Design of the UV Exposure Box

Table of Contents

[Introduction 2](#_Toc68892940)

[Basic System Architecture: 2](#_Toc68892941)

[UV LED Control System 3](#_Toc68892942)

[UV Light Sensing System 3](#_Toc68892943)

[UI (User Interface) System 3](#_Toc68892944)

[Fan Control System 3](#_Toc68892945)

[Temperature Sensing System 3](#_Toc68892946)

[UVExpoBox System Diagnostics 4](#_Toc68892947)

[UVExpoBox Current Measurement System 4](#_Toc68892948)

[System Event List 4](#_Toc68892949)

[Hardware Architecture 5](#_Toc68892950)

[Software Architecture 6](#_Toc68892951)

# Introduction

Basic Operation: The artist will put the image in the box, set the exposure parameters via Web Interface over WiFi, and the start the process. The internal processor will initiate the turning on of the UV LED lighting..

Text, whiteboard

Description automatically generated

The major components, from left to right:

1. LED Control Layer. Proposed name: **UVEB\_LEDControl**. This will entail turning on/off. If I can put in current sensing, it will include that as well.
2. UV Light Sensing Layer, **UVEB\_UVStats**. Measurement of light data captured.
3. UI. **UVEB\_UI**. There will be some LED status, but principal UI will be HTTP over WiFi. UI will encompass Alarms, Status, Start/Stop commands.
4. Fan Control handling. **UVEB\_FanControl**. Fan on/off. May use PWM to handle fan speed.
5. Temperature Sensing Layer. **UVEB\_TempStatus**. Discovery of 1-wire temp sensors, reading temp data.
6. System Diag handling. **UVEB\_SystemStats**. Aggregate data collection, for use by alarms and events.

# Basic System Architecture:

## UV LED Control System

The hardware components consist of the following:

1. A LED power circuit with a constant current LED driver.
2. A GPIO to MOSFET switch. The CPU (ESP32) will drive the switch from a GPIO based upon the control algorithm.

The software components:

1. The LED Control Layer (UVEB\_LEDControl).

Basic operation of the UV LEDs. They will turn on when the artist will command the system to go on after the parameters are set. The UV LEDs will turn off either when the artist’s parameters sensed are completed OR if there is an error detected.

TBD: current measurement to monitor the power consumption for the UV LED.

## UV Light Sensing System

The hardware components consist of:

1. UV Light sensor(s) that uses a ML8511 (detects 280-390nm light) and can interface with the ESP32 easily.
2. Cables and placement at the bottom of the exposure floor in the box.

The software with consist of interfacing the ESP32 with the suitable device library and will sample the UV data collected at periodic intervals. When the SW determines that the exposure parameters have been met as per artist selection, the SW will issue a System Event to be processed by the Control SW. If the SW determines that the device is not working (sampling) correctly, another System Event will be issued that indicates a system error.

## UI (User Interface) System

The hardware components consist of:

1. A WiFi enabled ESP32 system.
2. An OLED Touch Screen. TBD (OLED Display Module IIC 128x64 Pixel 12864 OLED Blue I2C 0.96inch)
3. Some minimal buttons interface.

We want most of the UI interactions between the artist and the UV box to be web based. The web page should be easy and complete, and working from a smart phone.

## Fan Control System

If the data from the temperature sensors indicates a high temperature, having a fan kick in to cool off the environment would help.

The hardware components are:

1. A PWM based PC Fan.
2. Power system for the Fan.
3. A MOSFET switch to interface the CPU (ESP32) to the power on the Fan, including PWM to slowly ramp up the power to avoid possible current in-rush.

The GPIO from the ESP32 will turn on or off the power. For using PWM, another GPIO pin. TBD.

## Temperature Sensing System

The hardware components are:

1. One (or more) 1-wire protocol-based temperature probe(s).
2. GPIO for accessing the temperature sensor(s).

The software will monitor on a periodic basis the temperature in the box.

## UVExpoBox System Diagnostics

## UVExpoBox Current Measurement System

TBD.

On order: Current Sensor ACS712

## System Event List

Actions in the box will be triggered by Events.

|  |  |  |
| --- | --- | --- |
| Event Name | Event Description | Event Trigger |
| UVEB\_SYSTEM\_ON | UVEB Turn System On | Power on |
| UVEB\_SYSTEM\_OFF | UVEB Turn System Off | Power Off |
| UVEB\_SYSTEM\_REBOOT | Reboot system | UI Command; Diag scan; Watchdog |
|  |  |  |
| UVEB\_UVLED\_ON | Turn on UV LEDs | UI Command |
| UVEB\_UVLED\_OFF | Turn off UV LEDs | UI Command; Exposure condition met; Temp Overheat |
|  |  |  |
| UVEB\_HITEMP\_WM | High Temperature Watermark | Temp sense trigger; Diag scan; |
| UVEB\_LOTEMP\_WM | Low Temperature Watermark | Temp sense trigger |
|  |  |  |
| UVEB\_FAN\_ON | Turn on the fans | Sense Too Hot; UI Command; |
| UVEB\_FAN\_OFF | Turn off the fans | Sense cooled off; System shutdown |
|  |  |  |
| UVEB\_CURR\_MAX | Current is at Max | Current measurement |
| UVEB\_CURR\_LOW | Current too low | “ “ “ |
|  |  |  |
|  |  |  |

TBD

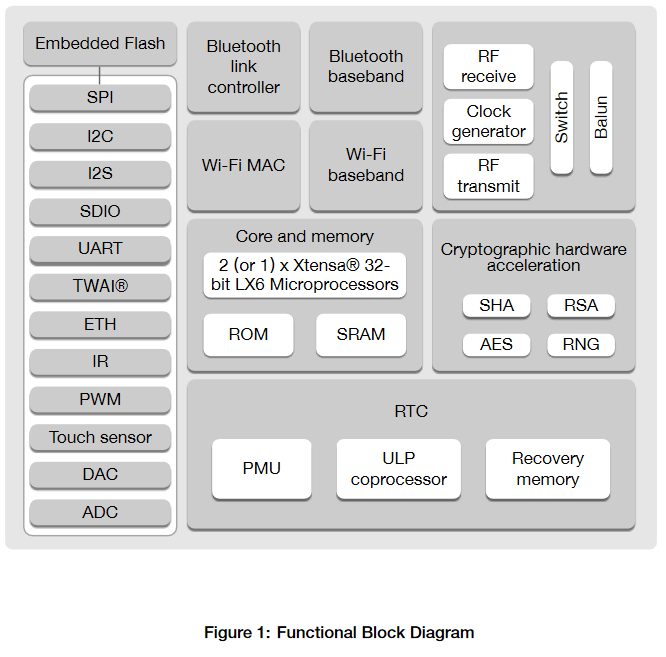
# Hardware Architecture

An snapshot of the Dev Kit:

A picture containing text, electronics, circuit

Description automatically generated

An overview diagram of the ESP32 DevKit architecture:



# Software Architecture

Having selected the ESP32 as the chief processing system for the UV Expo Box narrows down the SW platform choices, which is a good thing since it helps to focus the development efforts. There are two main development platforms for developing software for the ESP32:

1. PlatformIO plugin on VSCode
2. ESP-IDF plugin on VSCode

There are many pros and cons to each of these tools. But, choosing PlatformIO to be the chief tool makes the most sense, since we do not want to always be locked into a specific CPU/MCU and the PlatformIO has over 800 boards to support for a wide variety of IoT development, if there might be any thought of another Creative tool ion the horizon.

Certainly, the components listed above in the Basic System design section all have SW components that mirror the hardware components. But there are items to consider:

1. Boot code and Flash storage. The Dev Kits all have a Flash capable MCU, which ultimately means the flash will fill the RAM with executable code and run from there. There are already built SW functions for us to use w/o re-inventing the wheel.
2. Interrupt handling code. There are already built SW functions for us to use w/o re-inventing the wheel.
3. Low-layer code for the internal and external peripherals. There are already built SW functions for us to use w/o re-inventing the wheel.
4. Bare-metal firmware versus a Real-Time Operating System (RTOS). Both could be used here. I am exploring the inclusion of FreeRTOS. If we go with FreeRTOS, the SW can be written into well-defined Tasks, which is an extremely useful SW construct. Will make life (hopefully) easier.

Again, all of this can be done under PlatformIO.