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 курсовой работе
по дисциплине «Программное обеспечение измерительных процессов»

Тема: Разработка устройства активной системы охлаждения
ЮУрГУ – 120301.2021.087.ПЗ КР

Нормоконтролер:
доцент
_____/ С.В. Колодий
«____» _____2021 г.

Руководитель:
доцент
_____ / С.В. Колодий
«____» _____ 2021 г.

Авторы работы:
студенты группы КЭ-413
_____ / Д.А. Майер
_____ / А.А. Сергушкина
«_____» _____ 2021 г.

Работа защищена с оценкой
_____
«_____» _____ 2021 г.

Челябинск 2021

АННОТАЦИЯ

Майер Д.А., Сергушкина А.А. Разработка устройства активной системы охлаждения Челябинск: ЮУрГУ, КЭ-413, 24 с., 14 илл.

Заданием данной работы является разработка устройства активной системы охлаждения на базе микроконтроллера STM32. 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 температуру и рассчитывать скорость вращения.
3. Вывод значений температуры должен производиться через UART на компьютер.
4. Разработка детальной архитектуры в виде диаграмм в пакете Star UML.
5. Код на языке C++ с использование компилятора ARM 8.40.2.

Разработка программного обеспечения проводилась в двух программных пакетах:

- StarUML – для разработки архитектуры;
- IAR Embedded Workbench 8.40.2 – для разработки кода на языке C++.

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

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Программное обеспечение – это совокупность программ, выполняемых на определенном компьютере.

Система охлаждения – это совокупность устройств, состоящая из кулеров, обеспечивающих принудительный отвод теплоты от нагревающихся деталей устройства.

Система охлаждения служит для отвода тепла от наиболее нагретых деталей какого-либо устройства, поддерживая в системе оптимальную температуру.

Объектно-ориентированное программирование (**ООП**) позволяет разложить проблему на составные части, каждая из которых становится самостоятельным объектом. Каждый из объектов содержит свой собственный код и данные, которые относятся к этому объекту.

Любая программа, написанная посредством **ООП**, отражает в своих данных состояние физических предметов либо абстрактных понятий – объектов программирования, для работы, с которыми она предназначена.

Все данные об объекте программирования и его связях с другими объектами можно объединить в одну структурированную переменную. В первом приближении ее можно назвать объектом.

С объектом связывается набор действий, иначе называемых методами. С точки зрения языка программирования набор действий или методов – это функции, получающие в качестве обязательного параметра указатель на объект и выполняющие определенные действия с данными объекта программирования. Технология **ООП** запрещает работать с объектом иначе, чем через методы, таким образом, внутренняя структура объекта скрыта от внешнего пользователя.

Описание множества однотипных объектов называется классом.

Объект – это структурированная переменная, содержащая всю информацию о некотором физическом предмете или реализуемом в программе понятии.

Класс – это описание множества объектов программирования (объектов) и выполняемых над ними действий.

Класс можно сравнить с чертежом, согласно которому создаются объекты. Обычно классы разрабатывают таким образом, чтобы их объекты соответствовали объектам предметной области решаемой задачи.

Целью данного курсового проекта является разработка устройства активной системы охлаждения, которая будет измерять температуру и на основе этих показателей рассчитывать значение скорости вращения и передавать их на ПК.

Задачи курсового проекта:

1) Проанализировать технические требования к разработке.

2) Создать общую архитектуру программного обеспечения системы охлаждения.

3) Создать детальную архитектуру программного обеспечения.

4) Реализовать утвержденную архитектуру в виде кода C++.

5) Провести испытание.

1. АНАЛИЗ ТРЕБОВАНИЙ

1.1 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.

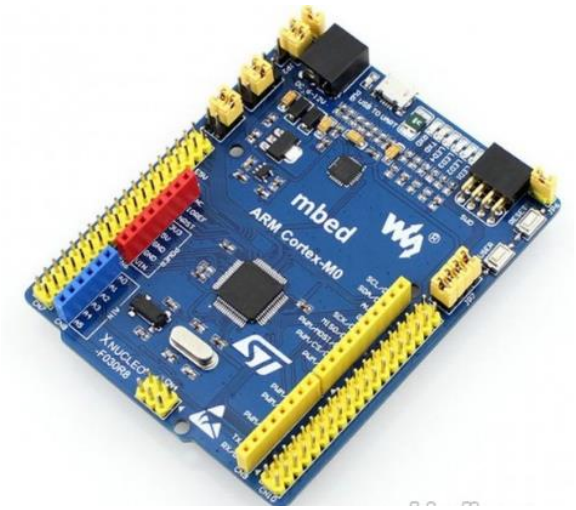

Рисунок 1 – Отладочная плата XNUCLEO–F411RE

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.

1.2 Устройство должно управлять скоростью вращения вентилятора в зависимости от температуры окружающей среды

1)Период измерения температуры и управления вентилятором должен быть 200 ms.

Измерения, которые производимые датчиком, должны быть периодическими, с частотой раз в 200 миллисекунд. Для реализации данного требования нам потребуется Операционная Система Реального Времени (RTOS), суть которой будет описана далее.

2) В качестве вентилятора выбран EVERCOOL EC3010H12B

С напряжением питания 12 вольт и рабочим током 0.11 ампер.

Рисунок 2 – Ветилятор EVERCOOL EC3010H12B

3) Для управления вентилятором выбран модуль Motor Control Shield

Данный модуль поддерживает управление 4 DC моторами, или же 2 шаговыми двигателями.

Рисунок 3 – Модуль Motor Control Shield

Для управления ШИМ сигналом и его выхода на модуль расширения необходимо настроить порты.

```
GPIOA::MODER::MODER10::Output::Set();     //подаем единицу на порт PA10
GPIOA::BSRR::BS10::High::Write();
GPIOB::MODER::MODER3::Output::Set();
GPIOB::BSRR::BR3::Low::Write();          //подаем 0 на порт PB3
GPIOB::MODER::MODER10::Output::Set();
GPIOB::MODER::MODER10::Alternate::Set();
GPIOB::AFRH::AFRH10::Af1::Set();         //Устанавливаем порт PB10 На альтренативную функцию
```

Рисунок 4 – Настройка портов на для ШИМ

Для подачи самого сигнала необходимо в регистр таймера записать значение скорости, которое зависин от температуры. Так как регистр принимает только целые значения, производится преобразования из float в uint32_t.

```
class Timer
{
  public:
    float SetPWM(float Speed)
    {
      uint32_t PWM = static_cast<uint32_t>(150.0f*Speed+850.0f);
      TIM2::CCR3::Write(PWM);
      return PWM;
    }
  private:
    uint32_t PWM=0;
};
```

Рисунок 5 – Настройка уровня ШИМ

1.3 Для управления скоростью вращения использовать следующую формулу

$Speed = P + I + D$,где
P - пропорциональная составляющая регулятора
I - интегральная составляющая регулятора
D - дифференциальная составляющая регулятора

$P(t) = K_p + e$,где
P - пропорциональная составляющая регулятора
e - ошибка между 23С и текущей измеренной температурой
Kp - пропорциональный коэффициент

$I = I_{i-1} + K_i \cdot e$,где
I - интегральная составляющая регулятора
e - ошибка между 23С и текущей измеренной температурой
Ki - интегральный коэффициент
I[i-1] = предыдущее значение интегральной составляющая регулятора

$D = K_d \cdot (e - e_{i-1})$,где
D - дифференциальная составляющая регулятора
e - ошибка между 23С и текущей измеренной температурой
e[i-1] - предыдущее значение ошибки между 23С и текущей измеренной температурой
Kd - дифференциальный коэффициент

Рисунок 6 – Формула ПИД регулятора

По определению, пропорционально-интегрально-дифференциальный регулятор – устройство в цепи обратной связи, используемое в системах автоматического управления для поддержания заданного значения измеряемого параметра, например, температуры воздуха.

Устройство подает управляющий или выходной сигнал на устройство регулирования, на основании полученных данных от датчиков или сенсоров. Контроллеры обладают высокими показателями точности переходных процессов и качеством выполнения поставленной задачи.

ПИД регулятор предназначен для поддержания на требуемом уровне некой величины — температуры, давления, уровня в резервуаре, расхода в трубопроводе, концентрации чего-либо и т.д., изменением управляющего воздействия на исполнительные механизмы, такие как автоматические регулирующие клапана, используя для этого пропорциональную, интегрирующую, дифференцирующую величины для своей настройки.

Целью использования является получение точного управляющего сигнала, который способен контролировать большие производства и даже реакторы электростанций.

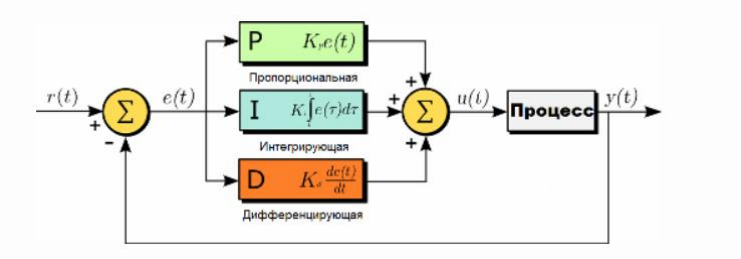

Рисунок 7 – Схема ПИД регулятора

Пропорциональный выходной сигнал дает пропорциональная составляющая. Сигнал этот приводит к противодействию текущему отклонению входной величины, подлежащей регулированию, от установленного значения. Чем больше отклонение — тем больше и сигнал. Когда на входе значение регулируемой величины равно заданному, то выходной сигнал становится равным нулю.

Интегрирующий. Интеграл по времени от величины рассогласования — есть основная часть интегрирующей составляющей. Она пропорциональна этому интегралу. Интегрирующий компонент используется как раз для исключения статической ошибки, поскольку регулятор со временем учитывает статическую погрешность.

Дифференцирующий. Темпу изменения отклонения величины, подлежащей регулированию, пропорциональна третья — дифференцирующая составляющая. Она необходима для того, чтобы противодействовать отклонениям (вызванным внешними воздействиями или задержками) от правильного положения, прогнозируемого в будущем.

ПИД-регуляторы применяют для поддержания заданного значения x0 некоторой одной величины, благодаря изменению значения u другой величины. Есть уставка или заданное значение x0, и есть разность или невязка (рассогласование) e = x0-x. Если система линейна и стационарна (практически это вряд ли возможно), то для задания u справедливы нижеследующие формулы:

$$u(t) = P + I + D = K_p\, e(t) + K_i \int_0^t e(\tau)\, d\tau + K_d \frac{de}{dt}$$

В этой формуле вы видите коэффициенты пропорциональности для каждого из трех слагаемых.

Практически в ПИД-регуляторах используют для настройки другую формулу, где коэффициент усиления применен сразу ко всем компонентам:

$$u(t) = K_p \left(e(t) + K_{ip} \int_0^t e(\tau)\, d\tau + K_{dp} \frac{de}{dt} \right)$$

Выбор значений ПИД регулятора

Первый – синтез регулятора, то есть вычисление параметров регулятора на основании модели системы. Данный метод позволяет очень точно рассчитать параметры регулятора, но он требует основательного погружения в ТАУ.

Второй метод – ручной подбор параметров (коэффициентов). Это метод эмперический, то есть проб и ошибок. Берем готовую систему, меняем один (или сразу несколько коэффициентов) регулятора, включаем регулятор и смотрим за работой системы. В зависимости от того, как ведет себя система с выбранными коэффициентами (недо/пере регулирование) опять меняем коэффициенты и повторяем эксперимент. И т. д. Ну, такой метод имеет право на жизнь, главное представлять, как изменение того или иного коэффициента повлияет на систему.

На рисунке 6 представлена реализация регулятора в коде.

```
if(temperature>23.0f)
{
    e = temperature - 23.0f;//разница между 23 градусами и измеренной температурой
    P = Kp * e;      //пропорциональный коэф
    {
      I = I + Ki * e;  // интегральный коэф
      D = Kd * (e - e0); //дифференциальный коэф

      e0 = e; //прошлое значение ошибки
      Speed = P + I + D; //формула скорости
      if (Speed >= 0.99f)
      {
        I = I - Ki * e;
        Speed = 1.0f;
      }
    }
  timer.SetPWM(Speed);
}
return Speed;
```

Рисунок 8 – Реализация ПИД регулятора в коде

В метод UpdateSpeed() приходит значение температуры, если оно больше 23 скорость будет считаться. *e* накапливает ошибку, рассчитываются параметры *P, I, D.* Если скорость доходит до 100%, для не допуска выхода скорости за 100% есть условие, при котором устанавливаем I прежнее значение. Это сделано для того чтобы, если скорости вращения недостаточно для охлажения, она не уходила в бесконечную величину. В случае, когда скорость меньше 100%, то происходит обычный расчет.

К измеренной температуре должен применяться фильтр вида:

$$\tau = \int \begin{pmatrix} 1 - e^{-\frac{dt}{RC}} & RC > 0\,\text{sec} \\ 1 & RC \le 0\,\text{sec} \end{pmatrix}$$

FilteredValue = OldFiltered + (Value – OldValue) · τ, где
dt - 200 мс;
Value – текущее нефильтрованное измеренное значение температуры;
oldValue - предыдущее фильтрованное значение.

Рисунок 9 – Фильтр

Цифровой фильтр – в электронике любой фильтр, обрабатывающий цифровой сигнал с целью выделения и/или подавления определённых частот этого сигнала.

Ниже представлена реализация фильтра в программе.

```
Filter(float RC, float dt)
{
   if (RC > 0)
   {
     tau = 1 - exp(-dt/RC);
   }
   else
   {
     tau=1;
   }
}
float FiltredValue(float tempval)
{
  FilteredValue = oldFilterValue + (tempval - oldFilterValue) * tau;
  oldFilterValue = FilteredValue;
  return FilteredValue;
}
```

Рисунок 10 – Реализация фильтра в коде

По формуле происходит сравнение если RC>0, tau рассчитывается по формуле или tau присваивается 1. Далее происходит фильтрация: фильтрованное значение получают из суммы прошлого отфильтрованного значения и разности нового нефильтрованного значения и прошлого фильтрованного умноженное на tau.

Далее фильтрованное значение становится прошлым и так по кругу. Само же фильтрованное значение передается в задачу.

1.4 Вывод значений температуры и скорости вентилятора должен производиться на ПК через интерфейс USART2

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. Период вывода информации на ПК должен быть 500ms.

2) Формат вывода:

"Температура: XXX.XX C"

"Скорость вращения: XXX.X %"

Данные с UART будут выводиться на ПК в программу Terminal.

1.5 Для индикации скорости вращения вентилятора необходимо использовать встроенные на плате светодиоды

Индикация должна быть осуществляться по следующему алгоритму:

- Все светодиоды потушены: скорость меньше 20% от максимальной
- Горит 1 светодиод: скорость от 20% до 40 %
- Горит 2 светодиод: скорость от 40% до 60 %
- Горит 3 светодиод: скорость от 60% до 80 %
- Горит 4 светодиод: скорость от 80% до 100 %

1.6 Разработка архитектуры в виде UML диаграмм в пакете StarUML.

UML можно условно называть графическим языком, предназначенным для моделирования компьютерных программ. Под моделированием понимается создание наглядной визуальной интерпретации чего-либо. UML позволяет создавать подобную интерпретацию программ высокоуровневой организации.

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.

1.7 Код на языке C++ 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.

1.8 При разработке должна использоваться Операционная Система Реального Времени FreeRTOS и C++ 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.

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.

2. РАЗРАБОТКА ОБЩЕЙ АРХИТЕКТУРЫ ПРОЕКТА

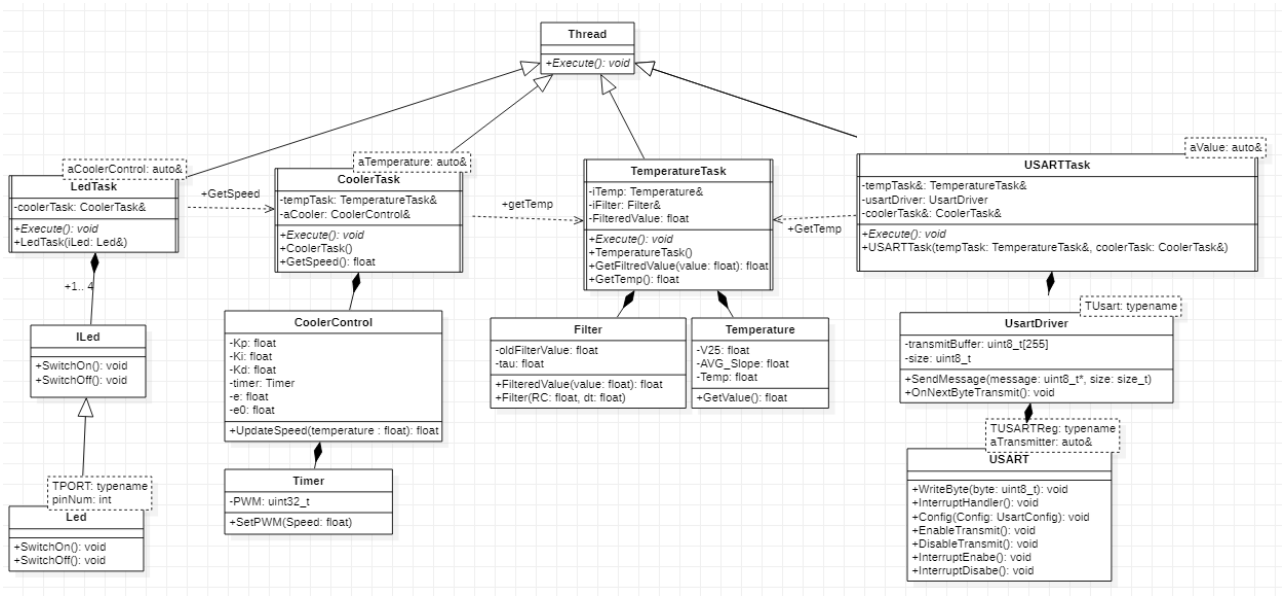

Рисунок 11 – Общая архитектура программы

Описание классов:

Проект включает в себя четыре активные задачи: **LedTask, CoolerTask, TemperatureTask, USARTTask**. Они наследуют класс **Thread**, позволяющий выполнять их на операционной системе реального времени (**RTOS**).

Класс **LedTask** отвечает за зажигание светодиодов в зависимости от уровня ШИМ;

Класс **ILed** виртуальный класс переключения светодиодов.

Класс **Led** зажигает светодиоды**.**

Для работы светодиодов необходима настройка их портов. Она приведена на рисунке

```
GPIOA::MODER::MODER5::Output::Set();
GPIOC::MODERPack<
    GPIOC::MODER::MODER5::Output,
    GPIOC::MODER::MODER8::Output,
    GPIOC::MODER::MODER9::Output
>::Set();
```

Рисунок 12 – Настройка портов светодиодов

Класс **CoolerTask** отвечает за уровень скорости вентилятора в зависимости от уровня температуры;

Класс **CoolerControl** по формуле с помощью ПИД регулятора преобразует значение температуры в уровень скорости и передает его в **Timer**.

Класс **Timer** отвечает за установку скорости вентилятора через ШИМ.

Класс **TemperatureTask** отвечает за снятие температуры с датчика температуры и его фильтрацию;

Класс **Filter** фильтрует принятые значения температуры;

Класс **Temperature** отвечает за получение значений температуры с АЦП;

Для получения данным температуры необходима предварительная настройка АЦП:

```
RCC::APB2ENR::ADC1EN::Enable::Set(); //включаем АЦП
ADC1::SQR3::SQ1::Channel18::Set();// настраиваем порт датчика т-ры
ADC_Common::CCR::TSVREFE::Enable::Set();// включаем порт датчика температуры
ADC1::CR1::RES::Bits12::Set();//разрядность 12бит
ADC1::CR2::CONT::SingleConversion::Set();//установ одиночное преобразование
ADC1::CR2::EOCS::SingleConversion::Set();//установ одиночное преобразование
ADC1::SQR1::L::Conversions1::Set();//количество измерений 1
ADC1::SMPR1::SMP18::Cycles84::Set();// скорость дискретизации
ADC1::CR2::ADON::Enable::Set();//
```

Рисунок 13 – Настройка АЦП

Получение температуры происходит по специальной формуле:

```
float  data =  static_cast<float>(ADC1::DR::Get()); //Get data from ADC;
Temp = ((((data*3.3F)/4096.0F - V25)/AVG_Slope)+25.0F); //rachet temperturi
```

Рисунок 14 – Формула для получения температуры

Класс **USARTTask** отвечает за передачу значений температуры и скорости вращения по USART на компьютер. Оно показано ниже:

```
sprintf(str, " Temperature: %3.2f C\n Speed: %3.2f %\n", temperature, speed);
usartDriver.SendMessage(str, strlen(str));
```

Рисунок 15 – Отправка данных на ПК

Класс **UsartDriver** отвечает за передачу данных по **USART**.

Класс **USART** выполняет передачу данных по **USART.**

Для работы USART необходима предварительная настройка портов для передачи данных.

```
RCC::APB1ENRPack<
  RCC::APB1ENR::TIM2EN::Enable,
  RCC::APB1ENR::USART2EN::Enable
  >::Set() ;
GPIOA::MODERPack<
  GPIOA::MODER::MODER2::Alternate, // Uart2 TX
  GPIOA::MODER::MODER3::Alternate  // Uart2 RX
  >::Set() ;
GPIOA::AFRLPack <
  GPIOA::AFRL::AFRL2::Af7, // Uart2 TX
  GPIOA::AFRL::AFRL3::Af7  // Uart2 RX
  >::Set() ;
  USART2::BRR::Write(UartSpeed9600);
  USART2::CR1::UE::Enable::Set();
  NVIC::ISER1::Write(1<<6);
```

Рисунок 16 – Настройка портов для USART

3. РАЗРАБОТКА ДЕТАЛЬНОЙ АРХИТЕКТУРЫ ПРОЕКТА

3.1 LedTask

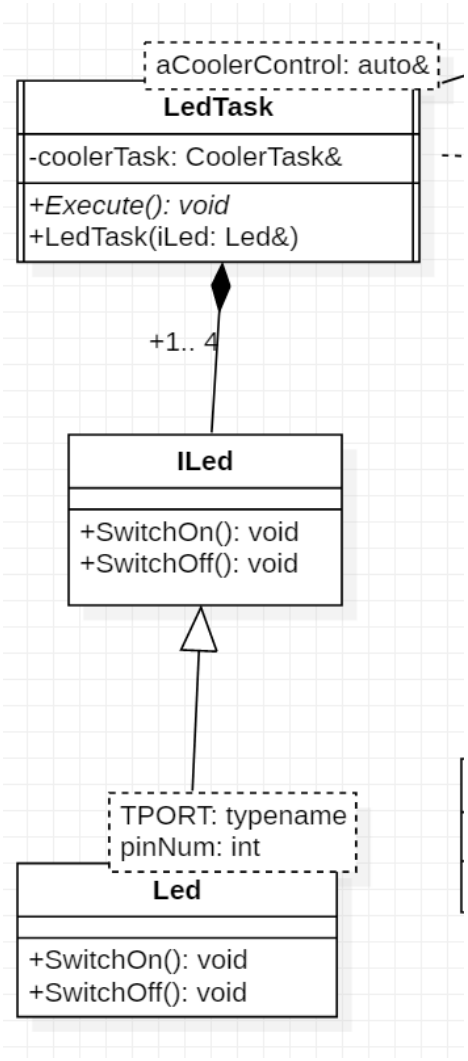

Рисунок 17 – Класс LedTask

Класс LedTask отвечает за зажигание светодиодов в зависимости от уровня скорости.

Приватные атрибуты:

coolerTask – ссылка на объект класса CoolerTask&.

Публичные методы:

Execute() – Бесконечный цикл;

LedTask() – задача.

Класс ILed отвечает за переключение светодиодов.

Публичные методы:

SwichOn () – зажигает светодиод;

SwichOff () – выключает светодиод.

Класс Led отвечает

Публичные методы:

SwichOn () – подает питание на определенный порт;

SwichOff () – подает 0 на определенный порт.

3.2 TemperatureTask

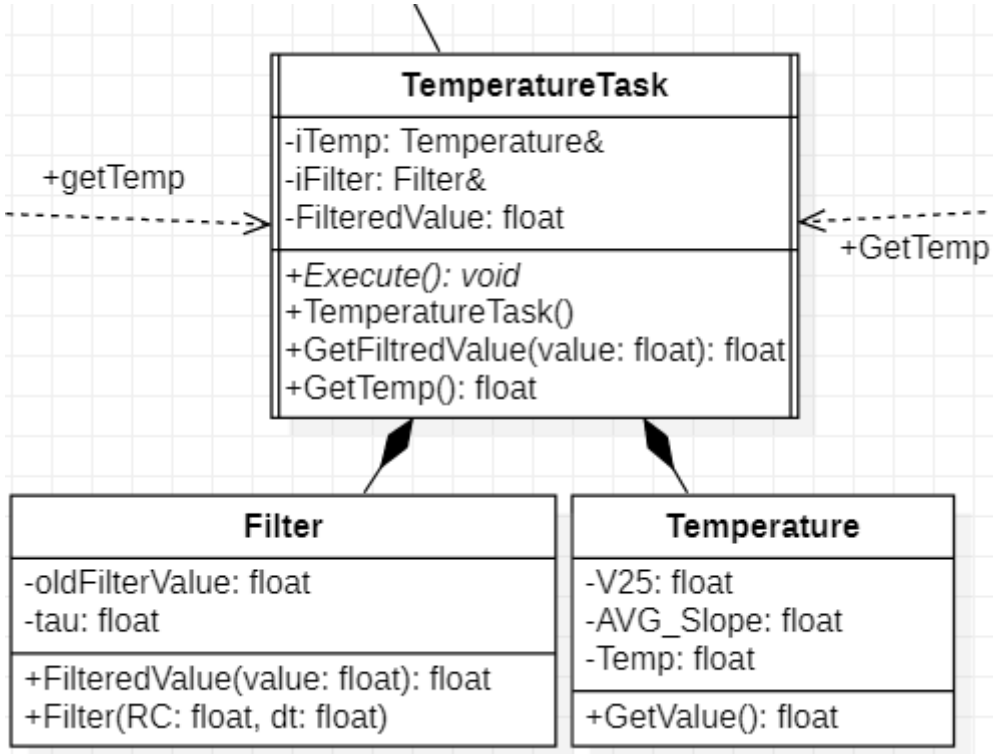

Рисунок 18 – Класс TemperatureTask

Класс TemperatureTask является активной задачей. Отвечает за получение температуры с датчика один раз в 200 миллисекунд.

Приватные атрибуты:

iTemp – ссылка на объект класса Temperature;

iFilter – ссылка на объект класса Filter;

FilteredValue: float – хранит фильтрованное значение температуры.

Публичные методы:

Execute() – включает в себя бесконечный цикл, в котором опрашивается сигнал.

TemperatureTask() – задача температуры.

GetFiltredValue(value: float) – возвращает отфильтрованное значение температуры.

GetTemp() – возвращает температуру.

Класс Filter:

Приватные атрибуты:

OldFilterValue – хранит прошлое отфильтрованное значение;

tau – хранит значение постоянной времени.

Публичные методы:

FilteredValue (value: float) – возвращает отфильтрованное значение температуры.

Filter (RC: float, dt: float) – хранит значения RC и dt.

Класс Temperature:

Приватные атрибуты:

AVG_slope – постоянная для расчета температуры.

V25 – постоянная для расчета температуры.

Temp – хранит значение температуры.

Публичный метод:

GetValue() – получает значение температуры микроконтроллера с помощью АЦП.

3.3 CoolerTask

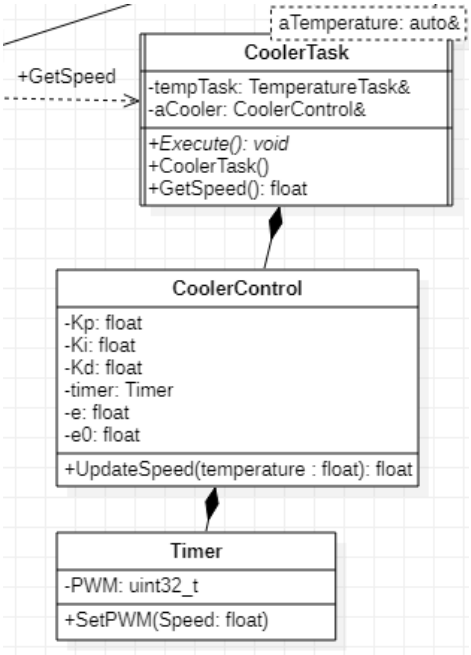

Рисунок 19 – Класс CoolerTask

Класс CoolerTask задача отвечает за уровень скорости вентилятора в зависимости от уровня температуры раз в 200 ms.

Класс CoolerTask:

Приватные атрибуты:

tempTask – ссылка на объект класса TemperatureTask

aCooler – ссылка на объект класса CoolerControl

Публичные методы:

Execute() – бесконечный цикл;

CoolerTask () – задача кулера.

GetSpeed() – Получает скорость вращения.

Класс CoolerControl:

Приватные атрибуты:

Kp – пропорциональный коэффициент;

Ki – интегральный коэффициент;

Kd – дифференцальный коэффициент.

timer – ссылка на класс Timer.

e – значение ошибки между 23 и измеренным значением температуры.

e0 – прошлое значение ошибки.

Публичные методы:

UpdateSpeed(temperature: float) – по формуле ПИД регулятора устанавливает скорость.

Класс Timer:

Приватные атрибуты:

PWM – хранит значение ШИМ.

Публичные методы:

SetPWM(Speed:float) – устанавливает в регистре сигнал ШИМ.

3.4 USARTTask

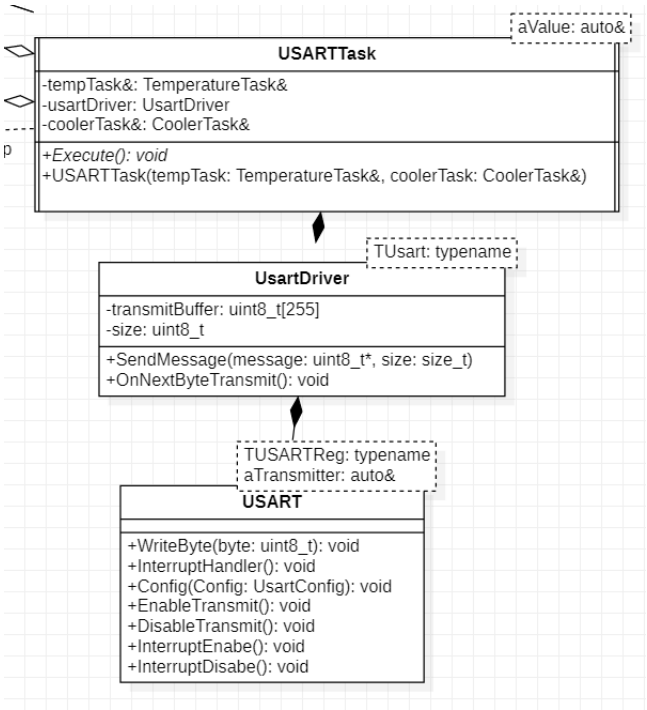

Рисунок 20 – класс USARTTask

Класс USARTTask задача отвечает за передачу значений температуры и скорости вращения по USART на компьютер раз в 500 ms.

Приватные атрибуты:

tempTask& – ссылка на объект класса TemperatureTask;

usartDriver – ссылка на объект класса UsartDrive;

coolerTask& – ссылка на объект класса CoolerTask.

Публичные методы:

Execute() – бесконечный цикл.

USARTTask(tempTask: TemperatureTask&, coolerTask: CoolerTask&) – активная задача.

Класс usartDriver:

Приватные атрибуты:

transmitBuffer – хранит значение, которое необходимо передать.

size – хранит размер строки, которую необходимо передать.

Публичные методы:

SendMessage – отвечает за отправку сообщения по **USART**.

OnNextByteTransmit () – отвечает за побайтовую передачу сообщения.

Класс USART:

Публичные методы:

WriteByte(byte: uint8_t) – записывает данные в регистр DR.

InterruptHandler() – проверяет флаги: Пуст ли регистр данных и разрешено ли прерывание по передаче.

Config (Config: UsartConfig) –

EnableTransmit() и DisableTransmit() – включают и выключают передачу данных.

InterruptEnable () и InterruptDisable () – разрешают и запрещают прерывания по передаче.

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работой было проанализировано техническое задание, создана общая и детальная архитектура, написан код на языке C++ и получено готовое рабочее устройство.