**FINAL PROJECT REPORT**

**Weather Monitoring and Evaluation Station for Pilots**

Colleen Staeger

Making Embedded Systems Class

November 2022

1. **APPLICATION DESCRIPTION**

Pilots need to be continuously monitoring weather conditions and forecasts to make an informed decision about the safety of flying in a plane. Getting in a plane before a large storm increases the risk of crashes and fatalities even if a pilot is instrument rated and can fly without good visibility. My solution is a weather monitoring and evaluation station that takes in temperature, pressure, humidity data to forecast the weather within the next 24 hours. The system will produce a flying condition report that breaks down the “safety conditions” due to weather in the area.

The following report will break down the hardware and software components, the development and testing, as well as propose future work to be done on the system.

1. **HARDWARE DESCRIPTION**

The embedded system is developed on a STM32L4 Discovery Kit which has an IoT node to allow for BLE/Wi-Fi development. The microcontroller on the board is a …

The board has an HTS221 temperature and humidity sensor, and an LPS22HB pressure sensor that will be used to collect data for a forecasting algorithm.

1. **SOFTWARE DESCRIPTION**
   1. Software Block Diagram

Diagram

Description automatically generated

* 1. State Machine
  2. Software Licensing and Origins
* STM32CubeMX was used to generate HAL and LL peripheral initialization and driver code. The code is licensed by ST and/or Arm as indicated in the header file for each module.
* The console.c/consoleIo.c/consoleCommands.c is from Elecia White’s repo and was used with modifications to add more commands for my application and updating the ConsoleIO wrapper for my board.
  1. Software Modules (that I wrote or contributed to)
     + Forecast\_algo.c
     + System\_SM.c
     + LPS22\_driver.c
       1. Module that initializes the sensor, can get the sensor id, and can read the current barometric pressure.
     + HTS221\_driver.c
       1. Module that initializes the sensor, can get the sensor id, and can read the temp and humidity.
     + ConsoleCommands.c
       1. Added Commands to read current temperature, current humidity, and current pressure.
     + Stm3214xx\_it.c
       1. Button interrupt is implemented here. Pressing the user button makes LED2 on the blink turn off and on. It is also part of the state machine that causes the system to wake up/ then go back to sleep when pressed.

1. **BUILD INSTRUCTIONS**
   1. Code was built with IAR Embedded Workbench. All necessary EWARM build files are included. The .IOC file from STMCube is also included in the git repo and can be used to configure the project again.
2. **TESTING AND DEBUGGING**

Because the system needs to be outside to accurately forecast, I created a few test scenarios to check that the forecaster was working correctly. I pulled data off the NOAA website from different days in Minneapolis history during days when it snowed, rained, was foggy, and clear and sunny. The following shows the outcome of each test.

* 1. Regular Mode
     1. In regular mode (reading from incoming sensors and forecasting on the spot), the console will print out the current temp, humidity, and a few pressure values over 5 mins. (The system needs multiple pressure readings over time to forecast). At room temperature inside the weather report produced favorable conditions as shown below.

Text

Description automatically generated

* 1. Snowy Conditions -

Text

Description automatically generated

* 1. Rainy Conditions
  2. Sunny Conditions
  3. Foggy Conditions

1. **GRADING** 
   1. **Fulfillment of Requirements Breakdown**

The Following Items must be achieved:

* Use a Cortex-M processor
  + Using the STM32L4 Discovery Kit for IOT which has a STM32L4 Series based Arm Cortex -M4 core
* Have a button that causes an interrupt
  + Using a button on the discovery kit with an interrupt that controls power resources
* Use at least three peripherals (only used 2 in my project)
  + I2C connected to a HTS221 Temp and Humidity Sensor
  + I2C connected to a LPS22HB Barometric Pressure Sensor
  + UART used for the command line interface
* Have a serial port output
  + Includes a Command Line Interface for debugging/ testing and outputting data to the user
  + Builds upon the console example in class by adding debugging commands to look at current temperature, humidity, and pressure values
* Implement an interesting algorithmic piece
  + Forecasting Algorithm that breaks down weather conditions for the pilot
* Implement a state machine
  + State machine shown above and implemented in System\_SM.c

1. **FUTURE WORK**

Yeah, there is a lot of future work that still needs to go into this project to be useful. Here is a summary of a few that I would need to implement in the future.

* Weatherproof Container/Battery
  + Currently all my testing must be done inside but in the future the system should be enclosed in a weatherproof container and have a battery with ADC charger breakout board included.
* Add IOT
  + The B-L475-IOT01A2 STM32 Disco Board that I used has the capability for wireless communication already built in. Unfortunately, I ran out of time to fully implement it, but the system should be able to wirelessly send back data to be user-friendly and accessible.
* Add more sensors/ data-capture devices
  + I would love to implement a rain-gauge and an anemometer, and a wind vane to gather more data for my system. As of right now I am somewhat limited in the forecasting I can do with just pressure, temperature, and humidity devices.

1. **SELF-GRADING**

|  |  |  |
| --- | --- | --- |
| **Criteria** | **Score** | **Explanation** |
| Project Meets Minimum Project Goals | 1.5 | Only was able to implement two peripherals, didn’t have time to get to any of the IOT functionality on the disco board working. The State Machine is also very basic. |
| Completeness of Deliverables | 1.5 |  |
| Clear Intentions and Working Code | 1.5 |  |
| Reusing Code | 2 |  |
| Originality and Scope of Goals | 1.5 | I think that it is an original idea but needs a lot more work to be usable. |