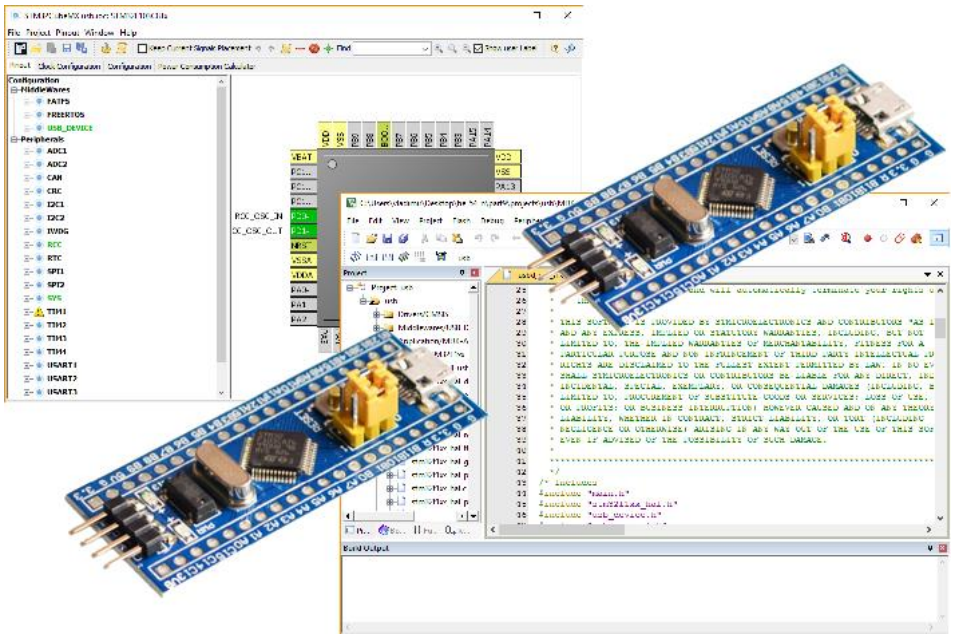
В.Н. Гололобов

Несколько простых опытов с модулем STM32F103C8



Москва – 2017

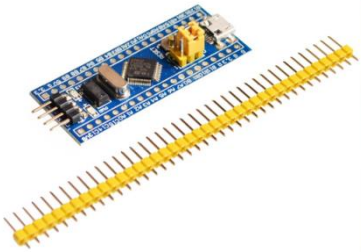
Оглавление

Кто-то из радиолюбителей, освоив микроконтроллеры PIC или AVR, собрался реализовать проект, который «не помещается» в эти микросхемы. Их внимание привлекают более мощные микроконтроллеры. Одним из таких микроконтроллеров может стать STM32, недорогой, но достаточно мощный представитель этих устройств.

Есть и другие радиолюбители, которые всегда стремятся использовать самые новые элементы в своей практике. И им может понравиться STM32.

Что нужно для первого знакомства с этим микроконтроллером?

Конечно, в первую очередь сам контроллер. На Aliexpress есть недорогие модули, например: STM32F103C8T6 ARM STM32, +CortexM Evaluation Board, STM32F103C8T6 STM32 STM32F103.



У меня модуль, показанный первым, поэтому для него я точно знаю, что нужен программатор и кабель для подключения к USB. Есть два варианта программирования: переходник USB/TTL (с напряжением 3.3В) и ST-Link, программатор и отладчик. А кабель нужен с microUSB для подключения к модулю. Нужен ли программатор для других вариантов, показанных выше, я не знаю. По описанию продавца это трудно понять.

Я не думаю, что все радиолюбители склонны начинать освоение микроконтроллера с изучения его описания на английском языке, с чтения руководств к программам и обдумыванию прочитанного. Поэтому предлагаю начать с очень простых экспериментов. Проделав их, вы можете определиться с собственными проектами, для реализации которых можно заглянуть и в описание контроллера, и в руководства.

1 Программы для работы с модулем

Кроме модуля, кабеля и программатора вам понадобятся программы. Сред разработки есть несколько, вы можете познакомиться с ними, но я предлагаю такой набор для начала: STM32CubeMx и Keil uVision.

Первая программа позволяет сделать выбор выводов, настроить конфигурацию и создать шаблон для реализации проекта.

Если вы заглянули в описание контроллера, вы поняли, что он существенно сложнее тех моделей, с которыми вы имели дело раньше. И подспорье в виде программы STM32CubeMx не будет лишним.

Начнём по порядку. Программу STM32CubeMx вы можете скачать на сайте производителя:

http://www.st.com/en/development-tools/stm32cubemx.html

Если вместе с программой вы получите и программу для программирования через USART, то хорошо, если нет, то поищите там же программу: *STM32 Flash Loader Demonstrator*.

Предположив, что вы будете использовать ST-Link, я советую вам запастись программой и драйвером, которые можно найти там же, введя в строку поиска по сайту: ST-Link. Выглядит это сегодня как STSW-LINK009. Это архив с нужной программой и драйвером для работы с St-Link.

Программу Keil uVision (сегодня это 5 версия) можно найти (как MDK-ARM) на сайте:

https://www.keil.com/download/product/

Бесплатная версия имеет ограничение до 32 кБайт, но для начала этого достаточно.

Устанавливаются все программы обычным образом. Если при установке программ вы получите предложение установить драйвер для ST-Link, то не отказывайтесь, позже пригодится.

Ещё раз напомню, что для программирования через USART приобретайте переходник USB/TTL такого вида:

Рис. 1.1. Переходник USB/TTL

У него есть переключение 5В и 3.3В. Я пробовал программировать при питании 5В, поскольку микроконтроллер не боится сигналов этого напряжения, но у меня ничего не получилось (позже стало, впрочем, получаться). И, пока не забыл, вам понадобятся провода для соединения выводов переходника и модуля STM32. Их тоже можно найти на сайте Aliexpress, если в поиск ввести: провода-перемычки.

Кабель для соединения модуля с портом USB, если вы выбрали такой же модуль, как это сделал я, тоже имеет особенность. Я надеялся, что подойдёт тот, что остался у меня от модуля Arduino Nano, но ошибся, поскольку у него был разъём, видимо, Mini, а не Micro. Для поиска на Aliexpress можно в строку поиска ввести: микро USB. Это будет нечто похожее на:

Рис. 1.2. USB кабель для STM32 с микроUSB для подключения к модулю

Собираясь заняться микроконтроллером STM32, закажите всё сразу. Я, например, не уверен, что программатор ST-Link позволяет снять защиту от программирования, если вы ненароком её установили. Программирование через USART позволяет это сделать.

2 Управление выходом

Вы дождались, когда пришли все посылки с заказанными модулем, проводами, переходниками и т.п., вы скачали все программы и теперь готовы приступить к работе. Возможно, вы уже почитали описание микроконтроллера, полистали руководства к программам. Я не исключаю, что и мои советы вам не нужны. Тем не менее…

Первый проект, кто бы мог подумать, будет «помигать светодиодом». Это не «из принципа», просто так вам будет понятнее, что вас ожидает. Итак. Запускаем программу STM32CubeMx.

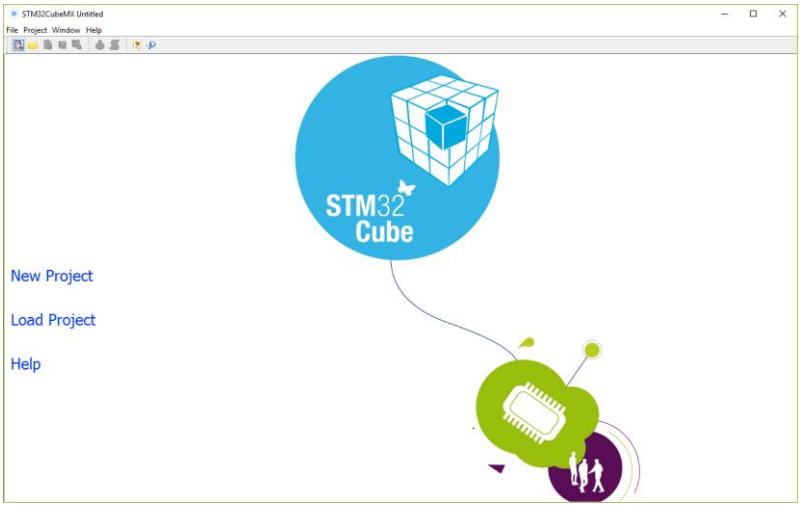


Рис. 2.1. Запуск программы STM32CubeMx

Выбираем из списка слева *New Project*. Попадаем в длинный список моделей. Чтобы облегчить себе выбор, воспользуемся заданием линейки микроконтроллеров.

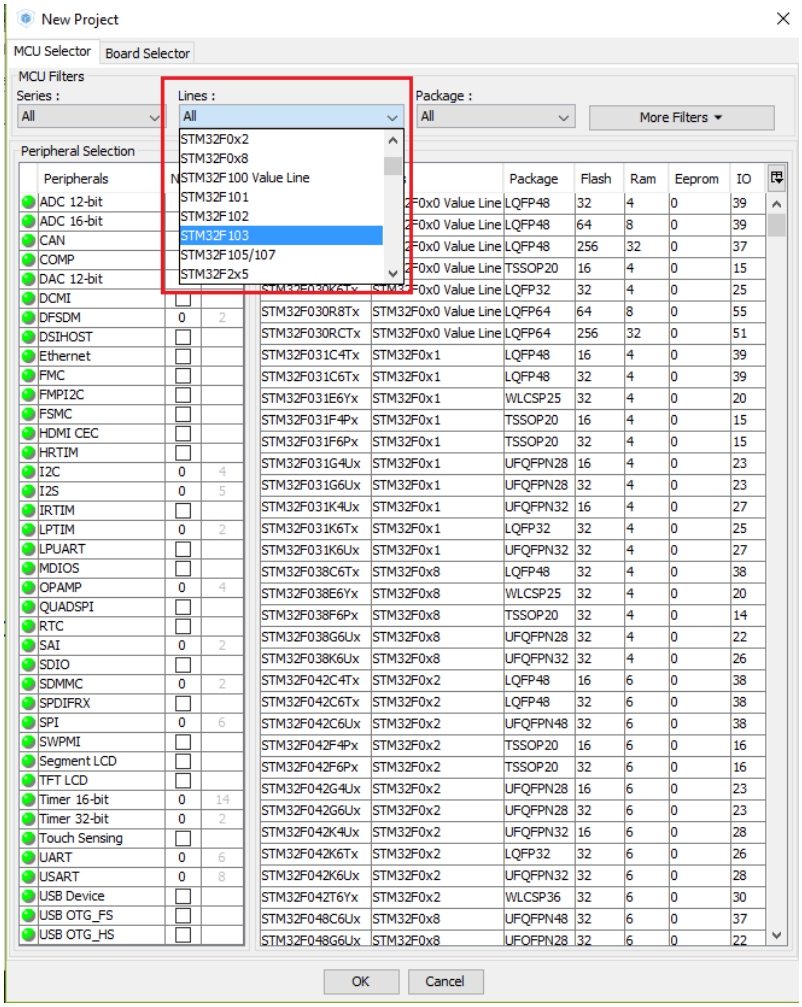


Рис. 2.2. Диалог выбора модели микроконтроллера

Теперь нужная модель появляется в начале списка. Выделяем её в списке и нажимаем кнопку **ОК**. В рабочем поле видна выбранная модель. Не трогая закладок, не раскрывая дерево периферии в левом окне, выделим вывод PC13 щелчком левой клавиши мышки, а из выпадающего меню выберем назначение вывода в качестве выхода (к этому выводу присоединён светодиод).

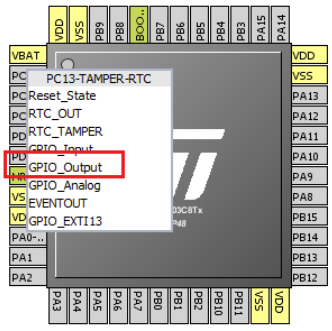


Рис. 2.3. Назначение вывода на выход

В данном случае не будем больше ничего делать, но запустим генерацию шаблона проекта: *Project->Generate Code*. В появившемся диалоге выберем место хранения проекта, название проекта и среду разработки, для которой нужно создать заготовку программы.

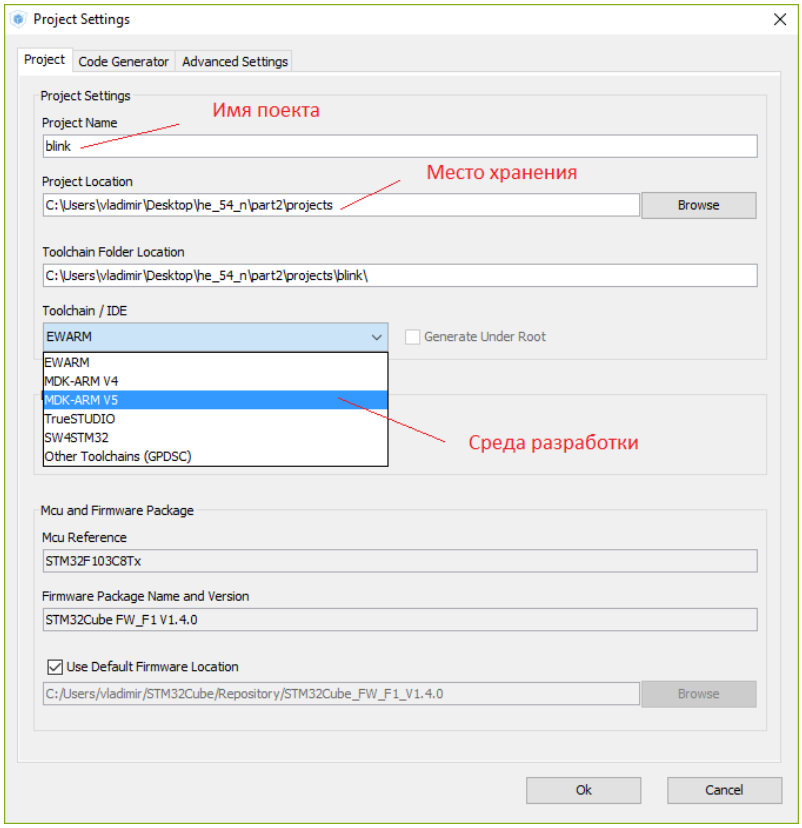


Рис. 2.4. Выбор среды разработки для генерации заготовки программы

Когда кнопка **OK** нажата, начинается создание шаблона, которое завершается предложением закрыть программу, открыть папку с проектом или открыть проект. После выбора *«Open Project»* начинается загрузка программы Keil.

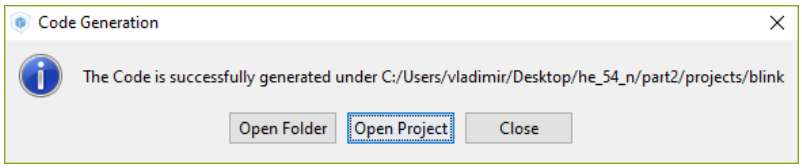


Рис. 2.5. Диалоговое окно, завершающее генерацию шаблона проекта

В появившемся окне программы мы увидим слева окошко навигатора по проекту, где можно найти файл *main.c*, который и откроем двойным щелчком левой клавиши мышки.

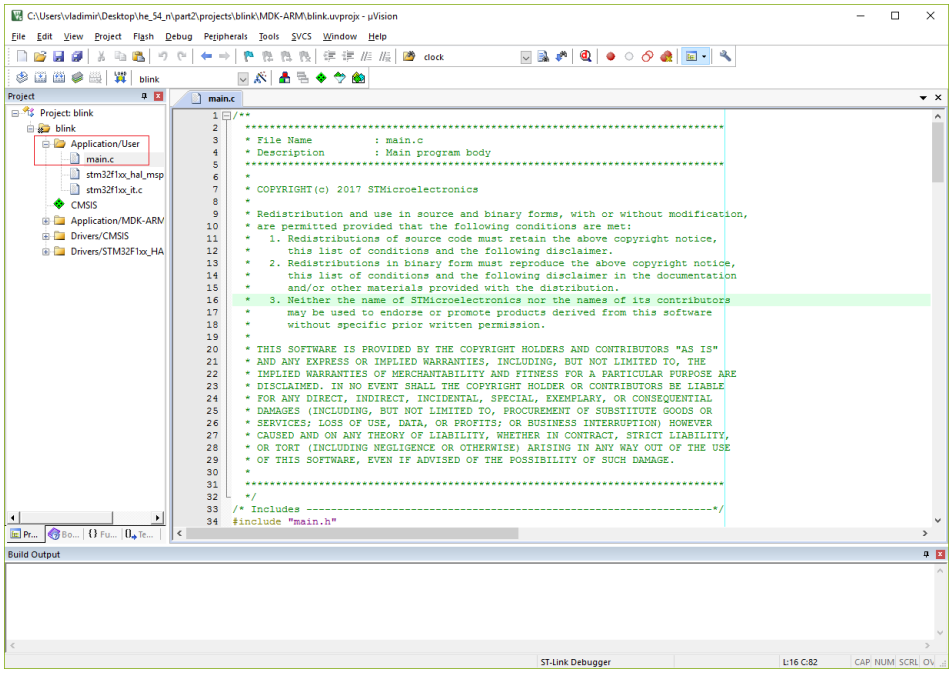


Рис. 2.6. Место, где можно найти основной файл программы

Перемещаясь по тексту шаблона к циклу *while*, обратите внимание, что шаблон текста программы размечен. Всё, что вы добавите в программу, должно лежать в тех местах, которые отмечены в тексте. Добравшись до нужного нам места, впишем две строки:

```
  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    GPIOC->ODR ^= GPIO_PIN_13;
        HAL_Delay(1000);
  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */

  }
  /* USER CODE END 3 */

}
```

```
GPIOC->ODR ^= GPIO_PIN_13;

HAL_Delay(1000);
```

2.7. Место основного цикла программы

Эти две строки результат поиска в Интернете, где я их и нашёл среди советов знающих людей. Те, кто привык к языку C++, не нуждаются в моих пояснениях. Они нужны только радиолюбителям, привыкшим к языку Си для PIC и AVR контроллеров.

GPIOC – означает порт C структуры вводов/выводов общего назначения. Доступ к элементам класса или структуры может быть осуществлён двумя операциями – операцией точка и операцией стрелка. Доступ через объект осуществляется с помощью операции точка, а доступ же через указатель на объект осуществляется через операцию стрелка. Если сместиться ниже в файле *main.c*, то можно найти такую строку:

```
GPIO_InitStruct.Pin = GPIO_PIN_13;
```

в этом случае доступ к свойству структуры осуществляется с помощью оператора точки.

GPIOx_ODR – это регистр вывода данных.

Оператор ^= побитовое исключающее «ИЛИ» совместно с оператором присваивания. А то, как можно переключать вывод микроконтроллера, вы найдёте, если внимательно просмотрите файл *stm32f1xx_hall_gpio.c*, который обнаружите, если раскроете раздел дерева проекта в части…

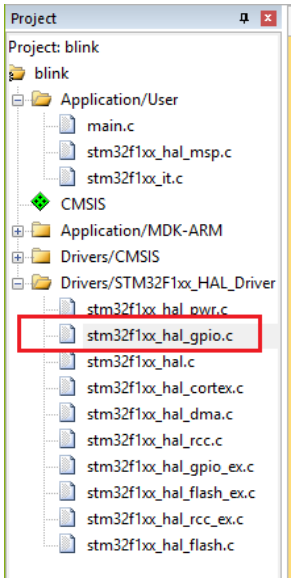


Рис. 2.8. Файл с настройками и функциями вводов-выводов общего назначения

Чтобы открыть ветку дерева проекта, достаточно щёлкнуть левой клавишей мышки по значку плюс слева от названия раздела. Просматривая файл, вы наверняка обнаружите функцию:

```
 void HAL_GPIO_TogglePin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)
{
  /* Check the parameters */
  assert_param(IS_GPIO_PIN(GPIO_Pin));
  GPIOx->ODR ^= GPIO_Pin;
}
```

Вам остаётся только переписать нужное выражение. Просмотр файлов, сгенерированных STM32CubeMx, очень часто даёт ответ на вопрос, как записать то или иное действие. Это относится к вопросу пауз. У компилятора для Keil нет привычных нам `delay_ms()`. Связано это, видимо, со сложной структурой тактирования периферии. Но есть пауза в миллисекундах, организованная на абстрактном уровне (HAL). Её мы и применили.

Далее. Для загрузки программы по USART нам нужен файл *.hex* (или *.bin*). Получить его можно, запустив сборку (постройку) проекта, но предварительно следует настроить проект с помощью диалога конфигурации:

Рис. 2.9. Кнопка инструментальной панели для настройки проекта

Можно использовать указанную выше иконку инструментального меню, можно использовать основное меню: *Project->Option for Target…*

В открывающемся диалоге на закладке *Output* достаточно установить флажок:

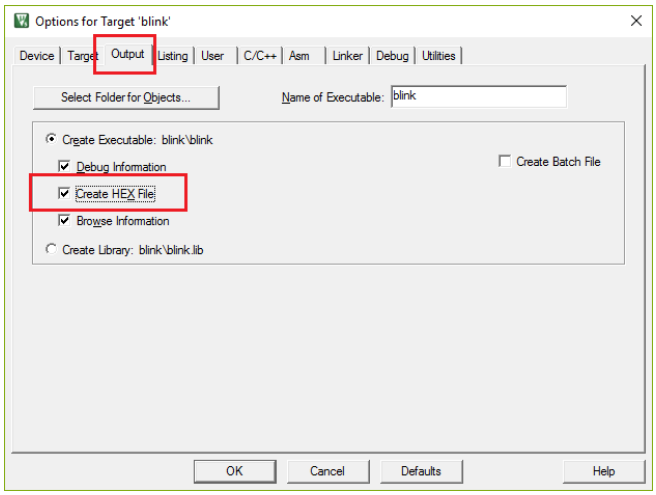


Рис. 2.10. Установка флага для создания hex-файла

Теперь можно запустить компиляцию и сборку проекта: *Project->Build Target*.

Если загрузку проекта осуществлять через USART, если вы используете такой же модуль, как использую я, то вам следует переместить перемычку boot0 (обычно соединение с землёй, а при программировании нужно соединение с «1»), включить кабели в гнёзда USB (возможно, нажать кнопку **reset** модуля) и ещё раз переключить кабель USB модуля.

Теперь можно запустить программу STM32 Flash Loader Demonstrator и выбрать виртуальный порт. Если всё сделано правильно, на светофоре вы увидите зелёный свет. Нажмите кнопку **Next**, чтобы увидеть модель своего микроконтроллера и его память программ. Следующее нажатие кнопки **Next** приведёт вас к странице, где вы должны выбрать hex-файл, используя кнопку навигации.

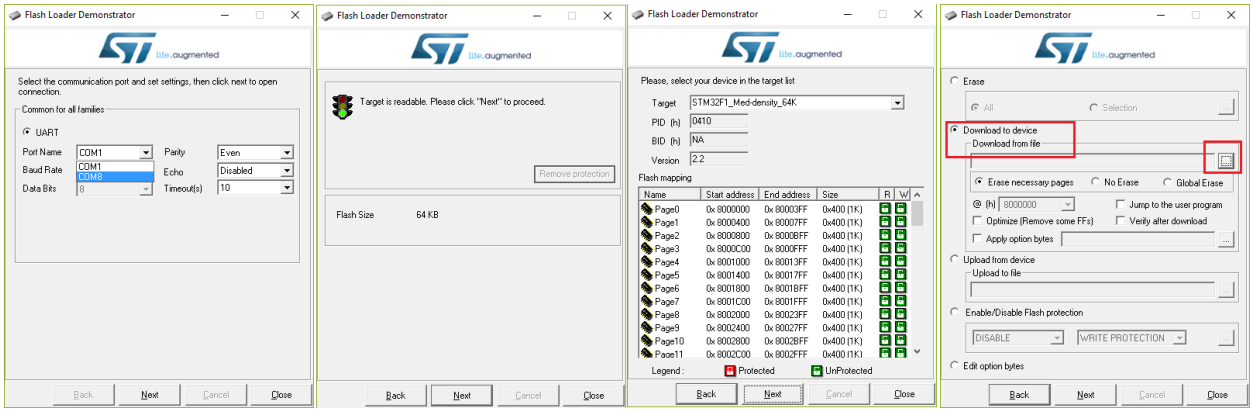


Рис. 2.11. Настройки программы программатора

Но в открывающемся окне проводника вам следует начать с того, чтобы выбрать hex-файл в списке расширений файлов:

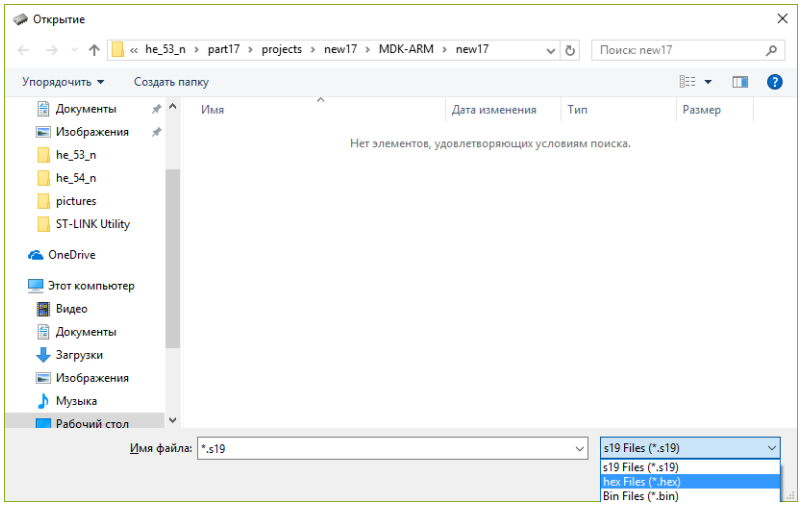


Рис. 2.12. Диалоговое окно выбора формата файла для загрузки в микроконтроллер

Теперь можно переместиться в папку проекта, где вы найдёте нужный файл: *C:\Users\vladimir\Desktop\...\projects\blink\MDK-ARM\blink*. Нажав кнопку проводника **Открыть**, вы вернётесь к программе загрузки, где клавиша **Next** загрузит вашу программу в модуль, о чём и сообщит вам на зелёном фоне окошка процесса загрузки. Переместите перемычку boot0 на место, нажмите кнопку **reset** на модуле, и вам остаётся наблюдать, как мигает светодиод, подключённый к выводу PC13.

Возвращаясь к программе Keil, можно добавить, что программа имеет симулятор отладчика, который следует настроить, повторив вызов диалога *Project->Build Target*. Теперь потребуется закладка *Debug*, на которой устанавливается флажок и нужно заменить настройки в окошках *Dialog.DLL* и *Parameter*:

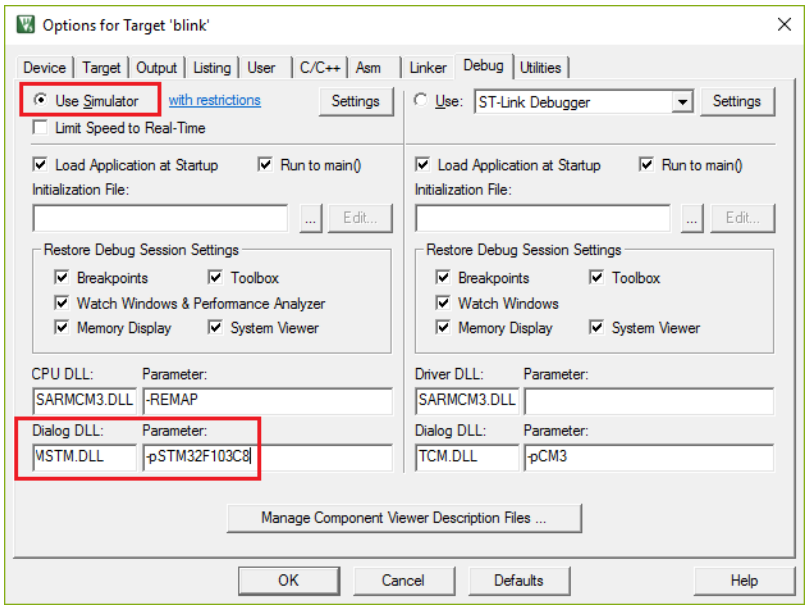


Рис. 2.13. Необходимая правка для работы отладчика с STM32F103

Вот эти параметры, их можно скопировать отсюда:

DARMSTM.DLL -pSTM32F103C8

Теперь можно запустить отладку: основное меню *Debug->Start/Stop Debug Session*. Там же в основном меню:

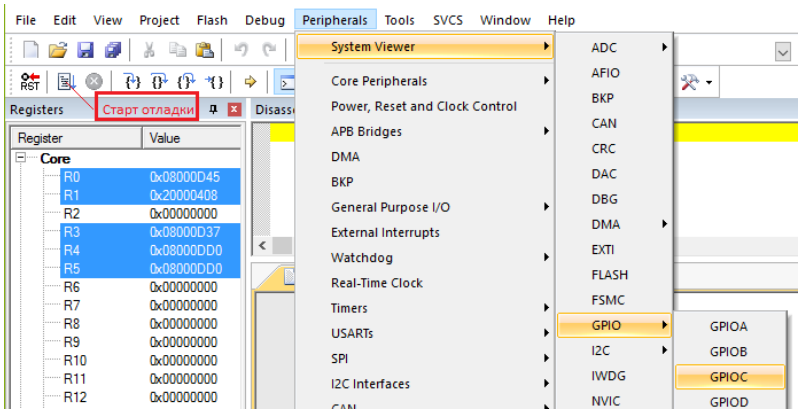


Рис. 2.14. Иконка запуска отладки и выбор окна данных порта C

В появившемся окне наблюдения за регистрами порта C после старта отладки можно наблюдать изменение состояния вывода:

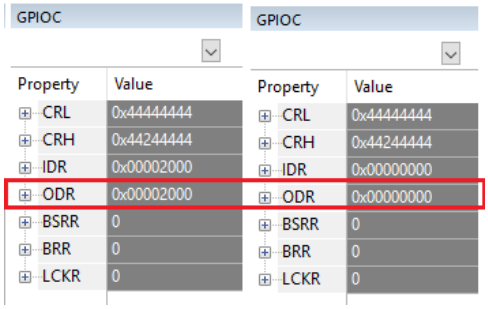


Рис. 2.15. Изменения состояния выводов порта

Многим программа «помигать светодиодом» кажется излишне простой, оскорбляющей их достоинство. Но, согласитесь, формирование таких сложных сигналов, как коммуникационные сигналы SPI, RS232, 1-wire и т.п., в своей основе имеют «помигать светодиодом», пусть и сложным образом. Да и основное назначение микроконтроллера – это включать и выключать что-то.

3 Кнопка на входе и светодиод на выходе

Мы научились (но не совсем) устанавливать вывод порта в высокое и низкое состояние. Теперь попробуем использовать кнопку для управления входом.

Начнём создание нового проекта с программы STM32CubeMx. Настроим вывод порта PC13 на выход, как делали это ранее, а вывод PC14 на вход (GPIO_Input).

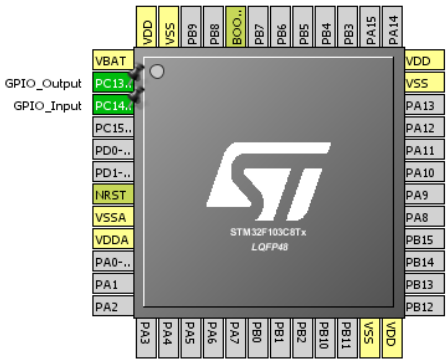


Рис. 3.1. Настройка выводов на вход и выход

Но теперь заглянем на закладку *Configuration*:

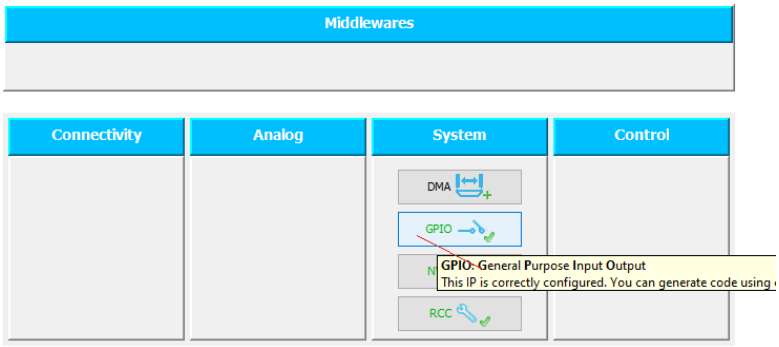


Рис. 3.2. Диалог для выбора конфигурации периферии

Зайдём в диалоговое окно *GPIO*:

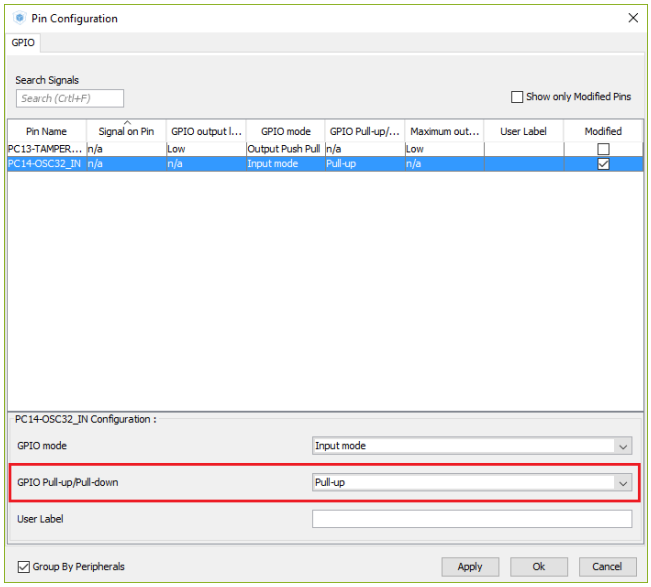


Рис. 3.3. Настройка подтягивающих резисторов

Выделив вывод PC14, настроим вход с подтягивающим резистором, используя выбор из выпадающего меню при нажатии кнопки со стрелкой справа.

Теперь можно сгенерировать заготовку программы и открыть проект в Keil.

Сейчас я хочу использовать запись, которая мне, сказать по правде, более привычна:

```
  while (1)
  {
      if (HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_14) == 0 ) {
          HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13,1);
      }
      else {HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13,0);}

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

Использование функций из файла *stm32f1xx_hall_gpio.c* мне понятнее. Это, увы, так. Однако после загрузки программы в модуль выяснилось, что программа работает так, как говорят: «с точностью до наоборот». Светодиод горит, когда кнопка не нажата, и гаснет при нажатой кнопке. Я пока не понимаю причину этого. Попробуем так:

```
  while (1)
  {
      if (HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_14) == 0 ) {
          HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13,GPIO_PIN_SET);
      }
      else {HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13,GPIO_PIN_RESET);}
}
```

Не помогло. Эту программу можно записать и иначе:

```
if (HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_14) == 0 )
    {
        (*GPIOC).BSRR = GPIO_BSRR_BS13;
    }
    else
        {
        GPIOC->BSRR = GPIO_BSRR_BR13;
        }
```

Выделенные цветом две строки можно записать одинаково. Относительно записи в нижней строке мы говорили – оператор стрелка применяется тогда, когда речь идёт о ссылке на объект. Но и в этом случае можно применять оператор точки, если разыменовать ссылку, что и показано в верхней строке.

После сборки программы можно проверить её работу в отладчике Keil. Напомню ещё раз, что следует изменить два параметра, о которых шла речь ранее: DARMSTM.DLL -pSTM32F103C8.

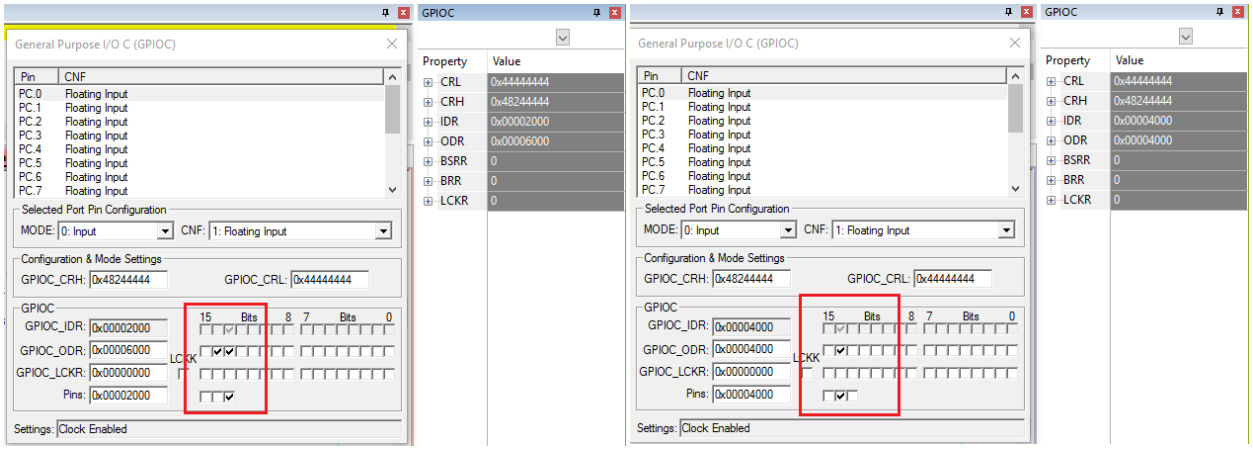


Рис. 3.4. Наблюдение за выводами порта

Отладчик показывает, что всё правильно – когда на входе PC14 единица (флажок установлен), на выходе PC13 низкий уровень (справа); когда на входе ноль (флажок сброшен), на выходе высокий уровень (слева). Так и было задумано. В чём же дело?

Всё просто. Достаточно было мультиметром посмотреть, что происходит на выходе, чтобы понять, что светодиод подключён не между выходом и землёй, как я предположил, а между выходом и плюсом. Что и вызвало у меня недоумение. Но это оказалось и полезно – мы познакомились с несколькими вариантами работы с выходом, что может быть полезно в дальнейшем.

Столь простая программа в действительности служит основой для многих устройств. К входу микроконтроллера подключаются датчики, многие из которых замыкают при их срабатывании контакты; аналогично подключаются кнопки управления и т.д. В частотности можно говорить об устройствах охраны, например, от пожара. В следующей главе мы рассмотрим этот вариант.

4 Устройство сигнализации

Мы объединим два эксперимента в новом для устройства сигнализации, чтобы показать, что не бывает простых программ, которые совсем бесполезны.

Многие системы сигнализации работают с датчиками, имеющими контакты реле (или транзистор, выполняющий эту функцию), которые замыкаются или размыкаются при срабатывании датчика. Так ряд пожарных датчиков имеет контакты, работающие на размыкание при пожаре. Сделано это по той причине, что датчики, как правило, располагаются вдали от пульта сигнализации. Проводных соединений бывает много, а, значит, возрастает вероятность того, что в каком-то месте соединение с датчиком будет нарушено. Чтобы проверить целостность соединения, датчик блокирован диодом, включённым встречно по отношению к штатному питающему напряжению, а при срабатывании датчика полярность питания меняют. Если ток в цепи протекает (через диод), то можно быть уверенным в том, что сработал датчик, а не произошёл обрыв проводов.

Но мы можем пока об этом не думать, поэтому используем предыдущий вариант – кнопка не нажата, всё хорошо. Кнопка нажата – тревога.

Итак.

Возьмём вторую программу, в которой изменим реакцию на нажатие кнопки программой из первого опыта.

```
while(1) {
if (HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_14) == 0 )
    {
      while (HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_14) == 0) {
            GPIOC->ODR ^= GPIO_PIN_13;
            HAL_Delay (500);
      }
    }
    else
        {
        GPIOC->BSRR = GPIO_BSRR_BR13;
        }
}
```

В сущности, я почти механически соединил две программы, добавив только цикл повторения «мигания» пока контакты замкнуты.

Далее необходимо проделать всё то, о чём говорилось в предыдущей главе для создания нового проекта. Открыв файл *main.c*, можно скопировать в него (после *while(1)*!) получившуюся программу. После настройки выхода и отладчика (как это сделано выше) можно либо загрузить программу в модуль STM32, либо запустить отладку. Хочу отметить, что количество фигурных скобок в тексте программы достаточно велико, в них можно запутаться. Разобраться со скобками поможет редактор Keil. Если выделить скобку, можно увидеть её пару:

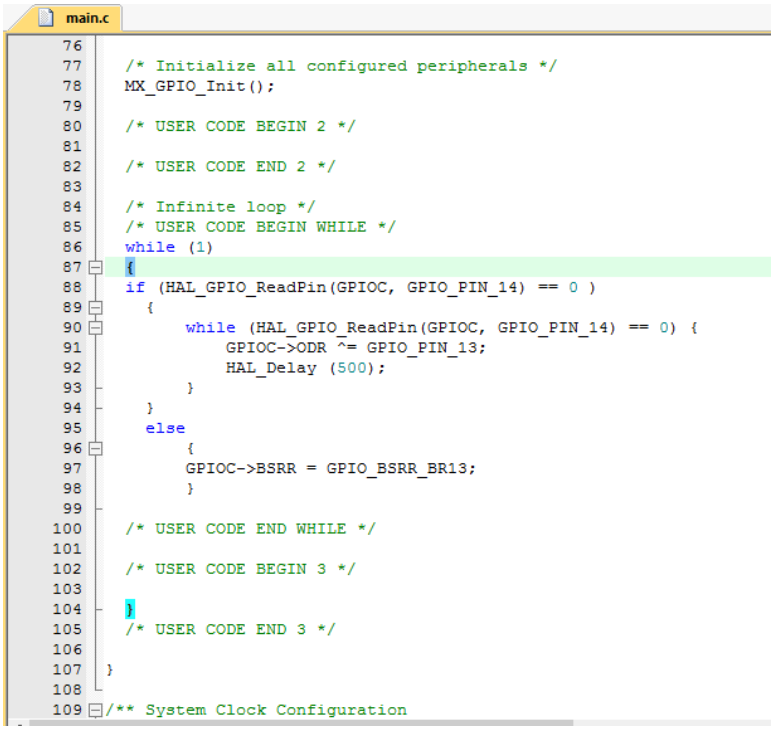


Рис. 4.1. Выделение парных скобок в редакторе

И в отладчике, и на плате светодиод исправно мигает, если нажать кнопку.

Конечно, для такого простого устройства можно выбрать и микроконтроллер попроще, это так, но!

Но, если вы надумаете подключить к устройству сигнализации радиоканал, если надумаете использовать Интернет, а такое может со временем случиться, STM32 «потянет» все эти задумки. А мне хотелось показать, что и простые программы могут оказаться весьма полезны. Главное не в сложности программы, а в том, чтобы знать, для чего программу (устройство, конечно) использовать.

Если у вас дачный кооператив, если на зиму кооператив нанимает сторожа для охраны дач, то добавьте к датчикам и светодиодам пульта ещё реле, включающее сирену, и охрана ваших дач станет гораздо надёжнее. Промышленные системы охраны, нет сомнений, гораздо сложнее. Но суть их работы не выходит за рамки приведённого выше примера.

5 Измерение постоянного напряжения с помощью АЦП

Многие модели микроконтроллеров имеют в своём составе аналого-цифровой преобразователь. STM32 не исключение, но АЦП в составе этой модели имеет 12 разрядов (и АЦП два, оба 16 канальные). То есть, использует 12 бит для получения значения постоянного напряжения в рамках, оговорённых в описании.

Поскольку речь зашла про описание, я позволю себе привести вводную часть про АЦП:

> *12-битовый АЦП последовательного приближения имеет до 18 переключаемых каналов, позволяющих измерять 16 внешних и два внутренних источника. Преобразование разных каналов может происходить в единичном, продолжающемся, сканирующем и бесконечном режимах. Результат оцифровки сохраняется в 16-битовом регистре с записью слева-направо или наоборот, по выбору. Аналоговое сторожевое устройство позволяет приложению обнаруживать, когда входное напряжение выходит за пределы (верхний или нижний порог), заданные пользователем. Тактовый генератор АЦП формируется сигналом от PCLK2 с помощью предделителя, и его частота не должна превышать 14 МГц.*

Настройку STM32CubeMx мы пока сведём к выбору первого канала ADC1, что соответствует выводу PA0. Создадим шаблон, откроем проект в Keil. Мы уже наслышаны о пользе обращения к файлам hal, поэтому просмотрим файл *stm32f1xx_hal_adc.c*, откуда выберем нужные нам функции. А в раздел переменных добавим такую:

```
/* Private variables-----*/
uint16_t dig = 0;
```

```
  while (1)
  {
      HAL_ADC_Start(&hadc1);
      dig = HAL_ADC_GetValue(&hadc1);
  /* USER CODE END WHILE */
  /* USER CODE BEGIN 3 */
  }
```

Настроим, как обычно, встроенный отладчик Keil и запустим отладку, выбрав АЦП в двух разделах:

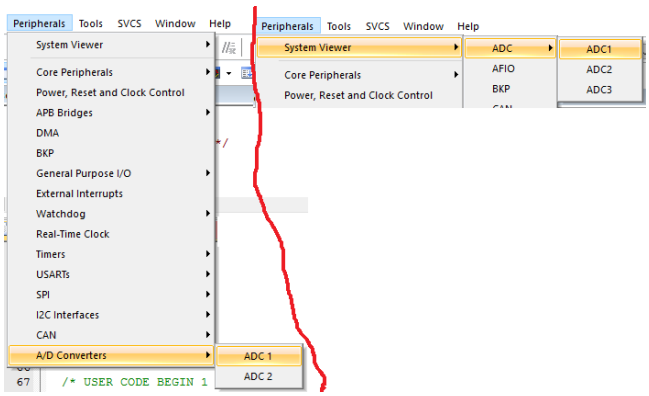


Рис. 5.1. Выбор окон настройки и наблюдения за АЦП

Диалоговое окно первого раздела позволяет управлять настройками АЦП, окно второго раздела показывает все регистры АЦП. В первом диалоге мы зададим напряжение 1В на входе АЦП, а во втором окне посмотрим полученное число после того, как запустим программу для отладки.

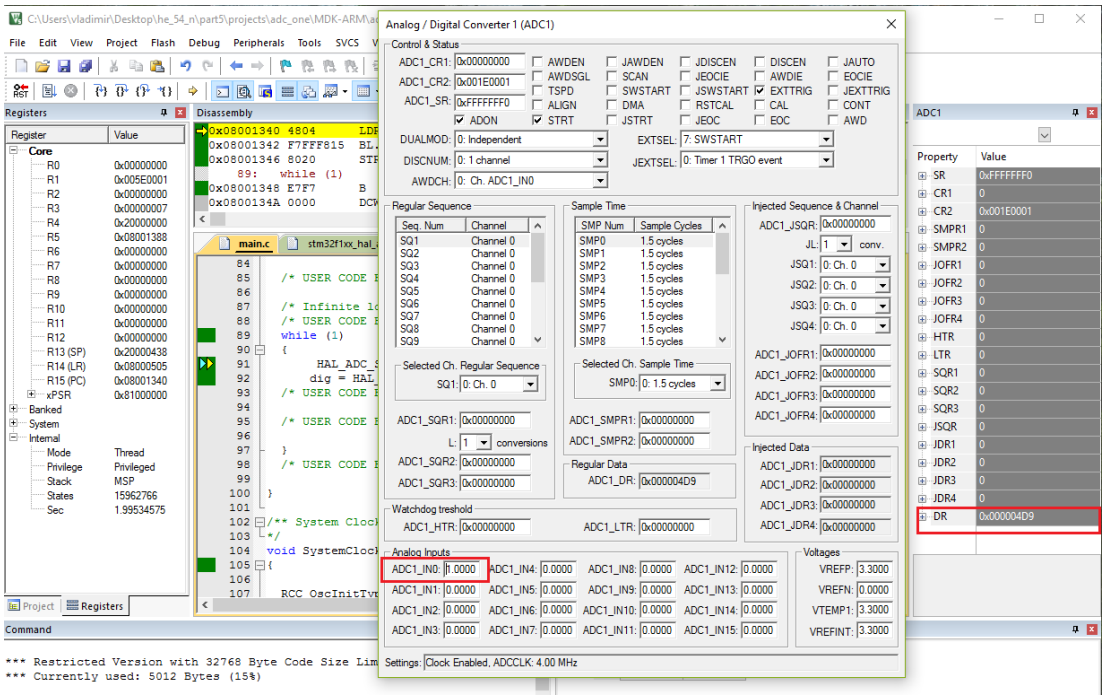


Рис. 5.2. Задание входного напряжения АЦП

Итак, при напряжении 1 В мы получаем шестнадцатеричное число 4D9, что соответствует десятичному 1241. Таким образом, разрешение нашего измерителя получается 0.8058 мВ. Осталось оттранслировать программу, загрузить в модуль и…

О результате измерения будет знать только микроконтроллер. И я очень сомневаюсь, что он расскажет нам о том, что у него получилось, сколько попыток что-то измерить мы ни сделай, без нашей помощи.

Результат измерения можно вывести на дисплей. Но дисплей нужно будет подключить к модулю. Нужно будет написать программу для работы с дисплеем. Поэтому я предпочитаю пока использовать встроенный USART для вывода информации, которую можно будет посмотреть каким-нибудь монитором COM-порта.

Есть, правда, один «подводный камень» – наша переменная 16-битовая, а через COM-порт отправляются байты. Чтобы решить эту проблему, попробуем добавить две переменные и сделать дополнительные операции:

```
/* Private variables ----------*/
uint16_t dig = 0;
uint8_t dig1 = 0;
uint8_t dig2 = 0;
  while (1)
  {
      HAL_ADC_Start(&hadc1);
      dig = HAL_ADC_GetValue(&hadc1);
      dig1 = dig>>8;
      dig2= dig&0xFF;
  /* USER CODE END WHILE */
  /* USER CODE BEGIN 3 */
  }
```

После запуска отладки в диалоге настройки АЦП вернём напряжение 1В, в основном меню используем раздел, позволяющий открыть окно наблюдения за переменными:

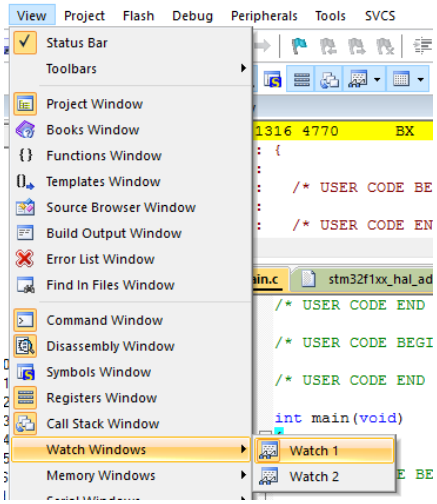


Рис. 5.3. Выбора окна наблюдения за переменными

Теперь, запустив программу, мы можем увидеть, как преобразилось наше напряжение в 1В:

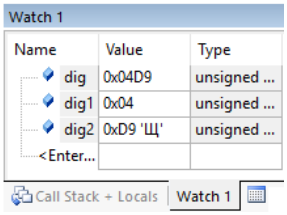
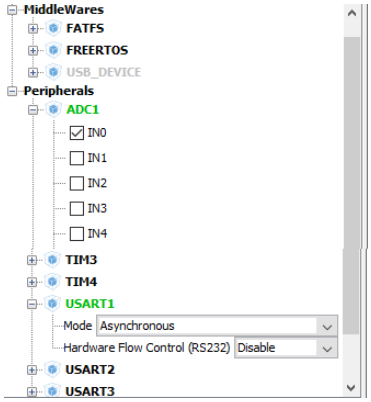


Рис. 5.4. Окно наблюдения за переменными

Шестнадцати битовое число мы разбили на два байта, которые можем отправить через COM-порт, а при необходимости вновь собрать в одно число. Осталось только…

6 Передача данных через USART

Я не буду настаивать, но пока предпочитаю «танцевать от печки», то есть, от программы STM32CubeMx, где мы выберем ADC1 и USART1:



Мы выбрали всё, что необходимо для работы. Пока на этом остановимся с настройками в программе STM32CubeMx. Как и раньше запускаем генерацию заготовки для программы и открываем Keil.

Рис. 6.1. Выбор периферии для проекта

Из предыдущей программы я переношу в текст новой программы всё, что наработано ранее. Затем заглядываю в файл *stm321xx_hal_uart.c*.

В этом файле есть то, как задать скорость работы порта: `huart1.Init.BaudRate = 115200;`

Здесь huart1 – это определение USART1 в структуре. Задание скорости работы порта я добавляю в раздел *main(void).* Есть и удобная функция отправки данных через порт. Правда, для этого я немного переделываю переменные. Вместо двух переменных для двух частей числа, измеренного АЦП, я использую массив с двумя ячейками, куда и запишу данные для отправки. Программа выглядит так:

```
/* Private variables------*/
uint16_t dig = 0;
uint8_t dig_part[2] = {0,0};
int main(void)
{
  huart1.Init.BaudRate = 115200;
  /* Reset of all peripherals,… */
  HAL_Init();
  /* Configure the system clock */
  SystemClock_Config();
  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_ADC1_Init();
  MX_USART1_UART_Init();
  ADC1->CR2 = 5;
HAL_Delay(1000);
  while (1)
  {
      HAL_ADC_Start(&hadc1);
      dig = HAL_ADC_GetValue(&hadc1);
      dig_part[0] = dig>>8;
      dig_part[1] = dig&0xFF;
```

```
        HAL_UART_Transmit(&huart1, dig_part, 2, 100);
        HAL_Delay(5000);
    }
}
```

Поскольку мы вписываем программу в шаблон, созданный программой STM32CubeMx, вписывать её следует в те разделы, которые размечены для этой цели. Добавление `ADC1->CR2 = 5;` призвано осуществить калибровку (если я правильно это понял из описания).

После сборки проекта и загрузки его в модуль можно проверить результат, но для этого потребуется монитор порта. У меня, я почти уверен, есть что-то на компьютере, но отыскать это… Есть другой выход. У меня установлена программа Atmel Studio, частью которой служит Data Visualizer (программа или утилита, может быть, не помню, плагин). Запустив и настроив эту программу, я получаю результат измерения напряжения батарейки:

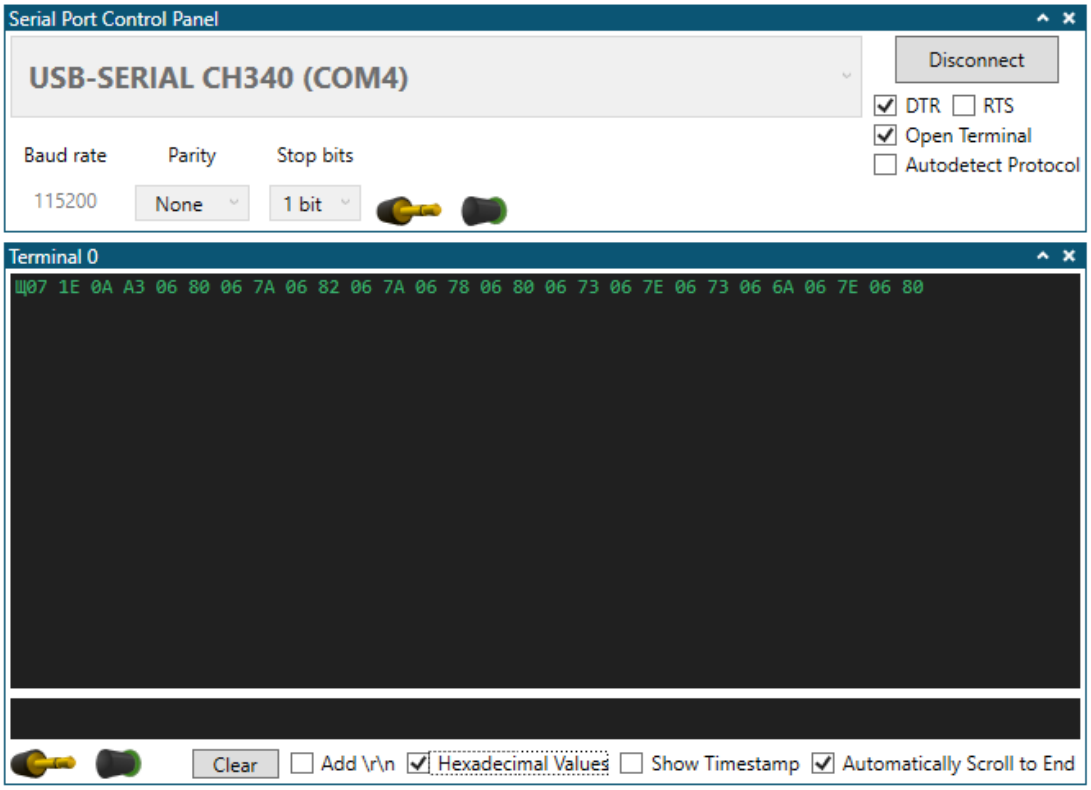


Рис. 6.1. Проверка значения оцифровки АЦП напряжения батарейки

Дальше приходится воспользоваться калькулятором. Десятичное число, которое соответствует шестнадцатеричному (0x680) на экране монитора, это 1664, и его нужно умножить на 0.8058 мВ. В итоге напряжение батарейки равно 1.341В.

Напряжение, измеренное точным мультиметром, оказывается равным 1.3554В. То есть, точность измерения порядка 1%. Это не так и плохо.

7 Измерение переменного напряжения с помощью АЦП

Как правило, и это правильно, переменное напряжение выпрямляют, чтобы измерить полученное постоянное напряжение. Но в этом опыте мы попробуем обойтись без выпрямления.

После настроек заготовки, сделанных точно так, как это было сделано в предыдущем опыте, и генерации шаблона откроем основной файл в программе Keil, куда перепишем почти всё, что было в файле *main.c* ранее.

Идея в данном случае такова – мы будем оцифровывать переменное синусоидальное напряжение частоты 1 кГц много раз, сравнивая два соседних измерения и выбирая наибольшее. Есть надежда, что мы встретим значение равное удвоенной амплитуде сигнала. Останется (правда, ручками на калькуляторе) определить величину переменного напряжения.

Программа должна выглядеть так:

```
 /* Includes ----------------*/
#include "main.h"
#include "stm32f1xx_hal.h"

/* USER CODE BEGIN Includes */
/* USER CODE END Includes */

/* Private variables -------*/
ADC_HandleTypeDef hadc1;
UART_HandleTypeDef huart1;

/* USER CODE BEGIN PV */
/* Private variables ----*/
uint16_t dig1 = 0; // временная переменная для хранения результата
uint16_t dig = 0; // окончательная переменная для хранения результата
uint8_t dig_part[2] = {0,0}; // массив для отправки данных
uint16_t ac1 = 0; // переменная для оцифровки
uint16_t ac2 = 0; // вторая переменная для оцифровки

/* USER CODE END PV */

/* Private function prototypes-------*/
void SystemClock_Config(void);
void Error_Handler(void);
static void MX_GPIO_Init(void);
static void MX_ADC1_Init(void);
static void MX_USART1_UART_Init(void);

/* USER CODE BEGIN PFP */
/* Private function prototypes -------*/
/* USER CODE END PFP */
/* USER CODE BEGIN 0 */
/* USER CODE END 0 */

int main(void)
{
```

```
    huart1.Init.BaudRate = 115200;

  /* USER CODE BEGIN 1 */
  /* USER CODE END 1 */
  /* MCU Configuration-----*/
  /* Reset of all peripherals…  */
  HAL_Init();

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

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_ADC1_Init();
  MX_USART1_UART_Init();
  ADC1->CR2 = 5;
  HAL_Delay(1000);

  /* USER CODE BEGIN 2 */
  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
      for(int i = 0; i<1000;i++) { // определяем максимум при оцифровке
      HAL_ADC_Start(&hadc1);
      ac1 = HAL_ADC_GetValue(&hadc1);
      HAL_ADC_Start(&hadc1);
      ac2 = HAL_ADC_GetValue(&hadc1);
      if(ac1>ac2) { dig1 = ac1;}
        else{dig1 = ac2;}
      if(dig1>dig) {dig = dig1;}
     }
      dig_part[0] = dig>>8; // начинаем подготовку данных для отправки
      dig_part[1] = dig&0xFF;
      HAL_UART_Transmit(&huart1, dig_part, 2, 100);
      HAL_Delay(1000);

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

В этот раз я сохранил значительную часть шаблона файла *main.c*, чтобы было понятнее, например, что почти все переменные в программе глобальные, кроме переменной i, объявленной локально. Локальные переменные уменьшают расход памяти, но к глобальным переменным всегда есть доступ в любом месте программы. Программа немного «корявая», но, надеюсь, понятная.

Ниже результат этого опыта:

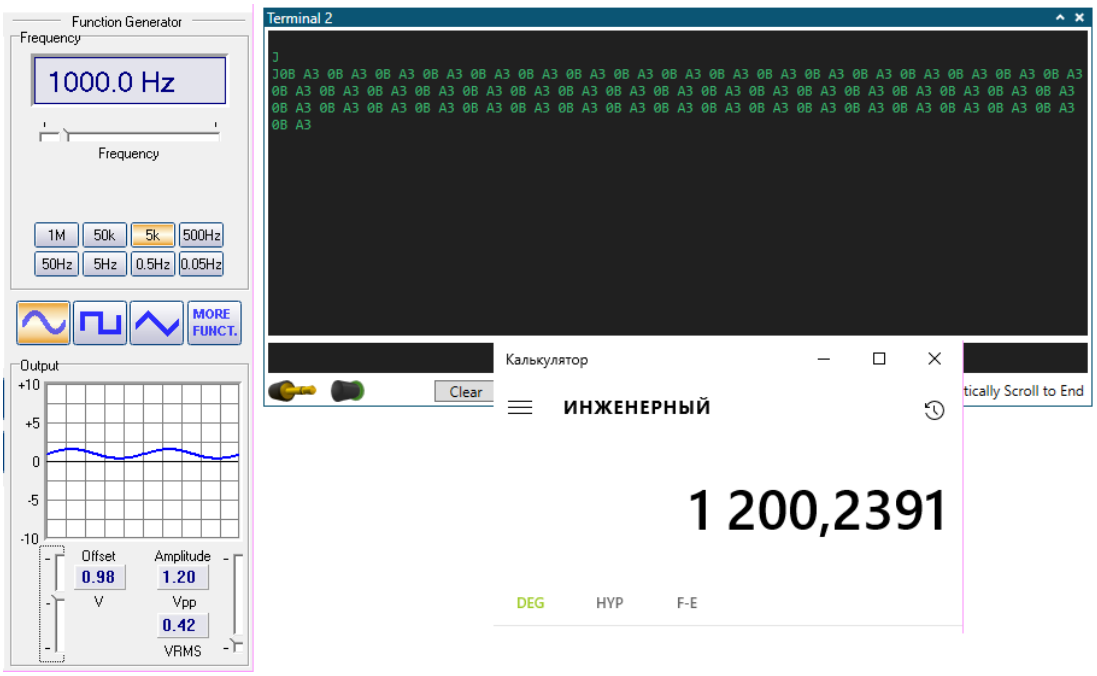


Рис. 7.1. Сравнение измеренной и заданной амплитуды синусоидального напряжения

На калькуляторе расчёт амплитуды сигнала. Конечно, можно выполнить эти расчёты микроконтроллером. Это так. Но это я предлагаю сделать вам. Добавьте, скажем, две кнопки; когда нажата первая кнопка, вы производите расчёт для синусоидального сигнала и выводите амплитудное и эффективное значение; когда нажата вторая кнопка, вы выводите значения для прямоугольного меандра. Не думаю, что есть смысл создавать подобное устройство, но в качестве эксперимента… отчего бы и нет?

8 Микроконтроллер и дисплей

Выводить данные при экспериментах на монитор COM-порта я считаю очень удобным. Но в тех случаях, когда вы разрабатываете какое-либо устройство, отображающее информацию, которое позже соберёте и будете использовать, вам понадобится дисплей.

Дисплеев существует множество – и знаковые, и графические, и цветные, и монохромные. В моём распоряжении недорогой дисплей, возможно, с контроллером 1602A, возможно, я уже не помню «фамилию» дисплея, с другим, но дисплей двухстрочный из серии 16x2.

К моему сожалению, я довольно долго пытался найти, но нет такого простого решения, которое позволило бы использовать всё готовое из библиотеки Keil. То есть, я не отрицаю, что нельзя исключить и того, что я плохо поискал. Но я не нашёл.

По своей привычке полагаться на то, что сделали другие добрые люди, я нашёл в Интернете несколько вариантов. Более того, я нашёл и скачал примеры программ для STM32F103: https://github.com/yohanes-erwin/stm32f103-keil

На указанной странице справа есть зелёная кнопочка, на которой написано «Clone or download», которая помогла мне обзавестись множеством примеров. Один из примеров программ

предназначался для работы с нужным мне дисплеем. Если не считать, что я «промахнулся», не подключив (обычно его держат на земле) вывод RW к модулю STM32F103C8, то в остальном пример нарисовал на дисплее батарейку и написал, что она разряжена.

Программа готова к тому, чтобы её открыть в Keil, оттранслировать и загрузить в модуль. Более того, среди примеров я обнаружил и программу для работы с АЦП и дисплеем. Но в этом месте удача покинула меня – я так и не смог заставить программу работать. То есть, она работала, но выводить на дисплей что-либо не захотела. Не захотела и всё тут. Пришлось обратиться к другим примерам работы с дисплеем, которые я загрузил с сайтов. Но и здесь мне не повезло. Ни один из примеров не дал желаемого результата. Вообще, конечно, возиться с чужими программами, если они не устраивают вас полностью, занятие, что называется «на любителя». Попробовав раз-другой, я махнул рукой на этот подход и взял за основу один из текстов, скачал datasheet для gdm1602 и стал…

Чтобы избавить себя от забот в части АЦП, я использую программу STM32CubeMx. В программе я задаю выводы на выход для отправки данные на дисплей (4 бита, что чаще всего делается) PB12-PB15 и выводы для управления (RS и E) PC13 и PC14. Раньше я этого не делал, но сейчас решил, что следует задать источник тактирования. В итоге это выглядит следующим образом:

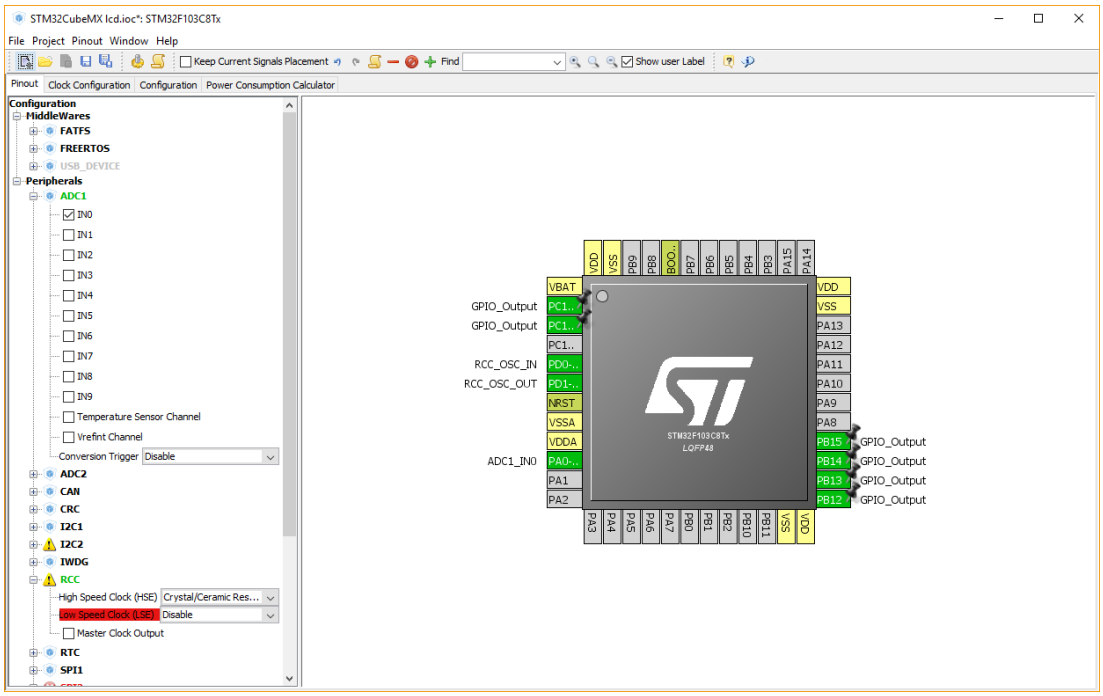


Рис. 8.1. Выбор выводов и настройка для подключения дисплея

После генерации заготовки программы, открыв Keil (удобно, что предложением сделать это завершается генерация шаблона), я стал писать свой вариант работы с АЦП и дисплеем, основываясь на описании дисплея и готовых файлов, которые у меня не захотели работать, когда я использовал их в программе.

К слову, пример, который я хотел использовать первоначально из набора примеров для STM32, послужил мне укором – не зря там используется переключение из режима записи в режим чтения, но, увы, пример не помог в моей работе. Быть может, вам он будет интереснее и полезнее того, что я изложу дальше: проект *adc-polling*.

Подключение дисплея к модулю STM32 я предпочёл простое, соединив вывод RW с землёй:

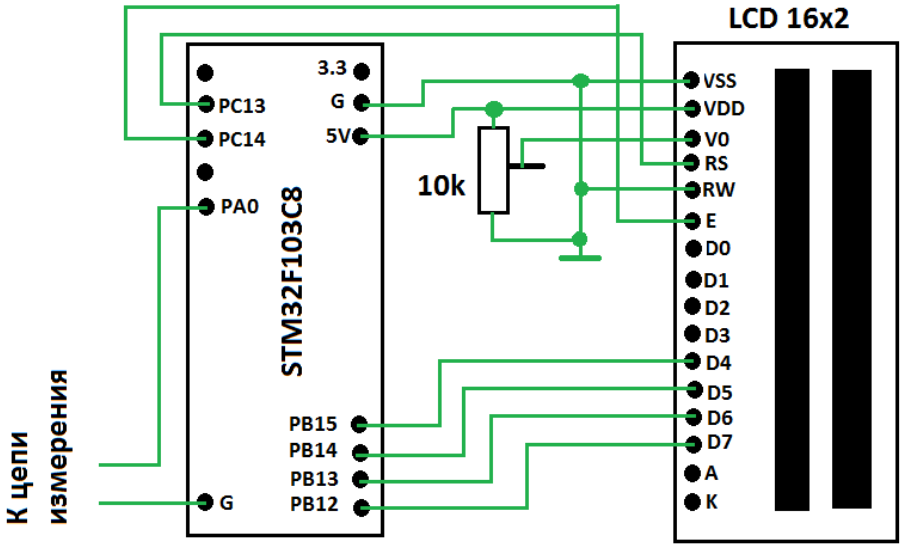


Рис. 8.2. Схема соединения модуля и дисплея

Для подключения дисплея можно было бы использовать один порт, но я не был уверен, что манипуляция с выводами управления не «зацепит» что-то в данных. Оригинал, с которого я списывал, обращается ко всем 8 битам данных дисплея, по причине чего эту часть пришлось переделывать. И, вообще, списывание всегда чревато ошибками (и не только своими). В частности, используя 16 битовые переменные (как в оригинале) я столкнулся с ошибками при преобразовании, что послужило к замене всех переменных с типа беззнаковых целых на беззнаковые байтовые.

Принцип работы с дисплеем не самый сложный. Есть команды настройки и управления, их список можно найти в таблице. Эти команды выполняются при сигнале RS равном нулю. Сами команды записываются, как и данные, через выводы данных, когда меняется сигнал E. Если верить диаграммам справки, это производится задним фронтом сигнала E, когда он из нуля переходит в единицу и обратно:

Write mode timing diagram

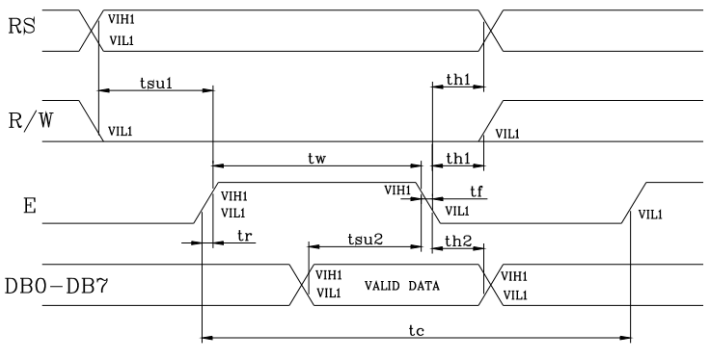


Рис. 8.3. Диаграмма сигналов при записи

Команды для дисплея вначале используются для его настройки. Выглядит это так:

```
void lcd_init(void) {
 GPIOC->BSRR = GPIO_BSRR_BR13; // RS = 0

   lcd_write_data(0x28); // задание 4 бита 2 линии 5x8 точек 00101000
   HAL_Delay(1);

   lcd_write_data(0xC); // включение дисплея 00001100
   HAL_Delay(1);

   lcd_write_data(0x6); // настройка режима ввода 00000110
   HAL_Delay(1);

   lcd_write_data(0x1); // очистка 00000001
   HAL_Delay(1);

   lcd_write_data(0x2); // возврат домой 00000010
   HAL_Delay(1);

GPIOC->BSRR = GPIO_BSRR_BS13;  // RS = 1
}
```

Паузы между командами определяются там же в справке, но я использовал задержки в 1 мс, чтобы не создавать программу для организации микросекундных пауз.

Правильность (или нет) инициализации вы можете проверить по следующей таблице:

Instruction Table

Instruction	Instruction code										Description	Execution time (fosc= 270 KHZ
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0		
Clear Display	0	0	0	0	0	0	0	0	0	1	Write "20H" to DDRA and set DDRAM address to "00H" from AC	1.53ms
Return Home	0	0	0	0	0	0	0	0	1	-	Set DDRAM address to "00H" From AC and return cursor to Its original position if shifted. The contents of DDRAM are not changed.	1.53ms
Entry mode Set	0	0	0	0	0	0	0	1	I/D	SH	Assign cursor moving direction And blinking of entire display	39us
Display ON/ OFF control	0	0	0	0	0	0	1	D	C	B	Set display (D), cursor (C), and Blinking of cursor (B) on/off Control bit.	
Cursor or Display shift	0	0	0	0	0	1	S/C	R/L	-	-	Set cursor moving and display Shift control bit, and the Direction, without changing of DDRAM data.	39us
Function set	0	0	0	0	1	DL	N	F	-	-	Set interface data length (DL: 8-Bit/4-bit), numbers of display Line (N: =2-line/1-line) and, Display font type (F: 5x11/5x8)	39us
Set CGRAM Address	0	0	0	1	AC5	AC4	AC3	AC2	AC1	AC0	Set CGRAM address in address Counter.	39us
Set DDRAM Address	0	0	1	AC6	AC5	AC4	AC3	AC2	AC1	AC0	Set DDRAM address in address Counter.	39us
Read busy Flag and Address	0	1	BF	AC6	AC5	AC4	AC3	AC2	AC1	AC0	Whether during internal Operation or not can be known By reading BF. The contents of Address counter can also be read.	0us
Write data to Address	1	0	D7	D6	D5	D4	D3	D2	D1	D0	Write data into internal RAM (DDRAM/CGRAM).	43us
Read data From RAM	1	1	D7	D6	D5	D4	D3	D2	D1	D0	Read data from internal RAM (DDRAM/CGRAM).	43us

Рис. 8.4. Таблица команд для дисплея 1602A

Функция записи данных выглядит следующим образом:

```
void lcd_write_data(uint8_t data) {
    uint8_t h_data; // старшая четвёрка данных
    uint8_t l_data; // младшая четвёрка данных

        h_data = (data >> 4);
        l_data = data & 0x0F;
        GPIOB->ODR = h_data<<12; // перемещаем половину данных
        (*GPIOC).BSRR = GPIO_BSRR_BS14; // записываем их
        HAL_Delay(1);
        GPIOC->BSRR = GPIO_BSRR_BR14;
        GPIOB->ODR = l_data<<12; // перемещаем вторую половину данных
        (*GPIOC).BSRR = GPIO_BSRR_BS14; // записываем её
        HAL_Delay(1);
        GPIOC->BSRR = GPIO_BSRR_BR14;
}
```

В режиме работы с 4 битами данные записываются дважды, вначале старшая часть, затем младшая. В данном случае порт B достаточно свободен, можно было бы использовать 8-битовый режим, но часто используют тот вариант включения, что показан на схеме, поэтому я остановился на таком варианте включения дисплея.

Отдельная функция команды нужна для перевода курсора, если это нужно, и для «возвращения домой». Всю программу я приведу ниже, а сейчас хочу сказать ещё об одном. Оказалось удобным использовать функцию `sprintf` из библиотеки *stdio*, что потребовало включения заголовочного файла. Эта функция отличается от `printf` тем, что записывает переменную в нужном формате в заданный буфер.

Первоначально я не использовал эту функцию, поэтому собирался выводить значение в милливольтах. Для этого я использовал массив для пяти символов. Позже я не менял массив, используя его в качестве буфера, работающего с функцией `sprintf`. Признаться, я был удивлён результату работы программы:

Рис. 8.5. Проверка программы на макетной плате

При эксперименте измерялось напряжение б/у «пальчиковой» батарейки. Это напряжение, измеренное через некоторое время мультиметром повышенной точности, составило 1.3570В. Ошибка около 1%. А моё внимание привлекло то, что выводилось 6 символов.

Конечно, следовало бы разобраться со всеми странностями этого эксперимента, но их было достаточно много. В частности я столкнулся с тем, что приходилось при загрузке программы в модуль подключать питание 3.3В, а для питания дисплея перебрасывать провода к выводам 5В. Когда я начинал проверять программирование STM32 с помощью перехода USB/TTL на микросхеме MAX232, у меня не получилось программировать при питании 5В. Но в этот раз, ошибаясь при многочисленных переключениях, я подключил MAX232 к 5-вольтовому питанию и был удивлён тем, что программирование состоялось.

Обычно принято разбивать программу на заголовочный файл и основной, но я не стал этого делать. И, как и обещал, я привожу всю программу:

```
/* Includes --------------*/
#include "main.h"
#include "stm32f1xx_hal.h"
#include <stdio.h>
/* USER CODE BEGIN Includes */

/* USER CODE END Includes */
//#define lcd_rs (*GPIOC).GPIO_PIN_13 // чтобы не забывать о подключении
//#define lcd_e  (*GPIOC).GPIO_PIN_14
//#define lcd_d4 (*GPIOB).GPIO_PIN_12
//#define lcd_d5 (*GPIOB).GPIO_PIN_13
//#define lcd_d6 (*GPIOB).GPIO_PIN_14
//#define lcd_d7 (*GPIOB).GPIO_PIN_15

/* Private variables ---------------*/
ADC_HandleTypeDef hadc1;

/* USER CODE BEGIN PV */
/* Private variables ---------------*/
uint16_t volt = 0;
char sVolt[5];
float o_volt;

/* USER CODE END PV */

/* Private function prototypes ----*/
void SystemClock_Config(void);
void Error_Handler(void);
static void MX_GPIO_Init(void);
static void MX_ADC1_Init(void);

/* USER CODE BEGIN PFP */
/* Private function prototypes ----*/
void lcd_init(void);
void lcd_write_str(char*str);
void lcd_set_cursor(uint8_t line,uint8_t pos);
void lcd_write_cmd(uint8_t cmd);
```

```
void lcd_write_data(uint8_t data);

/* USER CODE END PFP */

/* USER CODE BEGIN 0 */

/* USER CODE END 0 */

int main(void)
{

  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

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

  /* Reset of all peripherals… */
  HAL_Init();

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

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_ADC1_Init();
  ADC1->CR2 = 5;
  lcd_init();
  HAL_Delay(1000);
  /* USER CODE BEGIN 2 */

  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
      HAL_ADC_Start(&hadc1);
      volt = HAL_ADC_GetValue(&hadc1);
      o_volt = volt*0.8058/1000;
      sprintf(sVolt, "Vdc: %1.4f", o_volt);
      lcd_write_str(sVolt);

      HAL_Delay(5000);
      lcd_write_cmd(0x2); // возвращение домой

  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */

  }
  /* USER CODE END 3 */

}
```

```
/** System Clock Configuration
*/
void SystemClock_Config(void)
{

  RCC_OscInitTypeDef RCC_OscInitStruct;
  RCC_ClkInitTypeDef RCC_ClkInitStruct;
  RCC_PeriphCLKInitTypeDef PeriphClkInit;

    /**Initializes the CPU, AHB and APB busses clocks */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = 16;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_NONE;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

    /**Initializes the CPU, AHB and APB busses clocks */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_HSI;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_0) != HAL_OK)
  {
    Error_Handler();
  }

  PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_ADC;
  PeriphClkInit.AdcClockSelection = RCC_ADCPCLK2_DIV2;
  if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit) != HAL_OK)
  {
    Error_Handler();
  }

    /**Configure the Systick interrupt time  */
  HAL_SYSTICK_Config(HAL_RCC_GetHCLKFreq()/1000);

    /**Configure the Systick */
  HAL_SYSTICK_CLKSourceConfig(SYSTICK_CLKSOURCE_HCLK);

  /* SysTick_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(SysTick_IRQn, 0, 0);
}

/* ADC1 init function */
static void MX_ADC1_Init(void)
{
```

```
  ADC_ChannelConfTypeDef sConfig;

    /**Common config */
  hadc1.Instance = ADC1;
  hadc1.Init.ScanConvMode = ADC_SCAN_DISABLE;
  hadc1.Init.ContinuousConvMode = DISABLE;
  hadc1.Init.DiscontinuousConvMode = DISABLE;
  hadc1.Init.ExternalTrigConv = ADC_SOFTWARE_START;
  hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
  hadc1.Init.NbrOfConversion = 1;
  if (HAL_ADC_Init(&hadc1) != HAL_OK)
  {
    Error_Handler();
  }

    /**Configure Regular Channel */
  sConfig.Channel = ADC_CHANNEL_0;
  sConfig.Rank = 1;
  sConfig.SamplingTime = ADC_SAMPLETIME_1CYCLE_5;
  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }

}

/** Configure pins as
        * Analog
        * Input
        * Output
        * EVENT_OUT
        * EXTI
*/
static void MX_GPIO_Init(void)
{

  GPIO_InitTypeDef GPIO_InitStruct;

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOD_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13|GPIO_PIN_14, GPIO_PIN_RESET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_12|GPIO_PIN_13|GPIO_PIN_14|GPIO_PIN_15
                          |GPIO_PIN_3, GPIO_PIN_RESET);

  /*Configure GPIO pins : PC13 PC14 */
  GPIO_InitStruct.Pin = GPIO_PIN_13|GPIO_PIN_14;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
```

```
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

  /*Configure GPIO pins : PB12 PB13 PB14 PB15 */
  GPIO_InitStruct.Pin = GPIO_PIN_12|GPIO_PIN_13|GPIO_PIN_14|GPIO_PIN_15;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
}

/* USER CODE BEGIN 4 */

void lcd_write_data(uint8_t data) {
    uint8_t h_data;
    uint8_t l_data;

        h_data = (data >> 4);
        l_data = data & 0x0F;
        GPIOB->ODR = h_data<<12;
        (*GPIOC).BSRR = GPIO_BSRR_BS14;
        HAL_Delay(1);
        GPIOC->BSRR = GPIO_BSRR_BR14;
        GPIOB->ODR = l_data<<12;
        (*GPIOC).BSRR = GPIO_BSRR_BS14;
        HAL_Delay(1);
        GPIOC->BSRR = GPIO_BSRR_BR14;
}

void lcd_init(void) {
   GPIOC->BSRR = GPIO_BSRR_BR13; //rs = 0

      lcd_write_data(0x28); // function set  4bit 2line 5x8 dots
    HAL_Delay(1);

      lcd_write_data(0xC); // display on
    HAL_Delay(1);

      lcd_write_data(0x6); //entry mode set
    HAL_Delay(1);

      lcd_write_data(0x1); //clear 0x1
    HAL_Delay(1);

      lcd_write_data(0x2); // return to home
    HAL_Delay(1);

       GPIOC->BSRR = GPIO_BSRR_BS13;  //rs = 1
}

void lcd_write_str(char*str) {
      do {
            lcd_write_data(*str);
      }while(*++str);
}
```

```
void lcd_write_cmd(uint8_t cmd) {
    GPIOC->BSRR = GPIO_BSRR_BR13;
      lcd_write_data(cmd);
      GPIOC->BSRR = GPIO_BSRR_BS13;
}

void lcd_set_cursor(uint8_t line,uint8_t pos) {
      pos |= 0x80;
      if (line == 1) {
            pos += 0x40;
      }
      lcd_write_cmd(pos);
}


/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @param  None
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler */
  /* User can add his own implementation to report the HAL error return state
*/
  while(1)
  {
  }
  /* USER CODE END Error_Handler */
}

#ifdef USE_FULL_ASSERT

/**
   * @brief Reports the name of the source file and the source line number
   * where the assert_param error has occurred.
   * @param file: pointer to the source file name
   * @param line: assert_param error line source number
   * @retval None
   */
void assert_failed(uint8_t* file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line
number,
    ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line)
*/
  /* USER CODE END 6 */

}
```

```
#endif

/**
  * @}
  */

/**
  * @}
*/
/****************** (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```

Большая часть этой записи произведена программой STM32CubeMx, которую я оставил для лучшего понимания происходящего. Свою запись я выделил цветом. Очевидно, что использование заготовки избавляет от написания значительной части кода программы. Что я и хотел показать.

Если вам интересно провести аналогичный опыт, можете добавить кнопки, чтобы переключать измерение напряжения между измерением постоянного напряжения и переменного, как это было сделано ранее. Для переменного напряжения можно выводить действующее значение синусоидального напряжения и напряжения прямоугольного меандра, для которых существуют простые формулы пересчёта от амплитудного значения к действующему.

9 Использование USB

Модуль STM32F103C8 поддерживает разные протоколы коммуникации: это и CAN, и SPI, и I2C и другие. При наличии подходящих модулей или компонентов можно провести интересные эксперименты, используя эти возможности микроконтроллера. Но для первого знакомства, с моей точки зрения, желательно обойтись без дополнительных затрат. Поэтому я предлагаю познакомиться с работой модуля через порт USB. Для этого достаточно модуля, возможности его программировать и компьютера.

Начнём этот эксперимент с программы STM32CubeMx, где произведём соответствующие настройки.

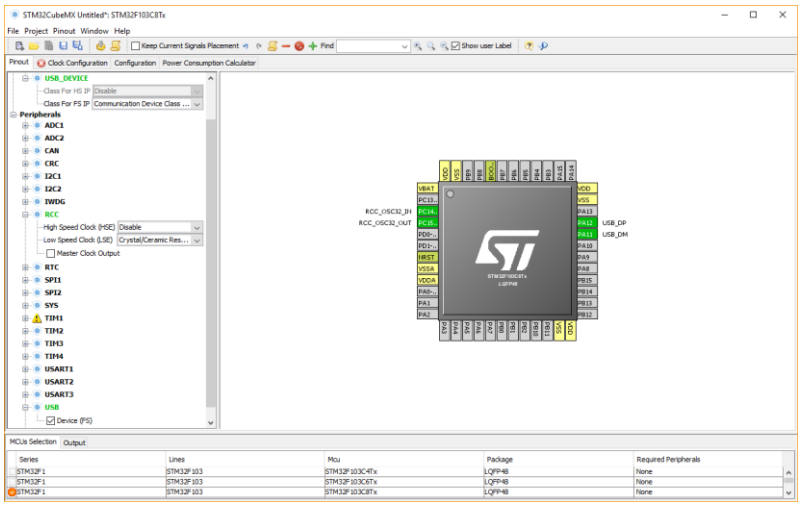


Рис. 9.1. Настройки выводов и выбор периферии

Если раньше было достаточно выбрать в окне периферии нужные устройства, то в данном случае предстоит продолжить настройки. В первую очередь следует настроить тактовые частоты для всей периферии. Пока нет достаточного опыта работы с микроконтроллером, лучше согласиться на автоматическую настройку.

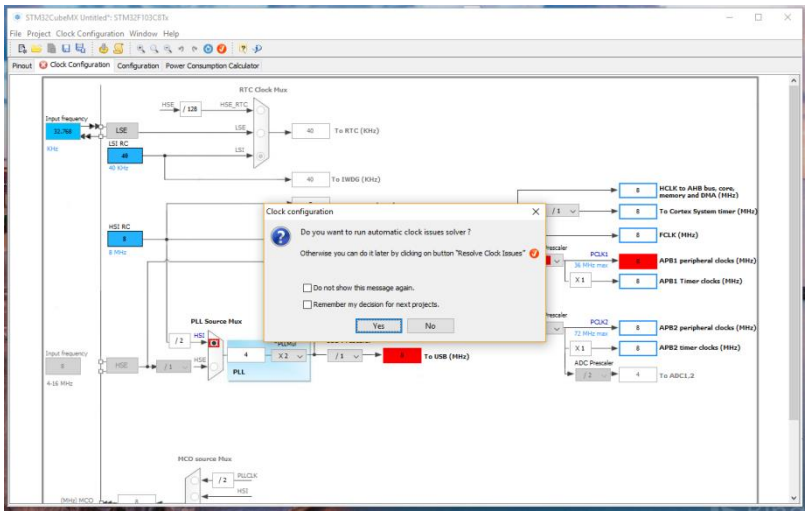


Рис. 9.2. Предложение автоматической настройки тактирования

Теперь можно генерировать заготовку для программы и открыть среду разработки Keil.

Я собираюсь отправить простую строку (массив), который выглядит как символы 1234. Вот этот массив в разделе переменных:

```
/* Private variables ---------------------*/
uint8_t snd[] = {0x31, 0x32, 0x33, 0x34};
```

Если открыть файл (он создан или добавлен программой STM32CubeMx) *usbd_cdc_if.c*, то можно найти функцию для отправки данных через порт USB:

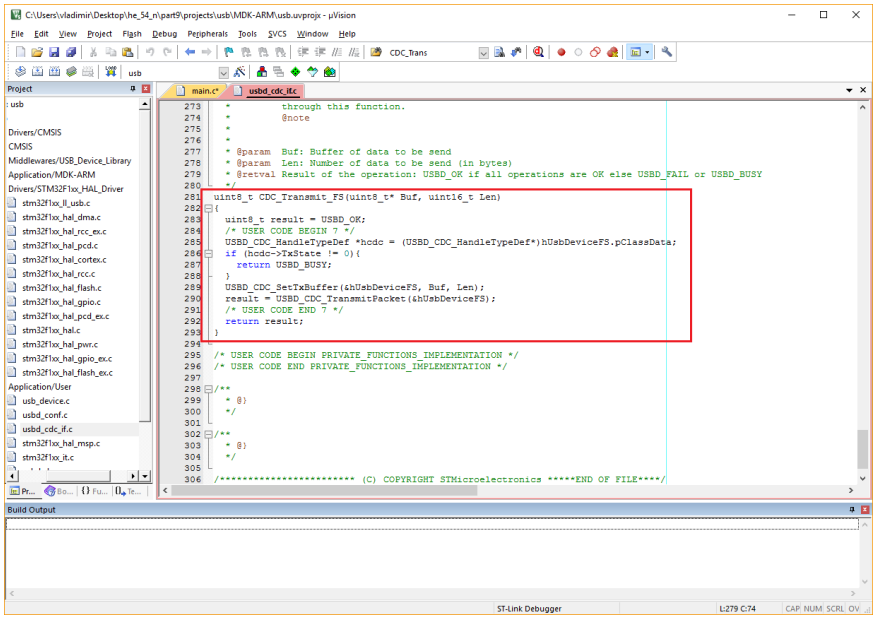


Рис. 9.3. Просмотр файла работы с USB

Добавим эту функцию в программу, указав наш массив и длину строки в 4 символа:

```
/* USER CODE BEGIN WHILE */
while (1)
{
    CDC_Transmit_FS(snd,4);
    HAL_Delay(1000);
/* USER CODE END WHILE */
}
```

Пауза в 1000 мс нужна, чтобы данные выводились не так быстро – USB порт работает со скоростью 12 Мбит/сек.

После добавления функции отправки появится значок сообщения о том, что есть некоторые проблемы. Если навести курсор мышки на этот значок, то можно увидеть сообщение:

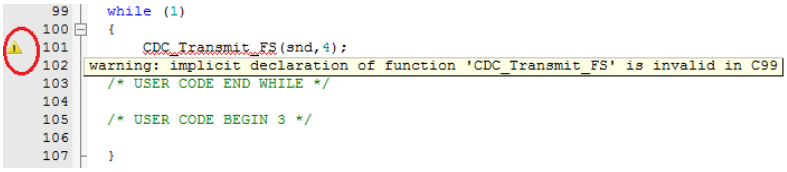


Рис. 9.4. Ошибка и подсказка в редакторе программы

Решение простое: следует добавить в раздел включений ещё один файл:

```
/* Includes ------------*/
#include "main.h"
#include "stm32f1xx_hal.h"
#include "usb_device.h"
#include "usbd_cdc_if.h"
```

Отметив на закладке *Output* необходимость создания hex-файла:

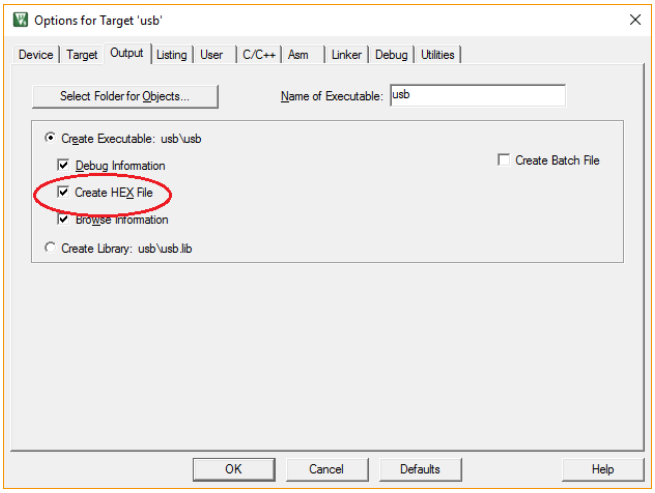


Рис. 9.5. Отметка для генерации hex-файла

…можно запустить сборку проекта, загрузить программу в модуль и убедиться в диспетчере устройств, что появилось новое устройство. Если включить модуль в порт USB до загрузки программы, то вы увидите:

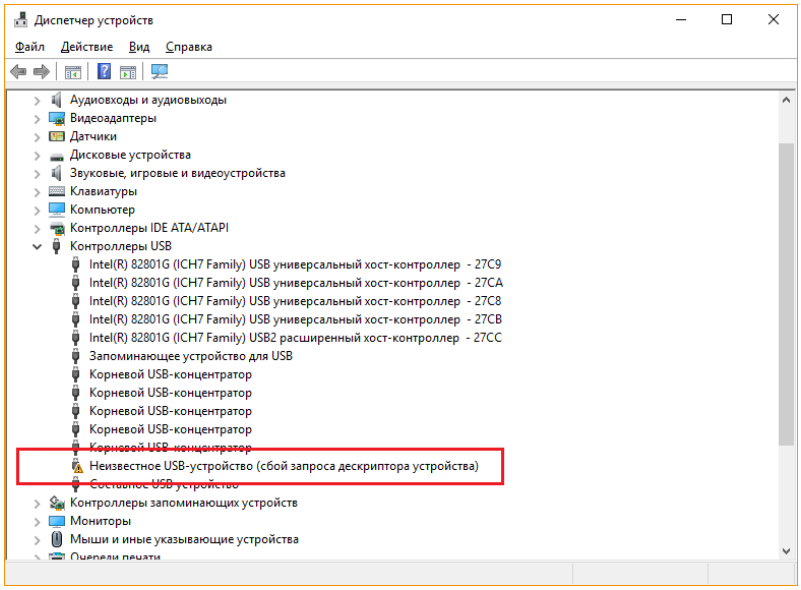


Рис. 9.6. Подключение модуля без настройки USB

Но после загрузки программы ситуация изменится:

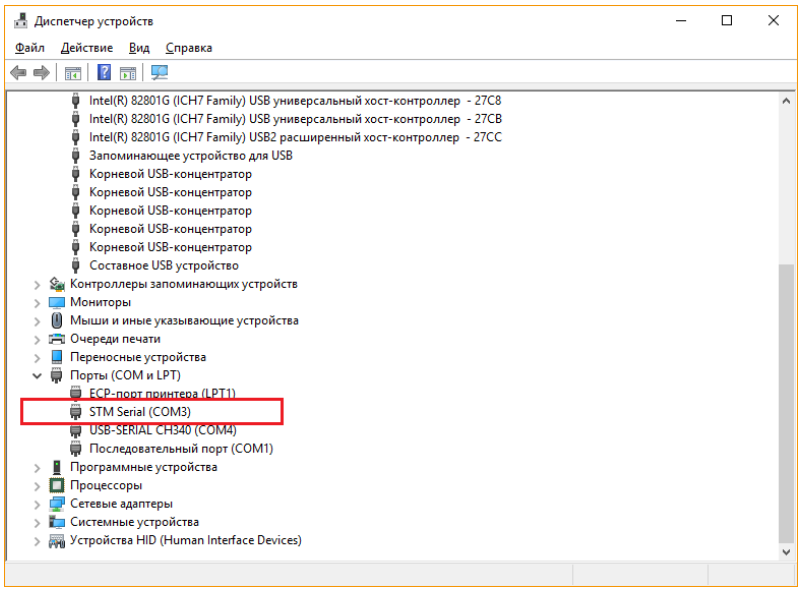


Рис. 9.7. Отображение модуля в диспетчере устройств

USB соединение реализуется в виде виртуального COM-порта, название которого, STM Serial, можно было изменить при настройке в STM32CubeMx. Если, конечно, вас не устраивает то, что он получил по умолчанию.

Осталось проверить работу программы. При работе по USART можно было использовать любой монитор для COM-порта. Можно ли это повторить сейчас я не знаю, поскольку использую программу Data Visualizer из состава Atmel Studio.

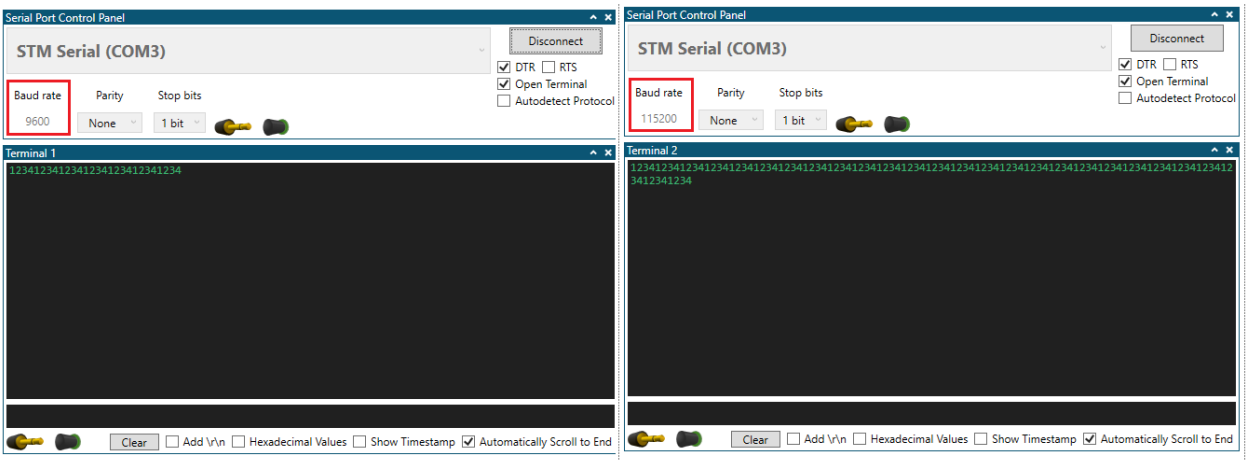


Рис. 9.8. Работа модуля с монитором при разных настройках монитора

И я не знаю, будет ли это доказательством, но вывод остаётся правильным, какую бы скорость обмена данными ни выбрать. Я привык к тому, что при работе RS232, если скорость обмена выбрана неверно, то и получить нужные символы нельзя.

Проверить скорость работы виртуального COM-порта можно иначе. Повторим в программе STM32CubeMx предыдущие настройки, но добавим ещё вывод PC13 на выход. Вновь полученную в результате генерации программу Keil пополним следующим кодом в основном цикле:

```
while (1)
{
    GPIOC->BSRR = GPIO_BSRR_BS13;
    CDC_Transmit_FS(snd,4);
    GPIOC->BSRR = GPIO_BSRR_BR13;
    CDC_Transmit_FS(snd,4);
/* USER CODE END WHILE */
}
```

Теперь светодиод на выводе PC13 мигает каждый раз после отправки 4 байтов.

Включим осциллограф, установим щуп осциллографа на вывод PC13 и посмотрим, что нам покажет экран.

При проведении этого эксперимента я исхожу из предположения, что микроконтроллер в основном цикле выполняет команды последовательно, переходя к следующей команде только тогда, когда завершится предыдущая. То есть, светодиод (вывод контроллера) изменит состояние тогда, когда завершится передача данных.

Осциллограф показывает следующее:

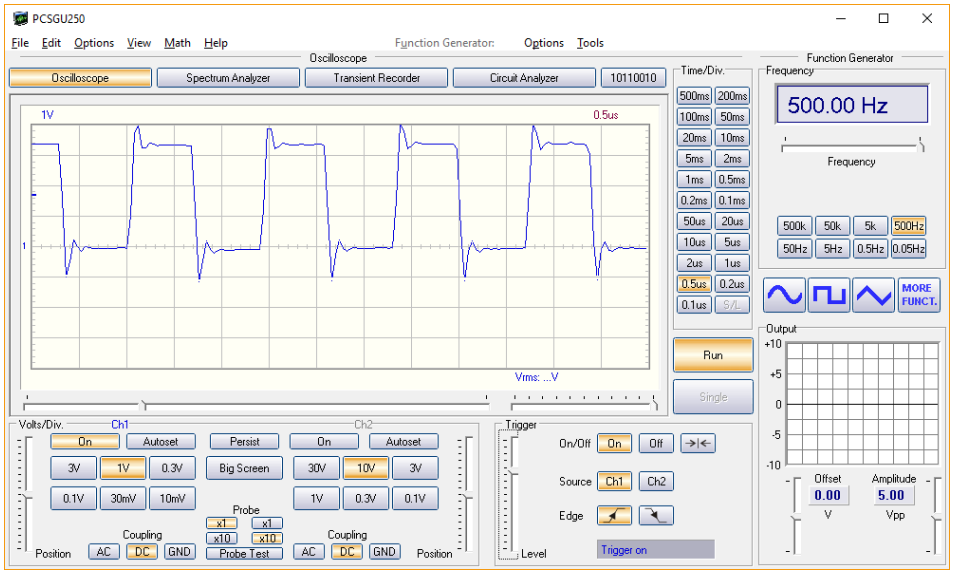


Рис. 9.9. Наблюдение за переключением светодиода

Длительность горения светодиода (или выключенного состояния) 0.7 мкс. За это время либо происходит передача 4 байтов данных, либо передача данных и работа основного цикла происходят параллельно, что тоже интересно. Поэтому, сменив осциллограф (частотный ресурс предыдущего почти исчерпан), подключив два канала к выводам модуля PA11 и PA12 (см. рис. 9.1), я наблюдаю такую картину на экране осциллографа:

Рис. 9.10. Сигналы на выводах порта

Во-первых, сигналы на выводах разностные; во-вторых, скорость изменения (6 битов за 0.5 мкс) порядка 12 Мбит/сек; и, наконец, вся передача данных завершается через 3 мкс.

Пора вспомнить старую присказку: если что-то ходит как утка, крякает как утка и выглядит как утка, то это, скорее всего, и есть утка. Вдобавок, основной цикл работает быстрее, чем происходит передача всех данных. Очень полезные, похоже, результаты, не так ли?

Завершение

Микроконтроллер STM32F103 достаточно мощный, чтобы провести ещё множество интересных экспериментов. В частности, если вам это будет полезно, вы можете проверить работу программ, которые можно скачать в качестве примеров для STM32F103 и программы Keil uVision5.

Для начинающих, пожалуй, работать с этим контроллером несколько сложно. Но, если вы освоились с PIC и AVR микроконтроллерами, вам, наверное, захочется посмотреть, что собой представляют и эти контроллеры.

Не буду вас убеждать, но мне показалось, что использовать для первого знакомства модуль STM32F103C8 вкупе с программами STM32Cube и Keil очень удобно. Позже, прочитав описания и руководства, вы можете обратиться к другим средам разработки, их довольно много. Но и в этом случае, думаю, вы согласитесь со мной, что написать несколько строк, чтобы получить работающее устройство – это неплохая замена многостраничному тексту кода. Впрочем, это дело убеждений и вкуса.