Lab 4 Report

Authors:

Arvin Nusalim, 2276767
Samantha Reksosamudra, 2276717

Date: Mar 9, 2024

EE/CSE 474 Winter 2024

Table of Contents

Introduction

The learning objectives in this project include implementing a real-time operating system (RTOS) to design an embedded system project. The goal is to create a system that addresses challenges in the real-world while applying a collection of engineering knowledge, skills, and creativity. Pursuing education in academia can sometimes become tiresome, however, academics still persevere through the help and support of the people around them. Hence, the inspiration for “Study Buddy,” a companion device built for a study session. This project incorporated tasks that demonstrate proficiency in managing time, scheduling tasks, and digital I/O operations. The challenge of this project was integrating different tasks and or components to work together sending and receiving data in real-time. This required an understanding of preemptive real-time task scheduling and the capability to work with novel sensors and displays.

System Design and Implementation

Part 1: Preemptive RTOS (Practice)

This section covers the design and implementation of executing multiple tasks at the same time, while running a background task at real-time. There are two tasks visible to the human: (1) flashing external LED and (2) playing a song on a passive buzzer. And a background task that computes a Fast Fourier Transform (FFT) on a random signal and prints the (real) wall clock time of the computation time. Using FreeRTOS, a real-time operating system kernel for microcontrollers, it provided simple yet robust functions that supported this project. For example, the Queue object enabled passing information back and forth between tasks in a FIFO system. The following tasks were created to achieve this goal:

- TaskBlink : flash an external LED that is on for 100 ms and off for 200 ms
- TaskPlaySong : plays a song on a passive buzzer at 1500 ms intervals
- TaskRT3p0 : generates an array of pseudo random doubles
- TaskRT3p1 : sends the array to TaskRT4 and receives a time value for 5 times
- TaskRT4 : computes FFT on the values in the array sent by TaskRT3p1 and sends it back to TaskRT3p1 once completed

Part 2: Study Buddy

This section covers the design and implementation of different hardware components controlling different tasks to operate (Study) Buddy. Each hardware component worked together to measure, calculate, and signal another component, which resulted in a synchronous operation. This project utilized the following hardware components:

- Sensors:
    - Servo motor:
        - **Wave** Buddy’s hand (if its eyes are closed)
        - It is located at Buddy’s elbow joints
    - Ultrasonic sensor:

        - **Detect** if Buddy's eyes is closed by user's hands or an object at a specific distance
        - It is located at Buddy's eyes
- User Interface:
    - Joystick module:
        - **Move up/down** to increase/decrease the desired amount of study time
        - **Click** to enter the desired amount
    - LCD display:
        - **Display** the desired amount of study time according to the user's input
        - **Display** the countdown
        - **Display** motivational quotes (if its eyes are closed)
        - It is located at Buddy's body (i.e. around the chest)
- Controller:
    - Arduino Mega:
        - Acts as the central processing unit.
        - **Reads** data from the sensors to decide how the Buddy will perform and control other background tasks

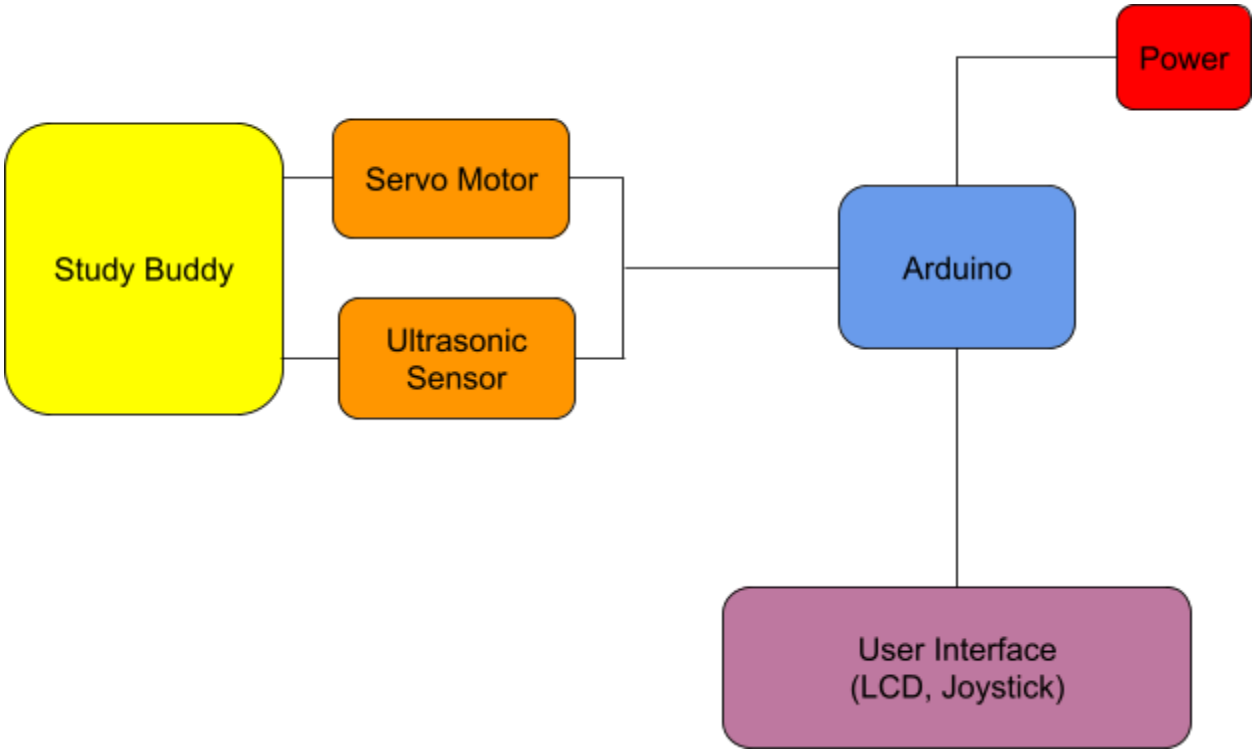

Fig.1 Flow diagram of the hardware components

Tasks that are currently not performed are suspended to achieve power efficiency, and will be resumed when they receive a signal to start. Queues were used to provide such signals, and active tasks could be halted momentarily as they waited until they received some data from a queue.

Code Structure

Part 1: Preemptive RTOS (Practice)

For the hardware timer setup, it was set to CTC mode and used a prescaler of 8 to control the passive buzzer. And two queues were created to: (1) send an array of random values from `TaskRT3p1()` to `TaskRT4()` and to (2) send the total processing time from `TaskRT4()` to `TaskRT3p1`. Also, the scheduled tasks are created using a pre-built function in FreeRTOS, which holds informations such as priority levels and task handles.

```
void setup() {
  Serial.begin(921600);
  while (!Serial) {} // Wait for the serial port to connect. Needed for native USB ports only.

  // Setup Timer 4
  TCCR4B = 0;
  TCCR4A = 0;
  TCCR4B |= (1 << WGM42) | (1 << CS41);    // set to CTC mode and prescaler 8
  TCCR4A |= (1 << COM4A0);                // toggle (buzzer) pin when TIMER4_ON_BIT reaches MAX

  // Create a queue to hold int values. Queue length is 1, and each item size is the size of int.
  queue1 = xQueueCreate(1, sizeof(int));
  queue2 = xQueueCreate(1, sizeof(int));

  // Check if the queue was created successfully
  if (queue1 == NULL || queue2 == NULL) {
    Serial.println("Queue creation failed");
    return; // Queue was not created and no memory was available
  }

  // Create tasks
  xTaskCreate(TaskBlink, "Blink", 128, NULL, 4, NULL); // Task for blinking the LED
  xTaskCreate(TaskPlaySong, "Song", 128, NULL, 4, NULL);
  xTaskCreate(TaskRT3p0, "GenerateArray", 128, NULL, 2, &RT3p0);
  xTaskCreate(TaskRT3p1, "PrintTime", 128, NULL, 4, &RT3p1);
  xTaskCreate(TaskRT4, "FFT", 2048, NULL, 4, &RT4);

  vTaskStartScheduler();
}
```

Fig.2 Setup function for Part 1

`TaskBlink()` is used to turn the external LED on for 100 ms and off for 200 ms. `vTaskDelay()` delays a task by a given number of ticks.

```
void TaskBlink(void *pvParameters) {
  pinMode(LED_PIN_T1, OUTPUT);

  for (;;) {
    digitalWrite(LED_PIN_T1, HIGH);
    vTaskDelay(100 / portTICK_PERIOD_MS);
    digitalWrite(LED_PIN_T1, LOW);
    vTaskDelay(200 / portTICK_PERIOD_MS);
  }
}
```

Fig.3 TaskBlink() function

TaskPlaySong() is used to play a song for a selected time, where the song is played three times and pauses 1.5 second between the playbacks. This task used several helper functions such as TaskPlayNote() that accesses a note (frequency) in the melody[] array and plays the frequency by setting the OCR4A value according to the frequency.

```
void TaskPlaySong(void *pvParameters) {
  pinMode(BUZZER, OUTPUT);

  for (;;) {
    if (task2_counter == 3) {
      vTaskSuspend(NULL);
    }
    // Play the song
    if (counter < (sizeof(melody) / sizeof(melody[0]))) {
      TaskPlayNote(melody[counter]);
    }
    // Else, reset counter and timestamp
    else {
      counter = 0;
      task2_counter++;
      vTaskDelay(1500 / portTICK_PERIOD_MS);
    }
  }
}
```

Fig.4 TaskPlaySong() function

In TaskRT3p0(), An array of N pseudo-random doubles is generated and it will halt itself after the array of N pseudo-random is ready. It uses a for-loop to access each slot in the N-value array, and generates a random double in the range of 1 between 10. Once it finished, the task will be suspended since it is only used once.

```
// Generates an array of N pseudo-random doubles
void TaskRT3p0(void *pvParameters) {
    static int min = 1;
    static int max = 10;

     // Serial.println("running TaskRT3p0");
     for (int i = 0; i < N; i++) {
       array[i] = random(min, max);
     }

     vTaskSuspend(NULL);

     // vTaskPrioritySet(RT3p1, 4);
     // vTaskPrioritySet(RT4, 3);
}
```

Fig.5 TaskRT3p0() function

TaskRT3p1 has a for-loop that runs 5 times and sends a pointer to the generated pseudo-random array to TaskRT4 using the Queue function. Then it waits for TaskRT4 to complete its FFT computation. After the loop, TaskRT3p1 will print the total computation time for all the FFT.

```
void TaskRT3p1(void *pvParameters) {
  TickType_t time = 0;
  TickType_t total = 0;
  for (;;) {
    for (int i = 0; i < 5; i++) {
      xQueueSend(queue1, &array, portMAX_DELAY);

      // Wait until this task is resumed
      while (xQueueReceive(queue2, &time, portMAX_DELAY) != pdPASS) {}
      total += time;
    }

    Serial.print("Total wall clock time for 5 FFTs: ");
    Serial.println(total - time);

    vTaskDelay(100 / portTICK_PERIOD_MS); // Prevent this task from hogging the CPU

    vTaskSuspend(NULL);
  }
}
```

Fig.6 TaskRT3p1 function with xQueueSend() and xQueueReceive()

TaskRT4() sets the input buffer of the FFT according to the requirements of the FFT library (e.g. real and imaginary parts). Then it waits for TaskRT3p1() to send the array data before performing FFT on the data. Then TaskRT4() will measure the computation time and send it back to TaskRT3p1() using a queue.

```
void TaskRT4(void *pvParameters) {
  int start = 0;
  uint16_t samples = N;
  double signalFrequency  = 1000;
  double samplingFrequency = 5000;   //Nyquist rate
  uint16_t N_SAMPLES = N;

  // double amplitude = 5;
  double cycles = (((samples) * signalFrequency) / samplingFrequency);
  double vReal[N_SAMPLES];
  double vImag[N_SAMPLES];
  for (;;) {
    // Receive the pointer data from Task RT3p1
    if (xQueueReceive(queue1, &array, portMAX_DELAY) == pdPASS) {

      // Start FFT on the data
      for (uint16_t i = 0; i < samples; i++) {
        vReal[i] = int8_t((array[i] * (sin((i * (twoPi * cycles)) / samples))) / 2.0);   /* Build data with positive and negative values*/
        vImag[i] = 0.0; //Imaginary part must be zeroed in case of looping to avoid wrong calculations and overflows
      }
      arduinoFFT fft = arduinoFFT(vReal, vImag, samples, samplingFrequency);

      fft.Windowing(FFT_WIN_TYP_HAMMING, FFT_FORWARD);
      TickType_t start = xTaskGetTickCount();
      fft.Compute(vReal, vImag, samples, FFT_FORWARD);
      TickType_t end = xTaskGetTickCount();
      TickType_t wallClockTime = (end-start) * (1000/configTICK_RATE_HZ);

      // Signal that FFT task has finished
      xQueueSend(queue2, &wallClockTime, portMAX_DELAY);
    }
  }
}
```

Fig.7 TaskRT4() function to perform FFT

Part 2: Study Buddy

The setup involves the creation of required queues to send data from one task to another, task initialization, and digital pin initializations for servo motor, ultrasonic sensor, joystick and LCD. Tasks such as displayQueue() and measureDist() are suspended initially as they are not needed when in the beginning stage..

```
void setup() {
  Serial.begin(921600);
  while (!Serial) {} // Wait for the serial port to connect. Needed for native USB ports only.

  // Create a queue to hold int values. Queue length is 1, and each item size is the size of int.
  distanceQueue = xQueueCreate(1, sizeof(int));
  setTimeQueue = xQueueCreate(1, sizeof(int));
  CDQueue = xQueueCreate(1, sizeof(int));
  quoteQueue = xQueueCreate(1, sizeof(int));

  // Servo setup
  myservo.attach(9);  // attaches the servo on pin 9 to the servo object

  // LCD setup
  lcd.begin(16, 2);   // Set number of columns and rows
  lcd.print("Enter study time:");

  // Ultrasonic sensor setup
  pinMode(trigPin, OUTPUT); // Sets the trigPin as an Output
  pinMode(echoPin, INPUT); // Sets the echoPin as an Input

  // Joystick setup
  pinMode(SW_PIN, INPUT);
  digitalWrite(SW_PIN, HIGH);

  // Create tasks
  xTaskCreate(inputTime, "joystick", 128, NULL, 3, &it);
  xTaskCreate(waveHand, "servo", 128, NULL, 2, &wh);
  xTaskCreate(measureDist, "sensor", 128, NULL, 2, &md);
  xTaskCreate(displayLCD, "LCD", 128, NULL, 2, &disp);
  xTaskCreate(displayCountdown, "countdown", 128, NULL, 2, &CD);
  xTaskCreate(displayQuote, "quote", 128, NULL, 1, &qt);

  // Suspend tasks that will be used later on
  vTaskSuspend(qt);
  vTaskSuspend(md);

  vTaskStartScheduler();
}
```

Fig.8 Setup() function for Part 2

inputTime() uses a joystick to control the time for the user which is displayed in the LCD. After the user sets the desired time, the button can be pressed to lock the time.
displayQueue() and measureDist() are started and tasks displayLCD() as well as inputTime() are suspended.

```
void inputTime(void *pvParameters) {

  for (;;) {

    if (!digitalRead(SW_PIN) && time != 0) {     // When switch is pressed,
      lcd.clear();                                      // clear the LCD display
      xQueueSend(CDQueue, &time, portMAX_DELAY);        // and send a signal to start the countdown

      // Suspend tasks already used and resume the next tasks
      vTaskSuspend(disp);
      vTaskResume(qt);
      vTaskResume(md);
      vTaskSuspend(it); // suspend this task
    }

    // read analog Y analog values
    yValue = analogRead(VRY_PIN);

    // If joystick is at the rightmost position, increment time
    if (yValue > 700) {
      time += 1;
    } else if (yValue < 300) {   // else if joystick is at leftmost position, decrement time
      if (time >= 1) {
        if (time == 10 || time == 100) {
          lcd.setCursor(0, 1);
          lcd.print("    ");
        }
        time -= 1;      // avoid negative values
      }
    }
    xQueueSend(setTimeQueue, &time, portMAX_DELAY);    // Send updated time to be displayed on LCD

    vTaskDelay(200 / portTICK_PERIOD_MS);

  }
}
```

Fig.9 inputTime() function

The data from the joystick in inputTime() is received by displayLCD() to be displayed in the LCD, where displayLCD() the number of minutes varied by the data from inputTime().

```
void displayLCD(void *pvParameters) {
  int minute = 0;
  for (;;) {
    if (xQueueReceive(setTimeQueue, &minute, portMAX_DELAY) == pdPASS) {     // Wait until receive time input from joystick
      // set the cursor to column 0, line 1
      // (note: line 1 is the second row, since counting begins with 0):
      lcd.setCursor(0, 1);

      // print the time input
      lcd.print(minute);

      lcd.setCursor(4, 1);
      lcd.print("mins");
    }
  }
}
```

Fig.10 displayLCD() function

displayCountdown() starts the countdown according to the time from inputTime() after the joystick button is pressed. When it reaches 00:00, it will print “CONGRATZ” in the LCD and delete all tasks in the project.

```
void displayCountdown(void *pvParameters) {
  // Setup initial values
  int minute = 0;
  int second = -1;

  for (;;) {
    // Start the countdown
    if (minute > 0 || second > -1 || (xQueueReceive(CDQueue, &minute, portMAX_DELAY) == pdPASS)) {
      if (second == -1) {   // change of display from 1 minute to 59 seconds
        minute--;
        second = 59;
      }

      vTaskDelay(1000 / portTICK_PERIOD_MS);    // delay by 1 second

      if (minute == 0 && second == 0) {
        lcd.clear();
        lcd.setCursor(4, 0);
        lcd.print("CONGRATZ");
        vTaskDelete(wh);
        vTaskDelete(md);
        vTaskDelete(qt);
        vTaskDelete(CD);
      }

      second--;   // decrement the 'seconds' value
    }
  }
}
```

Fig.11 displayCountdown() function

measureDist() uses an ultrasonic sensor to get the distance from the sensor to any object in front of it. If the distance is less than some specified number, then the task will send a signal to

waveHand() to move the servo motor and displayQuote() display a motivational quote on the LCD until the measured distance of the object is higher than the specified number.

```
void measureDist(void *pvParameters) {

  for (;;) {

    digitalWrite(trigPin, LOW);   // Clears the trigPin
    delayMicroseconds(2);

    digitalWrite(trigPin, HIGH);    // Sets the trigPin on HIGH state for 10 micro seconds
    delayMicroseconds(10);
    digitalWrite(trigPin, LOW);

    duration = pulseIn(echoPin, HIGH);    // Reads the echoPin, returns the sound wave travel time in microseconds

    distance = duration * 0.034 / 2;      // Calculating the distance

    if (distance < 5) {   // When an object is close to the sensor
      xQueueSend(distanceQueue, &distance, portMAX_DELAY);    // Send signal to move servo motor
      xQueueSend(quoteQueue, &distance, portMAX_DELAY);    // Send signal to display quote on LCD
    }
    vTaskDelay(200 / portTICK_PERIOD_MS);
  }
}
```

Fig.12 measureDist() function

waveHand() waves the servo motor everytime it receives a signal from the ultrasonic sensor in task measureDist().

```
void waveHand(void *pvParameters) {

  for (;;) {
    if (xQueueReceive(distanceQueue, &distance, portMAX_DELAY) == pdPASS) { // Wait until receive signal to execute

      myservo.write(90);   // move to position at 120 degrees
      vTaskDelay(500 / portTICK_PERIOD_MS);
      myservo.write(60);     // move to position at 0 degrees (original position)
      vTaskDelay(500 / portTICK_PERIOD_MS);
    }
  }
}
```

Fig.13 waveHand() function

displayQuote() displays a motivational quote “don’t give up” for 1 second on the LCD everytime it receives signal from the ultrasonic sensor in task measureDist().

```
void displayQuote(void *pvParameters) {
  char* printingQuote;

  for (;;) {
    if (xQueueReceive(quoteQueue, &distance, portMAX_DELAY) == pdPASS) {   // Wait for signal from ultrasonic sensor
      printingQuote = quote;    // Set the quote to be displayed
      lcd.setCursor(1, 1);      // Set cursor to the bottom line
      lcd.print(printingQuote);   // Print quote to LCD display
      vTaskDelay(1000 / portTICK_PERIOD_MS);  // Delay for a second to show the quote
      lcd.setCursor(1, 1);
      lcd.print("                ");    // Clear the bottom line
    }
  }
}
```

Fig.14 displayQuote() function

Results

Below is a visualization of the “Study Buddy” project and its highlighted features. Fig. 15 shows how a user can move the joystick to increase/decrease the desired time before inputting it.

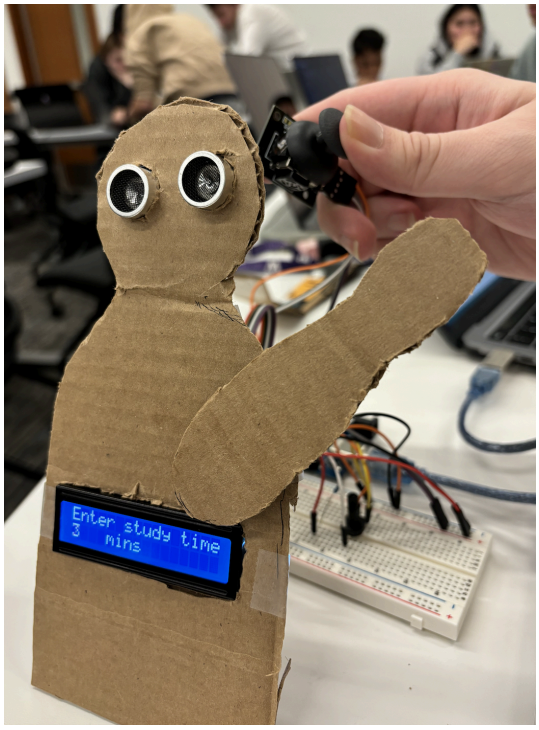

Fig. 15 A joystick module is used to input the desired time

A countdown timer will immediately start after the user inputted a specific amount of time in minutes. The countdown timer will show the number of minutes and seconds left on the LCD’s top row as shown in Figure 16.

Fig. 16 Countdown timer on Buddy

When Buddy's eyes are "covered," it will display a motivational quote on the display while waving its hand until the object covering its eyes is removed as shown in Figure 17.

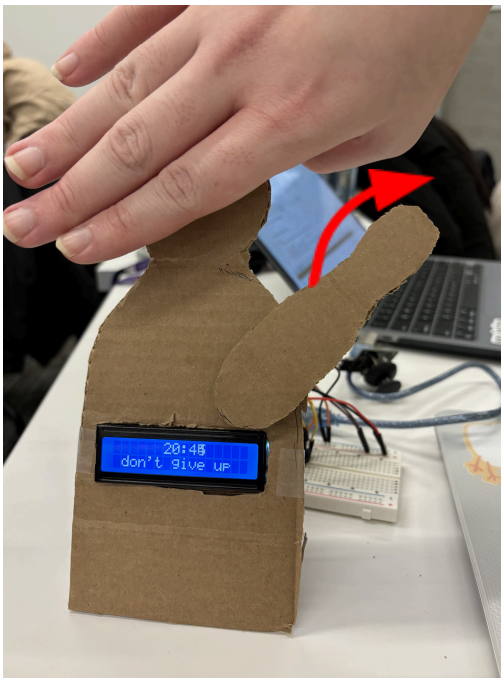

Fig. 17 Features when Buddy's eyes are "covered"

Once the countdown finished (e.g. 00:00 is reached), then the LCD will display “CONGRATZ” as shown in Figure 18. It is a signal that the user has finished, while reflecting a companion that is congratulating the user for finishing their study session.

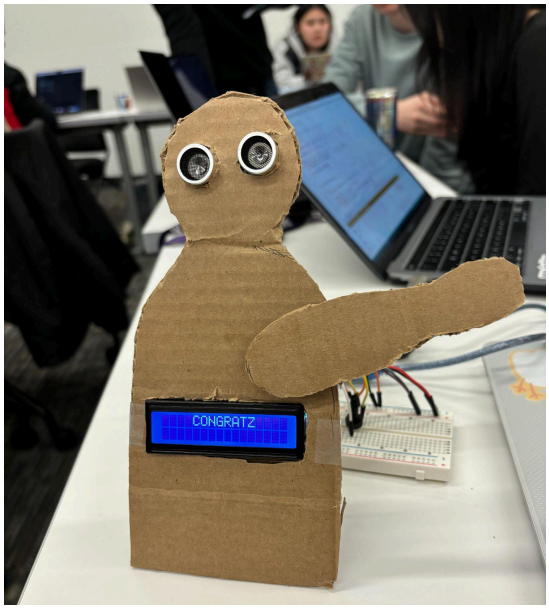

Fig. 18 Finishing state of Buddy

The hardware wiring setup is shown in Figure 19, where the all the hardware components (i..e ultrasonic sensor, joystick module, etc.) are wired to the Arduino board. The LCD uses a breadboard to connect each pin to the pins on the Arduino, while some pins are also being wired to a resistor and potentiometer to control the brightness of the LCD display.

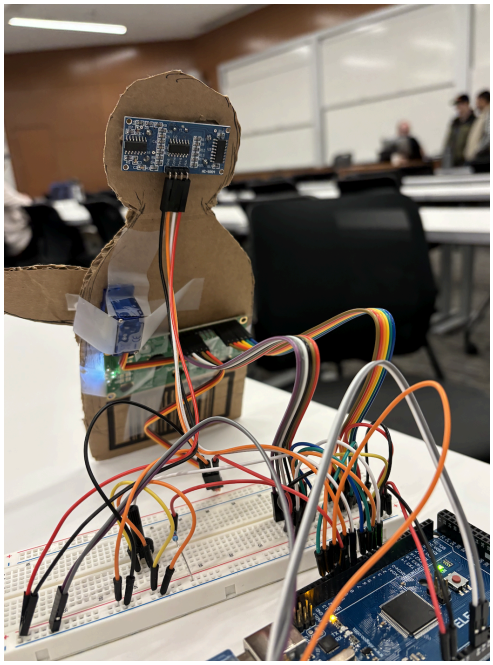

Fig. 19 Hardware wiring setup

Discussion

In the implementation of this project, there were a myriad of technical challenges that tested problem-solving skills and collaborative abilities. One of the challenges was integrating different tasks and or components to work together sending and receiving data in real-time. Another challenges was when displaying the time in LCD, specifically when transitioning from a higher place value to a lower place value (i.e. from 10 to 9 or 100 to 99), the issue persisted because the LCD placed the character in a lift alignment, resulting in the retention of the higher place value (i.e. 10 to 90 or 100 to 990). Work was divided accordingly to ensure swift and efficient progress, such that one member focuses on code structure and debugging, while another member focuses on code documentation and testing.

Before diving into the main part of this project, testing all the components needed as well as breaking down the idea into tasks are crucial. Regular work sessions provided a platform for discussing progress, addressing challenges, and coordinating efforts, ensuring that both members remained informed and engaged throughout the project. Peer code reviews played a crucial role in maintaining code quality and sharing knowledge among team members. For example, ensuring that there are function prototypes in the beginning of each sketch code explained what functions were used, and adding short comments on each key code line worked as a small note for anyone (i.e. team members or TA) to review what they are doing.

Suggestions

For this lab project, there were a lot of positive aspects that encouraged students' learning process in applying embedded systems to solve real-world problems. For example, the flexibility in project selection provided a space for students to design a personalized project while incorporating the skills learned inside or outside the classroom, where students are also encouraged to explore additional features such as using novel sensors and displays. However, it would have been interesting to explore more about IoT and focusing on connecting embedded devices to the internet and utilizing cloud services for data storage, processing, and analysis. Such platforms could include Raspberry Pi, ESP32, and integrating sensors, actuators and wireless communication modules.

Conclusions

In conclusion, the development of the "Study Buddy" project provided valuable insights into the complexities of real-time embedded system design and implementation. By utilizing a preemptive operating system (RTOS) and integrating various hardware components, this project aimed to address challenges encountered during study sessions through innovative solutions.

Overall, the "Study Buddy" project not only provided hands-on experience in embedded system development but also fostered teamwork, collaboration, and technical skills development.

Through this project, the team gained valuable insights into the importance of effective project management, communication, and problem-solving.

References

Arduino. "Timer Interrupts on Arduino." Arduino Official Documentation, 2022. https://www.arduino.cc.

ChatGPT. 2024. https://chat.openai.com.

Liquid Crystal Display (LCD) with Arduino https://docs.arduino.cc/learn/electronics/lcd-displays/

Servo Motor Basics with Arduino https://docs.arduino.cc/learn/electronics/servo-motors/

Ultrasonic Sensor HC-SR04 and Arduino - Complete Guide https://howtomechatronics.com/tutorials/arduino/ultrasonic-sensor-hc-sr04/

Appendices

Doxygen

https://github.com/sreksosamudra/Intro-to-Embedded-Systems-Lab/tree/main/lab4/doxygen_html

GitHub Release

https://github.com/sreksosamudra/Intro-to-Embedded-Systems-Lab/releases/tag/lab_4

Total Number of hours Spent

14 hours