GymHive Tracker

ECE 445 Design Document - Spring 2025

Team 28

Aryan Shah (aryans5)

Kushal Chava (kchav5)

TA: Aishee Mondal

Professor: Arne Fliflet

Date: 03-06-2025

Table of Contents

1. Introduction

1.1 Problem

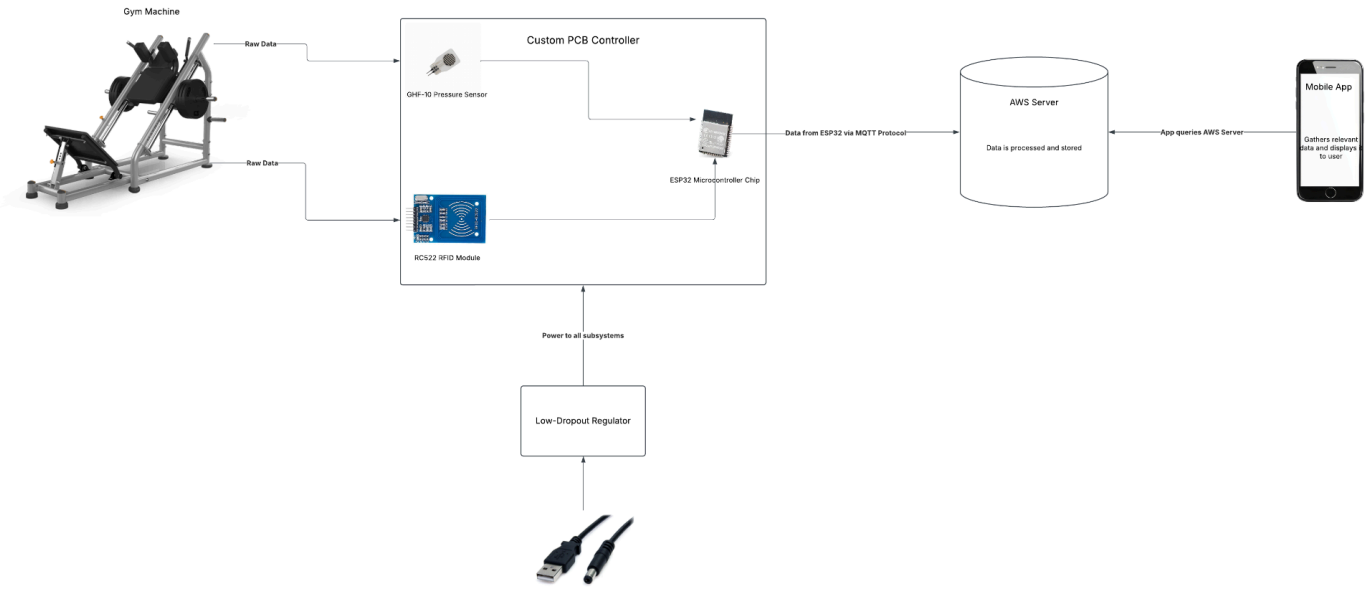
During peak gym times, equipment tends to get occupied quickly, which leads to long wait times and disrupted workout routines. Many gym-goers rely on consistent machines to track their progress weekly, but delays caused by occupied machines tend to force them to wait or alter their regimens. This inefficiency wastes time and reduces overall workout effectiveness for the athlete.

1.2 Solution

Our solution to this obstacle for gym-goers is the GymHive Tracker, a sensor-based system that monitors gym equipment utilization and provides individuals with real-time availability updates. We place our tracker at key contact points - such as pads, seats, or standing areas - and our system detects when a machine is occupied. To further optimize user satisfaction, we plan to implement a queue system where users can “check in” via their i-Card (simulating key fobs that commercial gym establishments typically give) on an RFID-enabled tag attached to each machine. Once checked in, users input their planned sets and reps, enabling the system to estimate wait times for those in line.

Rep and set tracking will be based on manual user input. Gym-goers can input their desired sets and reps and modify them during their turns. To improve accuracy, our system will use data analysis from user inputs to learn and adjust estimated set durations over time (which tend to vary depending on the type of gym equipment). The system will notify the next gym-goer in the queue when the current user finishes their last set, minimizing wasted time. Each gym machine will have a dedicated PCB containing an ESP32 microcontroller chip that transmits data remotely to a central AWS server for processing. Users access this data through a mobile app by scanning their I-Card onto the machine's RFID reader, allowing for a seamless process. By integrating features such as real-time tracking, adaptive set duration estimates, and estimated wait times, the GymHive Tracker will enhance workout efficiency and promote optimized gym operations.

1.3 Visual Aid



The diagram shown above provides a visual guide to our design. Starting from the left, we have an example of a gym machine commonly found in commercial gyms. This piece of gym equipment contains pads, bars, or other areas a user typically applies pressure onto while utilizing that machine. Our custom PCB would be placed onto these common contact points, and powered via USB connection into a nearby outlet. The PCB will be able to detect machine occupancy and user identification via protocols, which will be explained in detail later. This data would be continuously sent across packets to an AWS server, which will be queried by an app we design.

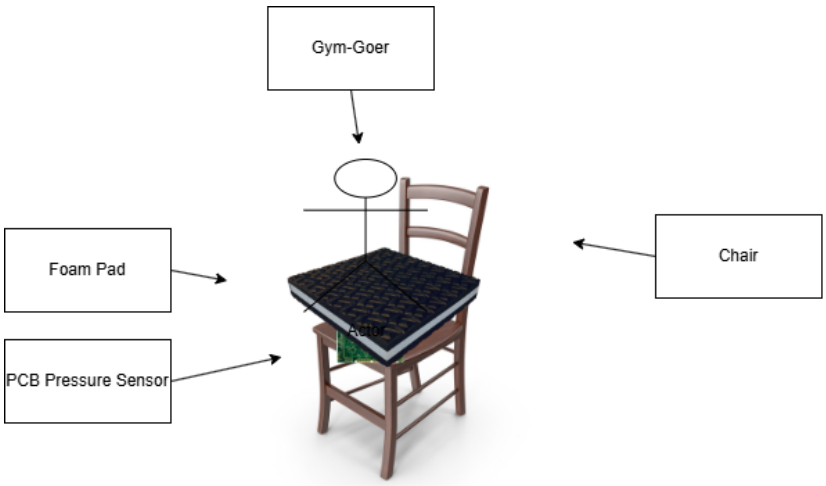
With this design, a gym member can approach any piece of gym equipment and hover their i-Card over it. The PCB will then detect the user via data from the i-Card sent wirelessly over RFID. The user can then see the equipment's status as occupied on the app. If the gym machine has already been occupied, however, the user can also join a queue system to wait for their turn on the machine. This queue system relies on manual user input of desired sets and reps, and will output an estimated wait time for that machine. Since this is only estimated, the user will also receive a notification when the occupied user is on their last set so they know their turn onto the machine is coming up.

1.4 High-Level Requirements

1. **Real-Time Equipment Monitoring:** The system must accurately detect gym equipment occupancy with at least 95% accuracy (+/- 5 lbs threshold for acceptance), filtering out random weight fluctuations
2. **Efficient Data Transmission and Display:** The user must be able to transmit PII (personally identifiable information) via RFID within 1 second of scanning an i-Card. The microcontroller must process occupancy data and user check-ins within 3 seconds and transmit updated availability to the central AWS server within 1 second of an occupancy change.
3. **Queue Management and User Notifications:** The system must estimate wait times with a maximum error margin of 20 +/- 3% error margin (set and rep times tend to vary drastically across users for some equipment) by analyzing user-inputted reps and sets.

2. Design

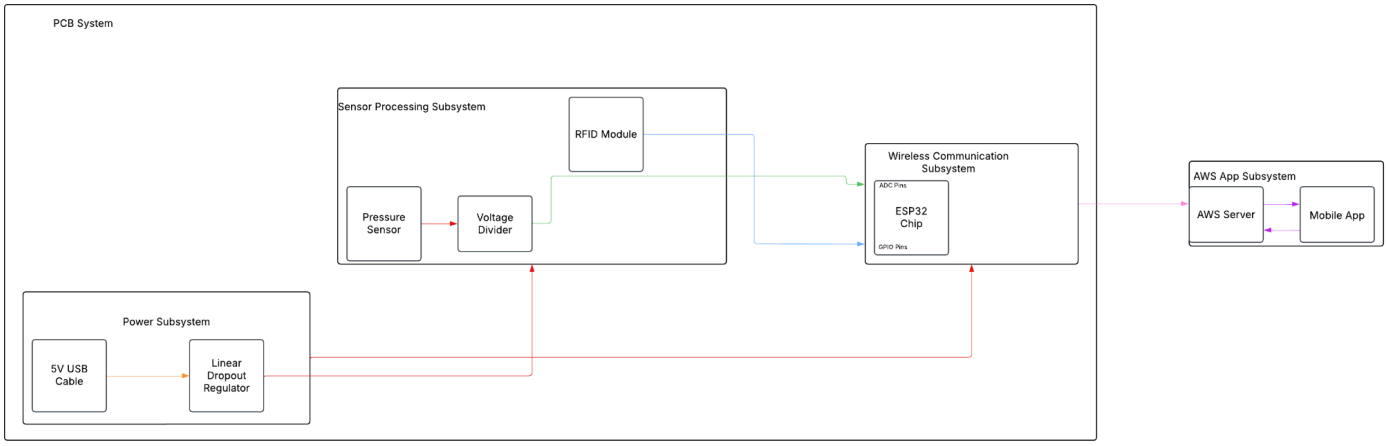
2.1 Physical Design



To test our PCB, it would be unfeasible to be able to acquire and alter the state of an actual piece of gym equipment. To simulate this scenario, we use a chair to act as our machine. Our PCB pressure sensor will be placed on top of the chair and a foam pad will be placed on top of it to act as the point of contact we would be using from our machine. This would also provide some weight fluctuations from the foam pad. This is important as it provides trivial weight onto the pressure sensor, which is mentioned in our requirements and tolerance analysis as a key factor in our final design. The

pressure sensor will have to determine whether the weight is coming from an actual individual as opposed to just the foam pad. The user will then apply pressure onto the foam pad to simulate their occupancy of the machine. The user can also test RFID functionality by hovering their i-Card over the foam pad on the chair. The user will use their normal smartphone to test our app's functionality.

2.2 Block Diagram



2.3 Functional Overview & Block Diagram Requirements

2.3.1 Pressure Sensing Subsystem

The Pressure Sensing Subsystem is the portion of our project responsible for detecting whether the gym equipment is occupied. The subsystem primarily relies on the GHF-10 component, a force pressure sensor that measures weight from 0 - 110 lbs and can interact with the ADC chip of our ESP32. Below are the operating characteristics found in the datasheet of our GHF-10 [1]:

Operating Characteristics

Table 1. Operating Characteristics (T_A = 25°C unless otherwise noted, F is positive)

Characteristic	Symbol	Min	Typ	Max	Unit
Force Range[1]	F_{OP}	0	—	500	N
Linearity[2]	—	—	0.99	—	—
Repeatability[3]	—	—	—	±2	%
Hysteresis[4]	—	—	—	±2	%
Drift[5]	—	—	—	6	%/log(s)
Response Time[6]	t_R	—	—	0.1	ms

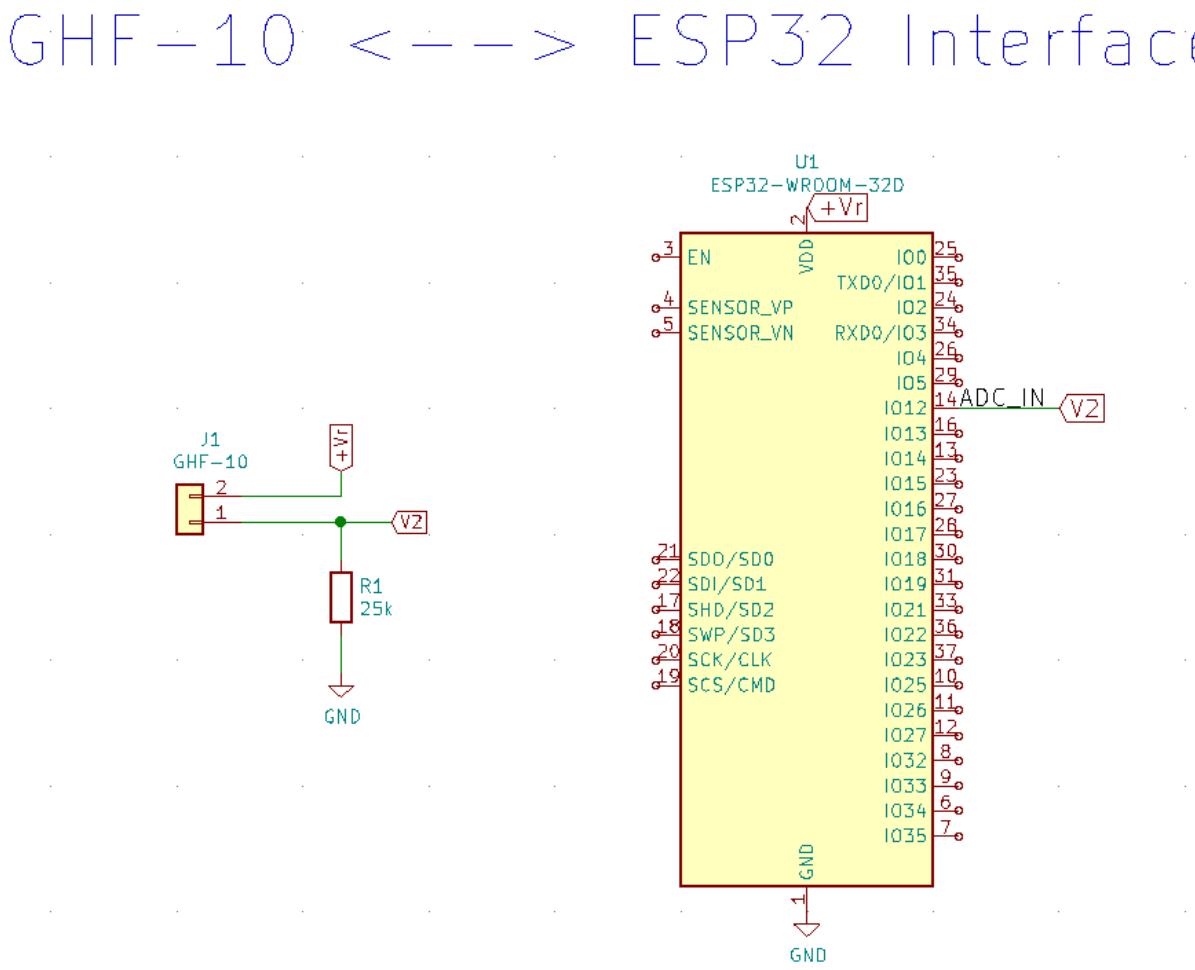
The GHF-10 relies on a voltage divider configuration, whose mathematical analysis follows in our tolerance analysis later in this report. To summarize, however, we require a measurable output of 50 lbs, which we specified as our minimum threshold for human occupancy. Because the output voltage of the GHF-10 will not be linear, we will set the output voltage at ~2.5V at 50lbs to ensure the reading is meaningful by the ADC pins. Due to the 12-bit resolution (which corresponds to raw outputs from 0 - 4095) of the ESP32 ADC pins, this 2.5V corresponds to approximately a raw output of 3102. The software requirements of this ADC protocol and the necessary code to continuously output user occupancy from the serial ports are explained below.

To summarize, our protocol will follow the following format: GHF-10 → Voltage Divider → ADC Pin → output from Arduino IDE → MQTT protocol. The software section will explain the software portion controlling the ESP32 and MQTT protocol. To ensure accuracy, we have created a requirements & verification table:

Requirement	**Verification**
The GHF-10 system must detect a short circuit within 10ms	• Power to the affected circuit is cut-off and reduced to 0-0.3V within 50ms • Time of cut-off is measured and entered into stored log files based on ADC output values • System must resume normal operation within 5 seconds • During testing, no permanent damage to relevant subsystem

	components after 10 trials
The GHF-10 force pressure sensor must detect occupancy weight of 50lbs or more	● Must detect applied weight within +/- 5lbs of accuracy ● Must maintain 95% accuracy in distinguishing actual occupancy from fluctuations in sensor readings ● Output voltage to the ADC pin must correlate to approximately 2.3 - 2.6 V to ensure reliable transmission
The GHF-10 system interfaces properly with the ADC pins of the ESP32	● Known weights of 10, 50, 100 lbs (more can be added if time allows) will be added onto the pressure sensor ● Each weight range must properly detect the correct occupancy weight ● A reasonable output voltage within +/- 0.3V must be applied and read from the ADC pins of the ESP32 ● Output voltage must not exceed 3.3V or output below 0.0V
The GHF-10 system will interface properly within a simulated gym environment	● Temperatures between 65, 70, and 75 degrees Fahrenheit will be simulated to ensure reliable operation across different gym environments ● The foam pad will likely act as an added insulator, which may increase the temperature ● Since the GHF-10 system is properly insulated, on the other hand, any rain/snowy conditions will not need to be tested
The GHF-10 system reliably updates to the ESP32 and AWS Server accordingly	● The microcontroller must process occupancy data (output from Arduino IDE code) within 3 seconds ● Updated availability must transmit to the central AWS server within 1

	second of an occupancy change, leading to a total response time from the GHF-10 → ESP32 → application of 4 seconds +/- 2 seconds
Software for interfacing between the GHF-10 and ESP32 is accurate	• The system must correctly determine the user occupancy based on the 50lbs threshold and signify a successful occupancy status accordingly • Forces below 50lbs will ensure that occupancy is not detected • The voltage output must be correctly read according to 12-bit resolution with an error margin of +/- 0.3V



Above is an example interface on the connection between our GHF-10 and ESP32. As we see, the GHF-10 peripheral is powered by the same 3.3V power supply of the ESP32. The output of the GHF-10, V2, is connected to IO12, a viable ADC channel input on the ESP32-WROOM-32D. Finally, we have R1 as 25kOhms in our voltage divider setup (as measured in our tolerance analysis). We are using a connector on our board to hook up the non-solderable GHF-10 component found on Digikey to, achieving the same result as a solderable component.

2.3.2 Microcontroller Subsystem

To interface with our components (namely, process pressure sensor data, handle communication via the RFID chip, and transmit data to the AWS server), we have chosen the ESP32-WROOM-32D IC. As mentioned, the IC will interface with all of our peripherals. It will be able to read in ADC data to process occupancy status from the GHF-10. It will be able to read in RFID tags via the SPI protocol, thereby allowing the

app to identify the user. Finally, it will handle all data transmission to our AWS IoT Core setup via the MQTT protocol. From the datasheet of the IC, we have the following peripheral schematic:

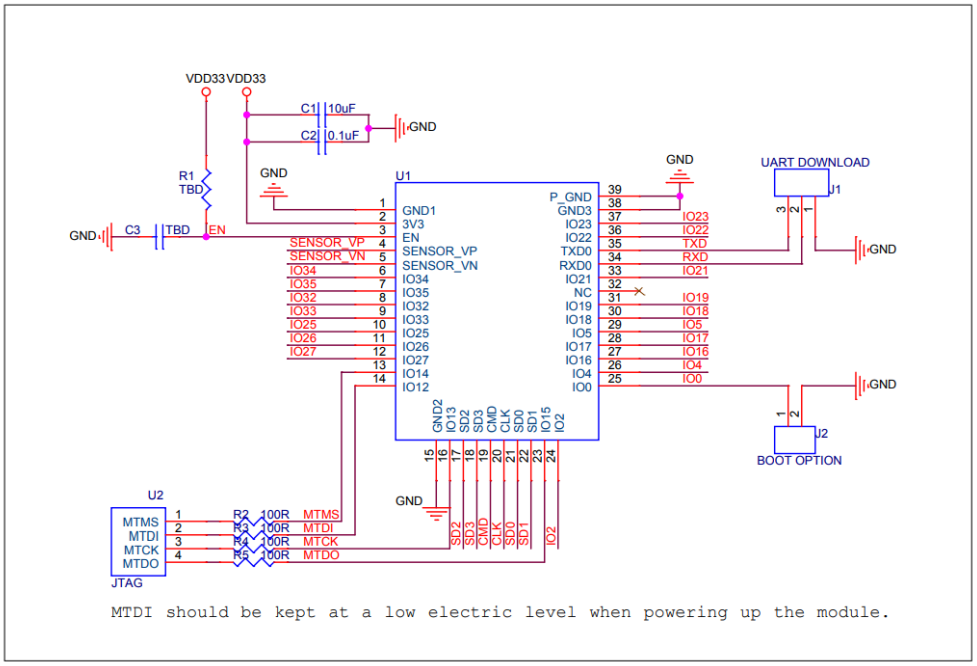


Figure 5: ESP32-WROOM-32D & ESP32-WROOM-32U Peripheral Schematics

As shown, the pins support a variety of protocols, including Wi-Fi, Bluetooth, SPI, and I2C, making it ideal to handle connection with our GHF-10, MFRC522, and other power peripherals. A sample schematic of our current ESP32 setup with the necessary additions for power (bare minimum circuitry for operation) will be later shown in the power subsystem. For now, we have detailed requirements & verifications in the following table:

Requirements	**Verification**
• The ESP32 must require proper voltage regulation	• Must receive a stable 3.3V power supply, stepped down from a 5V USB source by the AMS1117-3.3V • Voltage must remain within the ESP32's safe operation range, which is from 3.0 - 3.6V. Since none of our peripherals will exceed 3.3V, we will put this value at 3.4V to be safe • Output of voltage regulator must be confirmed with a multimeter to be 3.3 +/- 0.1V to be safe

	● Voltage fluctuations under load will be measured using an oscilloscope ● High computation loads will be placed to ensure voltage remains stable ● In the event of a short circuit, power must be cut off immediately to 0.0V to prevent damage to other components
● The ESP32 must be able to communicate securely with the AWS server	● The ESP32 must connect and maintain a stable connection to a designated Wi-Fi network, and print statements using its built-in WiFi.status() function will be tested in a loop of 500ms intervals ● RSSI signal strength and packet loss data will be monitored ● In the event of a network disruption (which can be simulated by manually shutting on/off a personal hotspot), will be tested 5 times, and each time, automatic reconnection must be met ● Using AWS built-in services, compare sent and received hash values for data integrity ● Measure round-trip latency using timestamps to ensure 1 second communication
● The ESP32 must interact with the GHF-10 using ADC	● Measure voltage output across varying levels of 0 to 110 lbs, and ensure that output is within 0 - 3.3V ● To validate, there will be no tolerance accepted above 3.3V for higher lbs of force. As long as we have ~2.5 V of output at 50 lbs, any tolerance below 3.3V for 110 lbs of max force is fine (as we just detect occupancy, not exact weight) ● Voltage readings using a multimeter should match the ESP32's analogRead() within +/-

	0.3V
● The ESP32 must correctly exchange data with the MFRC522 via SPI protocol	● The ESP32 should accurately read RFID tags within 25 +/- 10 mm (rated for a max of 50mm). ● Testing RFID tags at different distances from 5mm, 10mm, 25mm, 50mm, and a success rate of ~25 scans at each interval ● Attempt a failed read with an unsupported key fob type (for example, my apartment key fob) at 5 times, and must re-attempt scanning each time

2.3.3 RFID Subsystem

The RFID subsystem is responsible for allowing a user to hover over the gym equipment and being able to “check in” to the machine. The MFRC522 IC is a popular chip used alongside the ESP32 interfaced with the SPI protocol to provide RFID capability. Many commercial gyms give their members key fobs to check in, making it ideal if this concept were brought to market. Since we do not have access to one of these key fobs, however, we will use an i-Card that has RFID capability. Below is both the standard application scheme [3] and the schematic we have designed for the MFRC522 to interface with the ESP32:

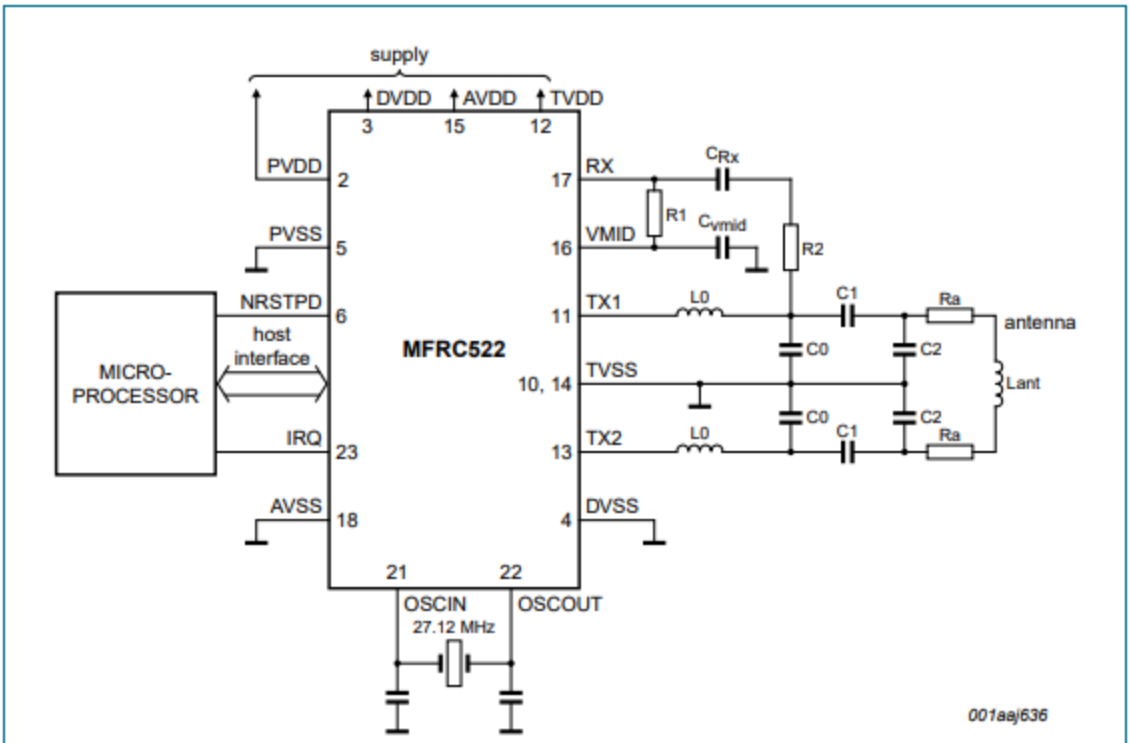
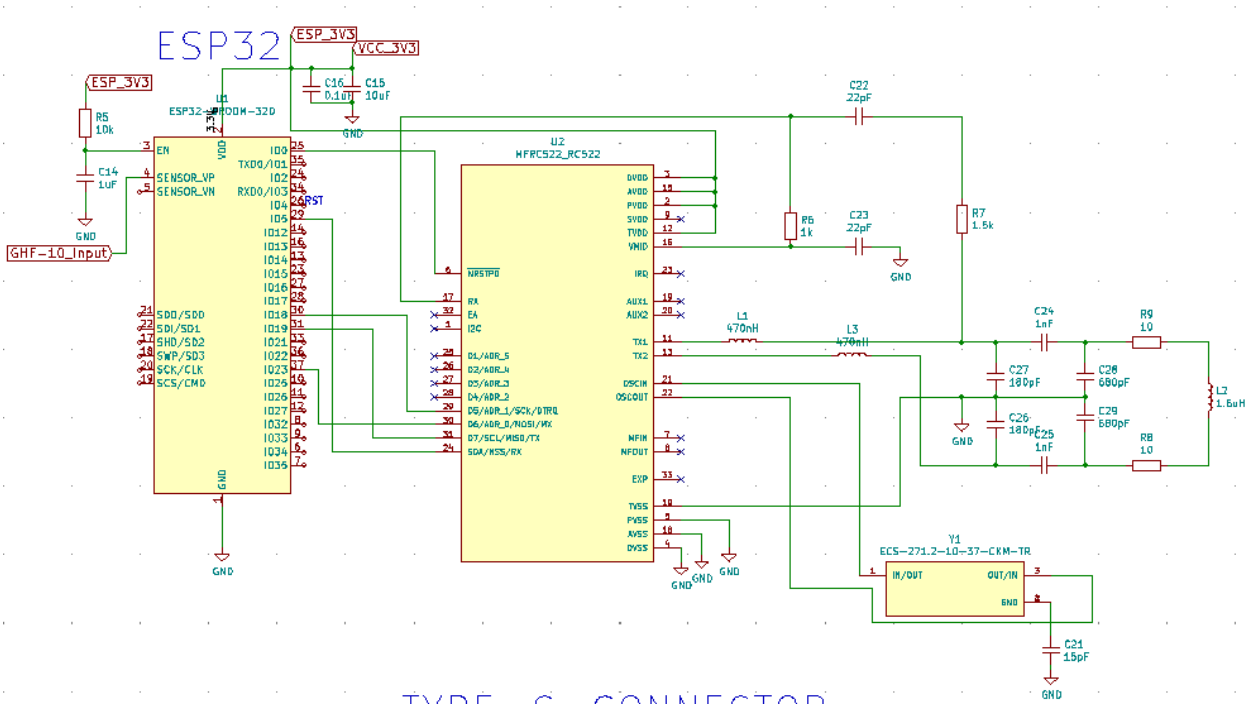


Fig 27. Typical application diagram



As shown, we have hooked up the MFRC522 using the SPI interface. The NRSTPD chip allows for reset requests to the chip, and the IRQ allows an interrupt request

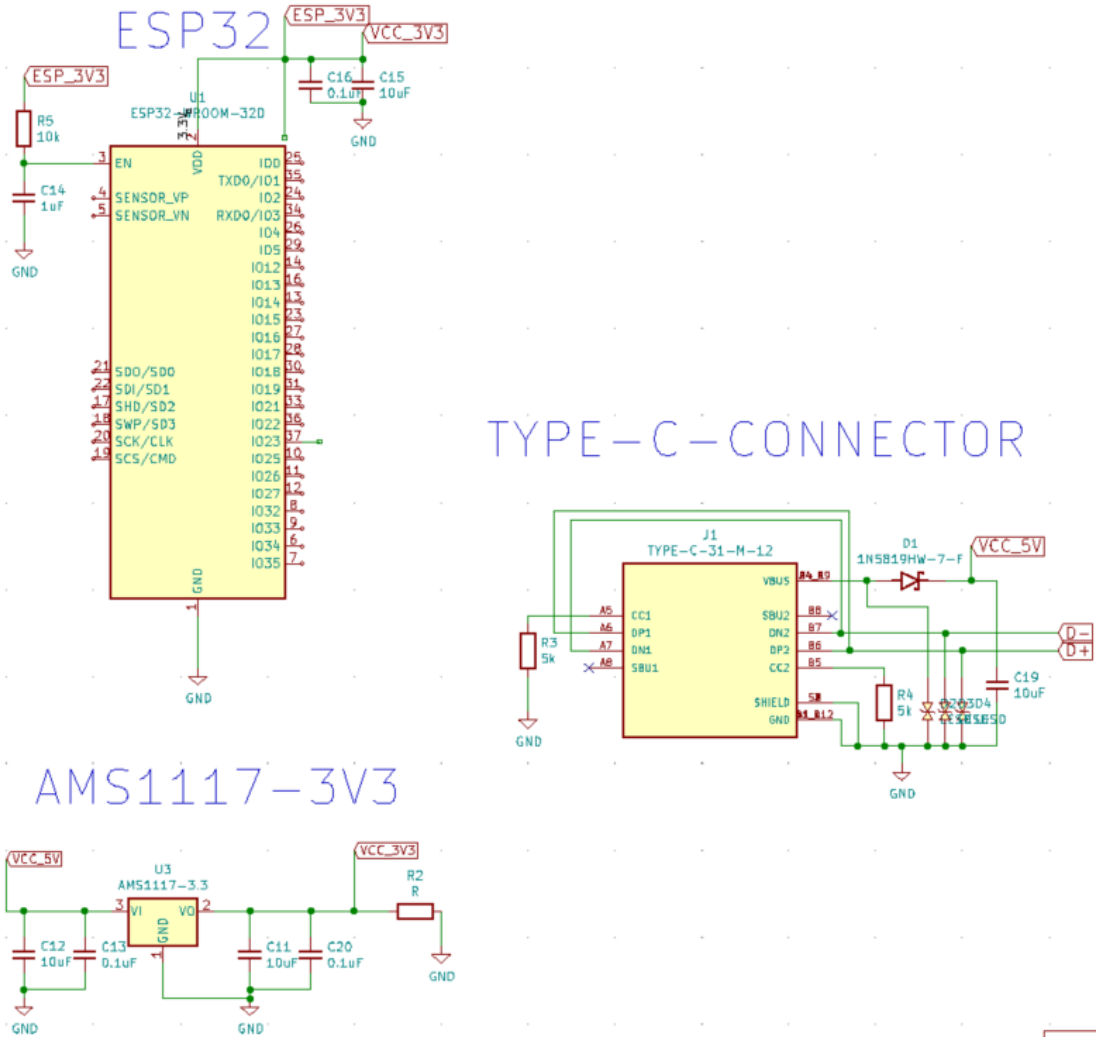
(signalling the microcontroller each time an event occurs). The datasheet also includes an antenna circuit (see pins TX1 and TX2), which is responsible for transmitting and receiving the RF signals that communicate with the RFID tags. Finally, a crystal oscillator with a 27.12MHz frequency using the ECS-271.2-10-37-CKM-TR is responsible for providing the clock signal for the MFRC522. This 27.12MHz frequency is a reference that generates the 13.56MHz carrier frequency used in the RFID communication. The clock signal precisely times all of the internal operations and processing of the MFRC522 with the ESP32.

Requirements	**Verification**
● Ensure that the MFRC522 is correctly supplied with power	● Ensure power supply is off at the beginning ● Ensure that DVDD, AVDD, TVDD, PVDD pins are physically connected to the positive voltage rail of the power supply ● Verify that DVSS, AVSS, PVSS, TVSS are physically connected to a ground rail ● Using a multimeter, check for short circuits between the power rails and ground (there should be 0 shorts) ● Turn on power supply and set it to the correct voltage for the MFRC522. Measure voltage at DVDD, AVDD, TVDD, PVDD. Ensure voltage is within 2.5V - 3.3V range for these positive rails. For the ground pins, they should measure 0V.
● The crystal oscillator must be correctly connected and oscillating	● Ensure power supply off ● Verify that the 27.12MHz crystal is physically connected to OSCIN and OSCOUT. Ensure capacitors are grounded ● Use an oscilloscope to probe the OSCIN and OSCOUT pins ● Verify that a stable oscillating signal at 27.12 +/- 0.5 MHz is present. This can be measured

	using the frequency of the signal.
● The antenna circuit must be properly connected in the PCB	● Ensure power supply off ● Verify all relevant components are physically connected to the pins in the correct configuration ● If available at the ECE 445, use a network analyzer to measure impedance and resonant frequency of the antenna circuit ● If not available, connect the MFRC522 to the microcontroller, and attempt to read an RFID tag. If the tag can be read, the antenna circuit is functioning as intended. Repeat this 5 times in a row with a 100% success rate.

2.3.4 Power Subsystem

Our power subsystem is primarily responsible for supplying power to all peripherals on the board. We utilize the TYPE-C-31-M-12 connector for integrating a standard USB-C power cord that provides continuous 5V power to the board. 3 primary peripherals need to be powered: the ESP32 (which runs from 0-3.3V), the MFRC522 (also runs 0-3.3V), and the GHF-10. The GHF-10's datasheet specifies that it does not run at a specified voltage, rather an output V2 that increases with added force. As mentioned in our tolerance analysis, we design a voltage divider configuration such that the GHF-10 also runs between 0-3.3V. As a result of this, we had to integrate the AMS1117-3V3 voltage regulator IC to step down the power to the ESP32 to 5V. The rest of the peripherals will then run off the ESP32's VDD pin which outputs 3.3V.



The image shown above is the schematic we have designed for our power subsystem, along with the necessary connections made to the ESP32. As per our professor's recommendation, we switched our choice from a 9V battery to a 5V USB-C connector because gym environments typically have plenty of outlets near machines and we had to deal with the power constraints of our PCB components.

The AMS1117-3.3 is a linear voltage regulator, working to dissipate extra voltage as heat. Paying close attention, we have the Vin as 5V, Vout as 3.3V, and Iout is the current drawn by the load connected to our 3.3V output. The core principle behind linear regulators lies behind Ohm's Law and power dissipation. Our regulator creates a voltage drop of 1.7V (5V - 3.3V). The power dissipated by the regulator must then be 1.7V * Iout. Since this power is dissipated as heat, linear regulators tend to physically warm up, however our difference between Vout and Vin is stable enough that the

AMS1117-3.3 can adjust resistance to maintain the stable 3.3V output without overheating.

Requirements	Verification
● The AMS1117-3.3 regulator shall provide a stable 3.3V +/- 0.1V output to the ESP32 at an input of 5V +/- 0.25V from the USB-C	● Apply a voltage of 5V +/- 0.25V via the USB-C connector ● Measure the voltage at the VCC_3V3 pin of the AMS1117-3.3 using a multimeter ● Using a multimeter and variable electronic load, increase load to 500mA and verify output voltage is stable within 3.2V and 3.4V
● The output voltage at the ESP32's VDD pin must be 3.3V +/- 0.1V	● Using a multimeter, ensure that the output voltage at the VDD pin of the ESP32 (ESP32_3V3) must be within the specified range ● Ensure that measured voltage is within 3.2V to 3.4V
● The power subsystem must be able to withstand continuous input of 5V without damage	● Connect USB-C power source to supply continuous 5V ● Under max load at 500mA, monitor all components to ensure no overheating (namely, the linear regulator) ● Ensure no components exceed 80 degrees Celsius under operation

2.4 Hardware Design

2.4.1 Operating Voltage & Regulation

Our goal is to utilize the microcontroller to provide stable 0 - 3.3 V output to the MFRC522 RFID chip and the GHF-10 pressure sensor. Found in the datasheet [3], we

have the following operating conditions for the ESP32:

5.2 Recommended Operating Conditions

Table 7: Recommended Operating Conditions

Symbol	Parameter	Min	Typical	Max	Unit
VDD33	Power supply voltage	3.0	3.3	3.6	V
I_{VDD}	Current delivered by external power supply	0.5	-	-	A
T	Operating ambient temperature	–40	-	85	°C

These values are in line with the supply voltage conditions listed in the MFRC522's datasheet [3]:

Table 1. Quick reference data

Symbol	Parameter	Conditions		Min	Typ	Max	Unit
V_{DDA}	analog supply voltage	$V_{DD(PVDD)} \leq V_{DDA} = V_{DDD} = V_{DD(TVDD)}$; $V_{SSA} = V_{SSD} = V_{SS(PVSS)} = V_{SS(TVSS)} = 0$ V	[1][2]	2.5	3.3	3.6	V
V_{DDD}	digital supply voltage			2.5	3.3	3.6	V
$V_{DD(TVDD)}$	TVDD supply voltage			2.5	3.3	3.6	V
$V_{DD(PVDD)}$	PVDD supply voltage		[3]	1.6	1.8	3.6	V
$V_{DD(SVDD)}$	SVDD supply voltage	$V_{SSA} = V_{SSD} = V_{SS(PVSS)} = V_{SS(TVSS)} = 0$ V		1.6	-	3.6	V

Finally, as mentioned, the GHF-10 pressure sensor does not work on a specified voltage, rather a voltage divider configuration that we will set up to work within 0 - 3.3V. With this in mind, it is very ideal for us to power both the pressure sensor as well as the MFRC522 RFID chip at these output values. The MFRC522 typically will run at around 3.3V and the GHF-10's output voltage should increase as force is applied.

2.5 Software Design

The following two subsections regarding the pressure sensor's ADC communication and the RFID's SPI communication rely on interface protocols that are accessible through the ESP32. This requires a need to set up and program the microcontroller. Luckily, the UIUC ECE 445 Course Page [13] has a section that details how to program the ESP32 using a USB-to-UART Bridge. This follows 3.3V logic, which is fine since our project design supports 3.3V power rails. Additionally, the programming will be done using an Arduino IDE in C/C++ code.

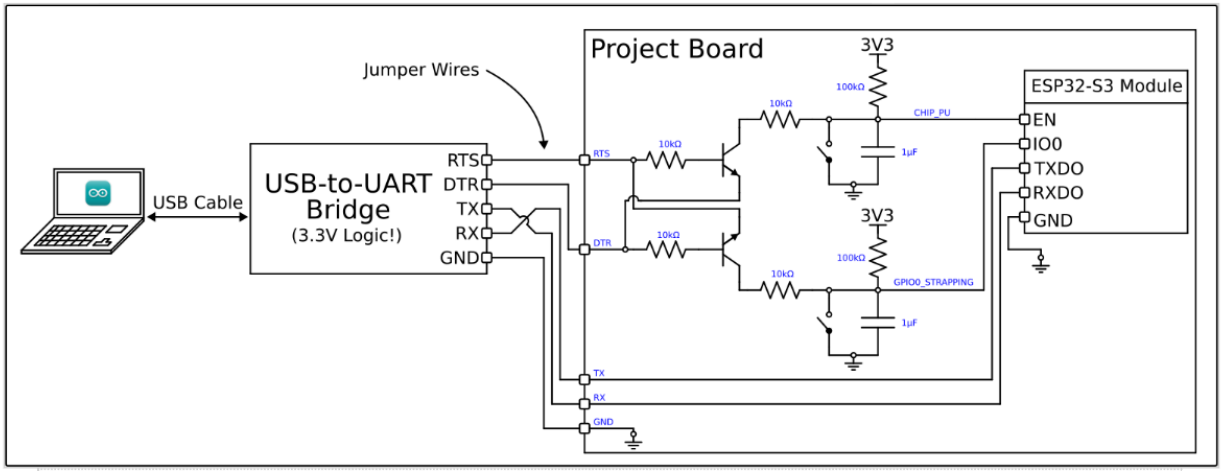


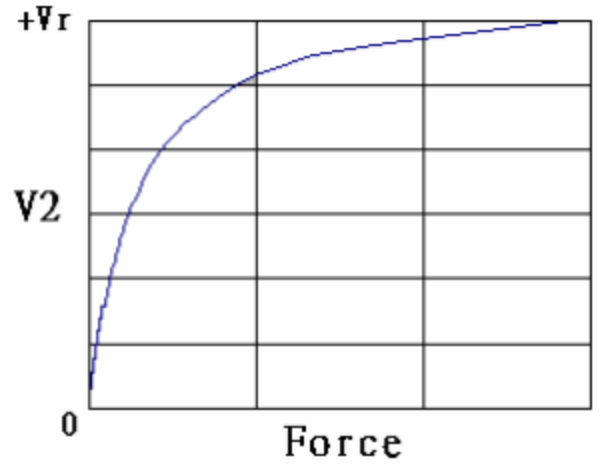
Diagram of programming setup and circuit

Note that the UART is a symmetric interface. TX on the bridge should go to RX on the ESP32-S3. RX on the bridge should go to TX on the ESP32-S3.

The preferred method of uploading to the ESP32-S3-WROOM uses the programming circuit to automate the use of GPIO0 to place the device into "Download Boot Mode and is a de-facto standard among ESP32 development boards. The circuit consists of four resistors and two transistors that form an open-collector output logic gate of sorts. The inputs are driven by UART control flow lines. When the Arduino IDE uploads, it will operate these control flow lines to automate the change to "Download Boot Mode" and final reset. Trying to program without this circuit on your board will get tedious.

2.5.1 Pressure Sensing ADC Communication

A key component of our PCB Controller is the ability to detect occupancy on a gym machine, which is handled by our GHF-10 pressure sensor. The GHF-10 pressure sensor relies on a voltage divider configuration to produce an output V2 which directly increases as added pressure is placed onto the component. The following diagram depicts the non-linear relationship between added force and the output V2 based on resistor values (calculated in our tolerance analysis):



From our tolerance analysis described below, we estimate a ~2.5V output at 50 lbs, our minimum threshold for detecting human occupancy. As mentioned in the ECE 445 Wiki, the ESP32 utilizes an Arduino IDE, likely with C/C++ development. The ESP32-Wroom-32D supports a 12-bit ADC (values from 0-4095). If we output 2.5V, we convert this using the following formula: (2.5V / 3.3V) * 4095 to get ~3102 as our ADC value output. This is the raw value we want from the ADC pin to detect successful occupancy. The following is sample code written, that will be modified as necessary during testing:

```
#define SENSOR_PIN 36 // ADC1 CH0 pin of the ESP32-Wroom-32D
#define VOLTAGE_MIN 2.5 // minimum voltage threshold for occupancy
#define VOLTAGE_MAX 3.3 // max voltage threshold of ESP32 pin
#define ADC_RESOLUTION 4095 // 12-bit resolution of ESP32 ADC pins

void setup()
{
    Serial.begin(250000); // initiate serial monitoring at 250000 baud
    analogReadResolution(12); // 12-bit resolution
}

void loop()
{
    int adc_output = analogRead(SENSOR_PIN); // read/store the adc value from the adc pin
    double voltage = (adc_output / ADC_RESOLUTION) * VOLTAGE_MAX; // use equation to convert the output to actual voltage

    if (voltage >= VOLTAGE_MIN) // if adc output > 2.5V
    {
        Serial.println("User occupied") // print success
    }
    else
    {
        Serial.println("User not found") // print no success
    }

    delay(500); // set a delay on the loop to run every 500ms to continuously monitor
}
```

2.5.2 RFID SPI Communication

One of the communication protocols supported by both the MFRC522 and the ESP32 includes the SPI protocol. Our reason for choosing SPI over I2C is that it has much

faster data transfer rates, is a full-duplex protocol, and has a dedicated chip select line. The following image from the MFRC522 datasheet [3] details the interface we need to set up by the SPI standard:

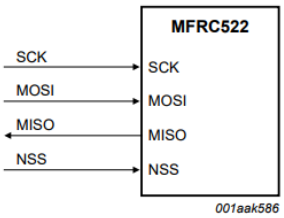


Fig 7. SPI connection to host

8.1.2.1 SPI read data

Reading data using SPI requires the byte order shown in Table 6 to be used. It is possible to read out up to n-data bytes.

The first byte sent defines both the mode and the address.

Table 6. MOSI and MISO byte order

Line	Byte 0	Byte 1	Byte 2	To	Byte n	Byte n + 1
MOSI	address 0	address 1	address 2	...	address n	00
MISO	X[1]	data 0	data 1	...	data n − 1	data n

[1] X = Do not care.

Remark: The MSB must be sent first.

8.1.2.2 SPI write data

To write data to the MFRC522 using SPI requires the byte order shown in Table 7. It is possible to write up to n data bytes by only sending one address byte.

The first send byte defines both the mode and the address byte.

Table 7. MOSI and MISO byte order

Line	Byte 0	Byte 1	Byte 2	To	Byte n	Byte n + 1
MOSI	address 0	data 0	data 1	...	data n − 1	data n
MISO	X[1]	X[1]	X[1]	...	X[1]	X[1]

[1] X = Do not care.

Remark: The MSB must be sent first.

8.1.2.3 SPI address byte

The address byte must meet the following format.

The MSB of the first byte defines the mode used. To read data from the MFRC522 the MSB is set to logic 1. To write data to the MFRC522 the MSB must be set to logic 0. Bits 6 to 1 define the address and the LSB is set to logic 0.

Table 8. Address byte 0 register; address MOSI

7 (MSB)	6	5	4	3	2	1	0 (LSB)
1 = read 0 = write	address						0

By analyzing the byte orders detailed in the datasheet, this means that when writing to a register of the ESP32, we must construct a first byte in which bit 7 is 0 and bits 6-1 contain the register address. After this, we can transmit the i-Card user ID data. If we need to read from the register, we must construct a byte where bit 7 is 1, and bits 6-1

contain the register address. We must then clock out the appropriate number of bytes, and the MFRC522 will send the requested data back ot the ESP32 using the MISO line.

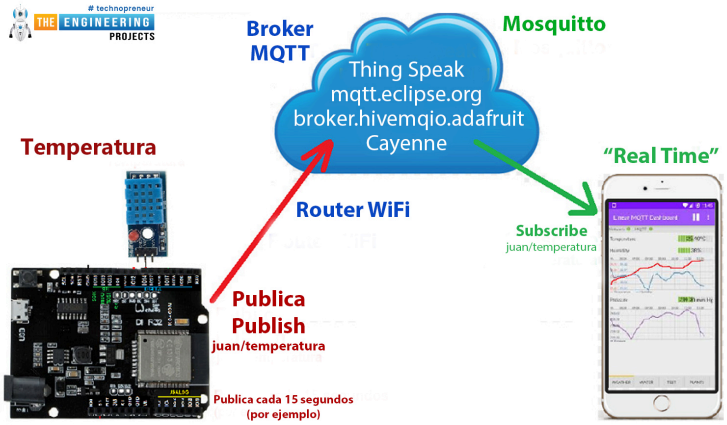
The following image details sample pseudocode that will need to be written to program onto the ESP32 to interface across the SPI protocol:

```
BEGIN PROGRAM
    // init hardware and spi
    Initialize SPI on ESP32
    Configure SPI settings according to datasheet
    Initialize MFRC522 module

    // send commands to MFRC522
    Reset MFRC522
    Configure MFRC522 settings

    // main loop to check rfid
    WHILE (true) DO
        Check if i-CARD card is detected
        IF card is present THEN
            Read card UID

            // r/w data
            IF operation is READ THEN
                Read data from i-CARD card
                Process and interpret the data

            ELSE IF operation is WRITE THEN
                Write data to RFID card
                Verify successful write operation

        END IF
        delay(500ms)
    END WHILE
END PROGRAM
```

2.5.3 AWS App Subsystem

The final software component of our project design involves the setup of an AWS app using the AWS IoT Core that utilizes the MQTT protocol for the ESP32. Taken from a sample project by Zoye Bella published in The Engineering Projects [12], this figure describes a basic overview of the MQTT protocol interface with the ESP32:

Fig 1: ESP32 MQTT Protocol



The MQTT (Message Queuing Telemetry Transport) is a messaging protocol that is designed for low-bandwidth applications. Due to the low resource-intensive nature necessary to simulate our project, it is a perfect way to interface with our AWS IoT Core. The protocol works by following a client-server model: The MQTT serves as the central server to manage all messages. The Client, which is our ESP32, will be the device that publishes these messages (ADC readings, RFID ID detection), to the broker.

Our overall communication flow will be as follows:
ESP32 → AWS IoT Core → AWS Services → Smartphone App

We begin by setting up the AWS IoT Core using Amazon's built-in AWS IoT Core Console. Since our program requires minimal resources and architecture, it can be designed within AWS' free services. Once this is established, we can write another program on the ESP32 to connect to the AWS IoT Core via MQTT (as the ESP32-Wroom-32D supports Wi-Fi). The sensor and ID data will likely be converted into a simple JSON format and published onto the MQTT, which supports secure connection to prevent user data from being breached. AWS Services (likely AWS Lambda) would be used to process and store the data. Using the REST API protocol, we will create an API gateway to connect to the AWS Lambda, which allows for data fetching from our mobile app. The mobile app will be a basic functional app, likely in React (a JavaScript Library).

2.6 Commercial Component Selection

2.6.1 Physical Design

The physical design of our simulated gym environment is simple enough, such that a majority of components will not be necessary nor will take away a majority of our budget. As seen in the physical design diagram, major components include a PCB

board, a foam pad, as well as a chair. The cost of our custom PCB has already been referenced in our cost analysis, and will not require any additional commercial components to purchase. A multi-purpose foam pad will cost ~$9.98 from Home Depot in Champaign, IL. The i-Card as well as a smartphone needed to interact with our application will both be free as we both own them. Finally, as mentioned in our software design section, utilizing the AWS IoT Core service for our purposes does not require enough usage or architecture to go beyond the free version of the service.

2.7 Tolerance Analysis

2.7.1 Pressure Sensor Force

When utilizing a pressure sensing module embedded within common points of contact in a gym machine, it is critical to be able to determine whether the equipment is truly being occupied. It is therefore important to consider the accuracy and reliability of the GHF-10 to ensure that it does not pick up random weight fluctuations and throw a false positive. As listed in our high-level requirements, our accuracy threshold is +/- 5lbs, leaving us little room for error when detecting miscellaneous environmental factors or other movement patterns that may throw our sensor off.

In response to this, we analyze the feasibility of this component, by evaluating the sensor response to applied weight and movement fluctuations using mathematical analysis:

We model the force applied on the sensor as:

$$F_{total} = F_{user} + F_{machine} + F_{fluctuation}$$

where:

- F_{user} is the force exerted by the gym-goer,
- $F_{machine}$ accounts for the resting weight of the equipment/padding onto the sensor that may affect sensor readings,
- $F_{fluctuation}$ represents noise due to minor weight shifts, mechanical vibrations, and temporary pressure variations.

The pressure sensor must differentiate between actual occupancy and these fluctuations. The sensor outputs a voltage proportional to the applied force:

$$V_{out} = kF_{total}$$

where k is a sensitivity constant specific to the sensor.

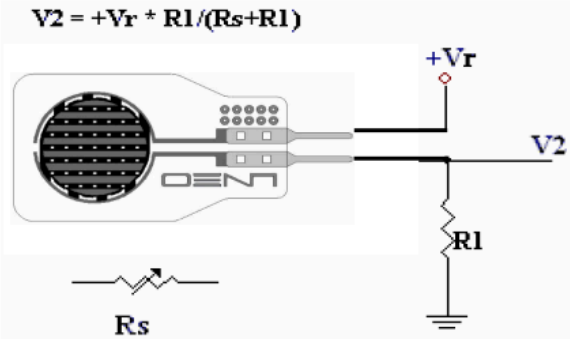
Given the specifications of the **GHF-10 pressure sensor**, the output readings could fluctuate with repeatability and hysteresis of ±2%, which translates to a weight fluctuation threshold of approximately 2-3 lbs. So we need to ensure a ±5 lbs accuracy requirement so that $\left|F_{fluctuation}\right| < 5$

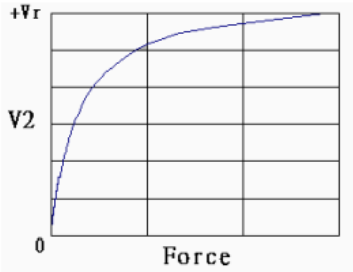
We implement signal filtering techniques such as:

- Low-pass filtering to smooth transient noise. [5]
- Kalman filtering to predict and correct erratic fluctuations. [7]

2.7.2 Pressure Sensor Voltage Divider

2. Using a fixed resistor R1 in a voltage divider configuration for an output V2 that increases with respect to added force.





Note:

1. R1 is 1Kohm to 100Kohm
2. V2 is 0 to +Vr

The figure above is an excerpt of the datasheet from the GHF-10 [1], detailing the circuit configuration necessary for the force sensor to work properly. As we know, a voltage divider is a circuit that produces the output voltage (V2), a fraction of the input voltage (+Vr). We need to set correct values for the resistors to keep our output voltage in line with the ADC pins of the ESP32 as well as the input voltage in line with our 3.3V power. As we can see from the diagram, as force is applied to the sensor, our resistance Rs decreases. If Rs decreases, the denominator of our fraction decreases and our overall value of V2 increases. This means that we are looking for the rate of change of the output voltage V2 connected to the ADC pin of our ESP32 microcontroller. We pay close attention to our non-linear relationship between force and +Vr, which is key to ensure we are interpreting our data properly.

We begin by establishing our given information. Our threshold for weight input should be above 50+ lbs, which accounts for the fluctuating weight threshold, pressure applied by the gym pad, and a realistic human gym member's minimum bodyweight. We want to

supply the ESP32's ADC pin within a reasonable range of 0-3.3V, being careful not to exceed those voltage requirements. The GHF-10 is rated for a maximum of 110 lbs, in which case the curve will flatten to protect against any unreasonably excessive voltage output.

With this in mind, let's assume we desire ~2.5V output at 50 lbs, enough to differentiate a user's occupancy. At +Vr = 3.3V and choosing Rs = 20kOhms (within the 1 kOhm to 100 kOhms limit set by the datasheet), we get:
5V = 3.3V * R1 / (20000 + R1)
5 * (20000 + R1) = 3.3 * R1
5 * 20000 + 2.5 * R1 = 3.3 * R1
R1 = 50000 / 0.8 = 25000 = 25kOhms.
This means that we can set our R1 to equal 25kOhms to get around 2.5V at 50lbs of force.

2.8 Cost Analysis

The total cost for the PCB's parts comes out to $33.55. Adding a 5% rough guideline as shipping cost and a 9% sales tax for Champaign County, IL, our total comes out to $38.40. At a budget of $50/member and 2 members at $100, this leaves plenty of room for more budget dedicated towards getting spare components as well as any other expenses that may come up. To estimate this value, let us use a moderate budget of 25% of our cost that requires re-ordering, which brings our total from $38.40 to $48. According to the admissions department at ECE Illinois, the average starting salary for an electrical engineer alumn is $87,769, which corresponds to ~$40/hr. A $40/hr salary * 2.5 (overhead costs for graduate work) * 150 hours is around $15,000 per team member. For 2 team members, this comes out to $30,000 total. Adding this to our parts total of $48, we get a total cost of ~$30,048.

Description	Manufacturer	Part #	Quantity	Cost (Total)
ESP32 Microcontroller	Espressif Systems	ESP32-WROOM-32D	1	Free (from 445 Lab)
RC522 RFID	NXP USA Inc.	MFRC52202HN 1,115	1	$7.91
Voltage Regulator	EVVO	AMS1117-3.3	1	$0.63
Crystal Oscillator	ECS	ECS-271-2-10-37-CKW-TR	1	$0.40
Schottky Diode	Diodes	1N5819HW-7-F	1	$0.25

	Incorporated			
USB-C	Korean Hroparts Elec	TYPE-C-31-M-12	1	$0.87
0.1 uF 0805 Capacitor	YAGEO	CC0805KRX7R9BB104	3	$0.24
10 uF 0805 Capacitor	Murata Electronics	GRM21BR61C106KE15K	4	$0.60
15 pF 0805 Capacitor	KYOCERA AVX	600F150JT250XT4K	1	$1.38
22 pF 0805 Capacitor	KYOCERA AVX	600F220FT250XT	2	$4.68
1 nF 0805 Capacitor	KEMET	C0805C102KDRACAUTO	1	$0.46
180 pF 0805 Capacitor	KYOCERA AVX	600F181FT250XT	2	$6.82
680 pF 0805 Capacitor	YAGEO	CC0805KRX7R9BB681	2	$0.20
470 nH 0805 Inductor	Samsung Electro-Mechanics	CIGT201210UHR47MNE	2	$0.30
1.6 uH 0805 Inductor	Murata Electronics	DFE201210U-1R5M=P2	1	$0.25
10kOhms 0805 Resistor	Susumu	RR1220P-103-D	1	$0.10
5kOhms 0805 Resistor	Vishay Dale Thin Film	PNM0805E5001BST5	3	$6.54
1kOhms 0805 Resistor	YAGEO	RC0805FR-071KL	1	$0.10
1.5kOhms 0805 Resistor	Susumu	RG2012P-152-B-T5	1	$0.10
10 Ohms 0805 Resistor	YAGEO	RC0805FR-0710RL	2	$0.22

2.9 Schedule

The following is a schedule that is split up based on weeks and rough deadlines of assignments due during the week or shortly following the week. This is to ensure that there is proper division of labor and that assignments are completed, especially with proper advance to ensure enough time is left over for any debugging. Note that because we are a 2-person team, each individual is expected to help out on every assignment if necessary. The role of the “Primary Contributor” simply means that the individual is responsible for the completion of the assignment. We designed our team contract agreement and schedule this way because it is the method our team has worked best since the first few weeks of the course. Some tasks may overlap across dates due to assignment deadlines being later in the week.

Week	Assignment	Primary Contributor
Week of Mar. 3rd - Mar. 10th	Design Document Teamwork Evaluation Breadboard Demo PCB Round 2 Order Component Ordering	Aryan Aryan and Kushal Kushal Aryan Aryan and Kushal
Week of Mar. 10th - Mar. 17th	Breadboard Demo PCB Round 2 Order Finalize Component Order	Kushal Aryan Aryan and Kushal
Week of Mar. 24th - Mar. 31	PCB Round 2 Soldering PCB Round 2 Debugging PCB Revisions for Round 3 Component Re-orders	Kushal Aryan Aryan Aryan and Kushal
Week of Mar. 31 - Apr. 7	PCB Round 3 Order Individual Progress Reports Component Re-orders PCB Round 2 Debugging Software Setup	Aryan Aryan and Kushal Aryan and Kushal Aryan Kushal
Week of Apr. 14 - Apr. 21	PCB Round 3 Debugging Software Setup Finalizing Design and Test Team Contract Assessment	Aryan Kushal Aryan and Kushal Aryan and Kushal
Week of Apr. 21 - Apr. 28	Mock Demo Prep Final Demo Prep Mock Presentation	Aryan and Kushal Aryan and Kushal Aryan and Kushal
Week of Apr. 28 - May 5	Final Demo Mock Presentation Final Paper	Aryan and Kushal Aryan and Kushal Aryan

	Lab Checkout Lab Notebook	Aryan and Kushal Aryan and Kushal

2.10 Risk Analysis

The nature of our design presents a low-risk to the user, both for our intended application as well as our simulated scenario for testing. When it comes to our physical simulation, we need to first establish proper safety with our PCB. We need to ensure safe power delivery to all our components and ensure all our peripherals stay within their power ratings. This involves rigorous testing in both our breadboard design as well as final PCB design with varying levels of power. Should any of our sensors fail or overheat, the testing must be concluded immediately and power should be cut off. Aside from this, there is little risk to the user. The foam pad that will be utilized will adequately protect the user from any damage to body parts. The gym user will simply be able to interact with the “gym equipment” as they normally would in any other scenario. If other situations arise during testing that pose a threat to any user of our device, we will be sure to mitigate these errors as best we can.

3. Ethics & Safety

- **User Data Privacy and Security**

  According to the IEEE Code of Ethics Section 1.1 [10], the intent of engineers should be to “hold paramount the safety, health, and welfare of the public”. By collecting and transmitting user data between the ESP32 microcontroller, AWS server, and mobile app, it is vital to encrypt this information using standard AES-256 encryption. Our goal will be to store as little personally identifiable information as needed, all of which will be securely encrypted.

- **Physical Safety of Equipment and Users**

  According to the ACM Code 1.2 [11], the goal of an engineer should be to “avoid harm… negative consequences, especially when those consequences are significant and unjust… include unjustified physical or mental injury, unjustified destruction…”. With this in mind, we understand how the custom PCB composed of pressure sensors and IMU modules must not only interfere with the intended

use of the gym equipment but also not pose any risk to the gym-goers. We will ensure that each electrical component is securely enclosed to mitigate exposure to electrical or flammable hazards. Our PCB operates at a low voltage from 0 - 3.3V, which already reduces risks of any electric shock or fire hazards. This will severely minimize any risk of harm to either the gym goers themselves, the equipment, or the gym establishment.

- **Support each other in maintaining Ethical Standards**

  According to the IEEE Code of Ethics 7.8.III.10 [10], we must support our team members in following this code of ethics and reporting violations without fear of retaliation. This will ensure transparency and accountability. We will include this in our team contract to ensure we all agree on it.

4. References

[1] Gentech International Ltd., *GHF10-500N Force Sensor Datasheet*, Accessed: Feb. 13, 2025. [Online]. Available: https://www.uneotech.com/uploads/product_download/tw/GHF10-500N%20ENG.pdf

[2] TDK InvenSense, *ICM-20948 Datasheet*, Ver. 1.3, Jun. 2016. Accessed: Feb. 13, 2025. [Online]. Available: https://invensense.tdk.com/wp-content/uploads/2016/06/DS-000189-ICM-20948-v1.3.pdf

[3] NXP Semiconductors, *MFRC522 Standard Communication Datasheet*, Accessed: Feb. 13, 2025. [Online]. Available: https://www.handsontec.com/dataspecs/RC522.pdf

[4] Espressif Systems, *ESP32 Series Datasheet*, Accessed: Feb. 13, 2025. [Online]. Available: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf

[5] Electronics Tutorials, *Active Low Pass Filter*, Accessed: Feb. 13, 2025. [Online]. Available: https://www.electronics-tutorials.ws/filter/filter_2.html

[6] Amazon Web Services, *Building an AWS IoT Core Device Using AWS Serverless and an ESP32*, Accessed: Feb. 13, 2025. [Online]. Available: https://aws.amazon.com/blogs/compute/building-an-aws-iot-core-device-using-aws-serverless-and-an-esp32/

[7] T. Lacey, *Kalman Filter Tutorial*. Available: https://web.mit.edu/kirtley/kirtley/binlustuff/literature/control/Kalman%20filter.pdf. Accessed: Feb. 13, 2025.

[8] ElectronicWings, *RFID-RC522 Interfacing with ESP32*, Accessed: Feb. 13, 2025. [Online]. Available: https://www.electronicwings.com/esp32/rfid-rc522-interfacing-with-esp32

[9] SparkFun, *SparkFun 9DOF IMU ICM-20948 Breakout Hookup Guide*, Accessed: Feb. 13, 2025. [Online]. Available: https://learn.sparkfun.com/tutorials/sparkfun-9dof-imu-icm-20948-breakout-hookup-guide/all

[10] IEEE, “IEEE Code of Ethics,” IEEE, 2020. [Online]. Available: https://www.ieee.org/about/corporate/governance/p7-8.html. [Accessed: 13-Feb-2025].

[11] ACM, “ACM Code of Ethics and Professional Conduct,” ACM, 2018. [Online]. Available: https://www.acm.org/code-of-ethics. [Accessed: 13-Feb-2025].

[12] Z. Bella, "ESP32 MQTT – Publish and Subscribe with Arduino IDE," *The Engineering Projects*, Nov. 2021. [Online]. Available: https://www.theengineeringprojects.com/2021/11/esp32-mqtt.html. [Accessed: 04-Mar-2025].

[13] "ESP32 Example," *ECE 445 Wiki*, Grainger College of Engineering, University of Illinois Urbana-Champaign. Accessed: Mar. 5, 2025. [Online]. Available: https://courses.grainger.illinois.edu/ece445/wiki/#/esp32_example/index