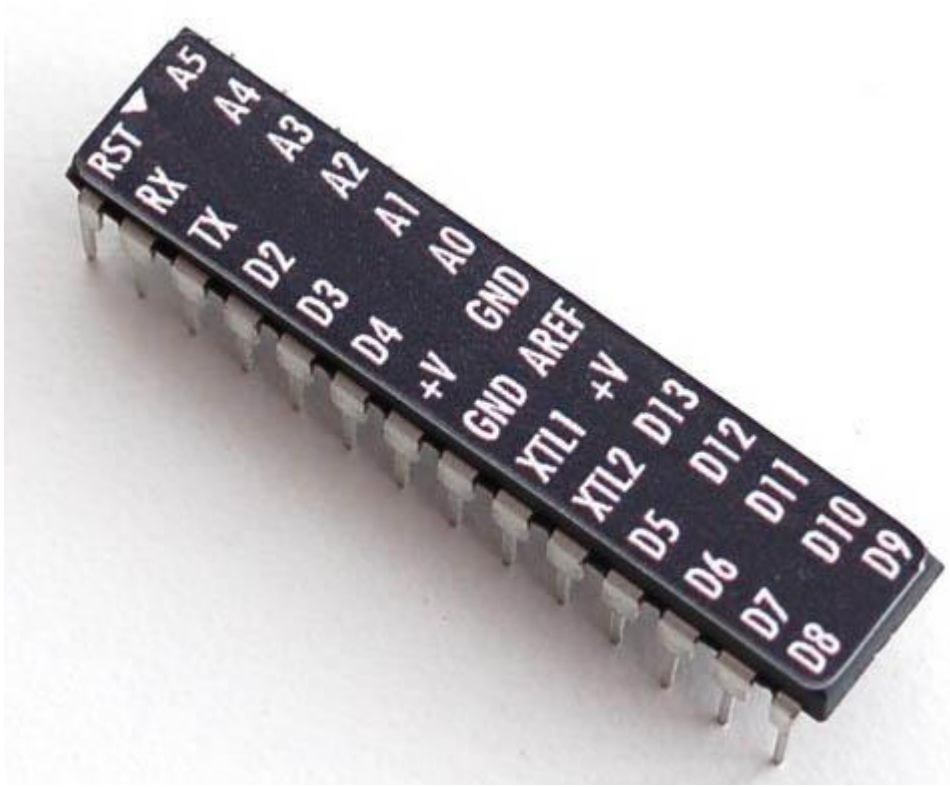
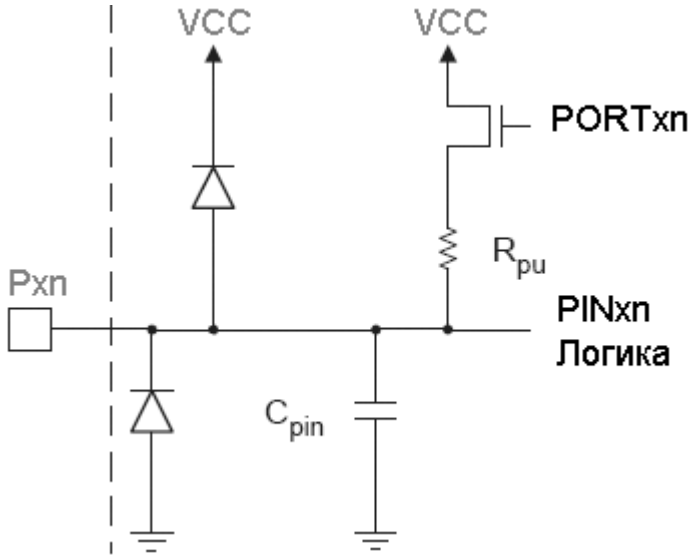
Доступ к портам I/O AVR на языке C (GCC, WinAVR)

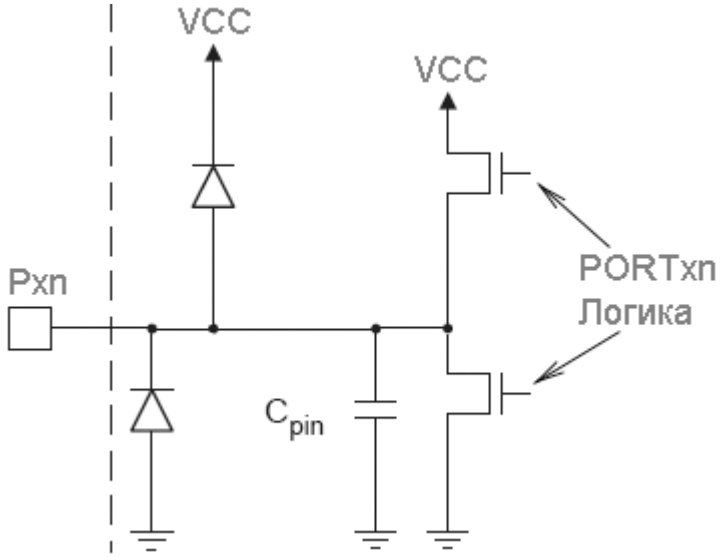


Меня часто спрашивают, как управлять ножками **GPIO** микроконтроллера **AVR**, и как читать их состояние. Несмотря на то, что это довольно просто, описано не только в даташите, но и во многих статьях, например [1], поток вопросов не уменьшается.

Все порты AVR (под AVR обычно подразумеваются микроконтроллеры популярных серий **megaAVR** и **tinyAVR** компании Atmel, например ATmega32A, примененный в макетной плате AVR-USB-MEGA16) обладают функционалом чтение-модификация-запись (**read-modify-write**) при работе с выводами микроконтроллера как обычными портами ввода вывода (general purpose **I/O** ports, GPIO). При этом можно поменять направление (задать что это - вход или выход) для каждой отдельной ножки GPIO (вывода порта AVR). Каждая ножка также имеет симметричный выходной буфер (два CMOS-ключа, один на +, другой на -), который может выдавать выходной ток от плюса питания (VCC, обычно +5V, drive source), или от земли (GND, минус источника питания, sink source). Мощность выходного драйвера более чем достаточна для прямого управления светодиодом (LED). Также для всех выводов портов, настроенных как вход, можно селективно подключить внутренний верхний нагрузочный резистор (**pull-up**), встроенный прямо в кристалл микроконтроллера. Все выводы имеют защищающие от перенапряжения диоды, подключенные к VCC и GND.



Упрощенная схема порта AVR, настроенного как вход (состояние по умолчанию, DDRxn == 0).



Упрощенная схема порта AVR, настроенного как выход (DDRxn == 1).

Примечания к рисункам:

Pxn - имя ножки порта микроконтроллера, где x буква порта (A, B, C или D), n номер разряда порта (7 .. 0).
C_{pin} - паразитная емкость порта.
VCC - напряжение питания.
R_{pu} - отключаемый нагрузочный верхний резистор (pull-up).
PORTxn - бит n регистра PORTx.
PINxn - бит n регистра PINx.
DDRxn - бит n регистра DDRx.

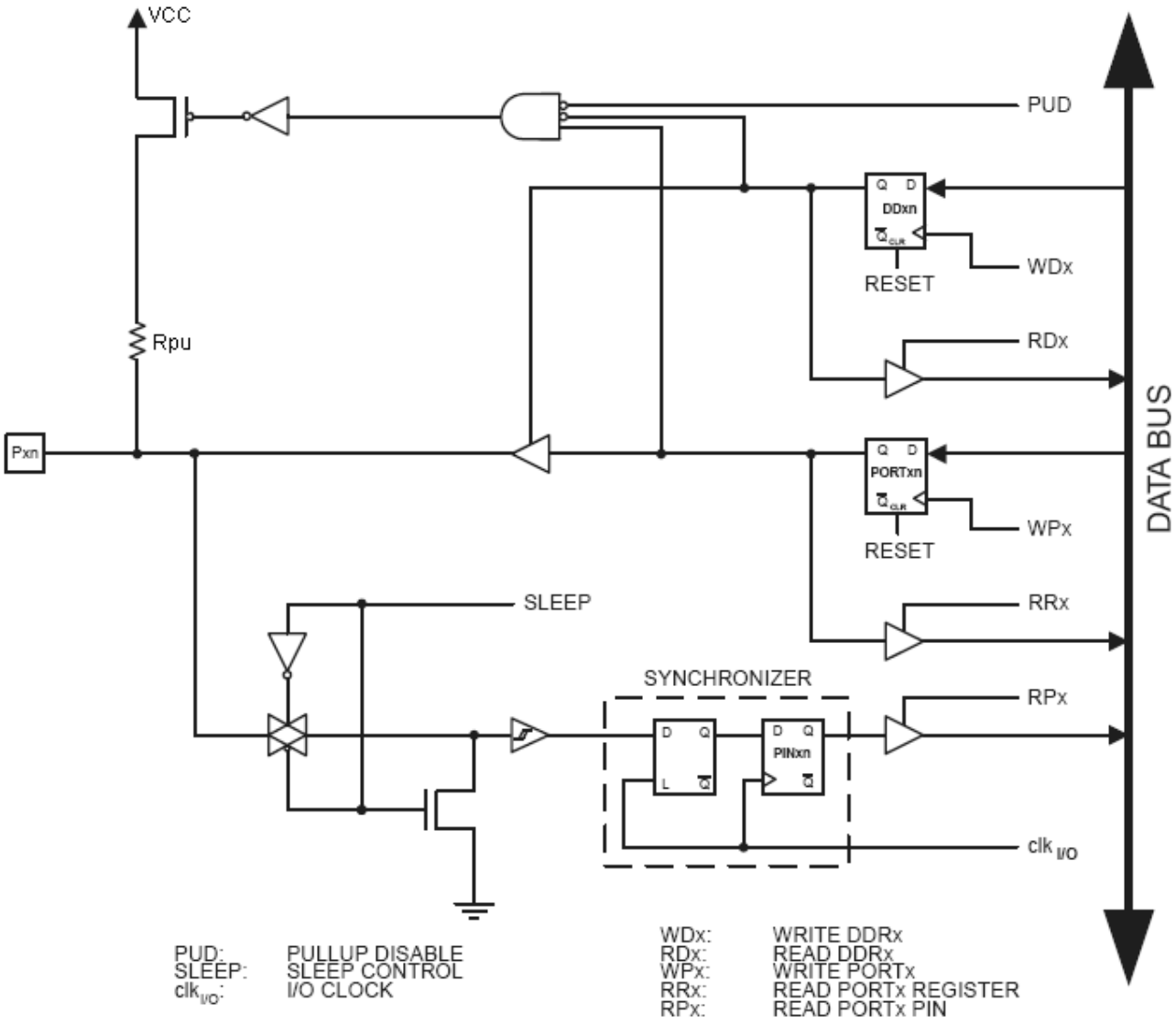
Каждый порт микроконтроллера AVR (обычно имеют имена A, B и иногда C или даже D) имеет 8 разрядов, каждый из которых привязан к определенной ножке корпуса. Каждый порт имеет три специальных регистра **DDRx**, **PORTx** и **PINx** (где x соответствует букве порта A, B, C или D). Назначение регистров:

DDRx	Настройка разрядов порта x на вход или выход.

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 x.

Регистр DDRx выбирает направление работы каждой отдельной ножки порта. Если в разряд регистра DDRx записана лог. 1, то соответствующая ножка будет сконфигурирована как выход. Ноль означает, что порт сконфигурирован как вход (состояние по умолчанию, которое устанавливается после сброса или включения питания). Если в разряд DDRx записан 0, и в соответствующий разряд PORTx записана 1, то порт не только сконфигурирован как вход, но к нему также еще и подключен внутренний верхний нагрузочный резистор (input **pull-up**). Если в разряд DDRx записан 0, и в соответствующий разряд PORTx также записан 0, то порт сконфигурирован как вход с высоким входным сопротивлением, что соответствует отключенному выходному состоянию (третье состояние), при этом можно для искусственного создания логических уровней подключать внешние нагрузочные резисторы (pull-up верхний на VCC или pull-down нижний на GND).

Если в разряд PORTx записана лог. 1, и в соответствующий разряд DDRx записана лог. 1, то порт сконфигурирован как выход, и на выходе будет лог. 1. Если в разряд PORTx записана лог. 0, и в соответствующий разряд DDRx записана лог. 1, то порт сконфигурирован как выход, и на выходе будет лог. 0. Т. е. биты PORTx управляют состоянием выходного порта, при условии что в соответствующий порту разряд DDRx записана лог. 1.



Логическая схема организации порта ввода/вывода (GPIO) микроконтроллера ATmega32A.

[**Предварительная настройка проекта для доступа к GPIO**]

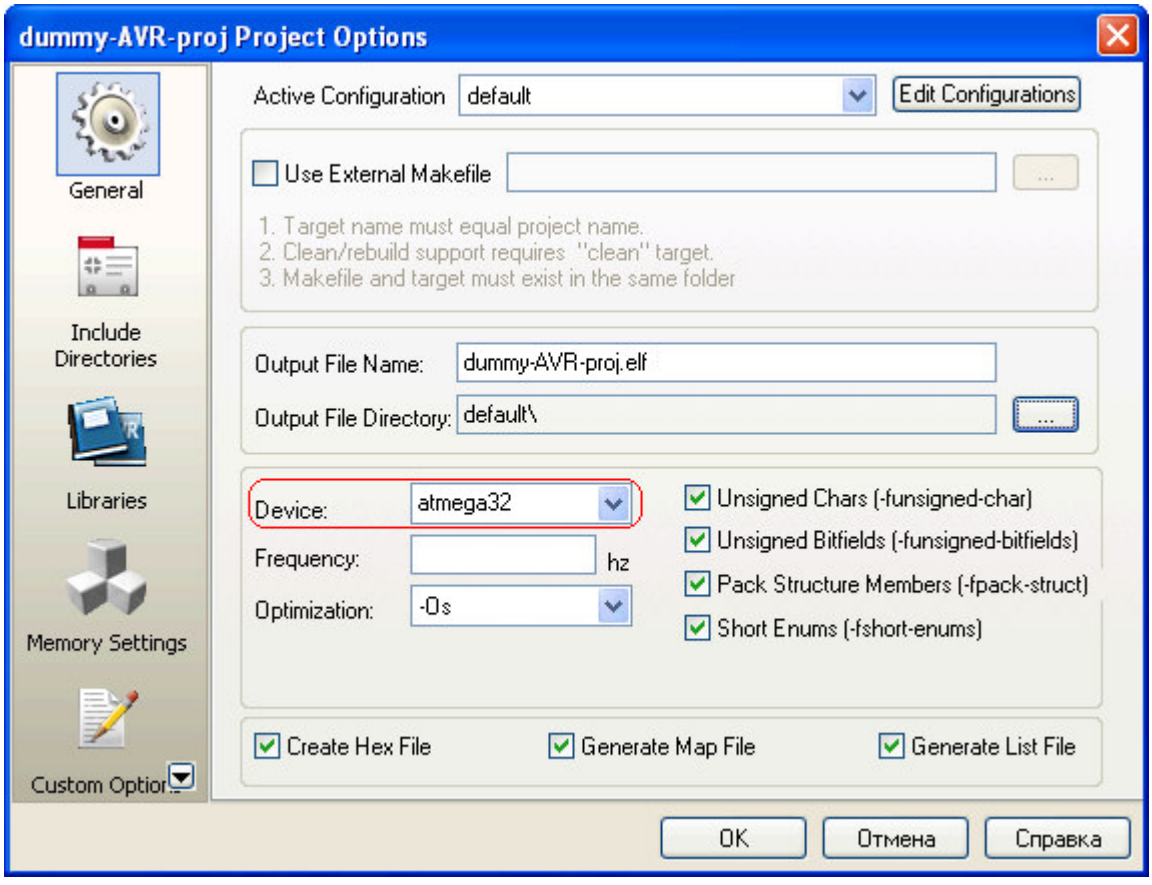
Теперь перейдем к языку C - как можно управлять ножками микроконтроллера AVR? Когда Вы используете **WinAVR GCC** (или соответствующий GCC-тулчейн от Atmel, который устанавливается вместе с **AVR Studio** или **Atmel Studio**), то нужно правильно настроить проект, чтобы нормально распознавались имена регистров AVR (имена DDRA, DDRB, PORTA, PORTB, PINA, PINB и другие), и преобразовывались в нужные адреса. Для этого имеется специальный файл заголовка *io.h*, который находится в папках инсталляции WinAVR (к примеру, это может быть папка c: \ WinAVR-20100110 \ avr \ include \ avr) или соответствующего тулчейна Atmel (для Atmel Studio это может быть папка c: \ Program Files \ Atmel \ Atmel Studio 6.0 \ extensions \ Atmel \ AVRGCC \ 3.3.2.31 \ AVRToolchain \ avr \ include \ avr). Для того, чтобы подключить заголовок *io.h*, нужно в код модуля (например файл main.c) добавить строку с директивой *#include*, и в настройках проекта правильно указать тип микроконтроллера. Вот как указывается директива *#include*:

```
#include < avr/io.h >
```

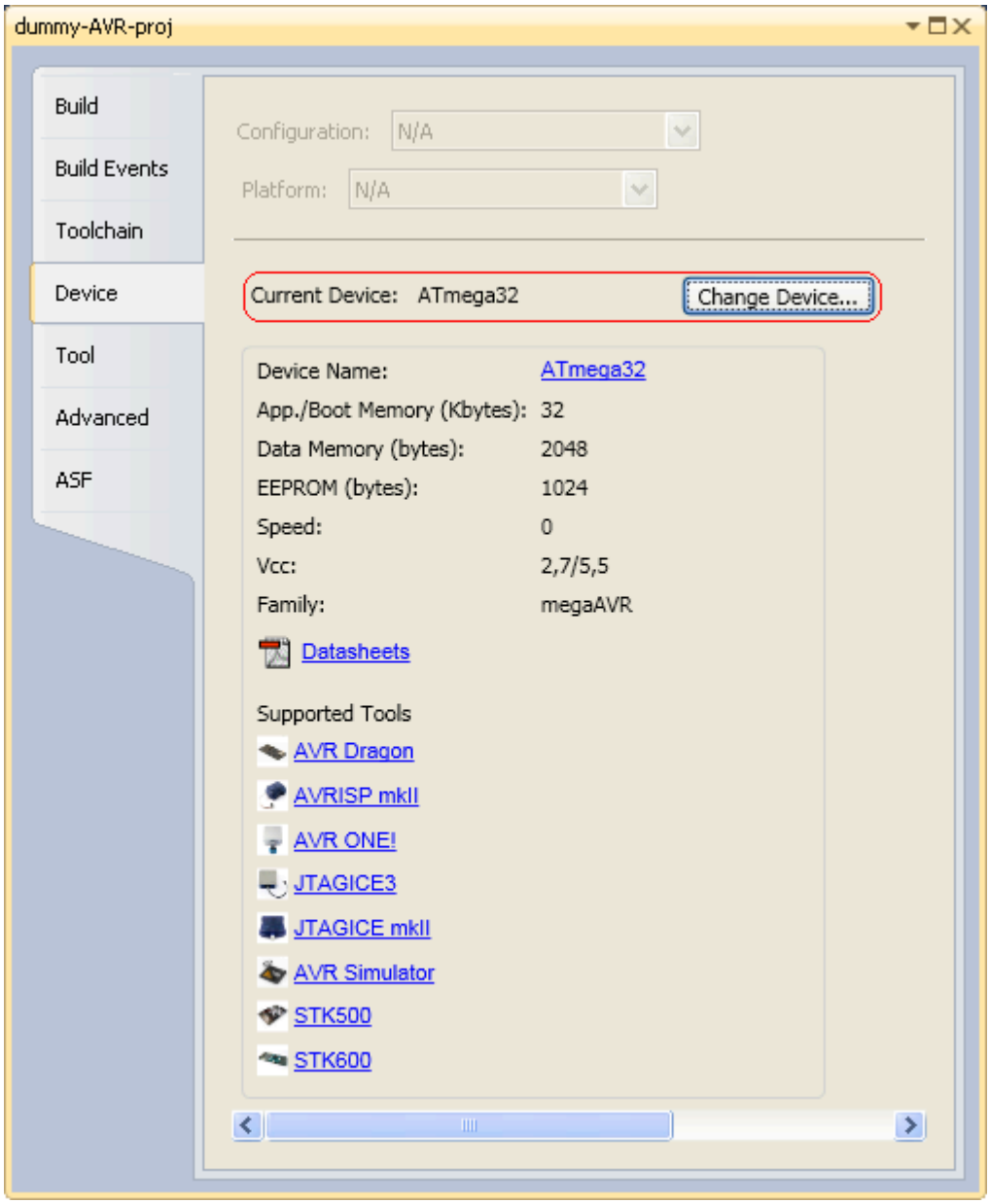
Имя (модель) процессора AVR может быть указано либо прямо в файле **Makefile** определением переменной **DEVICE**, или просто в настройках проекта (AVR Studio или Atmel Studio). Вот пример куска Makefile, где задан тип микроконтроллера ATmega32:

```
F_CPU = 12000000
DEVICE = atmega32
BOOTLOADER_ADDRESS = 0x7000
..
```

Вот так настраивается тип микроконтроллера в свойствах проекта **AVR Studio 4.19** (меню Project -> Configuration Options):



Вот так настраивается тип микроконтроллера в свойствах проекта **Atmel Studio 6.0** (меню Project -> Properties):



После того, как подключен файл io.h и задан тип микроконтроллера для проекта, можно в коде программы на языке C использовать имена регистров AVR. Через имена регистров осуществляется доступ к портам GPIO микроконтроллера.

[**Как работать с портами AVR как с выходами**]

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

Если у нас есть 8-битная переменная i, то мы можем присвоить её значение регистру PORTx, и тем самым установить ножки микроконтроллера в состояние, соответствующее значению переменной i:

```
1  uint8_t i=0x54;
2
3  PORTD = i;
```

Здесь показана работа с портом D, но Вы точно так же можете работать и с портами A, B, C, если будете использовать соответствующие имена регистров (DDRA, PORTA, DDRB, PORTB и т. п.).

[**Как работать с портами AVR как со входами**]

Вот как можно прочитать логические уровни из порта D в переменную i:

```
DDRD = 0; //настройка всех выводов порта D как входов
i = PIND; //прочитать все 8 ножек порта D и сохранить значение в i
```

[**Как работать с отдельными ножками порта AVR как с выходами**]

Есть возможность получить доступ к отдельным ножкам порта AVR. Это позволяет гибко использовать разряды порта для разных применений.

Некоторые из выводов 8-разрядного порта могут быть сконфигурированы и работать как выходы, причем остальные выводы порта могут работать как входы. Т. е. разные разряды одного порта могут выполнять разные функции в зависимости от потребностей пользователя.

Например, нам нужно, чтобы у порта D разряды 0, 2, 4, 6 (PD0, PD2, PD4, PD6) работали как входы (input), и разряды 1, 3, 5, 7 (PD1, PD3, PD5, PD7) работали как выходы (output). Тогда мы можем использовать код наподобие следующего:

```
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
```

Теперь мы можем вывести любое значение в разряды ножек порта (выходы) 1, 3, 5 и 7. Вот как можно установить эти разряды в лог. 1:

```
PORTD |= (1 << 1)|(1 << 3)|(1 << 5)|(1 << 7);
```

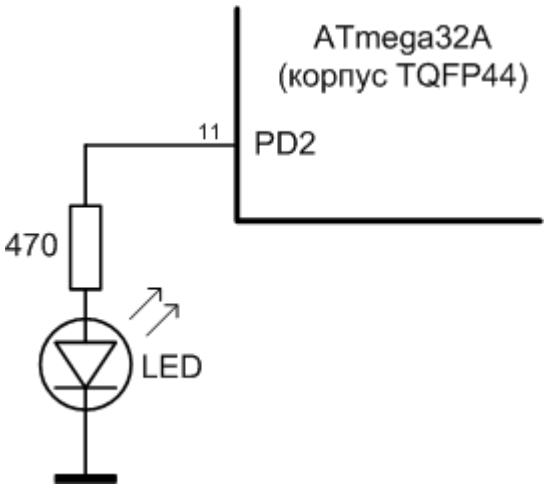
А так можно сбросить эти ножки в лог. 0:

```
PORTD &= ~((1 << 1)|(1 << 3)|(1 << 5)|(1 << 7));
```

Можно также использовать имена битов, соответствующие их номерам разрядов. Вот например, как можно работать с разрядами порта D через имена:

```
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 также для удобства назначать собственные мнемонические имена для разрядов и регистров. Вот как к примеру можно управлять светодиодом, подключенным к разряду 2 порта D:




```
#define LED PD2
#define LED_ON() PORTD |= (1 << LED)
#define LED_OFF() PORTD &= ~(1 << LED)

LED_ON();  //Зажечь светодиод
LED_OFF(); //Погасить светодиод
```

Здесь макросы LED_ON и LED_OFF заданы для удобного управление одним разрядом порта D. Они позволяют включать и выключать светодиод, подключенный к ножке порта PD2, и делают программу наглядной и простой.

Для составления масок есть также удобные макросы наподобие **_BV(n)**.

[**Чтение отдельных разрядов порта AVR**]

Чтение отдельных разрядов порта (ножек микроконтроллера) AVR также осуществляется очень просто. К примеру, настройте отдельные ножки (разряды 1 и 3) порта D как входы, и прочитайте их состояние в переменную:

```
DDRD &=~ ((1 << 1)|(1 << 3)); // Очистка битов 1 и 3 регистра направления порта D.
i = PIND;                     // Прочитать состояние всех 8 ножек порта D.
```

Теперь Вы можете узнать логическое состояние отдельных разрядов (1 или 3) с помощью использования битовых масок, наложенных на значение переменной.

```
if ((1 << PD3) & i)
{
   //Здесь можно выполнить некоторые действия, если PD3==1
   ..
}
else
{
   //Здесь можно выполнить некоторые действия, если PD3==0
   ..
}
```

Точно так же можно узнать состояние ножки PD1, если наложить на i маску (1<<PD1):

```
if ((1 << PD1) & i)
{
   //Здесь можно выполнить некоторые действия, если PD1==1
   ..
}
else
{
   //Здесь можно выполнить некоторые действия, если PD1==0
   ..
}
```

Второй способ - сдвинуть i вправо нужное число раз (для нашего примера 1 или 3 раза), и затем проверить значение младшего разряда i.

```
i >> =3;
if (1 & i)
{
   //Здесь можно выполнить некоторые действия, если PD3==1
   ..
}
else
{
   //Здесь можно выполнить некоторые действия, если PD3==0
   ..
}
```

Есть также удобные библиотечные макросы наподобие **bit_is_set**() или **bit_is_clear**(), имеющихся в файле *sfr_defs.h*, которые упрощают задачу проверки отдельных бит.

[**Бит PUD**]

Имеется также специальная возможность отключения всех внутренних нагрузочных резисторов AVR на всех портах сразу, если установить бит **PUD** (аббревиатура расшифровывается Pull-Up Disable) в регистре **SFIOR** (Special Function I/O Register). По умолчанию (после сброса или включения питания) этот бит сброшен, и не оказывает влияние на настройку нагрузочных резисторов.

Bit	7	6	5	4	3	2	1	0	
	ADTS2	ADTS1	ADTS0	–	ACME	PUD	PSR2	PSR10	SFIOR
Read/Write	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Регистр SFIOR и размещение в нем бита PUD.

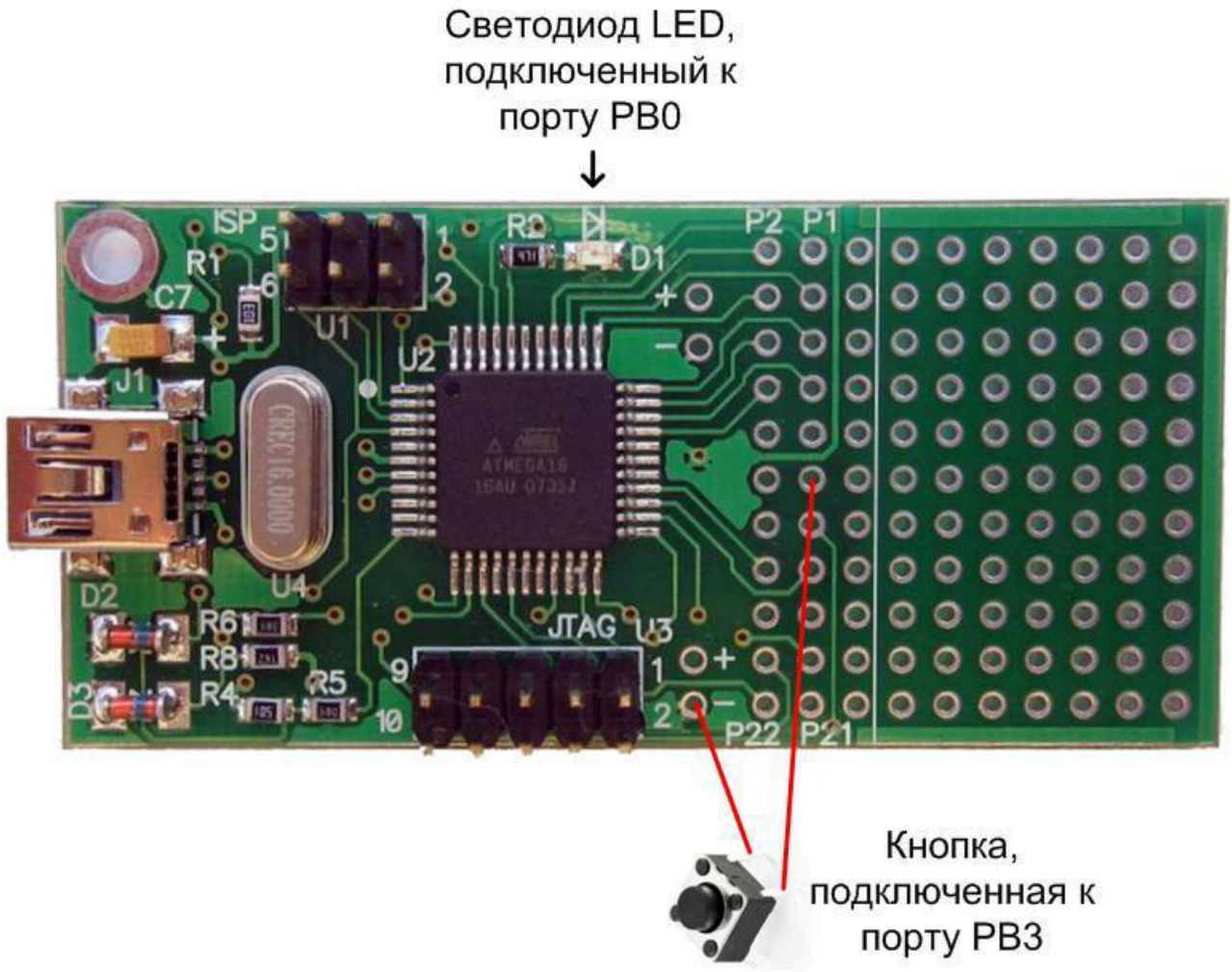
Обычно бит PUD не используют (он остается сброшенным) и настраивают подключением pull-up резисторов только через регистры DDRx. В таблице показано, как влияет на настройку порта бит PUD и биты PORTx, DDRx.

DDxn	**PORTxn**	**PUD (SFIOR)**	**I/O**	**pull-up**	**Пояснение**

0	0	X	Вход	Нет	Третье состояние (отключено, Hi-Z).
0	1	0	Вход	Да	Через порт Pxn будет течь ток через нагрузочный резистор, если на землю подключена внешняя цепь (нагрузка).
0	1	1	Вход	Нет	Третье состояние (отключено, Hi-Z).
1	0	X	Выход	Нет	Выход, замкнутый на землю (открыт нижний ключ буфера, верхний ключ закрыт).
1	1	X	Выход	Нет	Выход, замкнутый на плюс питания VCC (открыт верхний ключ буфера, нижний ключ закрыт).

[**Практический пример работы с портами GPIO макетной платы AVR-USB-MEGA16**]

Микроконтроллер может управлять электрическими сигналами (зажигать светодиоды, пищать динамиком, включать реле), получать сигналы из внешнего мира (например, нажатия кнопок, сигналы с датчиков). Для этого как раз и используются порты GPIO. В этом разделе приведен простейший для понимания пример таких подключений. К макетной плате AVR-USB-MEGA16 подключена кнопка к порту PB3, которую будет читать программа. Светодиод, подключенный к PB0, установлен на макетной плате.



Полный пример кода main.c, который иллюстрирует работу с портами микроконтроллера ATmega32A макетной платы AVR-USB-MEGA16:

```
#include "avr\io.h"
#include "avr\iom8.h"

int main(void)
{
   DDRB &= ~_BV(3);  //Сброс разряда 0 регистра DDRB в лог. 0 (PB3
                     // настроен как вход). Этот порт используется
                     // для чтения состояния кнопки. Кнопка подключена
                     // к ножке порта PB3 и к земле.
   PORTB |= _BV(3);  //Разрешить нагрузочный резистор на PB3.
   DDRB |= _BV(0);   //Установка разряда 0 регистра DDRB в лог. 1
                     // (PB0 настроен как выход).
   PORTB &= ~_BV(0); //Погасить светодиод, подключенный к PB0.
   while(1)
   {
      if (bit_is_clear(PINB, 3)) //Кнопка нажата?
      {
         //Да, кнопка нажата!
         PORTB |= _BV(0);                //Зажечь светодиод.
         loop_until_bit_is_set(PINB, 3); //Светодиод будет гореть,
                                         // пока кнопка нажата.
         PORTB &= ~_BV(0); //Погасить светодиод.
      }
   }
}
```

[**Как управлять портами микроконтроллера по USB**]

Мы уже разобрались, как программа микроконтроллера сама может управлять выводами портов - устанавливать в лог. 0 и лог. 1 (что можно использовать для управления лампочками, реле и другими периферийными устройствами), читать их состояние (можно использовать для принятия нажатия от кнопок, снятия информации с датчиков и различать другие события окружающего мира). Но как заставить компьютер передать команду через USB, чтобы микроконтроллер поменял состояние своей ножки/порта, или чтобы можно было считать в компьютер данные с GPIO, аналого-цифрового преобразователя, SPI или любого другого периферийного устройства микроконтроллера?

Ответ очевиден - нужно просто научиться наладить обмен данными между микроконтроллером и компьютером через USB. Если такой обмен будет налажен, то дальше уже дело техники - микроконтроллер сможет интерпретировать переданный ему байт как команду выполнить определенное действие (например, зажечь или погасить светодиод). И наоборот, микроконтроллер сможет передавать через USB какие-то данные, которые программа компьютера будет принимать и распознавать определенным образом.

Итак, вся загвоздка в том, как наладить обмен между микроконтроллером и компьютером через USB. Вопрос выбора практического решения этой задачи может оказаться весьма непрост - из-за того, что на рынке имеется очень много разных микроконтроллеров, и имеется очень много вариантов программных библиотек для обмена данными между устройствами USB. Но если уже имеются какие-то предпочтения, или выбрана модель

микроконтроллера или макетная плата, то выбрать уже проще. Здесь я кратко остановлюсь на возможных решениях для обмена данными по USB для макетных плат AVR-USB-MEGA16 и AVR-USB162. Далее для простоты программу для микроконтроллера буду называть **firmware** (эта программа будет работать как устройство **USB HID** или **USB CDC**), а программу на компьютере, которая обменивается данными с устройством USB, буду называть **ПО хоста**.

AVR-USB-MEGA16, работающая в качестве устройства USB

На этой макетной плате установлен микроконтроллер ATmega32A. Он не имеет на кристалле специально выделенного аппаратного контроллера USB. Поэтому протокол USB обрабатывается программно. В этом случае firmware микроконтроллера строится на основе популярной, многократно испытанной на деле библиотеке **V-USB**.

Написать firmware устройства USB HID для микроконтроллера на основе V-USB довольно просто, так имеется много хорошо документированных примеров таких устройств с открытым исходным кодом (как в составе самой библиотеки, так и в Интернете). Для ПО хоста V-USB предлагает примеры, написанные с помощью кроссплатформенной (Windows, Mac, Lunux) библиотеки **LibUSB**.

Для AVR-USB-MEGA16 есть также два готовых решения - [3] и [4], специально предназначенные для доступа (через USB из ПО хоста) не только к GPIO микроконтроллера, но и к его регистрам на чтение и запись. Благодаря этому можно получить полное управления над всеми портами микроконтроллера и его периферийными устройствами - можно управлять ножками и читать их значение, можно читать данные с аналого-цифрового преобразователя, можно пользоваться интерфейсом SPI, UART и проч.

[3] представляет из себя устройство класса USB HID, для которого не нужен драйвер. Однако из-за того, что для ПО хоста используется LibUSB, необходимо на компьютере установить так называемый драйвер фильтра - программную прослойку между LibUSB и периферийным устройством USB.

[4] - это устройство класса USB CDC (виртуальный COM-порт). ПО хоста, работающее с виртуальным COM-портом, можно написать довольно просто, так как есть много примеров программ и библиотек, которые передают и принимают данные через COM-порт. [4] организовано так, что содержит текстовый интерфейс команд, с помощью которых Вы обычной консоли (putty, TerraTerm, HyperTerminal и т. п.) можете управлять ножками микроконтроллера, читать их значение, можете получить доступ ко всем регистрам AVR. [5] содержит готовый пример (исходный код и скомпилированные бинарники, а также драйвер) firmware USB CDC, портированного на макетную плату AVR-USB-MEGA16.

Наладив обмен данными с AVR-USB-MEGA16 с помощью [3, 4, 5], Вы легко сможете управлять портами микроконтроллера через USB.

AVR-USB162, работающая в качестве устройства USB

Макетная плата AVR-USB162 (или её малогабаритный вариант AVR-USB162MU) выполнена на микроконтроллере AT90USB162. Этот микроконтроллер имеет в своем составе для работы с USB специальный аппаратный интерфейс, и для firmware используются уже другие библиотеки. Самые распространенные - LUFA [6] и библиотека Atmel для устройств AVR USB Series2 [7]. С помощью этих библиотек Вы легко сами можете создать собственное устройство USB HID или USB CDC.

Для ПО хоста USB HID можно использовать не только LibUSB, но и многие другие популярные библиотеки [8]. Наладив обмен данными с AVR-USB162 с помощью [7, 8], Вы легко сможете управлять портами микроконтроллера через USB.

[**Ссылки**]

1. Accessing AVR microcontroller ports with WinAVR GCC site:winavr.scienceprog.com.
2. Макетная плата AVR-USB-MEGA16.
3. AVR-USB-MEGA16: быстрая разработка USB приложений на C# при помощи класса-обертки ATMega16.
4. AVR-CDC: виртуальный COM-порт через Low-Speed USB (используется библиотека V-USB).
5. USB консоль для управления радиолюбительскими приборами.
6. LUFA - бесплатная библиотека USB для микроконтроллеров Atmel AVR.
7. AVR-USB162: где найти рабочие примеры кода firmware и ПО хоста.
8. Библиотеки для управления устройствами USB HID.
9. GPIO и альтернативные функции порта.