Timers-Arduino (/Timers-Arduino)

Arduino Timers and Interrupts

This complex subject is covered nicely here by RoboFreak from LetsMakeRobots

This tutorial shows the use of timers and interrupts for Arduino boards. As an Arduino programmer you will have used timers and interrupts without detailed knowledge, because all the low level hardware stuff is hidden by the Arduino API. Many Arduino functions use timers, for example the time functions: delay(), millis() and micros() and delayMicroseconds(). The PWM function analogWrite() uses timers, as do the tone() and the noTone() functions. And the Servo library uses timers and interrupts.

The original tutorial is here: http://letsmakerobots.com/node/28278

The Timers Library used in some examples below is available HERE:

If you are mainly interested in changing PWM frequencies See THIS PAGE.

What is a timer?

A timer or to be more precise a timer / counter is a piece of hardware built into the Arduino controller (other controllers have timer hardware, too). 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 prescaler for the timer, or the mode of operation and many other things.

The controller of the Arduino is the Atmel AVR ATMega 168 or ATmega328. 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 timers, where timer1 is a 16bit timer. The most important difference between 8bit and 16bit timer is the timer resolution. 8bits means 256 values where 16bit means 65536 values for higher resolution or longer count.

The controller for the Arduino Mega series is the Atmel AVR ATmega1280 or the ATmega2560. Again, identical but differs in memory size. These controllers have 6 timers. Timer 0, timer1 and timer2 are identical to the ATmega168/328. The timer3, timer4 and timer5 are all 16bit timers, similar to timer1.

All timers depend on the system clock of your Arduino system. Normally the system clock is 16MHz, but for the Arduino Pro 3.3V it 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.

Timer0: 8bit timer.

In the Arduino world timer0 is been used for the software Sketch timer functions, like delay() , millis() and micros() . If you change timer0 registers, this may influence the Arduino timer function. So you should know what you are doing.

Timer1: 16bit timer.

In the Arduino world the Servo library uses timer1 on Arduino Uno (timer5 on Arduino Mega).

Timer2: 8bit timer like timer0.

In the Arduino world the tone() function uses timer2.

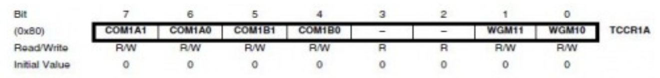
Timer3, Timer4, Timer5: 16bit Timers

Timer 3,4,5 are only available on Arduino Mega boards.

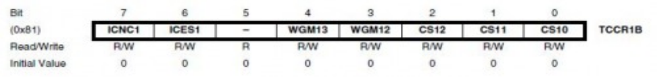
Timer Register

You can change the Timer behaviour through the timer register. The most important timer registers are:

TCCRx - Timer/Counter Control Register. The prescaler can be configured here.



Bit	7	6	5	4	3	2	1	0	
(0x80)	COM1A1	COM1A0	COM1B1	COM1B0	–	–	WGM11	WGM10	TCCR1A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	



Bit	7	6	5	4	3	2	1	0	
(0x81)	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/Write	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	

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 prescaler (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 prescaler.

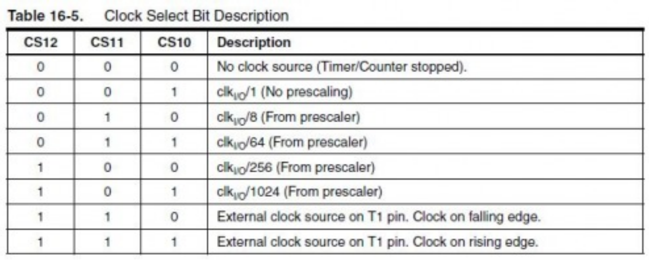


Table 16-5. Clock Select Bit Description

CS12	CS11	CS10	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	$clk_{I/O}$/1 (No prescaling)
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/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.

Timer modes

Timers can be configured in different modes.

PWM mode. (Pulse width modulation mode.): the OCxy outputs are used to generate PWM signals

CTC mode. Clear timer on compare match. When the timer counter reaches the compare match register, the timer will be cleared

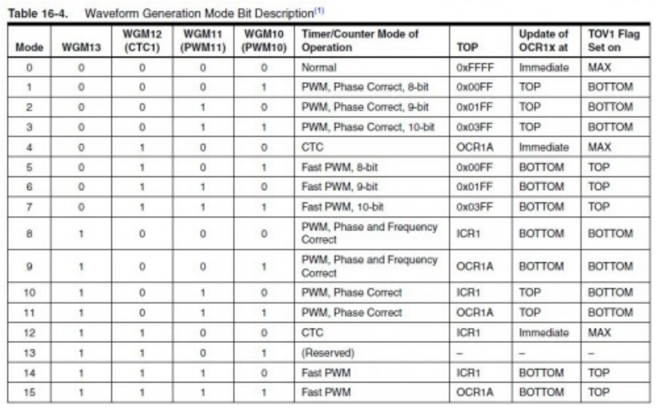


Table 16-4. Waveform Generation Mode Bit Description[(1)]

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 a 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 (globally) enabled / disabled with the function interrupts() / noInterrupts() . By default in the Arduino firmware interrupts are enabled. Interrupt masks are enabled / disabled by setting / 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 timers:

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 3Timers 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

Useful 3rd party libraries:

Some very useful 3rd party libraries exist, that uses timers:

- Ken Shirrifs IR library . Using timer2. Send and receive any kind of IR remote signals like RC5, RC6, Sony
- Timer1 and Timer3 library . Using timer1 or timer3. The easy way to write your own timer interrupt service routines.

RoboFreak has ported these libraries to different timers for Arduino Mega. All ported libraries can be found in the library linked at the top of this page.

Pitfalls:

There exist some pitfalls you may encounter, when programming your Arduino and making use of functions or libraries that uses timers.

- **Servo Library uses Timer1**. You can't use PWM on Pin 9, 10 when you use the Servo Library on an Arduino. For Arduino Mega it is a bit more difficult. The timer needed depends on the number of servos. Each timer can handle 12 servos. For the first 12 servos timer 5 will be used (losing PWM on Pin 44,45,46). For 24 Servos timer 5 and 1 will be used (losing PWM on Pin 11,12,44,45,46).. For 36 servos timer 5, 1 and 3 will be used (losing PWM on Pin 2,3,5,11,12,44,45,46).. For 48 servos all 16bit timers 5,1,3 and 4 will be used (losing all PWM pins).
- **Pin 11 has shared functionality PWM and MOSI**. MOSI is needed for the SPI interface, You can't use PWM on Pin 11 and the SPI interface at the same time on Arduino. On the Arduino Mega the SPI pins are on different pins.
- **tone() function** uses at least timer2. You can't use PWM on Pin 3,11 when you use the tone() function an Arduino and Pin 9,10 on Arduino Mega.

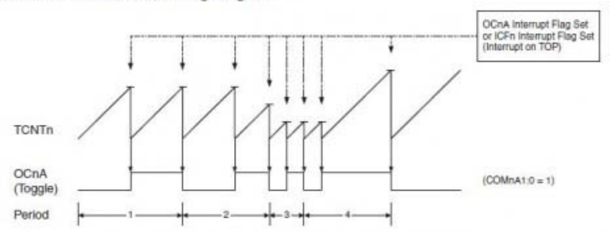
Examples:

Time for some examples.

(1) 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.

Figure 16-6. CTC Mode, Timing Diagram



```
/* Arduino 101: timer and interrupts
   1: Timer1 compare match interrupt example
   more infos: http://www.letmakerobots.com/node/28278
   created by RobotFreak
*/

#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 << WGM12);   // CTC mode
  TCCR1B |= (1 << CS12);    // 256 prescaler
  TIMSK1 |= (1 << 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...
}
```

(2) Blinking LED with timer overflow interrupt

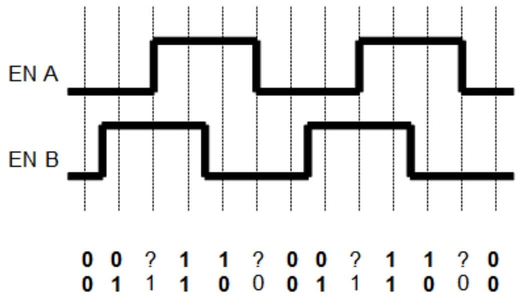
Same example like before but now we use the timer overflow interrupt. In this case timer1 is running in normal mode. The timer must be preloaded every time in the interrupt service routine.

```
/*
 * Arduino 101: timer and interrupts
 * 2: Timer1 overflow interrupt example
 * more infos: http://www.letmakerobots.com/node/28278
 * created by RobotFreak
 */

#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 << CS12);    // 256 prescaler
  TIMSK1 |= (1 << TOIE1);   // enable timer overflow interrupt
  interrupts();             // enable all interrupts
}
// interrupt service routine that wraps a user defined function
// supplied by attachInterrupt
ISR(TIMER1_OVF_vect)
{
  TCNT1 = 34286;            // preload timer
  digitalWrite(ledPin, digitalRead(ledPin) ^ 1);
}

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

(3) Reading quadrature encoders with a timer

The next example is part of RobotFreak's Ardubot project . It 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.



(Example code under construction)...

zz

(https://www.wikispaces.com/user/view/SupremeWa)

Help :)
SupremeWa (https://www.wikispaces.com/user/view/SupremeWa) Jun 10, 2015

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 <VirtualWire.h>

#include <SPI.h>
#include <Ethernet.h>

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