Code Explanation Document

Contents

DHT Sensor

Code

https://github.com/Redback-Operations/Elderly_Wearable_Tech/tree/main/IoT_Wearable/Varinder%20Singh/Sensors%20Codes/DHT22

Working

1. The code starts by including the necessary libraries: `Wire.h` for I2C communication, `DHT.h` for the DHT sensor library, `Adafruit_GFX.h` and `Adafruit_SSD1306.h` for the OLED display.

2. In the `setup()` function:

- `Serial.begin(115200)` initializes the serial communication at a baud rate of 115200.
- `pinMode(DHTPin, INPUT)` sets the pin mode for the DHT sensor pin as an input.
- `dht.begin()` initializes the DHT sensor library.
- `display.begin(SSD1306_SWITCHCAPVCC, 0x3C)` initializes the OLED display with the specified I2C address (0x3C).
- `display.display()` displays the initial content on the OLED.
- `delay(100)` introduces a delay of 100 milliseconds.
- `display.clearDisplay()` clears the OLED display.
- `display.display()` updates the OLED display with the cleared content.
- `display.setTextSize(1.75)` sets the text size for the OLED display to 1.75.
- `display.setTextColor(WHITE)` sets the text color to white for the OLED display.

3. In the `loop()` function:

- `Humidity = dht.readHumidity()` reads the humidity value from the DHT sensor and stores it in the `Humidity` variable.
- `Temperature = dht.readTemperature()` reads the temperature value in Celsius from the DHT sensor and stores it in the `Temperature` variable.
- `Temp_Fahrenheit = dht.readTemperature(true)` reads the temperature value in Fahrenheit from the DHT sensor and stores it in the `Temp_Fahrenheit` variable.

- `if (isnan(Humidity) || isnan(Temperature) || isnan(Temp_Fahrenheit))` checks if any of the sensor readings failed. If any reading failed, it prints an error message to the serial monitor and returns to the beginning of the `loop()` function.

- The code then prints the humidity, temperature in Celsius, and temperature in Fahrenheit values to the serial monitor.

- `display.setCursor(0, 0)` sets the cursor position on the OLED display to the top- left corner.

- `display.clearDisplay()` clears the OLED display.

- `display.setTextSize(1)` sets the text size for the OLED display to 1.

- `display.setCursor(0, 0)` sets the cursor position on the OLED display to the top- left corner.

- `display.print("Temperature: ")` prints the text "Temperature: " on the OLED display.

- `display.setTextSize(2)` sets the text size for the OLED display to 2.

- `display.setCursor(0, 10)` sets the cursor position on the OLED display to the second line.

- `display.print(Temperature)` prints the temperature value in Celsius on the OLED display.

- `display.print(" ")` prints a space character on the OLED display.

- `display.setTextSize(1)` sets the text size for the OLED display to 1.

- `display.cp437(true)` enables the use of extended ASCII characters.

- `display.write(167)` prints the degree symbol (°) on the OLED display.

- `display.setTextSize(2)` sets the text size for the OLED display to 2.

- `display.print("C")` prints the letter "C" (for Celsius) on the OLED display.

- `display.setTextSize(1)` sets the text size for the OLED display to 1.

- `display.setCursor(0, 35)` sets the cursor position on the OLED display to the fourth line.

- `display.print("Humidity: ")` prints the text "Humidity: " on the OLED display.

- `display.setTextSize(2)` sets the text size for the OLED display to 2.

- `display.setCursor(0, 45)` sets the cursor position on the OLED display to the fifth line.

- `display.print(Humidity)` prints the humidity value on the OLED display.

- `display.print(" %")` prints the percentage symbol (%) on the OLED display.

- `display.display()` updates the OLED display with the new content.

- `delay(1000)` introduces a delay of 1 second (1000 milliseconds) before repeating the loop.

This code reads the temperature and humidity values from a DHT sensor and displays them on an OLED display and the serial monitor. The temperature is displayed in both Celsius and Fahrenheit units.

LM35 Sensor

Code

https://github.com/Redback-Operations/Elderly_Wearable_Tech/tree/main/IoT_Wearable/Varinder%20Singh/Sensors%20Codes

Working

1. The code starts by including a comment section with credits and additional resources related to the LM35 temperature sensor and its usage with Arduino.

2. Two variables are declared: `val` (an integer) and `tempPin` (an integer initialized with the value 1, which represents the analog pin to which the LM35 sensor is connected).

3. In the `setup()` function:

   - `Serial.begin(9600)` initializes the serial communication at a baud rate of 9600.

4. In the `loop()` function:

   - `val = analogRead(tempPin)` reads the analog value from the LM35 sensor connected to the specified `tempPin` and stores it in the `val` variable.

   - `float mv = (val / 1024.0) * 5000` converts the analog value (`val`) to millivolts (`mv`). The Arduino's analog-to-digital converter (ADC) has a resolution of 10 bits (0-1023), and the reference voltage is 5V. Therefore, the formula `(val / 1024.0) * 5000` converts the analog value to its corresponding voltage in millivolts.

   - `float cel = mv / 10` calculates the temperature in Celsius (`cel`) by dividing the millivolt value (`mv`) by 10, as the LM35 sensor has a scale factor of 10 mV/°C.

   - `float farh = (cel * 9) / 5 + 32` converts the temperature from Celsius to Fahrenheit (`farh`) using the formula `(cel * 9/5) + 32`.

   - `Serial.print("TEMPRATURE = ")` prints the string "TEMPRATURE = " to the serial monitor.

   - `Serial.print(cel)` prints the temperature value in Celsius (`cel`) to the serial monitor.

   - `Serial.print("*C")` prints the degree Celsius symbol ("°C") to the serial monitor.

   - `Serial.println()` prints a new line character to the serial monitor.

   - `delay(1000)` introduces a delay of 1 second (1000 milliseconds) before repeating the loop.

5. The commented section `/* uncomment this to get temperature in farenhite ... */` contains code that can be uncommented to print the temperature in Fahrenheit instead of Celsius. If uncommented, it will print the temperature in Fahrenheit along with the Fahrenheit symbol ("°F") to the serial monitor.

This code reads the temperature value from an LM35 sensor connected to an analog pin of the Arduino board. It converts the analog value to its corresponding voltage in millivolts and then calculates the temperature in Celsius using the LM35's scale factor. The temperature value in Celsius is printed to the serial monitor. Additionally, there is an option to uncomment a section of code to print the temperature in Fahrenheit instead of Celsius.

MAX30102 Sensor

Code

https://github.com/Redback-Operations/Elderly_Wearable_Tech/tree/main/IoT_Wearable/Varinder%20Singh/Sensors%20Codes

Working

1. The code starts by including the necessary libraries: `Wire.h` for I2C communication, `MAX30105.h` for the MAX30102 sensor library, and `heartRate.h` for the heart rate calculation library.

2. An instance of the `MAX30105` class is created with the name `particleSensor`.

3. Constants are defined: `RATE_SIZE` (set to 4) determines the number of heart rate readings to average, and an array `rates` of size `RATE_SIZE` is declared to store the heart rate values. The variable `rateSpot` is used to keep track of the current position in the `rates` array, and `lastBeat` stores the time of the last detected beat.

4. Variables `beatsPerMinute` (float) and `beatAvg` (int) are declared to store the calculated beats per minute and the average heart rate, respectively.

5. In the `setup()` function:

   - `Serial.begin(115200)` initializes the serial communication at a baud rate of 115200.

   - `if (!particleSensor.begin(Wire, I2C_SPEED_FAST))` initializes the MAX30102 sensor using the I2C communication protocol. If the sensor is not found, an error message is printed, and the program enters an infinite loop.

   - `particleSensor.setup()` configures the sensor with default settings.

   - `particleSensor.setPulseAmplitudeRed(0x0A)` sets the red LED to a low brightness level to indicate that the sensor is running.

   - `particleSensor.setPulseAmplitudeGreen(0)` turns off the green LED.

6. In the `loop()` function:

   - `long irValue = particleSensor.getIR()` reads the infrared (IR) value from the sensor.

   - `if (checkForBeat(irValue) == true)` checks if a beat is detected using the `checkForBeat` function from the `heartRate.h` library.

   - If a beat is detected, the code calculates the time elapsed since the last beat (`delta`), updates the `lastBeat` time, and calculates the current beats per minute (`beatsPerMinute`).

- If the calculated `beatsPerMinute` value is within a valid range (between 20 and 255), it is stored in the `rates` array, and the `rateSpot` is updated to point to the next position in the array.

- The average heart rate (`beatAvg`) is calculated by summing up the values in the `rates` array and dividing by `RATE_SIZE`.

- The code then prints the IR value, current beats per minute (`beatsPerMinute`), and average beats per minute (`beatAvg`) to the serial monitor.

- If the IR value is less than 50000, it prints a message indicating that no finger is detected.

This code uses the MAX30102 sensor to detect heart rate and oxygen saturation levels. It reads the infrared (IR) value from the sensor and calculates the beats per minute (BPM) based on the detected beats. The code also averages the BPM values over a specified number of readings (`RATE_SIZE`) to provide a more stable heart rate measurement. The IR value, current BPM, and average BPM are printed to the serial monitor for monitoring and debugging purposes.

Gas Sensor (MQ-2)

Code

https://github.com/Redback-Operations/Elderly_Wearable_Tech/tree/main/IoT_Wearable/Varinder%20Singh/Sensors%20Codes

Working

1. The code starts by including a comment section with credits and additional resources related to the MQ-2 gas sensor and its usage with Arduino.

2. Pin assignments are made for the red LED (`redLed` - pin 12), green LED (`greenLed` - pin 11), buzzer (`buzzer` - pin 10), and the analog input pin for the gas sensor (`smokeA0` - pin A5).

3. A threshold value (`sensorThres`) is set to 400, which will be used to determine if the gas concentration is above or below the desired level.

4. In the `setup()` function:

- `pinMode(redLed, OUTPUT)` sets the red LED pin as an output.

- `pinMode(greenLed, OUTPUT)` sets the green LED pin as an output.

- `pinMode(buzzer, OUTPUT)` sets the buzzer pin as an output.

- `pinMode(smokeA0, INPUT)` sets the gas sensor analog input pin as an input.

- `Serial.begin(9600)` initializes the serial communication at a baud rate of 9600.

5. In the `loop()` function:

- `int analogSensor = analogRead(smokeA0)` reads the analog value from the gas sensor connected to the `smokeA0` pin and stores it in the `analogSensor` variable.

- `Serial.print("Pin A0: ")` prints the string "Pin A0: " to the serial monitor.

- `Serial.println(analogSensor)` prints the value of `analogSensor` (the gas sensor reading) to the serial monitor, followed by a new line.

- `if (analogSensor > sensorThres)` checks if the gas sensor reading (`analogSensor`) is greater than the threshold value (`sensorThres`).

- If the gas sensor reading is above the threshold:

- `digitalWrite(redLed, HIGH)` turns on the red LED.

- `digitalWrite(greenLed, LOW)` turns off the green LED.

- `tone(buzzer, 1000, 200)` generates a tone on the buzzer with a frequency of 1000 Hz for 200 milliseconds.

- If the gas sensor reading is below the threshold:

- `digitalWrite(redLed, LOW)` turns off the red LED.

- `digitalWrite(greenLed, HIGH)` turns on the green LED.

- `noTone(buzzer)` stops the tone on the buzzer.

- `delay(100)` introduces a delay of 100 milliseconds before repeating the loop.

This code reads the analog value from an MQ-2 gas sensor connected to an analog input pin of the Arduino board. It compares the sensor reading with a predefined threshold value (`sensorThres`). If the sensor reading is above the threshold, it indicates the presence of gas or smoke, and the code turns on the red LED and buzzer. If the sensor reading is below the threshold, it turns on the green LED and turns off the buzzer. The sensor reading is also printed to the serial monitor for monitoring and debugging purposes.

Ambient Light Sensor

Code

https://github.com/Redback-Operations/Elderly_Wearable_Tech/tree/main/IoT_Wearable/Varinder%20Singh/Sensors%20Codes

Working

1. The code starts with a comment section that provides credit for the source of the code (DFRobot Ambient Light Sensor).

2. In the `setup()` function:

   - `Serial.begin(9600)` initializes the serial communication at a baud rate of 9600 bps (bits per second).

3. In the `loop()` function:

   - `int val` declares an integer variable `val` to store the analog value read from the sensor.

   - `val=analogRead(0)` reads the analog value from the ambient light sensor connected to the analog input pin 0 (A0) and stores it in the `val` variable.

   - `Serial.println(val, DEC)` prints the value of `val` to the serial monitor in decimal format, followed by a new line.

   - `delay(100)` introduces a delay of 100 milliseconds before repeating the loop.

This code is designed to read the analog value from an ambient light sensor connected to the analog input pin 0 (A0) of the Arduino board. The analog value represents the intensity of the ambient light detected by the sensor. The `setup()` function initializes the serial communication at a baud rate of 9600 bps, which allows the Arduino to communicate with the serial monitor or other serial devices. In the `loop()` function, the code reads the analog value from the ambient light sensor using the `analogRead(0)` function and stores it in the `val` variable. The `Serial.println(val, DEC)` statement prints the value of `val` to the serial monitor in decimal format, allowing you to monitor the ambient light intensity. The `delay(100)` function introduces a delay of 100 milliseconds (0.1 seconds) before repeating the loop. This delay is optional and can be adjusted or removed based on your specific requirements. By monitoring the serial monitor, you can observe the changes in the ambient light intensity as the sensor detects different light levels in its surroundings.

Digital Vibration Sensor

Code

https://github.com/Redback-Operations/Elderly_Wearable_Tech/tree/main/IoT_Wearable/Varinder%20Singh/Sensors%20Codes

Working

1. The code starts with a comment section that provides credit for the source of the code (DFRobot Digital Vibration Sensor).

2. Constants are defined:

   - `SensorLED` is set to 13, which corresponds to the built-in LED on most Arduino boards.

   - `SensorINPUT` is set to 3, which is the digital input pin connected to the vibration sensor.

3. A variable `state` of type `unsigned char` is declared and initialized to 0.

4. In the `setup()` function:

   - `pinMode(SensorLED, OUTPUT)` sets the `SensorLED` pin (13) as an output for the built-in LED.

   - `pinMode(SensorINPUT, INPUT)` sets the `SensorINPUT` pin (3) as an input for the vibration sensor.

   - `attachInterrupt(1, blink, FALLING)` attaches an interrupt service routine (ISR) called `blink` to interrupt 1, which corresponds to the digital pin 3 on most Arduino boards. The `FALLING` parameter specifies that the ISR will be triggered on the falling edge of the signal (when the signal transitions from HIGH to LOW).

5. In the `loop()` function:

   - `if (state != 0)` checks if the `state` variable is not equal to 0.

   - If `state` is not 0, it means that a vibration was detected, and the following actions are performed:

     - `state = 0` resets the `state` variable to 0.

     - `digitalWrite(SensorLED, HIGH)` turns on the built-in LED.

     - `delay(500)` introduces a delay of 500 milliseconds (0.5 seconds).

   - If `state` is 0, `digitalWrite(SensorLED, LOW)` turns off the built-in LED.

6. The `blink()` function is an interrupt service routine (ISR) that is called when a falling edge is detected on the `SensorINPUT` pin (3). Inside this function, `state++` increments the `state` variable, indicating that a vibration was detected.

This code is designed to detect vibrations using a digital vibration sensor connected to the digital input pin 3 of the Arduino board. When a vibration is detected, the built-in LED on the Arduino board (pin 13) will turn on for 0.5 seconds (500 milliseconds) and then turn off. The vibration sensor is connected to the `SensorINPUT` pin (3), which is configured as an input in the `setup()` function. The `attachInterrupt()` function is used to set up an interrupt service routine (ISR) called `blink` that will be triggered on the falling edge of the signal from the vibration sensor. When a vibration is detected, the `blink` ISR is called, and it increments the `state` variable. In the `loop()` function, the code checks if the `state` variable is not equal to 0, indicating that a vibration was detected. If `state` is not 0, the built-in LED is turned on for 0.5 seconds, and then the `state` variable is reset to 0. The `delay(500)` function introduces a delay of 500 milliseconds (0.5 seconds) to keep the LED on for a visible duration after a vibration is detected.

Flame Sensor

Code

https://github.com/Redback-Operations/Elderly_Wearable_Tech/tree/main/IoT_Wearable/Varinder%20Singh/Sensors%20Codes

Working

1. The code starts with a comment section that provides credit for the source of the code (Circuit Digest - Interfacing Flame Sensor with Arduino).

2. In the `setup()` function:

- `pinMode(2, INPUT)` sets digital pin 2 as an input, which is connected to the output of the flame sensor.

- `pinMode(LED_BUILTIN, OUTPUT)` sets the built-in LED pin (`LED_BUILTIN`) as an output for indicating the presence of a flame.

- `Serial.begin(9600)` initializes the serial communication at a baud rate of 9600 bps (bits per second).

3. In the `loop()` function:

- `if (digitalRead(2) == 0)` checks if the digital input on pin 2 (connected to the flame sensor) is LOW, indicating the presence of a flame.

- If the digital input is LOW (flame detected):

  - `digitalWrite(LED_BUILTIN, HIGH)` turns on the built-in LED.

  - `Serial.println("** Fire detected!!! **")` prints the message "** Fire detected!!! **" to the serial monitor.

- If the digital input is HIGH (no flame detected):

  - `digitalWrite(LED_BUILTIN, LOW)` turns off the built-in LED.

  - `Serial.println("No Fire detected")` prints the message "No Fire detected" to the serial monitor.

- `delay(100)` introduces a delay of 100 milliseconds (0.1 seconds) before repeating the loop.

This code is designed to detect the presence of a flame using a flame sensor connected to digital pin 2 of the Arduino board. The flame sensor typically outputs a LOW signal when a flame is detected and a HIGH signal when no flame is present.

In the `setup()` function, the digital pin 2 is configured as an input to read the signal from the flame sensor. The built-in LED pin (`LED_BUILTIN`) is set as an output to indicate the presence of a flame visually. Additionally, the serial communication is initialized at a baud rate of 9600 bps to display messages on the serial monitor.

In the `loop()` function, the code reads the digital input from the flame sensor using `digitalRead(2)`. If the input is LOW (0), it means a flame is detected, and the following actions are performed:

- The built-in LED is turned on using `digitalWrite(LED_BUILTIN, HIGH)`.

- The message "** Fire detected!!! **" is printed to the serial monitor using `Serial.println("** Fire detected!!! **")`.

If the digital input is HIGH (1), it means no flame is detected, and the following actions are performed:

- The built-in LED is turned off using `digitalWrite(LED_BUILTIN, LOW)`.

- The message "No Fire detected" is printed to the serial monitor using `Serial.println("No Fire detected")`.

The `delay(100)` function introduces a delay of 100 milliseconds (0.1 seconds) before repeating the loop, allowing for a brief pause between readings.

By monitoring the built-in LED and the serial monitor output, you can detect the presence of a flame and take appropriate actions based on the sensor's output.

Tilt Sensor

Code

https://github.com/Redback-Operations/Elderly_Wearable_Tech/tree/main/IoT_Wearable/Varinder%20Singh/Sensors%20Codes

Working

1. The code starts with a comment section that provides credit for the source of the code (DFRobot Tilt Sensor).

2. Variables are declared:

   - `ledPin` is set to 13, which corresponds to the built-in LED on most Arduino boards.

   - `switcher` is set to 3, which is the digital input pin connected to the tilt sensor.

3. In the `setup()` function:

   - `pinMode(ledPin, OUTPUT)` sets the `ledPin` (13) as an output for the built-in LED.

   - `pinMode(switcher, INPUT)` sets the `switcher` pin (3) as an input for the tilt sensor.

4. In the `loop()` function:

   - `if(digitalRead(switcher)==HIGH)` checks if the digital input on the `switcher` pin (3) is HIGH, indicating that the tilt sensor is triggered (tilted).

   - If the tilt sensor is triggered (HIGH):

     - `digitalWrite(ledPin, HIGH)` turns on the built-in LED.

   - If the tilt sensor is not triggered (LOW):

     - `digitalWrite(ledPin, LOW)` turns off the built-in LED.

This code is designed to detect the tilt or orientation of a tilt sensor connected to digital pin 3 of the Arduino board. The tilt sensor typically outputs a HIGH signal when it is tilted or oriented in a specific direction, and a LOW signal when it is not tilted or oriented in that direction. In the `setup()` function, the digital pin 13 is configured as an output to control the built-in LED, and the digital pin 3 is configured as an input to read the signal from the tilt sensor.

In the `loop()` function, the code reads the digital input from the tilt sensor using `digitalRead(switcher)`, where `switcher` is the variable assigned to pin 3. If the input is HIGH, it means the tilt sensor is triggered (tilted), and the built-in LED is turned on using `digitalWrite(ledPin, HIGH)`. If the input is LOW, it means the tilt sensor is not triggered, and the built-in LED is turned off using `digitalWrite(ledPin, LOW)`. By observing the state of the built-in LED, you can determine whether the tilt sensor is triggered or not. When the LED is on, it indicates that the tilt sensor is tilted or oriented in the specific direction it is designed to detect. When the LED is off, it means the tilt sensor is not triggered or oriented in that direction. This code can be useful in various applications where you need to detect the orientation or tilt of an object, such as in robotics, gaming, or motion-sensing devices.

Capacitive Touch Sensor

Code

https://github.com/Redback-Operations/Elderly_Wearable_Tech/tree/main/IoT_Wearable/Varinder%20Singh/Sensors%20Codes

Working

1. The code starts with a comment section that provides credit for the source of the code (DFRobot Capacitive Touch Sensor).

2. Variables are declared:

   - `ledPin` is set to 13, which corresponds to the built-in LED on most Arduino boards.

   - `KEY` is set to 2, which is the digital input pin connected to the capacitive touch sensor.

3. In the `setup()` function:

   - `pinMode(ledPin, OUTPUT)` sets the `ledPin` (13) as an output for the built-in LED.

   - `pinMode(KEY, INPUT)` sets the `KEY` pin (2) as an input for the capacitive touch sensor.

4. In the `loop()` function:

   - `if(digitalRead(KEY)==HIGH)` checks if the digital input on the `KEY` pin (2) is HIGH, indicating that the capacitive touch sensor is touched.

   - If the touch sensor is touched (HIGH):

     - `digitalWrite(ledPin, HIGH)` turns on the built-in LED.

   - If the touch sensor is not touched (LOW):

     - `digitalWrite(ledPin, LOW)` turns off the built-in LED.

This code is designed to detect touch input from a capacitive touch sensor connected to digital pin 2 of the Arduino board. The capacitive touch sensor typically outputs a HIGH signal when it is touched or activated, and a LOW signal when it is not touched or activated. In the `setup()` function, the digital pin 13 is configured as an output to control the built-in LED, and the digital pin 2 is configured as an input to read the signal from the capacitive touch sensor. In the `loop()` function, the code reads the digital input from the capacitive touch sensor using `digitalRead(KEY)`, where `KEY` is the variable assigned to pin 2. If the input is HIGH, it means the touch sensor is touched or activated, and the built-in LED is turned

on using `digitalWrite(ledPin, HIGH)`. If the input is LOW, it means the touch sensor is not touched or activated, and the built-in LED is turned off using `digitalWrite(ledPin, LOW)`. By observing the state of the built-in LED, you can determine whether the capacitive touch sensor is touched or not. When the LED is on, it indicates that the touch sensor is touched or activated. When the LED is off, it means the touch sensor is not touched or activated. This code can be useful in various applications where you need to detect touch input, such as in user interfaces, control panels, or interactive devices.

OLED Display Example

Code

https://github.com/Redback-Operations/Elderly_Wearable_Tech/tree/main/IoT_Wearable/Varinder%20Singh/Sensors%20Codes/Display

Working

1. The necessary libraries are included: `SPI.h`, `Wire.h`, `Adafruit_GFX.h`, and `Adafruit_SSD1306.h`.

2. Constants are defined for the OLED display's width and height (`SCREEN_WIDTH` and `SCREEN_HEIGHT`), as well as the number of snowflakes in the animation example (`NUMFLAKES`) and the dimensions of the logo bitmap (`LOGO_HEIGHT` and `LOGO_WIDTH`).

3. An instance of the `Adafruit_SSD1306` class is created with the specified display dimensions and I2C parameters.

4. The `setup()` function:

   - Initializes the serial communication.

   - Checks if the OLED display is connected correctly and initializes it.

   - Displays an initial splash screen.

   - Calls various test functions to demonstrate different drawing capabilities:

     - `testdrawline()`: Draws lines in different directions on the display.

     - `testdrawrect()`: Draws rectangles (outlines) of increasing sizes.

     - `testfillrect()`: Draws filled rectangles of increasing sizes.

     - `testdrawcircle()`: Draws circles (outlines) of increasing sizes.

     - `testfillcircle()`: Draws filled circles of decreasing sizes.

     - `testdrawroundrect()`: Draws rounded rectangles (outlines) of increasing sizes.

     - `testfillroundrect()`: Draws filled rounded rectangles of increasing sizes.

     - `testdrawtriangle()`: Draws triangles (outlines) of increasing sizes.

     - `testfilltriangle()`: Draws filled triangles of decreasing sizes.

- `testdrawchar()`: Draws characters from the default font.
- `testdrawstyles()`: Draws text with different styles (inverse, scaled).
- `testscrolltext()`: Demonstrates scrolling text in different directions.
- `testdrawbitmap()`: Draws a small bitmap image (logo).
- `testanimate()`: Animates multiple instances of the logo bitmap, simulating falling snowflakes.

5. The `loop()` function is empty, as the code runs in the `setup()` function.

This code demonstrates various drawing functions and animations that can be achieved with the Adafruit_SSD1306 library on an OLED display connected to an Arduino board. It serves as a comprehensive example and reference for working with OLED displays in Arduino projects. The code is well-structured and organized, with each test function serving a specific purpose and demonstrating a particular drawing capability. The comments throughout the code provide explanations and context for the various functions and their purposes. The `testanimate()` function is particularly interesting as it demonstrates an animation of multiple instances of the logo bitmap, simulating falling snowflakes. This function showcases the ability to create dynamic and interactive displays using the OLED library. Overall, this code serves as an excellent starting point for anyone looking to work with OLED displays in their Arduino projects, providing a solid foundation and a wide range of examples to build upon.

Buzzer

Code

https://github.com/Redback-Operations/Elderly_Wearable_Tech/tree/main/IoT_Wearable/Varinder%20Singh/Sensors%20Codes

Working

1. The code starts with a comment section that provides credit for the source (Arduino SensorKit) and mentions additional resources.

2. A constant `BUZZER` is defined and assigned the value 5, which represents the pin number to which the buzzer is connected.

3. In the `setup()` function:

   - `pinMode(BUZZER, OUTPUT)` sets the `BUZZER` pin (5) as an output, allowing it to control the buzzer.

4. In the `loop()` function:

   - `tone(BUZZER, 85)` generates a tone on the buzzer connected to the `BUZZER` pin (5). The second argument, 85, specifies the frequency of the tone in Hertz (Hz). In this case, it produces a tone with a frequency of 85 Hz.

   - `delay(1000)` introduces a delay of 1000 milliseconds (1 second), allowing the tone to be played for 1 second.

   - `noTone(BUZZER)` stops the tone on the buzzer connected to the `BUZZER` pin (5), effectively turning it off.

   - `delay(1000)` introduces another delay of 1000 milliseconds (1 second), creating a pause before the next tone.

This code is designed to control a buzzer connected to digital pin 5 of the Arduino board. It generates a tone with a frequency of 85 Hz for 1 second, followed by a 1-second pause, and then repeats this cycle indefinitely. The `tone()` function is used to generate the tone on the buzzer. It takes two arguments: the pin number to which the buzzer is connected (`BUZZER`) and the frequency of the tone in Hertz (85 Hz in this case). The `delay()` function is used to introduce a delay in milliseconds, allowing the tone to be played for a specific duration (1 second) and creating a pause between tones. The `noTone()` function is used to stop the tone on the buzzer, effectively turning it off. This code can be useful in various applications where you need to generate audible tones or alerts using a buzzer, such as in alarms, notifications, or user feedback systems.