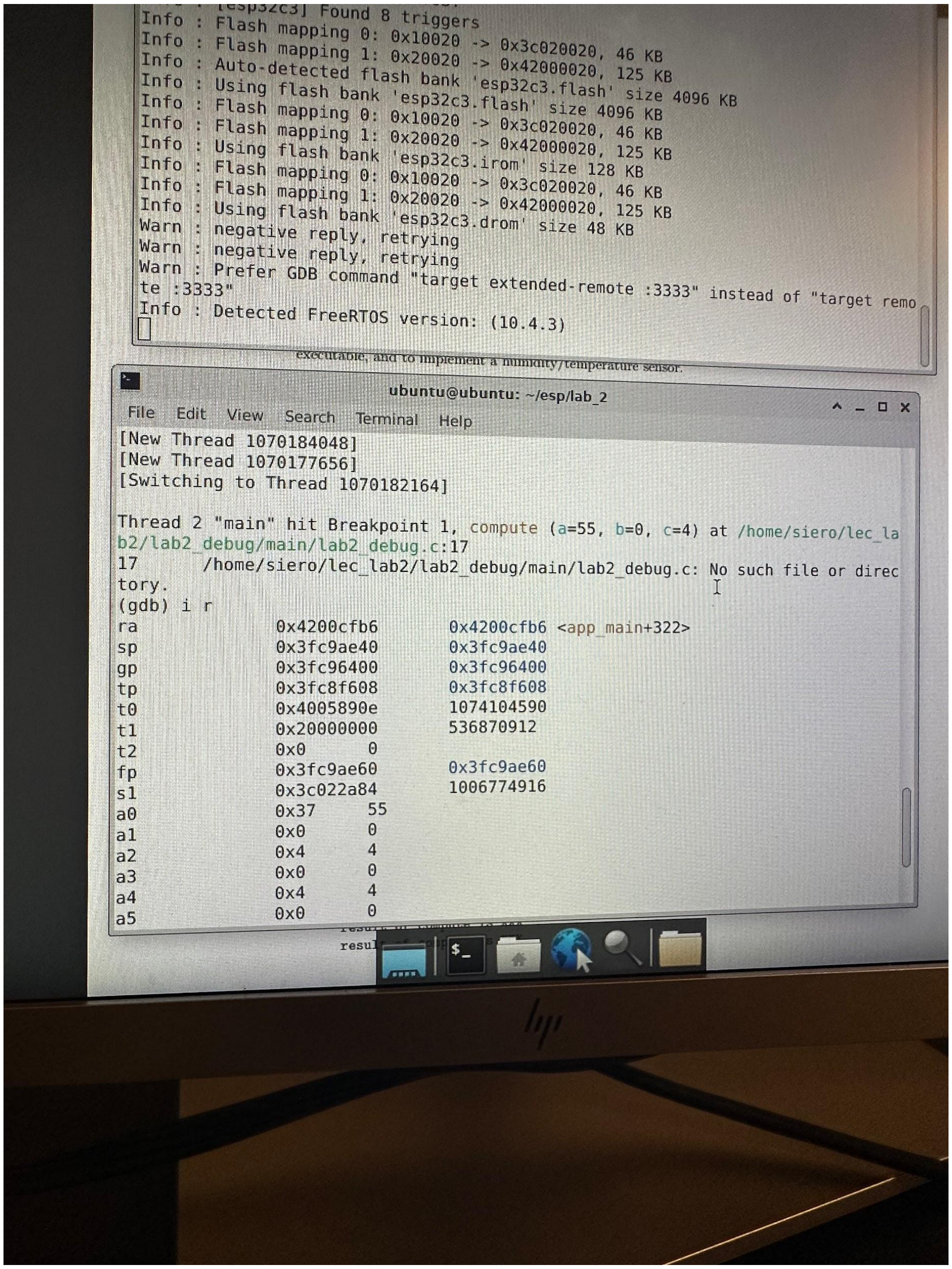
Jeffrey Yue

CSE121 Lab 2

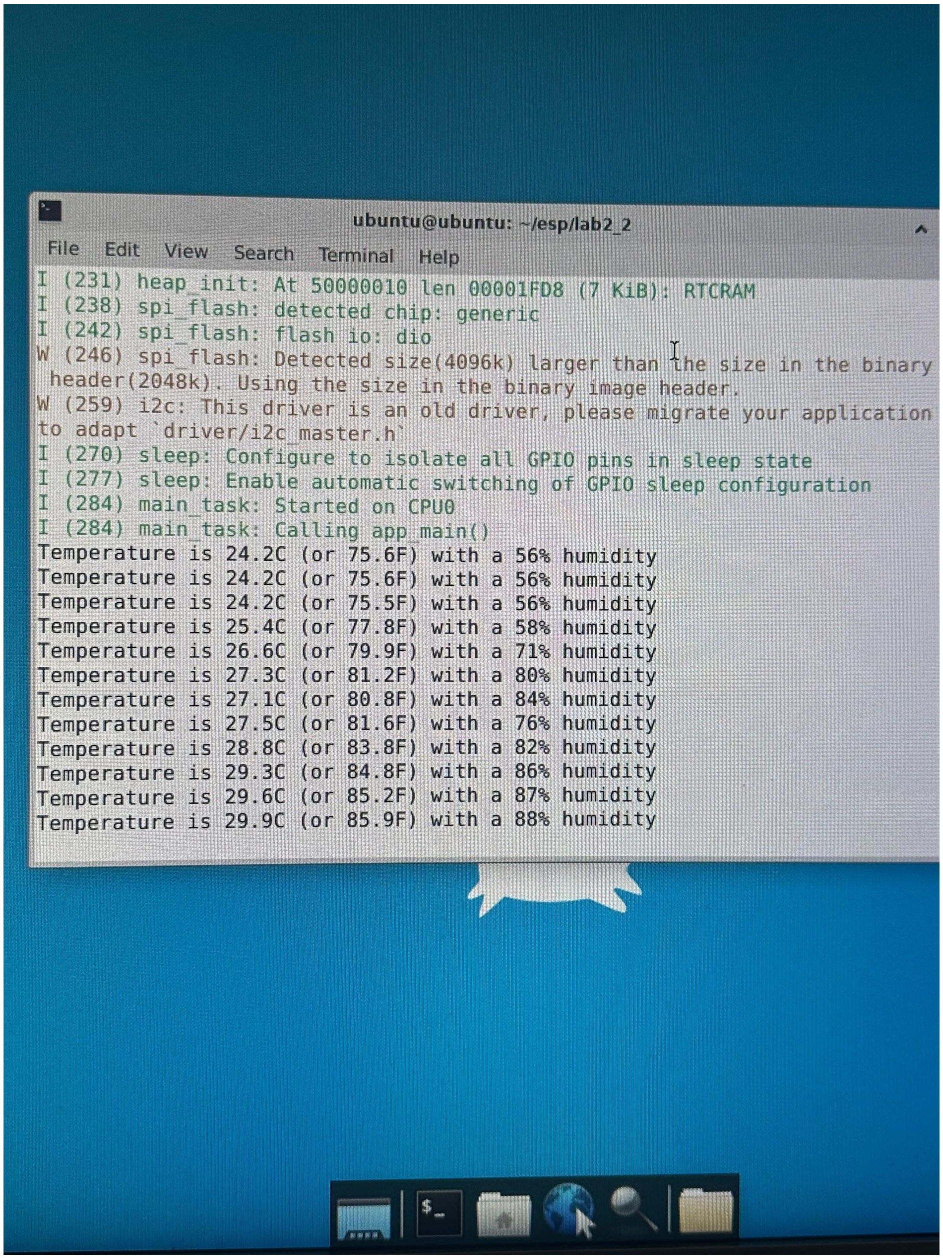
For part one of this lab, I used an esptool.py to convert an elf file to a bin file. Then I enforced the 60openocd rules found online. Running gdp, I used breakpoints to read the lines until finally it decoded the elf file to retrieve said values shown in the picture: a = 55, b = 0, and c = 4. The commands I used on gdp were “target remote :3333, b compute, c, and i r” to retrieve said values. Most of the process I looked up on ChatGPT or on the internet, shown below.



```
[esp32c3] Found 8 triggers
Info : Flash mapping 0: 0x10020 -> 0x3c020020, 46 KB
Info : Flash mapping 1: 0x20020 -> 0x42000020, 125 KB
Info : Auto-detected flash bank 'esp32c3.flash' size 4096 KB
Info : Using flash bank 'esp32c3.flash' size 4096 KB
Info : Flash mapping 0: 0x10020 -> 0x3c020020, 46 KB
Info : Flash mapping 1: 0x20020 -> 0x42000020, 125 KB
Info : Using flash bank 'esp32c3.irom' size 128 KB
Info : Flash mapping 0: 0x10020 -> 0x3c020020, 46 KB
Info : Flash mapping 1: 0x20020 -> 0x42000020, 125 KB
Info : Using flash bank 'esp32c3.drom' size 48 KB
Warn : negative reply, retrying
Warn : negative reply, retrying
Warn : Prefer GDB command "target extended-remote :3333" instead of "target remo
te :3333"
Info : Detected FreeRTOS version: (10.4.3)
```

```
[New Thread 1070184048]
[New Thread 1070177656]
[Switching to Thread 1070182164]

Thread 2 "main" hit Breakpoint 1, compute (a=55, b=0, c=4) at /home/siero/lec_la
b2/lab2_debug/main/lab2_debug.c:17
17      /home/siero/lec_lab2/lab2_debug/main/lab2_debug.c: No such file or direc
tory.
(gdb) i r
ra             0x4200cfb6       0x4200cfb6 <app_main+322>
sp             0x3fc9ae40       0x3fc9ae40
gp             0x3fc96400       0x3fc96400
tp             0x3fc8f608       0x3fc8f608
t0             0x4005890e       1074104590
t1             0x20000000       536870912
t2             0x0      0
fp             0x3fc9ae60       0x3fc9ae60
s1             0x3c022a84       1006774916
a0             0x37     55
a1             0x0      0
a2             0x4      4
a3             0x0      0
a4             0x4      4
a5             0x0      0
```

For part 2 of this lab, I used the i2c and shtc on the esp32c3 to detect the temperature and humidity of the environment, as shown in the picture below. Most of this was also looked up on ChatGPT or on the internet, all cited below.



```
I (231) heap_init: At 50000010 len 00001FD8 (7 KiB): RTCRAM
I (238) spi_flash: detected chip: generic
I (242) spi_flash: flash io: dio
W (246) spi_flash: Detected size(4096k) larger than the size in the binary
 header(2048k). Using the size in the binary image header.
W (259) i2c: This driver is an old driver, please migrate your application
to adapt `driver/i2c_master.h`
I (270) sleep: Configure to isolate all GPIO pins in sleep state
I (277) sleep: Enable automatic switching of GPIO sleep configuration
I (284) main_task: Started on CPU0
I (284) main_task: Calling app_main()
Temperature is 24.2C (or 75.6F) with a 56% humidity
Temperature is 24.2C (or 75.6F) with a 56% humidity
Temperature is 24.2C (or 75.5F) with a 56% humidity
Temperature is 25.4C (or 77.8F) with a 58% humidity
Temperature is 26.6C (or 79.9F) with a 71% humidity
Temperature is 27.3C (or 81.2F) with a 80% humidity
Temperature is 27.1C (or 80.8F) with a 84% humidity
Temperature is 27.5C (or 81.6F) with a 76% humidity
Temperature is 28.8C (or 83.8F) with a 82% humidity
Temperature is 29.3C (or 84.8F) with a 86% humidity
Temperature is 29.6C (or 85.2F) with a 87% humidity
Temperature is 29.9C (or 85.9F) with a 88% humidity
```

Resources:

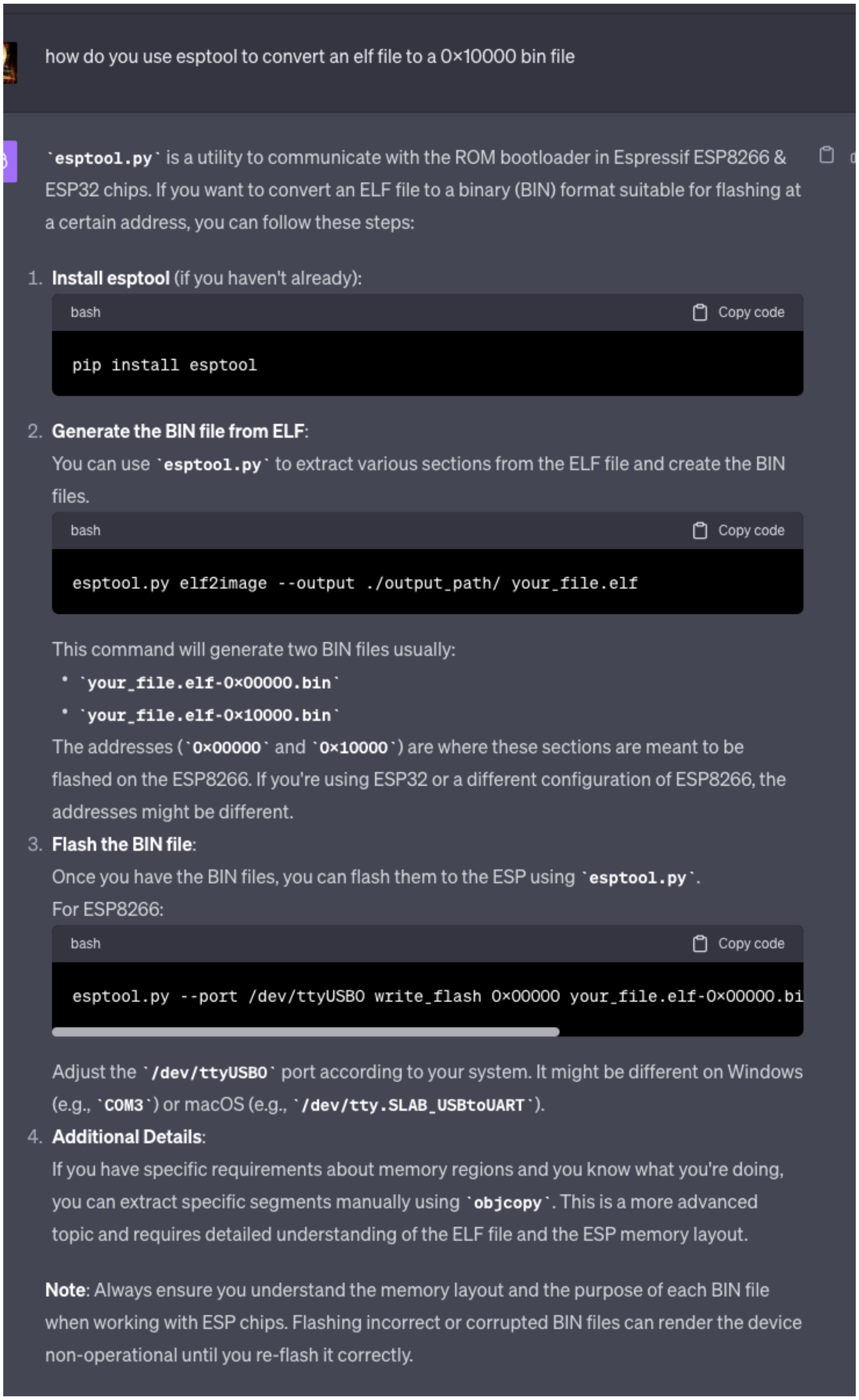
ChatGPT
chat.openai.com
https://docs.espressif.com/projects/esp-matter/en/latest/esp32/faq.html

APACHE license:
https://drive.google.com/drive/u/1/folders/15uqJ8zO42qNE-cszcmcdxg0Ajg6x94aC

OpenOCD Rules:
https://github.com/arduino/OpenOCD/blob/master/contrib/60-openocd.rules

Other:
https://esp32.com/viewtopic.php?t=3528
https://unix.stackexchange.com/questions/39370/how-to-reload-udev-rules-without-reboot
https://docs.espressif.com/projects/esp-idf/en/latest/esp32c3/api-guides/jtag-debugging/configure-builtin-jtag.html
https://stackoverflow.com/questions/41066653/openocd-error-libusb-open-failed-with-libusb-error-not-supported
https://stackoverflow.com/questions/72616674/error-libusb-open-failed-with-libusb-error-access-visual-studio-code

Chatgpt resources:



how do you use esptool to convert an elf file to a 0×10000 bin file

`esptool.py` is a utility to communicate with the ROM bootloader in Espressif ESP8266 & ESP32 chips. If you want to convert an ELF file to a binary (BIN) format suitable for flashing at a certain address, you can follow these steps:

1. **Install esptool** (if you haven't already):

```bash
pip install esptool
```

2. **Generate the BIN file from ELF**:
   You can use `esptool.py` to extract various sections from the ELF file and create the BIN files.

```bash
esptool.py elf2image --output ./output_path/ your_file.elf
```

   This command will generate two BIN files usually:
   - `your_file.elf-0×00000.bin`
   - `your_file.elf-0×10000.bin`

   The addresses (`0×00000` and `0×10000`) are where these sections are meant to be flashed on the ESP8266. If you're using ESP32 or a different configuration of ESP8266, the addresses might be different.

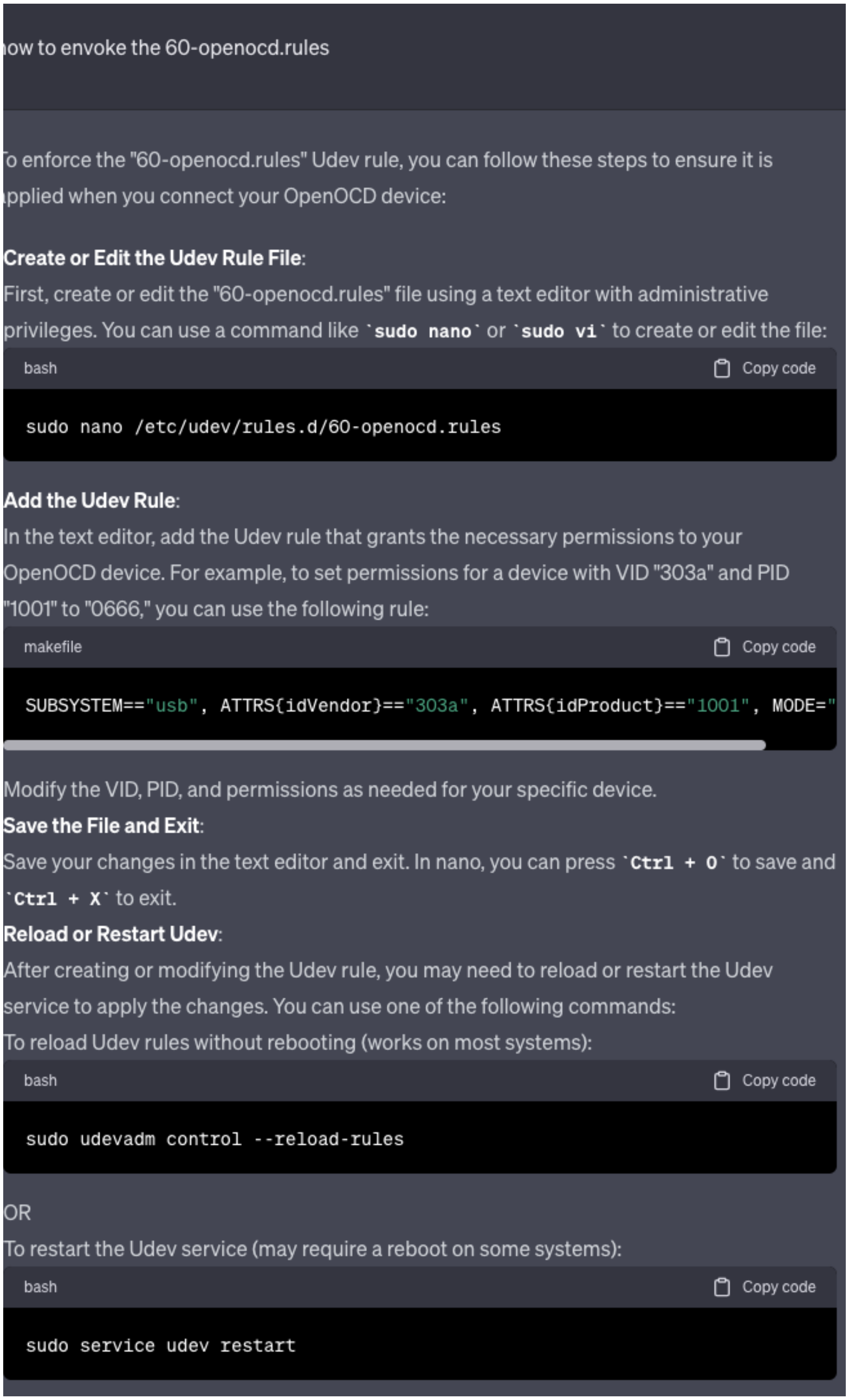
3. **Flash the BIN file**:
   Once you have the BIN files, you can flash them to the ESP using `esptool.py`.
   For ESP8266:

```bash
esptool.py --port /dev/ttyUSB0 write_flash 0×00000 your_file.elf-0×00000.bi
```

   Adjust the `/dev/ttyUSB0` port according to your system. It might be different on Windows (e.g., `COM3`) or macOS (e.g., `/dev/tty.SLAB_USBtoUART`).

4. **Additional Details**:
   If you have specific requirements about memory regions and you know what you're doing, you can extract specific segments manually using `objcopy`. This is a more advanced topic and requires detailed understanding of the ELF file and the ESP memory layout.

Note: Always ensure you understand the memory layout and the purpose of each BIN file when working with ESP chips. Flashing incorrect or corrupted BIN files can render the device non-operational until you re-flash it correctly.



ow to envoke the 60-openocd.rules

o enforce the "60-openocd.rules" Udev rule, you can follow these steps to ensure it is pplied when you connect your OpenOCD device:

Create or Edit the Udev Rule File:
First, create or edit the "60-openocd.rules" file using a text editor with administrative privileges. You can use a command like `sudo nano` or `sudo vi` to create or edit the file:

```bash
sudo nano /etc/udev/rules.d/60-openocd.rules
```

Add the Udev Rule:
In the text editor, add the Udev rule that grants the necessary permissions to your OpenOCD device. For example, to set permissions for a device with VID "303a" and PID "1001" to "0666," you can use the following rule:

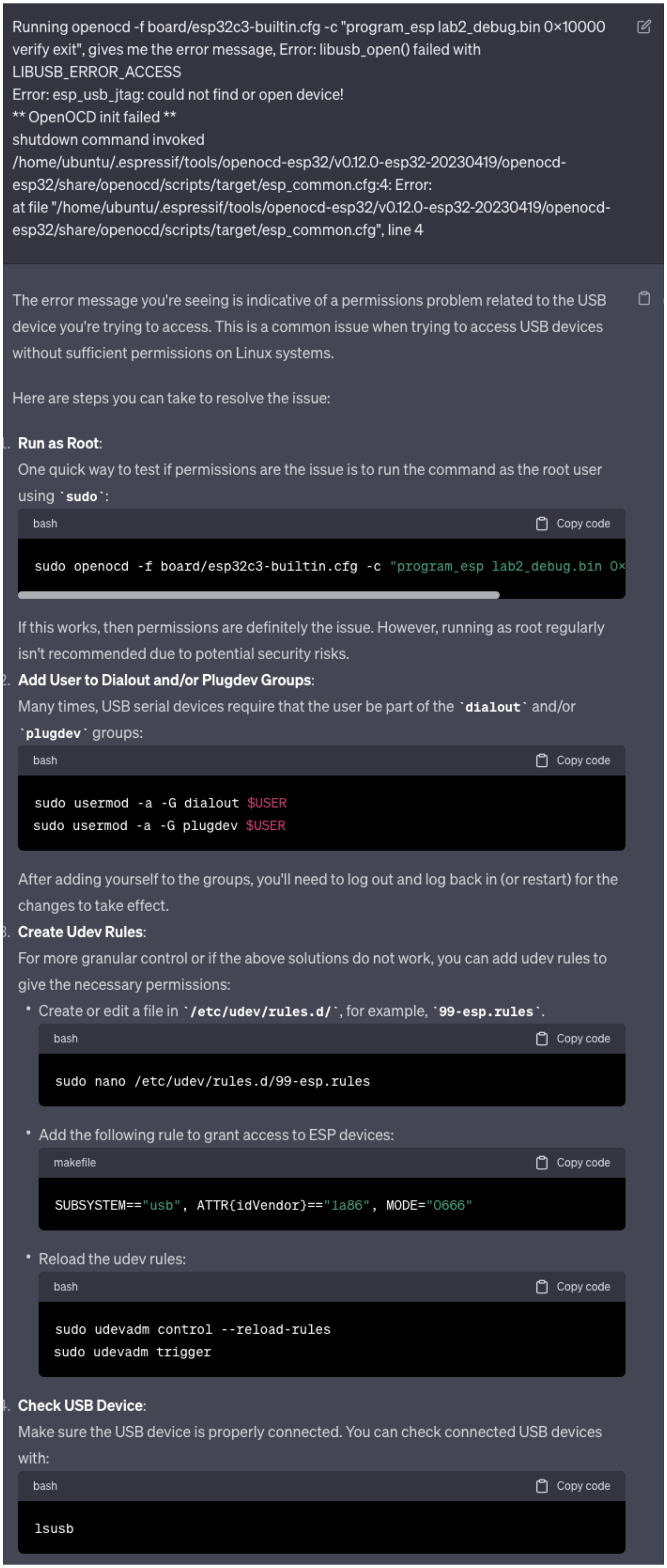
```makefile
SUBSYSTEM=="usb", ATTRS{idVendor}=="303a", ATTRS{idProduct}=="1001", MODE="
```

Modify the VID, PID, and permissions as needed for your specific device.
Save the File and Exit:
Save your changes in the text editor and exit. In nano, you can press `Ctrl + O` to save and `Ctrl + X` to exit.
Reload or Restart Udev:
After creating or modifying the Udev rule, you may need to reload or restart the Udev service to apply the changes. You can use one of the following commands:
To reload Udev rules without rebooting (works on most systems):

```bash
sudo udevadm control --reload-rules
```

OR
To restart the Udev service (may require a reboot on some systems):

```bash
sudo service udev restart
```



Running openocd -f board/esp32c3-builtin.cfg -c "program_esp lab2_debug.bin 0x10000 verify exit", gives me the error message, Error: libusb_open() failed with LIBUSB_ERROR_ACCESS
Error: esp_usb_jtag: could not find or open device!
** OpenOCD init failed **
shutdown command invoked
/home/ubuntu/.espressif/tools/openocd-esp32/v0.12.0-esp32-20230419/openocd-esp32/share/openocd/scripts/target/esp_common.cfg:4: Error:
at file "/home/ubuntu/.espressif/tools/openocd-esp32/v0.12.0-esp32-20230419/openocd-esp32/share/openocd/scripts/target/esp_common.cfg", line 4

The error message you're seeing is indicative of a permissions problem related to the USB device you're trying to access. This is a common issue when trying to access USB devices without sufficient permissions on Linux systems.

Here are steps you can take to resolve the issue:

1. **Run as Root**:
   One quick way to test if permissions are the issue is to run the command as the root user using `sudo`:

   ```bash
   sudo openocd -f board/esp32c3-builtin.cfg -c "program_esp lab2_debug.bin 0x
   ```

   If this works, then permissions are definitely the issue. However, running as root regularly isn't recommended due to potential security risks.
2. **Add User to Dialout and/or Plugdev Groups**:
   Many times, USB serial devices require that the user be part of the `dialout` and/or `plugdev` groups:

   ```bash
   sudo usermod -a -G dialout $USER
   sudo usermod -a -G plugdev $USER
   ```

   After adding yourself to the groups, you'll need to log out and log back in (or restart) for the changes to take effect.
3. **Create Udev Rules**:
   For more granular control or if the above solutions do not work, you can add udev rules to give the necessary permissions:
   - Create or edit a file in `/etc/udev/rules.d/`, for example, `99-esp.rules`.

     ```bash
     sudo nano /etc/udev/rules.d/99-esp.rules
     ```

   - Add the following rule to grant access to ESP devices:

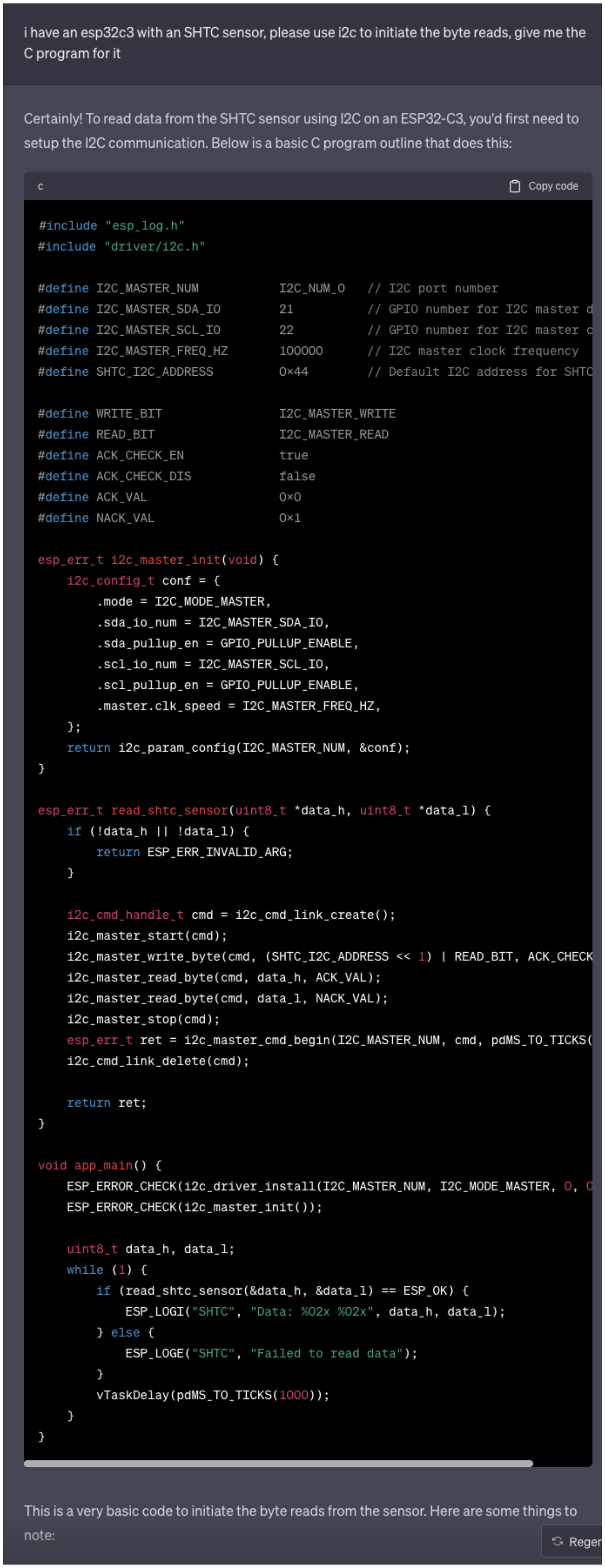
     ```makefile
     SUBSYSTEM=="usb", ATTR{idVendor}=="1a86", MODE="0666"
     ```

   - Reload the udev rules:

     ```bash
     sudo udevadm control --reload-rules
     sudo udevadm trigger
     ```

4. **Check USB Device**:
   Make sure the USB device is properly connected. You can check connected USB devices with:

   ```bash
   lsusb
   ```

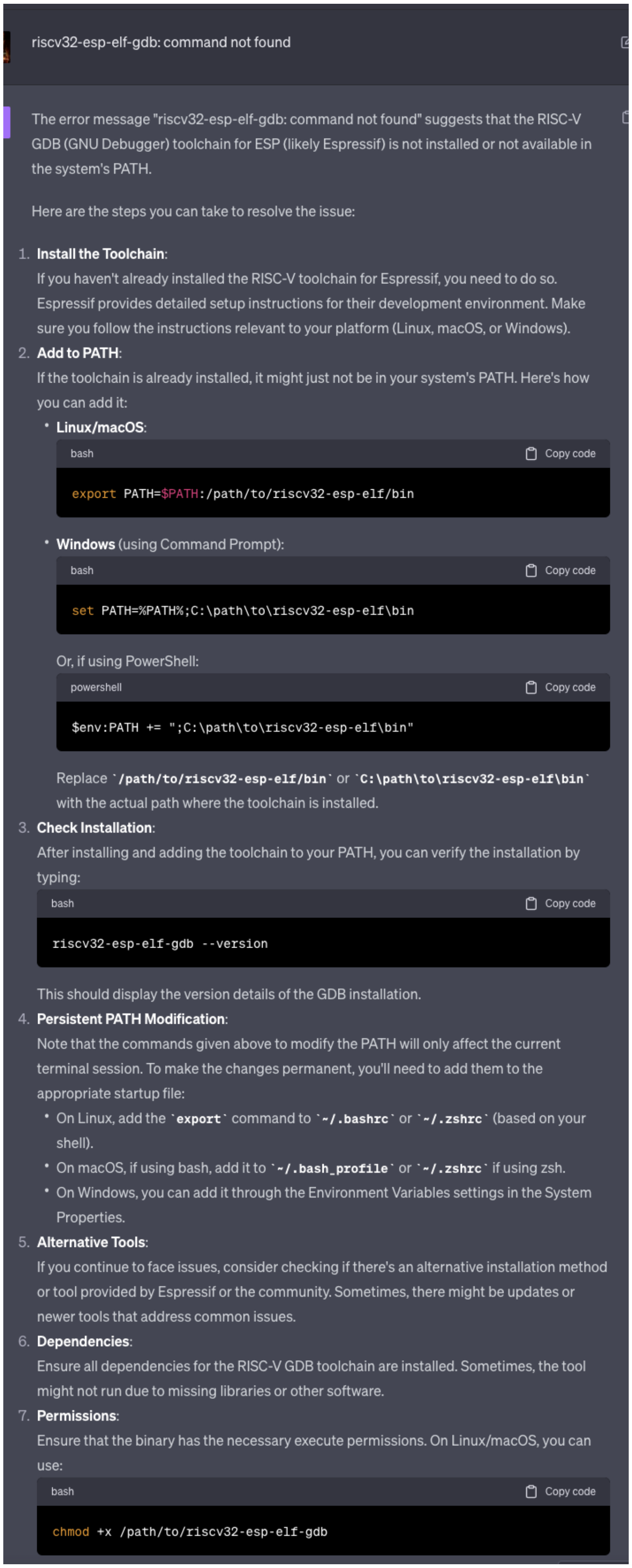


i have an esp32c3 with an SHTC sensor, please use i2c to initiate the byte reads, give me the C program for it

Certainly! To read data from the SHTC sensor using I2C on an ESP32-C3, you'd first need to setup the I2C communication. Below is a basic C program outline that does this:

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

#define I2C_MASTER_NUM          I2C_NUM_0   // I2C port number
#define I2C_MASTER_SDA_IO       21          // GPIO number for I2C master d
#define I2C_MASTER_SCL_IO       22          // GPIO number for I2C master c
#define I2C_MASTER_FREQ_HZ      100000      // I2C master clock frequency
#define SHTC_I2C_ADDRESS        0x44        // Default I2C address for SHTC

#define WRITE_BIT               I2C_MASTER_WRITE
#define READ_BIT                I2C_MASTER_READ
#define ACK_CHECK_EN            true
#define ACK_CHECK_DIS           false
#define ACK_VAL                 0x0
#define NACK_VAL                0x1

esp_err_t i2c_master_init(void) {
    i2c_config_t conf = {
        .mode = I2C_MODE_MASTER,
        .sda_io_num = I2C_MASTER_SDA_IO,
        .sda_pullup_en = GPIO_PULLUP_ENABLE,
        .scl_io_num = I2C_MASTER_SCL_IO,
        .scl_pullup_en = GPIO_PULLUP_ENABLE,
        .master.clk_speed = I2C_MASTER_FREQ_HZ,
    };
    return i2c_param_config(I2C_MASTER_NUM, &conf);
}

esp_err_t read_shtc_sensor(uint8_t *data_h, uint8_t *data_l) {
    if (!data_h || !data_l) {
        return ESP_ERR_INVALID_ARG;
    }

    i2c_cmd_handle_t cmd = i2c_cmd_link_create();
    i2c_master_start(cmd);
    i2c_master_write_byte(cmd, (SHTC_I2C_ADDRESS << 1) | READ_BIT, ACK_CHECK
    i2c_master_read_byte(cmd, data_h, ACK_VAL);
    i2c_master_read_byte(cmd, data_l, NACK_VAL);
    i2c_master_stop(cmd);
    esp_err_t ret = i2c_master_cmd_begin(I2C_MASTER_NUM, cmd, pdMS_TO_TICKS(
    i2c_cmd_link_delete(cmd);

    return ret;
}

void app_main() {
    ESP_ERROR_CHECK(i2c_driver_install(I2C_MASTER_NUM, I2C_MODE_MASTER, 0, 0
    ESP_ERROR_CHECK(i2c_master_init());

    uint8_t data_h, data_l;
    while (1) {
        if (read_shtc_sensor(&data_h, &data_l) == ESP_OK) {
            ESP_LOGI("SHTC", "Data: %02x %02x", data_h, data_l);
        } else {
            ESP_LOGE("SHTC", "Failed to read data");
        }
        vTaskDelay(pdMS_TO_TICKS(1000));
    }
}
```

This is a very basic code to initiate the byte reads from the sensor. Here are some things to note:

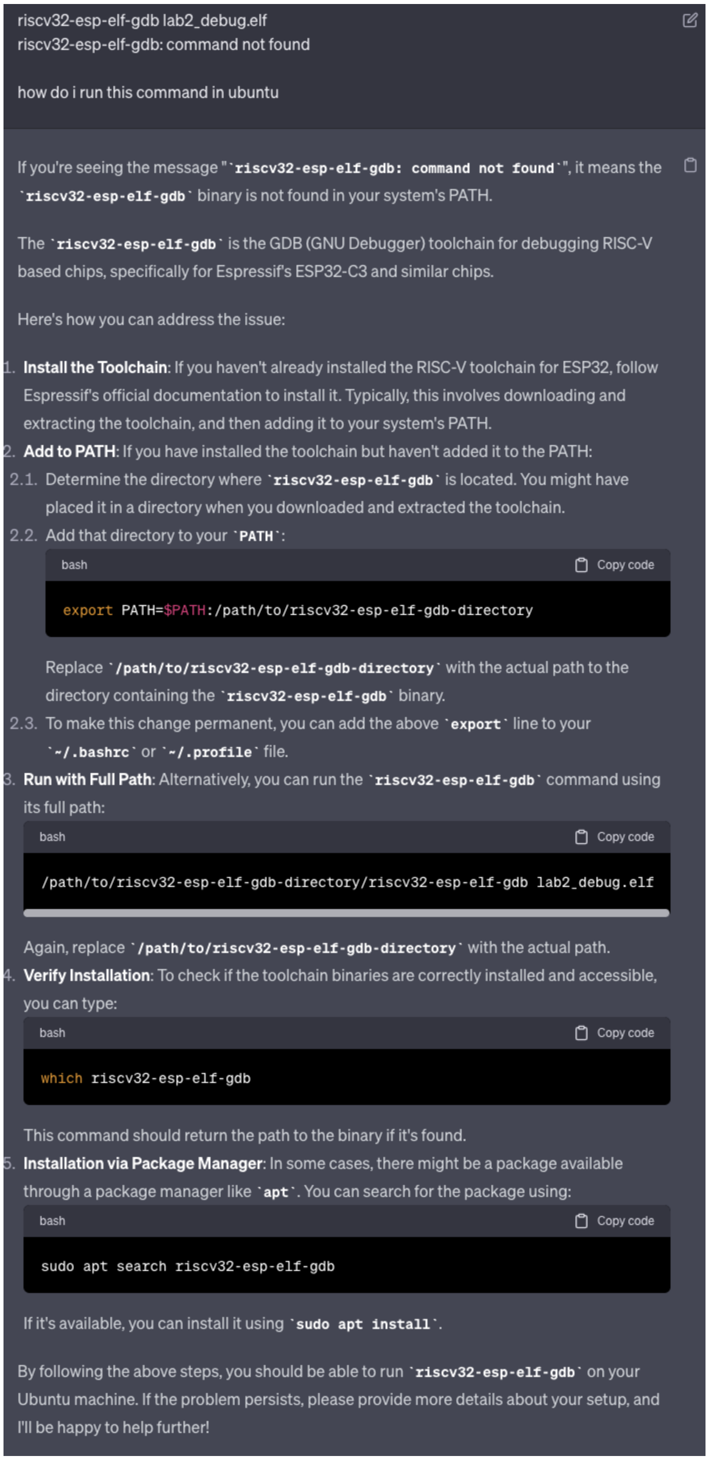


riscv32-esp-elf-gdb: command not found

The error message "riscv32-esp-elf-gdb: command not found" suggests that the RISC-V GDB (GNU Debugger) toolchain for ESP (likely Espressif) is not installed or not available in the system's PATH.

Here are the steps you can take to resolve the issue:

1. **Install the Toolchain**:
   If you haven't already installed the RISC-V toolchain for Espressif, you need to do so. Espressif provides detailed setup instructions for their development environment. Make sure you follow the instructions relevant to your platform (Linux, macOS, or Windows).
2. **Add to PATH**:
   If the toolchain is already installed, it might just not be in your system's PATH. Here's how you can add it:
   - **Linux/macOS**:
     ```bash
     export PATH=$PATH:/path/to/riscv32-esp-elf/bin
     ```
   - **Windows** (using Command Prompt):
     ```bash
     set PATH=%PATH%;C:\path\to\riscv32-esp-elf\bin
     ```
     Or, if using PowerShell:
     ```powershell
     $env:PATH += ";C:\path\to\riscv32-esp-elf\bin"
     ```
     Replace `/path/to/riscv32-esp-elf/bin` or `C:\path\to\riscv32-esp-elf\bin` with the actual path where the toolchain is installed.
3. **Check Installation**:
   After installing and adding the toolchain to your PATH, you can verify the installation by typing:
   ```bash
   riscv32-esp-elf-gdb --version
   ```
   This should display the version details of the GDB installation.
4. **Persistent PATH Modification**:
   Note that the commands given above to modify the PATH will only affect the current terminal session. To make the changes permanent, you'll need to add them to the appropriate startup file:
   - On Linux, add the `export` command to `~/.bashrc` or `~/.zshrc` (based on your shell).
   - On macOS, if using bash, add it to `~/.bash_profile` or `~/.zshrc` if using zsh.
   - On Windows, you can add it through the Environment Variables settings in the System Properties.
5. **Alternative Tools**:
   If you continue to face issues, consider checking if there's an alternative installation method or tool provided by Espressif or the community. Sometimes, there might be updates or newer tools that address common issues.
6. **Dependencies**:
   Ensure all dependencies for the RISC-V GDB toolchain are installed. Sometimes, the tool might not run due to missing libraries or other software.
7. **Permissions**:
   Ensure that the binary has the necessary execute permissions. On Linux/macOS, you can use:
   ```bash
   chmod +x /path/to/riscv32-esp-elf-gdb
   ```



riscv32-esp-elf-gdb lab2_debug.elf
riscv32-esp-elf-gdb: command not found

how do i run this command in ubuntu

If you're seeing the message "`riscv32-esp-elf-gdb: command not found`", it means the `riscv32-esp-elf-gdb` binary is not found in your system's PATH.

The `riscv32-esp-elf-gdb` is the GDB (GNU Debugger) toolchain for debugging RISC-V based chips, specifically for Espressif's ESP32-C3 and similar chips.

Here's how you can address the issue:

1. **Install the Toolchain**: If you haven't already installed the RISC-V toolchain for ESP32, follow Espressif's official documentation to install it. Typically, this involves downloading and extracting the toolchain, and then adding it to your system's PATH.
2. **Add to PATH**: If you have installed the toolchain but haven't added it to the PATH:
   2.1. Determine the directory where `riscv32-esp-elf-gdb` is located. You might have placed it in a directory when you downloaded and extracted the toolchain.
   2.2. Add that directory to your `PATH`:

   ```bash
   export PATH=$PATH:/path/to/riscv32-esp-elf-gdb-directory
   ```

   Replace `/path/to/riscv32-esp-elf-gdb-directory` with the actual path to the directory containing the `riscv32-esp-elf-gdb` binary.
   2.3. To make this change permanent, you can add the above `export` line to your `~/.bashrc` or `~/.profile` file.
3. **Run with Full Path**: Alternatively, you can run the `riscv32-esp-elf-gdb` command using its full path:

   ```bash
   /path/to/riscv32-esp-elf-gdb-directory/riscv32-esp-elf-gdb lab2_debug.elf
   ```

   Again, replace `/path/to/riscv32-esp-elf-gdb-directory` with the actual path.
4. **Verify Installation**: To check if the toolchain binaries are correctly installed and accessible, you can type:

   ```bash
   which riscv32-esp-elf-gdb
   ```

   This command should return the path to the binary if it's found.
5. **Installation via Package Manager**: In some cases, there might be a package available through a package manager like `apt`. You can search for the package using:

   ```bash
   sudo apt search riscv32-esp-elf-gdb
   ```

   If it's available, you can install it using `sudo apt install`.

By following the above steps, you should be able to run `riscv32-esp-elf-gdb` on your Ubuntu machine. If the problem persists, please provide more details about your setup, and I'll be happy to help further!