SEH500-EARTHQUAKE DETECTION(EARLY WARNING) SYSTEM
PROJECT REPORT

Introduction:

Earthquake are natural disasters that pose a threat to both people and infrastructure which often comes without warning and cause devastating consequences. They can be defined as sudden shaking of the ground caused by the passing of seismic waves through the earth's rocks. As we can also see what is happening in the world now the amount of earthquake around different parts of the globe have significantly increased. As Ruth states in the article Earthquakes In The Warming World," *Today, earthquakes are becoming more frequent and more severe as a result of global warming. While earthquakes cause tsunamis, new research is revealing that vast storms like hurricanes, along with other events brought on by climate change, can lead to an increase in seismic activity and thereby cause more earthquakes."(Atmos, 2023).*

Therefore there is a need for an earthquake detection system to provide real-time information hence giving early warning to minimize the loss of human lives and infrastructure.

Background Research:

It's not that there aren't systems in place for earthquake detection. There are several existing solutions for it. This is what we found:

- Seismometers and Seismic Networks:
    - The traditional method of using ground movement detection caused by seismic waves.
    - Disadvantage: Although they are effective, these systems often suffer from limited coverage and may provide only seconds to minutes of warning.
- Global Positioning System (GPS) and InSAR:
    - Donnellan, Andrea & Lyzenga, Gregory & Peltzer, Gilles & Webb, Frank & Heflin, M. & Zumberge, James in their research state*: "GPS is useful for understanding both in-interseismic and postseismic deformation. Interferometric synthetic aperture radar (InSAR) is also providing valuable crustal deformation data."*
    - The system indicates the likelihood of earthquakes.

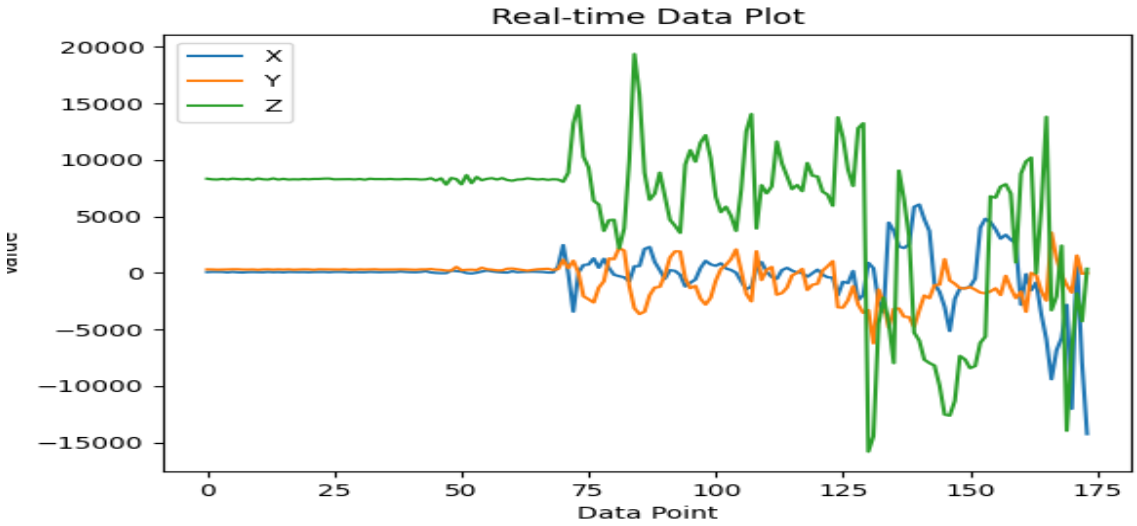
    - Disadvantage: These systems don't provide immediate warnings but aid in understanding long-term strain accumulation.
- Research:
    - It can't be said a system per se but in this scientists identify precursor signals or other anomalies that precede seismic events.
    - Disadvantage: Precursor readings aren't reliable.

Solution:

To come up with a more real-time solution we decided to use an accelerometer-based earthquake warning detection system.
Here are some advantages we found over the other systems:

- Real-time Monitoring:
    - Accelerometers provide continuous, real-time data, allowing for immediate detection and analysis of ground vibrations.
- Portability and Accessibility:
    - Accelerometers can be integrated into various devices, such as smartphones or IoT devices, enhancing their accessibility.
- High Resolution
    - Accelerometers offer high-resolution data, capturing detailed information about seismic activities and ground movements.

In our project, we used it to plot a real-time graph that gives us an indication that there is a deviation from normal movement.
A sample of the graph from our project when it detects unusual movements can be seen below:
The accelerometer was first stable upon shaking the microcontroller the data from the accelerometer changed substantially which can be noticed from the graph.



Project Work:

At the start of our project when we were brainstorming for our project idea we had several options, in which one of our ideas was to create a Sonar detector that would detect objects in the nearby sensor area using a HC-SR04 Ultrasonic sensor. However, after the professor's suggestion that the project was too complex for the time, we scratched the idea.

We decided to go for an earthquake detection system that can be used for early warnings. It looked feasible because our microcontroller has an in-built **FXOS8700CQ accelerometer**.

In our case, we only needed the accelerometer sensor to work with the I2C for communication. But whenever we started writing code for the setup of the FXOS8700CQ accelerometer and the I2C, we always had errors related to the Pins being different, we were using the wrong functions to initialize their components and we had the wrong files included in our project. And we wasted days in this process.
After a while, we came across an SDK example for our FRDM-K66 microcontroller named “encompass” which uses the magnetometer that is built with the FXOS8700CQ accelerometer. The encompass example was using the I2C protocol to communicate with the magnetometer.

The most challenging part was to figure out all of the configurations related to the accelerometer among the vast array of files.
After a few days, we figured it out and came to these conclusions

- pin_mux.h and pin_mux.c
- board.c and board.h
- Peripherals.h and peripherals.c
- Clock_config.c and clock_config.h
- Fsl_fxos.h and fsl_fxos.c
- Fsl_fxos_config.h and fsl_fxos_config.c and many other files

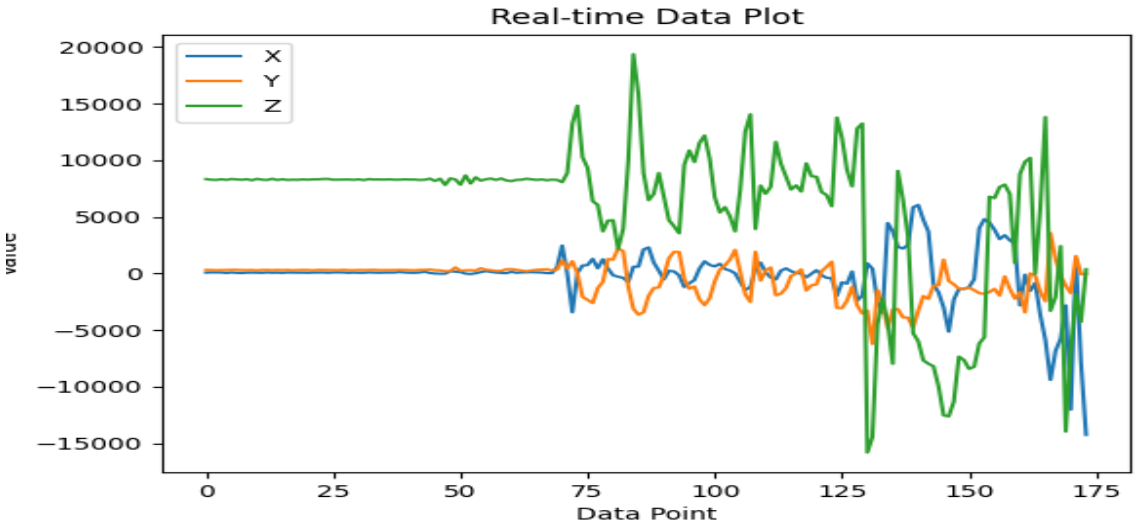
Through this we got a head start for our project as now we knew all the pins, functions, and header files that we needed to use for accelerometer and I2C.

After successfully setting up our project to read data from the accelerometer using the I2C communication we wanted to send this data to plot it using a tool such as processing IDE(which we substituted for VS Code with matplotlib library as processing IDE setup was complicated). **The other major obstacle we faced was we had the wrong idea of using UART** which was rectified by the prof we realized we were overcomplicating things and we were able to establish secure serial communication and were able to plot the graph. Here is just a part of the code we wrote(**the code we submitted does have the complete commented portions of UART for reference**).

```
//    char buffer[50];
//  hal_uart_handle_t handle;
//  const hal_uart_config_t actualConfiguration={
////             115200
//      .srcClock_Hz = 48000000,
//      .baudRate_Bps = 115200,
//      .parityMode = kHAL_UartParityDisabled,
//      .stopBitCount = kHAL_UartOneStopBit,
//      .enableRx = 1,
//      .enableTx = 1,
//      .enableRxRTS = 0,
//      .enableTxCTS = 0,
//      .instance = 0
//  };

//  const hal_uart_config_t *configuration = &actualConfiguration;

//    status_t uartInitStatus = HAL_UartInit(handle, configuration);
//        if (uartInitStatus != kStatus_HAL_UartSuccess) {
//            PRINTF("\r\nUART initialization failed!\r\n");
//        }
```

After that, we used **Putty** to check the values if our serial communication was working from our COM3 (where our microcontroller is connected). And we saw that we were able to get the accelerometer values through that port. After that, we started working on our Python code.
At first, we were not able to print a graph because we had set the **wrong baud rate** in our python so went back to our MCUexpresso code and checked for a suitable baud rate which was **115200.** Initially, we only used the matplotlib library to plot a graph but it was not working for some reason. So we researched more about Python libraries and got the **FuncAnimation** which is a part matplotlib.animation library and we were able to plot a graph.
The code that we used for visualization and sample output is shown below:

```python
import serial
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
# Initialize lists to store data
data_x = []
data_y = []
data_z = []
# Establish serial connection
ser = serial.Serial(
    "COM3", 115200
)  # Replace 'COM3' with your microcontroller's serial port and baud rate
# Function to handle incoming data
def handle_data(data):
    data = (
        data.decode().strip()
    )  # Decode byte data and remove trailing newline/whitespace
    if data:  # Check if the received data is not empty
        try:
            # Assuming the format is "X:value, Y:value, Z:value"
            values = data.split(",")  # Split data into individual values
            x_value = int(values[0].split(":")[1])  # Extract X value
            y_value = int(values[1].split(":")[1])  # Extract Y value
            z_value = int(values[2].split(":")[1])  # Extract Z value

            # Store the received values
            data_x.append(x_value)
            data_y.append(y_value)
            data_z.append(z_value)
        except (ValueError, IndexError):
            pass  # Ignore invalid data or index errors
# Update function for the animation
def update(frame):
    if ser.in_waiting > 0:
        data = ser.readline()  # Read data from the serial port
        handle_data(data)  # Handle the received data

        # Plot the received data
        plt.cla()  # Clear the previous plot
        plt.plot(data_x, label="X")  # Plot X values
        plt.plot(data_y, label="Y")  # Plot Y values
        plt.plot(data_z, label="Z")  # Plot Z values
        plt.xlabel("Data Point")
        plt.ylabel("Value")
        plt.title("Real-time Data Plot")
        plt.legend()  # Show legend
# Set up the animation
ani = FuncAnimation(plt.gcf(), update, interval=100)  # Update plot every 100ms
plt.show()  # Display the animated plot
# Close the serial connection (not reached in this example)
ser.close()
```



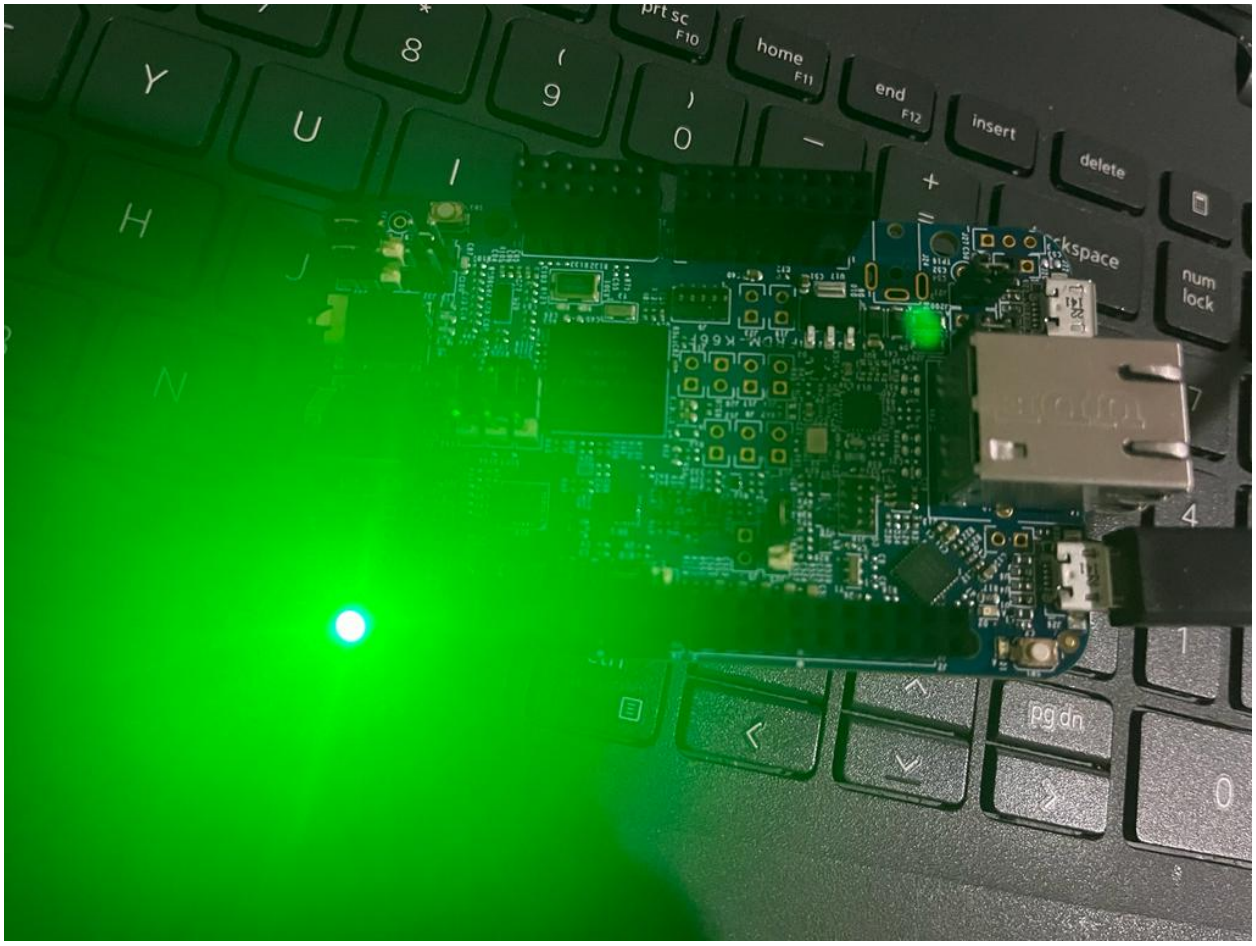
After that, we thought of adding an **interrupt** for both starting the application and shutting it down gracefully.

For the start, we thought of adding a **green LED** as an indication that the program has started reading values from the accelerometer. We took help from Lab 7 of our course to add both functionalities. But when we added the assembly code for the LED light to our project we were

not able to turn on the LED as it was throwing an error for some reason. After a bit of research, we got to know that the problem was persistent because **we were not setting up our clock configuration accordingly** for which we added this code:

```
// Setting up the clock config
__asm volatile("ldr r1, =0x40048038 \n"
               "ldr r0, [r1] \n"
               "orr r0, r0, #(1 << 13) \n"
               "str r0, [r1]");
```

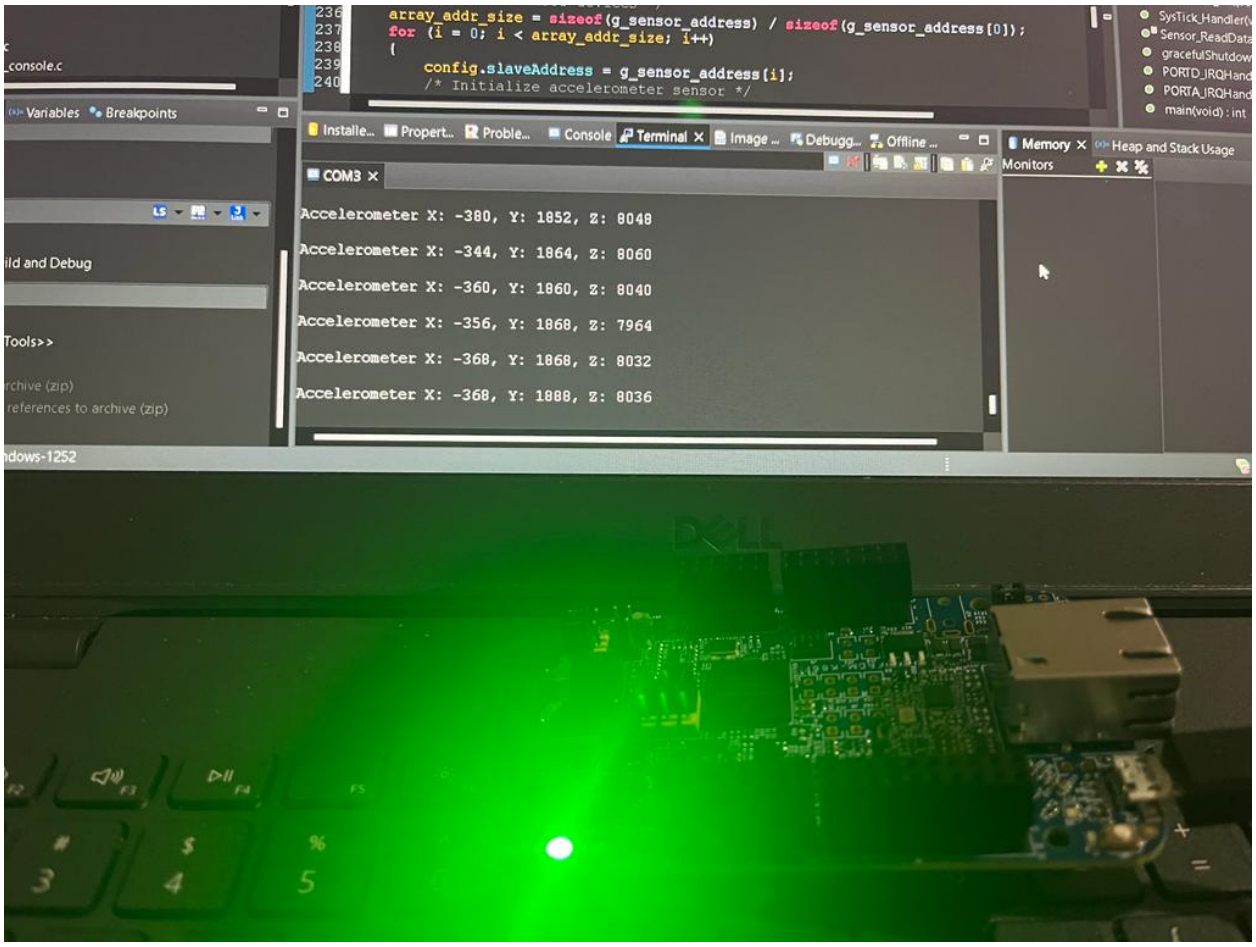
By adding this code we were able to set up our green LED Pins and configuration and were able to turn on the LED. **(a photo of our working LED is given below)**



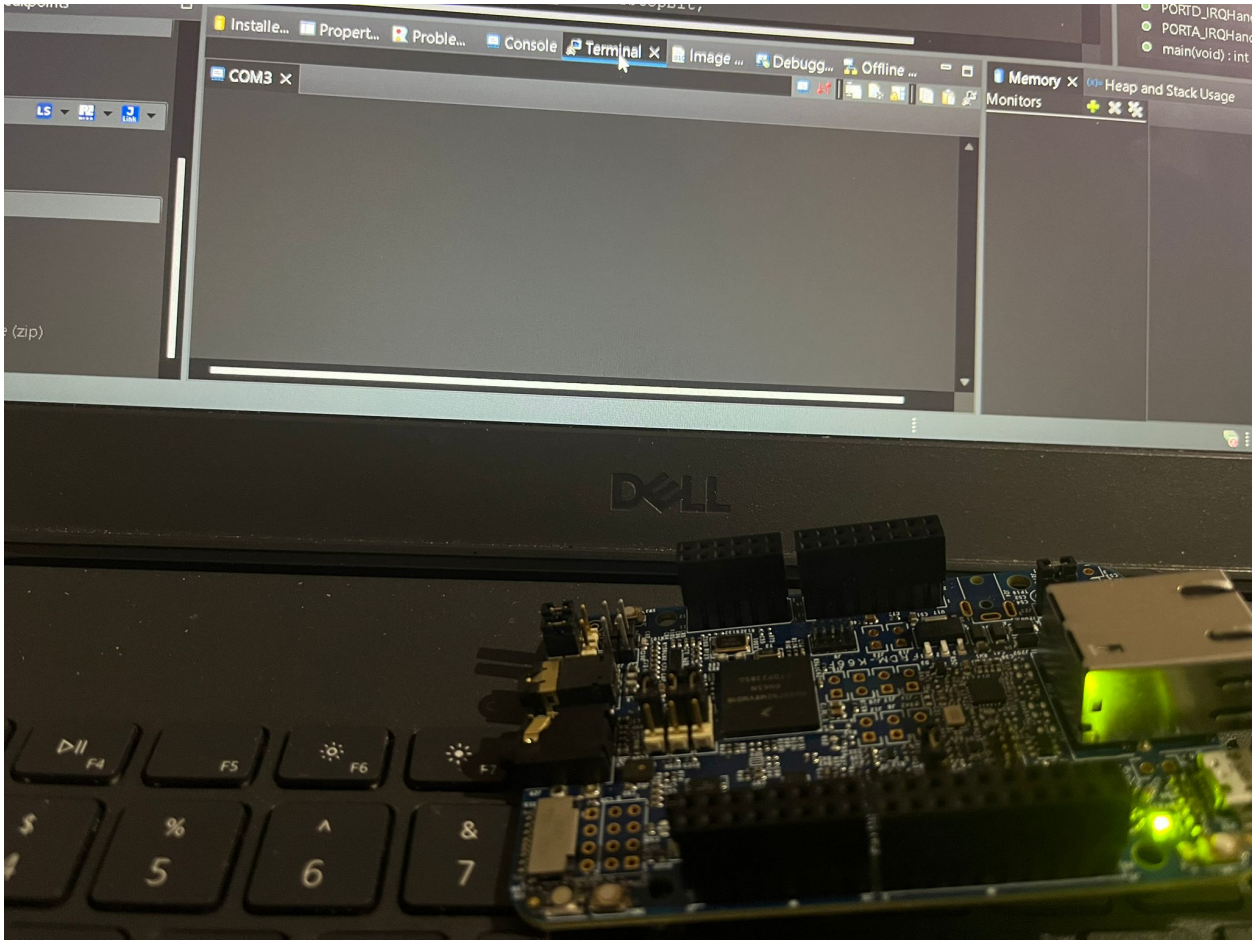
For the interrupt part, we started following the steps we were given in our Lab 7 document for setting up our interrupt handler but when we went to the config tools overview tab we saw that **some of the functionals groups were missing like BOARD_InitBUTTONsPins, BOARD_InitLEDsPins and BOARD_InitDEBUG_UARTPins.** For these files we went back to our Lab 7 assignment and started searching for these configurations and where they are

initialized. After a while, we found these configurations initialized in pin_mux.h and pin_mux.c files. We added these configs to our project's pin_mux.h and pin_mux.c files and added these functional groups manually to the functional group list in the config tools overview tab. After this we were able to add **two interrupt handlers** to our code which we are using to start our application using green LED as an indicator(SW2) and the second interrupt handler we are using shut down our application gracefully(SW3). **(a working example of our application is given below)**

When SW2 is pressed:



When SW3 is pressed:



Future Work:

Our project can be said to be a prototype for several future projects. As I mentioned, several other systems are already in place for earthquake detection and action. Using an accelerometer alone won't be sufficient. To enhance the capability of the detection system we can use the real-time data ability with other existing systems. For instance, an accelerometer-based system lacks coverage and depth assessments compared to other systems, hence we can use an accelerometer in conjunction with the seismic network which provides high coverage and we can also use the GPS/InSar system which provides in-depth analysis. Therefore we get the best of both worlds.

Using Accelerometer data from SmartPhones/IoT devices

This can also act as a prototype to explain how integrating Accelerometers into various devices, including smartphones and IoT devices be useful.

Integration into Smartphones:
Many modern smartphones already contain built-in accelerometers used for various functionalities like gaming, and fitness tracking. We can access these embedded sensors through software interfaces (APIs) provided by the device's operating system.

Then we can create a software application that can utilize the accelerometer data. Of course, we would have to encourage users to opt into earthquake detection features within applications, thereby giving us an extensive collective sensor network.
This approach significantly expands the coverage of earthquake detection systems and can potentially us with more extensive and real-time monitoring capabilities.

Integration into IoT devices:
We can also develop or integrate accelerometers in IoT devices such as smart home devices, wearables, environmental sensors, or in infrastructures themselves. And utilize the Wifi or Bluetooth features these devices provide to transmit data. This data can be sent to a centralized monitoring system where it can be analyzed.

Deploying these IoT devices in various locations can create a distributed network for comprehensive coverage.

All of this data from multiple IoT devices can be used to analyze trends, identify anomalies, and issue alerts if seismic activities are detected.

REFERENCES

[1]Atmos, “Earthquakes in a Warming World,” *Atmos*, Sep. 07, 2023. https://atmos.earth/earthquakes-in-a-warming-world/#:~:text=Today%2C%20earthquakes%20are%20becoming%20more

[1]“Networks of multiple seismometers are necessary to adequately monitor volcanoes | U.S. Geological Survey,” www.usgs.gov. https://www.usgs.gov/programs/VHP/networks-multiple-seismometers-are-necessary-adequately-monitor-volcanoes#:~:text=A%20seismometer%20is%20an%20instrument

Donnellan, Andrea & Lyzenga, Gregory & Peltzer, Gilles & Webb, Frank & Heflin, M. & Zumberge, James. (1999). Use of GPS and InSAR technology and its further development in earthquake modeling.

MCUXpressoIDE_11.8.0 SDK version 2.11.0 ecompass_peripheral