

CSE 121: Lab 2 Report

By Daniel Beltran

10/20/2023

Lab 2: Part 1

The first portion of the lab proved to be quite challenging as utilizing GDB without source code was new for myself. The first instance was creating a project directory and downloading the .elf file, provided by the staff. From the lab2_debug.elf file, I was able upload the image and grab the .bin file generated by the following command:

Bin Source:

<https://docs.espressif.com/projects/esptool/en/latest/esp32c3/esptool/basic-commands.html>

From this, the next step was to flash the files and run GDB. Unfortunately, it wasn't that simple as using the `idf.py flash` will effectively overwrite all files within the build directory. What does this mean and how does the setup work? Essentially, I would have to find another way to flash my files and along with making sure the initial setup (e.g. when using `./install.sh esp32c3`, `export.sh`, `idf.py set-target esp32c3`) as it tended to also overwrite the build directory. Each initial setup, I would have to remove the binary and elf files from the build directory to prevent any deletion. In order bypass this flashing overwritting, I used this command:

```
openocd -f board/esp32c3-built-in.cfg -c "program_esp filename.bin 0x10000 verify exit"
```

Flash Source:

<https://docs.espressif.com/projects/esp-idf/en/latest/esp32c3/api-guides/jtag-debugging/index.html>

However, before we can even flash there must be a .gdbinit file in place so that we can connect to port:3333 when calling on idf.py openocd. Straight from the lab documentation, these following lines of code will be necessary initiaing a client/server connection (e.g. board to gdb).

Below will show the lines added onto the .gdbinit file:

```
target remote :3333
set remote hardware-watchpoint-limit 2
mon reset halt
flushregs
b app_main
c
```

Another step that must be taken before getting into GDB is setting up the udev rules, pointed out by the staff. Essentially, I had to add these set of rules to properly setup my JTAG. The following directory is where the udev rules were added onto:

`/etc/udev/rules.d`

Udev Source:

<https://docs.espressif.com/projects/esp-idf/en/latest/esp32c3/api-guides/jtag-debugging/configure-built-in-jtag.html>

After adding these set of rules it was time to start the debugging process. One thing that held me back many times was the fact that two terminals were needed to: flash the bin file, establish the client/server connection, and run the RISCV command (riscv32-esp-elf-gdb -x gdbinit build/lab2_debug.elf, given by the professor).

```
ubuntu@ubuntu:~/esp/lab2_1$ idf.py openocd
Executing action: openocd
Note: OpenOCD cfg not found (via env variable OPENOCD_COMMANDS nor as a --openocd-commands argument)
OpenOCD arguments default to: "-f board/esp32c3-builtin.cfg"
OpenOCD started as a background task 21859
Executing action: post_debug
Open On-Chip Debugger v0.12.0-esp32-20230419 (2023-04-18-22:02)
Licensed under GNU GPL v2
For bug reports, read
    http://openocd.org/doc/doxygen/bugs.html
Info : only one transport option; autoselecting 'jtag'
Info : esp_usb_jtag: VID set to 0x303a and PID to 0x1001
Info : esp_usb_jtag: capabilities descriptor set to 0x2000
Warn : Transport "jtag" was already selected
Info : Listening on port 6666 for tcl connections
Info : Listening on port 4444 for telnet connections
Info : esp_usb_jtag: serial (34:85:18:02:36:78)
Info : esp_usb_jtag: Device found. Base speed 40000KHz, div range 1 to 255
Info : clock speed 40000 kHz
Info : JTAG tap: esp32c3.cpu tap/device found: 0x00005c25 (mfg: 0x612 (Espressif Systems), part: 0x0005, ver: 0x0)
Info : datacount=2 progbufsize=16
Info : Examined RISC-V core; found 1 harts
Info : hart 0: XLEN=32, misa=0x40101104
Info : starting gdb server for esp32c3 on 3333
Info : Listening on port 3333 for gdb connections
Info : accepting 'adb' connection
```

```

ubuntu@ubuntu: ~/esp/lab2_1/build$ cd /home/ubuntu/esp/lab2_1/build
/home/ubuntu/esp/lab2_1.bin binary size 0x2a2a0 bytes. Smallest app partition is
ubuntu@ubuntu:~/esp/lab2_1/build$ openocd -f board/esp32c3-build
0 verify exit"
Open On-Chip Debugger v0.12.0-esp32-20230419 (2023-04-18-22:02)
Licensed under GNU GPL v2
For bug reports, read
    http://openocd.org/doc/doxygen/bugs.html
Info : only one transport option; autoselecting 'jtag'
Info : esp_usb_jtag: VID set to 0x303a and PID to 0x1001
Info : esp_usb_jtag: capabilities descriptor set to 0x2000
Warn : Transport "jtag" was already selected
** program_esp input args <0x10000 verify exit> **
Info : esp_usb_jtag: serial (34:85:18:02:36:78)
Info : esp_usb_jtag: Device found. Base speed 40000KHz, div range 1
Info : clock speed 40000 KHz
Info : JTAG tap: esp32c3.cpu tap/device found: 0x00005c25 (mfg: 0x612
r: 0x0)
Info : datacount=2 progbufsize=16
Info : Examined RISC-V core; found 1 harts
Info : hart 0: XLEN=32, misa=0x40101104
Info : starting gdb server for esp32c3 on I3333
Info : Listening on port 3333 for gdb connections
Info : JTAG tap: esp32c3.cpu tap/device found: 0x00005c25 (mfg: 0x612 (E
r: 0x0)
Info : Reset cause (3) - (Software core reset)
Info : [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
** Programming Started **
Info : PROF: Erased 270336 bytes in 2004.37 ms
Info : PROF: Data transferred in 4173.32 ms @ 63.259 KB/s
Info : PROF: Wrote 270336 bytes in 4753.71 ms (data transfer time included)
** Programming Finished in 7971 ms **
** Verify Started **
Info : PROF: Flash verified in 458.586 ms
** Verify OK **
shutdown command invoked
ubuntu@ubuntu:~/esp/lab2_1/build$ cd ..
cd: command not found
ubuntu@ubuntu:~/esp/lab2_1/build$ cd ..
GNU gdb (esp32_gdb) 12.1-20221002
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

```

The pictures above show two different terminals that provide us with the client/server connection and running of GDB. Once into GDB it was quite confused until I had read on the different commands that can be used. The command “info registers” or “i r” showed a detailed description of all the registers within and what caught my attention was the compute function. However, that was merely to assist in displaying the registers and printing out their addresses, but how would we return inputs from a function? The use of “set args [arguemnts]” assisted in providing an

input to the compute function. Taking in the values, given in the lab documentation (502, 303, 404), I was able to successfully input the following commands:

```
set args 502 303 404
```

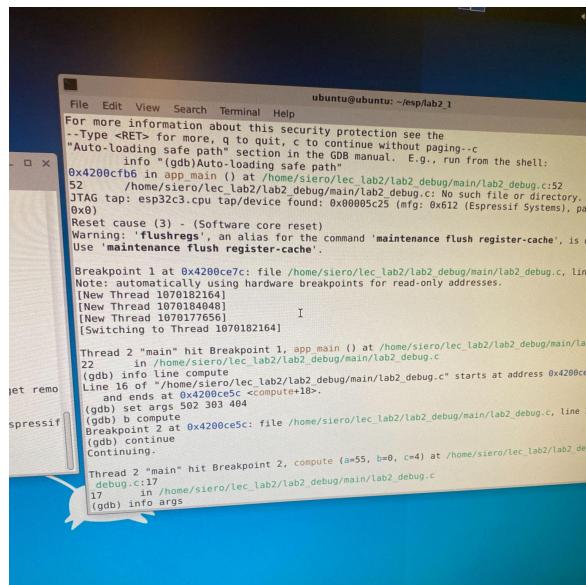
```
b compute
```

```
Continue
```

```
info args
```

```
finish
```

These set of commands gives us the outputs of $a = 55(0x37)$, $b = 0(0x0)$, and $c = 4(0x4)$. These values correspond to the integer inputs of the function compute. The list of commands are done this way since we are only provided an .elf file. The return address is 0x4200ce72 of the return value 59. The entry of the computer function is 00x4200ce5c. Below will be a list of photos detailing what is mentioned, previously.



The screenshot shows a terminal window titled "ubuntu@ubuntu: ~/esp/fab2_1". The terminal displays the following GDB session:

```
File Edit View Search Terminal Help
For more information about this security protection see the
--Type <RET> for more, q to quit, c to continue without paging--c
"Auto-loading safe path" section in the GDB manual. E.g., run from the shell:
info "(gdb)Auto-loading safe path"
52 /home/siero/lec_lab2/lab2_debug/main/lab2_debug.c:52
JTAG tap: esp32c3.cpu tap/device found: 0x00005c25 (mfg: 0x612 (Espressif Systems), par
0x0)
Reset cause (3) - (Software core reset)
Warning: 'flushregs', an alias for the command 'maintenance flush register-cache', is de
Use 'maintenance flush register-cache'.
Breakpoint 1 at 0x4200ce7c: file '/home/siero/lec_lab2/lab2_debug/main/lab2_debug.c', line
Note: automatically using hardware breakpoints for read-only addresses.
[New Thread 1070182164]
[New Thread 1070184048]
[New Thread 1070177656]
[Switching to Thread 1070182164]
Thread 2 "main" hit Breakpoint 1, app_main () at /home/siero/lec_lab2/lab2_debug/main/lab2_
22 in /home/siero/lec_lab2/lab2_debug/main/lab2_debug.c
(gdb) info line compute
Line 22 of "/home/siero/lec_lab2/lab2_debug/main/lab2_debug.c" starts at address 0x4200ce4
and ends at 0x4200ce5c <compute+18>.
(gdb) set args 502 303 404
(gdb) b compute
Breakpoint 2 at 0x4200ce5c: file /home/siero/lec_lab2/lab2_debug/main/lab2_debug.c, line 17
(gdb) continue
Continuing
Thread 2 "main" hit Breakpoint 2, compute (a=55, b=0, c=4) at /home/siero/lec_lab2/lab2_
debug.c:17
17 in /home/siero/lec_lab2/lab2_debug/main/lab2_debug.c
(gdb) info args
```

```
ubuntu@ubuntu: ~/esp/lab2_1
(gdb) info register
ra          0x4200cfb6      0x4200cfb6 <app_main+322>
sp          0x3fc9ae60      0x3fc9ae60
gp          0x3fc96400      0x3fc96400
tp          0x3fc8f608      0x3fc8f608
t0          0x4005890e      1074104598
t1          0x20000000      536870912
t2          0x0          0
fp          0x3fc9ae90      0x3fc9ae90
s1          0x3c02284       1086774916
a0          0x3b      59
a1          0x0          0
a2          0x4          4
a3          0x0          0
a4          0x37      55
a5          0x3b      59
a6          0x0          0
a7          0xa          10
s2          0x0          0
s3          0x0          0
s4          0x0          0
s5          0x0          0
s6          0x0          0
s7          0x0          0
s8          0x0          0
s9          0x0          0
s10         0x0          0
s11         0x0          0
t3          0x0          0
t4          0x0          0
t5          0x0          0
t6          0x0          0
pc          0x4200cfb6      0x4200cfb6 <app_main+322>
(gdb) info frame
Stack level 0, frame at 0x3fc9ae90:
  pc = 0x4200cfb6 in app_main (/home/siero/lec_lab2/lab2_debug/main/lab2_debug.c:52);
  saved pc = 0x4201de6
  call by register 0x3fc9aec0
  source language C
  Arglist at 0x3fc9ae90, args:
  Locals at 0x3fc9ae90, Previous frame's sp is 0x3fc9ae90
  Saved registers:
    ra at 0x3fc9ae8c, fp at 0x3fc9ae88, pc at 0x3fc9ae8c
(gdb) info register a5
a5          0x3b      59
(gdb) print a5
$5 = 59
No symbol "a5" in current context.
(gdb) print $a5
$2 = 59
(gdb) print $a0
$3 = 59
(gdb) disassemble 0x4200ce4a
Undefined command: "disassemble". Try "help".
(gdb) disassemble 0x4200ce4a
(gdb) disassemble code for function compute:
Dump of assembler code for function compute:
0x4200ce4a <>0>; addi    sp,sp,-32
0x4200ce4c <>2>;    sw     $0,28(%sp)
0x4200ce4e <>4>;    addi   $0,sp,32
```

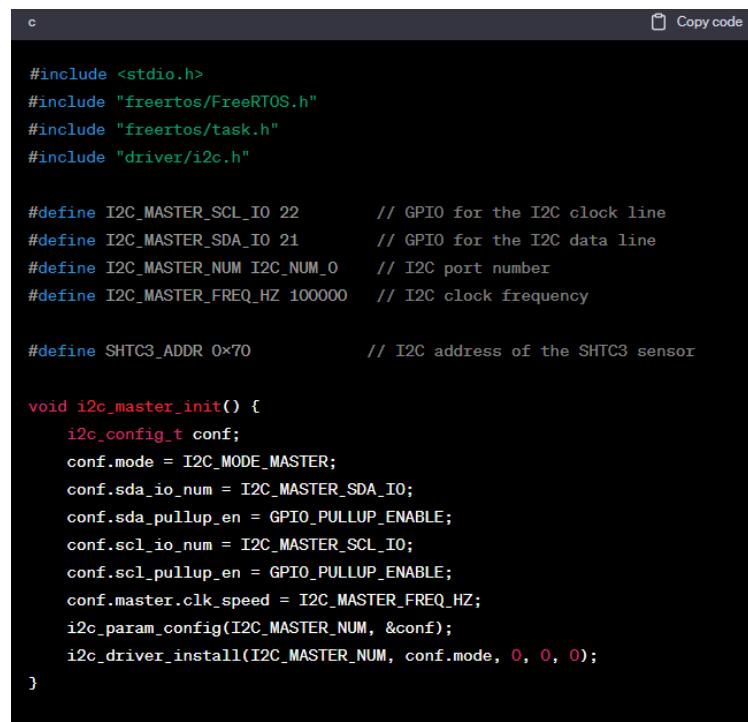
```
ubuntu@ubuntu: ~/esp/lab2_1
(gdb) info register
s8          0x0          0
s9          0x0          0
s10         0x0          0
s11         0x0          0
t3          0x0          0
t4          0x0          0
t5          0x0          0
t6          0x0          0
pc          0x4200cfb6      0x4200cfb6 <app_main+322>
(gdb) info frame
Stack level 0, frame at 0x3fc9ae90:
  pc = 0x4200cfb6 in app_main (/home/siero/lec_lab2/lab2_debug/main/lab2_debug.c:52);
  saved pc = 0x4201de6
  call by register 0x3fc9aec0
  source language C
  Arglist at 0x3fc9ae90, args:
  Locals at 0x3fc9ae90, Previous frame's sp is 0x3fc9ae90
  Saved registers:
    ra at 0x3fc9ae8c, fp at 0x3fc9ae88, pc at 0x3fc9ae8c
(gdb) info register a5
a5          0x3b      59
(gdb) print a5
$5 = 59
No symbol "a5" in current context.
(gdb) print $a5
$2 = 59
(gdb) print $a0
$3 = 59
(gdb) disassemble 0x4200ce4a
Undefined command: "disassemble". Try "help".
(gdb) disassemble 0x4200ce4a
(gdb) disassemble code for function compute:
Dump of assembler code for function compute:
0x4200ce4a <>0>; addi    sp,sp,-32
0x4200ce4c <>2>;    sw     $0,28(%sp)
0x4200ce4e <>4>;    addi   $0,sp,32
```

GDB Commands: <https://web.cecs.pdx.edu/~apt/cs491/gdb.pdf>

Set Args: https://visualgdb.com/gdbreference/commands/set_args

Lab 2: Part 2

After a long struggle with the first portion of the lab, first started researching what parts of the ESP32C3 board we'd be using and noticed that the SHTC3 sensor is built in within. Essentially, it gave me an idea of what to look for and used ChatGPT to start off the code. Essentially, I had given it the information of what type of board we'd be using, the protocol of I2C, and the functionality that is looked for, temperature and humidity. Below will show a screenshot of the code snippet given by ChatGPT, which served as a great starting point that needed alterations for it to work properly.



The screenshot shows a code editor window with a dark theme. The code is written in C and defines an I2C master configuration. It includes headers for stdio.h, freertos/FreeRTOS.h, freertos/task.h, and driver/i2c.h. It defines constants for I2C_MASTER_SCL_IO (22), I2C_MASTER_SDA_IO (21), I2C_MASTER_NUM (I2C_NUM_0), and I2C_MASTER_FREQ_HZ (100000). It also defines the I2C address of the SHTC3 sensor as 0x70. The code then initializes the I2C master using i2c_master_init(), setting up the configuration structure with mode set to I2C_MODE_MASTER, SDA and SCL pull-up enable, and a clock speed of 100000 Hz. Finally, it installs the driver with i2c_driver_install() for port 0, mode 0, frequency 0, and other parameters 0.

```
c
Copy code

#include <stdio.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "driver/i2c.h"

#define I2C_MASTER_SCL_IO 22          // GPIO for the I2C clock line
#define I2C_MASTER_SDA_IO 21          // GPIO for the I2C data line
#define I2C_MASTER_NUM I2C_NUM_0      // I2C port number
#define I2C_MASTER_FREQ_HZ 100000     // I2C clock frequency

#define SHTC3_ADDR 0x70              // I2C address of the SHTC3 sensor

void i2c_master_init() {
    i2c_config_t conf;
    conf.mode = I2C_MODE_MASTER;
    conf.sda_io_num = I2C_MASTER_SDA_IO;
    conf.sda_pullup_en = GPIO_PULLUP_ENABLE;
    conf.scl_io_num = I2C_MASTER_SCL_IO;
    conf.scl_pullup_en = GPIO_PULLUP_ENABLE;
    conf.master.clk_speed = I2C_MASTER_FREQ_HZ;
    i2c_param_config(I2C_MASTER_NUM, &conf);
    i2c_driver_install(I2C_MASTER_NUM, conf.mode, 0, 0, 0);
}
```

```

esp_err_t read_temperature_and_humidity(float *temperature_C, 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_ADDR << 1) | I2C_MASTER_READ, 1);
    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_RATE_MS);
    i2c_cmd_link_delete(cmd);

    if (ret != ESP_OK) {
        return ret;
    }

    // Replace the following with the correct data byte indices and conversions
    uint16_t raw_temp = (data[0] << 8) | data[1];
    uint16_t raw_humidity = (data[3] << 8) | data[4];

    *temperature_C = -45.0 + 175.0 * (float)raw_temp / 65535.0;
    *humidity = 100.0 * (float)raw_humidity / 65535.0;

    return ESP_OK;
}

```

```

void app_main() {
    float temperature_C;
    float humidity;

    i2c_master_init();

    while (1) {
        if (read_temperature_and_humidity(&temperature_C, &humidity) == ESP_OK)
            // Process and print the temperature and humidity data
        } else {
            printf("Failed to read data from the sensor\n");
        }

        vTaskDelay(2000 / portTICK_RATE_MS);
    }
}

```

The main source of difficulty was figuring out some issues with the code given by generative AI.

One thing that was causing the program to exit into the else statement was the fact that the AI only provided me with writing into the file and not reading. Through much of the trial and error, I had to input the address values of 0x7CA2, a 16-bit value representing the reading of temperature with the sensor SHTC3. We pass this in the second parameter of

`i2c_master_write` for the sole purpose of ensuring that the temperature is being read and written out on the terminal. Though, provided with starting code by ChatGPT assisted in the learning process of this part of the lab. Looking over Github repositories of the I2C protocol, it showed me how to utilize different functions along with the proper syntax from using the “driver/i2c.h” header file. Below will show the sources of what was stated.

SHTC3:https://sensirion.com/media/documents/643F9C8E/63A5A436/Datasheet_SHTC3.pdf

f

I2C Protocol Repo:

https://github.com/espressif/esp-idf/blob/master/examples/peripherals/i2c/i2c_simple/main/i2c_simple_main.c