AVR Урок 22. Изучаем АЦП. часть 3 |

Posted on

Урок 22

Часть 3

Продолжаем изучать **АЦП**. В **прошлой части** нашего занятия написали код полностью всех функций и даже посмотрели и оценили работу АЦП на практике.

Только происходило это ручным способом и мы раз в 500 милисекунд постоянно вручную вызывали процесс преобразования, ждали результат, что заметно тратит ресурсы нашего контроллера.

Для того. чтобы этого избежать, существует другой режим работы модуля АЦП в контроллерах AVR – это режим прерываний. Также есть ещё режим не ручного вызова преобразования, а циклического.

Вот такой режим мы сегодня и попробуем реализовать с помощью нашего кода.

Проект мы создадим новый с именем **MyADCISRLCD**, а код в главный модуль весь скопируем с предыдущего проекта, также все файлы включим те же. Это делается для того, чтобы у нас был отдельный проект для прерываний и отдельный для ручного режима.

Удалим полностью функцию запуска конвертирования из файла adc.c, а также из хедера её прототип за её дальнейшей ненадобностью ибо данный процесс у нас будет автоматизирован.

Немного перенастроим регистры в функции инициализации

```
void ADC_Init(void)

{

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

 |(1<<ADSC)//Запуск преобразования

 |(1<<ADFR)//Непрерывный режим работы АЦП
```

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

|(1<<ADIE);//Разрешение прерывания от АЦП

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

}
```

Вообщем, тут, я думаю также всё понятно, так как в **1 части** нашего занятия мы все регистры и их биты изучили. Также не забываем что весь код, кроме последней строки – это одна единая большая строка кода.

Над данной функцией добавим функцию-обработчик прерывания от АЦП

```
//————————————————-

ISR(ADC_vect)

{

}

//————————————————-
```

Для этого у контроллера существует вот такой вектор

Table 18. Reset and Interrupt Vectors

Vector No.	Program Address(2)	Source	Interrupt Definition
15	0x00E	ADC	ADC Conversion Complete

Здесь мы видим, что данное прерывание вызывается в момент окончания аналого-цифрового преобразования.

Также в файле adc.c нам потребуются две глобальне переменные для временного хранения старшей и младшей части регистровой пары adc

```
#include "adc.h"

//————————————————-
```

char high_adc=0,low_adc=0;

Заполним данные переменные значениями из регистра в обработчике прерывания

ISR(ADC_vect)

{

low_adc = ADCL;

high_adc = ADCH;//Верхняя часть регистра ADC должна быть считана последней, иначе не продолжится преобразование

Здесь есть хитрость. Читать нужно сначала младшую часть, так как следующее преобразование автоматически начинается именно в случае такого порядка считывания регистров.

When ADCL is read, the ADC Data Register is not updated until ADCH is read. Consequently, if the result is left adjusted and no more than 8-bit precision is required, it is sufficient to read ADCH. Otherwise, ADCL must be read first, then ADCH.

Из функции main() удалим объявление переменной **adc_value**, но инициализацию оставим

float n;

adc_value=0;

и запишем её в **main.h**, так как данная переменная у нас теперь будет глобальная и она должна быть видна в других модулях

#include <stdlib.h>

unsigned int adc_value;

Сегодня мы с вами впервые попытаемся использовать глобальные переменные, объявленные в одном модуле проекта, в других модуля.

Тут физика такая. Если мы просто попытаемся в другом модуле обратится к такой переменной, то она всё равно будет не видна. Её необходимо в модуль подключить, для этого существует специальный оператор **extern**. Вот с помощью него мы и подключим нашу переменную в файл adc.h

#include "adc.h"

```
//————————————————-

extern unsigned int adc_value;

//————————————————-

char high_adc=0,low_adc=0;
```

Теперь данная переменная будет видна везде и изменения в ней тоже.

Занесём в неё считанные значения из двух переменных в обработчике прерывания от АЦП

```
high_adc = ADCH;//Верхняя часть регистра ADC должна быть считана последней, иначе не продолжится преобразование

adc_value=high_adc*256+low_adc;
```

Можно здесь было также использовать и битовые сдвиги, но мы пока поступим просто.

В функции main() удалим вызов преобразоания, так как прерывание нам вызывать не придется, оно происходит автоматически, а можно и закомментировать

```
while(1)

{

  //adc_value = ADC_convert(); //Вызовем преобразование
```

Также нужно не забыть нам включить ещё прерывания глобальные после инициализации АЦП

```
ADC_Init(); //Инициализируем АЦП

sei();

clearlcd(); //Очистим дисплей
```

Соберём код, прошьём контроллер и попробуем работу нашего кода

Вот такой вот упрощённый механизм использования АЦП по прерываниям.

Предыдущая часть Программирование МК AVR Следующий урок

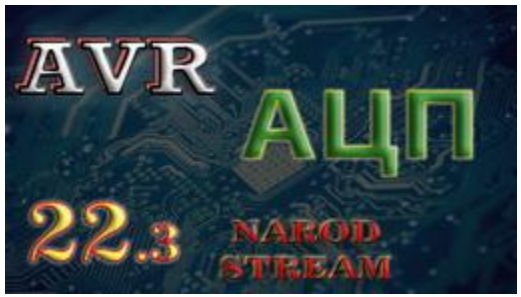
Исходный код

Программатор и дисплей можно приобрести здесь:

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

Дисплей LCD 16×2

Смотреть ВИДЕОУРОК (нажмите на картинку)



Post Views: 10 328