**AVR Урок 7. Кнопка**

**Урок 7**

**Кнопка**

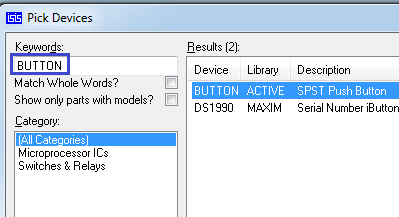
Сегодня мы расширим свой кругозор по изучению работы портов микроконтроллера и изучим второе назначение порта — работу на вход. И для изучения работы на вход мы применим обычную тактовую кнопку.

Как всегда, создадим проект в Atmel Studio, выберем Atmega8A, назовем проект Test04 и код также в main.c, как обычно, скопируем с проекта предыдущего урока.

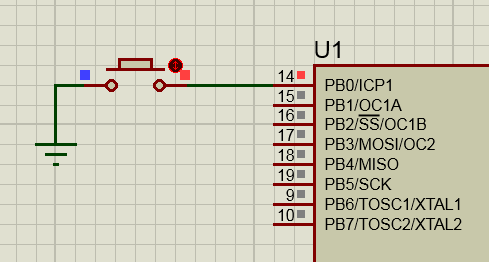
В качестве подопытного порта давайте возьмём порт B. Можно с успехом использовать любой порт. И в качестве ножки возьмем нулевую ножку. Итак у нас ножка B0.

Также опять мы соберём проект, скопируем и переименуем файл протеуса, откроем его и в свойствах контроллера покажем путь к новому проекту. Запустим на выполнение и убедимся, что всё работает.

 Добавим кнопку в протеусе, для этого в поиске компонентов найдём Button



Затем подключим нашу кнопку вот таким вот образом к ножке B0 контроллера



Судя по данной схеме, когда кнопка будет в нажатом состоянии, то Ножка PB0 у нас будет иметь низкий логический уровень, а при отжатом — непонятный уровень, поэтому мы применим подтягивающий к питанию резистор, который можно не паять физически, а подключить опционально определенной командой.

Для этого мы, во-первых настроим порт B. Мы можем объявить все ножки порта B на вход, так как нам не важны настройки остальных ножке, ибо мы их не используем

DDRD = 0xFF;

**DDRB = 0x00;**

В случае, когда мы работали с портом D на выход, биты регистра PORTD отвечали за уровень на соответствующих ножках. А в случае, когда порт инициализирован на вход, как наш порт B, то биты регистра PORTB будут уже отвечать за подтягивание к соответствующим ножкам порта резисторов на шину питания. Если будет логическая единица, то регистр будет подтягиваться, а если логический ноль — то не будет. Поэтому мы в 0 бите регистра установим 1

PORTD = 0b00000001;

**PORTB = 0b00000001;**

Соберём код и запустим его в протеусе. Мы видим, что на ножке **B0** у нас установилась логическая **1**, а если мы нажмём кнопку, то увидим, что на ней будет логический **0**, о чём свидетельствует синий цвет квадратиков на ножке и на кнопке.

В бесконечном цикле закомментируем весь код. В видеоверсии урока показано, как данную операцию можно выполнить одним движением (выделяем весь текст, который мы хотим закомментировать, затем нажимаем одновременно **Shift** и обычный слэш (не обратный))

// for(i=0;i<=7;i++)

// {

// PORTD = (1<<i);

// \_delay\_ms(500);

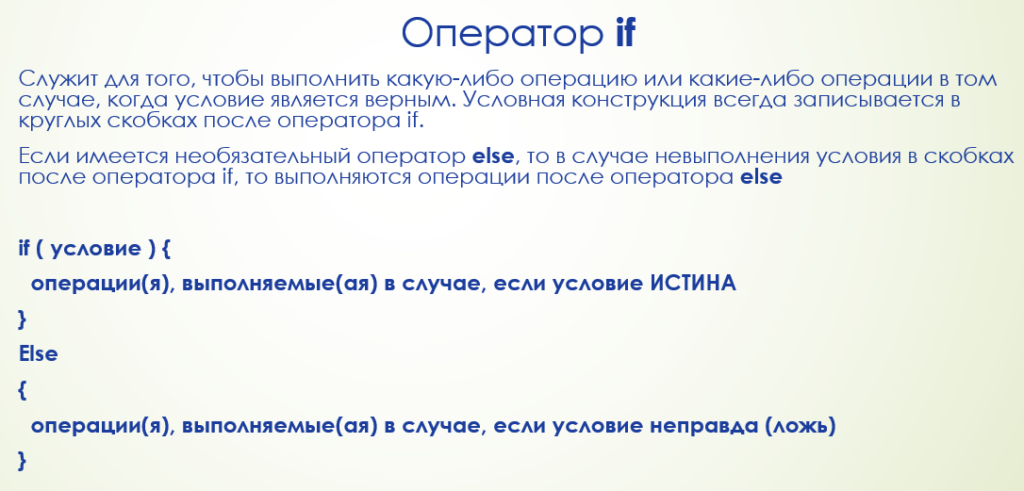
// }

}

В данном цикле мы и будем отслеживать состояние ножки PB0. Делается это с помощью определения состояния соответствующего бита в регистре PINB, который собственно за это и отвечает.

Чтобы нам следить за каким-либо действием или состоянием, нам необходимо будет обработать условие.

Условие в языке C добавляется с помощью команды **if**.



И в качестве условия мы возьмём состояние ножки 0 порта B или состояние бита 0 регистра **PINB**.

Как же можно получить состояние одного бита, ведь в языке C в отличие от ассемблера нет битовых операций?

Можно пойти на хитрость и применить вот такую конструкцию **PINB&0b00000001**.

Данная конструкция нам и проверит нулевой бит. То есть если в регистре PINB также будет 1 в нулевом его бите, то независимо от состояния остальных битов в данном регистре мы получим ненулевой результат, что также является истиной. То есть если ни с чем не сравнивать в условии результат, то условие эквивалентно сравниванием с нулём, только наоборот. Для истинности результат должен быть ненулевым — (**результат!=0**).

Но нам с вами наоборот нежелательно, чтобы ножка была в высоком логическом состоянии, так как кнопка у нас подключена к общему проводу. Поэтому мы должны поставить отрицание и написать код следующим образом

while(1)

{

**if (!(PINB&0b00000001))**

**{**

**}**

**else**

**{**

**}**

Теперь нам необходимо добавить тело условия. При выполнении условия, что кнопка нажата, мы будем зажигать светодиод на ножке D0. А если условие не будет выполняться (кнопка будет отжата), то мы будем его гасить. Также мы погасим данный светодиод и в начале программы. Поэтому получим следующий код

DDRB = 0x00;

PORTD = 0b0000000**0**;

PORTB = 0b00000001;

while(1)

{

  if (!(PINB&0b00000001))

  {

**PORTD = 0b00000001;**

  }

  else

  {

**PORTD = 0b00000000;**

  }

}

Теперь давайте пересоберём проект и пойдём в протеус смотреть, удалось ли нам что-то.

Чтобы у нас при сборке не было даже предупреждений, уберём объявление переменной i, так как она в коде не используется

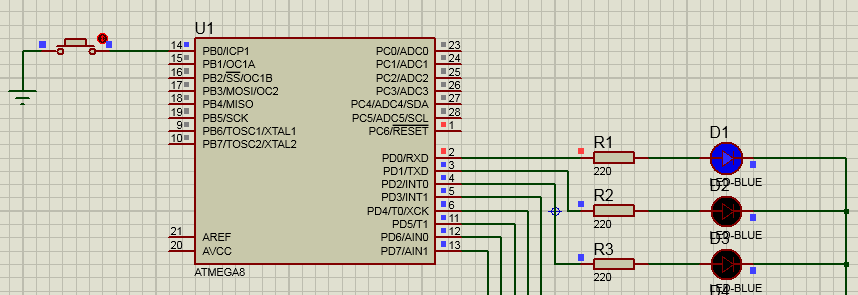
int main(void)

{

**// unsigned char i;**

    unsigned char butcount=0;

Запустим проект в протеусе и увидим, что при нажатии на кнопку у нас начинает светиться самый верхний светодиод



Казалось бы, что мы своей цели уже добились. Но чтобы сделать наш код более ответственным и совершенным, мы просто обязаны провести борьбу с дребезгом контактов, так как такое явление может иметь место, это только в протеусе всё идеально, на практике такое бывает не всегда.

И чтобы это как-то отследить и определить, что это было именно нажатие, а не дребезг, то мы будим отслеживать нажатие некоторое время, ну или некоторое количество тактов или циклов. Для этого в начале функции main() до бесконечного цикла мы добавим другую переменную (i нам ещё пригодится и мы её портить не будем). Назовём мы переменную **butcount**, так как имя переменной должно как-то само за себя говорить и тем самым достигается ещё большая читабельность кода

// unsigned char i;

**unsigned char butcount=0;**

И чтобы воспользоваться данной переменной, мы применим ещё одно условие. И у нас будет условие в условии. Это всё допустимо и очень широко используется. И в зависимости от этого условия мы данную переменную будем наращивать (инкрементировать). Условием будет у нас достижение данной переменной определённой величины. То есть попробуем сделать так, чтобы значение переменной не достигало **5**

if (!(PINB&0b00000001))

{

**if(butcount < 5)**

**{**

**butcount++;**

**}**

А когда значение данной переменной достигнет значения 5, то мы уже в данный цикл не попадём, а попадём мы в тело оператора **else**, который мы сейчас и добавим и в его теле напишем следующий код

  butcount++;

}

else

{

**PORTD = 0b00000001;**

}

То есть мы как раз после достижения пятёрки и будем обрабатывать нажатие кнопки и включать на нулевой ножке порта D высокое состояние.

По идее, здесь мы должны обнулить нашу переменную, но мы это будем делать также постепенно, используя тело оператора else, только другого — того, который у нас был и тело которого выполняется при низком уровне на ножке, к которой подключена кнопка. Вот таким будет его тело

else

{

**if (butcount > 0)**

**{**

**butcount—;**

**}**

**else**

**{**

**PORTD = 0b00000000;**

**}**

}

Данный код чем то похож на предыдущий, только здесь у нас идёт, наоборот декрементирование переменной, и как только её значение опять достигнет нуля, то мы и попадём в обработку отжатия кнопки, тем самым полностью избавимся от дребезга. И чем старее и некачественнее будет наша тактовая кнопка, тем большее значение переменной в условии мы будем применять.

Давайте теперь соберём проект и проверим его работу сначала в протеусе, а затем и на практике. Выглядит это приблизительно так. Интереснее конечно это смотреть в видеоуроке

