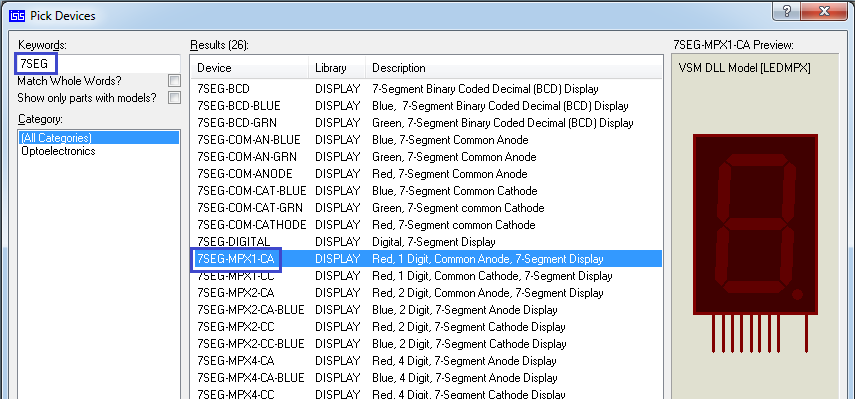
**AVR Урок 8. Семисегментный индикатор. Статическая индикация**

**Урок 8**

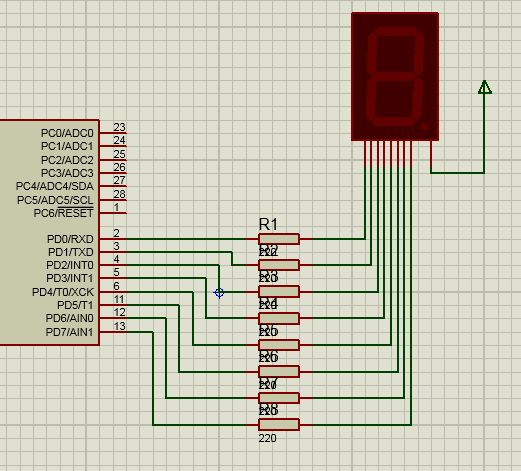
**Семисегментный индикатор. Статическая индикация**

Сегодня мы изучим, каким образом с помощью микроконтроллера **Atmega8a** можно управлять семисегментным светодиодным индикатором. Проект создадим также, как и обычно, назовём его Test05, удалим всё из main.c и скопируем туда содержимое одноименного файла из проекта прошлого занятия, также настроим симулятор в качестве программатора. Чтобы проверить, что в проекте всё нормально, попробуем его собрать. Также как и в прошлых занятиях, скопируем и переименуем файл протеуса, в свойствах контроллера там покажем путь к прошивке.

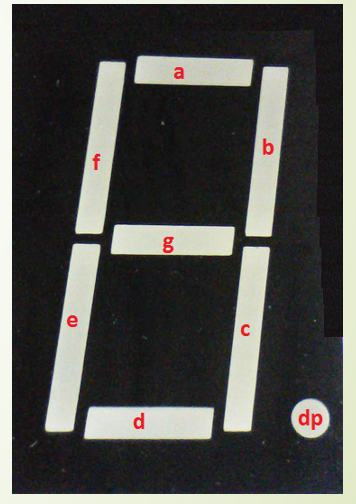
Удалим все светодиоды, так как они будут нам не нужны и найдём в библиотеке семисегментный светодиодный индикатор с общим анаодом. А с общим анодом мы возьмём из-за того, что они чаще встречаются, а также потому, что он у меня есть



Общий анод мы подключим к шине питания. И у нас сегменты будут светиться поэтому в том случае, когда на соответствующей ножке порта будет низкое напряжение, а когда высокое — не будут. Вот на этой основе мы и будем писать наш код. Подключаем вот так



Ну и прежде, чем писать какой-то код, давайте посмотрим, что из себя данный индикатор представляет и где и какой у него сегмент находится (принято обозначать сегменты латинскими буквами)



Поэтому, для того чтобы было легче данными сегментами управлять и выводить с помощью их свечения определённые цифры или символы, то мы соединим их по порядку — **a** с ножкой **D0**, **b** — с **D1** и так далее по порядку и будем в уме постоянно себе представлять вот такое вот двоичное число

image07

Скопируем данную строку в буфер обмена и добавим в наш код как комментарий, например вот таким образом

DDRB = 0x00;

PORTD = 0b00000000;      **// 1 2 3 4 5 6 7 8**

PORTB = 0b00000001; **//0b|dp|g|f|e|d|c|b|a|**

Обработчик кнопки мы трогать пока не будем, она нам возможно ещё пригодится и не будем поэтому портить проект.

Добавим следующий код в бесконечный цикл, сделав возможным отображение новой цифры через 500 милисекунд, используя функцию задержки

while(1)

{

**PORTD = ~0b00000110;**

**\_delay\_ms(500);**

**PORTD = ~0b01011011;**

**\_delay\_ms(500);**

**PORTD = ~0b01001111;**

**\_delay\_ms(500);**

**PORTD = ~0b01100110;**

**\_delay\_ms(500);**

**PORTD = ~0b01101101;**

**\_delay\_ms(500);**

**PORTD = ~0b01111101;**

**\_delay\_ms(500);**

**PORTD = ~0b00000111;**

**\_delay\_ms(500);**

**PORTD = ~0b01111111;**

**\_delay\_ms(500);**

**PORTD = ~0b01101111;**

**\_delay\_ms(500);**

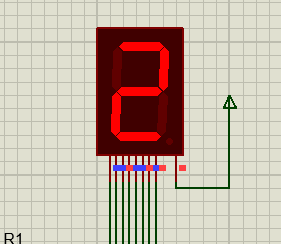
**PORTD = ~0b00111111;**

**\_delay\_ms(500);**

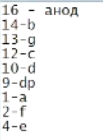
В коде мы видим что каждый раз мы заносим в порт новые значения, соответствующие определённой комбинации светящихся сегментов. Где ноль — там сегмент светится, где единица — нет.

А у нас-то наоборот — будут светиться единицы. Всё это достигается засчет операции инверсии. С точки зрения оптимизации лучше сделать всё-таки по первому варианту, но так как нам тут спешить некуда, то мы немного заодно повторим лишний раз пройденный материал по логическим операциям ибо они нам постоянно будут нужны.

Соберём код и проверим его сначала в протеусе



Код наш отлично работает. Давайте теперь посмотрим на живом контроллере и индикаторе. Для этого я взял ту же макетницу и собрал всё на ней, соединив с отладочной платой. У каждого отдельно-взятого индикатора схема распиновки своя. У моего она вот такая:



Прошьём контроллер и увидим, что всё у нас нормально работает (нагляднее конечно на видео, которое приложено внизу)

