INTRODUCTION & PROJECT DESCRIPTION

Embedded systems often require the simultaneous execution of multiple tasks. This lab focuses on the the implementation of concurrent tasks using the Arduino Mega microcontroller, by using the freeRTOS library.

In the initial segment of this lab, freeRTOS is used to run three parallel tasks:

1. Flashing an LED.
2. Sequentially playing the Mario theme song three times before termination.
3. Executing an FFT (Fast Fourier Transform) calculation five times. To achieve this, we employed two tasks: one that sends data and retrieves the computation time, and another that processes the data through FFT and then sends the computation duration back to the initial task. Efficient data transfer between these tasks is realized using queues.

For the project portion of the lab, I designed a Bluetooth-enabled smart plant care system. This system perpetually monitors parameters such as: soil moisture, ambient temperature, and light intensity. These metrics are displayed both on an LCD screen and a smartphone paired via bluetooth.

A highlight of this system is its automatic watering mechanism. Should the soil moisture decline beneath a pre-defined limit, a stepper motor is enabled turning on a water spigot to start irrigation. Given that evening is an optimal time for plant irrigation, the system is calibrated to detect low levels of ambient light, suggestive of evening or nighttime. Under such conditions, the watering process is triggered. This mechanism ensures that by dawn, the soil retains its ideal moisture content.

An additional feature of this system is the user's ability to override the automatic watering system. Through a Bluetooth-linked smartphone, users can manually initiate the watering process. This functionality is particularly handy in circumstances where external factors like fluctuating weather patterns or specific botanical requirements necessitate additional irrigation.

METHODS AND TECHNIQUES

For the initial part of this lab, besides utilizing the freeRTOS scheduler, an internal timer interrupt was initialized. This made the playback of the Mario theme more streamlined. Specifically, the interrupt was used to toggle a flag, which subsequently incremented a counter, functioning as a sort of "pseudo-clock".

In the project portion of the lab, the system exclusively relied on freeRTOS without the integration of a timer. Given the large number of external peripherals in the plant management system, the implementation of a Boolean flag system became necessary. These flags facilitate inter-task communication, signaling when to execute specific subtasks within a primary task. It is important to clarify that these "subtasks" are not standalone tasks; they are conditional statements influenced by the Boolean flags.

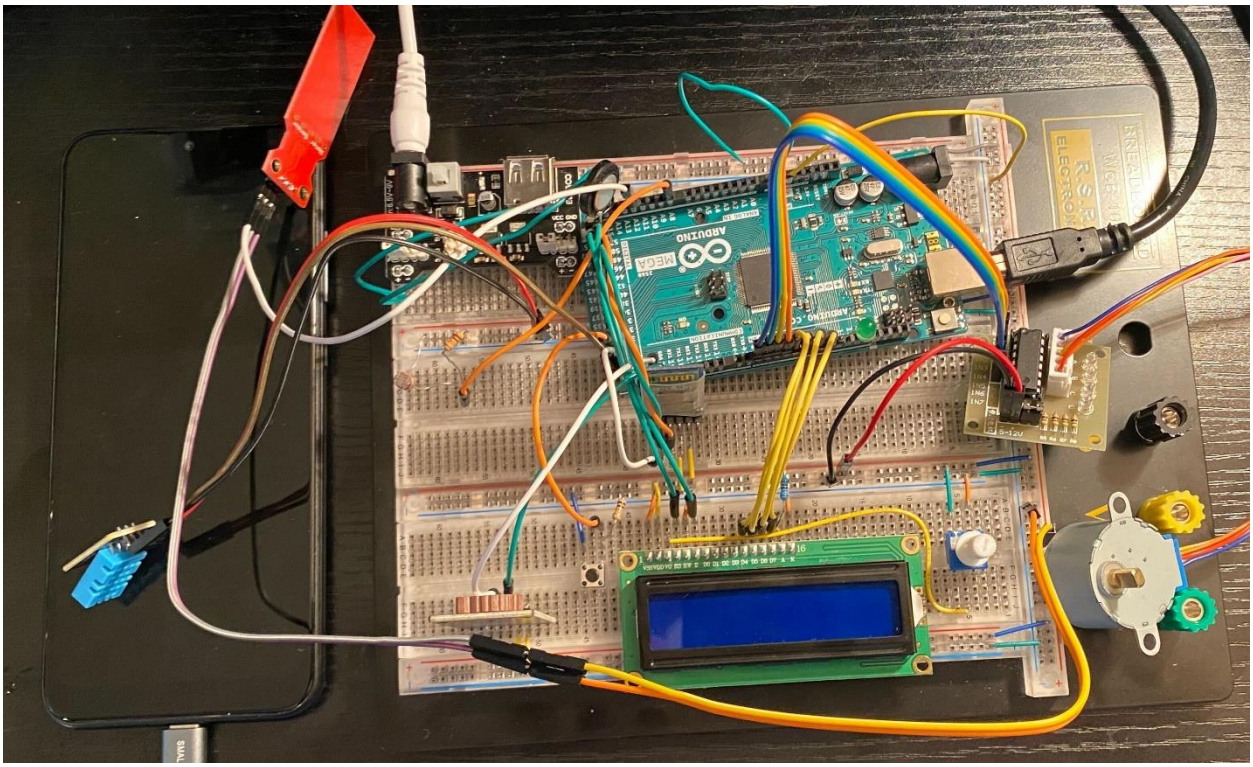

Figure 1 The experimental setup for this lab

EXPERIMENTAL RESULTS

PART B

In Part B, a timer interrupt facilitated the playback of the Mario theme song. Still, its operation was governed by a task managed by the freeRTOS scheduler. All other tasks were exclusively controlled by the freeRTOS scheduler without relying on interrupts. Much of the code for this segment was adapted from examples presented during lectures and lab sessions.

Initially, I crafted separate .ino files to verify the independent functionality of each task. Once confirmed, I consolidated them into a single .ino file that combined all tasks. Running this integrated file demonstrated correct performance as outlined in the lab instructions, showing simultaneous FFT computations, Mario theme song playback, and the continuous flashing of the LED.

FINAL PROJECT – Bluetooth Enabled Garden Management System

For the final project, I exclusively used the freeRTOS scheduler to manage all tasks. To enhance code efficiency and readability, I integrated numerous helper functions.

My first determination involved deciding which features should be governed by the scheduler and which should function as helper functions. To streamline this, I decided that all peripherals would be managed as tasks by the freeRTOS scheduler, while other functionalities, such as reading pins or printing to serial buffers, would serve as helper functions. The control logic for the system resided within these freeRTOS tasks.

Throughout the project's development, it became evident that tasks required multiple communication avenues. To address this, I incorporated global Boolean flags to enable efficient inter-

task communication. Unlike Part B, which had a linear data flow, this project saw each task receiving input from several others, leading to web-like intertask communication. This complexity made exhaustive testing challenging.

Nevertheless, I examined critical functions and scenarios. For instance, I checked how the system behaves when manual watering is toggled during automatic watering (it remained unaffected as anticipated) and whether manual watering can be initiated when automatic watering isn't active (it can). I also verified that manual watering could be activated both when the automated system was idle and active but not currently watering. Another significant test involved assessing the system's ability to distinguish between night and day for watering, and it successfully differentiated the two. While I conducted numerous tests, the intricate nature of this system and time constraints mean not every scenario could be examined. There may be untested cases that could disrupt and/or break the system.

In conclusion, the primary functions, as outlined in the project proposal, operated as planned. Given this, I consider the system successful in meeting the goals I set for myself.

CODE DOCUMENTATION

***NOTE – DOXYGEN DOCUMENTATION CAN BE FOUND IN "DOCUMENTATION" FOLDER WITHIN THE "PART_B" FOLDER AND THE "PART_C" FOLDER**

PART B

```
/**
 * @file PART_B.ino
 * @brief A program to flash an external LED, play the mario theme, and complete
FFT calcs using freeRTOS implemented on the Arudino Mega.
 * @author Jason Bentley
 * @date 8/16/2023
 *
 * @note acknowledgments
 *  - starter code from class
 *
 *
 * The file includes the following modules:
 * - @link group1 INITIALIZATION @endlink
 * - @link group2 TASKS @endlink
 * - @link group3 HELPER FUNCTIONS @endlink
 */


#include <Arduino_FreeRTOS.h>
#include <arduinoFFT.h>
#include <time.h>
#include "task.h"
#include "queue.h"
```

```
#define NOTE_E 659
#define NOTE_C 523
#define NOTE_G 784
#define NOTE_g 392
#define NOTE_R 0

#define TONE_PIN 52

#define NOTE_TIME  1000 //100ms

int sFlag = 0;
const int maxIterations = 3;
int iteration = 0;

long tune_timer   = 0; /*!<counter to store length of time which speaker has been
in a particular state*/
int tune_elem     = 0; /*!<counter to store the the position of the current note
in the mario[] array*/
int tune_state    = 0; /*!<counter to store the current state of the speaker*/
int note_timer    = 0; /*!<counter to store the length of time which a note has
been playing*/

int mario[] = {NOTE_E, NOTE_R, NOTE_E, NOTE_R, NOTE_R, NOTE_E, NOTE_R, NOTE_R,
NOTE_C, NOTE_R,
               NOTE_E, NOTE_R, NOTE_R, NOTE_G, NOTE_R, NOTE_R, NOTE_R, NOTE_R,
NOTE_R, NOTE_g,
               NOTE_R};

const long     samples   = 128;
const double   frequency = 5000;

double vRealRecieve[samples];
double vImag[samples];
double vRealSend[samples];

TaskHandle_t RT3, RT4;

QueueHandle_t q1 = xQueueCreate(samples, sizeof(double) * samples);
QueueHandle_t q2 = xQueueCreate(1, sizeof(TickType_t));



/**
 * @defgroup group1 INITIALIZATION
 * @{
```

```
 */

/**
 * @brief This function creates all tasks necessary to run this program as well
as intilizing TIMER4 interupt in oder to play the mario theme
 * @param none
 * @note see @link group2 TASKS @endlink for more information on tasks
 */
void setup() {

  for(int i = 0; i < samples; i++){
    vImag[i] = 0.0;
  }

  // initialize serial communication at 9600 bits per second:
  Serial.begin(19200);

  while (!Serial) {
    ; // wait for serial port to connect. Needed for native USB, on LEONARDO,
MICRO, YUN, and other 32u4 based boards.
  }

  TCCR4A  = 0;
  TCCR4B  = (1 << WGM42) | (1 << CS41) | (1 << CS40);
  OCR4A   = 25;
  TIMSK4 |= (1 << OCIE4A);
  sei();

  xTaskCreate(TaskBlink, "Blink", 128, NULL, 3, NULL);
  xTaskCreate(TaskSong, "Song", 128, NULL, 2, NULL);
  xTaskCreate(TaskRT3,"RT3", 256, NULL, 3, &RT3 );
  xTaskCreate(TaskRT4, "RT4", 256, NULL, 3, &RT4);


  vTaskStartScheduler();
}

/**
 * @brief Empty loop function necessary to compile *.ino file
 * @param none
 * @note Scheduling is handled by freeRTOS scheduler
 */
void loop(){}
/** @} */ // end of group1
```

```
/*-----------------------------------------------------*/
/*---------------------- Tasks ----------------------*/
/*-----------------------------------------------------*/
/**
 * @defgroup group2 TASKS
 * @{
 */

/**
 * @brief This task blinks and LED on for 100ms and off for 200ms
 * @param pvParameters pointer to allocate stack space for params
 */
void TaskBlink(void *pvParameters)
{
  pinMode(10, OUTPUT);

  for (;;)
  {
    digitalWrite(10, HIGH);
    vTaskDelay( 100 / portTICK_PERIOD_MS );
    digitalWrite(10, LOW);
    vTaskDelay( 200 / portTICK_PERIOD_MS );
  }
}


/**
 * @brief This task uses the interupt timer to play the mario theme song
 * @param pvParameters pointer to allocate stack space for params
 */
void TaskSong(void *pvParameters)
{
  pinMode(TONE_PIN, OUTPUT);

  while (iteration < maxIterations){
    playSpeaker();

    if (sFlag == 1){
      tune_timer++;
      note_timer++;
      sFlag = 0;
    }

  }
  vTaskDelete(NULL);
```

```
}

/**
 * @brief This task sends data to TaskRT4 and recieves computation time data from
TaskRT4
 * @param pvParameters pointer to allocate stack space for params
 */
void TaskRT3(void *pvParameters){

    TickType_t totalTime = 0, cTimeRecieve = 0;

    for(int i = 0; i < 5; i++){
        Serial.println();
        Serial.print("itteration: "); Serial.println(i+1);

        unsigned int tick = xTaskGetTickCount() + 1;
        srand(tick);

        for(int i = 0; i < samples; i++){
            vRealSend[i] = (double)rand() / RAND_MAX * (100) + 100;
        }

        if(xQueueSend(q1, &vRealSend, pdMS_TO_TICKS(100)) == pdPASS){
            Serial.println("Data Sent Successfully!");
            vTaskResume(RT4);
            vTaskSuspend(RT3);
        } else{
            Serial.println("Data Sending Failed!");
        }

        if(xQueueReceive(q2, &cTimeRecieve, portMAX_DELAY) == pdTRUE){
            totalTime = totalTime + cTimeRecieve;
            Serial.println("Computation Time Recieved Successfully");
        }else{
            Serial.println("Computation Time Recieving Failure!");
        }
    }
    Serial.println();
    Serial.print("WALL CLOCK TIME ELAPSED FOR 5 FFTs: ");
    Serial.println(totalTime * 1000 / configTICK_RATE_HZ);
    vTaskDelete(RT4);
    vTaskDelete(NULL);
}
```

```
/**
 * @brief This task recieves data from TaskRT3 compeltes an FFT and then sends computation time data to TaskRT3
 * @param pvParameters pointer to allocate stack space for params
 */
void TaskRT4(void *pvParameters){

    TickType_t cTimeSend = 0, startTime = 0, endTime = 0;

    while(1){
    if(xQueueReceive(q1, &vRealRecieve, portMAX_DELAY) == pdTRUE){
        Serial.println("Data Recieved, now processing...");

        startTime = xTaskGetTickCount();

        arduinoFFT FFT = arduinoFFT(vRealRecieve, vImag, samples, frequency);
        FFT.Windowing(FFT_WIN_TYP_HAMMING, FFT_FORWARD);
        FFT.Compute(FFT_FORWARD);

        endTime = xTaskGetTickCount();
        cTimeSend = endTime - startTime;
        Serial.println("Succesfully completed FFT...");
        Serial.println("Attempting to send data...");
    }else{
        Serial.println("Data for FFT was not recieved!");
    }

    if(xQueueSend(q2, &cTimeSend, pdMS_TO_TICKS(100) == pdPASS)){
        Serial.println("Time data was sent...");
        vTaskResume(RT3);
        vTaskSuspend(RT4);
    }else{
        Serial.println("Time data failed to send!");
    }
    }

}
/** @} */ // end of group2


/**
 * @defgroup group3 HELPER FUNCTIONS
 * @{
 */
```

```
/**
 * @brief This function controls which notes are played and for how long
 */
void playSpeaker() {
  //calculate the half period of a frequency
  int period_half = 5000/(mario[tune_elem] + 1);

  //if the speaker is off
  if(tune_state == LOW && tune_timer == period_half && tune_elem != 21){
    digitalWrite(TONE_PIN, HIGH);
    tune_timer = 0;
    tune_state = HIGH;
    return;
  }

  //if the speakers it on
  if(tune_state == HIGH && tune_timer == period_half && tune_elem != 21){
    digitalWrite(TONE_PIN, LOW);
    tune_timer = 0;
    tune_state = LOW;
    return;
  }

  //increment to next note if current note has played for NOTE_TIME amount of
time
  if(note_timer == NOTE_TIME && tune_elem != 21){
    tune_elem++;
    note_timer = 0;
    tune_timer = 0;
    return;
  }

  //if all notes have been played wait 4sec, then go back to beginning of tune[]
array
  if(tune_elem == 21 && tune_timer == 15000){
    tune_elem = 0;
    tune_timer = 0;
    note_timer = 0;
    iteration++;
    return;
  }
}

/**
 * @brief This interupt raises a flag on interupt
```

```
 */
ISR(TIMER4_COMPA_vect) {
  sFlag = 1;
  TIFR4 |= (1 << OCF4A);
}
/** @} */ // end of group3
```

FINAL PROJECT

```
/**
 * @file GardenManagmentSystem.ino
 * @brief A program to control an automated, bluetooth enabled irregation and
plant monnitoring system using freeRTOS implemented on the Arudino Mega.
 * @author Jason Bentley
 * @date 8/16/2023
 *
 * @note acknowledgments:
 * - Paul Stroffregen (AltSoftSerial)
 * - Maykon L. Capellari (ButtonDebounce)
 * - Mike McCauley (AccelStepper)
 * - Naguissa (uRTCLib)
 * - Adafruit (dht)
 *
 * The file includes the following modules:
 * - @link group1 INITIALIZATION @endlink
 * - @link group2 TASKS @endlink
 * - @link group3 HELPER FUNCTIONS @endlink
 */

//RTOS LIBRARIES
#include <Arduino_FreeRTOS.h>
#include "task.h"

//PERIPHERAL IO LIBRARIES
#include <AltSoftSerial.h>
#include <LiquidCrystal.h>
#include <ButtonDebounce.h>
#include <AccelStepper.h>
#include <uRTCLib.h>
#include <Arduino.h>
#include <dht.h>

//MOISTURE SENSOR PARAMS
#define DRENCHED     270  /*!< out of 1023 -> drenched soil threshold */
```

```
#define WET          230  /*!<out of 1023 -> wet soil threshold*/
#define ADEQUATE     190  /*!<out of 1023 -> adequate moisture soil threshold*/
#define SOMEWHAT_DRY 140  /*!<out of 1023 -> somewhat dry soil threshold*/
#define DRY          70   /*!<out of 1023 -> dry soil threshold*/
#define MOISUTRE_PIN A12
int moistureReading = 0;      /*!<store the soil mositure value*/
bool userCancel     = false;  /*!<a bool flag used to enable or disable auto
irregation. false = enable, true = disable*/
bool initFlag       = false;  /*!<a bool flag used to stop complex tasks from
running until simpler tasks have had a chance to complete intialization*/

//PUSH BUTTON PARAMS
#define PUSH_BUTTON_DIGITAL_PIN         29
#define DEBOUNCE_INTEGRATION_TIME_MILLI 10 /*!<push button delay before another
value is read*/

//TEMP AND HUMIDITY SENSOR PARAMS
#define DHT11_PIN  23
int dht_humidity = 0; /*!<store humidity data*/
int dht_temp     = 0; /*!<store temperature data*/

//STEPPER MOTOR PARAMS
#define MOTOR_TYPE  4        /*!<number of coils in the stepper motor being
used*/
bool water_flag     = false; /*!<a bool flag to indicate whether or not the
system should be irrigating*/
bool manual_control = false; /*!<a bool flag which overides all other flags and
will force the system to water if true*/


//BLUETOOTH PARAMS
#define TX_PIN   14
#define RX_PIN   15
#define BT_PIN   22   /*!<status pin communicates to arduino whether or not a
bluetooth connection has been established*/
#define BT_BAUD  9600 /*!<baud rate of bluetooth serial connection (HC-05/06
standard is 9600)*/
#define GET_INFO '1'
#define STOP     '3'
#define WATER    '2'
#define AUTO     '4'
char data         = ' ';    /*!<a variable to hold char recieved from bluetooth
serial buffer*/
bool BT_connected = false;  /*!<a bool for bluetooth connection status. false =
disconected, true = connected*/
```

```
//LED PARAMS
#define RS_PIN 50
#define E_PIN  51
#define D4_PIN 6
#define D5_PIN 7
#define D6_PIN 8
#define D7_PIN 9

//RTC PARAMS
#define RTC_I2C_ADR 0x68
char daysOfTheWeek[7][12] = {"Sunday", "Monday", "Tuesday", "Wednesday",
"Thursday", "Friday", "Saturday"};
uint8_t rtcData[5];

//LIGHT SENSE PARAMS
#define DARK_THRESHOLD 500
#define LIGHT_PIN      A8
int lightReading    = 0;
bool nightWaterFlag = false; /*!<a bool flag to store whether or not to enable
night time watering. false = disable, true = enable*/


TaskHandle_t bluetoothIO, displayDataOnLED, waterControlStepperMotor,
moistureMeasure, rtcIO, lightMeasurement, HumidityAndTemp;
ButtonDebounce pushButton(PUSH_BUTTON_DIGITAL_PIN,
DEBOUNCE_INTEGRATION_TIME_MILLI);
AltSoftSerial BT;
LiquidCrystal lcd(RS_PIN, E_PIN, D4_PIN, D5_PIN, D6_PIN, D7_PIN);


//================================================//
//====       I N I T I A L I Z A T I O N       ====//
//================================================//
/**
 * @defgroup group1 INITIALIZATION
 * @{
 */


/**
 * @brief This function creates all tasks necessary to run this program
 * @param none
 * @note see @link group2 TASKS @endlink for more information on tasks
 */
```

```
void setup(){
        xTaskCreate(Task_BluetoothIO, "BluetoothIO", 256, NULL, 3, &bluetoothIO);
        xTaskCreate(Task_displayDataOnLED, "displayDataOnLED", 256, NULL, 3,
&displayDataOnLED);
        xTaskCreate(Task_waterControlStepperMotor, "waterControlStepperMotor",
256, NULL, 2, &waterControlStepperMotor);
        xTaskCreate(Task_moistureMeasure, "moistureMeasure", 256, NULL, 2,
&moistureMeasure);
        xTaskCreate(Task_rtcIO, "rtcIO", 256, NULL, 2, &rtcIO);
        xTaskCreate(Task_lightMeasurement, "lightMeasurement", 256, NULL, 2,
&lightMeasurement);
        xTaskCreate(Task_HumidityAndTemp, "HumidityAndTemp", 256, NULL, 2,
&HumidityAndTemp);
        xTaskCreate(Task_Blink, "Blink", 128, NULL, 3, NULL);
        vTaskStartScheduler();
}

/**
 * @brief Empty loop function necessary to compile *.ino file
 * @param none
 * @note Scheduling is handled by freeRTOS scheduler
 */
void loop(){}
/** @} */ // end of group1


//================================================//
//====               T A S K S                ====//
//================================================//
/**
 * @defgroup group2 TASKS
 * @{
 */

/**
 * @brief This task intitates and controls the bluetooth connection. Information
receeived by
 * this task from a bluetooth device can control the physical peripherals of the
Garden Managment System.
 * @param pvParameters pointer to allocate stack space for params
 */
void Task_BluetoothIO(void *pvParameters){
        pinMode(BT_PIN, INPUT);
        BT.begin(BT_BAUD);
```

```
        //Check bluetooth connection, confirm connection on
        //BT connected device when connection is suceesful
        while (!BT_connected){
                if(digitalRead(BT_PIN) == HIGH){
                        BT_connected = true;
                        BT.println();
                        BT.println("Arduino connected");
                        BT.println("Initializing Tasks...");
                }
                vTaskDelay(100/portTICK_PERIOD_MS);
        }

        vTaskDelay(1000/portTICK_PERIOD_MS); //1 second delay to allow all tasks
to intialize
        BT.println("Tasks intialized...");

        for(;;){
                //if BT connection fails, search for new connection
                // and do nothing else until new connection is found
                if(digitalRead(BT_PIN) == LOW){
                        bool BT_connected = false;
                        while (!BT_connected){
                                vTaskDelay(20/portTICK_PERIOD_MS);
                                if(digitalRead(BT_PIN) == HIGH){
                                        BT_connected = true;
                                        BT.println();
                                        BT.println("Arduino connected");
                                }
                        }
                }

                //Display all information on bluetooth device
                if (BT.available()){
                        data = BT.read();
                        if (data == GET_INFO){
                                BT.println();
                                BTprintDate();
                                BTprintTime();
                                BTprintTempAndHumidity();
                                BTprintSoilMoisture();
                                BTprintLight();
                                BTprintAuto();

                        //Enable auto watering function
                        } else if (data == AUTO){
```

```
                                userCancel = false;
                                BT.println();
                                BT.println("Auto watering ENABLED");

                        //Disable auto watering function
                        } else if (data == STOP){
                                userCancel = true;
                                BT.println();
                                BT.println("Auto watering DISABLED");

                        //Enable manual watering function
                        } else if (manual_control == false && data == WATER){
                                manual_control = true;
                                BT.println();
                                BT.println("Manual watering in progress");

                        //Disable manual watering function
                        } else if (manual_control == true && data == WATER){
                                manual_control = false;
                        }
                }
        vTaskDelay(100/portTICK_PERIOD_MS); //100ms delay
        }
}



/**
 * @brief This task displays data on the LCD and controls the pushbutton logic.
The physical
 * push button scrolls between different data readouts or has other functionality
if
 * prompted on the LCD.
 * @param pvParameters pointer to allocate stack space for params
 */
void Task_displayDataOnLED(void *pvParameters){

        int button_counter    = 0;  //stores which screen the LCD should display
(out of 3 possible)
        int prev_button_state = LOW;//dummy variabel to hold which screen was
previously displayed

        lcd.begin(16,2);
        lcd.clear();
        lcd.setCursor(0,1);
```

```
        lcd.print("initializing...");

        vTaskDelay(3000/portTICK_PERIOD_MS); //3 second delay to allow other
tasks to initilize

        initFlag = true; //allow uninitialized tasks to begin running after other
tasks have finished initilizing

        for(;;){

                //logic to display various data on LCD
                if (button_counter == 0){
                        lcdPrintTimeAndDate();
                }else if (button_counter == 1){
                        lcdPrintTempAndHumidity();
                }else if (button_counter == 2){
                        lcdPrintLightAndMosiTure();
                }else if (button_counter >= 3){
                        button_counter = 0;
                }

                //pushbutton logic
                if(getButton() == HIGH && prev_button_state == LOW){
                        button_counter++;
                        prev_button_state = HIGH;
                }else if (getButton() == LOW){
                        prev_button_state = LOW;
                }


                vTaskDelay(50 / portTICK_PERIOD_MS); //50ms delay
                vTaskResume(HumidityAndTemp);
                vTaskResume(rtcIO);
                vTaskResume(moistureMeasure);
                vTaskResume(lightMeasurement);
        }
}


/**
 * @brief This task controls the servo motor which turns a water spigot on or off
 * depending on data recieved (via global bool flags) from other tasks.
 * @param pvParameters pointer to allocate stack space for params
 */
void Task_waterControlStepperMotor(void *pvParameters){
```

```
        AccelStepper myStepper(MOTOR_TYPE, 2, 4, 3, 5);

        myStepper.setCurrentPosition(0);
        myStepper.setMaxSpeed(200);
        myStepper.setAcceleration(100);

        vTaskSuspend(waterControlStepperMotor); //suspend task until is it
resumed in another task

        for(;;){
                if(water_flag){
                        myStepper.moveTo(2000);
                        myStepper.run();
                        if(myStepper.currentPosition() >= 2000){
                                myStepper.disableOutputs();
                        }
                        vTaskResume(moistureMeasure);

                } else if (!water_flag){
                        water_flag = false;
                        vTaskResume(bluetoothIO);

                        while(myStepper.currentPosition() != 0){
                                myStepper.moveTo(0);
                                myStepper.run();
                        }

                        myStepper.disableOutputs();
                        vTaskResume(displayDataOnLED);
                        vTaskSuspend(waterControlStepperMotor);
                }
                vTaskDelay(10/portTICK_PERIOD_MS); //10ms delay to allow smoother
stepper response
        }
}


/**
 * @brief This task measures the soil mositure and recieves data from other tasks
to control the
 * irregation logic of the system.
 * @param pvParameters pointer to allocate stack space for params
 */
void Task_moistureMeasure (void *pvParameters){
        bool enable = false;
```

```
	for(;;){
		moistureReading = getMoisture();

		//if all other tasks are initialized and watering is not enabled
		if(initFlag  && !enable){
			vTaskSuspend(displayDataOnLED);

			//if auto watering is turned on and soil mositure is DRY
			if(!userCancel && moistureReading <= DRY){
				enable = true;
				water_flag = true;
				startWateringAlertMessages();

			//if auto watering is tunred on and it is night time
			} else if (!userCancel && nightWaterFlag){
				enable = true;
				water_flag = true;
				nightWateringAlertMessages();

			//if watering is turnbed on manually
			} else if (manual_control){
				enable = true;
				water_flag = true;
				manualWateringAlertMessages();
			}
			vTaskResume(lightMeasurement);
			vTaskResume(waterControlStepperMotor);
		}

		vTaskResume(lightMeasurement);

		//If watering is currently enabled
		if(enable){

			//If soil is DRENCHED or it is not night time and soil is
not dry and manual watering is not turned on
			if((moistureReading >= DRENCHED || (!nightWaterFlag &&
moistureReading > DRY)) && !manual_control ){
				water_flag = false;
				enable     = false;
				finishWateringAlertMessages();

			//if there is data on the bluetooth buffer or the
pushbutton has been pushed
```

```
                        } else if(BT.available() || getButton() == HIGH){
                                data = BT.read();

                                //disbale watering and disable auto watering if
not manually enabled
                                if ((data == STOP || getButton() == HIGH) &&
!manual_control){
                                        water_flag = false;
                                        userCancel = true;
                                        enable     = false;
                                        cancelWateringAlertMessages();
                                        BT.println("Auto Watering Disabled");

                                //if manually enabled then only disable watering
but do not change state of auto watering
                                } else if ((data == WATER || getButton() == HIGH)
&& manual_control){
                                        water_flag     = false;
                                        manual_control = false;
                                        enable         = false;
                                        cancelWateringAlertMessages();

                                }
                        }
                }
        }

        vTaskDelay(100 / portTICK_PERIOD_MS); //100ms second delay
        vTaskSuspend(moistureMeasure);
}


/**
 * @brief This task reads and stores the rtc modules clock data.
 * @param pvParameters pointer to allocate stack space for params
 * @note the rtc can be reconfigured from inside this task during startup but
will require a restart of the system.
 */
void Task_rtcIO(void *pvParameters){
        uRTCLib rtc(RTC_I2C_ADR);
        URTCLIB_WIRE.begin();
        // 1. uncomment "rtc.set(40, 20, 17, 3, 8, 8, 23);" and change to desired
values
        // 2. upload the code to the arduino.
        // 3. comment out rtc.set() after setting desired date and time.
```

```
        // ***if you do not comment out the code the rtc will reset to those
values every time***

        // rtc.set(40, 20, 17, 3, 8, 8, 23);

        // *rtc.set(second, minute, hour, dayOfWeek, dayOfMonth, month, year)*
        // *set day of week (1=Sunday, 7=Saturday)*

        for(;;){
                rtc.refresh();
                rtcData[0] = rtc.year();
                rtcData[1] = rtc.month();
                rtcData[2] = rtc.day();
                rtcData[3] = rtc.hour();
                rtcData[4] = rtc.minute();
                vTaskDelay(50 / portTICK_PERIOD_MS); //50ms second delay
                vTaskSuspend(rtcIO);
        }

}


/**
 * @brief This task measures the light and and recieves mositure data to control
the nighttime watering logic.
 * @param pvParameters pointer to allocate stack space for params
 */
void Task_lightMeasurement(void *pvParameters){
        for(;;){
             lightReading = getLight();

             if(lightReading <= DARK_THRESHOLD && moistureReading < WET){
                nightWaterFlag = true;
             } else if (lightReading <= DARK_THRESHOLD && moistureReading >=
DRENCHED){
                nightWaterFlag = false;
                water_flag = false;
             } else if (lightReading >= DARK_THRESHOLD){
                nightWaterFlag = false;
             }

             vTaskDelay(50 / portTICK_PERIOD_MS);
             vTaskSuspend(lightMeasurement);
        }
}
```

```
/**
 * @brief This task measures and stores the air temperature and humidity
information
 * @param pvParameters pointer to allocate stack space for params
 */
void Task_HumidityAndTemp(void *pvParameters){
        dht DHT;
        for(;;){
                DHT.read11(DHT11_PIN);
                dht_humidity = DHT.humidity;
                dht_temp     = DHT.temperature;
                vTaskDelay(50 / portTICK_PERIOD_MS); //50ms second delay
                vTaskSuspend(HumidityAndTemp);
        }
}


/**
 * @brief This task blinks and LED on for 100ms and off for 200ms
 * @param pvParameters pointer to allocate stack space for params
 */
void Task_Blink(void *pvParameters){  // This is a task.
  pinMode(10, OUTPUT);
  for (;;)
  {
    digitalWrite(10, HIGH);   // turn the LED on (HIGH is the voltage level)
    vTaskDelay( 100 / portTICK_PERIOD_MS ); // wait for 100ms
    digitalWrite(10, LOW);    // turn the LED off by making the voltage LOW
    vTaskDelay( 200 / portTICK_PERIOD_MS ); // wait for 200ms
  }
}
/** @} */ // end of group2



//================================================//
//====     H E L P E R   F U N C T I O N S     ====//
//================================================//
/**
 * @defgroup group3 HELPER FUNCTIONS
 * @{
 */
```

```
/**
 * @brief Reads the ADC value of the photoresistor
 * @return ADC value of the photoresistor
 */
int getLight(){
        return analogRead(LIGHT_PIN);
}


/**
 * @brief Reads the ADC value of the water sensor
 * @return ADC value of the water sensor
 */
int getMoisture(){
        return analogRead(MOISUTRE_PIN);
}


/**
 * @brief Reads the digital value of the pushbutton
 * @return pushbutton state (HIGH or LOW)
 */
int getButton(){
        pushButton.update();
        return pushButton.state();
}


/**
 * @brief Prints the Date to the bluetooth serial buffer
 */
void BTprintDate(){
        BT.print("Date: ");
        if (rtcData[1] < 10){
                BT.print("0");
        }
        BT.print(rtcData[1]); BT.print("/");
        if (rtcData[2] < 10){
                BT.print("0");
        }
        BT.print(rtcData[2]); BT.print("/");
        BT.print("20"); BT.println(rtcData[0]);
}


/**
```

```
 * @brief Prints the Time to the bluetooth serial buffer
 */
void BTprintTime(){
        BT.print("Time: ");
        BT.print(rtcData[3]); BT.print(":");
        if (rtcData[4] < 10){
                BT.print("0");
        }
        BT.println(rtcData[4]);
}


/**
 * @brief Prints the Temp and Humidty to the bluetooth serial buffer
 */
void BTprintTempAndHumidity(){
        BT.print("Air Temp: ");
        int temp_fer = round((dht_temp * 9/5) + 32); //convert from celcius to
ferinheight
        BT.print(temp_fer);
        BT.print(char(176)); //ASCII degree symbol
        BT.println("F");

        BT.print("Air Humidity: ");
        BT.print(dht_humidity);
        BT.println("%");
}


/**
 * @brief Checks the soil mositure level and prints appropriate message to the
bluetooth serial buffer
 */
void BTprintSoilMoisture(){
        BT.print("Soil Moisture: ");
        if(moistureReading <= DRY){
                BT.println("WATER IMMMEDIATELY!!!");
        } else if (moistureReading > DRY && moistureReading <= SOMEWHAT_DRY){
                BT.println("Somewhat Dry - Watering Recommended");
        } else if (moistureReading > SOMEWHAT_DRY && moistureReading <=
ADEQUATE){
                BT.println("Ideal - No Watering Necessary");
        } else if (moistureReading > ADEQUATE && moistureReading <= DRENCHED){
                BT.println("Very Wet - DO NOT WATER!!!");
        }else if (moistureReading > DRENCHED){
```

```
                BT.println("DRENCHED - STOP WATERING IMMEDIATELY!!!");
        }
}


/**
 * @brief Checks the light levels and prints the appropriate message to the
bluetooth serial buffer
 */
void BTprintLight(){
        BT.print("Light Levels: ");
        if(lightReading <= DARK_THRESHOLD){
                BT.println("DARK");
        } else if (lightReading > DARK_THRESHOLD){
                BT.println("BRIGHT");
        }

}


/**
 * @brief Checks the state of automatic watering and prints the appropriate
message to the bluetooth serial buffer
 */
void BTprintAuto(){
        BT.print("Automatic Watering: ");
        if(userCancel){
                BT.println("DISABLED");
        } else if (!userCancel){
                BT.println("ENABLED");
        }

}


/**
 * @brief Prints the Date and Time to the LCD display
 */
void lcdPrintTimeAndDate(){
        lcd.clear();
        lcd.setCursor(0,0);
        lcd.print("Time: ");
        lcd.print(rtcData[3]); lcd.print(":");
        if (rtcData[4] < 10){
                lcd.print("0");
```

```
        }
        lcd.print(rtcData[4]);
        lcd.setCursor(0,1);
        lcd.print("Date: ");
        if (rtcData[1] < 10){
                lcd.print("0");
        }
        lcd.print(rtcData[1]); lcd.print("/");
        if (rtcData[2] < 10){
                lcd.print("0");
        }
        lcd.print(rtcData[2]); lcd.print("/");
        lcd.print("20"); lcd.println(rtcData[0]);
}


/**
 * @brief Prints the temperature and humidity to the LCD display
 */
void lcdPrintTempAndHumidity(){
        lcd.clear();
        lcd.setCursor(0,0);
        lcd.print("Air Temp: ");
        int temp_fer = round((dht_temp * 9/5) + 32); //convert C to F
        lcd.print(" ");
        lcd.print(temp_fer);
        lcd.print((char)223); //degree symbol for the LCD display
        lcd.print("F");
        lcd.setCursor(0,1);
        lcd.print("Air Humid: ");
        lcd.print(dht_humidity);
        lcd.print("%");
}


/**
 * @brief Prints the light levels and moisture levels to the LCD display
 */
void lcdPrintLightAndMosiTure(){
        lcd.clear();
        lcd.setCursor(0,0);
        lcd.print("Light: ");
        if(lightReading <= DARK_THRESHOLD){
                lcd.print("DARK");
        } else if (lightReading > DARK_THRESHOLD){
```

```
                lcd.print("BRIGHT");
        }
        lcd.setCursor(0,1);
        lcd.print("Soil: ");
        if(moistureReading <= DRY){
                lcd.print(" WATER NOW");
        } else if (moistureReading > DRY && moistureReading <= SOMEWHAT_DRY){
                lcd.print(" DRY");
        } else if (moistureReading > SOMEWHAT_DRY && moistureReading <=
ADEQUATE){
                lcd.print(" IDEAL");
        } else if (moistureReading > ADEQUATE && moistureReading < DRENCHED){
                lcd.print("VERY WET");
        }else if (moistureReading >= DRENCHED){
                lcd.print("DRENCHED");
        }

}


/**
 * @brief prints alert notification to bluetooth serial and LCD for extramly dry
soil
 */
void startWateringAlertMessages(){
        BT.println();
        BT.println("!!!ALERT!!!");
        BT.println("EXTREMLY DRY SOIL... COMENCING WATERING");
        BT.println("PRESS 'STOP' to cancel");
        lcd.clear();
        lcd.setCursor(0,0);
        lcd.print("WATERING NOW");
        lcd.setCursor(0,1);
        lcd.print("BUTTON TO STOP");
}


/**
 * @brief Prints alert notification to LCD when manual watering is enabled
 */
void manualWateringAlertMessages(){
        lcd.clear();
        lcd.setCursor(0,0);
        lcd.print("Manual Watering");
        lcd.setCursor(0,1);
```

```
        lcd.print("in progress...");
}


/**
 * @brief prints alert notification to bluetooth serial and LCD when watering has
been canceled
 */
void cancelWateringAlertMessages(){
        BT.println();
        BT.println("CANCELING WATERING");
        lcd.clear();
        lcd.setCursor(0,0);
        lcd.print("WATERING");
        lcd.setCursor(0,1);
        lcd.print("CANCELED");
        BT.println();
}


/**
 * @brief prints alert notification to bluetooth serial and LCD for when night
time watering begins
 */
void nightWateringAlertMessages(){
        BT.println();
        BT.println("Beginning nightly watering");
        BT.println("PRESS 'STOP' to cancel");
        lcd.clear();
        lcd.setCursor(0,0);
        lcd.print("NIGHT H20 NOW");
        lcd.setCursor(0,1);
        lcd.print("BUTTON TO STOP");
}

/**
 * @brief prints alert notification to bluetooth serial and LCD when watering has
been completed (not canceled)
 */
void finishWateringAlertMessages(){
        BT.println();
        BT.println("Watering Complete");
        lcd.clear();
        lcd.setCursor(0,0);
        lcd.print("WATERING");
```

```
        lcd.setCursor(0,1);
        lcd.print("Complete");
}


/** @} */ // end of group3
```

OVERALL PERFORMANCE SUMMARY

The lab demonstrations were completed without any hitches, and all the code reliably executed on the Arduino. In our exhaustive testing for the final project, we did not encounter any notable issues with the system's functionality. It's worth noting, however, that the moisture meter's sensitivity occasionally skewed the results, especially when measurements were on the verge of two distinct thresholds. With more time, implementing a software-based Schmitt trigger could introduce hysteresis to the system, reducing such irregularities.

Regarding the learning objectives, I feel confident in my understanding and application of the real-time preemptive operating system, freeRTOS. Furthermore, I successfully combined software and hardware elements in my automated gardening management system, incorporating unique sensors, actuators, and a Bluetooth module.

TEAM BREAKDOWN

I am the only member on my team and did all the work.

DICUSSION AND CONCLUSION

The most challenging part of this lab was the final project. My primary struggle revolved around the intertask communication system and crafting the necessary conditional statements to ensure the system's smooth operation. While the initial design of the system was intuitive, getting the various tasks to reliably communicate amongst themselves required some fine-tuning. In a few situations, I resorted to a trial-and-error approach to achieve the desired outcomes, but these instances were exceptions rather than the norm.

Overall I found this lab to be extremely useful and rewarding. Considering my current role as an electrical development engineer at a startup, I've come to appreciate the practicality of freeRTOS even more than the concepts covered in Lab 3. However, it's worth noting that Lab 3 was instrumental in laying the foundational understanding of these systems. I am fast approaching the firmware development cycle of my startup project, and after this lab I now feel significantly more confident in my ability to successfully develop efficient test code for the target device.

In conclusion, this lab offered invaluable hands on experience with freeRTOS. I particularly enjoyed the opportunity to merge hardware and software into a single system and then to see it functioning as I had envisioned. Even though the project's inherent complexity posed some significant challenges, I feel that lessons learned and the experienced gained about freeRTOS will surely serve me well in future projects and jobs.