AVR Урок 22. Изучаем АЦП. часть 2 |

Posted on

Урок 22

Часть 2

Сегодня мы продолжаем изучать очень интересную технологию, а для микроконтроллера – периферию – **аналго-цифровой преобразователь** или как его называют **АЦП**. В **прошлой части** нашего занятия мы познакомились, что такое вообще АЦП, также познакомились, как он организован в контроллере AVR, а также создали новый проект и настроили его.

Дальнейшая задача – реализация АЦП в нашем проекта.

Ну и чтобы нам данную задачу выполнить, нам нужны будут определённые функции для обращения к АЦП контроллера.

Для этого зайдём в файл **adc.c** и создадим функцию инициализацию нашего АЦП

```
#include "adc.h"

//————————————————-

void ADC_Init(void)

{

}
```

Также создадим на данную функцию прототип в хедер-файле adc.h для видимости её из внешних модулей, а также заодно и посмотрим всё содержимое данного файла

```
#ifndef ADC_H_

#define ADC_H_
```

```
#include "main.h"

void ADC_Init(void);

#endif /* ADC_H_ */
```

Продолжим теперь заполнять кодом тело данной функции. Так как в **прошлой части** мы хорошенечко ознакомились с регистрами, нам это особого труда не составит.

Начнем с управляющего регистра

```
void ADC_Init(void)

{

 ADCSRA |= (1<<ADEN) // Разрешение использования АЦП

 |(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0);//Делитель 128 = 64 кГц

}
```

Это не две строки, а одна, так писать в студии можно и даже нужно, так как код становится понятнее. А одна, потому что нет символа конца строки – точки с запятой.

Здесь мы включили бит **ADEN**, тем самым включили вообще модуль АЦП, а также установили делитель на 128, тем самым, помня то, что частота тактирования у нас 8 МГц и разделив её значение на 128, мы получили работу АЦП на частоте 64 кГц, что вполне нормально и надёжно, до 200 граничных далеко. Как видим, ничего сложного в инициализации регистра нет.

Также ещё в данной функции нам необходимо выбрать канал, к которому мы будем подключать измеряемое напряжение. У нас судя по схеме канал 0, поэтому соответствующий MUX мы и включим. А соответствующий MUX – это все нули в данных битах, поэтому ничего-то и включать не надо. Но мы ещё помним, что в регистре **ADMUX** у нас помимо всего прочего есть и управляющие биты, а именно биты REFS1 и REFS0, с помощью которых мы установим в качестве источника опорного напряжения внутренний источник на 2,56 вольта, а ADLAR мы не используем

```
 ADCSRA |= (1<<ADEN) // Разрешение использования АЦП

 |(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0);//Делитель 128 = 64 кГц

 ADMUX |= (1<<REFS1)|(1<<REFS0); //Внутренний Источник ОН 2,56в, вход ADC0
```

}

Ну вот, в принципе, и вся инициализация.

Вызовем эту функцию в главном модуле программы в функции main() где-нибудь вот тут

LCD_ini(); //Инициализируем дисплей

ADC_Init(); //Инициализируем АЦП

clearlcd(); //Очистим дисплей

Ну и также нам нужна будет в модуле adc.c ещё одна функция, которая будет инициализировать непосредственно начало процесса аналого-цифрового преобразования в нашем ADC

unsigned int ADC_convert (void)

{

}

Само собой нужен будет в хедер-файле прототип на неё

void ADC_Init(void);

unsigned int ADC_convert (void);

Данная функция нам вернёт значение из регистровой пары **ADC**, которая и будет содержать величину нашего электрического сигнала в единицах, выражающих отношение измеряемого сигнала к опорному и умноженных на количество возможных отрезков, которых у нас 1023, ну или 1024. Насчёт этого ходят много слухов, но в технической документации на контроллер в расчетной формуле содержится именно 1024. Но это нам не так важно.

Включим преобразование с помощью бита **ADSC**

unsigned int ADC_convert (void)

{

ADCSRA |= (1<<ADSC); //Начинаем преобразование

Теперь нам надо как-то отследить тот момент, когда данное преобразование закончится. А делается это

достаточно легко с помощью мониторинга того же бита ADSC, который по окончании процесса преобразования сам сбрасывается в **0** (When the conversion is complete, it returns to zero). Отслеживается данный бит с помощью условного цикла

ADCSRA |= (1<<ADSC); //Начинаем преобразование

while((ADCSRA & (1<<ADSC))); //проверим закончилось ли аналого-цифровое преобразование

Ну и по окончании вернём результат в виде беззнаковой величины

while((ADCSRA & (1<<ADSC))); //проверим закончилось ли аналого-цифровое преобразование

return (unsigned int) ADC;

Вернёмся теперь в нашу главную функцию main() и создадим там локальную переменную для хранения результата преобразования для дальнейшей с ним работы

int main(void)

{

unsigned int adc_value;

Вызовем функцию преобразования, которая нам положит в нашу переменную результат преобразования

while(1)

{

adc_value = ADC_convert(); //Вызовем преобразование

setpos(0,0);

Давайте сначала отобразим данную сырую величину, хотя бы посмотрим, что в ней есть. За основу мы пока возьмём код из наших часов, функция sprintf на помощь придёт в более поздних занятиях, время её пока не пришло и нам надо вообще понять, как преобразовываются символы. Это нам ой как пригодится в программировании светодиодных индикаторов

setpos(0,0);

sendcharlcd(adc_value/1000+0x30);//Преобразуем число в код числа

```
 sendcharlcd((adc_value%1000)/100+0x30);//Преобразуем число в код числа
 sendcharlcd((adc_value%100)/10+0x30);//Преобразуем число в код числа
 sendcharlcd(adc_value%10+0x30);//Преобразуем число в код числа
 _delay_ms(500);
}
```

Здесь мы разбиваем по цифрам четырёхзначную величину.

Теперь мы соберём код, прошьём контроллер и посмотрим наши результаты, покрутив резистор на 10 килоом

Вот так вот оно всё и работает.

Теперь давайте ещё на дисплее попробуем отобразить всё в вольтах, чтобы определить, какое у нас всё-таки напряжение на центральном контакте нашего переменного резистора. Для этого создадим переменную плавающего типа

unsigned int adc_value;

float n;

Также забудем про существование функции sprintf и попробуем получить плавающий тип на дисплее программным путём. Для этого сначала преобразуем наш сырой результат в плавающий тип явным образом, то есть та же цифра будет, но только тип другой, не забыв, конечно, перед этим поставить курсор в нужное место на дисплее. Для этого существует понятие в языке СИ явного преобразования типов и разделим преобразованный результат на 400

sendcharlcd(adc_value%10+0x30);//Преобразуем число в код числа

setpos(8,0);

n= (float) adc_value / 400;

Тут, конечно, возникает вопрос, а почему мы делим именно на 400. А вот почему.

Это ничто иное как 1024, разделённое на 2,56, то есть на наше опорное напряжение. Видимо, не зря разработчики контроллера выбрали именно такую величину опроного напряжения, чтобы всё делилось без остатка. Почему мы именно такое деление применяем. А потому что у нас есть формула в технической документации

$$ADC = \frac{V_{IN} \cdot 1024}{V_{REF}}$$

Вот поэтому и мы и вычислили её самую последнюю часть. Осталось теперь лишь только перевернуть ещё наоборот, выразив отсюда входное напряжение, так как неизвестное у нас именно оно. И мы получим, что оно будет у нас равно ADC, делённому на 400, что мы, собственно и сделали выше в коде. Я думаю, всё предельно стало теперь всем понятно.

Осталось самое интересное – отобразить всё это на экран, зная, то.что мы не можем работать с дисплеем с плавающим типом. А оказывается всё просто. Всё решается вот таким кусочком кода

```
 n= (float) adc_value / 400;

 sendcharlcd((unsigned char) n +0x30);//Преобразуем число в код числа

 sendcharlcd('.');//Преобразуем число в код числа

 sendcharlcd(((unsigned char) (n*10))%10 +0x30);//Преобразуем число в код числа

 sendcharlcd(((unsigned char) (n*100))%10 +0x30);//Преобразуем число в код числа

 _delay_ms(500);

}
```

Не пугайтесь, сейчас мы всё тут разрулим.

Сначала мы обратным преобразованием типов отсекаем вооще всю дробь и, зная, что дальше 9 мы не уйдём и у нас будет только одна цифра, да мы даже и дальше 2 тут не уйдём, у нас максимум 2,56, мы просто отображаем данную цифру.

Ну дальше по понятным всем причинам будет точка, обычная точка, никакая не плавающая.

Потом мы умножаем наш результат, преобразованный к плавающему типу на 10, тем самым, передвигаем запятую на один разряд в нём вправо и, преобразовав результат вычисления обратно в целочисленный тип, берём из него известным образом младшую цифру и отображаем её на дисплее после запятой.

Подобным образом поступим с цифрой следующей, только здесь мы умножаем результат на 100, что переносит в единицы уже вторую цифру после запятой. Можно продолжить дальше, но нам и двух цифр хватит.

Вот и всё!

Собираем код, прошиваем контроллер и смотрим наши интересные результаты, крутя наш резистор

Вроде бы и всё. Но мы с вами не испытали работу АЦП посредством прерываний.

Поэтому данные испытания мы проведём в **следующей части** нашего занятия.

Предыдущая часть Программирование МК AVR Следующая часть

Исходный код

Программатор и дисплей можно приобрести здесь:

Программатор (продавец надёжный) **USBASP USBISP 2.0**

Дисплей LCD 16×2

Смотреть ВИДЕОУРОК (нажмите на картинку)

Post Views: 13 208