NANYANG POLYTECHNIC
EG431D Data Acquisition

Lesson 2

Course: Diploma in Engineering (Electronic and Digital Engineering)

Module: EG431D Data Acquisition

Title: Sensors

Objective:

This lesson aims to equip learners with foundational knowledge of using an MCU as a Data Processing System and interfacing various types of sensors. By understanding sensor interface techniques, learners will be able to gather, process, and utilize sensor data effectively within embedded systems.

Learning Objectives:

- Gain familiarity with different real-world sensors and their applications.
- Describe the process of integrating analogue and digital sensors with a MCU, including sensor signal acquisition and data conversion techniques according to industry standards.
- Explain the current trends and technological advancements in sensor technologies, focusing on the role of analogue and digital sensors in modern embedded systems and IoT applications.
- Demonstrate how microcontroller platforms, such as Arduino, allow developers to interface with both analogue and digital sensors to collect, process, and interpret real-time data for a variety of applications.
- Develop practical application programs for sensor integration and data acquisition, addressing real-world problem-solving through embedded systems.

1. Overview

1.1 Sensor Basics

Sensors are sophisticated devices engineered to detect, measure, and relay information about various physical or environmental parameters, including temperature, pressure, motion, light, humidity, and chemical compositions. By capturing these parameters and converting them into data, sensors play a pivotal role in enabling systems to monitor and respond intelligently to environmental conditions. This capability is essential for applications ranging from industrial automation and environmental monitoring to healthcare, transportation, and consumer electronics, where continuous, precise data collection is critical for optimal operation.

Typically, a sensor consists of a sensing element, which interacts with the physical environment, and a transducer, which converts these interactions into electrical signals. These signals are then transformed into readable data, making them accessible for analysis, control, or immediate action. With recent advances in sensor technology—such as miniaturization, improved sensitivity, lower power consumption, and integration with wireless communication systems—sensors are now central to the Internet of Things (IoT) ecosystem, where they enable real-time, data-driven automation across a range of industries. Figure 1A depicts the various types of sensors that are available commercially.

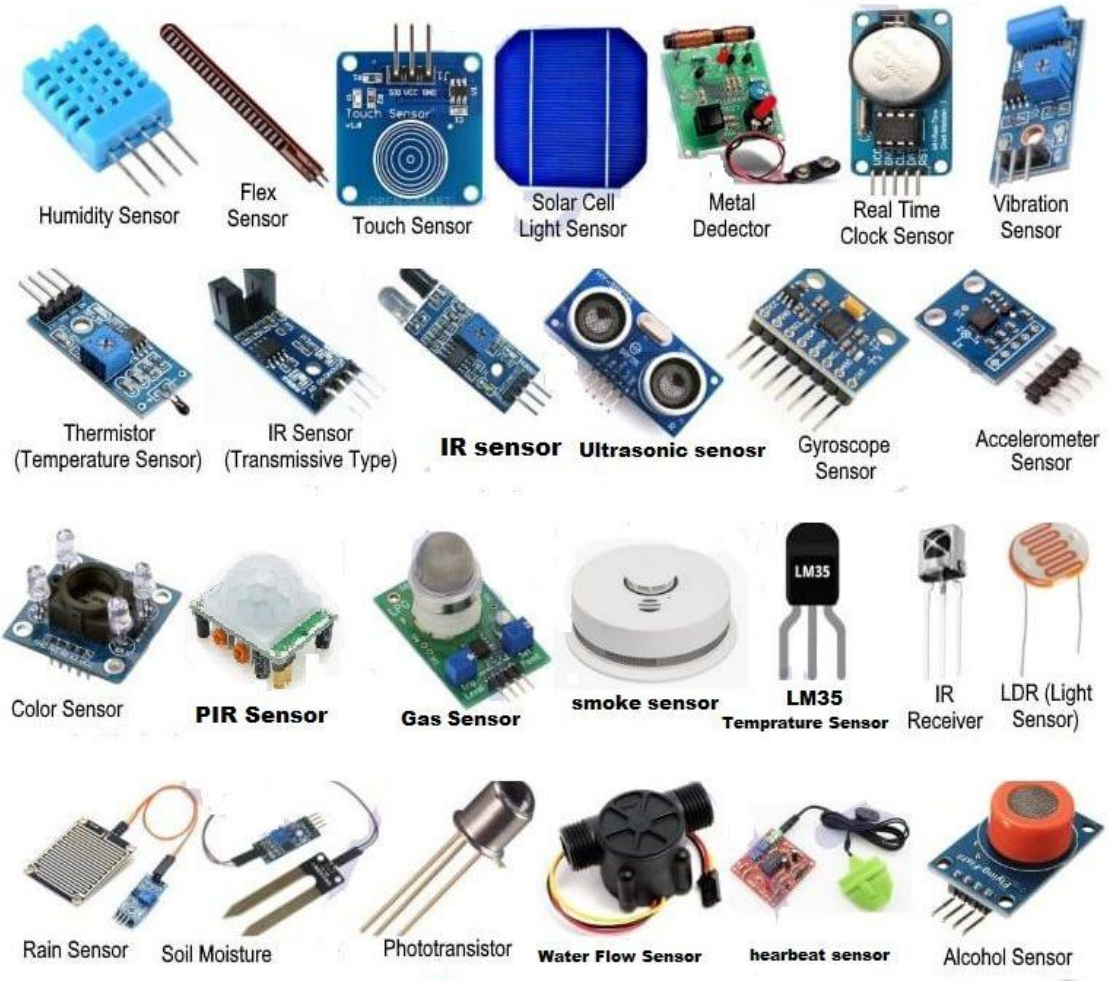


Figure 1A – Various types of Sensors

1.2 Sensor Classification

The classification of sensors is multi-faceted and provides engineers with a framework to select suitable sensor types for specific applications. Sensors can be categorized based on various criteria, including the type of measurement, energy requirements, output signal, and underlying physical principles.

For instance, by measurement type, sensors are classified into temperature, pressure, humidity, proximity, motion, and light sensors, each designed to detect a particular property. This classification aids in matching the sensor to the intended application, ensuring efficiency and accuracy.

By energy requirements, sensors are categorized as active or passive; active sensors, like ultrasonic sensors, emit energy (e.g., sound waves) to interact with the environment, while passive sensors, such as thermocouples, detect energy from external sources without emitting any.

Another classification is based on output signals: analog sensors produce continuous signals (e.g., voltage) proportional to the measurement, while digital sensors generate discrete signals, often binary, that are interpreted in binary states for ease of processing.

Finally, sensors are classified by their physical principles (Figure 1B), such as piezoelectric (e.g., pressure sensors), capacitive (e.g., touch sensors), or optical (e.g., light sensors). Each principle offers unique operational characteristics, contributing to the overall diversity of sensor applications and facilitating precise data capture in systems across fields like manufacturing, automotive safety, healthcare monitoring, and environmental surveillance. This form of classification is the most commonly used.

Type	Examples
Temperature	Thermistors, thermocouples
Pressure	Pressure gauges, barometers, ionization gauges
Optical	Photodiodes, phototransistors, infrared sensors, CCD sensors
Acoustic	Piezoelectric resonators, microphones
Mechanical	Strain gauges, tactile sensors, capacitive diaphragms, piezoresistive cells
Motion, vibration	Accelerometers, mass air flow sensors
Position	GPS, ultrasound-based sensors, infrared-based sensors, inclinometers
Electromagnetic	Hall-effect sensors, magnetometers
Chemical	pH sensors, electrochemical sensors, infrared gas sensors
Humidity	Capacitive and resistive sensors, hygrometers, MEMS-based humidity sensors
Radiation	Ionization detectors, Geiger-Mueller counters

Figure 1B – Sensors classified by Physical Principles

1.3 Sensor and MCU

When paired with a Microcontroller Unit (MCU), sensors form a powerful combination for real-time monitoring and automated control in embedded systems. In such setups, sensors collect raw data, such as temperature, light intensity, or motion levels, and relay this information to the MCU, where it undergoes processing. Acting as the system's core processor, the MCU interprets this data using programmed algorithms, converting it into meaningful information or initiating actionable commands.

For example, in an environmental monitoring system, sensors detect temperature and humidity and send this data to the MCU, which analyzes the readings and controls an HVAC system according to predefined conditions, ensuring optimal climate control. The MCU’s integrated memory and support for multiple communication protocols enable it to handle inputs from various sensors, store data, and communicate processed results to external devices or central servers.

This synergy between sensors and the MCU underpins the functionality of IoT networks and smart systems, allowing for rapid, automated responses and facilitating predictive maintenance, energy management, security, and operational efficiency across industries like home automation, industrial control, healthcare, and smart cities.

A. Sensor and MCU Application Example: Remote Health Monitoring

This diagram in Figure 1C shows an embedded system using an ATmega 328p MCU to manage and communicate with multiple sensors and modules. The MCU is connected to various devices through different communication protocols: I2C, UART, and SPI.

The OLED display and MAX30102 (a pulse oximeter and heart rate sensor) communicate with the MCU via the I2C interface, allowing real-time data display and health monitoring. The NEO-6M GPS module, which provides location data, connects through the UART protocol. Meanwhile, the SX1276 LoRaWAN transceiver, which enables long-range wireless communication, is connected through the SPI protocol.

These interfaces allow seamless data flow between the MCU and each module, making the system capable of applications like remote health monitoring with location tracking and data display.

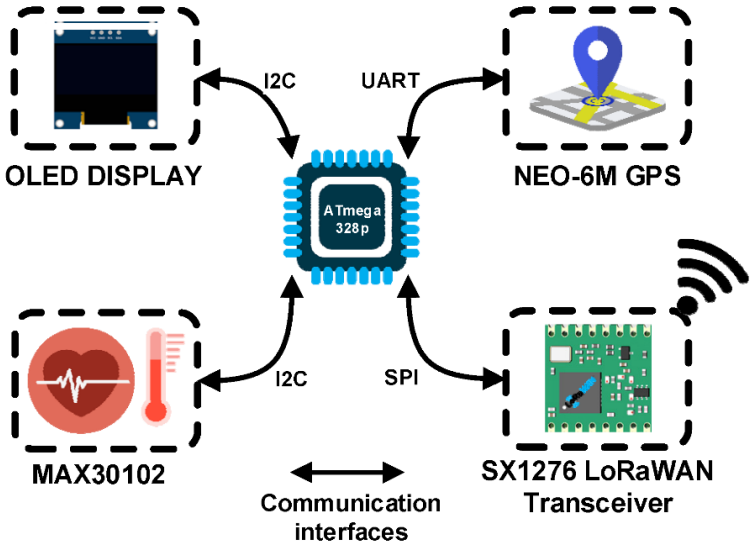


Figure 1C – Remote Health Monitoring.

B. Sensor and MCU Application Example: Real-time Temperature Monitoring

The diagram in Figure 1D illustrates the signal processing flow in a sensor-based embedded (MCU) system. It begins with a sensor detecting an environmental parameter, such as temperature, producing an analog signal.

This signal is then sent to an instrumentation amplifier, which increases the signal's amplitude to make it more suitable for further processing. The amplified signal passes through a low-pass filter, which removes high-frequency noise, ensuring that only the relevant frequency components reach the next stage.

The filtered analog signal is then fed into an Analog-to-Digital (A/D) converter, where it is digitized into an n-bit digital format at a specific sampling rate (f_s). This digital data is stored or processed in a computer or memory unit, which communicates with an embedded system or microcontroller for further analysis, control, or decision-making.

The inset graph displays a relationship between output voltage (V_{OUT}) and ambient temperature (T_A) for different temperature sensors, highlighting how temperature changes affect sensor output.

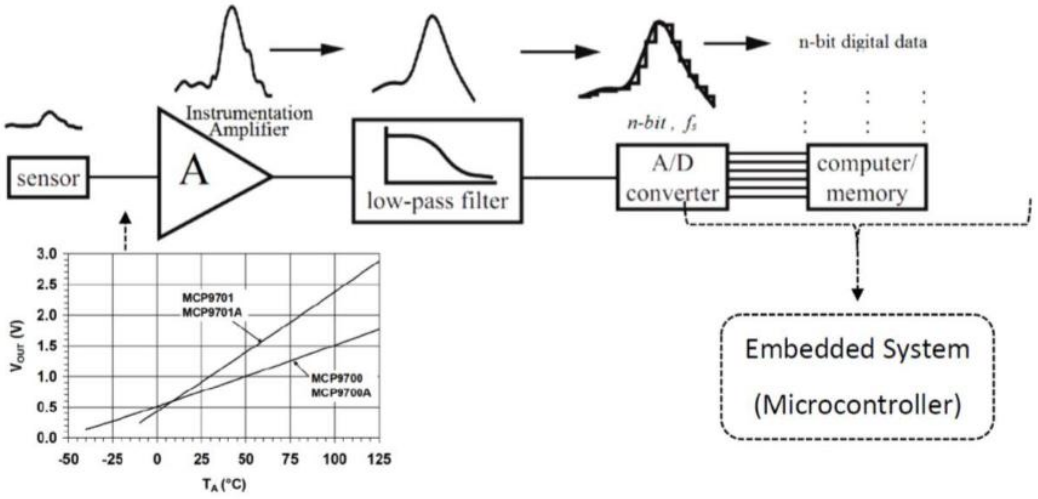


Figure 1D – Real-time Temperature Monitoring.

2. Sensor Interface

2.1 Digital Interface

The General-Purpose Input/Output (GPIO) port on an MCU serves as a critical digital interface for monitoring, controlling, and communicating with various external peripherals. MCUs are typically equipped with multiple GPIO ports, which are organized as 8, 16, or 32-bit digital parallel input/output ports, allowing them to handle multiple data lines simultaneously.

Each GPIO port can be individually configured as either an input or an output, depending on the application requirements. When configured as an input, the GPIO port reads digital signals from connected sensors, such as proximity sensors, momentary switches, or digital light sensors, and transmits this data to the MCU for processing. When set as an output, the GPIO port allows the MCU to control external devices or communicate directly with connected sensors or actuators.

This flexibility makes GPIO ports essential for tasks ranging from simple binary state monitoring to complex control applications. In addition, some MCU GPIO ports are multiplexed with alternative functions, such as USB, Serial (UART), I2C, or SPI interfaces. These alternative functions enable the MCU to communicate with more advanced or intelligent sensors and peripherals that require high-speed or specialized data transfer protocols. By leveraging multiplexed GPIO functionality, the MCU can handle multiple communication protocols efficiently, making GPIO ports indispensable for interfacing with both basic and complex external devices in embedded systems.

2.2 Analog Interface

The environment surrounding an MCU is predominantly analog, as most sensors (such as those measuring temperature, sound, or acceleration) produce analog outputs. However, for the MCU to process this information, it requires these signals in digital form. This is where the Analog-to-Digital Converter (ADC) plays a crucial role. The ADC converts analog signals into digital data, with the conversion accuracy determined by the ADC's resolution. Once the analog signal is digitized,

the MCU can process or analyze the information based on the original analog input value.

An ADC is an electronic circuit that provides a digital output directly proportional to its analog input. It measures the input voltage and delivers a binary number that represents the voltage level. The range of potential analog input signals is vast, covering sources like audio, video, medical and environmental data, and various industrial signals.

Some signals, such as temperature, change slowly, while others, like sound or vibration, vary periodically at frequencies up to tens of kilohertz. High-frequency signals, like those from video or radar, add further complexity. These varied signal characteristics have led to the development of numerous ADC types, each optimized for specific applications based on the frequency and nature of the analog signal.

Figure 2A depicts the ADC, typically functions as part of a larger system, known as a data acquisition system. On the right side of the diagram is the ADC, which has an analog input and a digital output. The ADC operates under the control of an MCU, which can initiate a conversion. Since the conversion process takes a finite amount of time, often a few microseconds or more, the ADC must signal the MCU when the conversion is complete so the data can be read. The ADC relies on a voltage reference, a stable and precise voltage source, which acts as a benchmark - similar to a ruler or measuring tape.

Essentially, the ADC compares the input voltage to this reference voltage, generating a digital output number based on this comparison. Like many digital or mixed-signal subsystems, the ADC also has a clock input, which sequences its internal processes. The clock frequency, which is subject to specific design constraints, influences the speed at which the ADC operates.

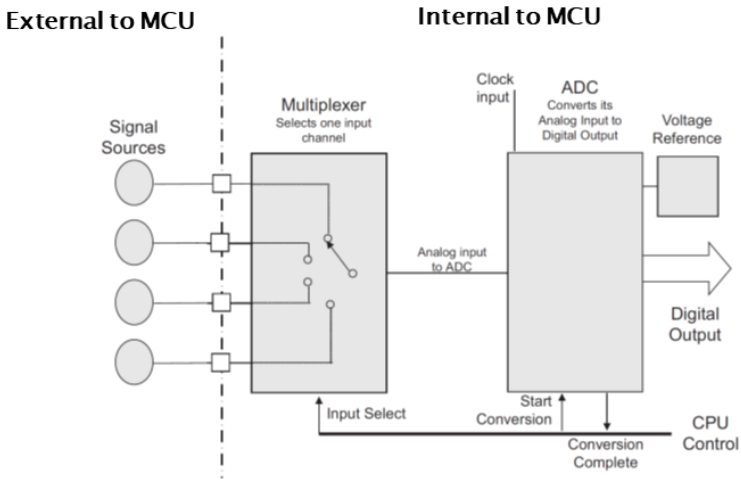


Figure 2A – 4 Channels Analog to Digital Converter.

When working with an ADC, it's often desirable to handle multiple input signals. While one option is to use multiple ADCs, this approach can be costly and consumes additional semiconductor space. A more efficient solution is to place an analog multiplexer before the ADC, which functions like a selector switch, allowing the user to choose one input at a time for conversion.

If the switching occurs fast enough, it can create the effect of simultaneous conversions for multiple inputs. Many MCUs include an integrated ADC with a built-in multiplexer, enabling multiple inputs through the MCU's pins. Figure 2A illustrates this concept with a 4-bit multiplexer. In this example, the MCU operates with 5V logic, using a 10-bit ADC with a reference voltage of 5V. The ADC produces a ratiometric output, meaning it interprets 5V as 1023 (the maximum value for a 10-bit ADC), and any voltage below 5V is represented as a proportion of this maximum value.

3. Arduino Framework

3.1 Input/Output Ports

The interface between Arduino and sensors is a key aspect of creating responsive, data-driven applications, as it enables Arduino to gather real-time information from its environment. Sensors detect various physical parameters, such as temperature, humidity, light, motion, and pressure, and convert these into electrical signals that the Arduino can read and interpret.

Arduino can interface with analog sensors via its analog input pins, which measure voltage variations, allowing it to read continuous data, such as temperature changes. Digital sensors, on the other hand, provide discrete signals and are connected to the digital input pins on the Arduino Board, making them ideal for binary or on-off measurements, such as motion detection.

For more complex sensors, Arduino supports communication protocols like I2C (Inter-Integrated Circuit) and SPI (Serial Peripheral Interface), which allow multiple devices to communicate efficiently with the Arduino over shared data lines. Additionally, UART (Universal Asynchronous Receiver-Transmitter) allows serial communication, often used for GPS modules and other serial devices.

Libraries available in the Arduino ecosystem simplify these sensor interfaces, allowing users to configure and read data from a wide range of sensors with minimal coding. This versatility in sensor interfacing enables the Arduino Board to serve as a robust platform for numerous applications, from simple hobby projects to advanced IoT systems and industrial automation.

A. Arduino UNO REV3 SMD

Figure 3A depicts an Arduino Uno Rev3 SMD, a variant of the popular and low-cost Arduino Uno board that uses an SMD (Surface-Mount Device) ATmega328 microcontroller instead of the standard DIP (Dual In-line Package) version.

It features 14 digital I/O pins (6 of which support PWM), 6 analog input pins, and a variety of power and ground pins. The board provides several power options, including a barrel jack for external power (7-12V), USB for programming and power, and a VIN pin for other external power sources.

Communication interfaces, such as UART (TX/RX), SPI (SCK, MISO, MOSI, SS), and I2C (SDA, SCL), allow easy connection to sensors, modules, and other microcontrollers. The pin layout also includes IOREF for voltage reference, AREF for analog reference, and the built-in LED on pin 13 for basic testing.

With its versatile connectivity options and compact SMD design, the Arduino Uno Rev3 SMD is ideal for prototyping, DIY projects, and educational purposes.

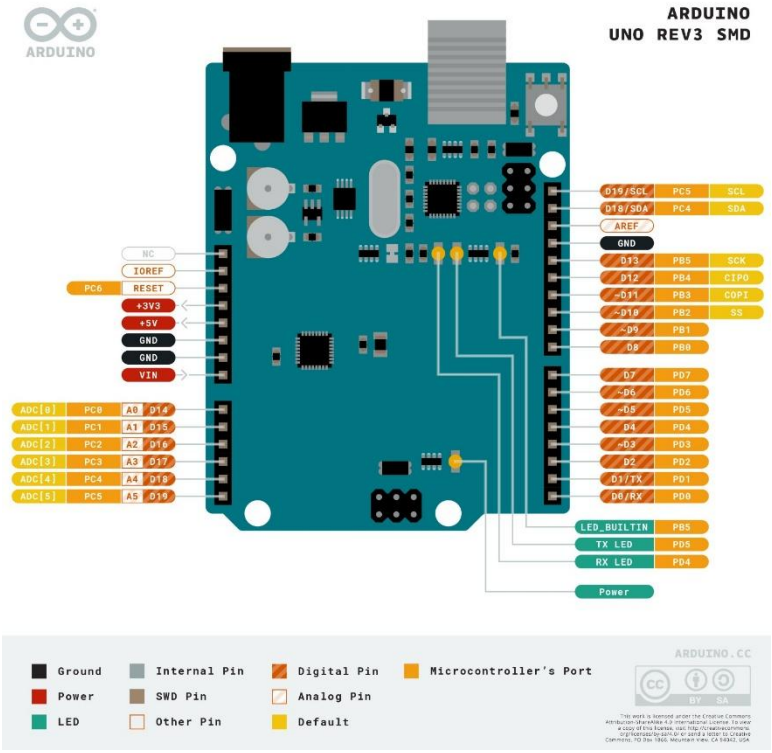


Figure 3A – Arduino UNO REV3 SMD

B. Arduino MKR WiFi 1010

Figure 3B illustrates the Arduino MKR WiFi 1010, an MCU board designed for IoT applications with built-in WiFi and Bluetooth capabilities.

The board features a range of pins and interfaces, including digital, analog, and communication pins, allowing versatile connections to sensors, actuators, and other peripherals.

It has seven analog input pins (A0-A6) and twelve digital pins (D0-D11), which support PWM (Pulse Width Modulation) on selected pins, enabling control over devices like motors and LEDs.

Communication interfaces include UART (TX/RX), I2C (SDA/SCL), and SPI (MISO, MOSI, SCK), offering flexibility for connecting various sensors and communication modules. The board supports a 3.7V Li-Po battery with an integrated charging circuit, making it suitable for portable applications. There are designated power pins (+5V, +3.3V, GND) for powering external components, and an onboard LED labeled "LED_BUILTIN" for basic output testing.

With these features, the MKR WiFi 1010 is a good platform for projects requiring wireless communication and data processing, especially in IoT and embedded systems.

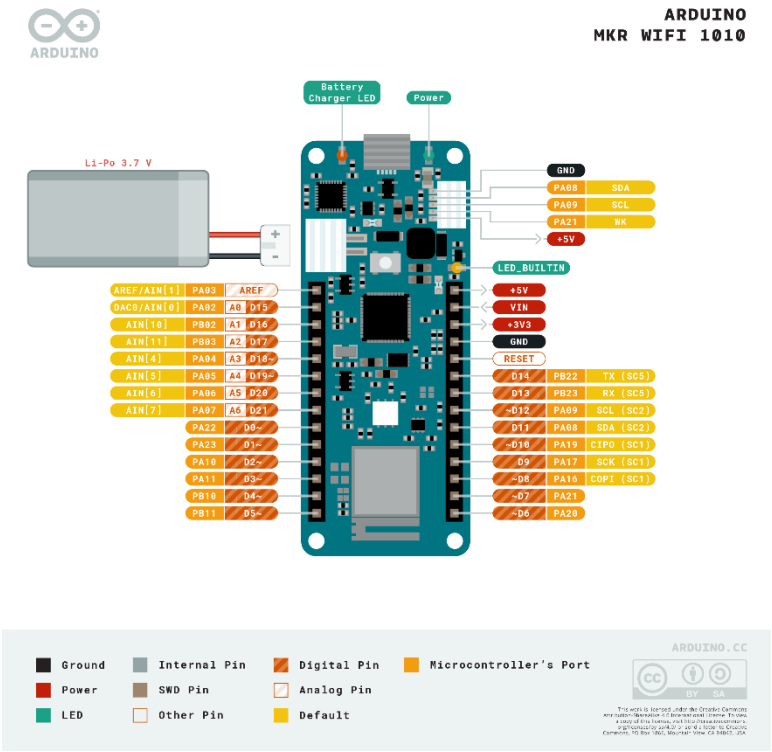


Figure 3B – Arduino MKR WiFi 1010 Board.

Figure 3C depicts an Arduino MKR WiFi 1010 board mounted on top of an MKR Connector Carrier board, which provides a convenient interface for connecting Grove-compatible sensors to the Arduino.

The Grove system consists of modular hardware components designed to simplify circuit building, allowing users to quickly set up connections without the

need for breadboards and jumper wires. This system uses a standardized connector to streamline connections, making prototyping faster and more intuitive.

This MCU hardware configuration will be used extensively throughout the practical exercise, assignments and project.

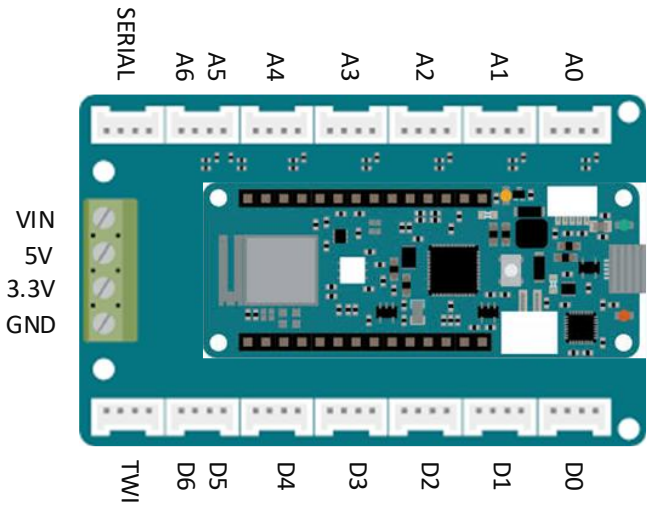


Figure 3C – Arduino MKR WiFi 1010 stacked on-top of the Arduino MKR Carrier board.

3.2 Data Logging

Arduino is a versatile and cost-effective MCU platform widely used as a data logger in scientific research, environmental monitoring, industrial processes, and other data collection applications. With its ability to interface with a broad range of sensors, Arduino can capture various environmental parameters, including temperature, humidity, pressure, light intensity, and even more specialized metrics like soil moisture and air quality.

Arduino-based data loggers are also popular due to their low power requirements, especially in battery-operated, solar-powered, or remote setups where energy efficiency is critical. With power management techniques, such as sleep modes and low-power sensors, Arduino can operate for extended periods in locations where frequent maintenance or battery replacement is not feasible.

In a typical data logging setup (Figure 3D), an Arduino is connected to sensors and a storage module, often an SD card module, where it can store data continuously or at defined intervals. The Arduino is programmed to read sensor values, format the data, and log it to the SD card, often with a timestamp for precise record-keeping. The addition of a Real-Time Clock (DS3231 RTC) module significantly enhances the setup by providing accurate timestamping, which is essential in applications where data is analyzed over time, such as tracking climate patterns or machine performance trends.

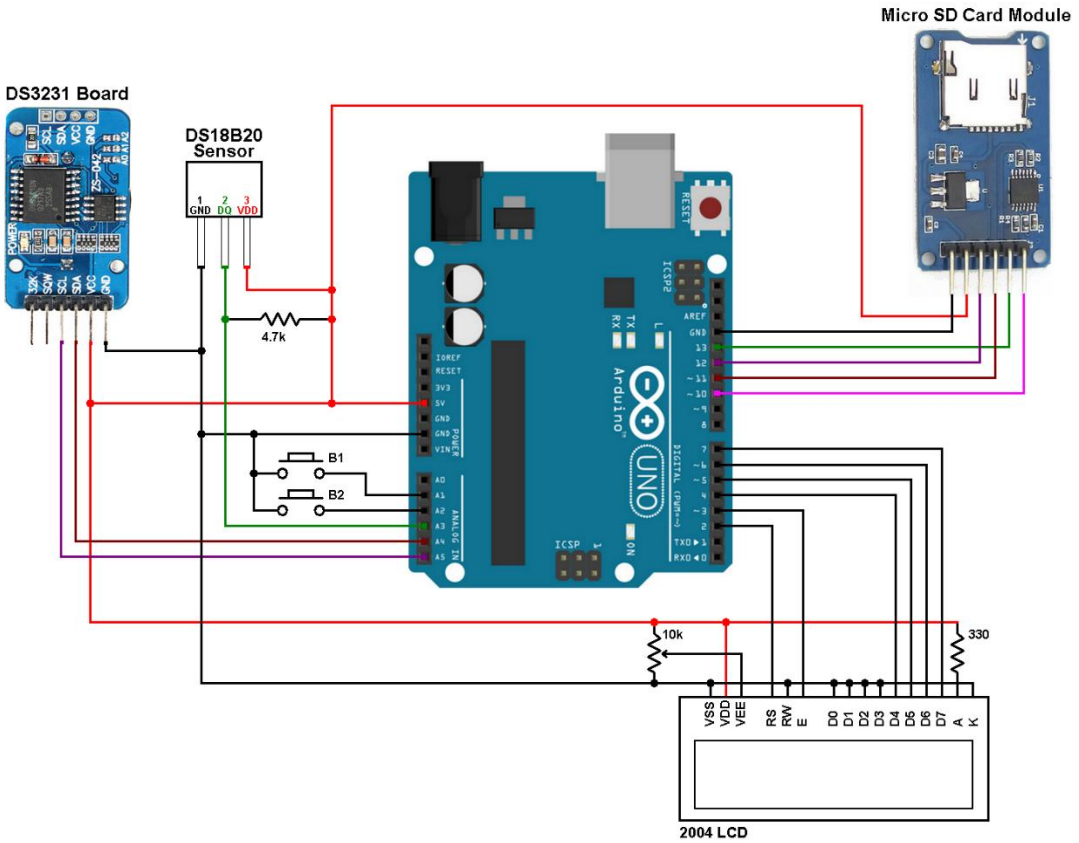


Figure 3D – An Arduino UNO Board as a Data Logger.

Arduino-based data logging is widely applicable across various fields, offering numerous examples where its capabilities are utilized to monitor, record, and analyze data over time. Here are some key examples:

A. Weather Stations: Arduino data loggers are commonly used in DIY and professional weather stations to monitor atmospheric conditions. By connecting temperature, humidity, barometric pressure, and wind speed sensors, an Arduino can log data over extended periods, helping researchers track climate patterns and weather changes. With the integration of an SD card and RTC module, data can be accurately timestamped, stored locally, or transmitted wirelessly to online platforms for real-time weather monitoring.

B. Environmental Monitoring: In environmental research, Arduino data loggers are employed to monitor natural habitats, water quality, soil moisture, and pollution levels. For instance, an Arduino can be connected to pH, turbidity, and dissolved oxygen sensors to log water quality in rivers or lakes, supporting studies on water pollution or ecosystem health. Similarly, soil moisture sensors connected to an Arduino can help track soil conditions in agricultural or conservation areas, enabling better resource management and supporting sustainable practices.

C. Industrial Equipment Monitoring: Arduino data loggers are valuable in industrial applications, where they are used to monitor the performance and health of machinery. By attaching vibration, temperature, and current sensors, Arduino can detect changes in equipment behavior, such as overheating or excessive vibration, which may indicate wear or a need for

maintenance. This data can be logged over time to perform predictive maintenance, reducing downtime and improving the efficiency of industrial operations.

D. **Health Monitoring:** Arduino data logging is also applicable in wearable and portable health devices. For instance, heart rate sensors (such as the MAX30102) and temperature sensors can be attached to an Arduino to record a user's health metrics over time. This can be used in home healthcare applications, where real-time or periodic monitoring of vital signs like heart rate, body temperature, and blood oxygen levels can support early diagnosis and better personal health management. With wireless communication modules, the data can be sent to mobile apps or healthcare providers for further analysis.

E. **Agricultural Monitoring:** In agriculture, Arduino-based data loggers help farmers monitor crops and soil health. Arduino can log soil moisture, temperature, and humidity data to provide insights into optimal irrigation and crop-growing conditions. By continuously monitoring these parameters, farmers can make data-driven decisions to improve crop yields and reduce water usage. Advanced setups may include additional sensors for sunlight and nutrient levels, enabling precision agriculture practices.

F. **Energy Consumption Tracking:** Arduino data loggers can monitor energy consumption in homes and businesses. By connecting current and voltage sensors, an Arduino can log power usage data over time. This data can help identify trends in electricity consumption, allowing for energy-efficient practices and the detection of devices consuming excessive power. Such a setup can be enhanced by integrating an Arduino with WiFi or cellular modules, enabling real-time tracking on mobile devices or web applications.

G. **Vehicle Data Logging:** Arduino is used to log data from vehicles, including speed, acceleration, engine temperature, and fuel usage. This data can be used to monitor driver behavior, detect performance issues, or support fuel efficiency improvements. Arduino-based vehicle loggers are particularly useful for fleet management, where tracking and analyzing data over time can help reduce operational costs, improve maintenance schedules, and enhance driver safety.

H. **Geological and Seismic Monitoring:** In geological studies, Arduino data loggers are used to monitor seismic activity, soil movement, and other geological changes. By connecting accelerometers, vibration sensors, and pressure sensors, an Arduino can log data on seismic events, supporting research on earthquakes, landslides, and volcanic activity. Such data can provide valuable insights into natural disasters and help develop early warning systems.

I. **Scientific Research Experiments:** Arduino data loggers are extensively used in scientific experiments where long-term data collection is necessary. For example, in biological research, Arduino can log environmental conditions in a lab, such as temperature and humidity, to study the growth patterns of bacteria or plants. Arduino's open-source nature makes it ideal for custom setups tailored to specific experimental needs, enabling scientists to collect precise and consistent data.

J. **Home Automation and Smart Home Monitoring:** Arduino data loggers are also popular in home automation, where they are used to monitor and control household conditions. For instance, an Arduino can be connected to temperature, humidity, and light sensors to track home climate data. This information can then be used to automatically adjust heating, lighting, or ventilation, improving energy efficiency and comfort. Data logging in smart homes can also monitor parameters like CO2 levels, helping to maintain healthy indoor air quality.

Each of these examples demonstrates how Arduino's flexibility and affordability make it an excellent platform for data logging across diverse applications. By combining sensors, storage modules, and communication interfaces, Arduino-based data loggers provide users with valuable insights, enabling data-driven decision-making, predictive maintenance, and improved efficiency across various fields.

This indicates Arduino as one of the suitable choices for applications requiring robust, long-term, and autonomous data collection systems. Overall, Arduino data loggers provide a highly adaptable solution for researchers, engineers, and hobbyists, offering an accessible yet powerful tool for capturing, analyzing, and managing data in real-world environments.

4. Analog Sensors

We are now prepared to explore interfacing analog sensors with the Arduino MKR WiFi 1010 Board using Arduino's streamlined approach for sensor integration. The Arduino Board is equipped with a high-resolution 12-bit Analog-to-Digital Converter (ADC), which provides enhanced precision in converting analog signals to digital form.

This 12-bit ADC allows the board to represent analog inputs with a digital resolution of 0 to 4095, offering much finer granularity in measurement compared to traditional 10-bit ADCs, which range from 0 to 1023. The Arduino Board has seven dedicated analog input pins (A0 to A6), enabling it to interface with up to seven analog sensors concurrently. Each input port can capture real-time data from connected sensors, making the Arduino Board ideal for applications requiring simultaneous monitoring of multiple environmental or physical parameters.

In this lesson, we will explore how to connect and utilize some commonly used analog sensors with the Arduino Board. Analog sensors such as angle, temperature, and sound sensors produce continuous voltage signals that reflect changes in their surroundings. These signals are then read by the ADC, which converts them into digital values for further processing by the Arduino.

The Arduino framework and its built-in analog signal acquisition functions (*analogRead()*, *analogReadResolution()*, etc.) make it easy to access and interpret these readings. Through this lesson, we will cover the setup and calibration of each sensor, examine how the Arduino processes analog inputs, and discuss ways to optimize accuracy and stability in data acquisition. This practical exploration will provide a solid foundation for understanding analog sensor integration, allowing you to harness the full capabilities of the Arduino Board for sensor-based applications.

4.1 Angle Sensor

An Angle Sensor (Figure 4A) is a rotary position sensor that utilizes a 10K Ohm potentiometer, enabling it to provide continuous, variable output for rotational control applications. It is ideal for applications that require precise adjustment, such as controlling volume levels, brightness settings, angular positioning, or regulating motor speed.

The potentiometer within the sensor functions as an adjustable resistor with three terminals: two terminals connect to the ends of a resistive element, creating a fixed resistance path, while the third terminal is connected to an adjustable wiper. As the wiper rotates along the resistive element, it alters the effective resistance between the wiper and each terminal, resulting in a proportional change in the output voltage.

This varying voltage corresponds to the angular position of the wiper, allowing the sensor to translate mechanical rotation into an analog electrical signal that can be read by MCUs or other analog-to-digital interfaces for precise control and monitoring. The sensor outputs a DC voltage ranging from 0V to 5V corresponding to the wiper position. The 12-bit ADC on the Arduino MKR WiFi 1010 Board translates the sensor output into digital word 0 (0V) to 4095 (Full Scale).

Schematic Diagram Sensor Hardware

Figure 4A – Angle Sensor with Grove Connector

The Angle Sensor's Grove Connector can connect to one of the Analog Ports A0 – A6 of the Arduino MKR WiFi 1010 Board through MKR Connector Carrier Board. In this lesson, the Sensor connects to Port A0 (Figure 4B). Listing 4A depicts a sample Arduino sketch processing the sensor output and displaying result to the Arduino IDE Serial Monitor (for text visualization).

The source codes for **all the lessons** are hosted in a GitHub repository located at:

https://github.com/NYP-SEG/EG431D.git

You can access the repository by either cloning it or downloading it as a .zip file. To clone, ensure Git is installed on your PC, then open a terminal and run the command: **git clone https://github.com/NYP-SEG/EG431D.git**, which will create a local copy of the repository. Alternatively, you can download the repository as a .zip file by visiting the link, clicking on the Code button, and selecting **Download ZIP**, then extract the file to a desired location on your computer. This will give you access to all the lesson sketches (source codes), which you can review, modify, and upload to your Arduino board as needed.

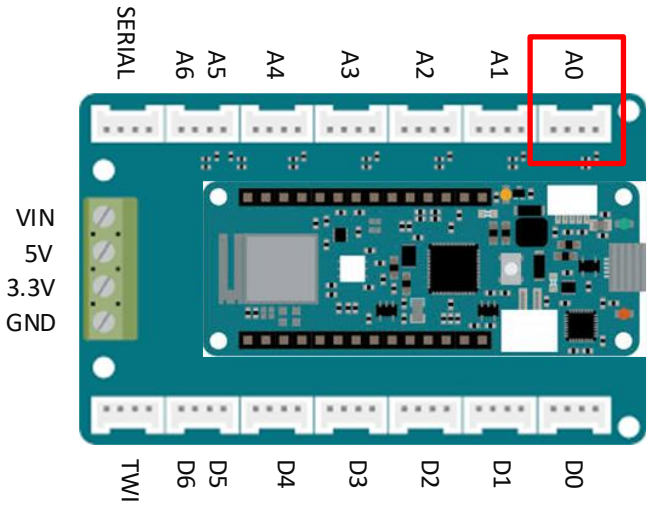


Figure 4B – Analog Port A0 on MKR Connector Carrier Board connects to the piggybacked Arduino MKR WiFi 1010 Board

The *angleSensor.ino* sketch demonstrates how to read and process data from a rotary angle sensor connected to the Arduino Board. The rotary angle sensor is connected to Port A0, with a reference voltage of 3.3V for the ADC. The setup initializes the serial port for debugging and sets the analog read resolution to 12 bits, ensuring precision for the sensor readings.

In the *loop()* function, the sensor's raw analog value is read using *analogRead()* and converted to voltage based on the ADC reference voltage and resolution. The calculated voltage is then transformed into an angle in degrees, based on the full-scale voltage (3.3V) and the full angle range of the sensor (300 degrees). The angle is displayed in the Serial Monitor along with a message explaining the angle measurement relative to the starting position. The loop runs continuously with a short delay of 100 milliseconds, providing real-time updates of the sensor's rotational position.

angleSensor.ino

```
#define ROTARY_ANGLE_SENSOR A0  // Angle Sensor connects to Arduino Analog A0 Port
#define ADC_REF       3.3       // Reference voltage of ADC is 3.3V.
#define FULL_VOLTAGE  3.3       // Full Scale Voltage of the Angle Sensor is at 3.3V (after attenuation)
#define FULL_ANGLE    300       // Full value of the rotary angle is 300 degrees

void setup()
{
    Serial.begin(9600);                 // Debug Serial Port
    analogReadResolution(12);           // Initialize Analog Resolution to 12 bits.
}

void loop()
{
    float voltage;
    int sensor_value = analogRead(ROTARY_ANGLE_SENSOR);   // Read value from Sensor
    voltage = (float)sensor_value * ADC_REF/4095;         // Converts raw ADC value to voltage
    float degrees = (voltage*FULL_ANGLE)/FULL_VOLTAGE;    // Change voltage to Angle
    Serial.println("The angle between the mark and the starting position:");
    Serial.println(degrees);
    delay(100);
}
```

Listing 4A – Sketch (source code) to read an Angle Sensor and displays angle to the Serial Monitor.

4.2 Temperature Sensor

Grove - Temperature Sensor (Figure 4C) uses an NTC Thermistor to detect the ambient temperature. The resistance of a thermistor increases when the ambient temperature decreases. It's this characteristic that we use to calculate the ambient temperature. The detectable range of this sensor is -40 - 125ºC, and the accuracy is ±1.5ºC.

The ambient temperature can be determined by solving the Steinhart–Hart equation, a mathematical equation widely used to convert the resistance of a thermistor into temperature. The Steinhart–Hart equation provides a highly accurate approximation for the temperature-resistance relationship of thermistors, especially for nonlinear temperature ranges.

The generic Steinhart–Hart equation is expressed as:

$$\frac{1}{T} = a + b \cdot ln(R) + c \cdot (ln(R))^3 \quad \textit{where:}$$

T = absolute temperature in Kelvin

R = resistance of the thermistor

a, b, c are Steinhart-Hart coefficients specific to the thermistor's characteristics

For NTC thermistors, the equation can be further simplified such that:

$$\frac{1}{T} = \frac{1}{T_0} + \frac{1}{B} \cdot ln\left(\frac{R}{R_0}\right) \quad \textit{where:}$$

T = absolute temperature in Kelvin

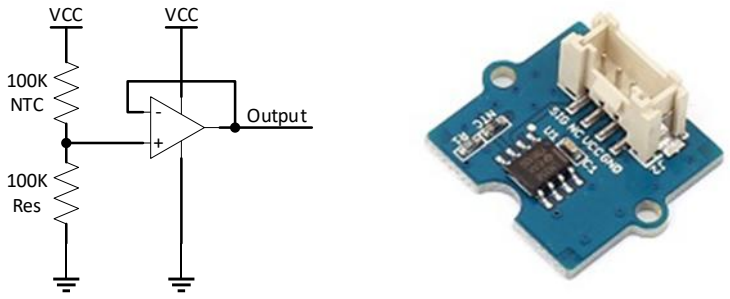
T_0 = 298.15K

B = NTC thermistor coefficient specific to the thermistor's characteristic

R = resistance of the thermistor

R_0 = resistance of the thermistor at T_0

By measuring the NTC thermistor's resistance in the circuit and applying the appropriate coefficients, we can solve the equation to obtain the highly accurate ambient temperature. This equation is particularly useful in environmental monitoring, HVAC systems, and other applications where precise temperature measurements are essential.



Schematic Diagram Sensor Hardware

Figure 4C – Grove – Temperature Sensor

In this lesson, Grove - Temperature Sensor is connected to Port A1 of the Arduino MKR WiFi 1010 Board through MKR Connector Carrier Board for data acquisition. The *tempSensor.ino* sketch (Listing 4B) demonstrates how to measure temperature using a thermistor-based Temperature Sensor connected to Port A1 of an Arduino Board and displays the results in the Arduino IDE's Serial Monitor. The thermistor's resistance varies with temperature, and this variation is converted into a temperature reading using the Steinhart-Hart equation.

The setup initializes the serial communication debugging and sets the analog read resolution to 12 bits for higher precision. The *loop()* function begins by reading the raw analog value from the sensor using *analogRead()*. This value is used to calculate the resistance (R) of the thermistor by applying the formula

$$\boldsymbol{R = R_0 \times (\frac{4095}{a} - 1)}$$, where $\boldsymbol{R_0}$ and $\boldsymbol{a}$ are

the nominal resistance of the thermistor at 25°C (100kΩ) and raw analog value from *analogRead()* respectively. The 4095 corresponds to the maximum ADC value for 12-bit resolution.

Next, the resistance is converted to temperature in degrees Celsius using a simplified Steinhart-Hart equation. The resulting temperature value is then printed to the Serial Monitor for real-time monitoring. The loop introduces a 100-millisecond delay, ensuring frequent updates while preventing excessive processing.

tempSensor.ino

```
#include <math.h>
#define TEMP_SENSOR  A1       // Grove - Temperature Sensor connects to Analog Port A1

const int B  = 4275;          // B value of the thermistor (from Datasheet)
const int R0 = 100000;        // R0 = 100k

void setup()
{
    Serial.begin(9600);
    analogReadResolution(12); // Initialize Analog Resolution to 12 bits.
}

void loop()
{
    int a = analogRead(TEMP_SENSOR);
    float R = 4095.0/a - 1.0;
    R = R0 * R;

    // converts R to temperature using simplified Steinhart-Hart equation
    float temperature = 1.0/(log(R/R0)/B + 1/298.15) - 273.15;
    Serial.print("temperature = ");
    Serial.println(temperature);
    delay(100);
}
```

Listing 4B – Sketch (source code) to measure ambient temperature and displays temperature value to the Serial Monitor.

4.3 Sound Sensor (Electret Microphone Amplifier MAX9814 with Auto Gain Control)

The Sound Sensor (Figure 4D) is designed to measure the sound pressure in the environment. The primary component of this module is an electret microphone coupled with an MAX9814 amplifier, which amplifies the microphone's output to make it easily detectable. The module outputs an analog signal corresponding to the detected sound intensity, allowing for continuous measurement of ambient noise levels. This analog output can be easily read by an Arduino Board, enabling simple sampling and real-time testing of sound levels in various applications, from noise monitoring to sound-based interaction projects.

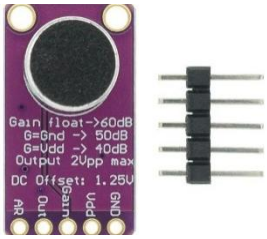


Figure 4D – Sound Sensor

In this lesson, the Sound Sensor is connected to Port A2 for data acquisition. The *soundSensor.ino* sketch (Listing 4C) demonstrates how to measure and analyze sound levels using the Sound Sensor connected to the Arduino Board's Port A2 and displays the results in the Arduino IDE's Serial Plotter (for graphical visualization). The sketch reads sound intensity values and calculates an average over a set of samples, providing a stable representation of the environmental noise level.

The setup initializes the serial communication at 115200 baud for high-speed data transmission to the Serial Plotter. The analog read resolution is set to 12 bits to improve the precision of the sensor readings, ensuring more accurate sound intensity measurements.

The *loop()* function begins by initializing a variable *sum* to accumulate the sound sensor readings. A *for* loop is used to collect 32 sound samples by repeatedly calling *analogRead()* on the *SOUND_SENSOR* pin. These individual readings are added together to calculate the total sound intensity over the 32 samples. After the loop completes, the *sum* is divided by 32 to compute the *average sound level*, providing a smoothed value that reduces noise and variability in the sensor's output. The calculated average is then transmitted to the Serial Plotter for real-time observation. A short delay of 1 millisecond is introduced before the loop repeats to allow continuous monitoring.

soundSensor.ino

```
#define SOUND_SENSOR A2          // Sound Sensor connects to Arduino Analog A2 Port

void setup()
{
    Serial.begin(115200);
    analogReadResolution(12);    // Initialize Analog Resolution to 12 bits.
}

void loop()
{
    long sum = 0;

    // Read 32 sound sound samples
    for(int i=0; i < 32; i++)
    {
        sum += analogRead(SOUND_SENSOR);
    }

    // Calculate an average sound from the 32 sound samples.
    sum = sum / 32;
    Serial.println(sum);
    delay(1);
}
```

Listing 4C – Sketch (source code) to measure sound intensity and displays sound intensity to the Serial Plotter.

4.4 Water Sensor

Grove - Water Sensor (Figure 4E) is a versatile module designed to detect the presence and level of water, making it suitable for a range of applications, from leak detection to fluid level monitoring. The sensor operates by measuring the conductivity between exposed traces on its surface, which changes when water or another conductive liquid makes contact. When water is detected, the sensor produces an analog output proportional to the water level or contact area, enabling it to distinguish between low, medium, and high levels of liquid. The analog signal can be read by an Arduino Board with an analog input, allowing for straightforward integration.

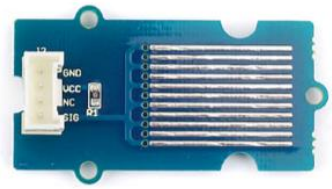


Figure 4E – Grove – Water Sensor

In this lesson, Grove - Water Sensor is connected to Port A3 for data acquisition. Listing 4D shows a sample Arduino sketch (waterSensor.ino) that reads the sensor's analog output, processes the data to calculate the conductivity between the exposed traces, and displays the results in the Arduino IDE's Serial Plotter. This setup enables users to observe real-time changes in conductivity, providing a visual representation of the sensor's readings and helping to identify the presence and level of water. By plotting the data, users can better understand variations in conductivity over time, enhancing the monitoring capabilities of applications involving water detection and fluid level tracking.

The *waterSensor.ino* sketch (Listing 4D) demonstrates how to measure water conductivity and provides an average value for enhanced reliability and accuracy. In the setup, the sketch initializes the serial communication at 115200 baud for monitoring purposes. It also sets the analog read resolution to 12 bits to increase the precision of the sensor readings, ensuring better measurement sensitivity for water conductivity levels.

In the *loop()* function, the sketch begins by initializing a variable *sum* to accumulate conductivity readings. A *for* loop collects 16 samples from the *WATER_SENSOR* pin using *analogRead()* and adds each value to the *sum* variable. This sampling process helps smooth out any fluctuations or noise in the sensor's output. After all 16 samples are collected, the total is divided by 16 to calculate the *average conductivity* value. This average is then transmitted to the Serial Plotter, providing a real-time display of the sensor's readings. A short delay of 1 millisecond is included at the end of the loop to allow continuous monitoring without excessive processing.

waterSensor.ino

```
#define WATER_SENSOR A3         // Grove - Water Sensor connects to Arduino Analog A3 Port

void setup()
{
    Serial.begin(115200);
    analogReadResolution(12);    // Initialize Analog Resolution to 12 bits.
}

void loop()
{
    long sum = 0;

    // Read 16 measured conductivity samples
    for(int i=0; i < 16; i++)
    {
        sum += analogRead(WATER_SENSOR);
    }

    // Calculate an average sound from the 16 conductivity samples.
    sum = sum / 16;
    Serial.println(sum);
    delay(1);
}
```

Listing 4D – Sample sketch (source code) to measure conductivity and displays conductivity value to the Serial Plotter.

5. Digital Sensors

Digital sensors interface with the Arduino MKR WiFi 1010 board by transmitting data in digital form, either as binary states or through serial communication protocols. These sensors typically connect to the digital input/output ports or designated communication ports on the board.

The Arduino MKR WiFi 1010 supports several digital communication protocols, including I2C, SPI, and UART, which are commonly used for interfacing with more complex digital sensors, such as temperature sensors, accelerometers, or GPS modules.

For instance, I2C allows multiple sensors to share the same two-wire bus (SDA and SCL), making it ideal for setups requiring multiple sensors. SPI offers faster communication speeds and is suitable for sensors requiring high-speed data transfer, while UART enables serial communication, often used with GPS or wireless modules.

Arduino's libraries simplify the integration of these protocols, allowing users to easily initialize and read data from digital sensors, facilitating tasks such as environmental monitoring, motion detection, and navigation.

5.1 Tact Switch and PIR Motion Sensor

Grove – Button (Figure 5A) is a mechanical momentary tact switch, functioning as a single-bit digital sensor. When pressed, the switch outputs a logic HIGH or "1" signal, and when released, it outputs a logic LOW or "0." This simple digital output can be easily read by an Arduino board through either an Analog or Digital input port, providing seamless integration into various projects. The Grove – Button's straightforward design makes it ideal for applications requiring a basic user interface or control input, such as triggering actions, toggling states, or initiating commands in DIY electronics, prototypes, or interactive devices.

Figure 5A – Grove – Button

Passive Infrared (PIR) Motion Sensor (Figure 5B) is a single-bit digital sensor designed for motion detection, primarily used to identify when a human has moved within or out of its range. PIR sensors detect changes in infrared radiation within their field of view, making them a popular choice for applications that require human presence detection, such as security systems, automatic lighting, and home appliances. They are compact, affordable, energy-efficient, reliable, and require minimal maintenance, which has led to their widespread use in both household and commercial settings.

PIR sensors, also known as "Passive Infrared," "Pyroelectric," or "IR motion" sensors, are favored for projects where human motion detection is essential. Despite their versatility, PIR sensors do have limitations: they cannot determine the number of people present, their exact distance from the sensor, or differentiate small pets from larger human movements. Their fixed lens typically limits them to specific distances and angles, but overall, PIR sensors are an ideal, cost-effective solution for general motion detection needs.

Like the Grove – Button, PIR Sensor also outputs a digital signal. When motion is detected within its sensing range, the PIR sensor generates a logic HIGH or "1" signal. In the absence of motion, it outputs a logic LOW or "0." This simple binary output makes it easy to interface with an Arduino Board, enabling straightforward integration for applications that require motion detection. The logic-based output can be used to trigger various actions, such as activating alarms, lighting systems,

or starting recordings in security applications, making the PIR sensor a versatile choice for motion-based projects.

Figure 5B – PIR Motion Sensor

In this lesson, Both Grove – Button and PIR Motion Sensor are connected to Port D0 and D1 respectively on the Arduino MKR WiFi 1010 Board through MKR Connector Carrier Board. An audible output device Grove – Buzzer (Figure 5C) connects to Port D2. The buzzer is not a sensor and is used as an audible indicator as part of the Button and PIR Motion Sensor operational requirement.

Figure 5C – Grove – Buzzer

The *buttonPIRSensor.ino* sketch (Listing 5A) integrates a Button, a PIR Motion Sensor, and a Buzzer to create an interactive Arduino-based monitoring system. The system's primary function is to trigger the Buzzer when both the Button is pressed and the PIR sensor detects motion. This provides a simple but effective demonstration of how digital sensors and actuators can work together to create responsive systems.

In the *setup()* function, the Arduino initializes the serial communication at a baud rate of 115200, allowing for real-time debugging and monitoring through the Serial Monitor. The Button and PIR sensor are configured as digital input devices using *pinMode()*, enabling the Arduino Board to read their states. The Buzzer is configured as a digital output device, which allows the Arduino to control whether the Buzzer is on or off based on specific input conditions.

In the *loop()* function, the system continuously reads the states of the Button and PIR sensor using the *digitalRead()* function. If both the Button and the PIR sensor are active (*logic HIGH*), the Buzzer is turned on by setting its output state to *HIGH*. If either the Button or the PIR sensor is inactive (*logic LOW*), the Buzzer is turned off by setting its output to *LOW*. This logic ensures that the Buzzer only activates when both conditions are met, adding an additional layer of interactivity or security to the system. The Button and PIR sensor states are printed to the Serial Monitor for debugging, formatted as tab-separated values to improve readability. A short delay of 100 milliseconds is added at the end of each loop to ensure stable and periodic updates.

buttonPIRSensor.ino

```
#define BUTTON 0              // Grove - Button connects to Arduino Digital D0 Port
#define PIR    1              // PIR Motion Sensor connects to Arduino Digital D1 Port
#define BUZZER 2              // Grove - Buzzer connects to Arduino Digital D2 Port

void setup()
{
    Serial.begin(115200);
    pinMode(BUTTON, INPUT);   // Set BUTTON (D0) as digital input
    pinMode(PIR, INPUT);      // Set PIR (D1) as digital input
    pinMode(BUZZER, OUTPUT);  // Set BUZZER (D2) as digital output
}

void loop()
{
    bool _button = digitalRead(BUTTON); // Read BUTTON (D0)
    bool _pir    = digitalRead(PIR);    // Read PIR (D1)
    if(_button == true && _pir == true) // Check if both BUTTON and PIR are asserted
    {
      digitalWrite(BUZZER, HIGH);       // If asserted, BUZZER (D2) is set to HIGH
    }
    else
    {
      digitalWrite(BUZZER, LOW);        // BUZZER (D2) is set to LOW if not asserted
    }
    Serial.print("Button = ");
    Serial.print(_button);
    Serial.print("\t");
    Serial.print("PIR = ");
    Serial.println(_pir);
    delay(100);
}
```

Listing 5A – Sample sketch (source code) to check for Grove – Button and PIR Motion Sensor activities, output to a Grove – Buzzer and to the Serial Monitor.

5.2 Ultrasonic Distance Sensor

Grove - Ultrasonic Ranger (Figure 5D) is a non-contact distance measurement sensor operating at a frequency of 40KHz. It works by emitting an ultrasonic pulse from its transmitter, then calculating the time it takes for the echo to return after reflecting off a distant object. The module's two front openings serve distinct functions: one is the transmitter (similar to a small speaker), which emits the ultrasonic waves, while the other acts as a receiver (like a microphone) to detect the returning echo.

Knowing the speed of sound in air—approximately 341 meters per second—the sensor can calculate the object's distance based on the interval between transmitting and receiving the pulse. This makes the ultrasonic sensor particularly suited for automatic distance measurement in challenging environments, where optical sensors may be unreliable due to smoke, dust, or other visual obstructions.

The ultrasonic sensor is precise, stable, and effective over a wide range, making it a versatile tool for measuring various parameters without physical contact with the medium. It can accurately measure (a) distance, (b) liquid or material levels, (c) object diameter, (d) the presence of objects, and (e) object position. This versatility makes the sensor ideal for use in industrial automation, robotics, environmental

monitoring, and any application requiring reliable, non-contact distance sensing across a range of environmental conditions.

In this lesson, Grove - Ultrasonic Ranger is connected to Digital Port D3 on the Arduino MKR WiFi 1010 Board via the MKR Connector Carrier Board. To use the sensor, a custom library is required, which can be downloaded from:

https://github.com/Seeed-Studio/Seeed_Arduino_UltrasonicRanger/archive/master.zip.

After downloading, extract the contents of the ZIP file and place the sensor's library folder in the **libraries** folder within the current working directory (e.g., EG431D/libraries). Once installed, restart the Arduino IDE, enabling access to the necessary functions to interface with the sensor seamlessly.

Detailed information on the ultrasonic sensor can be found at:

https://wiki.seeedstudio.com/Grove-Ultrasonic_Ranger/

Figure 5D – Grove – Ultrasonic Ranger

Listing 5B provides a sample Arduino sketch (*distanceSensor.ino*) that reads the sensor's output, calculates the measured distance, and displays the results in the Serial Monitor of the Arduino IDE.

The *distanceSensor.ino* sketch demonstrates how to use the Ultrasonic Ranger module with an Arduino Board to measure distances to obstacles. The sensor is connected to Port D3, and the sketch calculates and displays the distance in both inches and centimeters using the ultrasonic sensor's built-in functions.

In the *setup()* function, the serial communication is initialized at a baud rate of 115200, allowing for real-time distance measurements to be printed to the Serial Monitor. The Ultrasonic Ranger is initialized to prepare for distance calculations.

In the *loop()* function, the distance to an obstacle is measured in two units: inches and centimeters. First, *ultrasonic.MeasureInInches()* is called to calculate the distance in inches. The value is then printed to the Serial Monitor, followed by a short delay of 250 milliseconds. Next, the distance is measured in centimeters using *ultrasonic.MeasureInCentimeters()*, and the result is also printed to the Serial Monitor. This alternating measurement approach ensures both units are captured and displayed with a brief interval between readings.

distanceSensor.ino

```
#include "Ultrasonic.h"
#define ULTRASONIC_RANGER 3              // Grove - Ultrasonic Ranger connects to Arduino D3 Port

Ultrasonic ultrasonic(ULTRASONIC_RANGER); // Initialize Grove - Ultrasonic Ranger
void setup()
{
  Serial.begin(115200);                  // Initialize Serial Port
}

void loop()
{
  long RangeInInches;
  long RangeInCentimeters;
  Serial.println("The distance to a target obstacle: ");
  RangeInInches = ultrasonic.MeasureInInches();
  Serial.print(RangeInInches);      // 0~157 inches
  Serial.println(" inch");
  delay(250);

  // keep an interval between two measurements
  RangeInCentimeters = ultrasonic.MeasureInCentimeters();
  Serial.print(RangeInCentimeters); // 0~400cm
  Serial.println(" cm");
  delay(250);
}
```

Listing 5B – Sketch (source code) to measure distance using Grove – Ultrasonic Ranger and displays distance value to the Serial Monitor.

5.3 Digital Light Sensor

Grove - Digital Light Sensor (Figure 5E) is a precise, compact module designed to measure ambient light intensity. It is based on the TSL2561 I2C light-to-digital converter, which converts detected light into digital data, making it less susceptible to electrical noise compared to analog sensors. This feature allows for more stable readings in environments with fluctuating conditions.

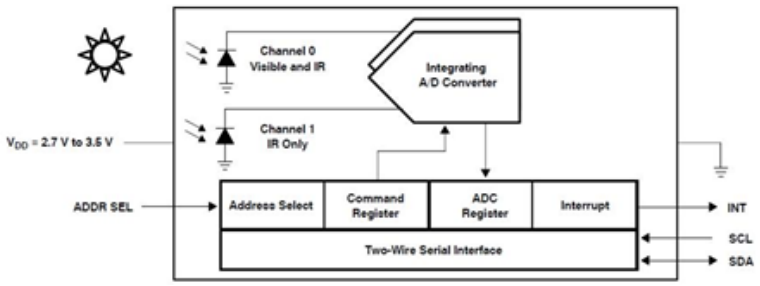
The sensor includes a photodetector and an onboard processing circuit that transmits light intensity as a digital signal, easily readable by an Arduino board through I2C communication (address 0x29). The I2C interface also supports multiple sensor connections on the same bus, making it well-suited for systems with diverse environmental monitoring needs.

The sensor offers selectable detection modes, letting users tailor light sensitivity for specific applications. It provides a high-resolution 16-bit digital output and operates in I2C Fast-Mode at 400 kHz, enabling quick data transfer for accurate measurements. Its wide dynamic range of 0.1 to 40,000 LUX allows it to detect light intensities across various environments, from dimly lit indoor settings to bright outdoor spaces. With an operational temperature range of -40°C to 85°C, the sensor is versatile enough for both indoor and outdoor use.

A notable feature is its programmable interrupt function, which allows users to define upper and lower light intensity thresholds. When the measured light surpasses or drops below these thresholds, the sensor can automatically trigger an interrupt, making it ideal for applications requiring immediate adjustments to changing light levels.

The sensor's broad range of detectable light intensities makes it ideal for applications such as automatic lighting control, screen brightness adjustment, and

environmental monitoring. With its compact form factor and easy integration into the Grove system, this sensor is a powerful tool for projects needing real-time light detection and control, from smart home systems and horticulture to portable devices that adapt to ambient light conditions.



Block Diagram Sensor Hardware

Figure 5E – Grove – Digital Light Sensor

In this lesson, the Grove – Digital Light Sensor is connected to the TWI (I2C) Communication Port on the Arduino MKR WiFi 1010 Board using the MKR Connector Carrier Board. To operate the sensor, a custom library is necessary, which can be downloaded from:

https://github.com/Seeed-Studio/Grove_Digital_Light_Sensor/archive/master.zip.

Detailed information on the light sensor can be found at:

https://wiki.seeedstudio.com/Grove-Digital_Light_Sensor/

Listings 5C and 5D contain sample Arduino sketches (*lightSensor.ino*, *lightSensorIR.ino*), which read the sensor's output, process the data to determine light intensities for visible and infrared spectrums, and display the results in the Arduino IDE's Serial Monitor.

The two sketches, *lightSensor.ino* and *lightSensorIR.ino*, demonstrate the use of the Digital Light Sensor (TSL2561) for measuring ambient light intensity and raw light channel values, respectively, using an Arduino.

lightSensor.ino

The sketch measures the ambient light intensity in Lux using the TSL2561 sensor. In the *setup()* function, *I2C communication* is initialized via *Wire.begin()*, and the serial communication is configured at 9600 baud. The light sensor is initialized using *TSL2561.init()*.

In the *loop()* function, the visible light intensity in Lux is read using *TSL2561.readVisibleLux()* and printed to the Serial Monitor with a 1-second interval.

lightSensorIR.ino

The sketch extends the functionality by reading raw infrared and full-spectrum light intensity values from the TSL2561 sensor. The *setup()* function initializes the I2C and serial communication, as well as the light sensor, similar to the previous sketch.

In the *loop()* function, raw data from the infrared channel (*TSL2561.readIRLuminosity()*) and full-spectrum channel (*TSL2561.readFSpecLuminosity()*) are retrieved and printed to the Serial Monitor. These raw values are not converted to Lux but provide deeper insights into the specific light components detected by the sensor. This capability is particularly useful for advanced applications like color detection, sunlight analysis, and object classification based on reflectivity.

Both sketches highlight the versatility of the TSL2561 sensor in measuring and analyzing light intensity, making it suitable for diverse applications in IoT and environmental sensing.

lightSensor.ino

```
#include <Wire.h>
#include <Digital_Light_TSL2561.h>

void setup() {
    Wire.begin();       // Initialize I2C Communication
    Serial.begin(9600); // Initialize Serial Port
    TSL2561.init();     // Initialize Light Sensor
}

void loop() {
    Serial.print("The Light value (Lux) is: ");
    Serial.println(TSL2561.readVisibleLux());
    delay(1000);
}
```

Listing 5C – Sketch (source code) to measure visible light intensity (Lux) using Grove – Digital Light Sensor and displays Lux value to the Serial Monitor.

lightSensorIR.ino

```
#include <Wire.h>
#include <Digital_Light_TSL2561.h>

void setup() {
    Wire.begin();
    Serial.begin(9600);
    TSL2561.init();
}

void loop() {
    Serial.print("The Infrared intensity value (Raw) is: ");
    // read Infrared channel raw intensity value only, not converted to Lux
    Serial.println(TSL2561.readIRLuminosity());

    Serial.print("The Full Spectrum intensity value (Raw) is: ");
    // read Full Spectrum channel raw intensity value only, not converted to Lux
    Serial.println(TSL2561.readFSpecLuminosity());
    delay(1000);
}
```

Listing 5D – Sketch (source code) to measure raw visible and IR light intensity values using Grove – Digital Light Sensor and displays raw intensity values to the Serial Monitor.

5.4 3-Axes Digital Accelerometer

Grove - 3-Axis Digital Accelerometer (±16g) (Figure 5F), based on the ADXL345 sensor is designed to measure acceleration along the X, Y, and Z axes with high accuracy. The ADXL345 is a low-power, MEMS (Micro-Electro-Mechanical System) accelerometer that can sense acceleration up to ±16g, making it suitable for applications ranging from basic tilt detection to more demanding uses like vibration monitoring, free-fall detection, and gesture recognition. The sensor provides digital output data and communicates with microcontrollers via an I2C (address 0x53) or SPI interface, offering flexibility in interfacing with an Arduino Board.

Figure 5F – Grove - 3-Axis Digital Accelerometer (±16g)

One of the key features of the ADXL345 is its high resolution, allowing it to detect changes in acceleration as small as 4 mg (0.004 g), making it suitable for applications that require fine-grained motion sensing. It also supports various measurement ranges (±2g, ±4g, ±8g, and ±16g), which can be configured based on the application's needs for sensitivity and range. The ADXL345's built-in features, such as tap detection, activity/inactivity monitoring, and orientation sensing, expand its utility in applications like portable electronics, fitness devices, gaming controls, and impact sensing.

In this lesson, the Grove - 3-Axis Digital Accelerometer (±16g) is connected to the TWI (I2C) Communication Port on the Arduino MKR WiFi 1010 Board using the MKR Connector Carrier Board. To operate the sensor, a custom library is necessary, which can be downloaded from:

https://github.com/Seeed-Studio/Accelerometer_ADXL345/archive/refs/heads/master.zip

Detailed information on the Grove - 3-Axis Digital Accelerometer can be found at:

https://wiki.seeedstudio.com/Grove-3-Axis_Digital_Accelerometer-16g/

Listing 5E contains a sample Arduino sketch (*accelSensor.ino*), which reads the sensor's output, processes the data to determine X, Y, Z accelerations, and displays the results in the Arduino IDE's Serial Monitor.

The *accelSensor.ino* sketch demonstrates the integration and functionality of the ADXL345 3-axis accelerometer, enabling precise monitoring of acceleration along the X, Y, and Z axes. The sketch is configured to detect activity, inactivity, tap, double-tap, and free-fall events. Additionally, it retrieves and displays raw accelerometer values and calculated acceleration in g-forces, making it ideal for various motion-sensing applications.

In the *setup()* function, the sketch initializes serial communication at a baud rate of 9600 to display debugging information in the Serial Monitor. The accelerometer is powered on using the *adxl.powerOn()*, and specific thresholds and parameters are configured for different functionalities. For activity and inactivity monitoring,

thresholds and timeouts are defined using *setActivityThreshold()* and *setInactivityThreshold()*, with monitoring enabled for all three axes. *Tap* and *double-tap* detection is configured to monitor only the *Z-axis*, while *free-fall* detection parameters are set for threshold and duration, enabling the detection of drops or sudden free-fall events. *Interrupts* for these functionalities are mapped to the *INT1* pin, and relevant interrupts are enabled, ensuring seamless detection and response.

The *loop()* function continuously reads data from the accelerometer. Using the *adxl.readXYZ()*, *raw X, Y,* and *Z* values are retrieved and displayed in the Serial Monitor. The *getAcceleration()* calculates acceleration in g-forces for each axis, providing more precise and user-friendly data. This processed acceleration data is also displayed on the Serial Monitor, making it easier to interpret and analyze motion events. A 500-millisecond delay is added to maintain periodic updates and ensure readability.

accelSensor.ino

```
#include <Wire.h>
#include <ADXL345.h>

ADXL345 adxl;   // Initialize ADXL345 instance of the ADXL345 library

void setup() {…
}

void loop() {
    int x, y, z;
    double xyz[3];
    double ax, ay, az;

    // Read the accelerometer values and store them in variables  x,y,z
    adxl.readXYZ(&x, &y, &z);

    // Output x,y,z values
    Serial.print("values of X , Y , Z: ");
    Serial.print(x);
    Serial.print(" , ");
    Serial.print(y);
    Serial.print(" , ");
    Serial.println(z);

    adxl.getAcceleration(xyz);
    ax = xyz[0];
    ay = xyz[1];
    az = xyz[2];
    Serial.print("X=");
    Serial.print(ax);
    Serial.println(" g");
    Serial.print("Y=");
    Serial.print(ay);
    Serial.println(" g");
    Serial.print("Z=");
    Serial.print(az);
    Serial.println(" g");
    Serial.println("**********************");
    delay(500);
}
```

Listing 5E – Sketch (source code) to measure X, Y, Z accelerations using Grove – 3-Axis Digital Accelerometer (±16g) and displays accelerations to the Serial Monitor (Some part of the sketch is not shown, please download the full code from the repository).

5.5 CO2, Temperature and Humidity Sensor

Grove – CO2, Temperature, and Humidity Sensor (Figure 5G) is an integrated environmental sensing module capable of measuring carbon dioxide (CO2) concentration, ambient temperature, and relative humidity. This sensor module combines multiple sensors, making it highly versatile for applications requiring comprehensive air quality monitoring, such as indoor air quality assessment, HVAC systems, and greenhouse environment control. The module features the SCD30

sensor, which employs NDIR (Non-Dispersive Infrared) technology for accurate CO2 detection and additional sensors for temperature and humidity measurements, ensuring reliable and stable readings across varying environmental conditions.

Figure 5G – Grove – CO2, Temperature, and Humidity Sensor

The sensor communicates with microcontrollers via an I2C interface (address 0x61), making it easy to integrate with an Arduino Board for real-time data collection and analysis. The CO2 sensing range is between 0 to 40,000 ppm, with high accuracy and a self-calibration feature, making it well-suited for monitoring CO2 levels in both residential and commercial spaces. In addition, the module provides temperature readings from -10°C to 60°C and humidity measurements from 0% to 100% RH, offering a complete view of ambient conditions.

In this lesson, the Grove – CO2, Temperature, and Humidity Sensor is connected to the TWI (I2C) Communication Port on the Arduino MKR WiFi 1010 Board using the MKR Connector Carrier Board. To operate the sensor, a custom library is necessary, which can be downloaded from:

https://github.com/Seeed-Studio/Seeed_SCD30/archive/refs/heads/master.zip

Detailed information on the Grove – CO2, Temperature, and Humidity Sensor can be found at:

https://wiki.seeedstudio.com/Grove-CO2_Temperature_Humidity_Sensor-SCD30/

The *CO2THSensor.ino* sketch (Listing 5F) illustrates how to use the CO2, Temperature, and Humidity Sensor (SCD30) with an Arduino Board to monitor essential environmental parameters in real time and displays the results in the Arduino IDE's Serial Monitor. This setup enables the collection of data for carbon dioxide (CO2) concentration, ambient temperature, and relative humidity, which are vital for applications such as air quality monitoring and climate control.

The *setup()* function initializes the hardware and communication protocols necessary for the sensor to operate. The I2C communication is established to facilitate data transfer between the Arduino and the SCD30 sensor. Serial communication is set up at a baud rate of 115200, allowing the sensor readings to be displayed on the Serial Monitor. A message, *SCD30 Raw Data*, is output to the Serial Monitor as an indicator that the initialization process has completed successfully. The sensor itself is initialized using *scd30.initialize()*, ensuring it is ready to collect data.

In the *loop()* function, the program continuously checks for new data availability using *scd30.isAvailable()*. If data is ready, the *scd30.getCarbonDioxideConcentration()* retrieves three key parameters: CO2 concentration, temperature, and humidity, which are stored in the *result* array. These values are then displayed on the Serial

Monitor. The CO2 concentration is reported in parts per million (ppm), offering insights into air quality. The temperature is shown in degrees Celsius (°C), providing real-time monitoring of ambient conditions, while the humidity is presented as a percentage (%), indicating relative moisture levels in the environment. A two-second delay is included in the loop to ensure periodic updates while preventing excessive data refresh rates.

CO2THSensor.ino

```
#include "SCD30.h"
#define SERIAL Serial

void setup() {
    Wire.begin();                    // Initialize I2C Communication
    SERIAL.begin(115200);            // Initialize Serial Port
    SERIAL.println("SCD30 Raw Data");
    scd30.initialize();              // Initialize Grove – CO2, Temperature, and Humidity Sensor
}

void loop() {
    float result[3] = {0};
    if (scd30.isAvailable()) {
        scd30.getCarbonDioxideConcentration(result);
        SERIAL.print("Carbon Dioxide Concentration is: ");
        SERIAL.print(result[0]);
        SERIAL.println(" ppm");
        SERIAL.println(" ");
        SERIAL.print("Temperature = ");
        SERIAL.print(result[1]);
        SERIAL.println(" ℃");
        SERIAL.println(" ");
        SERIAL.print("Humidity = ");
        SERIAL.print(result[2]);
        SERIAL.println(" %");
        SERIAL.println(" ");
        SERIAL.println(" ");
        SERIAL.println(" ");
    }
    delay(2000);
}
```

Listing 5F – Sketch (source code) to measure CO2 concentration, ambient temperature and humidity values using Grove – CO2, Temperature, and Humidity Sensor and displays results to the Serial Monitor.

5.6 Heart Rate and Pulse Oximetry Sensor (Advanced)

The MAX30100 (Figure 5H) is a compact, integrated sensor module designed for heart rate and pulse oximetry monitoring. It combines two LEDs (one red and one infrared), a photodetector, optimized optical components, and low-noise analog signal processing to detect and measure heart rate and blood oxygen saturation (SpO2) levels.

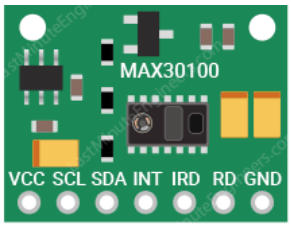


Figure 5H – MAX30100 Heart Rate and Pulse Oximetry integrated sensor module

The MAX30100 module illustrates the connections necessary for integrating the heart rate and pulse oximetry sensor with a microcontroller. It operates with a voltage input on the VCC pin, which can range from 3.3V to 5V. Communication is handled via the I2C interface (address 0x57), connecting to the corresponding I2C pins on a microcontroller such as an Arduino Board.

The INT (interrupt) pin is an optional feature that can be used to signal the microcontroller when new data is available, allowing for efficient power usage by eliminating the need for constant polling. Additionally, the module has two LED driver control pins: IRD for the infrared LED and RD for the red LED. These LEDs emit light at specific wavelengths to detect heart rate and blood oxygen levels, while the photodetector measures the reflected light, enabling accurate calculations.

The GND pin provides the ground connection, completing the electrical circuit. This comprehensive design, with dedicated pins for I2C communication, interrupts, and LED control, makes the MAX30100 module suitable for wearable and portable health monitoring systems.

In this lesson, the MAX30100 module is connected to the TWI (I2C) Communication Port on the Arduino MKR WiFi 1010 Board using the MKR Connector Carrier Board. To operate the sensor, a custom library is necessary, which can be downloaded from:

https://github.com/oxullo/Arduino-MAX30100/archive/refs/heads/master.zip

Detailed information on the module can be found at:

https://how2electronics.com/interfacing-max30100-pulse-oximeter-sensor-arduino/

https://lastminuteengineers.com/max30100-pulse-oximeter-heart-rate-sensor-arduino-tutorial/#google_vignette

The *heartRateSensor.ino* sketch (Listing 5G) demonstrates the use of an Arduino Board to interface with the MAX30100 Pulse Oximeter and Heart Rate Sensor. By employing the *MAX30100_PulseOximeter library*, the program monitors and displays heart rate and blood oxygen saturation (SpO2) levels in real-time using the Arduino Serial Monitor.

The *setup()* function is responsible for initializing the Arduino Board's serial communication and configuring the MAX30100 sensor. The serial communication is set up at a baud rate of 115200 for debugging and data visualization purposes. The MAX30100 sensor is initialized using *pox.begin()*, which returns *TRUE* if the initialization is successful. If the sensor fails to initialize, the program halts in an infinite loop, ensuring no further execution errors. Additionally, the infrared (IR) LED current is configured to *7.6mA* using *pox.setIRLedCurrent()*, a value that balances power consumption with measurement accuracy. To enhance interactivity, a callback function *onBeatDetected()* is registered using *pox.setOnBeatDetectedCallback()*, which outputs *Beat!* to the Serial Monitor whenever a heartbeat is detected.

The *loop()* function is the core of the program, where continuous monitoring of heart rate and SpO2 levels occurs. The *pox.update()* is called frequently to process sensor data in real time. Data reporting occurs every 1000 milliseconds (1 second), defined by the *REPORTING_PERIOD_MS* constant. During each reporting interval, the heart rate is retrieved using *pox.getHeartRate()*, and SpO2 is obtained using *pox.getSpO2()*. Both values are printed to the Serial Monitor for real-time

observation. If either value is 0, it indicates invalid readings, possibly due to poor sensor contact or incorrect placement.

This implementation highlights the flexibility and functionality of the MAX30100 sensor. The beat detection callback provides instant feedback when a heartbeat is identified, enhancing the user experience. The ability to adjust the IR LED current allows customization for different environments or user requirements, ensuring optimal performance. The program's design ensures real-time and periodic updates of vital metrics, making it suitable for a wide range of applications.

heartRateSensor.ino

```
#include <Wire.h>
#include "MAX30100_PulseOximeter.h"
#define REPORTING_PERIOD_MS 1000

// Initialize a Pulse Oximeter instance (beat detection, heart rate, SpO2)
PulseOximeter pox;
uint32_t tsLastReport = 0;

// Callback (registered below) fired when a pulse is detected
void onBeatDetected()
{ ···
}

void setup()
{ ···
}

void loop()
{
    // Make sure to call update as fast as possible
    pox.update();
    // Asynchronously dump heart rate and oxidation levels to the serial
    // For both, a value of 0 means "invalid"
    if (millis() - tsLastReport > REPORTING_PERIOD_MS) {
        Serial.print("Heart rate:");
        Serial.print(pox.getHeartRate());
        Serial.print("bpm / SpO2:");
        Serial.print(pox.getSpO2());
        Serial.println("%");
        tsLastReport = millis();
    }
}
```

Listing 5G – Sketch (source code) to measure the Heart Rate and SpO2 values using MAX30100 module and displays values to the Serial Monitor (Some parts of the sketch are not shown, please download the full code from the repository).

5.7 Thermopile Sensors Array (Advanced)

Passive infrared (PIR) sensor does a good job at detecting the presence of warm moving bodies in a room (or an enclosed area). However, these simple sensors won't detect motionless objects. Nor can they tell you the direction of any

movements within their field of view. To get around these sensing limitations, you need a more sophisticated passive IR sensing technology known as a Thermopile Sensors Array (TSA).

TSA measures radiated thermal infrared energy (thermal emission) from a warm object within its field of view. This resulted in a thermal image of the object, and the image quality is dependent on the TSA resolution. An object's temperature at an identified hotspot on the thermal image can be determined by applying Stefan-Boltzmann Law.

A TSA consists of thermistors (temperature-sensitive resistors) arranged in a two-dimensional grid, an infrared lens (typically made from silicon or germanium) to focus thermal radiation onto the sensor array, and dedicated signal processing hardware. This configuration enables the TSA to detect thermal patterns and measure temperature across a defined area, offering more detailed spatial information than a simple PIR sensor.

Advances in TSA technology have made it possible to use these arrays for non-contact human body temperature measurement, which is especially valuable in situations such as infectious disease outbreaks. By providing accurate temperature readings, TSA technology can help identify individuals with elevated body temperatures, a potential sign of infection. This capability is essential for early screening in public spaces, as elevated core body temperature, or fever, is often associated with active viral or bacterial infections.

AMG8833 TSA (Figure 5I) is a precision infrared sensor utilizing advanced MEMS technology. This sensor can detect temperature across a two-dimensional area, featuring an 8 × 8 grid of 64 pixels that can measure temperature variations in each pixel. Each pixel in the array provides an individual temperature reading, allowing the sensor to create a detailed heat map of the observed area.

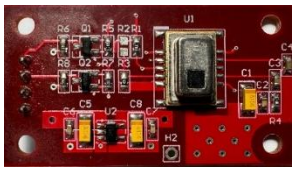


Figure 5I – AMG8833 TSA Module

The sensor's target temperature sensing range spans from 0 to 80°C, with an accuracy of ±2.5°C, making it suitable for precise, non-contact monitoring of object temperatures within this range. Additionally, AMG8833 features a 60° Field of View and a maximum detection range of up to 7 meters, enabling it to capture temperature data across a wide area, suitable for applications such as human presence detection, object monitoring, low-resolution thermal imaging, and environmental temperature sensing.

AMG8833 TSA communicates with microcontrollers using an I2C interface (address 0x68) allowing straightforward integration with an Arduino Board for real-time temperature data collection and analysis.

In this lesson, AMG8833 TSA is connected to the TWI (I2C) Communication Port on the Arduino MKR WiFi 1010 Board using the MKR Connector Carrier Board. To operate the sensor, a custom library is necessary, which can be downloaded from:

https://github.com/Seeed-Studio/Seeed_AMG8833/archive/refs/heads/master.zip

Detailed information on the AMG8833 TSA can be found at:

https://wiki.seeedstudio.com/Grove-Infrared_Temperature_Sensor_Array-AMG8833/

The *tempTSA.ino* sketch (Listing 5H) is designed to interface the AMG8833 Thermal Sensor Array (TSA) with an Arduino Board for real-time temperature data acquisition and visualization. This sensor captures thermal infrared radiation from an 8x8 grid of pixels, providing temperature values for a two-dimensional matrix, and displaying it in the Arduino IDE's Serial Monitor.

In the *setup()* function, the program initializes the serial communication at a baud rate of 115200, enabling high-speed communication with the Arduino Serial Monitor. The AMG8833 sensor is initialized using *sensor.init()*. If the initialization fails, the program outputs an error message *init failed!!!* and halts execution. Successful initialization results in the message *AMG8833 init OK!!!* being displayed on the Serial Monitor, ensuring the sensor is ready for operation.

The *loop()* function continuously reads and processes the temperature data from the sensor. The temperature values for all 64 pixels of the sensor are stored in an array named *pixels_temp* using *sensor.read_pixel_temperature()*. These values represent the thermal data captured by the sensor in degrees Celsius.

To display the thermal data, the program uses a *nested logic structure* to print the temperature values in an *8x8 grid* format on the Serial Monitor. Each temperature value is printed with a space for separation, and after every eighth value (completing a row), the program moves to the next line for better readability. Additional blank lines are printed after the matrix for visual separation between successive readings. The system pauses for 500 milliseconds after each reading, ensuring a consistent refresh rate.

The AMG8833 sensor allows for real-time thermal imaging and spatial temperature monitoring, with applications ranging from motion detection and human presence sensing to industrial and medical diagnostics. By presenting the data in an 8x8 grid, the program mimics the format of a thermal image, providing a clear visual representation of the heat distribution across the sensor's field of view.

This implementation can be extended for applications such as non-contact temperature monitoring, thermal mapping, and even integration with machine learning algorithms for gesture recognition or object detection. With further enhancements, such as graphical visualization tools or integration with cloud platforms, the system could serve as a compact and efficient solution for advanced thermal imaging applications.

tempTSA.ino

```
#include "Seeed_AMG8833_driver.h"
// Initialize an AMG8833 TSA instance
AMG8833 sensor;

void setup() {
    Serial.begin(115200);

    if (sensor.init()) {
        Serial.println("init failed!!!");
        while (1);
    }
    Serial.println("AMG8833 init OK!!!");
}

void loop() {
    u8 val = 0;
    float pixels_temp[PIXEL_NUM] = {0};

    // Read temperature of all the 64 pixels
    sensor.read_pixel_temperature(pixels_temp);
    // Print 8x8 pixels temperature value in 8 rows by 8 columns
    Serial.println("Temperature for 8x8 matrix are:");

    for (int i = 0; i < PIXEL_NUM; i++) {
        Serial.print(pixels_temp[i]);
        Serial.print("  ");
        if (0 == (i + 1) % 8) {
            Serial.println(" ");
        }
    }
    Serial.println(" ");
    Serial.println(" ");
    delay(500);
}
```

Listing 5H – Sketch (source code) to determine an array of temperature values within the Field of View using AMG8833 TSA module and displays values to the Serial Monitor.

6. Sensor Application Examples

These application examples illustrate how a combination of sensors can monitor environmental conditions, detect motion and health metrics, and enable automated responses, enhancing safety, convenience, and adaptability in both indoor and outdoor settings. By addressing specific needs in areas like agriculture and elderly care, these sensor systems provide real-time monitoring and automation, improving safety, efficiency, and overall quality of life.

6.1 Smart Home Environmental Monitoring and Automation System

This application focuses on creating a smart home system that monitors environmental conditions and automates various responses to improve comfort, safety, and efficiency.

Components and Roles

- **CO2, Temperature, and Humidity Sensor:** Monitors indoor air quality, temperature, and humidity levels.
- **Digital Light Sensor:** Adjusts indoor lighting based on ambient light conditions.
- **PIR Motion Sensor:** Detects human presence in different rooms, allowing the system to turn lights on or off based on occupancy.
- **Button Switch:** Serves as a manual control to activate/deactivate certain features, like toggling between "day" and "night" modes.
- **Temperature Sensor:** Measures temperature in various home zones to provide detailed climate control.
- **Sound Sensor:** Detects noise levels to trigger security features if unusual sounds are detected when the house is empty.
- **AMG8833 TSA:** Monitors heat patterns across rooms, detecting multiple occupants or pets, and adjusting the HVAC system accordingly.

Functionality

This system continuously monitors the indoor environment using temperature, CO2, and humidity sensors to ensure optimal air quality and comfort. Based on ambient light readings from the Digital Light Sensor, it adjusts artificial lighting to maintain consistent brightness throughout the day and conserve energy at night.

The PIR Motion Sensor and AMG8833 TSA work together to detect presence, adjusting heating or cooling systems based on occupancy to save energy in unused rooms. The Sound Sensor listens for unusual sounds to trigger security alerts.

The system's functionality can be manually adjusted using the Button Switch to switch between preconfigured modes, making the home environment both adaptive and user-friendly.

6.2 Home Occupancy and Motion Activity Monitoring for Elderly

This application is designed to monitor the motion activity and environmental conditions of an elderly individual living alone in a studio apartment, ensuring their safety, comfort, and well-being.

Components and Roles

- **PIR Motion Sensor:** Detects human motion activity.
- **AMG8833 TSA:** Monitors heat patterns, detecting occupants.
- **Sound Sensor:** Detects noise levels to trigger emergencies requests if unusual sounds are detected.
- **CO2, Temperature, and Humidity Sensor:** Monitors indoor air quality, temperature, and humidity levels.

Functionality

This monitoring system utilizes the AMG8833 Thermopile Sensor Array (TSA) and Passive Infrared (PIR) Motion Sensor to detect the presence and movements of elderly individuals in their living spaces. By continuously tracking daily activities, the system can identify irregularities such as prolonged inactivity, which may indicate potential health issues or emergencies. In such cases, caregivers or family members are immediately notified for timely intervention, ensuring prompt response and safety.

To further enhance security and responsiveness, a Sound Sensor is integrated to detect unusual noises, such as loud impacts or distress calls, that might signal an emergency situation. Simultaneously, a CO2, Temperature, and Humidity Sensor monitors air quality and environmental conditions, ensuring the apartment remains a safe and comfortable space. This sensor tracks metrics such as carbon dioxide levels, temperature, and humidity, enabling early detection of adverse conditions.

By combining motion tracking, sound detection, and environmental monitoring, the system provides real-time insights and alerts. This integration not only fosters a secure and supportive environment for the elderly but also delivers peace of mind to their loved ones, making it an invaluable solution for independent living and elderly care.

6.3 Comprehensive Elderly Care and Fall Detection System

This application is a wearable and environmental monitoring system designed to track the health and safety of elderly individuals, helping caregivers monitor vital signs, detect falls, and respond to emergencies.

Components and Roles

- **MAX30100 Sensor:** Continuously monitors heart rate and blood oxygen levels, providing alerts for abnormal readings.
- **3-Axis Digital Accelerometer:** Detects movement and orientation, identifying falls or sudden inactivity and triggering emergency notifications.
- **AMG8833 TSA:** Monitors body temperature and surrounding heat patterns, detecting potential signs of fever or hypothermia.
- **Temperature Sensor:** Measures ambient room temperature to ensure a safe, comfortable environment for the elderly individual.
- **Ultrasonic Ranger:** Measures distances to obstacles, providing mobility assistance by detecting obstructions in pathways.
- **Sound Sensor:** Detects loud noises, such as a fall or call for help, triggering alerts in case of potential emergencies.
- **Digital Light Sensor:** Monitors ambient light levels, automatically adjusting lighting to reduce the risk of falls in low-light conditions.
- **PIR Motion Sensor:** Detects motion to verify activity levels, identifying periods of inactivity that may indicate distress or require wellness checks.
- **Button Switch:** Acts as an emergency call button, allowing the user to manually summon help when needed.

Functionality

The Comprehensive Elderly Care and Fall Detection System continuously monitors both the health of the individual and their surroundings to ensure safety and comfort. The MAX30100 Sensor tracks vital signs, alerting caregivers if heart rate or blood oxygen levels fall outside of safe ranges.

The 3-Axis Digital Accelerometer detects falls or abnormal movement patterns, automatically triggering emergency notifications. The AMG8833 TSA provides non-contact body temperature monitoring, helping caregivers identify early signs of fever.

Room temperature is managed by the Temperature Sensor, while the Digital Light Sensor adjusts lighting to reduce fall risks. The Ultrasonic Ranger provides mobility assistance by identifying obstacles in the individual’s path. The Sound Sensor detects potential distress signals, and the PIR Motion Sensor tracks activity levels to ensure regular movement.

In emergencies, the Button Switch allows the individual to call for help, providing peace of mind and promoting independence while enabling timely intervention. This system enhances elderly care by integrating health monitoring, fall detection, and environmental safety, all within a user-friendly framework.

6.4 Smart Agricultural Monitoring System

This application leverages various sensors to monitor environmental and soil conditions in agricultural fields, helping farmers optimize crop health, conserve resources, and increase productivity.

Components and Roles

- **CO2, Temperature, and Humidity Sensor:** Measures ambient temperature and air quality parameters in greenhouses to maintain an ideal growing environment for plants.
- **Water Sensor:** Detects soil moisture levels to automate irrigation, ensuring crops receive adequate water without waste.
- **Digital Light Sensor:** Monitors sunlight levels, adjusting shade or artificial lighting for optimal plant growth in controlled environments.
- **Ultrasonic Ranger:** Measures the height of plants or water levels in tanks, allowing farmers to monitor crop growth and water reserves.
- **PIR Motion Sensor:** Detects the presence of animals or unauthorized personnel near the crops, triggering security alerts to protect the fields.
- **3-Axis Digital Accelerometer:** Monitors wind-induced vibrations in plants or structures, helping to assess wind damage risk and implement protective measures.
- **Angle Sensor:** Measures the tilt of irrigation equipment or solar panels, ensuring they are optimally positioned.
- **AMG8833 TSA:** Scans for heat signatures of pests or disease hot spots, providing early warnings for potential crop threats.

Functionality

This system continuously monitors environmental and soil conditions to help farmers maintain ideal growing conditions. The CO2, Temperature, and Humidity Sensor gather data on ambient temperature, air quality, and humidity levels, enabling adjustments in greenhouse settings.

The Water Sensor checks soil moisture, triggering the irrigation system when moisture is low, while the Digital Light Sensor adjusts lighting to maintain optimal sunlight exposure. The PIR Motion Sensor detects movement near crops to prevent animal or unauthorized access, and the Ultrasonic Ranger tracks plant height for growth analysis.

Additionally, the 3-Axis Digital Accelerometer detects high winds, prompting alerts if protection is needed. The AMG8833 TSA scans for abnormal heat patterns, indicating potential pest activity or disease. The Smart Agricultural Monitoring System enables data-driven crop management, improving yield and resource efficiency.

6.5 Condition-based Monitoring System for Predictive Maintenance of swimming pool motor pumps.

This system continuously monitors motor pump health, identify early signs of mechanical or thermal anomalies, and prevent unexpected failures through timely maintenance actions by analyzing vibrations, sound and detects abnormal heat patterns around the pump.

Components and Roles

- **3-Axis Digital Accelerometer:** Measures motor vibrations across x, y, and z axes.
- **Sound Sensor:** Measures sound from a running motor pump.
- **AMG8833 TSA:** Scans for heat signatures around the motor pump.

Functionality

The Condition-Based Monitoring System is a solution aimed at maintaining the health and operational efficiency of motor pumps. This system leverages three critical sensors: a 3-Axis Digital Accelerometer to monitor vibrations, a Sound Sensor to detect acoustic anomalies, and an AMG8833 TSA to identify heat signatures. Together, these sensors provide comprehensive insights into potential mechanical, acoustic, and thermal anomalies, enabling timely interventions to prevent unexpected failures.

The 3-Axis Digital Accelerometer measures vibrations along the x, y, and z axes to compute a resultant vibration magnitude. This value is compared against predefined thresholds to detect mechanical issues such as unbalanced loads, shaft misalignment, or bearing wear. Real-time vibration analysis ensures that mechanical faults are identified at an early stage, allowing preventive maintenance actions to mitigate serious damage. The accelerometer's precise readings help predict potential breakdowns and improve the pump's reliability.

The Sound Sensor complements vibration monitoring by capturing acoustic signals from the pump. Changes in noise intensity or irregular sound patterns can indicate potential problems such as cavitation, mechanical stress, or loose components. By continuously measuring and analyzing sound levels, the system provides additional diagnostic capability to detect anomalies early. This integration of acoustic monitoring enhances the overall reliability of the system by addressing sound-based indicators of mechanical issues.

The AMG8833 TSA enhances the system by detecting heat patterns across the pump's surface. It generates an 8x8 thermal grid, pinpointing areas where temperatures exceed standard operational limits. These thermal anomalies may indicate overheating, friction, electrical faults, or cooling system inefficiencies. Early detection of these hotspots facilitates prompt corrective measures, preventing critical failures caused by excessive heat.

The integration of vibration, acoustic, and thermal data provides a holistic view of the motor pump's health. Alerts are triggered when any of the vibration magnitude, noise levels, or temperature readings exceed their respective thresholds. If multiple anomalies occur simultaneously, a critical alarm is raised to prompt immediate maintenance. The system visualizes the collected data using an LCD display,

Arduino's Serial Monitor, or a cloud-based dashboard, enabling local and remote monitoring of pump health.

This predictive maintenance approach enhances operational reliability, minimizes unplanned downtime, and optimizes maintenance costs. By addressing mechanical, acoustic, and thermal issues proactively, the system extends the pump's lifespan and improves overall efficiency. Future iterations may incorporate machine learning for anomaly detection and cloud-based analytics for deeper insights, further advancing the capabilities of the monitoring system. This combination of sensors ensures a comprehensive monitoring solution for motor pumps, reducing risks and ensuring smooth operation.

- The End -