Министерство науки и высшего образования Российской Федерации

**Федеральное государственное автономное   
образовательное учреждение высшего образования   
«Южно-Уральский государственный университет   
(национальный исследовательский университет)»**

**Высшая школа электроники и компьютерных наук**

**Кафедра «Информационно-измерительная техника»**

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОМУ ПРОЕКТУ**

**по дисциплине «Программное обеспечение измерительных процессов»**

**ЮУрГУ – 270304.2021.308-103.ПЗ КП**

**Нормоконтролер Руководитель, доцент**

**С.В. Колодий С.В. Колодий**

**2021 г. 2021 г.**

**Автор работы**

**Студент группы КЭ-413**

**Н.И. Циоплиакис**

**2021 г.**

**Проект защищён с оценкой**

**2021 г.**

Министерство науки и высшего образования Российской Федерации

**Федеральное государственное автономное   
образовательное учреждение высшего образования   
«Южно-Уральский государственный университет   
(национальный исследовательский университет)»**

**Высшая школа электроники и компьютерных наук**

**Кафедра «Информационно-измерительная техника»**

**УТВЕРЖДАЮ**

**Заведующий кафедрой ИнИТ**

**А.Л. Шестаков**

**2021 г.**

**ЗАДАНИЕ**

**на курсовой проект студента**

**Циоплиакиса Николаоса Илиаса**

**Группа КЭ-413**

* **Дисциплина: Программное обеспечение измерительных процессов**
* **Тема проекта: Разработка метеостанции с передачей параметров по беспроводному интерфейсу**
* **Срок сдачи студентом законченного проекта: 21 мая 2021 г.**
* Перечень вопросов, подлежащих разработке:

Разработать устройство, способное измерять давление, влажность и температуру и рассчитывать точку росы и передавать эти данные по USART.

* Для разработки должна использоваться отладочная плата XNUCLEO-F411RE
* Программное обеспечение измерение должно измерять давление, влажность и температуру и рассчитывать точку росы
  + Период измерения должен быть 100 ms.
  + Для измерения давления, влажности и температуры должен использоваться датчик BME280
  + Общение с датчиком должно осуществляться по интерфейсу SPI1
  + Точка росы должна считаться на основе текущих показаний влажности и давления
* Вывод значений давления, влажности и температуры должен производиться на экран с жидкими чернилами
  + Общение с экраном должно осуществляться через интерфейс SPI2
  + Период вывода информации на экран должен быть 3 секунды.
  + Формат вывода:

"Давление: " XXX.XX [Units]  
"Влажность: " XXX.XX [Units]  
"Температура: " XXX.XX [Units]  
"Точка росы": " XXX.XX [Units]

* Передача значений по беспроводному интерфейсу должна осуществляться через модуль BlueTooth Bee HC-06
  + Для подключения модуля BlueTooth должна использоваться плата Accessories Shield или I/O Expansion Shield
  + Период вывода информации через BlueTooth модуль должен быть 1 секунда.
  + Общение с платой расширения должно осуществляться через USART2
  + Формат вывода:

"Давление: " XXX.XX [Units]  
"Влажность: " XXX.XX [Units]  
"Температура: " XXX.XX [Units]  
"Точка росы": " XXX.XX [Units]

* Архитектура должна быть представлена в виде UML диаграмм в пакете Star UML
* Приложение должно быть написано на языке С++ с использование компилятора ARM 8.40.2
* При разработке должна использоваться Операционная Система Реального Времени FreeRTOS и С++ обертка над ней
* По нажатию кнопки USER на плате XNUCLEO-F411RE единцы измерения температуры должны изменяться в следующей циклической последовательности F→K→C.

Календарный план

|  |  |  |
| --- | --- | --- |
| **Наименование разделов курсовой работы** | **Срок выполнения разделов работы** | **Отметка руководителя о выполнении** |
| Разработка общей архитектуры программы | 28 марта 2021 г. |  |
| Разработка кода каркаса программы | 4 апреля 2021 г. |  |
| Разработка детальной архитектуры модуля работы с датчиком | 11 апреля 2021 г. |  |
| Разработка кода для модуля работы с датчиком | 11 апреля 2021 г. |  |
| Разработка детальной архитектуры модуля работы с индикатором | 18 апреля 2021 г. |  |
| Разработка кода для модуля работы с индикатором | 18 апреля 2021 г. |  |
| Разработка детальной архитектуры модуля работы с USART и блутуз | 25 апреля 2021 г. |  |
| Разработка кода для модуля работы с USART и блутуз | 25 апреля 2021 г. |  |
| Разработка детальной архитектуры и кода для оставшихся модулей | 2 мая 2021 г. |  |
| Сдача и демонстрация работы устройства | 9 мая 2021 г. |  |
| Оформление пояснительной записки к курсовой работе | 20 мая 2021 г. |  |

Руководитель проекта / C.В. Колодий/

Студент /Н.И. Циоплиакис /

АННОТАЦИЯ

Циоплиакис Н.И. «**Разработка метеостанции с передачей параметров по беспроводному интерфейсу**» – Челябинск: , КЭ-413,  с., 15 ил., библиогр. cписок – 4 наименований.

В данной курсовой работе было разработано устройство для измерения температуры, влажности, давления и расчета точки росы на основе датчика BME280. Код программы реализован в IAR Embedded Workbench. Исходными данными является техническое задание. В ходе разработки были выполнены следующие задачи:

1. Анализ требований технического задания.
2. Разработка общей и детальной архитектуры проекта.
3. Формирование программы для микроконтроллера.
4. Оформление пояснительной записки к курсовому проекту.

Для оформления отчета использовался текстовый редактор MS Word 2016.

ОГЛАВЛЕНИЕ

[1 АНАЛИЗ ТРЕБОВАНИЙ ТЕХНИЧЕСКОГО ЗАДАНИЯ 10](#_Toc72683672)

[1.1 Для разработки должна использоваться отладочная плата XNUCLEO-F411RE 10](#_Toc72683673)

[1.2 Программное обеспечение измерение должно измерять давление, влажность и температуру и рассчитывать точку росы 11](#_Toc72683674)

[1.2.1 Период измерения должен быть 100 ms. 11](#_Toc72683675)

[1.2.2 Для измерения давления, влажности и температуры должен использоваться датчик BME280 12](#_Toc72683676)

[1.2.3 Общение с датчиком должно осуществляться по интерфейсу SPI1 12](#_Toc72683677)

[1.2.4 Точка росы должна считаться на основе текущих показаний влажности и давления 13](#_Toc72683678)

[1.3 Вывод значений давления, влажности и температуры должен производиться на экран с жидкими чернилами 13](#_Toc72683679)

[1.4 Общение с экраном должно осуществляться через интерфейс SPI2 13](#_Toc72683680)

[1.4.1 Период вывода информации на экран должен быть 3 секунды. 14](#_Toc72683681)

[1.4.2 Формат вывода: 14](#_Toc72683682)

[1.5 Передача значений по беспроводному интерфейсу должна осуществляться через модуль BlueTooth Bee HC-06 14](#_Toc72683683)

[1.5.1 Для подключения модуля BlueTooth должна использоваться плата Accessories Shield или I/O Expansion Shield 14](#_Toc72683684)

[1.5.2 Период вывода информации через BlueTooth модуль должен быть 1 секунда 15](#_Toc72683685)

[1.5.3 Общение с платой расширения должно осуществляться через USART2 15](#_Toc72683686)

[1.5.4 Формат вывода: 15](#_Toc72683687)

[1.6 Архитектура должна быть представлена в виде UML диаграмм в пакете Star UML 15](#_Toc72683688)

[1.7 Приложение должно быть написано на языке С++ с использование компилятора ARM 8.40.2 15](#_Toc72683689)

[1.8 При разработке должна использоваться Операционная Система Реального Времени FreeRTOS и С++ обертка над ней 16](#_Toc72683690)

[1.9 По нажатию кнопки USER на плате XNUCLEO-F411RE единцы измерения температуры должны изменяться в следующей циклической последовательности F→K→C. 16](#_Toc72683691)

[2 Функциональная схема устройства 17](#_Toc72683692)

[3 АРХИТЕКТУРА ПРОЕКТА 18](#_Toc72683693)

[3.1 Общая архитектура проекта 18](#_Toc72683694)

[3.2 Детальная архитектура 18](#_Toc72683695)

[3.3 Класс Measurement 18](#_Toc72683696)

[3.4 Класс BME280Driver 21](#_Toc72683697)

[3.5 Классы USART, UsartDriver 26](#_Toc72683698)

[3.6 Классы SPI, SpiDriver, SpiDriverOnlyTransmit 27](#_Toc72683699)

[3.7 Класс EPDDriver 28](#_Toc72683700)

[3.8 Класс ScreenDriver 29](#_Toc72683701)

[3.9 Класс Display 31](#_Toc72683702)

[3.10 Классы Port, Pin, Button 31](#_Toc72683703)

[3.11 Классы BluetoothUpdateTask, DisplayUpdateTask, ButtonPressTask, MeasurementTask 32](#_Toc72683704)

[4 ДЕМОНСТРАЦИЯ РАБОТЫ ПРОЕКТА 33](#_Toc72683705)

[4.1 Демонстрация работы дисплея на электронных чернилах 33](#_Toc72683706)

[4.2 Демонстрация работы Bluetooth-модуля 33](#_Toc72683707)

[4.3 Демонстрация работы кнопки выбора единиц измерения температурных величин 34](#_Toc72683708)

[ЗАКЛЮЧЕНИЕ 35](#_Toc72683709)

ВВЕДЕНИЕ

Микроконтроллер - это специальная микросхема, предназначенная для управления различными электронными устройствами. В корпусе микроконтроллера объеденены процессор, ОЗУ, ПЗУ и периферия. Микроконтроллеры выпускают десятки компаний, причем производятся 32-битные, 16, и даже 8-битные микроконтроллеры (как i8051 и аналоги) [1].

Центральным узлом микроконтроллера является процессор, предназначенный для выполнения машинных команд. Основными параметрами процессора является его разрядность и допустимая тактовая частота его работы.

Помимо процессора микроконтроллер, как правило, содержит внутри себя систему тактирования, систему памяти, включающую Flash память программ и оперативную память данных, и набор периферийных модулей, которые предназначены для получения информации и формирования сигналов управления. К периферийным модулям относятся, например, таймеры, аналого-цифровые и цифро-аналоговые преобразователи, контроллеры прямого доступа к памяти, компараторы, интерфейсы связи [2].

В качестве среды разработки используется среда IAR, т.к. у нее имеется практически полнофункциональная бесплатная версия. Кроме того, среда разработки IAR может использоваться с микроконтроллерами различных производителей и различных семейств. То есть, ознакомившись с работой в этой среде, можно легко переходить от одного типа микроконтроллеров к другому [2].

В результате выполнения курсовой работы должно быть получена **метеостанция на базе датчика BME280 с передачей параметров по беспроводному интерфейсу**.

# АНАЛИЗ ТРЕБОВАНИЙ ТЕХНИЧЕСКОГО ЗАДАНИЯ

## Для разработки должна использоваться отладочная плата XNUCLEO-F411RE

Для разработки будет использоваться отладочная плата XNUCLEO-F411RE, производства Waveshare Electronics [3]. В основе платы – ARM Cortex-M4 микроконтроллер STM32F411RET6. Эта отладочная плата представляет собой гибкую платформу, позволяющую разработчикам реализовать собственные идеи и в кратчайшие сроки сделать прототип будущего изделия. Разъемы ST Morpho платы XNUCLEO-F411RE обеспечивают полный доступ к линиям портов ввода/вывода (I/O) и дальнейшее периферийное расширение. Изделие поставляется с отдельным модулем ST-Link/ V2.

Технические характеристики микроконтроллера STM32F411RET6:

* ядро: ARM 32-Бит Cortex-M4;
* рабочая частота: 100МГц;
* рабочее напряжение: 1.7…3.6В;
* память: 512кБ Flash, 128кБ SRAM;
* интерфейсы: 1 x SDIO, 1 x USB 2.0 FS, 5 x SPI or 5 x I2S, 3 x USART, 3 x I2C;
* АЦП/ЦАП: 1 x АЦП (12 Бит, 16 каналов).

Остальные технические характеристики:

* SPX3819M5: регулятор напряжения 3,3 В;
* AMS1117-5.0: регулятор напряжения 5,0 В;
* CP2102: преобразователь USB в UART;
* разъем Arduino: для подключения щитов Arduino;
* интерфейс ICSP: Arduino ICSP;
* USB TO UART: для отладки;
* разъем USB: интерфейс связи USB;
* интерфейс SWD: для программирования и отладки;
* заголовки ST Morpho: доступ к VCC, GND и всем входам / выходам, прост в расширении;
* 6-12 В постоянного тока;
* пользовательская кнопка;
* кнопка сброса;
* индикатор питания;
* пользовательский светодиод;
* 500 мА быстрый самовосстанавливающийся предохранитель;
* индикатор Rx / Tx последовательного порта;
* кристалл 8 МГц;
* кристалл 32,768 кГц.

Комплектация:

* 1 х Отладочная плата (XNUCLEO-F411RE);
* 1 х Программатор (ST-LINK/V2 (mini));
* 1 х Кабель (USB Type A Plug to Micro B Plug Cable);
* 1 х Кабель (USB Type A Plug to Receptacle Cable).

## Программное обеспечение измерение должно измерять давление, влажность и температуру и рассчитывать точку росы

Предусмотрим в архитектуре проекта класс Measurement, предоставляющий результат измерения некоторой величины, выраженный в требуемых единцах измерения, а также хранящий набор допустимые для данной величины единицы измерения. Логику пересчета результата измерения конкретной физической величины при выборе различных единиц измерения можно реализовывать в дочерних классах класса Measurement.

### Период измерения должен быть 100 ms.

Предусмотрим задачу ОСРВ MeasurementTask с периодом 100 ms для снятия показаний датчиков.

### Для измерения давления, влажности и температуры должен использоваться датчик BME280

Для измерения давления, влажности и температуры будет исользоваться датчик BME280 производства Bosch Sensortec [4]. Датчик предназначен для измерения атмосферного давления, температуры и влажности, что позволяет создать на его основе небольшую метеостанцию.

Характеристики используемого модуля с датчиком BME280:

* Интерфейс: SPI, I2C;
* Напряжение питания: 3.3В;
* Диапазон измерений давления: 300–1100 гПа;
* Диапазон измерений температуры: -40 – +85 °C;
* Диапазон измерений влажности: 0 – 100 %;
* Энергопотребление: в режиме измерений – 2.74 нА, в спящем режиме: - 0.1 нА;
* Точность измерений: давление – 0.01 гПа, температура – 0.01°C,   
  влажность – 3%.

Для взаимодействия с датчиком BME280 предусмотрим класс BME280Driver.

### Общение с датчиком должно осуществляться по интерфейсу SPI1

Для взаимодействия с датчиком BME280 будем использовать SPI1. Выберем следующие пины для подключения датчика:

* pin B3 – SCK;
* pin B4 – MISO;
* pin B5 – MOSI;
* pin B6 – CS.

Для взаимодействия с SPI предусмотрим два класса: аппаратно-зависимый класс Spi, методы которого оборачивают обращения к регистрам микроконтроллера и работу с прерываниями, и класс верхнего уровня SpiDriver, предоставляющий возможность передачи и чтения данных по интерфейсу Spi.

### Точка росы должна считаться на основе текущих показаний влажности и давления

В классе BME280Driver предусмотрим метод CalcDewPoint(), расчитывающую точку росы. Метод CalcDewPoint() будет вызываться из метода класса BME280Driver Measure после измерния текущих значений температуры, давления, влажности:

void Measure()

{

temperature = ReadTemperature();

pressure = ReadPressure();

humidity =ReadHumidity();

dewPoint =CalcDewPoint();

}

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

Для вывода значений давления, влажности, температуры будем использовать экран с жидкими чернилами 4.2inch e-Paper Module производства Waveshare Electronics [5]. Для взамиодействия с этим модулем предусмотрим класс EPDDisplayDriver. Логику формирования черно-белого изображения, выводимого на экран, выделим в класс ScreenDriver. Также предусмотрим класс прикладного уровня Display, предоставляющий методы для взаимодействия с экраном, оборачивающие обращения к объетам классов EPDDisplayDriver и ScreenDriver.

## Общение с экраном должно осуществляться через интерфейс SPI2

Для взаимодействия с экраном будем использовать SPI2. Выберем следующие пины для подключения модуля экрана:

* pin B1 – CS;
* pin B2 – DC;
* pin B13 –CLK;
* pin B15 –DIN;
* pin C2 – BUSY;
* pin C3 – RST;

Для осуществления передачи данных по интерфейсу SPI предусмотрим еще один класс верхнего уровня SpiDriverOnlyTransmit, предоставляющий возможность передачи данных по интерфейсу Spi.

### Период вывода информации на экран должен быть 3 секунды.

Предусмотрим задачу ОСРВ DisplayUpdateTask с периодом 3000 ms для вывода на экран новых измеренных значений.

### Формат вывода:

"Давление: " XXX.XX [Units]  
"Влажность: " XXX.XX [Units]  
"Температура: " XXX.XX [Units]  
"Точка росы": " XXX.XX [Units]

В задаче DisplayUpdateTask будет сформировано изображение, соответствующее указанному формату, и отправлено на дисплей для отображения.

## Передача значений по беспроводному интерфейсу должна осуществляться через модуль BlueTooth Bee HC-06

Для передачи значений по беспроводному интерфейсу будет использован модуль BlueTooth Bee HC-06 [6]. Связь с модулем осуществляется через интерфейс USART, поэтому предусмотрим в архитектуре проекта классы для работы с этим интерфейсом. Создадим аппаратно-зависимый класс Usart, методы которого оборачивают обращения к регистрам микроконтроллера и работу с прерываниями, и класс верхнего уровня UsartDriver, предоставляющий возможность передачи и чтения данных по интерфейсу Usart.

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

Для подключения модуля BlueTooth Bee HC-06 будет использована плата  Accessories Shield [7].

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

Предусмотрим задачу ОСРВ BluetoothUpdateTask с периодом 1000 ms для вывода информации в BlueTooth модуль.

### Общение с платой расширения должно осуществляться через USART2

Для взаимодействия с платой расширения будем использовать USART2. Настроим следующие пины под USART2:

* pin A3 – RX;
* pin A2 – TX.

### Формат вывода:

"Давление: " XXX.XX [Units]  
"Влажность: " XXX.XX [Units]  
"Температура: " XXX.XX [Units]  
"Точка росы": " XXX.XX [Units]

В задаче BluetoothTask при помощи функции sprntf будет сформированa строка с соответствующим заданию форматированием и передана в BlueTooth Module по USART.

## Архитектура должна быть представлена в виде UML диаграмм в пакете Star UML

Архитектура проекта будет представлена в виде UML-диаграмм в пакете StarUML. StarUML - это инструмент UML (универсальный язык моделирования) производства MKLab. StarUML поддерживает большинство типов диаграмм, указанных в UML 2.0 .

## Приложение должно быть написано на языке С++ с использование компилятора ARM 8.40.2

Приложение будет написано на языке С++ с использованием компилятора ARM 8.40.2.

## При разработке должна использоваться Операционная Система Реального Времени FreeRTOS и С++ обертка над ней

При разработке будет использоваться Операционная Система Реального Времени FreeRTOS и С++ обертка над ней [7].

## По нажатию кнопки USER на плате XNUCLEO-F411RE единцы измерения температуры должны изменяться в следующей циклической последовательности F→K→C.

Для работы с кнопкой предусмотрим класс Button. Для обработки нажатий кнопки предусмотрим задачу ОСРВ ButtonPressTask c малым периодом (100 ms). При нажатии кнопки задача будет генерировать событие, которое будет отслеживаться задачей MeasureTask, циклично меняющей единицы измерения температурных величин.

# Функциональная схема устройства

На рисунке 1 приведена функциональная схема устройства, составленная на основе требований технического задания.

Bluetooth модуль

SPI1

Плата

Микро-контроллер

Unit Select

Кнопка выбора единицы измерения температуры

BME280

SPI2

VDD

GND

Программатор

Дисплей

UART2

Рисунок 1 – Функциональная схема устройства

Функциональная схема устройства состоит из следующих блоков:

1. микроконтроллер STM32F411RE;
2. датчик BME280;
3. дисплей Waveshare 4.2 inch e-Paper Module;
4. плата расширения с подключенным к ней Bluetooth-модулем;
5. кнопка выбора единицы измерения температуры;
6. программатор.

# АРХИТЕКТУРА ПРОЕКТА

Программирование осуществлялось на языке С++ с использованием операционной системы реального времени FreeRTOS.

## Общая архитектура проекта

Общая архитектура проекта, выполненная в программе StarUML, представлена на рисунке 2.

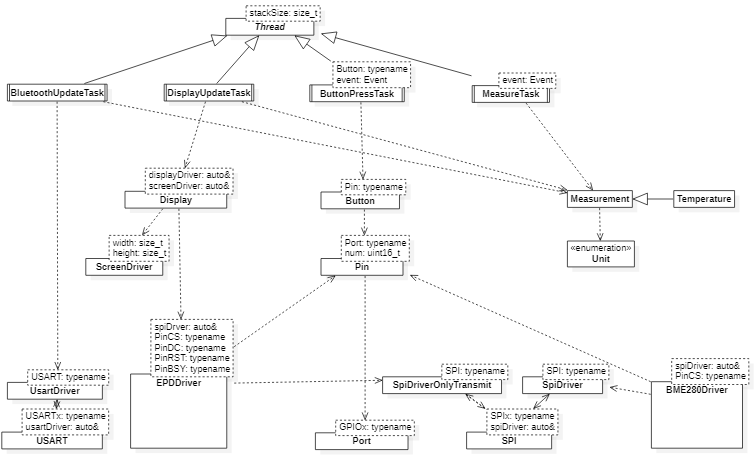


Рисунок 2 – Общая архитектура проекта

## Детальная архитектура

## Класс Measurement

Класс Measurement, описывающий значения измеряемых величин, имеет следующие компоненты:

* DataSourceFcn - указатель на функцию-поставщик данных;
* units - массив единиц измерения, используемых для выражения значения этой величины;
* countOfUnit - длина массива unit;
* defaultUnit - исходная единица измерения, в которой выражены значения, поставляемые источником данных DataSourceFcn;
* GetValue() - функция, возвращающая выраженное в defaultUnit значение измеряемой величины;
* GetValue(unit) - функция, возвращающая значение измеряемой величины, выраженное в выбранной единице измерения unit;
* quantityName - сторка наименования измеряемой величины;
* ShowUnitName(unit) - статическая функция, возвращающаяя строку обозначения единицы измерения unit.
* конструктор, создающий экземпляр класса на основе названия измеряемой величины, массива допустимых единиц измерений, длины этого массива, исходной величины измерения, указателя на функцию-поставщика данных;
* упрощенный конструктор для величин с одной допустимой единицей измерения, создающий экземпляр класса на основе названия измеряемой величины, единицы измерения, указателя на функцию-поставщика данных.

Дочерний класс Temperature, реализующий логику пересчета из одной температурной шкалы в другую, используется для описания величин, имеющих размерность температуры: temperature и dewPoint. Давление и влажность в рамках данной программы имеют по одной единице измерения, и ввиду отсутствия специальной логики для их описания достаточно базового класса.

На рисунке 3 приведены диаграммы классов Measurement и Temperature, а также экземпляры этих классов temperature, pressure, humidity, dewPoint, указатели на которые объединены в массив measurements.

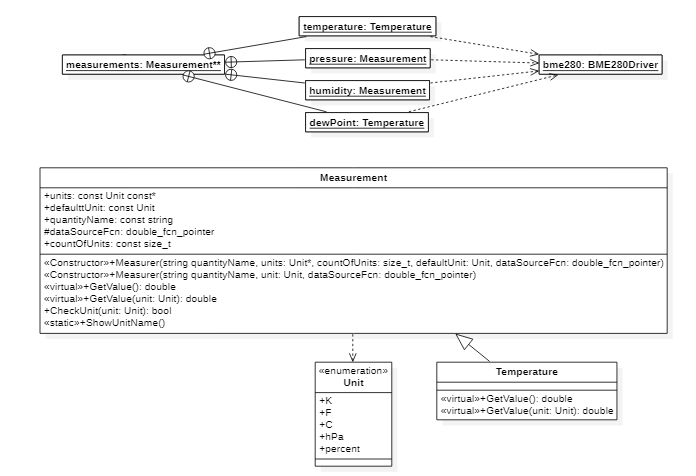


Рисунок 3 – Диаграммы классов Measurement и Temperature

## Класс BME280Driver

Класс bme280 используется для взаимодействия с датчиком BME280:

* метод StartMeasurements() запускает измерения датчика с   
  передискретизацией 1, устонавливает режим работы normal;
* метод InitializeCalibrateData() считывает калибровочные коэффициенты.
* метод Measure() обновляет поля temperature, pressure, humidity, dewPoint в соответствии с возвращаемыми значениями методов ReadTemperature(), ReadPressure(), ReadHumidity(), CalcDewPoint();
* Методы ReadTemperature(), ReadPressure(), ReadHumidity() отвечают за чтение «сырых» данных из регистров датчика и последующие введение поправок в соответствии с рекомендациями производителя датчика [3];
* Методы GetTemperature(), GetPressure(), GetHumidity(), GetDewPoint() предоставляют доступ на чтение полей temperature, pressure,humidity, dewPoint.

На рисунке 4 приведена диаграмма класса BME280Driver

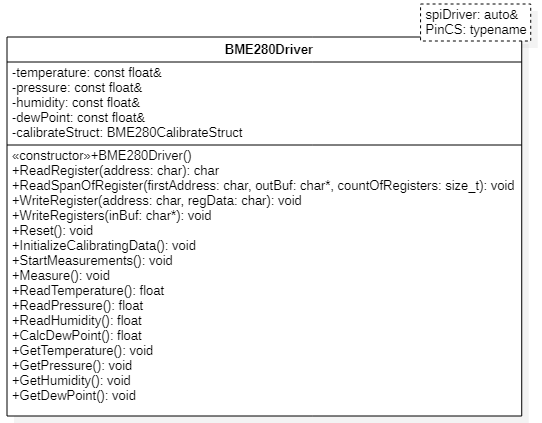


Рисунок 4 – Диаграмма класса BME280Driver

Наиболее существенной задачей, решенной при разработке BME280Driver, является задача чтения компенсационных параметров из регистров датчика и компенсация с их помощью систематических погршностей сырых данных датчика. На рисунке 5 приводятся регистры датчика, в которых записаны компенсационные параметры (Table 16) и регистры управления режимом измерений (Tables 19, 22) [4]:

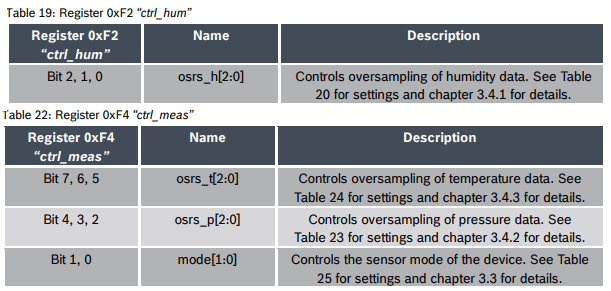
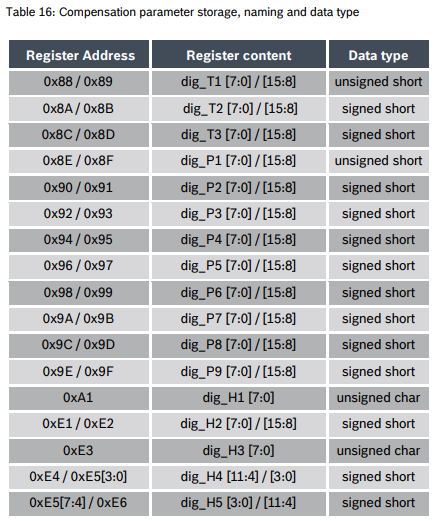
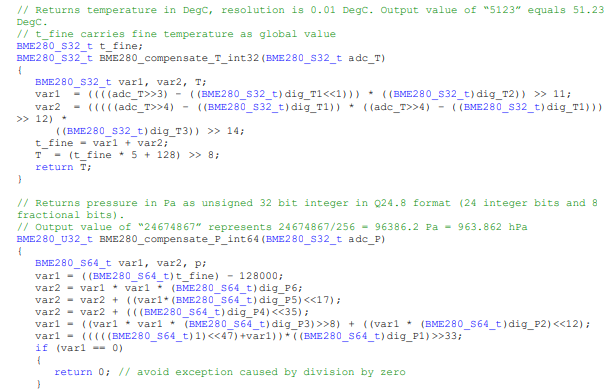
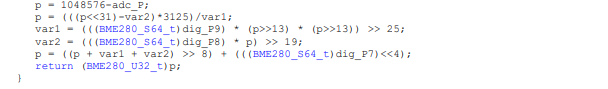
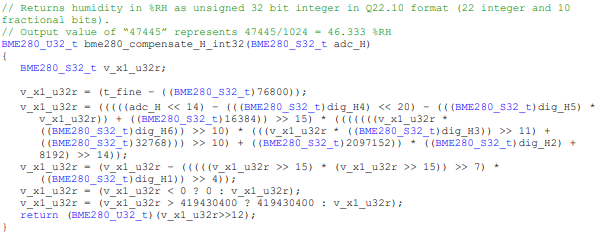


Рисунок 5 – Регистры с компенсационными параметрами

Ниже приведены компенсационные формулы, рекомендованные   
производителем [4]:







Соответствующий отрывок определения класса BME280Driver:

template<auto& spiDriver,typename CSPin>

class BME280Driver

{

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;

int8\_t dig\_H6;

} \_\_attribute\_\_((packed)) calibStruct;

double temperature;

double pressure;

double humidity;

double dewPoint;

uint32\_t tempInt=138347U;

uint32\_t pressInt;

public:

...

void InitializeCalibrateData()

{

ReadSpanOfRegisters(BME280Registers::Calibrate,reinterpret\_cast<char\*>(&calibStruct), 24);

calibStruct.dig\_H1=ReadRegister(BME280Registers::CalibrateDigH1);

ReadSpanOfRegisters(BME280Registers::CalibrateDigH2,reinterpret\_cast<char\*>(&calibStruct.dig\_H2), 3);

short byteDigH4 = ReadRegister(BME280Registers::CalibrateDigH4);

short byteDigH4H5 = ReadRegister(BME280Registers::CalibrateDigH4H5);

short byteDigH6 = ReadRegister(BME280Registers::CalibrateDigH5);

calibStruct.dig\_H4 = (byteDigH4<<4)|(byteDigH4H5&0x0F);

calibStruct.dig\_H5 = (byteDigH6<<4)|((byteDigH4H5&0xF0)>>4);

calibStruct.dig\_H6 = ReadRegister(BME280Registers::CalibrateDigH6);

}

void StartMeasurements()

{

WriteRegister(BME280Registers::CtrlHumidityMeasurement,BME280RegisterValues::SingleHumidityMeasurementMode);//humid

WriteRegister(BME280Registers::CtrlMeasurements,BME280RegisterValues::SingleMeasurementsStart);//temp and press

}

void Measure()

{

temperature = ReadTemperature();

pressure = ReadPressure();

humidity =ReadHumidity();

dewPoint =CalcDewPoint();

}

float ReadTemperature()

{

char pTemp[3] = {0};

ReadSpanOfRegisters(BME280Registers::TemperatureData,pTemp, 3);

uint32\_t tempRaw = (pTemp[0]<<16)+(pTemp[1]<<8)+pTemp[2];

tempRaw>>=4;

int32\_t val1 = ((((tempRaw>>3) - ((int32\_t) calibStruct.dig\_T1 <<1))) \*

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

int32\_t val2 = (((((tempRaw>>4) - ((int32\_t) calibStruct.dig\_T1)) \*

((tempRaw>>4) - ((int32\_t) calibStruct.dig\_T1))) >> 12) \*

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

uint32\_t tempInt = val1 + val2;

float tempFloat = ((tempInt \* 5 + 128) >> 8);

tempFloat /= 100.0f;

return tempFloat;

}

float ReadPressure()

{

char pPress[3] = {0};

ReadSpanOfRegisters(BME280Registers::PressureData,pPress, 3);

uint32\_t pressRaw = (pPress[0]<<16)+(pPress[1]<<8)+pPress[2];

pressRaw >>= 4;

int64\_t val1 = ((int64\_t) tempInt) - 128000;

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

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

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

val1 = ((val1 \* val1 \* (int64\_t)calibStruct.dig\_P3) >> 8) + ((val1 \* (int64\_t)calibStruct.dig\_P2) << 12);

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

if (val1 == 0) {

return 0;

}

int64\_t p = 1048576 - pressRaw;

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

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

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

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

pressInt = p;

float pressFloat = p;

pressFloat = pressFloat/ 25600;

return pressFloat;

}

float ReadHumidity()

{

char pHumid[2] = {0};

ReadSpanOfRegisters(BME280Registers::HumidityData,pHumid, 2);

uint16\_t humidRaw = (pHumid[0]<<8)+pHumid[1];

int32\_t humidRawSign = ((int32\_t)humidRaw)&0x0000FFFF;

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

v\_x1\_u32r = (((((humidRawSign << 14) - (((int32\_t)calibStruct.dig\_H4) << 20) -

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

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

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

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

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

((int32\_t)calibStruct.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;

float humidFloat = (v\_x1\_u32r>>12);

humidFloat /= 1024.0f;

return humidFloat;

}

...

};

## Классы USART, UsartDriver

В классе USART<USARTx, tranceiver> содержатся методы, оборачивающие работу с регистрами USARTx микроконтроллера, и обработчик прерываний USARTx, в котором вызываются методы драйвера tranceiver, содержащие конкретную логику действий, которые требуется совершить при прерываниях RXNE (буфер приема не пуст), TXE (буфер передачи пуст), IDLE (обнаружено отсутствие сигнала на входной линии).

template<typename USARTx, auto& tranceiver>

class USART

{

. . .

static void UsartInterruptHandler()

{

if(USARTx::CR1::RXNEIE::InterruptWhenRXNE::IsSet()&&USARTx::SR::RXNE::DataRecieved::IsSet())

{

tranceiver.OnNextByteReceive();

}

if(USARTx::CR1::TE::Enable::IsSet()&&USARTx::SR::TXE::DataRegisterEmpty::IsSet())

{

tranceiver.OnNextByteTransmite();

}

if(USARTx::CR1::IDLEIE::InterruptWhenIDLE::IsSet()&&USARTx::SR::IDLE::IdleLineDetected::IsSet())

{

tranceiver.OnIdleLineDetected();

}

}

};

Класс UsartDriver<USART> предоставляет возможность передачи и чтения данных по интерфейсу USART. Момент завершения приема входного сообщения фиксируется по наступлению прерывания IDLE.

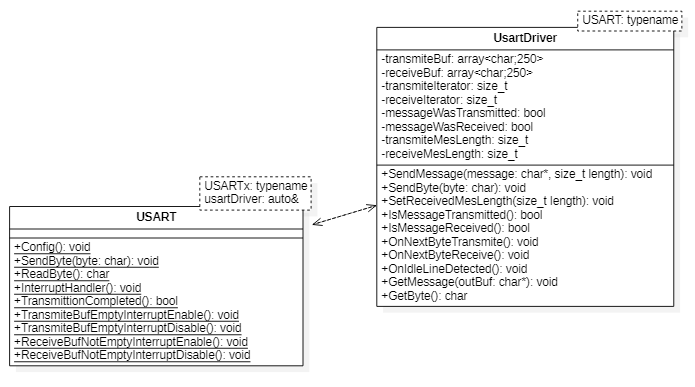


Рисунок 6 – Диаграммы классов USART, UsartDriver

## Классы SPI, SpiDriver, SpiDriverOnlyTransmit

Аналогично системе классов Usart-UsartDriver, для работы с протоколом SPI используются классы SPI, SpiDriver и SpiDriverOnlyTransmit. На рисунке 7 приведены диаграммы классов SPI, SpiDriver, SpiDriverOnlyTransmit.

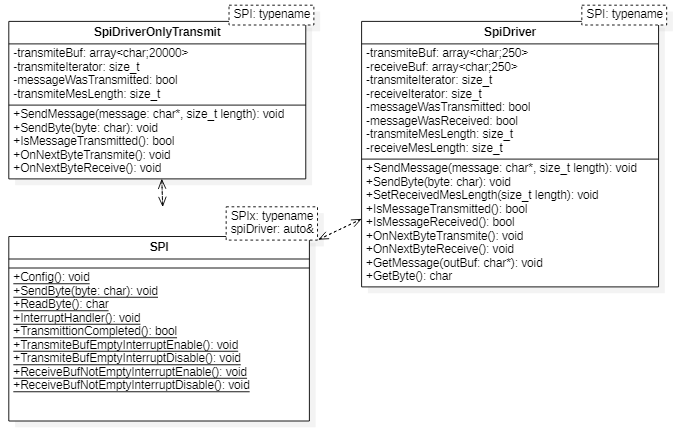
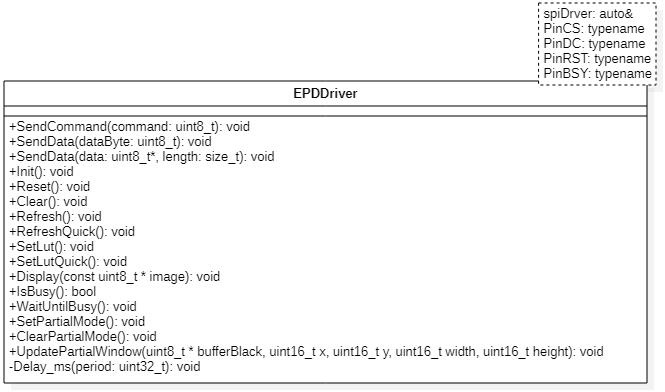


Рисунок 7 – Диаграммы классов SPI, SpiDriver,SpiDriverOnlyTransmit

## Класс EPDDriver

Класс предназначен для взамиодействия с 4.2inch e-Paper Module. Диаграмма класса приведена на рисунке 8.

 Рисунок 8 – Диаграмма класса EPDDriver

В шаблон класса включены следующие параметры:

* драйвер для работы с интерфейсом SPI;
* дескрипторы пинов, используемых для взаимодействия с модулем: СS, DC, RST, BSY.

Описание методов класса:

* SendСommand – используется для отправки комманд;
* SendData (2 перегрузки) – используется для отправки данных: либо по одному байту за сеанс связи, либо для передачи нескольких байтов за один сеанс связи;
* Init – обеспечивает запуск модуля, настройку частоты работы его процессора, настройку таблиц отображения, настройку таймингов модуля;
* Reset – обеспечивает запуск модуля;
* Clear – обеспечивает очистку дисплея модуля (заливка черным цветом);
* Refresh – устанавливает таблицы отображения, соответствующие нормальной длительности обновления изображения и запускает обновление изображения на дисплее модуля;
* RefreshQuick – устанавливает таблицы отображения, соответствующие сокращенной длительности обновления изображения и запускает обновление изображения на дисплее модуля;
* SetLut – устанавливает таблицы отображения, соответствующие нормальной длительности обновления изображения;
* SetLutQuick – устанавливает таблицы отображения, соответствующие сокращенной длительности обновления изображения;
* IsBusy – возвращает значение true, если модуль обрабатывает предыдущую команду (1 на ножке BSY);
* WhaitUntilBusy – ожидание готовности модуля к приему следующей команды;
* SetPartialMode – устанавливает режим частичного обновления дисплея модуля;
* ClearPartialMode – отключает режим частичного обновления дисплея модуля;
* UpdatePartialWindow – обеспечивает частичное обновление дисплея модуля в прямоугольнике, заданном x, y, width, height, в соответствии с передаваемым изображением bufferBlack;
* Delay\_ms – приватная функция, реализующая задержку требуемой длительности (в миллисекундах).

## Класс ScreenDriver

Класс ScreenDriver предоставляет инструменты для формирования черно-белого изображения, выводимого на дисплей. В шаблон класса передается длина и ширина изображения в пикселах. Диаграмма класса представлена на рисунке 9.

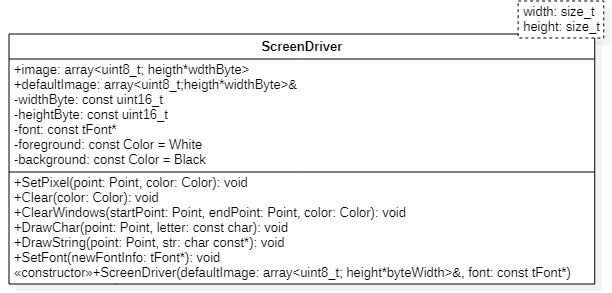


Рисунок 9 – Диаграмма класса ScreenDriver

Приватные параметры класса:

* widthByte – ширина двумерного массива байт, кодирующего изображение;
* heightByte – высота двумерного массива байт, кодирующего изображение;
* font – указатель на класс tFont, хранящий начертание букв шрифта;
* foreground ­– задает цвет переднего плана;
* background – задает цвет фона.

Публичные параметры класса:

* image – массив байт, кодирующий текущее изображение;
* defaultImage – массив байт, кодирующий заставку (фон);

Публичные методы класса:

* SetPixel – закрашивает прямоугольник 1×8 с указанными координатами заданным цветом;
* Clear – заливает изображение выбранным цветом;
* ClearWindows – заливает выбранным цветом заданный прямоугольник;
* ResetImage – сбрасывает изображение к стартовой заставке;
* SetFont – устанавливает шрифт (размер и начертание букв);
* DrawChar – добавляет символ в заданной точке изображения;
* DrawString – добавляет строку символов в заданной точке изображения.

## Класс Display

Класс прикладного уровня Display служит для упрощения взаимодействия с дисплеем (доступно лишь три метода). В шаблон класса передаются драйвер модуля-дисплея и драйвер для работы с изображениями, выводимыми на дисплей. Диаграмма класса представлена на рисунке 9.

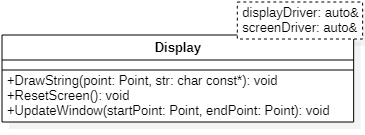


Рисунок 10 – Диаграмма класса Display

Методы класса Display:

* DrawString – добавляет строку символов в заданной точке изображения;
* ResetScreen – сбрасывает изображение к стартовой заставке;
* UpdateWindow – обновляет заданный прямоугольный участок дисплея всоответствии с сформированным изображением.

## Классы Port, Pin, Button

Диаграммы классов Port, Pin и Button представлены на рисунке 9

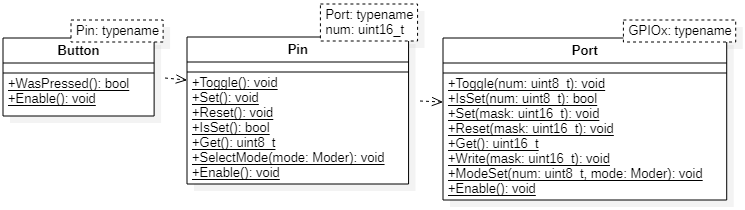


Рисунок 11 – Диаграммы классов Port, Pin и Button

## Классы BluetoothUpdateTask, DisplayUpdateTask, ButtonPressTask, MeasurementTask

Рассмотрим периоды и приоритеты запуска задач ОСРВ (диаграммы классов-задач представлены на рисунке 12).

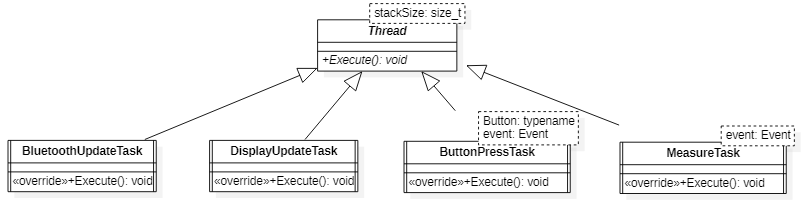


Рисунок 12 – Диаграммы классов BluetoothUpdateTask, DisplayUpdateTask, ButtonPressTask, MeasurementTask

* BluetoothUpdateTask: период – 1000ms, приоритет – lowest;
* DisplayUpdateTask: период – 3000ms, приоритет – lowest;
* ButtonPressTask: период – 100ms, приоритет – highest;
* MeasurementTask: период – 100ms, приоритет – priorityMax.

# ДЕМОНСТРАЦИЯ РАБОТЫ ПРОЕКТА

## Демонстрация работы дисплея на электронных чернилах

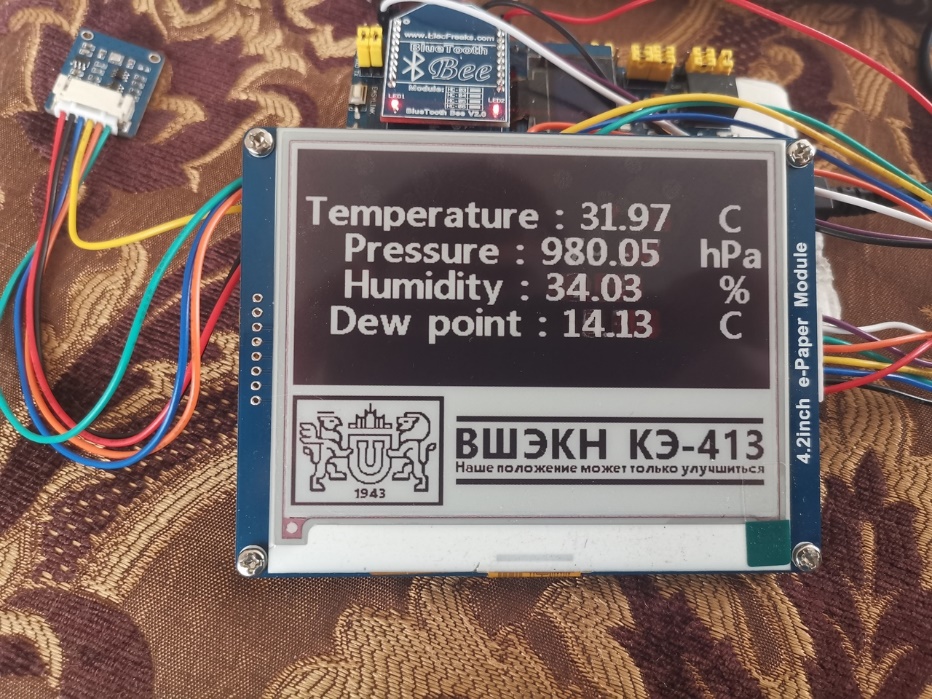


Рисунок 13 – Вывод результатов измерений на дисплей

## Демонстрация работы Bluetooth-модуля

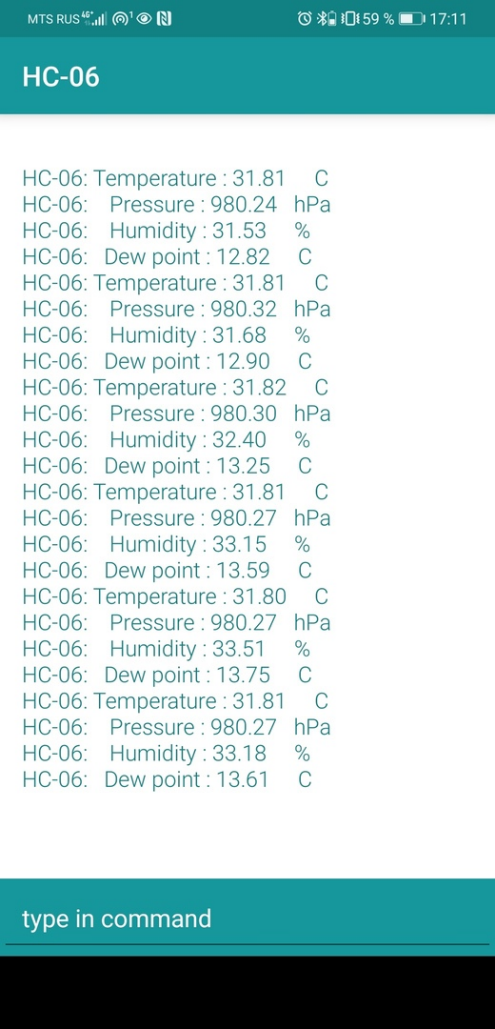


Рисунок 14 – Вывод результатов измерений в Bluetooth-терминал

## Демонстрация работы кнопки выбора единиц измерения температурных величин

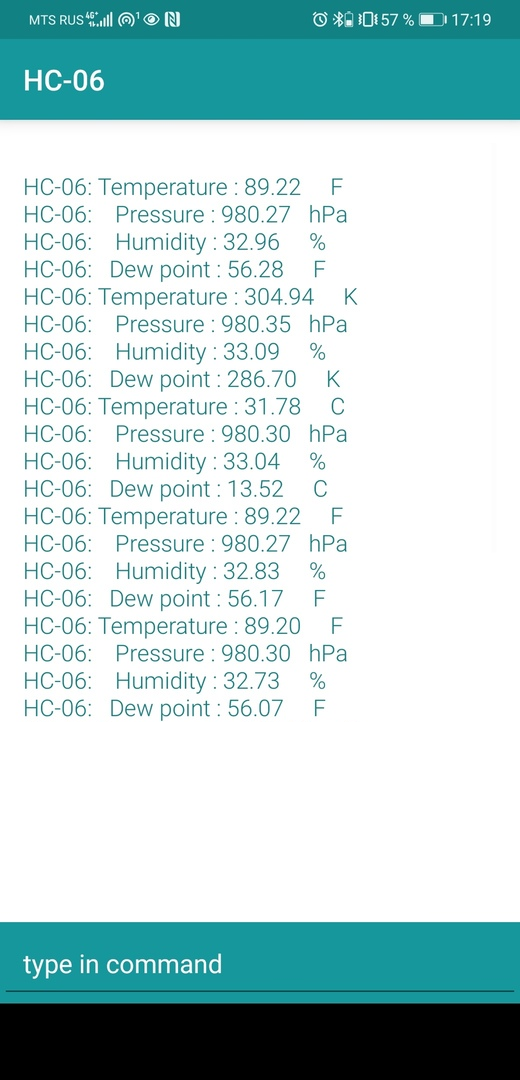


Рисунок 15 – Смена единиц измерения температурных величин

ЗАКЛЮЧЕНИЕ

В ходе работы над курсовым проектом было разработано программное обеспечение для метеостанции на базе датчика BME280. Вывод результатов измерения производится на смартфон по протоколу Bluetooth, а также на дисплей с электронными чернилами.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Назначение микроконтроллеров. – http://electrik.info/main/automation/549-chto-takoe-mikrokontrollery-naznachenie-ustroystvo-princip-raboty-soft.html.
2. Введение в микроконтроллеры STM32. – https://prog-cpp.ru/stm32-introduce.
3. XNUCLEO-F411RE – https://www.waveshare.com/wiki/XNUCLEO-F411RE.
4. BME280 Datasheet https://www.bosch-sensortec. com/media/boschsensortec/ downloads/datasheets/bst-bme280-ds002.pdf.
5. 4.2inch e-Paper Module – https://www.waveshare.com/wiki/4.2inch\_e-Paper\_Module.
6. BlueTooth Bee – https://www.waveshare.com/wiki/Dual-mode\_Bluetooth.
7. FreeRTOS и С++ обертка над ней – https://habr.com/ru/post/420467.