# Лабораторная работа №3

# Аналогово-цифровой преобразователь. Работа с отладчиком

**Цель работы:** научиться работать с АЦП, считывать значения с нескольких каналов АЦП, используя DMA, а также ознакомиться с возможностями отладчика.

**Оборудование и программное обеспечение:** плата UDK32F107V, среда разработки Cube IDE, отладчик ST-Link, беспаечная макетная плата, резистор переменный 10 кОм - 2шт.

## Теоретическая информация

### АЦП

Аналого-цифровые преобразователи (АЦП) — это устройства, предназначенные для преобразования аналоговых сигналов в цифровые. Для такого преобразования необходимо осуществить квантование аналогового сигнала, т. е. мгновенные значения аналогового сигнала ограничить определенными уровнями, называемыми уровнями квантования.

Квантование представляет собой округление аналоговой величины до ближайшего уровня квантования, т. е. максимальная погрешность квантования равна ±0,5h (h — шаг квантования).

К основным характеристикам АЦП относят число разрядов, время преобразования, нелинейность и др. Число разрядов — количество разрядов кода, связанного с аналоговой величиной, которое может вырабатывать АЦП.

В микроконтроллерах STM32 есть мощный модуль АЦП, который имеет действительно хорошие характеристики и интересные особенности:

- разрешение 12 бит
- однократное
- непрерывное
- по триггеру

- по таймеру
- удобное выравнивание битов результата
- генерирование прерываний и сигналов для DMA
- скорость оцифровки — до 0.9 MSPS с программируемым временем захвата и преобразования
- автокалибровка
- режим сканирования входов по списку

Необходимость в этом модуле возникает часто: просто потому, что природа вокруг нас не дискретна, а непрерывна, и всевозможные датчики обычно выдают именно аналоговый сигнал. Особенно это касается звука, но точно так же можно сделать и, к примеру, осциллограф: популярный китайский USB-осциллограф DSO Nano сделан именно на STM32F103.

Принцип оцифровки очень прост: входное напряжение сравнивается с опорными напряжениями V_REF- и V_REF+:

V_REF- нужно подключить к земле

V_REF+ по желанию: либо к питанию процессора (оно плавающее и шумное, поэтому этот вариант годится только для неточных измерений), либо к внешнему источнику опорного напряжения (ИОН)

Также есть возможность программно настроить эти ноги на прямое соединение с землёй и питанием.

Входное напряжение V_In будет измерено относительно V_REF- и V_REF+, и результат преобразования сложен в выходной регистр в такой пропорции, как покказано в таблице 1:

Таблица 1. Преобразование сигнала АЦП

| Напряжение | Результат |
|---|---|
| V_Ref- | |
| V_In | V_In/(V_Ref+ – V_Ref-)*4096 |
| V_Ref+ | 4096 |

К примеру, 1.2 В при питании АЦП от 3.3 В преобразуются в 1490.

# Регистры АЦП в STM32

**SR — регистр статуса**

0 бит: флаг AWD (Analog WatchDog). Входной сигнал пересёк значения регистров LTR или HTR.

1 бит: флаг EOC (End Of Conversion). После окончания преобразования переключается в 1. Сбрасывается вручную или при чтении регистра DR.

4 бит: флаг STRT (Start). Сигнализирует о начале преобразования.

**CR1 — первый регистр настроек**

0..4 биты: значение AWDCH (Analog WatchDog Channel). Задаёт номер канала для слежения вотчдогом.

5 бит: EOCIE (End Of Conversion Interrupt Enable). Включает прерывание по окончанию преобразования.

6 бит: AWDIE (Analog WatchDog Interrupt Enable). Включает прерывание по срабатыванию аналогового вотчдога.

7 бит: JEOCIE.

8 бит: SCAN. Включает режим сканирования каналов по списку в регистрах SQR1, SQR2, SQR3.

9 бит: AWDSGL (Analog WatchDog Single). Задаёт тип срабатывания вотчдога в режиме SCAN: на один канал (1) или на все (0).

10 бит: JAUTO.

11 бит: DISCEN (Discontinious mode Enabled). Включает «рваный» режим работы — АЦП включается по внешнему триггеру.

12 бит: JDISCEN.

13..15 биты: DISCNUM (Discontinious mode Number of channels). Количество каналов для преобразования в «рваном» режиме.

16..19 биты: DUALMOD (Dual Mode selection). Задаёт режим совместной работы двух АЦП.

22 бит: JAWDEN.

23 бит: AWDEN (Analog WatchDog Enabled). Включает аналоговый вотчдог.

**CR2 — второй регистр настроек**

0 бит: ADON (Analog/Digital converter On/off). Включает АЦП

1 бит: CONT (Continious coversion). Включает режим однократного (0) или зацикленных измерений (1).

2 бит: CAL (Calibration). Установка в 1 включает калибровку; после окончания калибровки сбрасывается в 0. Сначала нужно сбросить регистры.

3 бит: RSTCAL (Reset Calibration). Сброс регистров калибровки, точно так же устанавливаем в 1 и ждём сброса.

8 бит: DMA. Включает DMA.

11 бит: ALIGN. Выравнивает данные по правому (0) или левому (1) краю регистра.

12..14 бит: JEXTSEL.

15 бит: JEXTTRIG.

17..19 бит: EXTSEL (External event Select). Назначает номер события для запуска (TIM1 CC1, TIM1 CC2, TIM1 CC3, TIM1 CC4, TIM3 TRGO, TIM4 CC4, EXTI_11, SWSTART).

20 бит: EXTTRIG (External Trigger). Включает запуск преобразования по внешнему триггеру.

21 бит: JSWSTART.

22 бит: SWSTART (Start conversion). Запускает преобразование. После окончания сбрасывается.

23 бит: TSVREFE (Temp sensor and V_REF Enabled). Включает температурный сенсор и внутренний ИОН.

**DR — регистр результата измерения**

**SMPR1, SMPR2 — время преобразования**

Регистр настройки времени преобразования для каждого канала.

**HTR и LTR — пределы вотчдога**

Верхний и нижний пределы для аналогового вотчдога, аналогичны регистру DR.

**SQR1, SQR2, SQR3 — список каналов для сканирования**

Режим SCAN (бит SCAN в регистре CR1)

**DMA**

Direct memory access (DMA), или прямой доступ к памяти (ПДП) используется для быстрой передачи данных между памятью и периферийным устройством, памятью и памятью, или между двумя периферийными устройствами без участия процессора. В микроконтроллере STM32F107 есть два DMA контроллера,которые имеют в общем 12 каналов (7 для DMA1 и 5 для DMA2), каждый из которых специализируется на управлении запросами доступа к памяти от одного или более внешних устройств. DMA контроллеры имеют арбитр для обработки приоритетов между запросами на прямой доступ к памяти. DMA контроллер выполняет прямой обмен с памятью, разделяя системную шину с ядром Cortex-M3. DMA запрос может приостановить доступ ЦПУ к системной шине на несколько тактов шинны, если ЦПУ и DMA работают с одним адресатом (память или внешнее устройство). Матрица шин работает по алгоритму round-robin (циклическое планирование), гарантируя, таким образом, по крайней мере половину пропускной способности системной шины для ЦПУ (при обращении как к памяти, так и к периферии).

После события периферия посылает сигнал запроса на DMA контроллер. DMA контроллер обслуживает этот запрос в зависимости от приоритета канала. Как только DMA контроллер получает доступ к периферии, он посылает ей сигнал уведомления. Внешнее устройство снимает свой запрос, как только оно получает сигнал уведомления от DMA контроллера. Как только снимается сигнал запроса от периферии, DMA контроллер, в свою очередь, снимает сигнал уведомления. Если есть дополнительные запросы, периферия может инициализировать следующую транзакцию.

**JTAG**

**JTAG** (сокращение от англ. *Joint Test Action Group*; произносится «джей-та́г») — название рабочей группы по разработке стандарта IEEE 1149. Позднее это сокращение стало прочно ассоциироваться с разработанным этой группой специализированным аппаратным интерфейсом на базе стандарта IEEE 1149.1. Официальное название стандарта **Standard Test Access Port and Boundary-Scan Architecture**. Интерфейс предназначен для подключения сложных цифровых

микросхем или устройств уровня печатной платы к стандартной аппаратуре тестирования и отладки.

На текущий момент интерфейс стал промышленным стандартом. Практически все сколько-нибудь сложные цифровые микросхемы оснащаются этим интерфейсом для:

- выходного контроля микросхем при производстве
- тестирования собранных печатных плат
- прошивки микросхем с памятью
- отладочных работ при проектировании аппаратуры и программного обеспечения

Метод тестирования, реализованный в стандарте, получил название Boundary Scan (граничное сканирование). Название отражает первоначальную идею процесса: в микросхеме выделяются функциональные блоки, входы которых можно отсоединить от остальной схемы, подать заданные комбинации сигналов и оценить состояние выходов каждого блока. Весь процесс производится специальными командами по интерфейсу JTAG, при этом никакого физического вмешательства не требуется. Разработан стандартный язык управления данным процессом — Boundary Scan Description Language (BSDL).

Стандарт предусматривает возможность подключения большого количества устройств (микросхем) через один физический порт (разъем).

Порт тестирования (*TAP — Test Access Port*) представляет собой четыре или пять выделенных выводов микросхемы: TCK, TMS, TDI, TDO и (опционально) TRST.

JTAG-порт микросхемы и ячейки периферийного сканирования.

Функциональное назначение этих линий:

- **TDI** (*test data input* — «вход тестовых данных») — вход последовательных данных периферийного сканирования. Команды и данные вводятся в микросхему с этого вывода по переднему фронту сигнала TCK;

- **TDO** (*test data output* — «выход тестовых данных») — выход последовательных данных. Команды и данные выводятся из микросхемы с этого вывода по заднему фронту сигнала TCK;
- **TCK** (*test clock* — «тестовое тактирование») — тактирует работу встроенного автомата управления периферийным сканированием. Максимальная частота сканирования периферийных ячеек зависит от используемой аппаратной части и на данный момент ограничена 25…40 МГц
- **TMS** (*test mode select* — «выбор режима тестирования») — обеспечивает переход схемы в/из режима тестирования и переключение между разными режимами тестирования.
- В некоторых случаях к перечисленным сигналам добавляется сигнал **TRST** для инициализации порта тестирования, что необязательно, так как инициализация возможна путём подачи определённой последовательности сигналов на вход TMS.

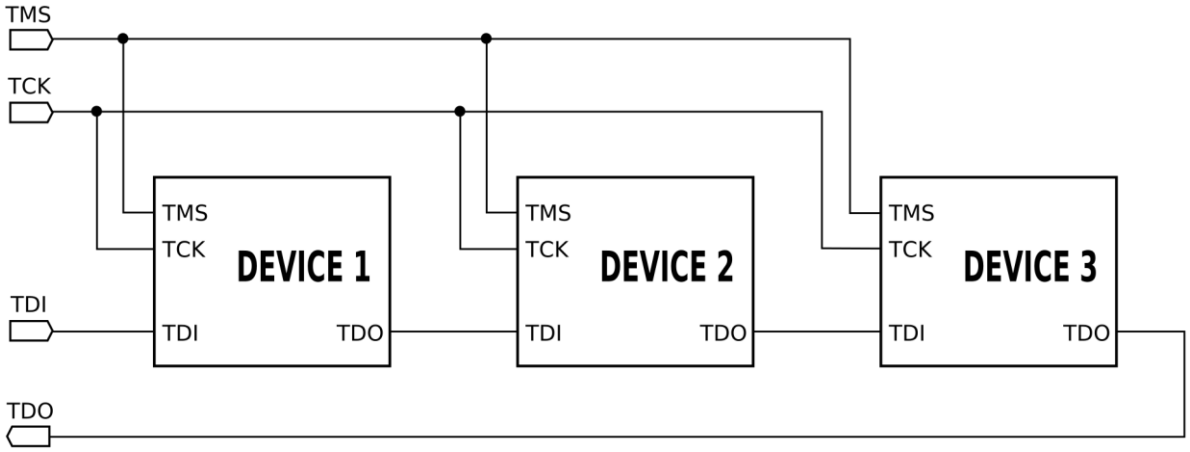


Рис. 1. Структурная схема подключения микроконтроллеров по JTAG

Работа средств обеспечения интерфейса JTAG подчиняется сигналам автомата управления, встроенного в микросхему. Состояния автомата определяются сигналами TDI и TMS порта тестирования. Определённое сочетание сигналов TMS и TCK обеспечивает ввод команды для автомата и её исполнение.

Если на плате установлено несколько устройств, поддерживающих JTAG, они могут быть объединены в общую цепочку. Уникальной особенностью JTAG является возможность программирования не только самого микроконтроллера (или ПЛИС), но и подключённой к его выводам микросхемы флэш-памяти. Причём существует два способа программирования флэш-памяти с использованием JTAG: через загрузчик с последующим обменом данными через память процессора, либо через прямое управление выводами микросхемы.

## Пример программы

Создаём проект в Cube IDE, настраиваем тактирование ориентируясь на предыдущие работы  и настраиваем пины PA3 и PC3 на аналоговый вход. Во вкладке “Parameter Settings” указываем настройки как на рис. 2.  Sampling Time — время (указывается в тактах), которое тратится на заряд внутреннего конденсатора.  АЦП работает так (очень упрощённо): поступающий сигнал сначала заряжает внутренний конденсатор, а потом происходит измерение напряжения в этом конденсаторе. Соответственно у поступающего сигнала должен быть большой ток чтоб быстро зарядить конденсатор. Если аналоговый сигнал слишком слаб, то нужно увеличить Sampling Time.

Устанавливаем параметры АЦП, как показано на рис. 2.

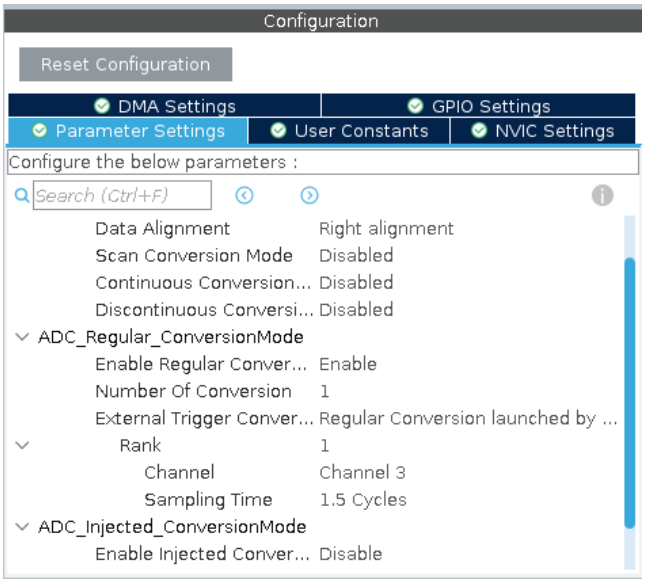


Рис. 2. Параметры АЦП для одноканального режима без DMA

Настраиваем пределитель частоты АЦП “ADC Prescaler” > /6 (Рис. 3). Частота АЦП не должна превышать 14 МГц.

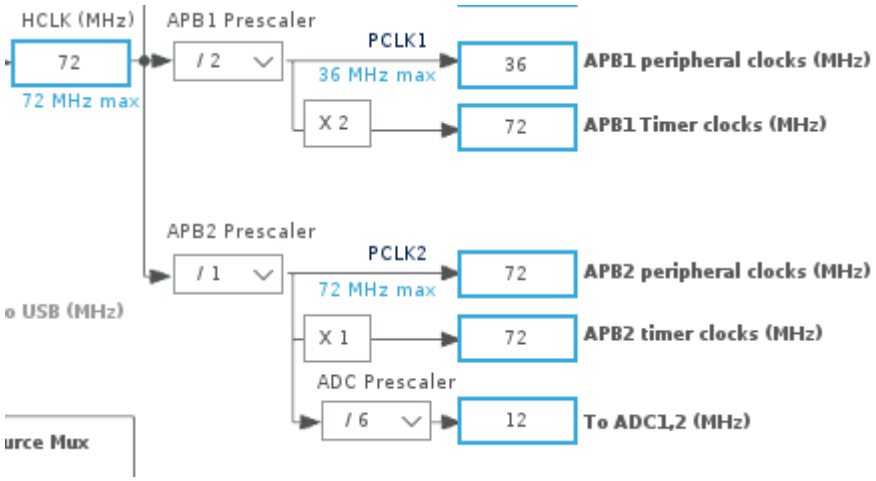


Рис. 3. Установка параметров предделителя частоты АЦП

Далее включаем отладчик в режиме “Trace Asynchronous Sw” (Рис. 4), это необходимо для работы режима SWV Data Console, через который происходит вывод отладочной информации, что отдалённо напоминает UART, но в отличие от последнего, SWV работает только на отображение информации и его нельзя использовать для отправки данных в микроконтроллер.

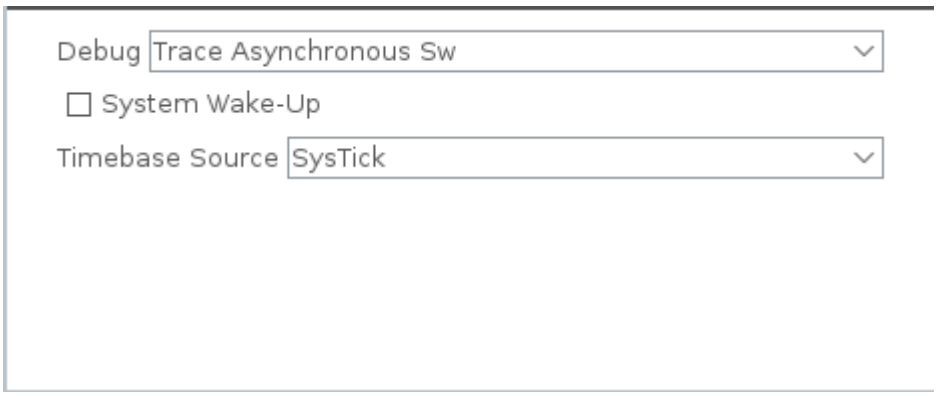


Рис. 4. Выбор режима отладки

Сохраняем настройки, нажав Ctrl+S и приступаем к разработке. Попробуем запустить АЦП в одноканальном режиме без режима DMA. Подключаем потенциометр, как показано на рис. 5.

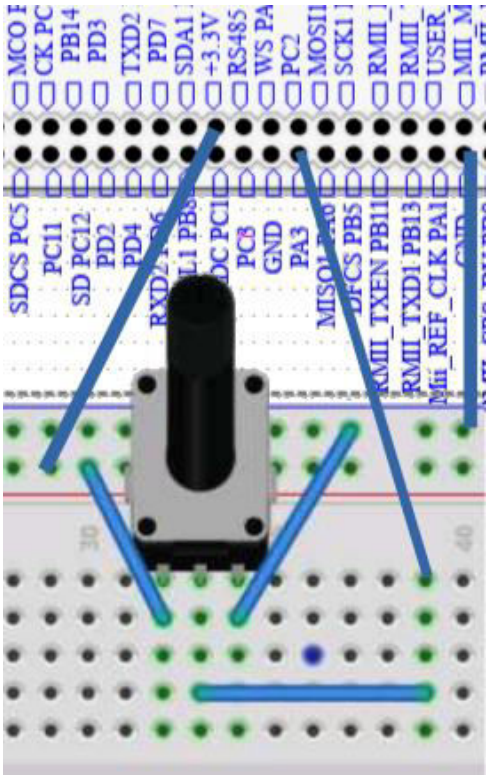


Рис. 5. Подключение потенциометра

```
/* USER CODE BEGIN PV */
uint16_t AdcValue0; //переменная для хранения данных с АЦП
/* USER CODE END PV */
/* USER CODE BEGIN 0 */
int _write(int32_t file, uint8_t *ptr, int16_t len) { //код для работы режима SWV
int i=0;
for(i=0 ; i<len ; i++) //собираем принятые данные в строку
ITM_SendChar((*ptr++)); //отправляем данные в порт SWV
return len;
}
/* USER CODE END 0 */
  /* USER CODE BEGIN 2 */
  HAL_ADCEx_Calibration_Start(&hadc1); //запускаем калибровку АЦП
  /* USER CODE END 2 */
  /* USER CODE BEGIN WHILE */
  while (1)  {
```

HAL_ADC_Start(&hadc1); // запускаем преобразование сигнала АЦП

HAL_ADC_PollForConversion(&hadc1, 100); // ожидаем окончания преобразования

AdcValue0 = HAL_ADC_GetValue(&hadc1); // читаем полученное значение в переменную adc

HAL_ADC_Stop(&hadc1); // останавливаем АЦП (не обязательно)

printf("AdcValue0: %u\n", AdcValue0); // %u означает, что данные нужно принять в формате unsigned int, а \n переносит каждое принятое значение на строку ниже

HAL_Delay(1000);

/* USER CODE END WHILE */

После написания программы компилируем её и если всё прошло успешно, запускаем отладку.

Во вкладке “Live Expressions” смотрим значение переменной AdcValue0 (рис. 6). Значение переменной при разрядности АЦП 12 бит, может иметь значение в диапазоне 0...4095.

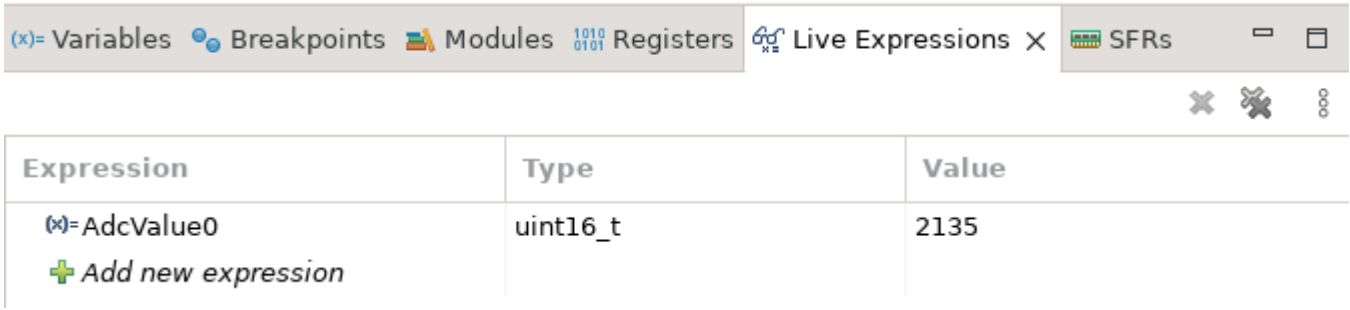


Рис. 6. Просмотр значения переменной в отладчике

Также попробуем глянуть, что пишет отладчик в порт SWO, но сначала надо включить его в меню: “Run>Debug configurations”. Debug Probe выбираем OpenOCD. Serial wire viewer (SWV) настраиваем на текущую тактовую частоту микроконтроллера - 72МГц.

Далее включаем “Window>Show view>SWV>SWV Data Console”, после включения этой опции появится окно SWV Data Console (Рис. 7).

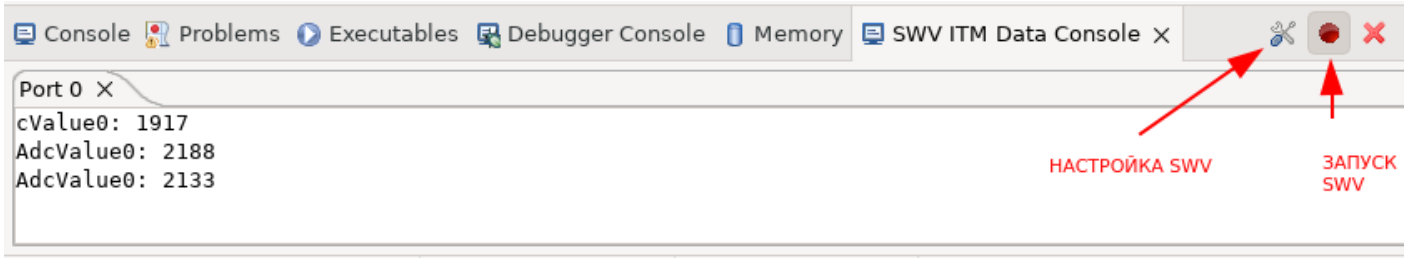


Рис. 7. Окно просмотра сообщений отладчика SWV

По умолчанию в параметрах SWV не выбран никакой порт, поэтому никакие сообщения изначально в это окно выводиться не будут. В параметрах SWV нужно активировать “Port 0”, как показано на рис. 8.

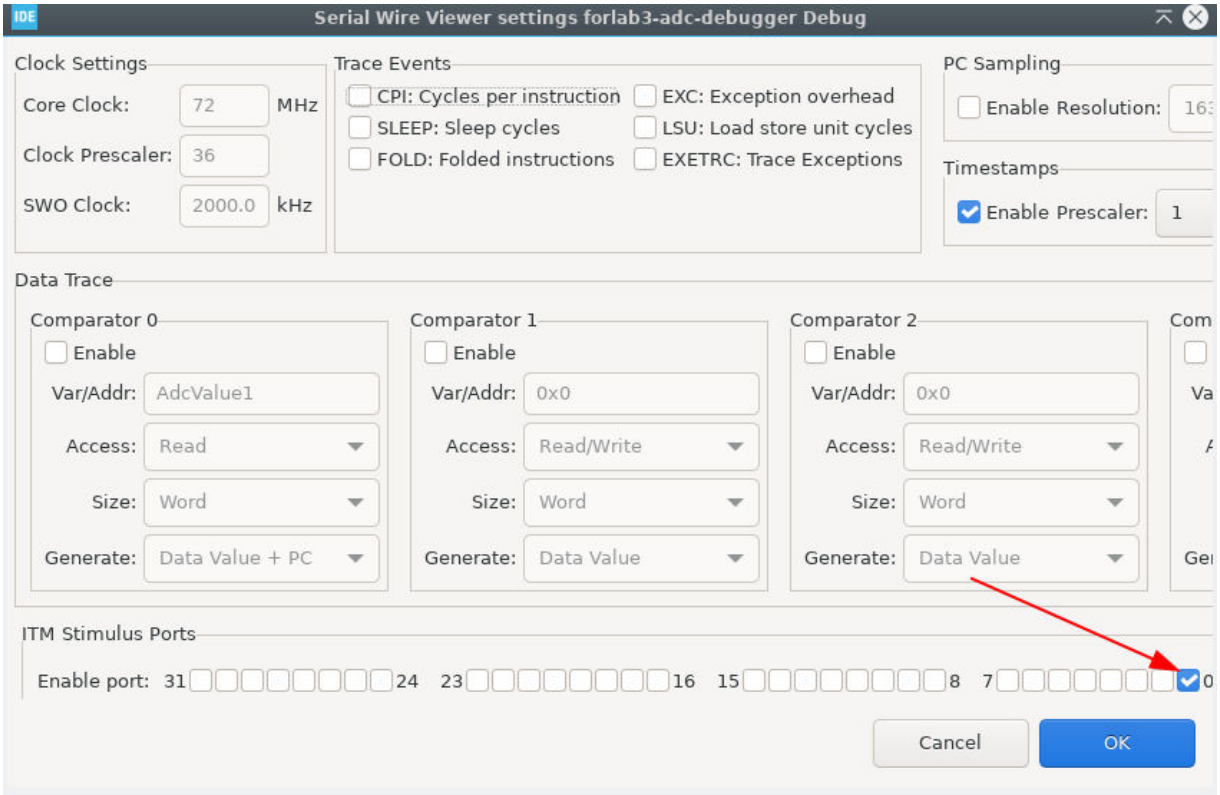


Рис. 8. Окно параметров SWV

Следующим шагом будет рассмотрено считывание значений АЦП с двух каналов и с применением DMA. Подключаем переменные резисторы, ка показано на рис 9 и настраиваем АЦП, как показано на рис. 10.

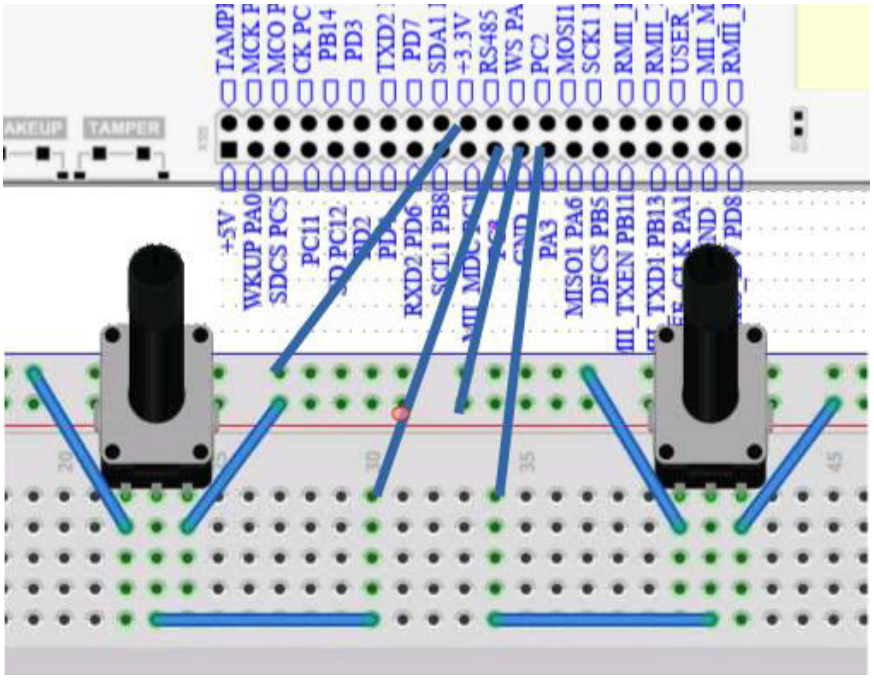

Рис. 9. Подключение переменных резисторов

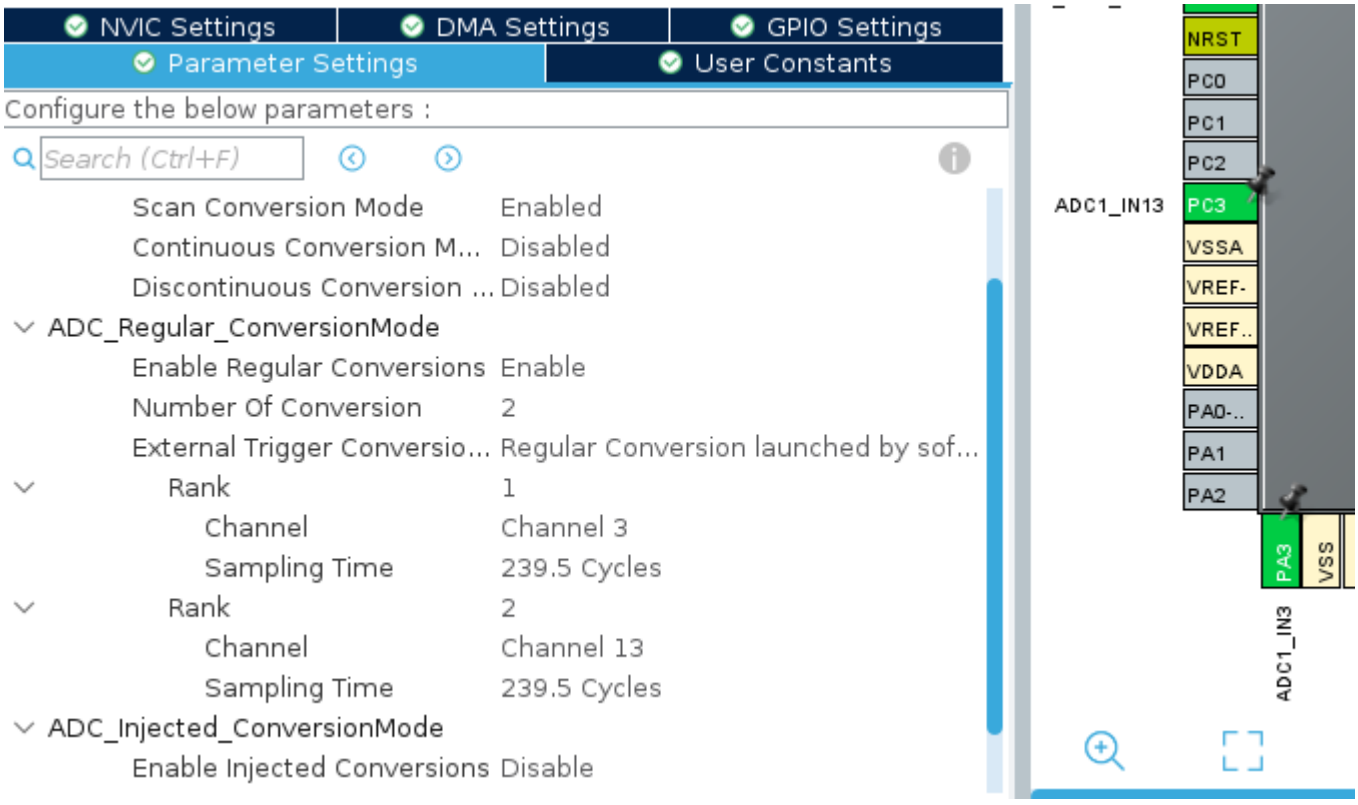


Рис. 10. Настройка АЦП

Также активируем DMA для АЦП (Рис. 11), что позволит организовать прямую передачу данных от АЦП сразу в память контроллера, без использования

ресурсов процессора, это позволит ускорить выполнение программы, что будет сильно актуально в более серьёзных проектах.

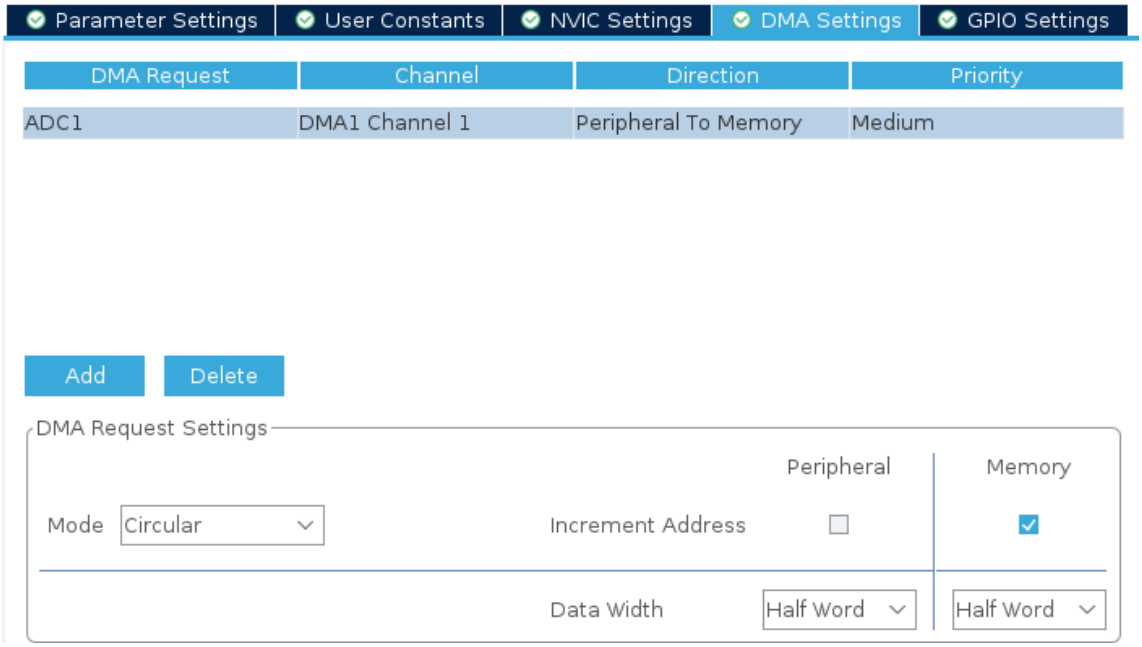


Рис. 11. Настройка DMA

```
/* USER CODE BEGIN Includes */
#include "stdio.h" //библиотека, в нашем случае необходимая для работы функций printf и sprintf
/* USER CODE END Includes */
/* USER CODE BEGIN PD */
#define ADC_CHANNELS_NUM 2 //количество каналов АЦП
#define FILTER_SMA_ORDER 15 //число измерений
/* USER CODE END PD */
/* USER CODE BEGIN PV */
uint16_t Filter_Buffer[FILTER_SMA_ORDER] = {0,};
uint16_t AdcValue0;
uint16_t AdcValue1;
float AdcVoltage0;
float AdcVoltage1;
uint16_t adcData[ADC_CHANNELS_NUM];
```

```
float adcVoltage[ADC_CHANNELS_NUM];
/* USER CODE END PV */
/* USER CODE BEGIN 0 */
int _write(int32_t file, uint8_t *ptr, int16_t len) {
int i=0;
for(i=0 ; i<len ; i++)
ITM_SendChar((*ptr++));
return len;
}
//Функция uint16_t Filter_SMA производит вычисление среднего
//арифметического, что позволяет отфильтровать шумы АЦП.

uint16_t Filter_SMA(uint16_t For_Filtered) {
	Filter_Buffer[FILTER_SMA_ORDER - 1] = For_Filtered;
	uint32_t Output = 0;
	for(uint8_t i = 0; i < FILTER_SMA_ORDER; i++)
	{
		Output += Filter_Buffer[i];
	}
	Output /= FILTER_SMA_ORDER;
	for(uint8_t i = 0; i < FILTER_SMA_ORDER; i++)
	Filter_Buffer[i] = Filter_Buffer[i+1];
	return (uint16_t) Output;
}
/* USER CODE END 0 */

/* USER CODE BEGIN 2 */
  HAL_ADCEx_Calibration_Start(&hadc1);
  HAL_ADC_Start_DMA(&hadc1, (uint32_t*)adcData, ADC_CHANNELS_NUM);
  HAL_ADC_Start_IT(&hadc1);
```

```
  HAL_TIM_Base_Start_IT(&htim4);
  /* USER CODE END 2 */
 while (1)
  {
          AdcValue0 = Filter_SMA(adcData[0]);
          AdcValue1 = adcData[1];
          AdcVoltage0 = adcVoltage[0];
          AdcVoltage1 = adcVoltage[1];
          printf("AdcVoltage0: %f\n", (float)AdcVoltage0);
          printf("AdcVoltage1: %f\n", (float)AdcVoltage1);
          HAL_Delay(100);
    /* USER CODE END WHILE */
/* USER CODE BEGIN 4 */
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim) {
    HAL_ADC_Start_IT(&hadc1);
}
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc) {
  if(hadc->Instance == ADC1)  {
    for (uint8_t i = 0; i < ADC_CHANNELS_NUM; i++) {
      adcVoltage[i] = adcData[i] * 3.3 / 4095;
    }
  }
}
/* USER CODE END 4 */
```

Следует заметить, что в компиляторе по умолчанию отключен float для функций printf, sprintf и scanf, поэтому необходимо эту опцию включить в “Project>Properties” (рис. 12).

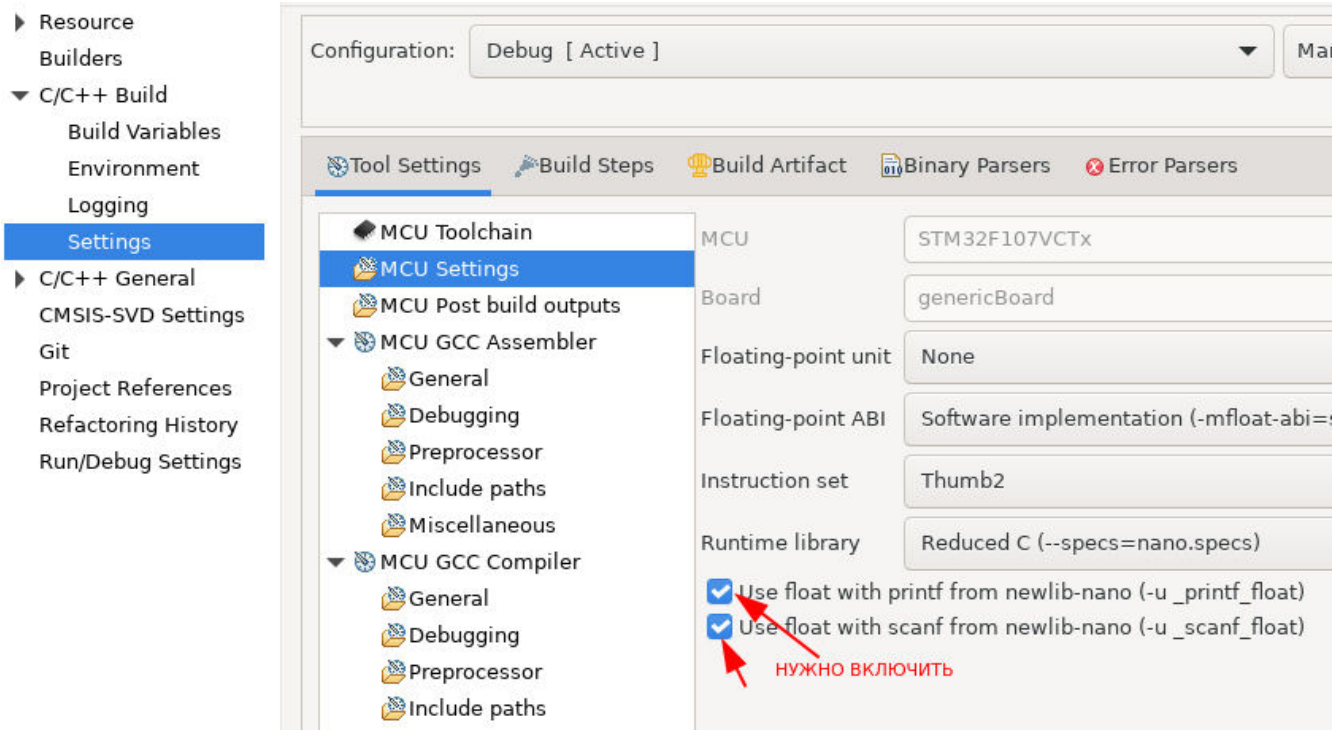


Рис. 12. Включение float для printf

Пробуем компилировать и если всё хорошо, то запускаем отладку. В сообщениях отладчика будет видно значение напряжения на обеих портах АЦП, а в переменных значение АЦП (рис 13).

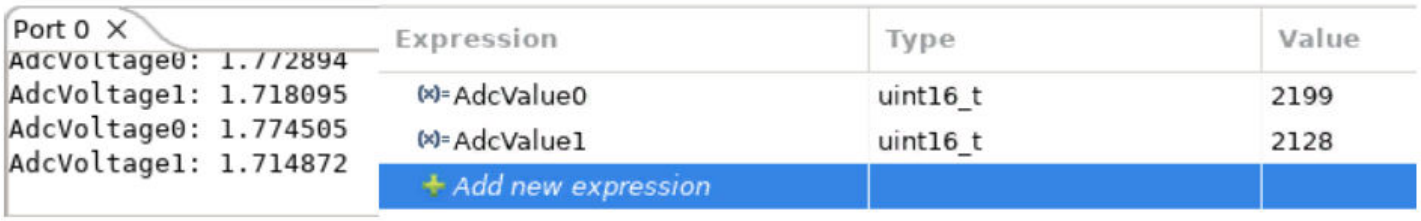


Рис. 13. Вывод значений АЦП в отладчик

Также значение АЦП можно увидеть при просмотре регистров. В АЦП значение хранится в регистре DR (рис. 14).

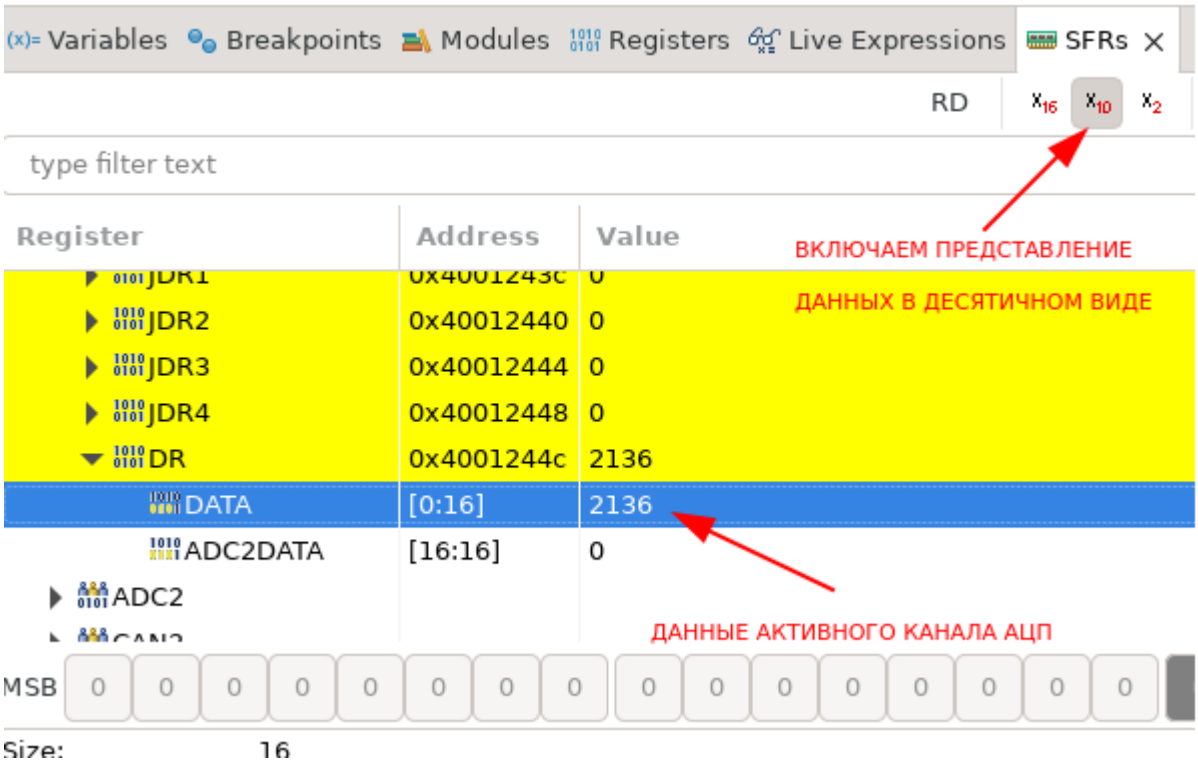


Рис. 14. Просмотр значений регистров АЦП

## Ход работы

1. На основе кода примера приложения создать свой проект в среде разработки и проверить его работоспособность.

2. Ознакомиться с работой используемых функций, просмотрев исходный код, а также комментарии в исходных файлах библиотеки и в режиме отладки.

3. Запрограммировать плату и продемонстрировать работу программы.