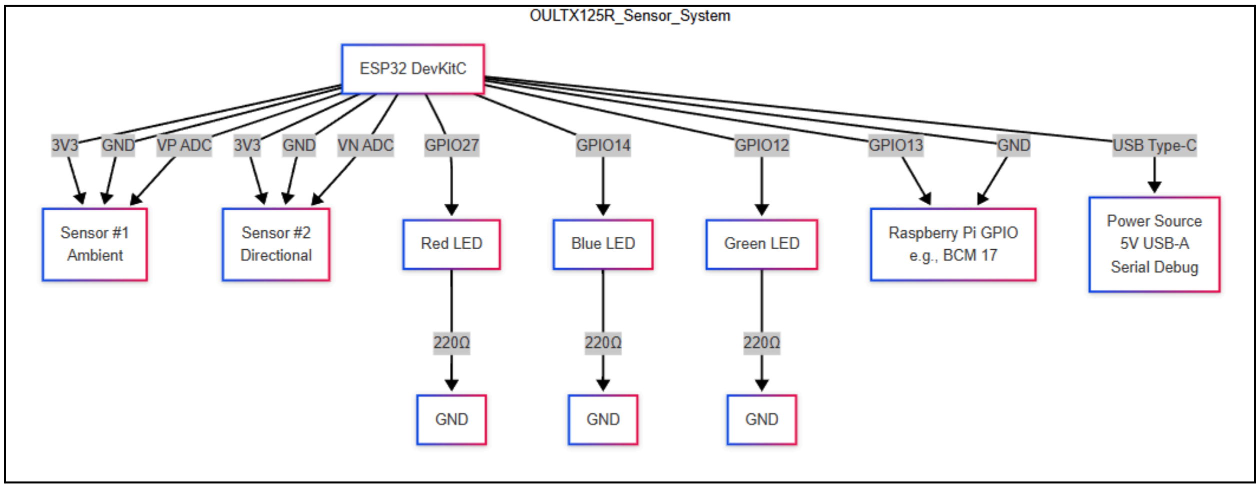
OULTX125R Integration Datasheet

Overview

The OULTX125R is an advanced dual-light sensor system designed for precise measurement of ambient and directional light levels in IoT applications. Built around an ESP32 DevKitC microcontroller, it delivers reliable data for smart home automation, energy management, and security systems. The sensor provides a standardized digital output on GPIO13 for seamless integration with external controllers like Raspberry Pi, complemented by three LED indicators for real-time operational feedback. Its compact design, low power consumption, and versatile connectivity options make it an ideal choice for developers enhancing environmental monitoring in diverse projects.

System Architecture

Pin Connections (Mermaid Diagram)



Connection Point	Function	Description
3V3	Power Supply	Provides 3.3V to sensor modules and LEDs, regulated from 5V USB input.
GND	Ground	Common ground for all components and external controllers (e.g., Raspberry Pi).
VP ADC	Ambient Sensor Input	Analog input for ambient light measurement.
VN ADC	Directional Sensor Input	Analog input for directional light measurement.
GPIO27	Red LED Control	Drives red LED to indicate low ambient light (<2000 ADC).
GPIO14	Blue LED Control	Drives blue LED to indicate low directional light (<2000 ADC).
GPIO12	Green LED Control	Drives green LED to indicate significant light difference (>100 ADC).
GPIO13	Digital Output	Outputs 3.3V HIGH when light difference > 100, LOW otherwise, for external use.
USB Type-C	Power and Debug	Supplies 5V power and Serial debug data (115200 baud) to Raspberry Pi or host.

The OULTX125R employs a modular architecture to ensure robust performance and straightforward integration:

- **Core Microcontroller**: The ESP32 DevKitC (dual-core, 240 MHz) processes analog light signals, calculates the difference between ambient and directional light, and drives outputs for integration and feedback.
- **Sensors**: Two light-sensitive modules detect:
  - **Ambient Light**: Measures overall environmental light (e.g., room lighting from lamps or natural sources).
  - **Directional Light**: Captures light from a specific direction (e.g., sunlight through a window).
- **Indicators**: Three LEDs provide visual status:
  - **Red LED** : Illuminates when ambient light is low (<2000 ADC value).
  - **Blue LED** : Illuminates when directional light is low (<2000 ADC value).
  - **Green LED** : Illuminates when the light difference exceeds 100 ADC units, signaling a significant environmental change.
- **Output**: GPIO13 delivers a digital signal (3.3V HIGH/0V LOW) when the light difference exceeds 100, enabling easy interfacing with external systems.
- **Debug Interface**: Serial output over USB (115200 baud) provides detailed sensor data, including ambient, directional, difference values, and GPIO13 state, for monitoring and calibration.
- **Power Management**: Powered via ESP32 USB Type-C (5V) or VIN pin, with internal regulation to 3.3V for sensors and LEDs, ensuring stable operation.

Specifications

General

- **Sensor Type**: Dual light intensity sensor
- **Microcontroller**: Expressif ESP32 DevKitC (Dual-core, 240 MHz)
- **Dimensions**: ~50mm x 25mm (ESP32) + 20mm x 15mm per sensor module
- **Weight**: ~25g (including LEDs)
- **Operating Voltage**: 3.3V or 5V DC
- **Power Consumption**: <500mW (typical)

Sensors

- **Type**: Light-sensitive modules with adjustable sensitivity
- **Sensing Range**: 0-100,000 lux (after calibration)
- **Output**: Analog (0-3.3V, mapped to 0-4095 via 12-bit ADC)
- **Response Time**: <10ms
- **Calibration**: Adjustable via onboard controls for optimal range

Indicators

- **Red LED**: Indicates low ambient light (<2000 ADC value)
- **Blue LED**: Indicates low directional light (<2000 ADC value)
- **Green LED**: Indicates significant light difference (>100 ADC value)

Input

- **Light Input**:
  - Sensor #1: Ambient light (connected to ESP32 ADC)
  - Sensor #2: Directional light (connected to ESP32 ADC)
- **Power Input**: 5V DC via ESP32 USB Type-C or VIN pin

Output

- **Primary Output**:
  - **Pin**: GPIO13 (digital, 3.3V HIGH/0V LOW)
  - **Condition**: HIGH when light difference (ambient vs. directional) > 100; LOW otherwise
  - **Purpose**: Signal for external controllers (e.g., Raspberry Pi GPIO)
- **Debug Output**:
  - Serial over USB (115200 baud)
  - Format: `Ambient: <value> | Directional: <value> | Diff: <value> | Output Pin 13: <HIGH/LOW>`
- **Indicator Output**:

  - Red LED: On when ambient < 2000
  - Blue LED: On when directional < 2000
  - Green LED: On when difference > 100

Performance

- **Resolution**: 12-bit ADC (0-4095)
- **Accuracy**: ±5% (post-calibration)
- **Sampling Rate**: 1 Hz (configurable)
- **Latency**: <50ms

Connectivity

- **Integration**: GPIO13 digital output for controllers (e.g., Raspberry Pi, Arduino)
- **Debug**: USB Type-C for Serial monitoring (accessible via Raspberry Pi)
- **Power**: USB Type-C (5V) or 5V/3.3V external supply

Environmental

- **Operating Temperature**: -20°C to 60°C
- **Humidity**: 10-90% RH (non-condensing)
- **IP Rating**: IP20 (indoor use)

Powering the Sensor

- **Primary Method**: Power via ESP32 USB Type-C port (5V DC).
  - Connect a USB Type-C cable from a USB-A port (e.g., Raspberry Pi 4, computer, or 5V USB adapter, minimum 500mA) to the ESP32 DevKitC USB port.
  - The ESP32 internally regulates 5V to 3.3V to power sensor modules and LEDs, ensuring stable operation across components.
  - USB connection simultaneously enables Serial debugging for real-time data monitoring, simplifying setup and troubleshooting.
- **Alternative**: 5V DC via ESP32 VIN pin (external supply, 500mA recommended).
- **Considerations**:

- Use a stable 5V/500mA USB source to prevent voltage drops that could affect sensor accuracy.
- Employ a high-quality, data-capable USB Type-C cable (not power-only) to ensure reliable power delivery and Serial communication.
- For portable or standalone setups, a 5V USB power bank can be used, maintaining compatibility with the USB Type-C interface.

Integration Guide

Application Scenarios

1. **Smart Home Lighting Control**:
    - **Setup**: Position the OULTX125R near a window in a 15x15 ft living room. Sensor #1 measures room-wide light (ambient), while Sensor #2 targets window light (directional).
    - **Operation**:
        - Morning: High directional light (e.g., sunrise, ADC 3500) compared to lower ambient (1500) sets GPIO13 HIGH, green LED on. The controller opens motorized blinds and dims smart lights to leverage natural light.
        - Midday: Balanced light levels (difference < 100) set GPIO13 LOW, minimizing adjustments for energy efficiency.
        - Evening: Low directional light (500) with moderate ambient (2000) sets GPIO13 HIGH, prompting the controller to brighten lights and close blinds for privacy.
    - **Benefits**: Enhances energy efficiency by optimizing natural light usage and maintains consistent room illumination.
2. **Security Monitoring**:
    - **Setup**: Install in a dark room or hallway for intrusion detection.
    - **Operation**: A sudden light source (e.g., flashlight, ADC spike to ~2500) causes a difference > 100, setting GPIO13 HIGH and illuminating the green LED. The controller triggers an alarm or sends a notification.
    - **Benefits**: Provides real-time detection of unauthorized light sources, enhancing security.
3. **Agricultural Lighting Optimization**:

   - **Setup**: Deploy in a greenhouse or indoor grow area.
   - **Operation**: Ambient sensor monitors overall light, directional sensor tracks sunlight. GPIO13 goes HIGH when natural light is insufficient (difference > 100), signaling the controller to activate grow lights.
   - **Benefits**: Optimizes plant growth conditions while minimizing power consumption.
4. **Office Environment Management**:
   - **Setup**: Place in an office with large windows and adjustable lighting.
   - **Operation**: High directional light from windows sets GPIO13 HIGH, prompting the controller to dim overhead lights. Low ambient light in meeting rooms triggers red LED and brighter lights. Significant differences (e.g., during presentations) adjust lighting dynamically.
   - **Benefits**: Improves occupant comfort and reduces energy costs in dynamic lighting environments.

Calibration Procedure

Accurate calibration is critical for reliable light measurements and preventing output saturation:

1. **Initial Setup**:
   - Connect sensors, LEDs, and power as per the diagram.
   - Power via USB Type-C for ease of debugging.
2. **Sensitivity Adjustment**:
   - In dim lighting (e.g., covered sensor or dark room), adjust sensor controls to achieve ADC readings of ~100-1000.
   - In bright lighting (e.g., direct sunlight or flashlight), adjust for ~2000-3500.
   - Avoid saturation (constant 4095 readings), which indicates excessive sensitivity.
3. **Verification**:
   - Monitor Serial output via USB to check `Ambient` and `Directional` values.
   - Test with varied lighting conditions (e.g., cover one sensor, illuminate the other) to ensure differential response.
   - Confirm GPIO13 toggles HIGH when light difference exceeds 100.
4. **Fine-Tuning**:

- Adjust thresholds in firmware (e.g., `<2000` for LEDs, `>100` for GPIO13) based on specific project requirements.
- Recalibrate if deploying in a new environment with different lighting conditions.

5. **Best Practices**:
   - Perform calibration in a controlled setting to avoid external light interference.
   - Regularly verify calibration to maintain accuracy over time.

Integration Steps

1. **Connect Sensor**:
   - Wire sensors, LEDs, and GPIO13 as shown in the diagram.
   - Power via USB Type-C (e.g., from Raspberry Pi USB-A port or 5V adapter).
   - Optionally connect GPIO13 to Pi GPIO (e.g., BCM 17) for digital signal.
2. **Calibrate Sensors**:
   - Adjust sensor sensitivity to ensure readings are within the expected range (avoid 4095 saturation).
3. **Read GPIO Output**:
   - Connect GPIO13 to Pi GPIO (e.g., BCM 17).

Use Python for Raspberry Pi:

```
import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BCM)
INPUT_PIN = 17
GPIO.setup(INPUT_PIN, GPIO.IN)
while True:
    print("Sensor:", "HIGH" if GPIO.input(INPUT_PIN) else "LOW")
    time.sleep(1)
```

-
- Run with `sudo python script.py`.

4. **Debug via USB on Raspberry Pi**:
   - Connect ESP32 USB Type-C to Pi USB-A port.
   - Identify device: Run `ls /dev/tty*` (e.g., `/dev/ttyUSB0` or `/dev/ttyACM0`).

Use Python to read Serial:

```
import serial
import time
ser = serial.Serial('/dev/ttyUSB0', 115200, timeout=1)
time.sleep(2)  # Wait for connection
while True:
    if ser.in_waiting > 0:
        line = ser.readline().decode('utf-8').rstrip()
        print(line)
    time.sleep(0.1)
```

-
- Install `pyserial`: `pip install pyserial`.
- Run with `python script.py`.
- Expected output: `Ambient: <value> | Directional: <value> | Diff: <value> | Output Pin 13: <HIGH/LOW>`.

5. **Monitor and Debug**:
    - Use Serial data to verify sensor readings, LED states, and GPIO13 output.
    - Adjust firmware thresholds for specific project needs (e.g., modify `diff > 100` or light < 2000).

Software

- **Firmware**: Preloaded C/C++ for ESP32 (Arduino-compatible)
- **Features**: Light difference detection, LED feedback, GPIO13 digital output, Serial debug output
- **Customization**: Modify thresholds (e.g., `diff > 100`, light < 2000) or sampling rate in firmware to suit specific applications
- **Source Code Availability**: Contact supplier for access to firmware source for advanced customization

Common Troubleshooting

- **No Serial Output on Raspberry Pi**:
    - Ensure ESP32 USB Type-C is connected to Pi USB-A; check `ls /dev/tty*` for device (e.g., `/dev/ttyUSB0`).
    - Verify Python script uses correct port and 115200 baud.
    - Confirm ESP32 is powered (USB LED on, typically visible on DevKitC).
- **Sensor Readings Saturated (4095)**:
    - Adjust sensor sensitivity controls to reduce output in bright light.
    - Test in dim light (expect 100-1000); bright light (2000-3500).
    - Ensure sensor outputs are connected to correct ADC inputs.
- **LEDs Not Lighting**:
    - Make sure LEDs are on: Check red (GPIO27), blue (GPIO14), green (GPIO12).
    - Verify anode to GPIO, cathode to 220Ω resistor to GND.
    - Ensure light levels meet thresholds (e.g., <2000 for red/blue LEDs).
    - Test LEDs independently by setting GPIO HIGH in firmware.
- **No GPIO13 Output**:
    - Check connection to Pi GPIO (e.g., BCM 17) and shared GND.
    - Verify light difference > 100 (vary light on one sensor, e.g., cover or illuminate).
    - Measure GPIO13 with a multimeter (3.3V HIGH, 0V LOW).
- **Unstable Readings**:
    - Secure all wiring (VCC to 3V3, GND to GND).
    - Minimize electrical noise (use short wires, avoid proximity to power sources).
    - Calibrate sensors in stable lighting conditions to ensure consistent output.

Safety and Compliance

- **Certifications**: CE, FCC (ESP32 module)
- **RoHS**: Compliant
- **Safety Notes**:
    - Ensure proper insulation of all connections to prevent short circuits.
    - Use only specified voltage (5V via USB or VIN) to avoid damage.
    - Avoid exposure to moisture or extreme temperatures beyond specified range.

Ordering Information

- **Part Number**: OULTX125R
- **Kit Contents**: ESP32 DevKitC, 2x light sensor modules, 3x LEDs (red, blue, green), 3x 220Ω resistors, jumper wires
- **Contact**: Supplier details available for bulk orders or custom configurations