Министерство науки и высшего образования Российской Федерации

Томский государственный университет
систем управления и радиоэлектроники

Кафедра телекоммуникаций и основ радиотехники

К. В. Савенко
Е. В. Рогожников
Э. М. Дмитриев

ПРОГРАММИРОВАНИЕ МИКРОКОНТРОЛЛЕРОВ И МИКРОПРОЦЕССОРОВ ДЛЯ СИСТЕМ БЕСПРОВОДНОЙ СВЯЗИ

Методические указания для выполнения лабораторных работ
для студентов направления 11.03.02 «Инфокоммуникационные технологии и системы связи»

Томск
2020

УДК 621.37

ББК 32.884.1

С12

Савенко К.В.

С12 Программирование микроконтроллеров и микропроцессоров для систем беспроводной связи: методические указания для выполнения лабораторных работ для студентов направления 11.03.02 «Инфокоммуникационные технологии и системы связи» / К. В. Савенко, Е. В. Рогожников, Э. М. Дмитриев. – Томск: Томск. гос. ун-т систем упр. и радиоэлектроники, 2020. – 69 с.

Настоящее методическое пособие для студентов направления 11.03.02 «Инфокоммуникационные технологии и системы связи» посвящено изучению микроконтроллеров и микропроцессоров для систем беспроводной связи и интернета вещей в программной среде STM32CubeMX и uVision IDE. В нём представлены указания для выполнения лабораторных работ, позволяющих углубленно изучить принцип работы встраиваемых систем. В методическом пособии представлены как основы работы с микроконтроллером, так и работа с различными аналоговыми и цифровыми датчиками.

УДК 621.37

ББК 32.884.1

Оглавление

Введение

В современном мире, интернет вещей – это наиболее быстро растущее явление в сфере информационных технологий. Устройства интернета вещей все чаще применяются в нашей повседневной жизни. Это смартфоны, голосовые помощники, различные беспроводные датчики и другие интеллектуальные устройства. Основной массой передаваемых данных в интернете вещей является информация, которая предается посредством межмашинной связи, между различными устройствами через интернет, без участия человека. Множество устройств могут обмениваться необходимой информацией, при этом основные вычисления могут производиться на удаленном сервере. Все это позволяет использовать в качестве узлов интернета вещей недорогие и энергоэффективные устройства, выполняющие определенные функции и соединенные в единую сеть посредством сети интернет. Как правило, данные устройства состоят из датчиков, которые регистрируют события окружающей среды, интерфейса для передачи данной информации в сеть интернет и микроконтроллера для управления данными узлами и преобразования информации.

В семи лабораторных работах данного методического пособия, представлены эксперименты, позволяющие углубленно изучить принцип работы встраиваемых систем. В методическом пособии представлены как основы работы с микроконтроллером, так и работа с различными аналоговыми и цифровыми датчиками.

Сборник начинается с ознакомления с графическим интерфейсом программы STM32CubeMX, где производятся основные настройки микроконтроллера. Далее производится обучение работе в программе uVision IDE, в которой непосредственно производится программирование микроконтроллера и написание программы для встраиваемой системы. Далее рассматривается работа с аналоговыми и цифровыми датчиками встраиваемых систем.

1 Лабораторная работа №1
«Начало работы со встроенными системами. Программирование МК»

Цель работы: ознакомиться с созданием проекта в Keil µVision 5 и изучить его структуру, настроить периферию микроконтроллера в CubeMX.

Задачи лабораторной работы:

1) Изучить основные функции и блоки CubeMX.
2) Настроить периферию микроконтроллера в CubeMX.
3) Изучить основные функции и структуры Keil µVision 5.
4) Произвести генерацию проекта и загрузить прошивку в плату B-L475E-IOT01A [1].

Оборудование и программное обеспечение: плата B-L475E-IOT01A, кабель USB, среда разработки Keil µVision 5, CubeMX.

Теоретический материал

STM32CubeMX – это графический инструмент, который позволяет очень легко конфигурировать микроконтроллеры и микропроцессоры STM32, а также генерировать соответствующий код C для ядра Arm Cortex-M.

Первый шаг состоит в выборе микроконтроллера или микропроцессора STMicroelectronics STM32, который соответствует требуемому набору периферийных устройств.

Второй шаг позволяет конфигурировать GPIO и настройку тактирования для всей системы, а также интерактивно назначать периферийные устройства либо на Arm Cortex-M, Cortex - A. Специальные утилиты позволяют легко приступить к работе с микропроцессорами STM32.

Для микроконтроллеров и микропроцессора Arm Cortex-M второй шаг заключается в настройке программного обеспечения с помощью средства разрешения конфликтов распиновки, помощника настройки дерева, калькулятора энергопотребления и утилиты, которая настраивает периферийные устройства (такие как GPIO или USART) и стеки промежуточного программного обеспечения (например, USB или TCP / IP).

В конце пользователь запускает генерацию, которая соответствует выбранным вариантам конфигурации. Этот шаг предоставляет код инициализации C для Arm Cortex-M, готовый для использования в нескольких средах разработки. [2]

Среда IDE µVision объединяет управление проектом, среду выполнения, средства сборки, редактирование исходного кода и отладку программ в единой мощной среде. µVision прост в использовании и ускоряет разработку встроенного программного обеспечения. µVision поддерживает несколько экранов и позволяет создавать отдельные макеты окон в любом месте визуальной поверхности.

Отладчик включает в себя традиционные функции, такие как простые и сложные точки останова, окна наблюдения и контроль выполнения, и обеспечивает полную видимость периферии устройства. [3]

Ход работы

1) Запустите программу CubeMX. Для этого на панели задач откройте меню «Пуск», «Все программы», «STMicroelectronics», «STM32Cube», «STM32CubeMX», «STM32CubeMX.exe».

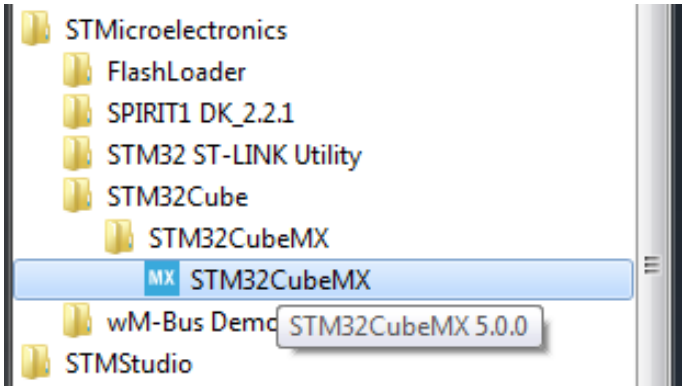

Рисунок 1.1 – CubeMX

STM32CubeMX представляет собой графический пользовательский интерфейс, который позволяет настраивать периферию микроконтроллера для вашего проекта в графическом режиме. На основе указанных вами данных STM32CubeMX сгенерирует настроенные библиотеки (HAL) для дальнейшей работы с проектом.

При первом запуске программы вы увидите окно, которое представлено ниже. В данном окне вы можете открыть уже созданный проект, создать новый проект для микроконтроллера или для платы разработчика, а также проверить обновления программы.

Нажмите на кнопку «ACCESS TO MCU SELECTOR» как показано на рисунке 1.2.

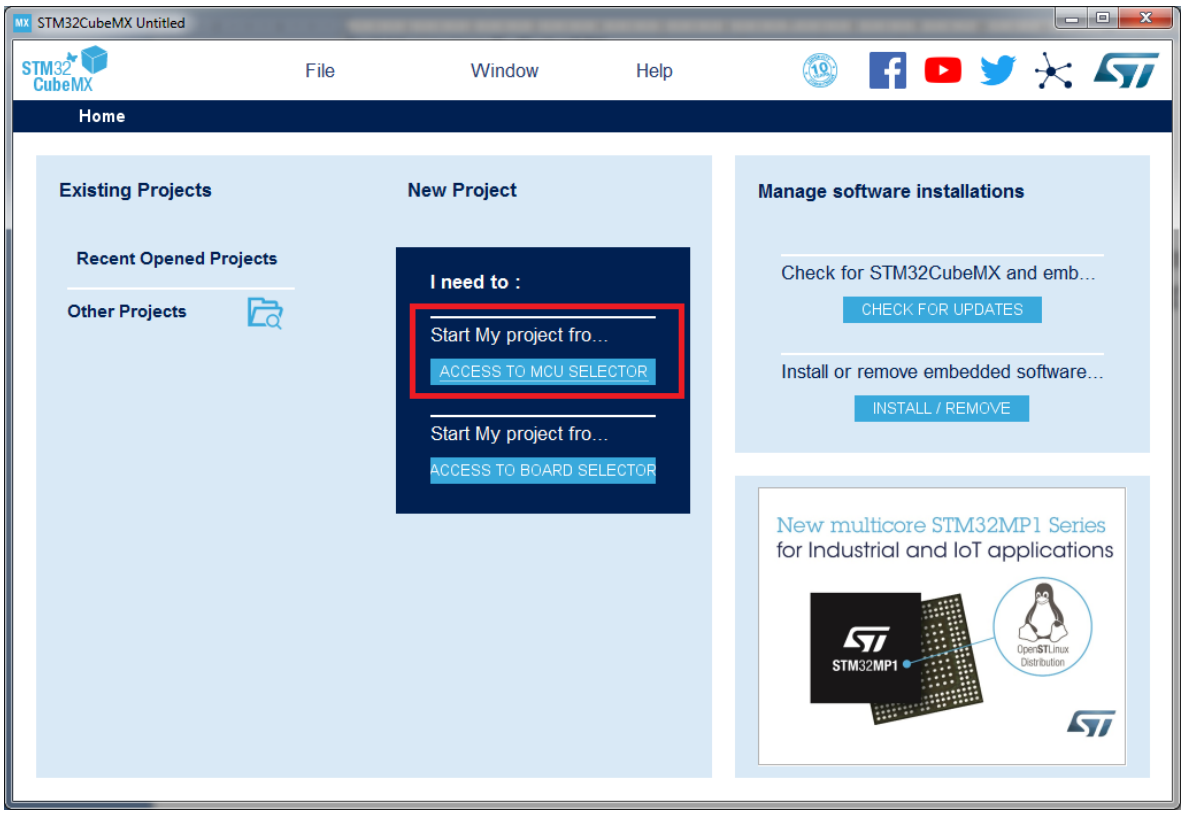

Рисунок 1.2 – Начальный экран программы

Далее откроется окно для выбора микроконтроллера, в списке «MCUs list» выберите микроконтроллер STM32L475VGTx и нажмите на кнопку «Start Project», как показано на рисунке 1.3.

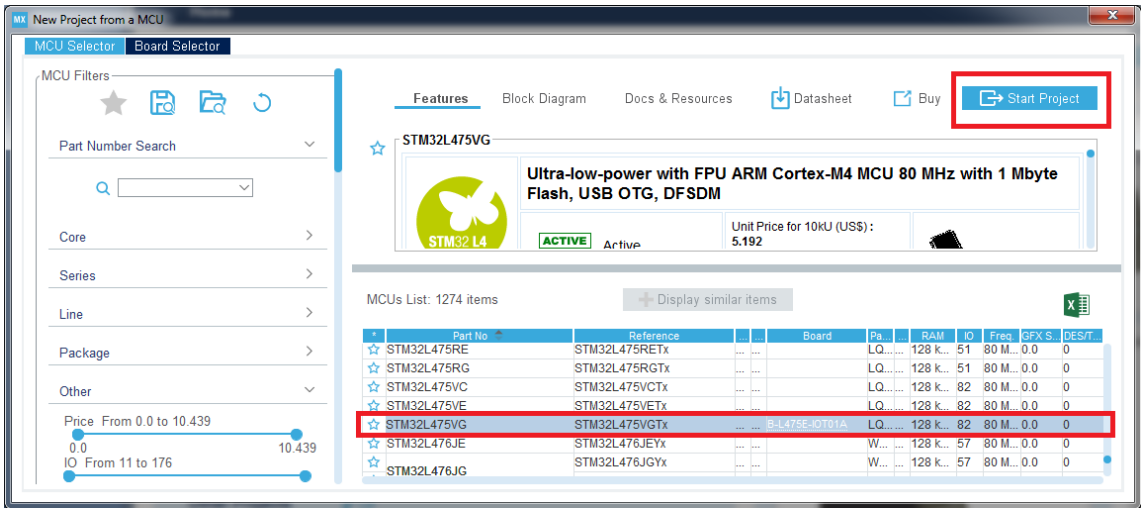

Рисунок 1.3 – Выбор микроконтроллера

Далее откроется рабочая область программы. Она изображена на рисунке 1.4. На данном рисунке выделены основные элементы программы:

1) Панель задач. На данной панели вы можете переключаться между Pinout & Configuration, Clock Configuration, Project Manager and Tools.

2) Окно периферии микроконтроллера. В данном окне по категориям разбиты все блоки микроконтроллера, для настройки каждой нужно дважды нажать левой кнопкой мыши по блоку.

3) В данном окне можно настраивать характеристики выбранного блока.

4) В данном окне показан Pinout микроконтроллера. В нем можно настроить функции каждого пина микроконтроллера отдельно.

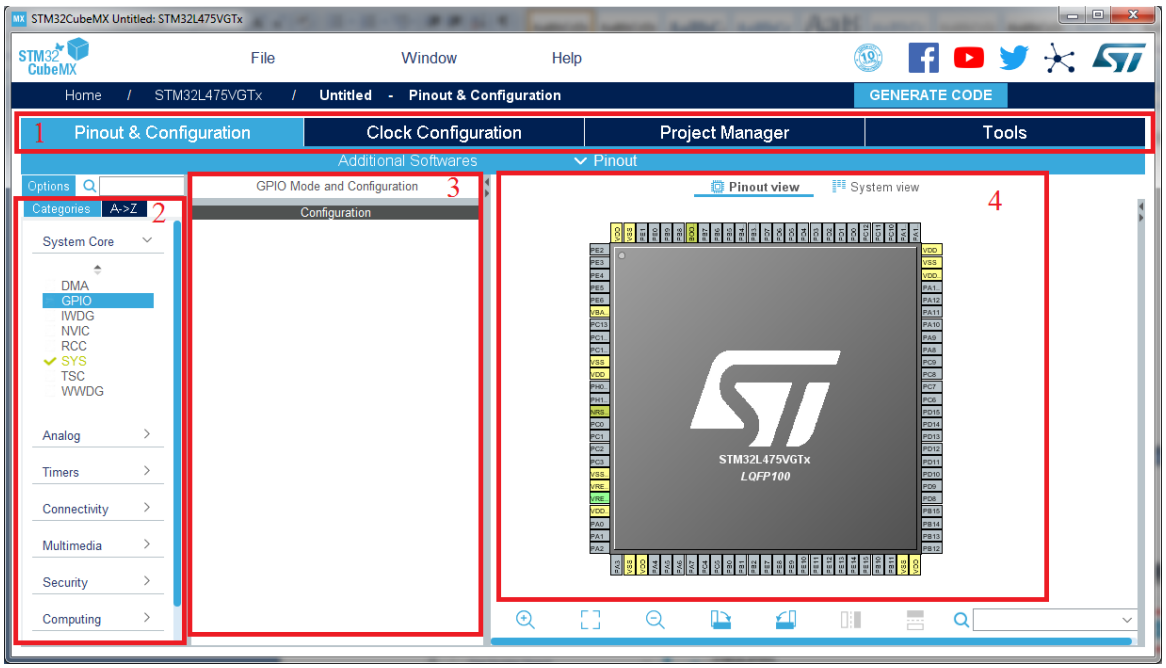

Рисунок 1.4 – Рабочая область программы

В данной лабораторной работе мы включим подключенный к микроконтроллеру светодиод. Для этого необходимо настроить GPIO микроконтроллера. В соответствии с документацией платы B-L475E-IOT01A можно узнать, что зеленый светодиод на плате подключен к пину номер 14, порта B микроконтроллера (PB14) [1].

Наведите курсор мыши на окно «Pinout view», покрутите колесико мышки чтобы увеличить или уменьшить изображение микроконтроллера. Зажав левую клавишу мыши можно перемещать изображение микроконтроллера в окне. Найдите пин PB14, и кликните на него левой кнопкой мыши. Появится вспомогательное меню, как показано на рисунке 1.5.

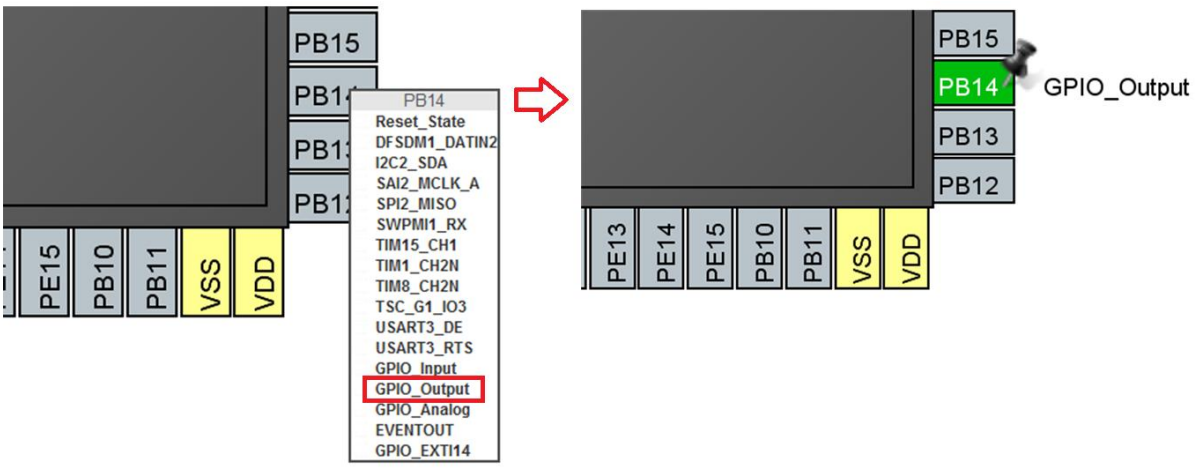

Рисунок 1.5 – Pinout configuration

Для того чтобы управлять светодиодом нужно настроить данный пин как «GPIO_Output». Данная настройка означает, что на данный пин может подаваться цифровой сигнал из микроконтроллера на внешнее устройство.

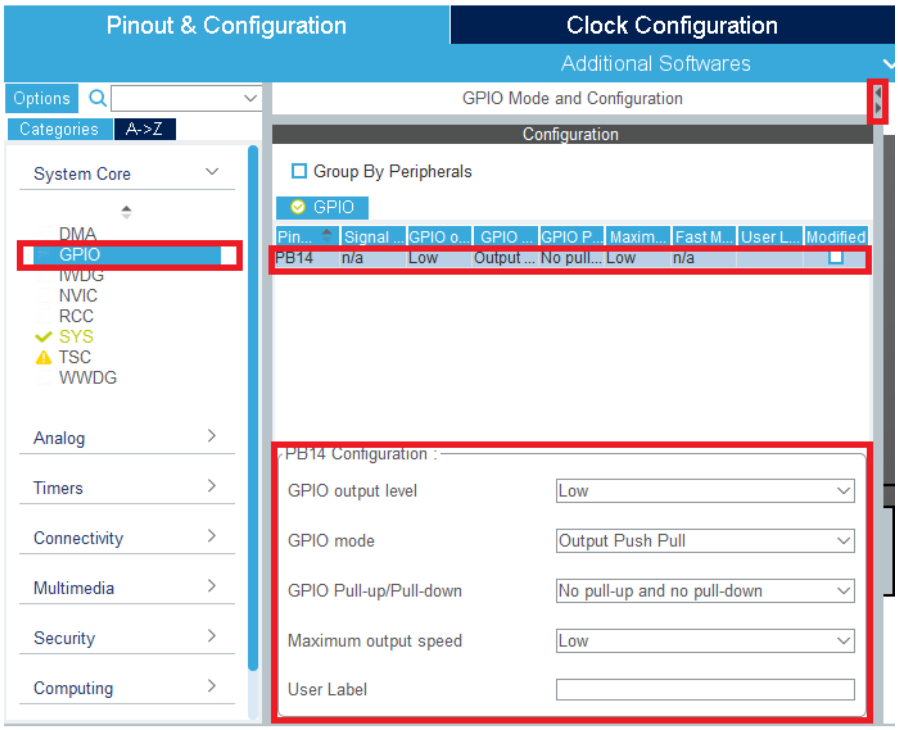

Рисунок 1.6 – GPIO configuration

В списке слева выберите «System Core», «GPIO», как показано на рисунке 1.6. После этого появится окно «GPIO Mode and Configuration» (Если данное окно не появилось, то его нужно развернуть, нажав на стрелочки, которые выделены на рисунке 1.6 в правом верхнем углу). Затем нажмите на строку PB14 в этом окне. Вы увидите в нижней части настройки пина PB14. Укажите следующую конфигурацию:

- GPIO output level – Low (уровень напряжения на данном пине после включения микроконтроллера);

- GPIO mode – Output Push Pull;
- GPIO Pull-up/Pull-down – No pull-up and no pull down;
- Maximum output speed – Low.

После того как GPIO настроены, можно приступать к настройке тактирования микроконтроллера. Микроконтроллеры STM32 могут использовать множество схем тактирования, от внутреннего генератора, от внешнего генератора, настраивать различную частоту для определенных блоков микроконтроллера. Для настройки тактирования в STM32CubeMX имеется удобный визуальный интерфейс. Чтобы открыть его нажмите на вкладку «Clock Configuration». Появится окно настройки тактирования, которое показано на рисунке 1.7.

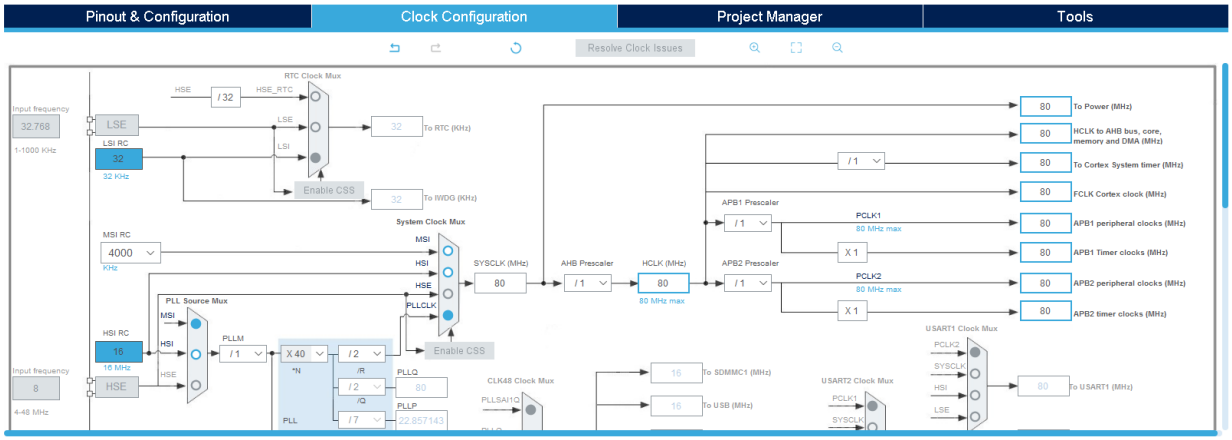

Рисунок 1.7 – Clock configuration

В блоке «PLL Source Mux» выберите «MSI», затем измените делители на «/1», «X40», «/2», в блоке «System Clock Mux» выберите «PLLCLK», как показано на рисунке 1.7. Проверьте чтобы настройки тактирования вашего проекта были такими же, как и на рисунке 1.7. Частоты всех блоков должны измениться на 80 MHz.

Далее откройте вкладку «Project Manager», вы увидите окно настройки вашего проекта. В поле «Project Name» введите название проекта «Blink». В поле «Project Location» укажите путь для сохранения проекта на жестком диске вашего компьютера. В поле «Toolchain/IDE» выберите «MDK-ARM V5», так как для дальнейшей работы мы используем программу KEIL Uvision 5.

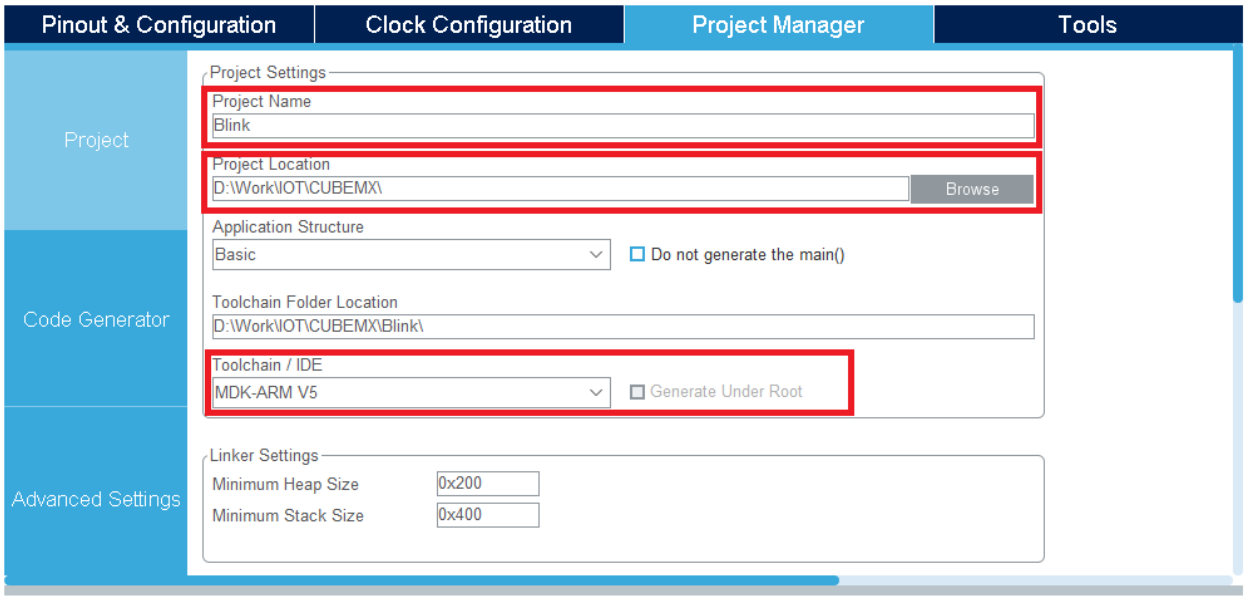

Рисунок 1.8 – Project configuration

Если вы внесли все необходимые настройки в ваш проект, можете нажать кнопку «GENERATE CODE». Начнется компиляция, если все настроено правильно вы увидите окно, которое показано на рисунке 1.9, нажмите кнопку «Open Project».

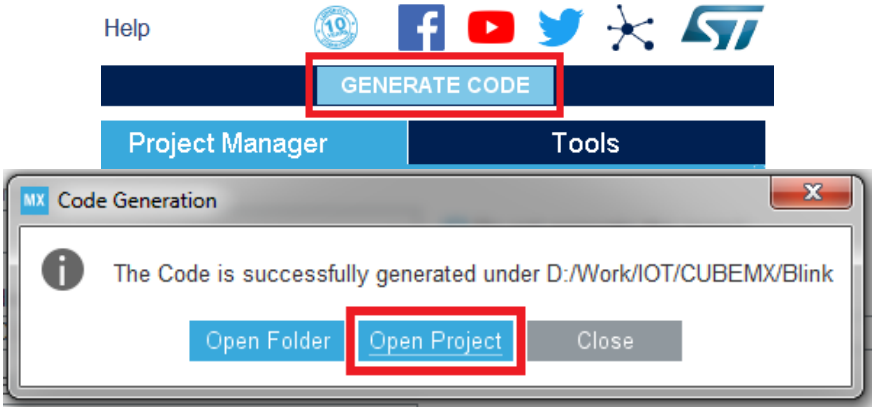

Рисунок 1.9 – Generate Code to KEIL Uvision 5

2) Далее откроется программа KEIL Uvision 5, с вашим проектом «Blink». Рабочая область программы показана на рисунке 1.10. Она включает в себя следующие блоки:

1 – Панель задач;

2 – File Toolbar;

3 – Build Toolbar;

4 – Окно иерархии проекта. В данном окне показаны все файлы и папки, которые содержит ваш проект;

5 – Рабочая область. Окно, в котором отображается содержимое открываемых файлов. В этом окне вы пишите код вашей программы;

6 – Build Output. Окно, в котором отображается процесс генерации проекта и наличие ошибок и предупреждений.

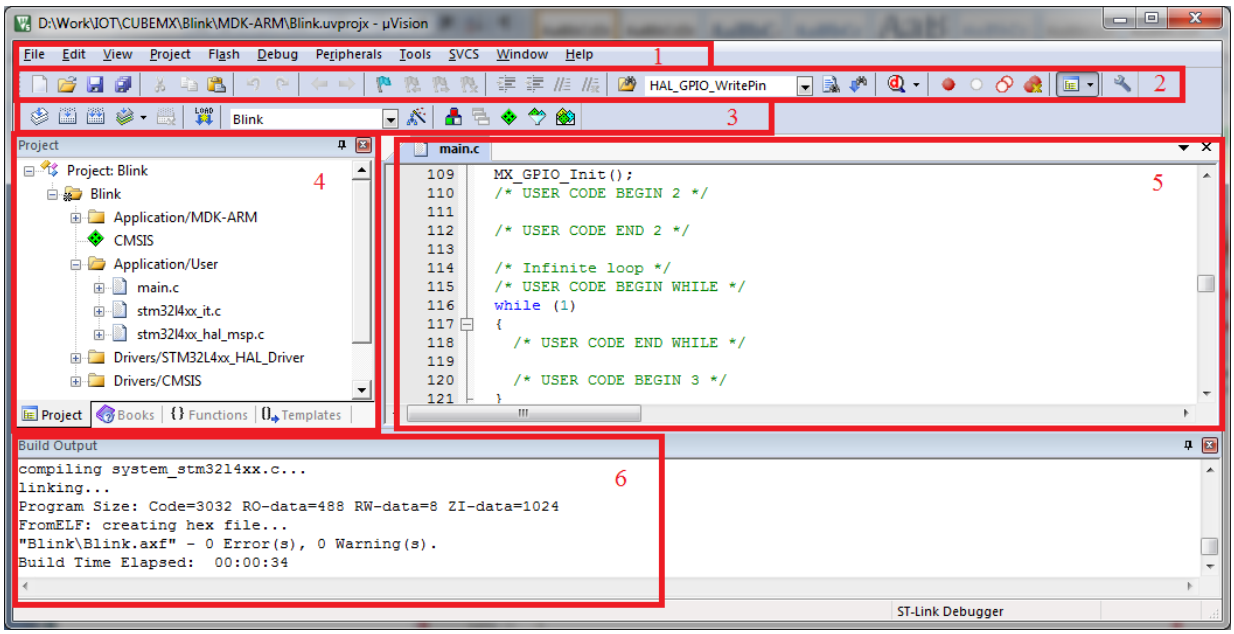

Рисунок 1.10 – KEIL Uvision 5

Пролистайте папки в окне иерархии проекта. В проекте расположены папки:

Application/MDK-ARM, Driver/CMSIS – в данных папках расположены низкоуровневые библиотеки для микроконтроллера STM32L4xx.c

Application/User – в данной папке расположены файлы, в которых пользователь будет писать свой основной код, а именно в файле main.c

Drivers/STM32L4xx_HAL_Driver – в этой папке расположены библиотеки «HAL» которые содержат множество удобных функций для работы с микроконтроллером. С данными функциями мы ознакомимся во время выполнения лабораторных работ.

Все эти библиотеки и файлы были автоматически сгенерированы программой STM32CubeMX, и содержат все те настройки, которые мы делали в графическом интерфейсе STM32CubeMX. Например, в программе STM32CubeMX вы настроили PB14. Данная программа сгенерировала функцию инициализации GPIO в файле main.c с этими настройками, как показано на рисунке 1.11.

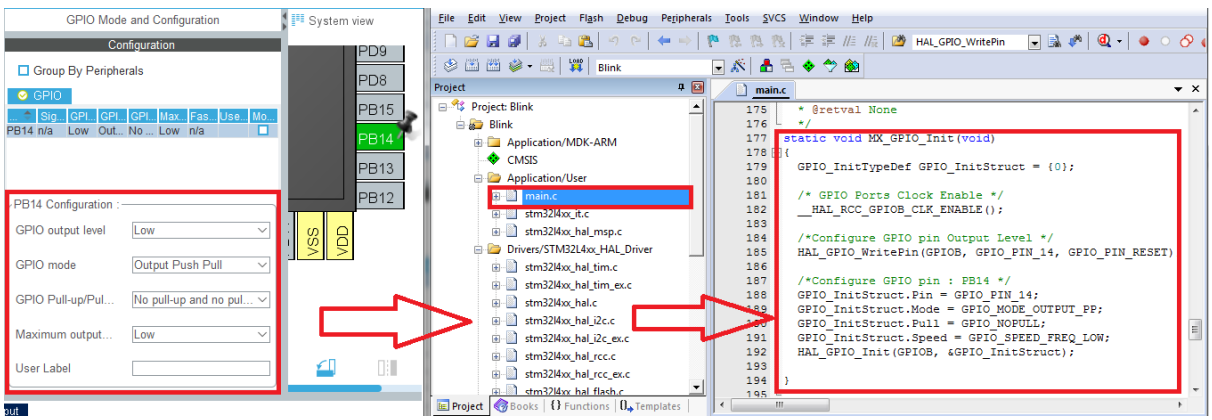

Рисунок 1.11 – Генерация функций программой CubeMX

Как мы видим на рисунке выше данный способ создания проекта экономит много времени и сил при настройке периферии микроконтроллера, но имеет и свои недостатки, которые мы рассмотрим в следующей лабораторной работе.

3) Ознакомимся подробнее с файлом main.c. Именно в этом файле описана основная часть работы вашей программы в микроконтроллере. Как и в любой программе на языке «С», команды, которые выполняет микроконтроллер расположены в функции «main». Их можно разделить на две части. Первая выполняется один раз при включении микроконтроллера (или перезагрузке), а вторая повторяется бесконечно, пока микроконтроллер включен и расположена в бесконечном цикле while (1). Каждая функция открывается и заканчивается скобками «{ }».

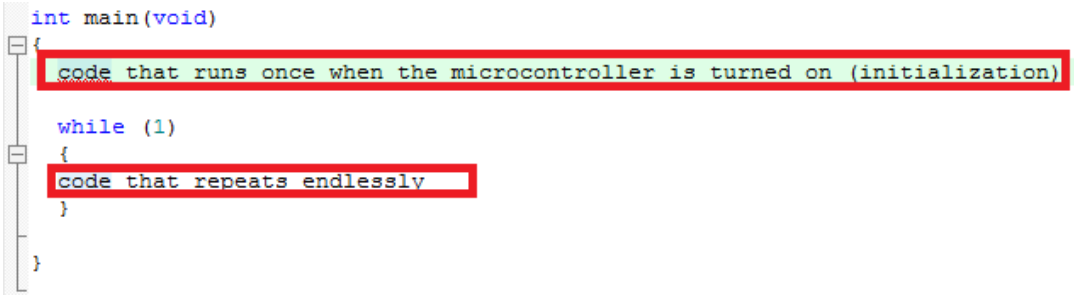

Рисунок 1.12 – Структура программы на языке «С»

Так как программа CubeMX сама генерирует код, то вы можете писать свой код только определенных для этого местах. Эти места выделены комментариями «USER CODE BEGIN…» и «USER CODE END…», как показано на рисунке 1.13. Если вы напишете код в другом месте, а потом измените настройки в CubeMX ваш код будет удален!

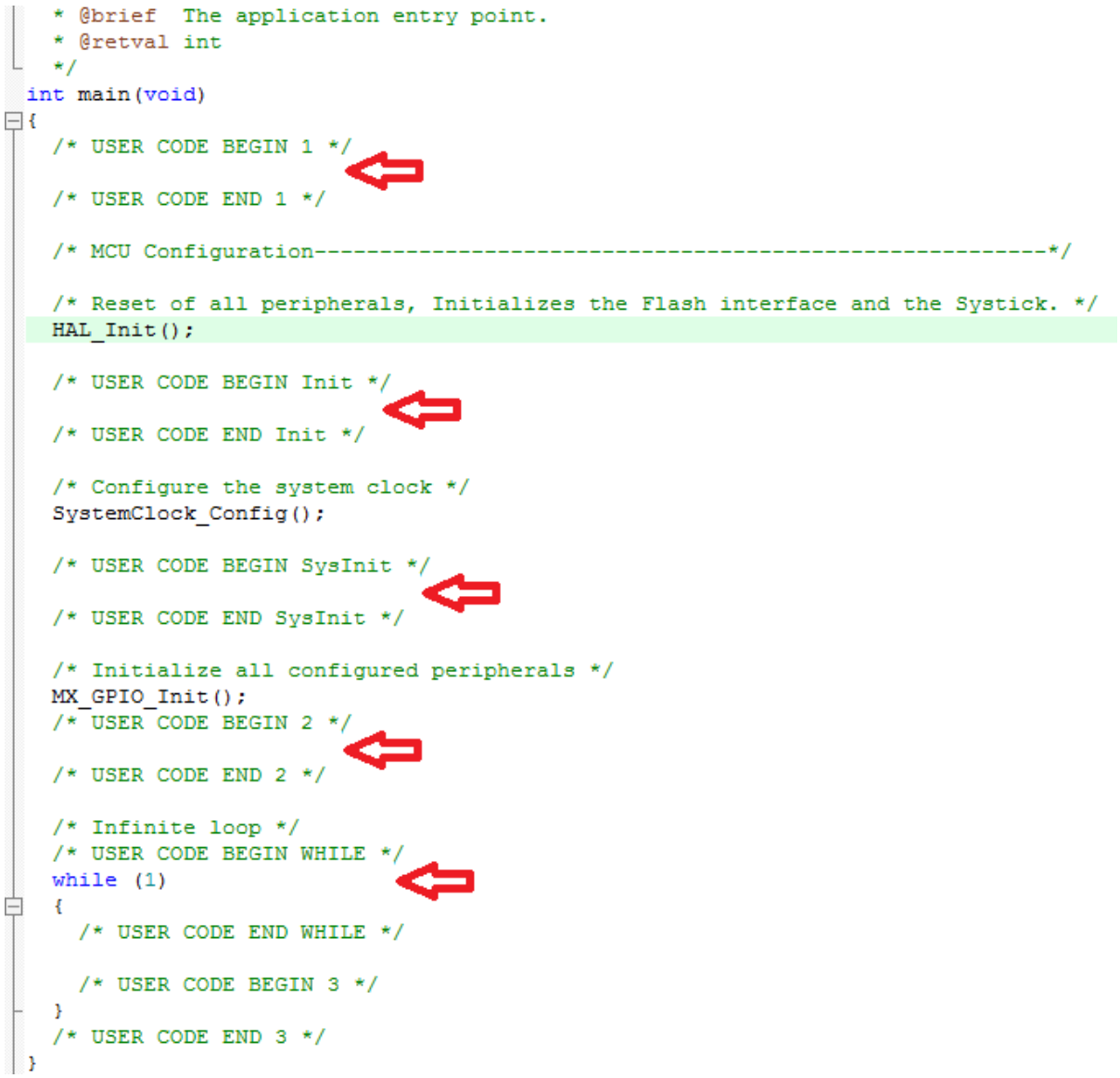

Рисунок 1.13 – Места для записи вашего кода

После того как вы узнали, как писать свою программу, напишем код для мигания светодиодом. Для этого мы должны подавать высокое, а затем низкое напряжение на пин микроконтроллера. Для этого в библиотеке HAL существует функция:

HAL_GPIO_WritePin (GPIOX, GPIO_PIN_X, GPIO_PIN_X);

Для начала напишем код между комментариями «USER CODE BEGIN WHILE» и «USER CODE END WHILE», и напишем функцию, которая подаст на порт GPIOB, на GPIO_PIN_14, высокое напряжение – GPIO_PIN_SET. Затем выставим задержку в 100 миллисекунд функцией HAL_Delay(100) и подадим низкое напряжение на этот пин – GPIO_PIN_RESET. Данный код пояснен на рисунке 1.14

```
  /* USER CODE BEGIN WHILE */
  while (1)
  {
      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_14, GPIO_PIN_SET);  // LED ON
      HAL_Delay(100);                                       // Delay 100 milliseconds
      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_14, GPIO_PIN_RESET);// LED OFF
      HAL_Delay(100);                                       // Delay 100 milliseconds
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}
```

Рисунок 1.14 – Blink code

4) Затем нужно сгенерировать ваш код. Для этого нажмите «Rebuild all target files» в Build Toolbar как показано на рисунке 1.15. Начнется генерация проекта и через несколько минут в окне Build Output будет написано «0 Errors 0 Warnings». Это означает что ошибок в проекте нет, и успешно создан файл в формате «.hex» для микроконтроллера.

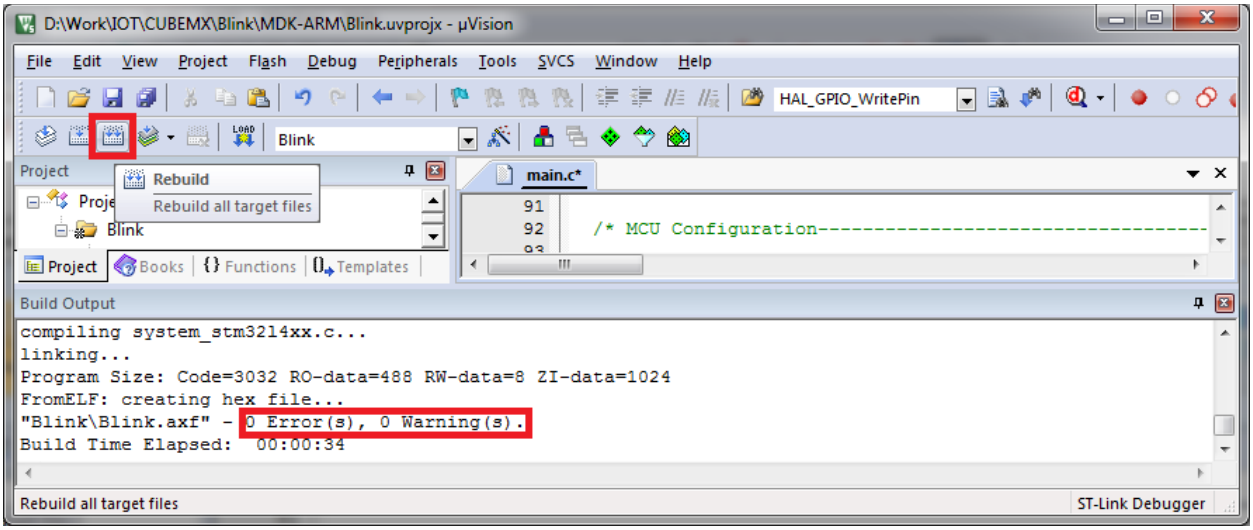

Рисунок 1.15 – Build

Присоедините плату B-L475E-IOT01A к вашему компьютеру кабелем USB в разъем под номером 1, как показано на рисунке 1.16. Должен загореться красный светодиод, как показано на рисунке 1.16 под номером 1.

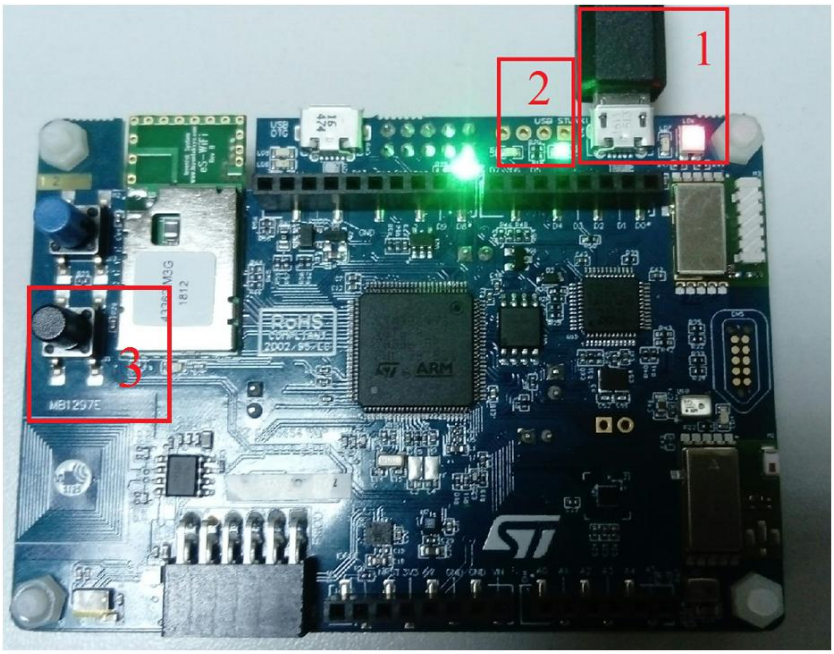

Рисунок 1.16 – B-L475E-IOT01A

Затем нужно загрузить вашу программу в микроконтроллер. Для этого нажмите «Download code to flash memory» в Build Toolbar как показано на рисунке 1.17. Начнется загрузка проекта и через несколько секунд в окне Build Output будет написано «Programming Done». Это означает, что программа успешно загружена.

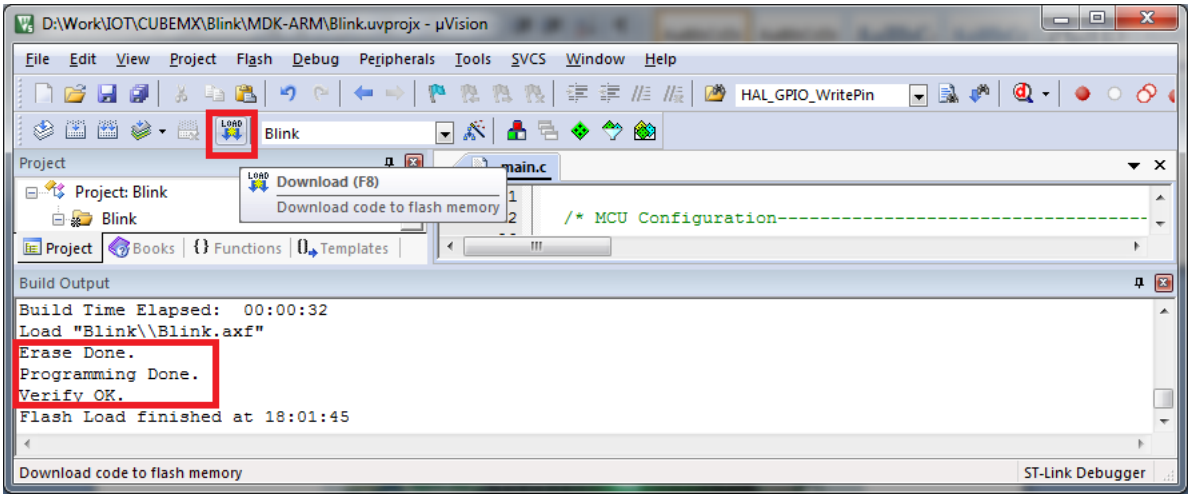

Рисунок 1.7 – Загрузка программы в память микроконтроллера

Нажмите кнопку RESET на плате B-L475E-IOT01A, как показано на рисунке 1.16 под номером 3. Если вы все сделали верно, то светодиод (номер 2 на рисунке 1.16) должен начать мигать.

Измените частоту миганий светодиода в вашем коде и повторите «Build», а затем «Download». Проверьте как изменилась работа светодиода на плате B-L475E-IOT01A.

Контрольные вопросы

1) Опишите основные блоки программы CubeMX.
2) Опишите назначение программы CubeMX.
3) Как настроить периферию микроконтроллера в CubeMX?
4) Опишите основные функции и структуры Keil µVision 5.
5) Опишите характеристики и назначение B-L475E-IOT01A.

Содержание отчета

1) Цель работы;
2) Подробное описание всех этапов проделанной работы;
3) Анализ проделанной работы;
4) Выводы по данной лабораторной работе.

2 Лабораторная работа №2
«Работа со встроенными системами. Программирование МК»
Основные функции в СИ.

Цель работы: ознакомиться с основными функциями языка программирования Си. Применить данные функции во встроенной системе.

Задачи лабораторной работы:

1) Изучить основные функции и операторы языка программирования Си.
2) Применить данные функции в проекте встроенной системы.
3) Произвести генерацию проекта и загрузить прошивку в плату B-L475E-IOT01A.

Оборудование и программное обеспечение: плата B-L475E-IOT01A, кабель USB, среда разработки Keil Uvusuion 5.

Теоретический материал

С – это язык программирования общего назначения, чрезвычайно популярный, простой и гибкий. Это машинно-независимый, структурированный язык программирования, который широко используется в различных приложениях. В языке СИ представлены типы данных приведенные в таблице 2.1.

Таблица 2.1 Типы данных в языке СИ

Тип данных	Описание	Диапазон
uint8_t	unsigned char (8-bit)	0…255
int8_t	signed char (8-bit)	-128…+127
uint16_t	unsigned short (16-bit)	0…65535
int16_t	signed short (16-bit)	-32768…+32767
uint32_t	unsigned int (32-bit)	0…65535
int32_t	signed int (32-bit)	-32768…+32767
uint64_t	unsigned long long (64-bit)	0 to 18,446,744,073,709,551,615
int64_t	signed long long (64-bit)	–9,223,372,036,854,775,808-9,223,372,036,854,775,807
float	single precision floating number (32-bit)	±1.175494E-38…±3.402823E+38
double	double precision floating number (64-bit)	2.22507385850720138e-308-1.79769313486231571e+308

Так же важно понимать представление данных в разных системах счисления, так как для работы с микроконтроллером нужно уметь работать с двоичными данными и их представление в шестнадцатеричном виде:

Для представления числа в шестнадцатеричной системе счисления используются «0x», для десятичной ничего не пишется, а в двоичной системе счисления записать в Keil нельзя (поэтому всегда нужно считать это в уме)

Пример:

Таблица 2.2 Разные системы счисления

Десятичная	Шестнадцатеричная	двоичная
93	0x5D	1011101

Арифметические операции

Арифметический оператор выполняет математические операции, такие как сложение, вычитание, умножение, деление и т. д. Над числовыми значениями (константами и переменными).

«+» сложение или одинарный плюс

«-» вычитание или унарный минус

«*» умножение

«/» деление

«%» остаток после деления (по модулю деления)

Операторы инкремента и декремента

В С-программировании есть два оператора приращения «++» и декремента «--» для изменения значения операнда (константы или переменной) на 1.

Операторы присваивания

Оператор присваивания используется для присвоения значения переменной. Наиболее распространенным оператором присваивания является знак равно «=». Операторы приведены в таблице 2.3.

Таблица 2.3 Операторы присваивания

=	+ =	- =	/ =	% =	* =
a = b	a + = b	a - = b	a / = b	a% = b	a * = b
b = a	a = a + b	a = a-b	a = a / b	a = a% b	a = a * b

Операторы отношений

Оператор отношений проверяет связь между двумя операндами. Если отношение истинно, возвращается 1; если отношение ложно, оно возвращает значение 0.

Оператор отношений используются в принятии решений и в циклах. Операторы приведены в таблице 2.4.

Таблица 2.4 Операторы отношений

Оператор	Обозначение	Пример	Оценивается как
==	Равно	5 == 3	0
>	Больше	5> 3	1
<	Меньше	5 <3	0
! =	Не равно	5! = 3	1
> =	Больше или равно	5> = 3	1
<=	Меньше или равно	5 <= 3	0

Логические Операторы

Выражение, содержащее логический оператор, возвращает 0 или 1 в зависимости от того, является ли выражение истинным или ложным. Логические операторы обычно используются при принятии решений в С-программировании.

«&&» Логическое И.

«||» Логическое ИЛИ.

«!» Логическое НЕ.

Битовые операторы

Во время вычислений математические операции, такие как сложение, вычитание, умножение, деление и т.д., преобразуются в битовый уровень, что ускоряет обработку и экономит электроэнергию.

Битовые операторы используются в С-программировании для выполнения операций на битовом уровне.

«&» Побитовое И

«|» Побитовое ИЛИ

«<<» Сдвиг влево

«>>» Сдвиг вправо

Ход работы

1) Откройте папку с проектом, который вы выполнили в лабораторной работе №1. Для этого запустите файл Blink.uvprojx в папке, в которой вы сохранили проект лабораторной 1, как показано на рисунке 2.1

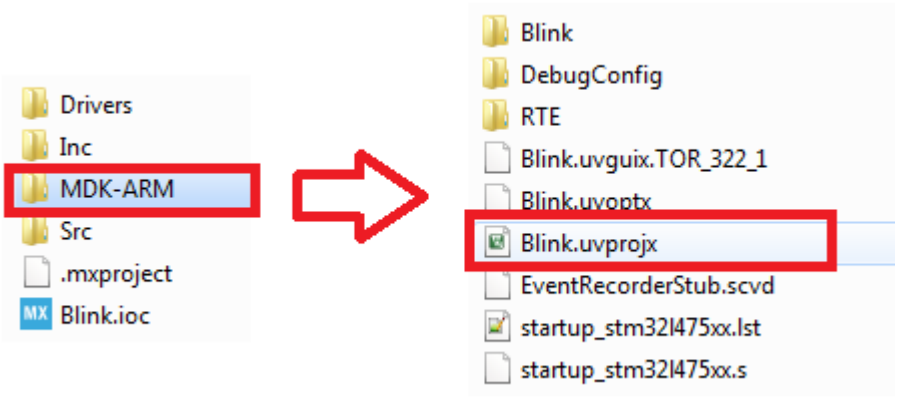

Рисунок 2.1 – Project location

2) Первым мы проверим работу оператора условия (if). Для этого откройте файл main.c. В разделе «/* USER CODE BEGIN PV */ /* USER CODE END PV */» приведите инициализацию переменно «a», типа unsigned char. В языке Си все переменные нужно сначала проинициализировать, то есть указать их тип и имя переменной.

```
64  /* Private variables ---------------------------------------------------------*/
65
66  /* USER CODE BEGIN PV */
67  uint8_t a;
68  /* USER CODE END PV */
69
70  /* Private function prototypes -----------------------------------------------*/
71  void SystemClock_Config(void);
72  static void MX_GPIO_Init(void);
```

Рисунок 2.2 – Переменные

3) Затем замените ваш код в цикле while (1) на условный оператор. Для этого сначала приравняйте переменную, а к нулю. Затем напишите условие: если a=1, то нужно включить светодиод. Иначе – нужно выключить светодиод (рисунок 2.3).

```
  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {

    a = 1;
       if (a == 1)
       {
       HAL_GPIO_WritePin(GPIOB, GPIO_PIN_14, GPIO_PIN_SET);
       }
       else
       {
       HAL_GPIO_WritePin(GPIOB, GPIO_PIN_14, GPIO_PIN_RESET);
       }
      /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}
```

Рисунок 2.3 – Цикл

4) Затем нужно сгенерировать ваш код. Для этого нажмите «Rebuild all target files» в Build Toolbar. Начнется генерация проекта и через несколько минут в окне Build Output будет написано «0 Errors 0 Warnings». Это означает что ошибок в проекте нет, и успешно создан файл в формате .hex для микроконтроллера.

Присоедините плату B-L475E-IOT01A к вашему компьютеру кабелем USB в разъем.

Затем нужно загрузить вашу программу в микроконтроллер. Для этого нажмите «Download code to flash memory» в Build Toolbar. Начнется загрузка проекта и через несколько секунд в окне Build Output будет написано «Programming Done». Это означает что программа успешно загружена.

Нажмите кнопку RESET на плате B-L475E-IOT01A. Если вы все сделали верно, то светодиод PB14 должен включиться.

Измените значение переменной «a» на 0. Сгенерируйте проект, затем загрузите в плату. Проверьте как ведет себя светодиод.

Измените значение переменной «a» на 4. Измените условие так: Если значение «a» больше 2, то нужно включить светодиод. Иначе – выключить светодиод

5) Добавьте в раздел «/* USER CODE BEGIN PV */ /* USER CODE END PV */» переменную «b», типа unsigned char. Присвойте ей значение 1. Переменной «a» присвойте значение 0. Измените оператор if так, чтобы проверялось условие: Если «a» ИЛИ «b» равно 1, то нужно включить светодиод, иначе выключить. Скомпилируйте и загрузите код.

```
/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
a = 1;
b = 0;
  if (a==1 && b==1)
  {
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_14, GPIO_PIN_SET);
  }
  else
  {
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_14, GPIO_PIN_RESET);
  }
  /* USER CODE END WHILE */
```

Рисунок 2.4 – Цикл с условием

6) Затем измените условие на: Если «a» И «b» равно 1, то нужно включить светодиод, иначе выключить. Объясните показания светодиода и работу данного кода.

7) Затем измените условие на: Если НЕ «a» И НЕ «b» равно 1, то нужно включить светодиод, иначе выключить. Объясните показания светодиода и работу данного кода.

8) Далее изучим цикл for. Для этого удалите строчки кода с оператором условия. Оставьте цикл while пустым, как показано на рисунке 2.5.

В область «/* USER CODE BEGIN 2 */ /* USER CODE END 2 */» запишите цикл for, как показано на рисунке 2.5. В данном цикле мы вводим переменную «i» равной 1, которая будет увеличиваться на 1 при выполнении цикла 1 раз. Пишем условие окончания цикла: – когда i будет больше чем 5, цикл не будет выполняться. В тело цикла напишите код мигания светодиодом, с задержкой 1000мс.

```
  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  /* USER CODE BEGIN 2 */
  for(int i=1; i<=5; i++)    //from 1 to 5 in increments of 1
{
      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_14, GPIO_PIN_SET);   // LED ON
      HAL_Delay(1000);                                       // Delay 1000 milliseconds
      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_14, GPIO_PIN_RESET); // LED OFF
      HAL_Delay(1000);                                       // Delay 1000 milliseconds
}
  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {


      /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}
```

Рисунок 2.5 – Цикл for

Скомпилируйте проект и загрузите в плату. Сколько раз загорелся светодиод? Это количество раз выполнилось тело цикла.

Так же циклом вы можете изменять параметры функции. Измените цикл for, как показано на рисунке 2.6. Затем в тело цикла напишите мигание светодиодом. Замените значение задержки между миганиями на «i» (HAL_Delay(i)). Скомпилируйте проект и загрузите в плату. Как меняется работа светодиода?

```
  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  /* USER CODE BEGIN 2 */
      //remove previous code
  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {

    for(int i=20; i<=50; i++)    //from 20 to 50 in increments of 1
{
      // your LED flashing code
}
      /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}
```

Рисунок 2.6 – Цикл for

9) Добавьте к этому коду второй цикл for, в котором частота мигания светодиодом будет уменьшаться. То есть нужно, чтобы частота мигания увеличивалась, а затем уменьшалась. Скомпилируйте код и загрузите в плату.

Контрольные вопросы

1) Опишите основные операции языка Си.
2) Опишите основные операторы языка Си.
3) Приведите примеры использования операторов языка Си?

Содержание отчета

1) Цель работы;
2) Примеры использования операций и операторов Си;
3) Подробное описание всех этапов проделанной работы;
4) Анализ проделанной работы;
5) Выводы по данной лабораторной работе.

3 Лабораторная работа №3
«Работа со встроенными системами. Программирование МК. Инициализация GPIO»

Цель работы: Получить навыки работы с документацией микроконтроллера. Настроить периферию микроконтроллера согласно данной документации без использования сторонних библиотек.

Задачи лабораторной работы:

1) Ознакомиться с документацией платы B-L475E-IOT01A.
2) Произвести настройку GPIO по документации микроконтроллера.
3) Произвести генерацию проекта и загрузить прошивку в плату B-L475E-IOT01A.

Оборудование и программное обеспечение: плата B-L475E-IOT01A, кабель USB, среда разработки Keil Uvusuion 5, B-L475E-IOT01A datasheets files.

Теоретический материал

Микроконтроллер – это компьютер, представленный в единой интегральной схеме, который предназначен для выполнения одной задачи и выполнения одного конкретного приложения. Он содержит память, программируемую периферию ввода-вывода, а также процессор, называемый также ядром. Разрядность ядра для микроконтроллеров STM32 составляет 32 бита.

Микроконтроллер в основном содержит следующие компоненты:

- Центральный процессор (ЦП);
- Оперативная память (RAM);
- Постоянная память (ПЗУ);
- Порты ввода / вывода;
- Таймеры и Счетчики;
- Управление прерываниями;
- Аналого-цифровые преобразователи;
- Цифро-аналоговые преобразователи.

В данной лабораторной работе мы рассмотрим GPIO.

GPIO – это порты ввода-вывода общего назначения. Через них микроконтроллер соединяется с внешними частями схемы (LEDs, кнопками и т.д.).

Микроконтроллеры семейства STM32 имеют несколько цифровых портов, называемых GPIOA, GPIOB, GPIOC.

Каждый порт имеет 16 бит и, следовательно, 16 электрических контактов. Контакты обозначаются как P_{xy}, где x - имя порта (A, B, ..., E), а y - бит (0, 1, ..., 15).

Например, контакт PC3 является битом 3 порта C. Каждый PIN-код также имеет альтернативную функцию, связанную с периферийным устройством, например, таймер, UART, SPI и т. д. Функциональная схема представлена на рисунке 3.1.

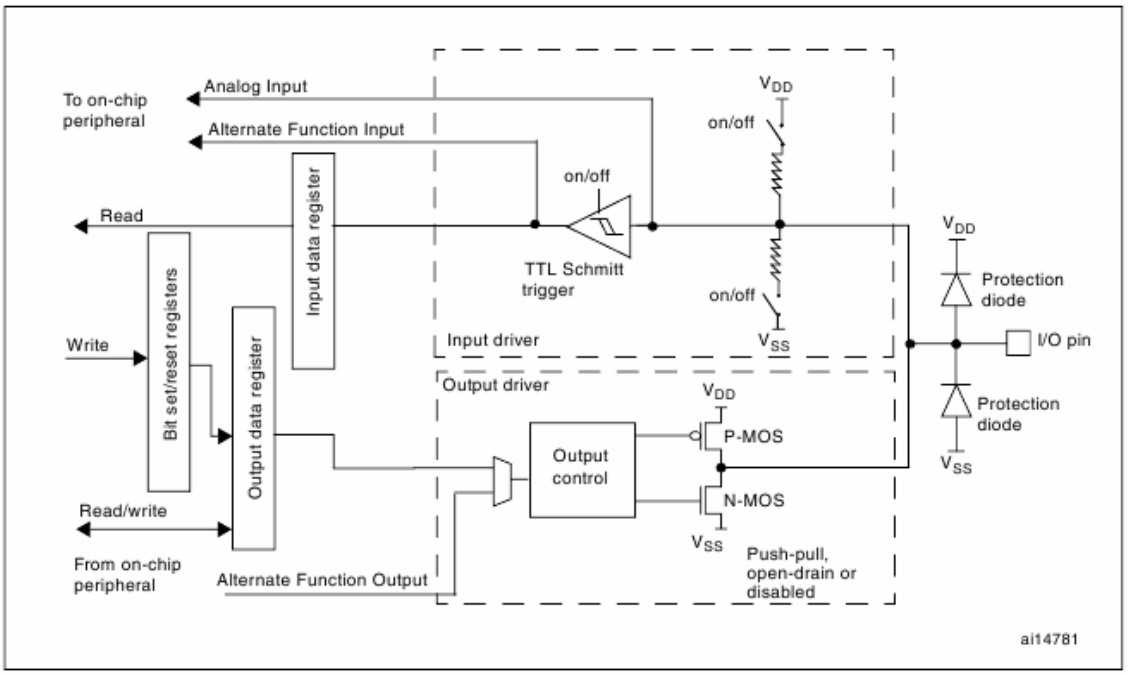

Рисунок 3.1 – Функциональная схема GPIO[2]

STM32 позволяет настроить GPIO различными способами:
- Input floating;
- Input pull-up;
- Input-pull-down;
- Analog;
- Output push-pull with pull-up or pull-down capability.

Ход работы

1) Запустите программу CubeMX. Для этого на панели задач откройте меню «Пуск», «Все программы», «STMicroelectronics», «STM32Cube», «STM32CubeMX», «STM32CubeMX.exe».

Нажмите на кнопку «ACCESS TO MCU SELECTOR». Далее откроется окно для выбора микроконтроллера, в списке «MCUs list» выберите микроконтроллер STM32L475VGTx и нажмите на кнопку «Start Project».

В данной лабораторной работе мы не будем настраивать GPIO визуальным интерфейсом, для понимания работы микроконтроллера.

Нажмите на вкладку «Clock Configuration». Появится окно настройки тактирования, которое показано на рисунке 3.2.

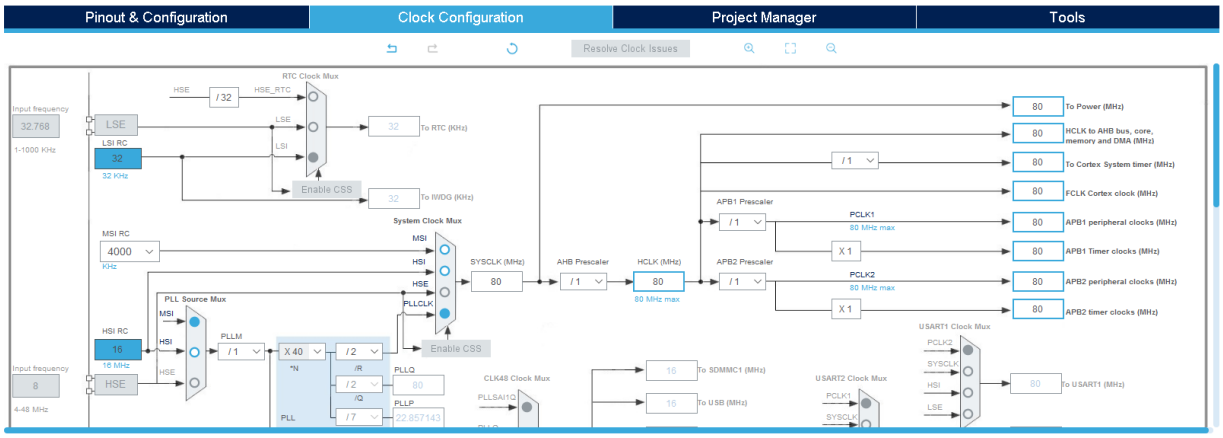

Рисунок 3.2 – Clock configuration

В блоке «PLL Source Mux» выберите «MSI», затем измените делители на «/1», «X40», «/2», в блоке «System Clock Mux» выберите «PLLCLK», как показано на рисунке 3.2. Проверьте чтобы настройки тактирования вашего проекта были такими же, как и на рисунке 3.2. Частоты всех блоков должны измениться на 80 MHz.

Далее откройте вкладку «Project Manager», вы увидите окно настройки вашего проекта. В поле «Project Name» введите название проекта «GPIO». В поле «Project Location» укажите путь для сохранения проекта на жестком диске вашего компьютера. В поле «Toolchain/IDE» выберите «MDK-ARM V5», так как для дальнейшей работы мы используем программу KEIL Uvision 5.

2) Если вы внесли все необходимые настройки в ваш проект, можете нажат кнопку «GENERATE CODE». Затем нажмите кнопку «Open Project».

Далее откроется программа KEIL Uvision 5, с вашим проектом «GPIO». Так как мы не настроили GPIO в CubeMX, то при генерации проекта они работать не будут.

Для настройки GPIO в данной лабораторной работе настроим один пин на выход для светодиода, а второй для подключения кнопки. Для этого необходимо обратиться к документации платы B-L475E-IOT01A [1].

В таблице «STM32L4 Discovery kit for IoT node I/O» можно увидеть, что LED2 подключен к порту Б, пину 14 (PB14). Для того чтобы управлять светодиодом нужно настроить его как двухтактный выход. Чтобы это сделать, обратимся к документации микроконтроллера STM32L475 [4]

Как видно из документации, каждый порт имеет 11 управляющих регистров:

- MODER: настраивает каждый бит как вход или выход;
- OSPEEDR: настраивает максимальную частоту выходного контакта;
- PUPDR: configures the internal pull-up or pull-down register;
- IDR: регистр входных данных;
- ODR: регистр выходных данных;
- BSRR: бит установлен / сброшен регистр;
- AFRL, AFRH: регистры конфигурации альтернативных функций;

- LCKR: регистр блокировки бит;

- OTYPER: конфигурация выходного типа (двухтактный или открытый сток).

Для того чтобы получить доступ к данным регистрам существуют структуры портов: GPIOA, GPIOB, GPIOC. Функции для работы с реестрами находятся внутри этих структур.

Получить доступ к управляющему регистру можно с помощью указателя структуры «->»: GPIOA -> ODR

3) в первую очередь нужно включить тактирование порта Б, для того чтобы GPIO порта Б могли работать. Для настройки тактирование у STM32L475 имеется регистр «AHB2ENR».

Откроем документацию микроконтроллера STM32L475 [4 пункт 6.4.17 AHB2 peripheral clock enable register (RCC_AHB2ENR)], в котором описан регистр AHB2ENR.

При этом если записать «1» – в соответствующий бит, тактирование будет включено, а если «0» – то выключено.

Узнаем, как в языке Си изменить значение одного бита в регистре:

STRUCTURE-> REGISTER &=~(1 << BIT_NUMBER);

В данной формуле мы выбираем указатель на REGISTER из соответствующей STRUCTURE. В этом регистре мы выбираем бит BIT_NUMBER, для этого мы 1 сдвигаем «<<» на нужное количество бит и делаем маску. Затем выполняем операцию «&=~» для того чтобы умножить весь регистр на эту маску и только этот бит изменится на «0». То есть мы инвертируем маску, все биты будут 1, а нужный нам бит 0, а затем выполняем операцию «И». В результате все биты не изменятся, а нужный бит изменится на «0».

STRUCTURE->REGISTER |= (1 << BIT_NUMBER);

В данной формуле мы выбираем указатель на REGISTER из соответствующей STRUCTURE. В этом регистре мы выбираем бит BIT_NUMBER, для этого мы 1 сдвигаем «<<» на нужное количество бит и делаем маску. Затем выполняем операцию «|=» для того чтобы умножить весь регистр на эту маску и только этот бит изменится на «1». То есть в маске все биты будут 0, а нужный нам бит 1. Затем мы выполняем операцию «ИЛИ». В результате все биты не изменятся, а нужный бит изменится на «1».

Запишем «1» в бит №1(GPIOBEN), для включения тактирования. Для этого в разделе /* USER CODE BEGIN 2 */ /* USER CODE END 2 */ запишем функцию изменения регистра AHB2ENR в структуре RCC:

RCC->AHB2ENR |= (1 << 1);

4) Настроим регистр MODE:

Откроем документацию микроконтроллера STM32L475 [4 пункт 8.4.1 GPIO port mode register (GPIOx_MODER) (x =A to I)], в ней описан регистр MODE.

MODER позволяет программисту определять функцию вывода GPIO.

Каждый вывод имеет 2 бита, которые допускают следующие конфигурации:

«00» – вход;

«01» – выход;

«10» – альтернативная функция;

«11» – аналоговый.

Значит, чтобы настроить PB14 как выход то нужно записать в регистр MODE14[1:0], «0» – в бит № 29, и «1» – в бит №28.

```
GPIOB->MODER &=~(1 << 29);
GPIOB->MODER |= (1 << 28);
```

5) Настроим регистр Output Type Register:

Откроем документацию микроконтроллера STM32L475 [4 пункт 8.4.2 GPIO port output type register (GPIOx_OTYPER) (x = A to I)], в ней описан регистр OTYPER.

OTYPER allows a programmer to configure the output stage of an output GPIO pin.

Each pin has 1 bits that permits the following configurations:

«0» – Push-pull;

«1» – Open Drain.

Значит, чтобы настроить PB14 как Push-pull (это наиболее подходит для подключения светодиода) то нужно записать в регистр OTYPER14 «0» – в бит № 14.

```
GPIOB->OTYPER &=~(1 << 14);
```

6) Настроим Output Speed Register:

Откроем документацию микроконтроллера STM32L475 [4 пункт 8.4.3 GPIO port output speed register (GPIOx_OSPEEDR), в ней описан регистр OSPEEDR.

OSPEEDR allows a programmer to define the speed of an output GPIOpin

Each pin has 2 bits that permits the following configurations:

«00» – Low speed;

«01» – Medium speed;

«10» – High speed;

«11» – Very high speed.

Значит, чтобы настроить PB14 как High speed (скорость в данном примере не важна, поэтому поставим самую большую) то нужно записать в регистр OSPEEDR 14 «1» – в бит № 29, «0» – в бит № 28,

```
GPIOB->OSPEEDR |=(1 << 29);
GPIOB->OSPEEDR &=~(1 << 28);
```

7) Настроим Pull-up/Pull-Down Register:

Откроем документацию микроконтроллера STM32L475 [4 пункт 8.4.4 GPIO port pull-up/pull-down register (GPIOx_PUPDR)], в ней описан регистр PUPDR:

PUPDR defines the presence of a pull-up or pull-down resistor (or none) at the GPIO pin

Each pin has 2 bits that permits the following configurations:

«00» – No pull-up, pull-down;

«01» – Pull-up;

«10» – Pull-down;

«11» – Reserved.

Значит, чтобы настроить PB14 как Pull-down (для устранения возможных шумов) то нужно записать в регистр PUPDR14 «1» – в бит № 29, «0» – в бит № 28,

```
GPIOB->PUPDR |= (1 << 29);
GPIOB->PUPDR &=~(1 << 28);
```

8) Настроим GPIO port bit set/reset register (GPIOx_BSRR):

Откроем документацию микроконтроллера STM32L475 [4 пункт 8.4.7 GPIO port bit set/reset register (GPIOx_BSRR)], в ней описан регистр BSRR.

Bits 31:16 BR[15:0]: Port x reset I/O pin y (y = 15 to 0) These bits are write-only. A read to these bits returns the value «0x0000».

«0» – No action on the corresponding ODx bit;

«1» – Resets the corresponding ODx bit;

«Note» – If both BSx and BRx are set, BSx has priority.

Bits 15:0 BS [15:0]: Port x set I/O pin y (y = 15 to 0). These bits are write-only. A read to these bits returns the value «0x0000».

«0» – No action on the corresponding ODx bit;

«1» – Sets the corresponding ODx bit.

Значит, чтобы зажечь светодиод на PB14 нужно записать в регистр BSRR «1» – в бит № 14

```
GPIOB-> BSRR |= (1 << 14);
```

А чтобы выключить светодиод на PB14 нужно записать в регистр BSRR «1» – в бит № 30

```
GPIOB-> BSRR |= (1 << 30);
```

В результате мы получим код представленный на рисунке 3.3.

```
  /* Initialize all configured peripherals */
  /* USER CODE BEGIN 2 */

RCC->AHB2ENR |=(1 << 1);      /* Enable GPIOB clock */
GPIOB->MODER &=~(1 << 29);    /* PB.14 Enable Output */
GPIOB->MODER |= (1 << 28);
GPIOB->OTYPER &=~(1 << 14);   /* PB.14 Push-Pull */
GPIOB->OSPEEDR |=(1 << 29);   /* PB.14 High speed */
GPIOB->OSPEEDR &=~(1 << 28);
GPIOB->PUPDR |= (1 << 29);    /* PB.14 is Pull-Down */
GPIOB->PUPDR &=~(1 << 28);
GPIOB-> BSRR |= (1 << 14);    /* PB.14 set bit */

  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}
```

Рисунок 3.3 – Включение светодиода

9) Затем нужно сгенерировать ваш код. Для этого нажмите «Rebuild all target files» в Build Toolbar. Начнется генерация проекта и через несколько минут в окне Build Output будет написано «0 Errors 0 Warnings». Это означает что ошибок в проекте нет, и успешно создан файл в формате «.hex» для микроконтроллера.

Присоедините плату B-L475E-IOT01A к вашему компьютеру кабелем USB в разъем.

Затем нужно загрузить вашу программу в микроконтроллер. Для этого нажмите «Download code to flash memory» в Build Toolbar. Начнется загрузка проекта и через несколько секунд в окне Build Output будет написано «Programming Done». Это означает что программа успешно загружена.

Нажмите кнопку RESET на плате B-L475E-IOT01A. Если вы все сделали верно, то светодиод PB14 должен включиться.

10) К плате B-L475E-IOT01A подключена синяя пользовательская кнопка. Она присоединена к порту C пину 13 (PC13).

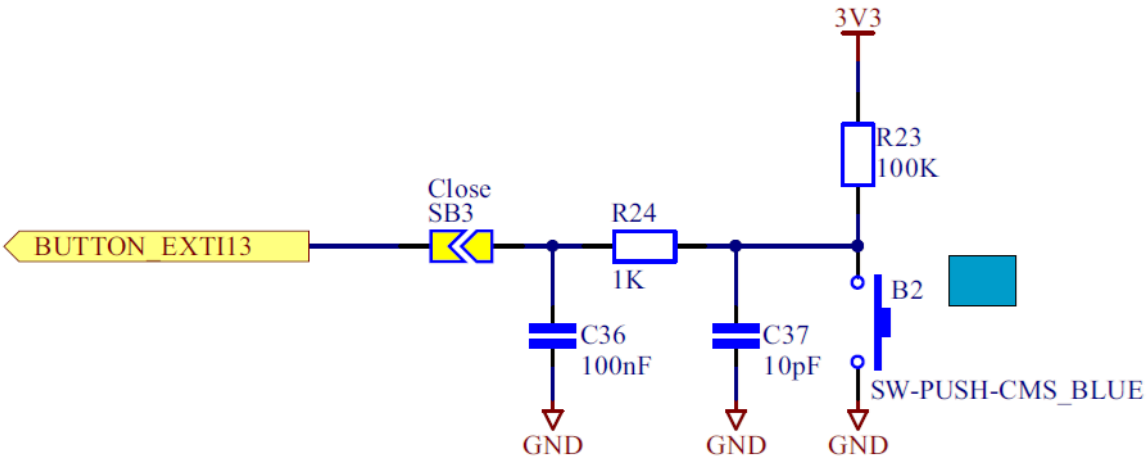

Рисунок 3.4 – User Button [1]

В разделе /* USER CODE BEGIN 2 */ /* USER CODE END 2 */ настройте этот пин на вход.

Для этого:

Регистр AHB2ENR настройте на тактирование порта C;

В регистре MODER порта C настройте пин 13 как INPUT;

В регистре PUPDR порта C настройте пин 13 как No pull-up, pull-down;

11) Затем в цикле while напишите условие: Если кнопка нажата – включить светодиод, иначе – выключить светодиод.

Для этого считайте значение регистра IDR. (Data Input Register):

Откроем документацию микроконтроллера STM32L475 [4 пункт 8.4.5 GPIO port input data register (GPIOx_IDR)].

Регистр IDR позволяет считывать состояние пина, который настроен как вход. Если значение высокого уровня, то бит будет равен «1», если низкого, то «0».

Кнопка по схеме B-L475E-IOT01A подключена резистором к VCC, значит при нажатии на кнопку регистр IDR PC13 будет равен «0», а когда кнопка не нажата равен «1».

Значит, чтобы проверить нажата ли кнопка нужно написать следующую функцию:

```
  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */
if ((GPIOC->IDR & (1 << 13)) == 0)
{
// ENABLE LED
}
else
  {
//DISABLE LED
  }

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}
```

Рисунок 3.5 – Условие

Скомпилируйте проект и загрузите в плату. Нажмите RESET на плате. Затем нажимайте на синюю кнопку на плате и проверяйте, как работает светодиод.

В данной лабораторной работе вы научились настраивать GPIO при помощи настройки регистров. Важно также сказать, что записывать информацию в регистр можно несколькими способами. Для понимания в этой лабораторной использовался наглядный способ записи бита в нужную позицию. Но многие разработчики библиотек предоставляют макросы. Макрос – это наглядное описание функции регистра. Например, запись байта в регистр можно сделать так:

A |= 0x1 ; или A |= MACRO_1 ;

То есть определенному набору байт присваивается соответствующее название. Это используется для лучшего понимания кода другими людьми и переносе вашего кода на другое оборудование. Так как при использовании одного макроса на разных микроконтроллерах он автоматически будет назначать разные регистры.

Так при настройке регистра AHB2ENR вы использовали запись:

RCC-> AHB2ENR |= (1 << 1);

Ее можно записать и через макрос:

RCC->AHB2ENR |= RCC_AHB2ENR_GPIOBEN;

Все макросы нужно искать в описании соответствующих библиотек, которые вы используете.

Контрольные вопросы

1) Как производится настройка периферии микроконтроллера?
2) Что такое GPIO?
3) Из чего состоит микроконтроллер?
4) Опишите способы настройки GPIO.
5) Опишите регистры для настройки GPIO.

Содержание отчета

1) Цель работы;
2) Подробное описание всех этапов проделанной работы;
3) Ответы на вопросы, представленные в тексте лабораторной работы;
4) Важные части вашего кода с пояснениями;
5) Анализ проделанной работы;
6) Выводы по данной лабораторной работе.

4 Лабораторная работа №4
«Работа с аналоговыми датчиками. АЦП»

Цель работы: Изучить принцип считывания показаний аналоговых датчиков микроконтроллером. Настроить периферию микроконтроллера для подключения датчика температуры.

Задачи лабораторной работы:

1) Произвести настройку АЦП по документации микроконтроллера.
2) Произвести калибровку датчика температуры.
3) Считать и проанализировать показания аналогового датчика.

Оборудование и программное обеспечение: плата B-L475E-IOT01A, кабель USB, среда разработки Keil Uvusuion 5, CubeMX.

Теоретический материал

В микроконтроллере STM32l475VG, который установлен на плате B-L475E-IOT01A присутствует аналоговый датчик температуры. Датчик температуры (TS) генерирует напряжение VTS, которое изменяется линейно в зависимости от температуры. Датчик температуры внутренне подключен к входам ADC1_IN17 и ADC3_IN17, которые используются для преобразования выходного напряжения датчика в цифровое значение.

Датчик обеспечивает хорошую линейность, но он должен быть откалиброван для получения хорошей точности измерения температуры. Так как меняется смещение датчика температуры от чипа к чипу из-за изменения процесса.

Рассчитать температуру можно двумя способами. Один использует данные из даташита к микроконтроллеру STM32l475xx, которые представлены в даташите [4 6.3.23 Temperature sensor characteristics].

Расчет температуры выполняется по формуле:

$$Temperature = (Voltage - V_{30}) / Avg\_Slope + TS\_CAL1\_TEMP; \quad (4.1)$$

$$Voltage = ADC\_Value / 4096 * V_{DDA}; \quad (4.2)$$

Где ADC_Value – значение АЦП при измерении температуры,

V_{DDA} = 3.3V – напряжение питания АЦП в плате B-L475E-IOT01A.

TS_CAL1_TEMP = 30 °C;

Для повышения точности измерения температурного датчика есть второй способ измерения. Каждое устройство имеет индивидуально откалиброванный на заводе ST датчик. Данные заводской калибровки датчика температуры хранятся STM32 в области памяти, доступной в режиме только для чтения, в таблица 4.1.

Таблица 4.1 Данные заводской калибровки датчика

Значение	Описание	Адрес памяти	Адрес памяти, определенный в библиотеке HAL
TS_CAL1	TS ADC необработанные данные, полученные при температуре of 30 °C (± 5 °C), VDDA = VREF+ = 3.0 V (± 10 mV)	0x1FFF 75A8 - 0x1FFF 75A9	TEMPSENSOR_CAL1_ADDR
TS_CAL2	TS ADC необработанные данные, полученные при температуре of 110 °C (± 5 °C), VDDA = VREF+ = 3.0 V (± 10 mV)	0x1FFF 75CA - 0x1FFF 75CB	TEMPSENSOR_CAL2_ADDR

Формула расчета:

$$Temperature = (((TS\_ADC\_DATA * VDDA / VREF) - TS\_CAL1) * (TEMPSENSOR\_CAL2\_TEMP - TEMPSENSOR\_CAL1\_TEMP)) / (TS\_CAL2 - TS\_CAL1) + TEMPSENSOR\_CAL1\_TEMP; \quad (4.1)$$

TS_ADC_DATA – необработанные данные датчика температуры, измеренные АЦП;

VDDA = 3.3V – напряжение питания АЦП в плате B-L475E-IOT01A;

VREF = 3.0V – напряжение питания АЦП при калибровке на заводе ST;

TS_CAL1 – эквивалент TS_ADC_DATA при температуре 30 °C (откалиброван на заводе);

TS_CAL2 – эквивалент TS_ADC_DATA при температуре 110 °C (откалибровано на заводе)

TEMPSENSOR_CAL1_TEMP = 30 °C;

TEMPSENSOR_CAL2_TEMP = 110 °C.

Ход работы

1) Запустите программу CubeMX. Для этого на панели задач откройте меню «Пуск», «Все программы», «STMicroelectronics», «STM32Cube», «STM32CubeMX», «STM32CubeMX.exe».

Нажмите на кнопку «ACCESS TO MCU SELECTOR». Далее откроется окно для выбора микроконтроллера, в списке «MCUs list» выберите микроконтроллер STM32L475VGTx и нажмите на кнопку «Start Project».

В вкладке «Analog» выберите «ADC1», затем в окне «Mode» поставьте галочку в пункте «Temperature Sensor Channel», как на рисунке 4.1.

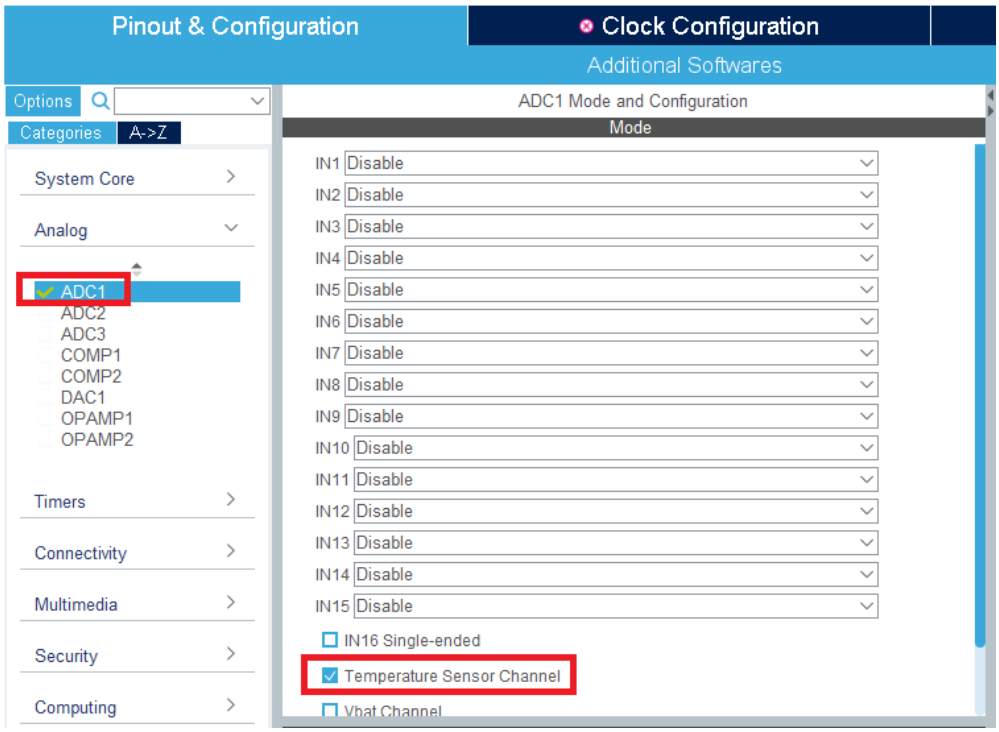

Рисунок 4.1 – ADC Mode

Во вкладке «Parameter Settings» выберите «Rank» -> «24.5 Cycles», как на рисунке 4.2.

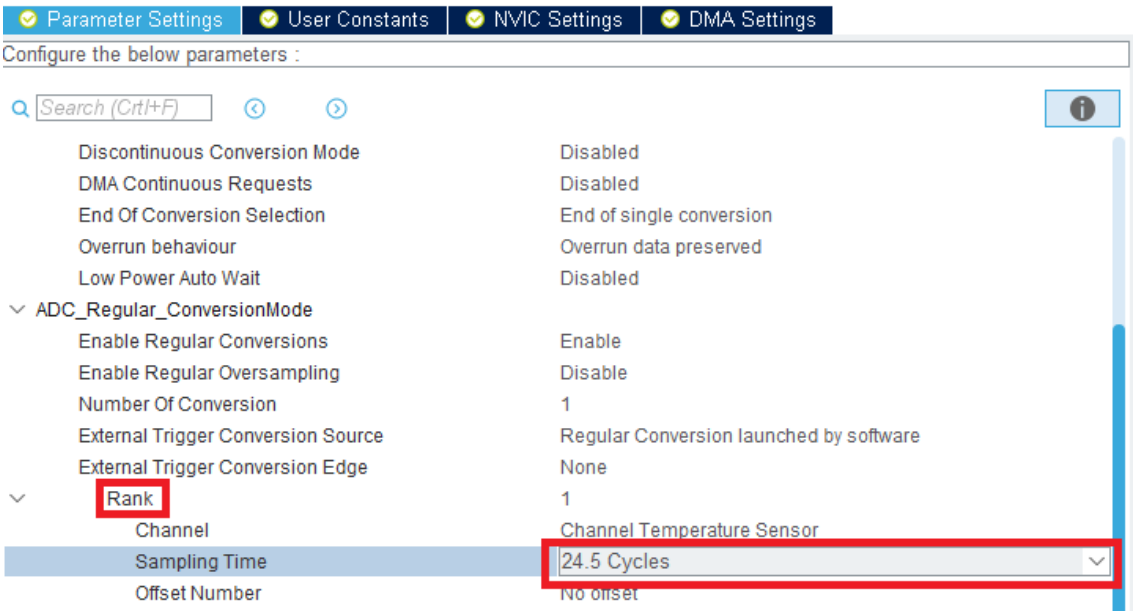

Рисунок 4.2 – Parameter Settings

Перейдите во вкладку «Clock Configuration», если появилось всплывающее окно, представленное на рисунке 4.3, то нажмите «Yes»

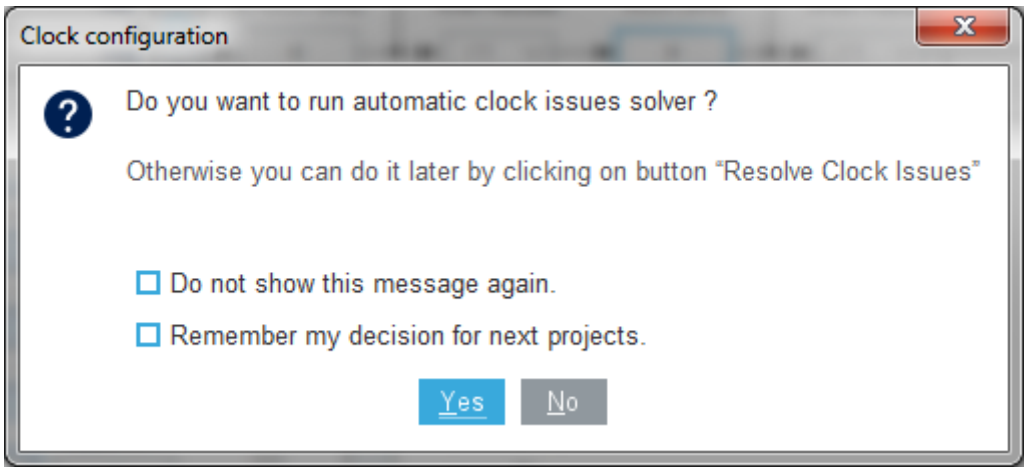

Рисунок 4.3 – Clock automatically Configuration

Настройте тактирование так же, как показано на рисунке 4.4.

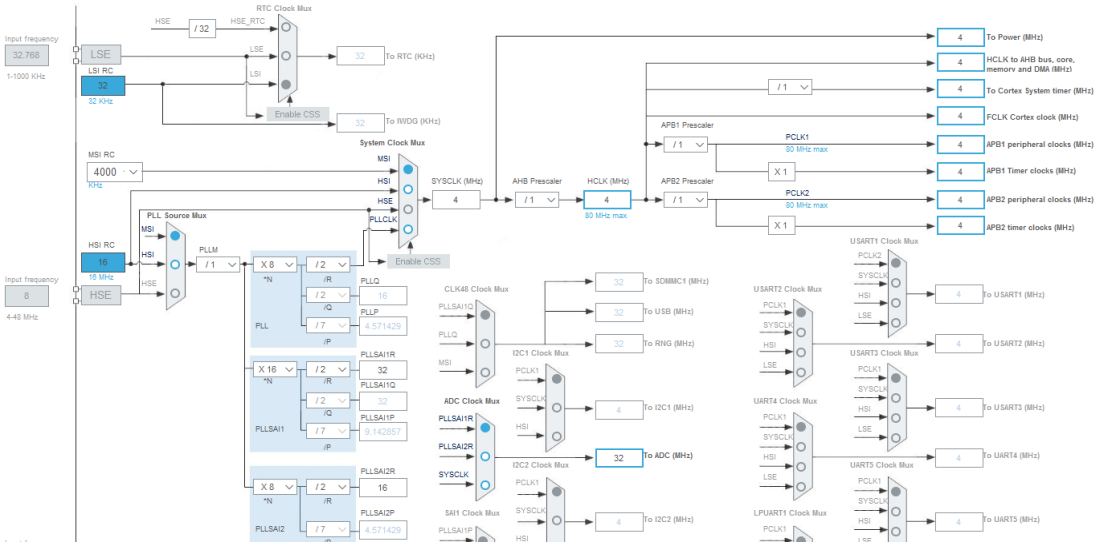

Рисунок 4.4 – Clock configuration

Далее откройте вкладку «Project Manager», вы увидите окно настройки вашего проекта. В поле «Project Name» введите название проекта «Analog_Sensor». В поле «Project Location» укажите путь для сохранения проекта на жестком диске вашего компьютера. В поле «Toolchain/IDE» выберите «MDK-ARM V5», так как для дальнейшей работы мы используем программу KEIL Uvision 5.

2) Если вы внесли все необходимые настройки в ваш проект, можете нажат кнопку «GENERATE CODE». Затем нажмите кнопку «Open Project».

Далее откроется программа KEIL Uvision 5, с вашим проектом «Analog_Sensor».

Откройте файл main.c. В раздел /* USER CODE BEGIN PD */ /* USER CODE END PD */ добавьте данные из таблицы 4.1:

```
#define V_30        0.76f;
#define Avg_Slope        0.0025f;
#define VDDA      3.3f.
```

f – в конце цифр, это тип данных float. Директива #define определяет идентификатор и последовательность символов, которой будет замещаться данный идентификатор при его обнаружении в тексте программы. Стандартный вид директивы:

#define имя_макроса последовательность_символов;

Обратим внимание, что в данном операторе отсутствует точка с запятой. Между идентификато-ром и последовательностью символов может быть любое число пробелов. Макрос завершается только переходом на новую строку.

В раздел /* USER CODE BEGIN PV */ /* USER CODE END PV */ добавьте переменные для расчета:

int16_t TS_ADC_DATA = 0;
float voltage, Temperature, Temperature2;
uint16_t TS_CAL1, TS_CAL2.

3) Далее нужно получить данные с АЦП. Для этого в цикле while (1) нужно выполнить код, который представлен на рисунке 4.5. В этом коде присутствует оператор if, он нужен для того чтобы считать данные с АЦП только тогда, когда преобразование завершится, и функция вернет значение HAL_OK.

```
  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    HAL_Delay(1000);
    HAL_ADC_Start(&hadc1);                                  // ADC conversion start
    if (HAL_ADC_PollForConversion(&hadc1, 100) == HAL_OK) //if the conversion is done
    {
        TS_ADC_DATA = HAL_ADC_GetValue(&hadc1);             //get the ADC value

    }
     HAL_ADC_Stop(&hadc1);                                  //stop ADC conversion
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}
```

Рисунок 4.5 – Get ADC value

Рассчитайте значение температуры по формуле 4.1. В результате у вас получится код, как на рисунке 4.6.

```
/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
  HAL_Delay(1000);
  HAL_ADC_Start(&hadc1);                                    // ADC conversion start
  if (HAL_ADC_PollForConversion(&hadc1, 100) == HAL_OK) //if the conversion is done
  {
      TS_ADC_DATA = HAL_ADC_GetValue(&hadc1);               //get the ADC value
      voltage = (float) TS_ADC_DATA/4096*VDDA;              //convert ADC value to volts
      Temperature = Your Code                               //Calculate the temperature value
  }
   HAL_ADC_Stop(&hadc1);                                    //stop ADC conversion
  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */
}
/* USER CODE END 3 */
```

Рисунок 4.6 – Аналоговый датчик

Затем нужно сгенерировать ваш код. Для этого нажмите «Rebuild all target files» в Build Toolbar. Начнется генерация проекта и через несколько минут в окне Build Output будет написано «0 Errors 0 Warnings». Это означает что ошибок в проекте нет, и успешно создан файл в формате «.hex» для микроконтроллера.

Присоедените плату B-L475E-IOT01A к вашему компьютеру кабелем USB в разъем.

Затем нужно загрузить вашу программу в микроконтроллер. Для этого нажмите «Download code to flash memory» в Build Toolbar. Начнется загрузка проекта и через несколько секунд в окне «Build Output» будет написано «Programming Done». Это означает что программа успешно загружена.

4) После этого запустите программу STMStudio. Эта программа позволяет смотреть значение переменных в микроконтроллере в режиме реального времени. Нажмите «Пуск», «Все программы», «STMStudio», «STMStudio.exe».

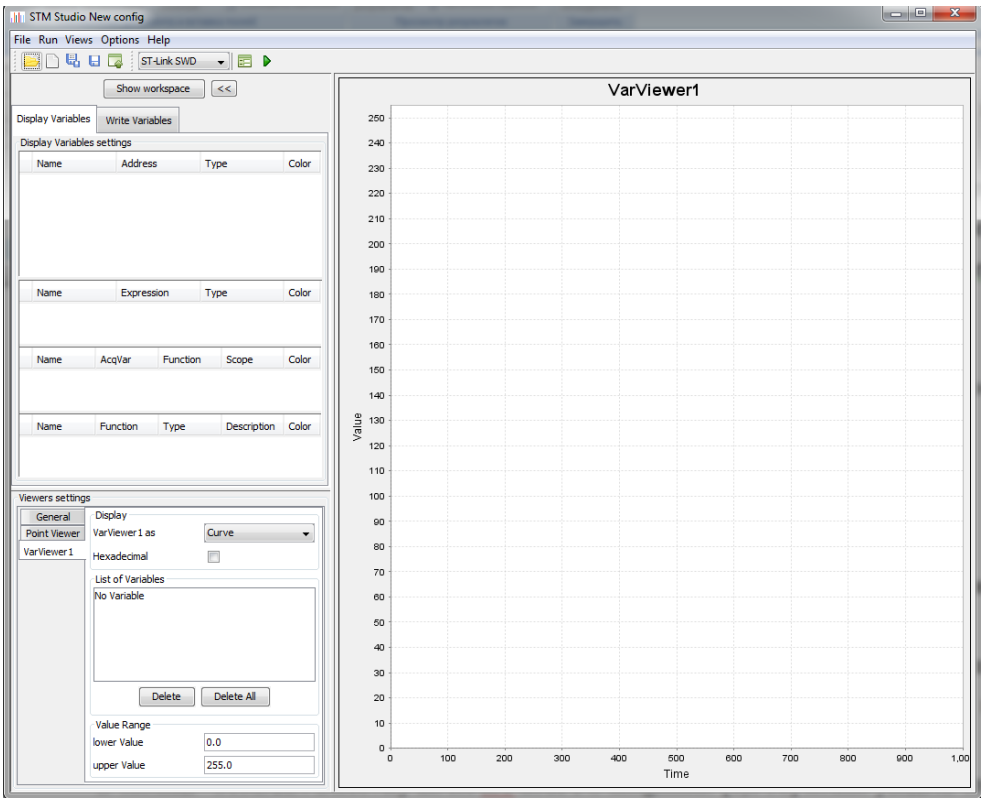

Рисунок 4.7 – Программа STMStudio

На рисунке 4.7 представлен внешний вид программы STMStudio. Она состоит из панели инструментов, «Display Variables» – где отображаются переменные вашей программы, «Viewers settings» – где вы можете настроить окно просмотра, «VarViewer» – окно просмотра переменных.

Чтобы открыть проект нажмите «File» –> «Import Variables». Откроется окно, которое видно на рисунке 4.8. Нажмите на кнопку «Browse» (Выделена красным на рисунке 4.8)

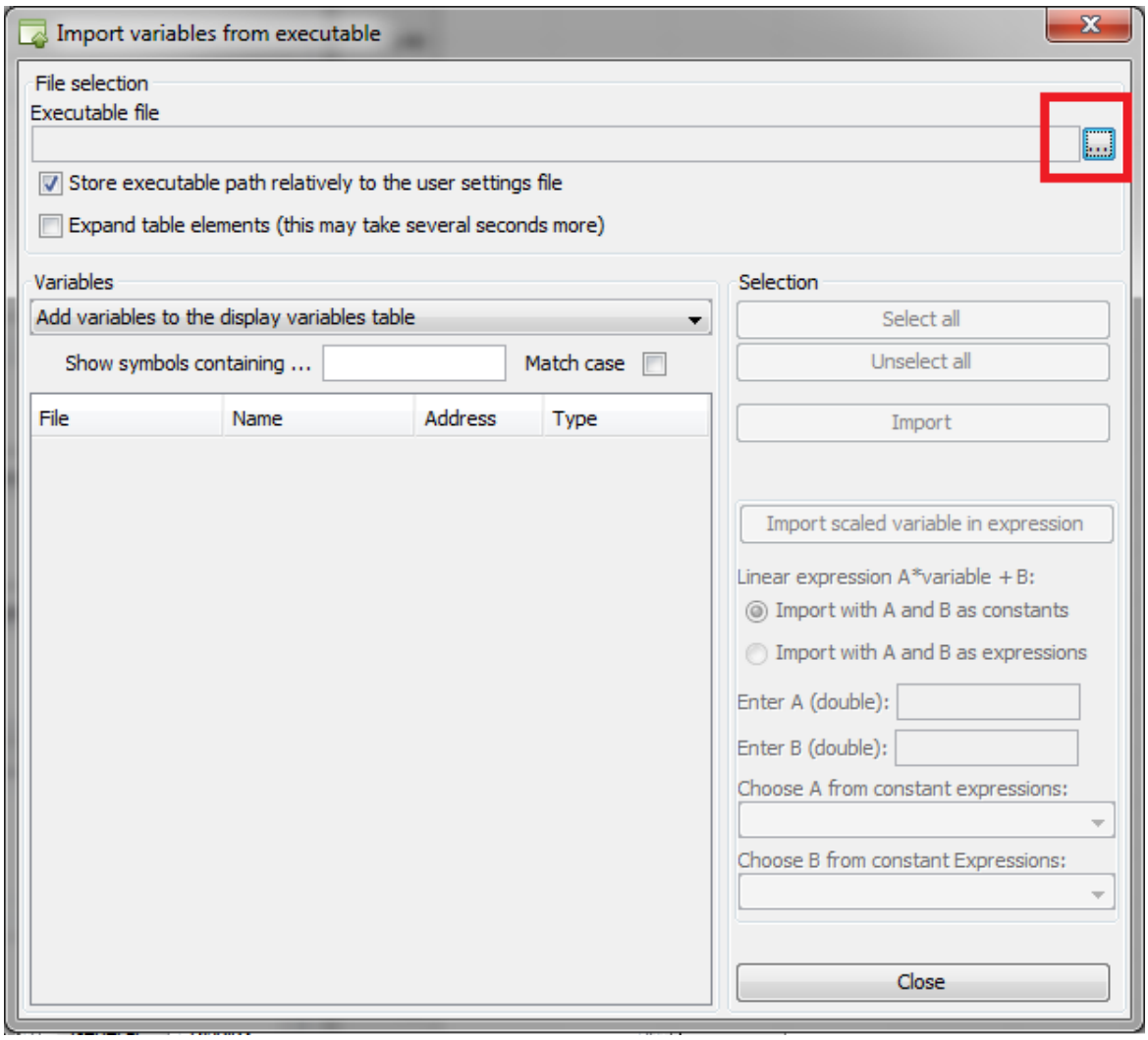

Рисунок 4.8 – Import variables

Далее откройте папку с вашим проектом. В данной папке выберите подпапку «MDK-ARM» и снова папку с вашим проектом, в ней есть файл «Analog_Sensor.axf».

В примере путь выглядит так:

/Analog_sensor/MDK-ARM/Analog_Sensor/ Analog_Sensor.axf;

Затем нажмите кнопку «Select executable file». В новом окне появится список всех переменных в микроконтроллере, как видно на рисунке 4.9.

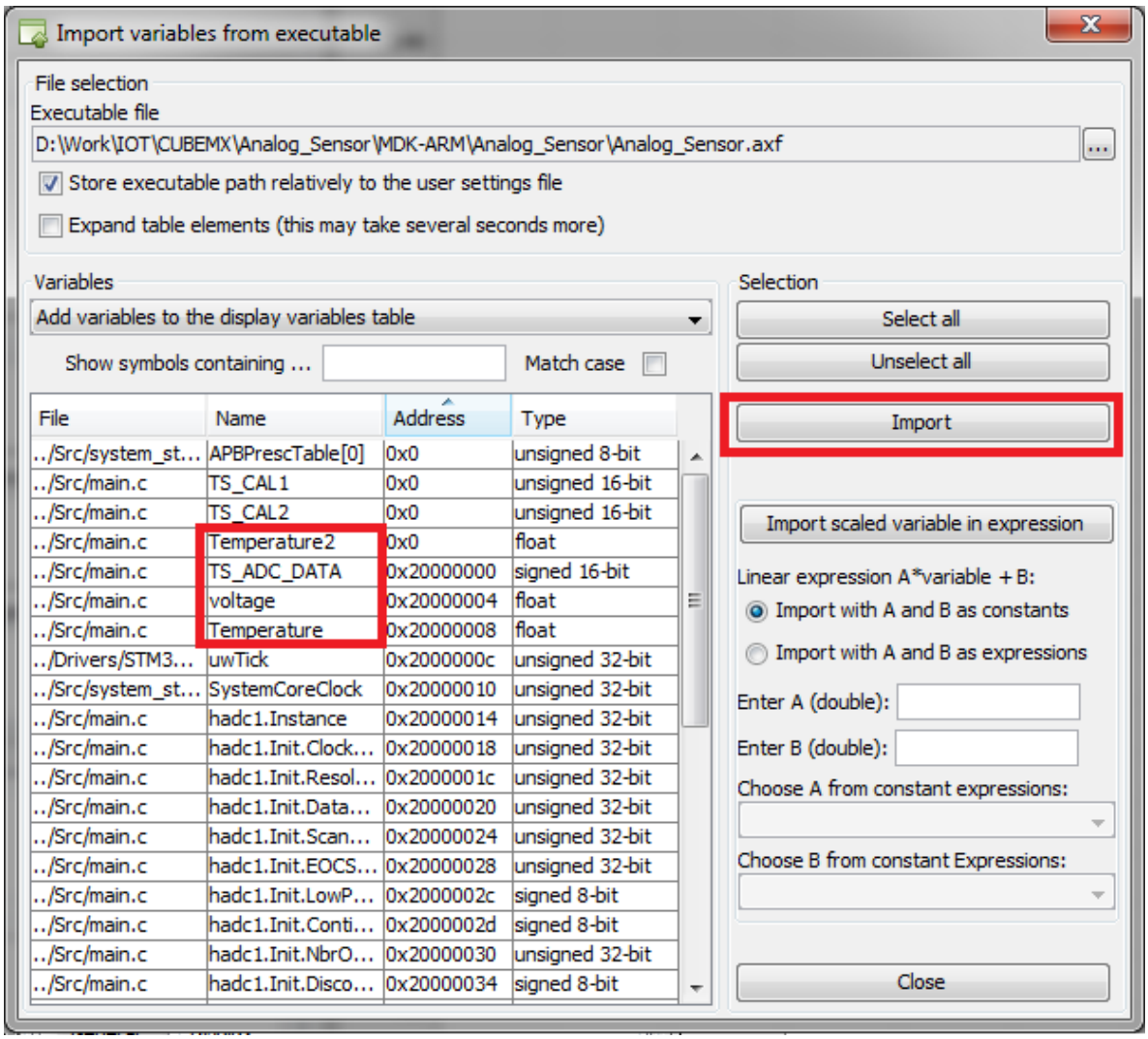

Рисунок 4.9 – Import variables

С зажатой клавишей Shift мышкой выделите переменные: Temperature, Temperature2, TS_ADC_DATA, voltage. Затем нажмите кнопку «Импорт», после этого нажмите кнопку «Close».

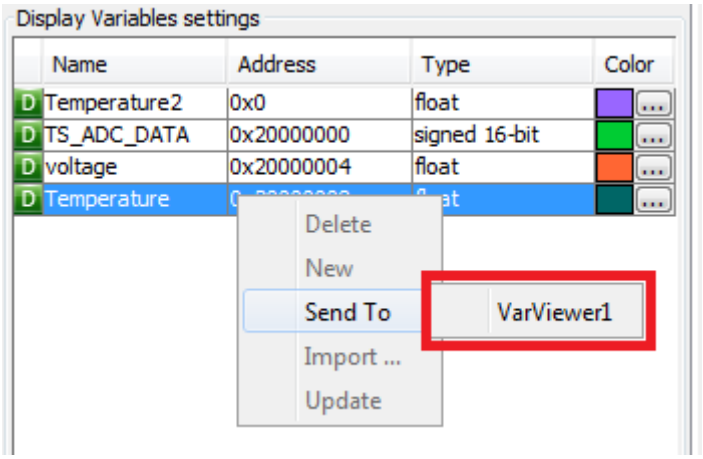

Рисунок 4.10 – Display Variables

Далее в окне «Display Variables», выберите нужную переменную, кликните правой кнопкой мыши и выберите «Send To» –> «VarViewer1». В окне «Viewers settings» вы увидите добавленные переменные (рисунок 4.10). Вы можете отображать данные как Curve, Bar Graph, Table. В этой работе выберите «Table».

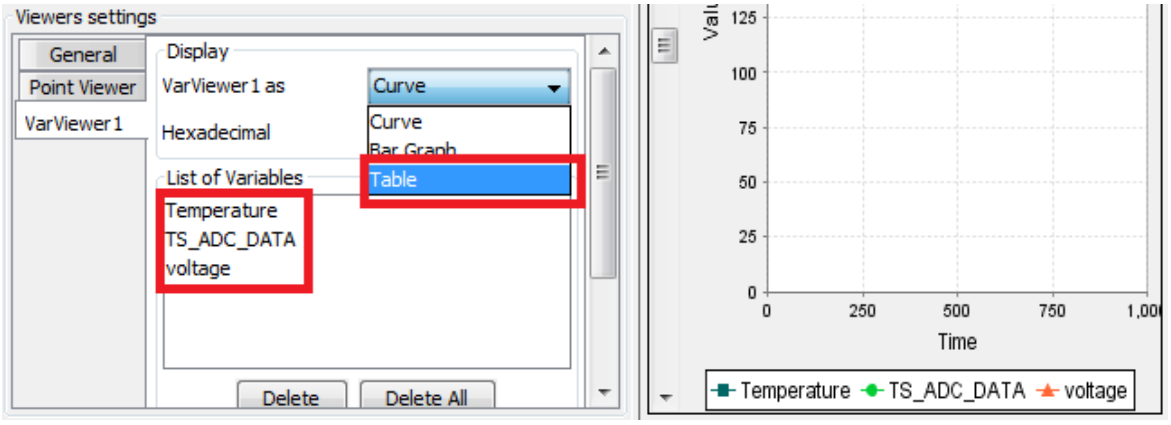

Рисунок 4.11 – Viewers settings

Затем на панели инструментов нажмите «Run» –> «Start S». Вы увидите значения температуры и других переменных в реальном времени. Запишите значение температуры в отчет. После этого нажмите «Run» –> «Stop S». Внимание! Нельзя одновременно смотреть значения переменных в STMStudio и выполнять загрузку кода в плату из Keil. Поэтому всегда нажимайте «Stop S» после просмотра переменных.

Откройте main.c вашего проекта в Keil, в цикле while (1) допишите строки кода:

TS_CAL1 = *TEMPSENSOR_CAL1_ADDR;
TS_CAL2 = *TEMPSENSOR_CAL2_ADDR.

Так вы добавили указатели (знак *) на область памяти, где лежат данные калибровки датчика температуры на заводе ST при 30°C и при 110°C. Далее самостоятельно запишите код расчета температуры по формуле 4.3. Сгенерируйте проект и загрузите его в плату. Запустите программу STMStudio и посмотрите показания температуры, которую вычислили по формуле 4.1 и

по формуле 4.3. Объясните полученный результат, запишите результаты в отчет. Какие показания более точны?

Контрольные вопросы

1) Опишите принцип преобразования аналогового сигнала в цифровой.
2) Какие виды датчиков вы знаете?
3) Где в первую очередь искать характеристики датчика или устройства?
4) Как производится настройка АЦП в программе CubeMX.
5) Как преобразовываются данные с АЦП в показания температуры.

Содержание отчета

1) Цель работы;
2) Подробное описание всех этапов проделанной работы;
3) Ответы на вопросы, представленные в тексте лабораторной работы;
4) Важные части вашего кода с пояснениями;
5) Графики и формулы, полученные в процессе выполнения лабораторной работы;
6) Анализ проделанной работы;
7) Выводы по данной лабораторной работе.

5 Лабораторная работа №5
«Работа с цифровыми датчиками. Барометр»

Цель работы: Изучить принцип считывания показаний цифровых датчиков микроконтроллером. Настроить периферию микроконтроллера для подключения датчика давления.

Задачи лабораторной работы:

1) Подключить датчик давления к микроконтроллеру.
2) Произвести настройку I2C по документации микроконтроллера.
3) Произвести калибровку датчика по документации LPS22HB.
4) Считать и проанализировать показания датчика.

Оборудование и программное обеспечение: плата B-L475E-IOT01A, кабель USB, среда разработки Keil Uvusuion 5, CubeMX.

Теоретический материал

Микроэлектромеханические системы (МЭМС) – устройства, объединяющие в себе микроэлектронные и микромеханические компоненты.

Механическим компонентом может быть примитивный инерциальный датчик и другие, способные определить характерные движения, которые пользователь проделывает со своим устройством.

МЭМС-устройства изготавливают на кремниевой подложке с помощью технологии микрообработки, аналогично технологии изготовления однокристальных интегральных микросхем. Типичные размеры микромеханических элементов лежат в диапазоне от 1 микрометра до 100 микрометров, тогда как размеры кристалла МЭМС-микросхемы имеют размеры от 20 микрометров до одного миллиметра[7].

LPS22HB – ультракомпактный пьезорезистивный датчик абсолютного давления, который функционирует как цифровой выходной барометр. Устройство содержит чувствительный элемент и интерфейс IC, который связывается через I2C или SPI с приложением.

Чувствительный элемент, который определяет абсолютное давление, состоит из подвесной мембраны, изготовленной с использованием специального процесса, разработанного ST. [5] Интерфейс I2C – это простой интерфейс с минимально возможным количеством линий связи. Интерфейс I2C использует всего две линии для связи ведущего устройства с ведомым:

- двунаправленная линия данных SDA;
- линия тактирования SCL, которая используется для синхронизации приема и передачи данных.

Есть ведущий(master) и ведомый (slave), такты генерирует master, ведомый лишь подтверждает прием байта. Всего на одной двухпроводной шине может быть до 127 устройств. Данные передаются последовательно по линии SDA.

Основной формат кадра обмена по I2C состоит из 11 бит. Сначала следует старт-бит, затем 8 бит данных, далее бит подтверждения приема, затем стоп-бит. Старт-бит определяется наличием спадающего фронта на лини SDA при высоком уровне сигнала на линии SCL. Эту комбинацию сигналов принято также называть стартовым состоянием, или состоянием «Старт». Стоп-бит (или состояние «Стоп») определяется наличием нарастающего фронта на линии SDA при высоком уровне на линии SCL. Состояния Старт и Стоп генерируются ведущим устройством, показано на рисунке 5.1 [8].

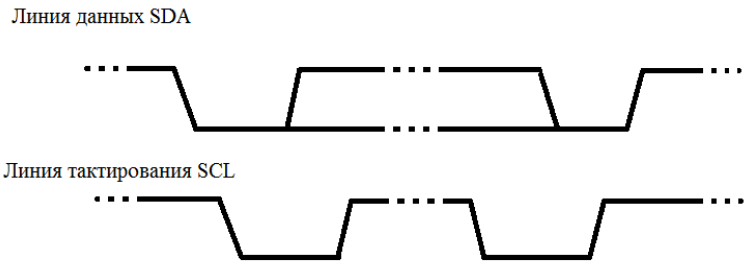

Рисунок 5.1 – I2C

Ход работы

1) Запустите программу CubeMX. Для этого на панели задач откройте меню «Пуск», «Все программы», «STMicroelectronics», «STM32Cube», «STM32CubeMX», «STM32CubeMX.exe».

Нажмите на кнопку «ACCESS TO MCU SELECTOR». Далее откроется окно для выбора микроконтроллера, в списке «MCUs list» выберите микроконтроллер STM32L475VGTx и нажмите на кнопку «Start Project».

Во вкладке «Connectivity» выберите «I2C2», затем в окне «Mode» выберите значение «I2C», значения во вкладке «Parameter Settings» установите, как на рисунке 5.2.

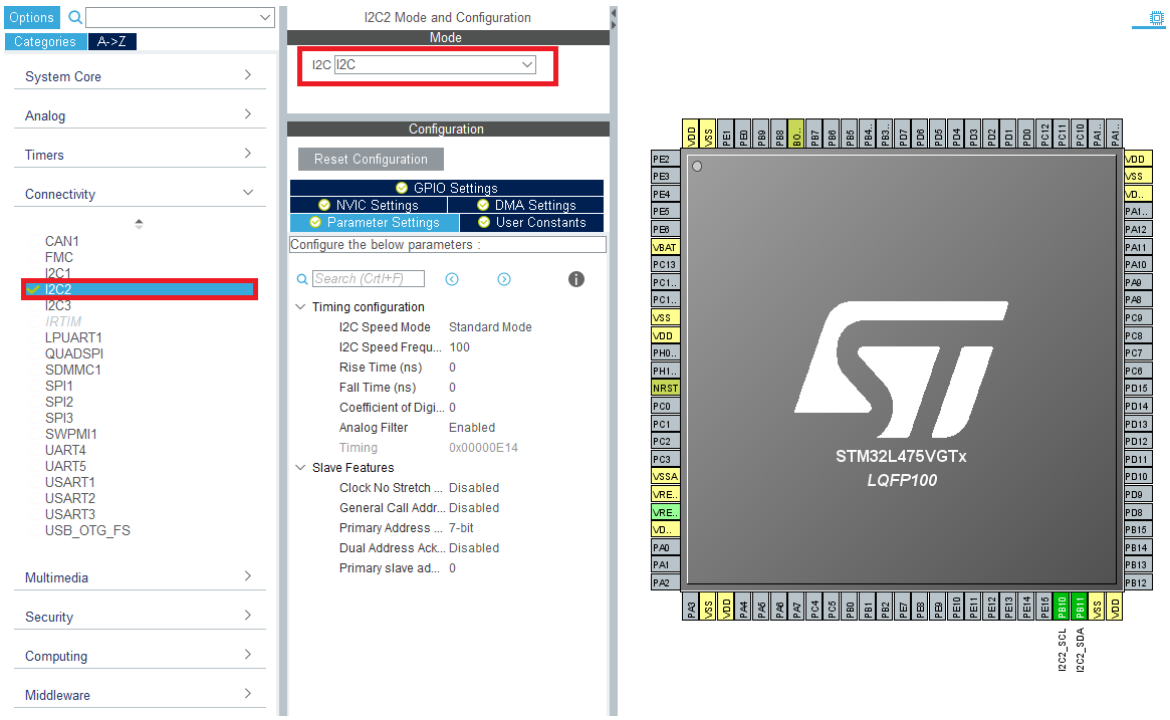

Рисунок 5.2 – I2C Mode

Перейдите во вкладку «Clock Configuration», если появилось всплывающее окно, представленное на рисунке 5.3, то нажмите «Yes»

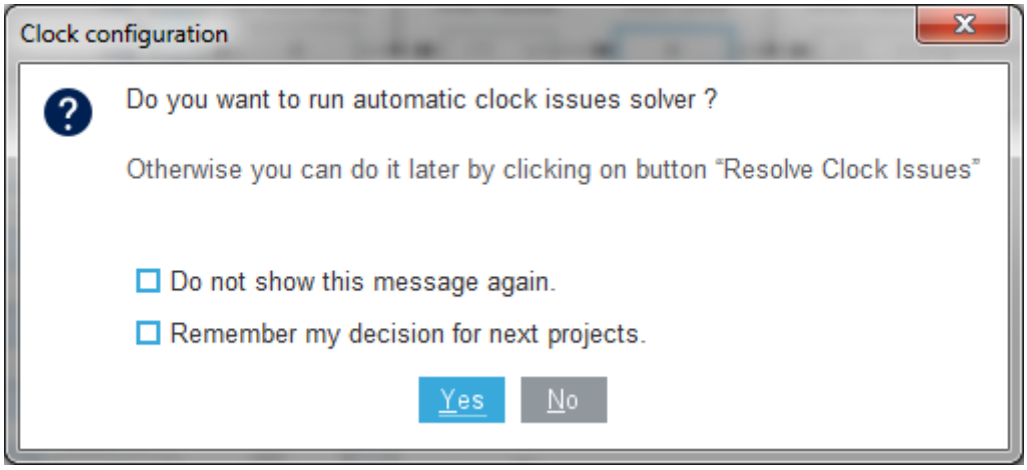

Рисунок 5.3 – Clock automatically Configuration

Настройте тактирование так же, как показано на рисунке 5.4.

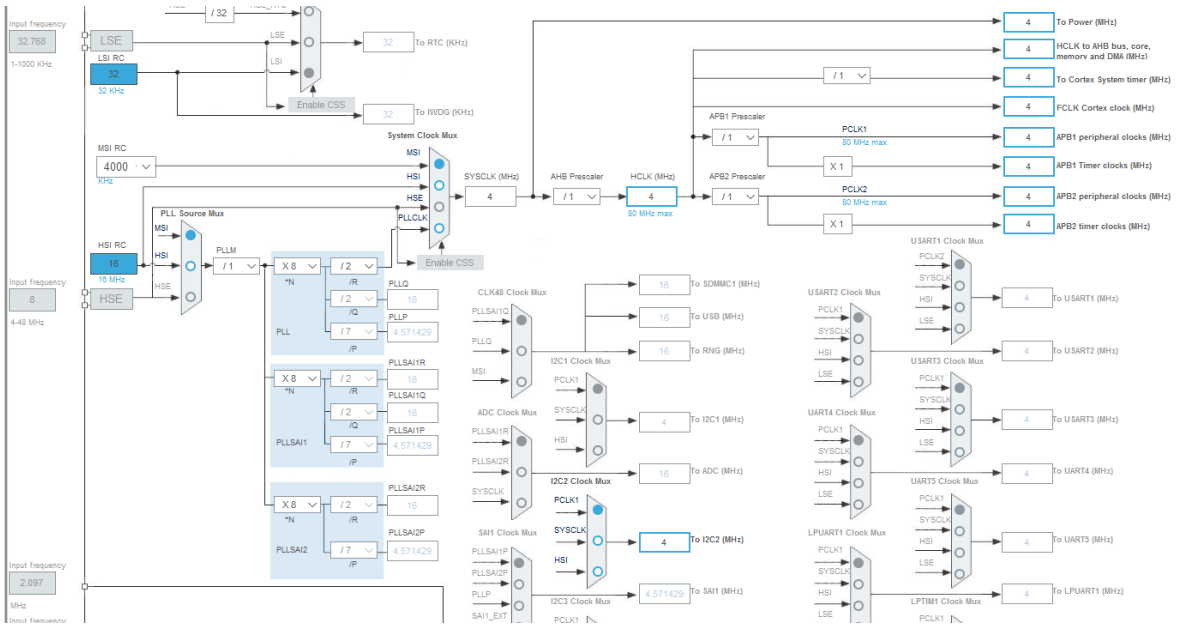

Рисунок 5.4 – Clock configuration

Далее откройте вкладку «Project Manager», вы увидите окно настройки вашего проекта. В поле «Project Name» введите название проекта «LPS22HB». В поле «Project Location» укажите путь для сохранения проекта на жестком диске вашего компьютера. В поле «Toolchain/IDE» выберите «MDK-ARM V5», так как для дальнейшей работы мы используем программу KEIL Uvision 5.

2) Если вы внесли все необходимые настройки в ваш проект, можете нажат кнопку «GENERATE CODE». Затем нажмите кнопку «Open Project».

Далее откроется программа KEIL Uvision 5, с вашим проектом «LPS22HB».

Передача и прием данных с датчика будет осуществляться по интерфейсу I2C. Для работы по этому интерфейсу в библиотеке HAL используются следующие функции:

*HAL_StatusTypeDef HAL_I2C_Master_Receive (I2C_HandleTypeDef *hi2c, uint16_t DevAddress,*
*uint8_t * pData, uint16_t Size, uint32_t Timeout)*

Parameters:
hi2c : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
DevAddress: Target device address
pData: Pointer to data buffer
Size: Amount of data to be sent
Timeout: Timeout duration
*HAL_StatusTypeDef HAL_I2C_Master_Transmit (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size, uint32_t Timeout)*

Parameters:

hi2c: Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

DevAddress: Target device address

pData: Pointer to data buffer

Size: Amount of data to be sent

Timeout: Timeout duration

Как видно из функции чтобы отправить данные по I2C необходимо указать шину I2C, адрес устройства на шине I2C, адрес регистра в который нужно писать, буфер с вашими данными, размер буфера (для работы с этим сенсором размер буфера = 1) и таймаут.

Для того чтобы считать данные, нужно выполнить функции сложнее. Так как микроконтроллер – это мастер. То нужно сначала отправить данные с адресом регистра, который вы хотите считать, а потом принять данные из этого регистра. Поэтому для удобства добавим эти операции в свои функции, чтобы не писать их каждый раз в дальнейшей работе.

Запишите в раздел /* USER CODE BEGIN 0 */ /* USER CODE END 0 */ функции для отправки данных и приема данных по шине I2C, как показано на рисунке 5.5. Запомните эти функции, они будут использоваться в следующих лабораторных работах.

```
/* USER CODE BEGIN 0 */
void write_register(uint16_t DevAddress, uint8_t register_pointer, uint8_t register_value)
{
    uint8_t data[2];
    data[0] = register_pointer;
    data[1] = register_value;
    HAL_I2C_Master_Transmit(&hi2c2, DevAddress, data, 2, 100);
}

void read_register(uint16_t DevAddress, uint8_t register_pointer, uint8_t* receive_buffer)
{
    HAL_I2C_Master_Transmit(&hi2c2, DevAddress, &register_pointer, 1, 100);
    HAL_I2C_Master_Receive(&hi2c2, DevAddress, receive_buffer, 1, 100);
}
/* USER CODE END 0 */
```

Рисунок 5.5 – Transmit and receive I2C functions

Далее добавьте инициализацию переменных, как на рисунке 5.6. Эти переменные понадобятся для работы с сенсором.

```
/* USER CODE BEGIN PV */
uint8_t WHO_I_AM;
int32_t PRESS,Pressure_millibar;
uint8_t buffer[3];
uint8_t i2c_receive_buf[1];
/* USER CODE END PV */
```

Рисунок 5.6 – Private variables

3) Теперь можно начать изучение датчика давления. Первое что нужно сделать при работе с датчиком по шине I2C – узнать адрес сенсора в документации.

Откройте документацию платы B-L475E-IOT01A [1 7.15 I2C addresses of modules used on MB1297] и в таблице 5.1 найдите адрес датчика LPS22HB.

Таблица 5.1 Устройства на шине I2C

Модули	Описание	SAD[6:0] + R/W	I2C write address	I2C read address
HTS221	Емкостный цифровой датчик для измерения относительной влажности и температуры	1011111x	0xBE	0xBF
LIS3MDL	3-х осевой магнитометр	0011110x	0x3C	0x3D
LPS22HB	МЭМС датчик давления	1011101x	0xBA	0xBB
LSM6DSL	3D акселерометр и 3D гироскоп	1101010x	0xD4	0xD5
VL53L0X	Датчик обнаружения жестов	0101001x	0x52	0x53
M24SR64-Y	Динамическая NFC / RFID метка	1010110x	0xAC	0xAD
STSAFE-A100	-	0100000x	0x40	0x41

Как видно в таблице 5.1 адрес сенсора написан в трех вариантах: в двоичном виде без младшего байта, в шестнадцатеричном виде если младший байт = 0, в шестнадцатеричном виде если младший байт = 1. Но функции HAL_I2C_Master_Receive и HAL_I2C_Master_Transmit сами изменяют младший байт на «1» при считывании, поэтому используйте только I2C write Address. Для датчика LPS22HB этот адрес будет равен «10111010» или в шестнадцатеричном виде «0xBA». Вы должны хорошо понимать перевод числа из одной системы счисления в другую, так как все числа нужно писать в шестнадцатеричном виде, а настраивать регистры в двоичном виде.

4) Откройте документацию датчика LPS22HB [5 Description,3.1-3.4,4.1-4.4] прочитайте описание и характеристики датчика. Затем нужно узнать значение регистра WHO_I_AM. WHO_I_AM – это регистр в котором сохранен идентификатор устройства. Это проверка подключен ли сенсор, и какой сенсор подключен по этому адресу. Откройте [5 - 9.4 WHO_AM_I (0Fh)]. Как видно адрес регистра WHO_AM_I равен 0Fh (для большинства сенсоров он равен 0Fh), а его значение 10110001b = B1h = 177d.

Теперь считаем данные с сенсора с адресом 0xBA, из регистра 0Fh, как показано на рисунке 5.7.

```
  /* USER CODE BEGIN 2 */
read_register(0xBA, 0x0F, i2c_receive_buf);
WHO_I_AM = i2c_receive_buf[0];
  /* USER CODE END 2 */
```

Рисунок 5.7 – WHO_AM_I

Затем нужно сгенерировать ваш код. Для этого нажмите «Rebuild all target files» в «Build Toolbar». Начнется генерация проекта и через несколько минут в окне «Build Output» будет написано «0 Errors 0 Warnings». Это означает что ошибок в проекте нет, и успешно создан файл в формате «.hex» для микроконтроллера.

Присоедините плату B-L475E-IOT01A к вашему компьютеру кабелем USB в разъем.

Затем нужно загрузить вашу программу в микроконтроллер. Для этого нажмите «Download code to flash memory» в Build Toolbar. Начнется загрузка проекта и через несколько секунд в окне Build Output будет написано «Programming Done». Это означает что программа успешно загружена.

Запустите программу STMStudio с проектом «LPS22HB» (описано в предыдущей лабораторной работе) и сделайте импорт переменной WHO_I_AM включите плату и запустите просмотр. Посмотрите значение переменной. Если оно равно «0xB1» в шестнадцатеричном виде, значит вы правильно запустили датчик и можете выполнять работу дальше, если нет – то вы допустили ошибку. Проверьте ваш код и исправьте ее.

5) Далее нужно проинициализировать сенсор. Это значит нужно настроить частоту работы, показания этого датчика и другие. Для этого существуют управляющие регистры. Изменяя их значения, вы изменяете работу датчика.

В датчике LPS22HB это регистры 10h, 11h, 12h. Откройте документацию LPS22HB и прочитайте значения всех битов регистра 10h [5 - 9.5 CTRL_REG1 (10h)]. После этого вы должны понять, что нужно настроить:

- Output data rate - 75 Hz;
- Low-pass filter enabled;
- Output registers not updated until MSB and LSB have been read;
- SPI Serial Interface Mode - Default value: 0 (We do not use SPI interface).

Поэтому значение регистра нужно написать 01011010 (5Ah), как описано в таблице 5.2.

Таблица 5.2 описание настройки регистра.

№	7	6	5	4	3	2	1	0
Name	0	ODR2	ODR1	ODR0	EN_LPFP	LPFP_ CFG	BDU	SIM
Need value	0	1	0	1	1	0	1	0

Далее изучите регистр CTRL_REG2 (11h) [5 - 9.6 CTRL_REG2 (11h)]. Его нужно настроить следующим образом:
- Reboot memory content – normal mode;
- FIFO disable;
- Stop on FIFO watermark – disable;
- Register address automatically incremented during a multiple byte access with a
- serial interface – disable;
- I2C enabled;
- Software reset – normal mode;
- One-shot enable – idle mode.

Далее изучите регистр CTRL_REG3 (12h) [5 - 9.7 CTRL_REG3 (12h)]. Его нужно настроить следующим образом:
All bytes – Default value.

В результате вы получите код инициализации датчика, как показано на рисунке 5.8.

```
  /* USER CODE BEGIN 2 */
read_register(0xBA, 0x0F, i2c_receive_buf);
WHO_I_AM = i2c_receive_buf[0];
//initialisation
write_register(0xBA, 0x10, 0x5A); // (5Ah = 1011010b)
write_register(0xBA, 0x11, 0x00); // (00h = 0000000b)
write_register(0xBA, 0x12, 0x00); // (00h = 0000000b)
  /* USER CODE END 2 */
```

Рисунок 5.8 – Инициализация

6) Если вы правильно настроили инициализацию, то датчик должен исправно работать и показывать данные об атмосферном давлении. Для того чтобы узнать, как считать данные о давлении с датчика откройте документацию LPS22HB и прочтите пункты 9.18-9.20. Информация о давлении это 32-битное число, которое разделено на 3 8-битных числа, которые находятся в регистрах PRESS_OUT_XL (28h), PRESS_OUT_L (29h), PRESS_OUT_H (2Ah). Нужно по очереди считать показания трех регистров, а затем соединить их в одно число.

Таблица 5.3 Структура данных

PRESS_OUT_H (2Ah)	PRESS_OUT_L (29h)	PRESS_OUT_XL (28h)
00010110 = 16h	00011010 = 1Ah	10111010 = BAh
PRESS_OUT = 00010110 00011010 10111010 = 161ABAh		

Аналогично тому как вы считали значения регистра WHO_AM_I, считайте по очереди значения регистров PRESS_OUT_XL (28h), PRESS_OUT_L (29h), PRESS_OUT_H (2Ah). И добавьте их в массив buffer, как показано на рисунке 5.9.

```
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    //// Barometer_READ
read_register(0xBA, 0x28, i2c_receive_buf);
buffer[0] = i2c_receive_buf[0];
read_register Your code ;
buffer[1] = i2c_receive_buf[0];
read_register Your code ;
buffer[2] = i2c_receive_buf[0];
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}
```

Рисунок 5.9 – Barometer read

Далее нужно собрать три 3 8-битных числа из массива buffer в одно 32-битное число. Для этого нужно сложить buffer [0] со сдвинутым на 8 бит buffer [1] и со сдвинутым на 16 бит buffer [2] (как объяснено в таблице 5.3).

Чтобы перевести это значение из LSB в миллибары нужно поделить это значение на Scaling Factor.

Pressure Value (LSB) = PRESS_OUT_H (2Ah) & PRESS_OUT_L (29h) & PRESS_OUT_XL (28h);

Press Value (hPa) = (Pressure value (LSB))/ Scaling Factor.

Это видно на рисунке 5.10.

Скомпилируйте полученный код, загрузите его в плату. Затем запустите программу STMStudio, импортируйте новые переменные для просмотра (PRESS, Pressure_millibar). Оцените полученный результат.

7) Затем добавьте код преобразования результатов сенсора (Pressure_millibar) в миллиметры ртутного столба. И выведите значение в STMStudio.

8) Запишите его в отчет. Найдите значение нормального атмосферного давления и сравните его с полученным вами. На сколько отличается значение? Что это означает? Как атмосферное давление изменяется от погоды? Как атмосферное давление изменяется от высоты?

```
  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    //// Barometer_READ
read_register(0xBA, 0x28, i2c_receive_buf);
buffer[0] = i2c_receive_buf[0];
read_register(Your code);
buffer[1] = i2c_receive_buf[0];
read_register(Your code);
buffer[2] = i2c_receive_buf[0];
PRESS = ((((uint32_t)buffer[2]) << 16)+(((uint32_t)buffer[1]) << 8) + buffer[0]);
Pressure_millibar = PRESS/4096;
HAL_Delay(2000);
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}
```

Рисунок 5.10 – Чтение данных барометра

Контрольные вопросы

1) Как работают МЭМС датчики?
2) Какие виды МЭМС датчиков вы знаете?
3) Как устроен датчик давления?
4) Как работает интерфейс I2C?
5) Из каких этапов состоит считывание данных с датчика?
6) Как производится инициализация датчика давления?

Содержание отчета

1) Цель работы;
2) Подробное описание всех этапов проделанной работы;
3) Ответы на вопросы, представленные в тексте лабораторной работы;
4) Важные части вашего кода с пояснениями;
5) Графики и формулы, полученные в процессе выполнения лабораторной работы;
6) Анализ проделанной работы;
7) Выводы по данной лабораторной работе.

6 Лабораторная работа №6
«Работа с цифровыми датчиками. Цифровой датчик температуры»

Цель работы: Подключить к микроконтроллеру цифровой датчик температуры. Произвести инициализацию и калибровку датчика. Считать данные и рассчитать значение температуры, измеренное датчиком.

Задачи лабораторной работы:

1) Подключить датчик температуры к микроконтроллеру.
2) Произвести настройку I2C по документации микроконтроллера.
3) Произвести калибровку датчика по документации HTS221.
4) Считать и проанализировать показания датчика.

Оборудование и программное обеспечение: плата B-L475E-IOT01A, кабель USB, среда разработки Keil Uvusuion 5, CubeMX, STMStudio.

Теоретический материал

HTS221 – это цифровой датчик влажности и температуры. Устройство включает в себя чувствительный элемент и интерфейс IC (интегральную схему), способный принимать информацию от чувствительного элемента и предоставлять цифровой сигнал приложению, связываясь через интерфейсы I²C / SPI.

Полная цепочка измерений состоит из малошумящего емкостного усилителя, который преобразует емкостный дисбаланс датчика влажности в аналоговый сигнал напряжения, а аналого-цифровой преобразователь преобразовывает в цифровую.

Преобразователь соединен со специальным аппаратным (HW) фильтром усреднения для удаления высокочастотного компонента и уменьшения трафика по последовательному интерфейсу.

Интерфейс IC (интегральная схема) откалиброван на заводе-изготовителе, и коэффициенты, необходимые для преобразования 16-битных значений АЦП градусы Цельсия, могут быть считаны через внутренние регистры датчика. Дальнейшая калибровка пользователем не требуется.

Блок-схема датчика HTS221 показана на рисунке 6.1.

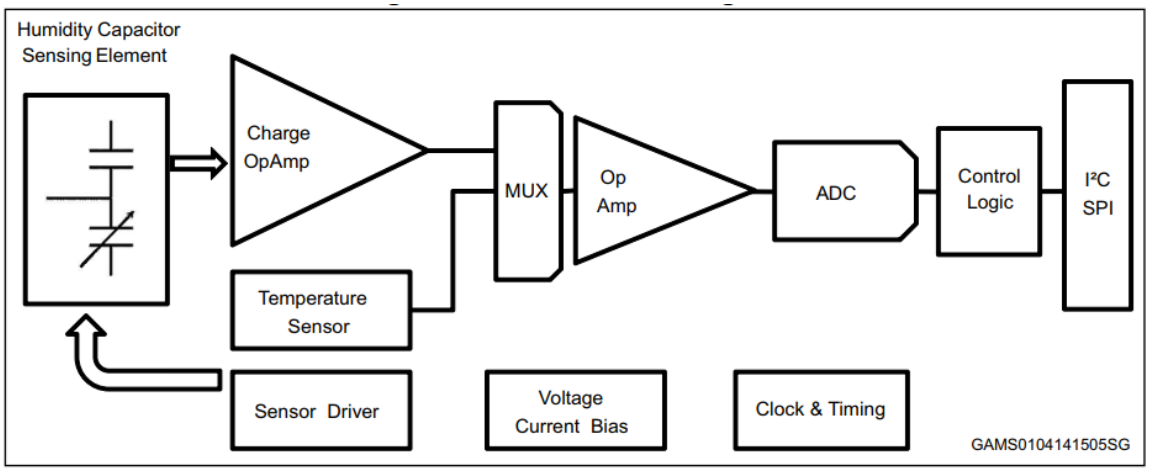

Рисунок 6.1 – Блок-диаграмма HTS221[6]

Датчик влажности HTS221 сохраняет значение температуры в необработанных значениях в двух 8-битных регистрах:

- TEMP_OUT_H (0x2B);
- TEMP_OUT_L (0x2A).

2 байта объединяются в 16-битное слово.

Самый старший бит (MSB) регистра TEMP_OUT_H указывает полярность:

- Если бит знака равен нулю, то считанное значение является положительным;
- Если бит знака равен единице, то считываемое значение является отрицательным: в этом случае мы берем два дополнения к полному слову.

Значение температуры T должно быть рассчитано путем линейной интерполяции текущих регистров (TEMP_OUT_H & TEMP_OUT_L) с калибровочными регистрами.

Ход работы

1) Запустите программу CubeMX. Для этого на панели задач откройте меню «Пуск», «Все программы», «STMicroelectronics», «STM32Cube», «STM32CubeMX», «STM32CubeMX.exe».

Нажмите на кнопку «ACCESS TO MCU SELECTOR». Далее откроется окно для выбора микроконтроллера, в списке «MCUs list» выберите микроконтроллер STM32L475VGTx и нажмите на кнопку «Start Project».

В вкладке «Connectivity» выберите «I2C2», затем в окне «Mode» выберите значение «I2C», значения во вкладке «Parameter Settings» установите, как на рисунке 6.2.

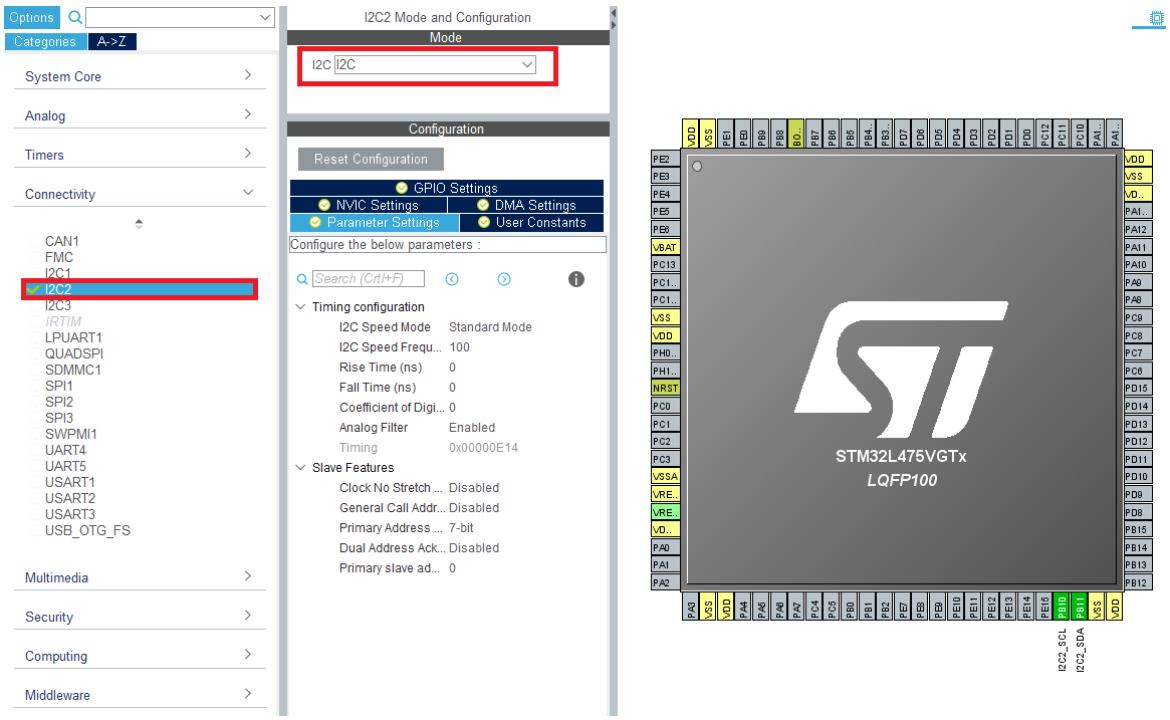

Рисунок 6.2 – I2C Mode

Настройте тактирование так же, как показано на рисунке 6.3.

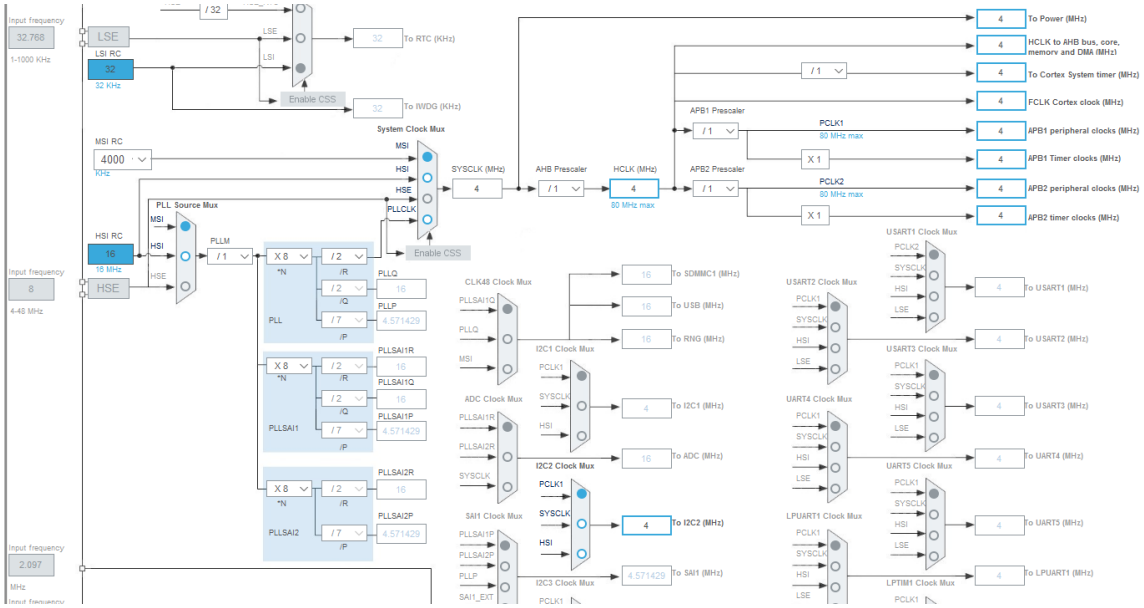

Рисунок 6.3 – Clock configuration

Далее откройте вкладку «Project Manager», вы увидите окно настройки вашего проекта. В поле «Project Name» введите название проекта «HTS221_Temperature». В поле «Project Location» укажите путь для сохранения проекта на жестком диске вашего компьютера. В поле «Toolchain/IDE» выберите «MDK-ARM V5», так как для дальнейшей работы мы используем программу KEIL Uvision 5.

Если вы внесли все необходимые настройки в ваш проект, можете нажать кнопку «GENERATE CODE». Затем нажмите кнопку «Open Project».

2) Далее откроется программа KEIL Uvision 5, с вашим проектом «HTS221_Temperature».

Запишите в раздел /* USER CODE BEGIN 0 */ /* USER CODE END 0 */ функции для отправки данных и приема данных по шине I2C, как показано на рисунке 6.4.

```
/* USER CODE BEGIN 0 */
void write_register(uint16_t DevAddress, uint8_t register_pointer, uint8_t register_value)
{
    uint8_t data[2];
    data[0] = register_pointer;
    data[1] = register_value;
    HAL_I2C_Master_Transmit(&hi2c2, DevAddress, data, 2, 100);
}

void read_register(uint16_t DevAddress, uint8_t register_pointer, uint8_t* receive_buffer)
{
    HAL_I2C_Master_Transmit(&hi2c2, DevAddress, &register_pointer, 1, 100);
    HAL_I2C_Master_Receive(&hi2c2, DevAddress, receive_buffer, 1, 100);
}
/* USER CODE END 0 */
```

Рисунок 6.4 – Transmit and receive I2C functions

3) Далее добавьте инициализацию переменных, как на рисунке 6.5. Эти переменные понадобятся для работы с сенсором.

```
/* USER CODE BEGIN PV */
uint8_t i2c_receive_buf[2], buffer[4], tmp;
int16_t T0_degC, T1_degC, T0_out, T1_out, T_out, T0_degC_x8_u16, T1_degC_x8_u16;
float Temperature;
/* USER CODE END PV */
```

Рисунок 6.5 – Private variables

Откройте документацию платы B-L475E-IOT01A [1 7.15 I2C addresses of modules used on MB1297] и найдите адрес датчика HTS221.

Для датчика HTS221 этот адрес будет равен «10111110» или в шестнадцатеричном виде «0xBE».

4) Откройте документацию датчика HTS221 [6 - 7.1 WHO_AM_I] и определите адрес и значение регистра WHO_AM_I.

Теперь считайте данные с сенсора HTS221, из регистра WHO_AM_I. Вы делали это аналогично в предыдущей работе. Сгенерируйте проект, подключите плату и загрузите в нее проект. Затем запустите STMStudio и посмотрите значение регистра WHO_AM_I аналогично предыдущей работе.

5) Затем нужно проинициализировать сенсор. Для этого нужно открыть [6 - 7.3 CTRL_REG1 (20h)]. В нем нужно настроить 4 байта:

- PD: power-down control – active mode;
- BDU: block data update – output registers not updated until MSB and LSB reading;
- ODR1, ODR0: output data rate selection – 12.5 Hz.

Таблица 6.1 CTRL_REG1

7	6	5	4	3	2	1	0
PD	Reserved				BDU	ODR1	ODR0

Запишите в устройство с адресом «0xBE», в регистр «0x20», значение «0x87 (10000111)».

Далее считайте показания датчика ADC_OUT (LSB) и преобразуйте в °C. Для этого прочитайте пункт 8 документации сенсора [6 - 8 Humidity and temperature data conversion]:

6) Считайте значение коэффициентов T0_degC_x8 и T1_degC_x8 из регистров 0x32 & 0x33.

```
read_register(0xBE, 0x32, i2c_receive_buf);
    buffer[0]=i2c_receive_buf[0];
read_register(0xBE, 0x33, i2c_receive_buf);
    buffer[1]=i2c_receive_buf[0];
```

7) Считайте MSB биты T1_degC и T0_degC из регистров 0x35 для вычисления T0_DegC and T1_DegC.

```
read_register(0xBE, 0x35, i2c_receive_buf);
    tmp=i2c_receive_buf[0];
```

8) Вычислите T0_degC и T1_degC

```
T0_degC_x8_u16 = (((uint16_t)(tmp & 0x03)) << 8) | ((uint16_t)buffer[0]);
T1_degC_x8_u16 = (((uint16_t)(tmp & 0x0C)) << 6) | ((uint16_t)buffer[1]);
    T0_degC = T0_degC_x8_u16 >> 3;
    T1_degC = T1_degC_x8_u16 >> 3;
```

9) Считайте из регистров 0x3C & 0x3D значение T0_OUT.

```
read_register(0xBE, 0x3C, i2c_receive_buf);
    buffer[0]=i2c_receive_buf[0];
read_register(0xBE, 0x3D, i2c_receive_buf);
    buffer[1]=i2c_receive_buf[0];
```

10) Считайте значение T1_OUT из регистров 0x3E & 0x3F.

```
read_register(0xBE, 0x3E, i2c_receive_buf);
    buffer[2]=i2c_receive_buf[0];
read_register(0xBE, 0x3F, i2c_receive_buf);
    buffer[3]=i2c_receive_buf[0];
```

11) Переведите 8-bit данные в 16-bit T0_out и T1_out

```
T0_out = (((uint16_t)buffer[1]) << 8) | (uint16_t)buffer[0];
T1_out = (((uint16_t)buffer[3]) << 8) | (uint16_t)buffer[2];
```

Общий код показан на рисунке6.6.

```
  /* USER CODE BEGIN 2 */
//Initialization
write_register(0xBE, 0x20, 0x87);

read_register(0xBE, 0x32, i2c_receive_buf);
buffer[0]=i2c_receive_buf[0];
read_register(0xBE, 0x33, i2c_receive_buf);
buffer[1]=i2c_receive_buf[0];
read_register(0xBE, 0x35, i2c_receive_buf);
tmp=i2c_receive_buf[0];
T0_degC_x8_u16 = (((uint16_t)(tmp & 0x03)) << 8) | ((uint16_t)buffer[0]);
T1_degC_x8_u16 = (((uint16_t)(tmp & 0x0C)) << 6) | ((uint16_t)buffer[1]);
T0_degC = T0_degC_x8_u16 >> 3;
T1_degC = T1_degC_x8_u16 >> 3;
read_register(0xBE, 0x3C, i2c_receive_buf);
buffer[0]=i2c_receive_buf[0];
read_register(0xBE, 0x3D, i2c_receive_buf);
buffer[1]=i2c_receive_buf[0];
read_register(0xBE, 0x3E, i2c_receive_buf);
buffer[2]=i2c_receive_buf[0];
read_register(0xBE, 0x3F, i2c_receive_buf);
buffer[3]=i2c_receive_buf[0];
T0_out = (((uint16_t)buffer[1]) << 8) | (uint16_t)buffer[0];
T1_out = (((uint16_t)buffer[3]) << 8) | (uint16_t)buffer[2];
  /* USER CODE END 2 */
```

Рисунок 6.6 – Инициализация датчика

12) Далее в цикле, while(1) считайте T_OUT (ADC_OUT) из регистров 0x2A & 0x2B.

13) Аналогично T0_out преобразовать 8-битные значения (MSB и LSB) в один 16-битный T_OUT

```
  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
HAL_Delay(1000);
read_register(0xBE, 0x2A, i2c_receive_buf);
buffer[0]=i2c_receive_buf[0];
read_register(0xBE, 0x2B, i2c_receive_buf);
buffer[1]=i2c_receive_buf[0];
T_out = Your code;

    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}
```

Рисунок 6.7 – Считывание показаний

14) Вычислите значение T [degC] путем линейной интерполяции, применяя следующую формулу:

$$T[\deg C] = \frac{(T1\_\deg C - T0\_\deg C)(T\_OUT - T0\_OUT)}{T1\_OUT - T0\_OUT} + T0\_\deg C \quad (6.1)$$

Когда рассчитываете формулу приведите ее к типу float

Temperature = (float) Your formula;

15) Сгенерируйте проект, загрузите его в плату. Включите STMStudio, добавьте новые переменные из проекта и посмотрите на показания температуры и переменных. Запишите все показания в отчет. Опишите принцип работы датчика, его характеристики. Для чего температура рассчитывается таким образом? Как обеспечивается калибровка?

Контрольные вопросы

1) Как работает датчик HTS221?
2) Как производится калибровка датчика температуры?
3) Как работает интерфейс обмена между микроконтроллером и датчиком по I2C?
4) Из каких этапов состоит считывание данных с датчика?
5) Как производится инициализация датчика?

Содержание отчета

1) Цель работы;
2) Подробное описание всех этапов проделанной работы;
3) Ответы на вопросы, представленные в тексте лабораторной работы;
4) Важные части вашего кода с пояснениями;
5) Графики и формулы, полученные в процессе выполнения лабораторной работы;
6) Анализ проделанной работы;
7) Выводы по данной лабораторной работе.

7 Лабораторная работа №7
«Работа с цифровыми датчиками. Цифровой датчик влажности»

Цель работы: Подключить к микроконтроллеру цифровой датчик влажности. Произвести инициализацию и калибровку датчика. Считать данные и рассчитать значение влажности, измеренное датчиком.

Задачи лабораторной работы:

1) Подключить датчик влажности к микроконтроллеру.
2) Произвести настройку I2C по документации микроконтроллера.
3) Произвести калибровку датчика по документации HTS221.
4) Считать и проанализировать показания датчика.

Оборудование и программное обеспечение: плата B-L475E-IOT01A, кабель USB, среда разработки Keil Uvusuion 5, CubeMX, STMStudio.

Теоретический материал

Относительная влажность (RH) – это отношение парциального давления водяного пара к равновесному давлению водяного пара при данной температуре. Относительная влажность зависит от температуры и давления интересующей системы. Такое же количество водяного пара приводит к более высокой относительной влажности холодного воздуха, чем теплого воздуха.

HTS221 – это цифровой датчик влажности и температуры. Устройство включает в себя чувствительный элемент и интерфейс IC (интегральной схемы), способный принимать информацию от чувствительного элемента и предоставлять цифровой сигнал приложению, связываясь через интерфейсы I²C / SPI с хост-контроллером.

Блок-схема датчика HTS221 показана на рисунке 7.1.

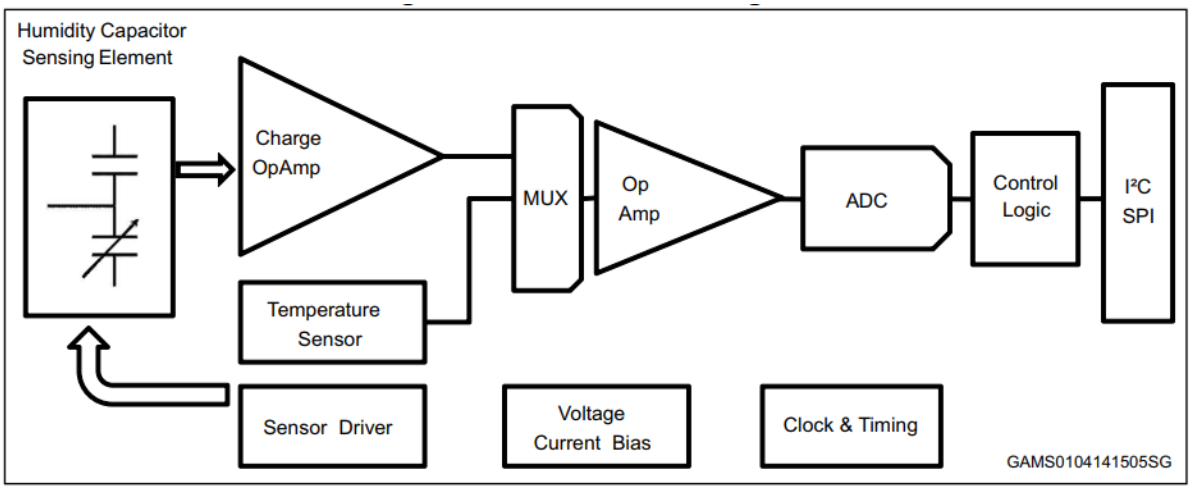

Рисунок 7.1 – Блок-диаграмма HTS221[6]

Датчик влажности HTS221 хранит значение влажности в необработанных значениях в двух 8-битных регистрах;

HUMIDITY_OUT_H (0x29);

HUMIDITY_OUT_L (0x28).

2 байта объединяются в 16-битное слово.

Значение относительной влажности RH следует рассчитывать путем линейной интерполяции регистров (HUMIDITY_OUT_H & HUMIDITY_OUT_L) с регистрами калибровки;

Устройство откалибровано на заводе-изготовителе, и коэффициенты, необходимые для преобразования 16-разрядных значений АЦП в% относительной влажности, могут быть считаны из регистров внутреннего датчика.

Ход работы

1) Запустите программу CubeMX. Для этого на панели задач откройте меню «Пуск», «Все программы», «STMicroelectronics», «STM32Cube», «STM32CubeMX», «STM32CubeMX.exe».

Нажмите на кнопку «ACCESS TO MCU SELECTOR». Далее откроется окно для выбора микроконтроллера, в списке «MCUs list» выберите микроконтроллер STM32L475VGTx и нажмите на кнопку «Start Project».

В вкладке «Connectivity» выберите «I2C2», затем в окне «Mode» выберите значение «I2C», значения во вкладке «Parameter Settings» установите, как на рисунке 7.3.

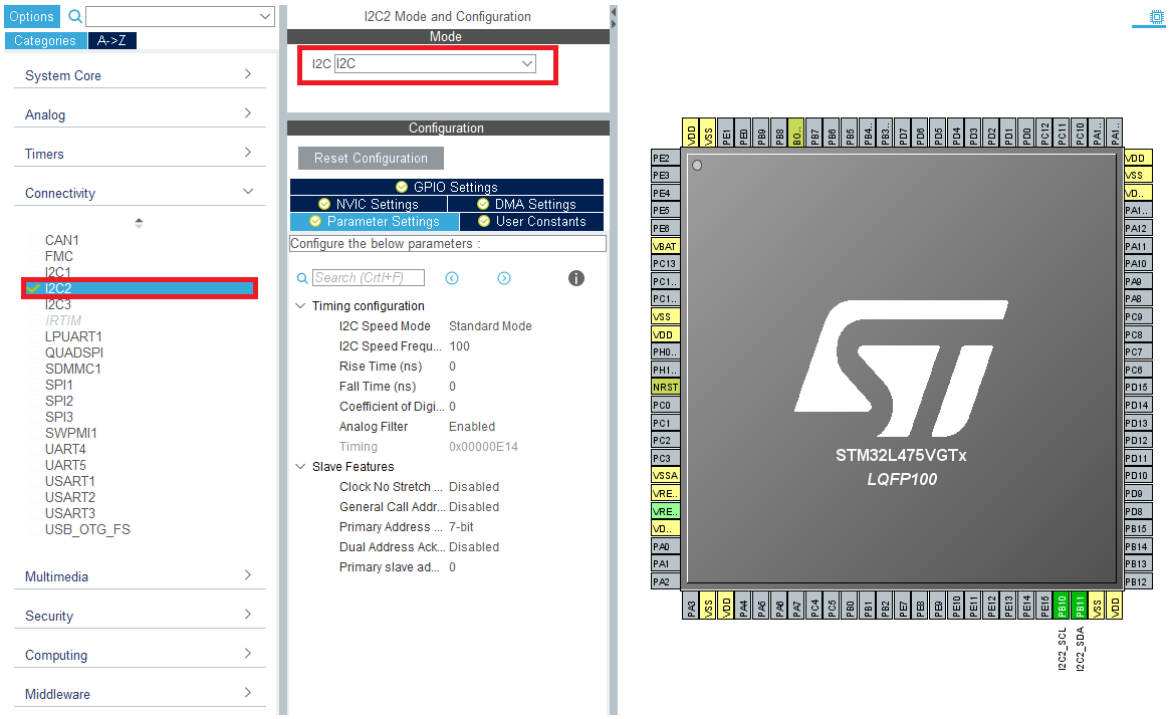

Рисунок 7.3 – I2C Mode

Настройте тактирование так же, как показано на рисунке 7.4.

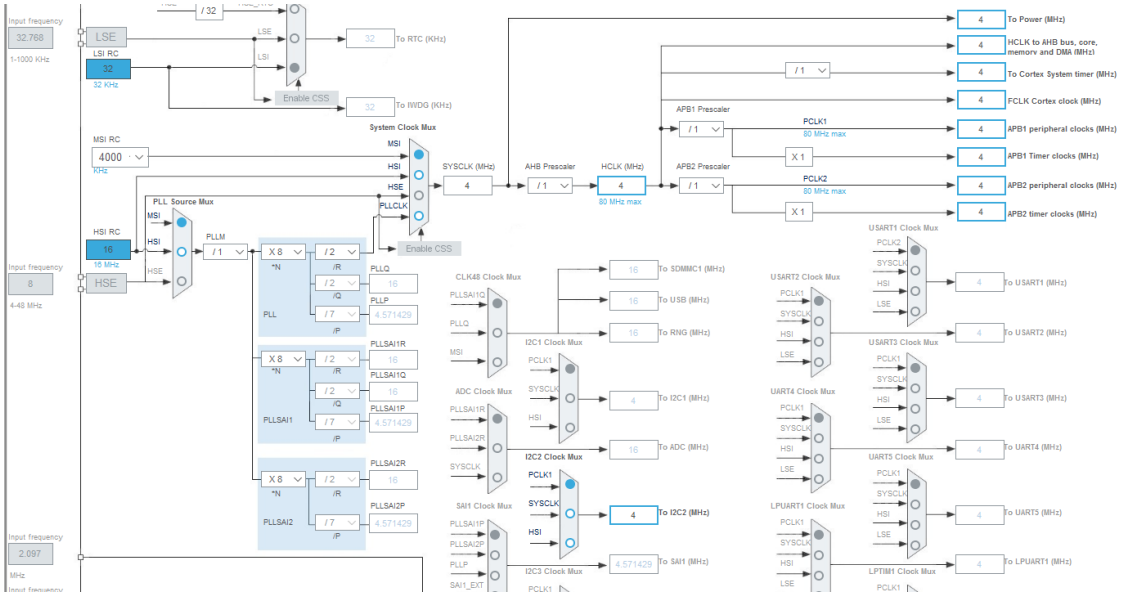

Рисунок 7.4 – Clock configuration

Далее откройте вкладку «Project Manager», вы увидите окно настройки вашего проекта. В поле «Project Name» введите название проекта «HTS221_ Humidity». В поле «Project Location» укажите путь для сохранения проекта на жестком диске вашего компьютера. В поле «Toolchain/IDE» выберите «MDK-ARM V5», так как для дальнейшей работы мы используем программу KEIL Uvision 5.

2) Если вы внесли все необходимые настройки в ваш проект, можете нажат кнопку «GENERATE CODE». Затем нажмите кнопку «Open Project».

Далее откроется программа KEIL Uvision 5, с вашим проектом «HTS221_ Humidity».

Запишите в раздел /* USER CODE BEGIN 0 */ /* USER CODE END 0 */ функции для отправки данных и приема данных по шине I2C, как показано на рисунке 7.5.

```
/* USER CODE BEGIN 0 */
void write_register(uint16_t DevAddress, uint8_t register_pointer, uint8_t register_value)
{
    uint8_t data[2];
    data[0] = register_pointer;
    data[1] = register_value;
    HAL_I2C_Master_Transmit(&hi2c2, DevAddress, data, 2, 100);
}

void read_register(uint16_t DevAddress, uint8_t register_pointer, uint8_t* receive_buffer)
{
    HAL_I2C_Master_Transmit(&hi2c2, DevAddress, &register_pointer, 1, 100);
    HAL_I2C_Master_Receive(&hi2c2, DevAddress, receive_buffer, 1, 100);
}
/* USER CODE END 0 */
```

Рисунок 7.5 – Transmit and receive I2C functions

Далее добавьте инициализацию переменных, как на рисунке 7.6. Эти переменные понадобятся для работы с сенсором.

```
/* USER CODE BEGIN PV */
uint8_t i2c_receive_buf[1], buffer[2];
int16_t H0_T0_out, H1_T0_out, H_T_out, H0_rh, H1_rh;
float humidity;
/* USER CODE END PV */
```

Рисунок 7.6 – Private variables

Откройте документацию платы B-L475E-IOT01A [1 - 7.15 I2C addresses of modules used on MB1297] и в таблице 5.1 найдите адрес датчика HTS221.

Для датчика HTS221 этот адрес будет равен «10111110» или в шестнадцатеричном виде «0xBE».

3) Откройте документацию датчика HTS221 [6 - 7.1 WHO_AM_I] и определите адрес и значение регистра WHO_AM_I.

Теперь считайте данные с сенсора HTS221, из регистра WHO_AM_I. Вы делали это аналогично в предыдущей работе. Сгенерируйте проект, подключите плату и загрузите в нее проект. Затем запустите «STMStudio» и посмотрите значение регистра WHO_AM_I аналогично предыдущей работе.

4) Затем нужно проинициализировать сенсор. Для этого нужно открыть [6 - 7.3 CTRL_REG1 (20h)]. В нем нужно настроить 4 байта:

- PD: power-down control – active mode;
- BDU: block data update – output registers not updated until MSB and LSB reading;
- ODR1, ODR0: output data rate selection – 12.5 Hz.

Запишите в устройство с адресом «0xBE», в регистр «0x20», нужное значение.

Прочитайте пункт 8 документации сенсора [6 - 8 Humidity and temperature data conversion].

Далее приступим к считыванию показаний сенсора (ADC_OUT) и переводу в RH %:

5) Считайте значение коэффициентов H0_rH_x2 и H1_rH_x2 из регистров 0x30 и 0x31.

```
read_register(0xBE, 0x30, i2c_receive_buf);
buffer[0]=i2c_receive_buf[0];
H0_rh = buffer[0] >> 1;
read_register(0xBE, 0x31, i2c_receive_buf);
buffer[1]=i2c_receive_buf[0];
H1_rh = buffer[1] >> 1;
```

6) Считайте значение H0_T0_OUT из регистров 0x36 и 0x37.

```
read_register(0xBE, 0x36, i2c_receive_buf);
buffer[0]=i2c_receive_buf[0];
read_register(0xBE, 0x37, i2c_receive_buf);
buffer[1]=i2c_receive_buf[0];
H0_T0_out = (((uint16_t)buffer[1]) << 8) | (uint16_t)buffer[0];
```

7) Считайте значение H1_T0_OUT из регистров 0x3A и 0x3B.

```
read_register(0xBE, 0x3A, i2c_receive_buf);
```

buffer[0]=i2c_receive_buf[0];
read_register(0xBE, 0x3B, i2c_receive_buf);
buffer[1]=i2c_receive_buf[0];
H1_T0_out = (((uint16_t)buffer[1]) << 8) | (uint16_t)buffer[0];

```
  /* USER CODE BEGIN 2 */
//Initialization
write_register(0xBE, 0x20, 0x87);
//---------------------------------------
read_register(0xBE, 0x30, i2c_receive_buf);
buffer[0]=i2c_receive_buf[0];
H0_rh = buffer[0] >> 1;
read_register(0xBE, 0x31, i2c_receive_buf);
buffer[1]=i2c_receive_buf[0];
H1_rh = buffer[1] >> 1;

read_register(0xBE, 0x36, i2c_receive_buf);
buffer[0]=i2c_receive_buf[0];
read_register(0xBE, 0x37, i2c_receive_buf);
buffer[1]=i2c_receive_buf[0];
H0_T0_out = (((uint16_t)buffer[1]) << 8) | (uint16_t)buffer[0];
read_register(0xBE, 0x3A, i2c_receive_buf);
buffer[0]=i2c_receive_buf[0];
read_register(0xBE, 0x3B, i2c_receive_buf);
buffer[1]=i2c_receive_buf[0];
H1_T0_out = (((uint16_t)buffer[1]) << 8) | (uint16_t)buffer[0];

  /* USER CODE END 2 */
```

Рисунок 7.7 – Инициализация датчика

Далее перейдите в цикл while(1) и считайте показания влажности.

8) Считайте значение влажности в необработанных счетах H_T_OUT из регистров 0x28 и 0x29.

```
  while (1)
  {
HAL_Delay(1000);
read_register(0xBE, 0x28, i2c_receive_buf);
buffer[0]=i2c_receive_buf[0];
read_register(0xBE, 0x29, i2c_receive_buf);
buffer[1]=i2c_receive_buf[0];
H_T_out = Your code buffer[0] and buffer[1] to 16 bit;
humidity= Your code;
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}
```

Рисунок 7.8 – Считывание данных

9) Вычислите значение RH [%] с помощью линейной интерполяции, используя приведенную ниже формулу:

$$H_{RH}[\%] = \frac{(H1\_rH - H0\_rH)(H\_T\_OUT - H0\_T0\_OUT)}{H1\_T0\_OUT - H0\_T0\_OUT} + H0\_rH \quad (7.1)$$

Обратите внимание, что коэффициенты H0_rH, H1_rH, H0_T0_OUT и H1_T0_OUT откалиброваны на заводе и сохранены в памяти, а значение H_T_OUT (0x28 и 0x29) зависит от температуры T, измеренной датчиком во время считывания относительной влажности. Таким образом, вы можете выполнить шаги 1, 2, 3 и 4 только один раз и циклически повторять только шаги 5 и 6.

Это значит, что шаги 1-4 нужно записать в область /* USER CODE BEGIN 2 */ /* USER CODE END 2 */, а шаги 5-6 в /* USER CODE BEGIN WHILE */ /* USER CODE END WHILE */.

Когда рассчитываете формулу приведите ее к типу float

Humidity = (float) Your formula;

Так как относительная влажность воздуха не может быть более 100%, и не может быть менее 0%, сделайте условие при помощи функции «if». Если значения влажности, показанные датчиком выходят за пределы, то нужно приравнивать их к 100% или 0%.

Скомпилируйте проект, подключите плату к компьютеру и загрузите код в плату. Запустите программу STMStudio и посмотрите показания влажности температуры.

Контрольные вопросы

1) Как работает датчик HTS221?
2) Как производится калибровка датчика влажности?
3) Как работает интерфейс обмена между микроконтроллером и датчиком по I2C?
4) Из каких этапов состоит считывание данных с датчика?
5) Как производится инициализация датчика?

Содержание отчета

1) Цель работы;
2) Подробное описание всех этапов проделанной работы;
3) Ответы на вопросы, представленные в тексте лабораторной работы;
4) Важные части вашего кода с пояснениями;
5) Графики и формулы, полученные в процессе выполнения лабораторной работы;
6) Анализ проделанной работы;
7) Выводы по данной лабораторной работе.

Список использованных источников

1. B-L475E-IOT01A – User manual [Электронный ресурс]. – Режим доступа: https://www.st.com/resource/en/user_manual/dm00347848.pdf
2. STM32CubeMX [Электронный ресурс]. – Режим доступа: https://www.st.com/en/development-tools/stm32cubemx.html
3. uVision IDE [Электронный ресурс]. – Режим доступа: http://www2.keil.com/mdk5/uvision/
4. Reference manual – STM32L4x5 and STM32L4x6 advanced Arm-based 32-bit MCUs [Электронный ресурс]. – Режим доступа: https://www.st.com/resource/en/reference_manual/dm00083560.
5. LPS22HB [Электронный ресурс]. – Режим доступа: https://www.st.com/en/mems-and-sensors/lps22hb.html
6. hts221 – Datasheet [Электронный ресурс]. – Режим доступа: https://www.st.com/resource/en/datasheet/hts221.pdf
7. MEMS [Электронный ресурс]. – Режим доступа: http://ru.wikibedia.ru/wiki/MEMS
8. I2C [Электронный ресурс]. – Режим доступа: http://old.symmetron.ru/suppliers/nxp/HCS08_11_h-1-2.shtml