Лабораторная работа № 12

Тема: Работа с портами ввода/вывода микроконтроллера.
Цель работы: Ознакомление с основами ввода/вывода, настройкой портов микроконтроллера и осуществление базовых операций ввода/вывода.

План работы:

Краткая теория.

Справка по системам счисления. В проекте используются 10-я, 2-я и 16-я системы счисления.

Число в 10-й системе	Число в 2-й системе	Число в 16-й системе
0	0b00000000	0x00
1	0b00000001	0x01
10	0b00001010	0x0A
255	0b11111111	0xFF

Регистры портов:
DDRD = 0 // порт D настроен на прием сигнала
DDRD = 1 // порт D настроен на выход сигнала

Если DDR=0:
PORTD = 0 // подтягивающее сопротивление отключено
PORTD = 1 // подтягивающее сопротивление подключено

Если DDR=1:
PORTD = 0 // на выходе порта D устанавливается логический 0
PORTD = 1 // на выходе порта D устанавливается логическая 1

PIND – Чтение логических уровней разрядов порта D

ATmega8 имеет три порта ввода/вывода: PORTB, PORTC и PORTD.
Каждый порт состоит из восьми бит(пинов) (например, PB0-PB7 для PORTB).

Примеры:
Установить все пины порта на **выход** можно командой:

```
DDRD = 0xFF;            //в 16-ой системе
Или
DDRD = 0b11111111;   //в 2-ой системе
Или
DDRD = 255;   //в 10-ой системе
```

Установить первые 4 пина порта D на выход сигнала, остальные на прием можно командой:

```
DDRD = 0b00001111;
Или
DDRD = 0x0F;
```

Если режим порта – выход, установить логическую «1» (5В) на выходе 0-го пина порта D:

```
DDRD = 0b00000001;   // сначала укажем режим – выход для 0-го пина, остальные - вход
PORTD = 0b00000001; // теперь можно установить «1» на выходе 0-го пина
```

Или:

```
DDRD = 0x01;
PORTD |= (1 << PD0);   // PORTD |= (1 << 0); так тоже будет работать
```

`1 << PD0` – это битовый сдвиг влево операнда «1» на количество разрядов, заданное PD0. PD0 - это номер бита (пина) на порте D, в данном случае, это 0. Таким образом, (1 << PD0) создает число, у которого только 0-й бит установлен в 1, т.е. получим: 00000001.
`PORTD |=` - это операция "ИЛИ с присваиванием". Она берет текущее значение в регистре PORTD (содержащем состояния всех пинов порта D) и выполняет "ИЛИ" с числом, стоящим справа, и записывает результат обратно в переменную `PORTD`. Таким образом, устанавливается бит `PD0` в «1», а остальные биты остаются неизменными.

Пример кода, случайно зажигающего один из 8-ми светодиодов, подключенных к порту B:

```
1  //зажигание одного из восьми светодиодов в случайном порядке на порту B
2  #define F_CPU 8000000
3  #include <avr/io.h>
4  #include <stdlib.h>      // библиотека стандартных ф-й, в т.ч. rand()
5  #include <util/delay.h>  // библиотека задержек
6
7  int main(void)
8  {
9      // Настройка порта B как вывода
10     DDRB = 0xFF;
11     // Устанавливаем "0" на всех пинах
12     PORTB = 0x00;
13
14     while(1)
15     {
16         // ф-я rand() возвращает случайное целое число от 0 до 32767*
17         int num = rand() % 8;  // получаем случайное число в диапазоне 0..7 (% - остаток от деления)
18         PORTB |= (1 << num);   // Устанавливаем пин с индексом num в "1"
19         _delay_ms(500);        // ждем 0.5сек
20         PORTB = 0x00;          // сбрасываем все пины в "0"
21     }
22 }
```

Часто при работе с регистрами используются маски.
Например строка:
`DDRB &= ~(1 << DDB0);`
установит «0» в последнем бите регистра DDRB, а остальные биты останутся без изменения, разберем подробнее.
Допустим начальное значение такое:
`DDRB = 0d11001101;`
Выражение (1 << `DDB0`) создаст маску вида 00000001 («1» в позицию с индексом «0»)
`~` побитовое инвертирование, т.е. получим: 11111110
`&=` выполнить логическое **И** и записать результат в ту же переменную:
11001101 исходное значение
11111110 маска
11001100 результат

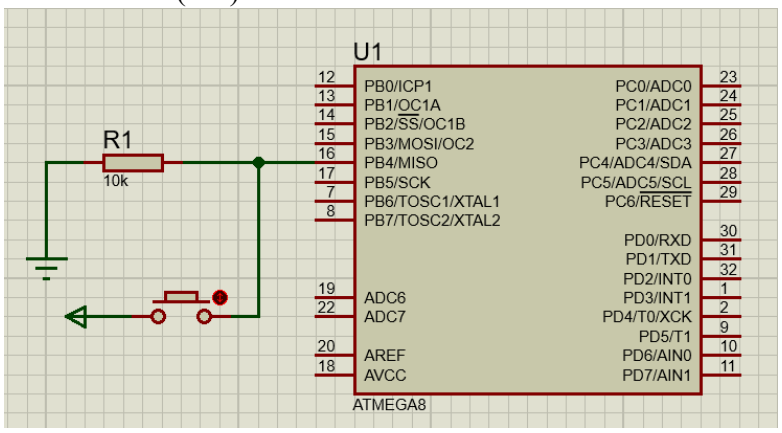
Если нужно установить «1» тогда строка будет выглядеть так:
`DDRB |= (1 << DDB0);`
Пример:
Допустим начальное значение такое:
`DDRB = 0d11001100;`
Выражение (1 << `DDB0`) создаст маску вида 00000001 («1» в позицию с индексом «0»)
`|=` выполнить логическое **ИЛИ** и записать результат в ту же переменную:
11001100 исходное значение
00000001 маска
11001101 результат

Если вывод сконфигурирован как выход, то единица в соответствующем бите регистра PORTx формирует на выводе сигнал высокого уровня, а ноль – формирует сигнал низкого уровня.
Если вывод сконфигурирован как вход, то единица в бите регистра PORTx подключает к выводу внутренний подтягивающий резистор, который обеспечивает высокий уровень на входе при отсутствии внешнего сигнала.

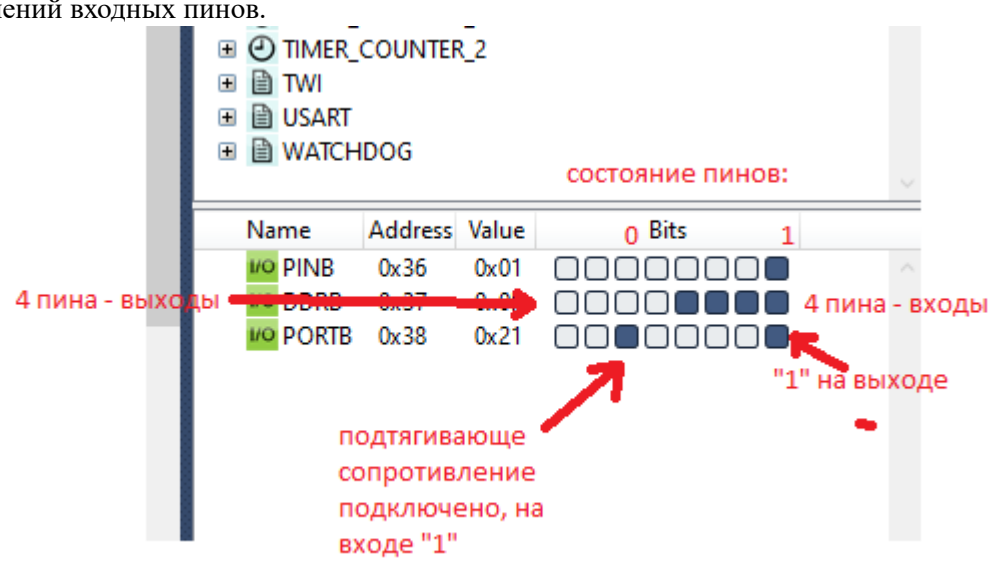
Пример:

```
//режим порта D - вход;
DDRD = 0x00;
//подтягиваем 0-й пин к «1»
PORTD |= (1 << PD0);
```

Если не используется внутренний подтягивающий резистор, то нужно использовать внешний (что бы не было неопределенного состояния), например можно «подтянуть» пин к земле («0»), а кнопкой подавать на него питание («1»):



Регистр PINx в микроконтроллерах AVR, таких как ATmega8, представляет собой регистр ввода, который содержит текущие состояния всех пинов соответствующего порта x. Этот регистр используется для чтения значений входных пинов.



Проверка состояния пина.
Пример.

```
// мл. 4 бита - выход, старшие 4 бита – вход
DDRB = 0x0F;
// подтягиваем 4-й пин к «1», т.е. к питанию. На нем будет всегда «1», если его не замкнуть на землю. Т.е. кнопка отжата – «1», кнопка нажата «0».
PORTB |= (1<<PB4);
```

```
// проверяем состояние пина, для этого используем регистр PINB
if (!(PINB & (1 << PINB4)))
```

(1 << PINB4): Это битовый сдвиг влево. **1** сдвигается на указанное количество позиций, например, для **PINB4**, результат будет **00010000**).
PINB & (1 << PINB0): Это битовая операция И между регистром **PINB** и полученной маской, созданной с помощью битового сдвига. Это позволяет извлечь текущее состояние пина PB4.
! – логическая операция **НЕ.**
00010000 // битовая маска `(1 << PINB4)` (**кнопка не нажата**, на пине «1»)
00010000 // состояние регистра PINB
00010000 // результат операции **И** (TRUE)
! – вернет FALSE

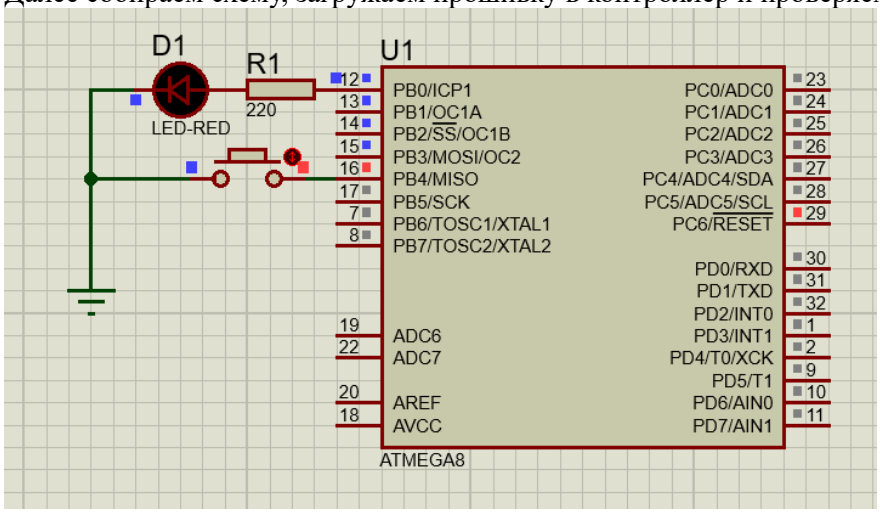
00000000 // битовая маска `(1 << PINB4)` (**кнопка нажата**, на пине «0»)
00010000 // состояние регистра PINB
00000000 // результат операции **И** (FALSE)
! – вернет TRUE

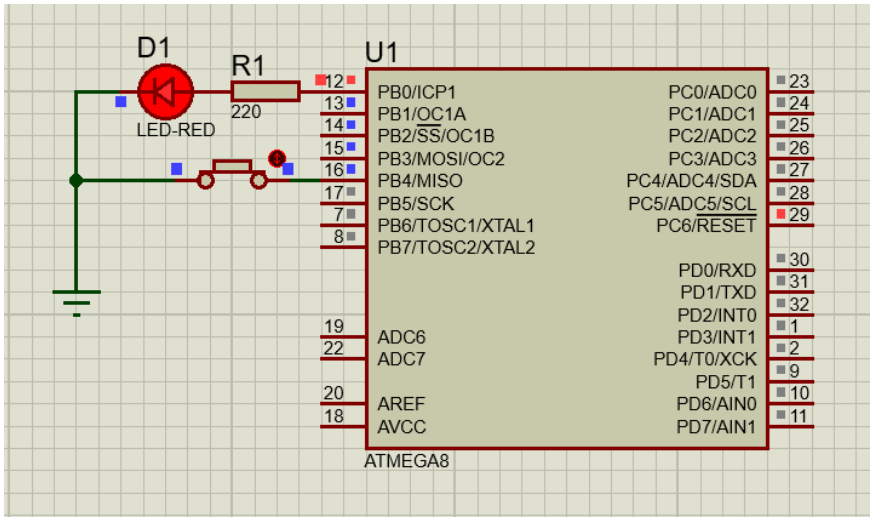
!(PINB & (1 << PINB0)): Это логическое отрицание. Если результат битовой операции И равен нулю (то есть пин PB0 находится в состоянии "нажат"), то логическое отрицание превращает это в **true**. Если результат битовой операции И равен единице (то есть пин PB0 находится в состоянии "не нажат"), то логическое отрицание превращает это в **false**.

Допишем код так, чтобы при нажатии кнопки на 4-м пине, на 0-м пине появлялась «1»

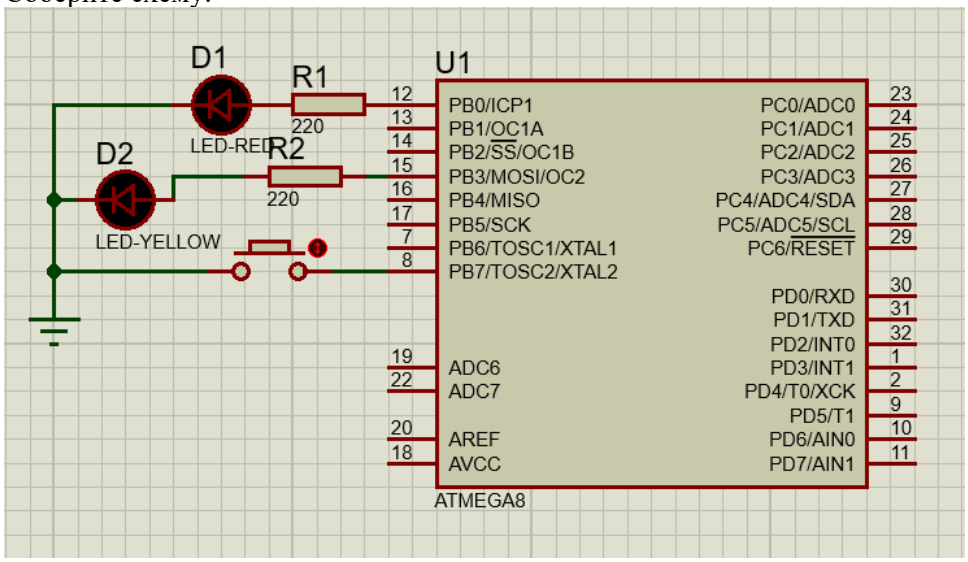
```
1  //зажигание светодиода при нажатии кнопки
2  #define F_CPU 8000000
3  #include <avr/io.h>
4
5  int main(void) {
6      // младшие 4 пина - выход, старшие 4 пина - вход
7      DDRB = 0x0F;
8      // подтягиваем 4-й пин к питанию
9      PORTB |= (1<<PB4);
10
11
12     while (1)
13         {
14         //проверяем состояние пина
15         if (!(PINB & (1 << PINB4))){
16             // если "0", включаем светодиод
17             PORTB |= (1<<PB0);
18             }
19         else {
20             // если "1", выключаем светодиод
21             PORTB &= ~(1<<PB0);
22             }
23         }
24     }
25
```

Далее собираем схему, загружаем прошивку в контроллер и проверяем:



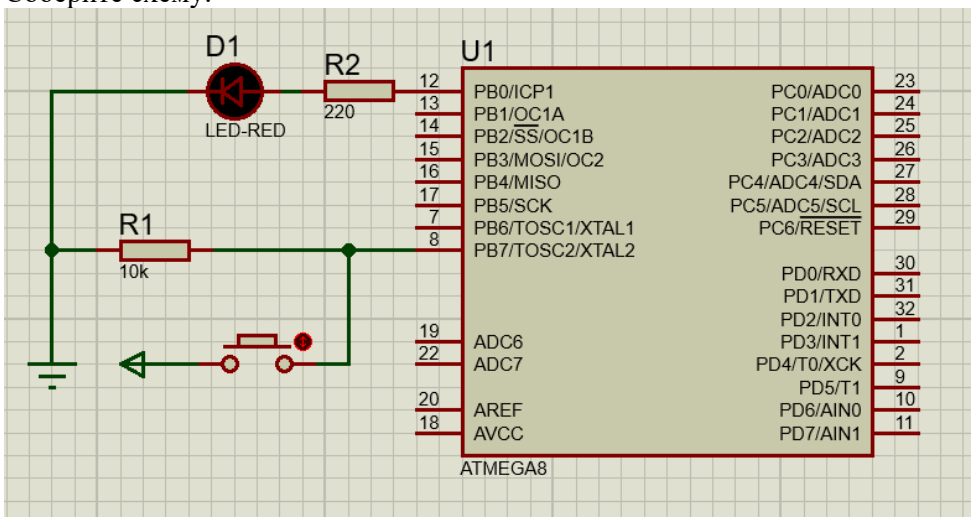


Задание 1:
Соберите схему:



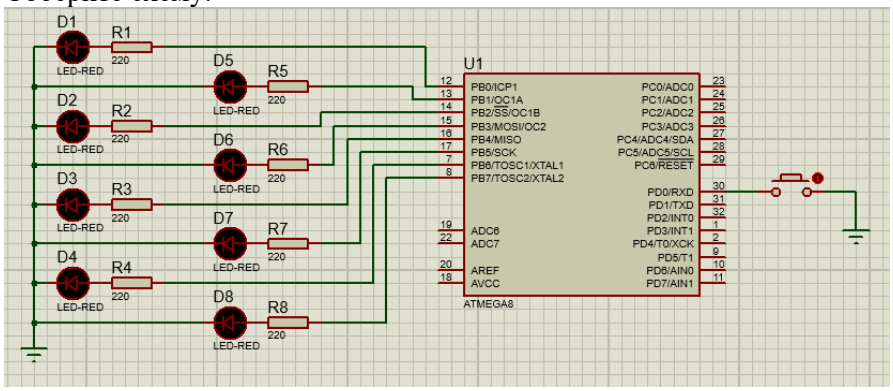
Измените код таким образом, чтобы при включении желтый светодиод горел, а красный нет. При нажатии кнопки красный должен загораться, а желтый гаснуть.

Задание 2.
Соберите схему:



Напишите код для микроконтроллера. При включении светодиод мигает с частотой 1Гц, при нажатии кнопки – 4Гц, при отпускании опять 1Гц.

Задание 3.
Соберите схему:



Напишите код, что бы при нажатии кнопки случайный светодиод мигал случайное количество раз (3..8). Напишите функцию для генерации случайных чисел в нужном диапазоне.

Отчет должен содержать:
- скриншот рабочего поля Proteus с собранной схемой;
- листинг исходного кода;
- выводы.