Timer Interrupt Overview

There is a 16Mhz clock that acts as a base clock in Arduino Uno, generally, 16Mhz is too fast for our application so we have to divide it by some number in order to make it useful for our day-to-day applications, that number which we use to divide is known as Prescaler, it helps us in bringing the high-frequency base clock down to match our application.

There are three counter registers in Arduino Uno, namely, Timer0, Timer1, and Timer2.

Timer0 and timer2 are 8 bit timers, meaning they can store a maximum counter value of 255. Timer1 is a 16 bit timer, meaning it can store a maximum counter value of 65535. Once a counter reaches its maximum, it will tick back to zero (this is called overflow). This means at 16MHz, even if we set the compare match register to the max counter value, interrupts will occur every 256/16,000,000 seconds (~16us) for the 8 bit counters, and every 65,536/16,000,000 (~4 ms) seconds for the 16 bit counter. Clearly, this is not very useful if you only want to interrupt once a second.

Instead you can control the speed of the timer counter incrementation by using something called a prescaler. A prescaler dictates the speed of your timer according the the following equation:

(timer speed (Hz)) = (Arduino clock speed (16MHz)) / prescaler

So a 1 prescaler will increment the counter at 16MHz, an 8 prescaler will increment it at 2MHz, a 64 prescaler = 250kHz, and so on. As indicated in the tables above, the prescaler can equal 1, 8, 64, 256, and 1024.

TimerOne Library

Link download: <https://code.google.com/archive/p/arduino-timerone/downloads>

This library is a collection of routines for configuring the 16 bit hardware timer called Timer1 on the ATmega168/328. There are 3 hardware timers available on the chip, and they can be configured in a variety of ways to achieve different functionality. The development of this library began with the need for a way to quickly and easily set the PWM period or frequency, but has grown to include timer overflow interrupt handling and other features. It could easily be expanded upon or ported to work with the other timers.

initialize(period) You must call this method first to use any of the other methods. You can optionally specify the timer's period here (in microseconds), by default it is set at 1 second. Note that this breaks analogWrite() for digital pins 9 and 10 on Arduino.

setPeriod(period) Sets the period in microseconds. The minimum period or highest frequency this library supports is 1 microsecond or 1 MHz. The maximum period is 8388480 microseconds or about 8.3 seconds. Note that setting the period will change the attached interrupt and both pwm outputs' frequencies and duty cycles simultaneously.

pwm(pin, duty, period) Generates a PWM waveform on the specified pin. Output pins for Timer1 are PORTB pins 1 and 2, so you have to choose between these two, anything else is ignored. On Arduino, these are digital pins 9 and 10, so those aliases also work. Output pins for Timer3 are from PORTE and correspond to 2,3 & 5 on the Arduino Mega. The duty cycle is specified as a 10 bit value, so anything between 0 and 1023. Note that you can optionally set the period with this function if you include a value in microseconds as the last parameter when you call it.

attachInterrupt(function, period) Calls a function at the specified interval in microseconds. Be careful about trying to execute too complicated of an interrupt at too high of a frequency, or the CPU may never enter the main loop and your program will 'lock up'. Note that you can optionally set the period with this function if you include a value in microseconds as the last parameter when you call it.

setPwmDuty(pin, duty) A fast shortcut for setting the pwm duty for a given pin if you have already set it up by calling pwm() earlier. This avoids the overhead of enabling pwm mode for the pin, setting the data direction register, checking for optional period adjustments etc. that are mandatory when you call pwm().

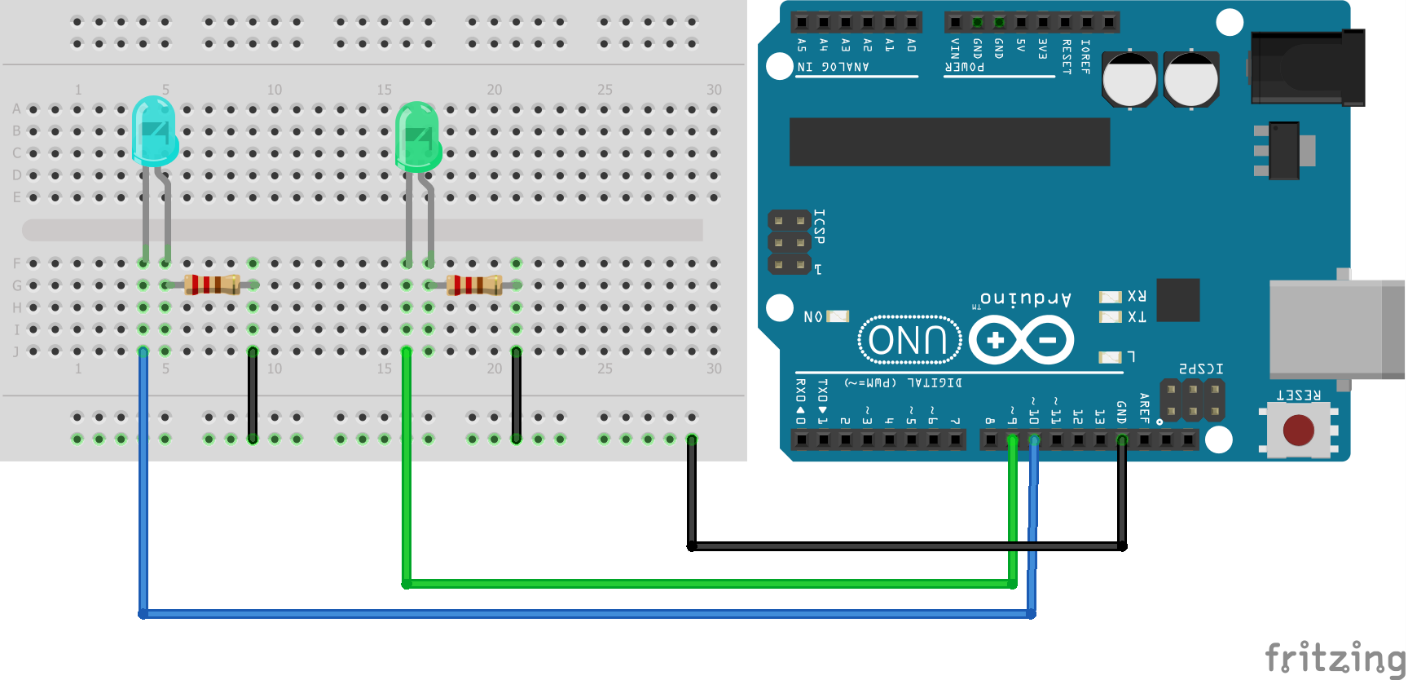
detachInterrupt() Disables the attached interrupt.

disablePwm(pin) Turns PWM off for the specified pin so you can use that pin for something else..

Hardware Required

* Arduino or Genuino Board
* LED
* 330 ohm resistor

Circuit



Code

/\*

\*  Timer1 library example

\*

\*/

#include "TimerOne.h"

void setup(){

 pinMode(10, OUTPUT);

 Timer1.initialize(500000); // initialize timer1, and set a 1/2 second period

 Timer1.pwm(9, 512); // setup pwm on pin 9, 50% duty cycle

 Timer1.attachInterrupt(callback);  // attaches callback() as a timer overflow interrupt

}

void callback(){

 digitalWrite(10, !digitalRead(10));

}

void loop(){

 // your program here...

}