Differential Measurement of Involuntary Breathing Movements

Jacob Seman
Lillian Tucker

Project 5
EECE244
Spring 2023
Dr. Mellenthin

Table of Contents:

Overview

A differential measurement was chosen for this project. Two MPU6050 sensors on a wire harness were connected to an ESP32, and readings from the y-axes are compared. The sensors are to be placed at the xiphoid process and on the sternum to detect contractions at the xiphoid and measure them relative to the stable area of the sternum. A custom list class was written to store the MPU sensor data and average it over a defined number of samples, to eliminate measurement noise from the accelerometers. A heart rate and oximeter sensor were also connected to the wire harness. Biometric information is output to an OLED display and states information regarding logging data to the SD card. An access point hosting a web server is created by the ESP32, and a smartphone may be used to connect to the access point and control the remote start & stop of the logging function, as well as the file download and delete function. Selecting “File/Logging Start” mounts the SD card and applies a normalization factor based on the starting conditions (also displayed on the OLED and indicated by an LED) before commencing logging data at 100 Hz. A static message is displayed on the OLED indicating the logging status and correction factor. The LED flashes with each line of buffered data that is written. Selecting “File/Logging Stop” finishes logging and unmounts the SD card for safe removal before continuing to display biometrics. With each logging process, the CSV filename is incremented, so there is no concern about overwriting previous data. The ESP32 has two cores, and threading was used to control the accelerometer and OLED with core #0, while the SD card writing, LED state, and state machine management was controlled by core #1. This allowed reaching a consistent sample rate.

Components

Adafruit MPU6050:

The components used to measure involuntary breathing movements are two Adafruit MPU6050s using an I2C interface. These sensors are attached to the corresponding GND and 3V3 pins of the ESP32 for power. Since these sensors use I2C to communicate, an SCL(serial clock) and SDA(serial data) pin are required to operate. On the ESP32, pin 22 is used for SCL, and pin 21 is used for SDA so the sensors are connected accordingly. The sensors are to be placed at the xiphoid process and on the sternum since most of the movement during involuntary breathing movement occurs in this area.

Sparkfun Pulse Oximeter:

The component used to measure heart rate and blood oxygen levels is an SPO2 sensor that uses an I2C interface. This sensor is placed on the padding of the middle finger on the right hand for optimal reading. This sensor is attached to the corresponding GND and 3V3 pins of the ESP32 for power. Since this sensor uses I2C to communicate, an SCL and SDA pin is required to operate. On the ESP32, pin 22 is used for SCL, and pin 21 is used for SDA so the sensor is connected accordingly.
There is an example library and code that can be downloaded and used to detect heart rate, blood oxygen levels, temperature, and reading status. There are 3 different reading statuses for the SPO2 monitor: 1 means there is no finger detected, 2 means there is a finger detected but isn't being properly detected, and 3 means there is a finger being properly detected to display heart rate and O2 levels.

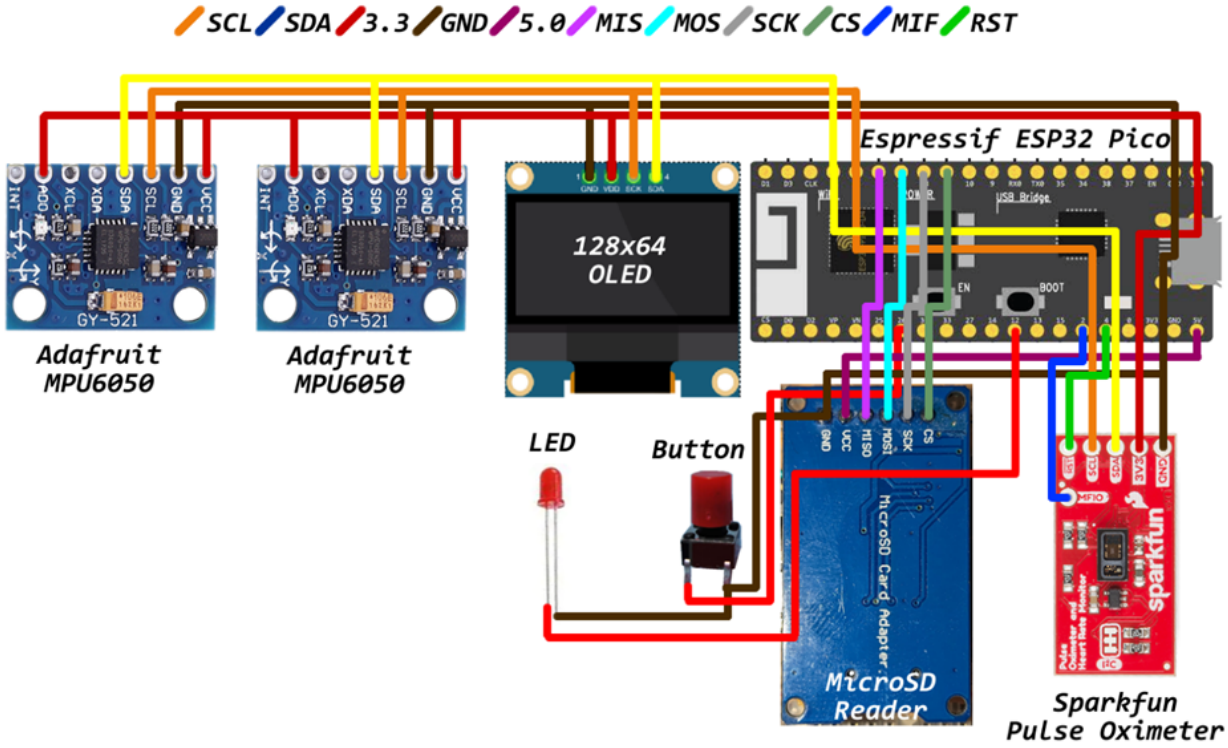
OLED Display Screen:

The component used to display heart rate and blood oxygen levels as well as displaying when the system is measuring and logging data is an OLED screen. This component is attached to the corresponding GND and 3V3 pins of the ESP32 for power. Since this component uses I2C to communicate, an SCL and SDA pin is required to operate. On the ESP32, pin 22 is used for SCL and pin 21 is used for SDA so the sensor is connected accordingly.

MicroSD Card Reader:

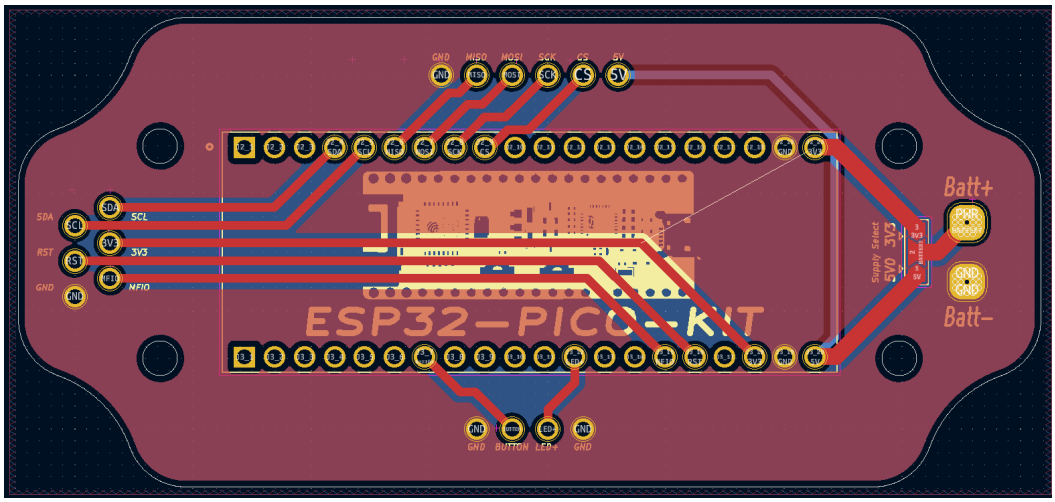
The component used to log and store the differential data from the accelerometers as well as the oximeter sensor data is a microSD card adapter that uses an SPI interface. This component is attached to the corresponding GND and 5V pins on the ESP32 for power. Since the component uses SPI to communicate, a MISO(master in slave out), MOSI(master out slave in), SCK(serial clock), and CS(chip select/slave select) pin are required to operate. On the ESP32 pin 19 is used for MISO, pin 23 is used for MOSI, pin 18 is used for SCK, and pin 5 is used for CS. A state machine is used for logging data to the microSD card. When using remote start to start logging on the microSD card, a static text is displayed to inform the user it is not safe to dismount the SD card. When using a remote stop to stop logging on the microSD card, a static text is displayed to inform the user when it is safe to dismount the SD card. An LED is used to show when data is being logged onto the microSD card. When data is being written onto the SD card, the LED will turn on. Otherwise, the LED will turn off.

Wiring Diagram



PCB Design

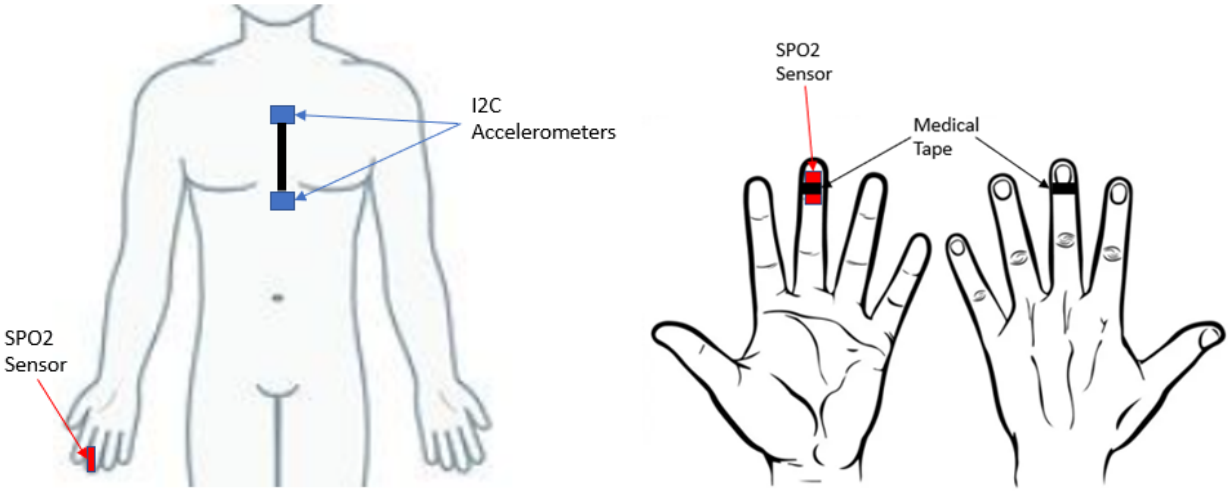
A dual-layer PCB design to mount the microcontroller into a plastic clamshell enclosure was created in KiCAD.



This PCB design provides pads for the I2C wire harness to the accelerometers and SPO2 meter, pads for the SD card reader, pads for the button and LED, and perforated pads for battery power with a solder-bridge option to select 3.3V or 5V supplies. The design shape fits the device enclosure and mounting pattern.

Placement

The I2C accelerometers are to be placed with one on the sternum and one over the xiphoid process. The accelerometers may be taped down over clothing provided the clothing is not excessively loose-fitting. The SPO2 sensor may be placed on the pad of a fingertip using tape or the provided velcro strap. To avoid intermittent readings from the SPO2 sensor, it is important to ensure the sensor is not pressing firmly into the pad of the fingertip - very light pressure that keeps the sensor in place without squeezing the fingertip is best. The wire harness is sufficiently long to accommodate any body shape and size, and may be oriented so that the subject is most comfortable. The microcontroller enclosure is not sensitive to orientation and may be placed nearby the subject rest area however is most convenient.



Program Code

The program code differs from typical Arduino programming in that it utilizes Free-RTOS tasks to make use of the two available cores on the ESP32-Pico. Free-RTOS is a real-time operating system kernel provided as standard with Arduino libraries. Using task pinning, functions are able to be assigned to a specific core and will run indefinitely on that core until the task is canceled, similar to the typical program loop, which by default only runs on core #1. This multi-core "threading" allows the grouping of functions that operate across like sensors, such as those using the I2C bus, or those using the SPI bus. In this specific case, the function "sensorRead" is pinned to core #0, while the function "sdWrite" is pinned to core #1. These functions were written such that all I2C bus devices are handled by core #0, while SPI and analog pin states are handled by core #1. In short, the accelerometer readings and OLED display updates are handled by one core, and all SD card writes as well as button state and LED pin state are handled by the other. This breaks up the tasks handled by the microcontroller into two parallel loops that run indefinitely.

Some conflicts can occur with multi-core threading, so care was taken to ensure that no data is written to the same location by both cores simultaneously. In this case, core #0 only reads accelerometer data, and writes temporary data to the "fauxList" class where the most recent 10 accelerometer difference samples are stored. Core #1 reads and averages this data before writing it

out to the SD card. Because the accelerometer readings are performed over I2C, and the SD card writes are performed over SPI, there is no conflict between the two processes.

Both cores are beholden to a state machine with four distinct states. When the microcontroller is powered on, peripherals are initialized and both threaded tasks are started, each with their own initialization routines before entering their primary loops. As part of this initialization, "loggingState" (an integer) is set to '0'. In this state, the accelerometers and pulse-oximeter sensors are read, averaged, and reported to the OLED display with some graphical elements. When the button is pressed, the SD card is started and "loggingState" is set to '3', where:

- A correction factor is calculated from the current accelerometer difference to be applied to all data during logging.
- The OLED is set to display a static warning and the correction factor that is applied.
- A filename name string is created and formatted with a number that is incremented for each logging session.
- A file with the previously created filename is opened for writing on the SD card.
- Column headers are printed to the file.
- "loggingState" is decremented to '2'.

After entering the next state, core #0 will only make sensor reads, and it will do so as quickly as possible. Core #1 will print a character buffer of formatted data at an interval controlled by a flag set by a hardware timer that runs from ICACHE RAM. This interval is able to be specified by setting the "DATAFREQ" definition, which is the rate (in Hertz) that the flag is set. During this state, the OLED is static to allow the sensor reading to happen at maximum speed. When the button is pressed a second time, "loggingState" is decremented to '1'.

In "loggingState" '1', the file is closed, the LED status is turned on, and the correction factor is disabled. After two seconds, the SD card is unmounted for removal, the LED is turned off, and "loggingState" is decremented to '0' to return to the initial state.

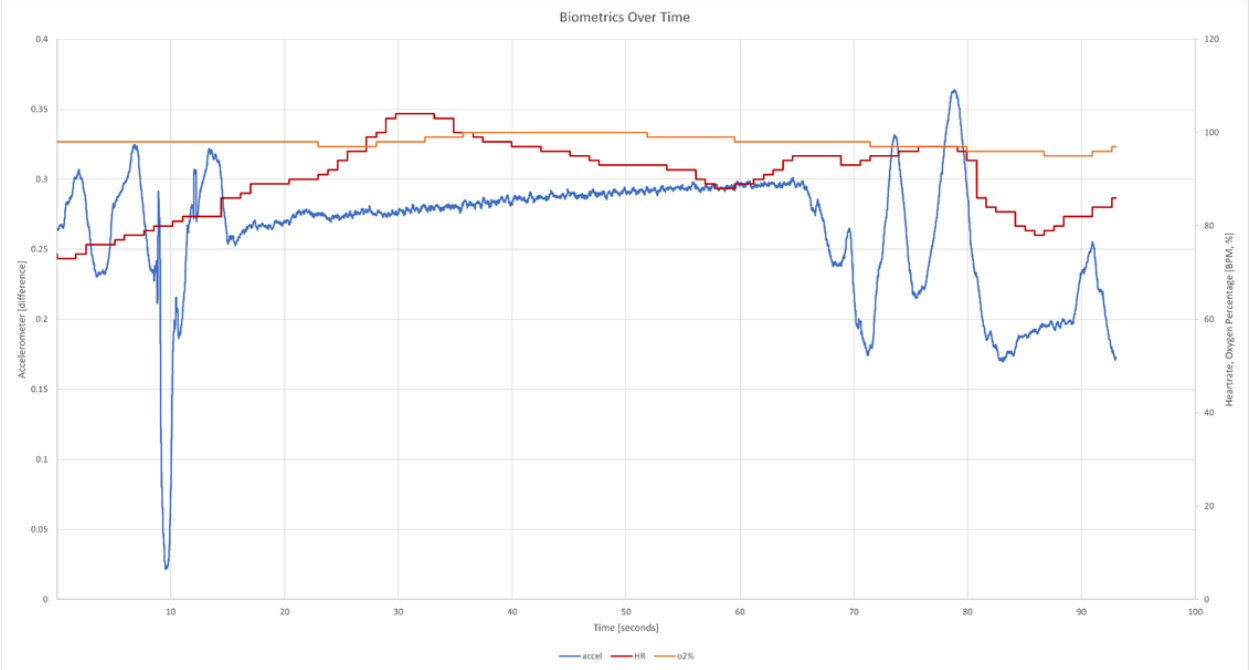
"loggingState" is changed from '0' to '3' anytime the webserver enters the "/loggingst" HTML page, and is changed from '2' to '1' anytime the webserver enters the "/logging" page. These pages are accessed by selecting the "Logging" header tab, and then toggling the "Logging Start/Stop" button.

File download and delete functionality is also available from the web server over WiFi. It is recommended to control the remote start/stop from a smartphone, and to access and download log files seperately from a laptop, although it is possible to download log files from a smartphone if needed. Multiple devices may connect to the microcontroller access point simultaneously, but it is not recommended to start or stop the logging process from anywhere but a single smartphone at a time.

Data

The data may be read by opening the CSV file with MS Excel and creating an XY chart using the data, or by importing the data into Matlab using the provided script. The data is written to the SD card as a CSV file, with a millisecond timestamp in the first column, accelerometer difference in the second column, heart rate in the third column, and oxygen percentage in the fourth column. Normal breathing, a brief breath hold, additional breathing, and then a short, exhaled breath hold are visible in the accelerometer difference plot below. The units for the accelerometer differential measurement are relative to the sternum and are a measure of the difference magnitude between the two accelerometers. Averaging over 10 samples proved effective at eliminating an appreciable degree of noise from the MPU data.

Output plot showing accelerometer differential measurement (blue), oxygen percentage (orange), and heart rate (red):



The data is collected at a 100Hz sample rate, and a MATLAB script to interpret and plot the data is included. The IBM movements are able to be interpreted as a regular oscillation in the "accel_difference" trace, and the beginning of these oscillations marks the onset of the body's physiological response to extended breath-holding. A scaling factor of x10 is applied to the accelerometer data in MATLAB to align the trace just below the heartrate and oxygen percentage traces.

Postural changes may appear on the accelerometer trace as fluctuations and offsets that do not align with any regular oscillations. Often these posture adjustments result in a new "at-rest" value for the accelerometer difference trace. Most postural adjustments are mitigated or minimized by the differential nature of the measurement, and the fact that the accelerometers are only polled on a single axis. The data is provided with a millisecond timestamp that is converted to seconds by the MATLAB script. This timestamp starts from zero for each logging event.

Specifications

Aspect:	Value:
Power Supply	5V 2A USB-Micro, Battery
SD Card Format	5Gb, Fat32
Maximum Sample Rate	~800Hz
Default Sample Rate	100Hz (Configurable)
Data Point Averaging	10 Samples (Default, Configurable)
Data Output	4 Columns (Incl. Millisecond Timestamp)
Data Format	*.CSV File

Troubleshooting

Oximeter Readings Intermittent/Status Reads 0:

If there is trouble getting a consistent reading from the oximeter sensor, check the placement of the sensor. Ensure the reader is completely covered by the padding of the finger. If readings are still inconsistent, check the pressure on the sensor. If there is too little or too much pressure, reading will not be consistent. Note that readings are usually less consistent when testing on an individual of a darker complexion.

Multi-Threaded I2C Components:

If using multiple I2C components on one set of I2C pins (SDA and SLC), ensure each component has a unique address. This can be done by using the Wire.beginTransmission() function with a unique address in the parenthesis. If each component does not receive a unique address, the components will try to write to the same place.

Cannot Connect to Web Server:

If there is trouble loading/connecting to the web server in your browser:

1. Ensure that the IP address of the web server is correct. The default is “192.168.4.1”.
2. Check that the password used to access the web server is correct. The default is “12345678”.
3. Make sure the libraries and devices are up to date.
4. Check the libraries; There are reports that the libraries including “wifiNINA.h”, “Ethernet”, and “webserver” can be unreliable and incompatible with certain codes and devices.
5. Try using a different browser or different device. The device was found to crash when logging was started from a Windows 10 laptop, but worked consistently without issue when used with an Android smartphone.

Linked below are forums regarding to issues and solutions with connecting and utilizing the web server:

https://www.reddit.com/r/esp32/comments/n5qb2m/esp32_board_seems_to_be_connecting_to_my_network/

https://github.com/espressif/arduino-esp32/issues/2977

https://github.com/espressif/arduino-esp32/pull/7686

Tutorials/References Used

WiFiNina webserver testing -
https://docs.arduino.cc/tutorials/uno-wifi-rev2/uno-wifi-r2-hosting-a-webserver

Initial Access Point testing - https://randomnerdtutorials.com/esp32-access-point-ap-web-server/

WiFi Webserver & SD data download - https://www.youtube.com/watch?v=zoYMU1tA3nI

Inserted webserver/client functions and CSS stylesheet
Modified HTML to add and support remote start/stop
Removed example file upload functionality
Bug tested, found pending pull request to resolve bug relating to our project (unable to start logging from PC, but works from Android smartphone)

Articles and Documents

Linked below are studies and reports for measuring involuntary breathing movements with prolonged breath holding:
https://www.mdpi.com/2673-7078/2/4/41/pdf
https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6908740/
https://www.tandfonline.com/doi/abs/10.1080/13102818.2010.10817856