Computer Science — Digital Systems
Spring Semester 2024

Project Report

Smart Home System with ESP32

Group members:
Vladyslav Horbatenko, *@ruc.dk
Salar Komeyshi, *@ruc.dk
Kacper Hvid, *@ruc.dk

Roskilde University
Aug 2024

Problem Statement

How can light-reactive blinds be designed to effectively maintain cooler indoor temperatures and reduce energy consumption in response to global warming?

Problem Area

Global warming is intensifying the frequency and severity of heatwaves, leading to increased reliance on air conditioning and cooling systems, which in turn contributes to higher energy consumption and greenhouse gas emissions. To mitigate these effects and maintain comfortable indoor temperatures without excessive energy use, there is a need for innovative solutions. One potential approach is the development of light-reactive blinds that can automatically adjust to varying light and heat conditions. These blinds would help regulate indoor temperatures by blocking excess sunlight during hot periods, thereby reducing the need for artificial cooling and conserving energy.
The challenge lies in designing efficient, responsive blinds that can significantly contribute to energy savings and environmental sustainability in both residential and commercial buildings.

Problem Description

The project aims to design light-reactive blinds to enhance indoor cooling and reduce energy use amid global warming. It involves developing a Smart Home Device using an ESP32 microcontroller to host an HTML webpage that displays real-time indoor conditions. The ESP32, equipped with a light sensor (and future temperature and humidity sensors), measures light levels and updates the webpage. A JavaScript simulation on the page visually demonstrates how the blinds adjust to different light conditions, helping to maintain cooler indoor temperatures by regulating sunlight exposure.

System overview

This project has these key components:

- **ESP32 Microcontroller**: Collects light intensity data and hosting the web server.
- **Light Sensor**: Measures light levels and sends this data to the ESP32.
- **Web Server**: Created using the ESP32, it hosts a Web-page to visualize sensor data and simulate a room.
- **WebSocket Communication**: Enables real-time data transfer between the ESP32 and the web client.

Environment Setup: VS Code with PlatformIO Extension

Visual Studio Code with the PlatformIO IDE extension is used to run the program.
The process begins with installing the PlatformIO extension from the VS Code Extensions marketplace. Once the extension is activated, a new project is created. The WebSocketsServer library by Markus Sattler is then added to the project within the PlatformIO environment. A `data/` folder is created, containing `index.html` and `script.js` files, followed by updating the `src/main.cpp` file with our code accordingly. With the general drivers for the ESP32 installed on the device, the program is ready for execution.

ESP32 Code and Set Up

Connecting the Light Sensor to ESP32

To receive data about light intensity we connect light sensor to 3.3V NC pin and resistor to GND, and connect them together with additional cable going to one of the analog readers, which is pin 34 in our case. Therefore we receive information about resistance based on the amount of light it detects. The ESP32 reads this data using the `analogRead()` function, and saves this as `lightValue` variable.

```
const int lightPin = 34; //Pin from which we read data
lightValue = analogRead(lightPin);
```

The `analogRead()` function reads data from the light sensor and returns a value between 0 and 4095. However, this raw data may not be directly useful, so we mapped it to a more meaningful range, such as 0 to 1000 lux, using the `map()` function:

```
lightValue = map(lightValue, 0, 4095, 0, 1000);
```

This step converts the raw sensor value into a format that can easily be interpreted as light intensity (in *lux*), making it more useful for practical applications. This conversion might not be the most correct, but we will keep it simple to work with this numbers.

WebSocket Server for Real-Time Communication

Once light intensity data is read and processed, ESP32 uses a WebSocket server to broadcast this data to all connected clients. WebSockets are chosen for their efficiency in real-time communication, allowing continuous updates without the need to repeatedly request data and how was suggested in course materials: IoT Tutorial.

The WebSocket server is initialized and set up to handle connections and events:

```
WebSocketsServer webSocket = WebSocketsServer(81);

void setup() {
    webSocket.begin();
    webSocket.onEvent(webSocketEvent);
}
```

The `webSocket.onEvent()` function attaches a callback function (`webSocketEvent`) that handles different WebSocket events such as connections, disconnections, and incoming messages, etc.

Broadcasting Sensor Data

The ESP32 broadcasts the sensor data every second using the following logic:

```
if (millis() - timer > 1000) {  // Check if a second has passed
    String json = "{\"light\":" + String(lightValue) + "}";
    webSocket.broadcastTXT(json);  // Broadcast the light value as JSON
    timer = millis();  // Reset the timer
}
```

Handling WebSocket Events

The `webSocketEvent` function is responsible for managing different types of WebSocket events that was taken from above mentioned tutorial from course: IoT Tutorial.

```
void webSocketEvent(uint8_t num, WStype_t type, uint8_t * payload, size_t length) {
    switch(type) {
        case WStype_DISCONNECTED:
            Serial.printf("[%u] Disconnected!\n", num);
            break;
        case WStype_CONNECTED:
            IPAddress ip = webSocket.remoteIP(num);
            Serial.printf("[%u] Connected from %d.%d.%d.%d\n", num, ip[0], ip[1], ip
    [2], ip[3]);
            webSocket.sendTXT(num, "Connected");
            break;
        case WStype_TEXT:
            Serial.printf("[%u] get Text: %s\n", num, payload);
            break;
    }
}
```

HTML code

HTML code, which also contains CSS code, allows us to display data from sensor in textual presentation at the same time with our P5.js code. It is split up in four parts: The Header, Styles, Body Content, Scripts.

The Header - The Header part gives our page its title, as well as its own favicon.
Styles - The Styles part defines how everything looks - CSS Code. This is for example the background colors, the cards colors, the placement of the cards, ensures the page is usable on a phone, etc.

The Body - The Body displays the page title, gets the data for the three sensor data cards and creates a placeholder for the p5.js.

Scripts - This links the placeholder for the p5.js code with the actual script and ensures its functionality.

JavaScript

Our JavaScript simulation visualizes light intensity data received from an ESP32 microcontroller via a WebSocket connection. The system provides a graphical representation of a window with curtains that change color based on the incoming light data, portraying a window's blinds opening and closing. This visualization helps us understand how the blinds' response to varying light levels can impact indoor conditions.

Variables - We initialise a few variables for the simulation, in this case these are the sensor data variables (Temp, Hum, Light), as well as the web socket and canvas.

```
// Global variables for sensor data
let temperature = 0;
let humidity = 0;
let lightIntensity = 0;

// p5.js related variables
let socket;
let canvas;
```

Setup Function: In the setup function we:

1. Create the p5.js canvas and attach it to the HTML element, with the ID `canvas-container`
2. Connect the simulation to the websocket to ensure it receives real time data.
3. Update the HTML elements and p5.js whenever there is new data.
4. Create the Boolean function to check if light intensity is below the threshold. As well as a function to get the color of the curtain, based on the light intensity.

Draw Function - Draws a simple scene with walls, floor, and a window. Gets and changes curtain color based on light intensity.

Draws a sun if light intensity is high:

```
if (!isCurtainGrey()) {
    fill("#FFC107");
    noStroke();
    ellipse(175, 150, 100, 100); // Static sun position for simplicity
}
```

Sum-up

Together, these three components; The ESP32, the HTML code, as well as the JavaScript, combine, and become: A Smart Home Device System, aimed at helping reduce energy consumption in response to global warming. Does the System achieve this goal? Arguably so, although we cannot delve into the hows and what in this mini-project.