International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 10 Issue: 10 | Oct - 2026 SJIF Rating: 8.659 ISSN: 2582-3930

Interfacing an I/O Device with a Raspberry PI Development Board: LED Interfacing Using GPIO Pins

Atharv Shankar Sathe
Department of E&TC
JSPM's BSP, Wagholi, Pune, India
atharvsathe1910@gmail.com

Neil Khude
Department of E&TC
JSPM's BSP, Wagholi, Pune, India
khudeneil1438@gmail.com

Shraddha Koymahale
Department of E&TC
JSPM's BSP, Wagholi, Pune, India
shraddhakoymahle28@gmail.com

Siddharth Kusgal
Department of E&TC
JSPM's BSP, Wagholi, Pune, India
siddharthkusgal@gmail.com

Durgesh Lohar
Department of E&TC
JSPM's BSP, Wagholi, Pune, India
lohardurgesh149@gmail.com

Prof. Dhaktode Gauri
Department of E&TC
JSPM's BSP, Wagholi, Pune, India
gauridhaktode123@gmail.com

***Abstract*—**The Raspberry Pi is an ultra-compact single-board computer that can interact seamlessly with external electronic devices through its General-Purpose Input/Output (GPIO) pins. This report explains the design, interfacing methodology, circuit safety, and programming implementation of connecting a Light Emitting Diode (LED) with a Raspberry Pi development board. The LED serves as a primary digital output transducer to demonstrate how high-level software instructions can govern physical hardware states deterministically. A suitable current-limiting resistor is calculated using Ohm's Law and wired in series with the LED, while a dedicated GPIO pin (BCM 17 / Physical Pin 11) is configured as an active digital output. A Python program utilizing the modern GPIO Zero library alternates the LED between illuminated (HIGH) and extinguished (LOW) states with calibrated time delays. The experiment establishes fundamental principles of embedded Linux hardware interfacing, 3.3V logic constraints, current sink/source limits, and robust exception handling. The architecture provides an extensible foundation for complex multi-actuator, sensor telemetry, and Internet of Things (IoT) platforms.

***Keywords*—**Raspberry Pi, GPIO, LED, Python, I/O interfacing, embedded systems, physical computing, single-board computer.

I. INTRODUCTION

A Raspberry Pi development board is a low-cost, compact computer widely utilized across modern engineering education, embedded automation, robotics, environmental monitoring, edge computing, and Internet of Things (IoT) applications. Unlike conventional desktop computers that restrict user interaction to standard USB peripherals and networking interfaces, the Raspberry Pi incorporates an accessible 40-pin header of General-Purpose Input/Output (GPIO) pins. These pins allow the central processing unit to exchange binary digital signals directly with external electronic circuits, acting as an essential bridge between software logic and physical electrical hardware.

The GPIO pins can be dynamically configured via software as inputs to read status voltages from sensors, pushbuttons, or optical switches, or as outputs to actuate indicator LEDs, buzzers, solid-state relays, and motor drivers. Mastering the interfacing of basic input/output (I/O) components represents the foundational milestone in embedded systems engineering.

In this experiment, a Light Emitting Diode (LED) is selected as the representative output device because its photometric response provides an instantaneous, unambiguous visual confirmation of circuit logic. When the designated GPIO pin is asserted to a logic HIGH state (3.3 V), forward electrical current flows through the series circuit, causing the semiconductor p-n junction of the LED to emit light. Conversely, when the GPIO pin transitions to logic LOW (0.0 V), the potential difference drops to zero, current stops flowing, and the LED turns off.

Because the Broadcom System-on-Chip (SoC) operates on sensitive 3.3 V CMOS logic with strict per-pin current thresholds (maximum 16 mA safe rating), connecting a bare LED directly to a GPIO pin would cause excessive current draw, immediately destroying the LED and irreparably burning the microcontroller output drivers. Therefore, integrating a calibrated current-limiting resistor in series is mandatory.

The primary objectives of this study are: (1) to safely design and assemble an external LED indicator circuit on a solderless breadboard; (2) to connect the circuit to the Raspberry Pi 40-pin header following exact physical and BCM pin mappings; (3) to author a deterministic Python control program using the GPIO Zero framework; and (4) to analyze the electrical switching characteristics, safety margins, and real-world embedded applications of digital I/O control.

II. PROPOSED SYSTEM AND ARCHITECTURE

A. Proposed System

The proposed system utilizes a Raspberry Pi development board (Model 4B / 3B+), a standard 5mm diffused red LED, a 330 Ω (1/4 W) carbon film current-limiting resistor, a solderless breadboard, and multi-strand male-to-female jumper wires. A Python application running under Raspberry Pi OS configures BCM GPIO 17 as an active output and cycles its electrical state between logic HIGH and LOW at one-second intervals.

B. System Architecture

The physical signal propagation pipeline is structured sequentially as:

Python Script → /dev/gpiomem → BCM2711 GPIO Output Buffer → Resistor (330Ω) → LED Anode → Ground Pin (0V)

The user application issues commands through high-level object-oriented calls. The Linux kernel translates these instructions into hardware register writes. The physical GPIO pin sources current through the resistor, which regulates current through the diode into the Raspberry Pi ground plane.

C. Software Architecture

The software architecture operates in user-space under the Debian-based Raspberry Pi OS. Rather than requiring low-level root kernel privileges, modern Raspberry Pi distributions utilize the `/dev/gpiomem` interface, which grants memory-mapped access to the GPIO peripheral registers for users belonging to the `gpio` group.

The application is authored in Python 3 using the official `gpiozero` library. GPIO Zero abstracts complex hardware setup by treating external components as intuitive software objects. Broadcom (BCM) pin numbering is employed natively, ensuring hardware portability across diverse Raspberry Pi hardware revisions. A continuous `while True` loop toggles the pin state, while a `try...finally` exception block guarantees that pins are released safely to a LOW impedance state upon program termination.



Fig. 1. System block diagram of Raspberry Pi GPIO I/O interfacing architecture.



Fig. 2. Electrical interfacing circuit schematic connecting Raspberry Pi GPIO 17 to LED.

III. HARDWARE-BASED SYSTEM

A. Raspberry Pi Development Board

The Raspberry Pi 4 Model B serves as the primary computing and control core (Fig. 3). Powered by a Broadcom BCM2711 64-bit quad-core ARM Cortex-A72 processor clocked at 1.5 GHz, the board provides high computational bandwidth alongside real-time peripheral control. The 40-pin GPIO header operates strictly at 3.3 V CMOS logic levels. Input voltages exceeding 3.3 V will cause catastrophic latch-up and destroy the SoC silicon. Each output pin can safely deliver up to 16 mA, with a cumulative current budget across all GPIO lines limited to 50 mA.

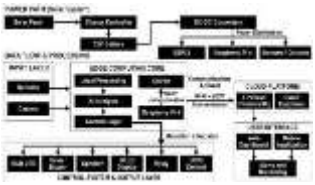


Fig. 3. Raspberry Pi 4 Model B single-board computer.

B. Light Emitting Diode (LED)

The LED is a specialized p-n junction semiconductor diode that emits incoherent narrow-spectrum light via electroluminescence when forward-biased. As illustrated in Fig. 5, the device possesses polar terminals: the longer lead indicates the Anode (positive terminal), while the shorter lead and adjacent flat notch on the plastic casing identify the Cathode (negative terminal). For standard 5mm red LEDs, the forward voltage drop (V_F) ranges between 1.8 V and 2.2 V, with typical luminous current ratings between 2 mA and 10 mA.

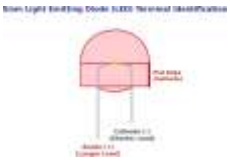


Fig. 5. 5mm LED terminal polarity and internal structure.

C. Current-Limiting Resistor

A fixed carbon-film resistor is connected in series to drop excess voltage and restrict forward loop current. Without this resistance, the LED presents negligible dynamic resistance once forward-biased, leading to thermal runaway and GPIO driver failure. Using Ohm's Law:

$$R = (V_{GPIO} - V_F) / I_F = (3.3\ V - 2.0\ V) / 0.00394\ A = 330\ \Omega$$

A standard 330 Ω (1/4 W, ±5% tolerance) resistor provides an optimal operating current of approximately 3.94 mA, ensuring vivid illumination while keeping pin load well below the 16 mA threshold.

D. Breadboard and Solderless Jumper Wires

A 400-point solderless breadboard provides the prototyping bed for component mounting. Terminal strips arranged in 5-hole vertical columns allow effortless interconnection between resistor leads and LED terminals. High-grade multi-strand male-to-female DuPont jumper wires establish reliable mechanical and electrical contact between the Raspberry Pi header and breadboard tie-points.

E. 40-Pin GPIO Header and Pin Selection

The Raspberry Pi 40-pin header incorporates two distinct numbering conventions: Physical (Board) numbering and Broadcom (BCM) GPIO numbering. Physical numbering designates pins sequentially from 1 to 40 based on physical location. BCM numbering reflects the internal channel designations of the Broadcom SoC silicon.

In this implementation, Physical Pin 11 is selected, corresponding to BCM GPIO 17. Ground return is established via Physical Pin 6 (Ground), which shares common zero-volt reference with the board power subsystem. The complete pinout layout is referenced in Fig. 4.



Fig. 4. Raspberry Pi 40-pin GPIO header reference pinout.

IV. CIRCUIT CONNECTION AND PROGRAM

A. Connection Procedure

Assembly must follow strict safety protocols:

1. Safely halt the operating system (`sudo poweroff`) and disconnect USB-C power before wiring.
2. Insert the red LED into the breadboard, noting the longer lead (Anode) and shorter lead (Cathode).
3. Insert one terminal of the 330 Ω resistor into the same breadboard tie column as the LED Anode.
4. Connect a female-to-male jumper wire from Raspberry Pi Physical Pin 11 (BCM 17) to the free lead of the resistor.
5. Connect a second jumper wire from Raspberry Pi Physical Pin 6 (GND) to the LED Cathode column.
6. Verify all connections, polarity, and seating before reconnecting power.

B. Python Program Implementation

The firmware code is authored in Python 3 using GPIO Zero:

```
from gpiozero import LED
from time import sleep

# Initialize LED on BCM GPIO 17 (Physical Pin 11)
led = LED(17)

print("Starting Raspberry Pi GPIO LED Control Program...")
print("Press Ctrl+C to safely terminate.")

try:
    while True:
        led.on()                # Set GPIO 17 HIGH (3.3V)
        print("LED Status: ON  [3.3V Sourced]")
        sleep(1)                # Hold state for 1.0 second

        led.off()               # Set GPIO 17 LOW (0.0V)
        print("LED Status: OFF [0.0V Sunk]")
        sleep(1)                # Hold state for 1.0 second

except KeyboardInterrupt:
    print("
Program interrupted by user (Ctrl+C).")

finally:
    led.off()                   # Ensure LED is extinguished
    led.close()                 # Release GPIO pin resources
    print("GPIO resources safely deallocated. Exit successful.")
```

C. Program Explanation

The script imports the `LED` abstraction from `gpiozero` and the blocking delay function `sleep` from `time`. The declaration `led = LED(17)` sets BCM pin 17 to digital output mode and ensures initial state is LOW. Inside the infinite loop, `led.on()` asserts 3.3V across the pin, driving forward current through the LED. The `sleep(1)` call holds execution for 1000 ms. `led.off()` drives output to 0.0V, shutting off current. The execution flowchart is illustrated in Fig. 7.

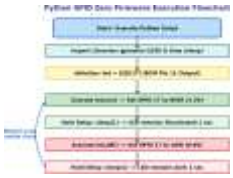


Fig. 7. Python GPIO Zero firmware execution flowchart.

D. Expected Results and Verification

Upon execution, the terminal displays real-time status output, and the breadboard LED blinks with a steady 50% duty cycle at a frequency of 0.5 Hz (1 second ON, 1 second OFF). Pressing Ctrl+C halts the loop gracefully, executing the `finally` block to prevent floating pin voltages.

V. WORKING OF THE SYSTEM

The operation of the system exemplifies the complete trajectory of digital physical computing. When the Python runtime executes `led.on()`, the underlying C-library writes a bitmask into the GPIO Output Set Register (`GPSET0`) of the Broadcom SoC. This turns on the high-side P-channel MOSFET inside the GPIO 17 output buffer, connecting the pin to the internal 3.3 V power rail.

Because the cathode terminal of the LED is tied to 0.0 V (Ground Pin 6), a forward bias voltage of 3.3 V appears across the series circuit. Approximately 2.0 V is dropped across the LED p-n junction, while the remaining 1.3 V is dropped across the 330 Ω resistor. Electrons and holes recombine within the direct bandgap of the Gallium Arsenide Phosphide (GaAsP) lattice, emitting visible photons at approximately 660 nm wavelength.

When `led.off()` is executed, the bitmask is written into the GPIO Output Clear Register (`GPCLR0`), switching on the low-side N-channel MOSFET and tying the pin to ground. The potential difference across the circuit becomes zero, stopping current flow instantly. Fig. 8 depicts the digital voltage and current waveforms measured across switching cycles.

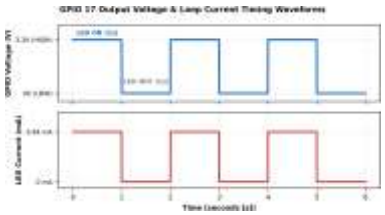


Fig. 8. GPIO 17 output voltage and loop current timing waveforms.

VI. APPLICATIONS

While simple in assembly, GPIO digital interfacing constitutes the core building block for sophisticated automation architectures:

- **Embedded Telemetry Status Indicators:** Displaying system health, Wi-Fi connectivity, MQTT broker links, and fault codes on edge IoT gateways.
- **Optical Alarm Annunciation:** Providing visual distress warnings in fire detection, gas leakage monitoring, and campus emergency smart poles.
- **Relay & Solenoid Driving:** Triggering optocoupled relay stages to control high-voltage 230V AC industrial lamps, water pumps, and solenoids.
- **Microcontroller Handshaking:** Providing digital strobe signals and interrupt pulses to synchronize coprocessors such as the ESP32 and Arduino.
- **Educational Prototyping:** Instructing students on Linux kernel memory mapping, Python scripting, and basic electronics principles.

VII. ADVANTAGES AND LIMITATIONS

A. Key Advantages

- **Simplicity & Rapid Deployment:** Requires only four passive components and minimal setup time.
- **High-Level Object Abstraction:** GPIO Zero provides readable, pythonic object orientation without raw pointer manipulation.
- **Energy Efficiency:** Consumes negligible power (less than 15 mW during active illumination).
- **Low Cost:** Assembled using readily accessible, budget-friendly components.

B. Technical Limitations & Electrical Constraints

- **Strict 3.3V Logic Barrier:** Direct connection to 5.0 V systems (such as legacy 5V Arduino boards) will burn the SoC input buffers.
- **Restricted Drive Capability:** Maximum current per pin is 16 mA; heavy loads (motors, coils) cannot be powered directly.
- **Non-Real-Time OS Jitter:** Because standard Raspberry Pi OS runs a non-preemptive Linux kernel, software delays can exhibit slight microsecond scheduling latency compared to bare-metal microcontrollers.

TABLE I. ELECTRICAL CHARACTERISTICS & OPERATING RATINGS

Parameter	Value	Operating Note
Logic HIGH Voltage (V_OH)	3.3 V DC	Minimum 3.0 V at rated current
Logic LOW Voltage (V_OL)	0.0 V DC	Maximum 0.4 V at rated sink
Safe Continuous Current	16 mA / pin	330 Ω resistor draws ~3.94 mA
Total 40-Pin Current Limit	50 mA max	Cumulative limit for all active pins
LED Forward Voltage (V_F)	1.8 V - 2.2 V	Nominal 2.0 V for 5mm Red LED
Resistor Power Dissipation	5.1 mW	Far below 250 mW (1/4 W) rating

VIII. FUTURE SCOPE

The basic LED circuit serves as a gateway to advanced embedded projects: (1) implementing software-based Pulse-Width Modulation (PWM) via `PWMLED` to achieve smooth 0-100% luminance dimming and fading; (2) integrating digital input devices such as tactile pushbuttons, PIR sensors, and ultrasonic sensors to trigger conditional lighting; (3) publishing device telemetry to cloud brokers (Firebase, ThingsBoard, Adafruit IO) via MQTT; and (4) interfacing optocoupled solid-state relays (SSRs) to switch mains electrical equipment autonomously.

IX. CONCLUSION

This research report presented the complete design, hardware assembly, electrical analysis, and Python software implementation for interfacing a Light Emitting Diode with a Raspberry Pi development board. The experiment proved that high-level software algorithms can reliably govern physical hardware states with absolute precision. By integrating a 330 Ω current-limiting resistor, the circuit established safe operation within the 3.3V CMOS electrical boundaries of the Broadcom SoC, drawing a conservative 3.94 mA. The study emphasized the necessity of correct LED lead polarity, BCM-to-physical pin translation, and clean GPIO deallocation via exception handling. The principles demonstrated here constitute the foundational architecture for IoT edge devices, industrial automation, and robotic telemetry.

ACKNOWLEDGMENT

The authors express their deepest gratitude to our project guide Prof. Dhaktode Gauri, and the faculty members and laboratory staff of the Department of Electronics and Telecommunication Engineering (E&TC), JSPM's Bhivarabai Sawant Polytechnic (BSP), Wagholi, Pune, for their technical mentorship, equipment support, and continuous encouragement during the implementation of this embedded computing project.

REFERENCES

[1] Raspberry Pi Ltd., "Raspberry Pi Documentation: GPIO Hardware and Pinout Specifications," Cambridge, UK, 2023. [Online]. Available: https://www.raspberrypi.com/documentation/computers/raspberry-pi.html

[2] E. Upton and G. Halfacree, *Raspberry Pi User Guide*, 4th ed. Indianapolis, IN: John Wiley & Sons, 2016.

[3] B. Scheid, *Raspberry Pi Hardware Reference*. New York, NY: Apress, 2017.

[4] GPIO Zero Contributors, "GPIO Zero: A Simple Interface to Everyday GPIO Devices on Raspberry Pi," ReadTheDocs, Rev. 2.0, 2023. [Online]. Available: https://gpiozero.readthedocs.io/

[5] S. Monk, *Programming the Raspberry Pi: Getting Started with Python*, 2nd ed. New York, NY: McGraw-Hill Education, 2016.

[6] D. Norris, *Raspberry Pi Projects for the Evil Genius*. New York, NY: McGraw-Hill Education, 2014.

[7] Broadcom Corporation, "BCM2711 ARM Peripherals Architecture Datasheet," Broadcom Silicon Documentation, 2020.

[8] M. Richardson and S. Wallace, *Getting Started with Raspberry Pi*, 3rd ed. Sebastopol, CA: Maker Media, 2016.

[9] Python Software Foundation, "Python 3 Language Reference Documentation," 2024. [Online]. Available: https://docs.python.org/3/

[10] P. Scherz and S. Monk, *Practical Electronics for Inventors*, 4th ed. New York, NY: McGraw-Hill, 2016.

[11] A. S. Tanenbaum and H. Bos, *Modern Operating Systems*, 4th ed. Boston, MA: Pearson, 2015.

[12] J. Blum, *Exploring Raspberry Pi: Architecture, Software, and Hardware*. Indianapolis, IN: Wiley, 2016.

[13] Texas Instruments, "Understanding CMOS Logic Voltage Levels and GPIO Interface Standards," Application Report SCLA007A, 2021.

[14] A. K. Ray and K. M. Bhurchandi, *Advanced Microprocessors and Peripherals*, 3rd ed. New Delhi: McGraw-Hill, 2012.

[15] C. Bell, *Beginning Sensor Networks with XBee, Raspberry Pi, and Arduino*. New York, NY: Apress, 2015.

APPENDIX: GPIO SAFETY AND CIRCUIT PROTECTION RULES

1. **Never exceed 3.3V:** Applying 5V signals directly to any GPIO pin will destroy the SoC.
2. **Always use a series resistor:** Never connect a bare LED across GPIO and GND.
3. **Power down during wiring:** Disconnect USB-C power before inserting breadboard jumpers.
4. **Observe current limits:** Do not draw more than 16 mA from a single pin or 50 mA total.
5. **Use external driver for inductive loads:** Never drive relays, motors, or buzzers directly.

BIOGRAPHIES



Prof. Dhaktode Gauri is a Lecturer in the Department of Electronics and Telecommunication Engineering (E&TC), JSPM's Bhivarabai Sawant Polytechnic (BSP), Wagholi, Pune, Maharashtra, India (e-mail: gauridhaktode123@gmail.com). Her academic research focuses on embedded microcontroller architectures, single-board computing, sensor networks, automation engineering, and IoT telemetry systems.



Atharv Shankar Sathe is with the Department of Electronics and Telecommunication Engineering (E&TC), JSPM's Bhivarabai Sawant Polytechnic (BSP), Wagholi, Pune, Maharashtra, India (e-mail: atharvsathe1910@gmail.com). His research interests include single-board computer architectures, embedded Linux systems, Python GPIO development, IoT edge telemetry, and intelligent automation systems.



Neil Hanumant Khude is with the Department of Electronics and Telecommunication Engineering (E&TC), JSPM's Bhivarabai Sawant Polytechnic (BSP), Wagholi, Pune, Maharashtra, India (e-mail: khudeneil1438@gmail.com). His technical interests include embedded robotics, IoT sensor interfacing, digital control systems, and edge AI applications.



Shraddha Sanjay Koymahale is with the Department of Electronics and Telecommunication Engineering (E&TC), JSPM's Bhivarabai Sawant Polytechnic (BSP), Wagholi, Pune, Maharashtra, India (e-mail: shraddhakoymahle28@gmail.com). Her primary research areas include smart campus monitoring, IoT telemetry, wireless networking, and microcontroller interfacing.



Siddharth Laximanputra Kusgal is with the Department of Electronics and Telecommunication Engineering (E&TC), JSPM's Bhivarabai Sawant Polytechnic (BSP), Wagholi, Pune, Maharashtra, India (e-mail: siddharthkusgal@gmail.com). His areas of interest encompass circuit design, hardware prototyping, sensor calibration, and embedded hardware security.



Durgesh Ganesh Lohar is with the Department of Electronics and Telecommunication Engineering (E&TC), JSPM's Bhivarabai Sawant Polytechnic (BSP), Wagholi, Pune, Maharashtra, India (e-mail: lohardurgesh149@gmail.com). His academic interests focus on power electronics, automation systems, smart infrastructure, and embedded system debugging.