OscarLiang.net

Sharing Knowledge and Ideas

Arduino Timer and Interrupt Tutorial

Arduino Timer and Interrupt Tutorial

This tutorial shows the use of timers and interrupts for Arduino boards. As Arduino programmer you have probably used timers and interrupts without even knowing it's there, because all the low level hardware stuff is hidden by the Arduino API. Many Arduino functions uses timers, for example the time functions: delay(), millis() and micros(), the PWM functions analogWrite(), the tone() and the noTone() function, even the Servo library uses timers and interrupts.

What is a timer?

A timer, A.K.A. counter is a piece of hardware built in the Arduino controller. It is like a clock, and can be used to measure time events.

The timer can be programmed by some special registers. You can configure the pre-scaler for the timer, or the mode of operation and many other things.

The Arduino board is based on the Atmel AVR ATmega168 or the ATmega328 microchip. These chips are pin compatible and only differ in the size of internal memory. Both have 3 timers, called Timer0, Timer1 and Timer2. Timer0 and Timer2 are 8bit timer, where Timer1 is a 16bit timer.

The most important difference between 8bit and 16bit timer is the timer resolution. 8bits means 256 values (two to the power of 8) where 16bit means 65536 values (two to the power of 16) which is much higher resolution.

The Arduino Mega series is based on the Atmel AVR ATmega1280 or the ATmega2560. They are almost identical to previous chips but only differs in memory size. These chips have 6 timers. First 3 timers (Timer 0, Timer1 and Timer2) are identical to the ATmega168/328. Timer3, Timer4 and Timer5 are all 16bit timers, similar to Timer1.

All timers depends on the system clock of your Arduino system. Normally the system clock is 16MHz, but the Arduino Pro 3/3V is 8Mhz, so be careful when writing your own timer functions. The timer hardware can be configured with some special timer registers. In the Arduino firmware, all timers were configured to a 1kHz frequency and interrupts are generally enabled.

To summarize:

```
Timer0:
 Timer0 is a 8bit timer.
 In the Arduino world Timer0 is been used for the timer functions, like delay(), millis() and micr
os(). If you change Timer0 registers, this may influence the Arduino timer function. So you should
 know what you are doing.
Timer1:
 Timer1 is a 16bit timer.
 In the Arduino world the Servo library uses Timer1 on Arduino Uno (Timer5 on Arduino Mega).
Timer2:
 Timer2 is a 8bit timer like Timer0.
 In the Arduino work the tone() function uses Timer2.
Timer3, Timer4, Timer5: Timer 3,4,5 are only available on Arduino Mega boards. These timers are al
l 16bit timers.
Timer Register
You can change the Timer behaviour through the timer register. The most important timer registers
are:
```

```
TCCRx - Timer/Counter Control Register. The pre-scaler can be configured here.
```

```
TCNTx - Timer/Counter Register. The actual timer value is stored here.
```

```
OCRx - Output Compare Register
```

```
ICRx - Input Capture Register (only for 16bit timer)
```

```
TIMSKx - Timer/Counter Interrupt Mask Register. To enable/disable timer interrupts.
```

```
TIFRx - Timer/Counter Interrupt Flag Register. Indicates a pending timer interrupt.
```

Clock select and timer frequency

Different clock sources can be selected for each timer independently. To calculate the timer frequency (for example 2Hz using Timer1) you will need:

1. CPU frequency 16Mhz for Arduino
2. maximum timer counter value (256 for 8bit, 65536 for 16bit timer)
3. Divide CPU frequency through the chosen pre-scaler (16000000 / 256 = 62500)
4. Divide result through the desired frequency (62500 / 2Hz = 31250)
5. Verify the result against the maximum timer counter value (31250 < 65536 success) if fail, choose bigger pre-scaler.

CS12	CS11	CS10	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	clkI/O/1 (No prescaling)
0	1	0	clkI/O/8 (From prescaler)
0	1	1	clkI/O/64 (From prescaler)
1	0	0	clkI/O/256 (From prescaler)
1	0	1	clkI/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.

Timer modes

Timers can be configured in different modes.

PWM (Pulse width modulation) mode – the OCxy outputs are used to generate PWM signals
CTC (Clear timer on compare match) mode – When the timer counter reaches the compare match register, the timer will be cleared.

Table 16-4. Waveform Generation Mode Bit Description[1]

Mode	WGM13	WGM12 (CTC1)	WGM11 (PWM11)	WGM10 (PWM10)	Timer/Counter Mode of Operation	TOP	Update of OCR1x at	TOV1 Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, Phase Correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, Phase Correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, Phase Correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCR1A	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	BOTTOM	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	BOTTOM	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	BOTTOM	TOP
8	1	0	0	0	PWM, Phase and Frequency Correct	ICR1	BOTTOM	BOTTOM
9	1	0	0	1	PWM, Phase and Frequency Correct	OCR1A	BOTTOM	BOTTOM
10	1	0	1	0	PWM, Phase Correct	ICR1	TOP	BOTTOM
11	1	0	1	1	PWM, Phase Correct	OCR1A	TOP	BOTTOM
12	1	1	0	0	CTC	ICR1	Immediate	MAX
13	1	1	0	1	(Reserved)	–	–	–
14	1	1	1	0	Fast PWM	ICR1	BOTTOM	TOP
15	1	1	1	1	Fast PWM	OCR1A	BOTTOM	TOP

What is an interrupt?

The program running on a controller is normally running sequentially instruction by instruction. An interrupt is an external event that interrupts the running program and runs a special interrupt service routine (ISR). After the ISR has been finished, the running program is continued with the next instruction. Instruction means a single machine instruction, not a line of C or C++ code.

Before an pending interrupt will be able to call a ISR the following conditions must be true:

- Interrupts must be generally enabled
- the according Interrupt mask must be enabled

Interrupts can generally enabled or disabled with the function *interrupts()* or *noInterrupts()*. By default in the Arduino firmware interrupts are enabled. Interrupt masks are enabled / disabled by setting or clearing bits in the Interrupt mask register (TIMSKx).

When an interrupt occurs, a flag in the interrupt flag register (TIFRx) is been set. This interrupt will be automatically cleared when entering the ISR or by manually clearing the bit in the interrupt flag register.

The Arduino functions *attachInterrupt*() and *detachInterrupt*() can only be used for external interrupt pins. These are different interrupt sources, not discussed here.

Timer interrupts

A timer can generate different types of interrupts. The register and bit definitions can be found in the processor data sheet (Atmega328 or Atmega2560) and in the I/O definition header file (iomx8.h for Arduino, iomxx0_1.h for Arduino Mega in the hardware/tools/avr/include/avr folder). The suffix x stands for the timer number (0..5), the suffix y stands for the output number (A,B,C), for example TIMSK1 (Timer1 interrupt mask register) or OCR2A (Timer2 output compare register A).

Timer Overflow:

Timer overflow means the timer has reached is limit value. When a timer overflow interrupt occurs, the timer overflow bit TOVx will be set in the interrupt flag register TIFRx. When the timer overflow interrupt enable bit TOIEx in the interrupt mask register TIMSKx is set, the timer overflow interrupt service routine ISR(TIMERx_OVF_vect) will be called.

Output Compare Match:

When a output compare match interrupt occurs, the OCFxy flag will be set in the interrupt flag register TIFRx . When the output compare interrupt enable bit OCIExy in the interrupt mask register TIMSKx is set, the output compare match interrupt service ISR(TIMERx_COMPy_vect) routine will be called.

Timer Input Capture:

When a timer input capture interrupt occurs, the input capture flag bit ICFx will be set in the interrupt flag register TIFRx. When the input capture interrupt enable bit ICIEx in the interrupt mask register TIMSKx is set, the timer input capture interrupt service routine ISR(TIMERx_CAPT_vect) will be called.

PWM and timer

There is fixed relation between the timers and the PWM capable outputs. When you look in the data sheet or the pinout of the processor these PWM capable pins have names like OCRxA, OCRxB or OCRxC (where x means the timer number 0..5). The PWM functionality is often shared with other pin functionality.

The Arduino has 3 timers and 6 PWM output pins. The relation between timers and PWM outputs is:

- Pins 5 and 6: controlled by Timer0
- Pins 9 and 10: controlled by Timer1
- Pins 11 and 3: controlled by Timer2

On the Arduino Mega we have 6 timers and 15 PWM outputs:

- Pins 4 and 13: controlled by Timer0
- Pins 11 and 12: controlled by Timer1
- Pins 9 and10: controlled by Timer2
- Pin 2, 3 and 5: controlled by timer 3
- Pin 6, 7 and 8: controlled by timer 4
- Pin 46, 45 and 44:: controlled by timer 5

Usefull 3rd party libraries

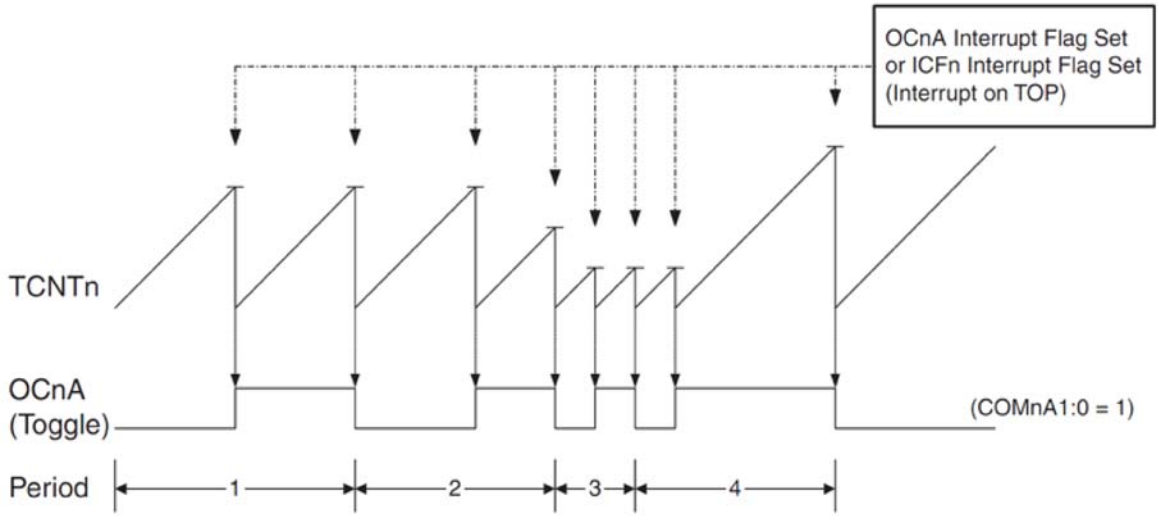
Some 3rd party libraries exists, that uses timers:

- Ken Shirrifs IR library using Timer2 – Send and receive any kind of IR remote signals
- Timer1 and Timer3 library using Timer1 or tiner3 – easy way to write your own timer interrupt service routines.

Examples

Blinking LED with compare match interrupt

The first example uses the Timer1 in CTC mode and the compare match interrupt to toggle a LED. The timer is configured for a frequency of 2Hz. The LED is toggled in the interrupt service routine.



[sourcecode language="cpp"]

```
/*
* timer and interrupts
* Timer1 compare match interrupt example
* more infos: http://blog.oscarliang.net
*/

#define ledPin 13

void setup()
{
pinMode(ledPin, OUTPUT);

// initialize Timer1
noInterrupts(); // disable all interrupts
TCCR1A = 0;
TCCR1B = 0;
TCNT1 = 0;

OCR1A = 31250; // compare match register 16MHz/256/2Hz
TCCR1B |= (1 &lt;&lt; WGM12); // CTC mode
TCCR1B |= (1 &lt;&lt; CS12); // 256 prescaler
TIMSK1 |= (1 &lt;&lt; OCIE1A); // enable timer compare interrupt
interrupts(); // enable all interrupts
}

ISR(Timer1_COMPA_vect) // timer compare interrupt service routine
{
digitalWrite(ledPin, digitalRead(ledPin) ^ 1); // toggle LED pin
}

void loop()
{
// your program here…
}
```

[/sourcecode]

Blinking LED with timer overflow interrupt

same example like before but now we use the timer overflow interrupt. In this case Timer1 is running in normal mode.

The timer must be pre loaded every time in the interrupt service routine.

[sourcecode language="cpp"]

```
/*
* timer and interrupts
* Timer1 overflow interrupt example
* more infos: http://blog.oscarliang.net
*/

#define ledPin 13

void setup()
{
pinMode(ledPin, OUTPUT);

// initialize Timer1
noInterrupts(); // disable all interrupts
TCCR1A = 0;
TCCR1B = 0;

TCNT1 = 34286; // preload timer 65536-16MHz/256/2Hz
TCCR1B |= (1 &lt;&lt; CS12); // 256 prescaler
TIMSK1 |= (1 &lt;&lt; TOIE1); // enable timer overflow interrupt
interrupts(); // enable all interrupts
}

ISR(Timer1_OVF_vect) // interrupt service routine that wraps a user defined function supplied by attachInterrupt
{
TCNT1 = 34286; // preload timer
digitalWrite(ledPin, digitalRead(ledPin) ^ 1);
}
```

```
void loop()
{
// your program here…
}
```

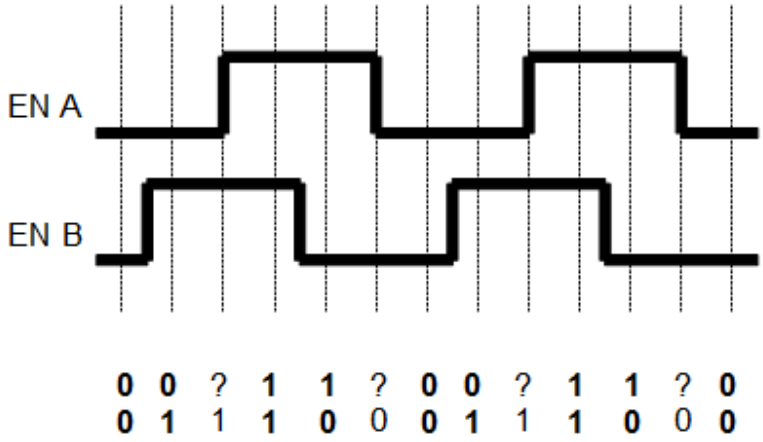
[/sourcecode]

Reading quadrature encoders with a timer

The next example uses Timer2 and the compare match interrupt to read the encoder inputs.

Timer2 is initialized by default to a frequency of 1kHz (1ms period). In the interrupt service routine the state of all encoder pins is read and a state machine is used to eliminate false readings. Using the timer interrupt is much easier to handle than using 4 input change interrupts.

The signals of a quadrature encoder is a 2bit Gray code. Only 1 bit is changing from state to state. A state machine is perfect to check the signal and count the encoder ticks. The timer must be fast enough, to recognize each state change. For the Pololu wheel encoders used here, the 1ms timer is fast enough.



The following example has been modified to work with Arduino V1.x

[sourcecode language="cpp"]

```
/*
* timer and interrupts
```

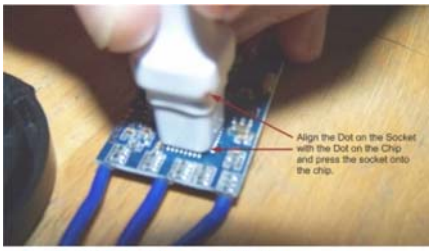
```
* Timer2 compare interrupt example. Quadrature Encoder
* more infos: http://blog.oscarliang.net
*
* Credits:
* based on code from Peter Dannegger
* http://www.mikrocontroller.net/articles/Drehgeber
*/
#if ARDUINO &gt;= 100
#include “Arduino.h”
#else
#include “WConstants.h”
#endif

// Encoder Pins
#define encLtA 2
#define encLtB 3
#define encRtA 11
#define encRtB 12
#define ledPin 13

#define LT_PHASE_A digitalRead(encLtA)
#define LT_PHASE_B digitalRead(encLtB)
#define RT_PHASE_A digitalRead(encRtA)
#define RT_PHASE_B digitalRead(encRtB)

static volatile int8_t encDeltaLt, encDeltaRt;
static int8_t lastLt, lastRt;
int encLt, encRt;

ISR( Timer2_COMPA_vect )
{
int8_t val, diff;

digitalWrite(ledPin, HIGH); // toggle LED pin
val = 0;
if( LT_PHASE_A )
val = 3;
```

```
if( LT_PHASE_B )
val ^= 1; // convert gray to binary
diff = lastLt – val; // difference last – new
if( diff &amp;amp; 1 ){ // bit 0 = value (1)
lastLt = val; // store new as next last
encDeltaLt += (diff &amp;amp; 2) – 1; // bit 1 = direction (+/-)
}

val = 0;
if( RT_PHASE_A )
val = 3;
if( RT_PHASE_B )
val ^= 1; // convert gray to binary
diff = lastRt – val; // difference last – new
if( diff &amp;amp; 1 ){ // bit 0 = value (1)
lastRt = val; // store new as next last
encDeltaRt += (diff &amp;amp; 2) – 1; // bit 1 = direction (+/-)
}
digitalWrite(ledPin, LOW); // toggle LED pin
}
void QuadratureEncoderInit(void)
{
int8_t val;

cli();
TIMSK2 |= (1&lt;&lt;OCIE2A);
sei();
pinMode(encLtA, INPUT);
pinMode(encRtA, INPUT);
pinMode(encLtB, INPUT);
pinMode(encRtB, INPUT);

val=0;
if (LT_PHASE_A)
val = 3;
if (LT_PHASE_B)
val ^= 1;
```

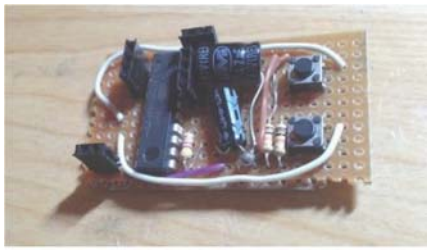
```
lastLt = val;
encDeltaLt = 0;

val=0;
if (RT_PHASE_A)
val = 3;
if (RT_PHASE_B)
val ^= 1;
lastRt = val;
encDeltaRt = 0;

encLt = 0;
encRt = 0;
}

int8_t QuadratureEncoderReadLt( void ) // read single step encoders
{
int8_t val;

cli();
val = encDeltaLt;
encDeltaLt = 0;
sei();
return val; // counts since last call
}

int8_t QuadratureEncoderReadRt( void ) // read single step encoders
{
int8_t val;

cli();
val = encDeltaRt;
encDeltaRt = 0;
sei();
return val; // counts since last call
}

void setup()
```

```
{
Serial.begin(38400);
pinMode(ledPin, OUTPUT);
QuadratureEncoderInit();
}
void loop()
{
encLt += QuadratureEncoderReadLt();
encRt += QuadratureEncoderReadRt();
Serial.print("Lt: ");
Serial.print(encLt, DEC);
Serial.print(" Rt: ");
Serial.println(encRt, DEC);
delay(1000);
}
[/sourcecode]
```

Related



Flash ESC SimonK Firmware Using Arduino Without USBasp adapter
In "DIY and Hacks"



How to Program ATtiny2313 ATtiny4313 using Arduino
In "DIY and Hacks"

Connect Raspberry Pi and Arduino with Serial USB Cable
In "Electronics"

Posted in Robot and tagged arduino, tutorial on 4th February 2013 [http://blog.oscarliang.net/arduino-timer-and-interrupt-tutorial/] . 14 Replies

New Naze32 rev6 | Upgrade of rev5

How to choose Motor and Propeller for Quadcopter and Multicopter

How To Use MOSFET - Beginner's Tutorial

Quadcopter Beginner Guide | Learn to Fly Drones

NEXT >

14 thoughts on “Arduino Timer and Interrupt Tutorial”

Stefan
10th June 2015 at 2:57 pm

Hi,
I need some help. i have 2 libraries that use one timer, How can I change the timer on one of them. I use arduino uno , ethernet shield, rf transmitter and receiver.
These are libraries that use:
#include
#include
#include

This is the function that use a timer.(form VirtualWire.h)

```
void vw_setup(uint16_t speed)
{
uint16_t nticks; // number of prescaled ticks needed
uint8_t prescaler; // Bit values for CS0[2:0]

#ifdef __AVR_ATtiny85__

// figure out prescaler value and counter match value
prescaler = _timer_calc(speed, (uint8_t)-1, &nticks);
```

```
if (!prescaler)
{
return; // fault
}

TCCR0A = 0;
TCCR0A = _BV(WGM01); // Turn on CTC mode / Output Compare pins disconnected

// convert prescaler index to TCCRnB prescaler bits CS00, CS01, CS02
TCCR0B = 0;
TCCR0B = prescaler; // set CS00, CS01, CS02 (other bits not needed)

// Number of ticks to count before firing interrupt
OCR0A = uint8_t(nticks);

// Set mask to fire interrupt when OCF0A bit is set in TIFR0
TIMSK |= _BV(OCIE0A);

#elif defined(__arm__) && defined(CORE_TEENSY)

// on Teensy 3.0 (32 bit ARM), use an interval timer
IntervalTimer *t = new IntervalTimer();
t->begin(TIMER1_COMPA_vect, 125000.0 / (float)(speed));

#else // ARDUINO

// This is the path for most Arduinos
// figure out prescaler value and counter match value
prescaler = _timer_calc(speed, (uint16_t)-1, &nticks);

if (!prescaler)
{
return; // fault
}

TCCR1A = 0; // Output Compare pins disconnected
TCCR1B = _BV(WGM12); // Turn on CTC mode
```

```
// convert prescaler index to TCCRnB prescaler bits CS10, CS11, CS12
TCCR1B |= prescaler;

// Caution: special procedures for setting 16 bit regs
// is handled by the compiler
OCR1A = nticks;

// Enable interrupt

#ifdef TIMSK1

// atmega168
TIMSK1 |= _BV(OCIE1A);

#else

// others
TIMSK |= _BV(OCIE1A);
#endif // TIMSK1

#endif // __AVR_ATtiny85__

// Set up digital IO pins
pinMode(vw_tx_pin, OUTPUT);
pinMode(vw_rx_pin, INPUT);
pinMode(vw_ptt_pin, OUTPUT);
digitalWrite(vw_ptt_pin, vw_ptt_inverted);
}
```

thanks in advance

camiloko
2nd April 2015 at 1:26 am

Hola, Estoy desarrollando un programa con webserver ethertnet shield & arduino mega 2560. Mi problema es que necesito encender un led por 5 min y que se apague por una hora, el

codigo es sencillo si no se utiliza el ethernet pero al utilizarlo no puedo usar delays por que la webserver por obvio razon se cae. ya probe con blinkwhithoutdelay no me da resultado, Algun codigo o sugerencia? gracias!

samtal
31st January 2015 at 7:29 am

Thanks Oscar.
This is a very clear and excellent interrupt explanation.
To save me time re-learning interrupt after years away (I am an 'oldy', from the times we used to write our own headers…), I copied the code to my Arduino Mega1280 board program with the hope it will run as-is.
Unfortunately, it did not.
What I want to achieve is a timely (1 per second) measuring routine, that currently runs free, with delay.

First, I was not sure if I have to add a library. (Probably not. Included in the built-in AVR lib?).
Next, I got some errors that I could clear by changing the semicolons into commas in TCCR1B |= (1 << WGM12); / – Not sure this is correct!
I still have an error: 'It' was not declared in this scope.
And last: I did not see in the interrupt setup any call to the ISR routine by its name. How can I give that ISR my own name?
Thanks

ed
12th April 2015 at 1:40 pm

Samtal, I tried both blink codes compiling for the Mega1280 and both compiled without errors.
I am not sure what you meant by changing semicolons into comma's
Maybe you are confused by a problem many websites have in displaying the so called 'fish hook' characters. if you see < (ampersand lt semicolon) that should be this character '<'
You do not set the interrupt routine to call a name. The interrupt is set by your registers and

then jumps to a predefined name.

Fastcash
16th April 2015 at 1:30 pm

Thanks for the explanations. Your answer helped me to understand that I had a problem with the character '<' .

Isaias
20th August 2014 at 5:31 am

Yeah, I know what it is. Thanks, Oscar!

Isaias
18th August 2014 at 9:18 pm

I know that this may seem pretty basic, but could any of you guys answer what does it mean "| =" ?

Oscar Post author
18th August 2014 at 9:57 pm

do you know what x += y is? it's the same here
x |= y;
is equivalent to
x = x | y;

Bell
13th July 2014 at 12:18 am

That should say ” &_l_t_;_&_l_t_;” or “ampersand ell tee semi ampersand ell tee semi” should be a pair of less than signs.
Apparently, the part that shows the code example takes the HTML literally and the part that shows the comments interprets it.

Bell
13th July 2014 at 12:08 am

There is an error in the “Blinking LED with compare match interrupt” that prevents it from working. It looks like a similar error may be in the other examples.

The constant Timer1_COMPA_vect should be TIMER1_COMPA_vect. This name depends on the model of AVR processor you are using, so check the AVR_GCC interrupts documentation if it still doesn’t work.

Also, what shows up in the example as << should be a pair of less than symbols – but that is probably obvious to anyone reading this page.

Thanks for the example though – despite having to troubleshoot it, it did save a lot of time figuring out how to do interrupt software on the arduino.

Javier
3rd September 2015 at 1:04 pm

Thanks it works for me!! Resume of what is needed to do:

– Changing in code “<” for “<"
– Changing "Timer1_COMPA_vect" for "TIMER1_COMPA_vect"

To do a 1Hz clock it is only needed to change:

OCR1A = 31250; // compare match register 16MHz/256/2Hz
for:
OCR1A = 62500; // compare match register 16MHz/256/1Hz

Alex
12th July 2013 at 2:16 am

Hey i was wondering if somone could provide a quick interupt ctc mode example of an inturrupt that occurs every second thanks :) just as a code example cheers

---

Janelle
27th June 2013 at 2:02 am

So, the servo library uses timer1, how does this affect the pins that you can use it on?

Pingback: Arduino Timer and Interrupt Tutorial -Arduino for Projects