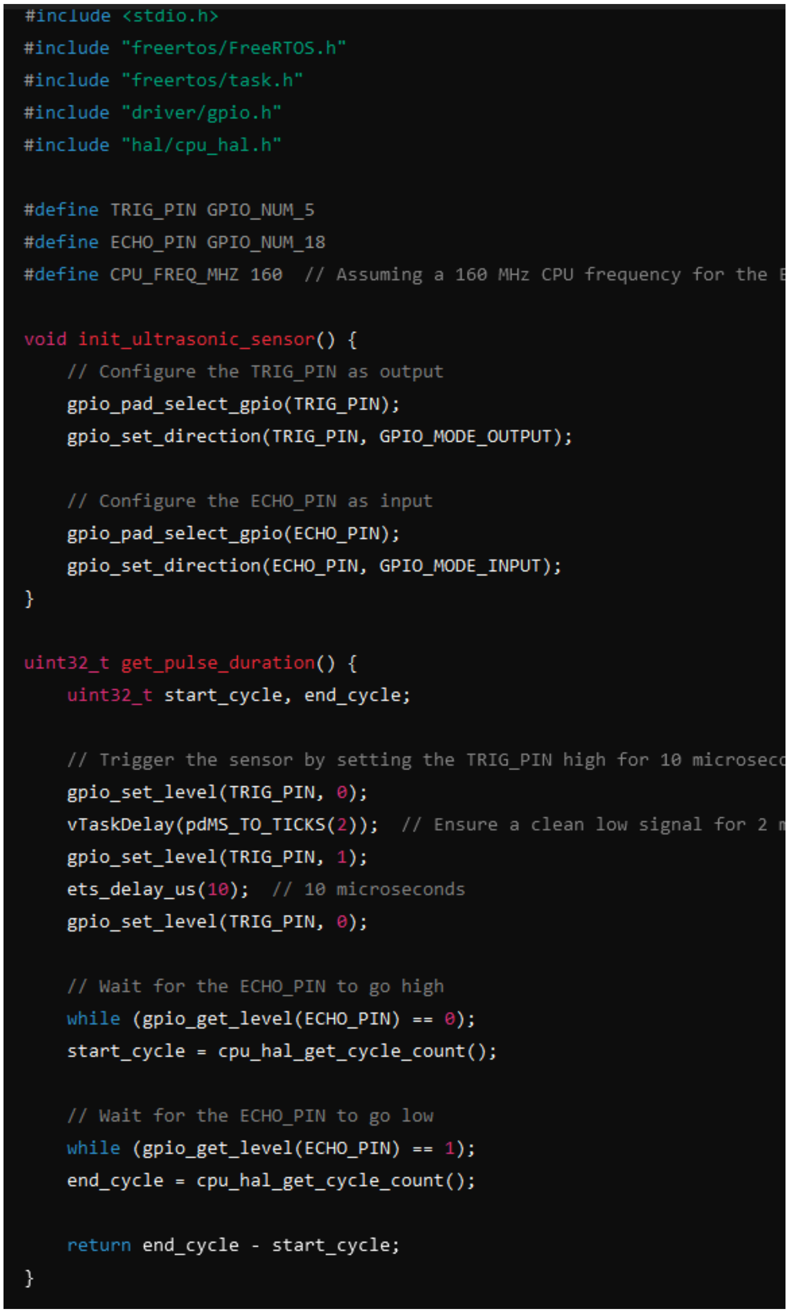
Lab 6 Report – Ryan King

Lab 6 Part 1

ChatGPT link: https://chatgpt.com/share/c0d2002a-c3f9-4e2a-a019-7222f519e48a

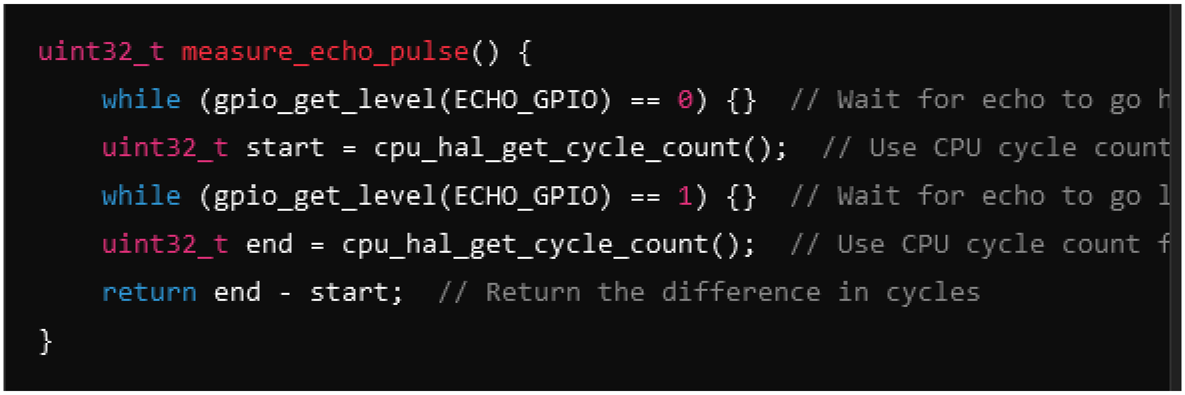
Code used and prompts:

“my professor says to use the hal/cpu_hal.h library for counters as it may have more precision. Can you elaborate?”



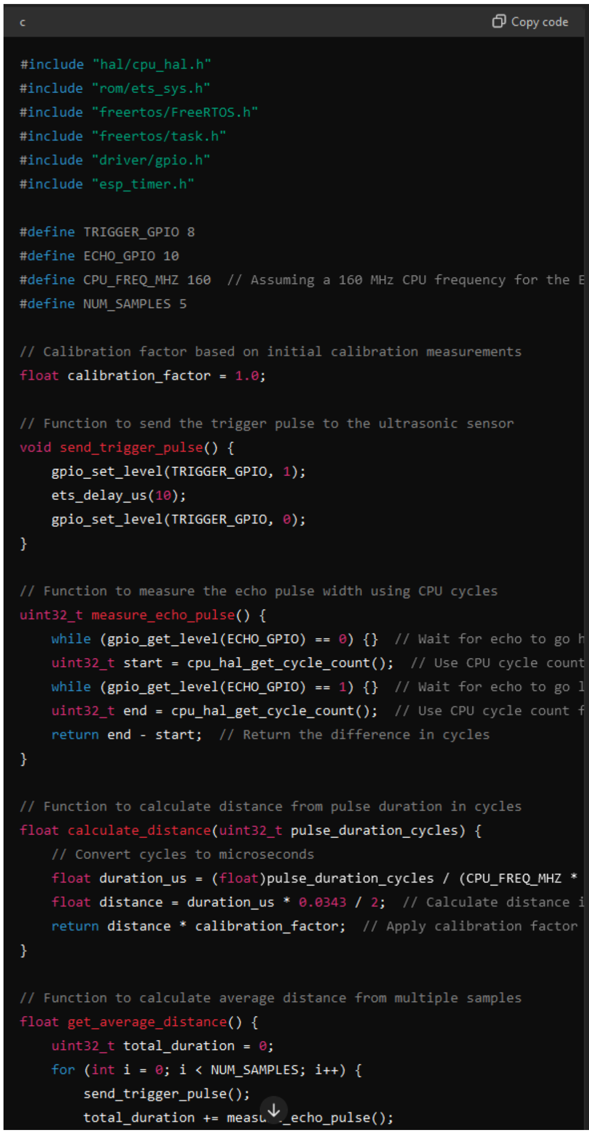
```c
#include <stdio.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "driver/gpio.h"
#include "hal/cpu_hal.h"

#define TRIG_PIN GPIO_NUM_5
#define ECHO_PIN GPIO_NUM_18
#define CPU_FREQ_MHZ 160  // Assuming a 160 MHz CPU frequency for the

void init_ultrasonic_sensor() {
    // Configure the TRIG_PIN as output
    gpio_pad_select_gpio(TRIG_PIN);
    gpio_set_direction(TRIG_PIN, GPIO_MODE_OUTPUT);

    // Configure the ECHO_PIN as input
    gpio_pad_select_gpio(ECHO_PIN);
    gpio_set_direction(ECHO_PIN, GPIO_MODE_INPUT);
}

uint32_t get_pulse_duration() {
    uint32_t start_cycle, end_cycle;

    // Trigger the sensor by setting the TRIG_PIN high for 10 microsec
    gpio_set_level(TRIG_PIN, 0);
    vTaskDelay(pdMS_TO_TICKS(2));  // Ensure a clean low signal for 2
    gpio_set_level(TRIG_PIN, 1);
    ets_delay_us(10);  // 10 microseconds
    gpio_set_level(TRIG_PIN, 0);

    // Wait for the ECHO_PIN to go high
    while (gpio_get_level(ECHO_PIN) == 0);
    start_cycle = cpu_hal_get_cycle_count();

    // Wait for the ECHO_PIN to go low
    while (gpio_get_level(ECHO_PIN) == 1);
    end_cycle = cpu_hal_get_cycle_count();

    return end_cycle - start_cycle;
}
```

“this is the code im using, please make the modifications accordingly and comment the lines where you made a change.”

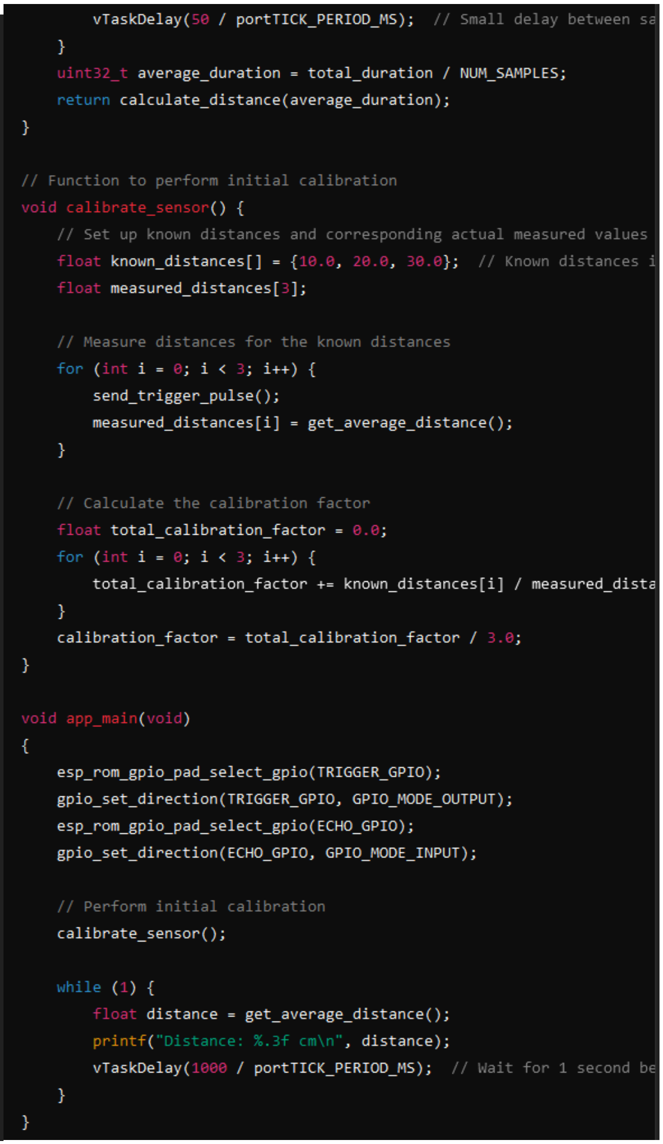


```c
uint32_t measure_echo_pulse() {
    while (gpio_get_level(ECHO_GPIO) == 0) {}  // Wait for echo to go h
    uint32_t start = cpu_hal_get_cycle_count();  // Use CPU cycle count
    while (gpio_get_level(ECHO_GPIO) == 1) {}  // Wait for echo to go l
    uint32_t end = cpu_hal_get_cycle_count();  // Use CPU cycle count f
    return end - start;  // Return the difference in cycles
}
```

“explain how I could calibrate the sensor”

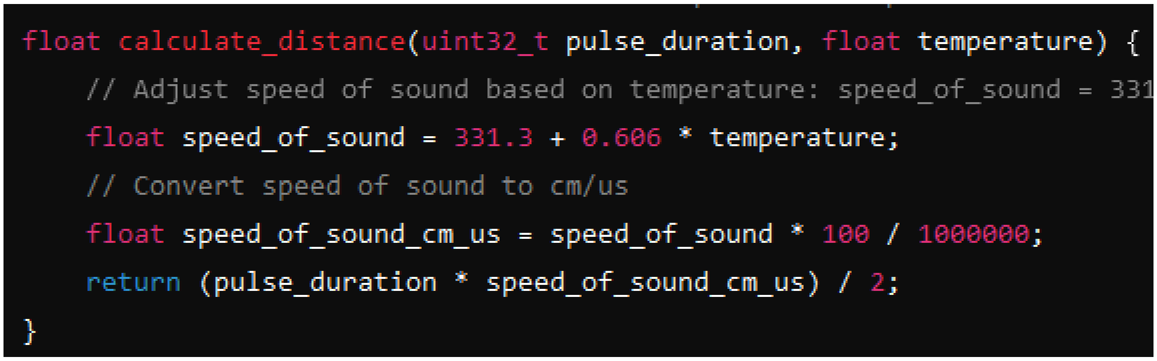


```c
#include "hal/cpu_hal.h"
#include "rom/ets_sys.h"
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "driver/gpio.h"
#include "esp_timer.h"

#define TRIGGER_GPIO 8
#define ECHO_GPIO 10
#define CPU_FREQ_MHZ 160  // Assuming a 160 MHz CPU frequency for the E
#define NUM_SAMPLES 5

// Calibration factor based on initial calibration measurements
float calibration_factor = 1.0;

// Function to send the trigger pulse to the ultrasonic sensor
void send_trigger_pulse() {
    gpio_set_level(TRIGGER_GPIO, 1);
    ets_delay_us(10);
    gpio_set_level(TRIGGER_GPIO, 0);
}

// Function to measure the echo pulse width using CPU cycles
uint32_t measure_echo_pulse() {
    while (gpio_get_level(ECHO_GPIO) == 0) {}  // Wait for echo to go h
    uint32_t start = cpu_hal_get_cycle_count();  // Use CPU cycle count
    while (gpio_get_level(ECHO_GPIO) == 1) {}  // Wait for echo to go l
    uint32_t end = cpu_hal_get_cycle_count();  // Use CPU cycle count f
    return end - start;  // Return the difference in cycles
}

// Function to calculate distance from pulse duration in cycles
float calculate_distance(uint32_t pulse_duration_cycles) {
    // Convert cycles to microseconds
    float duration_us = (float)pulse_duration_cycles / (CPU_FREQ_MHZ *
    float distance = duration_us * 0.0343 / 2;  // Calculate distance i
    return distance * calibration_factor;  // Apply calibration factor
}

// Function to calculate average distance from multiple samples
float get_average_distance() {
    uint32_t total_duration = 0;
    for (int i = 0; i < NUM_SAMPLES; i++) {
        send_trigger_pulse();
        total_duration += measu _echo_pulse();
```



```c
        vTaskDelay(50 / portTICK_PERIOD_MS);  // Small delay between sa
    }
    uint32_t average_duration = total_duration / NUM_SAMPLES;
    return calculate_distance(average_duration);
}

// Function to perform initial calibration
void calibrate_sensor() {
    // Set up known distances and corresponding actual measured values
    float known_distances[] = {10.0, 20.0, 30.0};  // Known distances i
    float measured_distances[3];

    // Measure distances for the known distances
    for (int i = 0; i < 3; i++) {
        send_trigger_pulse();
        measured_distances[i] = get_average_distance();
    }

    // Calculate the calibration factor
    float total_calibration_factor = 0.0;
    for (int i = 0; i < 3; i++) {
        total_calibration_factor += known_distances[i] / measured_dista
    }
    calibration_factor = total_calibration_factor / 3.0;
}

void app_main(void)
{
    esp_rom_gpio_pad_select_gpio(TRIGGER_GPIO);
    gpio_set_direction(TRIGGER_GPIO, GPIO_MODE_OUTPUT);
    esp_rom_gpio_pad_select_gpio(ECHO_GPIO);
    gpio_set_direction(ECHO_GPIO, GPIO_MODE_INPUT);

    // Perform initial calibration
    calibrate_sensor();

    while (1) {
        float distance = get_average_distance();
        printf("Distance: %.3f cm\n", distance);
        vTaskDelay(1000 / portTICK_PERIOD_MS);  // Wait for 1 second be
    }
}
```

“forget any previous optimizations, use my original code but modify it to include taking the temperature into account. You will include the following c file which grabs the temperature from an onboard chip.”



```
float calculate_distance(uint32_t pulse_duration, float temperature) {
    // Adjust speed of sound based on temperature: speed_of_sound = 331
    float speed_of_sound = 331.3 + 0.606 * temperature;
    // Convert speed of sound to cm/us
    float speed_of_sound_cm_us = speed_of_sound * 100 / 1000000;
    return (pulse_duration * speed_of_sound_cm_us) / 2;
}
```