**AVR Урок 10. Таймеры-счетчики. Прерывания**

**Урок 10**

**Таймеры-счетчики. Прерывания**

Сегодня мы узнаем, что такое **таймеры-счётчики** в микроконтроллерах и для чего они нужны, а также что такое **прерывания** и для чего они тоже нужны.

**Таймеры-счётчики** — это такие устройства или модули в микроконтроллере, которые, как видно из названия, постоянно что-то считают. Считают они либо до определённой величины, либо до такой величины, сколько они битности. Считают они постоянно с одной скоростью, со скоростью тактовой частоты микроконтроллера, поправленной на делители частоты, которые мы будем конфигурировать в определённых регистрах.

И вот эти таймеры-счётчики постоянно считают, если мы их инициализируем.

Таймеров в МК **Atmega8** три.

image00

Два из них — это **восьмибитные** таймеры, то есть такие, которые могут максимально досчитать только до 255. Данной величины нам будет маловато. Даже если мы применим максимальный делитель частоты, то мы не то что секунду не отсчитаем, мы даже полсекунды не сможем посчитать. А у нас задача именно такая, чтобы досчитывать до 1 секунды, чтобы управлять наращиванием счёта светодиодного индикатора. Можно конечно применить ещё наращивание переменной до определенной величины, но хотелось бы полностью аппаратного счёта.

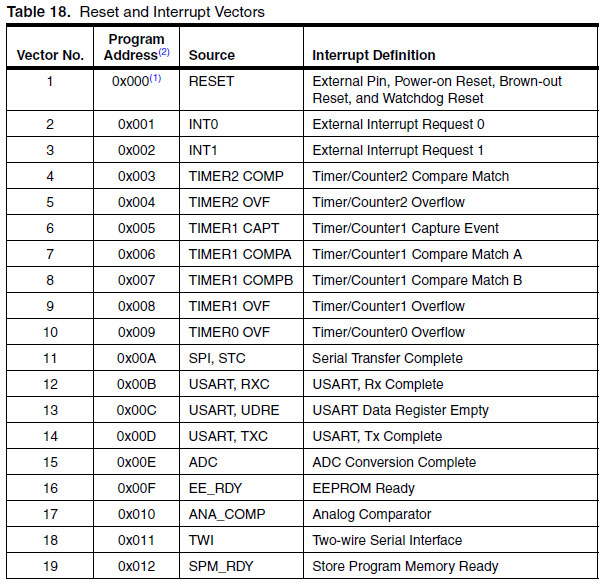
Но есть ещё один таймер — это полноправный **16-битный**таймер. Он не только **16-битный**, но есть в нём ещё определённые прелести, которых нет у других таймеров. С данными опциями мы познакомимся позже.

Вот этот 16-битный таймер мы и будем сегодня изучать и использовать. Также, познакомившись с данным таймером, вам ничего не будет стоить самостоятельно изучить работу двух других, так как они значительно проще. Но тем не менее 8-битные таймеры в дальнейшем мы также будем рассматривать, так как для достижения более сложных задач нам одного таймера будет недостаточно.

Теперь коротко о прерываниях.

**Прерывания** (**Interrupts**) — это такие механизмы, которые прерывают код в зависимости от определённых условий или определённой обстановки, которые будут диктовать некоторые устройства, модули и шины, находящиеся в микроконтроллере.

В нашем контроллере Atmega8 существует 19 видов прерываний. Вот они все находятся в таблице в технической документации на контроллер

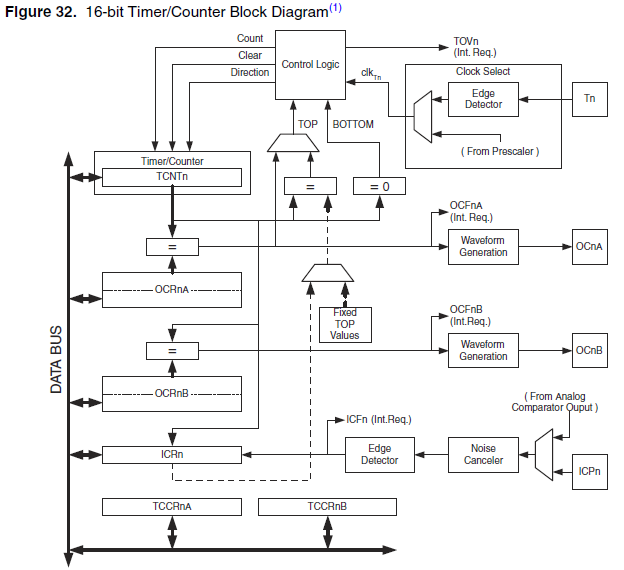


Какого типа могут быть условия? В нашем случае, например, досчитал таймер до определённой величины, либо например в какую-нибудь шину пришёл байт и другие условия.

На данный момент мы будем обрабатывать прерывание, которое находится в таблице, размещённой выше на 7 позиции — **TIMER1 COMPA**, вызываемое по адресу 0x006.

Теперь давайте рассмотрим наш 16-битный таймер или **TIMER1**.

Вот его структурная схема



Мы видим там регистр **TCNTn**, в котором постоянно меняется число, то есть оно постоянно наращивается. Практически это и есть счётчик. То есть данный регистр и хранит число, до которого и досчитал таймер.

А в регистры **OCRnA** и **OCRnB** (буквы n — это номер таймера, в нашем случае будет 1) — это регистры, в которые мы заносим число, с которым будет сравниваться чило в регистре TCNTn.

Например, занесли мы какое-нибудь число в регистр OCRnA и как только данное число совпало со значением в регистре счёта, то возникнет прерывание и мы его сможем обработать. Таймеры с прерываниями очень похожи на обычную задержку в коде, только когда мы находимся в задержке, тознаменатель

мы в это время не можем выполнять никакой код (ну опять же образно "мы", на самом деле АЛУ). А когда считает таймер, то весь код нашей программы в это время спокойно выполняется. Так что мы выигрываем колоссально, не давая простаивать огромным ресурсам контроллера по секунде или даже по полсекунды. В это время мы можем обрабатывать нажатия кнопок, которые мы также можем обрабатывать в таймере и многое другое.

Есть также регистр TCCR. Данный регистр — это регистр управления. Там настраиваются определенные биты, отвечающие за конфигурацию таймера.

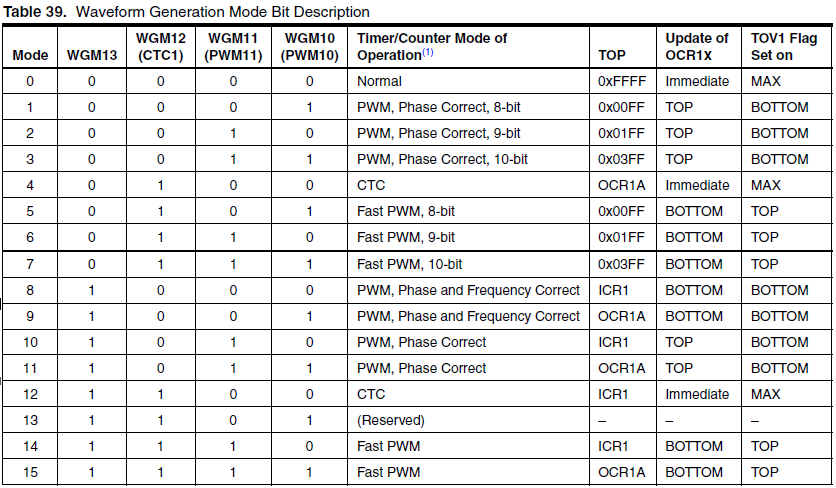
Также у таймера существует несколько режимов, с которыми мы также познакомимся немного позденее.

Он состоит из двух половинок, так как у нас конотроллер 8-битный и в нем не может быть 16-битных регистров. Поэтому в одной половинке регистра (а физически в одном регистре) хранится старшая часть регистра, а в другом — младшая. Можно также назвать это регистровой парой, состоящей из двух отдельных регистров TCCR1A и TCCR1B. Цифра 1 означает то, что регистр принадлежит именно таймеру 1.

Даный регист TCCR отвечает за установку делителя, чтобы таймер не так быстро считал, также он отвечает (вернее его определённые биты) за установку определённого режима.

За установку режима отвечают биты WGM

 знаменатель



Мы видим здесь очень много разновидностей режимов.

**Normal** — это обычный режим, таймер считает до конца.

**PWM** — это **ШИМ** только разные разновидности, то есть таймер может играть роль *широтно-импульсного модулятора*. С данной технологией мы будем знакомиться в более поздних занятиях.

**CTC** — это сброс по совпадению, как раз то что нам будет нужно. Здесь то и сравнивются регистры TCNT и OCR. Таких режима два, нам нужен первый, второй работает с другим регистром.

Все разновидности режимов мы в данном занятии изучать не будем. Когда нам эти режимы потребуются, тогда и разберёмся.

Ну давайте не будем томить себя документацией и наконец-то попробуем что-то в какие-нибудь регистры занести.

Код, как всегда, был создан из прошлого проекта. Для протеуса также код был скопирован и переименован с прошлого занятия, также в свойствах контроллера был указан путь к новой прошивке. Проекты мы назовем **Test07**.

Попробуем как всегда скомпилировать код и запустить его в протеусе. Если всё нормально работает, то начинаем добавлять новый код.

Добавим ещё одну функцию, благо добавлять функции мы на прошлом занятии научились. Код функции разместим после функции segchar и до функции main. После из-за того, что мы будем внутри нашей новой функции вызывать функцию segchar.

Мало того, мы создадим не одну функцию, а целых две. В одну функцию мы разместим весь код инициализации нашего таймеру, а другая функция будет являться обработчиком прерывания от таймера, а такие функции они специфичны и вызывать их не требуется. Когда возникнет необходимость, они вызовутся сами в зависимости от определённых условий, которые были оговорены выше.

Поэтому первую функцию мы назвовём **timer\_ini**

**//———————————————**

**void timer\_ini(void)**

**{**

**}**

**//———————————————**

Также давайте наши функции, а также какие-то законченные блоки с объявлением глобальных переменных, с прототипами функций будем отделять друг от друга вот такими чёрточками, которые за счет наличия двух слешей впереди компилятор обрабатывать не будет и примет их за комментарии. За счёт этих отчерчиваний мы будем видеть, где заканчивается одна функция и начинается другая.

Данная функция, как мы видим не имеет ни каких аргументов — ни входных, не возвращаемых. Давайте сразу данную функцию вызовем в функции main()

unsigned char butcount=0, butstate=0;

**timer\_ini();**

Теперь мы данную функцию начнём потихонечку наполнять кодом.

Начнем с регистра управления таймером, например с TCCR1B. Используя нашу любимую операцию "ИЛИ", мы в определённый бит регистра занесём единичку

void timer\_ini(void)

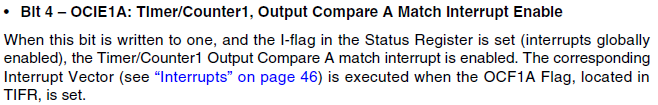
{

**TCCR1B |= (1<<WGM12); // устанавливаем режим СТС (сброс по совпадению)**

Из комментария мы видим, что мы работает с битами режима, и установим мы из них только бит WGM12, остальные оставим нули. Исходя из этого мы сконфигурировали вот такой режим:

image04

Также у таймера существует ещё вот такой регистр — **TIMSK**. Данный регистр отвечает за маски прерываний — **Interrupt Mask**. Доступен данный регистр для всех таймеров, не только для первого, он общий. В данном регистре мы установим бит **OCIE1A**, который включит нужный нам тип прерывания **TIMER1 COMPA**



TCCR1B |= (1<<WGM12); // устанавливаем режим СТС (сброс по совпадению)

**TIMSK |= (1<<OCIE1A); //устанавливаем бит разрешения прерывания 1ого счетчика по совпадению с OCR1A(H и L)**

Теперь давайте поиграемся с самими регистрами сравнения **OCR1A(H и L)**. Для этого придётся немного посчитать. Регистр **OCR1AH** хранит старшую часть числа для сравнения, а регистр **OCR1AL** — младшую.

Но прежде чем посчитать, давайте пока напишем код с любыми значениями данного регистра и потом поправим, так как дальше мы будем инициализировать делитель и он тоже будет учавствовать в расчёте требуемого времени счёта. Без делителя таймер будет слишком быстро считать.

TIMSK |= (1<<OCIE1A); //устанавливаем бит разрешения прерывания 1ого счетчика по совпадению с OCR1A(H и L)

**OCR1AH = 0b10000000; //записываем в регистр число для сравнения**

**OCR1AL = 0b00000000;**

**TCCR1B |= ();//установим делитель.**

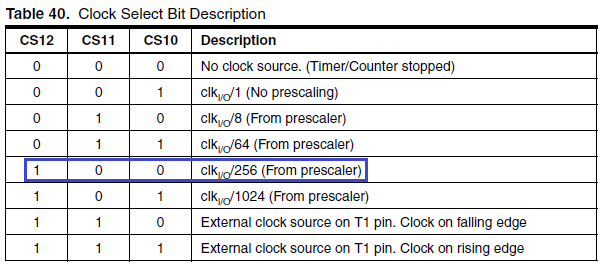
Пока никакой делитель не устанавливаем, так как мы его ещё не посчитали. Давайте мы этим и займёмся.

Пока у нас в регистре **OCR1A** находится число 0b1000000000000000, что соответствует десятичному числу 32768.

Микроконтроллер у нас работает, как мы договорились, на частоте 8000000 Гц.

Разделим 8000000 на 32768, получим приблизительно 244,14. Вот с такой частотой в герцах и будет работать наш таймер, если мы не применим делитель. То есть цифры наши будут меняться 244 раза в секунду, поэтому мы их даже не увидим. Поэтому нужно будет применить делитель частоты таймера. Выберем делитель на 256. Он нам как раз подойдёт, а ровно до 1 Гц мы скорректируем затем числом сравнения.

Вот какие существуют делители для 1 таймера



Я выделил в таблице требуемый нам делитель. Мы видим, что нам требуется установить только бит **CS12**.

Так как делитель частоты у нас 256, то на этот делитель мы поделим 8000000, получится 31250, вот такое вот мы и должны занести число в TCNT. До такого числа и будет считать наш таймер, чтобы досчитать до 1 секунды. Число 31250 — это в двоичном представлении 0b0111101000010010. Занесём данное число в регистровую пару, и также применим делитель

OCR1AH = 0b**01111010**; //записываем в регистр число для сравнения

OCR1AL = 0b**00010010**;

TCCR1B |= (**1<<CS12**);//установим делитель.

С данной функцией всё.

Теперь следующая функция — обработчик прерывания от таймера по совпадению. Пишется она вот так

**ISR (TIMER1\_COMPA\_vect)**

**{**

**}**

И тело этой функции будет выполняться само по факту наступления совпадения чисел.

Нам нужна будет переменная. Объявим её глобально, в начале файла

#include <util/delay.h>

//———————————————

**unsigned char i;**

//———————————————

Соответственно, из кода в функции main() мы такую же переменную уберём

int main(void)

{

~~unsigned char i;~~

Также закомментируем весь код в бесконечном цикле. Его роль теперь у нас будет выполнять таймер, и, я думаю, он с этим справится не хуже, а даже лучше, "никому" при этом не мешая.

**while(1)**

**{**

**// for(i=0;i<10;i++)**

**// {**

**//   while (butstate==0)**

**//   {**

**//     if (!(PINB&0b00000001))**

**//     {**

**//       if(butcount < 5)**

**//       {**

**//         butcount++;**

**//       }**

**//       else**

**//       {**

**//         i=0;**

**//         butstate=1;**

**//       }**

**//     }**

**//     else**

**//     {**

**//       if(butcount > 0)**

**//       {**

**//         butcount—;**

**//       }**

**//       else**

**//       {**

**//         butstate=1;**

**//       }**

**//     }**

**//   }**

**//   segchar(i);**

**//   \_delay\_ms(500);**

**//   butstate=0;**

**// }**

}

Теперь, собственно, тело функции-обработчика. Здесь мы будем вызывать функцию segchar. Затем будем наращивать на 1 переменную **i**. И чтобы она не ушла за пределы однозначного числа, будем её обнулять при данном условии

ISR (TIMER1\_COMPA\_vect)

{

**if(i>9) i=0;**

**segchar(i);**

**i++;**

}

Теперь немного исправим код вначале функции main(). Порт **D**, отвечающий за состояние сегментов, забьём единичками, чтобы при включении у нас не светился индикатор, так как он с общим анодом. Затем мы здесь занесём число 0 в глобавльную переменную i, просто для порядка. Вообще, как правило, при старте в неициализированных переменных и так всегда нули. Но мы всё же проинициализируем её. И, самое главное, чтобы прерывание от таймера работало, её недостаточно включить в инициализации таймера. Также вообще для работы всех прерываний необходимо разрешить глобальные прерывания. Для этого существует специальная функция **sei() — Set Interrupt**.

Теперь код будет вот таким

DDRB = 0x00;

PORTD = 0b**11111111**;

PORTB = 0b00000001;

**i=0;**

**sei();**

while(1)

Также ещё мы обязаны подключить файл библиотеки прерываний вначале файла

 знаменатель

#include <avr/io.h>

**#include <avr/interrupt.h>**

#include <util/delay.h>

Также переменные для кнопки нам пока не потребуются, так как с кнопкой мы сегодня работать не будем. Закомментируем их

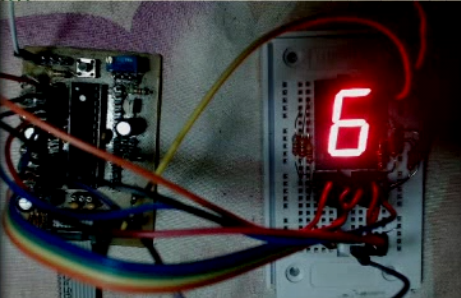
int main(void)

{

**//unsigned char butcount=0, butstate=0;**

timer\_ini();

Соберём наш код и проверим его работоспособность сначала в протеусе. Если всё нормально работает, то проверим также в живой схеме



Всё у нас работает. Отлично!

Вот такой вот получился секундомер. Но так как у нас даже нет кварцевого резонатора, то данный секундомер нельзя назвать точным.

Тем не менее сегодня мы с вами много чему научились. Мы узнали о прерываниях, также научились их обрабатывать, Научились работать с таймерами, конфигурировать несколько новых регистров микроконтроллера, до этого мы работали только с регистрами портов. Также за счёт всего этого мы значительно разгрузили арифметическо-логическое устройство нашего микроконтроллера.