Isabella Phung
CSE 121
Apr 10, 2024

Lab 1 Report

1.1:

In order to connect the raspberry pi to eduroam, nmcli was used. I utilized a bash script written by Mark Zakharov which installed nmcli and a variety of other libraries via flash drive and is only compatible with ubuntu server versions 22.04.3 and 22.04.4. The message is on the class slack located at the following link:
https://cse121-s24.slack.com/archives/C06SG8JP6JD/p1712605507833189
Information on nmcli and how to connect to a 801-1x network can be found here:
https://www.networkmanager.dev/docs/api/latest/nmcli.html

1.2:

The hello world code was acquired from the examples provided in the esp-idf folder, which can be found here: https://github.com/espressif/esp-idf/tree/master/examples/get-started/hello_world

1.3:

The blink led code was acquired from chatGPT using the following prompt “Could you write an esp32c3 program in C with esp-idf to flash the onboard LED?” with some slight modifications to the gpio pin and the “gpio_pad_select_gpio(LED_GPIO);” line.
Information about the board, the selected LED GPIO pins, and schematics can be found here:
https://github.com/esp-rs/esp-rust-board
Here is the link to the chat:
https://chat.openai.com/share/1cd8abe7-875d-47d3-b97c-72b7aad94a0e

You

Could you write an esp32c3 program in C with esp-idf to flash the onboard LED?

ChatGPT

Certainly! To create an ESP32-C3 program in C using the ESP-IDF (Espressif IoT Development Framework) to flash the onboard LED, you'll need to follow these steps. First, ensure you have the ESP-IDF environment set up on your computer. If you haven't done so, follow the setup instructions available on the Espressif website.

The ESP32-C3 has an onboard LED typically connected to a specific GPIO. The exact GPIO can vary depending on the module or development board you're using, so you'll want to check your board's documentation. For this example, I'll assume the onboard LED is connected to GPIO 2, which is common in many boards.

Here's a simple program that flashes the onboard LED:

1. **Create a project directory:** Follow the ESP-IDF's instructions to create a new project. You can name it `led_flash`.
2. **Replace the content of `led_flash/main/led_flash.c` with the following code:**

```c
#include <stdio.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "driver/gpio.h"
#include "sdkconfig.h"

#define LED_GPIO 2 // Change this to the correct GPIO pin for the LED on your board

void app_main(void)
{
    // Configure the IOMUX register for pad LED_GPIO (some boards need this)
    gpio_pad_select_gpio(LED_GPIO);

    // Set the GPIO as a push/pull output
    gpio_set_direction(LED_GPIO, GPIO_MODE_OUTPUT);

    while(1) {
        // Turn the LED on (Note: depending on your LED circuit, you might need to us
        gpio_set_level(LED_GPIO, 1);
        vTaskDelay(1000 / portTICK_PERIOD_MS); // Wait for 1000ms

        // Turn the LED off
        gpio_set_level(LED_GPIO, 0);
        vTaskDelay(1000 / portTICK_PERIOD_MS); // Wait for 1000ms
```

```
    }
}
```

This program uses FreeRTOS tasks to create a simple loop. It turns the LED on, waits for a second, turns it off, and waits for another second. This cycle repeats indefinitely.

1. **Configure the project:** Run `idf.py menuconfig` from your project directory to configure your project. If necessary, adjust the serial port under `Serial flasher config`.
2. **Build and flash the project:** Use the following commands in your project directory:

- `idf.py build` to compile the project.
- `idf.py -p (YOUR_COM_PORT) flash` to flash it to your ESP32-C3. Replace `(YOUR_COM_PORT)` with the correct serial port for your system.

After flashing, the ESP32-C3 will restart and begin running the program, flashing the onboard LED on and off every second.

Remember, if your development board uses a different GPIO for the LED, you'll need to adjust the `LED_GPIO` definition in the code. ↓