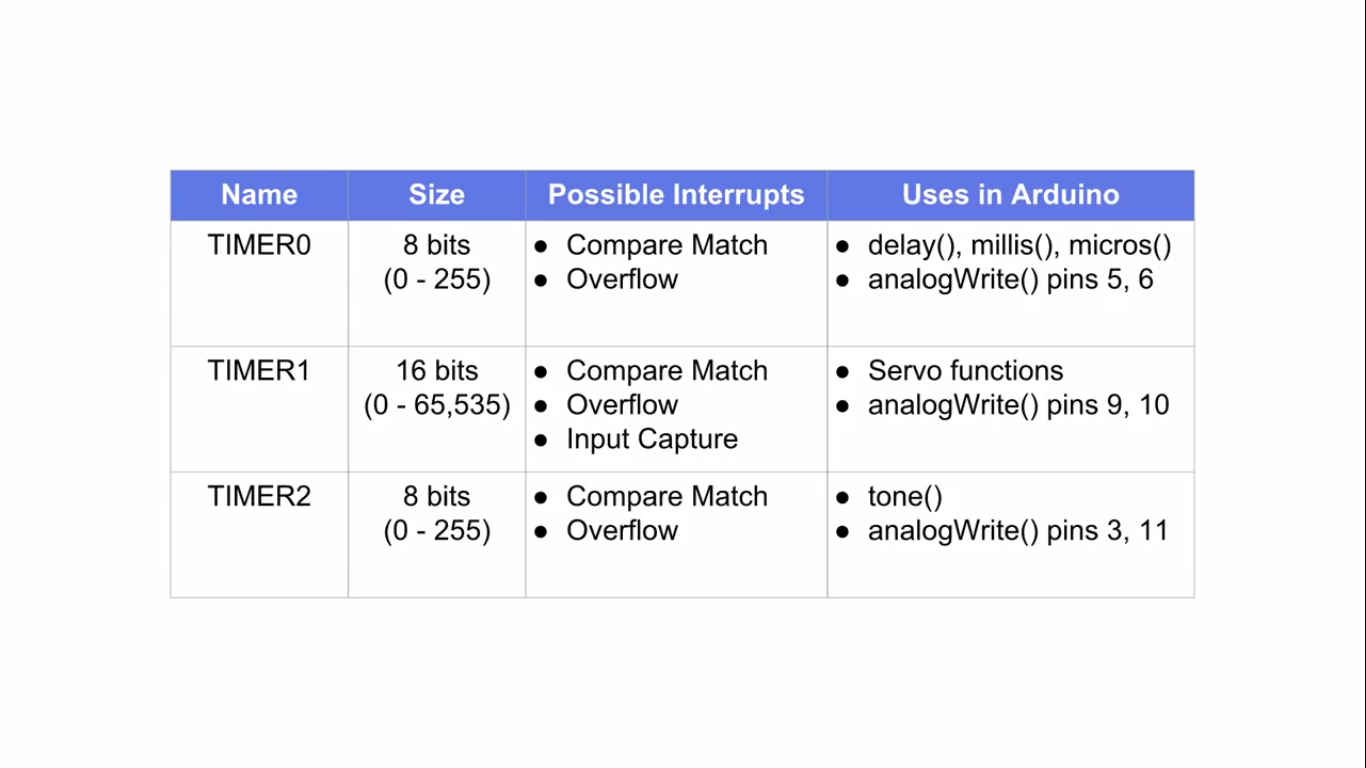
DOCUMENTATION: TIMER INTERRUPTS

A **timer** uses counter which counts at certain speed depending upon the clock frequency. In **Arduino** Uno it takes 1/16000000 seconds or 62nano seconds to make a single count. Meaning **Arduino** moves from one instruction to another instruction for every 62 nano second.

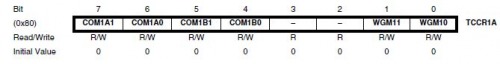
In Short, Generating one time timer interrupt after **'n' number of clock cycles are over**. Generating repetitive timer interrupt.

s



Timer0:  
Timer0 is a 8bit timer.   
In the Arduino world timer0 is been used for the timer functions, like  delay() ,[millis()](http://arduino.cc/en/Reference/Millis?utm_source=rb-community&utm_medium=forum&utm_campaign=arduino-101-timers-and-interrupts) and[micros()](http://arduino.cc/en/Reference/Micros?utm_source=rb-community&utm_medium=forum&utm_campaign=arduino-101-timers-and-interrupts). 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](http://arduino.cc/en/Reference/Servo?utm_source=rb-community&utm_medium=forum&utm_campaign=arduino-101-timers-and-interrupts) uses timer1 on Arduino Uno (timer5 on Arduino Mega).

Timer2:  
Timer2 is a 8bit timer like timer0.   
In the Arduino work the [tone()](http://arduino.cc/en/Reference/Tone?utm_source=rb-community&utm_medium=forum&utm_campaign=arduino-101-timers-and-interrupts) function uses timer2.  
  
Timer3, Timer4, Timer5:  
Timer 3,4,5 are only available on Arduino Mega boards. These timers are all 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 prescaler 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 choosen 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.

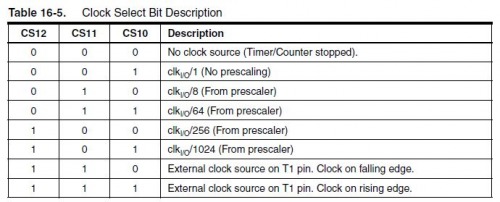
**OR**

**Formula for preloader value for required time in second**:

*TCNTn = 65535 – (16x1010xTime in sec / Prescaler Value)*

**To calculate preloader value for timer1 for time of 2 Sec:**

*TCNT1 = 65535 – (16x1010x2 / 1024) = 34285*

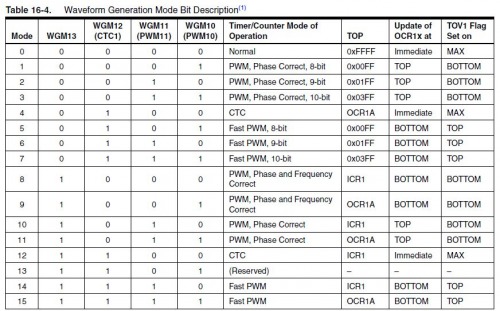


Timer modes:

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



Timer interrupts:

A timer can generate different types of interrupts. The register and bit definitions can be found in the processor data sheet ([Atmega328 261](http://www.atmel.com/dyn/resources/prod_documents/doc8271.pdf?utm_source=rb-community&utm_medium=forum&utm_campaign=arduino-101-timers-and-interrupts) or [Atmega2560 251](http://www.atmel.com/dyn/resources/prod_documents/doc2549.pdf?utm_source=rb-community&utm_medium=forum&utm_campaign=arduino-101-timers-and-interrupts)) 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 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

**Timer Overflow Interrupt:**

Whenever the timer reaches to its maximum value say for example (16 Bit-65535) the *Timer Overflow Interrupt*occurs. So, an ISR interrupt service routine is called when the Timer Overflow Interrupt bit enabled in the TOIEx present in timer interrupt mask register TIMSKx.

**ISR Format:**

**ISR(TIMERx\_OVF\_vect)**

**{**

**}**

**Output Compare Register (OCRnA/B):**

Here when the Output Compare Match Interrupt occurs then the interrupt service ISR (TIMERx\_COMPy\_vect) is called and also OCFxy flag bit will be set in TIFRx register. This ISR is enabled by setting enable bit in OCIExy present in TIMSKx register. Where TIMSKx is Timer Interrupt Mask Register.

**Timer Input Capture:**

Next when the timer Input Capture Interrupt occurs then the interrupt service ISR (TIMERx\_CAPT\_vect) is called and also the ICFx flag bit will be set in TIFRx (Timer Interrupt Flag Register). This ISR is enabled by setting the enable bit in ICIEx present in TIMSKx register.

Then set the preloader value 3035 for 4 seconds. Check the formula above to calculate the preloader value.

**float value = 3035;**

Next in void setup()

Next set the LED pin as OUTPUT pin and the Push buttons are set as INPUT pins

**pinMode(ledPin, OUTPUT);**

**pinMode(2,INPUT);**

**pinMode(4,INPUT);**

Next disable all the interrupts:

**noInterrupts();**

Next the Timer1 is initialized.

**TCCR1A = 0;**

**TCCR1B = 0;**

The preloader timer value is set (Initially as 3035).

**TCNT1 = value;**

Then the Pre scaler value 1024 is set in the TCCR1B register.

**TCCR1B |= (1 << CS10)|(1 << CS12);**

The Timer overflow interrupt is enabled in the Timer Interrupt Mask register so that the ISR can be used.

**TIMSK1 |= (1 << TOIE1);**

At last all interrupts are enabled.

**interrupts();**

Now write the ISR for Timer Overflow Interrupt which is responsible for turning LED ON and OFF usingdigitalWrite. The state changes whenever the timer overflow interrupt occurs.

**ISR(TIMER1\_OVF\_vect)**

**{**

**TCNT1 = value;**

**digitalWrite(ledPin, digitalRead(ledPin) ^ 1);**

**}**

In the void loop() the value of preloader is incremented or decremented by using the push button inputs

**if(digitalRead(2) == HIGH)**

**{**

**value = value+10; //Incement preload value**

**}**

**if(digitalRead(4)== HIGH)**

**{**

**value = value-10; //Decrement preload value**

**}**

So this is how a timer can be used to produce delay in Arduino program.

**Advantage of timer (peripheral) in microcontroller** is developers can accomplish lots of — timing/ counting/ delay related operations — very elegantly.

Whole philosophy behind their use is let hardware and registers of timer/counter peripheral do the job. We need not worry about handing delay/ timer/ counter related operations by writing custom code.

Instead we write code to use of timer/counter by doing the following:

1. Configure the timer / counter peripheral’s configuration registers
2. Initialize counters / timers
3. Monitor interrupt flags or handle interrupt via interrupt service subroutines
4. Access counter / timers , counted data.

By doing so microcontroller is free to execute other piece of code.

Timer in microcontroller can be used for following purpose:

1. Generating one time timer interrupt after ’n’ number of clock cycles are over.
2. Generating repetitive timer interrupt
3. Counting external pins pulse width
4. In some controllers — setting baud rates
5. In some controllers — Generating PWM is connected with timer.
6. Measure external pulse width
7. Time ticks for synchronizing operations of tiny real time micro controller’s OS
8. etc.

**Disadvantages:**

1. No other code (except perhaps other interrupt routines) can be executed.
2. CPU-time is wasted: If no other work must be processed it is still more efficient to set some powersaving-state and let a timer interrupt wake it up in time.

**Reference:**

[Arduino Timer Tutorial](https://circuitdigest.com/microcontroller-projects/arduino-timer-tutorial) :

<https://circuitdigest.com/microcontroller-projects/arduino-timer-tutorial> *AND*

<https://www.robotshop.com/community/forum/t/arduino-101-timers-and-interrupts/13072>