АННОТАЦИЯ

Смирнова И.А., Варганов С.В. Разработка метеостанции с передачей параметров по беспроводному интерфейсу. – Челябинск: ЮУрГУ, КЭ–413; 2020, 79 с., 26 ил., библиогр. список – 3 наим., 1 прил.

В данном курсовом проекте был разработан проект метеостанции, измеряющий значения влажности, давления, температуры, точки росы, выводящий полученные значения на дисплей с электронными чернилами и передающий данные через Bluetooth. Данный проект был реализован на основе отладочной платы XNUCLEO–F411RE.

Требования к разработке:

1. Использование отладочной платы XNUCLEO–F411RE.

2. Программное обеспечение измерение должно измерять давление, влажность и температуру с помощью датчика BM280 и рассчитывать точку росы.

3. Вывод значений давления, влажности и температуры должен производиться на экран с жидкими чернилами.

4. Передача значений по беспроводному интерфейсу должна осуществляться через модуль BlueTooth Bee HC–06.

5. Разработка архитектуры в виде UML диаграмм в пакете Star UML.

6. Код на языке С++ с использование компилятора ARM 8.40.2.

7. По нажатию кнопки на плате XNUCLEO–F411RE изменение единиц измерения температуры F→K→C.

Разработка программного обеспечения проводилась в двух программных пакетах:

• StarUML – для разработки архитектуры;

• IAR Embedded Workbench 8.40.2 – для разработки кода на языке С++.

Пояснительная записка к курсовому проекту оформлена в текстовом редакторе MS Word 2016.

Оглавление

[ВведениЕ 4](#_Toc40784867)

[1 Анализ требований 5](#_Toc40784868)

[1.1 Использование отладочной платы XNUCLEO–F411RE. 5](#_Toc40784869)

[1.2 Программное обеспечение измерение должно измерять давление, влажность и температуру и рассчитывать точку росы. 5](#_Toc40784870)

[1.3 Вывод значений давления, влажности и температуры должен производиться на экран с жидкими чернилами. 7](#_Toc40784871)

[1.4 Передача значений по беспроводному интерфейсу должна осуществляться через модуль BlueTooth Bee HC–06. 9](#_Toc40784872)

[1.5 Разработка архитектуры в виде UML диаграмм в пакете Star UML 11](#_Toc40784873)

[1.6 Код на языке С++ с использование компилятора ARM 8.40.2. 11](#_Toc40784874)

[1.7 При разработке должна использоваться Операционная Система Реального Времени FreeRTOS и С++ обертка над ней 13](#_Toc40784875)

[1.8 По нажатию кнопки USER на плате XNUCLEO–F411RE единицы измерения температуры должны изменяться в следующей циклической последовательности F→K→C. 13](#_Toc40784876)

[2 Разработка общей архитектуры проекта 14](#_Toc40784877)

[3 Разработка детальной архитектуры проекта 17](#_Toc40784878)

[3.1 AbstractHardware 17](#_Toc40784879)

[3.1.1 SPI 17](#_Toc40784880)

[3.1.2 Pin 18](#_Toc40784881)

[3.1.3 USART 19](#_Toc40784882)

[3.2 Applications 20](#_Toc40784883)

[3.2.1 Button 21](#_Toc40784884)

[3.2.2 Output 22](#_Toc40784885)

[3.2.2.1 Bluetooth 22](#_Toc40784886)

[3.2.2.2 Display 24](#_Toc40784887)

[3.2.2.3 Format 28](#_Toc40784888)

[3.2.3 Sensor 29](#_Toc40784889)

[4 Результат создания по 34](#_Toc40784890)

[4.1 Создание интерфейса на телефоне 34](#_Toc40784891)

[4.2 Проверка работоспособности 34](#_Toc40784892)

[Заключение 35](#_Toc40784893)

[Библиографический список 36](#_Toc40784894)

[ПРИЛОЖЕНИЕ А 37](#_Toc40784895)

Введение

Основным направлением в работе кафедры ИнИТ на современном этапе ее развития является соединение в учебном процессе вычислительной и измерительной техники для глубокой и всесторонней подготовки студентов. Одним из предметов, изучаемых на 8 семестре, является «Программное обеспечение измерительных процессов».

Программное обеспечение – это совокупность программ, выполняемых на определенном компьютере [1, стр. 55].

Метеостанция – измерительный прибор или комплекс, осуществляющий контроль за показателями метеорологических измерений (наблюдений за погодой и климатом) в установленные сроки по единой методике в определённой последовательности, и передаче собранных данных в Гидрометцентр или иным потребителям.

Целью данного курсового проекта является разработка метеостанции, которая будет измерять температуру, давление, влажность воздуха и на основе этих показателей рассчитывать значение точки росы.

Задачи курсового проекта:

1. Проанализировать технические требования к разработке.
2. Создать общую архитектуру программного обеспечения метеостанции.
3. Создать детальную архитектуру программного обеспечения.
4. Реализовать утвержденную архитектуру в виде кода С++.
5. Провести испытание.
6. Анализ требований
7. Использование отладочной платы XNUCLEO–F411RE.

XNUCLEO–F411RE – это отладочная плата от компании Waveshare (рисунок 1). В основе платы ARM Cortex-M4 микроконтроллер STM32F411RET6. ARM Cortex-M это 32-разрядный процессор, построенный по гарвардской архитектуре с разделением шины данных и кода. Процессор оптимизирован для недорогих и энергоэффективных микроконтроллеров.

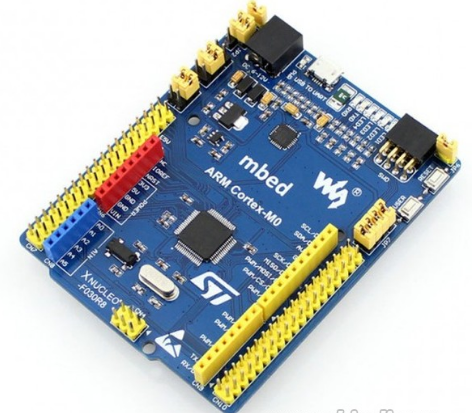


Рисунок 1 – Отладочная плата XNUCLEO–F411RE

Разъемы ST Morpho платы XNUCLEO-F411RE обеспечивают полный доступ к линиям портов ввода/вывода (I/O) и упрощают дальнейшее периферийное расширение.

К преимуществам данной платы можно отнести:

* arduino совместимость,
* подключение по USB,
* наличие MicroUSB порта,
* большое количество светодиодов, позволяющих отслеживать состояние платы,
* простое управление источником питания, выбором USART, пользовательской кнопкой и светодиодами с помощью джамперов. Преимуществом джамперов можно считать простоту в эксплуатации и отсутствие необходимости в пайке.

1. Программное обеспечение измерение должно измерять давление, влажность и температуру и рассчитывать точку росы.
2. Период измерения должен быть 100 ms.

Измерения, которые производимые датчиком, должны быть периодическими, с частотой раз в 100 миллисекунд. Для реализации данного требования нам потребуется Операционная Система Реального Времени (RTOS), суть которой будет описана далее.

1. Для измерения давления, влажности и температуры должен использоваться датчик BM280

Модуль BME280 (рисунок 2) предназначен для измерения атмосферного давления, температуры и влажности.

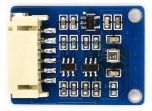


Рисунок 2– Датчик BME280

Датчик разработан Bosch Sensortec. По сравнению с первыми датчиками серии (BMP085 и BMP180) он имеет лучшие характеристики и меньшие размеры. Отличие от датчика BMP280 – наличие гигрометра, что позволяет измерять относительную влажность воздуха и создать на его основе маленькую метеостанцию.

Диапазон измерения температуры: -40~85°C (порог чувствительности 0.01°C, точность ±1°C).

Диапазон измерения влажности: 0~100% (порог чувствительности 0.008%, точность ±3%RH).

Диапазон измерения барометрического давления: 300~1100 кПА (порог чувствительности 0.18Па, точность ±1кПа).

1. Общение с датчиком должно осуществляться по интерфейсу SPIx (где х – любой не равный 1,2,3)

SPI (Serial Peripheral Interface) – последовательный синхронный стандарт передачи данных в режиме полного дуплекса, разработанный компанией Motorola для обеспечения простого и недорогого сопряжения микроконтроллеров и периферии. SPI также иногда называют четырехпроводным (англ. four–wire) интерфейсом. SPI является синхронным протоколом, в котором любая передача синхронизирована с общим тактовым сигналом, генерируемым ведущим устройством (процессором). Принимающая периферия (ведомая) синхронизирует получение битовой последовательности с тактовым сигналом. К одному последовательному периферийному интерфейсу ведущего устройства–микросхемы может присоединяться несколько микросхем. Ведущее устройство выбирает ведомое для передачи, активируя сигнал «выбор кристалла» (chip select) на ведомой микросхеме. Периферия, не выбранная процессором, не принимает участие в передаче по SPI.

Для подключения датчика был выбран интерфейс SPI2, Для его подключения потребуется обратиться к карте пинов, выбрать подходящие для использования с SPI4 и настроить их на нужный альтернативный режим.

1. Точка росы должна считаться на основе текущих показаний влажности и давления

Точка росы — это температура воздуха, при которой содержащийся в нём пар достигает состояния насыщения и начинает конденсироваться в росу.

Формула для приблизительного расчёта, где точка росы выражается через значения давления и влажности:

где – относительна влажность воздуха в процентах,

– температура в градусах цельсия,

– (постоянная) = 17,27,

– (постоянная) = 237,7.

1. Вывод значений давления, влажности и температуры должен производиться на экран с жидкими чернилами.

4.2inch e–Paper Module [B] (рисунок 3) – e–Ink дисплейный модуль с диагональю 4.2 дюйма, разрешением 400 х 300, со встроенным контроллером с SPI интерфейсом и поддержкой красного, черного и белого цветов свечения. Данный дисплей имеет такие преимущества, как ультранизкое энергопотребление, широкий угол обзора, хорошая контрастность даже без затрат электроэнергии делают дисплей хорошим выбором для таких приложений как электронные ценники, промышленные инструменты и т. д. Модуль не имеет задней подсветки и работает на отражение света как бумага. Дисплей характеризуется хорошей контрастность, в том числе при сильном (солнечном) освещении. Питание модуля 3.3 В – 5 В. Изделие можно подключить к таким популярным платам как Raspberry Pi/ Arduino/ Nucleo и многим другим, имеющим 3–проводной или 4–проводной SPI интерфейс.



Рисунок 3 – 4.2inch e–Paper Module

1. Общение с индикатором должно осуществляться через интерфейс SPI2

Для подключения дисплея через интерфейс SPI2 потребуется обратиться к карте пинов и выбрать подходящие для использования с SPI2 и настроить их на нужный альтернативный режим.

1. Период вывода информации на индикатор должен быть 3 секунды.

Для реализации данного требования будет использоваться система реального времени.

1. Формат вывода:

"Давление: " XXX.XX [Units]

"Влажность: " XXX.XX [Units]

"Температура: " XXX.XX [Units]

"Точка росы": " XXX.XX C"

Для правильного форматирования данных с последующим выводом на дисплей и передачи по беспроводному модулю Bluetooth, будет разработан отдельный класс, отвечающие за формирование формата данных.

1. Передача значений по беспроводному интерфейсу должна осуществляться через модуль BlueTooth Bee HC–06.

Bluetooth – открытый стандарт беспроводной связи с низким энергопотреблением, обеспечивающий передачу данных и звука между совместимыми устройствами.

В качестве модуля будет использован – Bluetooth Bee HC–06 (рисунок 4). Bluetooth Bee – это простой в использовании модуль Bluetooth SPP, совместимый с существующими разъемами Xbee, предназначенный для прозрачной настройки беспроводного последовательного соединения. Он может быть настроен на три основных режима Master, Slave или Loopback (HC–06 просто будет Slave).



Рисунок 4 – BlueTooth модуль

1. Для подключения модуля BlueTooth должна использоваться плата Accessories Shield или I/O Expansion Shield

Для реализации была выбрана плата Accessories Shield (рисунок 5).

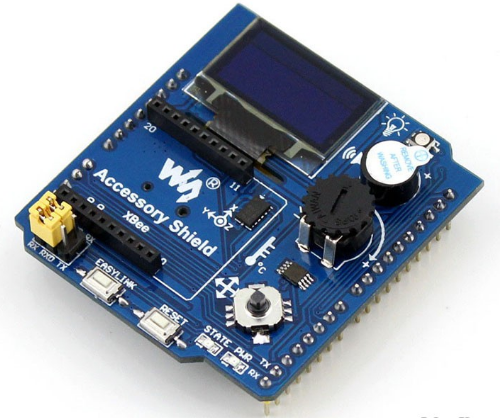


Рисунок 5 – Плата расширения Accessories Shield

Accessory Shield – это плата расширения, совместимая с популярными платформами для разработки электронных приложений, такими как Arduino UNO, Arduino Leonardo, NUCLEO, XNUCLEO. На плате установлены самые популярные дочерние модули, подходящие для реализации большинства задач.

Особенности:

– Разъем расширения для подключения плат Arduino;

– Разъем XBee для подключения беспроводных модулей;

– OLED дисплей 0.96–дюймов с разрешением 128x64 пикселей;

– RGB светодиод;

– Зуммер;

– Потенциометр 10кОм;

– 3–осевой цифровой акселерометр ADXL345;

– Температурный датчик LM75BDP;

– Джойстик 5–ти позиционный;

– Индикатор состояния XBee;

– Индикатор питания;

– Кнопка сброса модулей XBee и Arduino;

– Часы реального времени (RTC);

– Держатель батареи CR1220 для RTC;

– Драйвер RGB светодиода;

– Джампер выбора режима Отладка/Связь.

1. Период вывода информации через BlueTooth модуль должен быть 1 секунда.

Для реализации используем операционную систему реального времени.

1. Общение с платой расширения должно осуществляться через USART2

USART (Universal Synchronous–Asynchronous Receiver/Transmitter) – универсальный синхронно–асинхронный приёмопередатчик – аналогичный UART интерфейс, но дополнительно к возможностям UART, поддерживает режим синхронной передачи данных – с использованием дополнительной линии тактового сигнала.

1. Формат вывода:

"Давление: " XXX.XX [Units]

"Влажность: " XXX.XX [Units]

"Температура: " XXX.XX [Units]

"Точка росы": " XXX.XX C"

Для форматирования данных для передачи на дисплей и по Bluetooth будет использован единый класс преобразований.

1. Разработка архитектуры в виде UML диаграмм в пакете Star UML.

UML можно условно называть графическим языком, предназначенным для мо- делирования компьютерных программ. Под моделированием понимается созда- ние наглядной визуальной интерпретации чего-либо. UML позволяет создавать подобную интерпретацию программ высокоуровневой организации [2, с. 44].

StarUML – это проект с открытым кодом для разработки быстрых, гибких, расширяемых, функциональных и, главное, распространяемых бесплатно платформ UML/MDA для 32–разрядных систем Windows. Цель проекта StartUML – создание универсальной бесплатной платформы для моделирования, которая послужит аналогом для таких коммерческих проектов, как Rational Rose, Together и других.

UML – это постоянно расширяющийся стандарт, управлением которого занимается OMG (Группа Управления Объектами). StartUML поддерживает и будет поддерживать самые последние версии UML – как, к примеру, новый стандарт UML–UML 2.0, появившийся совсем недавно и уже поддерживаемый StartUML.

Простота использования является наиболее важной характеристикой в разработке приложений. Бесплатная платформа StarUML выгодно отличается от своих аналогов, в том числе и коммерческих, поддержкой множества особенностей, таких как быстрый диалог, управление с помощью клавиатуры, обзор диаграмм и многое другое. Кроме того, все эти дополнения понятны даже для неподготовленного пользователя.

1. Ко д на языке С++ с использование компилятора ARM 8.40.2.

IAR Embedded Workbench – Многофункциональная среда разработки приложений на языках C, C++ и ассемблере для целого ряда микроконтроллеров от различных производителей.

Основные преимущества пакета – дружественный пользовательский интерфейс и непревзойденная оптимизация генерируемого кода. Кроме этого реализована поддержка различных операционных систем реального времени и JTAG –адаптеров сторонних компаний.

В настоящее время IAR Embedded Workbench поддерживает работу с 8–, 16–, 32–разрядными микроконтроллерами от Atmel, ARM, NEC, Infineon, Analog Devices, Cypress, Microchip Technologies, Micronas, Dallas Semiconductor/Maxim, Ember, Luminary, NXP, OKI, Samsung, National Semiconductor, Texas Instruments, STMicroelectronics, Freescale, TI/Chipcon, Silicon Labs и Renesas. Для каждой платформы существует своя среда разработки, в частности ARM микроконтроллерам соответствует версия пакета IAR Embedded Workbench for ARM.

Программная среда включает в себя:

1. C/C++ компилятор – один из самых эффективных в своем роде. В нем также присутствует полная поддержка ANSI C.
2. Транслятор ассемблера, включающий в себя макроассемблер для программ реального времени и препроцессор для C/C++компилятора.
3. Компоновщик, поддерживающий более тридцати различных выходных форматов для совместного использования с внутрисхемными эмуляторами.
4. Текстовый редактор, настроенный на синтаксис языка Си и имеющий удобный пользовательский интерфейс, автоматическое выделение ошибок программного кода, настраиваемую инструментальную панель, подсветку блоков, а также удобную навигацию по именам подпрограмм, макросов и переменных.
5. Симулятор и отладчик в кодах Си и ассемблера. Отладчик позволяет просматривать области EEPROM, DATA, CODE, а также регистры ввода/вывода, устанавливать точки останова и аппаратные флаги, обрабатывать прерывания с предсказанием. Кроме этого, предусмотрен контроль стека и любых локальных переменных, режим пошагового выполнения программы. Тип отладчика и его настройки устанавливаются в свойствах проекта. Если отладчик отсутствует, то на помощь приходит симулятор, который, однако, не имеет возможности эмулировать работу процессора.
6. Менеджер проектов, облегчающий контроль и управление рабочими модулями.
7. Дополнительные утилиты для работы с оптимизированной CLIB/DLIB библиотекой.
8. При разработке должна использоваться Операционная Система Реального Времени FreeRTOS и С++ обертка над ней

Операционная система реального времени (ОСРВ, англ. real–time operating system, RTOS) — тип операционной системы, основное назначение которой — предоставление необходимого и достаточного набора функций для проектирования, разработки и функционирования систем реального времени на конкретном аппаратном оборудовании.

Под операционной системой реального времени понимается достаточно широкий набор программных инструментов прикладного и системного уровня, обеспечивающих работу других программ и библиотек, управляющих операциями ввода-вывода, принимающих, обрабатывающих и хранящих данные в различных форматах, взаимодействующих с пользователями различными способами, а главное, делающих это в таком масштабе времени, который удовлетворяет требованиям потребителя [3, с. 6].

Реальное время в операционных системах — это способность операционной системы обеспечить требуемый уровень сервиса в определённый промежуток времени.

FreeRTOS — многозадачная операционная система реального времени (ОСРВ) для встраиваемых систем. Портирована на 35 микропроцессорных архитектур.

1. По нажатию кнопки USER на плате XNUCLEO–F411RE единицы измерения температуры должны изменяться в следующей циклической последовательности F→K→C.

Для реализации данного требования используется Операционная система реального времени. Происходит опрос кнопки раз в 100 миллисекунд – так же, как и опрос датчика. За счет этого достигается эффект того, что при каждом измерении температуры значение измерения приходит в необходимых единицах измерения.

1. Разработка общей архитектуры проекта

Общая архитектура проекта представлена на рисунке 6.

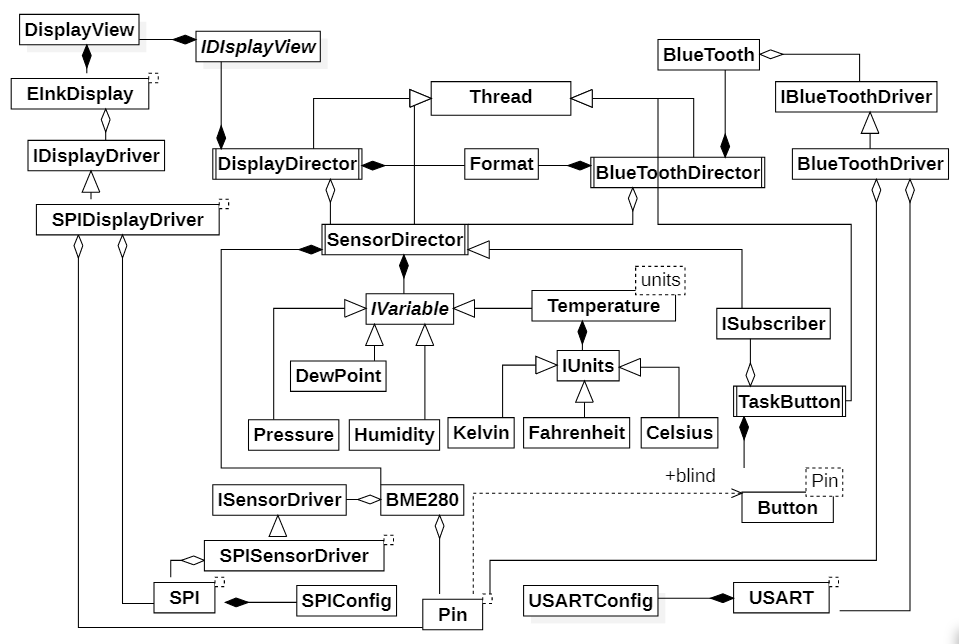


Рисунок 6 – Общая архитектура проекта

Для систематизации проекта, файлы были распределены по отдельным пакетам. Структура и содержание пакетов показано на рисунке 7.

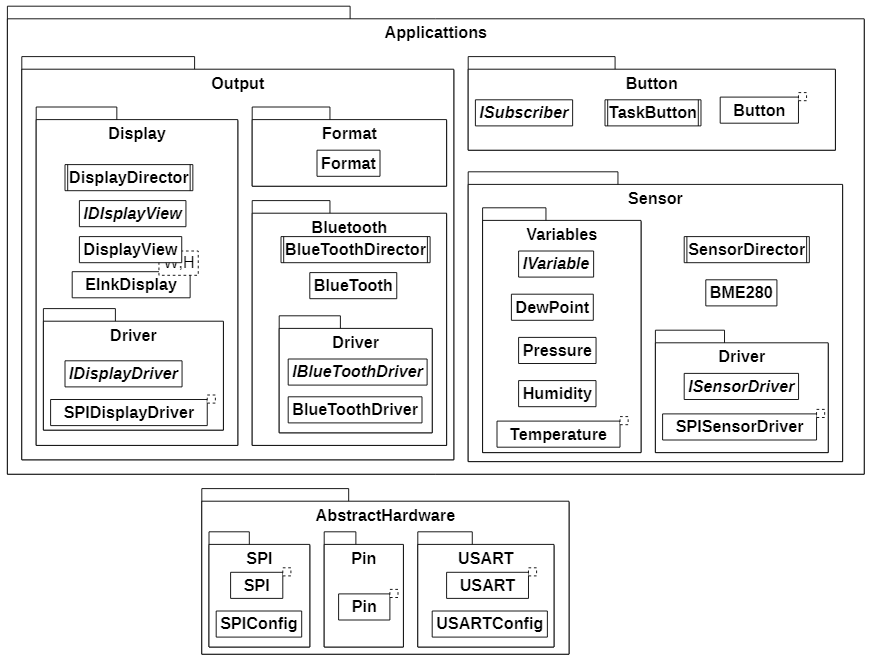


Рисунок 7 – Структура и содержимое пакетов проекта

Классы USART, SPI, PIN, отвечающие за аппаратную часть и использующие регистры микроконтроллера, расположены в пакете AbstractHardware. За универсальный синхронный асинхронный приемопередатчик отвечает класс USART. USART Config содержит структуру параметров для настройки USART. За настройку универсальных входов/выходов отвечает класс Pin. За последовательный периферийный интерфейс отвечает класс SPI. SPIConfig содержит структуру параметров для настройки SPI.

Классы работы с датчиком расположены в пакете Sensor. Для выполнения измерений используется датчик BME280. Для подключения датчика используются классы SPI и PIN. Настройка аппаратной части для общения с датчиком происходит в классе SensorDriver. ISensorDriver – интерфейс, служащий для унификации работы с драйверами. Класс, отвечающий за датчик – BME280. Код с датчика передается в класс задачи датчика SensorDirector. SensorDirector – активный класс, который раз в 100 мс. вызывает метод для обновления измерений, используя операционную систему реального времени FreeRTOS, за которую отвечает класс Thread.

Классы Temperture, Pressure, DewPoint и Humidity преобразуют входные данные, полученные от SensorDirector в требуемую величину, и возвращают кортеж со значением измеряемой величины и строкой, содержащей название единицы измерения. Классы измеряемых значений объединены в общий интерфейс IVariable.

Классы единиц измерения температуры, Kelvin, Fahrenheit и Celsius, объединены общим интерфейсом IUnits. В них значение, полученное от класса Temperature, преобразуется в требуемую величину и возвращается вместе со строкой, содержащей название единицы измерения температуры.

Классы, относящиеся к выводу данных, расположены в пакете Output.

Класс Format преобразует данные в формат, требуемый для вывода на дисплей и передачи через блютуз.

Классы работы с дисплеем расположены в пакете Display. Работа дисплея осуществляется через драйвер дисплея DisplayDriver. Для подключения дисплея используются классы SPI и PIN. Унификация работы с драйверами дисплея реализуется с помощью интерфейса IDisplayDriver. Класс EInkDisplay реализует работу с конкретным дисплей модулем. Класс DisplayView формирует вид картинки, выводимой на дисплее. Интерфейс IDisplayView позволяет унифицировать работу с видом дисплея. Активный класс DisplayDirector раз в три секунды обновляет дисплей, тем самым выводя на него значения, принятые от SensorDirector и преобразованные в классе Format.

Классы работы с блютузом расположены в пакете Bluetooth. Работа блютуза осуществляется через драйвер блютуза BlueToothDriver. Для подключения используются классы USART и PIN. Унификация работы с драйверами реализуется с помощью интерфейса IBlueToothDriver. Работа с конкретным Bluetooth модулем реализуется в классе Bluetooth. Активный класс BlueToothDirector передает данные, принятые от SensorDirector и преобразованные в классе Format, с частотой раз в секунду, используя ОСРВ Thread.

Классы работы с кнопкой расположены в пакете Button. Класс Button отвечает за состояние кнопки. Опрос состояния кнопки реализуется в активном классе TaskButton с помощью ОСРВ раз в 100 мс. При нажатии кнопки оповещается интерфейс ISubscriber, и у класса SensorDirector, наследующего данный интерфейс, вызывается метод, отвечающий за смену значений единиц температуры. Таким образом, по нажатию кнопки, единицы измерения температуры, отображаемые на дисплее и передающиеся через Bluetooth, должны изменяться в циклической последовательности F→K→C.

1. Разработка детальной архитектуры проекта
2. AbstractHardware

Пакет AbstractHardware содержит в себе классы, отвечающие за работу с аппаратной частью микроконтроллера и периферией (рисунок 8).

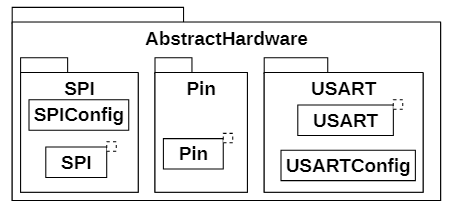


Рисунок 8 – Пакет AbstractHardware

1. SPI

Пакет SPI содержит в себе классы, требуемые для работы с интерфейсом SPI (рисунок 9).

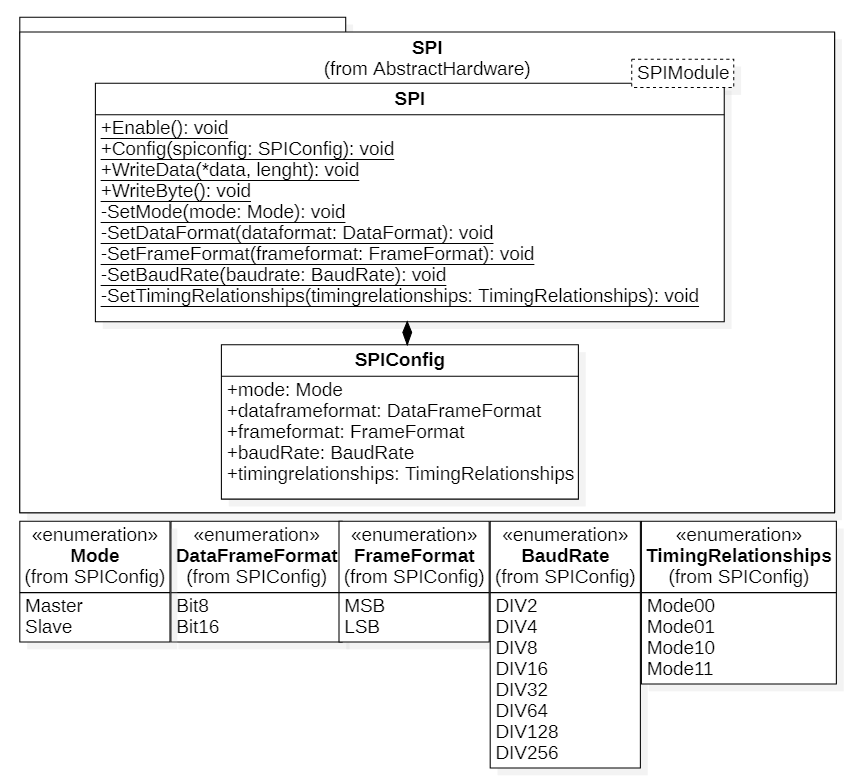


Рисунок 9 – Пакет SPI

Класс SPI осуществляет работу с интерфейсом SPI.

Публичные методы:

Config(spiconfig:SPIConfig) – первичная настройка.

WriteData(\*data, lenght)) – отправка данных.

WriteByte() – записать байт. Используется для заполнения массива данных байтами.

Приватные методы:

SetMode() – установка режима работы

SetDataFrameFormat() – установка размера кадра (8/16–бит)

SetFrameFormat() – установка формы кадра (первым идет LSB(Least Significant Bit – младший значащий бит) или MSB (Most Significant Bit – старший значащий бит))

SetBaudRate() – установка скорости передачи.

SetTimingRelationships() – установка режима простоя (логическая единица или ноль) и режима передачи (по переднему или заднему фронту) для SCK.

SPIConfig – структура для хранения параметров SPI::Config(); содержит классы типа “Enumeration”, показанные на архитектуре, с параметрами для настройки SPI::Config().

1. Pin

Пакет Pin содержит в себе класс Pin, отвечающий за настройку портов (Рисунок 10).

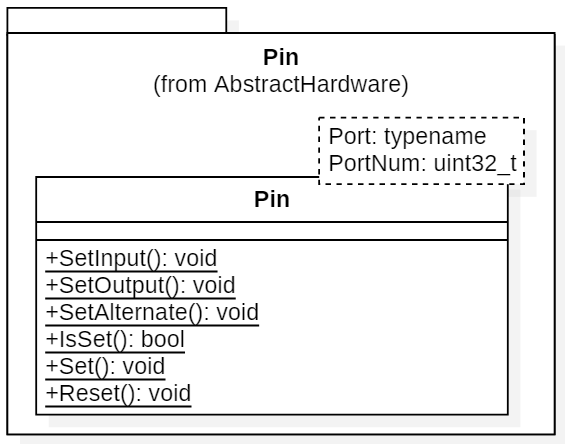


Рисунок 10 – Пакет Pin

Публичные методы:

SetInput() – режим входа (Кнопка).

SetOutput() – режим выхода.

SetAlternate() – альтернативный режим (SPI, USART).

IsSet() – состояние порта (Для опроса кнопки).

Set() – установка 1.

Reset() – установка 0.

1. USART

Пакет USART содержит классы, отвечающие за работу с универсальным синхронно–асинхронным приемопередатчиком USART (рисунок 11).

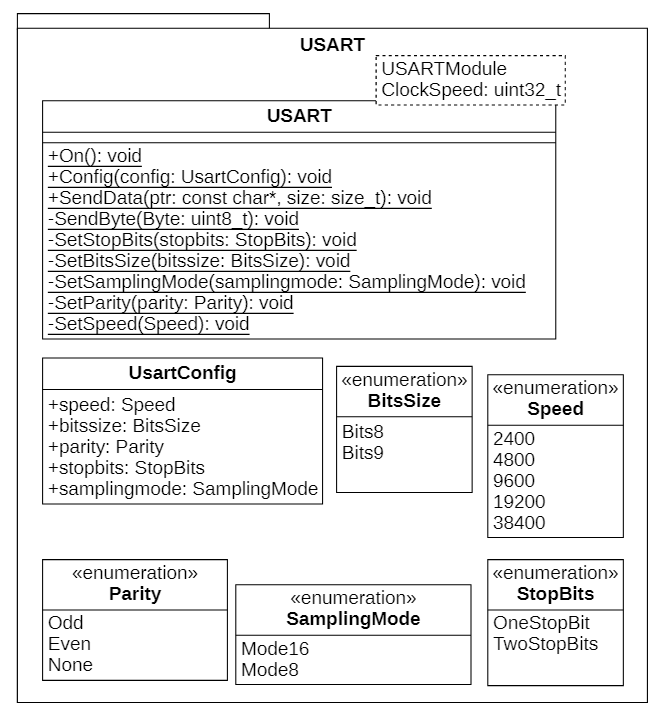


Рисунок 11 – Пакет USART

Класс USART отвечает за работу с универсальным синхронно–асинхронным приемопередатчиком USART.

Публичные методы:

On()– включение модуля USART.

Config() – первичная настройка USART.

SendData()– отправить данные.

Приватные методы:

SendByte() – отправить один бит данных.

SetStopBits() – настройка количества стопбитов для проверки передачи (1 или 2).

SetBitsSize() – настройка количества битов данных для передачи (8 или 9).

SetSamplingMode() – настройка режима дискретизации (1/8 или 1/16).

SetParity() – настройка проверки четности для проверки передачи (режим none– для отключения проверки четности).

SetSpeed() – настройка скорости приемопередатчика (по умолчанию настраиваем на 9600).

UsartConfig – структура с параметрами для настройки USART::Config(); содержит классы типа “Enumeration” с параметрами для настройки USART::Config().

1. Applications

Пакет Applications содержит в себе пакеты приложений Button, Output и Sensor (рисунок 12).

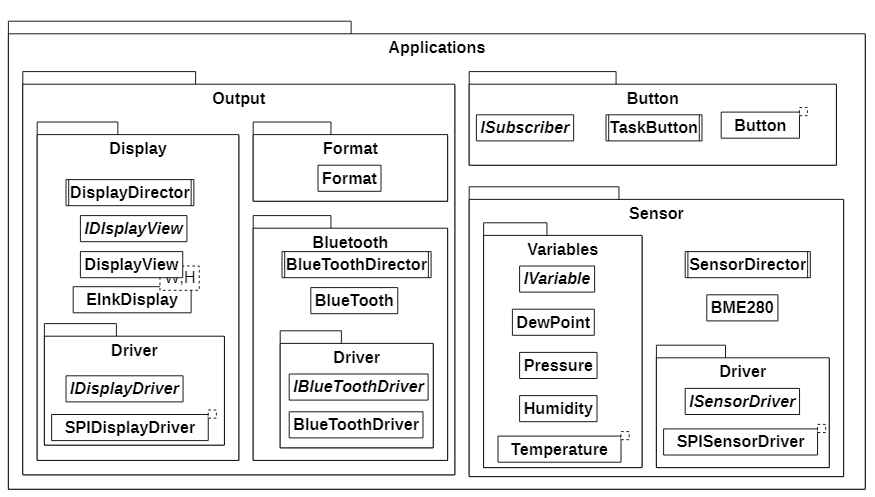


Рисунок 12 – Пакет Applications

1. Button

Пакет Button содержит в себе классы, отвечающие за работу с кнопкой (рисунок 13).

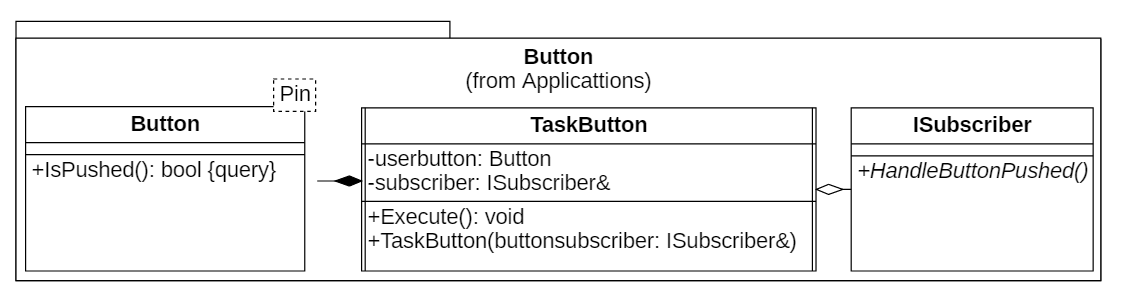


Рисунок 13 – Пакет Button

Класс Button отвечает за состояние кнопки.

Публичный метод:

IsPushed() – состояние кнопки.

Класс TaskButton отвечает за опрос кнопки.

Атрибуты:

userbutton – объект класса Button, создающий пользовательскую кнопку.

subscriber – ссылка на интерфейс ISubscriber.

Публичные методы:

TaskButton() – конструктор, инициализирует ссылку на интерфейс ISubscriber.

Execute() – метод, вызываемый операционной системой реального времени. Раз в 300 миллисекунд опрашивает состояние кнопки.

Интерфейс ISubscriber() отвечает за оповещение датчика о нажатии кнопки.

HandleButtonPushed() – Вирутальный метод, вызываемый при нажатии кнопки.

1. Output

Пакет Output содержит в себе пакеты классов устройств вывода данных (рисунок 14).

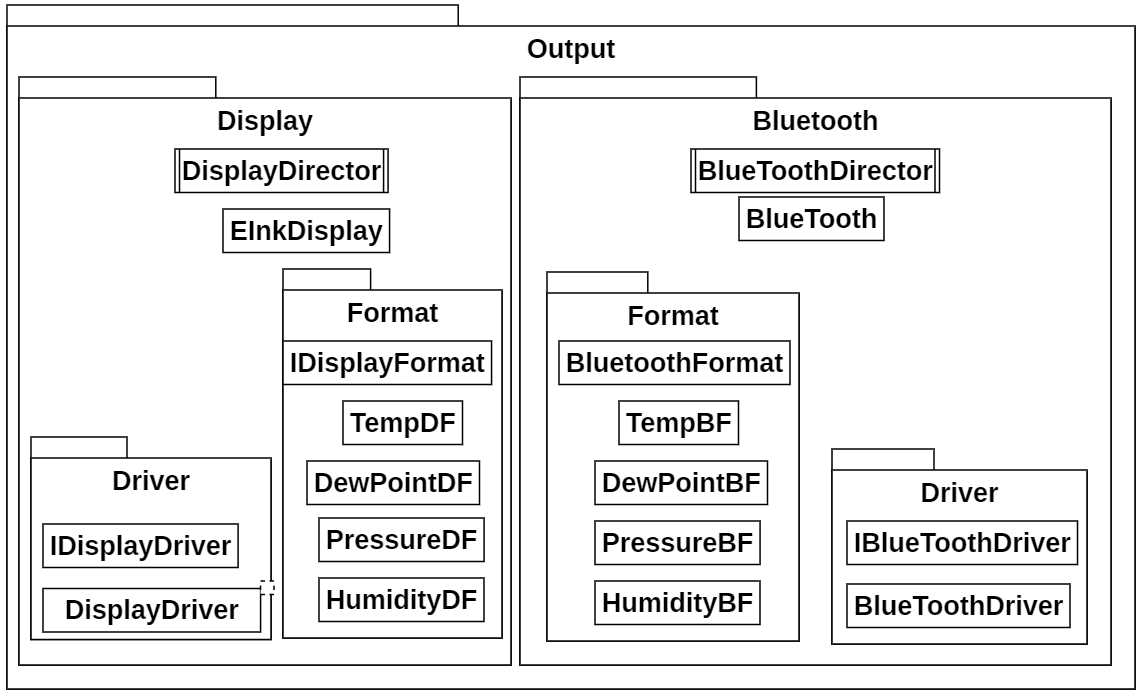


Рисунок 14 – Пакет Output

1. Bluetooth

Пакет Bluetooth содержит в себе классы, отвечающие за работу с Bluetooth–модулем (рисунок 15).

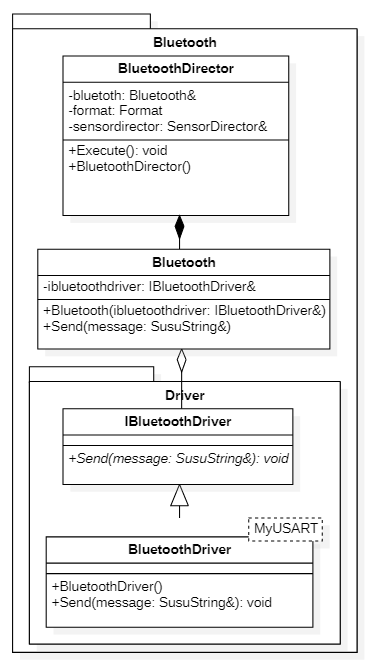


Рисунок 15 – Пакет Bluetooth

Класс BluetoothDriver отвечает за настройку и работу приемопередатчика USART, с помощью которого осуществляется передача данных на приемник по протоколу Bluetooth.

Публичные методы:

BluetoothDriver() – конструктор класса, в который передаются параметры настройки USART.

Send() – отправка данных.

Интерфейс IBluetoothDriver – для работы с драйверами Bluetooth

Виртуальная публичная функция, определяемая в интерфейсе:

Send() – отправить информацию на приемник через Bluetooth.

Класс Bluetooth отвечает за работу с конкретным Bluetooth-модулем.

Приватные атрибуты:

ibluetoothdriver : IBluetoothDriver& – ссылка на интерфейс IBluetoothDriver.

Публичные методы:

Bluetooth() – конструктор; служит для инициализации драйвера Bluetooth.

Send() – Отправка данных через Bluetooth. Переопределяет виртуальный метод.

Активный класс BluetoothDirector – отвечает за работу задачи Bluetooth.

Приватные атрибуты:

bluetooth : Bluetooth& – ссылка на класс Bluetooth.

format : Format – создать объект класса Format.

sensordirector : SensorDirector& – ссылка на класс SensorDirector.

Публичные методы:

BluetoothDirector() – конструктор; инициализирует ссылки на классы SensorDirector и Bluetooth.

Execute()– виртуальный метод операционной системы реального времени, определяемый в классе Thread и переопределяемый в классе BluetoothDirector; раз в секунду получает значения из SensorDirector, преобразует их через Format, и отправляет через Bluetooth.

1. Display

Пакет Display содержит в себе классы, отвечающие за работу с дисплеем (рисунок 16).

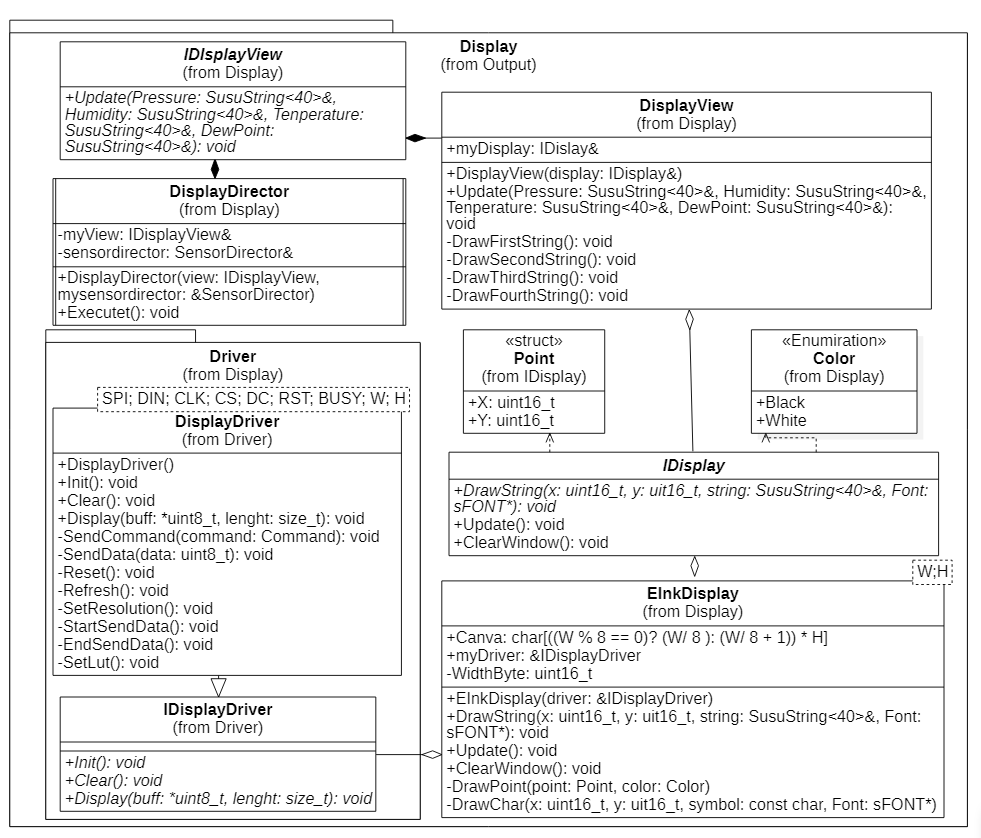


Рисунок 16 – Пакет Display

Пакет Driver содержит в себе классы, отвечающие за работу с драйверами дисплея.

Интерфейс IDisplayDriver отвечает за объединение драйверов общим интерфейсом.

Виртуальные методы:

Init() – инициализация дисплея.

Clear() – очистка дисплея.

Display() – отображение на дисплее.

Класс DisplayDriver – драйвер для подключения и работы с дисплеем. Для подключения дисплея используются классы SPI и PIN.

Параметры, передаваемые в шаблон:

SPI – выбор модуля SPI.

DIN – пин SPI MOSI.

CLK – пин SPI SCK.

CS – пин выбор микросхемы SPI (выбор ведомого устройства).

DC – пин выбора данных или команды (1 – данные, 0 – комманда).

RST – пин внешнего сброса.

BUSY – пин вывода значения занятости.

W – ширина дисплея.

H – высота дисплея.

Публичные методы:

DisplayDriver() – конструктор, в теле которого происходит настройка режима пинов и настройка выбранного модуля SPI.

Init() – инициализация дисплея. Переопределяет виртуальный метод.

Clear() – очистка дисплея. Переопределяет виртуальный метод.

Display() – отображение на дисплее. Переопределяет виртуальный метод.

Приватные методы:

SendCommand() – отправить команду.

SendData() – отправить данные.

Reset() – сброс.

Refresh() – обновить дисплей.

SetResolution() – установить разрешение.

StartSendData() – начать передачу данных.

EndSendData() – закончить передачу данных.

SetLut() – установка таблицы быстрого отображения.

Интерфейс IDisplay отвечает за объединение дисплеев общим интерфейсом.

Виртуальные методы:

DrawString() – вывод строки на дисплей.

Update() – обновление дисплея.

ClearWindow() – очистка дисплея.

EInkDisplay – класс, отвечающий за взаимодействие с дисплеем с электронными чернилами. Наследует интерфейс IDisplay.

Публичный атрибут:

myDriver – ссылка на интерфейс IDriver.

Приватные атрибуты:

Canva – холст, область вывода данных.

WidthByte – ширина в байтах.

Публичные методы:

EInkDisplay() – конструктор, инициализирующий ссылку на IDisplayDriver, в теле которого осуществляется инициализация драйвера.

DrawString() – вывод строки на дисплей. Переопределяет виртуальный метод.

Update() – обновление дисплея. Переопределяет виртуальный метод.

ClearWindow() – очистка дисплея. Переопределяет виртуальный метод.

DrawPoint() – метод рисования точки.

DrawChar() – метод рисования символа.

Структура Point хранит в себе координаты x и y.

Структура Color хранит в себе значения цветов.

IDisplayView – интерфейс отображения на дисплее. Отвечает за формирование картинки, отображаемой на дисплее.

Виртуальный метод:

Update() – обновляет дисплей и выводит на него значения в определенном формате.

DisplayView – вид отображения на дисплее. Позволяет разбить область дисплея на четыре области, требуемые для вывода значений и объединить их в общую картинку.

Приватный атрибут:

myDisplay() – ссылка на интерфейс IDisplay.

Публичные методы:

DisplayView() – инициализирует ссылку на интерфейс IDisplay.

Update() – обновляет дисплей и выводит на него значения в определенном формате. Переопределяет виртуальный метод.

DrawFirstString() – отображение первой строки.

DrawSecondString() – отображение второй строки.

DrawThirdString() – отображение третьей строки.

DrawFourthString() – отображение четвертой строки.

DisplayDirector – класс, отвечающий за задачу дисплея. Наследует ОСРВ.

Приватные атрибуты:

myView – ссылка на интерфейс IDisplayView.

sensordirector – ссылка на класс SensorDirector.

Публичные методы:

DisplayDirector() – конструктор, инициализирующий ссылки на IDisplayView и SensorDirector.

Execute() – метод, вызываемый операционной системой реального времени. Раз в три секунды получает значения из SensorDirector, преобразует их через Format, и обновляет дисплей.

1. Format

Пакет Format содержит в себе класс Format, отвечающий за преобразование данных, принятых от SensorDirector, в кортеж данных для вывода на дисплей и передачи через BlueTooth модуль (рисунок 17). Для формирования данных обратимся к порогу чувствительности датчика (п. 1.2).

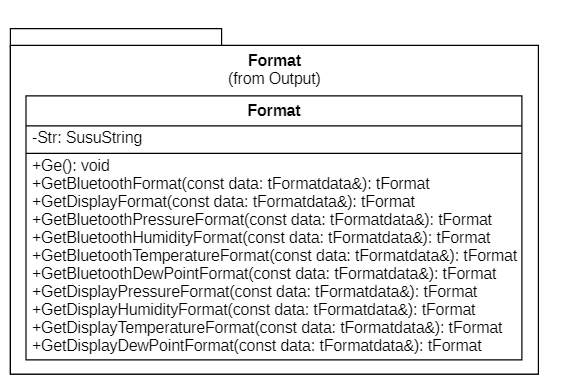


Рисунок 17 – Пакет Format

Порог чувствительности температуры 0.01°C. При отображении данных будем использовать два знака после запятой: .2f.

Порог чувствительности влажности 0.008%. При отображении данных будем использовать три знака после запятой: .3f.

Порог чувствительности барометрического давления 0.18Па, или 0,0013 мм.рт.ст. При отображении данных будем использовать три знака после запятой: .3f.

Публичные методы:

GetBlueToothFormat() – возвращает кортеж элементов, состоящий из отформатированных для вывода по Bluetooth строк со значениями температуры, давления, влажности и точки росы.

GetDisplayFormat() – возвращает кортеж элементов, состоящий из отформатированных для вывода на дисплей строк со значениями температуры, давления, влажности и точки росы.

Приватные методы:

GetBlueToothPressureFormat() – преобразует кортеж с данными давления в строку для вывода по Bluetooth.

GetBlueToothHumidityFormat() – преобразует кортеж с данными влажности в строку для вывода по Bluetooth.

GetBlueToothTemperatureFormat() – преобразует кортеж с данными температуры в строку для вывода по Bluetooth.

GetBlueToothDewPointFormat() – преобразует кортеж с данными точки росы в строку для вывода по Bluetooth.

GetDisplayPressureFormat() – преобразует кортеж с данными давления в строку для вывода на дисплей.

GetDisplayHumidityFormat() – преобразует кортеж с данными влажности в строку для вывода на дисплей.

GetDisplayTemperatureFormat() – преобразует кортеж с данными температуры в строку для вывода на дисплей.

GetDisplayDewPointFormat() – преобразует кортеж с данными точки росы в строку для вывода на дисплей.

1. Sensor

Пакет Sensor содержит в себе пакеты переменных, драйверов дисплея, и классы работы с дисплеем (рисунок 18).

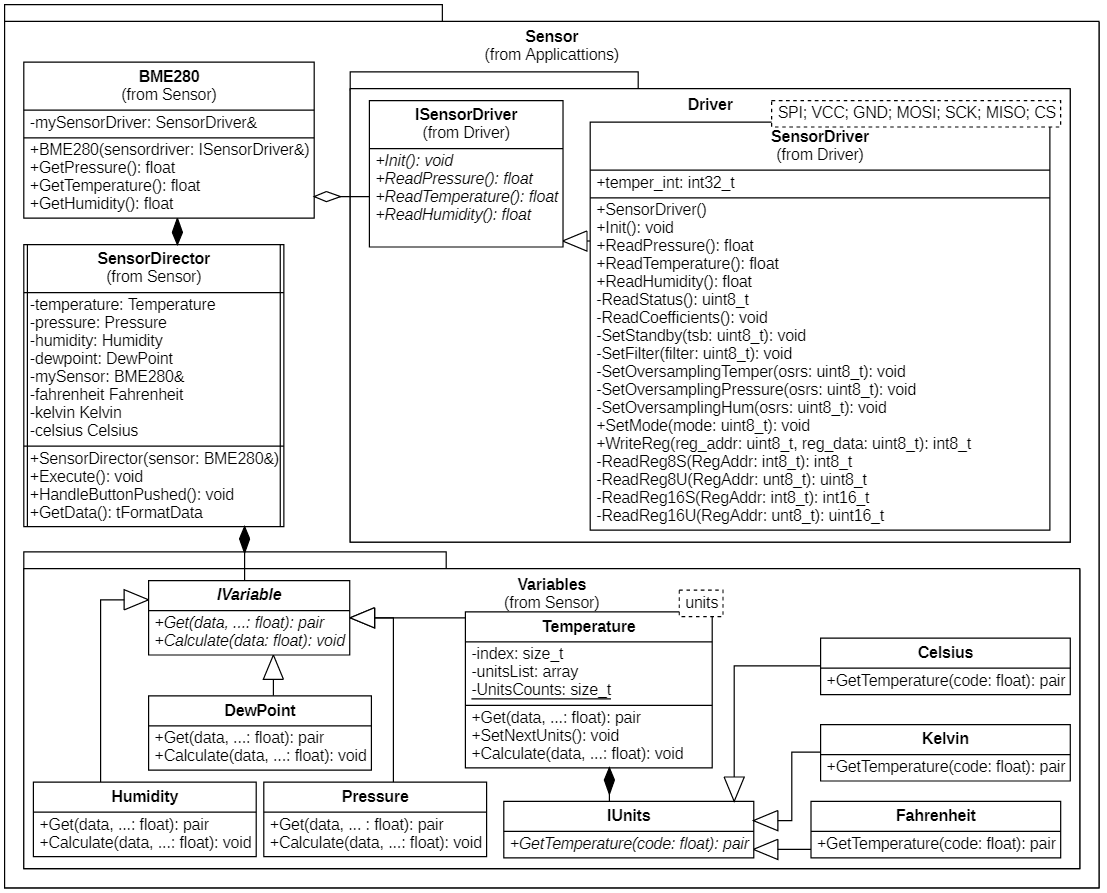


Рисунок 18 – Пакет Sensor

Пакет Variables содержит классы, отвечающие за работу с переменными: DewPoint, Humidity, Pressure, Temperature, классы единиц измерения температуры: Celsius, Kelvin, Fahrenheit, и интерфейсы IVariable и IUnits, обобщающие классы переменных и единиц измерения температуры.

Интерфейс IVariable – интерфейс переменных, объявляет виртуальные публичные методы:

Get() – возвращает кортеж данных, состоящий из значения переменной и единицы измерения переменной.

Calculate() – рассчитывает значение искомой переменной.

Классы Humidity, Pressure, DewPoint и Temperature наследуют интерфейс переменных IVariables, и отвечают, соответственно, за влажность, давление, точку росы и температуру.

Классы Humidity, Pressure, DewPoint имеют только наследуемые методы интерфейса IVariables.

Класс Temperature наследует методы интерфейса IVariables, а также имеет публичный метод:

SetNextUnits() – по нажатию кнопки изменяет единицу измерения температуры на следующую (F→K→C).

Приватные атрибуты:

index: size\_t – счетчик нажатий кнопки.

unitsList: array – массив единиц измерения.

UnitsCounts: size\_t – размер массива единиц измерения.

Интерфейс IUnits – интерфейс единиц измерения температуры, объявляет виртуальный метод:

GetTemperature() – рассчитывает значение температуры и возвращает кортеж данных , состоящий из значения температуры и единицы измерения.

Классы Celsius, Kelvin, Fahrenheit наследуют интерфейс единиц измерения IUnits и служат для расчета температуры в требуемых единицах измерения. Наследуют виртуальный метод интерфейса GetTemperature().

Интерфейс ISensorDriver – интерфейс драйверов датчика, объявляет виртуальные методы:

Init() – инициализация драйвера (первичная настройка).

ReadPressure() – возвращает значение давления.

ReadTemperature() – возвращает значение температуры.

ReadHumidity() – возвращает значение влажности.

Класс SensorDriver наследует интерфейс ISensorDriver и отвечает за работу с датчиком через SPI.

Публичные методы:

SensorDriver() – конструктор, в теле которого настраиваются порты и SPI.

Init() – инициализация драйвера (первичная настройка). Переопределяет виртуальный метод.

ReadPressure() – возвращает значение давления. Переопределяет виртуальный метод.

ReadTemperature() – возвращает значение температуры. Переопределяет виртуальный метод.

ReadHumidity() – возвращает значение влажности. Переопределяет виртуальный метод.

Приватные методы:

ReadStatus() – чтение статуса.

ReadCoefficients() – чтение калибровочных значений.

SetStandby() – настройка времени нахождения в режиме STANDBY. Датчик переходит в данный режим после того, как произведёт измерения.

SetFilter() – настройка коэффициента фильтрации.

SetOversamplingTemper() – настройка значения оверсемплинга (передискректизации) температуры.

SetOversamplingPressure() – настройка значения оверсемплинга (передискректизации) температуры.

SetOversamplingHum() – настройка значения оверсемплинга (передискректизации) температуры.

SetMode() – настройка режима датчика.

WriteReg() – запись байта в регистры датчика.

ReadReg8S() – чтение однобайтового знакового значения из регистров датчика.

ReadReg8U() – чтение однобайтового беззнакового значения из регистров датчика.

ReadReg8S() – чтение двухбайтового знакового значения из регистров датчика.

ReadReg8U() – чтение двухбайтового беззнакового значения из регистров датчика.

Класс BME280 отвечает за работу датчика BME280.

Приватный атрибут:

mySensorDriver – ссылка на интерфейс ISensorDriver.

Публичные классы:

BME280() – конструктор, инициализирующий ссылку на интерфейс ISensorDriver, в теле которого инициализируется драйвер датчика.

GetTemperature() – возвращает значение температуры в C, считанное с датчика BME280.

GetPressure() – возвращает значение давление, переведенное из Па в мм. рт. ст., считанное с датчика BME280.

GetHumidity() – возвращает значение влажности в %, считанное с датчика BME280.

Активный класс SensorDirector отвечает за работу с задачей датчика. Наследует операционную систему реального времени.

Приватные атрибуты:

temperature – объект класса Temperature.

pressure – объект класса Pressure.

humidity – объект класса Humidity.

dewpoint – объект класса DewPoint.

mySensor – ссылка на класс BME280.

fahrenheit – объект класса Fahrenheit.

kelvin – объект класса Kelvin.

celsius – объект класса Celsius.

Публичные методы:

SensorDirector() – конструктор, инициализирующий ссылку на класс BME280.

Execute() – метод, вызываемый операционной системой реального времени. считывает данные с датчика раз в 100 мс.

HandleButtonPushed() – при вызове вызывает у temperature метод SetNextUnits, который меняет единицу измерения температуры на следующую.

GetData() – возвращает кортеж из восьми элементов: значений и единиц измерения давления, влажности, температуры и точки росы.

1. Результат создания по
2. Создание интерфейса на телефоне

Для вывода измеренных значений на телефон было использовано приложение Bluetooth Electronics. Общий вид интерфейса показан на рисунке 19.

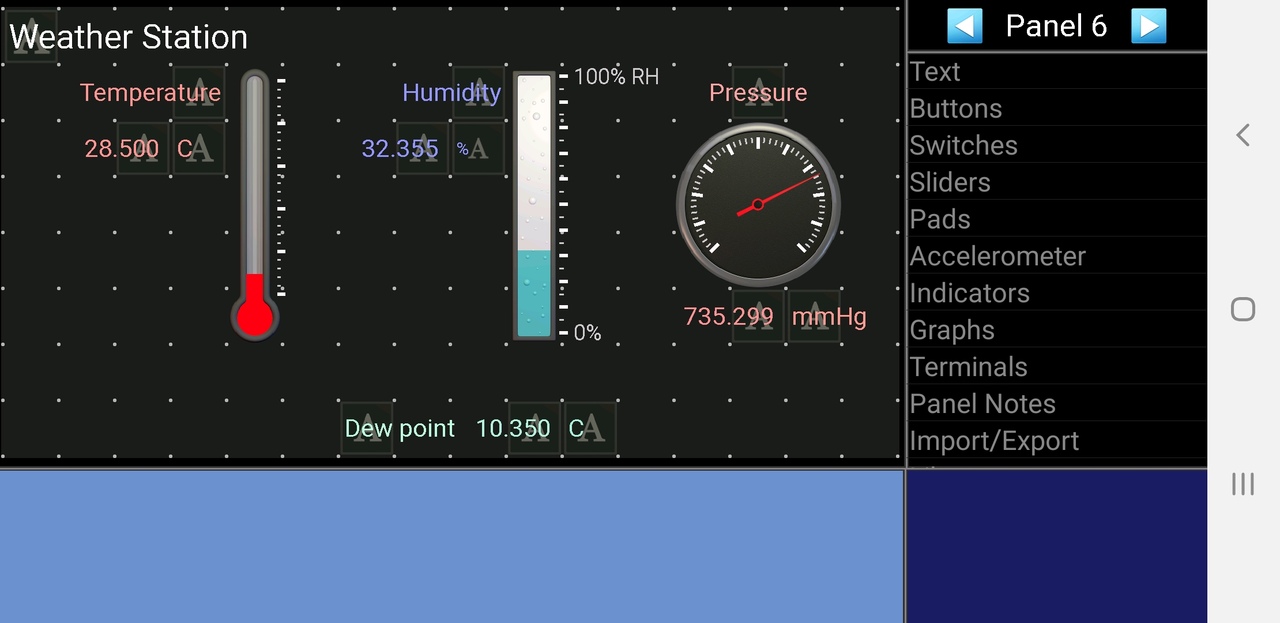


Рисунок 19 – Интерфейс WeatherStation

Значение температуры выводятся в числовых значениях и визуализируются с помощью изображения шкалы.

Значение влажности выводятся в числовых значениях и визуализируются с помощью изображения шкалы.

Значение давления выводятся в числовых значениях и визуализируются с помощью изображения стрелочного прибора.

Значение точки росы выводятся в числовых значениях и визуализируются с помощью изображения шкалы.

Значение температуры считывается из строки после специальных символов:

\*T Temperature\*,

где T – идентификатор для вывода температуры,

Temperature – переменная, хранящее значение температуры.

Значение единиц измерения температуры считывается из строки после специальных символов:

\* V Value\*,

где V – идентификатор для вывода единиц измерения,

Value – переменная, хранящее значение единиц измерения.

Значение влажности считывается из строки после специальных символов:

\*H Humidity\*,

где H – идентификатор для вывода влажности,

Humidity – переменная, хранящее значение влажности.

Значение давления считывается из строки после специальных символов:

\*P Pressure\*,

где P – идентификатор для вывода давления,

Pressure – переменная, хранящее значение давления.

Значение точки росы считывается из строки после специальных символов:

\*D DewPoint\*,

где D – идентификатор для вывода точки росы,

DewPoint – переменная, хранящее значение точки росы.

1. Проверка работоспособности

Данные, отображаемые при запуске проекта (рисунок 20).

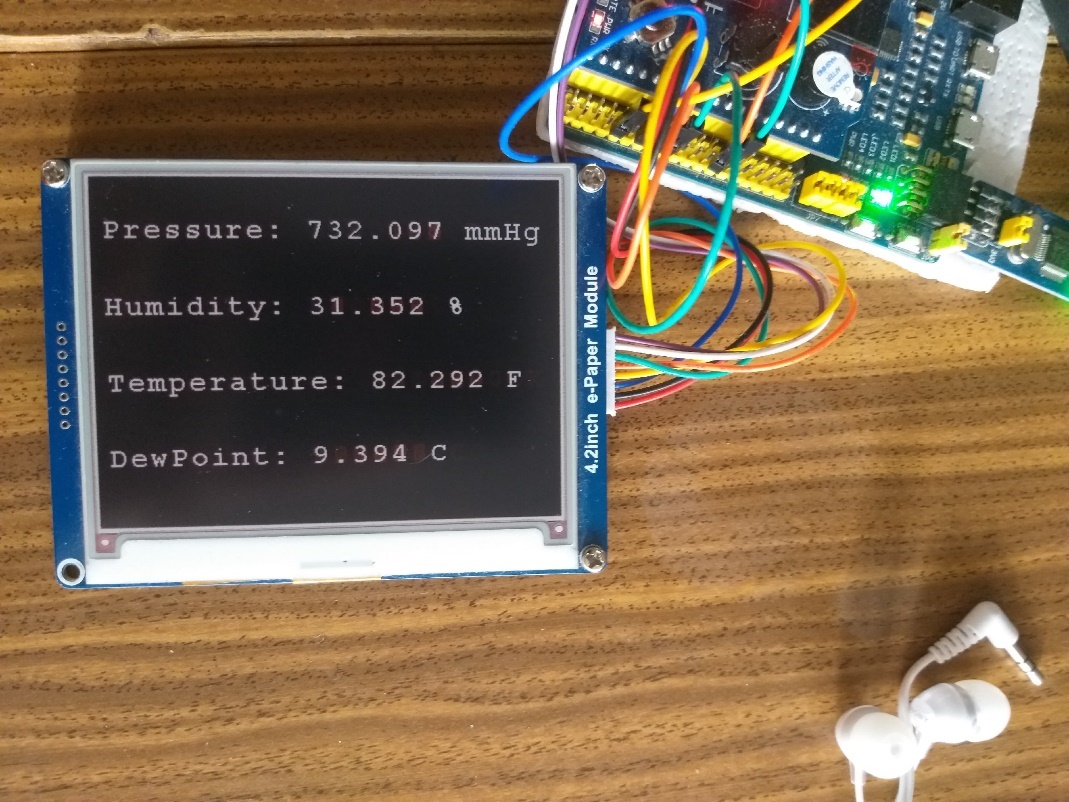
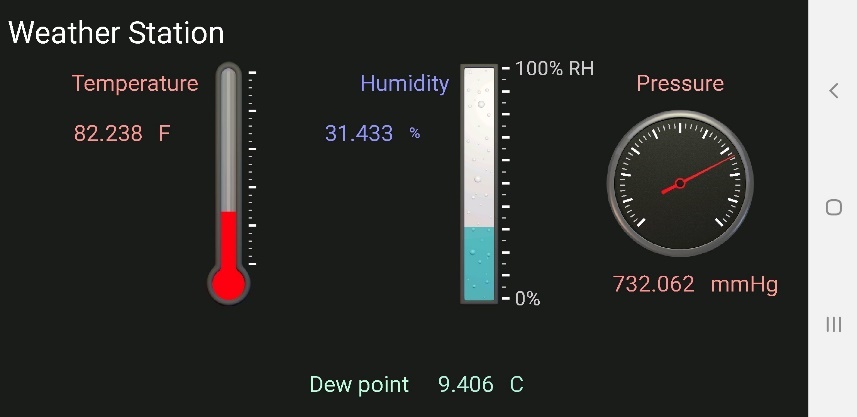
 

Рисунок 20 – Отображение данных после запуска программы

Нажали пользовательскую кнопку один раз (рисунок 21).

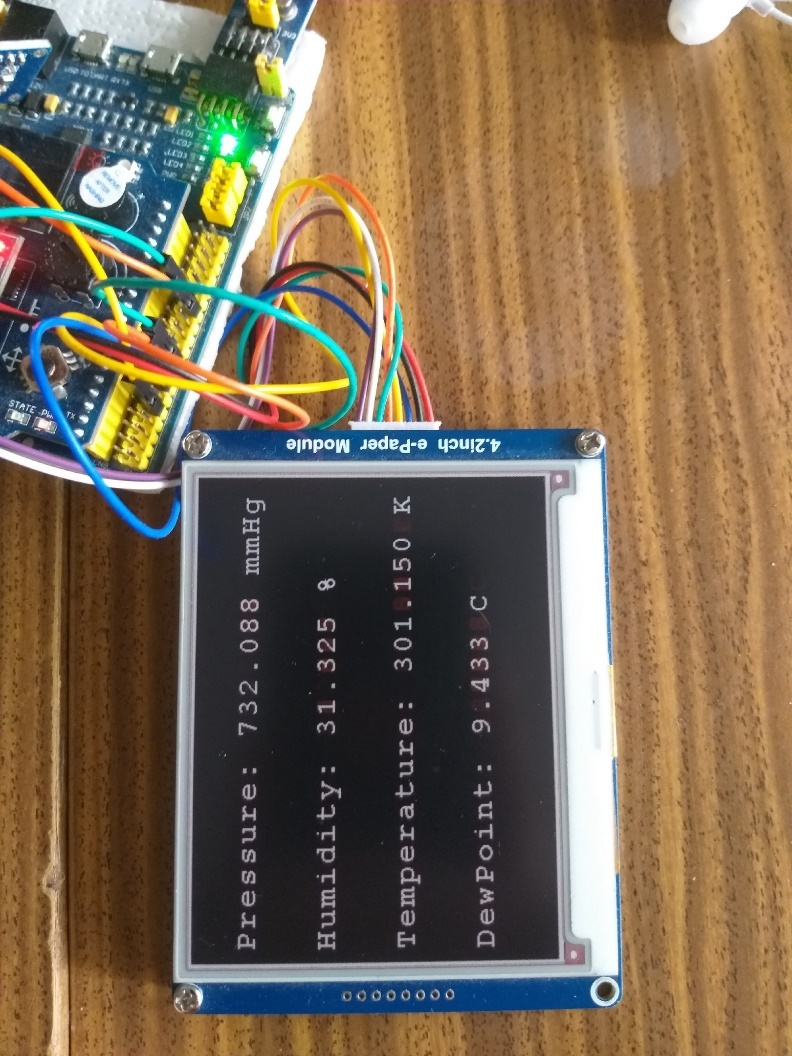
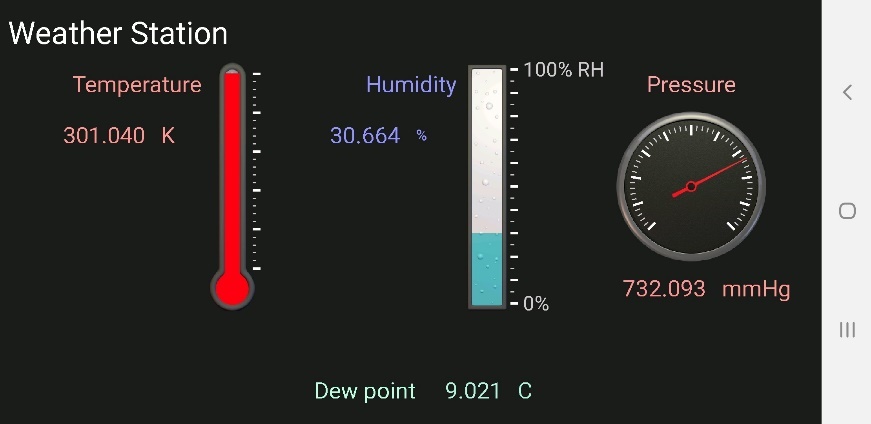
 

Рисунок 21 – Отображение данных после первого нажатия кнопки

Нажали пользовательскую кнопку второй раз (рисунок 22).

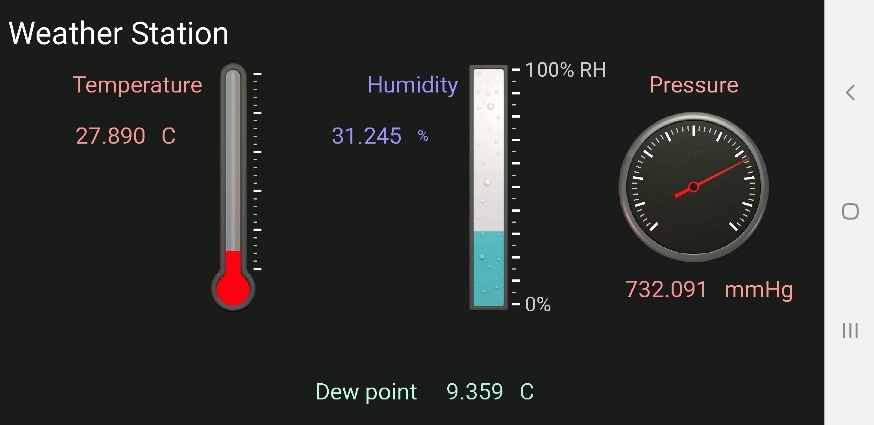
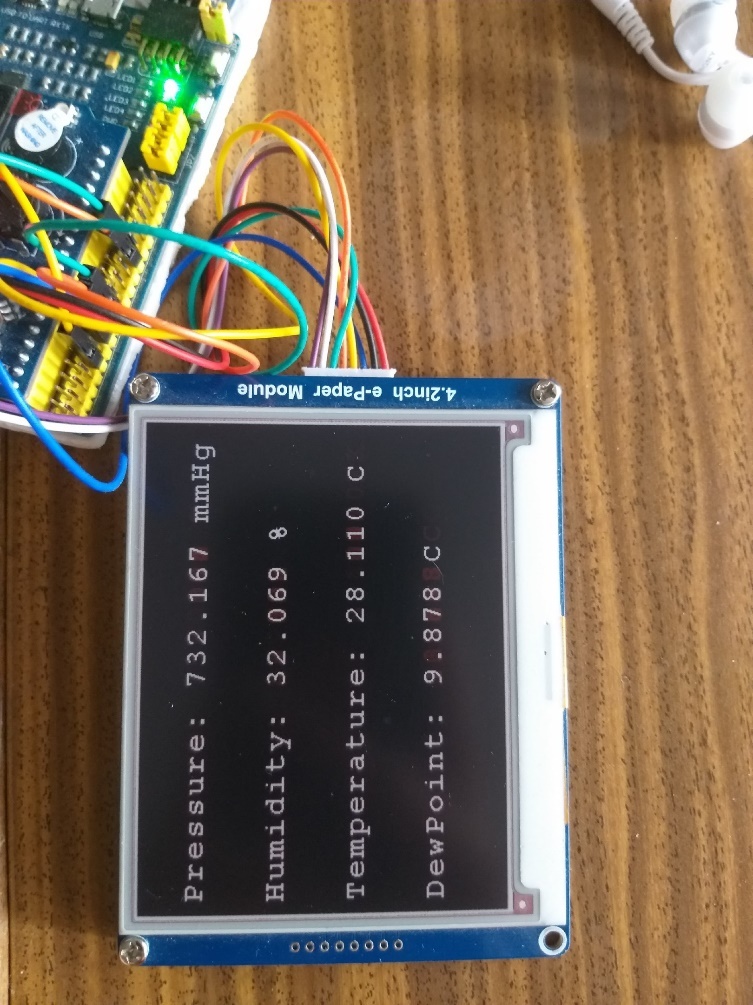


Рисунок 22 – Отображение данных после второго нажатия кнопки

Вывод: индикаторы реагируют на нажатие кнопки. Последовательность смены единиц измерения выполняется верно.

Поместили датчик рядом с холодным объектом. Показания индикатора Bluetooth (рисунок 23).

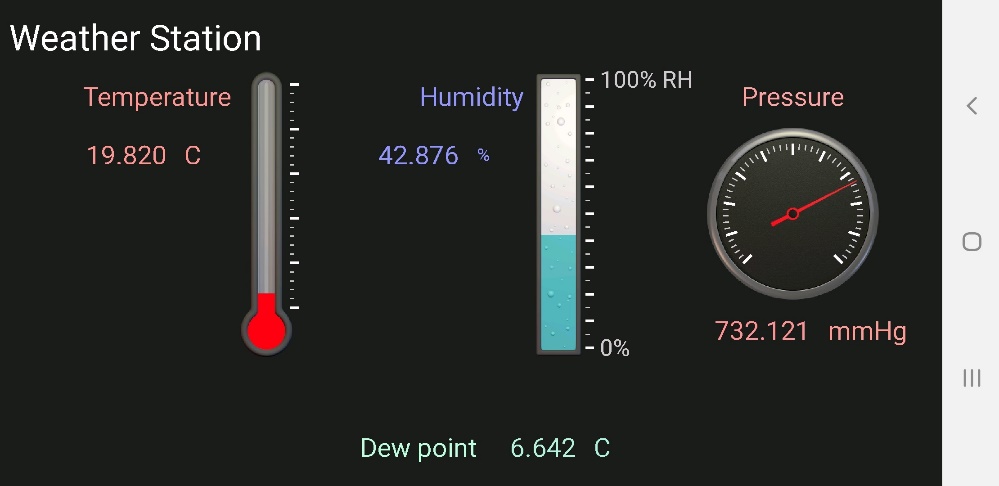


Рисунок 23 – Результат размещения датчика в холодной среде

Вывод: датчик реагирует на понижение температуры и изменение влажности.

Поместили датчик над теплым паром. Показания индикатора Bluetooth (рисунок 24).

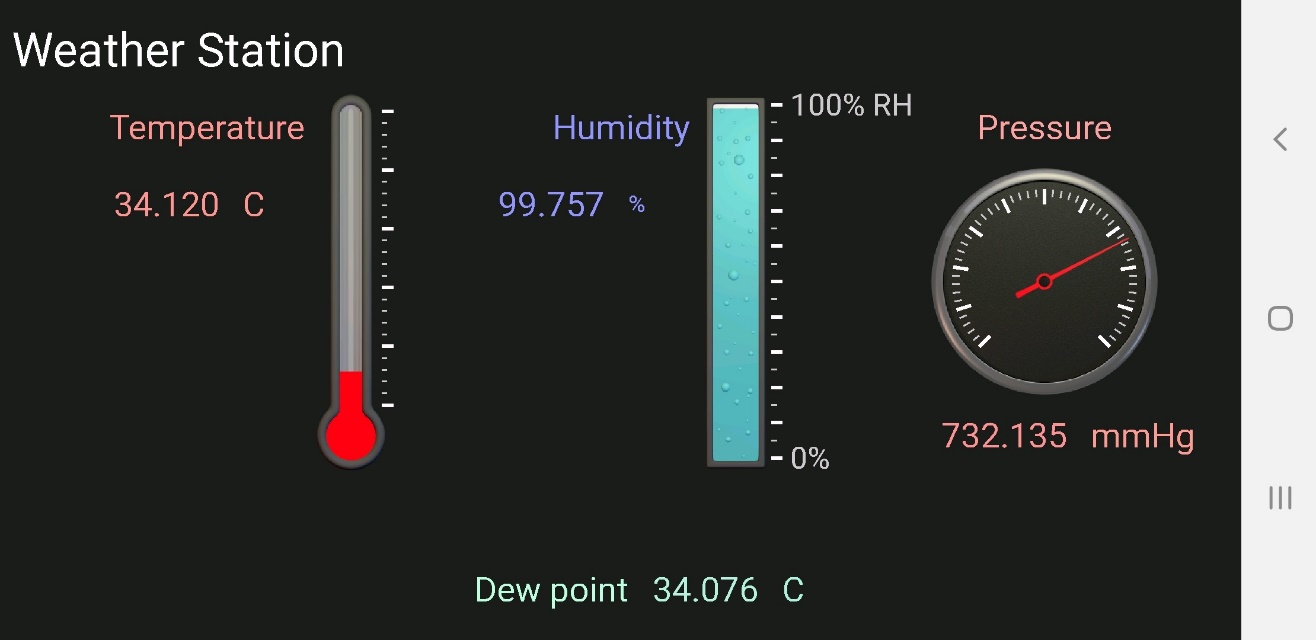


Рисунок 24 – Результат размещения датчика в теплой влажной среде

Вывод: датчик реагирует на повышение температуры и высокую влажности.

Поместили датчик за окном для измерения погодных условий на улице. Данные с сайта погоды (рисунок 25).



Рисунок 25 – Погода в городе

Данные, снятые с датчика, размещенного за окном (рисунок 26):

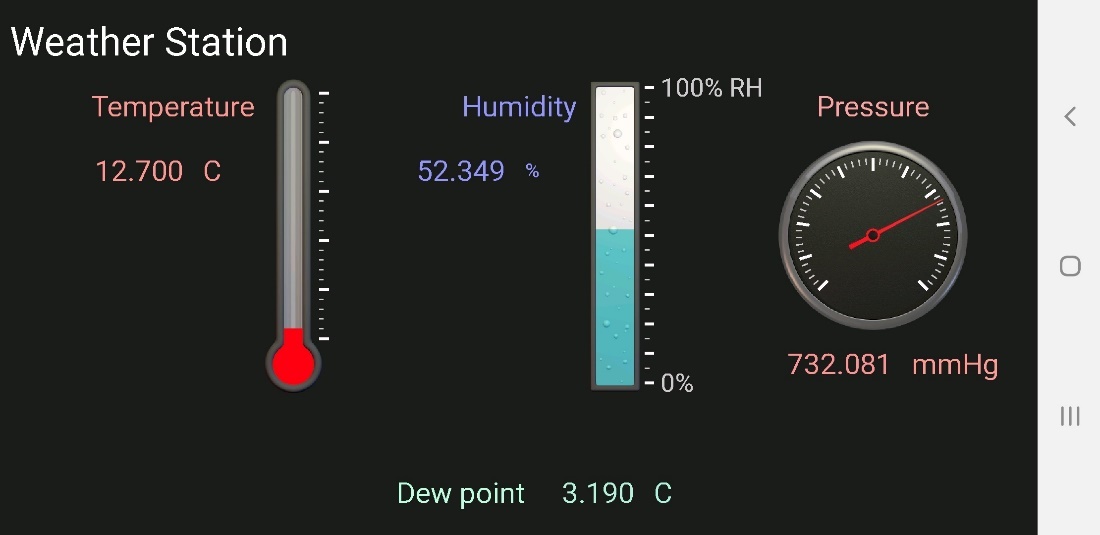


Рисунок 26 – Показания датчика

Вывод: данные снятые с датчика и данные с сайта погоды близки. Разницу в показаниях можно объяснить различием во времени снятия показаний (16:00 сайт, 16:30 датчик), а также отличием погодных условий внутри города в зависимости от района.

Заключение

В данном курсовом проекте была разработано программное обеспечение метеостанции, реализованной на основе отладочной платы XNUCLEO–F411RE, определяющей с помощью датчика BME280 значения температуры, влажности, давления и на основе этих показателей рассчитывающей значение точки росы.

Осуществлен вывод полученных значений на дисплей с жидкими чернилами и передача данных по беспроводному протоколу Bluetooth.

С помощью операционной системы реального времени FreeRTOS реализована работа с заданной в технических требованиях периодичностью.

Описан функционал и преимущества всех используемых программ и оборудования.

Проведены испытания, показывающие работоспособность метеостанции и ее способность реагировать на изменение окружающих климатических условий.

Библиографический список

1 Страуструп, Б. Программирование: принципы и практика с использованием С++ / Б. Страуструп. – М.: И.Д. Вильямс, 2016. – 1328 с.

2 Лафоре Р. Объектно-ориентированное программирование в C++ / Р. Лафоре. – СПб.: ПИТЕР, 2018. – 923 с.

3 Беспалов Д.А. Операционные системы реального времени и технологии разработки кроссплатформенного программного обеспечения. Часть 1: учебное пособие. / Д.А. Беспалов, С.М. Гушанский, Н.М. Коробейникова. – Ростов-на-Дону.: Издательство Южного федерального университета, 2019. – 139 с.

Приложения  
ПРИЛОЖЕНИЕ А

Листинг программы

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* | File : Pin.hpp

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#pragma once

template<typename Port, std::uint32\_t PortNum>

class Pin {

public:

static void Reset() {

Port::BSRR::Write((1U << PortNum ) << 16U);

}

static void Set() {

Port::BSRR::Write(1U << PortNum);

}

static void SetInput() {

static\_assert(PortNum <= 15U, "There are only 16 pins on port") ;

volatile auto value = Port::MODER::Get() ;

value &= ~ (3 << (PortNum \* 2U)) ;

value |= (Port::MODER::FieldValues::Input::Value << (PortNum \* 2U)) ;

Port::MODER::Write(value) ;

}

static void SetOutput() {

static\_assert(PortNum <= 15U, "There are only 16 pins on port") ;

volatile auto value = Port::MODER::Get() ;

value &= ~ (3 << (PortNum \* 2U)) ;

value |= (Port::MODER::FieldValues::Output::Value << (PortNum \* 2U)) ;

Port::MODER::Write(value);

}

static void SetAlternate() {

static\_assert(PortNum <= 15U, "There are only 16 pins on port") ;

volatile auto value = Port::MODER::Get() ;

value &= ~(3 << (PortNum \* 2U)) ;

value |= (Port::MODER::FieldValues::Alternate::Value << (PortNum \* 2U)) ;

Port::MODER::Write(value) ;

}

static bool IsSet() {

static\_assert(PortNum <= 15U, "There are only 16 pins on port") ;

return ((Port::IDR::Get() & (1 << PortNum)) != 0) ;

}

} ;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* End of file \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* | File : SPI.hpp

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#pragma once

#include "SPIConfig.hpp" //for SPI::Config(SPIConfig spiconfig)

template <typename SPIModule>

class SPI {

public:

static void Enable() {

SPIModule :: CR1 :: SPE :: Enable :: Set() ;

}

static void Config(SPIConfig spiconfig) {

SetMode(spiconfig.mode);

SetDataFormat(spiconfig.dataformat);

SetFrameFormat(spiconfig.frameformat);

SetBaudRate(spiconfig.baudrate);

SetTimingRelationships(spiconfig.timingrelationships);

SetBIDImode(spiconfig.bidimode);

SetCSmode(spiconfig.csmode);

SetCRC(spiconfig.crcen);

}

static void WriteData(uint8\_t \*data, size\_t lenght) {

//если длина нулевая (нечего передавать)

assert(lenght > 0);

for (size\_t index = 0; index < lenght; index++)

{

//передаем байт данных

WriteByteU(data[index]);

}

}

static void WriteByteU(uint8\_t Byte) {

//ждем, пока буфер передатчика не освободится

while (SPIModule :: SR :: TXE :: TxBufferNotEmpty :: IsSet()) {

}

//передаем байт данных

SPIModule :: DR :: Write(Byte) ;

//ждем, пока SPI освободится от предыдущей передачи

while (SPIModule :: SR :: BSY :: Busy :: IsSet()) {

}

}

static void WriteByteS(int8\_t Byte) {

//ждем, пока буфер передатчика не освободится

while (SPIModule :: SR :: TXE :: TxBufferNotEmpty :: IsSet()) {

}

//передаем байт данных

SPIModule :: DR :: Write(Byte) ;

//ждем, пока SPI освободится от предыдущей передачи

while (SPIModule :: SR :: BSY :: Busy :: IsSet()) {

}

}

static std::pair<int8\_t, bool> ReadByteS() {

/\*

Bit 0 RXNE: Receive buffer not empty

0: Rx buffer empty

1: Rx buffer not empty

\*/

int8\_t value = 0;

bool result = true;

int i = 0;

//ждем, пока закончит чтение

while ((SPIModule :: SR :: RXNE :: Value0 :: IsSet()) && (i< 1000)) {

i++ ;

}

if (i == 1000) {

result = false;

};

//читаем байт данных

value = SPIModule :: DR :: Get() ;

return std::make\_pair(value,result) ;

}

static std::pair<uint8\_t, bool> ReadByteU() {

uint8\_t value = 0U;

bool result = true;

int i = 0;

while ((SPIModule :: SR :: RXNE :: Value0 :: IsSet()) && (i< 1000)) {

i++ ;

}

if (i == 1000) {

result = false;

};

value = SPIModule :: DR :: Get() ;

return std::make\_pair(value,result) ;

}

private:

static void SetMode(Mode mode) {

switch (mode) {

case Mode::Master:

SPIModule :: CR1 :: MSTR :: Master :: Set() ;

break ;

case Mode::Slave:

SPIModule :: CR1 :: MSTR :: Slave :: Set() ;

break ;

default:

assert(false) ;

break ;

}

}

static void SetDataFormat(DataFormat dataformat) {

switch (dataformat) {

case DataFormat::Bit8:

SPIModule :: CR1 :: DFF :: Data8bit :: Set() ;

break ;

case DataFormat::Bit16:

SPIModule :: CR1 :: DFF :: Data16bit :: Set() ;

break ;

default:

assert(false) ;

break ;

}

}

static void SetFrameFormat(FrameFormat frameformat) {

switch (frameformat) {

case FrameFormat::MSB:

SPIModule :: CR1 :: LSBFIRST :: MsbFirst :: Set() ;

break ;

case FrameFormat::LSB:

SPIModule :: CR1 :: LSBFIRST :: LsbFirst :: Set() ;

break ;

default:

assert(false) ;

break ;

}

}

static void SetBaudRate(BaudRate baudrate) {

switch (baudrate) {

case BaudRate::DIV2:

SPIModule :: CR1 :: BR :: PclockDiv2 :: Set() ;

break ;

case BaudRate::DIV4:

SPIModule :: CR1 :: BR :: PclockDiv4 :: Set() ;

break ;

case BaudRate::DIV8:

SPIModule :: CR1 :: BR :: PclockDiv8 :: Set() ;

break ;

case BaudRate::DIV16:

SPIModule :: CR1 :: BR :: PclockDiv16 :: Set() ;

break ;

case BaudRate::DIV32:

SPIModule :: CR1 :: BR :: PclockDiv32 :: Set() ;

break ;

case BaudRate::DIV64:

SPIModule :: CR1 :: BR :: PclockDiv64 :: Set() ;

break ;

case BaudRate::DIV128:

SPIModule :: CR1 :: BR :: PclockDiv128 :: Set() ;

break ;

case BaudRate::DIV256:

SPIModule :: CR1 :: BR :: PclockDiv256 :: Set() ;

break ;

default:

assert(false) ;

break ;

}

}

static void SetTimingRelationships(TimingRelationships timingrelationships) {

switch (timingrelationships) {

case TimingRelationships::MODE00:

SPIModule :: CR1 :: CPOL :: Low :: Set() ;

SPIModule :: CR1 :: CPHA :: Phase1edge :: Set() ;

break;

case TimingRelationships::MODE01:

SPIModule :: CR1 :: CPOL :: Low :: Set() ;

SPIModule :: CR1 :: CPHA :: Phase2edge :: Set() ;

break;

case TimingRelationships::MODE10:

SPIModule :: CR1 :: CPOL :: High :: Set() ;

SPIModule :: CR1 :: CPHA :: Phase1edge :: Set() ;

break;

case TimingRelationships::MODE11:

SPIModule :: CR1 :: CPOL :: High :: Set() ;

SPIModule :: CR1 :: CPHA :: Phase2edge :: Set() ;

break;

default:

assert(false) ;

break ;

}

}

static void SetBIDImode(BIDImode bidimode) {

switch (bidimode) {

case BIDImode::LINE2:

SPIModule :: CR1 :: BIDIMODE :: Unidirectional2Line :: Set();

break ;

case BIDImode::LINE1:

SPIModule :: CR1 :: BIDIMODE :: Bidirectional1Line :: Set() ;

break ;

default:

assert(false) ;

break ;

}

}

static void SetCSmode(CSmode csmode) {

switch (csmode) {

case CSmode::AUTODIS:

SPIModule :: CR1 :: SSM :: NssSoftwareDisable :: Set();

SPIModule :: CR1 :: SSI :: Value0 :: Set();

break;

case CSmode::AUTOEN:

SPIModule :: CR1 :: SSM :: NssSoftwareDisable :: Set();

SPIModule :: CR1 :: SSI :: Value1 :: Set();

break;

case CSmode::SOFTDIS:

SPIModule :: CR1 :: SSM :: NssSoftwareEnable :: Set();

SPIModule :: CR1 :: SSI :: Value0 :: Set();

break;

case CSmode::SOFTEN:

SPIModule :: CR1 :: SSM :: NssSoftwareEnable :: Set();

SPIModule :: CR1 :: SSI :: Value1 :: Set();

break;

default:

assert(false) ;

break ;

}

}

static void SetCRC(CRCen crcen) {

switch (crcen) {

case CRCen::DISABLE:

SPIModule :: CR1 :: CRCEN :: CrcCalcDisable :: Set();

break ;

case CRCen::ENABLE:

SPIModule :: CR1 :: CRCEN :: CrcCalcEnable :: Set();

break ;

default:

assert(false) ;

break ;

}

}

};

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* End of file \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* | File : SPIConfig.hpp

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#pragma once

enum class Mode {

Master,

Slave

};

enum class DataFormat {

Bit8,

Bit16

};

enum class FrameFormat {

MSB,

LSB

};

enum class BaudRate {

DIV2,

DIV4,

DIV8,

DIV16,

DIV32,

DIV64,

DIV128,

DIV256

};

enum class TimingRelationships {

MODE00,

MODE01,

MODE10,

MODE11

};

enum class BIDImode {

LINE2,

LINE1

};

enum class CSmode {

AUTODIS,

AUTOEN,

SOFTDIS,

SOFTEN

};

enum class CRCen {

ENABLE,

DISABLE

};

struct SPIConfig

{

Mode mode;

DataFormat dataformat;

FrameFormat frameformat;

BaudRate baudrate;

TimingRelationships timingrelationships;

BIDImode bidimode;

CSmode csmode;

CRCen crcen;

};

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* End of file \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* | File : USART.hpp

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#pragma once

#include "USARTConfig.hpp" // for UsartConfig, tParity, tBitsSize

#include "usart2registers.hpp"

template <typename USARTModule, uint32\_t ClockSpeed>

class USART {

public:

static void On() {

USARTModule::CR1::UE::Enable::Set() ; //????????? USART2

USARTModule::CR1::TE::Enable::Set() ; //????????? ????????

}

static void Config(UsartConfig config) {

SetSamplingMode (config.samplingmode) ;

SetSpeed (config.speed) ;

SetStopBits (config.stopbits) ;

SetBitsSize (config.bitssize) ;

SetParity (config.parity) ;

}

static void SendData (const char\* ptr, size\_t size) {

assert(size > 0);

for (size\_t i = 0; i < size; i++) {

SendByte(\*ptr);

ptr++ ;

}

}

private:

static void SendByte(uint8\_t Byte) {

while (!USARTModule::SR::TXE::DataRegisterEmpty::IsSet()) {

} ;

USARTModule::DR::Write(Byte) ;

}

static void SetStopBits(StopBits stopbits) {

switch (stopbits) {

case StopBits::OneBit:

USARTModule::CR2::STOP::OneStopBit::Set() ;

break ;

case StopBits::TwoBits:

USARTModule::CR2::STOP::TwoStopBits::Set() ;

break ;

default:

assert(false) ;

break ;

}

}

static void SetBitsSize(BitsSize bitssize) {

switch (bitssize) {

case BitsSize::Bits8:

USARTModule::CR1::M::Data8bits::Set() ;

break ;

case BitsSize::Bits9:

USARTModule::CR1::M::Data9bits::Set() ;

break ;

default:

assert(false) ;

break ;

}

}

inline static uint32\_t over8 = 0 ;

static void SetSamplingMode(SamplingMode samplingmode) {

switch (samplingmode) {

case SamplingMode::Mode16:

USARTModule::CR1::OVER8::OversamplingBy16::Set() ;

over8 = 0 ;

break ;

case SamplingMode::Mode8:

USARTModule::CR1::OVER8::OversamplingBy8::Set() ;

over8 = 1 ;

break ;

default:

assert(false) ;

break ;

}

}

static void SetParity(Parity parity) {

switch (parity) {

case Parity::Even:

USARTModule::CR1::PCE::ParityControlEnable::Set() ;

USARTModule::CR1::PS::ParityEven::Set() ;

break ;

case Parity::Odd:

USARTModule::CR1::PCE::ParityControlEnable::Set() ;

USARTModule::CR1::PS::ParityOdd::Set() ;

break ;

case Parity::None:

USARTModule::CR1::PCE::ParityControlDisable::Set() ;

break ;

default:

assert(false) ;

break ;

}

}

static void SetSpeed(Speed speed) {

uint32\_t speednum ;

switch (speed) {

case Speed::Speed2400:

speednum = 2400 ;

break ;

case Speed::Speed4800:

speednum = 4800 ;

break ;

case Speed::Speed9600:

speednum = 9600 ;

break ;

case Speed::Speed19200:

speednum = 19200 ;

break ;

case Speed::Speed38400:

speednum = 38400 ;

break ;

default:

assert(false) ;

break ;

}

// 208 = 16000000/ (9600\*8\*(2-1)) ;

uint32\_t USARTDIVMANT = ClockSpeed/(speednum\*8\*(2-over8)) ;

uint32\_t USARTDIVFRACT = 16 \* (ClockSpeed % (speednum\*8\*(2-over8)))/(speednum\*8\*(2-over8));

USARTModule::BRR::Write((USARTDIVMANT<<4) | USARTDIVFRACT) ;

}

} ;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* End of file \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* | File : UsartConfig.hpp

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#pragma once

enum class SamplingMode {

Mode16 ,

Mode8 ,

} ;

enum class Parity {

Odd ,

Even ,

None ,

} ;

enum class BitsSize {

Bits8 ,

Bits9 ,

} ;

enum class Speed {

Speed2400 ,

Speed4800 ,

Speed9600 ,

Speed19200 ,

Speed38400 ,

} ;

enum class StopBits {

OneBit ,

TwoBits ,

} ;

struct UsartConfig {

Speed speed ;

BitsSize bitssize ;

Parity parity ;

StopBits stopbits ;

SamplingMode samplingmode ;

} ;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* End of file \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* | File : Button.hpp

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#pragma once

template<typename Pin>

class Button {

public:

Button() {

Pin :: SetInput();

}

bool IsPushed() {

return !Pin::IsSet() ;

} ;

} ;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* End of file \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* | File : ISubscriber.hpp

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#pragma once

struct ISubscriber {

virtual void HandleButtonPushed() = 0 ;

} ;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* End of file \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* | File : TaskButton.cpp

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "TaskButton.hpp"

TaskButton::TaskButton(ISubscriber& buttonsubscriber): subscriber(buttonsubscriber) {

}

void TaskButton::Execute() {

for(;;) {

if (userbutton.IsPushed()) {

subscriber.HandleButtonPushed() ;

}

Sleep(100ms) ;

}

};

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* End of file \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* | File : TaskButton.hpp

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#pragma once

#include "thread.hpp" //for thread

#include "ISubscriber.hpp" //for Subscriber

#include "Pin.hpp" //for Pin

#include "Button.hpp" //for Button

#include "gpiocregisters.hpp" //for gpiocregisters

class TaskButton : public OsWrapper::Thread<256> {

public:

TaskButton(ISubscriber& buttonsubscriber) ;

void Execute() override ;

private:

Button<Pin<GPIOC,13>> userbutton ;

ISubscriber& subscriber ;

} ;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* End of file \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* | File : BluetoothDriver.hpp

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#pragma once

#include <cstring> // for strlenght

#include "IBluetoothDriver.hpp" //for IBluetoothDriver

#include "Pin.hpp" //for TX

#include "USART.hpp" // for USART

#include "susudefs.hpp" // for SusuString

template <typename USART, typename RX, typename TX> // , typename TX

class BluetoothDriver: public IBluetoothDriver {

public:

BluetoothDriver() {

RX :: SetAlternate();

TX :: SetAlternate();

UsartConfig USART2Config ;

USART2Config.speed = Speed::Speed9600 ;

USART2Config.stopbits = StopBits::OneBit ;

USART2Config.bitssize = BitsSize::Bits8 ;

USART2Config.parity = Parity::None ;

USART2Config.samplingmode = SamplingMode::Mode8 ;

USART::Config(USART2Config) ;

USART::On() ;

}

void Send(SusuString<40>& message) override {

const char\* str = message.GetString() ;

auto size = std::strlen(str) ;

if (size <= message.Size) {

USART::SendData(message.GetString(), size) ;

}

}

} ;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* End of file \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* | File : IBluetoothDriver.hpp

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#pragma once

#include "susudefs.hpp" // for SusuStringView

struct IBluetoothDriver {

virtual void Send(SusuString<40>& message) = 0 ;

} ;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* End of file \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* | File : Bluetooth.hpp

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#pragma once

#include "susudefs.hpp" //for SusuStringView

#include "IBluetoothDriver.hpp" //for IBluetoothDriver

class Bluetooth {

private:

IBluetoothDriver& ibluetoothdriver ;

public:

Bluetooth (IBluetoothDriver& bluetoothdriver): ibluetoothdriver(bluetoothdriver) {

}

void Send (SusuString<40>& message) {

ibluetoothdriver.Send(message) ;

}

} ;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* End of file \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* | File : BluetoothDirector.hpp

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "BluetoothDirector.hpp"

BluetoothDirector::BluetoothDirector(Bluetooth& mybluetooth, SensorDirector& mysensordirector): bluetooth(mybluetooth), sensordirector(mysensordirector) {};

void BluetoothDirector::Execute() {

for (;;) {

Sleep(500ms) ;

tFormatData x = sensordirector.GetData();

auto f = Format::GetBlueToothFormat(x);

auto pres = std::get<0>(f) ;

bluetooth.Send(pres) ;

auto hum = std::get<1>(f) ;

bluetooth.Send(hum) ;

auto temp = std::get<2>(f) ;

bluetooth.Send(temp) ;

auto dewp = std::get<3>(f) ;

bluetooth.Send(dewp) ;

Sleep(500ms) ;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* End of file \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* | File : BluetoothDirector.cpp

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#pragma once

#include "Thread.hpp" // for Execute

#include "susudefs.hpp" //for susustring

#include "Bluetooth.hpp" //for blupup

#include "Format.hpp" //for tuple

#include "SensorDirector.hpp"

using namespace std ;

class BluetoothDirector: public OsWrapper::Thread<512> {

private:

Bluetooth& bluetooth ;

Format format ;

SensorDirector& sensordirector ;

public:

BluetoothDirector (Bluetooth& mybluetooth, SensorDirector& mysensordirector);

void Execute() override;

} ;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* End of file \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* | File : DisplayDriver.hpp

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#pragma once

#include "IDisplayDriver.hpp" //for IDisplayDriver

#include "SPIConfig.hpp" //for SPI::Config(SPIConfig spiconfig)

constexpr unsigned char LUT\_VCOM[] = {

0x00, 0x0E, 0x0E, 0x00, 0x00, 0x02,

0x00, 0x00, 0x00, 0x00, 0x00, 0x00,

0x00, 0x00, 0x00, 0x00, 0x00, 0x00,

0x00, 0x00, 0x00, 0x00, 0x00, 0x00,

0x00, 0x00, 0x00, 0x00, 0x00, 0x00,

0x00, 0x00, 0x00, 0x00, 0x00, 0x00,

0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00

};

constexpr unsigned char LUT\_W2W[] = {

0xA0, 0x0E, 0x00, 0x00, 0x00, 0x02,

0x00, 0x00, 0x00, 0x00, 0x00, 0x00,

0x00, 0x00, 0x00, 0x00, 0x00, 0x00,

0x00, 0x00, 0x00, 0x00, 0x00, 0x00,

0x00, 0x00, 0x00, 0x00, 0x00, 0x00,

0x00, 0x00, 0x00, 0x00, 0x00, 0x00,

0x00, 0x00, 0x00, 0x00, 0x00, 0x00

};

constexpr unsigned char LUT\_B2W[] = {

0xA0, 0x0E, 0x00, 0x00, 0x00, 0x02,

0x00, 0x00, 0x00, 0x00, 0x00, 0x00,

0x00, 0x00, 0x00, 0x00, 0x00, 0x00,

0x00, 0x00, 0x00, 0x00, 0x00, 0x00,

0x00, 0x00, 0x00, 0x00, 0x00, 0x00,

0x00, 0x00, 0x00, 0x00, 0x00, 0x00,

0x00, 0x00, 0x00, 0x00, 0x00, 0x00

};

constexpr unsigned char LUT\_B2B[] = {

0x50, 0x0E, 0x00, 0x00, 0x00, 0x02,

0x00, 0x00, 0x00, 0x00, 0x00, 0x00,

0x00, 0x00, 0x00, 0x00, 0x00, 0x00,

0x00, 0x00, 0x00, 0x00, 0x00, 0x00,

0x00, 0x00, 0x00, 0x00, 0x00, 0x00,

0x00, 0x00, 0x00, 0x00, 0x00, 0x00,

0x00, 0x00, 0x00, 0x00, 0x00, 0x00

};

constexpr unsigned char LUT\_W2B[] = {

0x50, 0x0E, 0x00, 0x00, 0x00, 0x02,

0x00, 0x00, 0x00, 0x00, 0x00, 0x00,

0x00, 0x00, 0x00, 0x00, 0x00, 0x00,

0x00, 0x00, 0x00, 0x00, 0x00, 0x00,

0x00, 0x00, 0x00, 0x00, 0x00, 0x00,

0x00, 0x00, 0x00, 0x00, 0x00, 0x00,

0x00, 0x00, 0x00, 0x00, 0x00, 0x00,

};

enum class CommandEInk : std::uint8\_t{

PanelSetting = 0x00,

PowerOff = 0x02,

PowerOn = 0x04,

BoosterSoftStart = 0x06,

DisplayStartTransmission1 = 0x10,

DisplayRefresh = 0x12,

DisplayStartTransmission2 = 0x13,

VCOMLUT = 0x20,

LUTW2W = 0x21,

LUTB2W = 0x22,

LUTW2B = 0x23,

LUTB2B = 0x24,

PLL = 0x30,

VcomDataIntervalSetting = 0x50,

ResolutionSetting = 0x61,

VCMDCSetting = 0x82,

PartialWindow = 0x90,

PartialIn = 0x91,

PartialOut = 0x92,

};

template <typename SPI, typename DIN, typename CLK, typename CS, typename DC,

typename RST, typename BUSY, int W, int H>

class DisplayDriver : public IDisplayDriver {

public :

DisplayDriver() {

SPIConfig SPI2Config;

SPI2Config.mode = Mode :: Master;

SPI2Config.dataformat = DataFormat :: Bit8;

SPI2Config.frameformat = FrameFormat :: MSB;

SPI2Config.baudrate = BaudRate :: DIV2;

SPI2Config.timingrelationships = TimingRelationships :: MODE11;

SPI2Config.bidimode = BIDImode :: LINE2;

SPI2Config.csmode = CSmode :: SOFTEN;

SPI2Config.crcen = CRCen :: DISABLE;

SPI :: Config(SPI2Config);

SPI:: Enable();

DIN :: SetAlternate();

CLK :: SetAlternate();

CS :: SetOutput();

DC :: SetOutput();

RST :: SetOutput();

BUSY :: SetInput();

}

void Init() override {

Reset(); //EPD\_4IN2BC\_Reset();

SendCommand(CommandEInk::BoosterSoftStart);//EPD\_4IN2BC\_SendCommand(0x06); // BOOSTER\_SOFT\_START

StartSendData();

SendData(0x17);//Booster Soft Start(BTST) BT\_PHA[7:0] SPI

SendData(0x17);//Booster Soft Start(BTST) BT\_PHB[7:0] SPI

SendData(0x17); //Booster Soft Start(BTST) BT\_PHC[5:0] SPI

EndSendData();

SendCommand(CommandEInk::PowerOff);

while(!BUSY::IsSet()) {for (int i = 0; i< 100000; i++) {};}; //EPD\_4IN2BC\_ReadBusy();//0: busy, 1: idle

SendCommand(CommandEInk::PowerOn);//EPD\_4IN2BC\_SendCommand(0x04); // POWER\_ON

while(!BUSY::IsSet()) {for (int i = 0; i< 100000; i++) {};}; //EPD\_4IN2BC\_ReadBusy();//0: busy, 1: idle

SendCommand(CommandEInk::PanelSetting);//EPD\_4IN2BC\_SendCommand(0x00); // PANEL\_SETTING

StartSendData();

SendData(0x3F); //Panel Setting (PSR) RES[1:0],REG,KW/R,UD,SHL,SHD\_N,RST\_N// LUT from OTP

EndSendData();

StartSendData();

SendCommand(CommandEInk::PLL); // PLL setting

SendData(0x3C); // 3A 100HZ 29 150Hz 39 200HZ 31 171HZ

SendCommand (CommandEInk :: VCMDCSetting);

StartSendData();

SendData(0x3A); //Display Refresh(DRF)

EndSendData();

SendCommand (CommandEInk :: VcomDataIntervalSetting);

StartSendData();

SendData(0x97); //Display Refresh(DRF)

EndSendData();

}

void Clear() override {

const std::uint8\_t BlackColor = 0x00;

SetResolution();

SendCommand(CommandEInk::DisplayStartTransmission1);

StartSendData();

for (int i = 0; i < W / 8 \* H; i ++) {

SendData(BlackColor);

}

EndSendData();

SendCommand(CommandEInk::DisplayStartTransmission2);

StartSendData();

for (int i = 0; i < W / 8 \* H; i ++) {

SendData(BlackColor);

}

EndSendData();

Refresh();

}

void Display(uint8\_t \*buff, size\_t lenght) override {

SendCommand(CommandEInk::PartialIn) ;

SendCommand(CommandEInk::PartialWindow) ;

StartSendData();

SendData(0 >> 8U) ;

SendData(0 & 0xf8) ;

SendData(((0 & 0xf8) + W - 1U) >> 8) ;

SendData(((0 & 0xf8) + W - 1U) | 0x07U) ;

SendData(0 >> 8) ;

SendData( 0 & 0xff) ;

SendData((0 + H - 1U) >> 8U) ;

SendData((0 + H - 1U) & 0xff) ;

SendData(0x01) ;

EndSendData();

SendCommand(CommandEInk::DisplayStartTransmission2);

StartSendData();

for (size\_t j = 0; j < lenght; ++j) {

SendData(buff[j]);

}

EndSendData();

Refresh();

}

private :

void SendCommand(CommandEInk command) {

DC :: Reset(); //DEV\_Digital\_Write(EPD\_DC\_PIN, 0);

CS :: Reset(); //DEV\_Digital\_Write(EPD\_CS\_PIN, 0);

SPI::WriteByteU(static\_cast<std::uint8\_t>(command)); //DEV\_SPI\_WriteByte(Reg);

CS::Set() ; // DEV\_Digital\_Write(EPD\_CS\_PIN, 1);

}

void SendData(std::uint8\_t data) {

// StartSendData();

SPI :: WriteByteU(data); //DEV\_SPI\_WriteByte(Data);

// EndSendData();//DEV\_Digital\_Write(EPD\_CS\_PIN, 1);

}

void Reset() {

RST :: Set(); //DEV\_Digital\_Write(EPD\_RST\_PIN, 1);

for (int i = 0; i < 1000000; i ++) {};

RST :: Reset(); //DEV\_Digital\_Write(EPD\_RST\_PIN, 0);

for (int i = 0; i < 1000000; i ++) {};

RST :: Set();//DEV\_Digital\_Write(EPD\_RST\_PIN, 1);

for (int i = 0; i < 1000000; i ++) {};

}

void Refresh() {

SetLut() ;

SendCommand(CommandEInk::DisplayRefresh); // EPD\_4IN2BC\_SendCommand(0x12); // DISPLAY\_REFRESH

while(!BUSY::IsSet()) {};//EPD\_4IN2BC\_ReadBusy();

}

void SetResolution() {

SendCommand (CommandEInk :: ResolutionSetting);

const std::uint8\_t WhiteColor = 0xff;

StartSendData();

SendData(W >> 8);

SendData(W & WhiteColor);

SendData(H >> 8);

SendData(H & WhiteColor);

EndSendData();

}

void StartSendData() {

DC :: Set() ;

CS :: Reset() ;

}

void EndSendData() {

CS :: Set() ;

}

void SetLut() {

unsigned int i;

SendCommand(CommandEInk::VCOMLUT);

StartSendData();

for (i = 0; i < 44; i ++) {

SendData(LUT\_VCOM[i]);

}

EndSendData();

SendCommand(CommandEInk::LUTW2W);

StartSendData();

for (i = 0; i < 42; i ++) {

SendData(LUT\_W2W[i]);

}

EndSendData();

SendCommand(CommandEInk::LUTB2W);

StartSendData();

for (i = 0; i < 42; i ++) {

SendData(LUT\_B2W[i]);

}

EndSendData();

SendCommand(CommandEInk::LUTW2B);

StartSendData();

for (i = 0; i < 42; i ++) {

SendData(LUT\_W2B[i]);

}

EndSendData();

SendCommand(CommandEInk::LUTB2B);

StartSendData();

for (i = 0; i < 42; i ++) {

SendData(LUT\_B2B[i]);

}

EndSendData();

}

};

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* End of file \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* | File : IDisplayDriver.hpp

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#pragma once

struct IDisplayDriver {

virtual void Init() = 0 ;

virtual void Clear() = 0 ;

virtual void Display(uint8\_t \*buff, size\_t lenght) = 0 ;

} ;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* End of file \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* | File : DisplayDirector.cpp

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "DisplayDirector.hpp"

#include <iostream>

DisplayDirector::DisplayDirector(IDisplayView& view, SensorDirector& mysensordirector ): myView(view), sensordirector(mysensordirector) { };

void DisplayDirector::Execute() {

for(;;){

Sleep(1500ms) ;

tFormatData x = sensordirector.GetData();

auto f = Format::GetDisplayFormat(x);

auto f0 = std::get<0>(f);

auto f1 = std::get<1>(f);

auto f2 = std::get<2>(f);

auto f3 = std::get<3>(f);

myView.Update(f0,f1,f2,f3);

Sleep(1500ms) ;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* End of file \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* | File : DisplayDirector.cpp

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#pragma once

#include "thread.hpp" //for thread

#include "IDisplayView.hpp"

#include "Format.hpp"

#include "susudefs.hpp"

#include "SensorDirector.hpp"

class DisplayDirector : public OsWrapper::Thread<512>{

public:

DisplayDirector(IDisplayView& view, SensorDirector& mysensordirector );

void Execute() override ;

private:

IDisplayView& myView;

SensorDirector& sensordirector ;

} ;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* End of file \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* | File : IDisplayView.hpp

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "IDisplayView.hpp"

#pragma once

class DisplayView : public IDisplayView {

public:

IDisplay& myDisplay;

DisplayView(IDisplay& display): myDisplay(display) { }

void Update(SusuString<40>& Pressure,SusuString<40>& Humidity,

SusuString<40>& Temperature, SusuString<40>& DewPoint) override {

myDisplay.ClearWindow() ;

DrawFirstString(Pressure);

DrawSecondString(Humidity);

DrawThirdString(Temperature);

DrawFourthString(DewPoint);

myDisplay.Update();

}

private:

void DrawFirstString(SusuString<40>& String) {

myDisplay.DrawString(10, 35, String, &Font24);

}

void DrawSecondString(SusuString<40>& String) {

myDisplay.DrawString(10, 100, String, &Font24);

}

void DrawThirdString(SusuString<40>& String) {

myDisplay.DrawString(10, 165, String, &Font24);

}

void DrawFourthString(SusuString<40>& String) {

myDisplay.DrawString(10, 230, String, &Font24);

}

};

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* End of file \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* | File : EInkDisplay.hpp

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#pragma once

#include "DisplayDriver.hpp"

#include "IDisplay.hpp"

#include <array> //for std::array

template <std::uint16\_t W, std::uint16\_t H>

class EInkDisplay : public IDisplay {

private:

static constexpr std::uint16\_t WidthByte = (W % 8 == 0)? (W / 8 ): (W / 8 + 1);

void DrawPoint(Point point, Color color) {

assert((point.X <= W) || (point.Y <= H)) ;

std::uint32\_t Addr = point.X / 8 + point.Y \* WidthByte;

std::uint32\_t Rdata = Canva[Addr];

if(color == Color::Black)

Canva[Addr] = Rdata & ~(0x80 >> (point.X % 8));

else

Canva[Addr] = Rdata | (0x80 >> (point.X % 8));

}

void DrawChar(uint16\_t x, uint16\_t y, const char symbol, sFONT\* Font) {

assert((x <= W) || (y <= H)) ;

uint16\_t Page, Column;

uint32\_t Char\_Offset = (symbol - ' ') \* Font->Height \* (Font->Width / 8 + (Font->Width % 8 ? 1 : 0));

const unsigned char \*ptr = &Font->table[Char\_Offset];

for (Page = 0; Page < Font->Height; Page ++ ) {

for (Column = 0; Column < Font->Width; Column ++ ) {

if (\*ptr & (0x80 >> (Column % 8))) {

DrawPoint( Point(x + Column, y + Page), Color::White);

}

else {

DrawPoint(Point(x + Column, y + Page),Color::Black);

}

if (Column % 8 == 7)

ptr++;

}

if (Font->Width % 8 != 0)

ptr++;

}

}

public:

unsigned char Canva[((W % 8 == 0)? (W/ 8 ): (W/ 8 + 1)) \* H];

IDisplayDriver& myDriver;

EInkDisplay(IDisplayDriver& driver): myDriver(driver) {

myDriver.Init();

}

void DrawString(uint16\_t x, uint16\_t y, SusuString<40>& String, sFONT\* Font) override {

const char\* str = String.GetString();

uint16\_t Xpoint = x;

uint16\_t Ypoint = y;

while (\* str != '\0') {

if ((Xpoint + Font->Width ) > W ) {

Xpoint = x;

Ypoint += Font->Height;

}

if ((Ypoint + Font->Height ) > H ) {

Xpoint = x;

Ypoint = y;

}

DrawChar(Xpoint, Ypoint, \* str, Font);

str ++;

Xpoint += Font->Width;

}

}

void Update() override {

myDriver.Display(Canva, sizeof(Canva));

}

void ClearWindow() override {

myDriver.Clear();

}

};

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* End of file \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* | File : IDisplay.hpp

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#pragma once

#include "susudefs.hpp"

#include "fonts.h"

struct Point

{

constexpr Point(std::uint16\_t xValue, std::uint16\_t yValue):

X(xValue),

Y(yValue)

{

}

std::uint16\_t X = 0U;

std::uint16\_t Y = 0U;

};

enum class Color: std::uint8\_t

{

Black = 0x00,

White = 0xFF

};

struct IDisplay

{

virtual void DrawString(uint16\_t x, uint16\_t y, SusuString<40>& String, sFONT\* Font) = 0 ;

virtual void Update() = 0 ;

virtual void ClearWindow() = 0 ;

} ;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* End of file \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* | File : IDisplayView.hpp

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#pragma once

struct IDisplayView {

virtual void Update(SusuString<40>& Pressure,SusuString<40>& Humidity,

SusuString<40>& Temperature, SusuString<40>& DewPoint) = 0 ;

} ;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* End of file \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* | File : Format.hpp

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#pragma once

#include "susudefs.hpp"

#include <iostream>

#include <string>

using tFormat = std::tuple<SusuString<40>,SusuString<40>,SusuString<40>,SusuString<40>> ;

using tFormatData = std::tuple<SusuString<5>, float,SusuString<5>, float, SusuString<5>, float,SusuString<5>, float> ;

class Format {

public:

static tFormat GetBlueToothFormat(const tFormatData& data) {

return std::make\_tuple(GetBlueToothPressureFormat(data), GetBlueToothHumidityFormat(data), GetBlueToothTemperatureFormat(data), GetBlueToothDewPointFormat(data));

}

static tFormat GetDisplayFormat(const tFormatData& data) {

return std::make\_tuple(GetDisplayPressureFormat(data), GetDisplayHumidityFormat(data), GetDisplayTemperatureFormat(data), GetDisplayDewPointFormat(data));

}

private:

static SusuString<40> GetBlueToothPressureFormat(const tFormatData& data) {

char str[40];

sprintf(str, "%s%.3f %s", "\*P", std::get<1>(data), "\*") ;

SusuString<40> pressure;

pressure.Set(str);

return pressure;

}

static SusuString<40> GetBlueToothHumidityFormat(const tFormatData& data) {

char str[40];

sprintf(str, "%s%.3f %s\n", "\*H", std::get<3>(data), "\*") ;

SusuString<40> humidity;

humidity.Set(str);

return humidity;

}

static SusuString<40> GetBlueToothTemperatureFormat(const tFormatData& data) {

char str[40];

sprintf(str, "%s%.3f %s%s%s\n", "\*T", std::get<5>(data),"\* \*V", (std::get<4>(data)).GetString(),"\*") ;

SusuString<40> temperature;

temperature.Set(str);

return temperature;

}

static SusuString<40> GetBlueToothDewPointFormat(const tFormatData& data) {

char str[40];

sprintf(str, "%s%.3f %s\n", "\*D", std::get<7>(data), "\*") ;

SusuString<40> dewpoint;

dewpoint.Set(str);

return dewpoint;

}

static SusuString<40> GetDisplayPressureFormat(const tFormatData& data) {

char str[40];

sprintf(str, "%s%.3f %s", "Pressure: ", std::get<1>(data), (std::get<0>(data)).GetString()) ;

SusuString<40> pressure;

pressure.Set(str);

return pressure;

}

static SusuString<40> GetDisplayHumidityFormat(const tFormatData& data) {

char str[40];

sprintf(str, "%s%.3f %s", "Humidity: ", std::get<3>(data), (std::get<2>(data)).GetString()) ;

SusuString<40> humidity;

humidity.Set(str);

return humidity;

}

static SusuString<40> GetDisplayTemperatureFormat(const tFormatData& data) {

char str[40];

sprintf(str, "%s%.3f %s", "Temperature: ", std::get<5>(data), (std::get<4>(data)).GetString()) ;

SusuString<40> temperature;

temperature.Set(str);

return temperature;

}

static SusuString<40> GetDisplayDewPointFormat(const tFormatData& data) {

char str[40];

sprintf(str, "%s%.3f %s", "DewPoint: ", std::get<7>(data), (std::get<6>(data)).GetString()) ;

SusuString<40> dewpoint;

dewpoint.Set(str);

return dewpoint;

}

};

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* End of file \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* | File : ISensorDriver.hpp

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#pragma once

struct ISensorDriver {

virtual void Init() = 0 ;

virtual float ReadTemperature() = 0 ;

virtual float ReadPressure() = 0 ;

virtual float ReadHumidity() = 0 ;

} ;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* End of file \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* | File : SensorDriver.hpp

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#pragma once

#include "ISensorDriver.hpp" //for ISensorDriver

#include "SPIConfig.hpp" //for SPI::Config(SPIConfig spiconfig)

#include <iostream>

// BME280 registers

constexpr uint8\_t ADDRESS = 0xEC; //BME280 I2C ADDRES (0x76<<1)

constexpr uint8\_t REG\_ID = 0xD0; //BME280 ID REGISTER

constexpr uint8\_t ID = 0x60; //BME280 I2C ID

constexpr uint8\_t REG\_SOFTRESET = 0xE0; //BME280 SOFT RESET REGISTER

constexpr uint8\_t SOFTRESET\_VALUE = 0xB6; //BME280 SOFT RESET VALUE

constexpr uint8\_t REG\_CTRL\_HUM = 0xF2; // Humidity measure control register

constexpr uint8\_t REGISTER\_STATUS = 0XF3; //BME280 STATUS REGISTER

constexpr uint8\_t REG\_CTRL\_MEAS = 0xF4; // Control register pressure and temperature measure

constexpr uint8\_t REG\_CONFIG = 0xF5; // Configuration register

constexpr uint8\_t REGISTER\_PRESSUREDATA = 0xF7;

constexpr uint8\_t REGISTER\_TEMPDATA = 0xFA;

constexpr uint8\_t REGISTER\_HUMIDDATA = 0xFD;

// Humidity oversampling control register (0xF2)

constexpr uint8\_t OSRS\_H\_MSK = 0x07;

constexpr uint8\_t OSRS\_H\_SKIP = 0x00;

constexpr uint8\_t OSRS\_H\_x1 = 0x01;

constexpr uint8\_t OSRS\_H\_x2 = 0x02;

constexpr uint8\_t OSRS\_H\_x4 = 0x03;

constexpr uint8\_t OSRS\_H\_x8 = 0x04;

constexpr uint8\_t OSRS\_H\_x16 = 0x05;

// Status register (0xF3)

constexpr uint8\_t STATUS\_MSK = 0x09;

constexpr uint8\_t STATUS\_MEASURING = 0x08;

constexpr uint8\_t STATUS\_IM\_UPDATE = 0x01;

// Temperature oversampling (osrs\_t [7:5]) (0xF4)

constexpr uint8\_t OSRS\_T\_MSK = 0xE0;

constexpr uint8\_t OSRS\_T\_SKIP = 0x00;

constexpr uint8\_t OSRS\_T\_x1 = 0x20;

constexpr uint8\_t OSRS\_T\_x2 = 0x40;

constexpr uint8\_t OSRS\_T\_x4 = 0x60;

constexpr uint8\_t OSRS\_T\_x8 = 0x80;

constexpr uint8\_t OSRS\_T\_x16 = 0xA0;

// Pressure oversampling (osrs\_p [4:2]) (0xF4)

constexpr uint8\_t OSRS\_P\_MSK = 0x1C;

constexpr uint8\_t OSRS\_P\_SKIP = 0x00;

constexpr uint8\_t OSRS\_P\_x1 = 0x04;

constexpr uint8\_t OSRS\_P\_x2 = 0x08;

constexpr uint8\_t OSRS\_P\_x4 = 0x0C;

constexpr uint8\_t OSRS\_P\_x8 = 0x10;

constexpr uint8\_t OSRS\_P\_x16 = 0x14;

// Sensor mode of the device (mode [1:0]) (0xF4)

constexpr uint8\_t MODE\_MSK = 0x03;

constexpr uint8\_t MODE\_SLEEP = 0x00;

constexpr uint8\_t MODE\_FORCED = 0x01;

constexpr uint8\_t MODE\_NORMAL = 0x03;

// Configuration register: set rate, filter and interface options (0xF5)

// Inactive duration in normal mode (t\_sb [7:5]) (0xF5)

constexpr uint8\_t STBY\_MSK = 0xE0;

constexpr uint8\_t STBY\_0\_5 = 0x00;

constexpr uint8\_t STBY\_62\_5 = 0x20;

constexpr uint8\_t STBY\_125 = 0x40;

constexpr uint8\_t STBY\_250 = 0x60;

constexpr uint8\_t STBY\_500 = 0x80;

constexpr uint8\_t STBY\_1000 = 0xA0;

constexpr uint8\_t STBY\_10 = 0xC0;

constexpr uint8\_t STBY\_20 = 0xE0;

// Time constant of the IIR filter (filter [4:2]) (0xF5)

constexpr uint8\_t FILTER\_MSK = 0x1C;

constexpr uint8\_t FILTER\_OFF = 0x00;

constexpr uint8\_t FILTER\_2 = 0x04;

constexpr uint8\_t FILTER\_4 = 0x08;

constexpr uint8\_t FILTER\_8 = 0x0C;

constexpr uint8\_t FILTER\_16 = 0x10;

// Compensation parameter storage

constexpr uint8\_t REGISTER\_DIG\_T1 = 0x88;

constexpr int8\_t REGISTER\_DIG\_T2 = 0x8A;

constexpr int8\_t REGISTER\_DIG\_T3 = 0x8C;

constexpr uint8\_t REGISTER\_DIG\_P1 = 0x8E;

constexpr int8\_t REGISTER\_DIG\_P2 = 0x90;

constexpr int8\_t REGISTER\_DIG\_P3 = 0x92;

constexpr int8\_t REGISTER\_DIG\_P4 = 0x94;

constexpr int8\_t REGISTER\_DIG\_P5 = 0x96;

constexpr int8\_t REGISTER\_DIG\_P6 = 0x98;

constexpr int8\_t REGISTER\_DIG\_P7 = 0x9A;

constexpr int8\_t REGISTER\_DIG\_P8 = 0x9C;

constexpr int8\_t REGISTER\_DIG\_P9 = 0x9E;

constexpr uint8\_t REGISTER\_DIG\_H1 = 0xA1;

constexpr int8\_t REGISTER\_DIG\_H2 = 0xE1;

constexpr uint8\_t REGISTER\_DIG\_H3 = 0xE3;

constexpr int8\_t REGISTER\_DIG\_H4 = 0xE4;

constexpr int8\_t REGISTER\_DIG\_H5 = 0xE5;

constexpr int8\_t REGISTER\_DIG\_H6 = 0xE7;

// Compensation parameter structure

struct {

uint16\_t dig\_T1;

int16\_t dig\_T2;

int16\_t dig\_T3;

uint16\_t dig\_P1;

int16\_t dig\_P2;

int16\_t dig\_P3;

int16\_t dig\_P4;

int16\_t dig\_P5;

int16\_t dig\_P6;

int16\_t dig\_P7;

int16\_t dig\_P8;

int16\_t dig\_P9;

uint8\_t dig\_H1;

int16\_t dig\_H2;

uint8\_t dig\_H3;

int16\_t dig\_H4;

int16\_t dig\_H5;

uint8\_t dig\_H6;

} CalibData;

template <typename SPI, typename MOSI, typename SCK,

typename MISO, typename CS>

class SensorDriver : public ISensorDriver {

public :

int32\_t temper\_int;

SensorDriver() {

MOSI :: SetAlternate();

SCK :: SetAlternate();

MISO :: SetAlternate();

CS :: SetOutput();

SPIConfig SPI4Config;

SPI4Config.mode = Mode :: Master;

SPI4Config.dataformat = DataFormat :: Bit8;

SPI4Config.frameformat = FrameFormat :: MSB;

SPI4Config.baudrate = BaudRate :: DIV16;

SPI4Config.timingrelationships = TimingRelationships :: MODE11;

SPI4Config.bidimode = BIDImode :: LINE2;

SPI4Config.csmode = CSmode :: SOFTEN;

SPI4Config.crcen = CRCen :: DISABLE;

SPI :: Config(SPI4Config);

SPI :: Enable();

}

void Init() override {

uint32\_t value32=0;

WriteReg(REG\_SOFTRESET,SOFTRESET\_VALUE);

while (ReadStatus() & STATUS\_IM\_UPDATE) ;

ReadCoefficients();

SetStandby(STBY\_1000);

SetFilter(FILTER\_4);

SetOversamplingTemper(OSRS\_T\_x4);

SetOversamplingPressure(OSRS\_P\_x2);

SetOversamplingHum(OSRS\_H\_x1);

value32 = ReadReg8U(REG\_CTRL\_MEAS);

value32 |= ReadReg8U(REG\_CTRL\_HUM) << 8;

SetMode(MODE\_NORMAL);

}

float ReadTemperature() override {

float temper\_float = 0.0f;

uint32\_t temper\_raw;

int32\_t val1, val2;

temper\_raw = ((ReadReg8U(REGISTER\_TEMPDATA) << 16U) | (ReadReg8U(REGISTER\_TEMPDATA + 0x01) << 8U) | (ReadReg8U(REGISTER\_TEMPDATA + 0x02))) & 0x00FFFFFF;

temper\_raw >>= 4;

val1 = ((((temper\_raw>>3) - ((int32\_t)CalibData.dig\_T1 <<1))) \*

((int32\_t)CalibData.dig\_T2)) >> 11;

val2 = (((((temper\_raw>>4) - ((int32\_t)CalibData.dig\_T1)) \*

((temper\_raw>>4) - ((int32\_t)CalibData.dig\_T1))) >> 12) \*

((int32\_t)CalibData.dig\_T3)) >> 14;

temper\_int = val1 + val2;

temper\_float = ((temper\_int \* 5 + 128) >> 8);

temper\_float /= 100.0f;

return temper\_float;

}

float ReadPressure() override {

float press\_float = 0.0f;

uint32\_t press\_raw, pres\_int;

int64\_t val1, val2, p;

ReadTemperature();

press\_raw = ((ReadReg8U(REGISTER\_PRESSUREDATA) << 16U) | (ReadReg8U(REGISTER\_PRESSUREDATA + 0x01) << 8U) | (ReadReg8U(REGISTER\_PRESSUREDATA + 0x02))) & 0x00FFFFFF;

press\_raw >>= 4;

val1 = ((int64\_t) temper\_int) - 128000;

val2 = val1 \* val1 \* (int64\_t)CalibData.dig\_P6;

val2 = val2 + ((val1 \* (int64\_t)CalibData.dig\_P5) << 17);

val2 = val2 + ((int64\_t)CalibData.dig\_P4 << 35);

val1 = ((val1 \* val1 \* (int64\_t)CalibData.dig\_P3) >> 8)

+ ((val1 \* (int64\_t)CalibData.dig\_P2) << 12);

val1 = (((((int64\_t)1) << 47) + val1)) \* ((int64\_t)CalibData.dig\_P1) >> 33;

if (val1 == 0) {

return 0; // avoid exception caused by division by zero

}

p = 1048576 - press\_raw;

p = (((p << 31) - val2) \* 3125) / val1;

val1 = (((int64\_t)CalibData.dig\_P9) \* (p >> 13) \* (p >> 13)) >> 25;

val2 = (((int64\_t)CalibData.dig\_P8) \* p) >> 19;

p = ((p + val1 + val2) >> 8) + ((int64\_t)CalibData.dig\_P7 << 4);

pres\_int = ((p >> 8) \* 1000) + (((p & 0xff) \* 390625) / 100000);

press\_float = pres\_int / 100.0f;

return press\_float;

}

float ReadHumidity() override {

float hum\_float = 0.0f;

int16\_t hum\_raw;

int32\_t hum\_raw\_sign, v\_x1\_u32r;

ReadTemperature();

hum\_raw = ((ReadReg8U(REGISTER\_HUMIDDATA) << 8U) | (ReadReg8U(REGISTER\_HUMIDDATA + 0x01)));

hum\_raw\_sign = ((int32\_t)hum\_raw)&0x0000FFFF;

v\_x1\_u32r = (temper\_int - ((int32\_t)76800));

v\_x1\_u32r = (((((hum\_raw\_sign << 14) - (((int32\_t)CalibData.dig\_H4) << 20) -

(((int32\_t)CalibData.dig\_H5) \* v\_x1\_u32r)) + ((int32\_t)16384)) >> 15) \*

(((((((v\_x1\_u32r \* ((int32\_t)CalibData.dig\_H6)) >> 10) \*

(((v\_x1\_u32r \* ((int32\_t)CalibData.dig\_H3)) >> 11) + ((int32\_t)32768))) >> 10) +

((int32\_t)2097152)) \* ((int32\_t)CalibData.dig\_H2) + 8192) >> 14));

v\_x1\_u32r = (v\_x1\_u32r - (((((v\_x1\_u32r >> 15) \* (v\_x1\_u32r >> 15)) >> 7) \*

((int32\_t)CalibData.dig\_H1)) >> 4));

v\_x1\_u32r = (v\_x1\_u32r < 0) ? 0 : v\_x1\_u32r;

v\_x1\_u32r = (v\_x1\_u32r > 419430400) ? 419430400 : v\_x1\_u32r;

hum\_float = (v\_x1\_u32r>>12);

hum\_float /= 1024.0f;

return hum\_float;

}

private :

uint8\_t ReadStatus() {

uint8\_t res = ReadReg8U(REGISTER\_STATUS)&0x09;

return res;

}

void ReadCoefficients() {

CalibData.dig\_T1 = ReadReg16U(REGISTER\_DIG\_T1);

CalibData.dig\_T2 = ReadReg16S(REGISTER\_DIG\_T2);

CalibData.dig\_T3 = ReadReg16S(REGISTER\_DIG\_T3);

CalibData.dig\_P1 = ReadReg16U(REGISTER\_DIG\_P1);

CalibData.dig\_P2 = ReadReg16S(REGISTER\_DIG\_P2);

CalibData.dig\_P3 = ReadReg16S(REGISTER\_DIG\_P3);

CalibData.dig\_P4 = ReadReg16S(REGISTER\_DIG\_P4);

CalibData.dig\_P5 = ReadReg16S(REGISTER\_DIG\_P5);

CalibData.dig\_P6 = ReadReg16S(REGISTER\_DIG\_P6);

CalibData.dig\_P7 = ReadReg16S(REGISTER\_DIG\_P7);

CalibData.dig\_P8 = ReadReg16S(REGISTER\_DIG\_P8);

CalibData.dig\_P9 = ReadReg16S(REGISTER\_DIG\_P9);

CalibData.dig\_H1 = ReadReg8U(REGISTER\_DIG\_H1);

CalibData.dig\_H2 = ReadReg16S(REGISTER\_DIG\_H2);

CalibData.dig\_H3 = ReadReg8U(REGISTER\_DIG\_H3);

CalibData.dig\_H4 = ((ReadReg8S(REGISTER\_DIG\_H4) \* 16) | (ReadReg8S(REGISTER\_DIG\_H4+0x01) & 0xF));

CalibData.dig\_H5 = ((ReadReg8S(REGISTER\_DIG\_H5+0x01) \* 16) | (ReadReg8S(REGISTER\_DIG\_H5) >> 4));

CalibData.dig\_H6 = ReadReg8S(REGISTER\_DIG\_H6);

}

void SetStandby(uint8\_t tsb) {

uint8\_t reg;

reg = ReadReg8U(REG\_CONFIG) & ~STBY\_MSK;

reg |= tsb & STBY\_MSK;

WriteReg(REG\_CONFIG,reg);

}

void SetFilter(uint8\_t filter) {

uint8\_t reg;

reg = ReadReg8U(REG\_CONFIG) & ~FILTER\_MSK;

reg |= filter & FILTER\_MSK;

WriteReg(REG\_CONFIG,reg);

}

void SetOversamplingTemper(uint8\_t osrs) {

uint8\_t reg;

reg = ReadReg8U(REG\_CTRL\_MEAS) & ~OSRS\_T\_MSK;

reg |= osrs & OSRS\_T\_MSK;

WriteReg(REG\_CTRL\_MEAS,reg);

}

void SetOversamplingPressure(uint8\_t osrs) {

uint8\_t reg;

reg = ReadReg8U(REG\_CTRL\_MEAS) & ~OSRS\_P\_MSK;

reg |= osrs & OSRS\_P\_MSK;

WriteReg(REG\_CTRL\_MEAS,reg);

}

void SetOversamplingHum(uint8\_t osrs) {

uint8\_t reg;

reg = ReadReg8U(REG\_CTRL\_HUM) & ~OSRS\_H\_MSK;

reg |= osrs & OSRS\_H\_MSK;

WriteReg(REG\_CTRL\_HUM,reg);

reg = ReadReg8U(REG\_CTRL\_MEAS);

WriteReg(REG\_CTRL\_MEAS,reg);

}

void SetMode(uint8\_t mode) {

uint8\_t reg;

reg = ReadReg8U(REG\_CTRL\_MEAS) & ~MODE\_MSK;

reg |= mode & MODE\_MSK;

WriteReg(REG\_CTRL\_MEAS,reg);

}

int8\_t WriteReg(uint8\_t reg\_addr, uint8\_t reg\_data) {

int8\_t rslt = 0;

CS :: Set() ;

CS :: Reset() ;

SPI :: WriteByteU(reg\_addr & 127);

SPI :: WriteByteU(reg\_data);

CS :: Set() ;

return rslt;

}

int8\_t ReadReg8S(int8\_t RegAddr) {

int8\_t rslt;

CS :: Set() ;

CS :: Reset() ;

SPI :: WriteByteS(RegAddr);

rslt = (SPI::ReadByteS()).first ;

SPI :: WriteByteS(RegAddr);

rslt = (SPI::ReadByteS()).first ;

CS :: Set() ;

return rslt;

}

uint8\_t ReadReg8U(uint8\_t RegAddr) {

uint8\_t rslt;

CS :: Set() ;

CS :: Reset() ;

SPI :: WriteByteU(RegAddr);

rslt = (SPI::ReadByteU()).first ;

SPI :: WriteByteU(RegAddr);

rslt = (SPI::ReadByteU()).first ;

CS :: Set() ;

return rslt;

}

uint16\_t ReadReg16U(uint8\_t RegAddr) {

uint8\_t buf [2] ;

assert(buf != nullptr) ;

buf[0] = ReadReg8U(RegAddr);

RegAddr = RegAddr + 0x01U;

buf[1] = ReadReg8U(RegAddr);

uint16\_t result = \*reinterpret\_cast<uint16\_t\*>(buf) ;

return result ;

}

int16\_t ReadReg16S(int8\_t RegAddr) {

int8\_t buf [2] ;

assert(buf != nullptr) ;

buf[0] = ReadReg8S(RegAddr);

RegAddr = RegAddr + 0x01;

buf[1] = ReadReg8S(RegAddr);

int16\_t result = \*reinterpret\_cast<int16\_t\*>(buf) ;

return result;

}

};

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* End of file \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* | File : Celsius.hpp

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#pragma once

#include "IUnits.hpp"

class Celsius : public IUnits {

public:

vFormat GetTemperature(float code) const override {

float value = code;

SusuString<5> name;

name.Set("C ");

return std::make\_pair(name,value) ;

}

} ;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* End of file \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* | File : Fahrenheit.hpp

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#pragma once

#include "IUnits.hpp"

class Fahrenheit : public IUnits {

public:

vFormat GetTemperature(float code) const override {

float value = (code \* 1.8F ) + 32.0F;

SusuString<5> name;

name.Set("F ");

return std::make\_pair(name,value) ;

}

} ;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* End of file \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* | File : Kelvin.hpp

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#pragma once

#include "IUnits.hpp"

class Kelvin : public IUnits {

public:

vFormat GetTemperature(float code) const override {

float value = code + 273.15F;

SusuString<5> name;

name.Set("K ");

return std::make\_pair(name,value) ;

}

} ;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* End of file \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* | File : IUnits.hpp

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#pragma once

#include "susudefs.hpp"

#include "IVariable.hpp"

struct IUnits {

virtual vFormat GetTemperature(float code) const = 0 ;

} ;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* End of file \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* | File : DewPoint.hpp

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#pragma once

#include <iostream>

#include <stdarg.h>

#include <cmath>

#include "IVariable.hpp"

static struct {

SusuString<5> dpname;

float dp;

} DPData;

class DewPoint: public IVariable {

public:

DewPoint(){}

vFormat Get() override {

return std::make\_pair(DPData.dpname, DPData.dp);

}

void Calculate(float data, ...) override {

va\_list args;

va\_start(args,data);

float T = data;

float H = va\_arg(args,float);

float a = 17.27F;

float b = 237.7F;

float f = ((a \* T)/(b + T)) + log(H/100);

DPData.dp = (b \* f)/(a - f);

char str[5];

sprintf(str, "%s","C ") ;

DPData.dpname.Set(str);

}

};

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* End of file \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* | File : Humidity.hpp

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#pragma once

#include "IVariable.hpp"

static struct {

SusuString<5> hname;

float h;

} HData;

class Humidity: public IVariable {

public:

Humidity() {} ;

vFormat Get() override {

return std::make\_pair(HData.hname, HData.h);

}

void Calculate(float data, ...) override {

SusuString<5> name;

char str[5];

sprintf(str, "%s","%") ;

HData.hname.Set(str);

HData.h = data;

}

};

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* End of file \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* | File : Pressure.hpp

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#pragma once

#include "IVariable.hpp"

static struct {

SusuString<5> pname;

float p;

} PData;

class Pressure: public IVariable {

public:

Pressure() {} ;

vFormat Get() override {

return std::make\_pair(PData.pname, PData.p);

}

void Calculate(float data, ...) override {

char str[5];

sprintf(str, "%s","mmHg") ;

PData.pname.Set(str);

PData.p = data;

}

};

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* End of file \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* | File : Temperature.hpp

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#pragma once

#include "IVariable.hpp"

#include "IUnits.hpp"

static struct {

SusuString<5> tname;

float t;

} TData;

template <const auto&... units>

class Temperature: public IVariable {

private:

static constexpr size\_t UnitsCounts = sizeof...(units) ;

std::array<const IUnits\*, UnitsCounts> unitsList = { &units...};

std::size\_t index = 0U;

public:

Temperature() {} ;

void SetNextUnits() {

(index < UnitsCounts) ? index++ : index = 0U ;

if (index == 3) {

index = 0U;

}

}

vFormat Get() override {

return std::make\_pair(TData.tname, TData.t);

}

void Calculate(float data, ...) override {

auto const& currentUnits = \*unitsList[index] ;

vFormat temperature = currentUnits.GetTemperature(data) ;

SusuString<5> name;

char str[5];

sprintf(str, "%s", (temperature.first).GetString()) ;

TData.tname.Set(str);

TData.t = temperature.second;

}

};

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* End of file \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* | File : IVariable.hpp

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#pragma once

using vFormat = std::pair<SusuString<5U>, float>;

struct IVariable {

virtual vFormat Get() = 0 ;

virtual void Calculate(float data, ...) = 0;

} ;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* End of file \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* | File : BME280.hpp

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#pragma once

#include "susudefs.hpp" //for SusuStringView

#include "ISensorDriver.hpp" //for IBluetoothDriver

class BME280 {

private:

ISensorDriver& mySensorDriver ;

public:

BME280(ISensorDriver& sensordriver): mySensorDriver(sensordriver) {

mySensorDriver.Init();

}

float GetTemperature() {

return mySensorDriver.ReadTemperature();

}

float GetPressure() {

return mySensorDriver.ReadPressure()\* 0.000750061683f;

}

float GetHumidity() {

return mySensorDriver.ReadHumidity();

}

} ;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* End of file \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* | File : SensorDirector.cpp

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "SensorDirector.hpp"

#include <iostream>

SensorDirector::SensorDirector(BME280& sensor) : mySensor(sensor) {};

void SensorDirector::Execute() {

for(;;) {

temperature.Calculate(mySensor.GetTemperature()) ;

pressure.Calculate(mySensor.GetPressure());

humidity.Calculate(mySensor.GetHumidity());

dewpoint.Calculate(mySensor.GetTemperature(),mySensor.GetHumidity());

}

}

void SensorDirector::HandleButtonPushed() {

temperature.SetNextUnits();

}

tFormatData SensorDirector::GetData() {

return std::make\_tuple((pressure.Get()).first,(pressure.Get()).second,

(humidity.Get()).first, (humidity.Get()).second,

(temperature.Get()).first, (temperature.Get()).second ,

(dewpoint.Get()).first, (dewpoint.Get()).second );

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* End of file \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "IVariable.hpp"

#include "IUnits.hpp"

#include "Kelvin.hpp"

#include "Fahrenheit.hpp"

#include "Celsius.hpp"

#include "Temperature.hpp"

#include "Pressure.hpp"

#include "Humidity.hpp"

#include "DewPoint.hpp"

#include "susudefs.hpp"

#include "Format.hpp"

#include "BME280.hpp"

class SensorDirector : public OsWrapper::Thread<512>, public ISubscriber {

public:

SensorDirector(BME280& sensor) ;

void Execute() override ;

void HandleButtonPushed() override ;

tFormatData GetData() ;

private:

static constexpr Fahrenheit fahrenheit = Fahrenheit();

static constexpr Kelvin kelvin = Kelvin();

static constexpr Celsius celsius = Celsius();

Temperature<fahrenheit, kelvin, celsius> temperature;

Pressure pressure;

Humidity humidity;

DewPoint dewpoint;

BME280& mySensor;

} ;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* End of file \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/