

## Lab 6 Report

I began with my code from Lab 2 part 2. I connected the esp32c3 with the SR04 and programmed from VSCode. I prompted chat gpt to modify my code to read the distance and speed of sound. There were almost no errors, this lab went very smoothly. Below is my original prompt, there were a few back and forths with code debugging but I only changed one function call and the connections that were wrong:

I have a working temp sensor. I need to complete this: Use the SR04 to measure the distance, AND also use the temperature sensor to "adjust" the speed of sound based on the current temperature. Assume that the temperature is always between 0 and 50C. The temperature (C) and distance (cm) should be shown in the monitor printed once per second. E.g: Distance: 3.5 cm at 23C Distance: 4.5 cm at 23C The distance to measure by the TAs would be from 10 to 20cm. You can calibrate the sensor, but there should be less than a 2 cm error. The distance is computed from the PCB board (not the sensor itself) to a flat surface. NOTE: To get accurate results, you may need to generate multiple pulses, you may also want to look for counters in the hal/cpu\_hal.h that has more precision. Here is my code for the temp sensor: #include <stdio.h> #include "driver/i2c.h" #include "freertos/task.h" // I2C configuration #define I2C\_MASTER\_SCL\_IO 8 // GPIO for SCL #define I2C\_MASTER\_SDA\_IO 10 // GPIO for SDA #define I2C\_MASTER\_NUM I2C\_NUM\_0 // I2C port number for master #define I2C\_MASTER\_FREQ\_HZ 100000 // I2C clock frequency #define SHTC3\_SENSOR\_ADDR 0x70 // SHTC3 I2C address // SHTC3 command codes #define SHTC3\_CMD\_WAKEUP 0x3517 #define SHTC3\_CMD\_SLEEP 0xB098 #define SHTC3\_CMD\_MEASURE 0x7CA2 // Measure temp and humidity with clock stretching // Initialize I2C master static void 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, }; i2c\_param\_config(I2C\_MASTER\_NUM, &conf); i2c\_driver\_install(I2C\_MASTER\_NUM, I2C\_MODE\_MASTER, 0, 0, 0); // Send command to SHTC3 sensor static esp\_err\_t shtc3\_write\_cmd(uint16\_t cmd) { uint8\_t cmd\_buf[2] = { (cmd >> 8) & 0xFF, cmd & 0xFF }; return i2c\_master\_write\_to\_device(I2C\_MASTER\_NUM, SHTC3\_SENSOR\_ADDR, cmd\_buf, sizeof(cmd\_buf), pdMS\_TO\_TICKS(1000)); } // Read temperature and humidity data (2 bytes each) static esp\_err\_t shtc3\_read\_data(uint8\_t \*data, size\_t len) { return i2c\_master\_read\_from\_device(I2C\_MASTER\_NUM, SHTC3\_SENSOR\_ADDR, data, len, pdMS\_TO\_TICKS(1000)); } // Convert raw sensor data to temperature (Celsius) static float get\_temperature(uint16\_t raw\_temp) { return -45 + 175 \* (raw\_temp / 65535.0); } // Convert raw sensor data to humidity (percentage) static float get\_humidity(uint16\_t raw\_hum) { return 100 \* (raw\_hum / 65535.0); } // Read and print temperature and humidity static void read\_and\_print\_data(void) { uint8\_t data[6]; // Power up the sensor shtc3\_write\_cmd(SHTC3\_CMD\_WAKEUP); vTaskDelay(pdMS\_TO\_TICKS(10)); // Start measurement shtc3\_write\_cmd(SHTC3\_CMD\_MEASURE); vTaskDelay(pdMS\_TO\_TICKS(15)); // Read 6 bytes: temperature (2 bytes), humidity (2 bytes),

```

and 2 CRC bytes shtc3_read_data(data, 6); // Convert raw data to temperature and humidity
uint16_t raw_temp = (data[0] << 8) | data[1]; uint16_t raw_hum = (data[3] << 8) | data[4]; float
temperature = get_temperature(raw_temp); float humidity = get_humidity(raw_hum); // Print the
results printf("Temperature is %.1f°C (or %.1f°F) with a %.1f%% Humidity\n", temperature,
temperature * 1.8 + 32, humidity); // Power down the sensor
shtc3_write_cmd(SHTC3_CMD_SLEEP); } void app_main(void) { i2c_master_init(); while (1) {
read_and_print_data(); vTaskDelay(pdMS_TO_TICKS(2000)); // Wait for 2 seconds } }

```

And here is the initial GPT response:

```

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

// I2C configuration for SHTC3 (same as your existing code)
#define I2C_MASTER_SCL_IO 8
#define I2C_MASTER_SDA_IO 10
#define I2C_MASTER_NUM I2C_NUM_0
#define I2C_MASTER_FREQ_HZ 100000
#define SHTC3_SENSOR_ADDR 0x70
#define SHTC3_CMD_WAKEUP 0x3517
#define SHTC3_CMD_SLEEP 0xB098
#define SHTC3_CMD_MEASURE 0x7CA2

// SR04 sensor configuration
#define TRIG_PIN 5
#define ECHO_PIN 6

// Function prototypes
static void i2c_master_init(void);
static esp_err_t shtc3_write_cmd(uint16_t cmd);
static esp_err_t shtc3_read_data(uint8_t *data, size_t len);
static float get_temperature(uint16_t raw_temp);
static void read_and_print_data(void);
static float measure_distance(float temperature);

// Initialize I2C master
static void 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,
    };
    i2c_param_config(I2C_MASTER_NUM, &conf);
    i2c_driver_install(I2C_MASTER_NUM, I2C_MODE_MASTER, 0, 0, 0);
}

```

// Read temperature

```

static float get_temperature(uint16_t raw_temp) {
    return -45 + 175 * (raw_temp / 65535.0);
}

```

```

static void read_and_print_data(void) {
    uint8_t data[6];
    shtc3_write_cmd(SHTC3_CMD_WAKEUP);
    vTaskDelay(pdMS_TO_TICKS(10));
    shtc3_write_cmd(SHTC3_CMD_MEASURE);
    vTaskDelay(pdMS_TO_TICKS(15));
    shtc3_read_data(data, 6);
    uint16_t raw_temp = (data[0] << 8) | data[1];
    float temperature = get_temperature(raw_temp);
    shtc3_write_cmd(SHTC3_CMD_SLEEP);

    float distance = measure_distance(temperature);

    printf("Distance: %.2f cm at %.1f°C\n", distance, temperature);
}

```

// Measure distance with SR04

```

static float measure_distance(float temperature) {
    // Set TRIG pin to LOW for a short delay
    gpio_set_level(TRIG_PIN, 0);
    ets_delay_us(2);

    // Send a 10µs pulse to TRIG pin
    gpio_set_level(TRIG_PIN, 1);
    ets_delay_us(10);
    gpio_set_level(TRIG_PIN, 0);

    // Measure ECHO pulse width
    int64_t start_time = 0, end_time = 0;
    while (gpio_get_level(ECHO_PIN) == 0) {

```

```

    start_time = esp_timer_get_time();
}
while (gpio_get_level(ECHO_PIN) == 1) {
    end_time = esp_timer_get_time();
}

// Calculate duration in microseconds
int64_t pulse_duration = end_time - start_time;

// Calculate speed of sound based on temperature
float speed_of_sound = 331.3 + 0.6 * temperature; // in m/s
float speed_of_sound_cm_us = speed_of_sound / 10000; // convert to cm/μs

// Calculate distance (in cm)
float distance = (pulse_duration * speed_of_sound_cm_us) / 2;
return distance;
}

void app_main(void) {
    // Initialize I2C for temperature sensor
    i2c_master_init();

    // Initialize GPIO for SR04
    gpio_config_t io_conf = {
        .pin_bit_mask = (1ULL << TRIG_PIN) | (1ULL << ECHO_PIN),
        .mode = GPIO_MODE_OUTPUT,
        .pull_up_en = GPIO_PULLUP_DISABLE,
        .pull_down_en = GPIO_PULLDOWN_ENABLE,
    };
    gpio_config(&io_conf);

    gpio_set_direction(ECHO_PIN, GPIO_MODE_INPUT);

    // Main loop
    while (1) {
        read_and_print_data();
        vTaskDelay(pdMS_TO_TICKS(1000)); // Wait for 1 second
    }
}

```