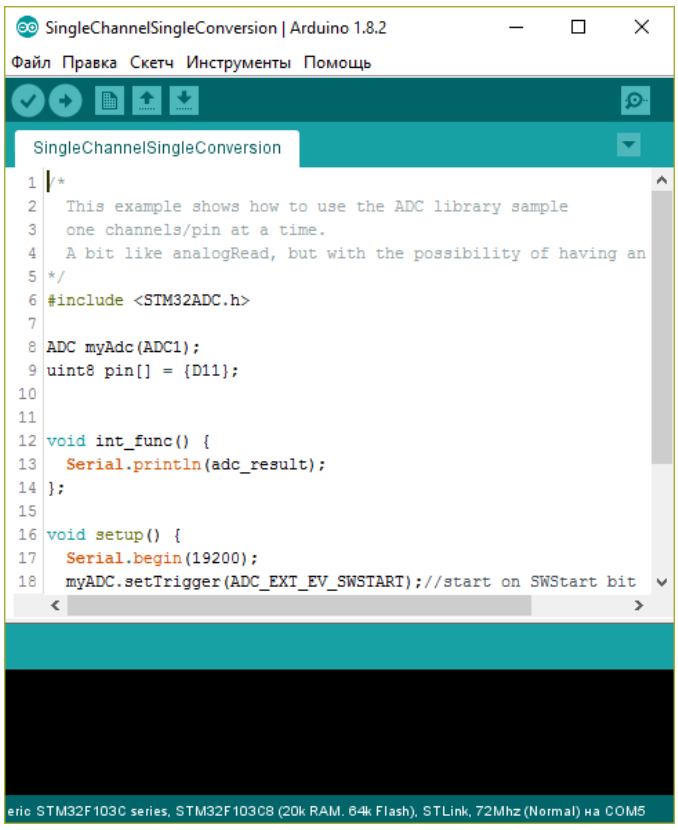
В.Н. Гололобов

Arduino & STM32F103C8



Москва - 2017

Оглавление

1 Настройка среды Arduino для поддержки STM32F103C8 3

2 Blink с помощью STM32 5

3 Приключения начинаются 7

4 Аналоговый вход 9

5 Полезные мелочи 12

6 Приключения продолжаются 15

7 Ещё немного о примерах 18

Для тех, кто изучал микроконтроллер, используя модуль Arduino, стала привычной среда разработки и язык этого модуля. Достигнув должного уровня в работе с микроконтроллерами, эти радиолюбители могли заинтересоваться более мощными микроконтроллерами, например, STM32.

Недорогой модуль STM32F103C8 очень подходящая кандидатура для этого. Внешнее сходство модуля с Arduino Nano обманчиво. Модуль STM32F103C8 потребует программатора ST-Link для программирования (или переходника USB/TTL с переключателем напряжения на 3.3В, на всякий случай) и кабеля USB с разъёмом Micro на втором конце. В этом случае вы можете проверить работу всех примеров, программируя модуль из среды Arduino.

1 Настройка среды Arduino для поддержки STM32F103C8

Во-первых, версия Arduino 1.7 поддерживала эту настройку, а версия 1.8.1, похоже, перестала поддерживать аналогичным образом, хотя появилась встроенная поддержка отдельных версий контроллеров ARM. Сегодня появилась версия 1.8.2, которая вновь позволяет работать с STM32F103C8, как это получалось раньше.

А получалось это так: необходимо скачать из Интернета пакет Arduino_STM32-мастер. Его следует разархивировать и добавить в то место, где установлена программа Arduino (*Program Files\Arduino*), в папку *hardware*.

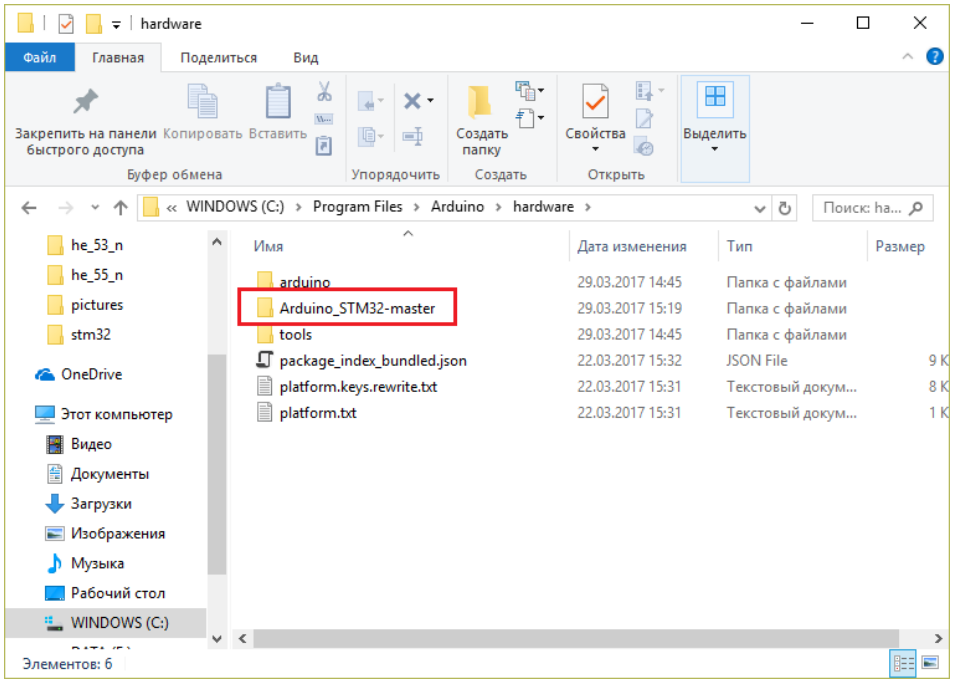


Рис. 1.1. Место добавления Arduino_STM32-мастер

При следующей загрузке программы плату можно обнаружить среди доступных модулей Arduino:

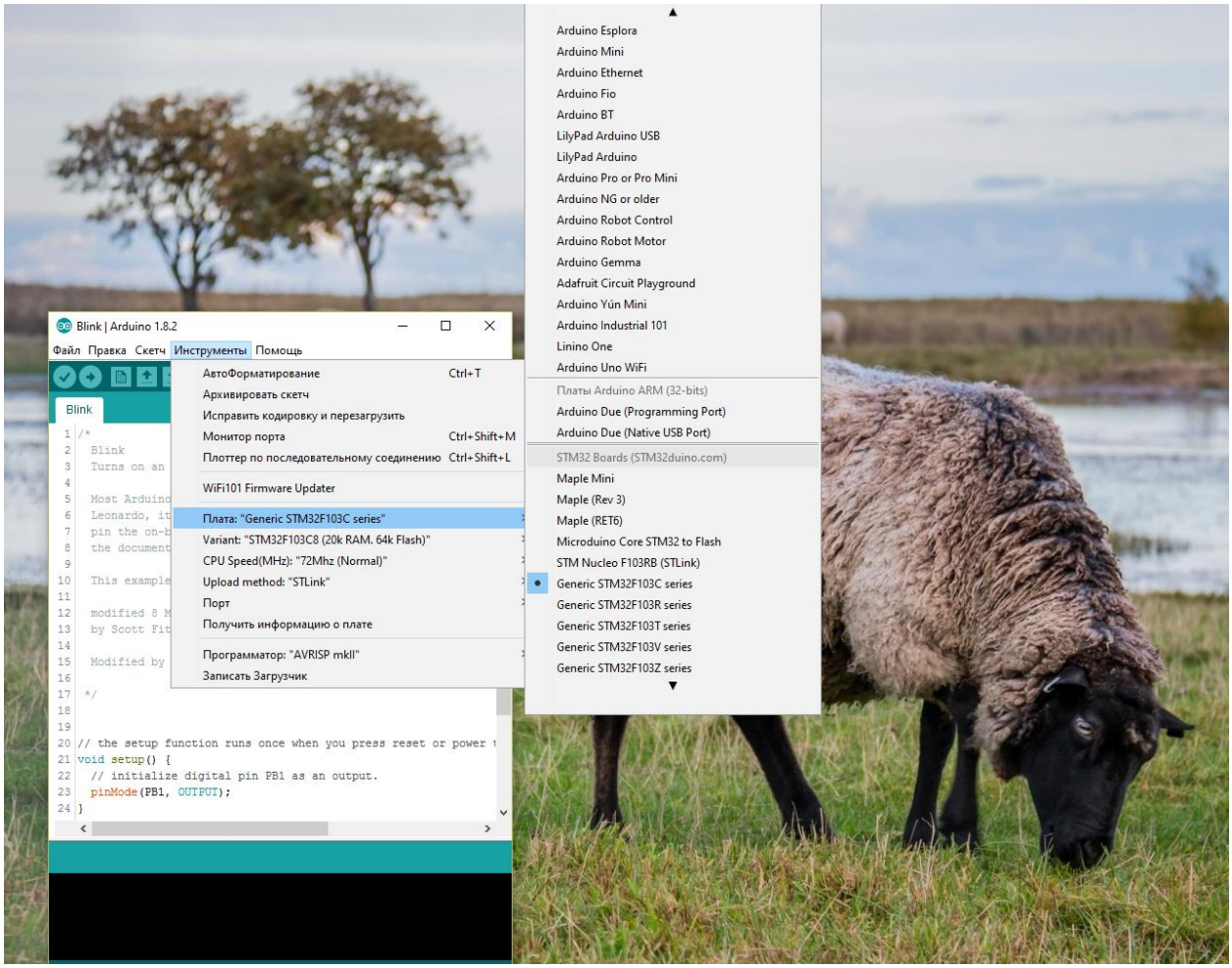


Рис. 1.2. Появление новых плат в разделе Инструменты

Если для программирования использовать переходник USB/TTL, то следует выбрать порт в этом же разделе. На моём компьютере несколько гнёзд USB, виртуальный порт появляется как COM4 или COM8, что зависит от гнезда USB порта, а в разделе *Upload method* следует выбрать Serial.

Если для программирования (как на рисунке выше) использовать программатор ST-Link, то следует его указать в разделе выбора метода загрузки. После подключения платы STM32, после выбора варианта микроконтроллера в меню *Инструменты* можно загружать в МК программы.

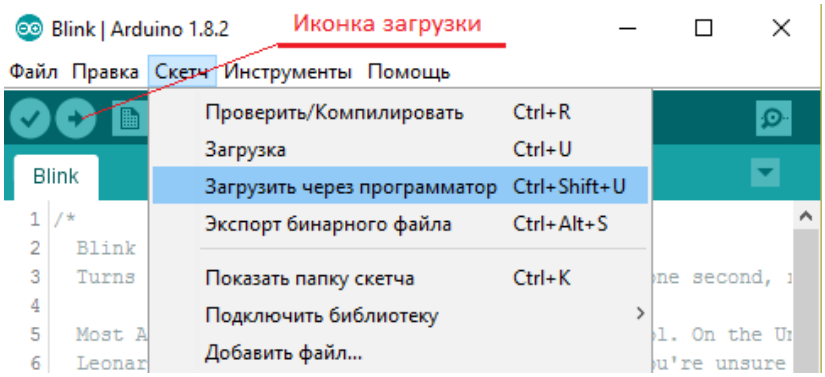


Рис. 1.3. Иконка загрузки и разделы меню

2 Blink с помощью STM32

Проверить правильность подключения и настройки удобнее всего с помощью программы *Blink*:

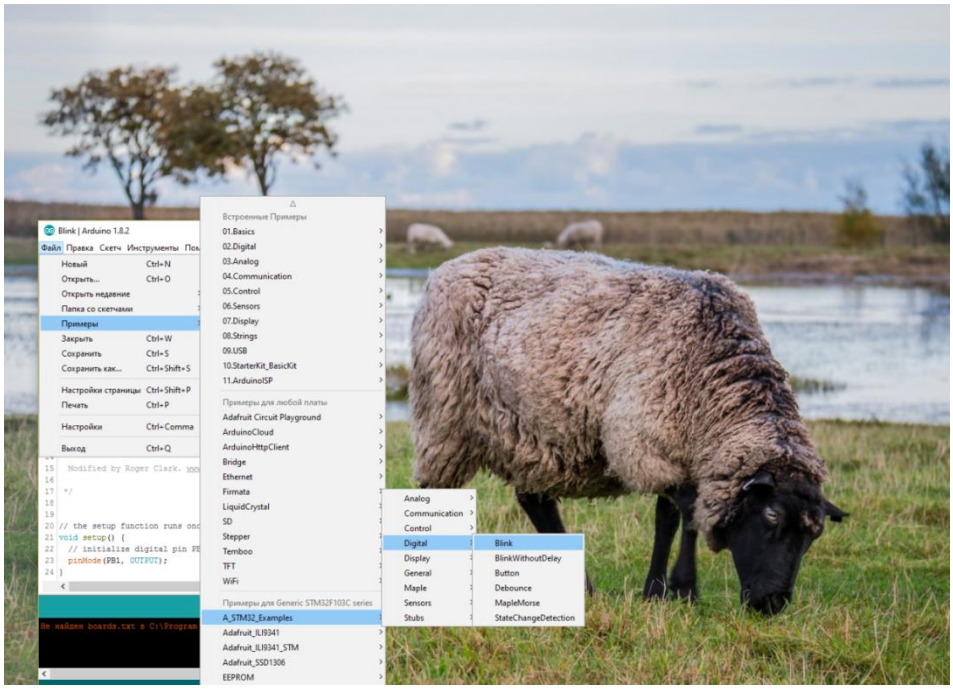


Рис. 2.1. Программа Blink для проверки

Лаконичная программа ничем, в сущности, не отличается от программы для Arduino. Но небольшую правку следует сделать. Длительность пауз лучше изменить, если платы STM32 новые, поскольку в них может уже быть загружена программа *Blink* с паузами в 1 секунду.

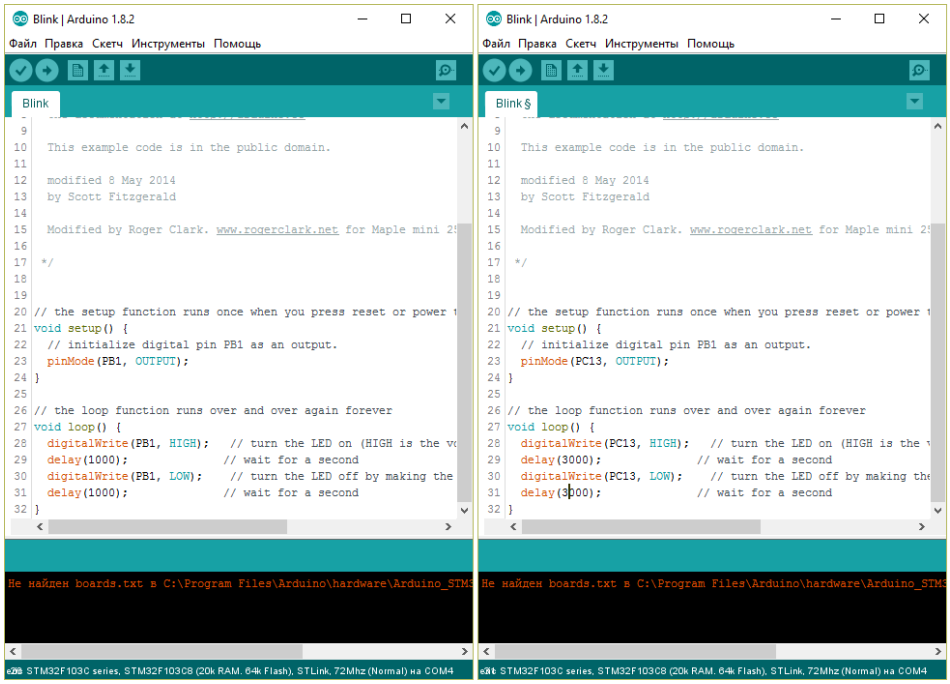


Рис. 2.2. Настройки в программе Blink

Светодиод начинает мигать сразу, но лучше проверить ещё раз, изменив паузу на 1 секунду. Убедившись, что всё работает, можно проверить работу примеров, приведённых выше.

Некоторые примеры требуют больше внимания. Вот один из них: *BlinkNcount*. После исправлений в тексте программы, после её загрузки через программатор ST-Link, модуль STM32 следует отсоединить от программатора и подключить к компьютеру с помощью USB кабеля. Подключение отображается появлением виртуального COM-порта, который идентифицируется как Maple Serial (COMx), где номер порта зависит от места включения, то есть, от гнезда USB. Однако работает виртуальный порт в этом случае по протоколу USB, в чём можно убедиться, если включить встроенный в программу Arduino монитор порта:

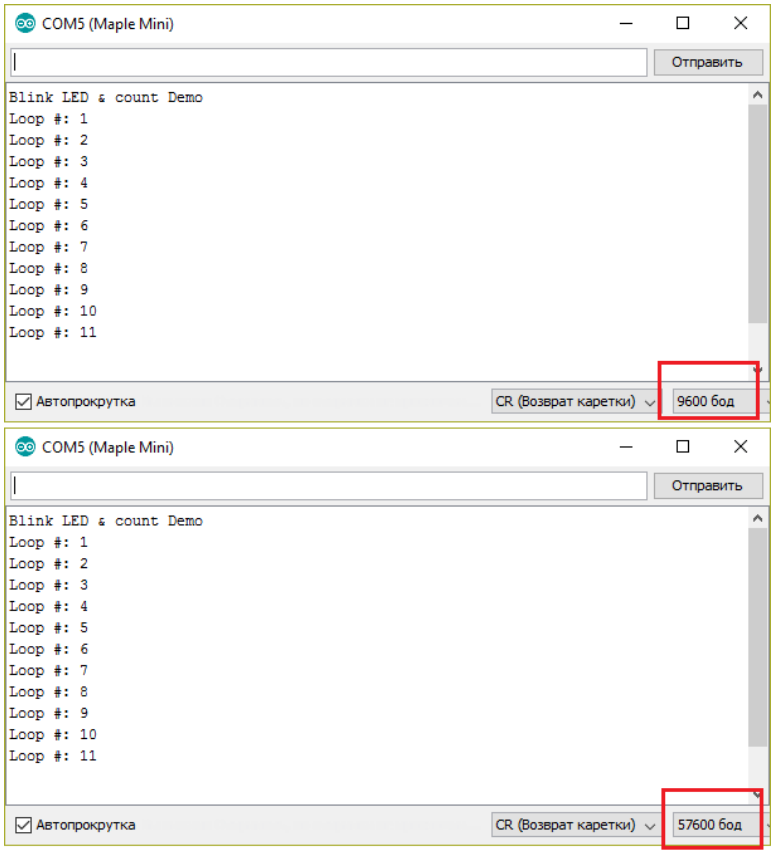


Рис. 2.3. Изменение скорости работы порта, не влияющее на работу программы

То есть, изменение скорости работы порта не влияет на его работу, что для COM-порта не характерно: неправильный выбор скорости работы приведёт к появлению либо случайных символов, либо невидимых управляющих символов вместо сообщения.

Правильно скомпилированная и загруженная программа заставит светодиод на плате STM32F103C8 быстро мигать. Когда же вы подключите монитор порта, скорость переключения светодиода понизится, и будет соответствовать указанной в программе.

3 Приключения начинаются

Небольшой обзор примеров, который я намеревался сделать, сразу споткнулся на примере использования АЦП.

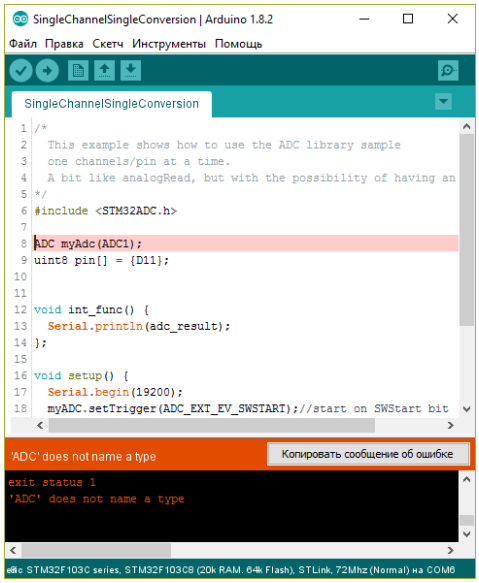


Рис. 3.1. Первые неприятности

Указанное сообщение, что ADC не является типом, справедливо, наверное. Открыв заголовочный файл *STM32ADC.h*, можно увидеть объявление класса:

```
class STM32ADC{…
```

Попробуем переделать это… чтобы споткнуться в другом месте:

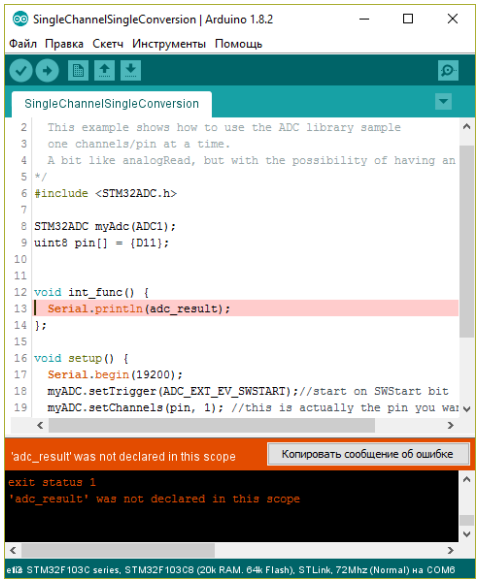


Рис. 3.2. Следующая ошибка

Попробуем исправить и это, попутно заменив вывод D11 на PA0.

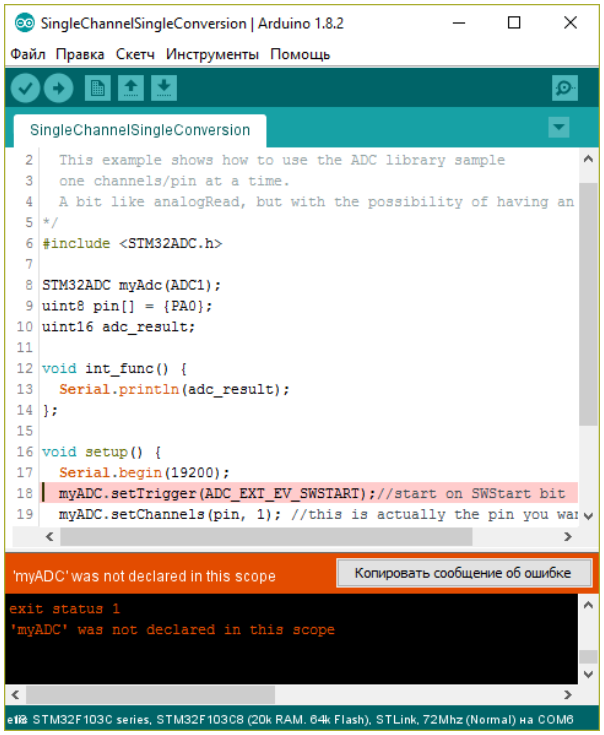


Рис. 3.3. Новая ошибка

Исправим это:

```
STM32ADC myADC(ADC1);
```

Но следующая ошибка касается команды: `myADC.attachADCInterrupt(int_func);`

Действительно, если заглянуть в заголовочный файл, то обнаружится такая функция:

```
void attachInterrupt(voidFuncPtr func, uint8 interrupt);
```

Что делать с прерыванием, я пока не знаю, поэтому строку убираю (пусть это будет комментарий). Но и это не помогает скомпилировать программу, поскольку обнаруживается следующая ошибка, ссылающаяся на раздел *utility* в библиотеке STM32ADC. А в этой утилите, если открыть заголовочный файл, ссылка на другую библиотеку…

Что ж, если что-то не получается, то следует разобраться в причинах. Но с этим, пожалуй, повременим, а попробуем примеры, которые компилируются и работают.

4 Аналоговый вход

Насколько я понимаю, использование входа порта для ввода аналоговых сигналов связано с работой АЦП. Вместе с тем, что приятно, пример *AnalogInput* компилируется и работает. Эта разновидность «мигания светодиодом» управляется (паузы) напряжением, считанным по заданному входу. Я использую вход PA0, который работает с ADC1.

На этот вход я подаю сигнал от генератора:

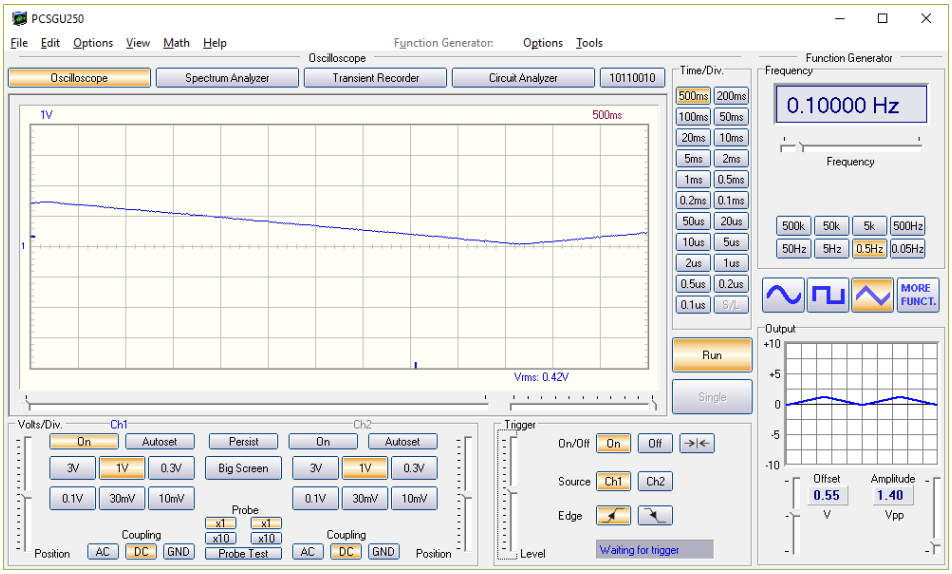


Рис. 4.1. Сигнал, подаваемый на аналоговый вход

С помощью такого сигнала можно наблюдать, как меняются интервалы включённого и выключенного состояния светодиода. Но это можно было наблюдать и с помощью потенциометра, как указано в комментарии к программе.

```
int sensorPin = PA0;    // Выбор входа для подключения потенциометра
int sensorValue = 0; // Переменная для хранения значения от входа

void setup() {
    // ОбъявлениesensorPin как INPUT_ANALOG:
    pinMode(sensorPin, INPUT_ANALOG);
    // Объявление вывода светодиода как OUTPUT (обозначения в оригинале
    // программы похожи на те, что на плате Arduino, а плата из Китая
    // маркирована иначе, но пока приведённые примеры работают).
    pinMode(PC13, OUTPUT);
}

void loop() {
    // Чтение значения с аналогового входа:
    sensorValue = analogRead(sensorPin);
    // Включение светодиода:
    digitalWrite(PC13, HIGH);
    // Остановка программы на <sensorValue> миллисекунд:
    delay(sensorValue);
    // Выключение светодиода:
```

```
    digitalWrite(PC13, LOW);
    // Остановка программы на <sensorValue> миллисекунд:
    delay(sensorValue);
}
```

В этом примере работа АЦП задаётся в совсем скрытом виде, но пример работает.

Ещё один пример называется *AnalogInOutSerial*. Он интересен тем, что АЦП принимает входной сигнал, отправляет его на монитор и формирует широтно-импульсный (PWM) сигнал на выходе контроллера.

Изменения, которые вносятся, в данном случае выглядят так:

```
const int analogInPin = PA0; // Аналоговый вход для потенциометра
const int pwmOutPin = PA9;    // PWM вывод для подключения светодиода
```

Если вывод PA0 уже использовался в предыдущем случае, то с выводом PA9 история более запутанная. Обозначение в оригинале программы 9 может относиться и к A9, но и к B9 – обозначениям на плате STM32F103C8. Листая описание на 1000 страницах, трудно сделать выбор.

Единственное, что ясно, так это то, что формируется PWD с помощью таймера. Таймеров у микроконтроллера несколько. Однако есть надежда, что это таймер 1, поскольку используется переменная:

```
int sensorValue = 0;    // Переменная для чтения значения от потенциометра
```

Целое значение – это 16 битов.

Я, разбираясь с выводом для PWM, использовал программу STM32CubeMx, что и вам советую. Вот настройки:

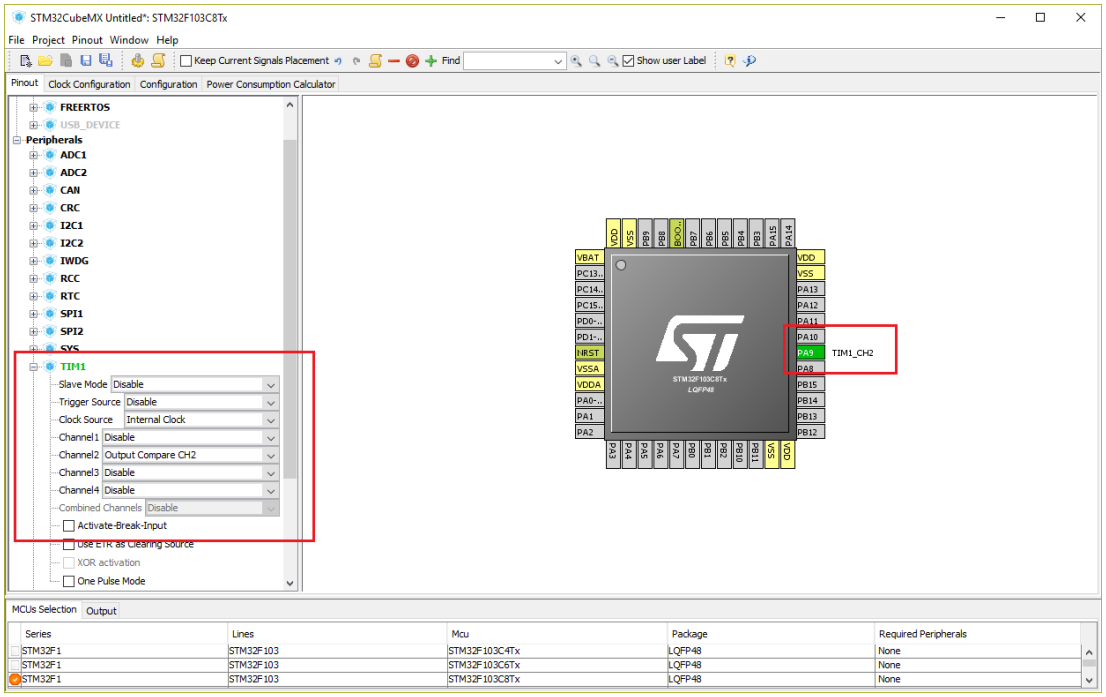


Рис. 4.2. Настройки таймера

После настройки можно видеть, с каким выводом эти настройки будут связаны. Если выбрать другой канал в настройках, то изменится и вывод.

После компиляции и загрузки программы, отключив программатор ST-Link, можно соединить модуль STM32F103C8 его кабелем для USB порта с этим портом. Запустив наблюдение с помощью монитора порта, можно увидеть значения на входе и выходе:

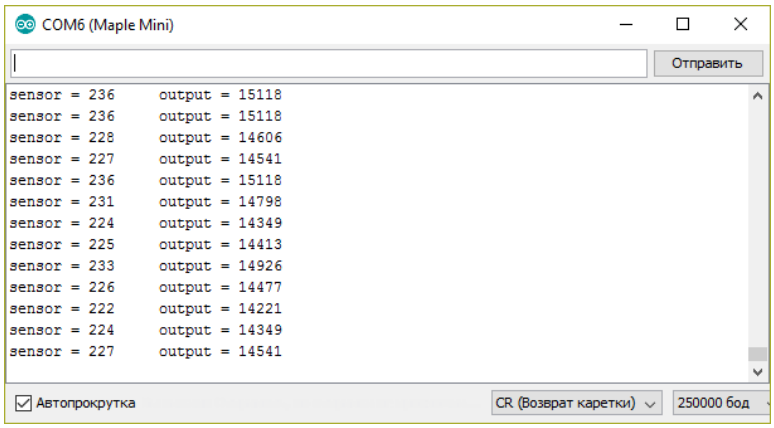


Рис. 4.3. Что показывает монитор порта

При проведении проверки использовался генератор ступенчатого напряжения на очень низкой частоте. Осциллограф на выводе PA9 (A9 на плате) показывает изменения выходного сигнала.

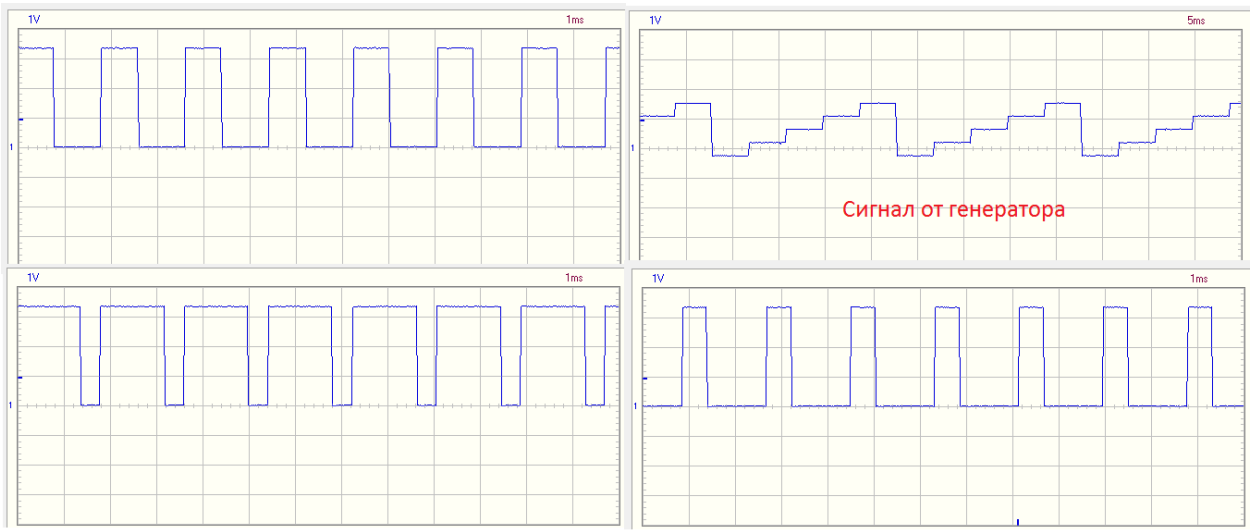


Рис. 4.4. Сигналы на входе A0 и выходе A9

5 Полезные мелочи

Начинающим подчас трудно справляться с поставленной перед собой задачей. Поэтому им весьма полезно разобраться в примерах, которые, как правило, приходят вместе с программами.

Один из примеров – это вывод чисел с плавающей точкой. Другими словами десятичных чисел. Среди примеров есть программа *Print_Float*. Загрузив её в модуль STM32F103C8, подключив модуль к порту USB, запустив в программе Arduino монитор порта, вы можете наблюдать вывод нескольких заданных в программе чисел:

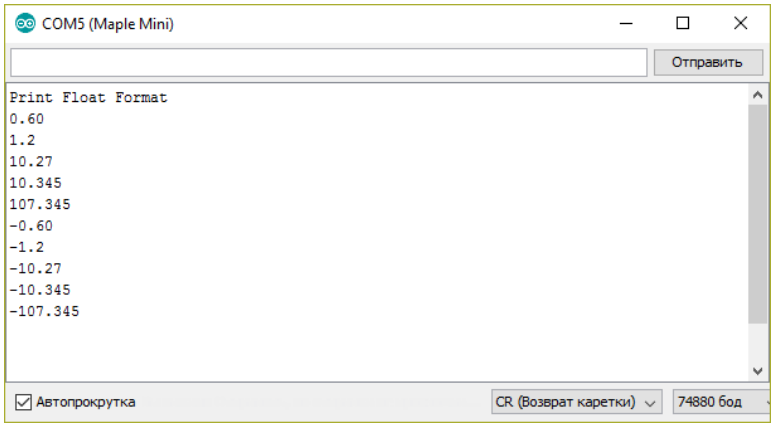


Рис. 5.1. Вывод десятичных чисел на монитор

Если не нажимать перед подключением к монитору кнопку сброса на плате, то строка *Print Float Format* не появится, поскольку её вывод вне основного цикла программы.

Вывод чисел в таком формате обязан функции *print_float()*. Зачем она может понадобиться?

АЦП модуля STM32 двенадцати битовое, то есть, имеет хорошее разрешение, что позволяет проводить достаточно точные измерения напряжения. Считывание напряжения с помощью АЦП мы уже использовали ранее. Попробуем прочитать напряжение батарейки и вывести это значение на монитор.

АЦП позволяет измерять напряжение, положим, от нуля до 3.3 В. Измеренное значение выражается числами от 0 до 4096, что даёт для одной «ступеньки» измерения значение 0.000806.

Переделаем немного программу, с которой работали выше, но начнём не с измерения, а с проверки, предположив, что АЦП отправляет число 1000.

Добавим глобальные переменные (выше разделов):

```
#define BAUD 9600
float volt;
int d_volt = 1000;
const float m = 0.000806;
```

Немного подправим функцию *print_float()*:

```
void print_float(float f, int num_digits)
{
```

```
    int f_int;
    int pows_of_ten[5] = {1, 10, 100, 1000, 10000};
    int multiplier, whole, fract, d, n; и т.д.
```

Удалим все числа, которые выводились в оригинале программы в основном цикле, оставив только вывод нашего значения, и добавим паузу:

```
      volt = d_volt*m;
      print_float(volt, 4);
      Serial.println();
      delay(1000);
```

Теперь после загрузки откомпилированной программы в модуль можно наблюдать на мониторе:

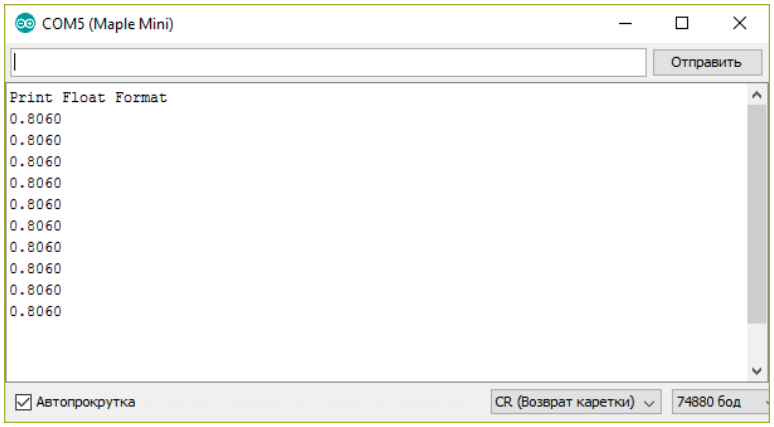


Рис. 5.2. Проверка нашего решения

Обращаясь к примеру, где мы считывали аналоговое значение с помощью АЦП, добавим это чтение на выводе PA0:

```
void setup()                    // run once, when the sketch starts
{
  pinMode(PA0, INPUT_ANALOG); и т.д.

  while(1)
  {
    d_volt = analogRead(PA0);
    volt = d_volt*m;
    print_float(volt, 4);
    Serial.println();
    delay(1000);
  }
```

Загружаем, подключаем батарейку 1.5 В и получаем:

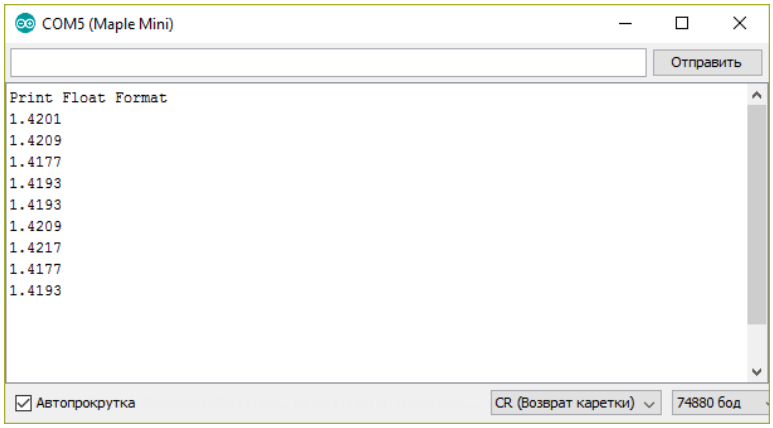


Рис. 5.3. Рабочее испытание программы

При этом возникает законный вопрос, а как мы читаем данные с помощью АЦП, используя функцию `analogRead()`? Мы выводим четыре знака после запятой, но не является ли это только арифметическим трюком?

Подобный вопрос возвращает нас к тому, что описано в главе «Приключения начинаются».

6 Приключения продолжаются

Открываем в программе Arduino файл с названием *SingleChannelSingleConversion*, начинаем исправлять текст так, как описано выше. Доходим до места, где компиляция застревает, ссылаясь на файл util_adc.c. Видимо, этот файл важен для работы программы. Открываем этот файл в блокноте или редакторе WordPad (я предпочитаю этот вариант). Запускаем поиск, ориентируясь на сообщение об ошибке.

Кстати, в Windows 10 лучше скопировать этот файл на рабочий стол, где и произвести все манипуляции. Если пытаться исправить его в директории установки Arduino, то не получается сохранить исправленный файл. А после правки на рабочем столе нет причин его не сохранять, хотя приходится заменять его с двух попыток – не любит ОС вмешательства в систему.

Итак, в этом файле ошибка находится по следующему «адресу»:

```
void enable_internal_reading(adc_dev *dev) {
  dev->regs->CR2 |= ADC_CR2_TSVREFE; // Было ADC_CR2_TSEREFE;
  }
```

Сам файл отыскивается в долгом пути, который начинается с *C:\Program Files\Arduino\hardware*, продолжается в папке *Arduino_STM32-master*, где есть папка *STM32F1*:

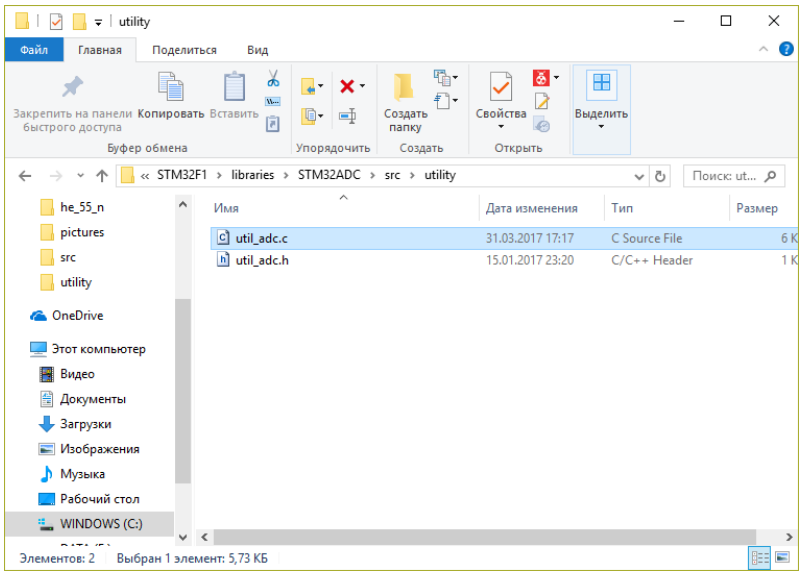


Рис. 6.1. Место, где находится файл с ошибкой

Последнее исправление касается переменной int adc_result; При компиляции появляется сообщение о её многократном определении. Чтобы не затевать долгие поиски, я убираю приставку adc. Теперь программа компилируется, загружается, но ничего не делает. Возможно, причина в том, что следовало внимательнее разобраться с последней ошибкой. Не исключаю этого, но я не такой ярый поклонник использования прерываний, чтобы сейчас на этом зацикливаться, поэтому убираю две строки исходного текста, оставив две в разделе void setup():

```
  Serial.begin(19200);
  myADC.setChannels(pin, 0); // актуальный вывод для измерения
```

В пустой ранее основной цикл программы я добавляю несколько строк:

```
void loop(){
myADC.startConversion();
delay(1000);
result = myADC.getData();
  Serial.println(result);
}; //end loop
```

Программа компилируется и работает:

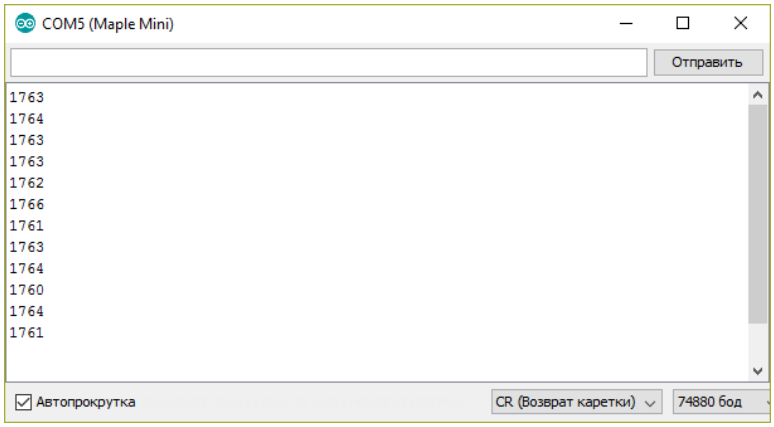


Рис. 6.2. Работа переделанной программы для одного канала и единичных измерений

В программу можно добавить преобразование в формат с плавающей точкой, но можно проверить и простым умножением полученного числа на коэффициент 0.000806. Получается, что напряжение батарейки равно 1.4218. А это вполне совпадает с тем значением, которое мы получили, используя простую функцию `analogRead()`.

Попутно можно определить и точность измерения. Если подключить выводы к источнику калибровочного напряжения, дающего 2.49942В, то прочитав:

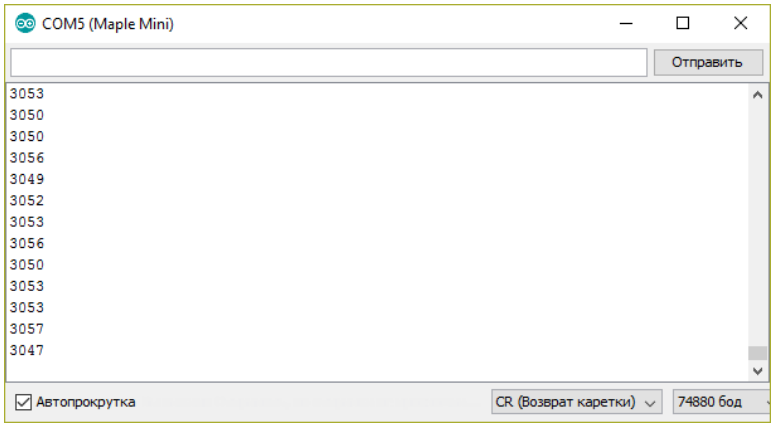


Рис. 6.3. Измерение калибровочного напряжения 2.5В

…возьмём среднее значение десяти измерений 3052.2, умножим на коэффициент 0.000806, чтобы получить 2.4601. В этом случае ошибка получается 1.6%. Но… Коэффициент, который мы применили для перевода полученного от АЦП значения в напряжение, мы получили при некотором предположении: конвертор измеряет постоянное напряжение до величины 3.3В. А это может оказаться и не совсем так. Особенно, если речь идёт о точных измерениях.

И последнее, программа для измерения из примеров (называется *AnalogReadSerial*):

```
void setup() {
Serial.begin(115200); // Нужно только при использовании USB/TTL переходника
pinMode(PA0, INPUT_ANALOG);
}
void loop() {
  int sensorValue = analogRead(PA0);
  Serial.println(sensorValue, DEC);
}
```

Даёт такие значения:

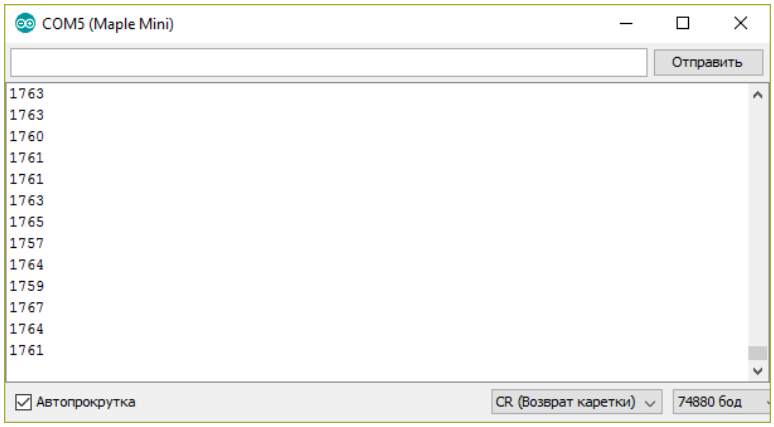


Рис. 6.4. Значения, показываемые программой, представленной выше

Можно сказать только: «Лаконичная программа, дающая очень хороший результат».

7 Ещё немного о примерах

Примеров для контроллера STM32F103 много.

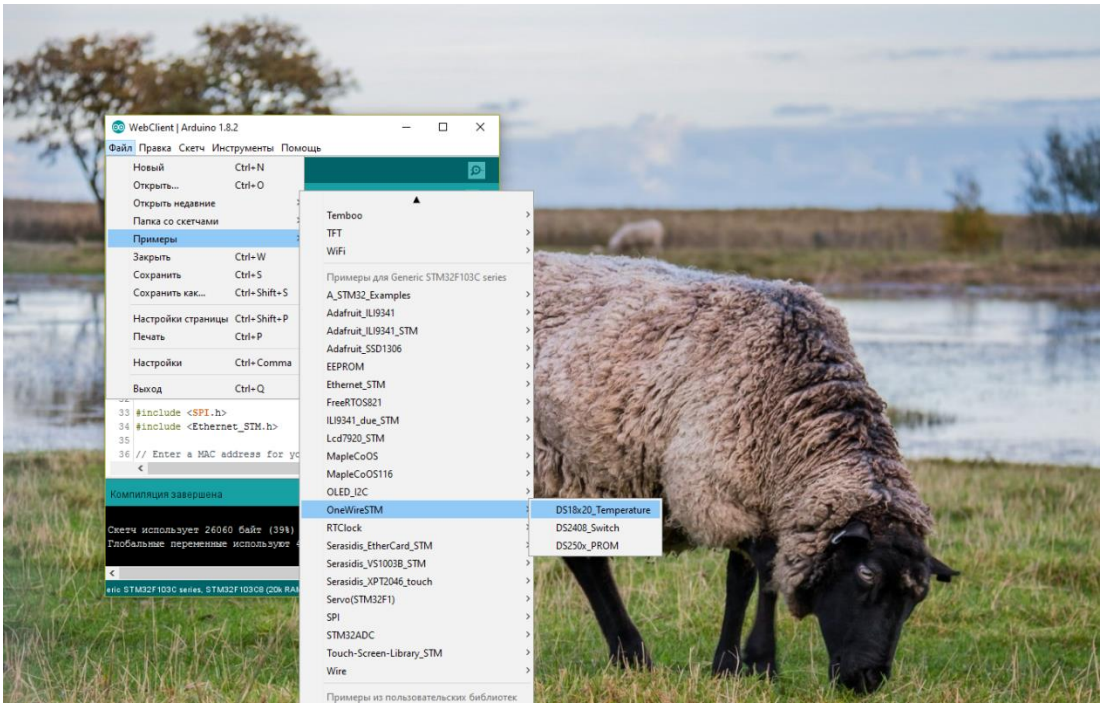


Рис. 7.1. Примеры из набора для программы Arduino

Некоторые примеры, видимо, потребуют правки, как это было показано выше. Другие примеры для проверки потребуют, видимо, дополнительных устройств. Например, это касается всех программ для *Ethernet*.

У меня есть некоторые компоненты, чтобы проверить правильность работы представленных примеров. Так есть датчик температуры DS18B20. Пример интересен тем, что обычно используется выход с открытым коллектором, и требуется принимать и передавать данные по этому выводу.

У микроконтроллера STM32 для работы по протоколу 1-wire используется USART. Все необходимые переключения выполняются внутри микросхемы.

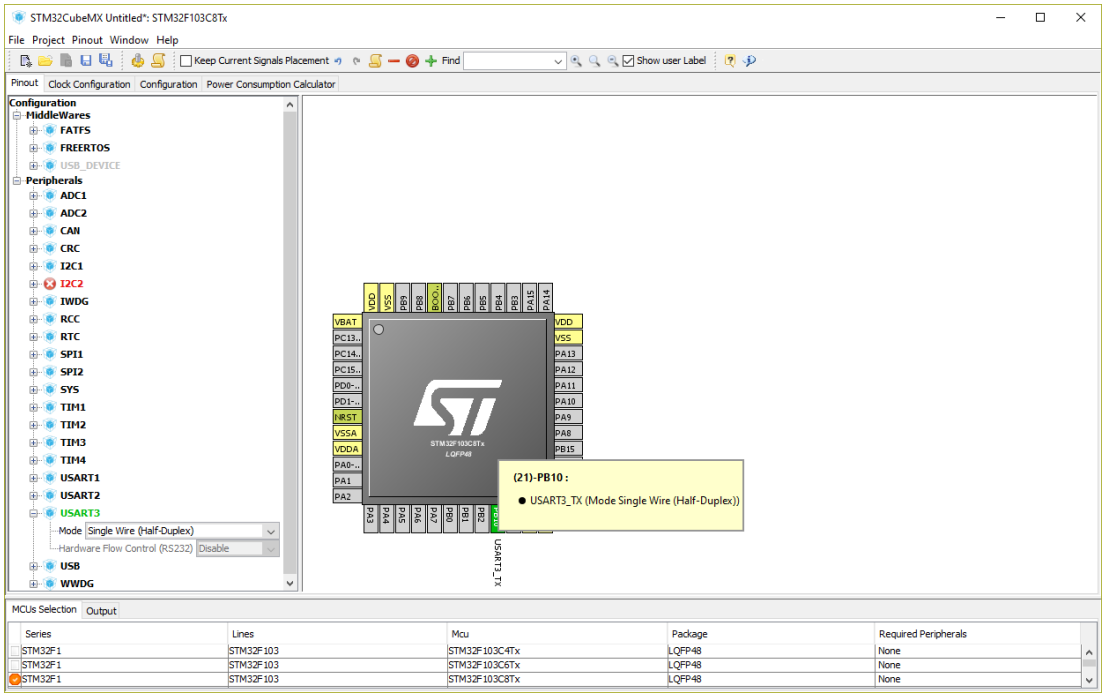


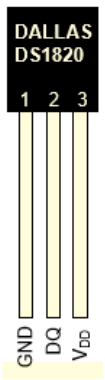
Рис. 7.2. Настройка STM32 в STM32CubeMx

В примере, отмеченном на рисунке 7.1, программа использует вывод 10. Но у модуля STM32F103C8 несколько выводов с номером десять. Поэтому я вновь использую программу Cube, чтобы определиться с нужным выводом.

Иногда получается, если оставить вывод так, как он обозначен в программе, но так бывает не всегда, лучше, это моё мнение, сразу определять вывод, как в этом случае, полным «именем» PB10.

Итак, программа из набора примеров *DS18x20_Temperature*. Программа компилируется без проблем и загружается. Остаётся на макетной плате обустроить датчик, и соединить его с модулем STM32.

Программа в комментариях напоминает о необходимости добавить резистор 4.7 кОм (он подключается между плюсом питания и выводом данных). Проводя эксперименты, полезно иметь под рукой «цоколёвку» компонентов. Я стараюсь не пренебрегать этим, что советую делать и вам.

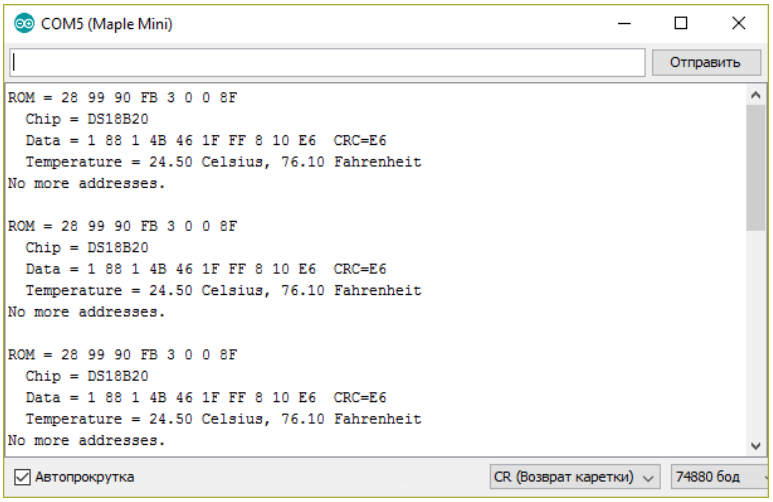


Резистор 4.7 кОм подключается между выводами DQ и Vdd.

Питающее напряжение для датчика 3-5 вольт. Поэтому можно использовать напряжение 3.3В, которое можно снять с выводов модуля STM32.

Рис. 7.3. Цоколёвка датчика температуры

После всех подключений можно запустить программу, чтобы наблюдать следующее:



```
ROM = 28 99 90 FB 3 0 0 8F
  Chip = DS18B20
  Data = 1 88 1 4B 46 1F FF 8 10 E6  CRC=E6
  Temperature = 24.50 Celsius, 76.10 Fahrenheit
No more addresses.

ROM = 28 99 90 FB 3 0 0 8F
  Chip = DS18B20
  Data = 1 88 1 4B 46 1F FF 8 10 E6  CRC=E6
  Temperature = 24.50 Celsius, 76.10 Fahrenheit
No more addresses.

ROM = 28 99 90 FB 3 0 0 8F
  Chip = DS18B20
  Data = 1 88 1 4B 46 1F FF 8 10 E6  CRC=E6
  Temperature = 24.50 Celsius, 76.10 Fahrenheit
No more addresses.
```

Рис. 7.4. Вывод результата работы программы на монитор порта

Все датчики имеют встроенные при изготовлении номера, по этому номеру происходит адресация к датчику. Программа показывает «адрес» датчика температуры (у меня он записан, а вы можете записать его, если повторяете проверку этой программы). Диалог датчика и модуля выглядит на экране осциллографа так:

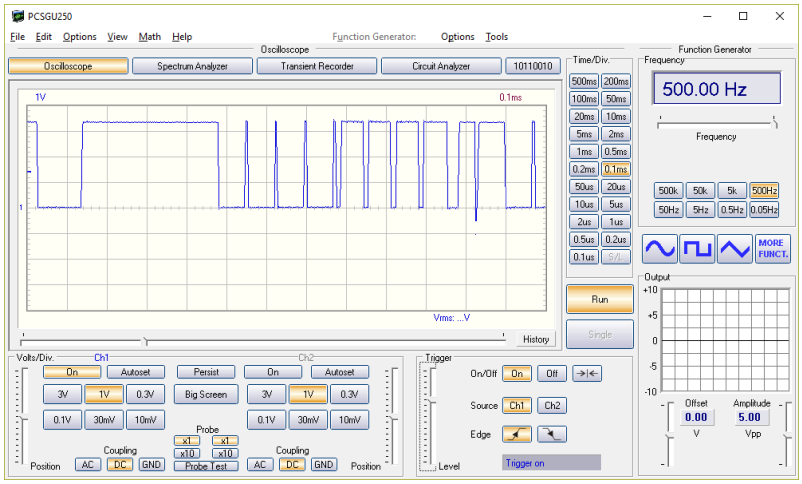


Рис. 7.5. Сигналы на экране осциллографа

Датчиков, подключаемых к двухпроводной линии, может использоваться несколько. Если добавить к модулю дисплей, то на него можно выводить температуру от всех датчиков. Такая конструкция при питании от батарейки вполне может применяться для наблюдения за температурой в теплице или инкубаторе, поскольку датчик показывает температуру достаточно точно.

Если протокол 1-wire требует двух проводов для соединения с датчиком температуры, то протокол SPI требует трёх проводов для диалога и ещё одного для подключения выбранного устройства, имеющего вывод выбора (CS).

Пример программы, работающей с SPI, не очень показателен, хотя в комментариях точно указаны выводы модуля STM32 для подключения внешнего устройства. У меня есть Ethernet модуль для Arduino, который работает по этому протоколу, и на плате есть гнездо для SD-карты, которая тоже работает по этому протоколу.

У модуля Ethernet сигнальные выводы одинаковы и для сети, и для SD-карты, разными будут выводы выбора между этими устройствами.

Чтобы не переделывать программу, я использую осциллограф для наблюдения за сигналами. Это покажет, «дышит» ли интерфейс SPI.

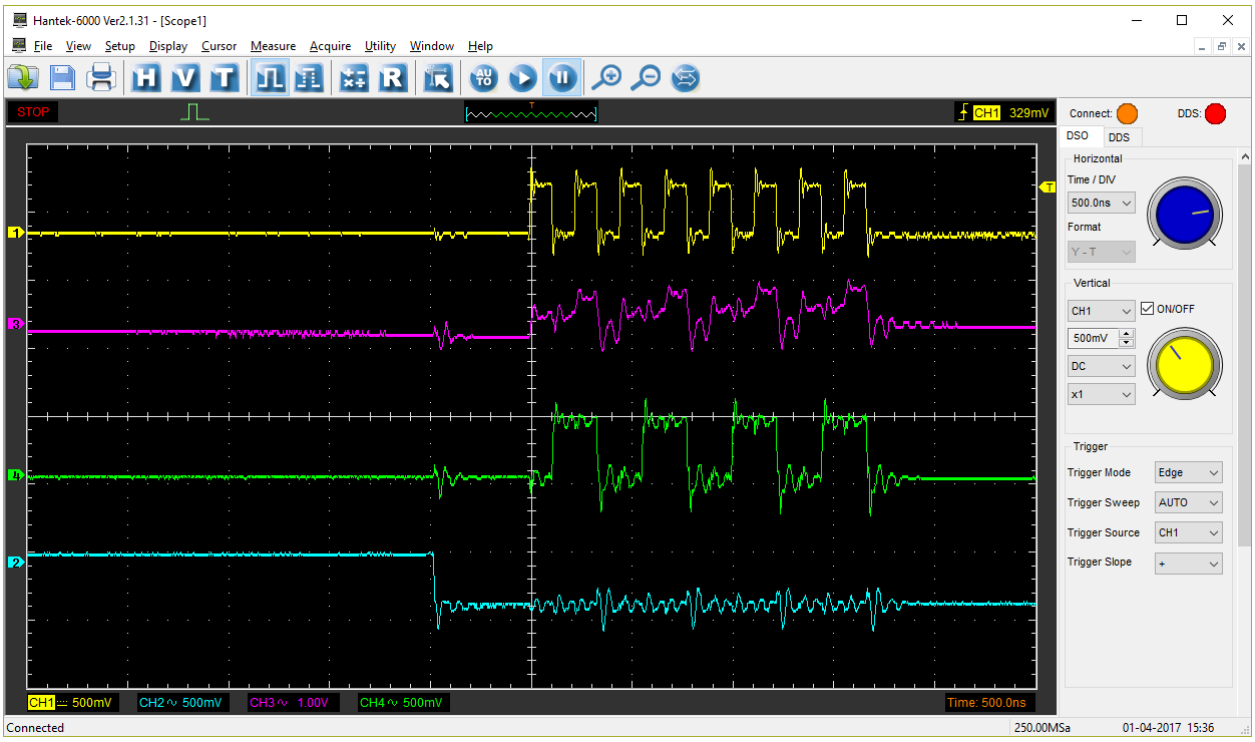


Рис. 7.6. Сигналы на выводах SPI

Самый верхний сигнал – это тактирующие импульсы. Самый нижний – это сигнал выбора устройства, над которым, скорее всего, отправленная команда 0x55. Программу можно было бы дописать, чтобы получить отклик, который можно вывести на монитор, но это потребует более глубокого погружения в протокол SPI и устройство модуля Ethernet.

Далее в мои намерения входил плавный переход к рассказу о связке программ STM32CubeMx и Keil uVision v.5. Но меня смутил сигнал на одном из выводов (второй на рисунке выше). Поэтому я решил проверить, как это выглядит в программе Keil.

Настройку работы программы осуществляет STM32CubeMx. Я не нашёл в Интернете подробного описания настройки, поэтому решил, что достаточно выбрать интерфейс SPI.

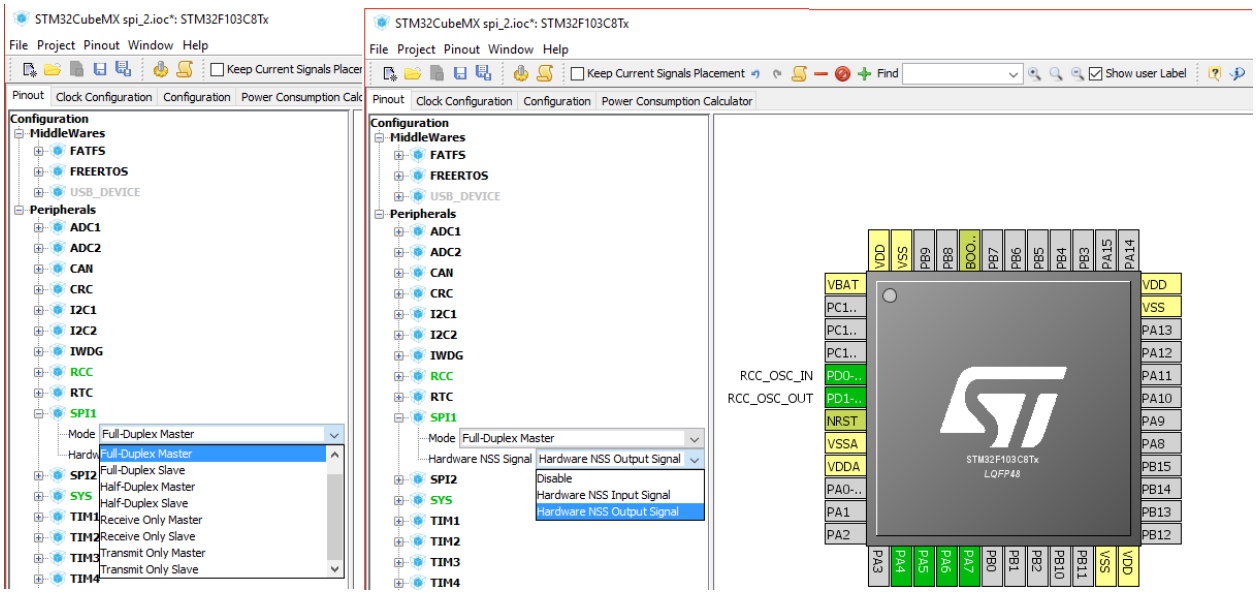


Рис. 7.7. Выбор настроек в STM32CubeMx

Выбор аппаратного сигнала NSS мне показался естественным, поскольку остальные сигналы формируются интерфейсом. Добавив переменную буфера для отправки и команду отправки, я ожидал появления сигналов на выводах порта А:

```
uint8_t SPI1_Buffer_Tx[] = {0x55}; и т.д.
HAL_SPI_Transmit(&hspi1, SPI1_Buffer_Tx, sizeof(SPI1_Buffer_Tx), 100);
HAL_Delay(1000);
```

Но к моему разочарованию я не увидел ни одного сигнала.

По некоторому размышлению я решил, что не понимаю значения аппаратного формирования этого сигнала, поскольку устройств, работающих по интерфейсу SPI, может быть несколько…

Удалив выбор аппаратного формирования этого сигнала (выбор *Disable*), я решил настроить вывод PA4 на выход (щелчок левой клавишей мышки по выводу PA4 на рисунке микросхемы, и выбор из выпадающего меню GPIO_Output). Добавление в программу «ручного» управления выводом:

```
uint8_t SPI1_Buffer_Tx[] = {0x55}; и т.д.
GPIOA->BSRR = GPIO_BSRR_BS4; и т.д.
GPIOA->BSRR = GPIO_BSRR_BR4;
HAL_SPI_Transmit(&hspi1, SPI1_Buffer_Tx, sizeof(SPI1_Buffer_Tx), 100);
GPIOA->BSRR = GPIO_BSRR_BS4;
HAL_Delay(1000);
```

…изменило ситуацию к лучшему – появились сигналы. Сравнивая сигнал данных и сигнал на выводе, который привлёк моё внимание:

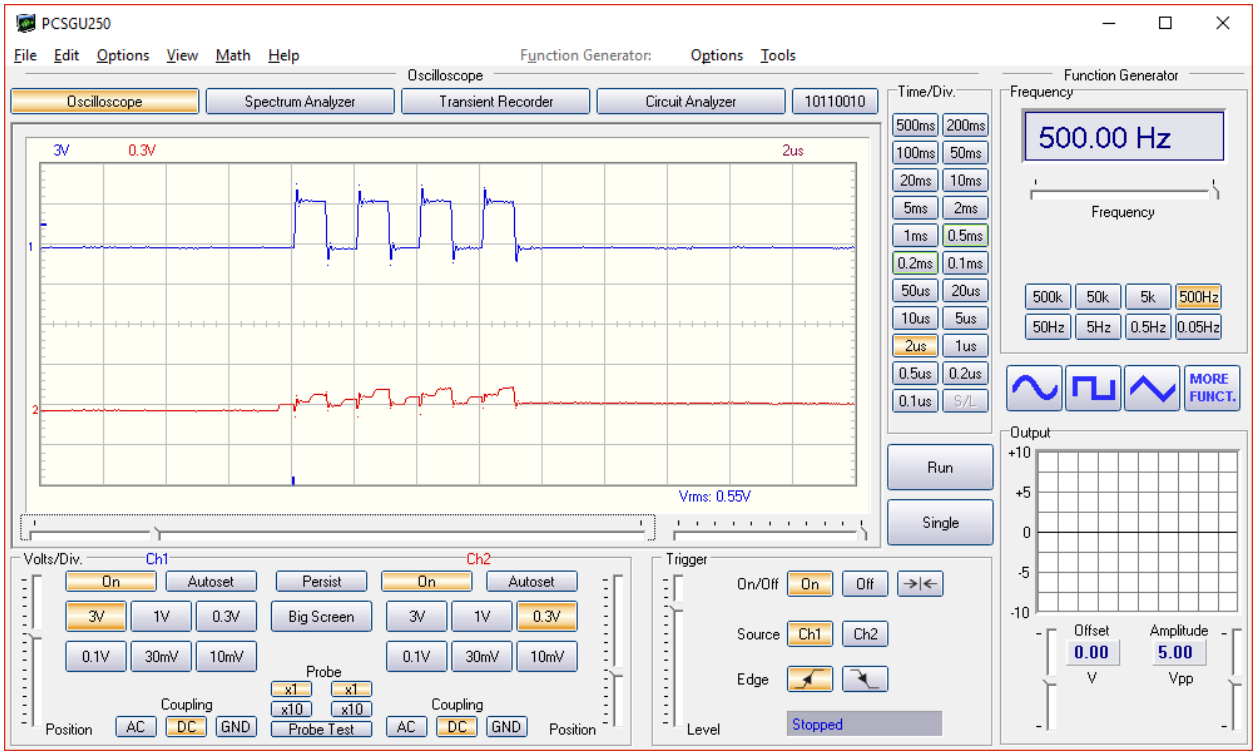


Рис. 7.8. Сигналы на выводах при использовании программы Keil

Я прихожу к выводу, что тот факт, что сигнал не отличается от того, что получен в программе, сделанной в Arduino, заставляет меня предполагать, что впереди ещё очень много работы…