Министерство науки и высшего образования Российской Федерации

Томский государственный университет
систем управления и радиоэлектроники

И. М. Васильев

Применение микроконтроллеров архитектуры ARM для сопряжения узлов аппаратуры робототехнических систем с помощью протоколов последовательной связи

Методические указания по выполнению лабораторных работ студентов

Томск
2024

УДК 004.382.4:004:423
ББК 3стд2-02
В 19

Рецензент:

Бикмуллин Э.А., технический директор ООО «Ди Эй Групп»

Васильев, Иван Михайлович
В 19 Применение микроконтроллеров архитектуры ARM для сопряжения узлов аппаратуры робототехнических систем с помощью протоколов последовательной связи: Васильев, И. М. Применение микроконтроллеров архитектуры ARM для сопряжения узлов аппаратуры робототехнических систем с помощью протоколов последовательной связи: Методические указания к лабораторным работам / И. М. Васильев. — Томск: ТУСУР, 2024. — 36 с.

Методические указания содержат рекомендации и материалы, необходимые для лабораторной работы студентов для инженерно-технических специальностей.

Одобрено на заседании кафедры УИ, протокол №4 от25ноября 2024 года.

УДК 004.382.4:004:423
3стд2-02

© Васильев И.М., 2024
© Томск. гос. ун-т систем упр. и радиоэлектронники, 2024

Оглавление

Общие положения 4

Лабораторная работа № 1. Работа с конфигуратором и отладочным интерфейсом микропроцессоров STM32. 5

Лабораторная работа № 2. Прием и передача данных по UART/USART. 14

Лабораторная работа № 3. Прием и передача данных по протоколу SPI. 19

Лабораторная работа № 4. Прием и передача данных по протоколу I2C. 27

Список рекомендуемой литературы 36

Общие положения

Данные методические указания разработаны для студентов, обучающихся в Томском государственном университете систем управления и радиоэлектроники (далее - Университет).

Структура учебного процесса предполагает выполнение студентами лабораторных работы, как по освоению теоретического материала, так и в рамках выполнения практических заданий. Рекомендации по выполнению лабораторных работ студентов приведены в соответствующих методических указаниях.

В ходе выполнения лабораторных работ студентам прививаются навыки работы с зарубежной технической документацией, работой с микропроцессором архитектуры ARM, проектирования электрических принципиальных схем простейших устройств на базе микропроцессоров, организации последовательной передачи данных между устройствами по стандартным протоколам.

Рекомендации подготовлены с целью помочь студентам в успешном освоении учебных дисциплин и подготовке к промежуточным этапам аттестации.

Лабораторная работа № 1.
Работа с конфигуратором и отладочным интерфейсом микропроцессоров STM32.

Цель работы: изучение принципов работы конфигуратора микропроцессора и интерфейса отладки SWD на базе созданного проекта.

Вводная часть

Микропроцессоры и микроконтроллеры играют ключевую роль в разработке различных электронных устройств. Одной из самых популярных линейкой микроконтроллеров является STM32, которая предлагает широкий спектр возможностей для работы с портами ввода/вывода (I/O). Понимание принципов работы этих портов является основой для создания эффективных и функциональных проектов.

1. Создание шаблона проекта

1.1. Для создания шаблона проекта откройте программу STM32CubeIDE и создайте новый проект, сохранив его в директорию по умолчанию согласно рисунку Рисунок 1.

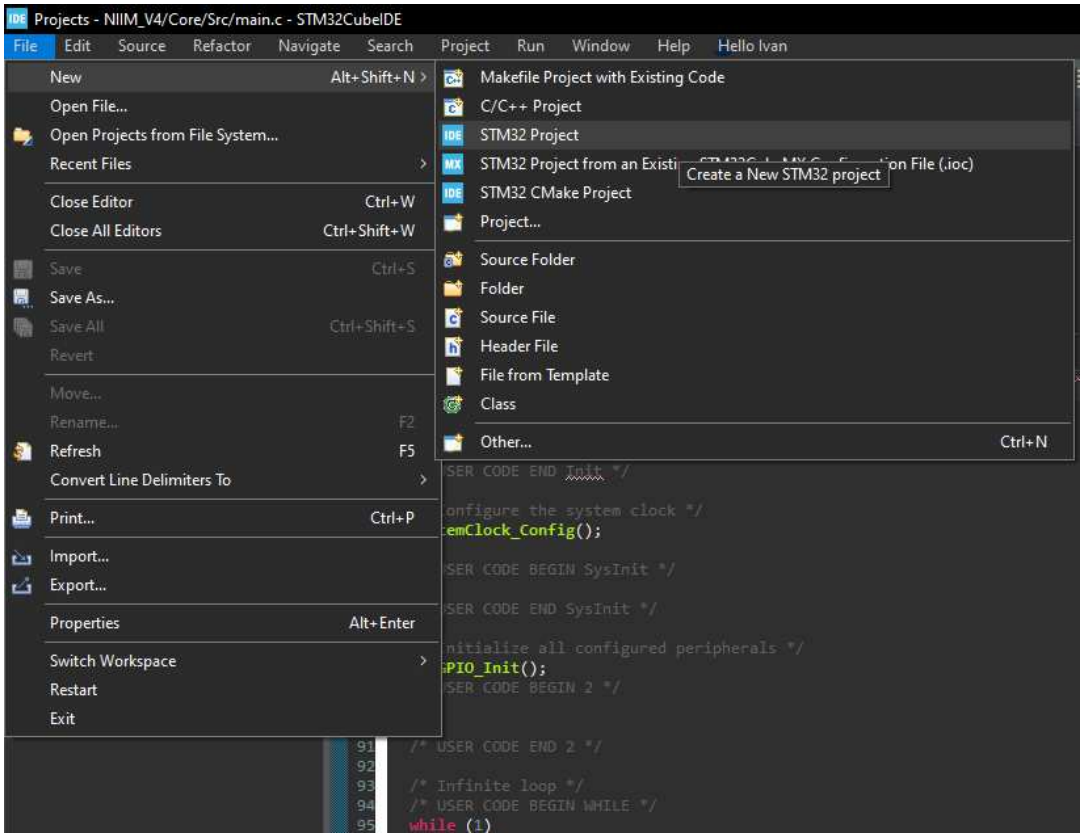

Рисунок 1 - Пример создания проекта в STM32CubeIDE.

1.2. Следующим шагом будет выбор микропроцессора. В рамках данных лабораторных работ используется STM32f103C6T6A. Данную маркировку так же можно увидеть на корпусе микропроцессора.

1.3. После сохранения проекта у вас откроется файл конфигурации с расширением *.ioc. Первым делом необходимо выбрать **источник тактовой частоты**. В нашем случае на плате установлен резонатор 8 МГц. Обратите внимание, что конфигуратор зарезервировал выделенные выводы, и в последующем не даст пользователю использовать их для других нужд. Примеры конфигурации тактовых выводов приведены на рисунках 2 и 3.

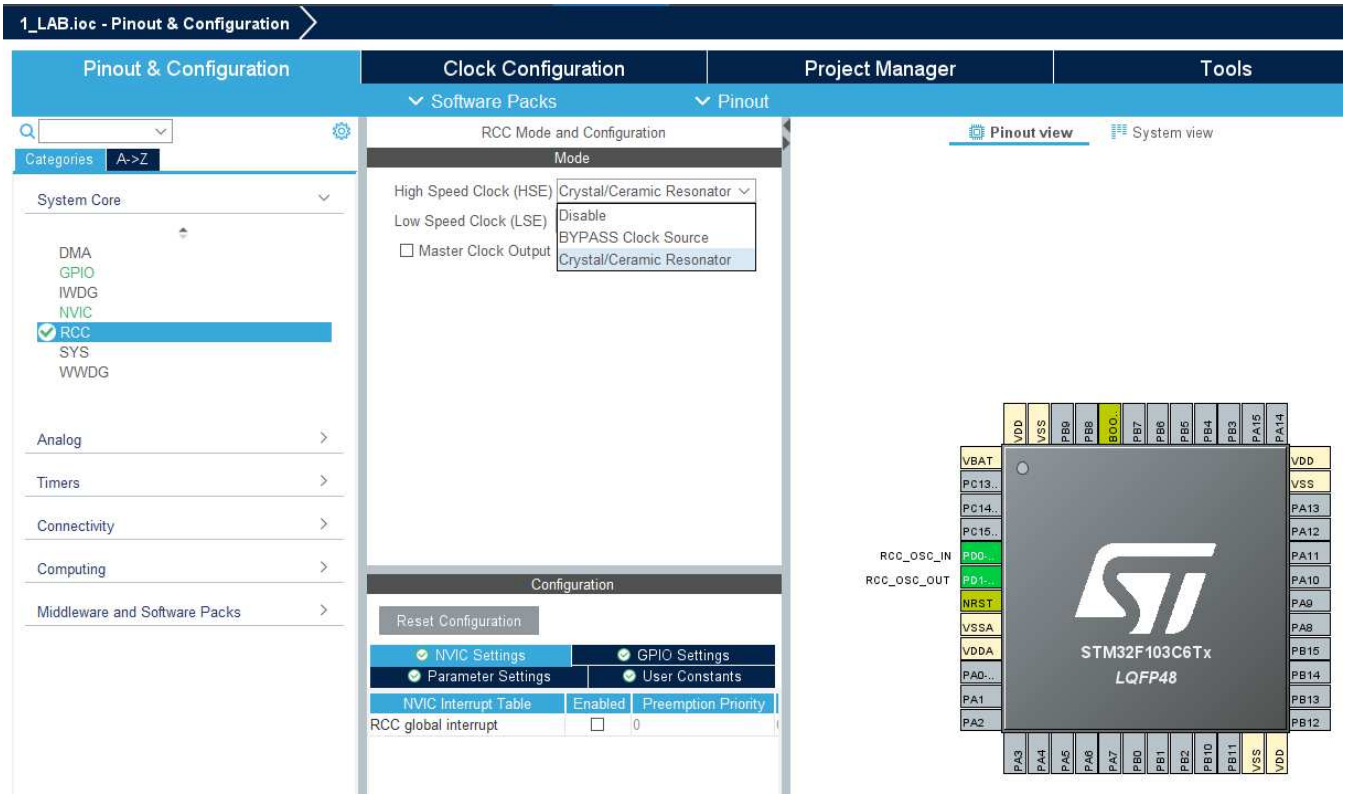

Рисунок 2 - Назначение источника тактовой частоты.

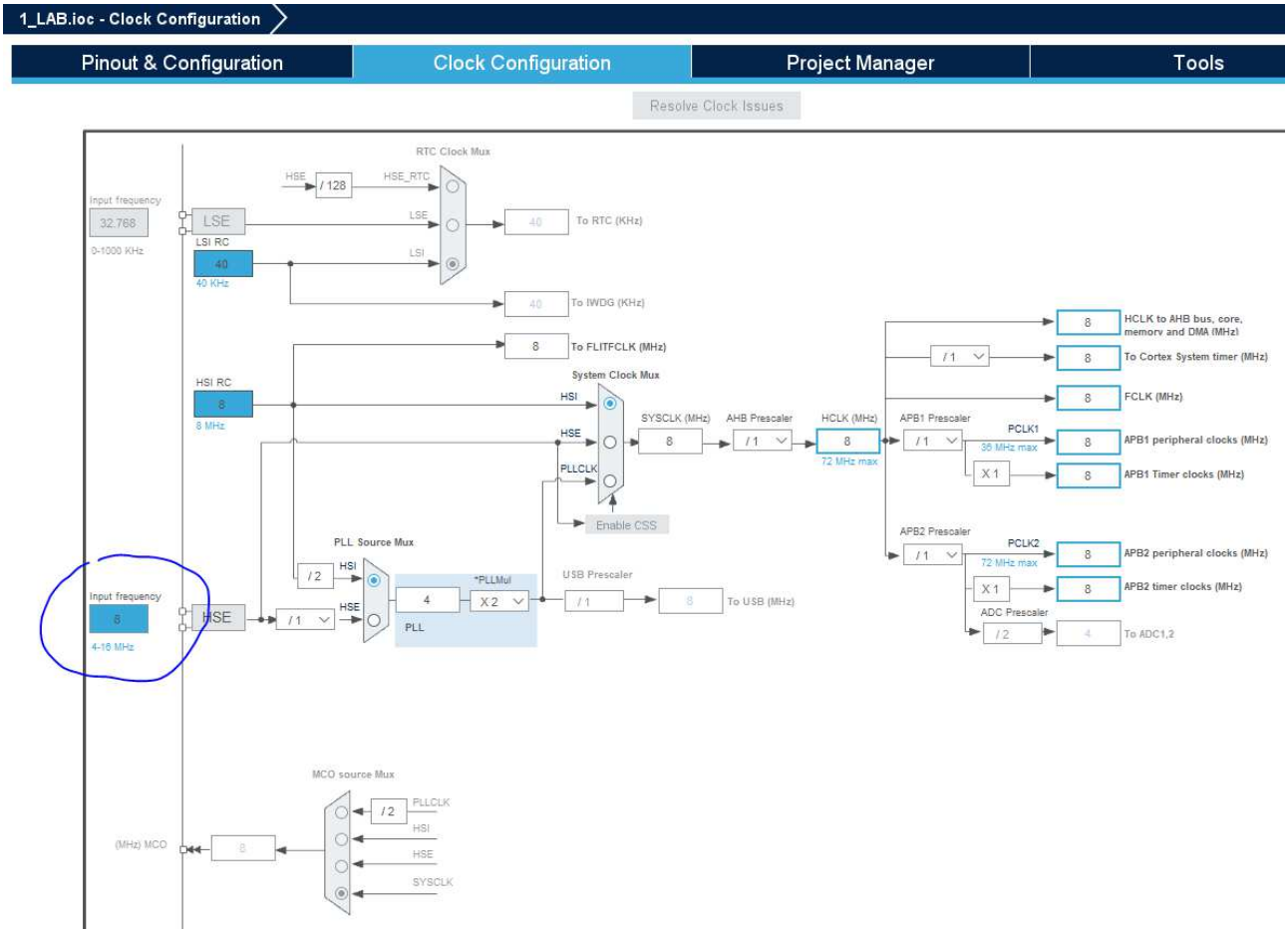

Рисунок 3 - Значение тактовой частоты в конфигураторе

1.4. Следующим шагом выберем интерфейс для отладки ПО на примере программатора ST-Link V2, который может использовать отладочный интерфейс Serial Wire Debug. Конфигурация интерфейсных выводов приведена на рисунке 4.

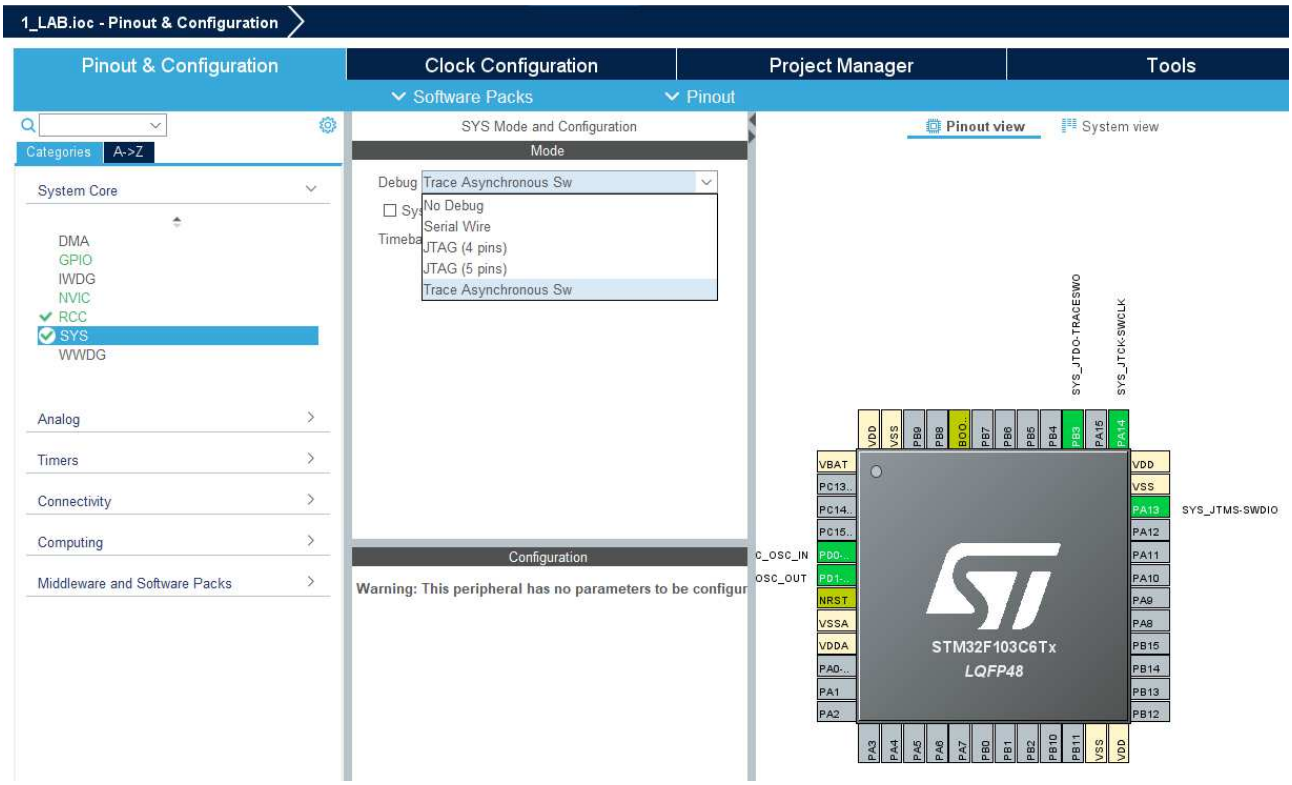

Рисунок 4 - Пример назначения выводов отладочного интерфейса в конфигураторе.

1.5. Для настройки любого вывода микропроцессора на выход достаточно выбрать его, а затем присвоить доступную через конфигуратор функцию. В данном примере используется вывод PC13 как логический выход с названием «LED_BUILTIN» согласно рисунку 5. Конфигуратор удобен тем, что предлагает только доступные функции, и доступные выводы, что в последствии не приводит к коллизиям при переназначении функционала вывода.

Использование конфигуратора для настройки выводов микропроцессора значительно упрощает процесс разработки, особенно для начинающих. Благодаря интуитивно понятному интерфейсу, разработчики могут быстро находить и назначать необходимые функции, избегая ошибок, связанных с неправильным подключением или настройкой. Например, при выборе вывода PC13 как логического выхода, разработчик может легко задать его состояние (высокий или низкий уровень), что позволяет сразу же управлять подключенными устройствами, такими как светодиоды или реле. Это особенно полезно в проектах, где требуется быстрая проверка работоспособности.

Кроме того, конфигуратор может автоматически генерировать код, который затем можно интегрировать в основной проект. Это позволяет сократить время на написание и отладку кода, так как многие настройки уже будут учтены. Также, благодаря возможности визуального представления конфигурации, разработчики могут лучше понимать, как различные компоненты взаимодействуют друг с другом. В конечном итоге, использование конфигуратора не только повышает эффективность разработки, но и способствует созданию более надежных и устойчивых к ошибкам систем.

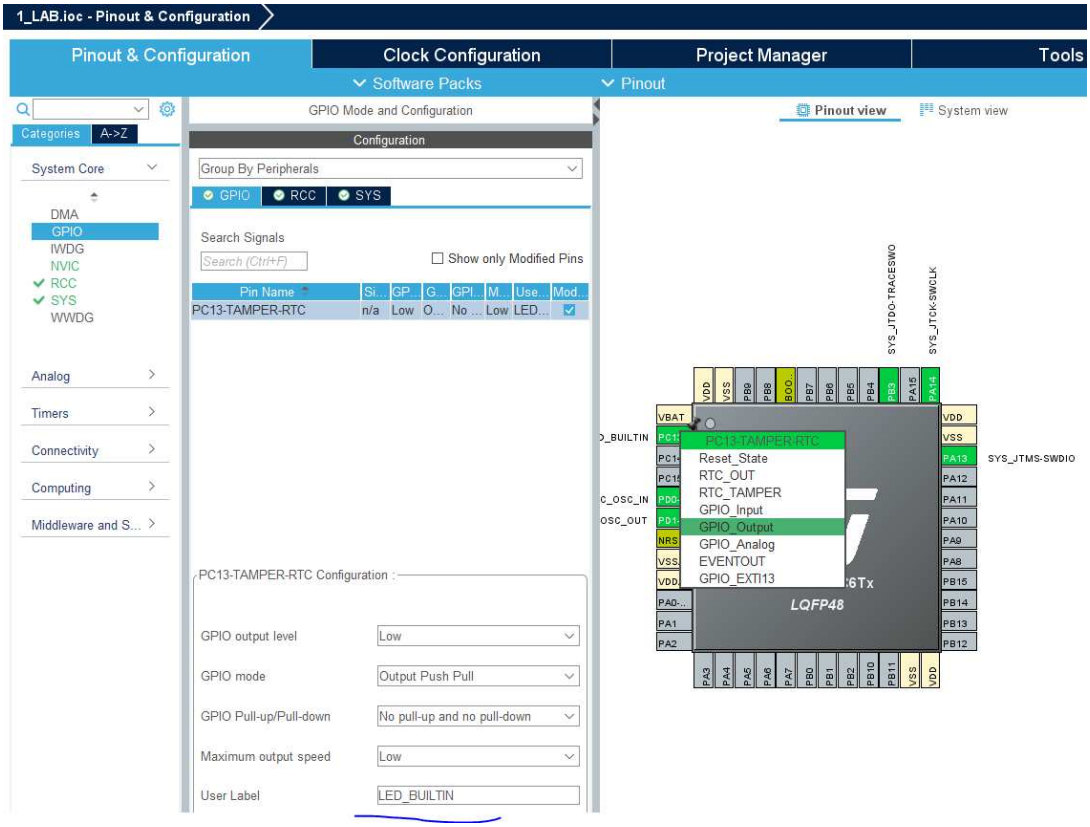

Рисунок 5 - Пример назначения выводов GPIO

1.6. Следующим шагом следует выставить оптимальные настройки для генерации проекта согласно рисунку 6.

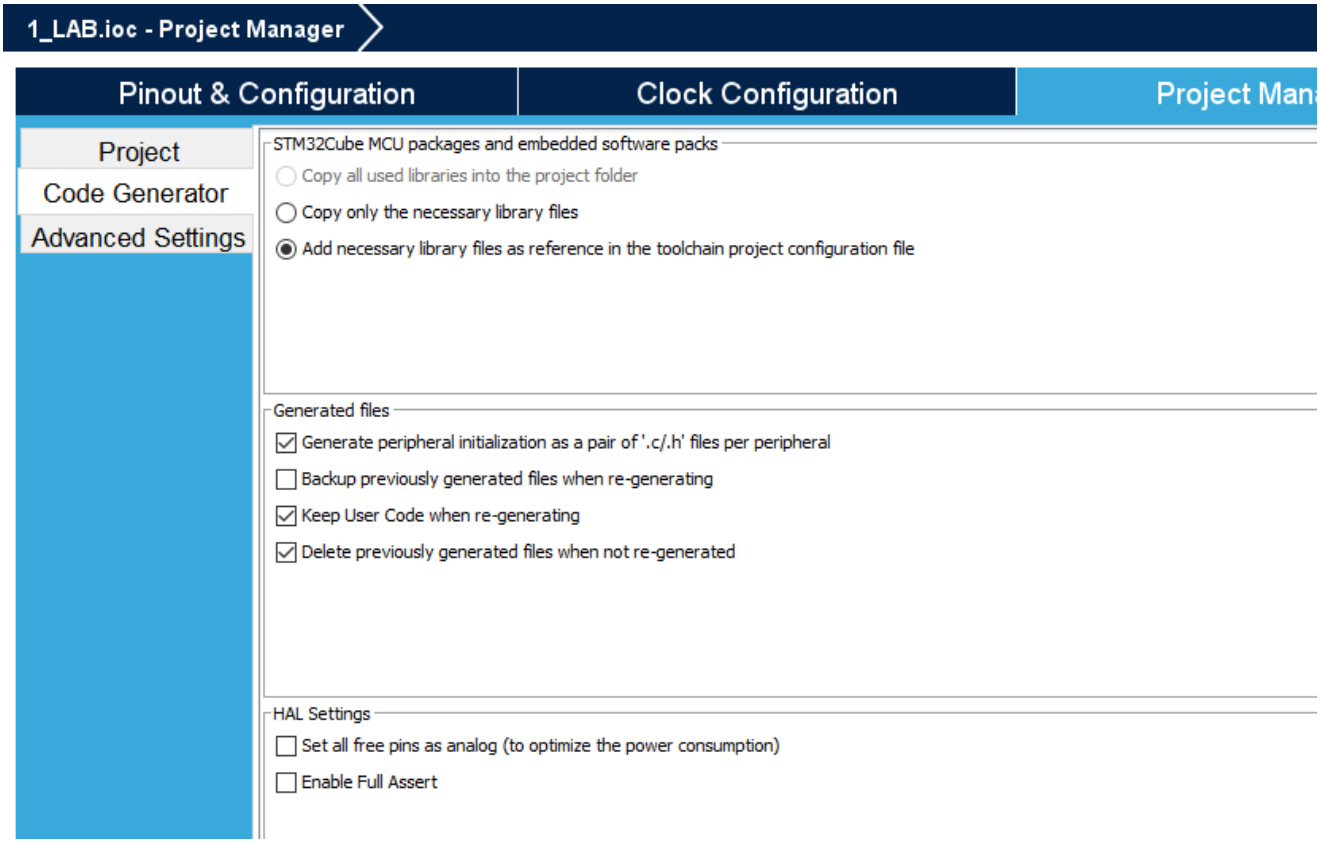

Рисунок 6 - Настройки генератора проекта.

1.7. После сохранения файла конфигурации проект автоматически сгенерируется и в окне будет видна структура проекта. Структура должна соответствовать рисунку 7.

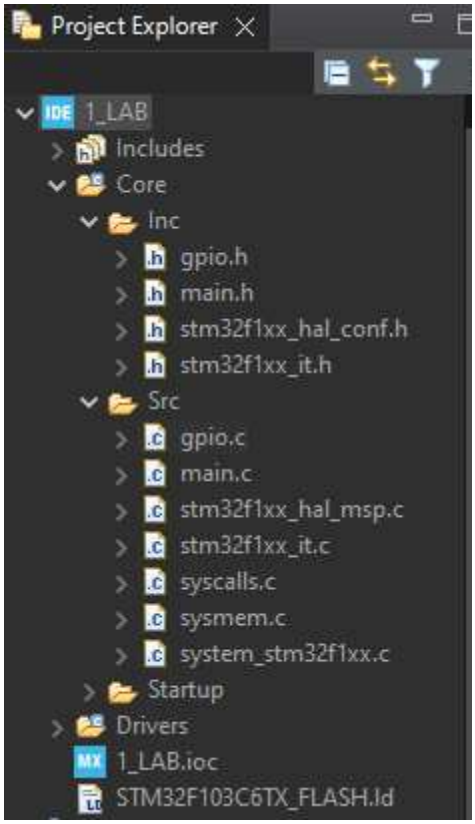

Рисунок 7 - Типовая структура проекта STM32CubeIDE.

В данном проекте основное тело программы реализовано в файле **main.c.** Инициализация **GPIO** содержится в файле **GPIO.c**. Эти файлы объединены в проекте и позволяют ссылаться на выполняемые функции внутри каждого файла через заголовочные файлы с расширением ***.h.**

Файл **STM32F1xx_it.c** содержит в себе функции обработчиков **прерываний**.

Файл **STM32F1xx_hal_msp.c** содержит в себе набор инструкций для использования HAL функций. **HAL** (**H**ardware **A**bstraction **L**ayer) – набор библиотек, созданный разработчиком микропроцессора для использования абстрактных функций, что упрощает пользователям написание кода. Описание абстрактных функций приведено в п 3.11.2 руководства https://clck.ru/3EQMXD.

2. Создание кода программы и отладка

2.1. Для написания простейшей программы по миганию светодиода необходимо в файле main.c вставить соответствующие инструкции для микроконтроллера.

```
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  /* USER CODE BEGIN 2 */

  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */
              HAL_GPIO_TogglePin(GPIOC, LED_BUILTIN_Pin);
//переключаем состояние встроенного светодиода
              HAL_Delay(250);
//задержка 250 мс
    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}
```

2.2. Для включения микропроцессора необходимо подготовить файлы инструкций, собрав проект согласно рисунку 8.

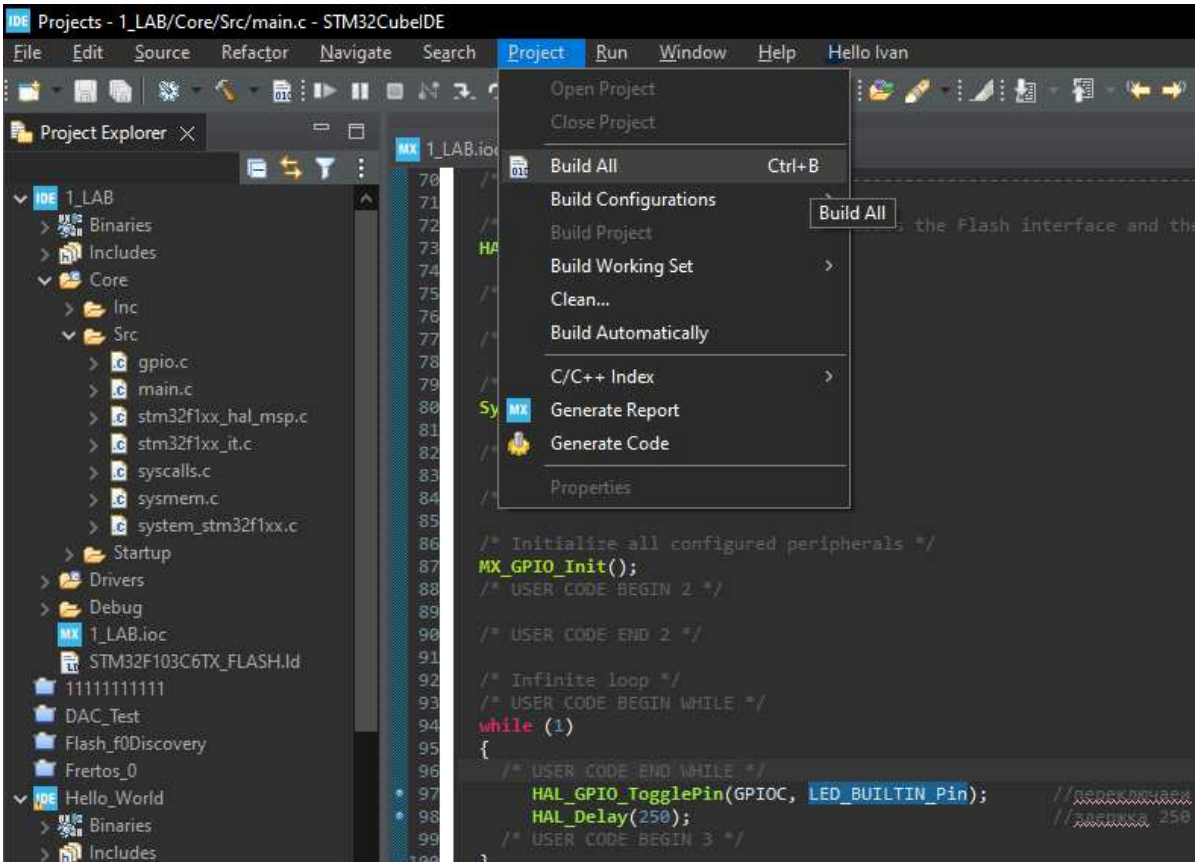

Рисунок 8 - Сборка проекта STM32CubeIDE.

2.3. Подключите программатор ST-Link V2 согласно маркировке на корпусе к выводам микропроцессора. Для удобства пользователей на плате BluePill выводы SWD выведены на отдельный разъем согласно рисунку 9.

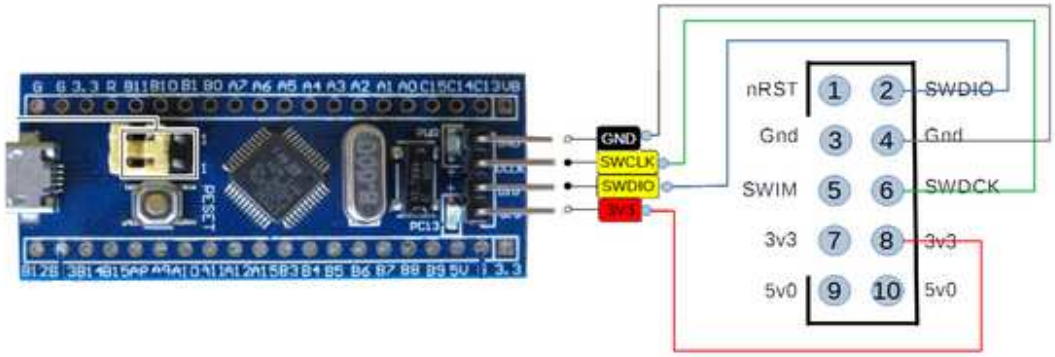

Рисунок 9 - Схема подключения программатора ST-Link V2 к отладочной плате.

2.4. Установите точки останова на выделенных строках, нажав в соответствующую область 2 раза левой кнопкой мыши. Графический интерфейс при успешной операции должен соответствовать рисунку 10.

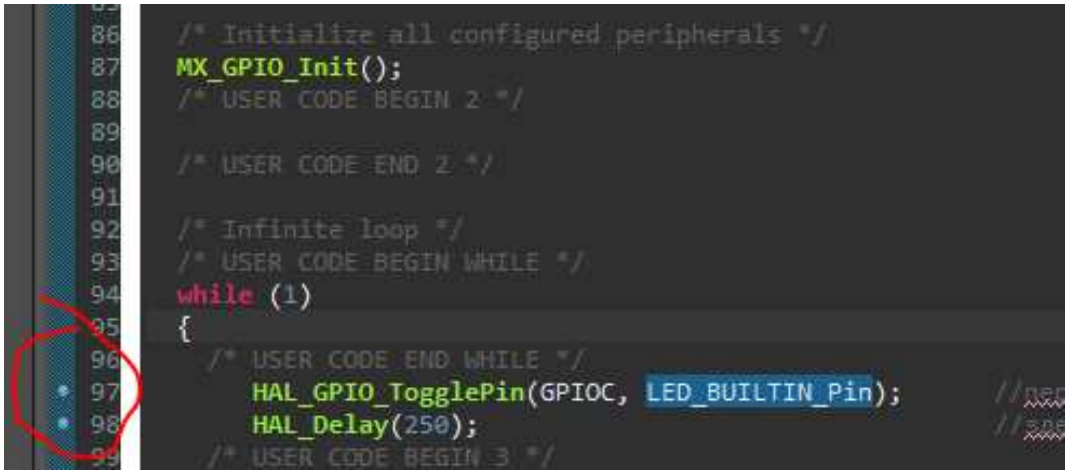

Рисунок 10 - Установка точек останова в среде STM32CubeIDE.

2.5. Запустите программу в режиме Debug нажав соответствующую кнопку в панели инструментов согласно рисунку 11.

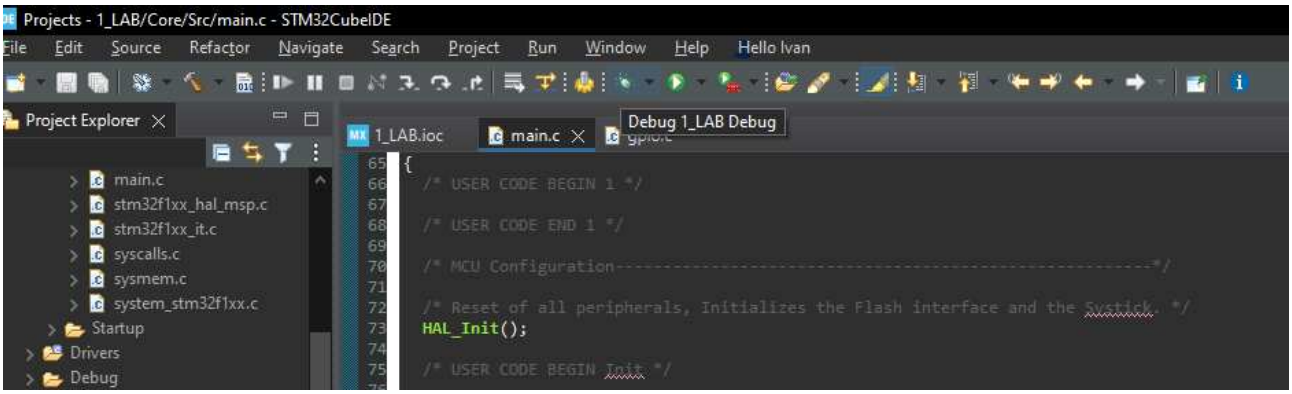

Рисунок 11 - Запуск программы в режиме Debug.

Убедитесь что встроенный светодиод меняет свое состояние по нажатии на кнопку «Resume» графического интерфейса программы.

2.6. Запустите программу в обычном режиме нажав соответствующую кнопку в панели инструментов согласно рисунку 12.

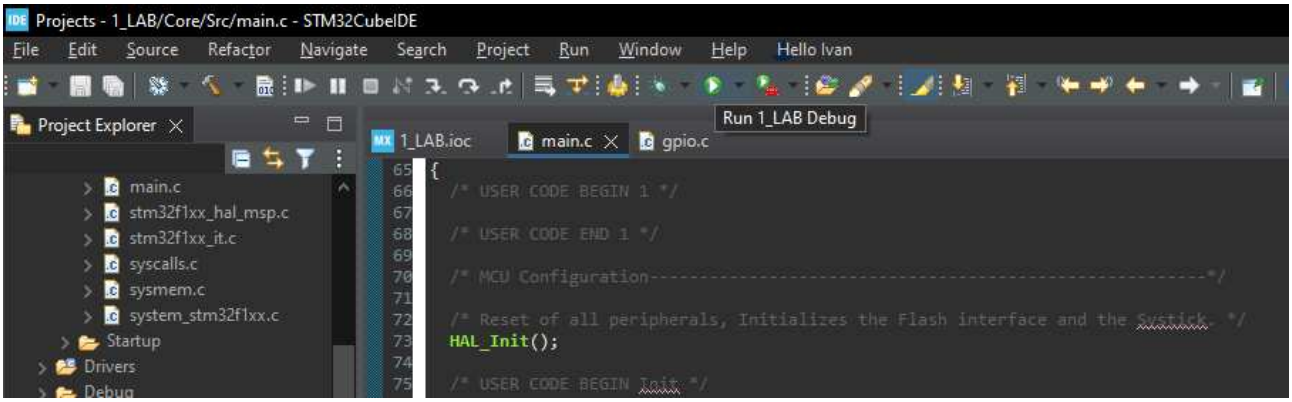

Рисунок 12 - Запуск написанной программы в обычном режиме

Убедитесь, что встроенный светодиод меняет свое состояние всё время.

3. Выполните одно из предложенных заданий для самостоятельной работы для закрепления базовых навыков.

Варианты заданий:

№	Задание
1.	Организовать бегущие огни на светодиодах.
2.	Организовать бегущие огни, меняющие направление по нажатию кнопки.
3.	Организовать бегущие огни, меняющие режим по нажатию кнопки. Кол-во режимов не менее трех.
4.	Организовать плавное свечение светодиода, регулируемое кнопками.
5.	Организовать устранение дребезга тактовой кнопки.
6.	Двумя кнопками регулировать яркость свечения светодиода.
7.	Организовать плавное изменение свечения RGB светодиода во всем видимом спектре
8.	Переменным резистором регулировать скорость бегущих огней.
9.	Переменным резистором организовать шкалу на 6и светодиодах
10.	Организовать вывод данных с датчика влажности в последовательный монитор
11.	Организовать вывод данных с датчика температуры в последовательный монитор
12.	Переменным резистором дискретно регулировать частоту звучания ноты в пределах одной октавы
13.	Организовать вывод данных с датчика температуры и датчика влажности в последовательный монитор
14.	Организовать вращение сервопривода на угол, соответствующий углу поворота переменного резистора
15.	Переменным резистором регулировать скорость вращения шагового двигателя
16.	Переменным резистором регулировать частоту выдаваемой с помощью ШИМ синусоиды
17.	Регулировать переменным резистором свечение RGB светодиода во всем видимом спектре

4. Составление отчета. В отчет включить схему соединения электронных устройств между собой и описание работы программы. Обязательно укажите аппаратные модули, используемые микропроцессором и обоснование их применения.

Лабораторная работа № 2.

Прием и передача данных по UART/USART.

Цель работы: изучение принципов работы интерфейсов UART и USART на микроконтроллерах STM32, а также разработка и реализация системы передачи данных между двумя устройствами с использованием данного интерфейса

Вводная часть

Интерфейс UART является одним из самых распространенных способов передачи данных между микропроцессорами и другими устройствами. Он обеспечивает асинхронную передачу данных, что позволяет передавать информацию без необходимости синхронизации тактовых сигналов между передающим и принимающим устройствами. Микропроцессоры STM32, благодаря своей высокой производительности и широким возможностям, являются идеальной платформой для реализации проектов, использующих UART/USART.

Краткие теоретические сведения

UART - Universal Asynchronous Receiver/Transmitter (универсальный асинхронный приемник/передатчик). Это устройство, встроенное в микроконтроллер, отвечает за преобразование входящих и исходящих байтов в последовательный поток данных.

Схема подключения двух устройств по стандарту UART приведена на рисунке 13.

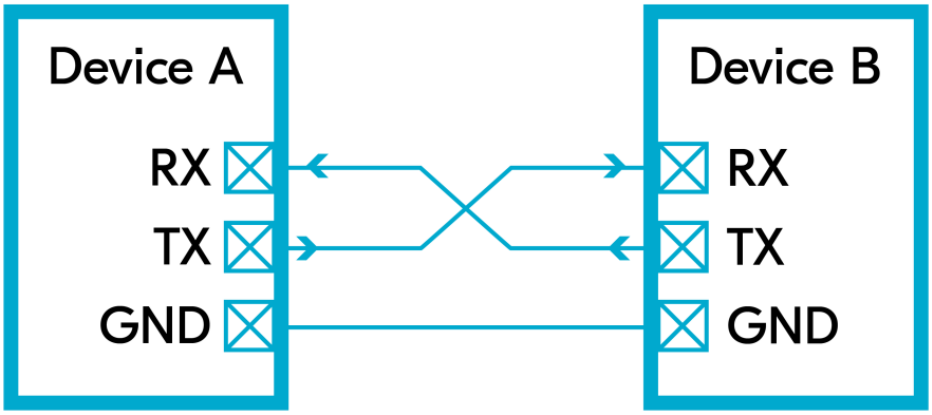

Рисунок 13 - Стандартная схема подключения устройств по интерфейсу UART.

В данном случае выводы Rx и Tx обозначают направленность данных. Rx обозначает Receive, т.е. прием. Tx в свою очередь Transmit, т.е. передатчик. Логическая схема UART включает в себя логические входы и выходы, которые соответствуют полупроводниковой технологии, такой как КМОП или ТТЛ. Этот физический уровень обычно подходит для использования внутри одного устройства, однако для длинных соединений он не всегда эффективен. Это связано с низким уровнем защиты от электрических повреждений и помех, что делает его менее пригодным для коммутируемых соединений на больших расстояниях. Логика UART позволяет производить передачу и прием одновременно, что обозначается термином дуплекс.

Пример передачи информации приведен на рисунке 14

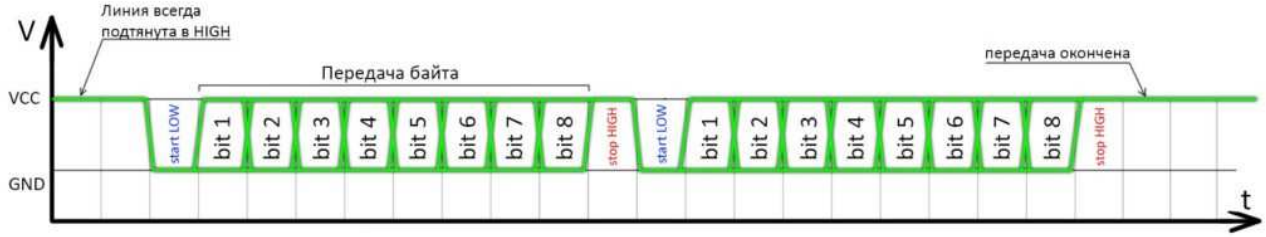

Рисунок 14 - Пример передачи информации по протоколу UART.

Передача начинается со стартового бита, который сигнализирует о начале отправки битов, и вызывает срабатывание прерывание аппаратного модуля обработки данных. Завершение слова данных происходит с помощью стопового бита.

Для успешного асинхронного приема и передачи данных сопрягаемые устройства должны быть настроены на одну и ту же скорость передачи данных. Самые распространенные скорости это 9600 и 115200 бод/с.

При уходе внутренней частоты тактирования возможен прием ложной информации. Для минимизации погрешностей при асинхронном приеме данные принимаются через равные промежутки времени, которые равны половине стартового бита. Пример сэмплирования приведен на рисунке 15.

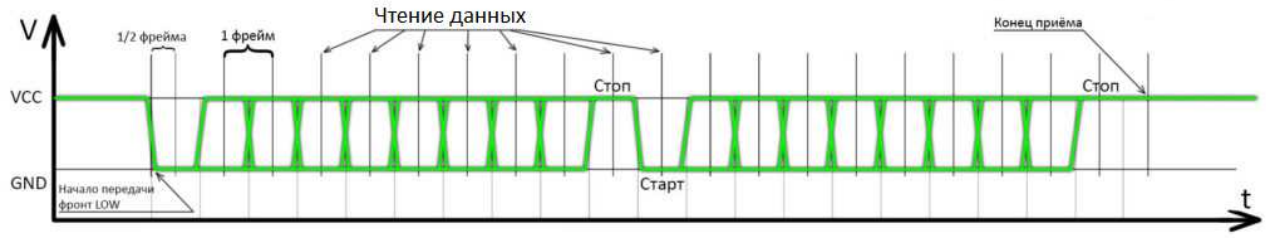

Рисунок 15 - Сэмплирование данных при заданной скорости асинхронного обмена данными.

USART - **U**niversal **S**ynchronous/**A**synchronous **R**eceiver/**T**ransmitter. Это схожий протокол передачи данных, однако его отличие от UART заключается в том, что частота сэмплирования принимающего устройства не зависит от внутренней частоты и поступает по отдельному проводу из вне.

Кроме того, для обнаружения ошибок в процессе передачи UART/USART может добавлять контрольный бит в поток данных. Это так называемей **бит чётности** (**Parity bit**), который устанавливается в «0» или в «1», если отправляемый байт является четным или нечетным числом соответственно. Таким образом, если мы отправляли байт «0b00000001» и приняли бит четности «1», то произошел сбой при передачи данных и необходимо провести повторную отправку. Бит четности – это **дополнительная** опция и при неправильной настройке интерфейса для сопряжения с другим устройством этот бит может быть прочитан как бит данных. Обратите внимание, что количество бит данных в пакете так же может быть отличным от 8-и, а длительность стоп бита – как 1.0, так и 1,5 от длины старт бита.

Таким образом, критерий **успешной** передачи данных это **соответствие скорости** сопрягаемых устройств, а так же **одинаковая конфигурация передачи данных** – наличие/отсутствие бита четности, количество бит данных в пакете и длительность стоп-бита.

1. Создание проекта.

1.1. Создайте новый проект на основе созданной ранее конфигурации согласно рисунку 16.

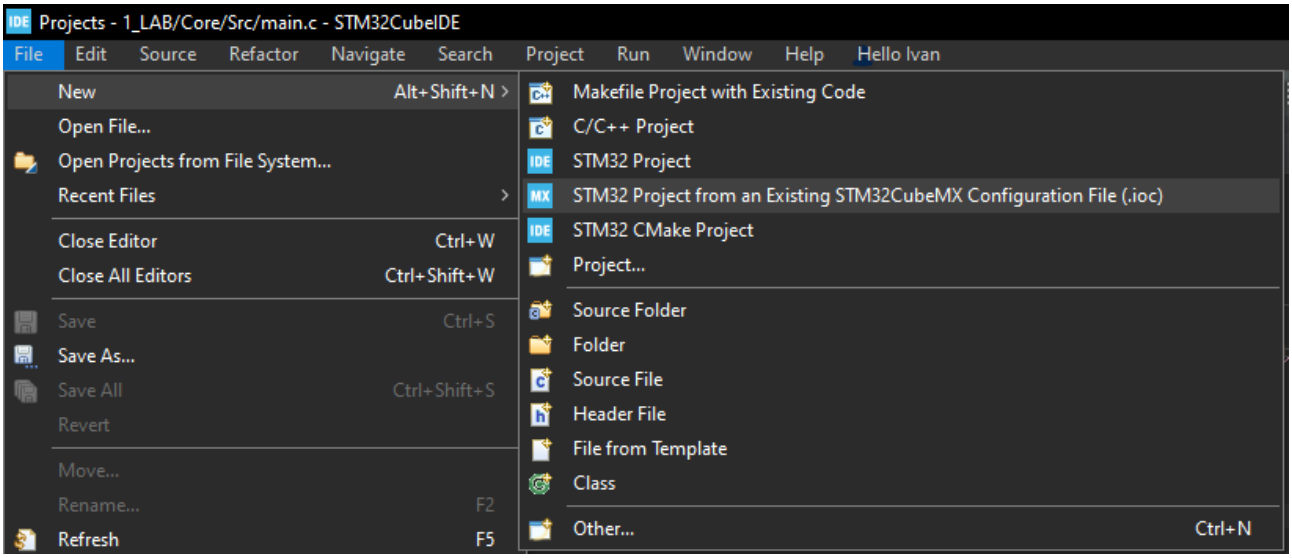

Рисунок 16 - Создание проекта на основе имеющегося файла конфигурации.

В конфигураторе укажите тип используемого интерфейса USART1. Конфигуратор предложит подходящие выводы, в данном случае это **PA9** и **PA10**. Для последующей отладки включите прерывание по приему данных согласно рисунку 17.

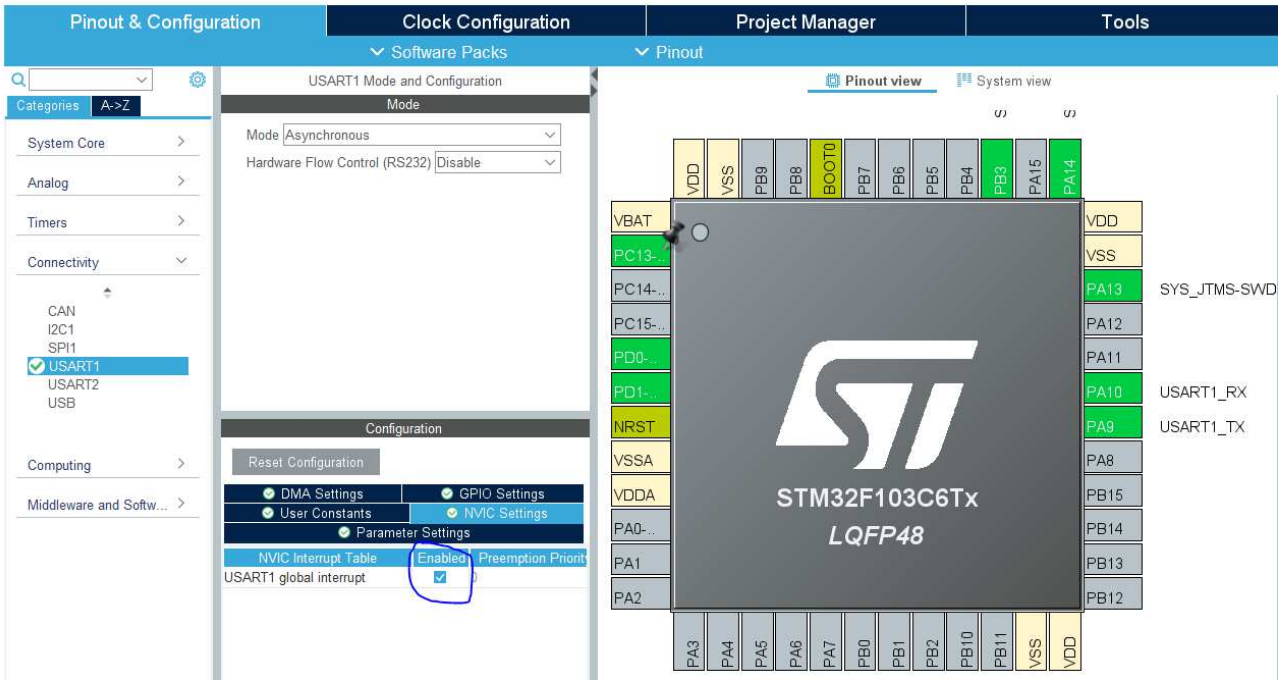

Рисунок 17 - Конфигурация интерфейса USART1.

1.2. Создайте новый проект на основе созданной конфигурации. Сохраните конфигурацию, и убедитесь, что в файле stm32f1xx_**it**.c появился обработчик прерываний **USART1_IRQHandler**.

1.3. Подключите преобразователь интерфейсов USB to TTL HW-597 к отладочной плате согласно распиновке конфигуратора и рисунку 13. Обязательно соедините линии GND, дополнительное питание на преобразователь USB to TTL подавать не обязательно.

1.4. Откройте терминал последовательного порта на ПК (Terminal1_9_b). Допускается использовать онлайн сервисы, например https://www.serialterminal.com.

2. Создание кода программы и отладка

2.1. В основное тело программы добавьте строки:

```
int main(void)
{
  /* USER CODE BEGIN 1 */
  volatile uint8_t str[1]={1};
// выделяем область памяти для хранения принятых, либо отправляемых данных
  /* USER CODE END 1 */
……
……
  while (1)
  {
    /* USER CODE END WHILE */
  HAL_UART_Transmit_IT(&huart1, str, sizeof(str));
// отправляем данные по инициализированному UART, пакет данных размером с выделенный ранее массив
  HAL_UART_Receive_IT(&huart1, str, 1);
// разрешаем прием по прерыванию только одного байта, т.к. вторым байтом отправляется символ переноса строки

              if (str[0]==0x31)//если приняли байт «1» в кодировке ANSI
              {
  HAL_GPIO_WritePin(GPIOC, LED_BUILTIN_Pin, GPIO_PIN_SET);
//зажигаем светодиод
              }
            else
            {
HAL_GPIO_WritePin(GPIOC, LED_BUILTIN_Pin, GPIO_PIN_RESET);
// иначе - тушим
            }
              HAL_Delay(500);
  }
```

2.2. В настройках терминала укажите параметры, выставляемые в конфигураторе при настройке интерфейса UART согласно рисунку 18.

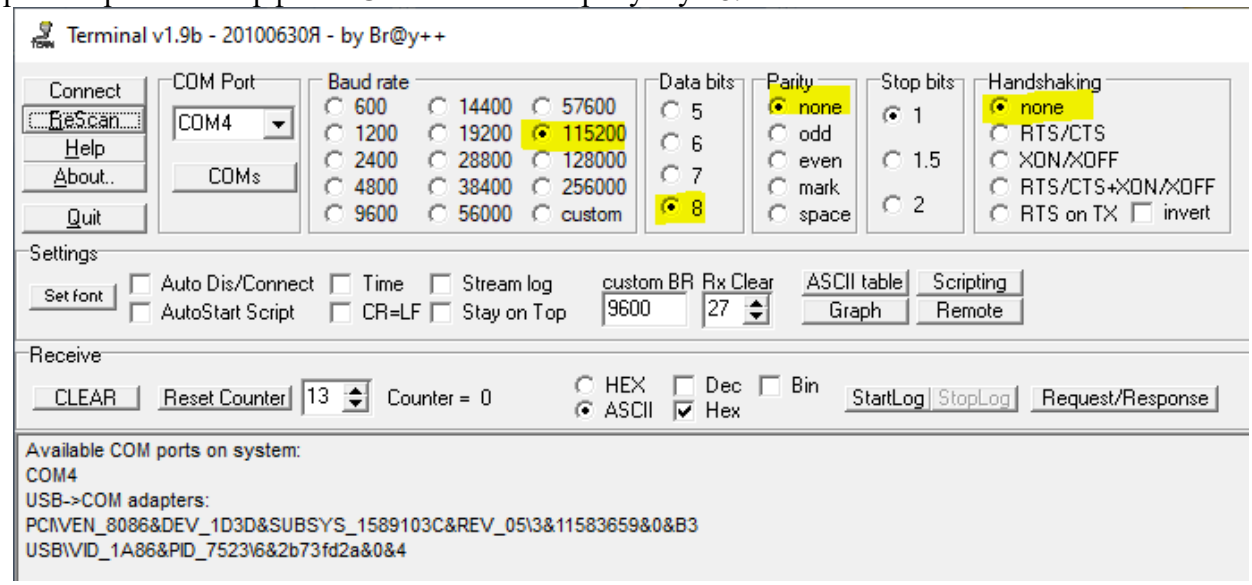

Рисунок 18 - Настройки терминала приема данных по UART.

2.3. Исследуйте работу программы в пошаговом режиме, установив точку останова согласно рисунку 19. При принятии байта 0x31 («1» в кодировке ANSI) светодиод должен загораться.

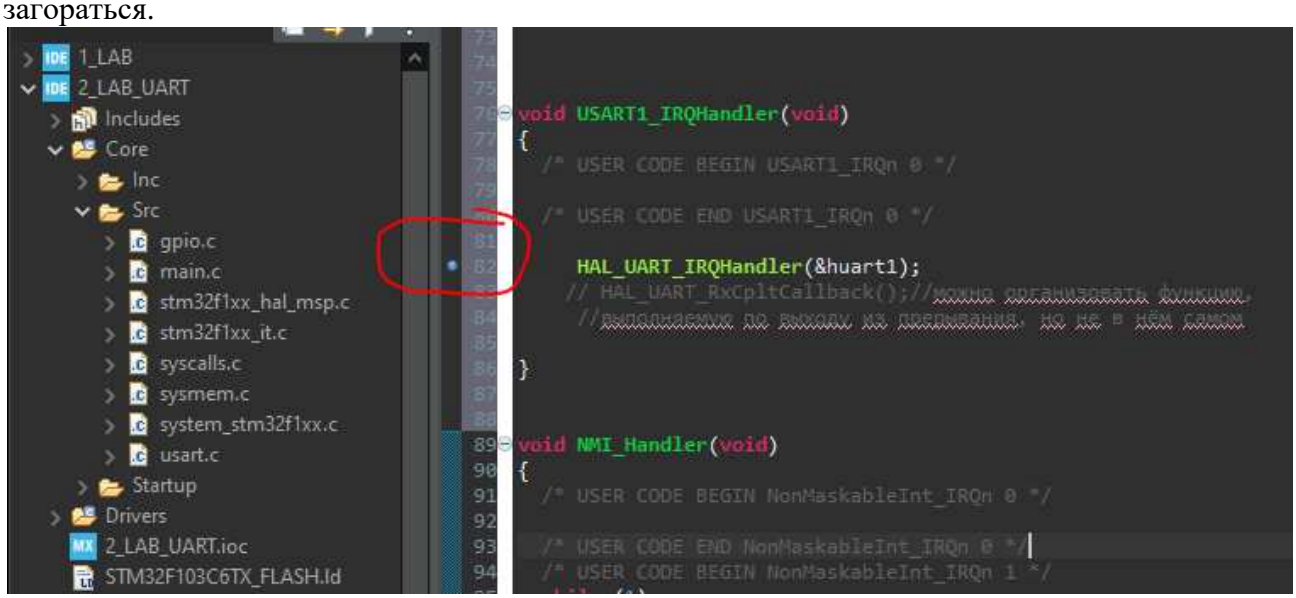

Рисунок 19 - Точка останова для проверки принятого байта.

3. Выполните одно из предложенных заданий для самостоятельной работы для закрепления полученных знаний и навыков.

Варианты заданий:

№	Задание
1.	Выводить в терминал состояние кнопок, подключенных к микропроцессору. Допускается кнопки подключать к произвольным выводам.
2.	Выводить в терминал яркость светодиода в процентах, регулируемую с помощью ШИМ и переменного резистора.
3.	Менять состояние светодиода не по символу «1», а по команде «LED_ON», «LED_OFF».
4.	Провести сопряжение между двумя микропроцессорами, либо микроконтроллерами для передачи данных между ПК. Параметры протокола согласовывать с преподавателем.
5. *	Управлять с помощью нажатия кнопок на плате Arduino светодиодами, подключенными к плате STM32 Blue Pill.
6. *	Организовать кодовый замок с управлением от терминала, подключенного к ПК. Замок представлен в виде сервопривода, пользователю доступна смена пароля. Пароль устанавливается при первом включении и сохраняется во внутренней энергонезависимой памяти устройства
* - задачи повышенной сложности	

4. Составление отчета. В отчет включить схему соединения электронных устройств между собой, выбранное задание, предполагаемый результат выполнения задания, описание работы программы и полученный результат. Обязательно укажите аппаратные модули, используемые микропроцессором и обоснование их применения.

Лабораторная работа № 3.

Прием и передача данных по протоколу SPI.

Цель работы: изучение принципов работы интерфейса SPI а также разработка и реализация системы передачи данных между двумя устройствами с использованием данного интерфейса.

Вводная часть

Последовательный интерфейс периферийных устройств (SPI, Serial Peripheral Interface) представляет собой один из наиболее распространённых протоколов для обмена данными между микроконтроллерами и периферийными устройствами. Понимание работы SPI позволит эффективно интегрировать различные компоненты в свои проекты, обеспечивая надёжный и быстрый обмен информацией. Исследование этого протокола является важным шагом для создания более сложных электронных систем.

Краткие теоретические сведения

SPI (**S**erial **P**eripheral **I**nterface) — это протокол, разработанный для **синхронной** передачи данных между микроконтроллерами и периферийными устройствами, такими как датчики, дисплеи, ячейки памяти и многое другое. Он использует несколько выводов для передачи данных и управления, что позволяет достичь высокой скорости обмена информацией.

Подключение устройств по стандарту SPI включает следующие линии:

MOSI (**M**aster **O**ut **S**lave **I**n) — линия, по которой данные передаются от ведущего устройства (Master) к ведомому (Slave).

MISO (**M**aster **I**n **S**lave **O**ut) — линия, по которой данные передаются от ведомого устройства к ведущему.

SCK (**S**erial **C**lo**ck**) — линия тактового сигнала, генерируемого ведущим устройством для синхронизации передачи данных.

SS, CS (**S**lave **S**elect, **C**hip **S**elect) — линия, используемая для выбора **конкретного** ведомого устройства, с которым будет происходить обмен данными.

Данный интерфейс является **синхронным**, сопрягаемые устройства тактируются частотой **ведущего устройства**.

Передача данных по SPI начинается с того, что устройство **Master** устанавливает **низкий** уровень на линии SS для выбора конкретного ведомого устройства. После этого Master генерирует тактовые импульсы на линии SCK, которые синхронизируют передачу данных. На каждом тактовом импульсе данные передаются по линии MOSI, а принимаемые данные считываются по линии MISO. Процесс передачи продолжается до тех пор, пока не будет завершена передача всех необходимых байтов.

Организация работы выводов приведена на рисунке 20.

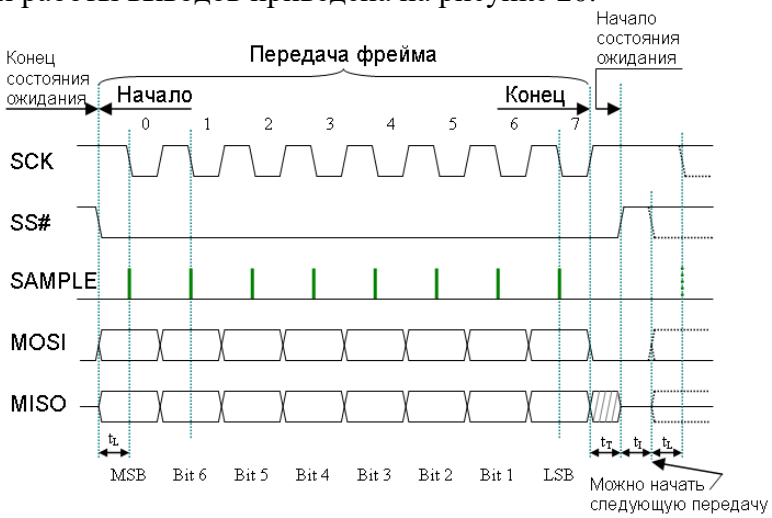

Рисунок 20 - Диаграммы состояний выводов при передаче данных по интерфейсу SPI.

Преимущества SPI:

- Высокая скорость передачи данных, что делает его подходящим для приложений, требующих быстрой скорости обработки информации.
- Простота реализации, так как протокол не требует сложных алгоритмов управления.
- Возможность подключения нескольких ведомых устройств к одному ведущем и возможность обращаться к каждому на прямую.

Однако у SPI есть и **недостатки**:

- Необходимость использования дополнительных линий для подключения нескольких ведомых устройств, что может усложнить схем при большом количестве ведомых устройств.
- Отсутствие встроенной поддержки для обнаружения ошибок, что требует дополнительных мер для обеспечения надежности передачи.

Из критериев успешного сопряжения можно выделить допустимую частоту синхронизации ведомых устройств, очередность отправки старшего разряда – MSB first или LSB first. Пример побитовой передачи данных приведен на рисунке 21. Промежуточные данных хранятся в буфере, и вычитывать их из буфера нужно только в тот момент, когда контроллер выдаст команду о целостности полученных данных. Микроконтроллер обычно в таких случаях вызывает прерывание с соответствующим вектором, пользователю остается только написать инструкции для обработчика прерываний.

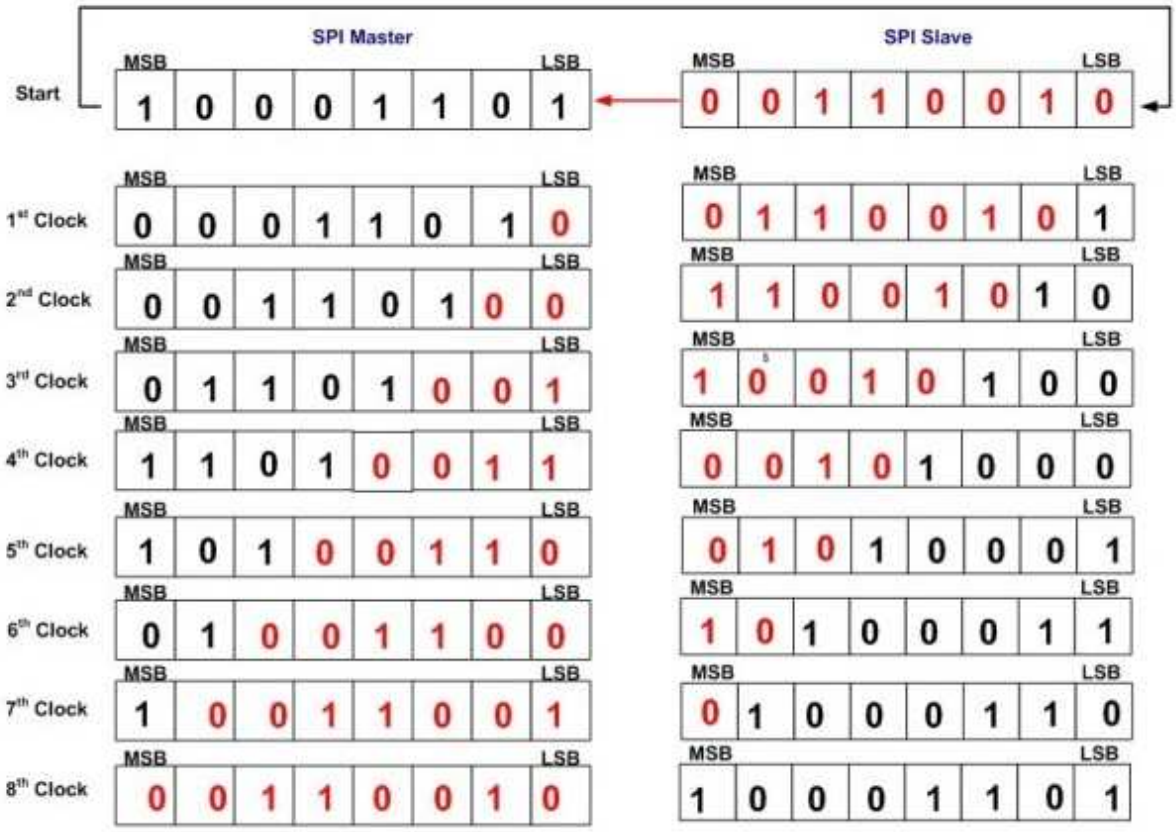

Рисунок 21 - Побитовая передача данных протокола SPI.

Схема подключения нескольких SPI устройств в независимом режиме приведена на рисунке 22.

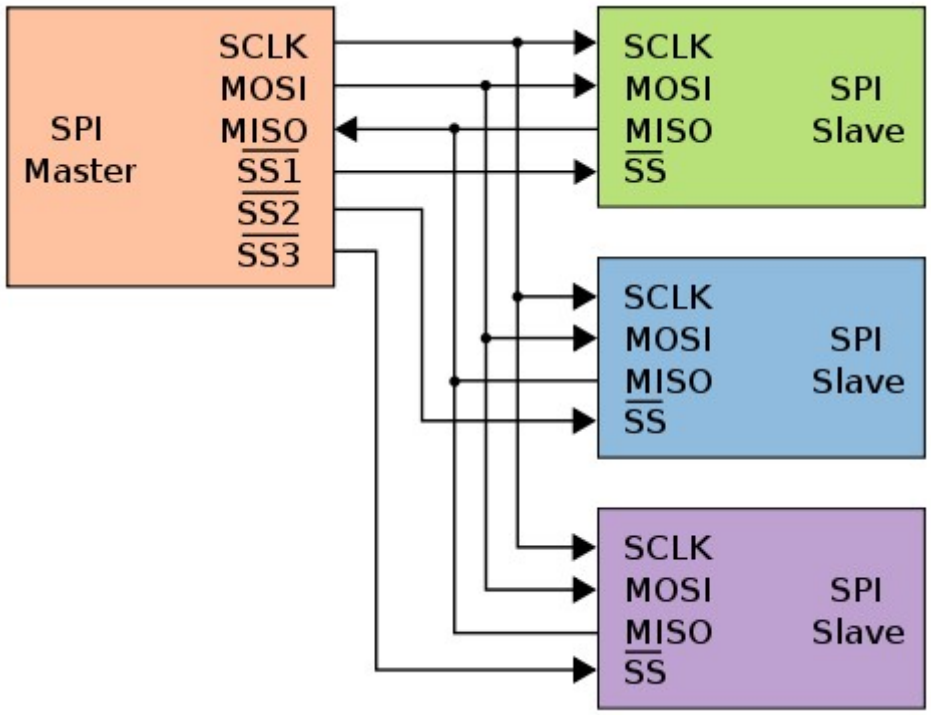

Рисунок 22 - Схема подключения нескольких SPI устройств в независимом режиме.

Данная лабораторная работа посвящена работе с устройством MAX7219. Это простейший сдвиговый регистр с управлением по интерфейсу SPI.

1. Создание проекта.

1.1. Создайте новый проект на основе созданной ранее конфигурации согласно рисунку 23.

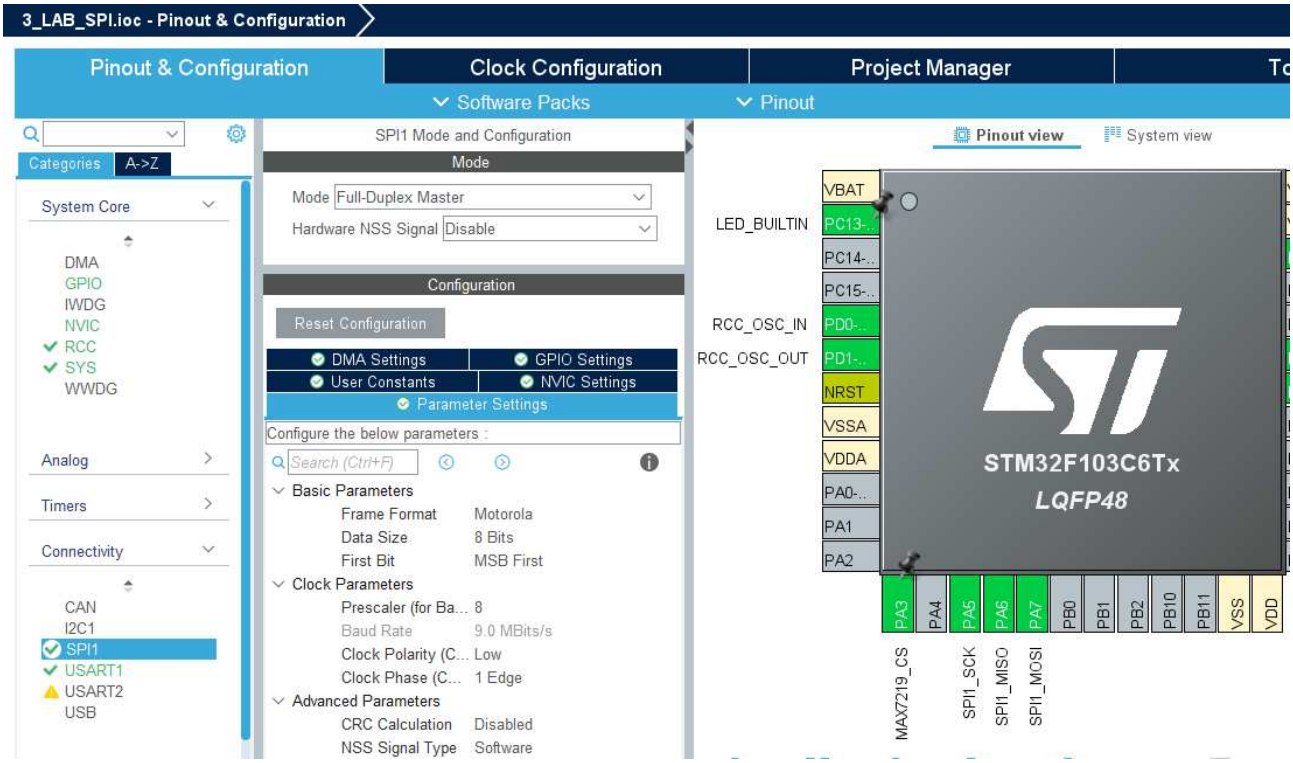

Рисунок 23 - Конфигурация выводов SPI

1.2. Разрешите прерывание SPI согласно рисунку 24

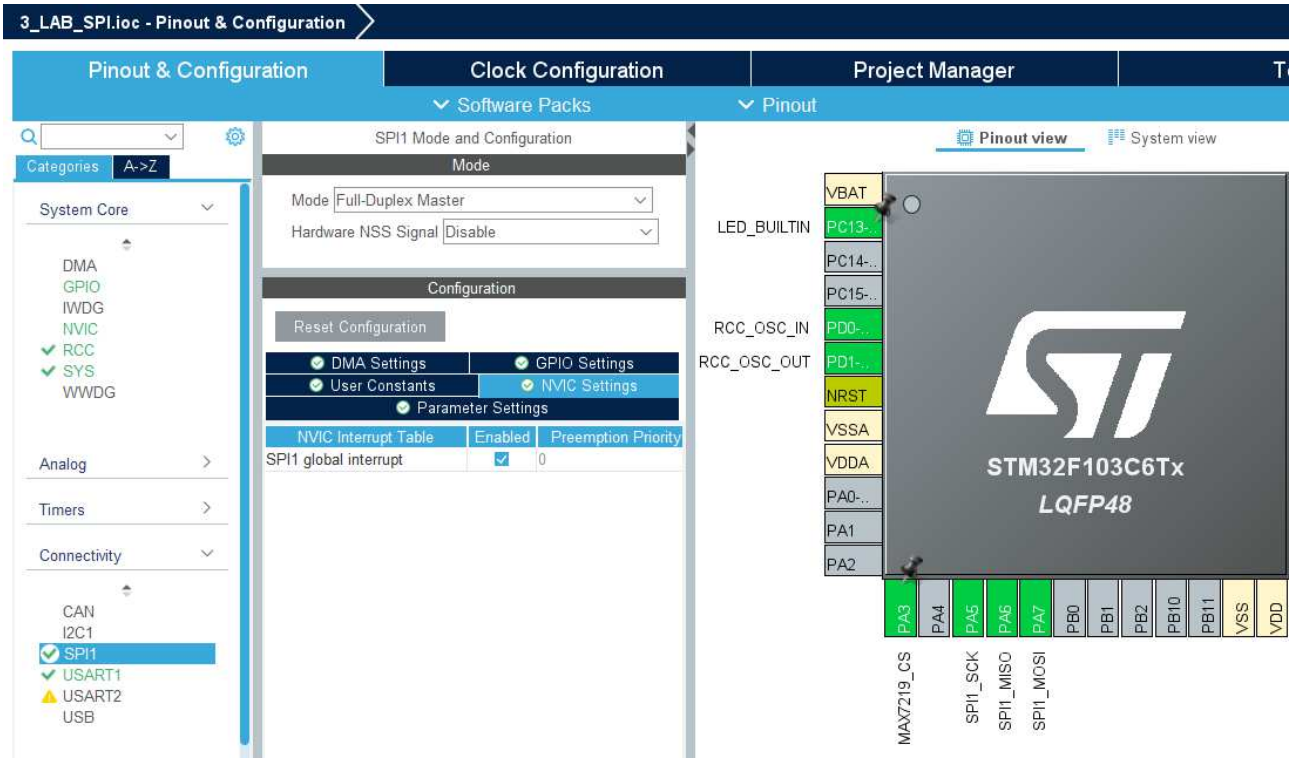

Рисунок 24 - Настройка прерываний по SPI.

1.3. Дополнительно установите ножку PA3 в режим выхода согласно рисунку 25. Данный шаг нужен, что бы управлять линией SS ведомого устройств. Имя вывода - **MAX7219_CS**.

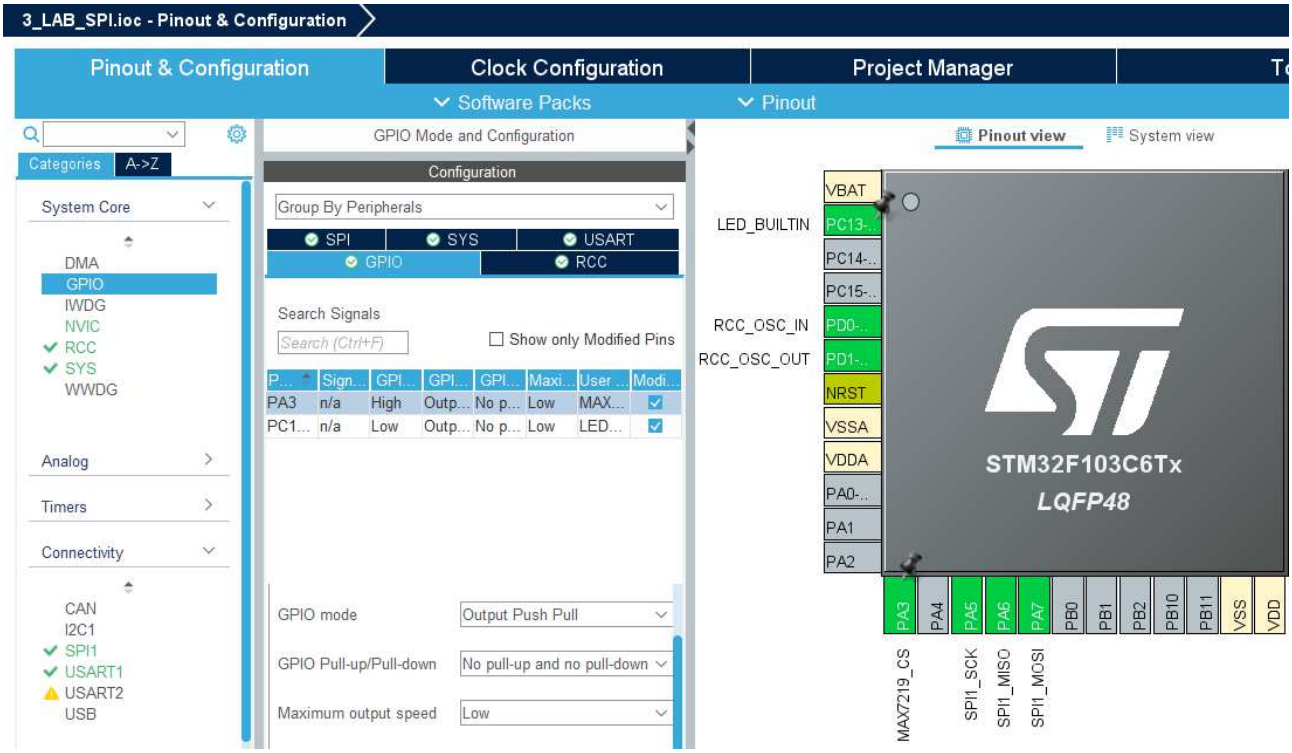

Рисунок 25 - Настройка управляющего вывода CS SPI.

2. Создание кода программы и отладка.

2.1. Обозначим адреса регистров управления микросхемы MAX7219 согласно листу данных на устройство.

```
/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */

typedef enum {
   REG_NO_OP          = 0x00,
   REG_DIGIT_0        = 0x01,
   REG_DIGIT_1        = 0x02,
   REG_DIGIT_2        = 0x03,
   REG_DIGIT_3        = 0x04,
   REG_DIGIT_4        = 0x05,
   REG_DIGIT_5        = 0x06,
   REG_DIGIT_6        = 0x07,
   REG_DIGIT_7        = 0x08,
   REG_DECODE_MODE    = 0x09,
   REG_INTENSITY      = 0x0A,
   REG_SCAN_LIMIT     = 0x0B,
   REG_SHUTDOWN       = 0x0C,
   REG_DISPLAY_TEST   = 0x0F,
} MAX7219_REGISTERS;

/* USER CODE END Includes */
```

REGISTER	ADDRESS					HEX CODE
	D15–D12	D11	D10	D9	D8	
No-Op	X	0	0	0	0	0xX0
Digit 0	X	0	0	0	1	0xX1
Digit 1	X	0	0	1	0	0xX2
Digit 2	X	0	0	1	1	0xX3
Digit 3	X	0	1	0	0	0xX4
Digit 4	X	0	1	0	1	0xX5
Digit 5	X	0	1	1	0	0xX6
Digit 6	X	0	1	1	1	0xX7
Digit 7	X	1	0	0	0	0xX8
Decode Mode	X	1	0	0	1	0xX9
Intensity	X	1	0	1	0	0xXA
Scan Limit	X	1	0	1	1	0xXB
Shutdown	X	1	1	0	0	0xXC
Display Test	X	1	1	1	1	0xXF

ВНИМАНИЕ!!! Вновь вводимые строки в проект обязательно ставьте в допустимые области, например между воротами, т.к. это **защищенная от затирания область** кода при обновлении проекта через конфигуратор.

```
  /* USER CODE BEGIN 1 */
//место для вашего кода
  /* USER CODE END 1 */
```

2.2. Вставьте функцию отправки команды устройству вне основного тела программы

```
void max7219_SendData(uint8_t addr, uint8_t data)
{
   HAL_GPIO_WritePin(MAX7219_CS_GPIO_Port, MAX7219_CS_Pin, GPIO_PIN_RESET);

   HAL_SPI_Transmit_IT(&hspi1,&addr, 1);
   HAL_SPI_Transmit_IT(&hspi1, &data, 1);

   HAL_GPIO_WritePin(MAX7219_CS_GPIO_Port, MAX7219_CS_Pin, GPIO_PIN_SET);
}
```

Разберем эту функцию подробнее

HAL_GPIO_WritePin(MAX7219_CS_GPIO_Port, MAX7219_CS_Pin, GPIO_PIN_SET);»

Данная строка устанавливает вывод CS в «0» для начала передачи данных ведомому устройству.

HAL_SPI_Transmit_IT(&hspi1, &addr, 1);

Данная строка является указателем абстрактной функции для передачи данных по SPI через прерывание.

Вот краткое объяснение каждого из аргументов в данном случае:

&hspi1: Это указатель на структуру, содержащую параметры SPI. Здесь & обозначает, что мы передаем адрес переменной hspi1, что делает его указателем на эту переменную. В данном случае пользуем SPI1, в некоторых процессорах данных интерфейсов может быть и больше, поэтому HAL функция, написанная для общей линейки микропроцессоров предусматривает обращение именно через адрес.

&addr: Это указатель на переменную addr, которая содержит данные для передачи. В данном случае мы указываем адрес регистра, в который хотим записать данные.

1: Это целочисленный аргумент, указывающий количество байт, которые нужно передать. В данном случае это 1 байт.

В функции «HAL_SPI_Transmit_IT(&hspi1, &**data**, 1);» после передачи адреса регистра передается данные, которые мы хотим в него записать. Протокол передачи данных указан в технических данных на микросхему MAX7219 согласно рисунку 26.

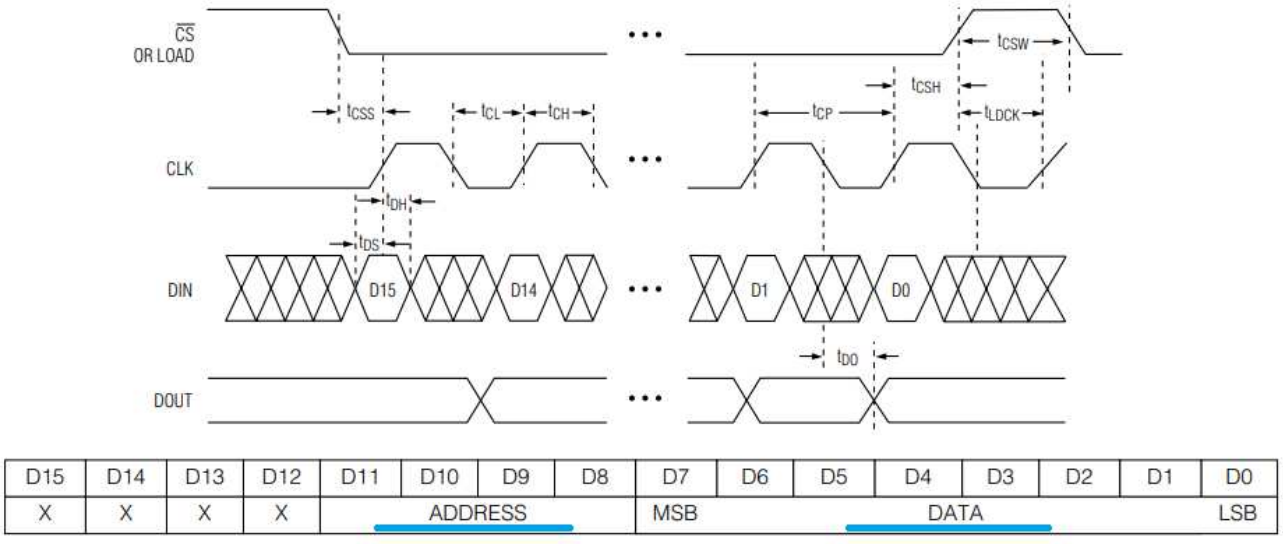

Рисунок 26 - Последовательность передачи данных в протоколе SPI

HAL_GPIO_WritePin(MAX7219_CS_GPIO_Port, MAX7219_CS_Pin, GPIO_PIN_SET);

Отпускаем CS и освобождаем шину SPI.

2.3. Добавьте вставьте функцию инициализации устройства

```
/* USER CODE BEGIN 0 */
uint8_t MAX7219_INIT()

{
   max7219_SendData(REG_SCAN_LIMIT, 0x07);  // limit off
   max7219_SendData(REG_DECODE_MODE, 0x00); // decode off
   max7219_SendData(REG_SHUTDOWN, 0x01);
}
/* USER CODE END 0 */
```

Согласно техническим данным микросхема MAX7219, как и многие периферийные устройства по умолчанию отключены.

Initial Power-Up

*On initial power-up, all control **registers are reset**, the display is **blanked**, and the MAX7219/MAX7221 **enter shutdown mode**. **Program** the display driver prior **to** display **use**. Otherwise, it will initially be set to scan one digit, it will not decode data in the data registers, and the intensity register will be set to its minimum value.*

Данные для выбора режима работы приведены на рисунке 27.

MODE	ADDRESS CODE (HEX)	REGISTER DATA							
		D7	D6	D5	D4	D3	D2	D1	D0
Shutdown Mode	0xXC	X	X	X	X	X	X	X	0
Normal Operation	0xXC	X	X	X	X	X	X	X	1

Рисунок 27 - Данные для выбора режима работы MAX7219.

В основное тело программы на этапе инициализации введем функцию инициализации устройства и очистки дисплея

```
/* USER CODE BEGIN 2 */
MAX7219_INIT();
for (int i = 1; i < 9; i++)
   max7219_SendData(i, 0x00);
/* USER CODE END 2 */
```

С помощью цикла в данном случае мы принудительно отключаем все светодиоды, отправляя в регистры с адресами 0x01…0x08 байт 0x00.

2.4. Основной код программы:

```
/* USER CODE BEGIN WHILE */
while (1)
{       max7219_SendData(0x01, 0b00000001);
        HAL_Delay(250);
        max7219_SendData(0x02, 0b00000010);
        HAL_Delay(250);
        max7219_SendData(0x03, 0b00000100);
        HAL_Delay(250);
        max7219_SendData(0x04, 0b00001000);
        HAL_Delay(250);
```

```
        max7219_SendData(0x05, 0b00010000);
        HAL_Delay(250);
        max7219_SendData(0x06, 0b00100000);
        HAL_Delay(250);
        max7219_SendData(0x07, 0b01000000);
        HAL_Delay(250);
        max7219_SendData(0x08, 0b10000000);
        HAL_Delay(250);
        HAL_GPIO_TogglePin(LED_BUILTIN_GPIO_Port, LED_BUILTIN_Pin);
        for (int i = 1; i < 9; i++)       //clear display
        max7219_SendData(i, 0x00);
        HAL_Delay(250);
    /* USER CODE END WHILE */
```

Здесь по очереди отправляется содержимое Data в каждую строку. MAX7219 запоминает полученные данные в каждом из своих регистров и выводит их на светодиодную матрицу. Номер строки соответствует адресу каждого регистра.

2.5. Подключите матрицу светодиодов согласно рисункам 22 и 23. Запустите программу в отладочном режиме и убедитесь что светодиодная матрица меняет свое состояние согласно написанному коду.

3. Выполните одно из предложенных заданий для самостоятельной работы для закрепления базовых навыков.

Варианты заданий:

№	Задание
1.	Счётчик нажатия кнопок: Реализовать систему, которая будет отображать счётчик нажатия двух кнопок на двух матрицах. При нажатии первой кнопки первый счётчик увеличивается, при нажатии второй счетчик увеличивается. Счётчик должен отображаться в диапазоне от 0 до 9, и когда он достигает 9, при следующем нажатии он должен возвращаться к 0.
2.	Простейшая анимация: Создать анимацию "бегающего огонька" на нескольких матрицах. Например, можно сделать так, чтобы один пиксель перемещался по матрицам из левого верхнего угла в правый нижний, а затем возвращался обратно.
3.	Бегущая строка через UART: Организовать бегущую строку символов, принимаемых через терминал по протоколу UART. Пользователь может вводить текст, который будет отображаться на матрицах. Можно добавить возможность прокрутки текста, если он превышает длину матриц.
4.	Часы: Создать простые цифровые часы, которые будут отображать текущее время на матрицах. Можно использовать модуль RTC (Real Time Clock) для получения точного времени и обновления отображения каждую секунду.
5.	Игра "Угадай число": Создать игру, в которой пользователь должен угадать случайно сгенерированное число от 0 до 9. На матрицах будет отображаться количество попыток, а при правильном ответе — анимация празднования.

4. Составление отчета. В отчет включить схему соединения электронных устройств между собой, выбранное задание, предполагаемый результат выполнения задания, описание работы программы и полученный результат. Обязательно укажите аппаратные модули, используемые микропроцессором и обоснование их применения.

Лабораторная работа № 4.

Прием и передача данных по протоколу I2C.

Цель работы: изучение принципов работы интерфейса I2C, а также разработка и реализация системы передачи данных между двумя устройствами с использованием данного интерфейса

Вводная часть

I2C (Inter-Integrated Circuit) — это широко используемый протокол для обмена данными между микроконтроллерами и различными периферийными устройствами. Он был разработан для упрощения связи между компонентами, позволяя подключать до 127 устройств. Этот протокол имеет широкое распространение и отличается простотой в использовании, но более сложным процессом освоения принципа обмена данными.

Краткие теоретические сведения

I2C (IIC - Inter-Integrated Circuit) — это протокол, разработанный для синхронной передачи данных между устройствами. Он использует только **2** вывода, что делает его простым в реализации и эффективным для подключения сразу нескольких устройств.

Подключение устройств по стандарту I2C включает следующие линии:

SDA (Serial Data Line) — линия, по которой передаются данные между устройствами.

SCL (Serial Clock Line) — линия тактового сигнала, генерируемого ведущим устройством для синхронизации передачи данных.

Передача данных по I2C начинается с того, что устройство Master (ведущее устройство) отправляет стартовый сигнал (**START**), после чего оно передает адрес Slave (ведомого) устройства. Если ведомое устройство распознает свой адрес, оно отвечает подтверждением (**ACK**). Далее Master отправляет данные по линии **SDA**, синхронизируя передачу с помощью сигналов на линии SCL. Ведомое устройство может отправить подтверждение после получения каждого байта данных.

Важным моментом является отправка команды на чтение/запись от ведомого устройства. Адрес ведомого устройства обычно содержит в себе последние 7 бит передаваемого байта, а первым битом Master обозначает ведомому устроству будет ли это команда на чтение или запись

Процесс передачи продолжается до тех пор, пока не будет завершена передача всех необходимых байтов, после чего Master отправляет сигнал остановки, завершая обмен данными (STOP). Диаграммы передачи адреса и данных приведены на рисунке 28.

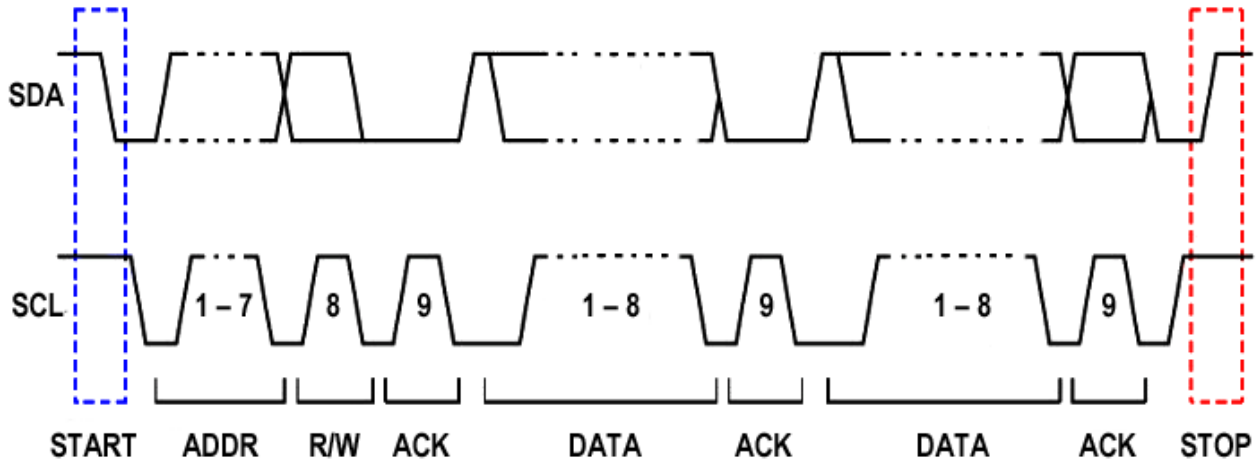

Рисунок 28 - Диаграммы передачи адреса и данных по протоколу I2C.

Каждое ведомое устройство, подключенное к шине I2C считывает адрес, и если адрес совпадает, то отправляет обратно ACK. Те устройства, что не отправили бит подтверждения отключаются от шины и не мешают передачи данных между ведущим и ведомым устройством. Таким образом можно подключить несколько устройств с разными адресами всего лишь по двум линиям. Структурная схема подключения устройств в шину приведена на рисунке 29.

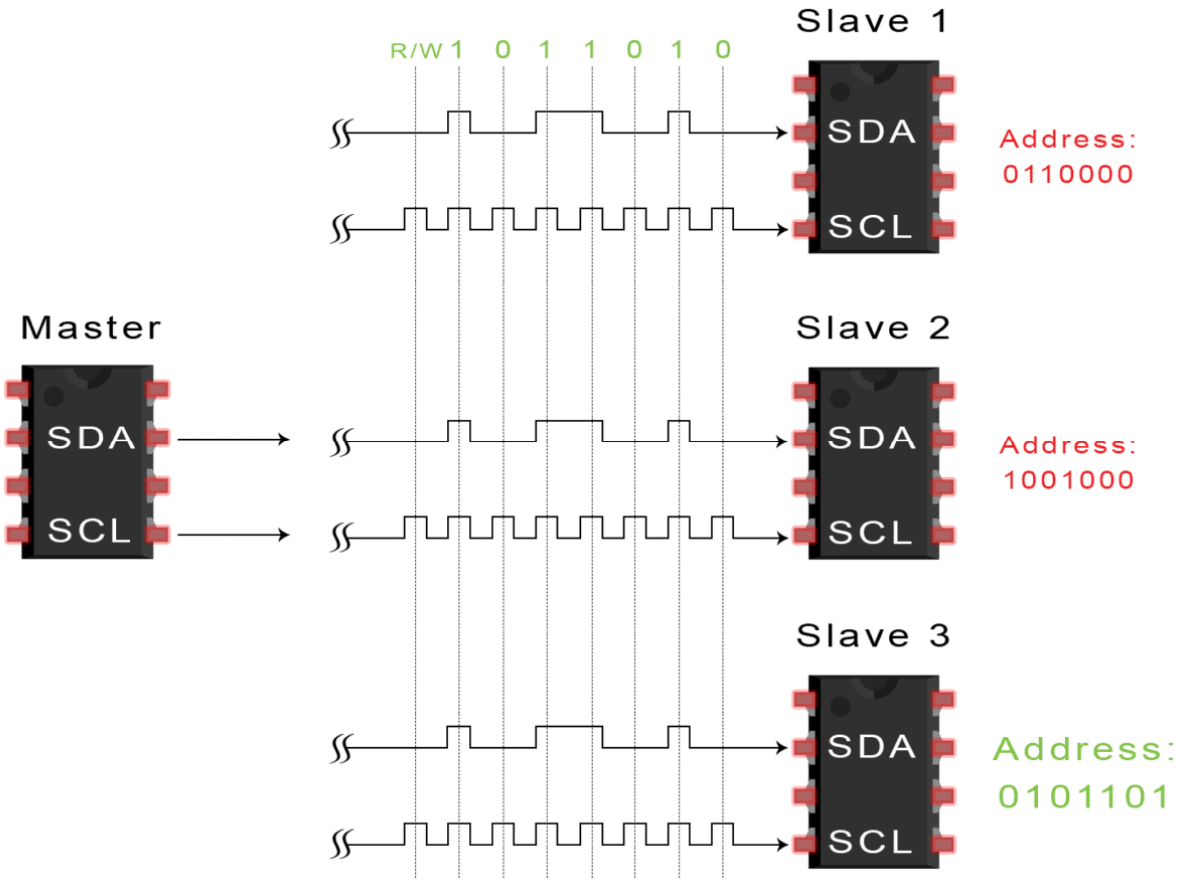

Рисунок 29 - Структурная схема подключения устройств в шину I2C.

После того как мастер принял бит подтверждения от ведомого устройства, он может считать с него данные. Для этого он должен обратиться к устройству с заданным адресом **повторно, указав его адрес,** но отправив бит чтения/записи «1». После этого нужно указать адрес регистра из которого мы хотим считать данные. Если нужно считать несколько байт, то в конце нужного принятого байта мастер отправляет бит неподтверждения принятых данных (NACK), что означает конец передачи пакета. Протокол чтения пакета данных приведен на рисунке 30.

Burst Read Sequence

Master	S	AD+W		RA		S	AD+R			ACK		NACK	P
Slave			ACK		ACK			ACK	DATA		DATA		

Рисунок 30 - Последовательность чтения нескольких байт подряд по протоколу I2C.

Преимущества I2C:

- Для передачи и приема данных от всех устройств в шине нужно только 2 линии.
- Адресное обращение позволяет подключать по стандартному протоколу до 128 устройств. Однако и это число можно увеличить при использовании ведомых устройств с 10-битным адресом.

Недостатки:

- Относительно медленный протокол для передачи данных, т.к. мы передаем не только информацию, но и адрес устройства.
- Сложный по сравнению с SPI протокол обмена данными

Из критериев успешного сопряжения так же можно выделить допустимую частоту синхронизации ведомых устройств и очередность отправки старшего разряда – MSB first или LSB first.

Данная лабораторная работа посвящена работе с устройством MPU6050. Это трехуровневый гироскоп-акселерометр, работающий по шине I2C.

1. Создание проекта.

1.1. Создайте новый проект на основе созданной ранее конфигурации согласно рисунку 31.

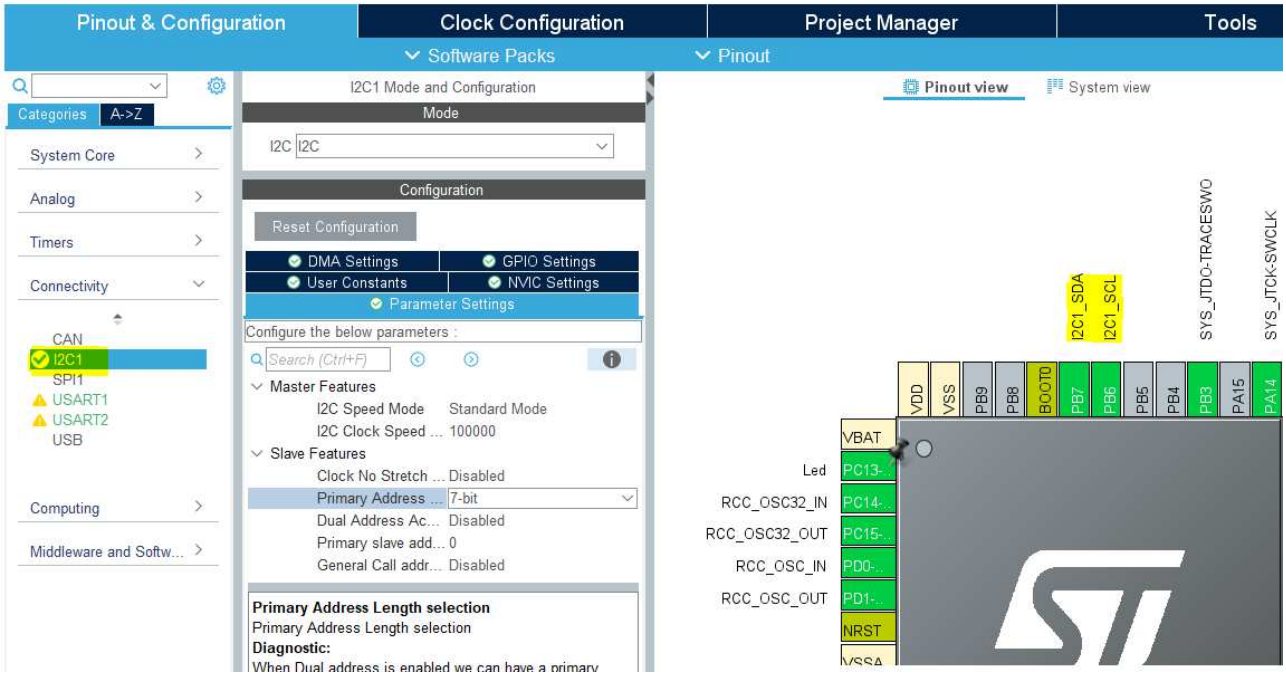

Рисунок 31 - Конфигурация I2C.

Так же включите прерывания для допустимых событий согласно рисунку 32.

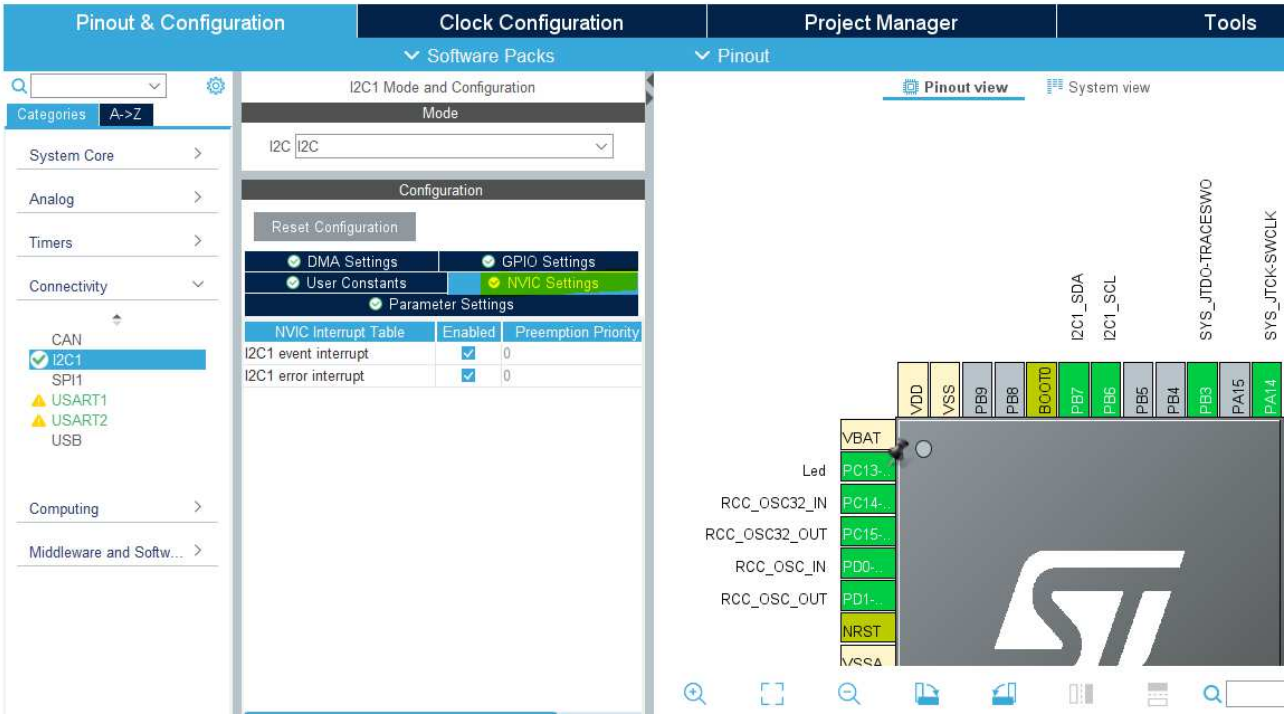

Рисунок 32 - Конфигурация прерываний I2C.

1.2. Для обозначения адресов каждого регистра управления микросхемы MPU6050 подключите файл «MPU6050.h» в файле main.c согласно рисунку 33.

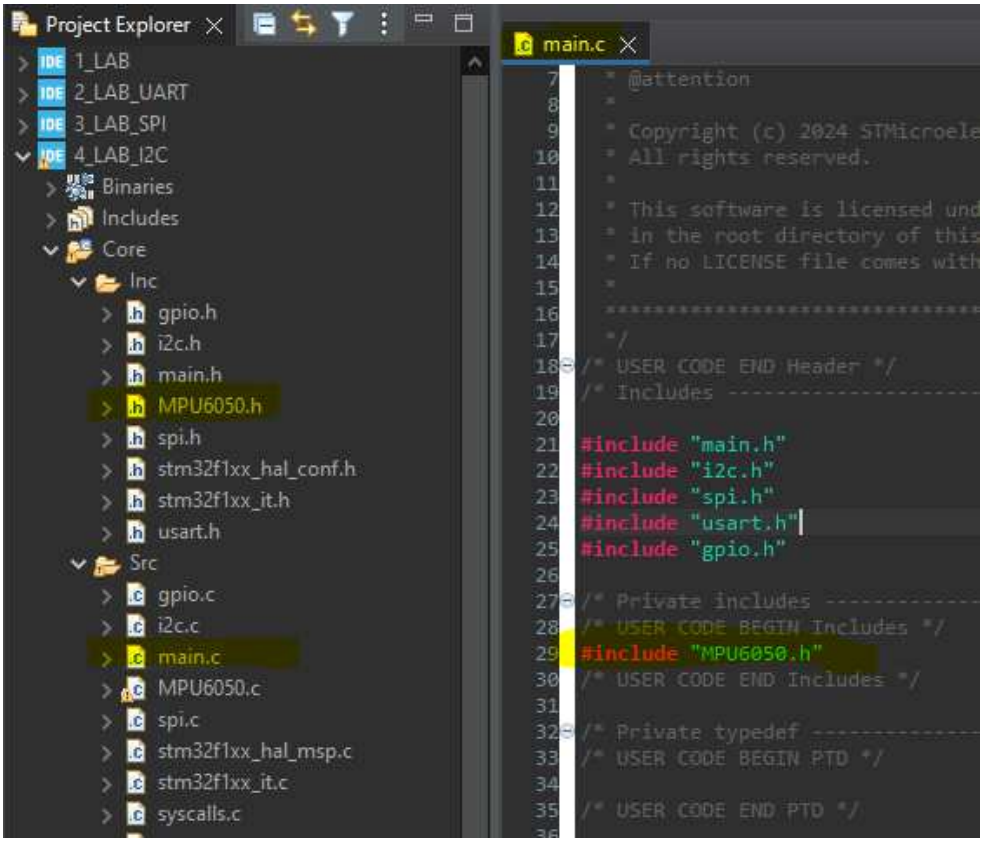

Рисунок 33 - Расположение и подключение в проект файла «MPU6050.h».

2. Создание кода программы и отладка.

2.1. Добавьте в код основного тела программы определение переменных, необходимых для приема данных

```
int main(void)
{
 /* USER CODE BEGIN 1 */
  int16_t Accel_X_RAW;     //данные с акселерометра по оси X
  int16_t Accel_Y_RAW;      //--//--
  int16_t Accel_Z_RAW;     //--//--

  int16_t Gyro_X_RAW;      //данные с гироскопа по оси X
  int16_t Gyro_Y_RAW;      //--//--
  int16_t Gyro_Z_RAW;      //--//--

  uint8_t Rec_Data[14];        //буфер хранения сырых данных
 /* USER CODE END 1 */
```

2.2. Далее нам нужно убедиться, что MPU6050 подключен к шине I2C. для этого достаточно считать байт данных из его регистра «WHO_AM_I». Описание регистра представлено на рисунке 34.

4.32 Register 117 – Who Am I
WHO_AM_I

Type: Read Only

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
75	117	-	WHO_AM_I[6:1]						-

Description:

This register is used to verify the identity of the device. The contents of *WHO_AM_I* are the upper 6 bits of the MPU-60X0's 7-bit I^2C address. The least significant bit of the MPU-60X0's I^2C address is determined by the value of the AD0 pin. The value of the AD0 pin is not reflected in this register.

The default value of the register is 0x68.

Bits 0 and 7 are reserved. (Hard coded to 0)

Parameters:

WHO_AM_I Contains the 6-bit I^2C address of the MPU-60X0.

The Power-On-Reset value of Bit6:Bit1 is 110 100.

Рисунок 34 - Описание регистра WHO_AM_I MPU6050.

2.3. Временно добавьте функцию чтения этого байта в основном цикле программы

```
 /* USER CODE BEGIN WHILE */
 while (1)
 {
HAL_I2C_Mem_Read_IT (&hi2c1, MPU6050_ADDRESS_AD0_LOW,
MPU6050_RA_WHO_AM_I, 1, &check, 1);
  /* USER CODE END WHILE */
```

В данном случае микроконтроллера дается инструкция для чтения от устройства с адресом **MPU6050_ADDRESS_AD0_LOW** содержимое регистра с адресом **MPU6050_RA_WHO_AM_I** в количестве **одного** байта и с использованием прерывания, т.е. программа задержится только на выдачу команды для отправки байта и продолжит выполнять свои дальнейшие инструкции. Прием байта осуществиться в обработчике прерываний.

2.4. Зайдите в режим Debug, установите точку останова согласно рисунку, и убедитесь, что мы принимаем байт с содержимым 0x68 согласно рисунку .

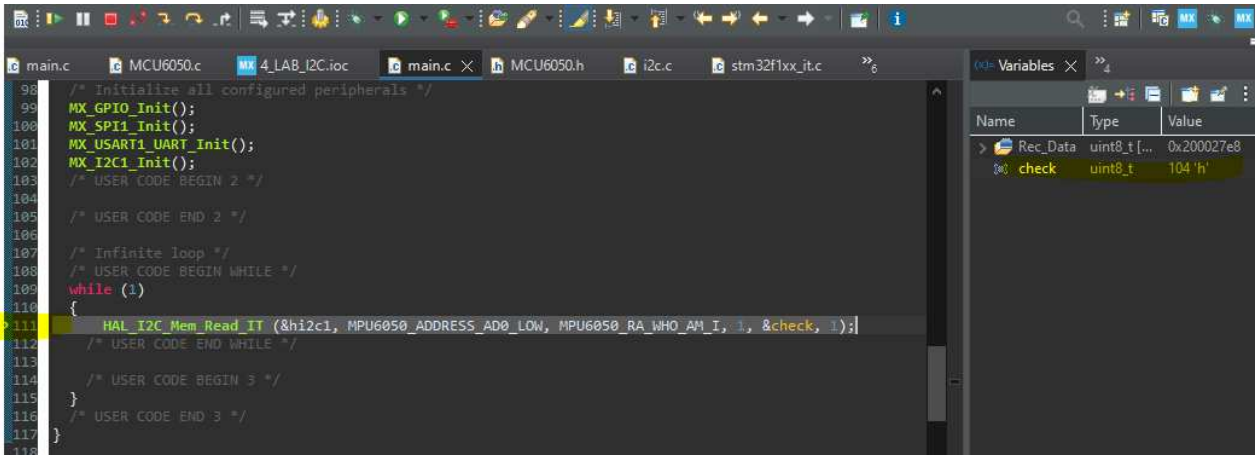

Рисунок 35 - Пример просмотра содержимого переменных в отладочном режиме.

2.5. Далее необходимо разрешить работу гироскопа. Для этого, согласно описанию регистров устройства необходимо сделать следующее:

- Согласно техническим данным на MPU6050 по умолчанию он находится в спящем режиме. Для того что бы разрешить работу в нормальном режиме достаточно обнулить бит SLEEP регистра PWR_MGMT_1;
- Так же рекомендуется установить конфигурацию частоты сэмплирования данных и диапазон измерений акселерометра и гироскопа.

Добавим функцию инициализации в файл MPU6050.c

```
#include "MPU6050.h"

#include "main.h"
#include "i2c.h"
#include "usart.h"
#include "gpio.h"


        void MPU6050_Init(void)
        {
        uint8_t check=0;
        uint8_t Data = 0;

        HAL_I2C_Mem_Read(&hi2c1,MPU6050_ADDRESS_AD0_LOW,
MPU6050_RA_WHO_AM_I,1,&check,1,1000);
```

```
if (check == 104)// если считали данные из регистра
            {
// power management register 0X6B we should write all 0's to wake the sensor up
Data = 0;
HAL_I2C_Mem_Write(&hi2c1, MPU6050_ADDRESS_AD0_LOW,
MPU6050_RA_PWR_MGMT_1, 1,&Data, 1, 1000);
// Set DATA RATE of 1KHz by writing SMPLRT_DIV register
Data = 0x07;
HAL_I2C_Mem_Write(&hi2c1, MPU6050_ADDRESS_AD0_LOW,
MPU6050_RA_SMPLRT_DIV, 1, &Data, 1, 1000);

// Set accelerometer configuration in ACCEL_CONFIG Register
// XA_ST=0,YA_ST=0,ZA_ST=0, FS_SEL=0 -> +- 2g
Data = 0x00;
HAL_I2C_Mem_Write(&hi2c1, MPU6050_ADDRESS_AD0_LOW,
MPU6050_RA_ACCEL_CONFIG, 1, &Data, 1, 1000);

// Set Gyroscopic configuration in GYRO_CONFIG Register
// XG_ST=0,YG_ST=0,ZG_ST=0, FS_SEL=0 -> +- 250 +-/s
Data = 0x00;
HAL_I2C_Mem_Write(&hi2c1, MPU6050_ADDRESS_AD0_LOW,
MPU6050_RA_GYRO_CONFIG, 1, &Data, 1, 1000);

uint8_t str[30]={0};//инициализируем буфер-массив

sprintf(str,"MPU6050 is ready!\r\n");//преобразуем массив в строку
HAL_UART_Transmit(&huart1, str, 18, 100);//выводим строку по UART
}
else
{
uint8_t str[30]={0};
sprintf(str,"We have a MPU6050 problem..\r\n");
HAL_UART_Transmit(&huart1, str, 27, 100);

            }
}
```

2.6. Добавьте обращение к функции инициализации до основного цикла программы и строки чтения данных из регистров хранения измерений:

```
/* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_SPI1_Init();
  MX_USART1_UART_Init();
  MX_I2C1_Init();
  /* USER CODE BEGIN 2 */
  MPU6050_Init();
  /* USER CODE END 2 */
```

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

            HAL_I2C_Mem_Read(&hi2c1,MPU6050_ADDRESS_AD0_LOW,
MPU6050_RA_ACCEL_XOUT_H, 1, Rec_Data, 6, 1000);

Accel_X_RAW = (int16_t)(Rec_Data[0] << 8 | Rec_Data [1]);
Accel_Y_RAW = (int16_t)(Rec_Data[2] << 8 | Rec_Data [3]);
Accel_Z_RAW = (int16_t)(Rec_Data[4] << 8 | Rec_Data [5]);


HAL_I2C_Mem_Read(&hi2c1,MPU6050_ADDRESS_AD0_LOW,
MPU6050_RA_GYRO_XOUT_H, 1, Rec_Data, 6, 1000);

Gyro_X_RAW = (int16_t)(Rec_Data[0] << 8 | Rec_Data [1]);
Gyro_Y_RAW = (int16_t)(Rec_Data[2] << 8 | Rec_Data [3]);
Gyro_Z_RAW = (int16_t)(Rec_Data[4] << 8 | Rec_Data [5]);

    /* USER CODE END WHILE */

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

2.7. В режиме debug, поставив точку останова в основном цикле программы убедитесь, что микропроцессор процессор считывает нужные значения измерений согласно рисунку 36

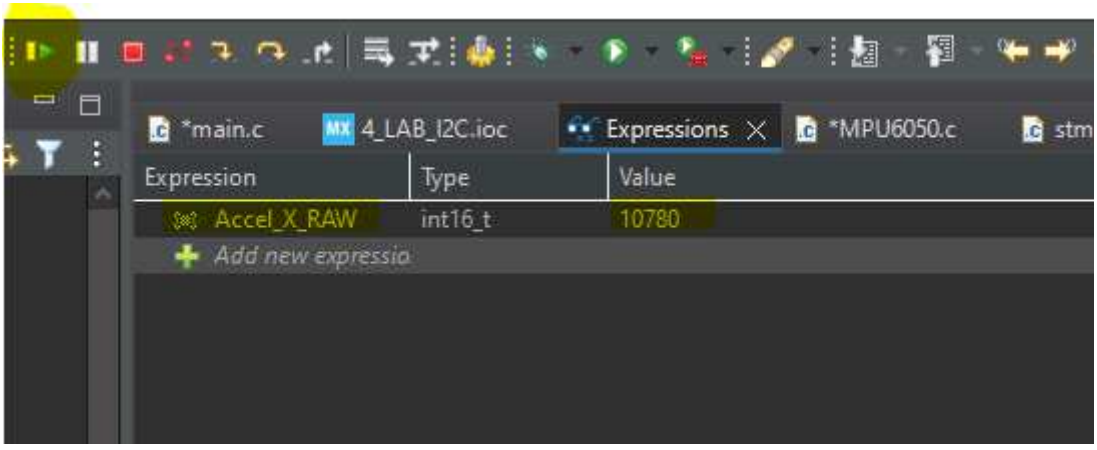

Рисунок 36 - Пример отображения переменных в реальном времени.

3. Выполните одно из предложенных заданий для самостоятельной работы для закрепления базовых навыков.

Варианты заданий:

№	Задание
1.	Создайте систему, которая будет отображать угол наклона устройства на матрицах. Используйте данные акселерометра для определения угла наклона по осям X и Y. На матрицах можно отображать угол наклона в диапазоне от -90° до +90° с помощью графической визуализации (например, стрелка или индикатор).
2.	Реализуйте систему, которая будет отслеживать движение устройства. Если устройство наклоняется или перемещается, на матрицах будет отображаться анимация (например, бегущая точка или мигающий индикатор), сигнализирующая о том, что движение было зафиксировано.
3.	Реализуйте систему, которая будет отслеживать движение устройства. Если устройство наклоняется или перемещается, на матрицах будет отображаться анимация (например, бегущая точка или мигающий индикатор), сигнализирующая о том, что движение было зафиксировано.
4.	Создайте игру, в которой пользователь должен угадать направление наклона устройства. На матрицах будет отображаться случайное направление (вверх, вниз, влево, вправо), и пользователь должен наклонить устройство в указанном направлении. При правильном ответе — анимация празднования.
5.	Реализуйте систему, которая будет отслеживать активность пользователя (например, сидит или стоит) на основе данных акселерометра. На матрицах можно отображать статус активности (сидит/стоит) с соответствующей анимацией.

4. Составление отчета. В отчет включить схему соединения электронных устройств между собой, выбранное задание, предполагаемый результат выполнения задания, описание работы программы и полученный результат. Обязательно укажите аппаратные модули, используемые микропроцессором и обоснование их применения.

Список рекомендуемой литературы

1. Ключарёв, А.А., Кочин, К.А., Фоменкова, А.А. Программирование микроконтроллеров STM32: учеб. пособие / А.А. Ключарёв, К.А. Кочин, А.А. Фоменкова. - Санкт-Петербург: Санкт-Петербургский государственный университет аэрокосмического приборостроения, 2023. - 196 с.

2. Иоффе, В.Г., Графкин, А.В., Графкин, В.В. Архитектура, принципы функционирования и программные средства микроконтроллеров STM32 / В.Г. Иоффе, А.В. Графкин, В.В. Графкин. - Самара: Самарский национальный исследовательский университет имени академика С. П. Королёва, 2021. - 490 с.

3. Конченков, В.И., Скакунов, В.Н. Семейство микроконтроллеров STM32. Программирование и применение: учебное пособие / В.И. Конченков, В.Н. Скакунов. - Волгоград: Волгоградский государственный технический университет, 2015. - 78 с.

4. Люлява, Д.В., Дуксин, Н.А., Тарасов, И.Е. Работа с отладочной платой EasyMX PRO v7 FOR STM32. Часть 1: лабораторный практикум / Д.В. Люлява, Н.А. Дуксин, И.Е. Тарасов. - Москва: МИРЭА - Российский технологический университет, 2023. - 169 с.

5. Морохин, Д.В., Мясников, В.И., Иванов, А.В. Микропроцессорные системы на основе микроконтроллеров STM32: лабораторный практикум / Д.В. Морохин, В.И. Мясников, А.В. Иванов. - Тольятти: Поволжский государственный технологический университет, 2023. - 112 с.