Making Embedded Systems Course - Orange Stars cohort

Air Quality Monitor Project

13th June 2022

By snkYmkrct

MOTIVATION

From measuring CO2 levels in a crowded room for Covid-19 (and other respiratory infections) prevention, to tracking air pollution in the city, to checking if the soldering fume extractor is working properly, portable air quality monitors are useful tools to have around. And luckily, they are (relatively) cheap and easy to DIY.

The purpose of this project is to build one of these air quality monitors, using 2 sensors initially, a true CO2 sensor, and an air particulate sensor, with the possibility to add others in the future (like a volatile organic compounds sensor, etc).

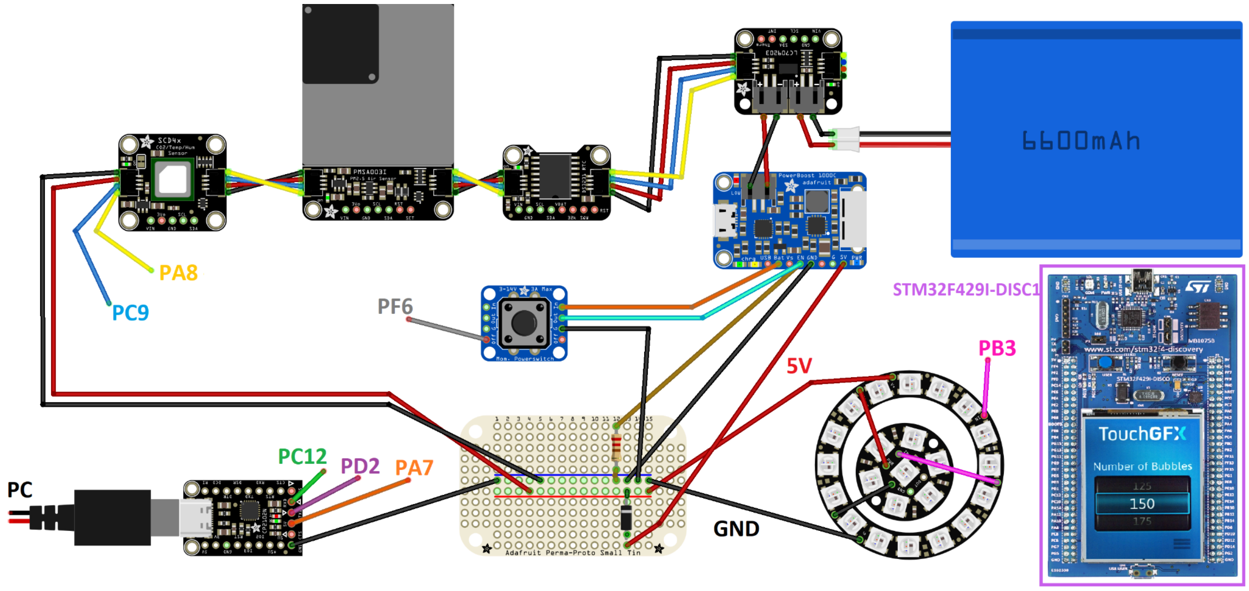
It was designed to be used as both a portable device on battery, and a room monitor, with the addition of a RGB lamp for distance notification.

HARDWARE

The 32F429IDISCOVERY (with the Cortex-M4 STM32F429ZIT6 microcontroller) class board was used for this project, with the addition of a few breakout boards from Adafruit.

- STM32F429I-DISC1 Discovery board - using the touch screen interface
- PMSA003I air particulate sensor breakout board
- SCD-40 Co2 sensor breakout board
- DS3231 Precision RTC breakout board
- Adafruit I2C LC709203F LiPoly / LiIon Fuel Gauge and Battery Monitor
- Adafruit CP2102N breakout board - USB to Serial Converter
- WS2812B (Neopixel) LED rings
- Adafruit Push-button Power Switch Breakout
- Adafruit PowerBoost 1000C Charger + Power supply
- Rechargeable LiPo Battery

Fritzing diagram



The Fritzing diagram shows the physical connections of the components. Since a Fritzing model could not be found for the main board, the image shows the pins to which the breakout boards are attached.

The I2C pins are PA8 (SCL) and PC9 (SDA). This is the interface used by both sensors, the RTC, and the battery level monitor.

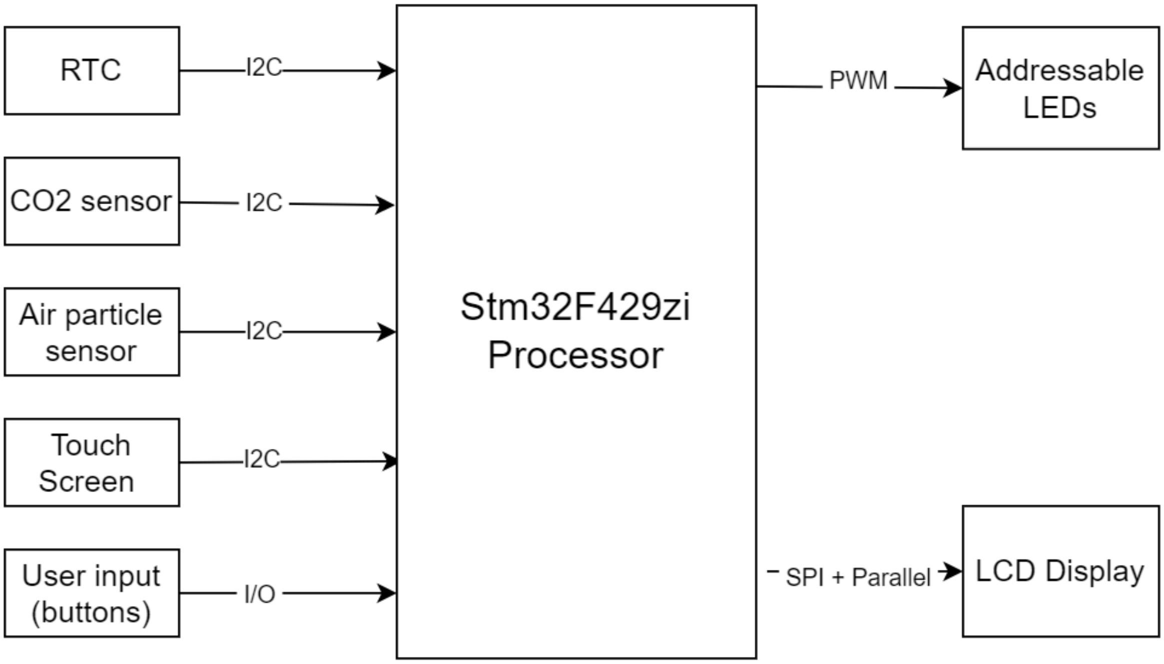
The Neopixel rings get the data on the GPIO pin PB3, using DMA.

The console communication to the PC happens through the CP2102N converter, using UART pins PC12 (Rx), PD2 (Tx), and PA7 (power pin from PC signaling that the serial cable is connected during functioning).

Since the Power Switch Breakout uses a P-FET to connect and disconnect the IN pin to the OUT pin, not a mechanical ON/OFF switch, it can receive auto shut off commands from the microcontroller. This is done through the board GPIO pin PF6.

The PowerBoost 1000C is used both for charging the battery, and for supplying 5V to the circuit, when enabled using the Power Switch.

Hardware Block Diagram



SOFTWARE

Toolchain

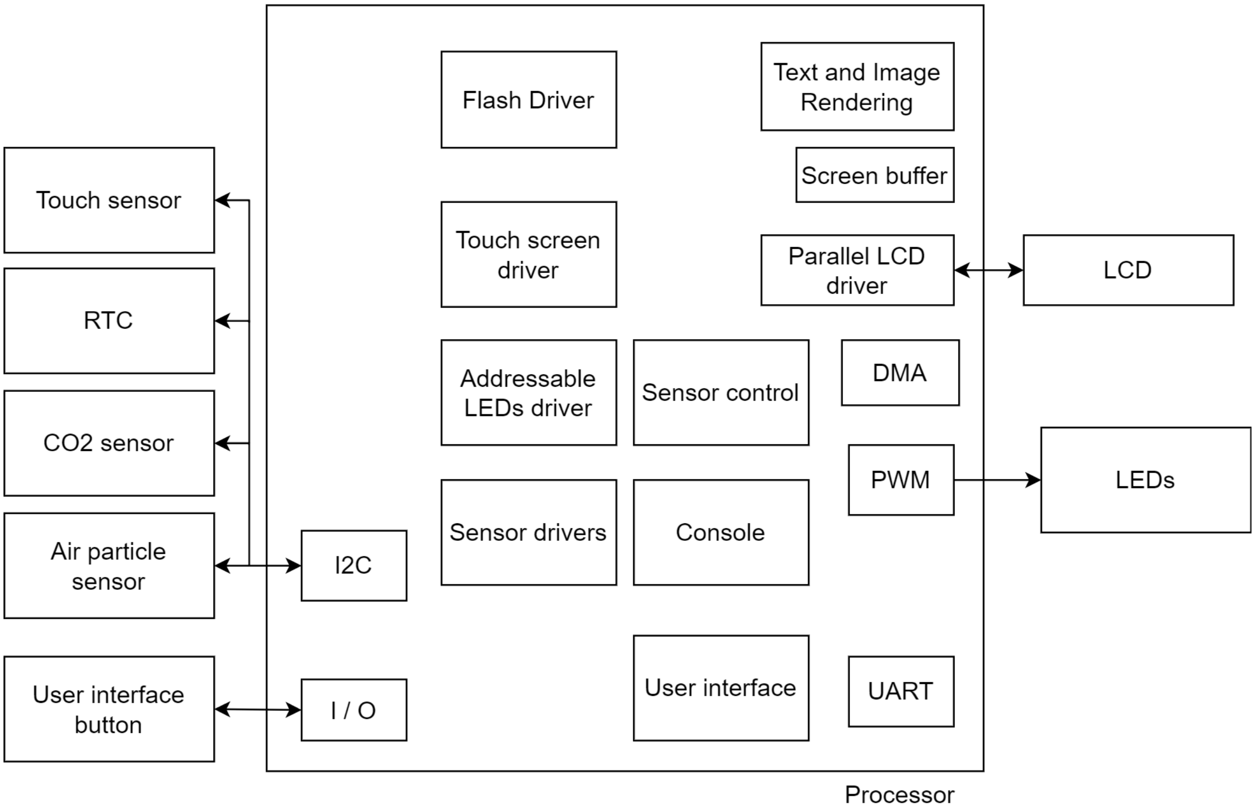
This project was developed on Windows 11, using STM32CubeIDE version 1.9.0, and TouchGFX Designer version: 4.19.1

TouchGFX Designer provides an excellent environment for developing graphical user interfaces on the Discovery board, and a very useful simulator.

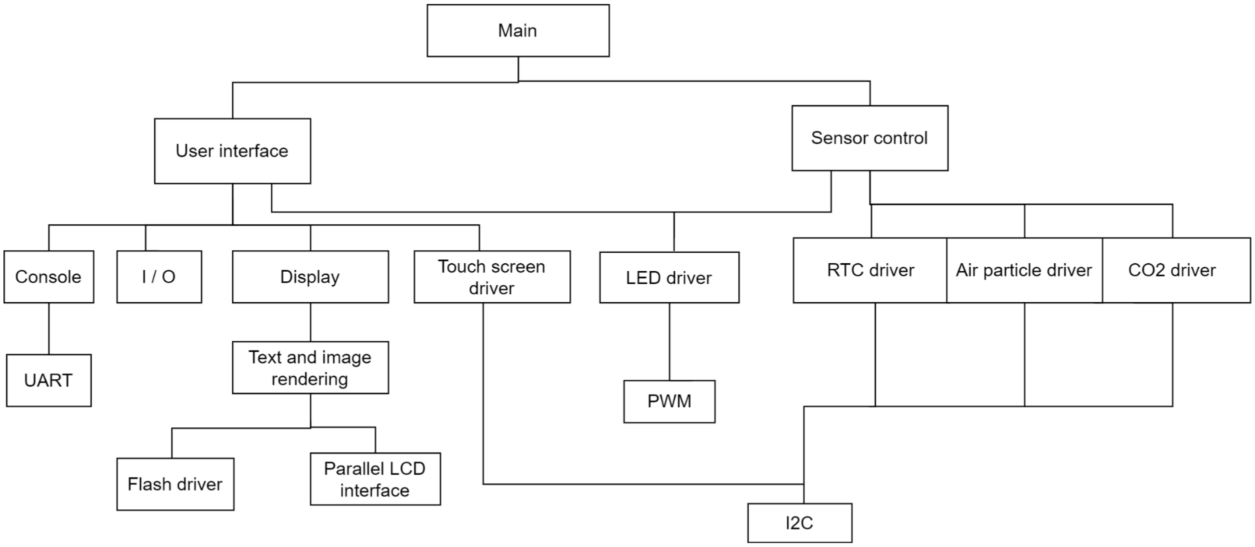
However, getting it to work seamlessly with CubeIDE is a bit of a complicated process, that was published earlier in the class, at this link:

https://github.com/snkYmkrct/Making_Embedded_Systems_Course/tree/main/TouchGFX%20%26%20STM32CUBE%20Example

Software Block Diagram



Software Hierarchy of Control



Mode of operation

The project is created to work as an independent device, powered by battery. It is currently in a temporary enclosure, but a much smaller custom design could be used. A charging port is provided to charge the battery, or to keep the device plugged in if desired.

It is powered on through a momentary push button connected to the Power Switch Breakout. The on-board FET will enable power to the PowerBoost, which in turn supplies 5V to the main microcontroller board, and the Neopixel rings.

The board starts and the GUI will show up on the display. After a few moments of initialization, the user can see the data values from the peripherals on the screen, updated periodically.

Different colors are used to show the data, making obvious what ranges the values from the CO2 and air particulate sensors are in.

The time from RTC is shown on a digital clock, and updated every second.

The battery levels are shown by displaying a different image for every 25% range.

Using the touch screen to press the right arrow button will take the user to a second screen, where the data is displayed in dynamic graphs, updated at the same rate as the first screen.

Further future development is planned for the GUI, like adding a screen for settings (where the user could turn the neopixels on and off, or set the RTC date and time), and multiple functioning modes.

In the main loop of the program, the sensor data is read periodically, stored in memory, and sent to the graphical interface as needed.

The Neopixels are also set to display colors based on the values read from the CO2 sensor. It can show either green, orange, or red, depending if CO2 values are in normal range, elevated, or very high.

The Neopixel lamp is attached through a magnetic connector, which means it can be removed, to reduce power consumption, making the project even more portable.

Another magnetic connector provides an interface to a USB data cable that can be used for serial communication with a computer. The connection can be detected reading a GPIO pin, and the on board green LED is turned on and off as indicator.

Pressing the on board user button causes an interrupt, that when the serial cable is connected, starts the console interface. This provides a series of commands that can be used to test and debug the system.

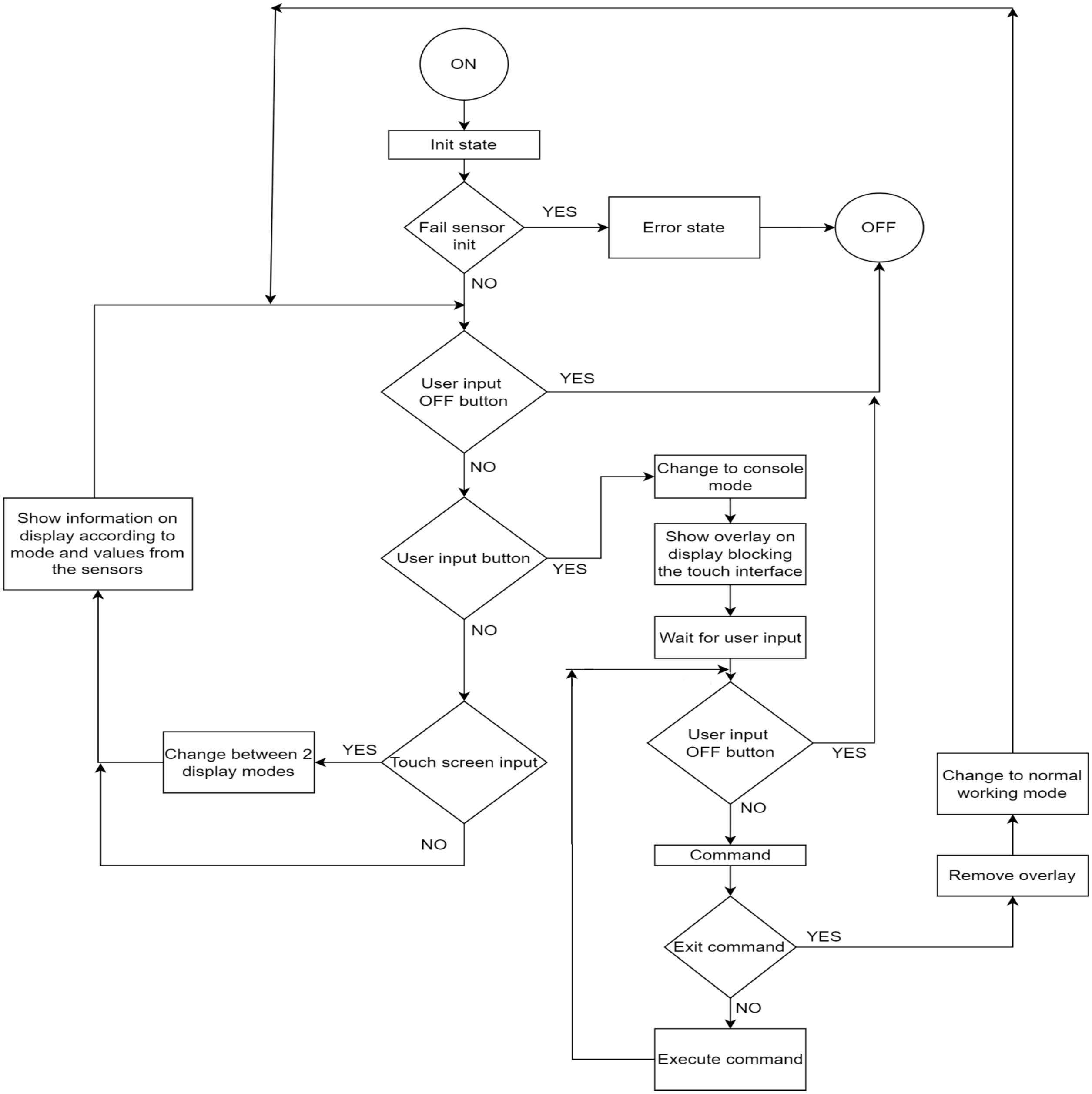
In console mode, the sensor data is updated only when a command is issued, the neopixels can be tested with different colors, and the GUI will display an overlay that blocks touch screen interfacing, and lets the user know the console is started.

Typing the exit command returns the system to the usual working mode.

Pressing the start / stop button again will power off the system, from any state.

Future development includes sending commands to auto shut off the system after a time period, to save battery.

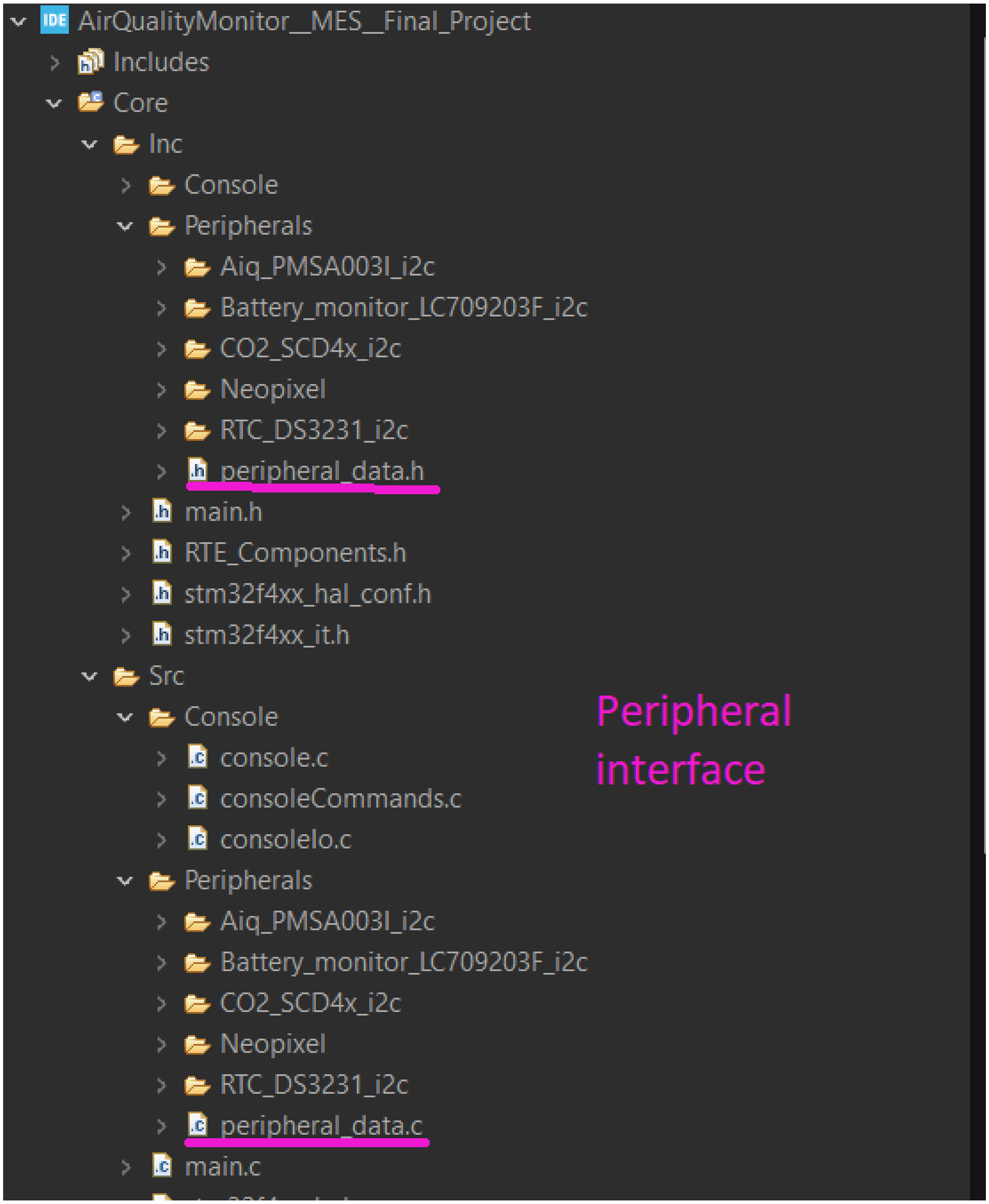
Flowchart state machine



The Code

Most of the code was generated using TouchGFX Designer and STM32Cube, and initializes most board peripherals to default values. The code doesn't use any RTOS, and relies on direct interaction with the ST HAL library.

As described in the link from the Toolchain section, ST BSP libraries were included for the LCD and touch controllers.



```
AirQualityMonitor_MES_Final_Project
  Includes
  Core
    Inc
      Console
      Peripherals
        Aiq_PMSA003I_i2c
        Battery_monitor_LC709203F_i2c
        CO2_SCD4x_i2c
        Neopixel
        RTC_DS3231_i2c
        peripheral_data.h
      main.h
      RTE_Components.h
      stm32f4xx_hal_conf.h
      stm32f4xx_it.h
    Src
      Console
        console.c
        consoleCommands.c
        consoleIo.c
      Peripherals
        Aiq_PMSA003I_i2c
        Battery_monitor_LC709203F_i2c
        CO2_SCD4x_i2c
        Neopixel
        RTC_DS3231_i2c
        peripheral_data.c
      main.c
```

Peripheral interface

The Console, and Peripheral controllers each have their own folder, appropriately named, and the main program accesses the data through an interface.

The Console controller is reused and adapted from the example provided in class.

The CO2 sensor library is provided freely by Sensirion, and can be used almost as is - with minor adaptations for the specific hardware.

The air particulate sensor, and the battery monitor drivers were ported and adapted from the corresponding Adafruit Arduino libraries. Credit is given in the code files.

The Neopixel and RTC drivers were written from multiple examples found online. Since none of the libraries tested worked as-is, the code had to be heavily modified and simplified for the purpose of this project.

The system settings for the Neopixels had to be precise, to obtain the correct signal frequency. For a 72Mhz clock, a timer with 0 prescaler, 90-1 counter, PWM output generation was used, and the data sent with the help of DMA.

The HAL functions for I2C (sensors), UART (console), and DMA (Neopixels) communication were used in the written / adapted code.

Reimplementing __io_putchar() and __io_getchar() allowed for serial communication to be done with regular calls of printf and scanf.

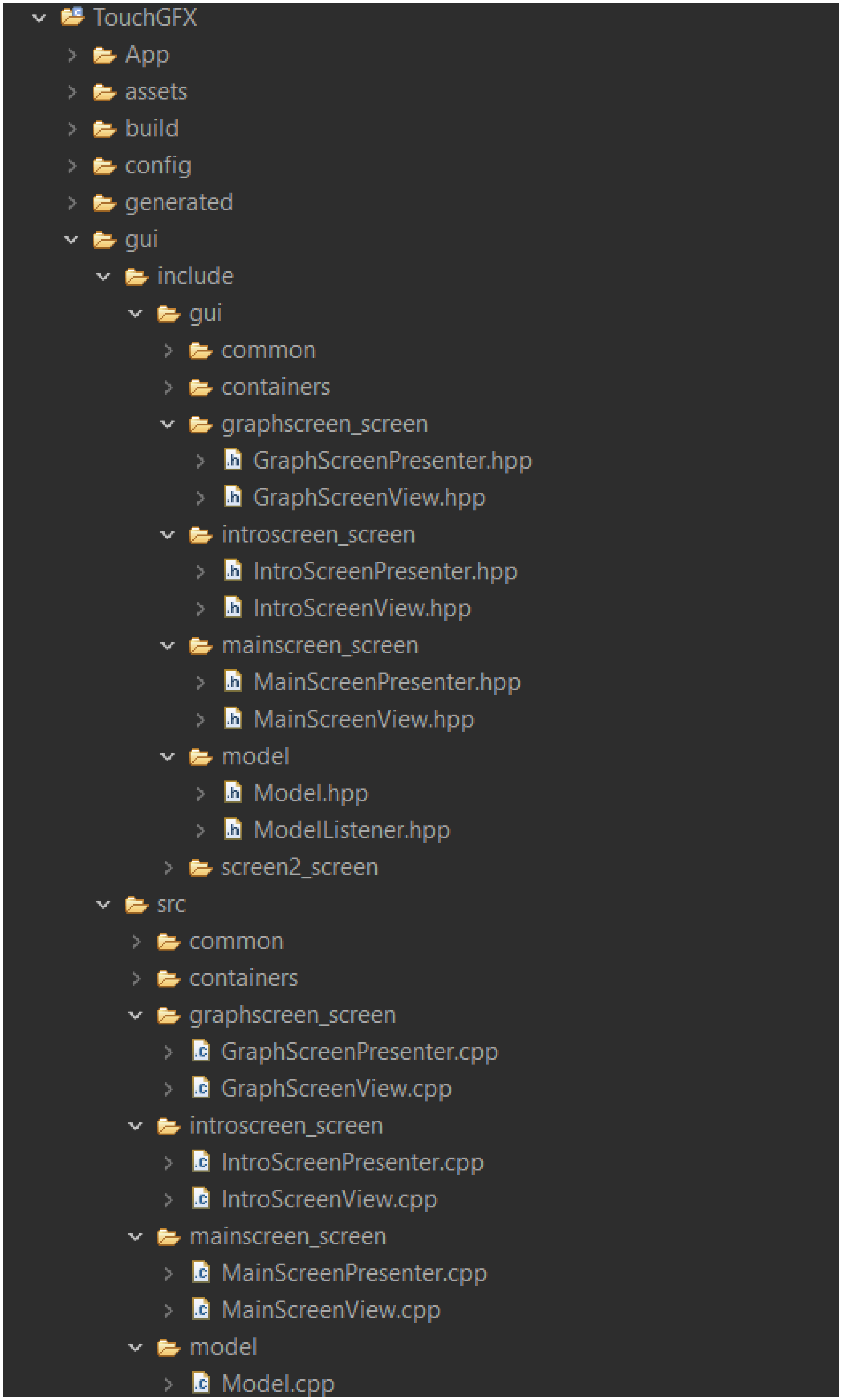
Pressing the on board user button generates an interrupt that sets a global variable, and the console-started flag.

The main loop checks for the console-started flag, and runs either the console process, or the process to update the data from the sensors.

The modified code in the main.c file will be between /* USER CODE BEGIN */ and /* USER CODE END */ comments that stop it from being deleted during automatic code generation.

For the GUI, the TouchGFX provided Model-View-Presenter (derivation of the MVC) pattern structure was used. The Model takes the data from the peripherals through the implemented interface, and notifies the Presenters, so the information displayed in the Views is updated.

The icons used in the UI were taken from https://www.flaticon.com/ free license, with credit.

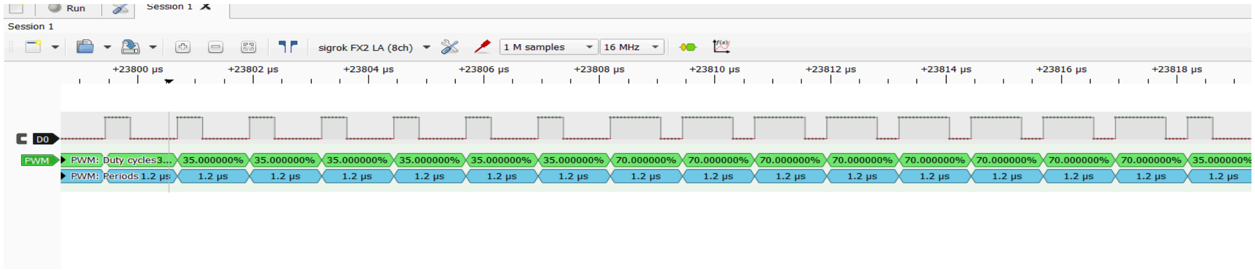


```
TouchGFX
  App
  assets
  build
  config
  generated
  gui
    include
      gui
        common
        containers
        graphscreen_screen
          GraphScreenPresenter.hpp
          GraphScreenView.hpp
        introscreen_screen
          IntroScreenPresenter.hpp
          IntroScreenView.hpp
        mainscreen_screen
          MainScreenPresenter.hpp
          MainScreenView.hpp
        model
          Model.hpp
          ModelListener.hpp
        screen2_screen
    src
      common
      containers
      graphscreen_screen
        GraphScreenPresenter.cpp
        GraphScreenView.cpp
      introscreen_screen
        IntroScreenPresenter.cpp
        IntroScreenView.cpp
      mainscreen_screen
        MainScreenPresenter.cpp
        MainScreenView.cpp
      model
        Model.cpp
```

Testing and debugging

STM32CubeIDE provides a debugging interface over the integrated ST-Link.
Serial prints, and sending commands over UART were also used for testing.

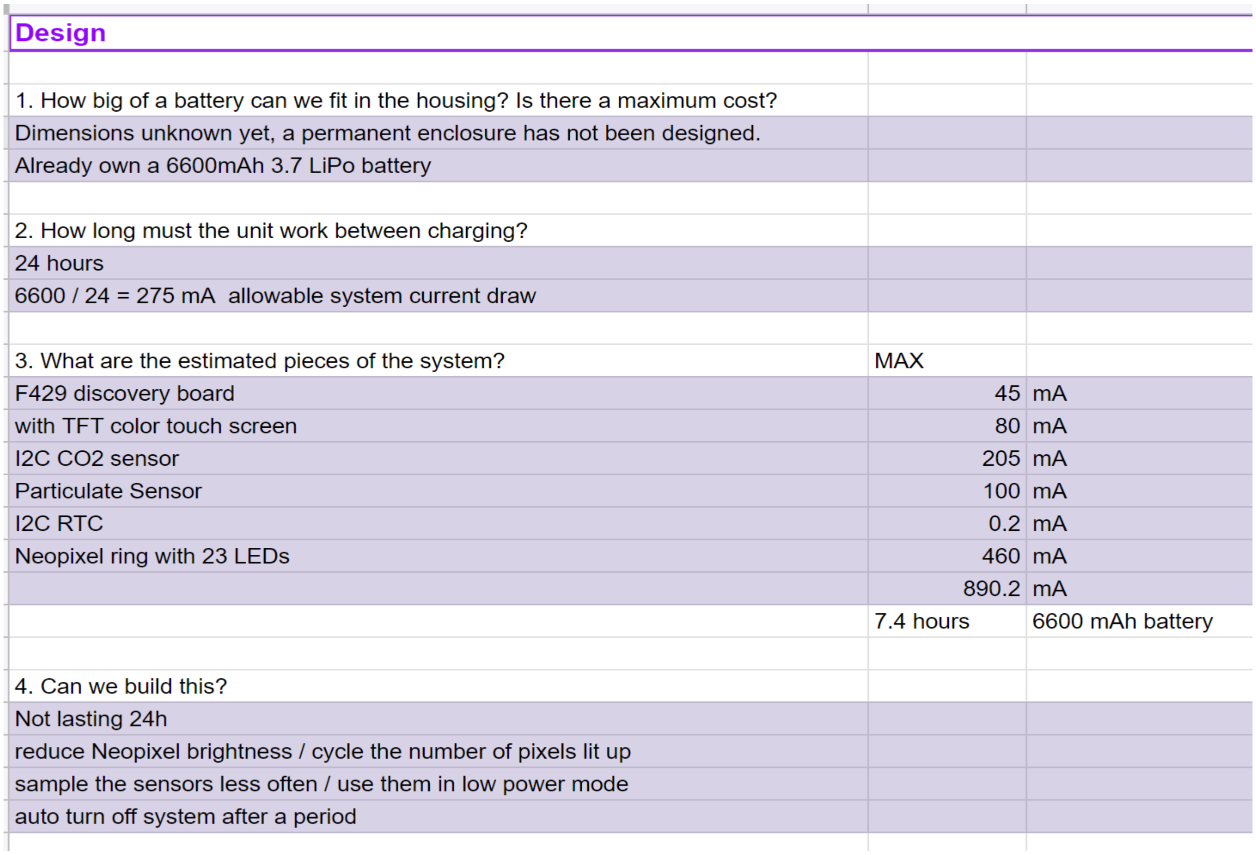
Debugging the Neopixel code was done also using a logic analyzer, to make sure the signal had the correct frequency, and the LEDs were getting the correct data.



SYSTEM POWER USAGE

Estimated

During the Week 10 class exercise, this was the estimated power consumption.

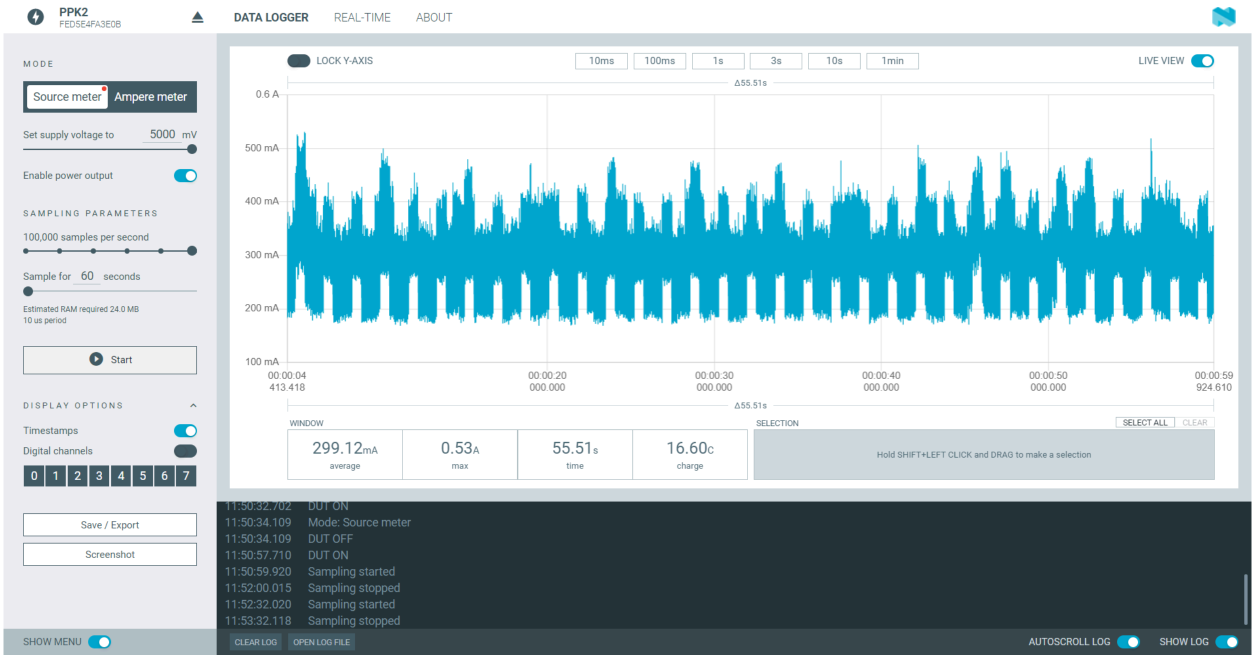


Design		
1. How big of a battery can we fit in the housing? Is there a maximum cost?		
Dimensions unknown yet, a permanent enclosure has not been designed.		
Already own a 6600mAh 3.7 LiPo battery		
2. How long must the unit work between charging?		
24 hours		
6600 / 24 = 275 mA allowable system current draw		
3. What are the estimated pieces of the system?	MAX	
F429 discovery board	45	mA
with TFT color touch screen	80	mA
I2C CO2 sensor	205	mA
Particulate Sensor	100	mA
I2C RTC	0.2	mA
Neopixel ring with 23 LEDs	460	mA
	890.2	mA
	7.4 hours	6600 mAh battery
4. Can we build this?		
Not lasting 24h		
reduce Neopixel brightness / cycle the number of pixels lit up		
sample the sensors less often / use them in low power mode		
auto turn off system after a period		

Actual

Using the Power Profiler Kit II, a graph of the power consumption over time can be created.

This is a 1 minute sample:



Set supply voltage to 5000 mV

Enable power output

100,000 samples per second

Sample for 60 seconds

Estimated RAM required 24.0 MB
10 us period

Timestamps

Digital channels

299.12mA average | 0.53A max | 55.51s time | 16.60C charge

```
11:50:32.702   DUT ON
11:50:34.109   Mode: Source meter
11:50:34.109   DUT OFF
11:50:57.710   DUT ON
11:50:59.920   Sampling started
11:52:00.015   Sampling stopped
11:52:32.020   Sampling started
11:53:32.118   Sampling stopped
```

The system is running with the Neopixels at lower brightness, and only certain colors are displayed, so the consumption is lower than the maximum estimated. They can also be completely removed, and since they are the largest power consumer, the battery life should be greatly improved.

But the system is still far from running for the desired period of time (at least 24h) on a single battery charge.

Other power reduction methods can be explored in future work, like lower sensor sampling rates, adding the possibility to turn off the LCD, and microcontroller sleep modes.

SELF ASSESSMENT

Criteria	Assessment
Project meets minimum goals	3
Completeness of deliverables	2.5
Clear intentions and working code	2.5
Reusing code	2.5
Originality and scope of goals	2
Power analysis	2.5
Version control was used.	3