Lab 8 report

CSE121

Inhle Cele

Contents

Lab 8 Overview

- **Objective**: The goal of this report is to document the process of reimplementing the original Lab 1 (designed for C/C++ on an ESP32 with Ubuntu) into Rust. This includes setting up the environment, implementing a "Hello World" application, and creating a blinking LED program using Rust with the ESP-IDF framework.

Lab 8.1: setup Pi4

1. Environment Preparation

1.1 Install Rust Toolchain and ESP32 Support: The Rust toolchain and Espressif support were set up using the `espup` utility:

```
cargo install espup

espup install

source $HOME/export-esp.sh
```

1.2 Clone ESP-IDF Template for Rust: To streamline development, the ESP-RS ESP-IDF Template was used:

```
git clone https://github.com/esp-rs/esp-idf-template.git

cd esp-idf-template
```

1.3 Ensure Dependencies Are Installed: Packages necessary for ESP32 development were installed:

```
sudo apt update

sudo apt install build-essential cmake ninja-build libssl-dev
```

1.4 Verify ESP-IDF Installation: The ESP-IDF framework was installed and linked with the Rust toolchain.

Lab 8.2: Run hello world in ESP32

1. "Hello World" Implementation

The `src/main.rs` file was modified to print "Hello, World!" and my name:

```
fn main() {
    // Link runtime patches
    esp_idf_svc::sys::link_patches();

    // Initialize the ESP Logger
    esp_idf_svc::log::EspLogger::initialize_default();

    log::info!("Hello, world!");
    log::info!("Inhle Cele");
}
```

Output: When flashed to the ESP32, the serial monitor displayed the expected log messages.

Lab 8.3: Flash LED on ESP32

1. Blinking LED Implementation

A program was written to toggle the onboard LED connected to GPIO7:

```
use esp_idf_hal::prelude::*;
use esp_idf_hal::gpio::*;

fn main() {
    esp_idf_svc::sys::link_patches();
    esp_idf_svc::log::EspLogger::initialize_default();

    let peripherals = Peripherals::take().unwrap();
    let mut led = peripherals.pins.gpio7.into_output().unwrap();

    loop {
        led.set_high().unwrap();
        log::info!("LED ON");
        std::thread::sleep(std::time::Duration::from_secs(1));

        led.set_low().unwrap();
```

```
            log::info!("LED OFF");
            std::thread::sleep(std::time::Duration::from_secs(1));
        }
}
```

Output: The onboard LED blinked at a 1-second interval, with logs confirming the state changes.

Challenges and Resolutions

1. **Issue:** The espup installation stalled while downloading toolchains.
    - **Resolution:** Manually downloaded missing files and placed them in the appropriate directories.
2. **Issue:** Errors linking ESP-IDF libraries with Rust.
    - **Resolution:** Ensured that the export-esp.sh script was correctly sourced and all dependencies were installed.
3. **Issue:** GPIO pin access required proper peripheral handling.
    - **Resolution:** Utilized the esp-idf-hal crate for safe and idiomatic Rust GPIO control.

ChatGPT chat export:

https://chatgpt.com/share/674fda85-4be8-8008-b6aa-2324f5e2c026

This PDF includes all the context of this conversation except the formatted report.

Conversation Logs

User shared the following structure and actions performed:

1. Issues encountered while installing `espup` and setting up the Rust environment.

2. Discussions on how to duplicate the `lab8` folder.

3. The `main.rs` code content provided by the user for the ESP32 implementation using Rust:

```rust
fn main() {
    // It is necessary to call this function once. Otherwise some patches to the runtime
    // implemented by esp-idf-sys might not link properly. See https://github.com/esp-rs/esp-idf-template/issues/71
    esp_idf_svc::sys::link_patches();

    // Bind the log crate to the ESP Logging facilities
    esp_idf_svc::log::EspLogger::initialize_default();

    log::info!("Hello, world!");
    log::info!("Inhle Cele");
}
```

Discussions on challenges:

- The stalled downloading of `idf_tool_xtensa_elf_clang.libs.tar.xz`.

- Manual suggestions on using tools like `wget` and re-ensuring robust `export PATH`.

Conclusion

The environment was successfully set up for ESP32 development using Rust.

The "Hello World" and LED blinking programs were implemented and tested successfully on the ESP32.

The Rust-based implementation provided safer memory handling and better logging capabilities than the original C/C++ approach.