Motion Detection & Alarming System

ISFAQ EVAN DIPRO 20255692

IBTISHAM MUHIT 20255696

JAHIDUL ISLAM 20255665

SEYAM HOSASIN 20255707

FARJANA RAHMAN BRISTY 20255703

Contents

1 Simulation Code Documentation

1.1 Project Title

ESP32-Based PIR Motion Detection Security System with Telegram Alert Notification (Pure C Simulation)

1.2 Project Overview

This project simulates a smart security system using a PIR (Passive Infrared) motion sensor, buzzer alarm, WiFi connection, and Telegram notification service. The system continuously monitors an area for movement and sends alerts whenever motion is detected. The implementation is written entirely in Standard C and simulates the behavior of an ESP32-based IoT security device.

1.3 Objectives

- Detect motion using a PIR sensor.
- Trigger an alarm when movement is detected.
- Send instant notifications through Telegram.
- Allow remote control using Telegram commands.
- Maintain a motion detection counter.
- Simulate an embedded IoT security system using Standard C.

1.4 System Features

1.4.1 Motion Detection

Detects movement using a simulated PIR sensor. Tracks the current motion status.

1.4.2 Alarm System

Activates buzzer when motion is detected and deactivates it when the area becomes clear.

1.4.3 Telegram Notifications

Sends alerts when motion occurs and when the area clears; also provides system status information.

1.4.4 Remote Commands

The system supports the following commands:

1.5 Technologies Used

- C Language – Main programming language.
- Standard Library – Input/Output operations.
- String Library – Command processing.
- Time Library – Future timing operations.
- ESP32 Concept – IoT system simulation.

Command	Description
`/on`	Activate system
`/off`	Deactivate system
`/status`	Show current status
`/reset`	Reset motion counter
`/motion_on`	Simulate motion
`/motion_off`	Stop motion simulation
`/quit`	Exit program

Table 1: Simulation commands

- Telegram Bot API Concept – Alert notifications.

1.6 System Architecture

Input Layer: PIR Sensor
Processing Layer: Motion Detection Logic, Command Processor, Alert Generator
Output Layer: Buzzer Alarm, Telegram Notifications, Status Display
Flow: PIR Sensor → Motion Detection → Alarm Trigger → Telegram Alert

1.7 Global Variables

- `motionState` – stores whether motion is currently detected.
- `systemEnabled` – determines if the security system is active.
- `motionCount` – counts total motion detection events.
- `buzzerState` – stores buzzer status.
- `pirSensorVal` – stores simulated PIR sensor value.

1.8 Functional Description

1.8.1 connectWiFi()

Simulates WiFi connection; displays connection status.

1.8.2 sendTelegram()

Simulates sending Telegram messages. Takes message text as input and prints the notification.

1.8.3 readMotion()

Reads the PIR sensor state; returns HIGH or LOW.

1.8.4 turnBuzzerOn() / turnBuzzerOff()

Activate / deactivate the alarm buzzer.

1.8.5 handleMotion()

Processes motion events: increases motion counter, activates buzzer, sends Telegram alerts, and resets alarm when motion stops.

1.8.6 processCommand()

Interprets user commands (`/on`, `/off`, `/status`, `/reset`, `/motion_on`, `/motion_off`) and modifies system behaviour accordingly.

1.8.7 printStatus()

Displays current system information: system state, buzzer state, PIR state, and motion count.

1.8.8 setup()

Initialises the system, configures simulated GPIO pins, connects to WiFi, warms up the PIR sensor, and starts Telegram service.

1.8.9 main()

Runs the main control loop: reads user commands, processes motion detection continuously, and handles the system accordingly.

1.9 Program Workflow

1. System starts.
2. GPIO pins are configured.
3. WiFi connection is established.
4. Telegram bot becomes active.
5. User enters commands.
6. Motion is simulated.
7. Alarm activates.
8. Telegram alert is sent.
9. Motion count is updated.
10. System continues monitoring.

1.10 Sample Execution

System Startup:

```
ESP32 Motion Detection System
Pure C Simulation
[WiFi] Connecting... Connected!
[TELEGRAM] System is ONLINE
```

Motion Detection:

```
/motion_on
[PIR] Motion detected!
[BUZZER] ON
[TELEGRAM] Motion Detected! Total Count: 1
```

Motion Ends:

```
/motion_off
[PIR] No motion
[BUZZER] OFF
[TELEGRAM] Area is Clear
```

1.11 Advantages

- Simple implementation.
- Easy to understand.
- Demonstrates IoT concepts.
- Supports remote monitoring.
- Real-time alert simulation.
- Expandable for actual ESP32 hardware.

1.12 Limitations

- Uses simulated hardware.
- Does not connect to actual WiFi.
- Telegram communication is simulated.
- No real sensor integration.

1.13 Future Improvements

- Real ESP32 implementation.
- Actual Telegram Bot API integration.
- Camera image capture.
- Cloud database logging.
- Multiple sensor support.
- Mobile application integration.
- Face recognition security.
- Intruder image transmission.

1.14 Conclusion

The ESP32 PIR Motion Detection and Telegram Alert System successfully demonstrates the basic principles of an IoT-based security system using Standard C. The project combines motion detection, alarm activation, wireless communication, and remote control features into a single application. It serves as an excellent educational project for understanding embedded systems, IoT security, and event-driven programming.

2 Real Hardware Code Documentation

2.1 Project Title

ESP32 PIR Motion Detection and Telegram Alert System (Hardware Implementation)

2.2 Project Overview

This project implements a real-time smart security system on an ESP32 microcontroller. It uses a physical PIR motion sensor, a buzzer for local alarm, and the Telegram Bot API for instant remote notifications. The system continuously monitors a physical area, triggers an alarm on motion detection, and allows remote control via Telegram commands. The code is written in standard C for the ESP32, using the Arduino framework for hardware abstraction.

2.3 Objectives

- Detect physical movement using a PIR sensor connected to ESP32.
- Activate a buzzer alarm for a fixed duration when motion is detected.
- Send instant motion alerts and status updates via the Telegram messaging platform.
- Accept remote commands (system on/off, status, counter reset) through Telegram.
- Implement a state machine with alarm, cooldown, and idle phases for reliable operation.
- Maintain a persistent motion detection counter.

2.4 System Features

2.4.1 Motion Detection

Reads actual digital signal from a PIR sensor (HIGH when motion is present). Includes a built-in warm-up delay of 30 seconds for sensor stabilization.

2.4.2 Alarm System

Activates buzzer immediately upon motion detection; keeps buzzer on for 25 seconds (configurable). After the alarm period, the system enters a 3-second cooldown phase to prevent immediate re-triggering.

2.4.3 Telegram Notifications

Sends `"Motion Detected!"` on each new motion event, `"Cooldown Start"` when the alarm ends, and `"System Online"` at startup. Provides status and counter values on request.

2.4.4 Remote Commands

Commands are processed from incoming Telegram messages.

Command	Description
`/on`	Enable the security system
`/off`	Disable the system (buzzer forced OFF)
`/status`	Request current system state & count
`/reset`	Reset motion counter to zero

Table 2: Hardware commands

2.4.5 State Machine

- **IDLE** (state 0) – waiting for motion.
- **ALARM** (state 1) – motion detected, buzzer active for 25 s.
- **COOLDOWN** (state 2) – 3 s quiet period before returning to IDLE.

2.5 Technologies Used

- ESP32 Microcontroller – central processing unit.
- C Language – application logic.
- Arduino Framework – hardware abstraction and development environment.
- WiFi Library – connect to local wireless network.
- HTTPClient Library – send HTTPS requests to Telegram Bot API.
- Telegram Bot API – receive commands and send messages.
- PIR Motion Sensor – detect physical movement.
- Active Buzzer – audible alarm indication.

2.6 System Architecture

Hardware Layer: PIR sensor → GPIO2 (INPUT_PULLDOWN), Buzzer → GPIO4 (OUTPUT).
Software Processing Layer: Main loop polls Telegram and runs motion handling; command processor interprets incoming text; motion state machine controls alarm/cooldown timing.
Communication Layer: Telegram Bot API over HTTPS GET requests.
Data Flow:

$$PIRSensor \rightarrow DigitalRead \rightarrow StateMachine \rightarrow Buzzer + TelegramAlert$$

$$TelegramUpdate \rightarrow CommandProcessor \rightarrow SystemControl$$

2.7 Global Variables and Data Structures

SystemData Structure

```
typedef struct {
    int state;
    int motionCount;
    int systemEnabled;
    unsigned long stateStartTime;
    unsigned long lastMsgTime;
} SystemData;
```

- `state` – 0=IDLE, 1=ALARM, 2=COOLDOWN.
- `motionCount` – number of motion events detected.
- `systemEnabled` – 1 if system active, 0 if disabled.
- `stateStartTime` – timestamp when state was entered.
- `lastMsgTime` – timestamp of last Telegram message sent.

A global instance `sys` holds the system state. The variable `lastUpdateID` stores the latest Telegram update processed for offset-based polling.

2.8 Functional Description

2.8.1 connectWiFi()

Establishes connection to the configured WiFi network. Prints connection progress to serial; blocks until connected.

2.8.2 sendTelegram(char *msg, int force)

Sends a text message to the Telegram chat. Spaces in the message are URL-encoded as `%20`. The `force` parameter, when 0, respects a 3-second rate limit using `lastMsgTime`. Builds an HTTPS URL and performs a GET request via `HTTPClient`. Prints the returned HTTP code and updates `sys.lastMsgTime`.

2.8.3 getUpdates(char *buffer)

Polls Telegram for new incoming messages. Constructs a URL with `offset = lastUpdateID + 1`. Parses the `update_id` from the JSON response to update `lastUpdateID`. The raw response is stored in `buffer` for command processing.

2.8.4 processCommand(char *msg, SystemData *s)

Interprets incoming Telegram messages.

- `/on` – enables system, resets state to IDLE.
- `/off` – disables system, turns off buzzer, resets state.
- `/status` – replies with current system state and motion count.
- `/reset` – sets `motionCount` to zero.

2.8.5 handleMotion(SystemData *s)

Runs the motion-triggered state machine.

- **IDLE (0):** If PIR reads HIGH, increment counter, buzzer ON, send `"Motion Detected!"`, transition to ALARM.
- **ALARM (1):** Wait until 25 s have elapsed, then buzzer OFF, send `"Cooldown Start"`, transition to COOLDOWN.
- **COOLDOWN (2):** Wait 3 s, then return to IDLE.

2.8.6 initSystem(SystemData *s)

Resets all system fields to default startup values (state=0, motionCount=0, systemEnabled=1, timestamps=0).

2.8.7 setup()

Initialises hardware and software: serial communication at 115200 baud, configures GPIO pins (PIR input with pulldown, buzzer output), initialises system data, connects to WiFi, warms up PIR sensor for 30 seconds, and sends `"System Online"` via Telegram.

2.8.8 loop()

Main control loop. Repeatedly polls Telegram for new commands, processes them, and – if the system is enabled – calls `handleMotion()`; otherwise forces the buzzer OFF. A 200 ms delay controls the loop frequency.

2.9 Program Workflow

1. ESP32 boots → serial initialised.
2. GPIO pins configured.
3. System data structure initialised.
4. WiFi connection established.
5. PIR sensor warm-up (30 s).
6. `"System Online"` sent.
7. Loop forever:
   a) Fetch latest Telegram messages.
   b) Execute any received command.
   c) If system enabled: read PIR and run state machine.
   d) If system disabled: ensure buzzer is OFF.
   e) Small delay (200 ms).

2.10 Sample Serial Output

```
Connecting........
WiFi Connected!
PIR Warming up...
Ready!
Telegram HTTP Code: 200   (System Online)

STATE: ALARM
Telegram HTTP Code: 200   (Motion Detected!)
...
(25 seconds later)
STATE: COOLDOWN
Telegram HTTP Code: 200   (Cooldown Start)
...
(3 seconds later)
STATE: IDLE
```

2.11 Advantages

- Real hardware implementation with actual sensors and actuators.
- Reliable state machine prevents repeated triggers (alarm + cooldown phases).

- Remote control and monitoring via widely used Telegram app.
- Simple URL-based Telegram API calls – no complex libraries needed.
- Minimal external dependencies (WiFi, HTTPClient).
- Easy to extend with additional sensors or notifications.

2.12 Limitations

- Basic URL-encoding only replaces spaces; other special characters may break the request.
- Polling mechanism (getUpdates) is not real-time and relies on delays.
- No deep JSON parsing – command detection uses `strstr`, which can be fooled by substrings.
- No secure certificate validation (uses default HTTPClient configuration).
- Buzzer and PIR pins are fixed; changing requires recompilation.
- WiFi credentials and bot token are hard-coded in source.

2.13 Future Improvements

- Implement proper JSON parsing (e.g., ArduinoJson) for robust Telegram message handling.
- Add full URL encoding for all special characters.
- Use HTTPS with certificate verification for enhanced security.
- Support additional commands (e.g., `/motion_on` for test, `/motion_off`).
- Store credentials in non-volatile memory (SPIFFS/NVS).
- Integrate an LCD or OLED display for local status.
- Log motion events to a cloud database or SD card.
- Add OTA (Over-The-Air) firmware updates.
- Implement a web interface for configuration.

2.14 Conclusion

The ESP32 PIR Motion Detection and Telegram Alert System demonstrates a complete, working IoT security solution. It successfully combines hardware interfacing, network communication, and a state-machine based alarm logic. Remote control via Telegram provides user-friendly access, while the cooldown mechanism ensures robust real-world operation. This implementation serves as a practical foundation for more advanced home-automation and security projects.