**Лабораторная работа №4**

**Работа с АЦП**

1. ***Теоритическая часть***

Микроконтроллер STM32F4хх имеет на борту 3 12-ти разрядных АЦП. Каждое АЦП может быть подключено к любому из 24 аналоговых входов. Более того, каждое из АЦП может сканировать эти входы, снимая с них данные в заданном пользователем порядке.

По окончании преобразования АЦП может выдать прерывание. Вообще АЦП может выдать одно из трёх прерываний: Об окончании преобразования обычного (регулярного) канала, об окончании преобразования по инжекторному каналу и событие по Watchdog.

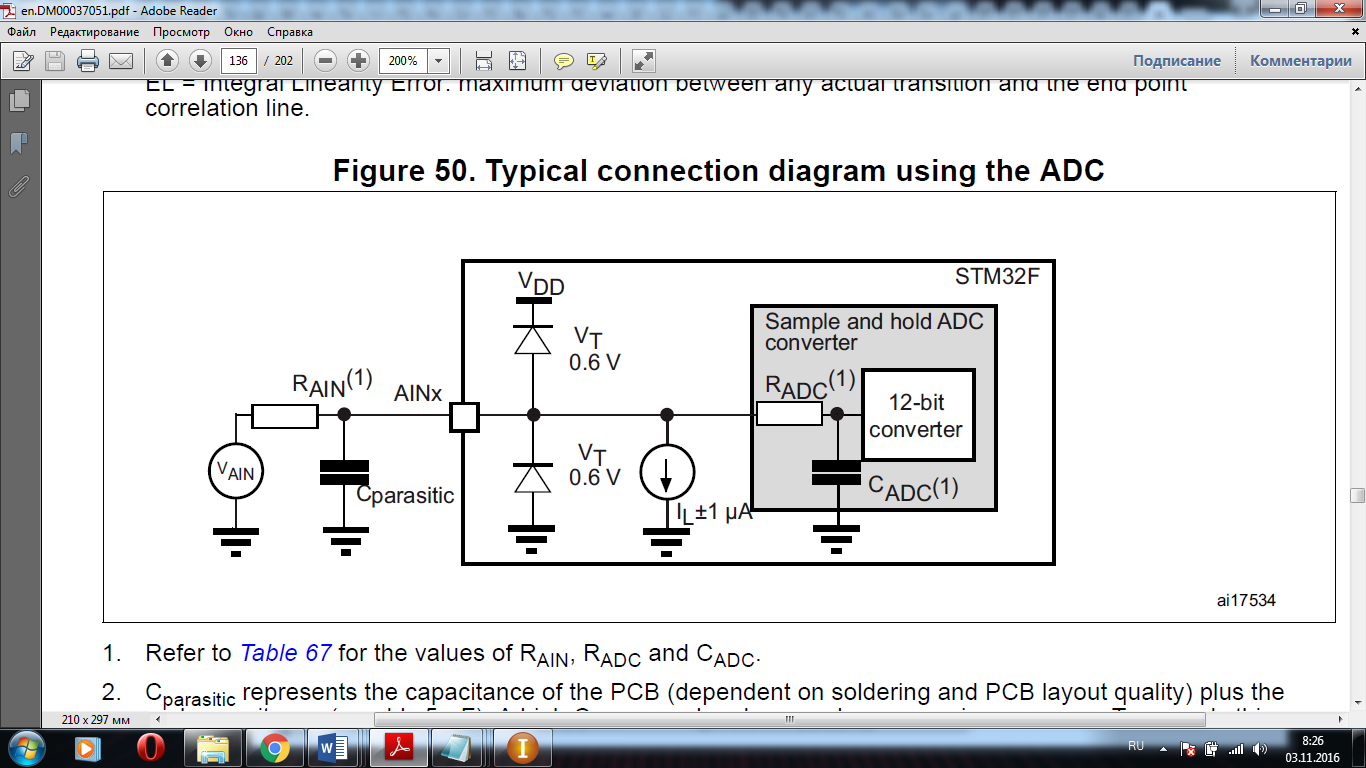
В режиме сканирования прерывание об окончании преобразования выдаётся только по завершении всего сканирования. И при использовании регулярных каналов, в которых данные записываются всегда в один и тот же регистр, вы будете получать результаты только последнего преобразования.

Что бы этого не происходило, в микроконтроллере предусмотрено наличие так называемых инжекторных каналом, имеющих в своём наличии 4 разных регистра для записи данных. Т.е. если вам надо сканировать не более 4-х каналов, то результаты преобразований вы не потеряете. Т.к. каждый канал будет писать данные в свой регистр.

Для параллельного снятия данных сразу по нескольким каналам, предусмотрена возможность одновременного запуска нескольких АЦП. Данный режим получил название Dual Mode.

АЦП имеет хорошую разрешающую способность 12 бит и очень высокую скорость преобразования, составляющую 2,4 МСемпла в одиночном режиме и 7,2 МСемпла – в тройном режиме. Как и в большинстве современных МК, присутствует генератор опорного напряжения. Гибкая система настроек встроенного аналогового мультиплексора позволяет задавать любые последовательности преобразования аналоговых каналов (за исключением одновременного преобразования одного канала на нескольких АЦП). Настройки АЦП позволяют производить однократные и циклические измерения. Для проведения преобразования на максимальных скоростях необходимо соблюдать диапазон напряжения питания 2,4…3,6 В. При снижении напряжения до 1,8 (1,7) В, скорость преобразования снижается до 1,2 МСемплов. Для контроля внутренней температуры микроконтроллера встроен температурный датчик. На его выходе формируется напряжение в зависимости от окружающей температуры. Выход датчика через мультиплексор подключается к АЦП. Используя температурный датчик, можно измерять температуру от -40 до 125°C с точностью ±1,5°C.





Типовая схема подключения с использованием АЦП

1. ***Практическая часть***

***2.1 Настройка библиотек***

В проекте нужны следующие компоненты (как включаются компоненты смотрите в лабораторной работе 1):

* M4 CMSIS Core;
* CMSIS BOOT;
* RCC;
* GPIO;
* ADC (модуль отвечает за АЦП)
* TIM ( нужен для настройки таймера)
* MISC (нужен для настройки и обработки прерываний);

В файле main.c подключаем заголовочные файлы:

#include "stm32f4xx.h"

#include "stm32f4xx\_gpio.h"

#include "stm32f4xx\_rcc.h"

#include "stm32f4xx\_adc.h"

#include "stm32f4xx\_tim.h"

#include "misc.h"

#include "config.h" //подключаем созданный файл конфигурации

* 1. ***Инициализация аппаратных средств***

void Board\_Init(void

{

GPIO\_LEDS\_Init();

ADC1\_Init();

Timer3\_Init();

Timer4\_Init();

}

* 1. ***Настройка портов ввода вывода для работы со светодиодами и для АЦП***

Для работы светодиодов необходимо произвести инициализацию GPIO:

* инициализировать порты ввода/вывода для светодиодов (смотрите лабораторную работу №1);
* инициализация портов ввода/вывода для АЦП.

Включаем тактирование портов ввода/вывода для АЦП (Для ADC1 на РА1 с использованием IN1):

RCC\_AHB1PeriphClockCmd(RCC\_AHB1Periph\_GPIOA, ENABLE);

* 1. ***Инициализация АЦП***

GPIO\_ADC1\_Init();

Включаем тактирование АЦП:

RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_ADC1, ENABLE);

**Дополнительная структура настройки АЦП:**

ADC\_CommonInitTypeDef ADC\_CommonInitStruct;

ADC\_CommonStructInit(&ADC\_CommonInitStruct);

ADC\_CommonInitStruct.ADC\_Mode = ADC\_Mode\_Independent;

ADC\_CommonInitStruct.ADC\_Prescaler = ADC\_Prescaler\_Div2;

ADC\_CommonInitStruct.ADC\_DMAAccessMode=

ADC\_DMAAccessMode\_Disabled;

ADC\_CommonInitStruct.ADC\_TwoSamplingDelay= ADC\_TwoSamplingDelay\_5Cycles;

Инициализируем АЦП периферийных устройств в соответствии с заданными параметрами в ADC\_CommonInitStruct.

ADC\_CommonInit(&ADC\_CommonInitStruct);

Разберем некоторые параметры:

* ADC\_Mode – настраивает АЦП для работы в автономном режиме или нескольких. Этот параметр может принимать значения:
* ADC\_Mode\_Independent (АЦП работает независимо от других)
* ADC\_DualMode\_RegSimult\_InjecSimult
* ADC\_DualMode\_RegSimult\_AlterTrig
* ADC\_DualMode\_InjecSimult
* ADC\_DualMode\_RegSimult
* ADC\_DualMode\_Interl
* ADC\_DualMode\_AlterTrig
* ADC\_TripleMode\_RegSimult\_InjecSimult
* ADC\_TripleMode\_RegSimult\_AlterTrig
* ADC\_TripleMode\_InjecSimult
* ADC\_TripleMode\_RegSimult
* ADC\_TripleMode\_Interl
* ADC\_TripleMode\_AlterTrig.
* ADC\_Prescaler – указывает предделитель, используемый для настройки частоты тактового сигнала на АЦП. Частота является общим для всех АЦП. Параметр может принимать значения:
* ADC\_Prescaler\_Div2 (Предделитель устанавливаем на 2)
* ADC\_Prescaler\_Div4
* ADC\_Prescaler\_Div6
* ADC\_Prescaler\_Div8
* ADC\_TwoSamplingDelay – настраивает задержку между 2 сэмплами (фазами) выборки. Этот параметр может принимать значения от 5 до 20 тактов
* ADC\_TwoSamplingDelay\_5Cycles
* ADC\_TwoSamplingDelay\_6Cycles
* ADC\_TwoSamplingDelay\_7Cycles
* ADC\_TwoSamplingDelay\_8Cycles
* и т. д …
* ADC\_DMAAccessMode – настраивает режим прямого доступа к памяти для режима нескольких АЦП. Может принимать значения:
* ADC\_DMAAccessMode\_Disabled
* ADC\_DMAAccessMode\_1
* ADC\_DMAAccessMode\_2
* ADC\_DMAAccessMode\_3

**Основная структура настройки АЦП:**

ADC\_InitTypeDef ADC\_InitStruct;

ADC\_StructInit(&ADC\_InitStruct);

ADC\_InitStruct.ADC\_Resolution=ADC\_Resolution\_12b;

ADC\_InitStruct.ADC\_ScanConvMode=DISABLE;

ADC\_InitStruct.ADC\_ContinuousConvMode=DISABLE; ADC\_InitStruct.ADC\_ExternalTrigConvEdge=ADC\_ExternalTrigConvEdge\_None;

ADC\_InitStruct.ADC\_ExternalTrigConv=ADC\_ExternalTrigConv\_T1\_CC1;

ADC\_InitStruct.ADC\_DataAlign = ADC\_DataAlign\_Right;

ADC\_InitStruct.ADC\_NbrOfConversion = 1;

ADC\_Init(ADC1, &ADC\_InitStruct);

ADC\_EOCOnEachRegularChannelCmd(ADC1, ENABLE);

ADC\_Cmd(ADC1, ENABLE);

Рассмотри некоторые функции:

* ADC\_EOCOnEachRegularChannelCmd(ADC\_TypeDef\*ADCx,FunctionalState NewState) -- включает или выключает End–of–Conversion (EOC – конечное преобразование) на каждом очередном преобразовании канала.
* ADCx : где х может быть 1, 2 или 3 для выбора АЦП периферийного устройства.
* NewState : новое состояние выбранного флага ADC EOC растет Этот параметр может быть: ENABLE (включить) или DISABLE (отключить).

Рассмотрим некоторые параметры:

* ADC\_Resolution -- настраивает двойной режим разрешения АЦП (Выбираем число значащих разрядов преобразования) Может принимать значения:
* ADC\_Resolution\_12b
* ADC\_Resolution\_10b
* ADC\_Resolution\_8b
* ADC\_Resolution\_6b
* ADC\_ScanConvMode -- указывает, выполняется ли преобразование в Scan (многоканальности) или Single (один канал). Этот параметр может быть установлен для ENABLE (включения) или DISABLE (отключения).

То есть этот параметр определяет будет ли АЦП сканировать несколько каналов. Если этот режим включен, то АЦП будет последовательно оцифровывать данные с заданных каналов в заданной последовательности. И каналы, и последовательность легко можно задать. Но возникает небольшая проблема со снятием данных.

* ADC\_ContinuousConvMode -- указывает, выполняется ли преобразование в непрерывном или одиночном режиме. Этот параметр может быть установлен для ENABLE (включения) или DISABLE (отключения).

Т.е этот режим, если включен, запускает следующее преобразование сразу по окончании предыдущего. Так можно добиться максимальной скорости работы АЦП. В нашем случае это не надо, и данная функция отключена.

* ADC\_ExternalTrigConvEdge -- настраивает запуск преобразования по какому-либо событию, например, переполнению таймера. Этот параметр может быть значением:
* ADC\_ExternalTrigConvEdge\_None (без внешнего триггера)
* ADC\_ExternalTrigConvEdge\_Rising
* ADC\_ExternalTrigConvEdge\_Falling
* ADC\_ExternalTrigConvEdge\_RisingFalling
* ADC\_DataAlign -- указываем, в какую сторону данные выравниваются (регистр 16-ти битный, а значащих данных всего 12 или того меньше бит. Отсюда вопрос: в какую сторону выровнять данные.

Пример:

С АЦП пришло значение 12 бит. Если выравнивание вправо, то все просто:

0b0000xxxxxxxxxxxx, x - это 0 или 1. Если выравнивание влево:

0bxxxxxxxxxxxx0000)

Этот параметр может быть значением:

* ADC\_DataAlign\_Right (выравнивание вправо)
* ADC\_DataAlign\_Left (выравнивание влево)
* ADC\_ExternalTrigConv -- устанавливает какие именно события запустят преобразования. Т.к. триггер отключен, то эта функция не используется.

Этот параметр может быть значением:

* ADC\_ExternalTrigConv\_T1\_CC1
* ADC\_ExternalTrigConv\_T1\_CC2
* ADC\_ExternalTrigConv\_T1\_CC3
* ADC\_ExternalTrigConv\_T2\_CC2
* ADC\_ExternalTrigConv\_T2\_CC3
* ADC\_ExternalTrigConv\_T2\_CC4
* ADC\_ExternalTrigConv\_T2\_TRGO
* ADC\_ExternalTrigConv\_T3\_CC1
* ADC\_ExternalTrigConv\_T3\_TRGO
* ADC\_ExternalTrigConv\_T4\_CC4
* ADC\_ExternalTrigConv\_T5\_CC1
* ADC\_ExternalTrigConv\_T5\_CC2
* ADC\_ExternalTrigConv\_T5\_CC3
* ADC\_ExternalTrigConv\_T8\_CC1
* ADC\_ExternalTrigConv\_T8\_TRGO
* ADC\_ExternalTrigConv\_Ext\_IT11
* ADC\_NbrOfConversion -- указывает число переходов АЦП, который будет проводиться с использованием секвенсеров (устройство для записи в реальном времени) для регулярной группы каналов. Этот параметр должен находиться в диапазоне от 1 до 16.

То есть число каналов, которые будет сканировать МК. Сюда записывается требуемое значение, а ниже, если это число больше 1 и ADC\_ ScanConvMode=ENABLE, описывается какие каналы и в какой последовательности они будут сканироваться

* 1. ***Настройка таймеров, приоритета и источника прерываний с помощью NVIC, запуск прерываний***
* Для управления прерываниями производим инициализацию NVIC (подробнее смотрите в лабораторной работе №2)
* Инициализировать таймеры (смотреть лабораторную работу №3)
  1. ***Настраиваем конкретный канал***

В нашем случае это всего один канал, потому настройка будет выглядеть так:

ADC\_RegularChannelConfig(ADC1, channel,1,ADC\_SampleTime\_480Cycles);

ADC\_SoftwareStartConv(ADC1);

while(ADC\_GetFlagStatus(ADC1, ADC\_FLAG\_EOC) == RESET);

return ADC\_GetConversionValue(ADC1);

Разберем несколько функций:

1. ADC\_RegularChannelConfig(ADC\_TypeDef\*ADCx,ADC\_Channel,Rank,ADC\_SampleTime) -- настраивает для выбранного канала АЦП регулярного его соответствующего ранга в секвенсор и его время выборки.

Параметры:

* ADCx: где х может быть 1, 2 или 3 ,Номер настраиваемого АЦП.
* ADC\_Channel: задает снимаемый канал (с Channel1- Channel18)
* Rank: ранг в обычной группе секвенсор. Этот параметр должен быть в пределах от 1 до 16. Показывает в каком порядке этот канал будет оцифровываться. В нашем случае канал один, потому и rank=1.
* ADC\_SampleTime: Образец значение времени, чтобы установить для выбранного канала. задаёт за какое время будет произведена оцифровка. Чем медленнее, тем точнее. Этот параметр может быть одним из следующих значений:
* ADC\_SampleTime\_3Cycles
* ADC\_SampleTime\_15Cycles
* ADC\_SampleTime\_28Cycles
* ADC\_SampleTime\_56Cycles
* ADC\_SampleTime\_84Cycles
* ADC\_SampleTime\_112Cycles
* ADC\_SampleTime\_144Cycles
* ADC\_SampleTime\_480Cycles

1. ADC\_SoftwareStartConv(ADC\_TypeDef \* ADCx ) -- включает выбранное программное обеспечение ADC, запускает преобразование каналов
2. ADC\_GetFlagStatus(ADC\_TypeDef\* ADCx,ADC\_FLAG) -- проверяет, является ли указанный флаг АЦП установлен или не.

Параметры:

* ADCx,: где х может быть 1, 2 или 3 для выбора АЦП периферийного устройства.
* ADC\_FLAG ,: задает флаг для проверки. Этот параметр может быть одним из следующих значений:
* ADC\_FLAG\_AWD: флаг Аналоговый сторожевой
* ADC\_FLAG\_EOC: Конец флага преобразования
* ADC\_FLAG\_JEOC: Конец флага преобразования введенного группы
* ADC\_FLAG\_JSTRT: Начало флага преобразования введенного группы
* ADC\_FLAG\_STRT: Начало регулярного флага преобразования группы
* ADC\_FLAG\_OVR: перерасход флаг

1. ADC\_GetConversionValue(ADC\_TypeDef\*ADCx) -- возвращает последний ADCx результат преобразования данных для обычного канала.

То есть мы ожидаем, когда преобразование закончится и считываем результат

while(ADC\_GetFlagStatus(ADC1, ADC\_FLAG\_EOC) == RESET);

return ADC\_GetConversionValue(ADC1);