Title

April 23, 2025

Introduction

Text

Contents

Features

Text

Problem Statement

Text

Problem Solution

Text

Status

Text

Lessons Learned

Text

Results

Text

Future Work

Text

Conclusion

Text

Proper References

Text

Proper Images

Text

Code in Appendix

Text

Overview

The balloon project was created in VSCode using the PlatformIO (PIO) extension along with LaTeX using the Strawberry Perl and MaTeX extensions for live rendering of LaTeX PDFs. There are two primary coding languages involved in this project. The first is C++, which was used to write the codebase for the Adafruit Clue nRF52840 microcontroller. This board utilizes the BMP280 and LSM6DS33 sensors to measure altitude, pressure, and temperature, while the LSM6DS33 also provides accelerometer data to calculate the rate of change in acceleration, also known as jerk.
There are two primary objectives of this project:

1. Measure the rate of change of acceleration (jerk) using the LSM6DS33 accelerometer.

2. Compare the temperature, pressure, and altitude data from the BMP280 to the ideal standard atmospheric data.

Python was used to write the post-processing scripts, which output standard statistical measurements such as mean, standard deviation, variance, and others. This portion of the code also generates various graphs showing relationships such as time vs. altitude, time vs. jerk, and altitude vs. acceleration.
Regarding the Clue board itself, when the chip detects a change in acceleration, it writes data to a CSV file and outputs different LED light colors based on jerk values. The C++ portion can be described as the **Data Acquisition** portion, and the Python portion can be described as the **Post-Processing** portion.
The board contains onboard storage (2MB QSPI Flash), and outputs include the NeoPixel RGB LEDs and data being written to QSPI Flash. An additional file required is the CircuitPython driver `adafruit-circuitpython-clue_nrf52840_express-en_US-9.2.7.uf2`, which is used to access the data stored on the Flash.

Code Documentation

1.1 Codebase Structure

The codebase is organized into several folders, each serving a specific purpose. The main folder structure is as follows:

- Balloon/
  - Data/
    - live_serial_monitoring.py
    - post_data_processing.py
    - static_data_test.csv
    - dynamic_data_test.csv
    - README.md
  - Documentation/
    - LaTeX/
      - docs.aux
      - docs.fdb_latexmk
      - docs.fls
      - docs.log
      - docs.pdf
      - docs.synctex.gz
      - docs.tex
      - README.md
    - changelog.txt
    - README.md
  - Images/
    - image.png
    - README.md
  - src/
    - main.cpp

    - testing.exe
    - README.md
- Utilities/
    - adafruit-circuitpython-clue_nrf52840_express-en_US-9.2.7.uf2
    - README.md

1.2 src

1.2.1 main.cpp

The src folder contains the main.cpp file, which serves as the core of the program that operates the Clue board. At the beginning of this file, seven external libraries are included to support sensor readings, LED control, communication, and data logging:

1. **Adafruit_LSM6DS33.h:** Used to collect data from the LSM6DS33 6-DOF accelerometer and gyroscope.
2. **Adafruit_BMP280.h:** Used to read barometric pressure and temperature data from the BMP280 sensor.
3. **Adafruit_NeoPixel.h:** Used to control the onboard NeoPixel RGB LEDs.
4. **Wire.h:** Enables I^2C communication for interfacing with the sensors.
5. **Adafruit_SPIFlash.h:** Provides access to the QSPI flash memory used for onboard data storage.
6. **SdFat.h:** A lightweight FAT filesystem library used to write sensor data to the flash storage.
7. **Adafruit_FlashTransport.h:** Handles low-level data transport between the processor and the flash memory.

Function Prototypes

The next section in main.cpp is the **Function Prototypes** header. This section declares all the functions before they are used in the setup() and loop() functions. The purpose of doing this is to keep setup() and loop() near the top of the file for easier readability and quick access.

Global Objects and Variables

In the next section of `main.cpp` is the **Global Objects** header, which contains all global objects used throughout the file. Most of these are sensor objects declared as variables so they can be referenced later in the program. This section also includes the creation of the datalogging CSV file. This is where future sensor data will be written during operation.

Following that is the **Global Variables** header. This section defines the acceleration array used to store a four-point history of acceleration data. This historical data is required to compute the jerk, which is the rate of change of acceleration. The array stores acceleration values in the X, Y, and Z directions. In the next section of the code, the corresponding jerk values for each axis are calculated and stored.

Another key item defined in this header is the internal clock. This is an `unsigned long` variable, capable of storing large non-negative values, and is used to keep track of elapsed time on the board in milliseconds.

setup()

The next section of the `main.cpp` file is the `setup()` function. This is where the I^2C bus routes, the BMP280, the LSM6DS33, and the NeoPixel are all initialized. The accelerometer's range and data rate are configured, and the NeoPixel is started with no LEDs turned on. At the beginning of the function, `Serial.begin(115200)` is called to set up the serial communication between the board and the computer. This is used for printing data for debugging or monitoring. The portion of code that is commented out is meant for live serial data monitoring using the VSCode PlatformIO Serial Monitor. By keeping this section commented out, it allows the program to log data directly to the flash storage instead.

After setting up the serial communication, the I^2C bus and sensor communication channels are initialized to ensure proper data flow between the sensors and the board. The code includes sensor checks that confirm whether the LSM6DS33, BMP280, NeoPixel, and QSPI flash have been successfully initialized. If any of these checks fail, a warning message is printed to indicate the lack of communication. Additionally, the FAT filesystem is mounted on the QSPI flash. Since the Clue board lacks the ability to delete existing data manually, the `setup()` function clears any previously stored data to ensure that there is sufficient space for a new recording session. This is important for avoiding data loss during the final experiment.

The final portion of the `setup()` function writes the CSV header row to the flash file. This header includes the time, pressure, altitude, temperature, acceleration, and jerk, and sets up the file structure for consistent logging in the `loop()`.

loop()

Now analyzing the `loop()` function, this is the core of the program. The `loop()` continuously updates the accelerometer data, computes the jerk, and controls the NeoPixels based on those jerk values. When live serial monitoring is enabled, the function logs data to the Serial Monitor in real-time. Otherwise, it records data to flash storage at a frequency of 1 Hz (once every second).

To prevent rapid or excessive LED color changes, a threshold of 1.0 m/s^3 was introduced. Only when the computed jerk exceeds this threshold does the NeoPixel LED change color.

At the beginning of this function, several `unsigned long` variables are declared, specifically `const` and `static` types. The `const` variable sets a fixed time interval for updates, while the `static` variables serve as persistent time trackers within the loop. These are used to implement internal timing for jerk calculations.

The next section of the loop retrieves altitude, pressure, and temperature data from the sensors. These values are then stored into four-point history buffers used for jerk computation. Because the data is sampled at 1 Hz, the program waits at least four seconds to collect four data points before attempting any jerk calculation.

Once enough data is available, the `computeJerk()` function is called to calculate the rate of change of acceleration in the X, Y, and Z directions. If the computed jerk in any axis exceeds the 1.0 m/s^3 threshold, the NeoPixel LED is activated and displays a color corresponding to the axis in which the threshold was surpassed.

Jerk thresholds:

Axis	**Jerk < 0**	**Jerk > 0**
X	Blue	Red
Y	Yellow	Green
Z	White	Purple

If no jerk value in the X, Y, or Z direction exceeds the threshold, the LED turns black to indicate an off state. After this, the collected data, including the internal clock, altitude, pressure, temperature, acceleration (X, Y, Z), and jerk (X, Y, Z), is printed to the CSV file.

scanI2Devices()

Beginning with the utility function `scanI2Devices()`, this function scans the I^2C bus for connected devices and enables proper communication with them. It begins by checking for the presence of I^2C devices at various addresses. If no error is returned (error code 0), an I^2C device is found and can be communicated with. If the error code is 4, it indicates an unknown address or communication failure. If the variable `nDevices` is 0, it means no I^2C devices were detected on the bus.

getAltitude()

The next utility function is `getAltitude()`. This function reads the current altitude from the BMP280 barometric pressure sensor. It calculates altitude using a hardcoded reference sea level pressure of 101,325 Pascals, which represents the ideal atmospheric pressure at sea level. This assumption allows the function to convert the measured pressure into an estimated altitude above sea level using the ideal standard atmosphere calculations.

getAcceleration()

The next utility function is `getAcceleration()`, which reads acceleration data from the LSM6DS33 sensor. It captures the current acceleration in the X, Y, and Z directions and updates the corresponding global variables. These values are then

updateAccelHistory()

The next function is `updateAccelHistory()`, which manages the four-point acceleration history used for jerk calculation. It shifts the previously stored acceleration values back by one index, making room for the newest value. The most recent acceleration reading is then stored in the front of the array, ensuring that the history remains up to date for accurate derivative computations.

computeJerk()

The last utility function is `computeJerk()`, which calculates the jerk, defined as the rate of change of acceleration. This function uses the four-point acceleration history to compute the numerical derivative for each axis (X, Y, and Z). The resulting jerk values are then passed to the global variables, allowing the `loop()` function to make decisions based on sudden changes in motion, such as triggering LED responses.

1.3 Data

1.3.1 `live_serial_monitoring.py`

The purpose of this file is to enable live debugging of the microcontroller. It monitors the serial output from the board, captures the data in real time, and converts it into a CSV format for easier analysis. This script is primarily used to test the functionality of the code during development and ensure that sensor readings and calculations are being processed correctly.

1.3.2 `post_data_processing.py`

This file is used to analyze the data after the experiment has concluded. It begins by importing the CSV file that contains all recorded data. Using the Python library `pandas`, the script performs statistical analysis by calculating key metrics such as mean, standard deviation,

variance, and other relevant statistical properties.

Following the statistical analysis, the script runs a standard atmosphere calculator to generate ideal atmospheric values based on sea level conditions. These values are hardcoded using reference pressure and temperature at sea level. To make accurate comparisons between measured and theoretical data, the values must be adjusted using known conditions from the actual testing location. The standard atmosphere calculator facilitates this comparison between ideal and experimental data.

After this, the script generates a series of plots. Each plot overlays smoothed data on top of the raw data, which is displayed in a greyed-out background. The smoothing is performed using a Gaussian filter from the `scipy` module to reduce noise and highlight trends. For plots where smoothing is less effective, scatterplots are used instead to better visualize the raw data.

1.4 Documentation

1.4.1 `changelog.txt`

This file is used to document changes made throughout the codebase. It serves as a changelog, keeping track of updates, bug fixes, and feature additions. These updates are reflected in the version history and are synchronized with the repository on GitHub.

1.4.2 `LaTeX`

This folder is used to store the engine files required to render LaTeX documents as PDFs. Files such as `docs.aux`, `docs.fdb_latexmk`, `docs.fls`, `docs.log`, and `docs.synctex.gz` are automatically generated during the LaTeX compilation process and should not be modified. These engine files are responsible for compiling the LaTeX source into a live-updating PDF.

The two important files in this folder are `docs.tex` and `docs.pdf`. The `docs.tex` file is the primary LaTeX source file where the report and documentation are written and modified. The `docs.pdf` file is the compiled output that reflects the current state of the LaTeX document.

1.5 Images

This folder is used to store all the graphs generated by the `post_data_processing.py` script. Any plot created during the post-processing phase is automatically compiled and saved to this directory. It serves as a centralized location for visual outputs, making it easy to access and include figures in documentation or reports.

1.6 Utilities

This file contains the driver used to extract flash data from the Clue board. The file `adafruit-circuitpython-clue_nrf52840_express-en_US-9.2.7.uf2` is dragged into the boot directory when the Clue board is connected and enters boot mode. Once loaded, it allows access to the internal storage, enabling the extraction of logged data from the device.

Hardware

The microcontroller used in this project is the **Adafruit Clue nRF52840**, which features a 64 MHz Cortex-M4 processor and 1 MB of flash memory for code storage.
Two primary sensors are integrated into the system:

- **BMP280** — used to measure altitude, atmospheric pressure, and temperature.
- **LSM6DS33** — used to collect three-axis accelerometer data.

The system can be powered either via a USB connection or a **3x AAA battery pack**.
For onboard storage, the project utilizes **2 MB of QSPI Flash**, which is formatted with a FAT32 filesystem to support file-based data logging.
The primary output device is the **NeoPixel RGB LED**, which visually responds to changes in jerk (rate of acceleration change).

Software Requirements

The software tools used in this project include **PlatformIO** for embedded development, **Python** for post-processing and analysis, **C++** for firmware programming, and **LaTeX** for technical documentation. Additionally, the **MaTeX** and **Strawberry Perl** extensions are used to enable live rendering of LaTeX documents within the development environment.
The following libraries are utilized within the PlatformIO project configuration (`lib_deps`) to support sensor integration, data storage, and peripheral control:

- `adafruit/Adafruit Unified Sensor`
- `adafruit/Adafruit BMP280 Library`
- `adafruit/Adafruit LSM6DS`
- `adafruit/Adafruit NeoPixel@^1.12.5`
- `adafruit/SdFat - Adafruit Fork@^2.2.54`
- `adafruit/Adafruit SPIFlash@^5.1.1`