

Lab 4

Part 1 Process

The objective of the first part of the lab was to read values from the IMU and display on the monitor which direction the board is tilted. It only tracks tilt on the x and y axis, meaning the output could be any combination of “UP”, “DOWN”, “LEFT”, or “RIGHT.” Most of my code was very similar to how the code for the temperature sensor works. I first gave ChatGPT the assignment description to generate some starter code. The query and the output is shown below. It generated a main function and two sub functions. The first sub function is used to initialize I2C. I copied my initialization code from Lab 3 into here since I know it works without error. The other subfunction is used to communicate with the accelerometer, parse its output, and then to write strings to the ESP_LOG for the monitor output. This function starts by writing the register addresses that contain the accelerometer output values to the IMU’s input bus. The IMU then outputs the relevant acceleration data onto its output bus which my function then reads. The data for each axis is returned as two separate 8 bit values, one for the upper bits and one for the lower. I used a bit shift technique and bitwise masking to get the final values into a single 16 bit value. These output values are then checked to see if they are in the correct range to indicate a movement. I set my dead zone boundaries to be +-500 for each axis to try and filter out the excess noise. STRCPY is used to copy the strings I talked about earlier into empty character arrays which are used for the monitor output (one for each axis). These character arrays are then combined using sprintf and then sent to the ESP_LOG output.

The last function is main. Main starts by calling the initialization function to initialize I2C. It then writes a command to the IMU which turns on the accelerometer and gyroscope. After that an infinite while loop continuously calls the IMU read function to get the output.

Part 2 Process

Part 2 was relatively straightforward. I followed the instructions to get bluetooth working on the PI. Once it was working I made a copy of the bludroid HID project into my own projects folder. I changed the name of the bluetooth device and then flashed it onto the ESP32c3. I was having difficulties with getting it to connect to the PI so I just used my Windows laptop instead. They communicated with each other with no problems. This confirmed that the basic bluetooth functionality was working, and the volume up/down demo was working. I studied how the code worked. I found the demo function where the actual control functionality was implemented. I modified it using the built in mouse manipulation function so that it would move the mouse left and right and

then pause for 5 seconds (as per the lab instructions). I flashed it and it worked as intended.

Part 3 Process

Part 3 was a matter of simply combining parts 1 and 2. I again copied the bludroid code over into a new project. I then copied all of my gyro code into the main HID script. The initialization process largely remained the same. I modified the main function so that it would call the initialization function and also send the command to the IMU to turn on the accelerometer. The demo task function was modified such that it calls the read gyro function that I created before. The read gyro function had the dead zone if statement checks stripped out and was modified to return the raw x and y values out of the IMU. All of the monitoring functions were also removed. The x and y values are stored in an array and then returned to the demo function. The raw x and y values are then filtered through a series of if statements that check if their value is above 500 or below -500. This creates the dead zones like before, but the function has been moved to the demo function instead of doing it in the gyro read function. If the x and y values are within the necessary ranges, then new x and y variables are set to 255 or -255 (depending on the boards orientation) and sent to the mouse movement function in order to move the mouse. This function then loops to repeat the process.

Sources

ChatGPT was my only source this time, and it was only used to generate some starter code for part 1 and to try and figure out how the ESP_LOGI works, along with some other syntax.

Can you write me a c script that uses i2c to communicate with the imu sensor on the ESP32C3, that uses esp_logi to print up, down, left, or right depending on the boards tilt?

ChatGPT

Certainly! Here's a basic C script that communicates with an IMU sensor over I2C on an ESP32-C3 and uses `esp_logi` to print "up," "down," "left," or "right" based on the board's tilt. Please note that this is a simplified example and may require adjustments to work with your specific IMU sensor and configuration.

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

static const char *TAG = "IMU";
```

```
static i2c_port_t i2c_port = I2C_NUM_0;
static uint8_t imu_address = 0x68; // Replace with your IMU's
I2C address

void imu_init() {
    i2c_config_t conf;
    conf.mode = I2C_MODE_MASTER;
    conf.sda_io_num = GPIO_NUM_21; // Replace with your SDA GPIO
pin
    conf.scl_io_num = GPIO_NUM_22; // Replace with your SCL GPIO
pin
    conf.sda_pullup_en = GPIO_PULLUP_ENABLE;
    conf.scl_pullup_en = GPIO_PULLUP_ENABLE;
    conf.master.clk_speed = 100000; // I2C clock speed

    ESP_ERROR_CHECK(i2c_param_config(i2c_port, &conf));
    ESP_ERROR_CHECK(i2c_driver_install(i2c_port, conf.mode, 0,
0, 0));
}

void imu_read_acceleration(int16_t *acceleration) {
    uint8_t data[6];
    i2c_cmd_handle_t cmd = i2c_cmd_link_create();

    i2c_master_start(cmd);
    i2c_master_write_byte(cmd, (imu_address << 1) |
I2C_MASTER_READ, true);
    i2c_master_read(cmd, data, 6, I2C_MASTER_LAST_NACK);
    i2c_master_stop(cmd);

    ESP_ERROR_CHECK(i2c_master_cmd_begin(i2c_port, cmd, 1000 /
portTICK_RATE_MS));
    i2c_cmd_link_delete(cmd);

    // Assuming data contains the raw accelerometer readings
```

```
// You'll need to parse your IMU's data format here

// Calculate the tilt direction based on the accelerometer
data
if (acceleration[2] > 10000) {
    ESP_LOGI(TAG, "Board is tilted up");
} else if (acceleration[2] < -10000) {
    ESP_LOGI(TAG, "Board is tilted down");
} else if (acceleration[0] > 10000) {
    ESP_LOGI(TAG, "Board is tilted left");
} else if (acceleration[0] < -10000) {
    ESP_LOGI(TAG, "Board is tilted right");
}
}

void app_main() {
    imu_init();
    int16_t acceleration[3];

    while (1) {
        imu_read_acceleration(acceleration);
        vTaskDelay(1000 / portTICK_RATE_MS); // Adjust the delay
as needed
    }
}
```

In this example:

1. `imu_init` initializes the I2C communication and configures the I2C bus and pins.
2. `imu_read_acceleration` reads the raw accelerometer data from the IMU sensor and then determines the board's tilt direction based on the accelerometer readings. You'll need to replace this part with the actual code to read data from your specific IMU sensor and interpret its output correctly.

3. The `app_main` function sets up the I2C communication and enters a loop where it periodically reads the accelerometer data and prints the tilt direction using `esp_logi`.

Remember to configure the correct I2C pins, I2C address, and data format for your specific IMU sensor. Additionally, make sure to install and configure the ESP-IDF development environment to compile and run this code on your ESP32-C3 board.



Can you give me an example of the use of this i2c command? `esp_err_t ret = i2c_master_write_to_device`

ChatGPT

Certainly! The `i2c_master_write_to_device` function is typically used in ESP-IDF for writing data to an I2C device. Here's an example of how you can use it to write data to an I2C device using ESP32-C3:

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

static const char *TAG = "I2C";
static i2c_port_t i2c_port = I2C_NUM_0; // Use the appropriate
I2C port
static uint8_t device_address = 0x68;    // Replace with the I2C
address of your device

void i2c_init() {
    i2c_config_t conf;
    conf.mode = I2C_MODE_MASTER;
    conf.sda_io_num = GPIO_NUM_21;    // Replace with the SDA pin
number
    conf.sda_pullup_en = GPIO_PULLUP_ENABLE;
    conf.scl_io_num = GPIO_NUM_22;    // Replace with the SCL pin
number
    conf.scl_pullup_en = GPIO_PULLUP_ENABLE;
```

```
    conf.master.clk_speed = 100000; // I2C clock speed (100
kHz)

    ESP_ERROR_CHECK(i2c_param_config(i2c_port, &conf));
    ESP_ERROR_CHECK(i2c_driver_install(i2c_port, conf.mode, 0,
0, 0));
}

esp_err_t i2c_write_data(uint8_t *data, size_t data_len) {
    i2c_cmd_handle_t cmd = i2c_cmd_link_create();

    // Start a write transaction
    i2c_master_start(cmd);

    // Write the device address with a write bit (0) to the bus
    i2c_master_write_byte(cmd, (device_address << 1) |
I2C_MASTER_WRITE, true);

    // Write data to the device
    i2c_master_write(cmd, data, data_len, true);

    // Stop the transaction
    i2c_master_stop(cmd);

    // Send the command
    esp_err_t ret = i2c_master_cmd_begin(i2c_port, cmd, 1000 /
portTICK_RATE_MS);

    // Free the command link
    i2c_cmd_link_delete(cmd);

    return ret;
}

void app_main() {
```

```
i2c_init();

// Example data to write to the I2C device
uint8_t data_to_send[] = {0x01, 0x02, 0x03};
size_t data_len = sizeof(data_to_send);

esp_err_t ret = i2c_write_data(data_to_send, data_len);
if (ret == ESP_OK) {
    ESP_LOGI(TAG, "Data write successful");
} else {
    ESP_LOGE(TAG, "Data write failed");
}
}
```

In this example:

1. `i2c_init` initializes the I2C communication by configuring the I2C bus and pins.
2. `i2c_write_data` is a function that writes data to the I2C device. It starts by creating an I2C command link, writes the device's address with the write bit set, writes the data to the device, and then stops the transaction.
3. In the `app_main` function, we call `i2c_write_data` with the example data `data_to_send`, and it will write this data to the I2C device. The result is logged using `esp_logi`.

Please replace the I2C bus, device address, and GPIO pins with the appropriate values for your specific setup. Also, adjust the data and data length accordingly to match your use case.



how do I read the icm-42670-p gyro data on the esp32c3 using i2c

3 / 3

ChatGPT

To read gyro data from the ICM-42670-P gyro sensor on the ESP32-C3 using I2C, you can follow a similar approach as in my previous example for I2C communication. However, you need to configure the I2C communication and use the ICM-42670-P's specific I2C address and data format.

Here's a simplified example of how to read gyro data from the ICM-42670-P sensor using I2C:

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

static const char *TAG = "ICM42670";
static i2c_port_t i2c_port = I2C_NUM_0; // Use the appropriate
I2C port
static uint8_t gyro_address = 0x68;      // ICM-42670-P's I2C
address

void i2c_init() {
    i2c_config_t conf;
    conf.mode = I2C_MODE_MASTER;
    conf.sda_io_num = GPIO_NUM_21;  // Replace with your SDA
pin number
    conf.sda_pullup_en = GPIO_PULLUP_ENABLE;
    conf.scl_io_num = GPIO_NUM_22;  // Replace with your SCL
pin number
    conf.scl_pullup_en = GPIO_PULLUP_ENABLE;
    conf.master.clk_speed = 100000; // I2C clock speed (100
kHz)

    ESP_ERROR_CHECK(i2c_param_config(i2c_port, &conf));
    ESP_ERROR_CHECK(i2c_driver_install(i2c_port, conf.mode, 0,
0, 0));
}

esp_err_t read_gyro_data(int16_t *gyro_data) {
    uint8_t data[12]; // The ICM-42670-P provides 6 bytes of
gyro data for X, Y, and Z axes

    i2c_cmd_handle_t cmd = i2c_cmd_link_create();
```



```
// Start a write transaction
i2c_master_start(cmd);

// Write the device address with a write bit (0) to the bus
i2c_master_write_byte(cmd, (gyro_address << 1) |
I2C_MASTER_WRITE, true);

// Select the gyro data register address (e.g., 0x1D for
X-axis gyro data)
i2c_master_write_byte(cmd, 0x1D, true);

// Start a read transaction
i2c_master_start(cmd);

// Write the device address with a read bit (1) to the bus
i2c_master_write_byte(cmd, (gyro_address << 1) |
I2C_MASTER_READ, true);

// Read the gyro data
i2c_master_read(cmd, data, 12, I2C_MASTER_LAST_NACK);

// Stop the transaction
i2c_master_stop(cmd);

// Send the command
esp_err_t ret = i2c_master_cmd_begin(i2c_port, cmd, 1000 /
portTICK_RATE_MS);

// Free the command link
i2c_cmd_link_delete(cmd);

if (ret == ESP_OK) {
    // Assuming the data format is little-endian and two's
complement
    gyro_data[0] = (int16_t)((data[0] << 8) | data[1]);
```

```
        gyro_data[1] = (int16_t)((data[2] << 8) | data[3]);
        gyro_data[2] = (int16_t)((data[4] << 8) | data[5]);
    }

    return ret;
}

void app_main() {
    i2c_init();
    int16_t gyro_data[3];

    while (1) {
        esp_err_t ret = read_gyro_data(gyro_data);

        if (ret == ESP_OK) {
            ESP_LOGI(TAG, "Gyro Data - X: %d, Y: %d, Z: %d",
gyro_data[0], gyro_data[1], gyro_data[2]);
        } else {
            ESP_LOGE(TAG, "Failed to read gyro data");
        }

        vTaskDelay(1000 / portTICK_RATE_MS); // Adjust the
delay as needed
    }
}
```

In this example:

1. `i2c_init` initializes the I2C communication by configuring the I2C bus and pins.
2. `read_gyro_data` reads the gyro data from the ICM-42670-P sensor. It starts by creating an I2C command link, writes the device's address and selects the appropriate register for gyro data. Then, it reads 12 bytes of data (2 bytes per axis) and converts the data to meaningful gyro values. Be sure to consult the ICM-42670-P datasheet for the correct register addresses and data format.
3. In the `app_main` function, we continuously read and log the gyro data in a loop. Adjust the delay as needed to control the data read rate.



how do I use sprintf to combine two strings

ChatGPT

To combine two strings using the `sprintf` function in C, you can use the format specifier `%s` to insert the content of one string into another. Here's an example:

```
c
#include <stdio.h>

int main() {
    char firstString[] = "Hello, ";
    char secondString[] = "world!";
    char combinedString[50]; // Allocate a buffer to hold the
    combined string

    // Use sprintf to combine the two strings
    sprintf(combinedString, "%s%s", firstString, secondString);

    printf("Combined String: %s\n", combinedString);

    return 0;
}
```

In this example, we have two strings, `firstString` and `secondString`, and we want to combine them into `combinedString`. We use the `sprintf` function with the format string `"%s%s"` to insert the content of `firstString` and `secondString` into `combinedString`.

Remember to allocate a buffer (in this case, `combinedString`) with enough space to hold the resulting combined string. In practice, you should ensure that the buffer size is large enough to accommodate the combined string, as `sprintf` doesn't perform bounds checking.

The `%s` format specifier is used to insert a string, and it will copy the content of the string into the resulting string until the null-terminating character `'\0'` is encountered in the source string.



how do i assign a string to a character array in c

ChatGPT

In C, you can assign a string to a character array using either array initialization or string copy functions. Here are two common methods:

1. Array Initialization: You can initialize a character array with a string literal directly, which automatically assigns the characters to the array:

c

```
char myCharArray[] = "Hello, World!";
```

In this example, `myCharArray` is a character array that contains the string "Hello, World!".

2. Using String Copy Functions (e.g., `strcpy`): You can use the `strcpy` function from the `<string.h>` library to copy a string into a character array:

c

```
#include <stdio.h>
#include <string.h>
```

```
int main() {
    char myCharArray[20]; // Create a character array with
                           // enough space

    strcpy(myCharArray, "Hello, World!"); // Copy the string
    into the array

    printf("myCharArray: %s\n", myCharArray);

    return 0;
```

```
}
```

In this example, we declare a character array `myCharArray` and then use `strcpy` to copy the string "Hello, World!" into it. You should make sure that the character array is large enough to accommodate the string and a null-terminating character, which is automatically added by `strcpy`.

Please be cautious when using `strcpy` to ensure that the source string fits within the destination character array to avoid buffer overflows. It's a good practice to specify the size of the destination array to avoid potential buffer overflows, as shown in the example above.



how do I print bits in c

2 / 2

ChatGPT

To print the individual bits of a value in C, you can use bitwise operations and loops to iterate through the bits and print them one by one. Here's a basic example of how you can do this:

c

```
#include <stdio.h>
```

```
void print_bits(unsigned int value) {  
    int num_bits = sizeof(value) * 8; // Assuming 32 bits,  
    adjust for your platform
```

```
    for (int i = num_bits - 1; i >= 0; i--) {  
        int bit = (value >> i) & 1;  
        printf("%d", bit);  
    }  
    printf("\n");  
}
```

```
int main() {  
    unsigned int num = 42; // Change to your desired value
```

```
printf("Binary representation of %d: ", num);  
print_bits(num);  
return 0;  
}
```

In this example, the `print_bits` function takes an unsigned integer `value` and iterates through its bits from the most significant bit to the least significant bit using a loop. It uses bitwise operations to extract each bit (0 or 1) and prints it to the console. The `sizeof(value) * 8` calculates the number of bits in the integer, assuming a 32-bit integer. Adjust this value for your specific platform if necessary.

Remember that this example prints the binary representation of an integer. You can modify it to print bits for other types, such as `char`, `short`, or `long`, by adjusting the `sizeof` and loop parameters accordingly.



what is the maximum value a uint16_t can carry

ChatGPT

A `uint16_t` is an unsigned 16-bit integer data type in C/C++. It can represent values from 0 to 65,535. In other words, the maximum value a `uint16_t` can hold is $2^{16} - 1$, which is equal to 65,535. This data type can store 16 bits, allowing it to represent values in the range from 0 to 65,535 without a sign bit.



how do I print hex values in c

ChatGPT

To print hex values in C, you can use the `printf` function with the `%x` format specifier. Here's how you can print hex values:

c

```
#include <stdio.h>
```

```
int main() {  
    int decimalValue = 255; // Replace with your desired value
```

```
// Printing as a hexadecimal value
printf("Decimal value %d in hexadecimal is: %x\n",
decimalValue, decimalValue);

return 0;
}
```

In this example, we use the `%x` format specifier to print the integer `decimalValue` in hexadecimal format. The `%x` specifier tells `printf` to interpret the argument as an integer and print it in hexadecimal notation.

You can use `%X` instead of `%x` to print the hexadecimal value in uppercase letters. For example:

```
c
printf("Decimal value %d in hexadecimal (uppercase) is: %X\n",
decimalValue, decimalValue);
```

This code will print the hexadecimal value in uppercase, which is commonly used for hexadecimal representations.



how do I convert uint16_t into 2s complement in c

ChatGPT

In C, you don't need to explicitly convert a `uint16_t` into 2's complement because `uint16_t` is an unsigned data type and already represents values using 2's complement notation. The 2's complement representation is used for signed integer types like `int16_t`.

If you have a `uint16_t` value that you want to interpret as a signed 16-bit value (i.e., as an `int16_t`), you can simply cast it. Keep in mind that if the original `uint16_t` value exceeds the range of `int16_t`, the conversion may lead to unexpected results.

Here's an example of how you can cast a `uint16_t` to an `int16_t`:

```
c
```

```
#include <stdint.h>
#include <stdio.h>

int main() {
    uint16_t uintValue = 65535; // A uint16_t value (maximum
    value)

    int16_t intValue = (int16_t)uintValue; // Cast to int16_t

    printf("Original uint16_t value: %u\n", uintValue);
    printf("Converted int16_t value: %d\n", intValue);

    return 0;
}
```

In this example, we cast the `uint16_t` value to an `int16_t`. Note that if the `uint16_t` value is within the range of `int16_t`, the cast will simply interpret it as a signed 16-bit value. However, if the `uint16_t` value is larger than what can be represented as a positive `int16_t`, it will overflow and the result may not be what you expect.



how do i convert an unsigned value into 2s complement in c

ChatGPT

To convert an unsigned value into its two's complement representation in C, you can follow these steps:

1. Determine the maximum value that can be represented by the signed data type you want to convert to. For example, if you are converting to a signed 8-bit integer (`int8_t`), the maximum value it can represent is 127.
2. If the original unsigned value is greater than or equal to half of the maximum value plus 1, subtract the maximum value from it.

Here's a code example demonstrating this process:

c


```
#include <stdio.h>
#include <stdint.h>

int main() {
    // Original unsigned value
    uint16_t unsignedValue = 300; // Change to your desired
    value

    // Determine the maximum value for the signed data type
    int16_t maxSignedValue = (1 << (sizeof(int16_t) * 8 - 1)) -
    1;

    // Convert the unsigned value to two's complement
    int16_t twosComplementValue;

    if (unsignedValue >= (uint16_t)(maxSignedValue + 1)) {
        twosComplementValue = unsignedValue -
(unsigned16_t)(maxSignedValue + 1);
    } else {
        twosComplementValue = unsignedValue;
    }

    printf("Original unsigned value: %u\n", unsignedValue);
    printf("Two's complement value: %d\n", twosComplementValue);

    return 0;
}
```

In this example, we determine the maximum value that can be represented by an `int16_t` (a signed 16-bit integer). If the original `unsignedValue` is greater than or equal to half of the maximum value plus 1, we subtract the maximum value from it to obtain the two's complement representation. Otherwise, we leave it as is.

Make sure to adjust the code for the specific data types and ranges you are working with. This code assumes that you want to convert to a signed 16-bit integer (`int16_t`), but you can adapt it for other data types as needed.



Casting wont work. I need to actually do the 2s complement operation

ChatGPT

If you need to manually convert an unsigned value to its two's complement representation, you can perform the two's complement operation by inverting the bits and then adding 1. Here's how you can do it in C:

C

```
#include <stdio.h>
#include <stdint.h>

int main() {
    // Original unsigned value
    uint16_t unsignedValue = 300; // Change to your desired
value

    // Determine the maximum value for the signed data type
    int16_t maxSignedValue = (1 << (sizeof(int16_t) * 8 - 1)) -
1;

    // Convert the unsigned value to two's complement manually
    int16_t twosComplementValue;

    if (unsignedValue <= (uint16_t)maxSignedValue) {
        // Value fits within positive range
        twosComplementValue = unsignedValue;
    } else {
        // Value is too large for positive range, calculate
two's complement
        twosComplementValue = ~unsignedValue + 1;
    }

    printf("Original unsigned value: %u\n", unsignedValue);
    printf("Two's complement value: %d\n", twosComplementValue);
}
```

```
    return 0;  
}
```

In this code, we determine if the original unsigned value fits within the positive range of the signed data type (in this case, `int16_t`). If it does, we simply assign it to `twosComplementValue`. If it's too large to fit in the positive range, we calculate its two's complement by inverting the bits using the bitwise NOT operator (`~`) and then adding 1. This manual operation will correctly convert the value to two's complement.



can you explain to me how to use ESP_LOGI to print a string for the esp32c3
ChatGPT

Certainly! To use `ESP_LOGI` to print a string in the ESP32-C3 development environment, you'll need to follow these steps:

1. Include the required header files:
 - `#include "esp_log.h"`: This header file provides the logging functions and macros.
 - `#include "esp_system.h"`: This header file contains system-related functions.
2. Initialize the logging system. You typically do this in your `app_main` function or another initialization function.
3. Use the `ESP_LOGI` macro to print your log message with the desired log level and tag.

Here's a simple example:

```
c  
#include "esp_log.h"  
#include "esp_system.h"  
  
void app_main() {  
    // Initialize the logging system with a default log  
    verbosity level  
    esp_log_level_set("*", ESP_LOG_INFO);  
}
```

```
// Your application code

// Use ESP_LOGI to print an informational message
ESP_LOGI("MyApp", "Hello, ESP32-C3!");

// More of your application code

// Deinitialize the logging system (optional)
esp_log_level_set("*", ESP_LOG_NONE);
}
```

In this example:

- We include the necessary header files, `esp_log.h` and `esp_system.h`.
- We initialize the logging system using `esp_log_level_set`. In this case, we set the log verbosity level to `ESP_LOG_INFO` for all log tags ("*"). You can adjust the verbosity level and specify tags for finer control.
- Inside your application code, you can use `ESP_LOGI` to print an informational message. The first argument is the tag, which helps identify the source of the log message. The second argument is the log message itself.
- Optionally, you can deinitialize the logging system when your application is done using `esp_log_level_set` with `ESP_LOG_NONE` to disable all logging.

To view the logs generated by `ESP_LOGI`, you need to use a tool like `idf_monitor` or `esptool.py` to monitor the serial output from your ESP32-C3 device.

Make sure you have the ESP-IDF development environment set up correctly and that you have configured your project for ESP32-C3 development.