Isabella Phung
4/25/2024
ECE103

Lab 2: I2C

2.1:

Compute 1st argument is 33 and is passed via register a0
Compute 2nd argument is 0 and is passed via register a1
Compute 3rd argument is 4 and is passed via register a2
Return value is 37 and is returned via register a0
The entry point of the "compute" function is at address 0x4200bb7e
The return point of the “compute” function is at address 0x4200bba6

Process, difficulties encountered, and various sources I consulted:
Flashing elf to gdb

- I asked chat-gpt to see if it could point me in the right direction after many different attempts searching on google. The chat is located here. It gave me a better idea to look into esptools write-format. Further documentation regarding this command was found here, but it didn’t provide much detail about the offset. There was a lot of trial and error with various values that I thought would work, but I only figured out the appropriate command thanks to a suggestion by Andrew Yegiayan on slack, who implied that looking at the build commands run by other projects would help. It was only after that, when I decided to try using the ninja command in the build folder with the .elf file located there to create a .bin file and then flashing it using this command “esptool.py write_flash 0x0 bootloader/bootloader.bin 0x10000 lab2.bin 0x8000 partition_table/partition-table.bin”, the values of which I got from when I flashed my hello_world project to my board.

Issues dealing with gdb:

- Couldn’t get openocd to work at first because it didn’t have access to the usb port. I had to use the temporary fix of using chmod to change write permissions as proposed by user JaeJun Lee.
- Described at start of this official openocd documentation, getting openOCD to work requires creating a file in the rules.d folder. I attempted to use this file at first which is intended for generic openOCD, but it didn’t work. After further research, I found this file provided by the official openOCD esp32 developers.
- After getting openOCD working, I was struggling to get gdb to work correctly. At first I got a warning that auto-loading was declined which would prevent me from using the .gdbinit file I created for this specific lab. While I have tried more permanent solutions where I’m supposed to edit a .gdbinit file located at the home directory or in a specific config folder, none of these solutions worked for me. The only solution that did make the warning message go away was including the option “-iex "set auto-load safe-path ." as mentioned in official gdb documentation. So my full command was “riscv32-esp-elf-gdb -iex "set auto-load safe-path /" -x .gdbinit build/lab2_1.elf”
- Even after fixing that, when I would run gdb, it would start up but never open up the (gdb) prompt, but this was fixed with one ctrl+c, which would interrupt the running program.

Other resources accessed:
[Learning about DIVU]
[Learning about REMU]
[Had trouble creating a breakpoint at a specific address with gdb]

2.2:
I utilized code from chatGPT from this chat. The only edits required was changing the gpio pins, adding in the sleep function, which was just copied and edited from the wake up command, and creating separate read functions for the humidity and temperature data. Information regarding the pinouts and commands could be found at the official esp32c3 github and the corresponding temperature sensor datasheet.
I've included images of the first chatGPT response in the appendix. While this chat includes additional questions I queried chatGPT with, none of them provided me with additional information that the first response didn't already provide.

Appendix:
2.1 ChatGPT:

You

I have a esp32c3 and a .elf file, how do I flash it to the board using idf.py?

ChatGPT

Flashing an `.elf` file to an ESP32-C3 board using the ESP-IDF's `idf.py` tool is quite straightforward. Here are the steps to follow:

Prerequisites

Make sure you have the following ready:

1. **ESP-IDF Environment Setup:** Ensure that the ESP-IDF (Espressif IoT Development Framework) is installed and set up on your computer. If not, you can follow the official installation guide from Espressif's ESP-IDF Programming Guide.
2. **Python:** The ESP-IDF tools are based on Python, so ensure Python is installed.
3. **USB Driver:** Make sure the USB driver for the ESP32-C3 board is installed if required (depending on your operating system).
4. **Connect Your ESP32-C3 Board:** Connect the ESP32-C3 to your computer via a USB cable.

Steps to Flash the `.elf` File

1. **Open a Command Prompt or Terminal:**

Open a command line interface (CLI) where the ESP-IDF tools are accessible. If ESP-IDF was set up correctly, you should have access to the `idf.py` script from the command line.

2. **Navigate to Your Project Directory:**

Change the directory to where your `.elf` file is located, or where your ESP-IDF project is located if the `.elf` file is part of an ESP-IDF project.

bash

```bash
cd path/to/your/project
```

3. **Set the Target (if not set):**

Before you flash, make sure to set the target to ESP32-C3 if it's not already set. This is crucial to ensure the build configurations and flashing are set for the correct chip.

bash

```bash
idf.py set-target esp32c3
```

4. **Flash the `.elf` File:**

Use the `idf.py` tool to flash the `.elf` file. Typically, you would use `idf.py flash` to build and flash your project. However, since you already have an `.elf` file, you need to specify the `.elf` file directly to the flash command using the `-e` flag or just use the `idf.py` command to flash the built binary:

```
idf.py -p PORT flash
```

Replace `PORT` with the COM port or device file where your ESP32-C3 is connected, such as `COM3` on Windows or `/dev/ttyUSB0` on Linux.Note: The `idf.py flash` command by default looks for the `.elf` and other necessary files in the `build` folder of an ESP-IDF project. If you need to flash a standalone `.elf` file that is not part of a build directory, you might have to use lower-level tools like `esptool.py` (included in ESP-IDF) for more direct control:

bash Copy code

```bash
esptool.py --chip esp32c3 -p PORT write_flash -z 0x10000 /path/to/your/file.elf
```

Again, replace `PORT` with the appropriate port and adjust the flash address (`0x10000` is a common starting point) according to where the `.elf` file needs to be written on the ESP32-C3.

Final Steps

After flashing, you may want to monitor the output from the ESP32-C3 to confirm it's running as expected:

bash Copy code

```bash
idf.py -p PORT monitor
```

This will open a serial connection to the ESP32-C3 and show you the output logs. To exit the monitor, type `Ctrl-]`.

That's it! Your `.elf` file should now be flashed to your ESP32-C3 board, and you can start seeing the output or debugging as needed.

2.2 ChatGPT:

You

Create an ESP-IDF program for ESP32C3 that reads temperature and humidity SHTC3 with address 0×70 and uses the checksum.

ChatGPT

To create an ESP-IDF program that reads temperature and humidity data from the SHTC3 sensor on an ESP32C3, you need to ensure you have the ESP-IDF development environment properly set up. You can download and install ESP-IDF from the Espressif website if you haven't done so already.

Here, I will provide a simple program using the ESP-IDF that reads data from the SHTC3 sensor, checks the checksum for data integrity, and prints the temperature and humidity values to the console. The SHTC3 communicates via the I2C protocol, and we'll set up the ESP32C3 to communicate with it.

Step 1: Setup I2C Driver

First, set up the I2C master driver. You need to configure the GPIO pins used for I2C, the I2C port, and speed.

c Copy code

```c
#include <stdio.h>
#include "freertos/FreeRTOS.h"
```

```c
#include "driver/i2c.h"
#include "esp_log.h"

static const char *TAG = "SHTC3";

#define I2C_MASTER_SCL_IO 10                /*!< GPIO number for I2C master clock */
#define I2C_MASTER_SDA_IO 9                 /*!< GPIO number for I2C master data  */
#define I2C_MASTER_NUM I2C_NUM_0            /*!< I2C port number for master dev */
#define I2C_MASTER_FREQ_HZ 400000           /*!< I2C master clock frequency */
#define I2C_MASTER_TX_BUF_DISABLE 0         /*!< I2C master doesn't need buffer */
#define I2C_MASTER_RX_BUF_DISABLE 0         /*!< I2C master doesn't need buffer */

#define SHTC3_SENSOR_ADDR 0x70              /*!< Slave address of the SHTC3 sensor */

/**
 * @brief i2c master initialization
 */
static esp_err_t i2c_master_init(void)
{
    int i2c_master_port = I2C_MASTER_NUM;
    i2c_config_t conf = {
        .mode = I2C_MODE_MASTER,
        .sda_io_num = I2C_MASTER_SDA_IO,
        .scl_io_num = I2C_MASTER_SCL_IO,
        .sda_pullup_en = GPIO_PULLUP_ENABLE,
```

```
        .sda_pullup_en = GPIO_PULLUP_ENABLE,
        .scl_pullup_en = GPIO_PULLUP_ENABLE,
        .master.clk_speed = I2C_MASTER_FREQ_HZ,
    };
    i2c_param_config(i2c_master_port, &conf);
    return i2c_driver_install(i2c_master_port, conf.mode,
                              I2C_MASTER_TX_BUF_DISABLE, I2C_MASTER_RX_BUF_DISABLE, 0
}
```

Step 2: Reading from the SHTC3

Now, implement the function to send commands to the SHTC3 and read the temperature and humidity data.

c Copy code

```c
#include "sdkconfig.h"
#include "string.h"

#define SHTC3_READ_COMMAND 0x7CA2  /*!< Command to read temperature and humidity */

/**
 * @brief Check CRC for SHTC3
 */
```

```c
static bool check_crc(uint16_t value, uint8_t received_crc) {
    uint32_t data = value;
    uint32_t crc = 0xFF;

    for (int i = 0; i < 16; i++) {
        crc ^= (data & 0x8000) ? 0x131 : 0x00;
        data <<= 1;
    }

    return (crc & 0xFF) == received_crc;
}

/**
 * @brief Read temperature and humidity from SHTC3
 */
static esp_err_t shtc3_read_temperature_humidity(float *temperature, float *humidity)
{
    uint8_t data[6];
    i2c_cmd_handle_t cmd = i2c_cmd_link_create();
    i2c_master_start(cmd);
    i2c_master_write_byte(cmd, (SHTC3_SENSOR_ADDR << 1) | I2C_MASTER_WRITE, true);
    i2c_master_write_byte(cmd, SHTC3_READ_COMMAND >> 8, true);
    i2c_master_write_byte(cmd, SHTC3_READ_COMMAND & 0xFF, true);
```

```
    i2c_master_start(cmd);
    i2c_master_write_byte(cmd, (SHTC3_SENSOR_ADDR << 1) | I2C_MASTER_READ, true);
    i2c_master_read(cmd, data, 6, I2C_MASTER_LAST_NACK);
    i2c_master_stop(cmd);
    esp_err_t ret = i2c_master_cmd_begin(I2C_MASTER_NUM, cmd, 1000 / portTICK_PERIOD_
    i2c_cmd_link_delete(cmd);

    if (ret == ESP_OK) {
        uint16_t temp_raw = (data[0] << 8) | data[1];
        uint8_t temp_crc = data[2];
        uint16_t hum_raw = (data[3] << 8) | data[4];
        uint8_t hum_crc = data[5];

        if (check_crc(temp_raw, temp_crc) && check_crc(hum_raw, hum_crc)) {
            *temperature = -45 + 175 * ((float) temp_raw / 65535);
            *humidity = 100 * ((float) hum_raw / 65535);
        } else {
            ESP_LOGE(TAG, "CRC check failed");
            ret = ESP_FAIL;
        }
    } else {
        ESP_LOGE(TAG, "Failed to read data from SHTC3 sensor");
    }
```

```
    return ret;
}
```

Step 3: Main Task

Finally, implement the main task to periodically read the sensor data.

c Copy code

```c
void app_main(void)
{
    ESP_ERROR_CHECK(i2c_master_init());

    float temperature = 0, humidity = 0;
    while (1) {
        if (shtc3_read_temperature_humidity(&temperature, &humidity) == ESP_OK) {
            ESP_LOGI(TAG, "Temperature: %.2f°C, Humidity: %.2f%%", temperature, humic
        } else {
            ESP_LOGE(TAG, "Failed to get sensor readings");
        }
        vTaskDelay(pdMS_TO_TICKS(2000)); // delay between reads
    }
}
```

This program initializes the I2C interface, reads temperature and humidity from the SHTC3 sensor, performs a CRC check, and prints the values to the ESP32C3's console. Adjust GPIO pin numbers as necessary for your specific hardware setup.