Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ

КАФЕДРА КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.04.01 Информатика и вычислительная техника**

МАГИСТЕРСКАЯ ПРОГРАММА **09.04.01/05 Современные интеллектуальные программно-аппаратные комплексы.**

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ

НА ТЕМУ:

Исследование средств разработки программного обеспечения для микроконтроллеров WCH

Студент ИУ6-21М — 27.09.2024 — И.С. Марчук
(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель — 27.09.2024 — С.В. Ибрагимов
(Подпись, дата) (И.О.Фамилия)

Н.К. 27.09.2024

2024 г.

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

З А Д А Н И Е
на выполнение научно-исследовательской работы

по теме «Исследование средств разработки программного обеспечения для микроконтроллеров WCH»

Студент группы ИУ6-21М

Марчук Иван Сергеевич
(Фамилия, имя, отчество)

Магистерская программа 09.04.01/05 Современные интеллектуальные программно – аппаратные комплексы

Направленность НИР (исследовательская, практическая, производственная, др.)
Исследовательская

Источник тематики (кафедра, предприятие, НИР) Кафедра

График выполнения НИР: 25% к 4 нед., 50% к 7 нед., 75% к 11 нед., 100% к 14 нед.

Техническое задание: *Изучить возможности программирования микроконтроллера CH582M в средах MounRiver Studio и VisualStudio Code*

Оформление научно-исследовательской работы:

Расчетно-пояснительная записка на 25-30 листах формата А4.
Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.)

Дата выдачи задания «07» февраля 2024 г.

Руководитель НИР 07.02.2024 С.В. Ибрагимов
(Подпись, дата) (И.О.Фамилия)

Студент 07.02.2024 И.С. Марчук
(Подпись, дата) (И.О.Фамилия)

Примечание: Задание оформляется в двух экземплярах.

РЕФЕРАТ

Расчетно-пояснительная записка 45 страниц, 51 рисунок, 2 таблицы, 19 источников.

МИКРОКОНТРОЛЛЕР, RISC-V, WCH, СРЕДА РАЗРАБОТКИ, BLUETOOTH LOW ENERGY, ПРОГРАММАТОР, КОМПИЛЯТОР.

Объектом исследования является микроконтроллер CH582M от компании WCH. Этот микроконтроллер перспективен и актуален на данный момент. Он имеет архитектуру RISC-V, большое количество периферии, в том числе BLE и 2 порта USB, и большое количество памяти, но при этом продается по низкой цене.

В рамках данного исследования моя цель - проанализировать возможности разработки программ для CH582M при помощи различных инструментов. В будущем это исследование поможет в настройке окружения на ранних этапах разработки проектов, включающих этот микроконтроллер. А также поможет с выбором микроконтроллера при проектировании устройств.

ОГЛАВЛЕНИЕ

ОБОЗНАЧЕНИЯ, ОПРЕДЕЛЕНИЯ И СОКРАЩЕНИЯ

Радио модуль (Радиочастотный модуль) – это (обычно) небольшое электронное устройство, используемое для передачи и/или приема радиосигналов между двумя устройствами. Часто является встраиваемым компонентом системы. Беспроводная связь может осуществляться с помощью оптической связи или с помощью радиочастотной (RF) связи;

WCH – NanjingQinhengMicroelectronics, крупный китайский производитель устройств TTL-to-USB и микроконтроллеров;

BLE – Bluetooth с низким энергопотреблением (Bluetooth Low Energy), выпущенная в декабре 2009 года версия спецификации ядра беспроводной технологии Bluetooth, наиболее существенным достоинством которой является сверхмалое пиковое энергопотребление, среднее энергопотребление и энергопотребление в режиме простоя.;

RISC-V – расширяемая открытая и свободная система команд и процессорная архитектура на основе концепции RISC, предназначенная для создания процессоров/микроконтроллеров и разработки ПО;

UART – Универсальный асинхронный приёмопередатчик, узел вычислительных устройств, предназначенный для организации связи с другими цифровыми устройствами.;

IDE – Интегри́рованная среда́ разрабо́тки (integrated development environment), комплекс программных средств, используемый программистами для разработки программного обеспечения (ПО);

PATH – переменная окружения Unix-подобных операционных систем, DOS, OS/2 и Microsoft Windows, представляющая собой набор каталогов, в которых расположены исполняемые файлы. В основном, каждый выполняемый процесс или сеанс пользователя имеет собственную переменную PATH;

МК – Микроконтроллер;

SPI – Serial Peripheral Interface последовательный интерфейс для связи МК с другими внешними устройствами;

USB (Universal Serial Bus) – последовательный интерфейс для подключения периферийных устройств к вычислительной технике;

USB-HID – Human Interface Devices;

Datasheet – Техническая спецификация, карта данных, даташит. Это документ, который объединяет в себе технические характеристики продукта, материала, компонента (например, электронного) и предназначен для использования инженером-конструктором. Создается компанией-разработчиком (производителем) и начинается со вводной страницы, за которой следуют списки конкретных характеристик.

ВВЕДЕНИЕ

В современном мире беспроводные технологии играют все более важную роль и находят все большее применение в различных сферах нашей жизни. Одной из самых популярных и востребованных технологий является Bluetooth, которая позволяет передавать данные без использования проводных подключений. Эта технология находит применение в устройствах самого разного назначения — от бытовой электроники до промышленных систем автоматизации.

Микроконтроллер CH582M от компании WCH удачно сочетает в себе оба этих прогрессивных направления — поддержку Bluetooth и архитектуру RISC-V, а также предоставляет встроенную поддержку аппаратного USB-интерфейса. Эти характеристики делают его привлекательным решением для различных проектов, особенно с учетом его конкурентоспособной цены на рынке.

Для более детального изучения возможностей микроконтроллера CH582M я приобрел несколько отладочных плат и программатор, что позволило мне провести серию практических экспериментов. В рамках данной работы предполагается исследование возможностей программирования этого микроконтроллера как в собственной среде разработки MounRiver Studio, так и в Visual Studio Code с использованием специально настроенного набора инструментов.

Таким образом, данная работа направлена на изучение возможностей программирования микроконтроллера CH582M в собственной среде MounRiver Studio, а также в среде VisualStudio Code со специально настроенным набором инструментов.

1 Программирование микроконтроллера CH582M в MounRiver Studio

1.1 Установка ПО для программирования в MounRiver Studio

Для программирования своих микроконтроллеров компания WCH предоставляет неплохую среду разработки со всеми необходимыми функциями. В ней есть компилятор и сборщик, настроен OpenOCD для внутрисхемной отладки и прошивки микроконтроллеров, а также WCH-LinkUtility для настройки программатора WCH-Link.

Загрузить среду можно с официального сайта компании WCH [1]. Я выбирал версию со встроенными утилитами (embedded), рисунок 1.

На сайте mounriver [2] есть статья на английском языке для ознакомления с основным функциями среды.

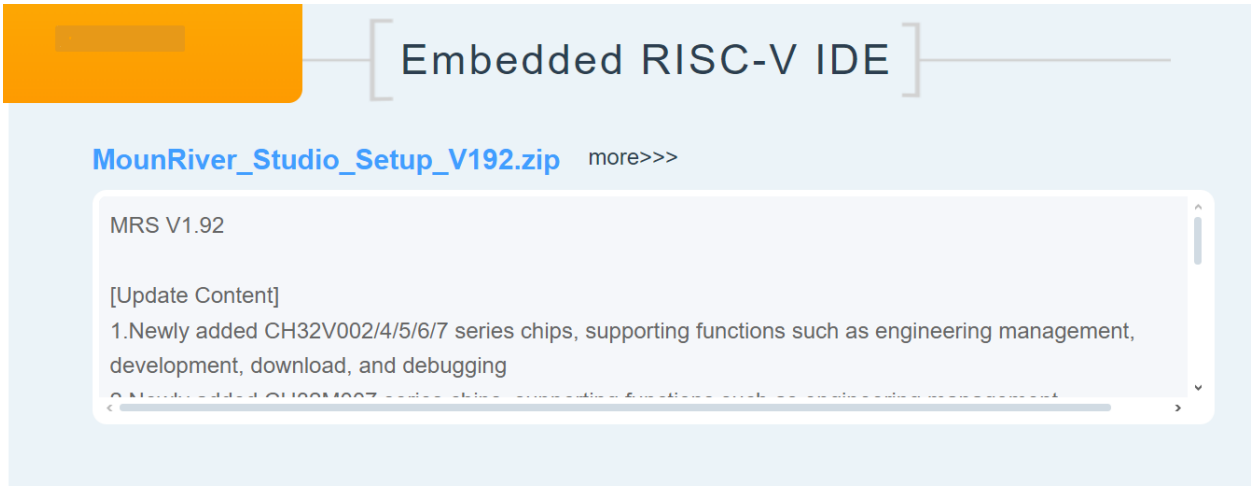

Рисунок 1 – Раздел со ссылкой на установочный файл MounRiver

1.2 Создание пустого проекта.

После установки откроется сама среда. Для создания проекта в верхнем меню нужно выбрать «File» -> «New» -> «MounRiver Project» (рисунок 2).

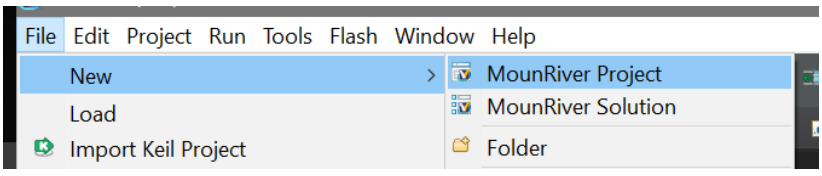

Рисунок 2 – Пункт меню создания нового проекта

После этого откроется окно создания проекта (рисунок 3). В нем можно настроить название папки с проектом и её расположение (расположение должно быть без пробелов и кириллицы). Здесь же предлагается выбрать архитектуру микроконтроллера для того, чтобы загрузить начальный шаблон проекта. Этот шаблон содержит минимальную работоспособную программу, которую в дальнейшем можно будет переделать под свои нужды, что будет гораздо проще чем создавать проект с нуля.

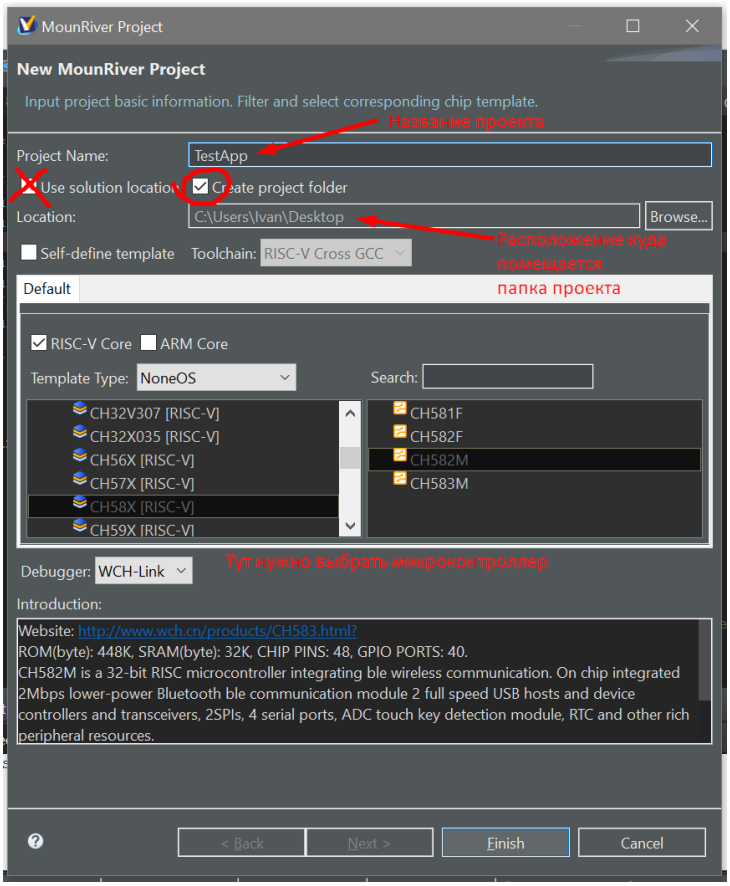

Рисунок 3 – окно создания проекта с настройками для CH582M

После нажатия кнопки Finish проект отобразится во вкладке Project Explorer (что видно на рисунке 4). Если же проект не открылся или был случайно удален из списка проектов, его можно открыть снова перейдя в расположение проекта и найдя там файл «.wvproj».

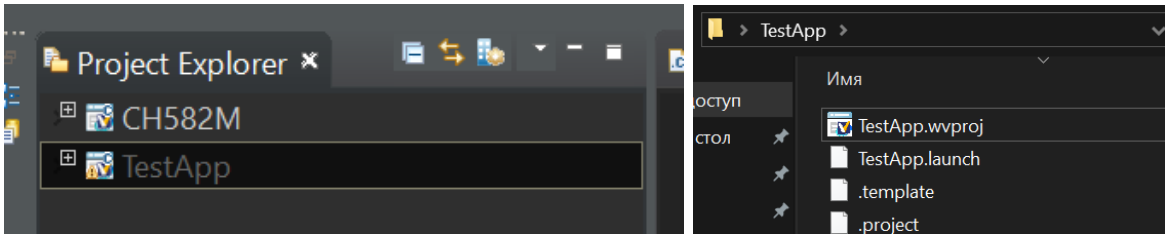

Рисунок 4 – Слева на Project Explorer отображаются 2 открытых проекта: «TestApp» и «CH582M», справа выделен файл wvproj в директории проекта

1.3 Структура проекта

Нажав на значок проекта, можно увидеть его структуру (рисунок 5).

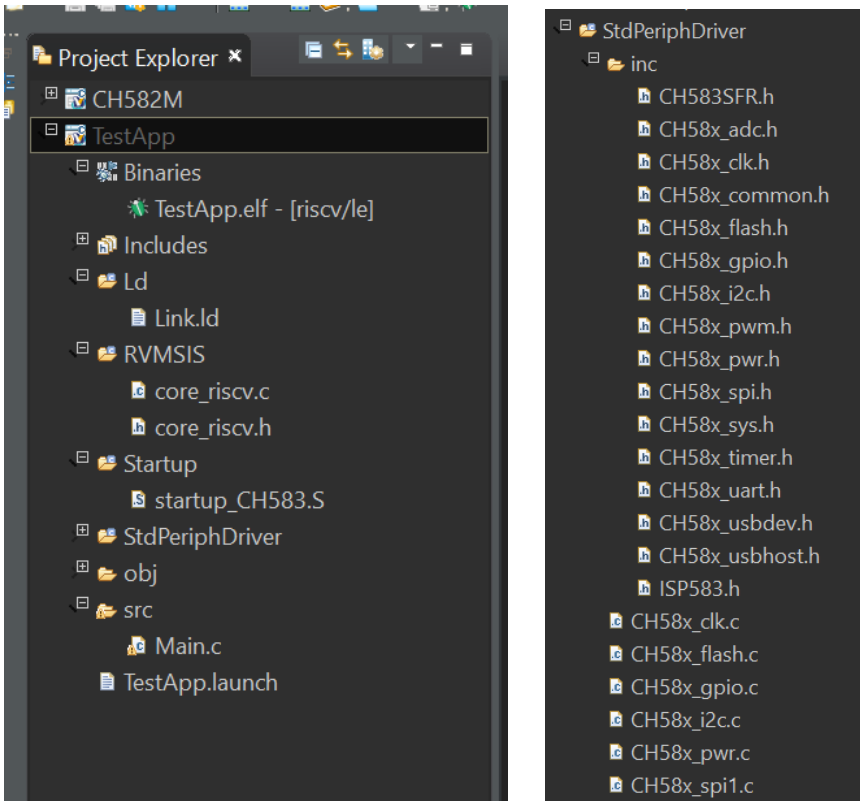

Рисунок 5 – Структура проекта «TestApp» слева, открытая папка StdPeriphDriver с библиотеками периферии микроконтроллера справа

Рассмотрим отдельно каждую из папок:

– «Binaries» – тут хранится ссылка на скомпилированный elf файл прошивки загружаемый в МК из папки «Obj»;

– «Includes» – подключенные библиотеки и компиляторы;

– «Ld» – содержит файл «Link.ld»;

– «RVMSIS» – содержит файлы с функциями для работы с ядром RISC-V;

– «Startup» – содержит логику инициализации регистров микроконтроллера;

– «StdPeriphDriver» – библиотека кода для работы с периферией МК, все функции в нем являются «оберткой» вокруг регистров мк и оперируют ими на прямую;

– «src» – папка с исходными текстами пользовательских программ;

– «obj» – Все скомпилированные временные файлы;

– «TestApp.launch» – файл с настройками проекта в среде MounRiver

Все файлы с готовыми функциями можно редактировать, но для базовых и средних по сложности проектов это не нужно. Хотя, при разработке мне пришлось не один раз заходить в папку «StdPeriphDriver», чтобы посмотреть принцип работы функций периферии. Также все функции снабжены подробными комментариями (к сожалению, на китайском), которые тоже сильно помогли мне в написании прошивок.

1.4 Разбор программы, демонстрирующей работу UART

Микроконтроллер CH583M имеет достаточно обширную периферию, среди которой есть аппаратный TTL порт протокола UART. В созданном по умолчанию проекте написан «Hello world» код, демонстрирующий работу UART1. Для понимания работы алгоритма, я перевел комментарии на русский язык.

Листинг 1 кода Main.c из стартового проекта (с переведенными комментариями):

```
#include "CH58x_common.h"

// буферы для отправки и приема данных
uint8_t TxBuff[] = "This is a tx example\r\n";
uint8_t RxBuff[100];
uint8_t trigB;

/*********************************************************************
 * @fn      main
 * @brief   Основная функция
 */
int main(){

    // длинна полученной строки
```

```
    uint8_t len;

    // внутренний генератор, на плате есть внешний кварц на 32Мгц
    SetSysClock(CLK_SOURCE_PLL_60MHz);
    // частота работы камня 60 МГц, а частота кварца 32
    //  Хитрость заключается в том, что если вы используете PLL то выход с генератора
    //  сразу умножается на 15 и получаем частоту = 480 МГц, а уже из нее путем
    //  деления получаем нужное нам значение.

    /* Настройка последовательного порта 1:
     * Сначала настройте режим порта ввода-вывода,
     * затем настройте последовательный порт */
    GPIOA_SetBits(GPIO_Pin_9);

    // обращмемся к регистру A
    // Подтяжка RX-пина к питанию
    GPIOA_ModeCfg(GPIO_Pin_8, GPIO_ModeIN_PU);
    // Конфигурация TXD-выхода, обратите внимание на то, чтобы сначала обеспечить
    //   высокую подтяжку порта ввода-вывода
    GPIOA_ModeCfg(GPIO_Pin_9, GPIO_ModeOut_PP_5mA);

    UART1_DefInit();


#if 1

    // Проверьте последовательный порт для отправки строки
    UART1_SendString(TxBuff, sizeof(TxBuff));

    // Метод запроса: Отправка после получения данных
    // Мой главный цикл
    while(1){
        len = UART1_RecvString(RxBuff);

        if(len){
            UART1_SendString(RxBuff, len);
        }
    }

#endif


#if 0 // Метод прерывания: Отправка после получения данных
    UART1_ByteTrigCfg(UART_7BYTE_TRIG);
    trigB = 7;
    UART1_INTCfg(ENABLE, RB_IER_RECV_RDY | RB_IER_LINE_STAT);
    PFIC_EnableIRQ(UART1_IRQn);
#endif


    // зацикливание в конце программы
    while(1);
}

/*********************************************************************
 * @fn      UART1_IRQHandler
 * @brief   Функция обработки прерывания UART1
 * @return  none
 */
__INTERRUPT
__HIGH_CODE
void UART1_IRQHandler(void){
    volatile uint8_t i;

    switch(UART1_GetITFlag()){
        case UART_II_LINE_STAT:{ // Ошибка состояния строки
            //UART1_GetLinSTA();
            break;
        }

        case UART_II_RECV_RDY: // Данные достигают заданной точки срабатывания

            // в цикле от 0 до 7
            for(i = 0; i != trigB; i++){
                // записываем в строку RxBuff 1 байт
                RxBuff[i] = UART1_RecvByte();
                // отправка одного байта с передачей по значению
                UART1_SendByte(RxBuff[i]);
            }
            break;
```

```
        // Время приема истекло, временно завершен один кадр приема данных
        case UART_II_RECV_TOUT:

            // чтение нескольких байт из буфера последовательного порта
            //   и запись их в RxBuff. Возвращает количество символов.
            i = UART1_RecvString(RxBuff);

            // отправляет в последовательный порт i байт
            //    начиная с переданного адреса RxBuff
            UART1_SendString(RxBuff, i);
            break;

        case UART_II_THR_EMPTY: // Буфер отправки пуст, вы можете продолжить отправку
            break;

        case UART_II_MODEM_CHG: // Поддерживается только последовательный порт 0
            break;

        default:
            break;
    }
}
```

В начале программы функции main происходит настройка портов TX и RX. Затем вызывается функция инициализации UART1_DefInit(). Удерживая клавишу Ctrl можно перейти к исходному коду этой функции.

Листинг 2 функции UART1_DefInit(), отвечающей за инициализацию UART:

```
void UART1_DefInit(void){
    UART1_BaudRateCfg(115200);
    // FIFO control register
    // FIFO включен, триггерная точка равна 4 байтам
    R8_UART1_FCR = (2 << 6) | RB_FCR_TX_FIFO_CLR | RB_FCR_RX_FIFO_CLR | RB_FCR_FIFO_EN;
    R8_UART1_LCR = RB_LCR_WORD_SZ;// Line control register
    R8_UART1_IER = RB_IER_TXD_EN; // Interrupt enable register
    R8_UART1_DIV = 1;             // Prescaler divisor register
}
```

В datasheet этого микроконтроллера есть подробное описание каждого из регистров. Для примера разберу что делает каждый из регистров в функции UART1_DefInit (таблица 1).

Таблица 1 – Описание регистров использующихся в UART1_DefInit и присваиваемых им значений

Название регистра и описание	Подаваемое значение и описание
R8_UARTx_FCR – регистр включения и настройки работы FIFO стека UART;	(2 << 6) – RB_FCR_FIFO_TRIG RW Выбор точек запуска для получения прерывания FIFO и аппаратного управления потоком данных: 00: 1 byte; 01: 2 bytes; **10: 4 bytes;** 11: 7 bytes. То есть прерывание, доступное для приема данных, генерируется при получении 4 байт, и вывод RTS автоматически аннулируется.

	RB_FCR_TX_FIFO_CLR – Бит включения очистки и моментальной очистки TX стека FIFO 1 - Очистить данные FIFO для передатчика RB_FCR_RX_FIFO_CLR – Бит включения очистки и моментальной очистки RX стека FIFO RB_FCR_FIFO_EN – бит включения FIFO: 1 - Включить 8-битный FIFO;
R8_UARTx_LCR – Line control register	Значение «#define RB_LCR_WORD_SZ 0x03» Это настраиваемая константа, обозначающая длинну передоваемого слова UART: RW, UART word bit length: 00-5bit, 01-6bit, 10-7bit, 11-8bit Здесь стоит значение 11, то есть размер передаваемого слова 8 бит
R8_UARTx_IER – Регистр включения прерываний	Значение «#define RB_IER_TXD_EN 0x40» - Бит включения прерывания на выводе UART TXD = 1
R8_UARTx_DIV - используется для вычисления внутренних опорных тактовых импульсов UART, младшие 7 бит являются значимыми. Формула: Делитель = Fsys*2 / внутренние опорные тактовые импульсы UART, максимальное значение равно 127.	Значение 1

В листинге 1 представлена конструкция выбора режима работы UART: когда UART работает в главном потоке и когда он работает через прерывания.

Работа в главном потоке основывается на двух методах: UART1_RecvString(RxBuff) и UART1_SendString(RxBuff, len). Исходный код этих функций приведён в листинге 3.

Листинг 3 – исходный код функций RecvString и SendString

```
uint16_t UART1_RecvString(uint8_t *buf){
    uint16_t len = 0;

    while(R8_UART1_RFC){ // пока в стеке есть данные (счетчик не равен 0)
        *buf++ = R8_UART1_RBR; // переписываем байт из регистра в буфер программы
        len++;
    }
    return (len); // размер полученной строки
}

void UART1_SendString(uint8_t *buf, uint16_t l){
    uint16_t len = l;

    while(len){                             // проходимся по длине строки
        if(R8_UART1_TFC != UART_FIFO_SIZE){ // если стек еще не заполнен
            R8_UART1_THR = *buf++;          // записываем текущий байт из буфера программы
            len--;
        }
    }
}
```

Приведу расшифровку регистров, используемых в этих программах из Datasheet (таблица 2).

Таблица 2 – Описание регистров и присваиваемых им значений, использующихся в функциях RecvString и SendString

Название регистра или константы	Назначение
R8_UART1_RFC (Read only)	Счетчик полученных данных в текущем приемном стеке FIFO
R8_UART1_RBR (Read only)	Регистр буфера приема данных. Если бит DATA_RDY в LSR равен 1, полученные данные могут быть считаны из этого регистра; Если FIFO_EN равно 1, то данные, полученные из сдвигового регистра UART RSR, сначала будут сохранены в FIFO приемника, а затем считаны через регистр.
R8_UART1_TFC (Read only)	Счетчик данных в текущем стеке TX FIFO
UART_FIFO_SIZE (Константа)	Глубина стека. Равна 8
R8_UART1_THR (Write Only)	Регистр удержания передачи. Включая стек FIFO передатчика, используется для записи передаваемых данных; если значение FIFO_EN равно 1, записанные данные сначала будут сохранены в FIFO передатчика, а затем выведены один за другим через передающий сдвиговый регистр TSR.

1.5 Программа управления светодиодом через UART

В качестве следующего шага я решил написать программу для управления с помощью консоли светодиодом на отладочной плате (рисунок 6).

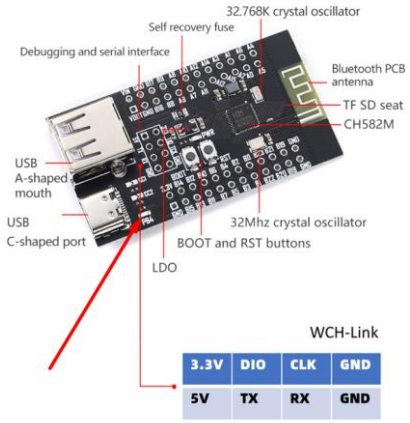

Рисунок 6 – Внешний вид используемой мною отладочной платы (красной стрелкой отмечен светодиод подключенный к одному из выводов МК)

Исходный файл Main.c я модифицировал следующим образом, листинг 4.

Листинг 4 – Программа управления светодиодом через консоль

```
#include "CH58x_common.h"

// буферы для отправки и приема данных
uint8_t TxBuff[] = "This is a tx example\r\n";
uint8_t RxBuff[100];

//   Основная функция
int main(){

    // длинна полученной строки
    uint8_t len;
    // внутренний генератор, на плате есть внешний кварц на 32Мгц
    SetSysClock(CLK_SOURCE_PLL_60MHz);
    // частота работы камня 60 МГц, а частота кварца 32
    //  Хитрость заключается в том, что если вы используете PLL то выход с генератора
    //  сразу умножается на 15 и получаем частоту = 480 МГц, а уже из нее путем
    //  деления получаем нужное нам значение.

    // PB4 как выход
    GPIOB_ModeCfg(GPIO_Pin_4, GPIO_ModeOut_PP_5mA);
    // (Гасим светодиод) Установка низкого выходного уровня на пине порта GPIOB
    GPIOB_ResetBits(GPIO_Pin_4);



    /* Настройка последовательного порта 1:
     * Сначала настройте режим порта ввода-вывода,
     * затем настройте последовательный порт */
    GPIOA_SetBits(GPIO_Pin_9);

    // обращмемся к регистру A
    // Подтяжка RX-пина к питанию
    GPIOA_ModeCfg(GPIO_Pin_8, GPIO_ModeIN_PU);
    // TXD-Сконфигурируйте двухтактный выход, обратите внимание на то, чтобы сначала обеспечить
    //   высокую подтяжку порта ввода-вывода
    GPIOA_ModeCfg(GPIO_Pin_9, GPIO_ModeOut_PP_5mA);
    UART1_DefInit();
    // Проверьте последовательный порт для отправки строки
    UART1_SendString(TxBuff, sizeof(TxBuff));

    // Метод запроса: Отправка после получения данных
    // Мой главный цикл
    while(1){
        len = UART1_RecvString(RxBuff);

        if(len){

            if(RxBuff[0] == 'a'){
                // Зажигаем светодиод
                GPIOB_SetBits(GPIO_Pin_4);
            } else if(RxBuff[0] == 'b'){
                // Гасим светодиод (низкий уровень на выходе)
                GPIOB_ResetBits(GPIO_Pin_4);
            }
            UART1_SendString(RxBuff, len);
        }
    }

    // зацикливание в конце программы // надо будет выяснить для чего todo
    while(1);
}
```

В данном примере я оставил обработку UART в главном потоке. При получении строки микроконтроллер анализирует первый символ этой строки, если это символ «a», микроконтроллер устанавливает единицу на выводе 4, если это символ «b», микроконтроллер устанавливает ноль на выводе 4.

1.6 Прошивка написанной программы в микроконтроллер

1.6.1 Программаторы

Чтобы загрузить прошивку в микроконтроллер нужен программатор, компания WCH выпускает несколько моделей программаторов, предназначенных под разные задачи [3]. Там же есть таблица (рисунок 7), по которой можно определить, какой программатор подходит для микроконтроллера

Function items	WCH-Link-R1-1v1	WCH-LinkE-R0-1v3	WCH-DAPLink-R0-2v0	WCH-LinkW-R0-1v1
RISC-V mode	√	√	×	√
ARM-SWD mode-HID device	×	×	√	×
ARM-SWD mode-WINUSB device	√	√	√	√
ARM-JTAG mode -HID device	×	×	√	×
ARM-JTAG mode -WINUSB device	×	√	√	√
ModeS key to switch mode	×	√	√	√
2-wire way upgrade firmware offline	×	√	×	×
Serial port upgrade firmware offline	√	×	×	×
USB upgrade firmware offline	√	×	√	√
Controllable 3.3V/5V power output	×	√	√	√
High-speed USB2.0 to JTAG interface	×	√	×	×
Wireless mode	×	×	×	√
Download tools	MounRiver Studio WCH-LinkUtility Keil uVision5	MounRiver Studio WCH-LinkUtility Keil uVision5	WCH-LinkUtility Keil uVision5	MounRiver Studio WCH-LinkUtility Keil uVision5
Keil supported versions	Keil V5.25 and above	Keil V5.25 and above	Supported in all versions of Keil	Keil V5.25 and above

Рисунок 7 – Характеристики программаторов WCH-Link

В случае CH582M необходимо наличие режима RISC-V, а также возможность прошивки через последовательный порт «Serial port upgrade firmware offline».

Исходя из необходимых характеристик я купил WCH-Link-R1-1v1

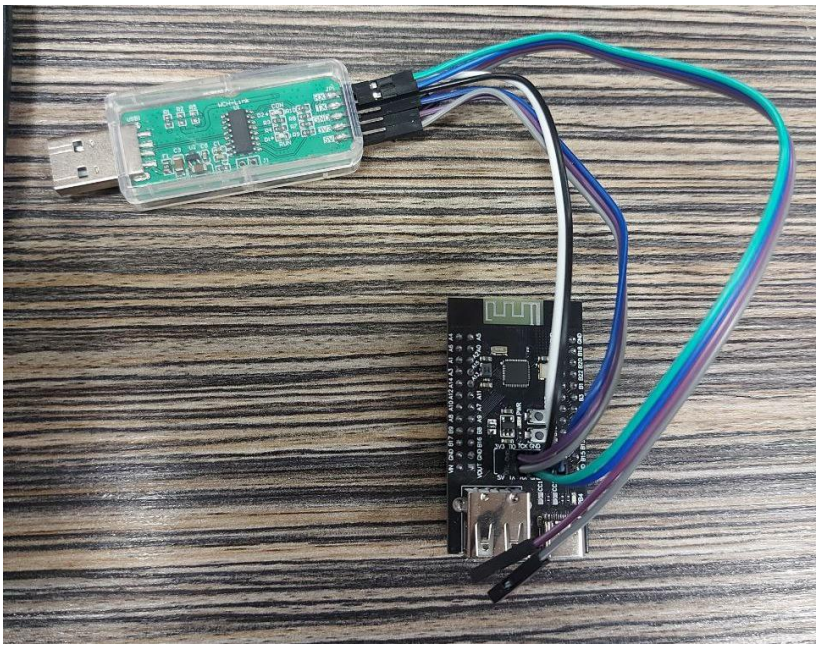

Рисунок 8 – Подключенный к отладочной плате программатор

Подключение осуществляется по следующей схеме:

- «SWCLK» программатора в «TCK» платы;
- «SWDIO» программатора в «TIO» платы;
- «RX» программатора в «TX» платы;
- «RX» программатора в «TX» платы;
- «GND» программатора в «GND» платы;
- «5V» программатора в «5V» платы;
- «3V3» программатора в «3V3» платы.

Драйвер для «WCH-Link» загружается совместно с «WCH-LinkUtility» если его загружать отдельно [4] а также доступен после установки MoonRiver в каталоге «C:\WCH.CN\WCHLinkDrv».

1.6.2 Загрузка программы в микроконтроллер

После подключения контроллера к USB через программатор, его можно прошить, выбрав в меню пункт «Run» - «Run». Если все подключено правильно, в консоли отобразится информация о запуске OpenOCD.

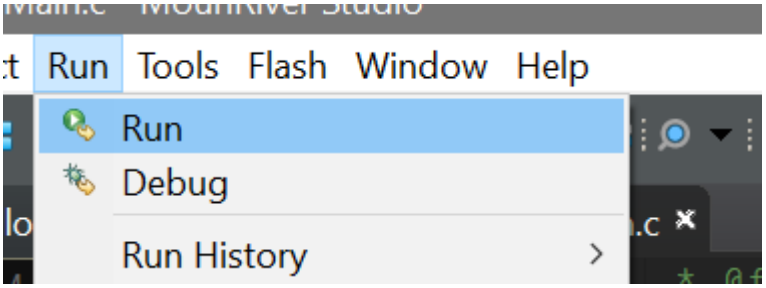

Рисунок 9 – Опции «Run» (прошить микроконтроллер) и «Debug» (запустить отладочный режим)

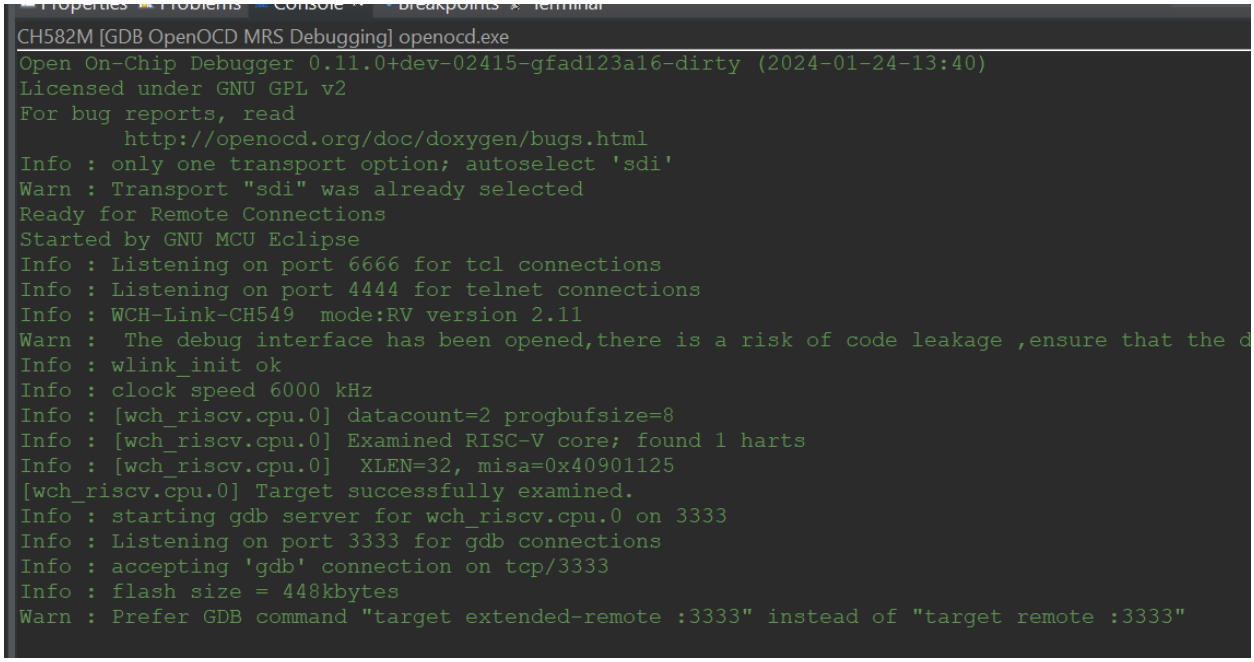

Рисунок 10 – Окно консоли с выводом логов OpenOCD

Также в этом же меню есть возможность программирования микроконтроллера в отладочном режиме.

После загрузки программы я открыл терминал встроенный в MounRiver. При вводе символа на клавиатуре он сразу же отправлялся по последовательному порту в микроконтроллер.

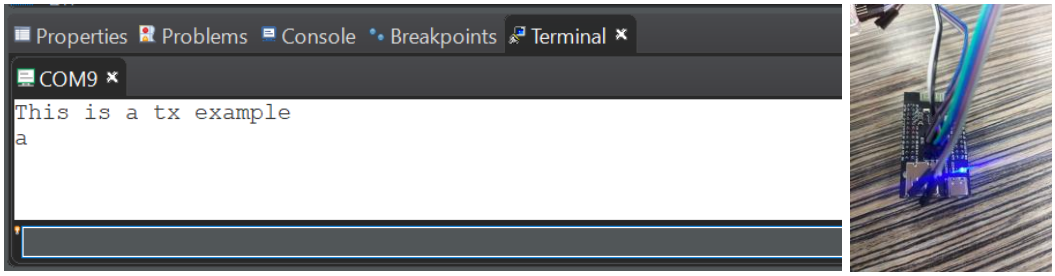

Рисунок 11 – Открытый терминал слева, отладочная плата с включенным светодиодом справа

При отправке через консоль символа 'a', светодиод включался, при отправке 'b' выключался.

Терминал я открывал с такими параметрами:

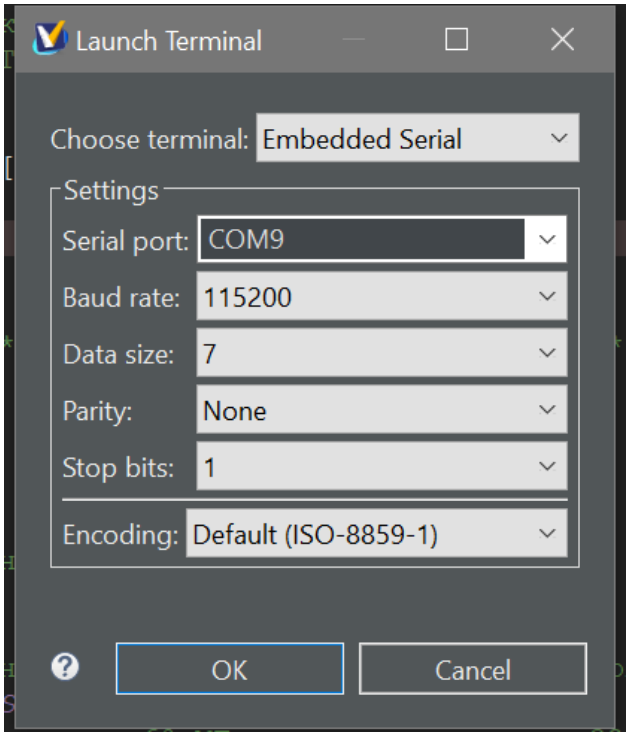

Рисунок 12 – Настройка терминала для коммуникации с микроконтроллером

1.6.3 Включение Debug режима

Чтобы загрузить прошивку в микроконтроллер под отладчиком нужно включить Debug режим, так как с производства он идет отключенным. Для этого нужно перейти в меню «Tools» - «WCH In-System Programmer».

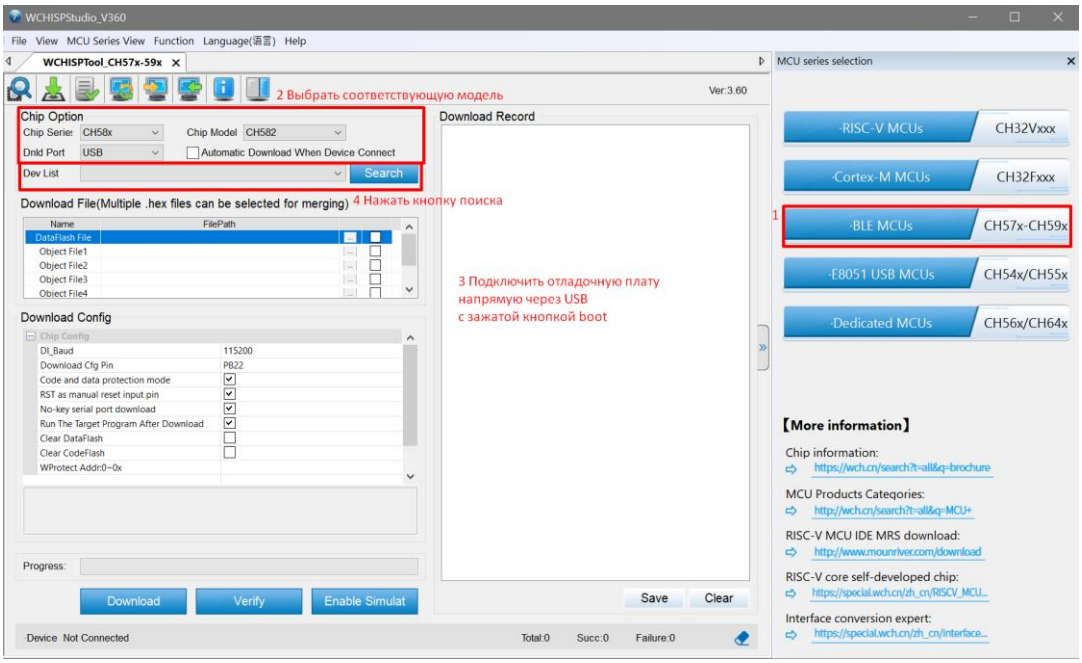

Рисунок 13 – окно WCHISPStudio

Появится окно WCHISPStudio. Последовательность действий для включения отладочного режима следующая:

1. Справа нужно выбрать пункт соответствующий прошиваемому микроконтроллеру «BLE MCUs»;
2. В секции «Chip Option» нужно выбрать нужный микроконтроллер (CH582)
3. Я зажал кнопку boot и подключил контроллер напрямую по USB, не через программатор
4. После этого в окне «Download Record» должна высветиться надпись «Device#0 UID…», обозначающая что микроконтроллер был найден программой
5. В течении 30 секунд необходимо нажать кнопку «Enable Simulat», и подтвердить выбор в диалоговом окне, после чего в окне «Download Record» появится надпись «Succ to Enable»

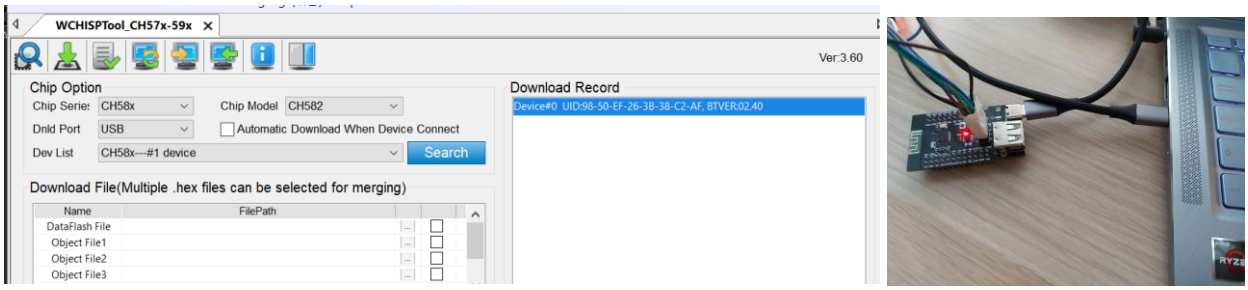

Рисунок 14 – интерфейс «WCHISPTool» с найденым микроконтроллером слева и отладочная плата с микроконтроллером, подключенным напрямую по USB справа

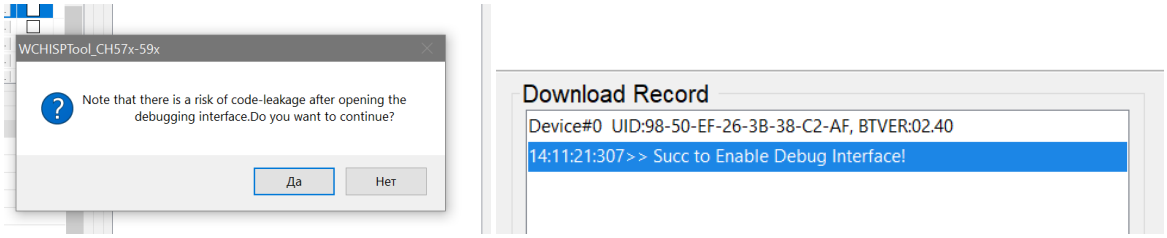

Рисунок 15 – Окно подтверждения включения отладочного режима слева и уведомление об успешно включенном отладочном режиме справа

1.6.4 Настройка режима отладки в MounRiver для пошагового выполнения команд

После включения отладочного режима на микроконтроллере необходимо настроить отладочный режим для проекта. Для этого нужно выбрать пункт меню «Run» - «Debug Configurations». Откроется окно настроек конфигурации. Для того чтобы добавить новый профиль настроек необходимо два раза нажать на поле «GDB OpenOCD MRS Debugging» слева. Чуть ниже появится новое поле с названием проекта (рисунок 16), а справа - его настройки (рисунок 17).

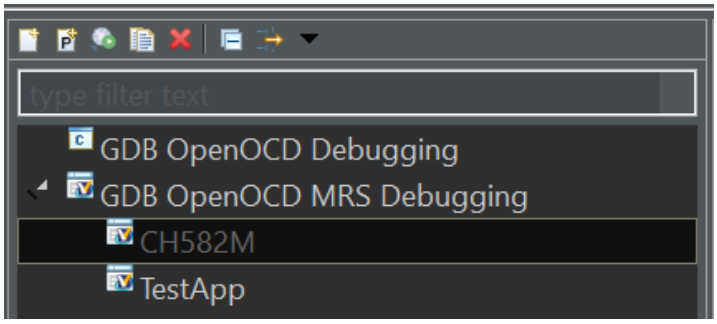

Рисунок 16 – Список конфигураций отладочного режима

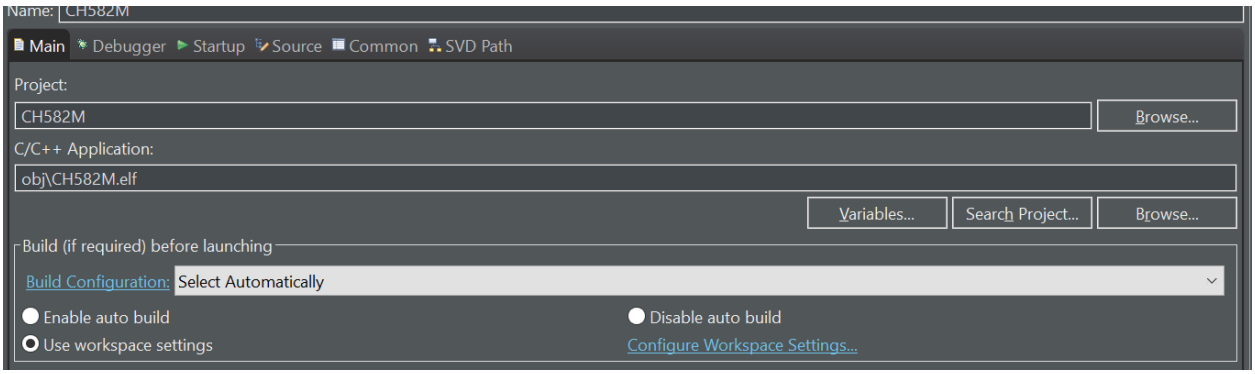

Рисунок 17 – Настройки вкладки «Main»

В настройках на вкладке «Main» автоматически указываются название проекта и путь до «elf» файла, если «elf» файл не был прописан автоматически, путь до него необходимо указать вручную (он расположен в папке «obj» проекта).

Дальше во вкладке «SVD Path» необходимо указать путь до SVD файла для данного микроконтроллера (рисунок 18).

Этот файл находится в следующей директории: «C:\MounRiver\MounRiver_Studio\template\wizard\WCH\RISC-V\CH58X\NoneOS\CH58Xxx.svd». В этом же каталоге находятся шаблоны проектов, которые пригодятся позже.

Есть еще сокращенный путь «${eclipse_home}template/wizard/WCH/RISC-V/CH58X/NoneOS/CH58Xxx.svd»

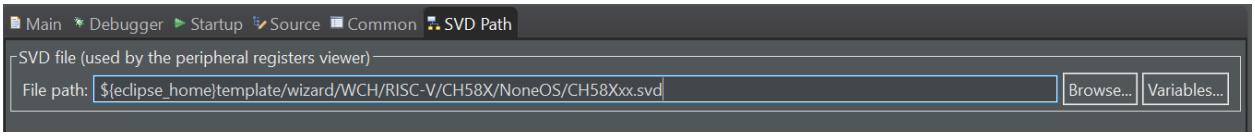

Рисунок 18 – Настройки вкладки «SVD Path»

Затем необходимо нажать кнопку «Apply» для сохранения настроек.

И сразу же можно запустить отладку нажатием кнопки «Debug», но перед этим необходимо подключить программатор с микроконтроллером.

После этих действий режим отладки можно вызывать из меню «Run» - «Debug».

После запуска этого режима становятся активными несколько кнопок в панели инструментов, такие как проход по команде, в команду, остановка и так далее (рисунок 19).

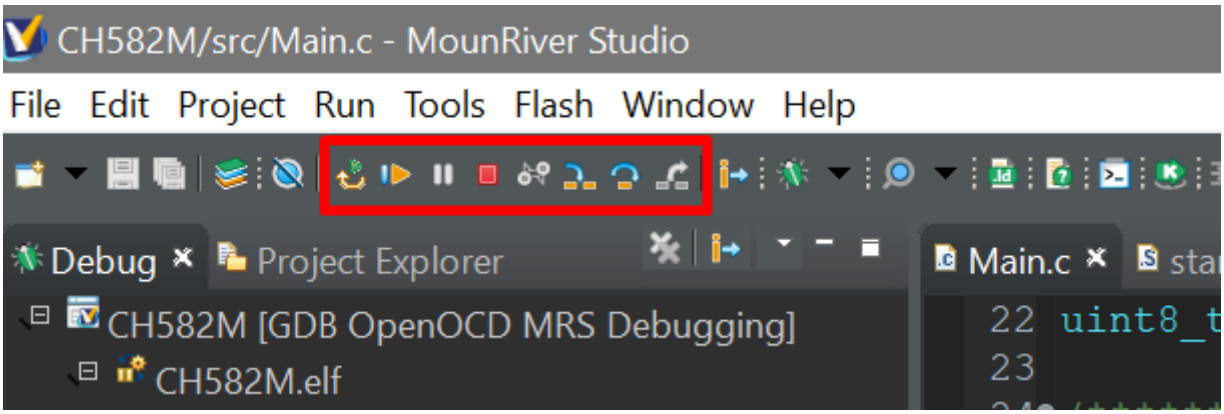

Рисунок 19 – Команды отладочного режима

После запуска отладочного режима появляются два новых окна (рисунок 20). Слева код ассемблера, с указателем исполняемой команды, справа список отслеживаемых переменных. А в окне с программой подсвечивается текущая исполняемая команда.

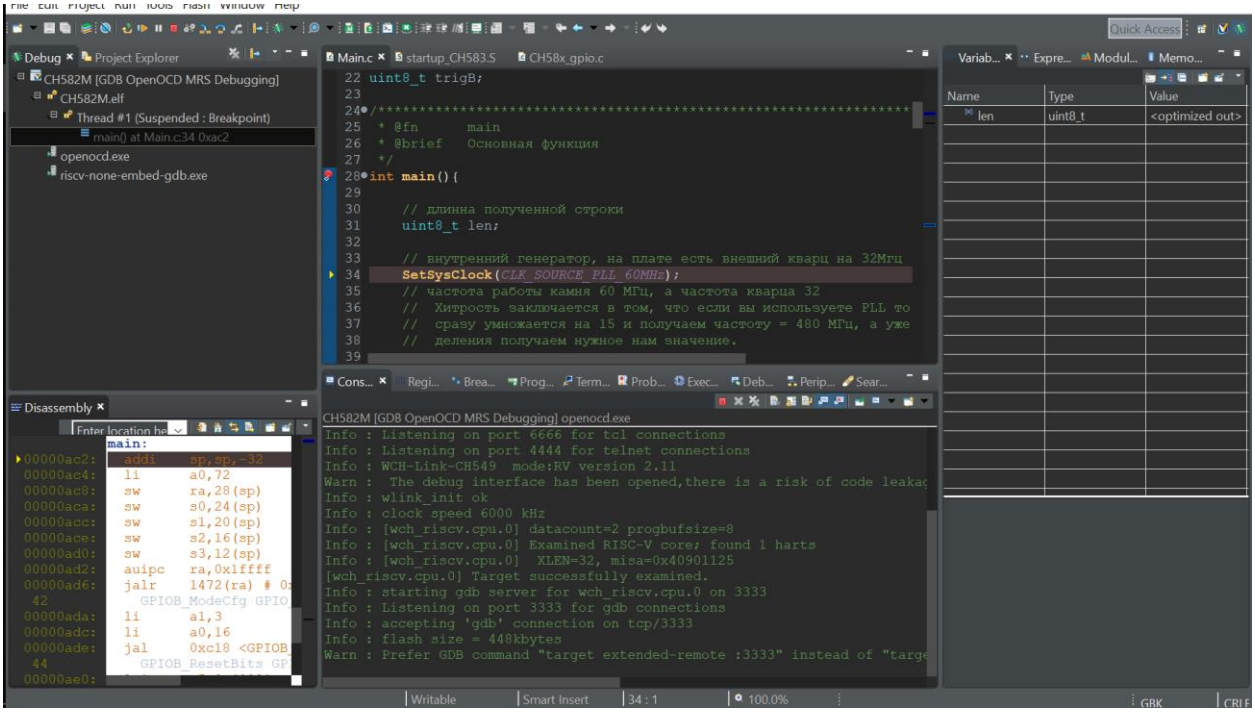

Рисунок 20 – Интерфейс MounRiver в отладочном режиме

В основной программе можно расставлять точки остановки, нажав слева от команды правой кнопкой мыши и выбрав пункт «Toggle Breakpoint» (рисунок 21).

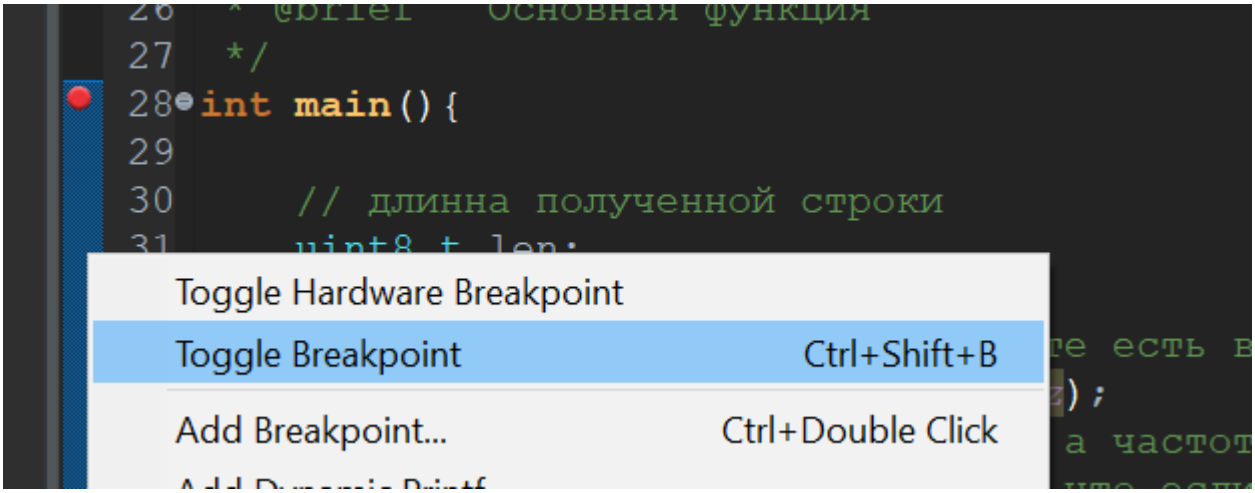

Рисунок 21 – пункт меню «Toggle Breakpoint»

1.7 Другие примеры программ для микроконтроллера CH582

Я нашел актуальный репозиторий с большим количеством примеров проектов для микроконтроллера CH582 на площадке github[5]. А также менее удобный для просмотра на сайте компании WCH[6].

Среди них я считаю важным отметить проекты, которые я планирую изучить для разработки своего дипломного проекта (создания Bluetooth USB клавиатуры). Я выделил список :

- COM-порт, работающий через «device USB», для назначения клавиш клавиатуры и других настроек допустим через специализированное ПО без прошивки микроконтроллера) [7];
- USB-HID устройство, работающее через device USB, для работы клавиатуры в проводном режиме [8];
- Bluetooth HID клавиатура, основной код для работы клавиатуры «по воздуху» [9];
- Bluetooth UART порт, для назначения настроек клавиатуры по Bluetooth [10];
- Примеры режимов сна и управления питанием микроконтроллера, для более экономного расхода заряда батареи клавиатуры [11];
- И возможно usb iap (In Application Programming) — это процесс программирования части пользовательской Flash во время выполнения программы, цель IAP это облегчение процесса обновления программ прошивки в продукте через зарезервированный порт связи уже после выпуска продукта [12].

2 Установка ПО для программирования в VSCode совместно с CMake и openocd

Есть статья, которая описывающая процесс настройки VSCode для микроконтроллера CH32x [13]. Шаги настройки окружения частично подходят и для CH58x, однако настройка проекта иначе, так как контроллеры имеют разную архитектуру.

Для программирования в Visual Studio Code понадобятся следующие программы:

- GCC (GNU Compiler Collection), набор компиляторов
- CMake, предназначен для сборки скомпилированного кода.
- Ninja, предназначен для облегчения процесса сборки CMake…
- OpenOCD, (Open On-Chip Debugger) позволяет выполнять внутрисхемное программирование и отладку микросхем. То есть благодаря этой утилите скомпилированную, а затем собранную прошивку можно загрузить на микроконтроллер.

2.1 Установка компилятора GCC

В первую очередь был установлен компилятор GCC для RISC-V архитектуры. Относительно свежий релиз мне удалось найти в репозитории xpack-dev-tools [14], релиз основан на GCC 13.2.0.

Есть страница с последними релизами [15], я загрузил пакет xpack-riscv-none-elf-gcc-13.2.0-2-win32-x64.zip

Архив с компилятором я поместил в папку «C:\Program Files\». Для того чтобы использовать этот компилятор при сборке проекта его необходимо было добавить в переменную среды PATH, указав путь до каталога bin (рисунок 22).

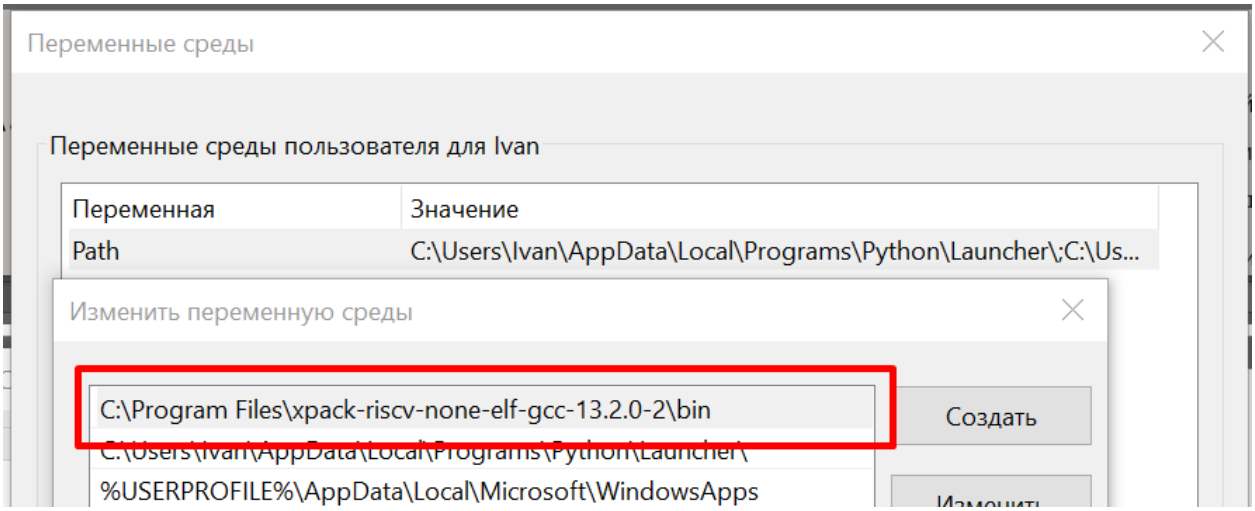

Рисунок 22 – Путь до компилятора в переменных среды

После этого для проверки установки, я запросил текущую версию gcc в командной строке, выполнив «riscv-none-elf-gcc --version», (рисунок 23).

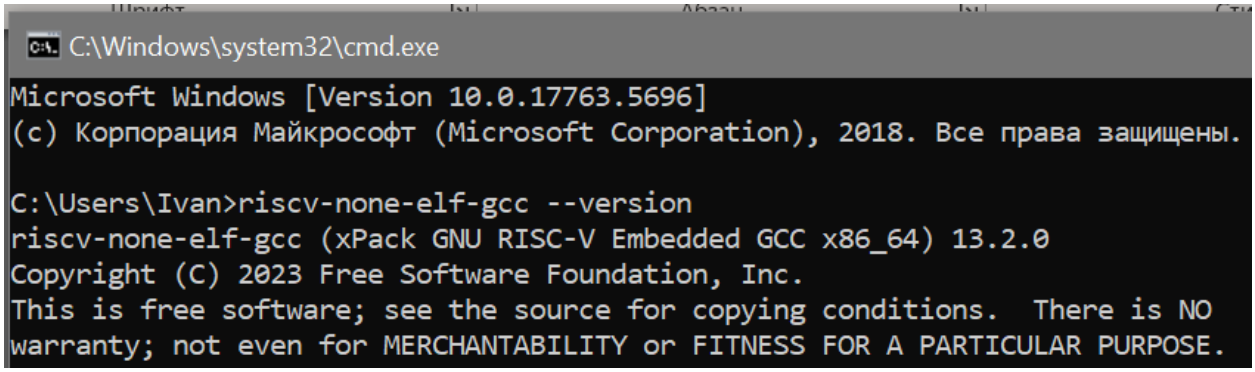

Рисунок 23 – Проверка версии GCC через консоль

2.2 Установка CMake

CMake очень легко устанавливается с официального сайта [16]. В процессе установки необходимо выбрать опцию «добавить в переменную PATH», (рисунок 24).

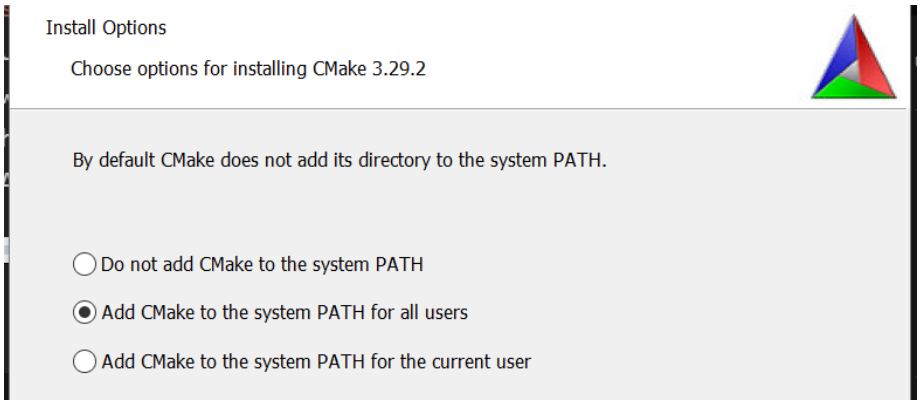

Рисунок 24 – Настройка «добавить в переменную PATH» при установке CMake

Также желательно устанавливать CMake в директорию не содержащую пробелов и ни в коем случае кириллицы, (рисунок 25).

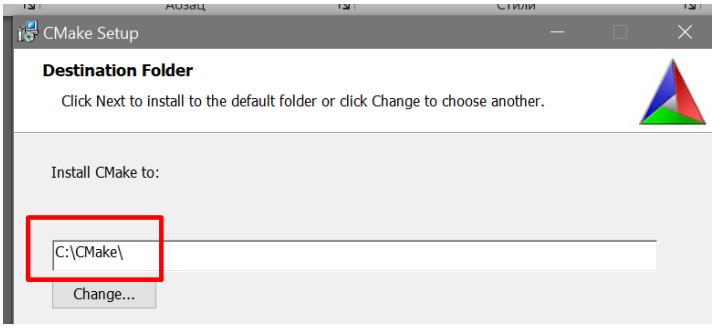

Рисунок 25 – мой путь установки CMake

После этого для проверки установки, я запросил текущую версию CMake в командной строке, выполнив «cmake --version», (рисунок 26).

```
C:\Users\Ivan>cmake --version
cmake version 3.30.3

CMake suite maintained and supported by Kitware (kitware.com/cmake).
```

Рисунок 26 – Проверка версии CMake через консоль

2.3 Установка Ninja

Последнюю версию программы также можно скачать с официального репозитория [17].

Для этого приложения тоже необходимо прописывать путь в Path, однако для упрощения работы я поместил ninja.exe в папку с файлами Cmake, до которой уже был прописан путь, (рисунок 27).

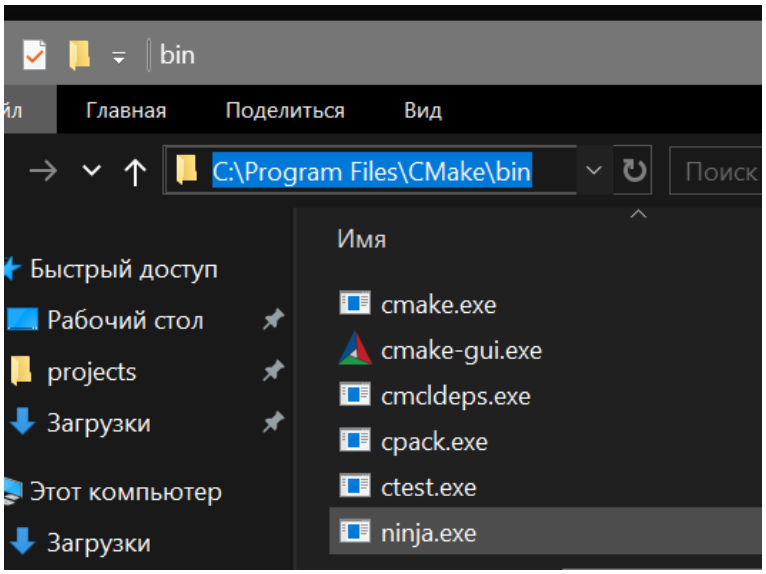

Рисунок 27 – Файл ninja помещенный в директорию «bin»

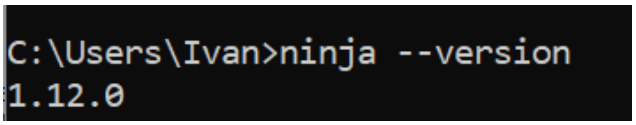

Рисунок 28 – Проверка версии Ninja в консоли

Проверку установки я проделал командой ninja --version, (рисунок 28).

2.4 Установка OpenOCD

Оригинальная версия утилиты не подходит, поэтому одним из возможных решений является просто взять версию из MounRiver, выполнив простые действия:

1. Скопировать директорию <MounRiver>/toolchain/OpenOCD в произвольное место. Я поместил папку с OpenOCD в каталог CMake (рисунок 29);

2. Скопировать файл OpenOCD/bin/wch-riscv.cfg в директорию OpenOCD/share/openocd/scripts/interface (по умолчанию openocd ищет скрипты в директории OpenOCD/share/openocd/scripts).

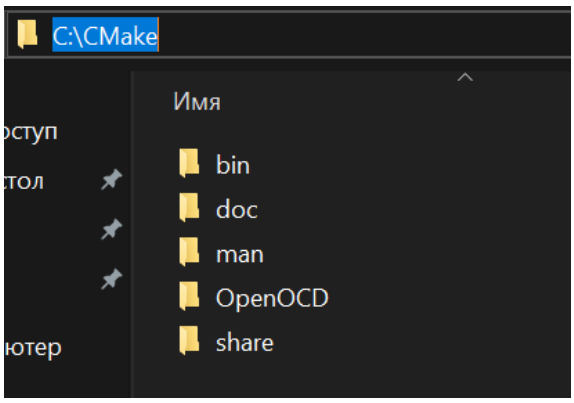

Рисунок 29 – Папка OpenOCD расположенная в директории CMake

Также для того, чтобы исполняемый файл openOCD был доступен из VisualStudio я добавил его в переменные PATH, (рисунок 30).

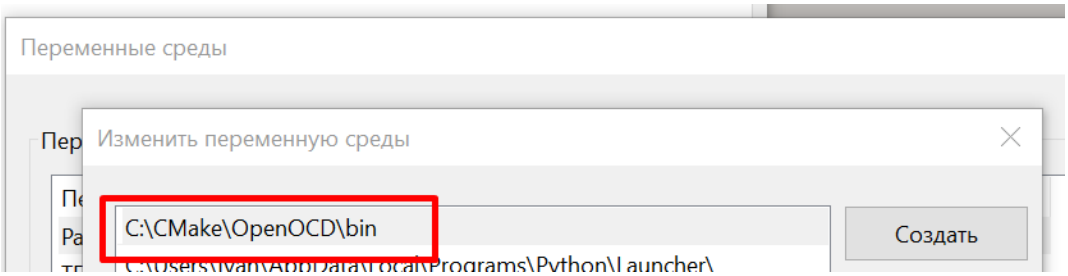

Рисунок 30 – Исполняемый файл OpenOCD в переменной среды PATH

Затем я решил проверить работу установленного OpenOCD, запустив его с необходимым файлом конфигурации. Без подключенного программатора вывод выглядит так, (рисунок 31).

```
C:\Users\Ivan>openocd -f "interface/wch-riscv.cfg"
Open On-Chip Debugger 0.11.0+dev-02415-gfad123a16-dirty (2024-01-24-13:40)
Licensed under GNU GPL v2
For bug reports, read
        http://openocd.org/doc/doxygen/bugs.html
Info : only one transport option; autoselect 'sdi'
Warn : Transport "sdi" was already selected
Ready for Remote Connections
Info : Listening on port 6666 for tcl connections
Info : Listening on port 4444 for telnet connections
Error: WLink Open Error
```

Рисунок 31 – Работа OpenOCD без программатора

С подключенным программатором вывод выглядит вот так, (рисунок 32) (установка драйвера программатора была описана выше):

```
C:\Users\Ivan>openocd -f "interface/wch-riscv.cfg"
Open On-Chip Debugger 0.11.0+dev-02415-gfad123a16-dirty (2024-01-24-13:40)
Licensed under GNU GPL v2
For bug reports, read
        http://openocd.org/doc/doxygen/bugs.html
Info : only one transport option; autoselect 'sdi'
Warn : Transport "sdi" was already selected
Ready for Remote Connections
Info : Listening on port 6666 for tcl connections
Info : Listening on port 4444 for telnet connections
Info : WCH-Link-CH549  mode:RV version 2.11
Warn :  The debug interface has been opened,there is a risk of code leakage
 ,ensure that the debug interface has been closed before leaving factory !
Info : wlink_init ok
Info : clock speed 6000 kHz
Info : [wch_riscv.cpu.0] datacount=2 progbufsize=8
Info : [wch_riscv.cpu.0] Examined RISC-V core; found 1 harts
Info : [wch_riscv.cpu.0]  XLEN=32, misa=0x40901125
[wch_riscv.cpu.0] Target successfully examined.
Info : starting gdb server for wch_riscv.cpu.0 on 3333
Info : Listening on port 3333 for gdb connections
```

Рисунок 32 – Работа OpenOCD с программатором

Перед извлечением программатора из компьютера окно консоли лучше закрыть иначе программа завершится с ошибкой, (рисунок 33).

```
Error: failed read at 0x11, status=5980062
Error: Maybe the device has been removed
Assertion failed: target->arch_info, file ./src/target/riscv/riscv.h, line
279
```

Рисунок 33 – Ошибка OpenOCD при отключении программатора

2.5 Настройка VSCode для сборки и отладки программы

После установки VSCode необходимо добавить в неё 2 расширения. «CMake Tools» (для удобства работы с Cmake), «Comand Variable» (для удобной настройки скриптов в «tasks.json»), (рисунок 34).

Рисунок 34 – Установленные расширения

Также необходимо установить стандартные библиотеки C++. На сайте Visual Studio Code даже есть инструкция как это сделать [18].

Во-первых необходимо установить еще одно расширение – инструменты C/C++ (рисунок 35).

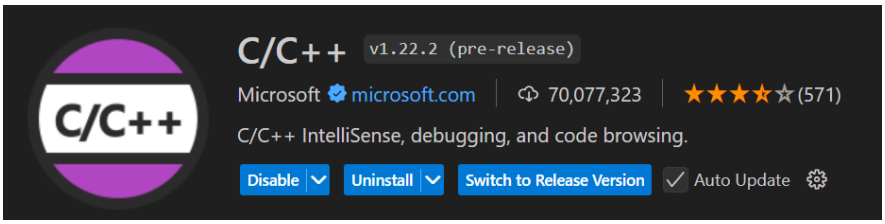

Рисунок 35 – Установленное расширение C/C++

Таже необходимо установить средства сборки visualStudio [19] (рисунок 36).

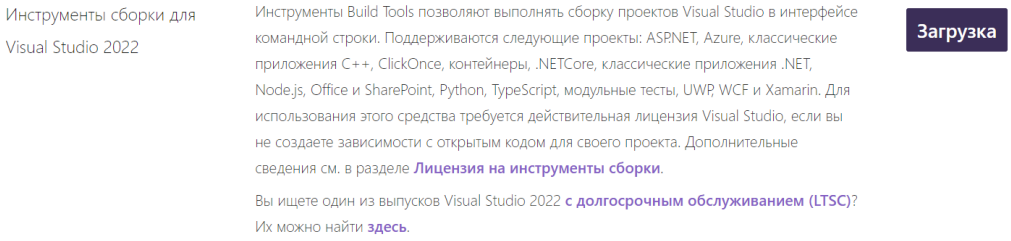

Рисунок 36 – Инструменты сборки для Visual Studio 2022

После загрузки установщика пакетов (vs_BuildTools.exe), нужно выбрать в нем набор пакетов для «разработки классических приложений на C++» (рисунок 37).

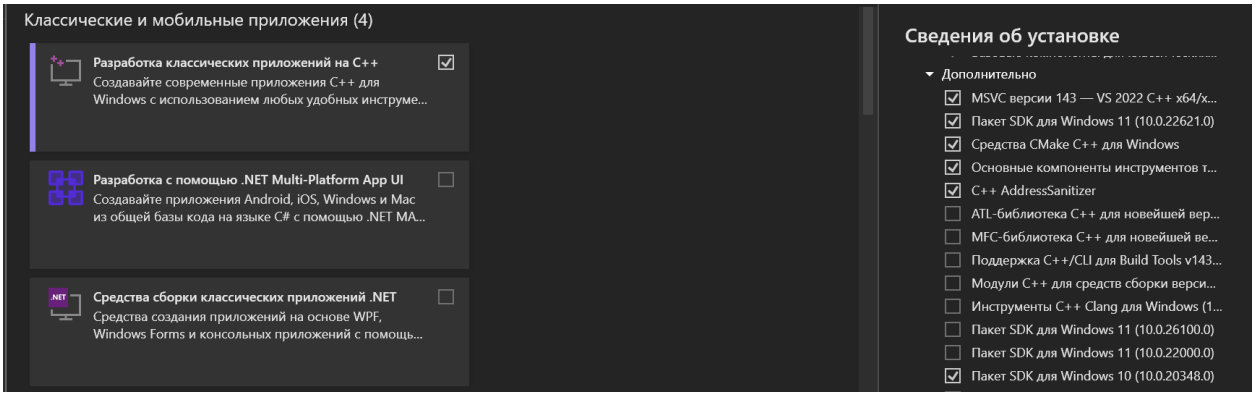

Рисунок 37 – Выбранный пакет инструментов сборки для классических приложений

2.6 Подготовка шаблона проекта для сборки и прошивки через VSCode

Для создания проекта в первую очередь необходимо получить его шаблон. После установке Moun River Studio, шаблоны проектов будут располагаться в директории «<MounRiver>\MounRiver_Studio\template\wizard», а для конкретно микроконтроллера CH582M в дирректории «<MounRiver>\MounRiver_Studio\template\wizard\WCH\RISC-V\CH58X\NoneOS\CH582M.zip» (рисунок 38).

Архив необходимо извлечь в папку проекта, в любой дирректории не содержащей в пути пробелов и кириллицы (рисунок 39).

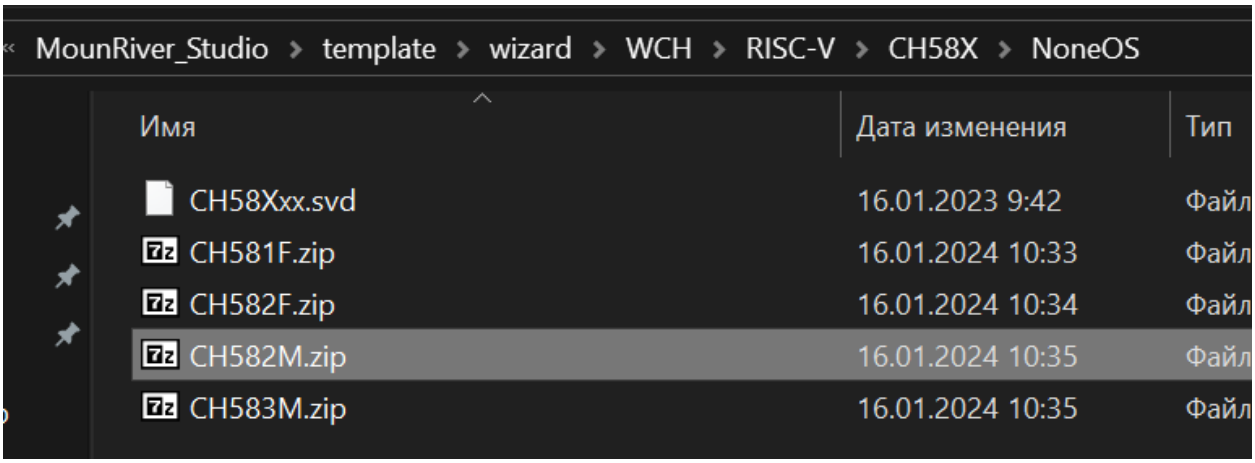

Рисунок 38 – Папка с шаблонами проектов

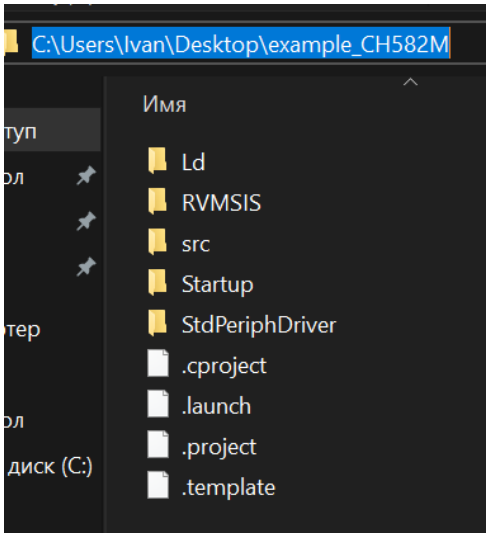

Рисунок 39 – Скопированный шаблон проекта

Дальше папку с проектом можно открывать в Visual Studio Code, в ней необходимо создать директории «build» и «.vscode», а также файлы «CmakeLists.txt» и «.vscode/tasks.json» (рисунок 40).

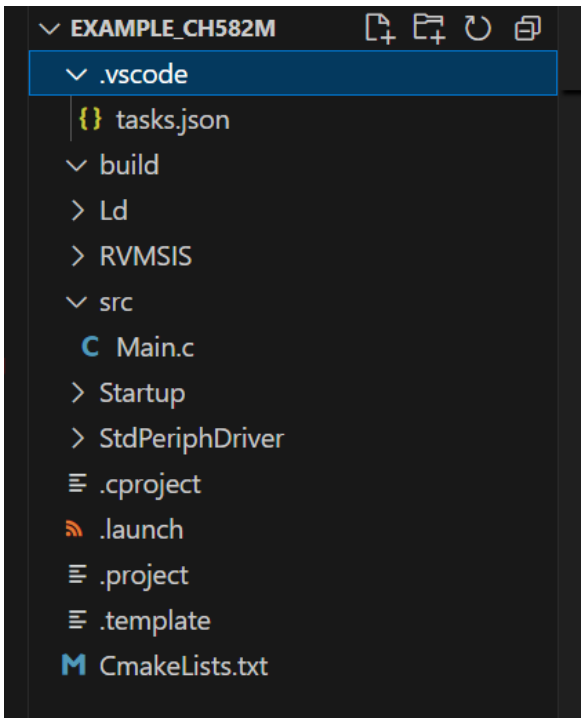

Рисунок 40 – полученная структура проекта в среде VSCode

CMakeLists отвечает за алгоритм действий сборщика CMake, его структуру я привел в листинге 5. В файл необходимо подключить набор инструментов GCC («toolchain»), в строке «set(TOOLPATH …)». Я взял набор из установленной MounRiver, в дирректории «C:/MounRiver/MounRiver_Studio/toolchain/RISC-V Embedded GCC12/bin/».

Листинг 5 – Содержимое файла CMakeLists.txt

```
set(CMAKE_SYSTEM_NAME Generic)
set(CMAKE_SYSTEM_VERSION 1)
set(CMAKE_TRY_COMPILE_TARGET_TYPE "STATIC_LIBRARY")
cmake_minimum_required(VERSION 3.20)

# Настройки набора инструментов
set(TOOLPATH "C:/MounRiver/MounRiver_Studio/toolchain/RISC-V Embedded
GCC12/bin/riscv-none-elf-")

if (WIN32)
    MESSAGE(STATUS "Now is windows!")
    set(CMAKE_C_COMPILER ${TOOLPATH}gcc.exe)
    set(CMAKE_CXX_COMPILER ${TOOLPATH}g++.exe)
    set(CMAKE_ASM_COMPILER ${TOOLPATH}gcc.exe)
    set(CMAKE_AR ${TOOLPATH}ar.exe)
    set(CMAKE_OBJCOPY ${TOOLPATH}objcopy.exe)
    set(CMAKE_OBJDUMP ${TOOLPATH}objdump.exe)
    set(SIZE ${TOOLPATH}size.exe)
elseif (UNIX)
    MESSAGE(STATUS "Now is UNIX-like OS!")
    set(CMAKE_C_COMPILER ${TOOLPATH}gcc)
    set(CMAKE_CXX_COMPILER ${TOOLPATH}g++)
    set(CMAKE_ASM_COMPILER ${TOOLPATH}gcc)
    set(CMAKE_AR ${TOOLPATH}ar)
    set(CMAKE_OBJCOPY ${TOOLPATH}objcopy)
    set(CMAKE_OBJDUMP ${TOOLPATH}objdump)
    set(SIZE ${TOOLPATH}size)
else ()
    MESSAGE(STATUS "Unsupported system!")
endif ()

set(CMAKE_TRY_COMPILE_TARGET_TYPE STATIC_LIBRARY)

# Настройки проекта
project(myProgram C CXX ASM) # имя проекта (имя собранного файла, я использую
workspaceFolderBasename)
set(CMAKE_CXX_STANDARD 11)
set(CMAKE_C_STANDARD 99)

# Уровень компиляции
add_compile_options(-O3)

# Уровень компиляции информации
add_compile_options(-Wall)

# Путь к файлу заголовка
include_directories(
    RVMSIS
    StdPeriphDriver/inc
)
```

```
# Макрос
add_definitions(-DDEBUG=1)

# Файл с исходным кодом
file(GLOB_RECURSE SOURCES
    "Startup/*.S"
    "RVMSIS/*.c"
    "StdPeriphDriver/*.c"
    "src/Main.c"
)

# Параметры линковки
set(LINKER_SCRIPT  ${CMAKE_SOURCE_DIR}/Ld/Link.ld)
add_link_options(-nostartfiles
                 -Xlinker --gc-sections
                 -Wl,--print-memory-usage
                 -Wl,-Map,${PROJECT_NAME}.map
                 --specs=nano.specs
                 --specs=nosys.specs)
add_link_options(-T ${LINKER_SCRIPT})

# Компиляция исполняемого файла
add_executable(${PROJECT_NAME}.elf ${SOURCES} ${LINKER_SCRIPT})

# Ссылка на статическую библиотеку
target_link_libraries(
    ${PROJECT_NAME}.elf
    ${CMAKE_SOURCE_DIR}/StdPeriphDriver/libISP583.a
)

# Вывод "hex" и "bin".
set(HEX_FILE ${PROJECT_BINARY_DIR}/${PROJECT_NAME}.hex)
set(BIN_FILE ${PROJECT_BINARY_DIR}/${PROJECT_NAME}.bin)
add_custom_command(TARGET ${PROJECT_NAME}.elf POST_BUILD
    COMMAND ${CMAKE_OBJCOPY} -Oihex $<TARGET_FILE:${PROJECT_NAME}.elf> ${HEX_FILE}
    COMMAND ${CMAKE_OBJCOPY} -Obinary $<TARGET_FILE:${PROJECT_NAME}.elf> ${BIN_FILE}
)
```

Также можно использовать установленый ранее набор инструментов однако тогда при компиляции будет некорректно отображаться размер занимаемой программой RAM микроконтрллера. Также для установленного GCC необходимо поменять следующие строки:

В tasks.json

```
"'program ${input:workspaceFolderForwardSlash}/build/myProgram-ninja.hex verify reset exit'"],
```

поменял на

```
"'program ${input:workspaceFolderForwardSlash}/build/myProgram.hex verify reset exit'"],
```

В CmakeLists.txt

```
set(TOOLPATH "C:/MounRiver/MounRiver_Studio/toolchain/RISC-V Embedded GCC12/bin/riscv-none-elf-")
```

Заменить на

```
set(TOOLPATH "C:/Program Files/xpack-riscv-none-elf-gcc-13.2.0-2/bin/riscv-none-elf-")
```

А также

```
# Уровень компиляции
add_compile_options(-O3)
```

Заменить на

```
# Уровень компиляции
add_compile_options(-O3 -march=rv32g)
```

2.7 Компиляция и прошивка программы в Visual Studio Code

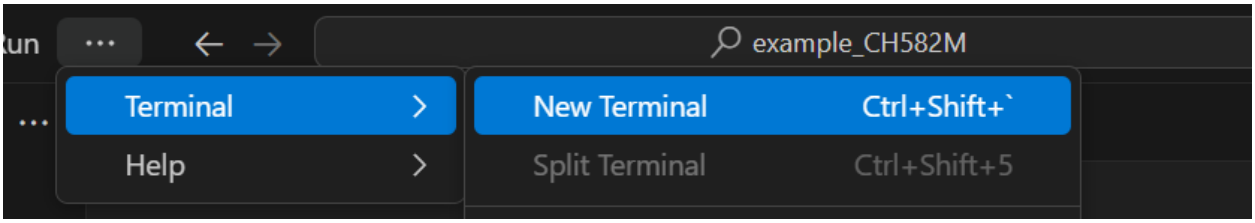

Рисунок 41 – Запуск терминала в VSCode

В папке «src» расположен основной файл программы Main.c. Чтобы скомпилировать программу в исполняемый файл, необходимо открыть терминал (рисунок 41) и выполнить последовательность команд:

- «cd build», чтобы прейти в директорию, в которой будет производиться сборка проекта
- «del * -Recurse -Force», если компиляцию потребуется сделать несколько раз необходимо удалить предыдущие файлы.
- «cmake -GNinja ..», чтобы запустить сборку проекта (рисунок 42)
- «ninja», чтобы запустить компиляцию исходного кода в бинарные файлы

```
PS C:\Users\Ivan\Desktop\example_CH582M> cd build
PS C:\Users\Ivan\Desktop\example_CH582M\build> cmake -GNinja ..
-- Now is windows!
-- The C compiler identification is GNU 12.2.0
-- The CXX compiler identification is GNU 12.2.0
-- The ASM compiler identification is GNU
-- Found assembler: C:/MounRiver/MounRiver_Studio/toolchain/RISC-V Embedded GCC12/b
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: C:/MounRiver/MounRiver_Studio/toolchain/RISC-V Emb
gcc.exe - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: C:/MounRiver/MounRiver_Studio/toolchain/RISC-V E
.exe - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Configuring done (1.5s)
-- Generating done (0.1s)
-- Build files have been written to: C:/Users/Ivan/Desktop/example_CH582M/build
PS C:\Users\Ivan\Desktop\example_CH582M\build>
```

Рисунок 42 – Вывод cmake в терминале в процессе сборки проекта

После выполнения команды «ninja» vscode скрипт покажет обьем заполненной памяти (рисунок 43), а в директории build можно будет найти скомпилированные файлы (рисунок 44).

```
Memory region         Used Size  Region Size  %age Used
           FLASH:        3464 B       448 KB      0.76%
             RAM:        3356 B        32 KB     10.24%
```

Рисунок 43 – объем занятой программой памяти

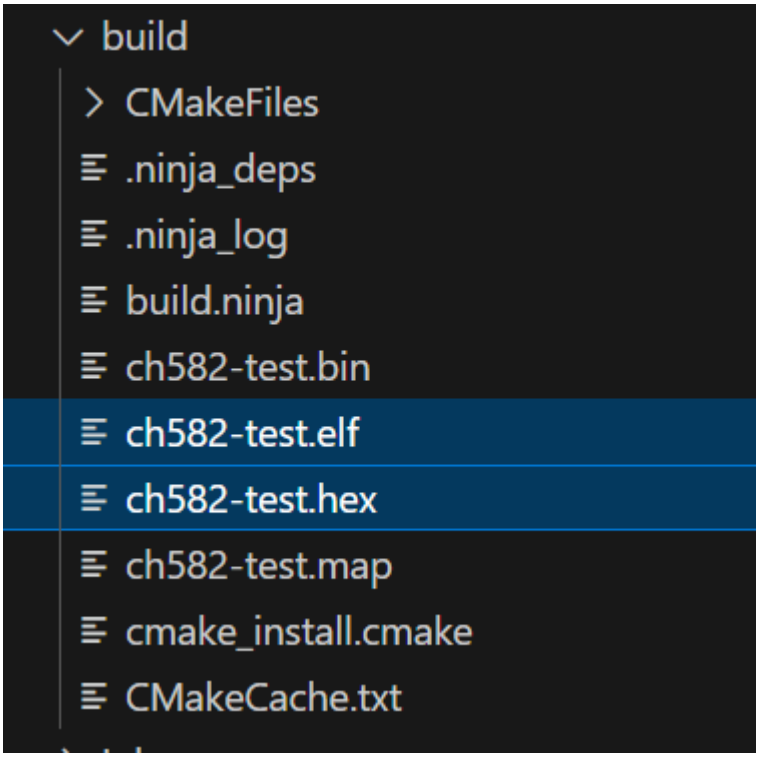

Рисунок 44 – Скомпилированные «elf» и «hex» файлы в дирректории «build»

После получения исходных файлов «elf» и «hex», можно подключать программатор чтобы записать на микроконтроллер бинарный файл прошивки.

Для этого в терминале необходимо запустить OpenOCD с помощью команды «openocd.exe -f interface/wch-riscv.cfg -c "program C:/Users/Ivan/Desktop/example_CH582M/build/ch582-test.hex verify reset exit"».

В качестве аргументов команды указываются файл скрипта «wch-riscv.cfg» и полный путь до бинарного файла прошивки (рисунок 45).

```
PS C:\Users\Ivan\Desktop\example_CH582M\build> openocd.exe -f interface/wch-riscv.cfg -c "program C:/Users/Ivan/Desktop
/example_CH582M/build/ch582-test.hex verify reset exit"
Open On-Chip Debugger 0.11.0+dev-02415-gfad123a16-dirty (2024-01-24-13:40)
Licensed under GNU GPL v2
For bug reports, read
        http://openocd.org/doc/doxygen/bugs.html
Info : only one transport option; autoselect 'sdi'
Warn : Transport "sdi" was already selected
Ready for Remote Connections
```

Рисунок 45 – вывод OpenOCD при программировании микроконтроллера

После того как программа по умолчанию была успешна прошита я решил поменять код Main.c на программу включения светодиода через консоль, из-за убранного алгоритма работы UART через прерывания обьем используемой программой памяти сократился (рисунок 46).

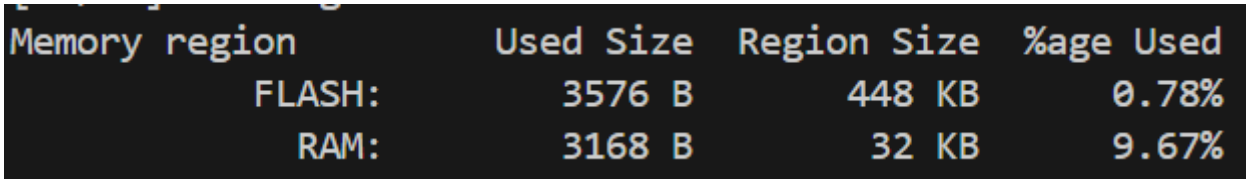

Рисунок 46 – объём памяти занятый программой включения светодиода через консоль

2.8 Настройка Serial Monitor в VSCode

Теперь чтобы проверить работу записанной в микроконтроллер программы необходимо открыть монитор порта, через который с микроконтроллером можно будет обмениваться данными. Для проверки прошивки можно открыть порт в MounRiver, как я показывал выше, или даже через монитор порта в среде Arduino (рисунок 47) (Плату можно при этом выбрать любую, так как микроконтроллер мы прошивать не будем). Удобство этого способа в отличие от MounRiver состоит в том, что из Arduino IDE можно отправлять строку из нескольких симовлов без переноса строки.

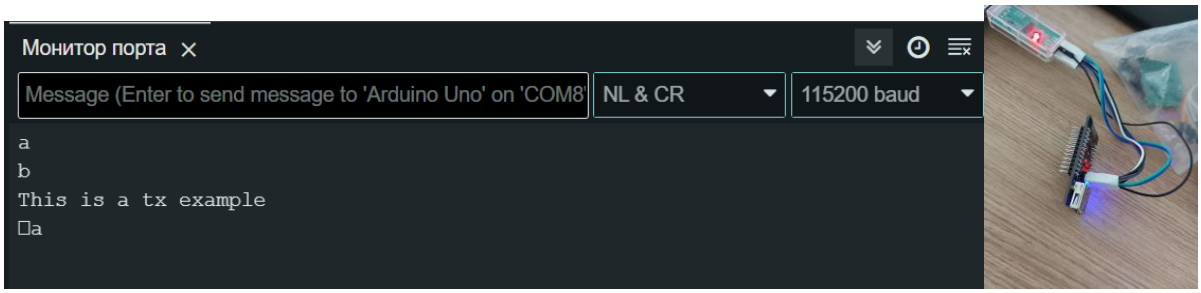

Рисунок 47 – Процесс управления светодиодом на отладочной плате через монитор порта ArduinoIDE

Однако Visual Stuio Code тоже имеет специальный плагин для отправки и приема данных по последовательному порту, называется он Serial Monitor (рисунок 48).

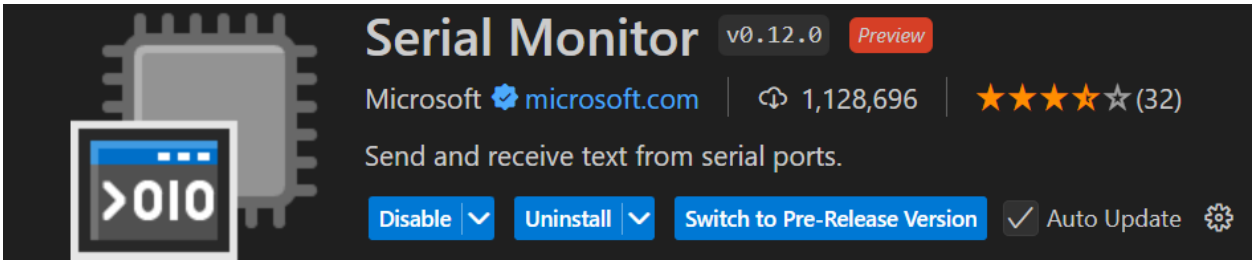

Рисунок 48 – Плагин для VSCode, «Serial Monitor»

После установки плагина, для начала работы необходимо нажать кнопку «Start Monitoring», а также выставить остальные параметры как на рисунке 49.

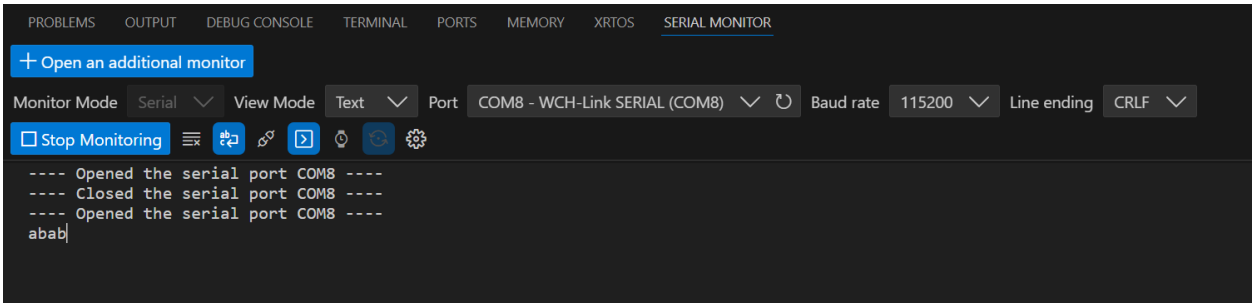

Рисунок 49 – Интерфейс плагина «Serial Monitor»

Кнопкой «Toggle terminal mode» (рисунок 50) можно переключать режим отправки введенных данных, посимвольная отправка как в MounRiverStudio или оправка введенной строки (рисунок 51).

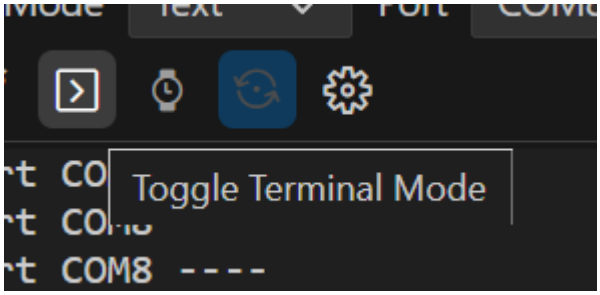

Рисунок 50 – Кнопка «Toggle terminal mode»

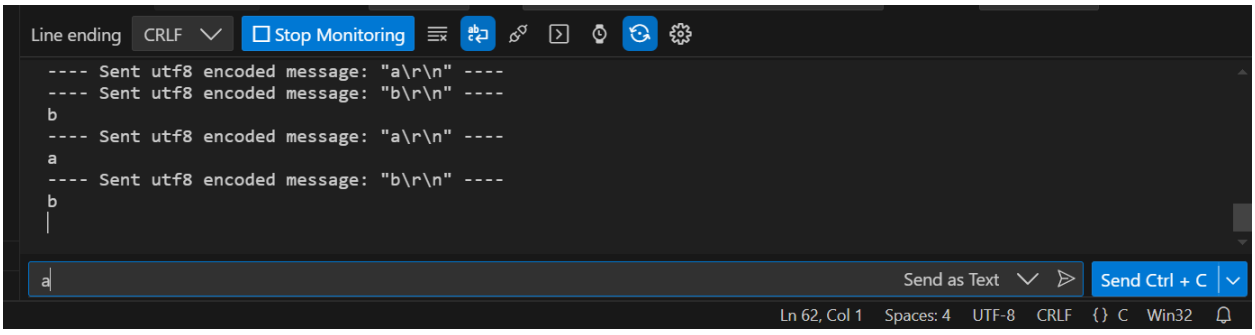

Рисунок 51 – Пример работы с платой с построчной передачей команд

2.9 Настройка задач «tasks.json» для автоматической сборки и загрузки программы

Теперь для удобства можно добавить в visual studio макросы для более удобной компиляции и прошивки данного проекта.

Для этого необходимо настройть файл «tasks.json», содержимое файла представлено в листинге 6.

Листинг 6 – Файл «tasks.json»

```
{
  "version": "2.0.0",
  "tasks": [
    // Сборка проекта
    {
      "type": "shell",
      "label": "build cmake+ninja",
      "dependsOrder": "sequence",
      "dependsOn": [
          "clear_build_folder",
          "Cmake",
          "runNinja"
      ],
      "group": {
          "kind": "build",
          //"isDefault": true
      }
    },

    {
      "label": "clear_build_folder",
      "type": "shell",
      "command": "del * -Recurse -Force",
      "options": {
          "cwd": "${workspaceFolder}/build",
      }
    },
    {
      "label": "Cmake",
      "type": "shell",
      "command": "cmake -GNinja ..",
      "options": {
          "cwd": "${workspaceFolder}/build",
      },
    },
    {
      "label": "runNinja",
      "type": "shell",
      "command": "ninja",
      "options": {
          "cwd": "${workspaceFolder}/build",
      },
    },

    // загрузка на микроконтроллер
    {
        "type": "shell",
        "label": "flash_controller",
        "group": "build",
        "command": "openocd",
```

```
        "args": ["-f", "interface/wch-riscv.cfg", "-c",
        "'program ${input:workspaceFolderForwardSlash}/build/myProgram-ninja.hex
verify reset exit'"],
    }
  ],
  "inputs": [
    {
      "id": "workspaceFolderForwardSlash",
      "type": "command",
      "command": "extension.commandvariable.transform",
      "args": {
        "text": "${workspaceFolder}",
        "find": "\\\\",
        "replace": "/",
        "flags": "g"
      }
    }
  ]
}
```

Настройка данного файла позволяет автоматизировать консольные команды, рассмотренные выше, для сборки, компиляции и загрузки программы на микроконтроллер.

После сохранения файла при нажатии сочетания клавиш «Ctrl» + «Shift» + «B» появляется меню макросов (рисунок 52).

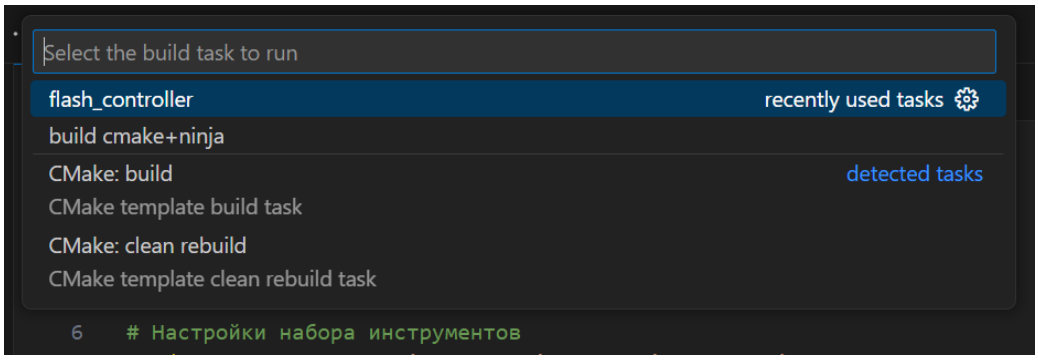

Рисунок 52 – Меню макросов сборки проекта

Настройкой файла «tasks.json» я добавил 2 новых пункта:

– «build cmake+ninja», выполняет очистку дирректории «build», а затем команды сборки «cmake -GNinja ..» и «ninja»;

– «flash_controller», выполняет команду прошивки микроконтроллера через openOCD.

Вывод

Микроконтроллер CH582M от компании WCH имеет отличные характеристики и должен отлично показать себя в перспективе разработки моего дипломного проекта. В этой научно-исследовательской работе я рассмотрел работу с этим микроконтроллером в среде MounRiver от компании WCH, а также в специально настроенной среде Visual Studio Code. Я разобрал принцип работы базовой программы по передаче данных через последовательный интерфейс, а также отдельные функции этой программы. У микроконтроллера замечательная и понятная база готового кода (с учетом наличия переводчика с китайского языка), также в сети есть большое количество готовых проектов использующих совершенно разную периферию микроконтроллера, из которых можно брать фрагменты кода.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. MounRiver Studio download [Электронный ресурс]. URL: http://mounriver.com/download (Дата обращения: 30.05.2024)
2. MounRiver Studio Quick Start [Электронный ресурс]. URL: http://mounriver.com/help (Дата обращения: 30.05.2024)
3. WCH-Link Emulation Debugger Module [Электронный ресурс]. URL: https://www.wch-ic.com/products/WCH-Link.html (Дата обращения: 30.05.2024)
4. Загрузка WCH-LinkUtility.ZIP [Электронный ресурс]. URL: https://www.wch.cn/downloads/WCH-LinkUtility_ZIP.html (Дата обращения: 30.05.2024)
5. «SoCXin»-«CH583» examples [Электронный ресурс]. URL: https://github.com/SoCXin/CH583/tree/master/src/EVT/EXAM (Дата обращения: 30.05.2024)
6. CH583 evaluation [Электронный ресурс]. URL: https://www.wch.cn/downloads/CH583EVT_ZIP.html (Дата обращения: 30.05.2024)
7. «SoCXin»-«CH583» USB Device COM-порт [Электронный ресурс]. URL: https://github.com/SoCXin/CH583/blob/master/src/EVT/EXAM/USB/Device/COM/src/Main.c (Дата обращения: 30.05.2024)
8. SoCXin»-«CH583» USB-HID [Электронный ресурс]. URL: https://github.com/SoCXin/CH583/blob/master/src/EVT/EXAM/USB/Device/HID_CompliantDev/src/Main.c (Дата обращения: 30.05.2024)
9. «SoCXin»-«CH583» BLE-HID Keyboard [Электронный ресурс]. URL: https://github.com/SoCXin/CH583/tree/master/src/EVT/EXAM/BLE/HID_Keyboard (Дата обращения: 30.05.2024)
10. «SoCXin»-«CH583» BLE-UART [Электронный ресурс]. URL: https://github.com/SoCXin/CH583/tree/master/src/EVT/EXAM/BLE/BLE_UART (Дата обращения: 30.05.2024)

11. «SoCXin»-«CH583» Power Management [Электронный ресурс]. URL: https://github.com/SoCXin/CH583/tree/master/src/EVT/EXAM/PM (Дата обращения: 30.05.2024)

12. «SoCXin»-«CH583» In Application Programming [Электронный ресурс]. URL: https://github.com/SoCXin/CH583/tree/master/src/EVT/EXAM/IAP (Дата обращения: 30.05.2024)

13. «VSCode+CMake+openocd для программирования и отладки ch32» [Электронный ресурс]. URL: https://habr.com/ru/articles/786872/ (Дата обращения: 30.05.2024)

14. riscv-none-elf-gcc-xpack [Электронный ресурс]. URL: https://github.com/xpack-dev-tools/riscv-none-elf-gcc-xpack (Дата обращения: 30.05.2024)

15. xPack GNU RISC-V Embedded GCC v14.2.0-2 [Электронный ресурс]. URL: https://github.com/xpack-dev-tools/riscv-none-elf-gcc-xpack/releases (Дата обращения: 30.05.2024)

16. CMake Download [Электронный ресурс]. URL: https://cmake.org/download/ (Дата обращения: 30.05.2024)

17. ninja releases [Электронный ресурс]. URL: https://github.com/ninja-build/ninja/releases (Дата обращения: 30.05.2024)

18. Configure VS Code for Microsoft C++ [Электронный ресурс]. URL: https://code.visualstudio.com/docs/cpp/config-msvc (Дата обращения: 30.05.2024)

19. Visual Studio 2022 download https://visualstudio.microsoft.com/ru/downloads/ (Дата обращения: 30.05.2024)