Программирование AVR на примерах Arduino и не только
Часть 2

МпУА Колкер А.Б. Ф-т автоматики НГТУ
a.kolker@corp.nstu.ru

AVR Studio,Wiring

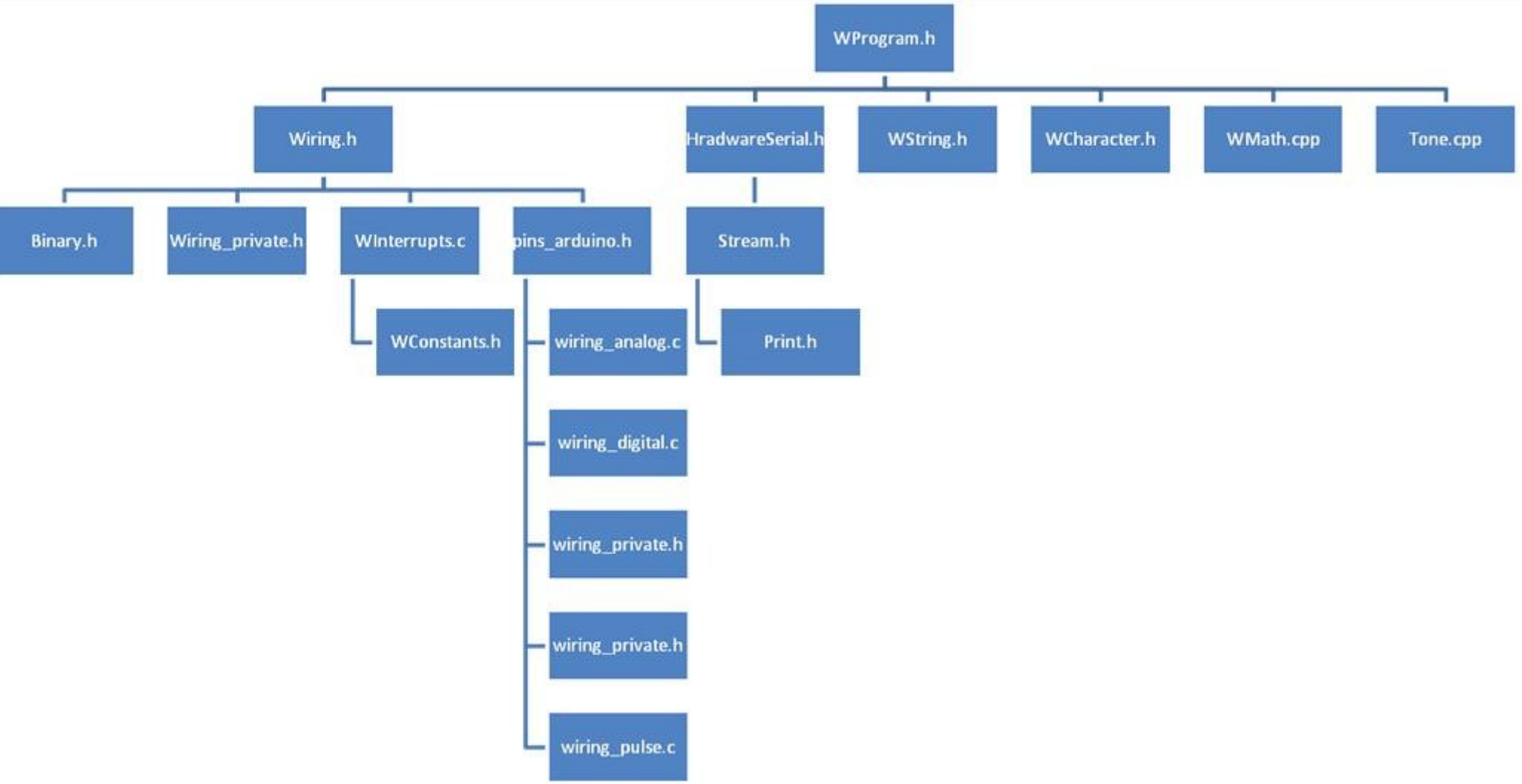
- Arduino — это готовая отладочная плата и очень простой язык для программирования, упрощающие начало работы с микроконтроллерами ценой размера и быстродействия программ. Компиляция — gcc.
- С недавних пор Atmel добавила поддержку загрузчика Arduino в AVR Studio, то есть можно писать загружать без программатора программы написанные хоть на C, хоть на C++, хоть на Asm. Более того — можно в AVR Studio писать код на языке Processing/Wiring.
- Для работы понадобится AVR Studio V6 и выше.

  Насколько потеря размера и быстродействия будет существенным?

Почувствуйте разницу test.cpp

```
#include <util/delay.h>
#define F_CPU 8000000UL
#define DEVICE = atiny13


int main(void)
    {
    TCNT0 =0;
    DDRB = 1<<2 ;
    while (1)
          {
          PORTB ^=1<<2;
          _delay_ms(2000);
           }
    }
```

Основы ядра Arduino



- HardwareSerial.h — класс коммуникаций
- Wiring.h — математика и константы
- Print.h — печать
- Wiring.c - инициализация таймеров и функции подсчета времени
- pins_arduino.c — порты платы=> порты МК
- Wcharacter.h — константы,напр HIGH
- String.h — класс для работы со строками
- wiring_analog.h — DI и ШИМ
- wiring_digital.c — DI/DO
- Winterrupts.c
-
- main.cpp → setup() , loop()

Make.sh

```
rm *.hex
rm *.o

avr-gcc -mmcu=atmega2560 -I. -DF_CPU=8000000UL -Os -o ./test2560.o ./test.cpp.
avr-gcc -mmcu=attiny13 -I. -DF_CPU=8000000UL -Os -o ./test13.o ./test.cpp.
avr-gcc -mmcu=atmega328 -I. -DF_CPU=8000000UL -Os -o ./test328.o ./test.cpp.


avr-objcopy -O binary ./test13.o ./test13.hex
avr-objcopy -O binary ./test2560.o ./test2560.hex
avr-objcopy -O binary ./test328.o ./test328.hex


#sudo avrdude -P /dev/ttyACM0  -c  avrisp  -b 19200 -p attiny13 -U flash:w:test.hex
```

Ls -l *.hex

-rwxrwxr-x 1 alex-i5 alex-i5 **72** окт. 12 15:48 test13.hex

-rwxrwxr-x 1 alex-i5 alex-i5 **294** окт. 12 15:48 test2560.hex

-rwxrwxr-x 1 alex-i5 alex-i5 **166** окт. 12 15:48 test328.hex

Blink — стандартный пример 1106(AT328) /1624 (AT2560)

```
/*
  Blink
  Turns on an LED on for one second, then off for one second, repeatedly.
   This example code is in the public domain.
 */

// Pin 13 has an LED connected on most Arduino boards.
// give it a name:
int led = 13;

// the setup routine runs once when you press reset:
void setup() {
  // initialize the digital pin as an output.
  pinMode(led, OUTPUT);
}

// the loop routine runs over and over again forever:
void loop() {
  digitalWrite(led, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);               // wait for a second
  digitalWrite(led, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);               // wait for a second
   }
```

Главная идея

Arduino IDE – это надстройка над основным кодом, которая предоставляет удобный инструмент для работы с периферией контроллера, а также основные математические функции. Благодаря такой архитектуре ядро легко может быть доработано, а также обеспечена работа на новых контроллерах.

Незначительная потеря быстродействия и менее компактный код компенсируется наличием широкого спектра открытых библиотек*.

* эффект для лабораторных стендов и исследовательских мелкосерийных проектов, где не очень важна стоимость массового производства.

Фьюзы

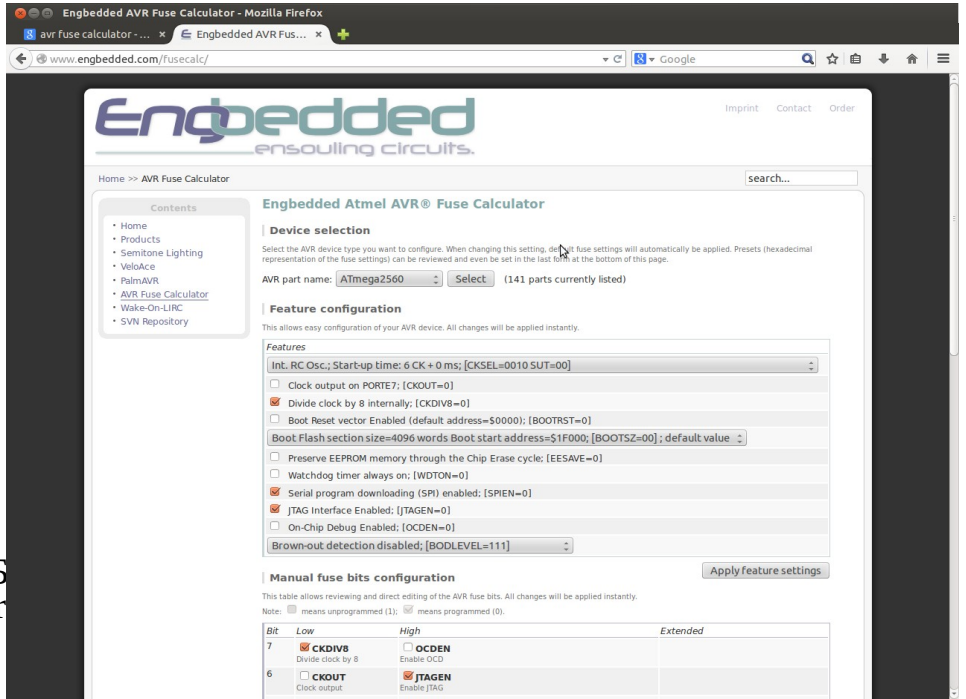
http://www.engbedded.com/fusecalc

Конфигурация микроконтроллера задается служебными регистрами, доступными только в режиме программирования. В AVR Обычно 3 регистра фьюзов:

1 LOW

2 HIGH

3 EXTENDED



- OCDEN — fuse разрешает работу схемы внутреннего отладчика (On Chip Debug ENable). Не оставляйте установленным этот бит в коммерческих продуктах! Иначе вашу программу можно будет считать из памяти МК.

- JTAGEN — fuse бит разрешает работу интерфейса программирования-отладки JTAG. По сравнению с SPI-интерфейсом, JTAG обладает расширенными возможностями. Не рекомендуется без необходимости оставлять этот бит установленным, т. к. в этом случае потребляемый МК ток возрастает.

- SELFPRGEN — бит, разрешающей программе МК производить запись в память программ, т. е. производить самопрограммирование.

- DWEN — fuse бит, разрешающий работу DebugWire - это интерфейс отладки по одному проводу. Не рекомендуется оставлять его установленным в коммерческих изделиях.
-
- EESAVE — fuse бит, после установки которого при стирании памяти МК содержимое EEPROM данных будет сохраняться нетронутым, т. е. не будет стерто.
-
- SPIEN — fuse бит, разрешающий работу интерфейса внутрисхемного программирования МК по SPI. Этот бит может быть легко переустановлен при помощи параллельного программатора (или JTAG, если таковой разрешен и имеется в МК). Все МК выпускаются с установленным битом SPIEN, снять его по интерфейсу SPI невозможно.
-
- WDTON — fuse бит, после установки которого сторожевой таймер WDT включается сразу после подачи питания и не может быть отключен программно. Если бит не установлен, то включением и отключением WDT можно управлять программно.

- Группа fuse битов BODLEVEL. Может быть либо один такой бит, либо несколько, тогда они нумеруются, начиная с нуля. Значение этих fuse битов определяет порог срабатывания схемы BOD — детектора уровня питающего напряжения, при снижении напряжения питания ниже этого уровня произойдет "сброс" МК.
-
- BODEN — fuse бит, включающий схему аппаратного детектора недопустимого уровня
- питающего напряжения, т.е. схему BOD.
-
- RSTDISBL — fuse бит, отключающий сигнал внешнего сброса от вывода микроконтроллера и подключающий к нему схему порта ввода-вывода. Этот бит имеется только в тех МК, у которых вывод аппаратного сброса RESET совмещен с одинм из портов ввода-вывода. Ошибочная установка этого fuse бита может отключить RESET и вы не сможете больше прошивать по ISP. Не устанавливайте этот бит, если намерены продолжать работать с МК при помощи последовательных программаторов. "Оживить" МК с установленным RSTDISBL можно только параллельным программатором и не для всех МК.

- CKDIV8 — fuse бит, включающий предварительное деление частоты кварцевого (или иного имеющегося) тактового генератора на 8. То есть при включенном этом бите и применении кварцевого резонатора на 8 МГц реальная тактовая частота МК составит 1 МГц.
-
- CKOUT — fuse бит, разрешающий вывод тактовой частоты на один из выводов МК (для тактирования других устройств).
-
- SUT1 и SUT0 — fuse биты, управляющие режимом запуска тактовых генераторов МК. Связаны с ниже описываемыми битами, определяющими тип и частоту тактового генератора, причем связь весьма хитрая и запутанная. При ошибочной их установки возможны ситуации неустойчивого запуска генератора или неоднократного сброса МК в процессе подачи на него питания.
-
- CKOPT — бит, определяющий режим работы встроенного генератора тактовой частоты для работы с кварцевыми резонаторами. Реально изменяет коэффициент усиления встроенного инвертора в схеме генератора и значит выходное напряжение на ножке XTAL2. Ошибочная установка может приводить к неустойчивому запуску кварцевого генератора, вплоть до возбуждения его не на той гармонике, что надо (из-за этого бита кварц запускался или только при питании МК напряжением не выше 3,6В, или только после прикосновения к выводу XTAL1 пинцетом). Отчасти аппаратно решается конденсаторами

- Группа битов CKSEL0…CKSEL3 — fuse биты, комбинация которых определяет тип и частоту работающего тактового генератора. Всего возможно до 16 комбинаций, однако не все определены для всех типов МК. Ошибочная установка комбинации этих битов может сделать МК «мертвым» — он не будет работать в схеме без подачи тактового сигнала на ножку XTAL1.
-
- PLLCK — fuse бит, разрешающий использование встроенного синтезатора частоты для тактирования ядра МК.
-
- BOOTRST — fuse бит, определяющий адрес, с которого будет начато исполнение программы после сброса — если бит установлен, то начало программы будет не с адреса 0000h (как обычно), а с адреса области загрузчика (Boot Loader).
-
- Группа fuse битов BOOTSZ — два fuse бита, определяющие размер области памяти программ, выделяемой для загрузчика (Boot Loader). Комбинация этих битов, в частности, определяет точку начала исполнения программы после сброса, если установлен бит BOOTRST.

Brown-out-detection

Супервизор питания. Не даст запуститься микроконтроллеру, если питание не достигнет нужной величины

- 111 — disabled
- 110 — 1.8v
- 101 — 2.7v
- 110 — 4.3v

Что у «UNO» в фьюзах?

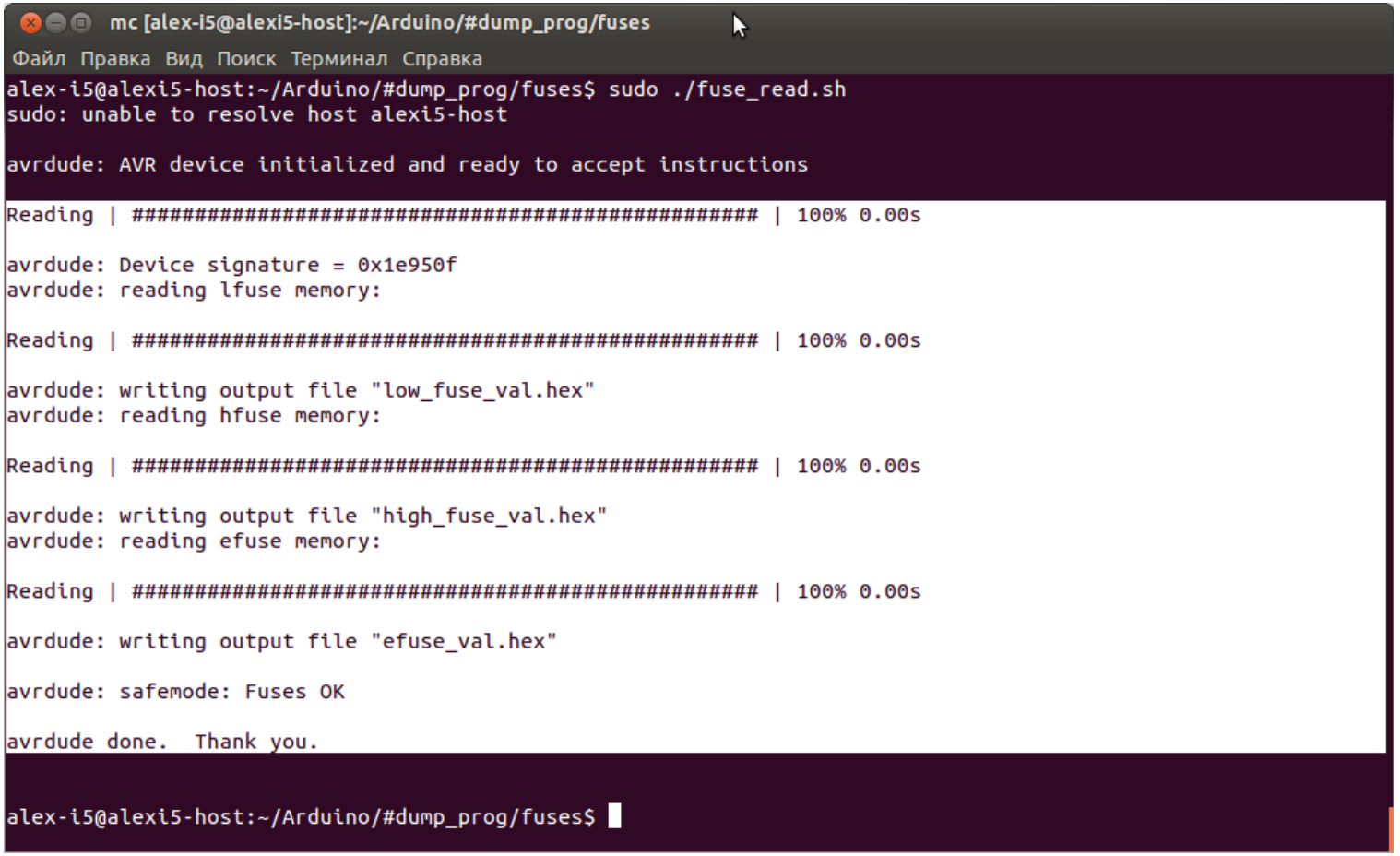
- Возьмем универсальный программатор USBASP
- Зацепим его на порт ISCP

  avrdude -c usbasp -p atmega328P

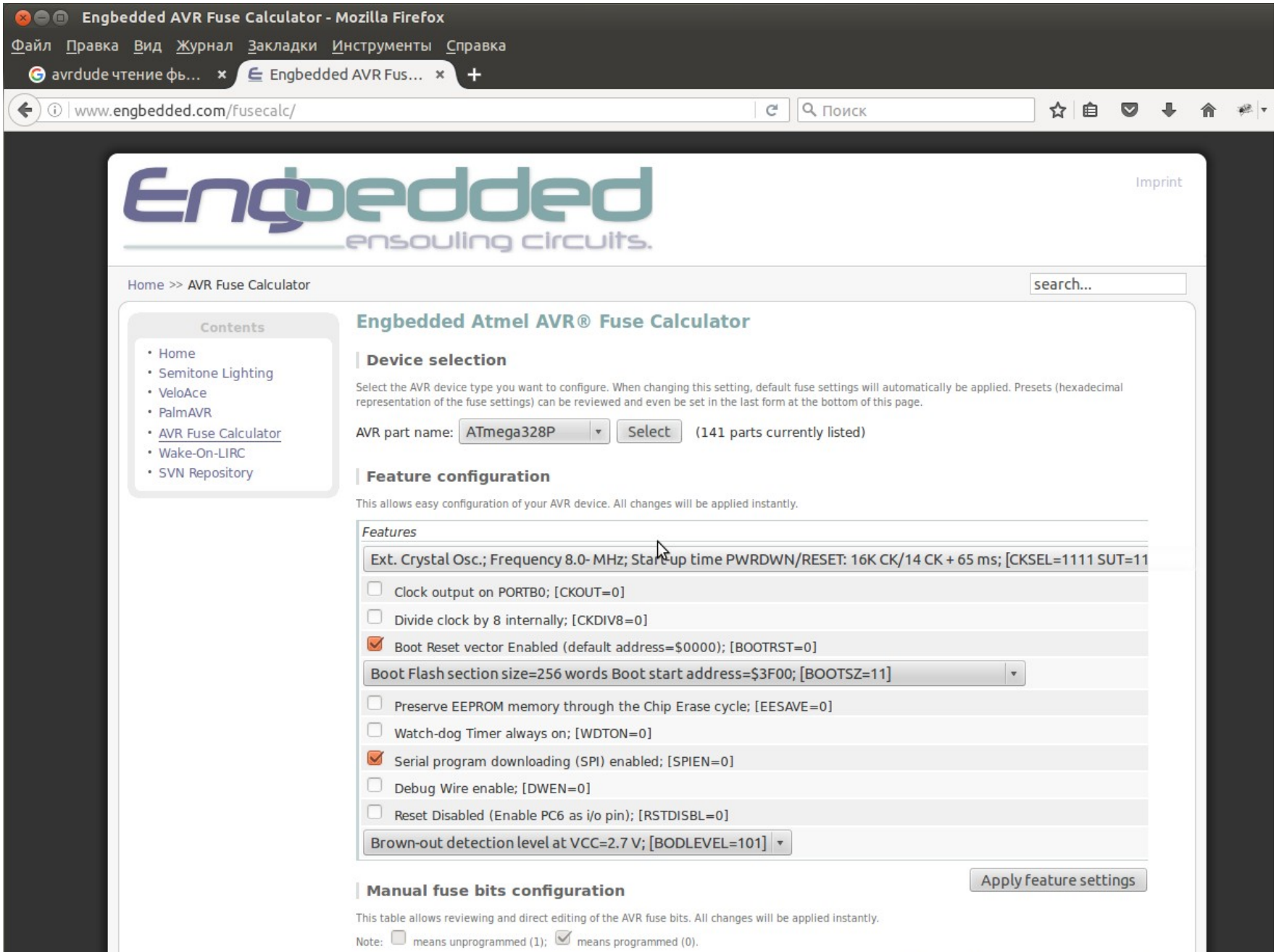
  -U lfuse:r:low_fuse_val.hex:h

  -U hfuse:r:high_fuse_val.hex:h

  -U efuse:r:efuse_val.hex:h


mc [alex-i5@alexi5-host]:~/Arduino/#dump_prog/fuses
Файл Правка Вид Поиск Терминал Справка
alex-i5@alexi5-host:~/Arduino/#dump_prog/fuses$ sudo ./fuse_read.sh
sudo: unable to resolve host alexi5-host
avrdude: AVR device initialized and ready to accept instructions
Reading | ## | 100% 0.00s
avrdude: Device signature = 0x1e950f
avrdude: reading lfuse memory:
Reading | ## | 100% 0.00s
avrdude: writing output file "low_fuse_val.hex"
avrdude: reading hfuse memory:
Reading | ## | 100% 0.00s
avrdude: writing output file "high_fuse_val.hex"
avrdude: reading efuse memory:
Reading | ## | 100% 0.00s
avrdude: writing output file "efuse_val.hex"
avrdude: safemode: Fuses OK
avrdude done. Thank you.
alex-i5@alexi5-host:~/Arduino/#dump_prog/fuses$

L:oxff H:0xde E:0x05



Brown-out detection level at VCC=2.7 V, [BODLEVEL=101]

Apply feature settings

Manual fuse bits configuration

This table allows reviewing and direct editing of the AVR fuse bits. All changes will be applied instantly.

Note: ☐ means unprogrammed (1); ☑ means programmed (0).

Bit	Low	High	Extended
7	☐ **CKDIV8** Divide clock by 8	☐ **RSTDISBL** External reset disable	
6	☐ **CKOUT** Clock output	☐ **DWEN** debugWIRE Enable	
5	☐ **SUT1** Select start-up time	☑ **SPIEN** Enable Serial programming and Data Downloading	
4	☐ **SUT0** Select start-up time	☐ **WDTON** Watchdog Timer Always On	
3	☐ **CKSEL3** Select Clock Source	☐ **EESAVE** EEPROM memory is preserved through chip erase	
2	☐ **CKSEL2** Select Clock Source	☐ **BOOTSZ1** Select boot size	☐ **BODLEVEL2** Brown-out Detector trigger level
1	☐ **CKSEL1** Select Clock Source	☐ **BOOTSZ0** Select boot size	☑ **BODLEVEL1** Brown-out Detector trigger level
0	☐ **CKSEL0** Select Clock Source	☑ **BOOTRST** Select reset vector	☐ **BODLEVEL0** Brown-out Detector trigger level

Apply manual fuse bit settings

Current settings

These fields show the actual hexadecimal representation of the fuse settings from above. These are the values you have to program into your AVR device. Optionally, you may fill in the numerical values yourself to preset the configuration to these values. Changes in the value fields are applied instantly (taking away the focus)!

Low	High	Extended	Action	*AVRDUDE arguments*
0x FF	0x DE	0x 05	Apply values Defaults Apply manual changes to the values on the left side, or load factory default values for the selected device.	`-U lfuse:w:0xff:m -U hfuse:w:0xde:m -U efuse:w:0x05:m` Select (try triple-click) and copy-and-paste this option string into your avrdude command line. You may specify multiple -U arguments within one call of avrdude.

References

All information based on database **ATmega328P.xml** build **1**.
Unreviewed original XML backend database from Atmel. Probably buggy! Please report.

User interface version: 0.9.2.

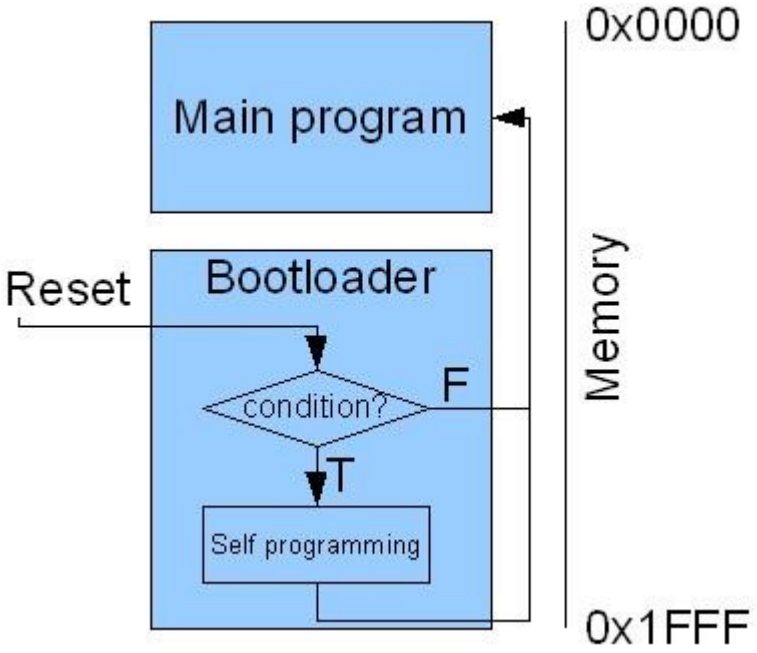
If you find bugs in the user interface or the database backend(s), please report them.

Что делает загрузчик?

После Reset активируется загрузчик, который лежит в нижней области памяти и слушает порт RX TX Reset GND и записывает программу.

Если ничего нет- начинается исполнение программы.

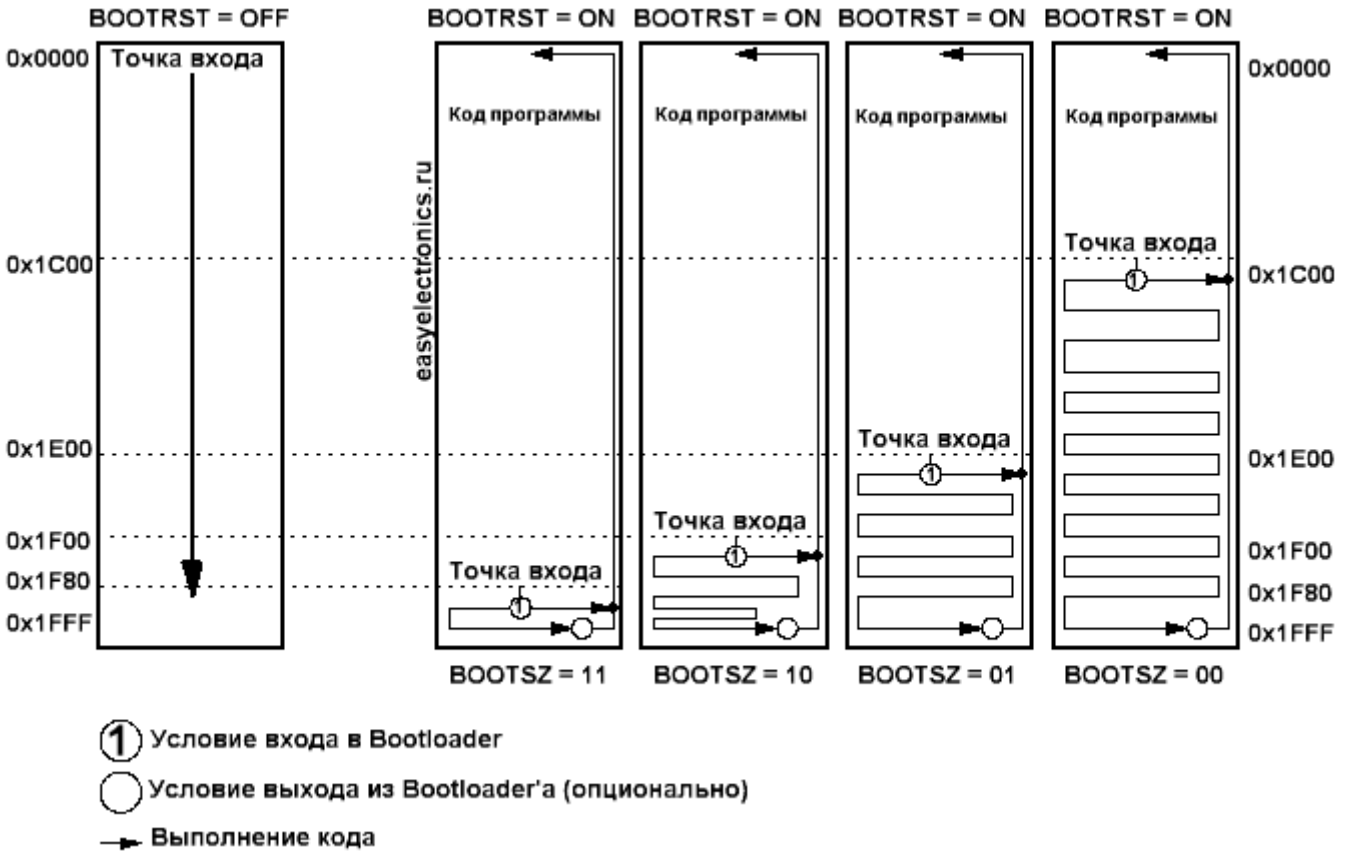
- В Ардуино распаян USB драйвер (или FTDI232, или еще один МК, который умеет «дернуть» МК когда нужно).



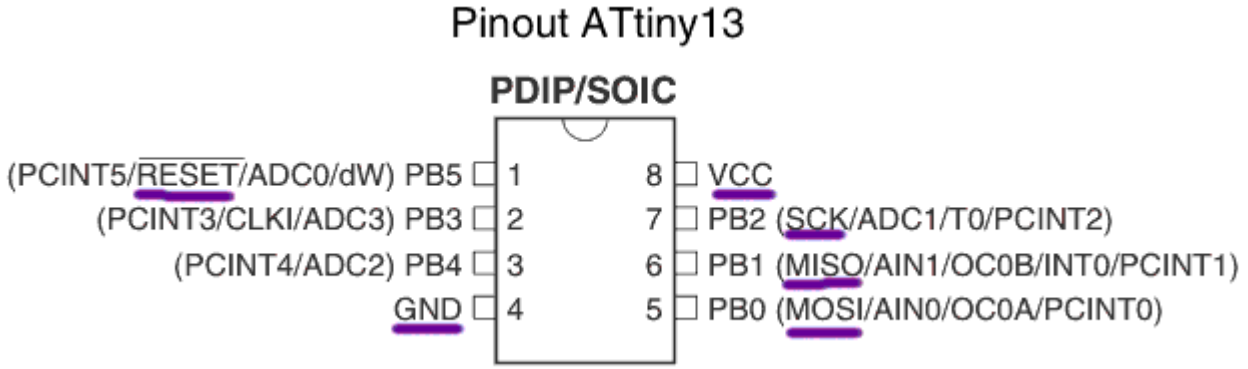
Побочный «баг»

- Из за установленного вектора инициализации, Watchdog работает некорректно!
- Решение: откорректированный загрузчик.

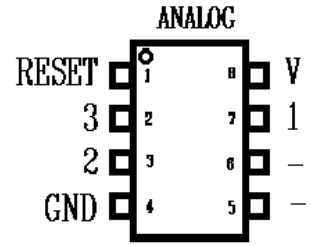
Различные bootsize

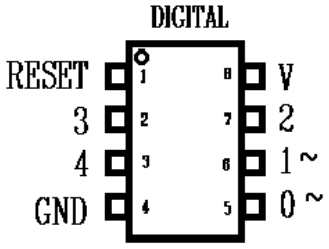


Цоколевка AT Tiny13 выводы многофункциональны



ATTINY 45/85





commands	values
analogRead(pin)	= 0 to 1023
pinMode(pin,value)	INPUT or OUTPUT
analogWrite(pin,value)	0 to 255
digitalRead(pin)	= HIGH or LOW
digitalWrite(pin,value)	HIGH or LOW

~ = PWM

a.kolker@corp.nstu.ru

Порты микроконтроллера

- Один и тот же порт микроконтроллера программным путем может быть переведен в различные режимы,что обеспечивает дополнительную функциональность

Блок-схема пина

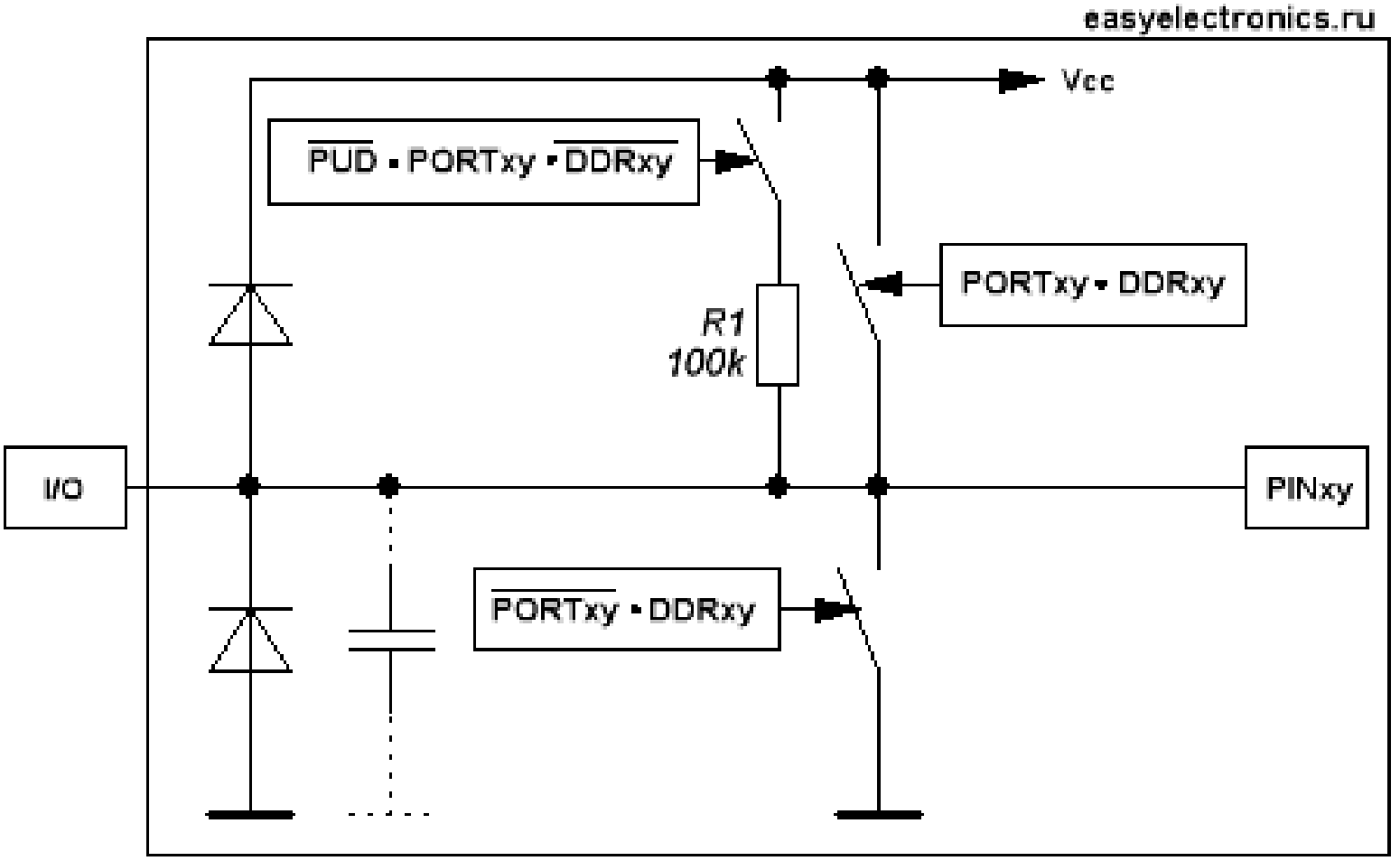


Схема одного вывода порта МК AVR без учета доп функций

Блок-схему можно прочитать так : "любой вывод микроконтроллера может быть настроен как на вход (Input) так и на выход (Output), причем читать состояние пина можно в любой момент, даже когда пин настроен как выход или альтернативная функция (АФ)".

Диоды D1 и D2, предназначены для защиты вывода от напряжений, превышающих напряжение питания (электростатика (ESD) и т.п.), а емкость Cpin это паразитная емкость вывода.

Настройка вывода

Настройка вывода на низком уровне осуществляется через три регистра:

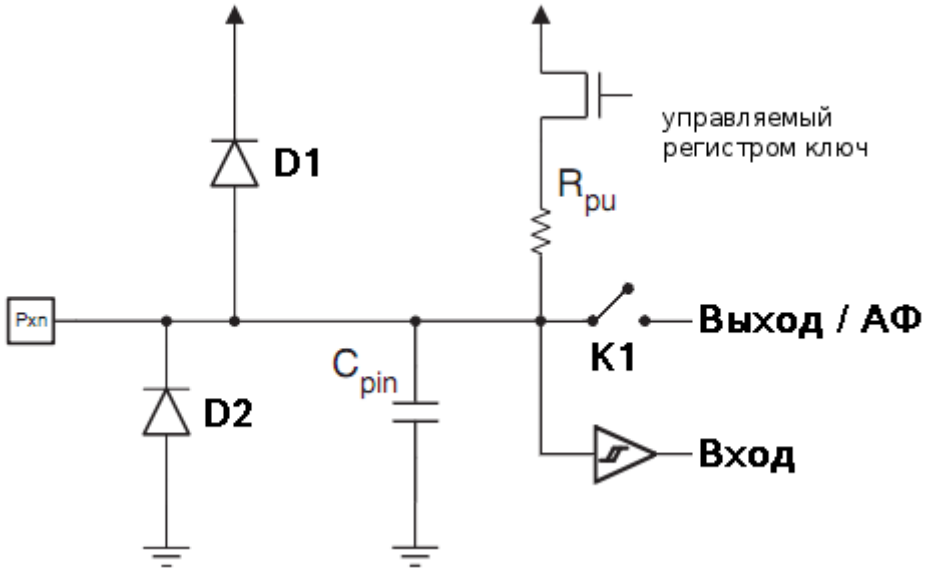
DDRx — Data Direction Register (bit 0<- 1->)

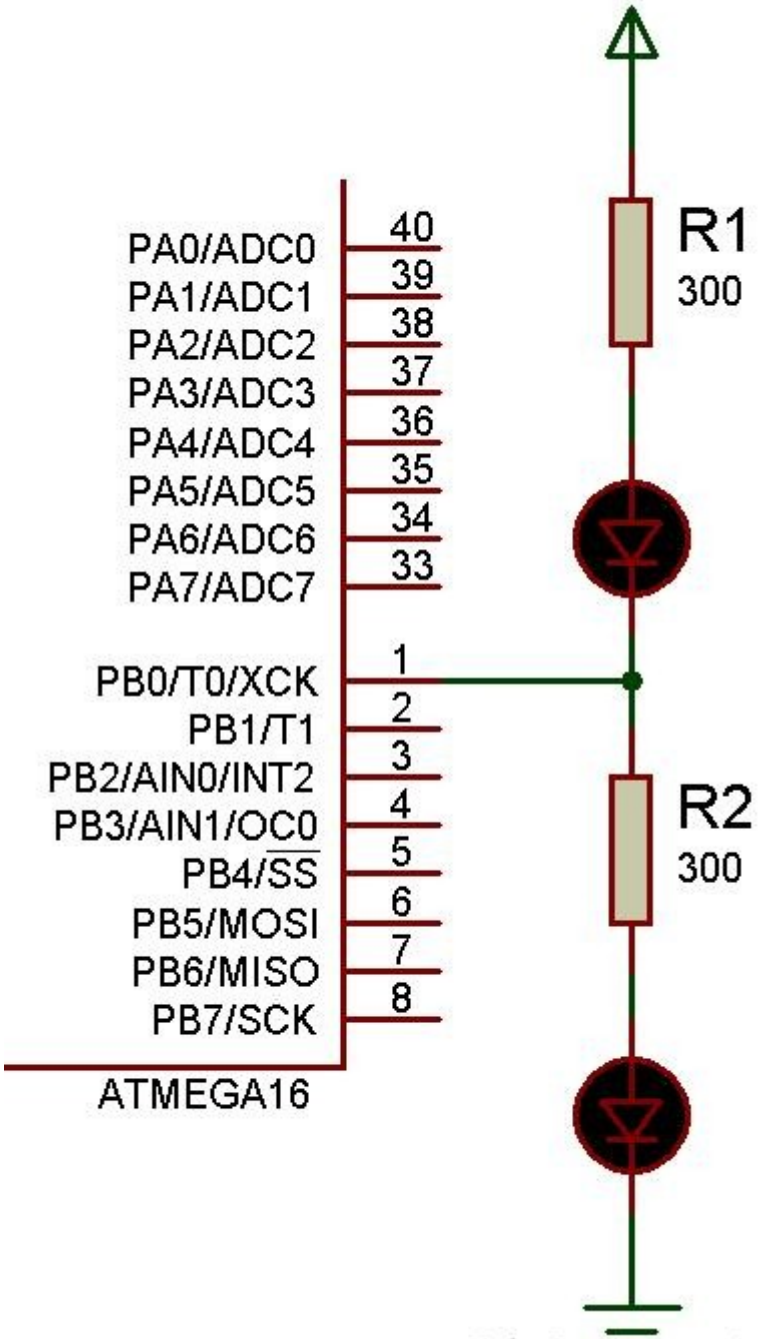
PORTx - Включить подтягивающий резистор

PINx

- x- имя порта. Порт-8 выводов, можно конфигурировать одной командой.

  При программировании на среде Arduino предусмотрены «сервисные» функции, облегчающие жизнь.
- После старта все порты в IN


управляемый
регистром ключ
D1
R_{pu}
Pxn
Выход / АФ
K1
C_{pin}
D2
Вход



матики НГТУ
a.kolker@corp.nstu.ru

Управление 2мя диодами с одного выхода

- PB0 режим OUTPUT ,0 — светится верхний (10ма)
- PB0 режим OUTPUT ,1 — светится нижний (10ма)
- PB0 режим INPUT не светится ни один (во всяком случае ярко)

Советы для реального устройства

- Hi-z — сопротивление очень велико. Если висит в воздухе, то ловит «помехи» (в общем случае 50HZ).
- PullUp — безопасно,если монтажник уронит отвертку, если воздух-то 1

Arduino functions (digital)

- pinMode(pin,INPUT) - высокоимпедансный
- pinMode(pin,INPUT_PULLUP) — с подтяжкой
- pinMode(pin,OUTPUT) — выход
-
- digitalRead(pin)
- digitalWrite(pin,HIGH);
- digitalWrite(pin,LOW);

  Примечание: 13 вывод платы Arduino использовать как вход нельзя из за распайки светодиода.
-

Просто контроллер*

- DDRx = управление разрядами порта вход/выход . Выход =1.
- PORTx = Управление состоянием выходов порта x (если соответствующий разряд настроен как выход), или подключением внутреннего pull-up резистора (если соответствующий разряд настроен как вход)
- PINx Чтение логических уровней разрядов порта

  * в io.h конвертируется в абсолютные адреса при компиляции программы, соотв требуется корректное указание типа МК,

Обычный контроллер

```
DDRD  = 0xFF;   //настройка всех выводов порта D как выходов
PORTD = 0x0F;   //вывод в разряды порта D значений 00001111

uint8_t i=0x54;
PORTD = i;

DDRD=0;                                  // Сброс всех битов в 0 (00000000).
DDRD |= (1 << 1)|(1 << 3)|(1 << 5)|(1 << 7); // Использование операции сдвига
                                         // битов << и логической ИЛИ (OR)
                                         // чтобы установить биты 1, 3, 5, 7
                                         // в лог. 1. Остальные разряды
                                         // останутся в состоянии лог. 0
```

Обычный контроллер

Можно использовать имена Битов

-
- PORTD |=   (1 << PD5)|(1 << PD7); // установка битов 5 и 7
- PORTD &= ~((1 << PD5)|(1 << PD7)); // сброс битов 5 и 7
- PORTD |=   (1 << PD1);   // установка бита 1
- PORTD &= ~ (1 << PD1);   // сброс бита 1
-
- можно использовать define
-
- #define LED PD2
- #define LED_ON() PORTD |= (1 << LED)
- #define LED_OFF() PORTD &= ~(1 << LED)
-
- можно использовать имена бит (зачем?)
-
- PORTD |=   (1 << PD5)|(1 << PD7)

- LED_ON(); //Зажечь светодиод
- LED_OFF(); //Погасить светодиод
-
- #----------

К вопросу о программировании Arduino

```
void setup()
  {
  //один раз после reset — конфигурируем начальное состояние
  }
void loop()
  {
  // тело программы — вызов в вечном цикле
  }
```

А в обычном контроллере:

#define

int a=1; глобальные переменные

main()

{

}

ВАЖНО: в Arduino Main уже есть и он занят!

Что еще Tone:

Tone() - генератор меандра

tone(pin, frequency)

tone(pin, frequency, duration)

Параметры

pin: номер порта вход/выхода, на котором будет генерироваться сигнал

frequency: частота сигнала в Герцах

duration: длительность сигнала в миллисекундах

- Воспроизводиться одновременно может только один сигнал. Если сигнал уже воспроизводится на одном порту, то вызов Tone() с номером другого порта в качестве параметра ни к чему не приведет, если же Tone() будет вызвана с тем же номером порта, то будет установлена новая частота сигнала.

- Использование функции Tone() помешает использовать ШИМ на портах вход/выхода 3 и 11 (кроме MEGA)

- NoTone() - останавливает сигнал

shiftOut()

shiftOut(dataPin, clockPin, bitOrder, value)

Выводит побитно байт на DO

Параметры

dataPin: номер порта вход/выхода, на который выводятся биты (int)

clockPin: номер порта по которому производится синхронизация (int)

bitOrder: используемая последовательность вывода бит. MSBFIRST (Most Significant Bit First) — слева или LSBFIRST (Least Significant Bit First) — справа.

value: значение (байт) для вывода (byte)

AnalogWrite() - ШИМ

Выдает ШИМ на порт,которые это умеет делать.

Частота ШИМ сигнала приблизительно 490 Hz.

- analogWrite(pin, value)

  Параметры

  pin: порт вход/выхода на который подаем ШИМ сигнал.

  value: период рабочего цикла значение между 0 (полностью выключено) and 255 (сигнал подан постоянно).

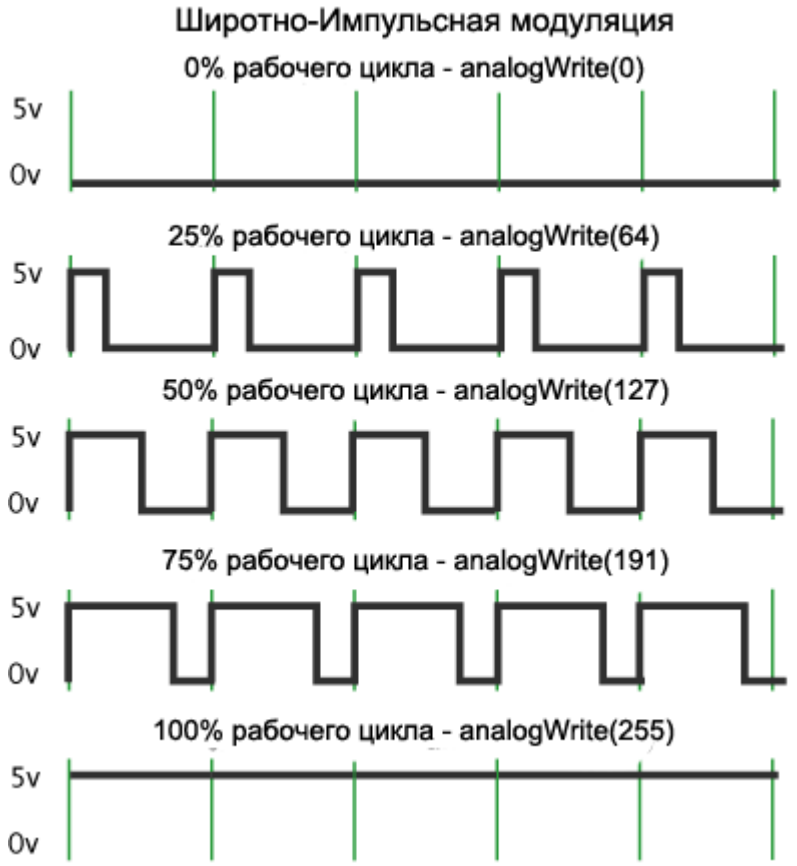
  Примечание:

  На большинстве плат Arduino (на базе микроконтроллера ATmega168 или ATmega328) ШИМ поддерживают порты 3, 5, 6, 9, 10 и 11, на плате Arduino Mega порты с 2 по 13. На более ранних версиях плат Arduino analogWrite() работал только на портах 9, 10 и 11.

  Для вызова analogWrite() нет необходимости устанавливать тип вход/выхода функцией pinMode().

  Функция analogWrite никак не связана с аналоговыми входами и с функцией analogRead.

AnalogWrite()



МпУА Колкер А.Б. Ф-т автоматики НГТУ
a.kolker@corp.nstu.ru

pulseIn()

Считывает длину сигнала на заданном порту (HIGH или LOW). Например, если задано считывание HIGH функцией pulseIn(), функция ожидает пока на заданном порту не появиться HIGH. Когда HIGH получен, включается таймер, который будет остановлен когда на порту вход/выхода будет LOW. Функция pulseIn() возвращает длину сигнала в микросекундах. Функция возвращает 0, если в течение заданного времени (таймаута) не был зафиксирован сигнал на порту.

- Возможны некоторые погрешности в измерение длинных сигналов. Функция может измерять сигналы длиной от 10 микросекунд до 3 минут.

  Синтаксис
- pulseIn(pin, value)
- pulseIn(pin, value, timeout)
- Параметры

- pin: номер порта вход/выхода, на котором будет ожидаться сигнал. (int)
- value: тип ожидаемого сигнала — HIGH или LOW.
- timeout (опционально): время ожидания сигнала (таймаут) в микросекундах; по умолчанию - одна секунда. (unsigned long)

- Вернет: Длина сигнала в микросекундах или 0, если сигнал не получен до истечения таймаута. (unsigned long)

Аналоговые входы

- analogRead(pin) (вернет 10 бит)

  0 до 5 вольт будет преобразовано в значение от 0 до 1023, это 1024 шага с разрешением 0.0049 Вольт

- analogReference() - где брать опорное напряжение
  - DEFAULT: стандартное опорное напряжение 5 В (на платформах с напряжением питания 5 В) или 3.3 В (на платформах с напряжением питания 3.3 В)
  - INTERNAL: встроенное опорное напряжение 1.1 В на микроконтроллерах ATmega168 и ATmega328, и 2.56 В на ATmega8.
  - EXTERNAL: внешний источник опорного напряжения, подключенный к выводу AREF
  -

UART / USART

Последовательный интерфейс для связи между различными устройствами.

- UART- аппаратный порт микроконтроллера
- USART (soft) программная эмуляция

Mega2560 имеет на борту 4 аппаратных UART (указаны пины платы arduino, пины контроллера смотрите по схемы), последовательность rx tx :

Serial 0,1 (для arduino дублируется в софтварный порт в сторону программатора)

Serial1 19,18

Serial2 17,16

Serial3 15,14

Важно: совершенно не стоит их цеплять напрямую к rs232, ибо там +-12в

–

UART

Использование:

Serial.begin(speed)

Serial.begin(speed,config)

speed: можно указывать не только стандартные значения, но для rs232 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, or 115200

config SERIAL_5N1 SERIAL_6N1 SERIAL_7N1 SERIAL_8N1 (the default) SERIAL_5N2 SERIAL_6N2 SERIAL_7N2 SERIAL_8N2 SERIAL_5E1 SERIAL_6E1 SERIAL_7E1 SERIAL_8E1 SERIAL_5E2 ERIAL_6E2 SERIAL_7E2 SERIAL_8E2 SERIAL_5O1 SERIAL_6O1 SERIAL_7O1 SERIAL_8O1 SERIAL_5O2 SERIAL_6O2 SERIAL_7O2 SERIAL_8O2

UART

```
Serial.println("Hello Computer");
Serial1.println("Hello Serial 1");
Serial2.println("Hello Serial 2");
Serial3.println("Hello Serial 3");
```

```
 if (Serial.available() > 0) {
        // read the incoming byte:
        incomingByte = Serial.read();

        // say what you got:
        Serial.print("I received: ");
        Serial.println(incomingByte, DEC);
    }
```

Используем прерывание

Вариант 1: SerialEvent

```
void serialEvent(){
//statements
}
```

Arduino Mega only:

```
void serialEvent[1,2,3](){
//statements
}
```

А как быть с исходящим буфером?

Напрямую лазим в регистр порта

Каждый UART порт имеет 3 регистра конфигурации и статуса:

UCSR0A - статус

UCSR0B - конфигурация

UCSR0C - конфигурация

USCR0A

UCSR0A Bit # Name Description

- **bit 7 RXC0 USART Receive Complete. Set when data is available and the data register has not be read yet.**
- **bit 6 TXC0 USART Transmit Complete. Set when all data has transmitted.**
- bit 5 UDRE0 USART Data Register Empty. Set when the UDR0 register is empty and new data can be transmitted.
- bit 4 FE0 Frame Error. Set when next byte in the UDR0 register has a framing error.
- bit 3 DOR0 Data OverRun. Set when the UDR0 was not read before the next frame arrived.
- bit 2 UPE0 USART Parity Error. Set when next frame in the UDR0 has a parity error.
- bit 1 U2X0 USART Double Transmission Speed. When set decreases the bit time by half doubling the speed.
- bit 0 MPCM0 Multi-processor Communication Mode. When set incoming data is ignored if no addressing information is provided.

USCR0B

UCSR0B Bit # Name Description

- bit 7 RXCIE0 RX Complete Interrupt Enable. Set to allow receive complete interrupts.
- bit 6 TXCIE0 TX Complete Interrupt Enable. Set to allow transmission complete interrupts.
- bit 5 UDRIE0 USART Data Register Empty Interrupt Enable. Set to allow data register empty interrupts.
- bit 4 RXEN0 Receiver Enable. Set to enable receiver.
- bit 3 TXEN0 Transmitter enable. Set to enable transmitter.
- bit 2 UCSZ20 USART Character Size 0. Used together with UCSZ01 and UCSZ00 to set data frame size. Available sizes are 5-bit (000), 6-bit (001), 7-bit (010), 8-bit (011) and 9-bit (111).
- bit 1 RXB80 Receive Data Bit 8. When using 8 bit transmission the 8th bit received.
- bit 0 TXB80 Transmit Data Bit 8. When using 8 bit transmission the 8th bit to be submitted.

UCSR0C Bit # Name Description

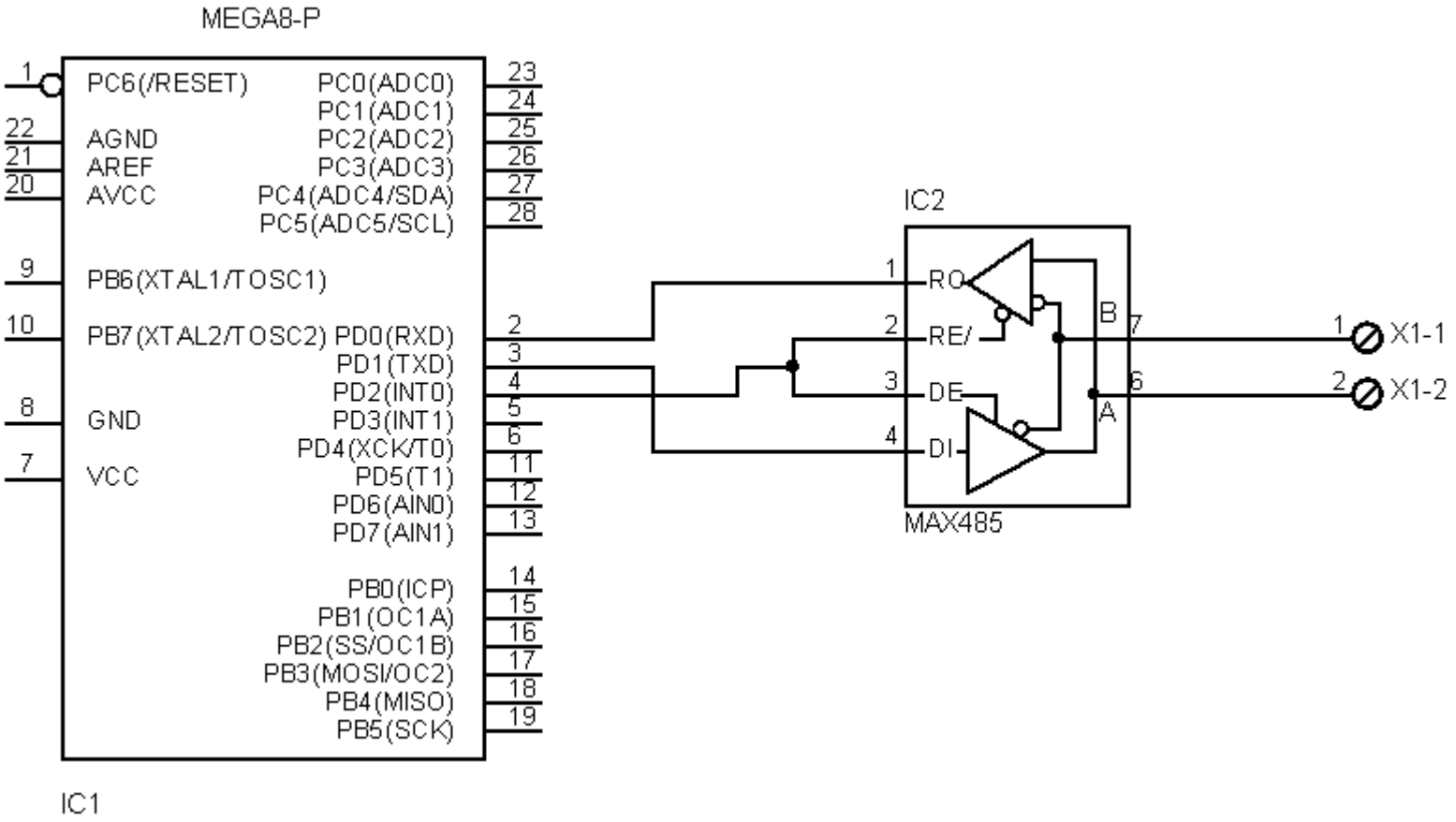
- bit 7
- bit 6 UMSEL01
- UMSEL00 USART Mode Select 1 and 0. UMSEL01 and UMSEL00 combined select the operating mode. Available modes are asynchronous (00), synchronous (01) and master SPI (11).
- bit 5
- bit 4 UPM01
- UPM00 USART Parity Mode 1 and 0. UPM01 and UPM00 select the parity. Available modes are none (00), even (10) and odd (11).
- bit 3 USBS0 USART Stop Bit Select. Set to select 1 stop bit. Unset to select 2 stop bits.
- bit 2
- bit 1 UCSZ01
- UCSZ00 USART Character Size 1 and 0. Used together with with UCSZ20 to set data frame size. Available sizes are 5-bit (000), 6-bit (001), 7-bit (010), 8-bit (011) and 9-bit (111).
- bit 0 UCPOL0 USART Clock Polarity. Set to transmit on falling edge and sample on rising edge. Unset to transmit on rising edge and sample on falling edge.

Как эту штуку использовать

```
while (!(UCSR0A & (1 << TXC0))); // Wait for the transmision to complete
  while (!(UCSR0A & (1 << UDRE0))); // Wait for empty transmit buffer
```

Зачем это надо? Например для RS485

Схема включения MAX485



Software.Serial

В некоторых МК только один или вообще ни одного UART. А хочется. Выход: SoftwareSerial. Займет 2 DI порта (DI-RX,DO-WR).

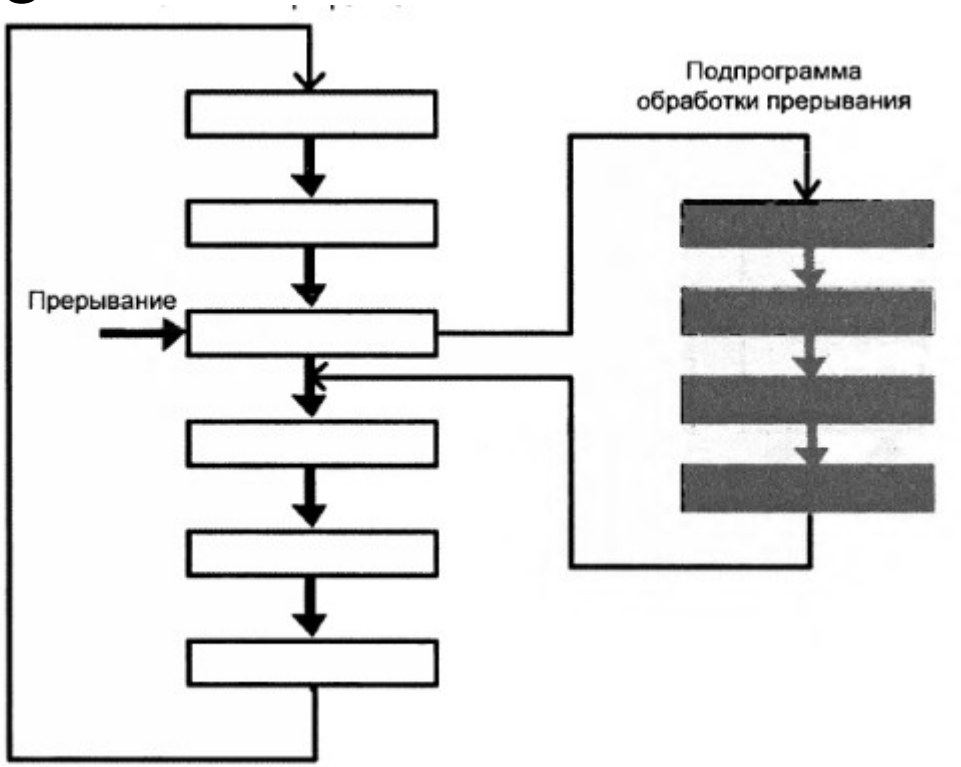
Ограничения: на прием нормально будет работать только один порт, RX может быть назначен только порт с прерыванием.

http://arduino.cc/en/Reference/softwareSerial

```
SoftwareSerial mySerial(10, 11); // RX, TX

void setup()
{
  // Open serial communications and wait for port to open:
  Serial.begin(57600);
  while (!Serial) {
    ; // wait for serial port to connect. Needed for Leonardo only
  }

  Serial.println("Goodnight moon!");

  // set the data rate for the SoftwareSerial port
  mySerial.begin(4800);
  mySerial.println("Hello, world?");
}

void loop() // run over and over
{
  if (mySerial.available())
    Serial.write(mySerial.read());
  if (Serial.available())
    mySerial.write(Serial.read());
}
```

Сага о прерываниях

- http://arduino.cc/en/Reference/Interrupts И не только



Зачем и как?

- Прерывание - это остановка выполнения текущей программы на время выполнения некоторой подпрограммы - «обработчика прерывания».
- В однозадачной среде позволяет незамедлительно реагировать на критические события.
- Регистры и состояние основной программы возвращается «как было» по возвращению назад из обработчика.
- Важно! Вызов прерывания занимает как минимум 4 цикла (могут потребоваться дополнительные циклы для завершения текущих инструкций) . Возврат назад- ровно 4 цикла.

Приоритет основных прерываний

1. Аппаратный сброс.

2. Внешние прерывания.

3. Таймеры-счётчики.

4. SPI.

5. Последовательный порт RS-232.

6. АЦП.

7. Готовность EEPROM.

8. Компаратор.

все прерывания будут вызываться ТОЛЬКО при установленном флаге I в слове состояния

AT TINY13

1 0x0000 RESET External Pin, Power-on Reset,
Brown-out Reset, Watchdog Reset
2 0x0001 INT0 External Interrupt Request 0
3 0x0002 PCINT0 Pin Change Interrupt Request 0
4 0x0003 TIM0_OVF Timer/Counter Overflow
5 0x0004 EE_RDY EEPROM Ready
6 0x0005 ANA_COMP Analog Comparator
7 0x0006 TIM0_COMPA Timer/Counter Compare Match A
8 0x0007 TIM0_COMPB Timer/Counter Compare Match B
9 0x0008 WDT Watchdog Time-out
10 0x0009 ADC ADC Conversion Complete

МпУА Колкер А.Б. Ф-т автоматики НГТУ
a.kolker@corp.nstu.ru

Источники прерываний
(на примере AT Mega8 приоритета >>)

- Адрес Источник прерывания Описание
- 0x0000 RESET Сигнал сброса
- 0x0001 INT0 Внешний запрос на прерывание по входу INT0
- 0x0002 INT1 Внешний запрос на прерывание по входу INT1
- 0x0003 T/C1 Захват по таймеру T/C1
- 0x0004 T/C1 Совпадение с регистром сравнения A таймера T/C1
- 0x0005 T/C1 Совпадение с регистром сравнения B таймера T/C1
- 0x0006 T/C1 Переполнение счётчика T/C1
- 0x0007 T/C0 Переполнение счётчика T/C0
- 0x0008 SPI Передача данных по интерфейсу SPI завершена
- 0x0009 UART Приём данных приемопередатчиком UART завершен
- 0x000A UART Регистр данных UART пуст
- 0x000B UART Передача данных приемопередатчиком UART завершена
- 0x000C ANA_COMP Прерывание от аналогового компаратора

Вектора ATTINY13

he most typical and general program setup fo
r the Reset and Interrupt Vector Addresses in
ATtiny13 is:

```
Address Labels Code                Comments
0x0000         rjmp   RESET         ; Reset Handler
0x0001         rjmp   EXT_INT0      ; IRQ0 Handler
0x0002         rjmp   PCINT0        ; PCINT0 Handler
0x0003         rjmp   TIM0_OVF      ; Timer0 Overflow Handler
0x0004         rjmp   EE_RDY        ; EEPROM Ready Handler
0x0005         rjmp   ANA_COMP      ; Analog Comparator Handler
0x0006         rjmp   TIM0_COMPA    ; Timer0 CompareA Handler
0x0007         rjmp   TIM0_COMPB    ; Timer0 CompareB Handler
0x0008         rjmp   WATCHDOG      ; Watchdog Interrupt Handler
0x0009         rjmp   ADC           ; ADC Conversion Handler
;
0x000A  RESET: ldi    r16, low(RAMEND); Main program start
0x000B    out    SPL,r16            ; Set Stack Pointer to top of RAM
0x000C         sei                 ; Enable interrupts
0x000D    <instr>  xxx
  ...    ...    ...  ...
```

Вектора AT MEGA 328

```
Address  Labels Code            Comments
0x0000         jmp   RESET      ; Reset Handler
0x0002         jmp   EXT_INT0   ; IRQ0 Handler
0x0004         jmp   EXT_INT1   ; IRQ1 Handler
0x0006         jmp   PCINT0     ; PCINT0 Handler
0x0008         jmp   PCINT1     ; PCINT1 Handler
0x000A         jmp   PCINT2     ; PCINT2 Handler
0x000C         jmp   WDT        ; Watchdog Timer Handler
0x000E         jmp   TIM2_COMPA ; Timer2 Compare A Handler
0x0010         jmp   TIM2_COMPB ; Timer2 Compare B Handler
0x0012         jmp   TIM2_OVF   ; Timer2 Overflow Handler
0x0014         jmp   TIM1_CAPT  ; Timer1 Capture Handler
0x0016         jmp   TIM1_COMPA ; Timer1 Compare A Handler
0x0018         jmp   TIM1_COMPB ; Timer1 Compare B Handler
0x001A         jmp   TIM1_OVF   ; Timer1 Overflow Handler
0x001C         jmp   TIM0_COMPA ; Timer0 Compare A Handler
0x001E         jmp   TIM0_COMPB ; Timer0 Compare B Handler
0x0020         jmp   TIM0_OVF   ; Timer0 Overflow Handler
0x0022         jmp   SPI_STC    ; SPI Transfer Complete Handler
0x0024         jmp   USART_RXC  ; USART, RX Complete Handler
0x0026         jmp   USART_UDRE ; USART, UDR Empty Handler
0x0028         jmp   USART_TXC  ; USART, TX Complete Handler
0x002A         jmp   ADC        ; ADC Conversion Complete Handler
0x002C         jmp   EE_RDY     ; EEPROM Ready Handler
0x002E         jmp   ANA_COMP   ; Analog Comparator Handler
0x0030         jmp   TWI        ; 2-wire Serial Interface Handler
0x0032         jmp   SPM_RDY    ; Store Program Memory Ready Handler
;
0x0034   RESET: ldi   r16, high(RAMEND); Main program start
0x0035         out   SPH,r16     ; Set Stack Pointer to top of RAM
0x0036         ldi   r16,   low(RAMEND)
0x0037         out   SPL,r16
0x0038         sei             ; Enable interrupts
0x0039         <instr>  xxx
```

- … … … …

AT MEGA 2560 имеет 53 прерываний

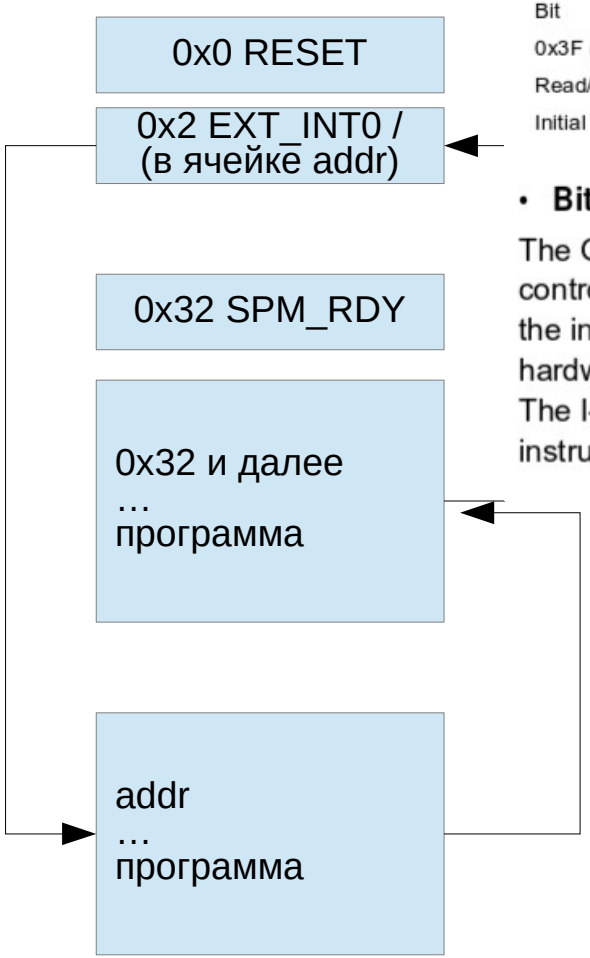
1 $0000(1) RESET External Pin, Power-on Reset, Brown-out Reset, Watchdog
- Reset, and JTAG AVR Reset
- 2 $0002 INT0 External Interrupt Request 0
- 3 $0004 INT1 External Interrupt Request 1
- 4 $0006 INT2 External Interrupt Request 2
- 5 $0008 INT3 External Interrupt Request 3
- 6 $000A INT4 External Interrupt Request 4
- 7 $000C INT5 External Interrupt Request 5
- 8 $000E INT6 External Interrupt Request 6
- 9 $0010 INT7 External Interrupt Request 7
- 10 $0012 PCINT0 Pin Change Interrupt Request 0
- 11 $0014 PCINT1 Pin Change Interrupt Request 1
- 12 $0016(3) PCINT2 Pin Change Interrupt Request 2
- 13 $0018 WDT Watchdog Time-out Interrupt
- 14 $001A TIMER2 COMPA Timer/Counter2 Compare Match A
- 15 $001C TIMER2 COMPB Timer/Counter2 Compare Match B
- 16 $001E TIMER2 OVF Timer/Counter2 Overflow
- 17 $0020 TIMER1 CAPT Timer/Counter1 Capture Event
- 18 $0022 TIMER1 COMPA Timer/Counter1 Compare Match A
- 19 $0024 TIMER1 COMPB Timer/Counter1 Compare Match B
- 20 $0026 TIMER1 COMPC Timer/Counter1 Compare Match C
- 21 $0028 TIMER1 OVF Timer/Counter1 Overflow
- 22 $002A TIMER0 COMPA Timer/Counter0 Compare Match A
- 23 $002C TIMER0 COMPB Timer/Counter0 Compare match B
- 24 $002E TIMER0 OVF Timer/Counter0 Overflow
- 25 $0030 SPI, STC SPI Serial Transfer Complete
- 26 $0032 USART0 RX USART0 Rx Complete
- 27 $0034 USART0 UDRE USART0 Data Register Empty
- 28 $0036 USART0 TX USART0 Tx Complete
- 29 $0038 ANALOG COMP Analog Comparator
- 30 $003A ADC ADC Conversion Complete
-

МпУА Колкер А.Б. Ф-т автоматики НГТУ
a.kolker@corp.nstu.ru

31 $003C EE READY EEPROM Ready

- 32 $003E TIMER3 CAPT Timer/Counter3 Capture Event
- 33 $0040 TIMER3 COMPA Timer/Counter3 Compare Match A
- 34 $0042 TIMER3 COMPB Timer/Counter3 Compare Match B
- 35 $0044 TIMER3 COMPC Timer/Counter3 Compare Match C
- 36 $0046 TIMER3 OVF Timer/Counter3 Overflow
- 37 $0048 USART1 RX USART1 Rx Complete
- 38 $004A USART1 UDRE USART1 Data Register Empty
- 39 $004C USART1 TX USART1 Tx Complete
- 40 $004E TWI 2-wire Serial Interface
- 41 $0050 SPM READY Store Program Memory Ready
- 42 $0052(3) TIMER4 CAPT Timer/Counter4 Capture Event
- 43 $0054 TIMER4 COMPA Timer/Counter4 Compare Match A
- 44 $0056 TIMER4 COMPB Timer/Counter4 Compare Match B
- 45 $0058 TIMER4 COMPC Timer/Counter4 Compare Match C
- 46 $005A TIMER4 OVF Timer/Counter4 Overflow
- 47 $005C TIMER5 CAPT Timer/Counter5 Capture Event
- 48 $005E TIMER5 COMPA Timer/Counter5 Compare Match A
- 49 $0060 TIMER5 COMPB Timer/Counter5 Compare Match B
- 50 $0062 TIMER5 COMPC Timer/Counter5 Compare Match C
- 51 $0064 TIMER5 OVF Timer/Counter5 Overflow
- 52 $0066 (3) USART2 RX USART2 Rx Complete
- 53 $0068 (3) USART2 UDRE USART2 Data Register Empty
- 54 $006A(3) USART2 TX USART2 Tx Complete

Вектора прерываний



SREG – AVR Status Register

The AVR Status Register – SREG – is defined as:

Bit	7	6	5	4	3	2	1	0	
0x3F (0x5F)	I	T	H	S	V	N	Z	C	SREG
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – I: Global Interrupt Enable**

The Global Interrupt Enable bit must be set for the interrupts to be enabled. The individual interrupt enable control is then performed in separate control registers. If the Global Interrupt Enable Register is cleared, none of the interrupts are enabled independent of the individual interrupt enable settings. The I-bit is cleared by hardware after an interrupt has occurred, and is set by the RETI instruction to enable subsequent interrupts. The I-bit can also be set and cleared by the application with the SEI and CLI instructions, as described in the instruction set reference.

1. RESET

Аппаратный сброс контроллера происходит при включении питания, снижении напряжения питания ниже минимально допустимого, срабатывании сторожевого таймера или при подаче логического "0" на вывод #RESET.

При сбросе ВСЕГДА вызывается обработчик, находящийся по адресу BOOTRST и программа "начинает всё сначала", так как при сбросе происходит полная переинициализация всех I/O-регистров. Содержимое регистрового файла и RAM не определено.

BOOTEST — устанавливается фьюзами и может быть отличен от 0 если есть загрузчик.

Пин Reset может быть переопределен (и уже не являться таковым)

2. Внешние прерывания

Внешние прерывания- изменения состояния пина, поддерживающего INT. Количество зависит от типа МК. В МК прерывания управляются регистрами:

GIMSK, MCUCR, GIFR;

EIMSK, EICR, EIFR;

GIMSK, MCUCR.

- Первый в каждой из этих групп - это регистр масок прерываний. Контроллер вызывает соответствующий обработчик только если установлен соответствующий бит в регистре масок.
- Во втором находятся биты управления реакцией на входной сигнал. Прерывание может вызываться при прямом или обратном фронте сигнала или логическим "0" на входе.
- В третьем находятся флаги прерываний. Флаги аппаратно устанавливаются при вызове прерывания и сбрасываются при возврате из обработчика

Программные/аппаратные прерывания

Для того,чтобы аппаратные прерывания работали, пин МК должен быть настроен как **ВХОД**. Если пин настроен как выход, то аппаратные прерывания становятся программными (нет смысла для МК).

- Прерываний пять (Mega328) (INT0 INT1 или любой PCINT0,1,2, обработчик, один) За то,кому назначаются прерывания отвечают PCMSK2, PCMSK1 и PCMSK0.
- Если прерывание используется для «побудки», то сигнал на нем должен оставаться достаточно долго, в течение времени «просыпания» контроллера

2. Внешние прерывания (Mega328)

Разрешение или запрещение внешних прерываний регулируется регистром GICR. Установка соотв. бита разрешает прерывание.

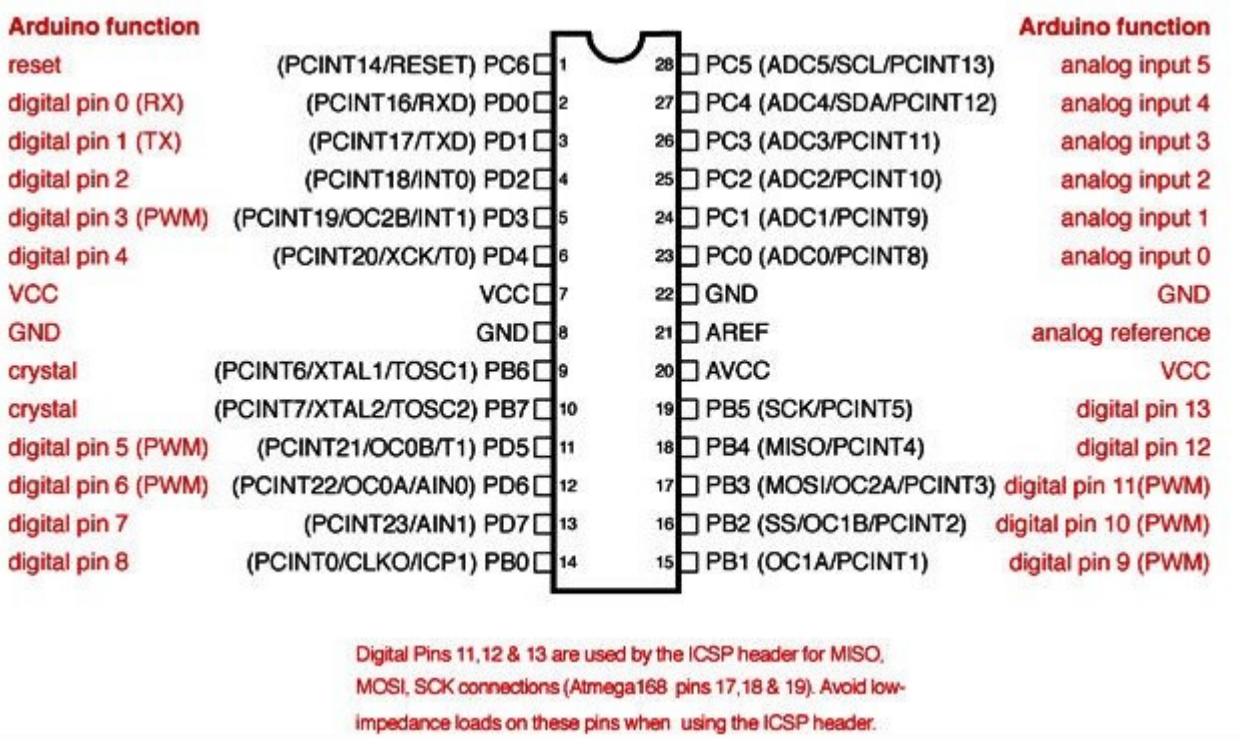
Bit	7	6	5	4	3	2	1	0	
	INT1	INT0	INT2	–	–	–	IVSEL	IVCE	GICR
Read/Write	R/W	R/W	R/W	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Условия внешних прерываний

Внешнее прерывание может происходить по одному из условий:

- по низкому уровню на выводах INT
- по любому изменению логического уровня на выводах INT
- по спадающему фронту сигнала на выводах INT
- по нарастающему фронту на выводах INT, INT

ВАЖНО! Так умеетют работать только порты INT, но не PCINT (см документацию МК)



Digital Pins 11,12 & 13 are used by the ICSP header for MISO, MOSI, SCK connections (Atmega168 pins 17,18 & 19). Avoid low-impedance loads on these pins when using the ICSP header.

Условием генерации прерывания управляют регистры MCU*(MCUCR)

- Bit 1 – IVSEL Interrupt Vector Select — где искать векторы. 1 = начало FLASH 0 = начало RAM (куда указывает BOOTSZ Fuses)
- Bit 0 – IVCE: Interrupt Vector Change Enable — разрешение менять IVSEL (сам обнулится через 4 цикла)
-

MCUCR – MCU Control Register

Bit	7	6	5	4	3	2	1	0	
0x35 (0x55)	–	BODS(1)	BODSE(1)	PUD	–	–	IVSEL	IVCE	MCUCR
Read/Write	R	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Условие обработки — регистр EICRA

Bit	7	6	5	4	3	2	1	0	
(0x69)	–	–	–	–	ISC11	ISC10	ISC01	ISC00	EICRA
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

ISC11	ISC10	Назначение (Применительно только к INT1)
0	0	Прерывание по уровню 0
0	1	Прерывание по любому изменению лог. уровня
1	0	Прерывание по изм. уровня «вниз»
1	1	Прерывание по изм. уровня «вверх»

ISC11	ISC10	Назначение (Применительно только к INT0)
0	0	Прерывание по уровню 0
0	1	Прерывание по любому изменению лог. уровня
1	0	Прерывание по изм. уровня «вниз»
1	1	Прерывание по изм. уровня «вверх»

PCINT

- PCINT — целый порт генерирует прерывание (один обработчик на порт). Можно указать «кому можно» в регистре PCMSK

PCMSK0 – Pin Change Mask Register 0

Bit	7	6	5	4	3	2	1	0	
(0x6B)	PCINT7	PCINT6	PCINT5	PCINT4	PCINT3	PCINT2	PCINT1	PCINT0	PCMSK0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7:0 – PCINT[7:0]: Pin Change Enable Mask 7...0**

Each PCINT[7:0] bit selects whether pin change interrupt is enabled on the corresponding I/O pin. If PCINT[7:0] is set and the PCIE0 bit in PCICR is set, pin change interrupt is enabled on the corresponding I/O pin. If PCINT[7:0] is cleared, pin change interrupt on the corresponding I/O pin is disabled.

Arduino

Interrupts()/noInterrupts() - вкл/выкл прерывания

```
void loop()
{
  noInterrupts();
  // critical, time-sensitive code here
  interrupts();
  // other code here
}
```

ВАЖНО

- ВАЖНО! PCINT не настраиваемое внешнее прерывание, в отличие от INT и всегда работает по принципу Any logical change
- Если началась обработка какого-нибудь прерывания, **никакое другое не может быть вызвано(автоматом устанавливается запрет)**, даже с большим приоритетом.
- Приоритет играет роль, только если несколько вызовов возникли **ОДНОВРЕМЕННО** или для порядка отложенных прерываний

Программирование МК

- Настраиваем MCUCR (MCU Control Register - регистр управления) и GICR (General Interrupt Control Register - общий регистр управления прерываниями)
- При вызове обработчика запрещаем любые другие прерывания (cli() а по завершению и в начале программы возвращаем его назад sei())

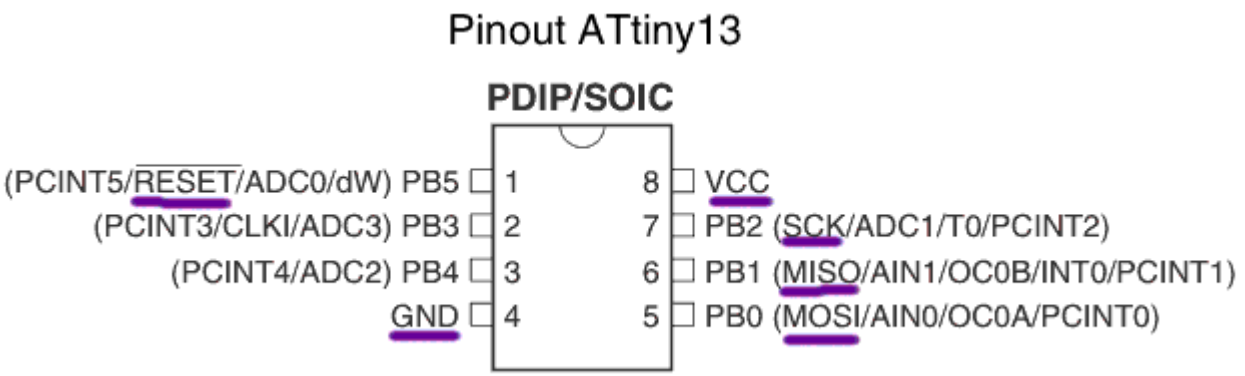
Макросы обработчика прерываний

Определены макросы

```
#include <avr/interrupt.h>
ISR(ADC_vect) // или SIGNAL(..)
{
    // Пользовательский код здесь
}
```

Сигналы (см. приложение)

Пример кода




```
include <avr/io.h> //библиотека ввода/вывода
#include <avr/interrupt.h> //библиотека прерываний
#define F_CPU 8000000UL
#define DEVICE = atiny13

#define nop() {asm("nop");} // немного удобств

unsigned int ms,x; //декларирование переменных
volatile unsigned int counter=0;

// функция задержки на заданное число тактов
void delay_tk(int ms)
   {
   for (x=ms; x>0; x--)
   nop (); //задержка на один такт
   }

//обработка прерываний
SIGNAL(SIG_INTERRUPT0)
   {
   //Тут размещается текст программы которую надо выполнить при срабатывании прерывания INT0
   PORTB ^=1<<3; // меняем состояние светодиода
   counter++;
   }

int main( void ) //главная программа
   {
   GIMSK=0b01000000; //разрешаем прерывание int0 -  кнопка, подключенная к 5В через ножку МК6
   MCUCR=0x03; // срабатывание по нарастанию фронта -  для кнопки
   DDRB = 1<<3 ; // ногу МК 3 (PB4) - к светодиоду в режим выхода
   sei(); //глобальное разрешение прерываний
   for(;;) //бесконечный цикл
      {
       //Тут размещают программу которая выполняется /в свободное от выполнения
      //обработчика прерывания
       delay_tk(100);
      }
   }
```

Вопрос

- Почему этот код будет компилировться, но не будет надежно работать?

Правила создания обработчика

- Обработчик должен быть так прост и быстр, насколько это вообще возможно, он не имеет параметров и ничего не возвращает
- Только одно прерывание может выполняться в один момент, все прерывания «одинакового приоритета», хотя при одновременном появлении нескольких сигналов будет обработано с меньшим номером.
- Если в момент обработки прерывания пришел еще один сигнал,он будет игнорирован(отложен).
- Для общения с телом обработчика используйте глобальные переменные с пометкой violate.
- Вызов занимает минимум 4 цикла CPU, возврат 4 цикла

Распиновка прерываний на Arduino

Соотв. Выводы контроллера см. в DS

	Int.0	Int.1	Int.2	Int.3	Int.4	Int.5
Uno, Ethernet	2	3				
Mega2560	2	3	21	20	19	18
Leonardo	3	2	1	7		

Syntax

attachInterrupt(interrupt, ISR, mode)

interrupt: номер прерывания

ISR — обработчик прерывания

mode — режим обработки сигнала:

- LOW прерывание вызывается если 0
- CHANGE — изменение состояния
- RISING- 0->1
- FALLING 1->0
-

Примечания для Arduino

- Внутри обработчика прерывания delay() не работает — почему?
- Значение mills() будет равно тому,что было присвоено на входе в обработчик

Таймеры

Arduino пользуется тремя таймерами

- Таймер 0 (Системное время, ШИМ 5 and 6)

  Используется для хранения счетчика времени работы программы. Функция millis() возвращает число миллисекунд с момента запуска программы, используя ISR глобального приращения таймера 0. Таймер 0 также используется для реализации ШИМ на выводах 5 и 6.
- Таймер 1 (ШИМ 9 и 10)

  Используется для реализации ШИМ для цифровых выводах 9 и 10.
- Таймер 2 (ШИМ 3 и 11)

  Используется для управления выходами ШИМ для цифровых выводов 3 и 11.

Остальные таймеры доступны для работы

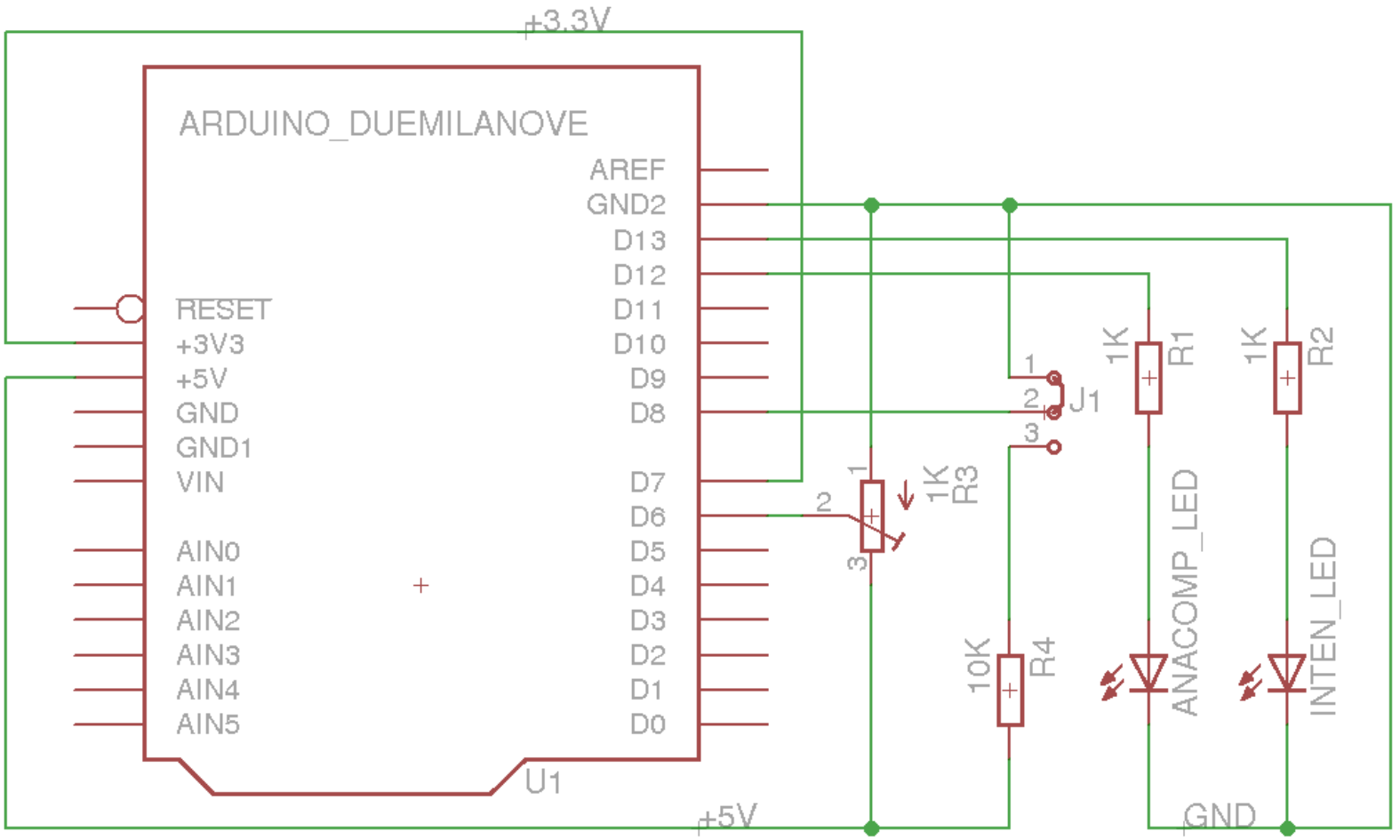
```
define TIMER_CLOCK_FREQ 2000000.0 //2MHz for /8 prescale from 16MHz

//Установка Таймера2.
//Конфигурирует 8-битный Таймер2 ATMega168 для выработки прерывания
//с заданной частотой.
//Возвращает начальное значение таймера, которое должно быть загружено в TCNT2
//внутри вашей процедуры ISR.
//Смотри пример использования ниже.
unsigned char SetupTimer2(float timeoutFrequency){
unsigned char result; //Начальное значение таймера.

//Подсчет начального значения таймера
result=(int)((257.0-(TIMER_CLOCK_FREQ/timeoutFrequency))+0.5);
//257 на самом деле должно быть 256, но я получил лучшие результаты с 257.

//Установки Таймер2: Делитель частоты /8, режим 0
//Частота = 16MHz/8 = 2Mhz или 0.5 мкс
//Делитель /8 дает нам хороший рабочий диапазон
//так что сейчас мы просто жестко запрограммируем это.
TCCR2A = 0;
TCCR2B = 0<<CS22 | 1<<CS21 | 0<<CS20;

//Подключение прерывания по переполнению Timer2
TIMSK2 = 1<<TOIE2;

//загружает таймер для первого цикла
TCNT2=result;

return(result);
• }
```

Прерывание от АЦП, компаратора и остальные

В wiring в явном виде в виде других функций не реализовано, и тут время программирования gcc (AVR со своей мощью никуда не делся...)

```
•
• // Настройка аналогового компаратора
• void anaCompSetup()
• {
•   ACSR  |= _BV(ACIE);                    // Разрешить прерывание аналогового компаратора
•   DIDR1 |= _BV(AIN1D) | _BV(AIN0D);      // Выключить цифровые входы контактов Digital 6 и 7 (для снижения энергопотребления)
• }
•
• // Обработчик прерывания аналогового компаратора
• ISR(ANALOG_COMP_vect)
• {
•   anacompState = ACSR & _BV(ACO);        // Запомнить состояние аналогового компаратора
•   updateLed    = true;                   // Обновить состояние светодиода ANACOMP_LED
• }
```

Прерывания от компаратора



МпУА Колкер А.Б. Ф-т автоматики НГТУ
a.kolker@corp.nstu.ru

Регистр конфигурации прерывания

ACIS1 ACIS0 Режим прерывания

- 0 0 Прерывание при изменении состояния
- 0 1 Не используется
- 1 0 Прерывание по заднему фронту
- 1 1 Прерывание по переднему фронту
-

```
•
• /*
•  Скетч, демонстрирующий работу прерываний микроконтроллера и использование аналогового компаратора.
•
•  Автор (D) Андрей Шаройко <vanyamboe@gmail.com>, 2012
•  http://sites.google.com/site/vanyambauseslinux/electronics/analogovyj-komparator-arduino
•  */
• #define INTEN_LED_PIN    13          // К выводу Digital 13 подключен светодиод "прерывания разрешены"
• #define INTEN_SW_PIN     8           // К выводу Digital 8 подключен переключатель "разрешить/запретить прерывания"
• #define ANACOMP_LED_PIN  12          // К выводу Digital 12 подключен светодиод "состояние компаратора"
•
• volatile bool updateLed = false;
• volatile bool anacompState = false;
•
• // Настройка скетча
• void setup()
• {
•   pinMode(INTEN_LED_PIN, OUTPUT);    // Контакт светодиода INTEN_LED в режим вывода
•   pinMode(INTEN_SW_PIN, INPUT);      // Контакт переключателя INTEN_SW в режим ввода
•   pinMode(ANACOMP_LED_PIN, OUTPUT);  // Контакт светодиода ANACOMP_LED в режим вывода
•
•   anaCompSetup();                    // Настройка аналогового компаратора
• }
•
• // Итерация основного цикла программы
• void loop()
• {
•   if (digitalRead(INTEN_SW_PIN) == HIGH)   // В зависимости от положения переключателя
•     noInterrupts();                        // запретить прерывания
•   else                                     // или
•     interrupts();                          // разрешить прерывания
•
•   if (SREG & 0x80)                         // Если прерывания разрешены
•     digitalWrite(INTEN_LED_PIN, HIGH);     // включить светодиод INTEN_LED
•   else                                     // В противном случае
•     digitalWrite(INTEN_LED_PIN, LOW);      // выключить светодиод INTEN_LED
•
•   if (updateLed) {                         // Если произошло прерывание аналогового компаратора
•     updateLed = false;                     // Отметить, что данное событие обработано
•     if (anacompState)                      // И если компаратор в состоянии логической единицы
•       digitalWrite(ANACOMP_LED_PIN, HIGH);  // включить светодиод ANACOMP_LED
•     else                                   // В противном случае
•       digitalWrite(ANACOMP_LED_PIN, LOW);   // выключить светодиод ANACOMP_LED
•   }
• }
•
• // Настройка аналогового компаратора
• void anaCompSetup()
• {
•   ACSR  |= _BV(ACIE);                      // Разрешить прерывание аналогового компаратора
•   DIDR1 |= _BV(AIN1D) | _BV(AIN0D);        // Выключить цифровые входы контактов Digital 6 и 7 (для снижения энергопотребления)
• }
•
• // Обработчик прерывания аналогового компаратора
• ISR(ANALOG_COMP_vect)
• {
•   anacompState = ACSR & _BV(ACO);          // Запомнить состояние аналогового компаратора
•   updateLed    = true;                     // Обновить состояние светодиода ANACOMP_LED
• }
•
```

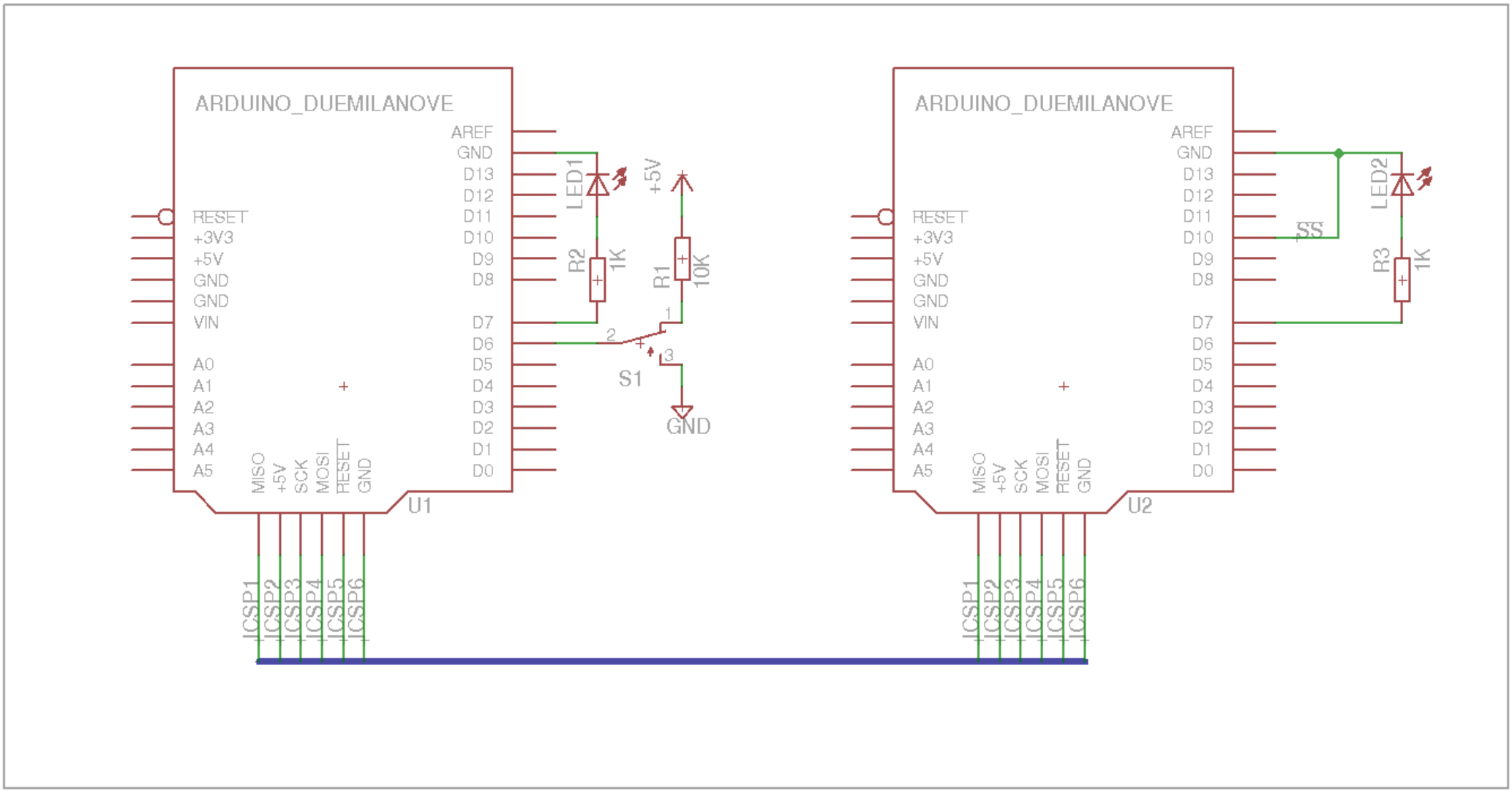
SPI

Последовательный периферийный интерфейс SPI (Serial Peripheral Interface) является высокоскоростным интерфейсом передачи данных между микроконтроллерами и например микросхемами памяти Flash.

Соединив два AVR, по интерфейсу возможно успешно обмениваться данными со скоростью до 4 Мбит/сек.,

(Его же мы зачастую используем и для программирования МК)

SPI


ARDUINO_DUEMILANOVE
AREF
GND
D13
D12
D11
D10
D9
D8
D7
D6
D5
D4
D3
D2
D1
D0
RESET
+3V3
+5V
GND
GND
VIN
A0
A1
A2
A3
A4
A5
MISO
+5V
SCK
MOSI
RESET
GND
U1
ICSP1
ICSP2
ICSP3
ICSP4
ICSP5
ICSP6
LED1
R2
1K
+5V
R1
10K
1
2
3
S1
GND
ARDUINO_DUEMILANOVE
U2
SS
LED2
R3
1K