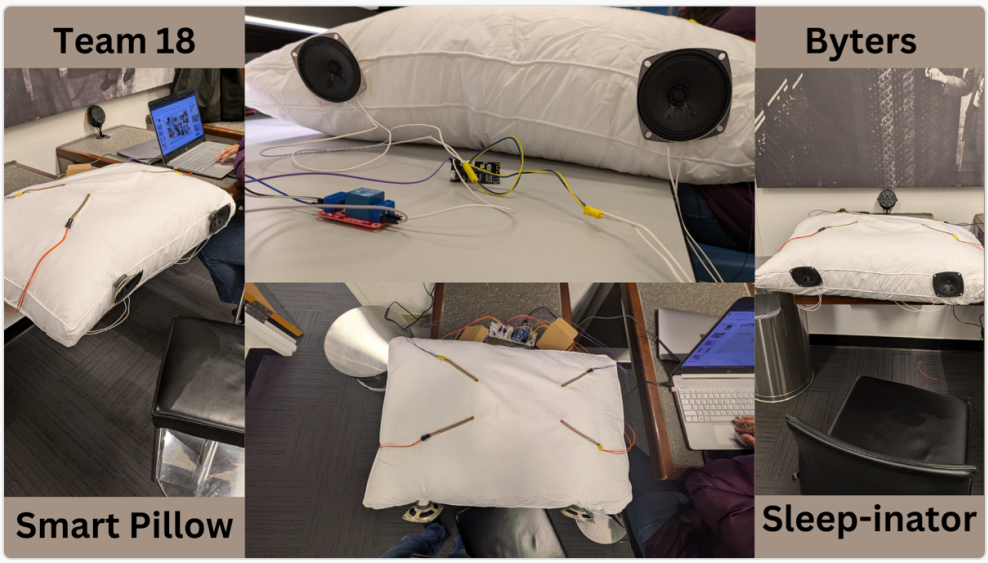
Team Byters Final Submission

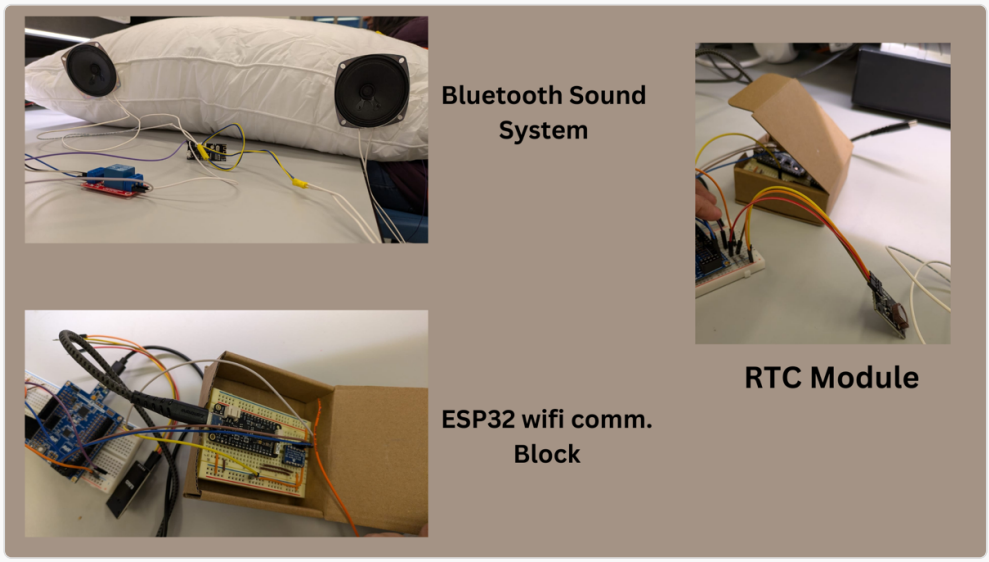
- **Team Number:** 18
- **Team Name:** Byters
- **Team Members:** Siddharth Ramanathan and Urvi Haval
- **Github Repository URL:** https://github.com/sidram1705/Sleep-inator.git
- **Description of test hardware:** Analog Flex Sensors, RTC module DS1307, MH-M38 Bluetooth Audio Transfer Module, 5V one channel Relay module, Atmega328PB, Macbook Air M1

Demo Video

Click here for demo video

Images





Project Summary

Device Description

- Sleep-inator is a smart pillow to equipped with bluetooth speakers and flexi-sensors. The goal of the project is to help people sleep better and monitor their sleep.

- The ATMega328 PB acts as the main MCU for this project as it reads the sensor data from flex sensors through the ADC functionality, acts as the power source for the BT speaker system and transmits sensor data to ESP32.

Inspiration

- In today's increasingly noisy world, finding uninterrupted sleep is harder than ever. From bustling city sounds to unexpected household noises, these disturbances can impact sleep quality and overall well-being.

- With modern lifestyles often leading to poor sleep quality, the idea was to create a product that not only monitors sleep patterns but also actively aids in improving sleep through adaptive technology.

Device Functionality

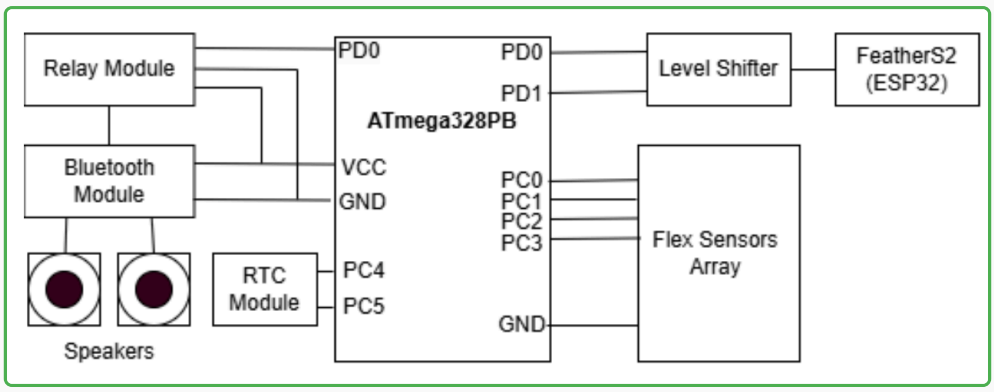
- The project has 3 main parts: The flex sensor array, the Bluetooth module and the RTC. The code starts the RTC to begin recording the time from 00:00:00, and keeps on updating. The flex sensors are constantly running in parallel, but the values are not read for every ADC trigger. The functionality of the code is to read the sensor values at every 30th minute in order to see if the previous and the current readings are different. In our demonstration, we've set the reading duration to be 10 seconds.

- Once the 10 seconds have elapsed, the sensor values are read from all four sensors in a loop. In each of these individual iterations, the sensor value is read and we compare the difference of this value and the previous reading of

this particular sensor 10 seconds ago to be greater than a particular threshold (Threshold value = 10). If this condition is met, we say that there has been a bend in the sensor and this triggers the relay which connects the Bluetooth module. So effectively this brings our main functionality where the bluetooth speakers can play songs or white noise as long as there's movement detected on the pillow. Here's the pseudocode for the above explained mechanism:

```
{
    If 10 seconds have elapsed:
        loop(for each sensor)
    {
        If (current sensor values - previous sensor values) > 10
        {
            Relay_on
        }
        Previous sensor value = current sensor value
    }
}
```

System Block Diagram



Firmware Overview

RTC:

The real time clock is an I2C device, which led us to write a custom driver. The code structure for the RTC is as follows:

- The i2c.c file has register and pin configuration commands which were provided in the Atmega328PB's datasheet. The I2C_Init initialises sets the prescalar to 1 and the SCL frequency to operate in 100kHz. The I2C_Start function sends the start condition of the I2C communication, the I2C_stop send the stop condition, I2C_Write loads the data which is received as a function parameter, the I2C_Read_ACK will read with acknowledgement and the I2C_Read_NACK will read without acknowledgement.
- The rtc.c has functions to Set time and Get time. The RTC_Init function just calls the I2C_Init. The RTC_SetTime function follows an ITC data write operation by calling the I2C functions in the following

sequence: I2C_Start(), I2C_Write(address), I2C_Write(data) and I2C_Stop(). The RTC_GetTime function follows an ITC data write operation by calling the I2C functions in the following sequence: I2C_Start(), I2C_Write(address), I2C_Read_ACK(seconds), I2C_Read_ACK(minutes), I2C_Read_NACK(hours) and I2C_Stop().

Main file:

It begins by configuring the UART for communication, setting up the ADC for analog-to-digital conversion, initializing the relay pin as an output, and preparing the RTC for timekeeping using I2C communication.

- The RTC is set to a default time (12:00:00 PM), and the seconds are recorded at every 10-second intervals.
- Inside the while loop, the program continually reads the current time from the RTC, converts it from BCD format to decimal format, and formats it as a string to send over UART for logging.
- Every 10 seconds, determined by comparing the current seconds with the initial seconds, the program reads sensor values from multiple ADC channels. It compares the current sensor readings to previous values, calculating the difference to detect significant changes which are defined by a threshold.
- If a significant change is found in any sensor, the relay is toggled ON; otherwise, it remains OFF. Each sensor value and the current relay status are transmitted via UART for monitoring.

Application Logic

1. The RTC module is used to track time and trigger actions every 10 seconds.
2. The flex sensors are connected to the ADC channels and the functionality of a multiplexer is achieved by using ADMUX.
3. At every 10th second, the logic checks if the current value of the sensors is different from the previous value by a threshold.
4. When the sensors are bent (person is not asleep), the relay toggles and the bluetooth speakers are turned ON. When the sensors are not bent after the next 10 seconds and their value is constant, the relay toggles and the bluetooth speakers are turned OFF.
5. The OFF time of the relay is sent over to the Blynk app using ESP32.

Software Requirements Specification & Results

SRS01: Flex sensor array software performance

Validation: Initially the plan was to achieve multiplexing using an hardware MUX, but instead the ADMUX register on ATmega328PB was used for its flexibility. Channel switching was implemented successfully by masking the required bits with the respective channel and the ADMUX register. This software requirement also entailed coming up with an appropriate threshold for how different the sensor value is from its previous one. This was achieved by testing various cases by applying pressure on all areas of the pillow.

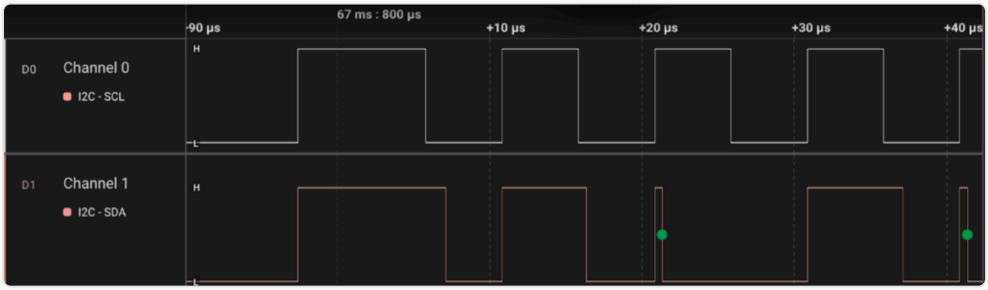
```
        void ADC_init() {
            ADMUX = (1 << REFS0);
```

```
        ADCSRA = (1 << ADEN) | (1 << ADPS2) | (1 << ADPS1);  //prescaler of
    }

    uint16_t ADC_read(uint8_t channel) {
        ADMUX = (ADMUX & 0xF0) | (channel & 0x0F);
        ADCSRA |= (1 << ADSC);
        while (ADCSRA & (1 << ADSC));
        return ADC;
    }
```

SRS02: Implementing the RTC drivers

Validation: We chose the DS1307 as the RTC. It was interfaced with the microcontroller via the I2C protocol. The microcontroller's I2C pins were configured with the appropriate pull-up resistors. The driver development consisted of writing the necessary functions according to the configuration required by the driver and the microcontroller. The successful working was verified on the Salae Logic Analyzer.



SRS03: Tracking user's sleep time using ESP32 and the Blynk app

Validation: The project uses the Feather-S2 to transmit the duration for which the relay remains turned off to the Blynk server. This was successfully achieved by establishing communication between the ATmega328PB and the ESP32 and configuring the datastream on the Blynk console. Initially the ESP32 was not able to receive the data from the microcontroller due to some problems in the PC's COM PORT. The cause was verified by using the logic analyzer. The implementation was programmed using the Arduino IDE, ensuring reliable integration and efficient data transmission.

Hardware Requirements Specification & Results

HRS01: Implementing the flex sensor array

Validation: The flex sensor is not always accurate and did not work the exact same every time. Also, implementing multiple of these in the project was a challenge. Initially, we used the sensors with a voltage divider but upon soldering and attaching them to the pillow we received some junk values. Upon using them with OPAMPs (LM358), the values were significantly consistent. In the current implementation, there are random spikes in the values of the sensor in every 8 or 10 rounds. However, it does not significantly change the working of our project.

HRS02: Bluetooth speaker connection and working

Validation: The bluetooth speakers are soldered to the bluetooth module and the logic relay. The relay is toggled OFF when there is no change in the value of the flex sensor from the previous value, measured every 10th second.

Trickiest Part of the Implementation

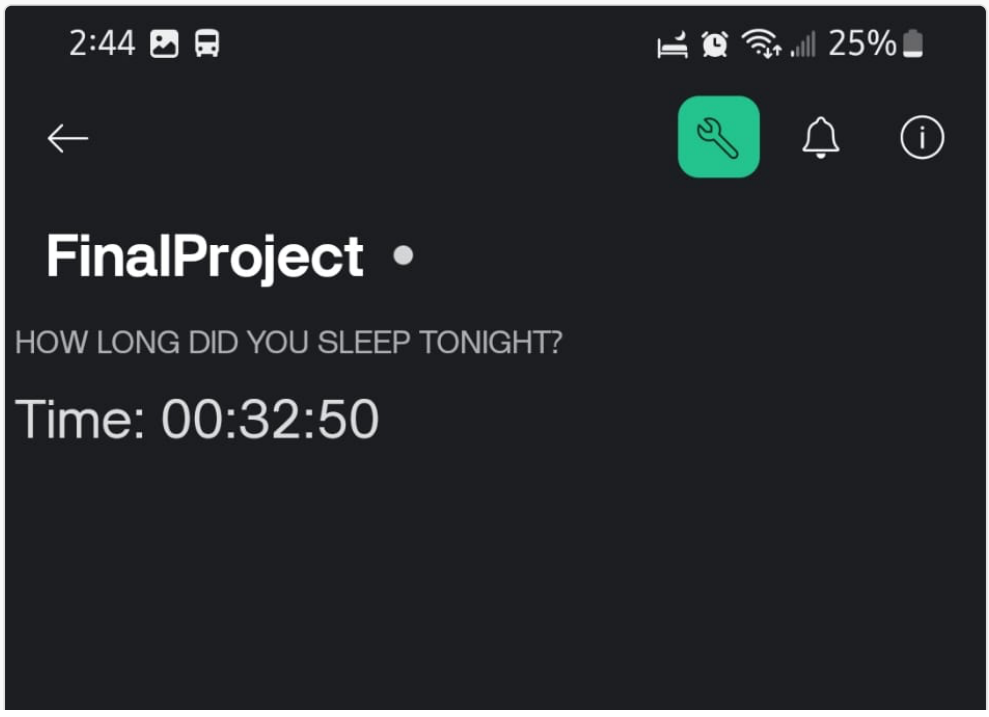
While the sensors work perfectly on the breadboard, when we soldered them and pasted them on the pillow, there seems to be some error in the connections. The flex sensors are giving random values after a few rounds of checking. After a discussion with the TAs, we soldered the flex sensors again.

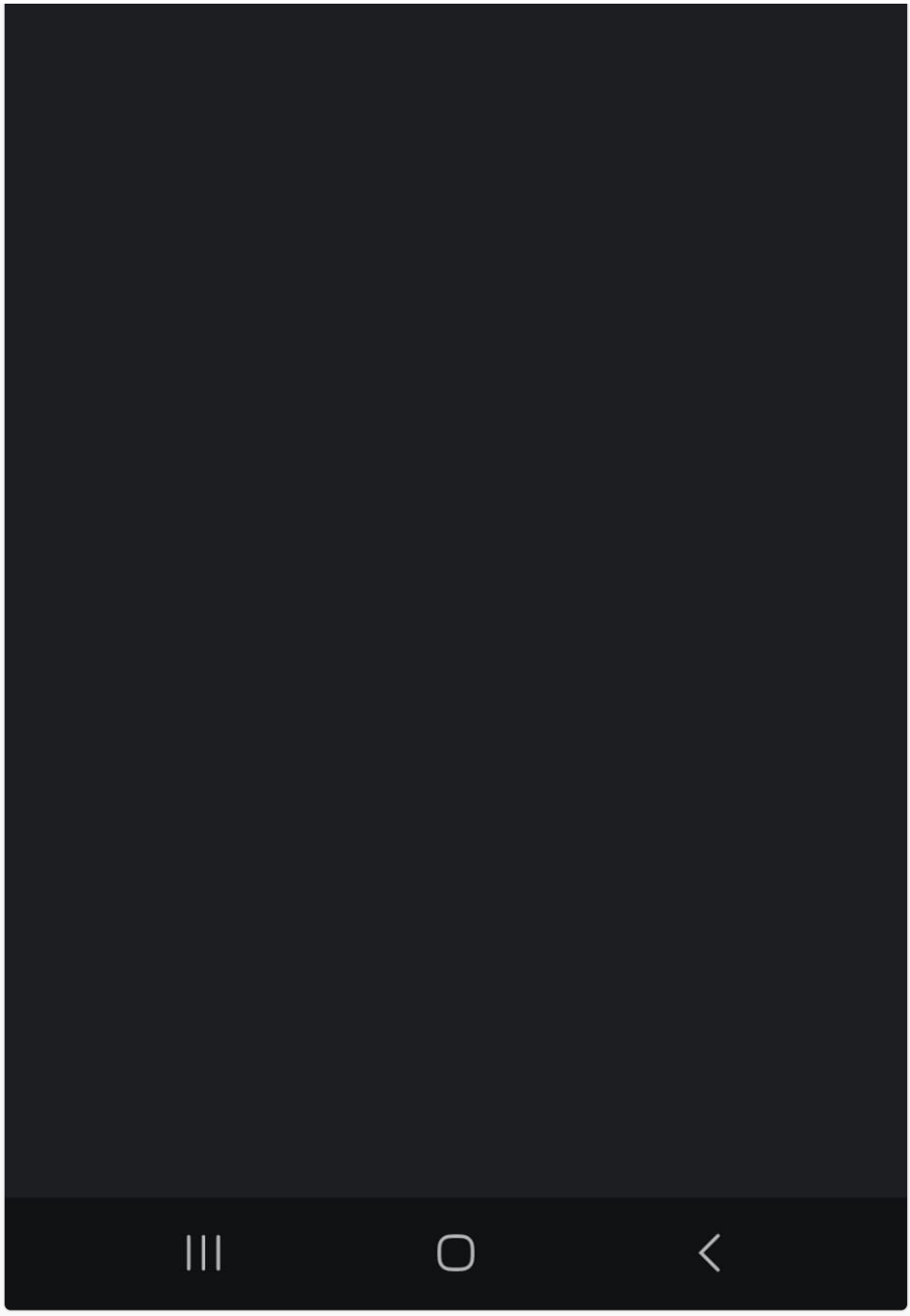
Conclusion

The project combined various topics that we learned during ESE5190. We were able to integrate RTC, ADC, UART and drivers for making a really fun project. We also learned how to effectively debug any problem that we might face while working with embedded systems - and the problems are pretty common than what we had thought before. Our project idea was fairly straightforward and we did not aim to introduce too complicated concepts right from the start, which helped us build things as we went forward. Speaking of things about the project we are proud of, we were able to implement a software driver from scratch. Using the logic analyzer to see and understand the way I2C works bit by bit was humbling (but fun nonetheless). We did face a few problems with the flex sensors having a mind of their own, but after talking to TAs and researching more about the sensors, we were able to solder them correctly and use a hot-glue gun to make sure there was no contact. The next step for the project would be trying to make the pillow as usable as possible, and not just a showcase project. One step we took towards that was sticking the flex sensors on the pillow using command strips so that the pillow can be washed by simply removing the electronics off of the velcro. The Blynk app can also be scaled to provide more analytics regarding the person's sleep over a period of days. The circuitry also needs to be made non-invasive, as currently there is a bunch of wires lying on the side of the pillow. Overall, this was a really fun and informative project. All thanks to Nick and the teaching staff (>W<) !

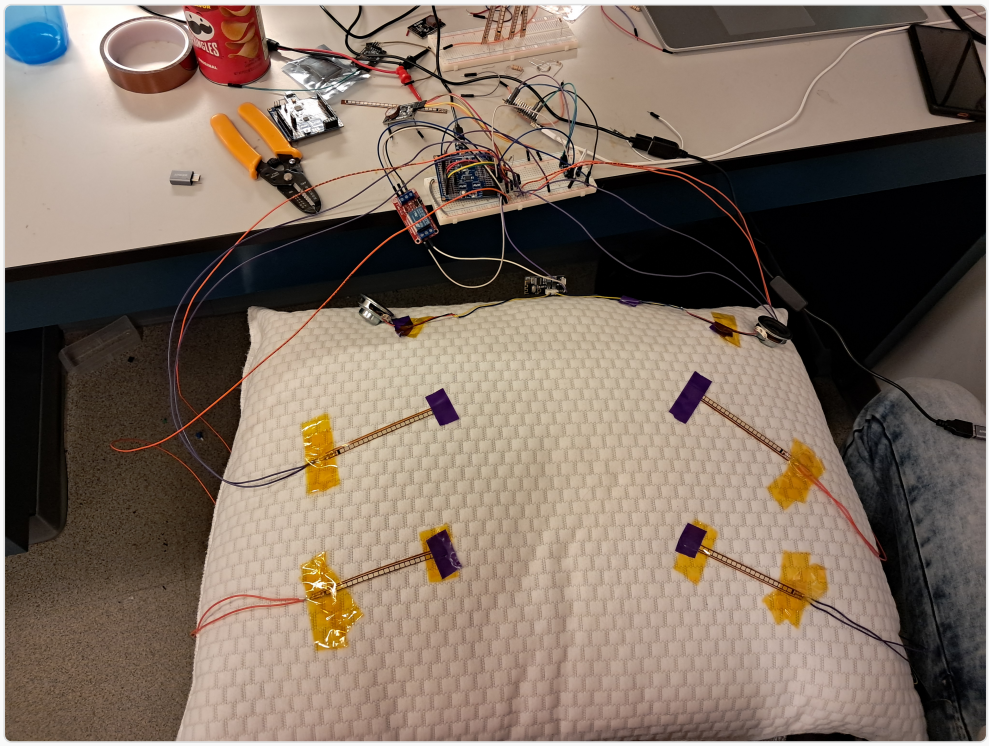
Images and Screenshots

Blynk App:

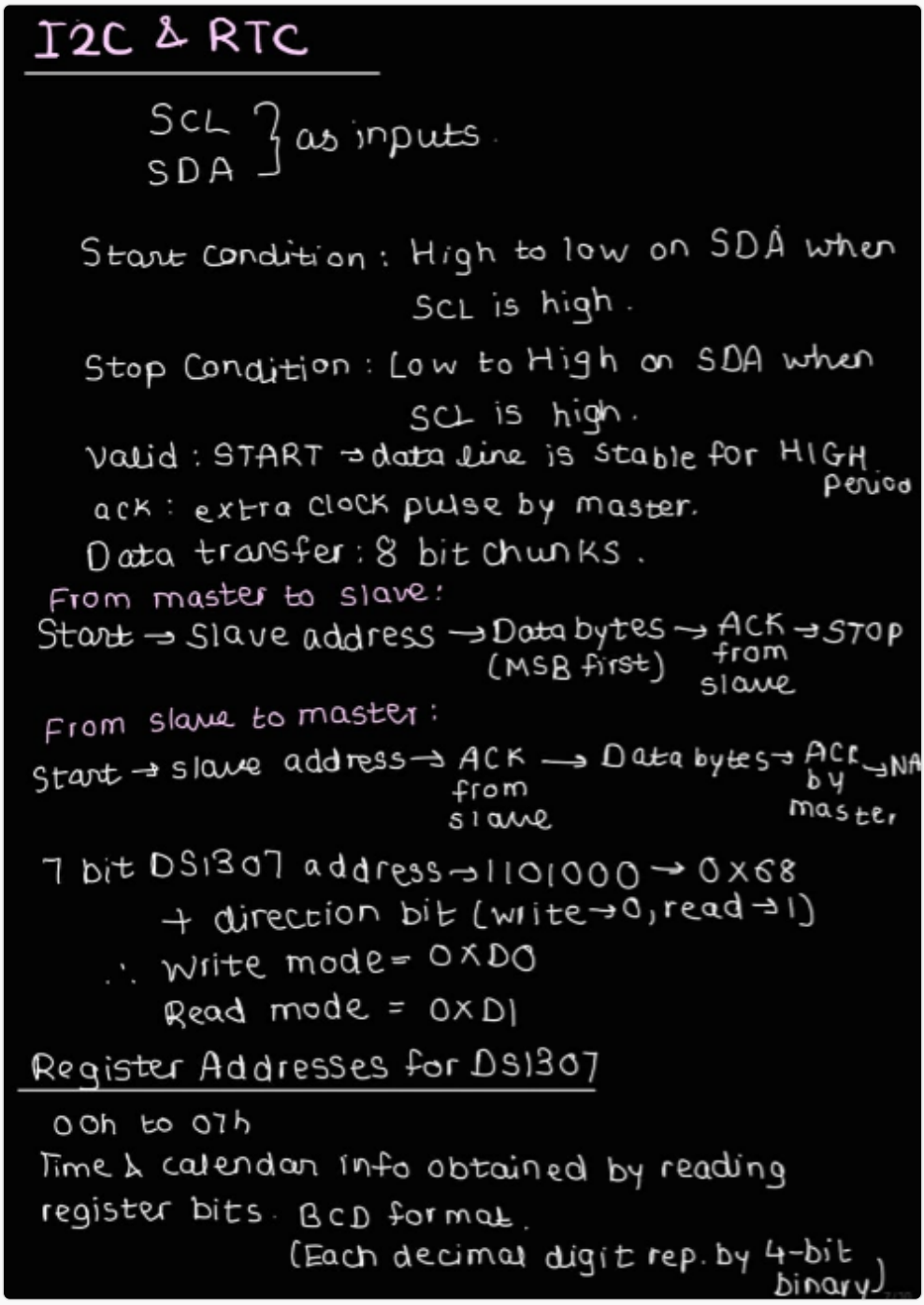




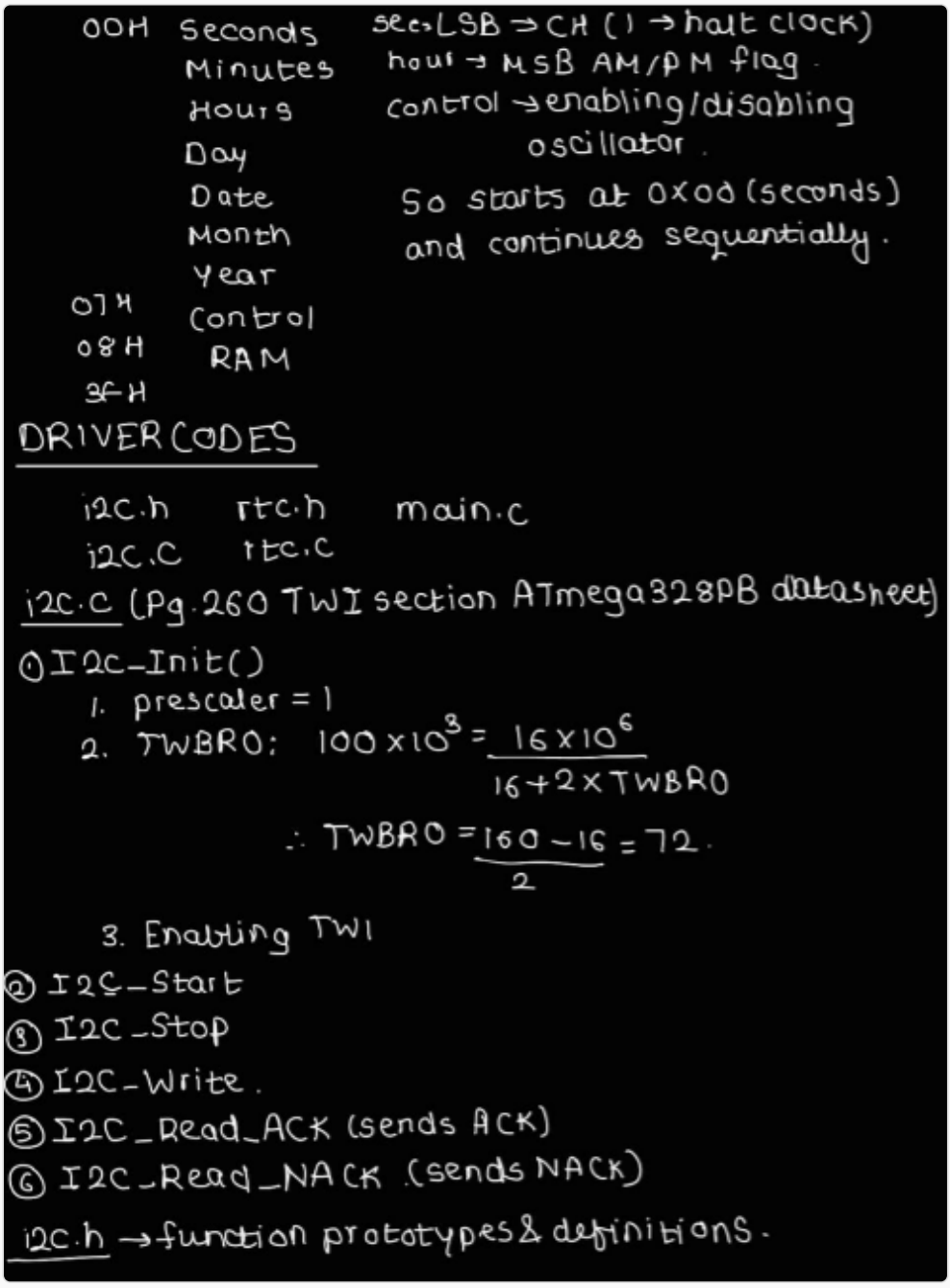
The hero during the MVP presentation:



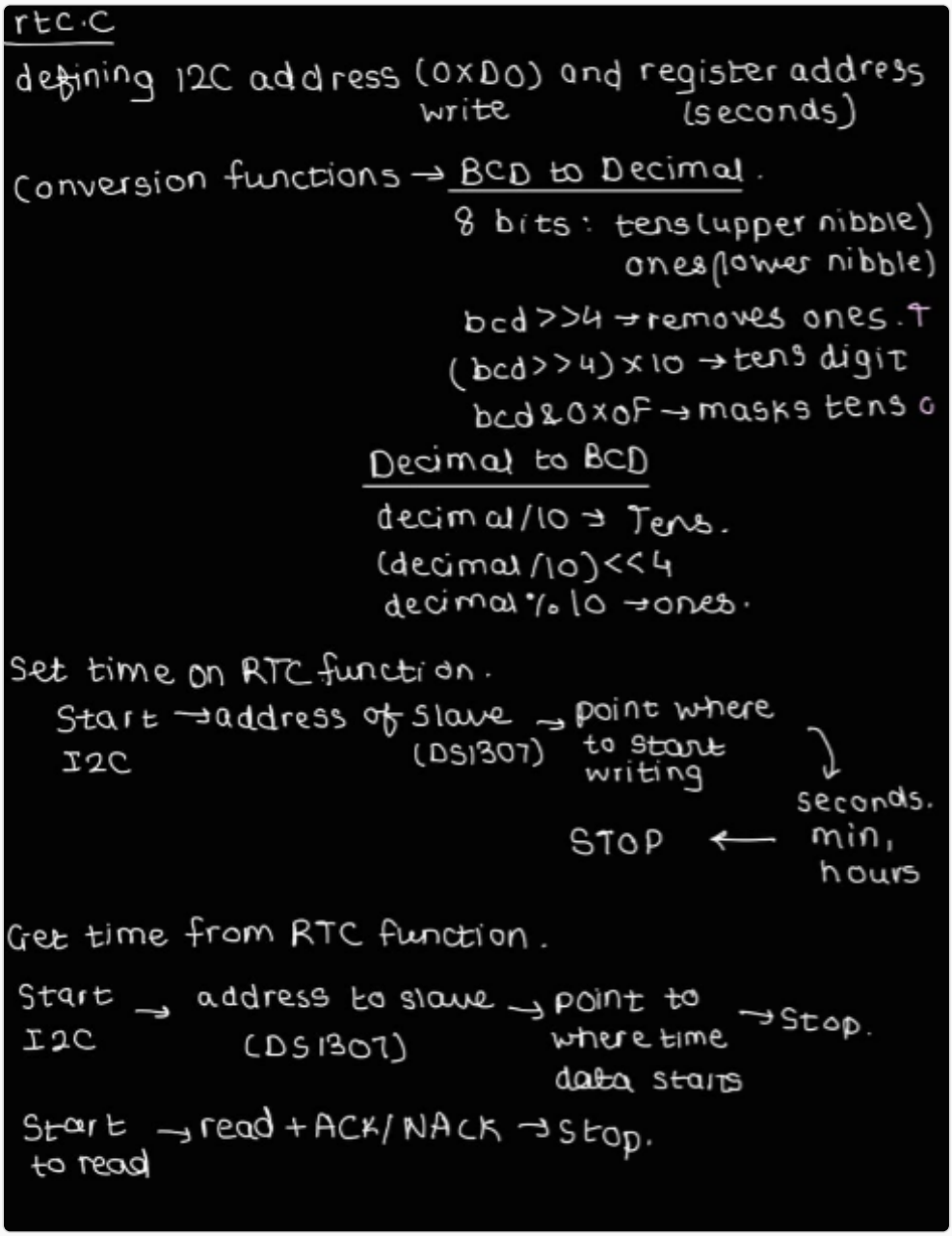
Trying to learn i2c :) :



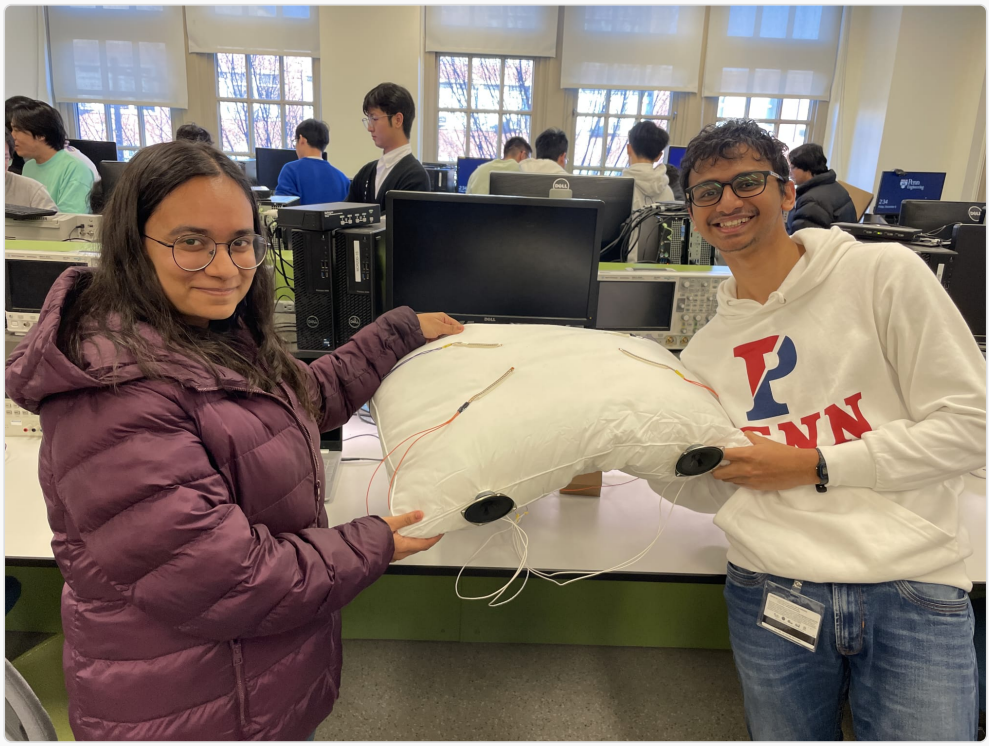
Trying to learn i2c part 2 :) :



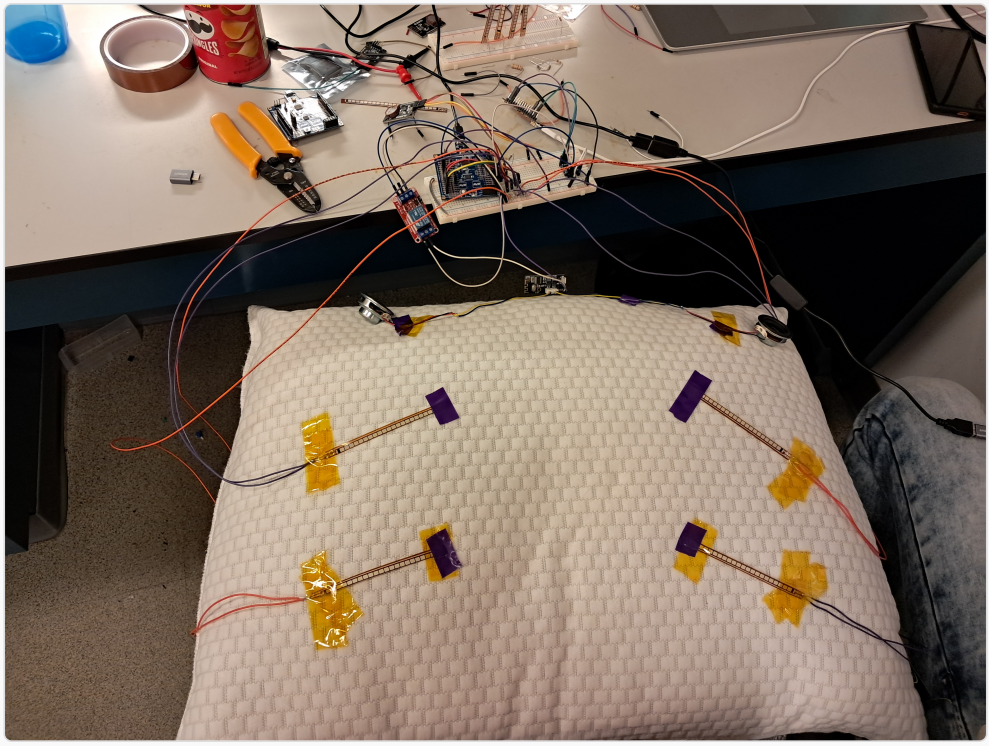
Trying to learn i2c (you get it now):



Showing it off



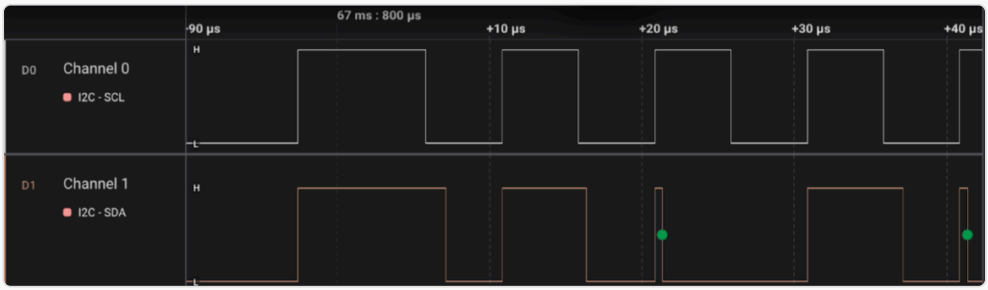
MVP



Software Requirements Specifications

- **SRS01: Setting up the RTC drivers**

  Validation: The RTC module (DS1307) had a verified output from the logic analyzer, showing the expected I2C behaviour.



- **SRS02: Configuring the software MUX on Atmega328PB**

  Validation: Successfully implemented channel switching by masking the required bits with the respective channel and the ADMUX register. Functionality demonstrated in the video.

Hardware Requirements Specifications

- **HRS01: The relay toggles ON when the value of the sensor changes significantly from the previous value.**

  Validation: We set up the flex sensors on a breadboard initially. After doing a few trial runs, we decided on a threshold of 10. If the absolute value of the sensor change is greater than 10, the relay toggles ON. This functionality can be observed in the demo video.

  Demo Video

Final Tasks

We plan to make a dashboard for displaying the analytics regarding the person's sleep. We also plan on making a 3D-printed casing to keep the microcontroller.

Device Demo

Link to the demo: Device Demo

Trickiest Part of the Implementation

While the sensors work perfectly on the breadboard, when we soldered them and pasted them on the pillow, there seems to be some error in the connections. The flex sensors are giving random values after a few rounds of checking. After a discussion with the TAs, we have decided to solder the flex sensors more accurately again.

We are still in the process of integrating the ESP32 and sending the sleep data (time elapsed since the toggle last turned OFF). We also plan to make the pillow look better by adding a pillow cover and hiding the wires and the microcontroller in a 3D-printed casing.

Help Needed from the Teaching Team

Currently we just need to figure out a way to solder the sensors or make the decision to completely use new sensors because of the junk values we are getting.

Weekly Sprints

Sprint Review #2

Current State of Project

Multiple (4) flex sensors can work together on different ADC channels. When the sensor is bent, the relay turns ON and starts the Bluetooth speaker. This does not happen constantly; instead, the code checks for every 1000th ADC value and if it has changed, the Bluetooth module turns on.

Last Week's Progress

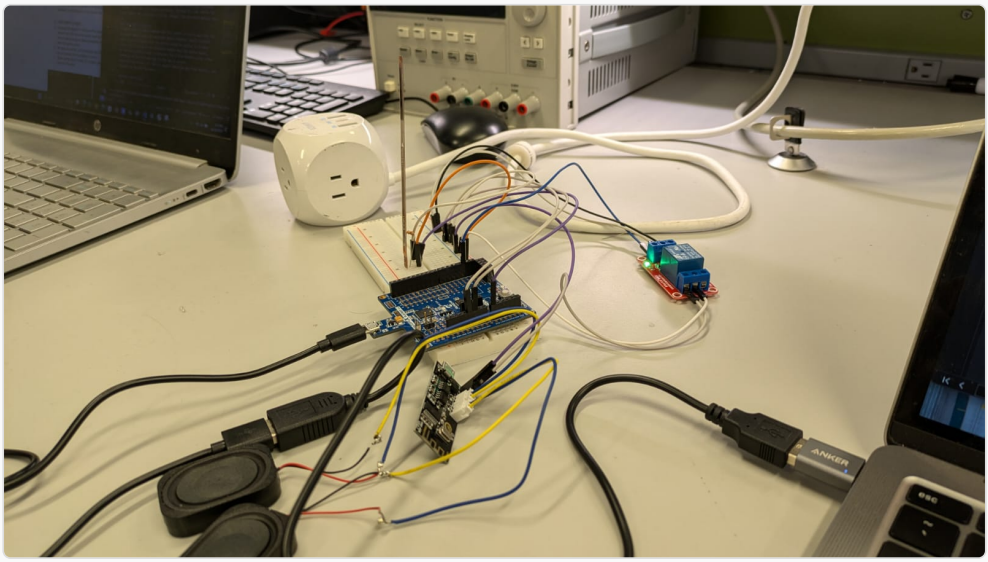
Integrated the Bluetooth module and flex sensor in such a way that the Bluetooth module turns OFF when the sensor readings remain constant (which basically means that the person using the pillow is in deep sleep) and it turns on the Bluetooth module when the person moves. We have also written code to integrate the 4 flex sensors using a MUX. Currently, we are just waiting for the MUX to arrive. Began writing drivers for the RTC module (which we got from Detkin), and figured out how the TWI works. The logic behind the code is as follows:

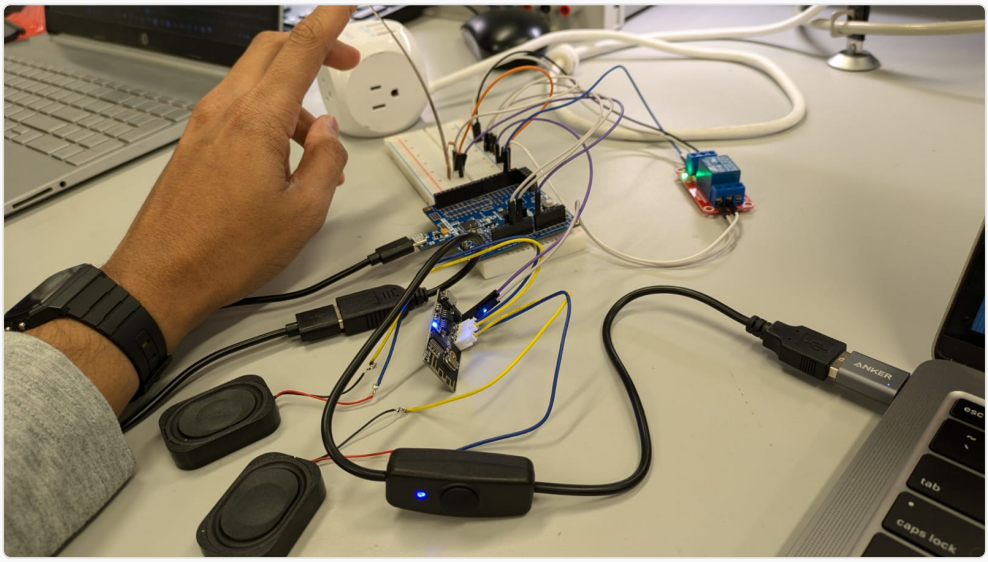
```
{
    if (counter >= 1000) {
        currentValue = sensorValue;  // Record the current sensor value
        if (abs(currentValue - previousValue) < 5) {  // Threshold for minor cha
            toggle_relay(0);  // Turn off the relay
        } else {
            toggle_relay(1);  // Turn on the relay
        }

        // Update the previous value
        previousValue = currentValue;
        counter = 0;  // Reset counter
    }
}
```

By defining a threshold for the difference of the current and previous value of the sensor, we ensure that not every small movement is considered a movement by the person.

- Notes
- Images of the Bluetooth turning ON and OFF





Next Week's Plan

The multiplexer should arrive. With this, we can use the 4 flex sensors on a single ADC channel instead of using multiple channels. We plan to code the Blynk app with the ESP32 to send over the time data obtained from the RTC module. We shall show the RTC in our code after writing the drivers and changing the manual counter in the code to time-based monitoring.

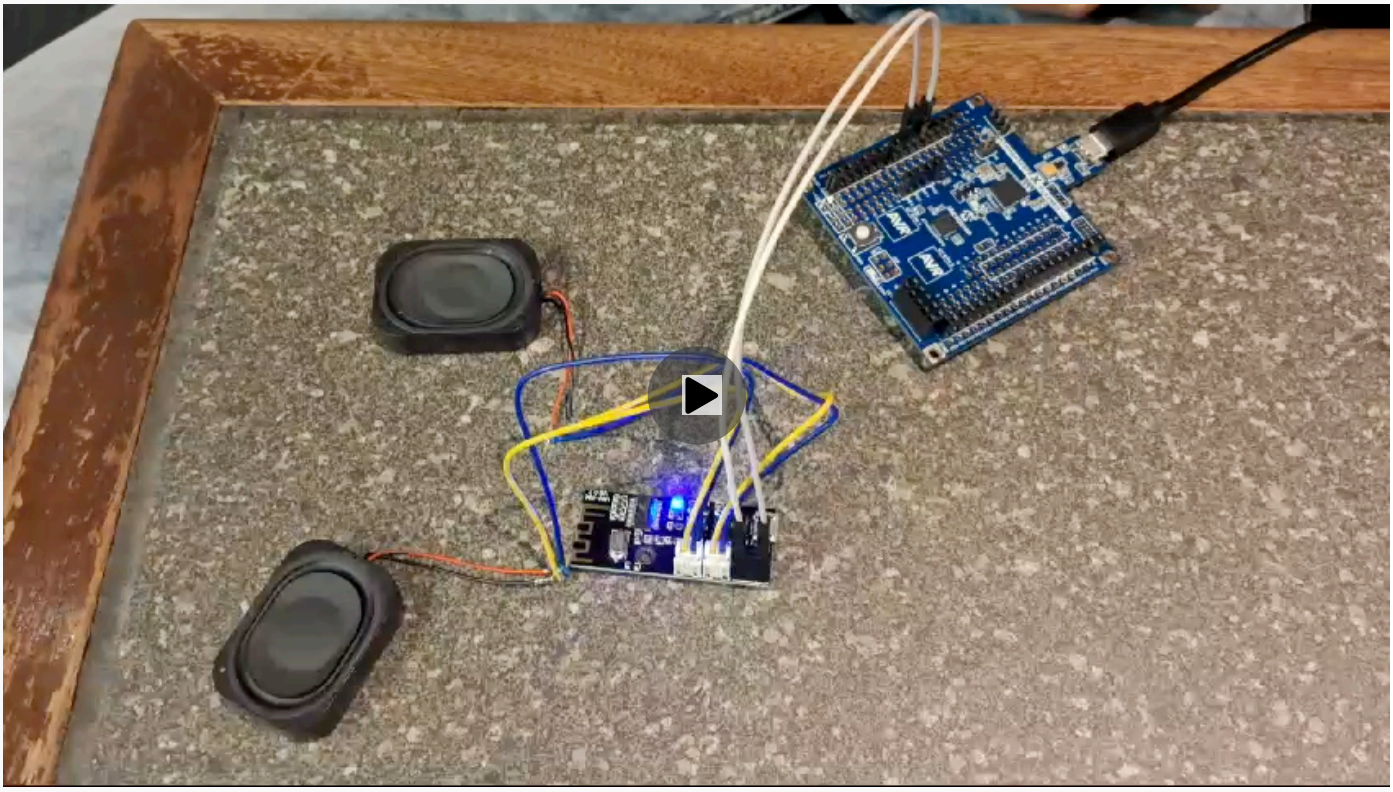
Sprint Review #1

Current State of Project

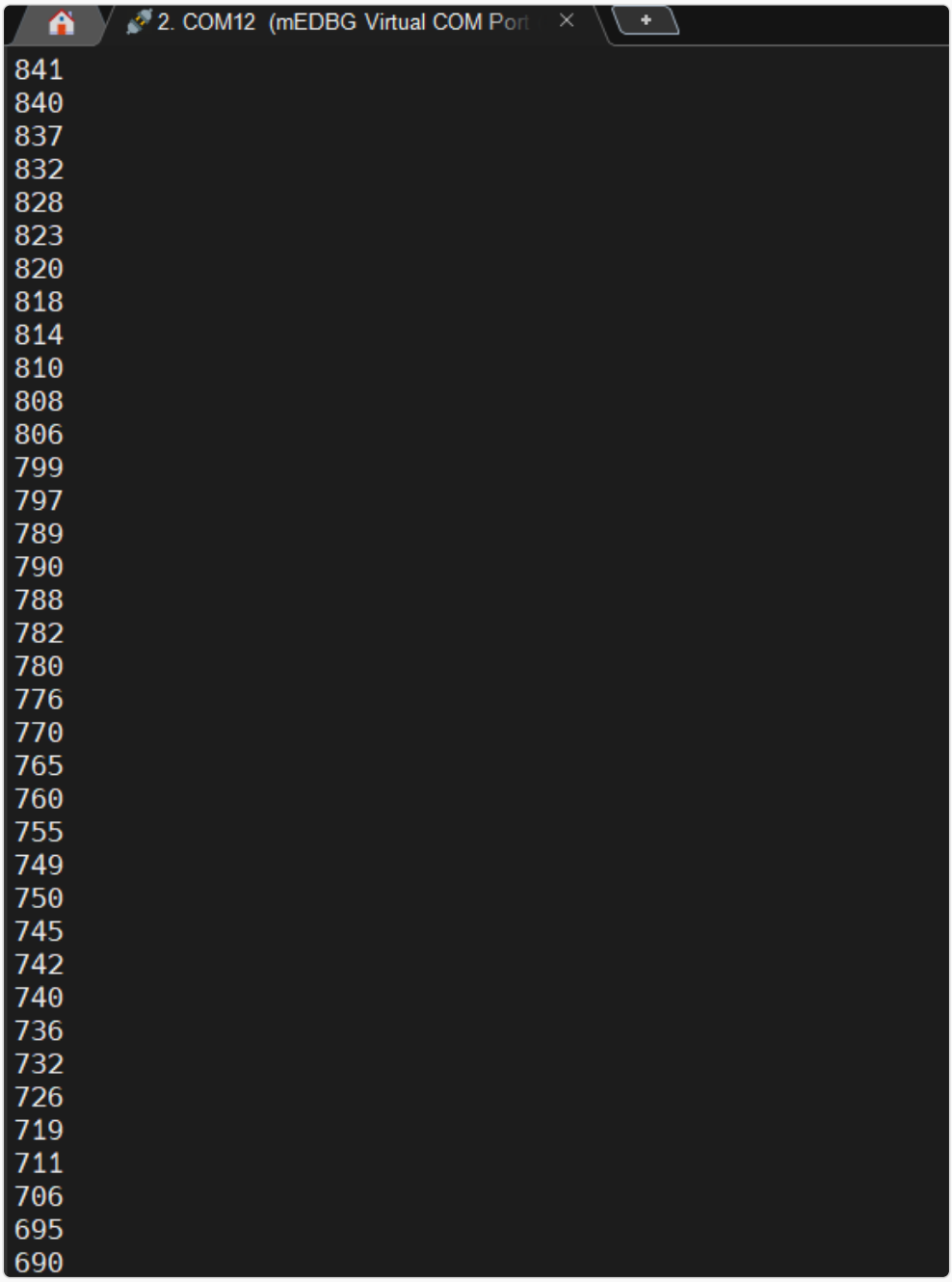
- Tested out the flex sensor from Detkin. Made a demo circuit with an LED to check if the resistance values change when the sensor bends. The sensor works well.
- Bluetooth module works along with the speakers that we took from the Detkin lab.

Last Week's Progress

- Bluetooth module arrived from Amazon. We tested it out with the speakers from Detkin. Completed the soldering for the BT module and connected it to the smartphone to see if the Bluetooth connection is established. Initially tested it with a power supply in the lab and then powered it with the ATmega.
- Video of the speaker working:

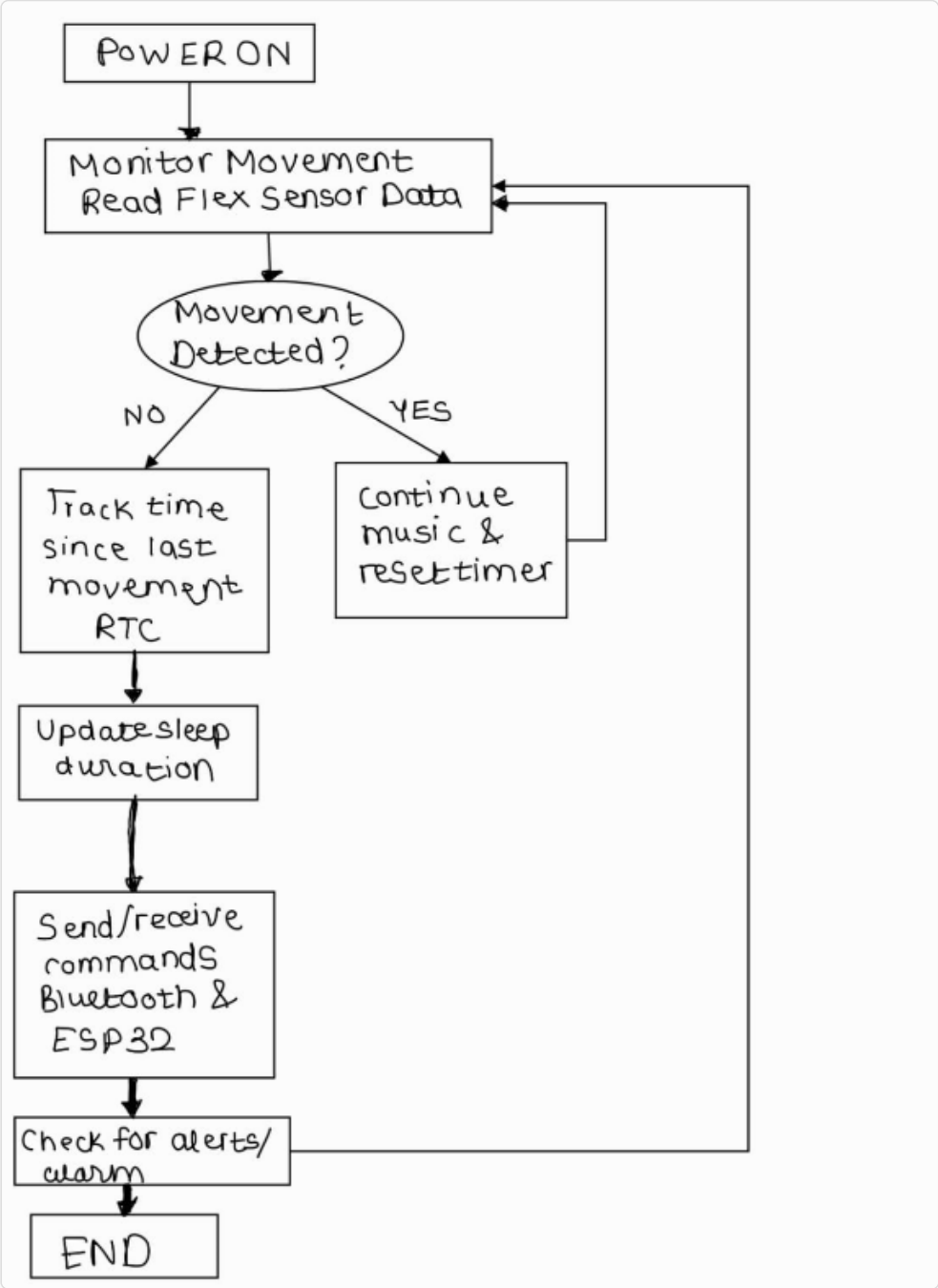


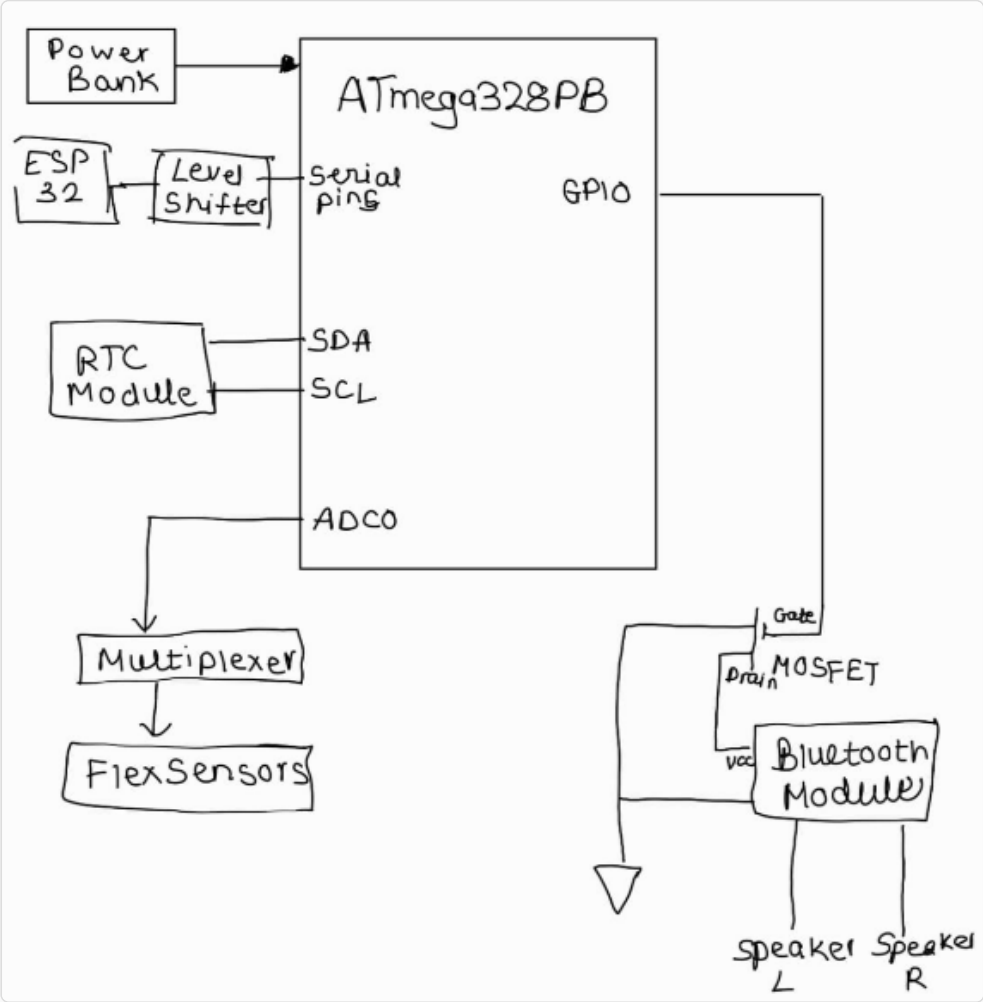
- Took the flex sensor from Detkin and tested if the resistances change as the sensor bends. Printed the ADC values on a serial monitor. Tested it with the long as well as the short sensor. Modified the code so that the sensor is not constantly reading the values and is reading at an interval of 5 seconds using a delay. We can always adjust this based on how the pillow functions. The serial monitor output is pasted below.



The C code for a singular flex sensor when attached to the LED is pasted at: FlexSensorSample.c

Since we have not received our order for the 4 flex sensors yet, we have not proceeded to interfacing the array of sensors. We plan to do this using a multiplexer and connect them all to a single ADC channel (ADC0).


POWER ON
Monitor Movement
Read Flex Sensor Data
Movement Detected?
NO
YES
Track time since last movement RTC
continue music & resettimer
Update sleep duration
Send/receive commands Bluetooth & ESP32
Check for alerts/ alarm
END



Next Week's Plan

- **Flex Sensors:** Waiting for our flex sensors to arrive so we can test out different ones and choose the perfect fit. Also, then we will be able to add the multiplexer and experiment with all 4 sensors on a single ADC channel. This will also mean writing the logic for how long the sensor needs to be unbent to know if the person is asleep.
- **Bluetooth Speakers:** Waiting for the speakers to arrive.

Sprint Review Trial

Current State of Project

- Waiting for the components to arrive.
- Started to figure out the pseudocode and the structure of the firmware.

Last Week's Progress

- Finalized project after talking to TAs, got it approved.
- Searched for components, created BOM.

- Ordered the first set of components, yet to receive.
- Got access to RPL.

Next Week's Plan

- Test the components and choose the right flex sensor.
- Order the remaining flex sensors based on the one we finalize.
- Think about component placement for the final product.

Sleep-inator: Final Project Proposal

1. Abstract

Sleep-inator is a smart pillow equipped with Bluetooth speakers and flex sensors. The goal of the project is to help people sleep and monitor their sleep to generate analytics.

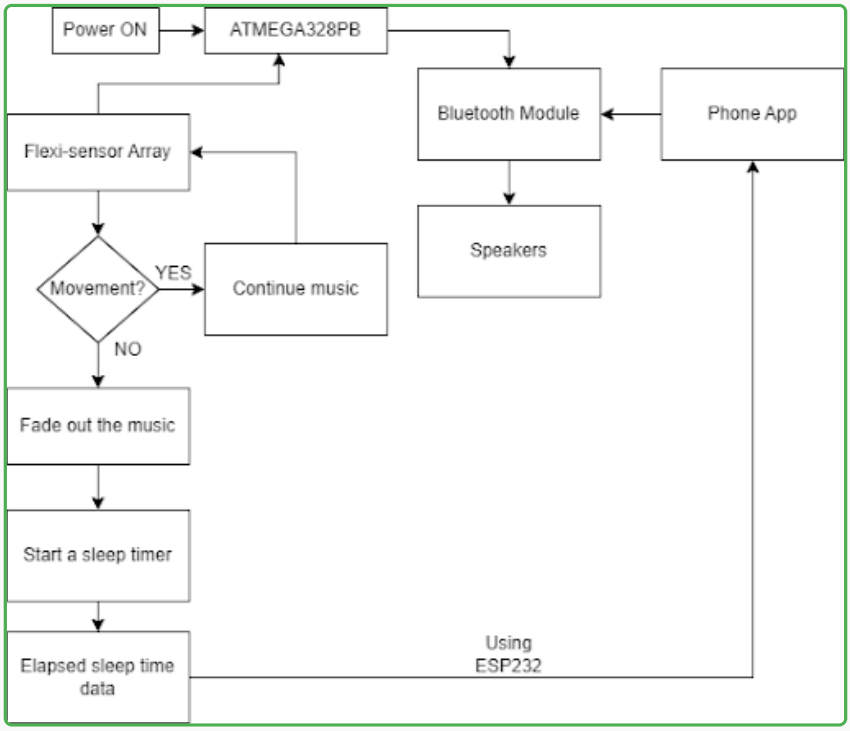
2. Motivation

In today's increasingly noisy world, finding uninterrupted sleep is harder than ever. From bustling city sounds to unexpected household noises, these disturbances can impact sleep quality and overall well-being. The project is minimalistic and provides a simple solution to help people sleep. The underlying objective of this project is to test out how many functionalities can be implemented with such a setup.

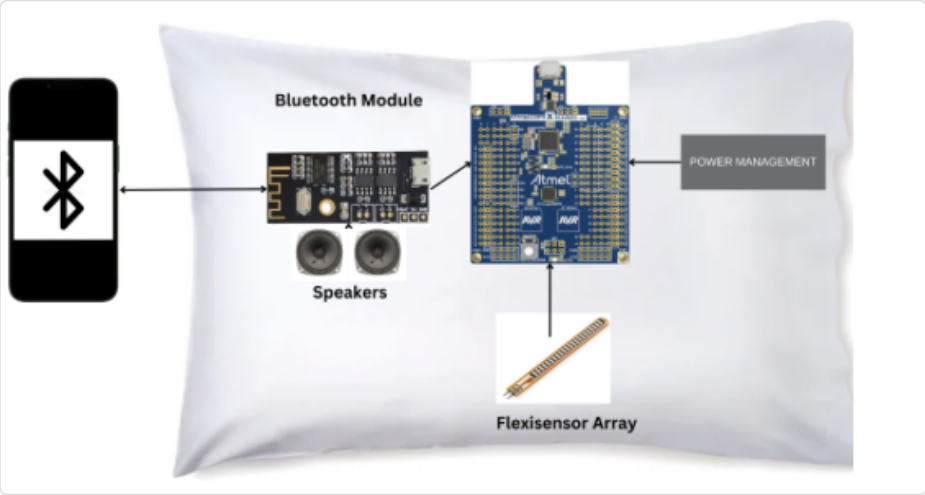
3. Goals

- Help people sleep and monitor their sleep to generate analytics using flex sensors to monitor movement.
- Transmit data over a WiFi module for further analysis.
- Communicate data from a smartphone to play music via a Bluetooth module.
- Test the extent of functionalities implementable with minimal components, leveraging interrupts, SPI communication, and power management.

4. System Block Diagram



5. Design Sketches



Overview

The ATMEGA328PB constantly monitors the sensor data and performs UART communication to send the data to an ESP32, which creates its own server to transmit data using WiFi. Concurrently, the Bluetooth module receives power from the ATMEGA, and songs are transmitted from the smartphone to the BT module via Bluetooth. The code will include multiple interrupts, including one to put the BT system into sleep mode.

Users

People who have issues sleeping and wish to fix their sleeping schedules are the target users for this project.

Definitions and Abbreviations

- MCU - Microcontroller Unit
- BT - Bluetooth

Functionality

- Arduino IDE - for the ESP32 codes
- MPLAB IDE - for the ATMEGA C codes
- Cloud analytics software - Thingsboard, Thingspeak, or a custom dashboard linked to a GitHub Pages website
- VSCode - for documentation
- GitHub and Git - for collaboration

Overview

The ATMega328PB acts as the main MCU for this project, reading sensor data from flex sensors via ADC, powering the BT speaker system, transmitting sensor data to the WiFi module, and handling tasks such as interrupt handling.

Definitions and Abbreviations

- MCU - Microcontroller Unit
- ADC - Analog to Digital Converter
- BT - Bluetooth

Functionality

- **ATMEGA328PB** - Microcontroller for power management and peripheral controls.
- **M38 Wireless Bluetooth Receiver Module** - Bluetooth module for 2-8 ohm, 3-5W speakers.
- **Speaker - 3" Diameter - 4 Ohm 3 Watt** - Compatible with the BT module.
- **Long Flex Sensor** - Measures pillow movement.
- **ESP32 Module** - Manages WiFi communication for analytics dashboard updates.
- **Multiplexers** - Scale down output channels for multiple sensors.

8. Components

- Long Flex Sensor
- ESP32 Module
- Speaker - 3" Diameter - 4 Ohm 3 Watt
- M38 Wireless Bluetooth Receiver Module
- ATMEGA328PB (provided)

9. Final Demo

1. Detect movements from the flex sensor when the user rests.
2. Play songs from the phone via Bluetooth connection.
3. Manage power efficiently, prioritizing tasks during deep sleep.
4. Show analytics hosted on a webpage.

10. Methodology

The ATMEGA328PB monitors sensor data and performs SPI communication with the ESP32, which creates a WiFi server to transmit data. Concurrently, the Bluetooth module receives power from the ATMEGA, and songs are transmitted from the smartphone. Multiple interrupts will handle tasks such as putting the BT system into sleep mode.

11. Evaluation

- Evaluate flex sensor sensitivity and optimize multi-sensor outputs.
- Ensure Bluetooth sound system control via ATMEGA.
- Collect sufficient data for analytics and prepare a working prototype ahead of submission.

References

- BT speaker idea
- Flex sensor working