**AVR Урок 13. ШИМ. Мигаем светодиодом плавно. Часть 1**

**ШИМ. Мигаем светодиодом плавно**

Сегодня мы изучим возможность использования **широтно-импульсной модуляции** в микроконтроллере **AVR**, или, как говорят в народе, **ШИМ**.

В технической документации мы будем видеть чаще аббревиатуру **PWM** или **pulse-width modulation**, что преводится имено также.

Что же такое вообще широтно-импульсная модуляция.

**ШИМ** — это управление свечением светодиодов, вращением двигателей, и прочими устройствами необычным способом, при котором данное управление осуществляется не приложенным напряжением к контактам, а квадратными импульсами. При этом напряжение будет только двух видов — высокое (**1**) и низкое (**0**). При данном способе результирующее напряжение вычисляется как среднее по времени между временем высокого состояния в одном импульсе и временем низкого состояния. Мы вычисляем отношение времени (или широты) высокого состояния к общему периоду импульса. Называем мы это скважностью импульса. То есть чем больше в периоде напряжение находилось в высоком состоянии, тем больше скважность, а, следовательно, тем больше и результирующее среднее напряжение. То есть, чтобы найти результирующее напряжение, нам необходимо и достаточно вычисленную скважность умножить на напряжение и разделить на 100, так как скважность как правило измеряется в процентах. Например, если у нас в квадратном импульсе широта логического нуля равна широте логической единицы, то скважность у нас будет 50 процентов, и, если напряжение будет 5 вольт, то среднее результирующее напряжение мы получим равное 2,5 вольт и т.д. Лучшую картину объяснения данной ситуации мы можем увидеть, посмотрев видеоурок, ссылка на который дана в конце данной статьи.

Это конечно очень упрощённое понятие ШИМ. Есть более серьёзные разъяснение данной технологии, но нам для наших экспериментов этого будет вполне достаточно.

То есть, подведя итоги объяснению, мы управляем результирующим напряжением, а также и свечением светодиода, угловой скоростью электродвигателя и прочими значениями за счёт изменения скважности импульсов.

Но всё-таки самое интересное, как же всё-таки всё это организовано в нашем микроконтроллере?

В микроконтроллере **AVR широтно-импульсную модуляцию** можно организовать как **программно**, так и **аппаратно**.

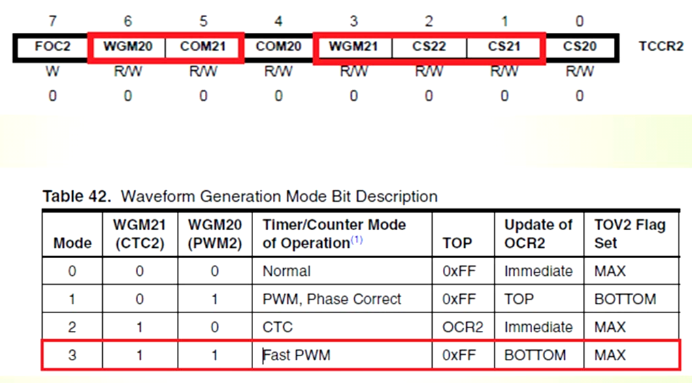
**Программная**организация **ШИМ**— это когда мы включим на определённое время на ножке контроллера логическую единицу, а затем на определённое время — логический ноль и так по циклу. Плюсы данного способа — это то, что мы можем организовать **ШИМ**на абсолютно любой ножке любого порта контроллера, а минусом — то, что всё это будет связано с немалыми затратами на процессорное время, и возможно даже будет сопряжено с какими-то ошибками, вытекающими из этого.

Поэтому всегда в любой технологии мы стараемся придерживаться всё-таки именно аппаратного способа реализации.

**Аппаратная**организация **ШИМ**в МК **AVR** происходит на уровне **таймера 2**.

Как мы помним из предыдущего занятия, таймеров в конкретном нашем микроконтроллере Atmega8 три. И таймеры 0 и 2 являются восьмибитными. Но на уровне 2-го таймера как раз и организована широтно-импульсная модуляция.

Давайте посмотрим вот такую вот картину



Здесь иллюстрируется то, какие мы должны включить биты в регистре **TCCR** второго таймера, чтобы запустить широтно-импульсную модуляцию.

Биты WGM как раз и включают таймер в резим PWM (ШИМ). С остальными битами разберёмся позже, если это потребуется.

А сейчас, чтобы это всё прочувствовать, нам необходим проект, который мы и создадим сейчас.

Проект создаем таким же образом, как и на прошлых занятиях. Назовём его Test10, создадим и добавим файлы PWM.c и PWM.h, последний автоматически оформится при создании соответствующими директивами прероцессора. А в главном файле Test01.c мы напишем стандартный код

#include «main.h»

//—————————————-

void port\_ini(void)

{

PORTB=0x00;

DDRB=0x08;

}

//—————————————-

int main(void)

{

  port\_ini();

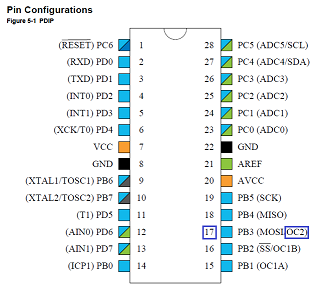
  while(1)

  {

  }

}

Как мы видим в коде, в порте B мы включили на выход 3 ножку. В распиновке контроллера мы видим, что у данной ножки есть ещё альтернативное обозначение **OC2**, которое и означает возможность этой ножки работать непосредственно с ШИМ



Файл main.h мы можем даже подключить с прошлого занятия по LCD и немного исправим код, подключив туда уже новый наш модуль PWM

#ifndef MAIN\_H\_

#define MAIN\_H\_

#define F\_CPU 8000000UL

#include <avr/io.h>

#include <avr/interrupt.h>

#include <util/delay.h>

#include <stdio.h>

#include <stdlib.h>

**#include «PWM.h»**

#endif /\* MAIN\_H\_ \*/

Ну и теперь начнём писать код в новую библиотеку для работы непосредственно с ШИМ — в файл PWM.c.

Там мы также подключим заголовочный файл и создадим каркас функции для работы с ШИМ

**#include «PWM.h»**

**void init\_PWM\_timer(void)**

**{**

**}**

А в заголовочный файл нашей библиотеки мы добавим прототип данной функции, а также подключим библиотеку для работы с прерываниями

#ifndef PWM\_H\_

#define PWM\_H\_

**#include <avr/interrupt.h>**

**void init\_PWM\_timer(void);**

#endif /\* PWM\_H\_ \*/

И вызовем сразу эту функцию в main()

port\_ini();

**init\_PWM\_timer();**

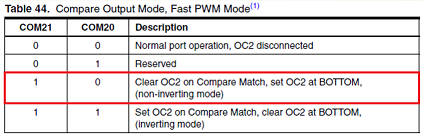
Начнём теперь писать код непосредственно в функцию. Сначала запишем все нули в регистр **ASSR**, который существует у таймера, но мы его никак не используем

void init\_PWM\_timer(void)

{

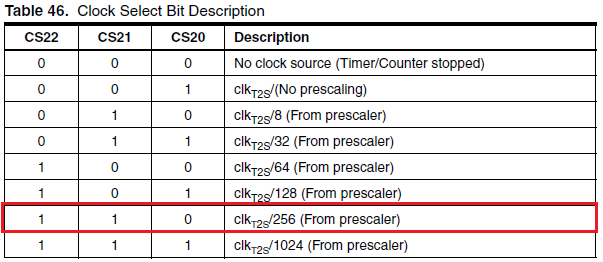
**ASSR=0x00;**

Дальше уже займёмся управляющим регистром нашего таймера. Назначение битов WGM и то, какие именно из них мы включим, мы разобрали. Теперь биты COM20 и COM21, отвечающие за режим самого ШИМ. Мы выберем вот такой режим и соответственно ему и включим данные биты



А включим мы режим, при котором при начале счёта напряжение на ножке OC2 будет находиться в высоком логическом состоянии, а как только мы досчитаем до определённой цифры, то ножка перейдёт в низкое состояние и будет в нём находиться до окончания счёта. И так по кругу.

И останется нам только настроить делитель. Слишком большая частота нам не нужна. Но когда была слишком маленькая, также было заметно мерцание, поэтому давайте включим следующую величину



Поэтому мы включим все три бита.

В итоге значение регистра станет вот таким

ASSR=0x00;

**TCCR2=0b01101110; //Fast PWM, Clear OC2 on Compare Match, clkT2S/256 (From prescaler) (реальная частота получится 8мгц/256 = 31250 гц)**

Теперь попробуем собрать проект. Проект нормально собрался. В следующей части мы продолжим работать с ШИМ.

**AVR Урок 13. ШИМ. Мигаем светодиодом плавно. Часть 2**

**ШИМ. Мигаем светодиодом плавно**

Продолжаем работать с функцией, отвечающей за работу с ШИМ.

Дальше заносим 0 в регистр счёта **TCNT2**, то есть обнуляем счётчик

TCCR2=0b01101110; //Fast PWM, Clear OC2 on Compare Match, clkT2S/256 (From prescaler) (реальная частота получится 8мгц/256 = 31250 гц)

**TCNT2=0x00; // Timer Value = 0**

Далее проинициализируем регистр **OCR2**, который как раз и задаёт значение скважности. Пока зададим ноль, а в основном коде уже будем задавать другие нужные нам значения. То есть при нуле практически импульсов у нас не будет вообще

TCNT2=0x00; // Timer Value = 0

**OCR2=0x00; //Output Compare Register = 0 — в нашем случае это скважность шим (0 — это 0, а FF — 100 %)**

Регистр **TIMSK** также символически не трогаем, сложив его по "ИЛИ" с нулём, что вообще ни к чему не приводит, просто мы демонстративно покажем, что мы его не трогаем

  OCR2=0x00; //Output Compare Register = 0 — в нашем случае это скважность шим (0 — это 0, а FF — 100 %)

**TIMSK|=0x00;**

}

Перейдём в главный модуль Test10.c и в функции main() установим какую-нибудь скважность, например 50

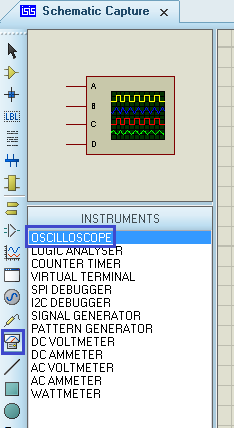
init\_PWM\_timer();

**OCR2 = 0;**

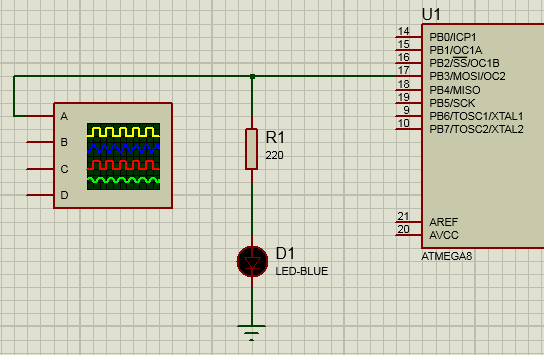
Ну конечно же это будет не 50 процентов, 50/255\*100, что составит приблизительно 19,6 процентов.

Соберём код.

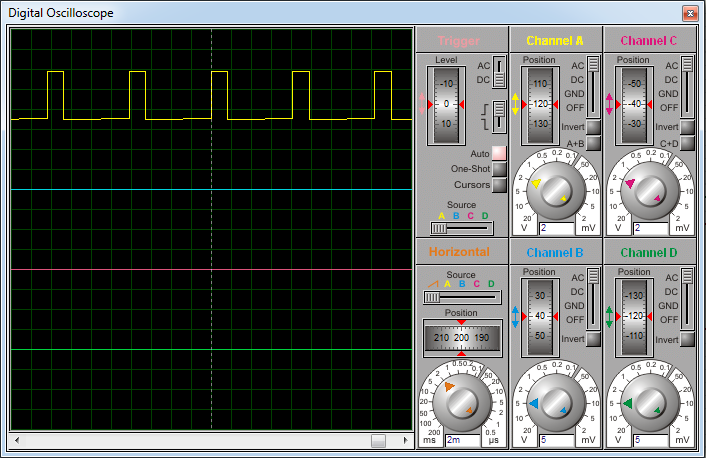
Для протеуса скопируем и один из первых проектов и подключим через токоограничивающий резистор светодиод к ножке 19, через которую мы и управляем ШИМом. Также подключим измерительный прибор вот здесь



и подключим его к той же ножке



Запустим проект в протеусе и получим вот такую картину



Теперь давайте попробуем сочинить код в бесконечный, который будет управлять плавно скважностью от 0 до 100 процентов, ну а, соответственно OCR2 мы будем менять от 0 до 255. Я думаю, используя знания прошлых занятий, нам уже будет это сделать очень несложно. Сначала добавим переменную для статуса нашего ШИМ. Будет 2 вида статуса — нисходящий и восходящий — 0 и 1.

int main(void)

{

**unsigned char pwm\_state=0;**

  port\_ini();

Ну, а теперь, собственно, цикл

while(1)

{

**if (pwm\_state==0)**

**{**

**OCR2++;**

**if (OCR2>254)**

**{**

**pwm\_state=1;**

**}**

**}**

**if (pwm\_state==1)**

**{**

**OCR2—;**

**if (OCR2<1)**

**{**

**pwm\_state=0;**

**}**

**}**

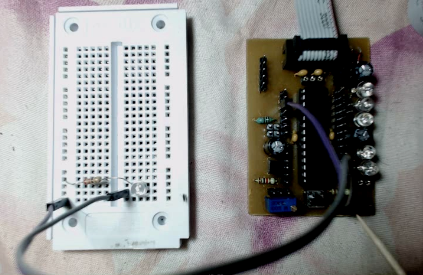
**\_delay\_ms(3);**

}

Благодаря такому коду у нас скважность будет плавно то прибавляться, то убавляться.

Соберем код и попробуем пока посмотреть в протеусе, как будет меняться скважность. В рамках текстовой версии занятия мы данного эффекта не увидим, поэтому в этом случае лучше посмотреть видеоурок, так как осцилограмма ведёт себя очень красиво.

Теперь попробуем подключить контроллер со светодиодом вот таким вот образом



 и прошьём контроллер. Приблизительно такой будет эффект

