Technische Universität München

Department of Informatics

Real-Time Systems (IN2060)

Project Report

Quirky Lamp

Author

Vasil Sarafov (sarafov@cs.tum.edu)

Lecturer

Dr. Alexander Lenz (alex.lenz@tum.de)

Thursday 25th January, 2018

Winter Term 17/18

Contents

1 Introduction **2**

2 Hardware **2**
2.1 Components 2
2.2 Electronics Schematics 3

3 Software **5**
3.1 Baseline 5
3.2 Architecture 5
3.2.1 Services 5
3.2.2 Scheduler 6
3.2.3 Serial Logger 6
3.3 Toolchain 7

4 Further Work **7**

5 Additional Materials **8**

1 Introduction

The following document provides a not so formal description of a simple IoT powered lamp that was created as part of a student project for the Real-Time Systems lecture, given by Dr.Alexander Lenz at the Technical University of Munich during the 2017/18 Winter Term.

The described system is called "Quirky Lamp" and utilizes the **Wi-Fi** and Bluetooth capable **ESP32** system on chip (SoC). It collects data from a photo resistor, sound and temperature **sensors** and distributes it via **MQTT**. A multicoloured lightning from an **LED strip** provides the basic lamp functionality. In addition to that, the lamp is capable of dynamically adjusting its lightning based on sound noises (e.g. reacting in real time on music).

The reason for creating this project is purely educational. I wanted to evaluate the ESP32 chip as I have experience with its predecessor - the ESP8266. The project does not solve any significant (or real) problem but rather aspires to be an engineering product that brings enjoyment to its creator.

The document is structured as follows. Sections 2 and 3 provide an overview of the hardware configuration and the software stack respectively. Chapter 4 describes known issues with the current state of the system and suggests improvements. Section 5 outlines additional materials such as references to the actual code base, pictures and demonstration videos of the system.

2 Hardware

In this Chapter we provide a more detailed description of the hardware components that were utilized in the project. Section 2.1 lists every single one of them and Section 2.2 depicts the corresponding electronics schematics.

2.1 Components

At the core of the system is the **ESP32** SoC. It is an upgraded successor of the very popular ESP8266 chip. ESP32 is created by the Chinese manufacturer Espressif Systems and is a low-power, low-cost Wi-Fi and dual-mode Bluetooth capable SoC. At its heart are a dual-core Tensilica Xtensa LX6 processor with a clock rate of up to 240 MHz, 448 KiB of ROM, 520 KiB of SRAM, up to 4 MiB embedded flash memory, Wi-Fi, Bluetooth modules and cryptographic hardware acceleration.

Additional information with references to the original manufacturing datasheet can be found in [5]. In our project we utilized the official ESP32 Core development module which provides an easy USB-to-Serial programming interface.

A simple photo resistor is used for measuring the *surrounding* lightning level (the measured value does not affect the lamp functionality). The temperature is measured by a TMP36 sensor and the sound level is captured by a KY-037 sensor.

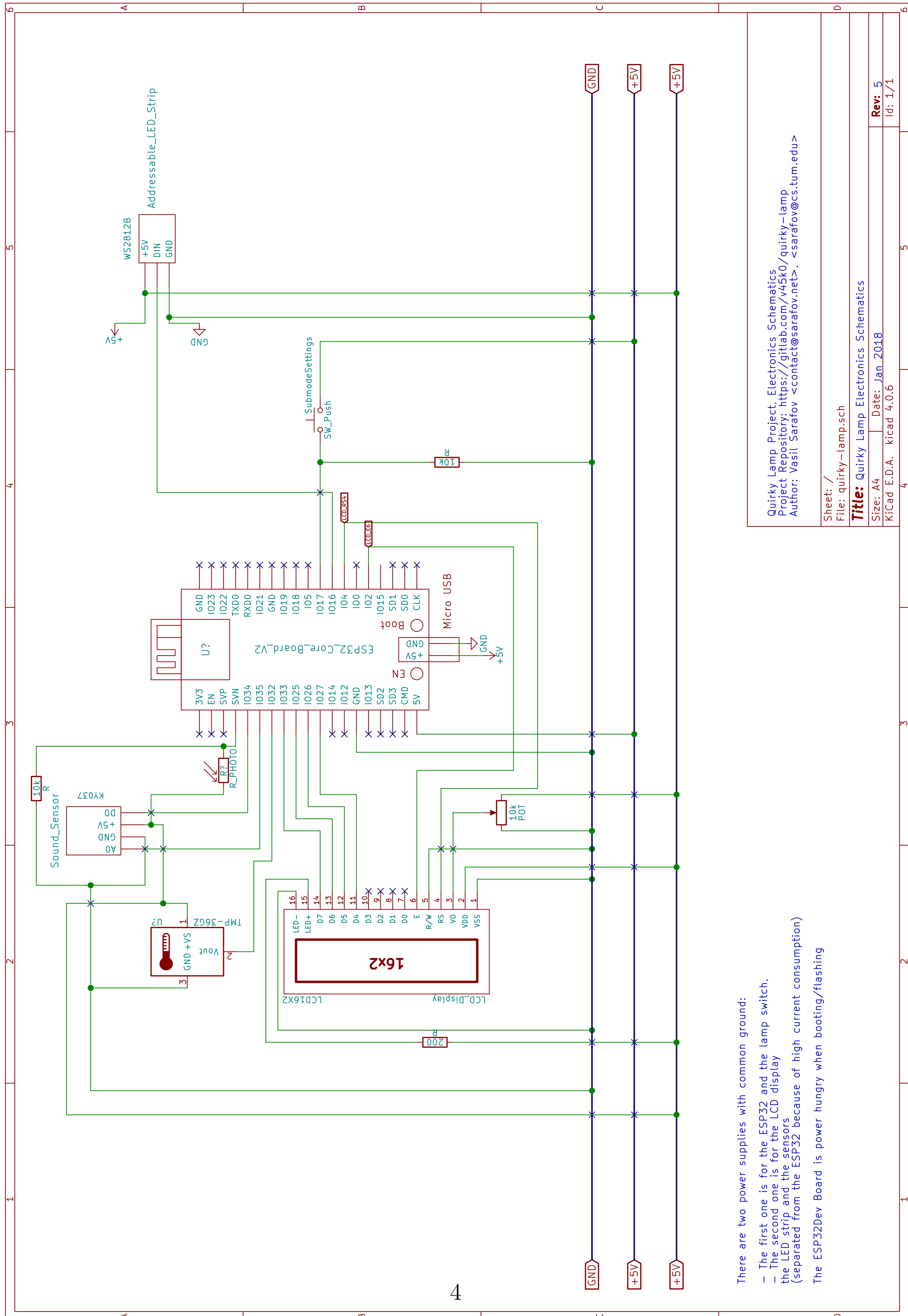
The lightning functionality of the lamp is achieved via a WS2812B-based strip with 60 individually addressable LEDs. With a push button the user is able to iterate through the different working modes of the lamp. Currently there are 7:

- Disabled - the lamp is not active, i.e. the LED strip is turned off
- White lightning - all 60 LEDs glow in white colour
- Yellow lightning - all 60 LEDs glow in yellow colour
- Pink lightning - all 60 LEDs glow in pink colour
- Green lightning - all 60 LEDs glow in green colour
- Purple lightning - all 60 LEDs glow in purple colour
- Reactive lightning - the colour and number of active LEDs is determined by the measured sound level

To provide a better user experience, an ADM1602K 16x2 LCD monitor is used to display the collected sensor data and the time elapsed since the system was booted. Data from the ESP32 is transferred to the LCD's microcontroller using a 4-bit data bus.

2.2 Electronics Schematics

The electronics schematics were created using the open source KiCad Software [6].


WS2812B
Addressable_LED_Strip
+5V
DIN
GND
SubmodeSettings
SW_Push
10k
R
ESP32_Core_Board_V2
U?
Boot
EN
Micro USB
Sound_Sensor
KY037
R_PHOTO
TMP-36GZ
Vout
LCD16X2
LCD_Display
16x2
200
10k
POT
Quirky Lamp Project, Electronics Schematics
Project Repository: https://gitlab.com/v45k0/quirky-lamp
Author: Vasil Sarafov <contact@sarafov.net>, <sarafov@cs.tum.edu>
Sheet: /
File: quirky-lamp.sch
Title: Quirky Lamp Electronics Schematics
Size: A4
Date: Jan 2018
Rev: 5
KiCad E.D.A. kicad 4.0.6
Id: 1/1
There are two power supplies with common ground:
- The first one is for the ESP32 and the lamp switch,
- The second one is for the LCD display
the LED strip and the the sensors
(separated from the ESP32 because of high current consumption)
The ESP32Dev Board is power hungry when booting/flashing

3 Software

3.1 Baseline

The baseline of the software stack, which the project's firmware is built on top of, is the official ESP32 IoT Development Framework Software Development Kit (SDK) provided by the manufacturer. It is a C code base, large part of which is closed source, that provides the core software components needed for interacting with the chip. This includes the hardware abstraction layer (HAL), a bootloader, a TCP/IP stack based on lwIP, core C and cryptographic libraries, a memory allocator, a file system implementation and many more. For more information about the SDK, please refer to its repository [8].

In addition to that, Espressif Systems provides a C++ interface to its SDK which implements the Wiring framework [9, 10]. This allows ESP32 projects to benefit from the large variety of arduino libraries and is exactly what our system does.

On top of the C++ Wiring interface sits our firmware. In the next section we will take a closer look of its architecture.

3.2 Architecture

The architecture of the firmware is rather simple. It consists of *services* (Section 3.2.1) which are the basic execution units, i.e. they are what processes are on a typical operating system without the isolation and protection mechanisms between them. Our services provide a common execution interface and are used to separate and easily reuse different business logic across the project.

A scheduler (Section 3.2.2) is utilized to organize the execution order of the services and meet their timing requirements.

A serial interface logger (Section 3.2.3) allows the developer to keep track of the firmware's action in real time.

3.2.1 Services

Currently there are 4 active services which all derive from the base service class to provide a common API.

When started, the **network service** is responsible for establishing and maintaining a WiFi connection to a network whose credentials are specified at compile time.

When executed, the service sends all queued data packets to an MQTT message broker. Message Queuing Telemetry Transport (MQTT) is a standardised and very lightweight data networking protocol which is based on the publish-subscribe communication pattern. It sits on top of TCP and and provides 3 different quality of service (QoS) levels. Because of its small overhead, easy client implementation and ability to provide asynchronous communication between peers in a distributed network, MQTT is very suitable for embedded devices. For more information about MQTT refer to the official specification [2]. A deeper analysis of MQTT's overhead and comparison with other IoT communication protocols can be found in [1].

In our case a self hosted MQTT message broker was used which is available at *mqtt.sarafov.net*. It takes advantage of the Mosquitto project [3]. The captured data can be then viewed from any MQTT client which can connect to the broker, provided that it has the needed authentication credentials. We used a simple MQTT client for android to remotely inspect the data.

The **sensor service** collects data from the temperature and sound sensors and reads the level of the photoresistor. The measured values are then displayed on the LCD monitor and are passed to the network service in order to be dispatched to the MQTT message broker.

The **sound lamp service** controls the LED lamp. When executed it polls the status of the push button data pin to register whether the user has changed the lamp's active mode (described in Section 2.1). If the reactive lightning mode is activated, the service measures the sound level and turns on the appropriate amount of LEDs.

The **watchdog service** periodically monitors the amount free heap memory space. If it is critically low, it sends a notification to the MQTT message broker via the networking service. It is helpful for detecting unwanted memory leaks.

3.2.2 Scheduler

The scheduler creates *tasks* from the services that need to run. A *task* has a starting time and can be either one-shot, meaning it must be executed only once, or periodic, meaning that it needs to be executed periodically. The implemented scheduler is non-preemptive and works on the "earliest starting time first" strategy.

3.2.3 Serial Logger

For developing purposes a serial logger was implemented. It is used to display useful information to the developer when the system is attached to a serial monitor.

The developer can then keep better track of what is currently happening on the microcontroller.

At the moment the logger has three different levels of verbosity which can be activated/deactivated separately at compile time:

1. Information - for general clear text messages
2. Warning - to signal events that require the developer's attention and shouldn't normally occur
3. Debug - to inspect the content of internal variables/data structures

3.3 Toolchain

The Quirky Lamp project takes advantage of the PlatformIO **ecosystem** [4]. PlatformIO is an open source product which provides a unified, highly extensible and fully automatized development environment for programming different microcontrollers. It abstracts the typical and very often cumbersome toolchain setup process that a developer must execute to start programming on a specific microcontroller. It solves this problem by providing ready configurations for popular chips and development boards. In addition to that, a library manager, cross-platform IDE, unit testing framework and unified debugger are available as separate components.

In our case, we use PlatformIO Core to install, configure and easily utilize the xtensa-gcc toolchain, which is the closed source ESP32 toolchain provided by the chip manufacturer. Furthermore the library manager is used to automatically install some of the project dependencies. For more information please refer to Section 5.

4 Further Work

The project has successfully fulfilled its initial requirements. However, there are some known issues and a lot of potential for further improvements.

The ESP32 analog-to-digital-converter (ADC) accuracy is not high and surprisingly varies on each device. The conversion function is neither linear nor monotonically increasing as it should be [11]. This implies a good correction algorithm for mapping the sensor ADC data to real values. The current implementation presents a rather bad one (the mapping function was experimentally determined and is valid only for certain value ranges).

As mentioned in Chapter 3.1 the MQTT connection to the broker is not encrypted. This can be further improved by adding/enabling TLS both device and server sides.

The collected lightning data is not very accurate since a photoresistor is used. This can be enhanced by utilizing a decent light sensor.

The Wiring Framework interface, which is used by this project, takes advantage only of a single CPU core of the ESP32 chip. The used LED strip drivers utilize of busy loops to synchronize the main controller with the strip's microcontroller. The busy waiting causes the SDK's watchdog to reboot the system when a hard WiFi-Stack deadline is missed (happens nondeterministically). Because of this the networking and sound lamp services (Section 3.2.1) conflict with one another. A temporarily implemented workaround is to disable the MQTT message publications when the lamp is active and disable the sound lamp service when MQTT messages must be distributed. A permanent solution to the problem is to swap out the used LED strip driver with one which synchronizes the communication asynchronously via interrupts. Taking advantage of the second CPU core is not possible when the Wiring Framework interface is used.

Instead of displaying the elapsed time since the system was booted on the LCD monitor, the correct time of day can be shown. This can be achieved by utilizing the Simple Network Time Protocol (SNTP).

5 Additional Materials

The additionally provided materials are:

- the firmware codebase (with references to the used third-party libraries)
- the electronics schematics design files
- the LaTeXcode for this report

and can be found on the project's git repository: https://gitlab.com/v45k0/quirky-lamp.

Pictures of the final system and demonstration videos are available on https://cloud.sarafov.net/s/pQDZIBtIGIHIl7m

References

[1] Comparison of IoT Data Protocol Overhead
Vasil Sarafov, Jan Seeger - Seminar Future Internet SS17, Technical University of Munich: http://bit.ly/2E8Mtjr

[2] MQTT Version 3.1.1, OASIS Standard, 29 October 2014
MQTT is a Client Server publish/subscribe messaging transport protocol. It is light weight, open, simple, and designed so as to be easy to implement

[3] Mosquitto
Eclipse Mosquitto™ is an open source (EPL/EDL licensed) message broker that implements the MQTT protocol versions 3.1 and 3.1.1.

[4] http://platformio.org/
An open source ecosystem for IoT development. Cross-platform IDE and unified debugger. Remote unit testing and firmware updates

[5] http://esp32.net/
A community driven collection of resources for the ESP32 chip

[6] http://kicad-pcb.org/
A Cross Platform and Open Source Electronics Design Automation Suite

[7] https://github.com/espressif/esp-idf/issues/164
ESP32 ADC accuracy issue

[8] https://github.com/espressif/esp-idf
Espressif IoT Development Framework. Official development framework for ESP32.

[9] http://wiring.org.co/reference/
Arduino core for the ESP32

[10] https://github.com/espressif/arduino-esp32
Wiring is an open-source programming framework for microcontrollers.

[11] https://github.com/espressif/esp-idf/issues/164
ESP32 analog-to-digital-converter accuracy issue