11/5/2021

Ahmed S Ahmed

Student ID: 81441928

Report 2

EECS 221 [IOT]

Table of Contents

[1. The Pi Server 2](#_Toc87003128)

[1.1. Hardware 2](#_Toc87003129)

[1.1.1. System Components 2](#_Toc87003130)

[1.1.2. Pin Connections 2](#_Toc87003131)

[1.2. Software Overview 2](#_Toc87003132)

[1.3. Design Notes 3](#_Toc87003133)

[1.3.1. Initialization Block 3](#_Toc87003134)

[1.3.2. TCP Server 3](#_Toc87003135)

[1.3.3. Sensor Thread 4](#_Toc87003136)

[1.3.4. App Thread 4](#_Toc87003137)

[1.3.6. Enhancements 4](#_Toc87003138)

[1.4. Demo 4](#_Toc87003139)

[2. The ESP Client 5](#_Toc87003140)

[2.1. Hardware 5](#_Toc87003141)

[2.1.1. System Diagram 5](#_Toc87003142)

[2.1.2. Pin Connections 5](#_Toc87003143)

[2.2. Software Overview 5](#_Toc87003144)

[2.2.1. Setup 5](#_Toc87003145)

[2.2.2. Loop 6](#_Toc87003146)

[2.2.2.1. Enhancements 6](#_Toc87003147)

[2.3. Demo 6](#_Toc87003148)

# The Pi Server

## Hardware

As the Pi board doesn’t support an ADC, there must be an ADC connected with the Photo LDR. I used an ESP board as an ADC, as I don’t have ADC chip “MCP3008”. The ESP is connected to the Photo LDR using an analog pin. At the same time, the Pi gets the sensor readings periodically over the UART.

NOTE: The ESP board connected to the Pi is part of the Server system, as shown in the below figure

### System Components



### Pin Connections

|  |  |
| --- | --- |
| **Raspberry Pi Header** |  |
| GPIO23 | ESP\_INT (D1) |
| GPIO15 (UART RX) | ESP\_UART\_TX (D4) |
| GND | ESP\_GND |
| GPIO13 | Red Led |
| GPIO19 | Green Led |
| GPIO26 | Blue Led |

* Note: The ESP board is connected to the photo LDR using A0 pin.

## Software Overview



## Design Notes

### Initialization Block

Initialization block is responsible for initializing required HWs and global values, such as:

1. Initializing the communication with the ESP over UART.
2. Initializing the 3 LED PWM modules to control the LED intensity using the PWMIO module.
3. Initializing the global lists and dictionaries.

### TCP Server

The TCP server thread is responsible for the following:

1. Create the TCP socket and bind it to the board IP.
2. Listen to incoming connections.
3. Upon connection, wait until data is received from the ESP Client board.
4. Handle receiving multiple readings in the same message.
5. Parse and validate the message.
6. Apply the required LED behavior.
7. Signal the App thread after parsing the data to consume the data.

### Sensor Thread

The sensor thread is responsible for the following:

1. Read a new sensor value from the ESP board.
2. It signals the ESP board using an interrupt pin.
3. Then waits for the ESP board to send the sensor value over the UART connection.
4. Process the received data and extract the sensor value.
5. Update the sensor value to a shared memory location with the App thread.

### App Thread

The Application thread is responsible for the following:

1. Waits for a signal from the TCP thread to start the processing.
2. Once signaled, it gets the sensor value stored in shared memory with the sensor thread.
3. Combine the local sensor value with the Client received value.
4. It keeps accumulating records until it has 8 records (2 seconds window).
5. Calculates the average value for the 8 records and decides the led value.
   1. Client (ESP) has more light: Turn on the **Blue** LED
   2. Server (Pi) has more light: Turn on the **Green** LED.

#### ADC Code (ESP board)

The ESP board connected to the Pi as an ADC has a code that does the following:

1. Initialize the Serial0 and Serial1; Serial1 is used to communicate the sensor values to the Pi.
2. Configure an interrupt pin. The Pi uses the pin to inform the ESP to take a reading.
3. Upon rising edge interrupt, the ESP reads the sensor and sends the value to Pi over the UART connection.
4. Then waits for a new interrupt.

### Enhancements

* The data exchange between the Server and the Client is in JSON format. To be as close as possible to actual applications.
* Accessing the shared memory (Sensor values) between multiple threads needs a sync technique. A locking mechanism is used to guarantee data integrity.

## Demo

The video demo link is uploaded to this link.

# The ESP Client

## Hardware

### System Diagram



### Pin Connections

|  |  |
| --- | --- |
| **ESP8266 Header** |  |
| D1 | Red Led |
| D3 | Green Led |
| D2 | Blue Led |
| A0 | Photo LDR |

## Software Overview

## Setup

1. Initialize the serial with 115200 BaudRate.
2. Initialize the LED pins.
3. Initialize the analog pin connected to the photo LDR.
4. Initialize a timer interrupt module.
5. Configure the WiFi and connect to the home network.
6. Connect to the Server. In case of connection failure, start 5 seconds timer to retry.
7. Upon connection with the Server, start a 0.25-second periodic timer that initiates sending the data.

## Loop

1. In case the Server is not connected, loop on the connection retry flag to re-initiate the connection. A timer interrupt sets the retry flag.
2. When sending timer fires:
   1. Read the sensor value.
   2. Compose the JSON data object.
   3. Send the response to Server.
3. Loop for data received from the Server:
   1. Parse the server JSON response.
   2. Apply the command on the connected LEDs:
      * Client (ESP) has more light: Turn on the **Blue** LED.
      * Server (Pi) has more light: Turn on the **Green** LED.
4. Check on the total connection time to initiate a disconnection from the Server after 2 minutes.

### Enhancements

* Use timer interrupt module instead of Millis() function to enhance timing accuracy and code structure.
* Use Arduino JSON library to compose/decompose server requests/responses to make the project closer to actual projects.
* Include a timer to retry the Server connectivity in case the Server was down.

## Demo

The video demo link is uploaded to this link.