

# Timer interrupts

Posted on [February 28, 2012](#)

This article will discuss AVR and Arduino timers and how to use them in Arduino projects or custom AVR circuits.

What is a timer?

Like in real life, in microcontrollers a timer is something you set to trigger an alert at a certain point in the future. When that point arrives, that alert interrupts the microprocessor, reminding it to do something, like run a specific piece of code.

Timers, like external interrupts, run independently from your main program. Rather than running a loop or repeatedly calling `millis()`, you can let a timer do that work for you while your code does other things.

So suppose you have a device that needs to do something –like blink an LED every 5 seconds. If you are not using timers but just conventional code techniques, you'd have to set a variable with the next time the LED should blink, then check constantly to see if that time had arrived. With a timer interrupt, you can set up the interrupt, then turn on the timer. The LED will blink perfectly on time, regardless of what your main program was just doing

How do timers work?

Timers work by incrementing a counter variable known as a *counter register*. The counter register can count to a certain value, depending on its size (usually 8 bits or 16 bits). The timer increments this counter one step at a time until it reaches its maximum value, at which point the counter *overflows*, and resets back to zero. The timer normally sets a flag bit to let you know an overflow has occurred. This flag can be checked manually, or you can have the timer trigger an interrupt as soon as the flag is set. And as with any other interrupt, you can specify an Interrupt Service Routine (ISR) to run your own code when the timer overflows. The ISR will automatically reset the overflow flag, so using interrupts is usually your best option for simplicity and speed.

To increment the counter value at regular intervals, the timer must have a *clock source*. The clock provides a consistent signal. Every time the timer detects this signal, it increases its counter by one.

Since timers are dependent on the clock source, the smallest measurable amount of time will be the period of the clock. If you provide a 16 MHz clock signal to a timer, the timer resolution



(or timer period) is:

$$T = 1 / f \quad (f \text{ is the clock frequency})$$

$$T = 1 / (16 * 10^6)$$

$$T = (0.0625 * 10^{-6}) \text{ s}$$

The timer resolution thus is 0.0625 milliseconds

For 8 MHz this would be 0.125 milliseconds

and for 1 MHz exactly one millionth of a second

## Follow “Arduino”

Get every new post delivered to your Inbox.

Join 76 other followers



Build a website with WordPress.com

You can supply an external clock source for use with timers, but usually the chip's internal clock is used as the clock source. The 16 MHz crystal that is usually part of a setup for an Atmega328 can be considered as part of the internal clock.

### Different timers

In the standard Arduino variants or the 8-bit AVR chips, there are several timers at your disposal.

The ATmega8, ATmega168 and ATmega328 have three timers: Timer0, Timer1, and Timer2. They also have a watchdog timer, which can be used as a safeguard or a software reset mechanism. The Mega series has 3 additional timers.

#### Timer0

Timer0 is an 8-bit timer, meaning its counter register can record a maximum value of 255 (the same as an unsigned 8-bit byte). Timer0 is used by native Arduino timing functions such as `delay()` and `millis()`, so unless you know what you are doing, timer 0 is best left alone.

#### Timer1

Timer1 is a 16-bit timer, with a maximum counter value of 65535 (an unsigned 16-bit integer). The Arduino Servo library uses this timer, so keep that in mind if you use this timer in your projects.

#### Timer2

Timer2 is an 8-bit timer that is very similar to Timer0. It is used by the Arduino `tone()` function.

#### Timer3, Timer4, Timer5

The AVR ATmega1280 and ATmega2560 (found in the Arduino Mega variants) have an additional three timers. These are all 16-bit timers, and function similarly to Timer1.

### Configuring the timer register

In order to use these timers the built-in timer registers on the AVR chip that store timer settings need to be configured. There are a number of registers per timer. Two of these registers –the Timer/Counter Control Registers- hold setup values, and are called TCCRxA and TCCRxB, where x is the timer number (TCCR1A and TCCR1B, etc.). Each register holds 8 bits, and each bit stores a configuration value. The [ATmega328 datasheet](#) specifies those as follows:

#### TCCR1A

Bit	7	6	5	4	3	2	1	0	TCCR1A
0x80	COM1A1	COM1A0	COM1B1	COM1B0	–	–	WGM11	WGM10	
ReadWrite	RW	RW	RW	RW	R	R	RW	RW	
Initial Value	0	0	0	0	0	0	0	0	

#### TCCR1B

Bit	7	6	5	4	3	2	1	0	TCCR1B
0x81	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	
ReadWrite	R/W	R/W	R	R/W	R/W	R/W	R/W	R/w	
Initial Value	0	0	0	0	0	0	0	0	

The most important settings are the last three bits in TCCR1B, CS12, CS11, and CS10. These determine the timer clock setting. By setting these bits in various combinations, you can make the timer run at different speeds. This table shows the required settings:

#### Clock Select bit description

CS12	CS11	CS10	Description
0	0	0	No clock source (Timer/Counter stopped)
0	0	1	$\text{clk}_{i/o}/1$ (No prescaling)
0	1	0	$\text{clk}_{i/o}/8$ (From Prescaler)
0	1	1	$\text{clk}_{i/o}/64$ (From Prescaler)
1	0	0	$\text{clk}_{i/o}/256$ (From Prescaler)
1	0	1	$\text{clk}_{i/o}/1024$ (From Prescaler)

1	1	0	External clock source on T1 pin. Clock on falling edge
1	1	1	External clock source on T1 pin. Clock on rising edge

By default, these bits are set to zero. Suppose you want to have Timer1 run at clock speed, with one count per clock cycle. When it overflows, you want to run an Interrupt Service Routine (ISR) that toggles a LED tied to pin 13 on or off. Below you will find the Arduino code for this example, for completeness I use avr-libc routines wherever they don't make things overly complicated.

First, initialize the timer:

```
// avr-libc library includes
#include <avr/io.h>
#include <avr/interrupt.h>
#define LEDPIN 13

void setup()
{
  pinMode(LEDPIN, OUTPUT);
  // initialize Timer1
  cli();          // disable global interrupts
  TCCR1A = 0;     // set entire TCCR1A register to 0
  TCCR1B = 0;     // set entire TCCR1B register to 0
                  // (as we do not know the initial values)

  // enable Timer1 overflow interrupt:
  TIMSK1 |= (1 << TOIE1); //Atmega8 has no TIMSK1 but a TIMSK regi:

  // Set CS10 bit so timer runs at clock speed: (no prescaling)
  TCCR1B |= (1 << CS10); // Sets bit CS10 in TCCR1B
  // This is achieved by shifting binary 1 (0b00000001)
  // to the left by CS10 bits. This is then bitwise
  // OR-ed into the current value of TCCR1B, which effectively set
  // this one bit high. Similar: TCCR1B |= _BV(CS10);

  // enable global interrupts:
  sei();
}
```

The register TIMSK1 is the *Timer/Counter1 Interrupt Mask Register*. It controls which interrupts the timer can trigger. Setting the TOIE1 bit (=Timer1 Overflow Interrupt Enable) tells the timer to trigger an interrupt when the timer overflows. It can also be set to other bits to trigger other interrupts. More on that later.

When you set the CS10 bit, the timer is running, and since an overflow interrupt is enabled, it will call the `ISR(TIMER1_OVF_vect)` whenever the timer overflows.

Next define the ISR:

```
ISR(TIMER1_OVF_vect)
{

    digitalWrite(LEDPIN, !digitalRead(LEDPIN));
    // or use: PORTB ^= _BV(PB5); // PB5 =pin 19 is digitalpin 13
}
```

Now you can define `loop()` and the LED will toggle on and off regardless of what's happening in the main program. To turn the timer off, set `TCCR1B = 0` at any time.

How fast will the LED blink with this code?

Timer1 is set to interrupt on an overflow, so if you are using an ATmega328 with a 16MHz clock. Since Timer1 is 16 bits, it can hold a maximum value of  $(2^{16} - 1)$ , or 65535. At 16MHz, we'll go through one clock cycle every  $1/(16 \times 10^6)$  seconds, or  $6.25 \times 10^{-8}$  s. That means 65535 timer counts will pass in  $(65535 \times 6.25 \times 10^{-8} \text{s})$  and the ISR will trigger in about 0.0041 seconds. Then again and again, every four thousandths of a second after that. That is too fast to see it blink. If anything, we've created an extremely fast PWM signal for the LED that's running at a 50% duty cycle, so it may appear to be constantly on but dimmer than normal. An experiment like this shows the amazing power of microprocessors – even an inexpensive 8-bit chip can process information far faster than we can detect.

Timer prescaling and preloading

To control this you can also set the timer to use a *prescaler*, which allows you to divide your clock signal by various powers of two, thereby increasing your timer period. For example, if you want the LED blink at one second intervals. In the TCCR1B register, there are three CS bits to set a better timer resolution. If you set CS10 and CS12 using:

`TCCR1B |= (1 << CS10);` and `TCCR1B |= (1 << CS12);`, the clock source is divided by 1024. This gives a timer resolution of  $1/(16 \times 10^6 / 1024)$ , or 0.000064 seconds (15625 Hz). Now the timer will overflow every  $(65535 \times 6.4 \times 10^{-5} \text{s})$ , or 4.194s.

If you would set only CS12 using `TCCR1B |= (1 << CS12);` (or just `TCCR1B=4`), the clock source is divided by 256. This gives a timer resolution of  $1/(16 \times 10^6 / 256)$ , or 0.000016 sec (62500 Hz) and the timer will overflow every  $(65535 \times 0.000016 \text{s})$  1.04856 sec.

Suppose you do not want an 1.04856 sec interval but a 1 sec interval. It is clear to see that if the counter wasn't 65535 but 62500 (being equal to the frequency), the timer would be set at

1sec. The counter thus is  $65535 - 62500 = 3035$  too high. To have more precise 1 second timer we need to change only one thing – timer's start value saved by TCNT1 register (Timer Counter ). We do this with `TCNT1=0x0BDC`; BDC being the hex value of 3035. A Value of 34286 for instance would give 0.5 sec  $((65535 - 34286) / 62500)$

The code looks as follows:

```
// avr-libc library includes
#include <avr/io.h> // can be omitted
#include <avr/interrupt.h> // can be omitted
#define LEDPIN 13
/* or use
DDRB = DDRB | B00100000; // this sets pin 5 as output
                        // pins
*/
void setup()
{
  pinMode(LEDPIN, OUTPUT);

  // initialize Timer1
  cli();           // disable global interrupts
  TCCR1A = 0;      // set entire TCCR1A register to 0
  TCCR1B = 0;      // set entire TCCR1A register to 0

  // enable Timer1 overflow interrupt:
  TIMSK1 |= (1 << TOIE1);
  // Preload with value 3036
  //use 64886 for 100Hz
  //use 64286 for 50 Hz
  //use 34286 for 2 Hz
  TCNT1=0x0BDC;
  // Set CS10 bit so timer runs at clock speed: (no prescaling)
  TCCR1B |= (1 << CS12); // Sets bit CS12 in TCCR1B
  // This is achieved by shifting binary 1 (0b00000001)
  // to the left by CS12 bits. This is then bitwise
  // OR-ed into the current value of TCCR1B, which effectively set
  // this one bit high. Similar: TCCR1B |= _BV(CS12);
  // or: TCCR1B= 0x04;

  // enable global interrupts:
  sei();
}

ISR(TIMER1_OVF_vect)
{
  digitalWrite(LEDPIN, !digitalRead(LEDPIN));
}
```

```

}

void loop() {}

```

## CTC

But there's another mode of operation for AVR timers. This mode is called Clear Timer on Compare Match, or *CTC*. Instead of counting until an overflow occurs, the timer compares its count to a value that was previously stored in a register. When the count matches that value, the timer can either set a flag or trigger an interrupt, just like the overflow case.

To use CTC, you need to figure out how many counts you need to get to a one second interval. Assuming we keep the 1024 prescaler as before, we'll calculate as follows:

$$(\text{target time}) = (\text{timer resolution}) * (\# \text{ timer counts} + 1)$$

and rearrange to get

$$\begin{aligned}
 (\# \text{ timer counts} + 1) &= (\text{target time}) / (\text{timer resolution}) \\
 (\# \text{ timer counts} + 1) &= (1 \text{ s}) / (6.4\text{e-}5 \text{ s}) \\
 (\# \text{ timer counts} + 1) &= 15625 \\
 (\# \text{ timer counts}) &= 15625 - 1 = 15624
 \end{aligned}$$

You have to add the extra +1 to the number of timer counts because in CTC mode, when the timer matches the desired count it will reset itself to zero. This takes one clock cycle to perform, so that needs to be factored into the calculations. In many cases, one timer tick isn't a huge deal, but if you have a time-critical application it can make all the difference in the world.

Now the `setup()` function to configure the timer for these settings is as follows:

```

void setup()
{

  pinMode(LED_PIN, OUTPUT); // you have to define the LED_PIN as say :
                             // or so earllier in yr program

  // initialize Timer1
  cli();                    // disable global interrupts
  TCCR1A = 0;              // set entire TCCR1A register to 0
  TCCR1B = 0;              // same for TCCR1B

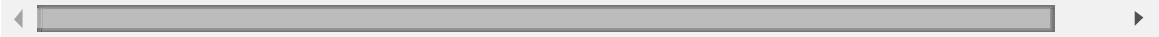
```

```
// set compare match register to desired timer count:
OCR1A = 15624;

// turn on CTC mode:
TCCR1B |= (1 << WGM12);

// Set CS10 and CS12 bits for 1024 prescaler:
TCCR1B |= (1 << CS10);
TCCR1B |= (1 << CS12);

// enable timer compare interrupt:
TIMSK1 |= (1 << OCIE1A);
sei();          // enable global interrupts
}
```



And you need to replace the overflow ISR with a compare match version:

```
ISR(TIMER1_COMPA_vect)
{
  digitalWrite(LEDPIN, !digitalRead(LEDPIN));
}
```

The LED will now blink on and off at precisely one second intervals. And you are free to do anything you want in `loop()`. As long as you don't change the timer settings, it won't interfere with the interrupts. With different mode and prescaler settings, there's no limit to how you use timers.

Here's the complete example in case you'd like to use it as a starting point for your own project.

```
// Arduino timer CTC interrupt example
//
// avr-libc library includes
#include <avr/io.h>
#include <avr/interrupt.h>
#define LEDPIN 13
void setup()
{
  pinMode(LEDPIN, OUTPUT);
  // initialize Timer1
  cli();          // disable global interrupts
  TCCR1A = 0;     // set entire TCCR1A register to 0
```



```

TCCR1B = 0;      // same for TCCR1B

// set compare match register to desired timer count:
OCR1A = 15624;

// turn on CTC mode:
TCCR1B |= (1 << WGM12);

// Set CS10 and CS12 bits for 1024 prescaler:
TCCR1B |= (1 << CS10);
TCCR1B |= (1 << CS12);

// enable timer compare interrupt:
TIMSK1 |= (1 << OCIE1A);

// enable global interrupts:
sei();
}

void loop()
{
  // main program
}

ISR(TIMER1_COMPA_vect)
{
  digitalWrite(LEDPIN, !digitalRead(LEDPIN));
}

```

Remember that you can use the built-in ISRs to extend timer functionality. For example, if you wanted to read a sensor every 10 seconds, there's no timer set-up that can go this long without overflowing. However, you can use the ISR to increment a counter variable in your program once per second, then read the sensor when the variable hits 10. Using the same CTC setup as in our previous example, the ISR would look something like this:

```

ISR(TIMER1_COMPA_vect)
{
  seconds++;
  if(seconds == 10)
  {
    seconds = 0;
    readSensor();
  }
}

```

For a variable to be modified within an ISR, it is good custom to declare it as volatile. In this case, you need to declare `volatile byte seconds;` or similar at the start of the program.

#### A WORD ON THE ATMEGA8

The Atmega8 seems to give people problems with use of the timers, one reason is that it doesn't have a TIMSK1 register (in fact it doesn't have a TIMSKn register), it does have a TIMSK register though that is shared amongst the 3 timers. As I do not have an Atmega8 (like the early Arduino NG) I can not test it, but if you encounter problems, the following programs will help:

```
// this code sets up counter0 with interrupts enabled on an Atmega
// beware, it may generate errors in Arduino IDE
// as 'milis' uses timer0
#include <avr/io.h>
#include <avr/io.h>

void setup()
{
  DDRD &= ~(1 << DDD4); // Clear the PD4 pin
  // PD0 is now an input

  PORTD |= (1 << PORTD4); // turn On the Pull-up
  // PD4 is now an input with pull-up enabled

  TIMSK |= (1 << TOIE0); // enable timer interrupt

  TCCR0 |= (1 << CS02) | (1 << CS01) | (1 << CS00);
  // Turn on the counter, Clock on Rise

  sei();
}

void loop()
{
  // Stuff
}

ISR (TIMER0_OVF_vect)
{
  // interrupt just fired, do stuff
}
```

A 1 sec flasher using the timer 1 CTC mode for the Atmega 8 would look like this:

```
void setup()
{
  pinMode(13,OUTPUT);
  /* or use:
  DDRB = DDRB | B00100000; // this sets pin 5 as output
                        // without changing the value of the other
  */
  // Disable interrupts while loading registers
  cli();
  // Set the registers
  TCCR1A = 0; //Timer Counter Control register
  // Set mode
  TCCR1B = (1 << WGM12); // turn on CTC mode
  // Set prescale values (1024). (Could be done in same statement
  // as setting the WGM12 bit.)
  TCCR1B |= (1 << CS12) | (1 << CS10);
  //Enable timer compare interrupt==> TIMSK1 for ATmega328,
  //TIMSK for ATmega8
  TIMSK |= (1 << OCIE1A);
  // Set OCR1A
  OCR1A = 15624;
  // Enable global interrupts
  sei();
}

void loop(){}

ISR (TIMER1_COMPA_vect) {
  digitalWrite(13, !digitalRead(13));
  //PORTB ^= _BV(PB5); // as digitalWrite(13,x) is an Arduino
  //function, direct writing to the port may be preferable
}
```

It is obvious that this is very akin to the CTC program presented earlier for the Atmega328 and in fact will work on the Atmega238 as well by renaming 'TIMSK' to 'TIMSK1'

### Other Atmega chips:

TCCR0 should be TCCR0A in ATmega164P/324P/644

### Attiny

The Attiny series has timer interrupts too. This code sets up a 50uS timer in CTC mode on the Attiny85 (pag 79 datasheet)

```
TCCR0A = (1 << WGM01);    //CTC mode. set WGM01
TCCR0B = (2 << CS00);      //divide by 8  sets
OCR0A = F_CPU/8 * 0.000050 - 1;    // 50us compare value
TIMSK |= (1<<OCIE0A);        //set interrupt

ISR(TIMER0_COMPA_vect)
{
    // code of choice!
}
```

More on timers [here](#)

[here](#)

[here](#)

[here](#)

[here](#)

[here](#)

[here](#)

[pwm generation by timers](#)

[here \(Atmega8\)](#)

[Atmega8 Datasheet](#)

[Atmega328 Datasheet](#)

---

#### SHARE THIS:



5 bloggers like this.

---

#### RELATED

[Arduino hardware interrupt](#)

With 2 comments

[Timer Interrupts re-visited](#)

In "Arduino"

[Flashing an LED with Attiny13: Timer,](#)

[Watchdog timer and Sleep](#)

In "Attiny"

This entry was posted in [Uncategorized](#) by [Arduino](#). Bookmark the [permalink](#) [\[https://arduino diy.wordpress.com/2012/02/28/timer-interrupts/\]](https://arduino diy.wordpress.com/2012/02/28/timer-interrupts/) .

54 THOUGHTS ON "TIMER INTERRUPTS"



Beyaz

on [December 29, 2012 at 9:08 am](#) said:

Best explanation of internal timers. Very clear. Thank you very much



Arduino

on [September 14, 2014 at 10:05 pm](#) said:

My pleasure, I expanded it a bit

Pingback: [Utilizando interrupção interna ou por tempo no Arduino | daltonhioki](#)

Pingback: [#2 – Touché meets Pd and Max/MSP « Stefano Trento](#)

Pingback: [#2 – Touché meets Pd and Max/MSP |](#)



carloschsa

on [April 16, 2013 at 12:03 am](#) said:

I appreciate your work. Thanks



Arduino

on **June 20, 2013 at 8:39 pm** said:

Thank you

Pingback: [lucadentella.it](http://lucadentella.it) – Allegro A4988 e Arduino (3)

comcomAude

on **June 12, 2013 at 2:38 am** said:

Thx you very much, this help a lot !



Arduino

on **June 20, 2013 at 8:29 pm** said:

Great



Tuyen

on **June 20, 2013 at 4:23 pm** said:

thank you, it's very usefull, but i think lib should be set:

#include

#include



Tuyen

on [June 20, 2013 at 4:25 pm](#) said:

```
#include
```

```
#include
```



Arduino

on [September 14, 2014 at 10:27 pm](#) said:

Ha Tuyen, sorry for my late reactio, Indeed the comment section of wordpress is not really suitable for code. The includes are: avr/io.h and avr/interrupt.h, both between 'fishhooks'



**Eduardo**

on [June 29, 2013 at 3:19 am](#) said:

hi guys,

i had a problem with this code using a Atmega8

"Interrupt.cpp: In function 'void setup()':

Interrupt.pde:-1: error: 'TIMSK1' was not declared in this scope"

i look on the datasheet and this register has the name TIMSK without "1" but it isn't works.....someone can help me? thans



Arduino

on [August 30, 2013 at 8:34 am](#) said:

I did not specifically have the atmega8 in mind but you could try altering the name



Arduino

on [September 14, 2014 at 10:25 pm](#) said:

it is over a year that i replied you and looking back at my reply i may have been a bit too hasty and not addressed things well. You seem not to be the only one who is having trouble with TIMSK1 and the Atmega8.

It may not help you anymore but maybe someone else wuith the same problem is helped by this code:

in case the includes drop in the code due to wordpress peculiarities:

they read avr/io.h and avr/interrupt.h, both between 'fishhooks'

// this code sets up counter0 and with interrupts enabled

```
#include
```

```
#include
```

```
int main(void)
```

```
{
```

```
  DDRD &= ~(1 << DDD4); // Clear the PD4 pin
```

```
  // PD0 is now an input
```

```
  PORTD |= (1 << PORTD4); // turn On the Pull-up
```

```
  // PD4 is now an input with pull-up enabled
```

```
  TIMSK |= (1 << TOIE0); // enable timer interrupt
```

```
  TCCR0 |= (1 << CS02) | (1 << CS01) | (1 << CS00);
```

```
  // Turn on the counter, Clock on Rise
```

```
  sei();
```

```
  while (1)
```

```
  {
```

```
    // we can read the value of TCNT0 hurray !!
```

```
  }
```

```
}
```

```
ISR (TIMER0_OVF_vect)
```

```
{
```

```
  // interrupt just fired
```

```
}
```



Pingback: [“Необычное” поведение режима CTC таймера1 | MyLinks](#)



chuck

on [August 20, 2013 at 5:28 am](#) said:

Just wanted to drop you a line to say thanks for the best explanation I have found on the timer/interrupt features. Nice work. Saved me a lot of time rather than digging through the depths of the 448 page data sheet.



Arduino

on [August 27, 2013 at 7:41 pm](#) said:

Thank you



Jan Kromhout

on [August 26, 2013 at 9:27 pm](#) said:

Great for this, have had a lot of fun to understanding this topic together with a scoop!



Arduino

on [August 27, 2013 at 7:41 pm](#) said:

Thank you



davinci

on **October 4, 2013 at 10:47 pm** said:

Thanks, by far best explanation I've encountered!



Arduino

on **October 8, 2013 at 10:55 pm** said:

Thank you



fanzeyi

on **March 23, 2014 at 7:56 pm** said:

Thank you so much! This helps me a lot!



Arduino

on **March 23, 2014 at 9:23 pm** said:

My pleasure



zenmonkey760

on **May 21, 2014 at 5:06 am** said:

Your explanation has propelled my understanding of timer interrupts like no other piece I've read. Thank you so much!



Arduino

on **May 21, 2014 at 5:33 am** said:

My pleasurw



The Dark Knight

on **August 17, 2014 at 12:16 pm** said:

Amazing work!It saved a lot of time.The data sheet is just too long!



Arduino

on **August 18, 2014 at 7:52 pm** said:

Thanks , my pleasure, glad you liked it



iforce2d

on **September 12, 2014 at 6:13 pm** said:

Wow, a very nice explanation, thankyou!



Arduino

on **September 12, 2014 at 7:41 pm** said:

My pleasure



Arduino



on **September 14, 2014 at 10:28 pm** said:

I have in fact expanded it a bit today



nerdant

on **October 14, 2014 at 4:36 am** said:

Great article!

In the section “Timer prescaling and preloading”, I believe there is an line of code missing in your code snippet.

In the “ISR(TIMER1\_OVF\_vect)” function you must re-load the TCNT1 register with your preset value. So this line should be added:

```
TCNT1=0x0BDC;
```

This article here: <http://www.hobbytronics.co.uk/arduino-timer-interrupts> helped me out.



Arduino

on **October 15, 2014 at 6:38 am** said:

Thanks.

I have checked my original code simply by running it - without that extra line- and it works fine, exactly as it should work.

I know the link you provide, as I even provide it at the end of my article . Since the link doesn't give any explanation, I guess they just made a mistake by adding that line. It seems a bit counter-intuitive to have to redefine a parameter.

Anyway, thanks for your observation and comment  
Always good to see how others do something.



Robert

on [March 20, 2015 at 11:30 am](#) said:

Yes it solves my problem, thanks.

R



les

on [October 18, 2014 at 7:39 am](#) said:

this is one of the most informative articles I hav ever encountered!  
thanx a million! but im stil having some kind of a problem., I cant  
differentiate between the timer interrupt configuration that cannot  
interfere with functions like millis (), analogwrite () ,etc. and the ones  
that can safely be used without worrying about those timer  
dependent functions! can you help me out please! thank you in  
advance for your respons!



Arduino

on [October 18, 2014 at 6:15 pm](#) said:

Les, thanks. I am on mobile so I will be brief for now, most  
timers are used for some function, but timer 0 is the one that  
is used most by the system. Timer1 is safe u less u use the  
servo library and timer2 is safe unless u use the tone  
function



Stan

on [February 11, 2015 at 9:53 pm](#) said:

Hello!

Thank you so much for this! Searched for hours before I found this.

I have a question though. Can a timer be started at a different time in program than setup()?

I see you state that setting CS bits starts it, but is there a way to start it and stop it eventually or will it go on forever?



Arduino

on **February 19, 2015 at 1:16 am** said:

I am getting a bit worried. I left a reply, but it seems not to have been posted. anyway there are several ways to stop the timer. One can do it by clearing the interrupt or by resetting the CS10, 11 and 12 bits in the control register:  
`TCCRxB &= ~(_BV(CS10) | _BV(CS11) | _BV(CS12));`



Spencer

on **April 8, 2015 at 2:37 am** said:

Thank you so much, because of this article I feel I am very close to being able to complete my project. However I am having an issue getting your example to work. I have tried to use the snippet for using a variable to increment and count many seconds (ultimately I need to count minutes), but it doesn't seem to be working. Here is the code which supposedly checks the timer, it compiles but does not ever blink the LED. Thank you in advance for any insights!

```
ISR(TIMER1_COMPA_vect)
{
  int seconds;
  seconds++;
  if (seconds == 2) {
    seconds = 0;
    // execute code here
    digitalWrite( 13, digitalRead( 13 ) ^ 1 );
  }
}
```

```
}  
}
```



Arduino

on **April 22, 2015 at 4:39 pm** said:

seems you are xor-ing the LED but not sure if you do it right.  
Try : `digitalWrite(13, !digitalRead(13));`



spencerjroberts

on **April 23, 2015 at 4:49 am** said:

Thanks for the reply! I figured it out: I just had to make the "seconds" variable a global variable.



Arduino

on **April 23, 2015 at 11:36 pm** said:

OK thanks for the feedback  
I am happy you got it working



haris

on **April 26, 2015 at 1:58 pm** said:

Great article,  
sir,  
i want to read encoders value precisely from robot's wheel using  
mega328 timers to trace path of my automatic guided vehicle, how  
can i do it ? where should i start this job?  
my encoders gives smooth square wave,  
need help,

kindly explain



Arduino

on **April 27, 2015 at 7:35 pm** said:

You can use the output of your decoders to generate a hardware interrupt on INT0 or INT1 (pin D2, D3 /physical pin 4 or 5)

That ill let you keep track of the number of rotations and thus the driven distance. However I think you need a bit more to also know the direction. As I do not know what decoders you have, I am not sure if they also indicate change of direction



haris

on **April 28, 2015 at 5:37 pm** said:

To the first i have 2 disk type encoders with 36 holes on each of them which interrupt IR beam from trnsmr to rcvr on rotation and gives 36 pulses ,

2nd, yes u r right, i hv to detect direction of my bot, i m thinking to connect 2 wires from H-bridge to digital input of arduino , HIGH or LOW digital input will tell status of direction of my bot.

my teacher gave me task of counting using timers,cuz interrupt is disturbing controller countinously while running other task

can i connect sqr wave coming from encoders to timer pin of arduino as external clock by setting CS10,CS11,CS12 bit to 1 ? if yes how ? and then run CTC mode for counting





Zohaib

on **May 16, 2015 at 7:58 pm** said:

I have written a code for one second delay to blink led using Atmega 2560,timers.I want to get 5sec delay .how can i get this by using this code...

Here is my code:

```
#include
#include
#define LEDPIN 13
#define LEDPINn 12
int seconds=0;

void setup()
{

pinMode(LEDPIN, OUTPUT);

// initialize Timer1
noInterrupts();
TCCR1A = 0; // set entire TCCR1A register to 0
TCCR1B = 0; // same for TCCR1B

// set compare match register to desired timer count:
OCR1A = 156.24;//10ms delay
// turn on CTC mode:
TCCR1B |= (1 << WGM12);
// Set CS10 and CS12 bits for 1024 prescaler:
TCCR1B |= (1 << CS10);
TCCR1B |= (1 << CS12);
// enable timer compare interrupt:
TIMSK1 |= (1 << OCIE1A);
// enable global interrupts:
interrupts();
pinMode(LEDPINn,OUTPUT);
noInterrupts();
TCCR3A=0;
TCCR3B=0;
```

```

OCR3A=15624;//100 msec delay
TCCR3B |= (1 << CS30);
TCCR3B |= (1 << CS32);
TIMSK3 |= (1 << OCIE3B);
interrupts();

}

void loop()
{
  // do some crazy stuff while my LED keeps blinking
}
ISR(TIMER1_COMPA_vect)
{

digitalWrite(LEDPIN, !digitalRead(LEDPIN));

}

ISR(TIMER3_COMPA_vect)
{
digitalWrite(LEDPIN, !digitalRead(LEDPIN));
}

```



Arduino

on **May 16, 2015 at 10:13 pm** said:

That is not so hard and in fact I describe it in the article. Keep a counter in your interrupt routine, increasing the counter with 1 every time the interrupt is called. Then when it comes to 5 you clear the counter and toggle the LED.

```

ISR(TIMER1_COMPA_vect)
{
  seconds++;
  if(seconds == 5)
  {
    seconds = 0;
    digitalWrite(LEDPIN, !digitalRead(LEDPIN));
  }
}

```

ofcourse you have to declare your variable in the setup.

Pingback: [Studies On Criticality No. 1 – first steps | studio algorhythmics](#)



Ver

on [August 29, 2015 at 3:29 am](#) said:

Very informative! Thank you so much.



Arduino

on [August 30, 2015 at 12:31 am](#) said:

i am glad you liked it



Jeff

on [October 20, 2015 at 5:06 pm](#) said:

ARDUINO SKETCH TO SLOWLY DIM LIGHT BULB (part 1)

```
#include
```

```
/*
```

```
Dim_PSSR_ZC_Tail
```

This sketch is a sample sketch using the ZeroCross Tail(ZCT) to generate a sync pulse to drive a PowerSSR Tail(PSSRT) for dimming ac lights.

Connections to an Arduino Duemilanove:

1. Connect the C terminal of the ZeroCross Tail to digital pin 2 with a 10K ohm pull up to Arduino 5V.
2. Connect the E terminal of the ZeroCross Tail to Arduino Gnd.
3. Connect the PowerSSR Tail +in terminal to digital pin 4 and the -in terminal to Gnd.

```
*/
```

```
#include
```

```
volatile int i=0; // Variable to use as a counter
```

```
volatile boolean zero_cross=0; // Boolean to store a "switch" to tell us if we have crossed zero
```

```
int PSSR1 = 4; // PowerSSR Tail connected to digital pin 4
```

```
int dim = 32; // Default dimming level (0-128) 0 = on, 128 = off
```

```
int freqStep = 60; // Set to 60hz mains
```

```
int LED = 0; // LED on Arduino board on digital pin 13
```

```
void setup()
```

```
{
```

```
pinMode(LED, OUTPUT);
```

```
pinMode(4, OUTPUT); // Set SSR1 pin as output
```

```
attachInterrupt(0, zero_cross_detect, RISING); // Attach an Interrupt to digital pin 2 (interrupt 0),
```

```
Timer1.initialize(freqStep);
```

```
Timer1.attachInterrupt(dim_check, freqStep);
```

```
}
```

```
void loop() // Main loop
```

```
{
```

```
dim = 0;
```

```
delay(500);
```

```
dim = 1;
```

```
delay(500);
```

```
dim = 2;
```

```
delay(500);
```

```
dim = 3;
```

```
delay(500);
```

```
dim = 4;
```

```
delay(500);
```

```
dim = 5;
```

```
delay(500);
```

```
dim = 6;
```

```
delay(500);  
dim = 7;  
delay(500);  
dim = 8;  
delay(500);  
dim = 9;  
delay(500);  
dim = 10;  
delay(500);  
dim = 11;  
delay(500);  
dim = 12;  
delay(500);  
dim = 13;  
delay(500);  
dim = 14;  
delay(500);  
dim = 15;  
delay(500);  
dim = 16;  
delay(500);  
dim = 17;  
delay(500);  
dim = 18;  
delay(500);  
dim = 19;  
delay(500);  
dim = 20;  
delay(500);  
dim = 21;  
delay(500);  
dim = 22;  
delay(500);  
dim = 23;  
delay(500);  
dim = 24;  
delay(500);  
dim = 25;  
delay(500);  
dim = 26;  
delay(500);  
dim = 27;  
delay(500);
```

```
dim = 28;
delay(500);
dim = 29;
delay(500);
dim = 30;
delay(500);
dim = 31;
delay(500);
dim = 32;
delay(500);
dim = 33;
delay(500);
dim = 34;
delay(500);
dim = 35;
delay(500);
dim = 36;
delay(500);
dim = 37;
delay(500);
dim = 38;
delay(500);
dim = 39;
delay(500);
dim = 40;
delay(500);
dim = 41;
delay(500);
dim = 42;
delay(500);
dim = 43;
delay(500);
dim = 44;
delay(500);
dim = 45;
delay(500);
dim = 46;
delay(500);
dim = 47;
delay(500);
dim = 48;
delay(500);
dim = 49;
```

```
delay(500);  
dim = 50;  
delay(500);  
dim = 51;  
delay(500);  
dim = 52;  
delay(500);  
dim = 53;  
delay(500);  
dim = 54;  
delay(500);  
dim = 55;  
delay(500);  
dim = 56;  
delay(500);  
dim = 57;  
delay(500);  
dim = 58;  
delay(500);  
dim = 59;  
delay(500);  
dim = 60;  
delay(500);  
dim = 61;  
delay(500);  
dim = 62;  
delay(500);  
dim = 63;  
delay(500);  
dim = 64;  
delay(500);  
dim = 65;  
delay(500);  
dim = 66;  
delay(500);  
dim = 67;  
delay(500);  
dim = 68;  
delay(500);  
dim = 69;  
delay(500);  
dim = 70;  
delay(500);
```

```
dim = 71;  
delay(500);  
dim = 72;  
delay(500);  
dim = 73;  
delay(500);  
dim = 74;  
delay(500);  
dim = 75;  
delay(500);  
dim = 76;  
delay(500);  
dim = 77;  
delay(500);  
dim = 78;  
delay(500);  
dim = 79;  
delay(500);  
dim = 80;  
delay(500);  
dim = 81;  
delay(500);  
dim = 82;  
delay(500);  
dim = 83;  
delay(500);  
dim = 84;  
delay(500);  
dim = 85;  
delay(500);  
dim = 86;  
delay(500);  
dim = 87;  
delay(500);  
dim = 88;  
delay(500);  
dim = 89;  
delay(500);  
dim = 90;  
delay(500);  
dim = 91;  
delay(500);  
dim = 92;
```



```
delay(500);  
dim = 93;  
delay(500);  
dim = 94;  
delay(500);  
dim = 95;  
delay(500);  
dim = 96;  
delay(500);  
dim = 97;  
delay(500);  
dim = 98;  
delay(500);  
dim = 99;  
delay(500);  
dim = 100;  
delay(500);  
dim = 101;  
delay(500);  
dim = 102;  
delay(500);  
dim = 103;  
delay(500);  
dim = 104;  
delay(500);  
dim = 105;  
delay(500);  
dim = 106;  
delay(500);  
dim = 107;  
delay(500);  
dim = 108;  
delay(500);  
dim = 109;  
delay(500);  
dim = 110;  
delay(500);  
dim = 111;  
delay(500);  
dim = 112;  
delay(500);  
dim = 113;  
delay(500);
```

```
dim = 114;
delay(500);
dim = 115;
delay(500);
dim = 116;
delay(500);
dim = 117;
delay(500);
dim = 118;
delay(500);
dim = 119;
delay(500);
dim = 120;
delay(500);
dim = 121;
delay(500);
dim = 122;
delay(500);
dim = 123;
delay(500);
dim = 124;
delay(500);
dim = 125;
delay(500);
dim = 126;
delay(500);
dim = 127;
delay(500);
dim = 128;
delay(500);
}
```

// Functions

```
void dim_check() { // This function will fire the triac at the proper
time
if(zero_cross == 1) { // First check to make sure the zero-cross has
happened else do nothing
if(i>=dim) {
delayMicroseconds(100); //These values will fire the PSSR Tail.
digitalWrite(PSSR1, HIGH);
delayMicroseconds(50);
digitalWrite(PSSR1, LOW);
}
```

```
i = 0; // Reset the accumulator
zero_cross = 0; // Reset the zero_cross so it may be turned on
again at the next zero_cross_detect
} else {
i++; // If the dimming value has not been reached, increment the
counter
} // End dim check
} // End zero_cross check
}

void zero_cross_detect()
{
zero_cross = 1;
// set the boolean to true to tell our dimming function that a zero
cross has occurred
}
```



Jeff

on **October 20, 2015 at 5:07 pm** said:

### ARDUINO SKETCH TO SLOWLY DIM LIGHT BULB (part 2)

Hello and thanks for this great resource. I'm an arduino newb and am currently struggling with trying to make a light bulb dim very slowly (full brightness to totally off) as smoothly as I possibly can, ideally over the course of say 2-5 minutes.

At this point my hardware is behaving as advertised and it's a matter of modifying the original sample sketch I was provided by the seller of the hardware solution I'm using (ZeroCross Tail & PowerSSR Tail with an Arduino Duemilanove Atmega 328).

Modifying the parameters I was given in the original sketch the way I have has led me to a roadblock (SEE SKETCH in "part 1" comment above). I'm sure there's a more elegant way to accomplish what I have –

too many individual lines of code with the "dim" and "delay" going incremental for all 128 steps. Regardless, this code is working with the hardware to dim the bulb but the steps (128 steps @ 500ms

each) still seem too apparent visually, especially toward the tail end of the fade to darkness and more importantly the overall time the fade takes happens too quickly.

So now the problem is how can I add more steps in the dimming process (more than the 128 in the sample sketch I started with) as well as have the delay arguments less than 500ms?

I was told in order to get a smoother longer seamless transition from light to dark I would need to modify the timer settings. Any advice or a direction you can point me in to better focus my time to find a solution would be GREATLY appreciated?



Arduino

on **October 21, 2015 at 2:31 pm** said:

Jeff your code is indeed a bit inefficient. For one thing why would you want to have more than 128 steps? the 128 steps regulate from ON to OFF. Do you really want to divide that into say 256 levels? the difference between those steps will be small. Sure it is possible, but let me handle the 500ms question first.

as you can see in yr code, there is a delay of 500ms between each step. If you want that to be less you have to alter that to say 200, or 100, whatever you like.

Now obviously you dont want to do that 128 times, especially not if you may want to chose another delay later. I am not sure if you wrote/changed that part of the code, but this is exactly the sort of thing that asks for a FOR NEXT loop.

I am not sure how you ended up at my 'timer interrupt' post as I have a post that exactly deals with AC dimming:

<https://arduino diy.wordpress.com/2012/10/19/dimmer-arduino/> and gives you programs how to do this. Basically

you have to replace tour entire 'void loop()' by the following:

```
void loop() {  
  for (int i=5; i <= 128; i++)  
  {  
    dimming=i;  
    delay(10);  
  }  
}
```

```
}
```

That will take care of it.

Now with regard to your number of steps: I suggest you first change yr code as I just instructed, before you start messing with the steps. For now let me say that I spot an error in your code. Change the line

```
"int freqStep = 60; // Set to 60hz mains"
```

into

```
"int freqStep = 65; // Set to 60hz mains"
```

the 65 is not your grid frequency it is yr steplength for 128 steps for 60Hz. as it is equal to  $8333/128$  So if it is 60 you already have 138 steps. If you want more steps you have to lower the number 'freqStep.

let me explain.

I presume your board uses a double phase rectification of the grid frequency for its Zerocrossing pulse. That means that there will be a 120Hz signal after the rectification.

120Hz is equal to a period of 8333 uSec. Meaning that you have 8333uSec to do your phase cutting your dimming level depends on when in that cycle you do your phase cutting. If you take steps of 60uS you have 139 steps in that cycle. If you take steps of 65 usecs you have 128 steps ( $128 \times 65 = 8333$ ).

So suppose you want 1000 steps, yr frequency step needs to be 8.333. This ofcourse would be a totally impractical number of steps but it is possible

