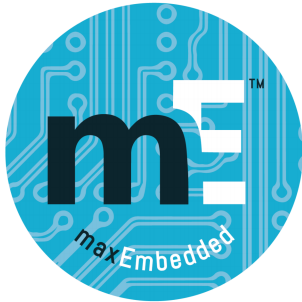


[maxEmbedded Index](#)[Categories »](#)[Tools](#)

Search This Site...



a guide

# Embedded

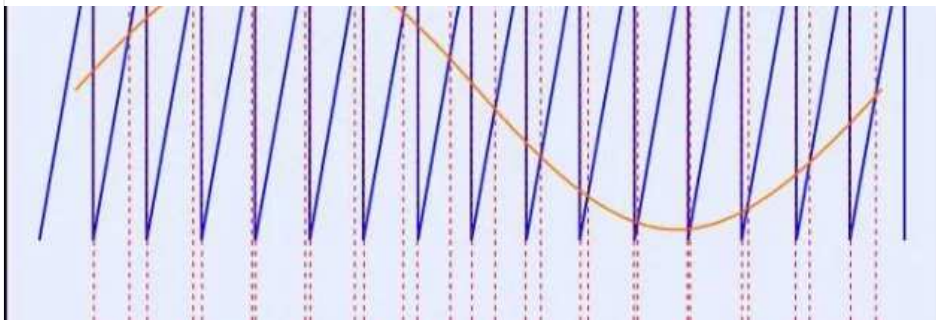
embedded

computer vision

[Home](#)[mE Index](#)[Getting Started](#)[Atmel AVR »](#)[Electronics](#)[More »](#)[About »](#)[Home](#)[Atmel AVR](#)

AVR Timers – PWM Mode – Part II

Posted by [Mayank](#) on Jan 7, 2012 in [Atmel AVR](#), [Microcontrollers](#) | [153 comments](#)



## AVR Timers – PWM Mode – Part II

This article is in continuation with the previous PWM post. Learn how to program the timers to operate in PWM mode! So let's begin!



Hello folks! Long time no see! :)

In my [previous post](#), we have discussed the basic concepts of PWM. Let's summarize it first:

- PWM stands for Pulse Width Modulation.
- It can be generated by comparing predetermined waveform with a reference voltage level or by making simple analog circuits.
- Duty Cycle of a PWM waveform is given by the following relation.

Search maxEmbedded

Search for:

Search

Popular

Recent

Random

MON  
20

### [The ADC of the AVR](#)

Posted by Mayank in Atmel AVR, Microcontrollers

FRI  
24

### [AVR Timers – TIMERO](#)

Posted by Mayank in Atmel AVR, Microcontrollers

TUE  
06

### [RF Module Interfacing without Microcontrollers](#)

Posted by Mayank in Electronics

THU  
16

### [LCD Interfacing with AVR](#)

Posted by Mayank in Atmel AVR, Microcontrollers

THU  
20

### [Making an RF Car](#)

Posted by Yash in Electronics, Robotics

Browse maxE by Categories

Browse maxE by Categories

Select Category



$$\text{Duty Cycle} = \frac{T_{on}}{T_{on} + T_{off}} \times 100 \%$$

- There are three modes of PWM operation – Fast PWM, Phase Correct PWM and Frequency and Phase Correct PWM
- How to choose timer, operation mode and compare output mode for generating the desired PWM.

So now, without much hassle, let's see how to implement it using the AVR microcontrollers. Before we proceed, I suggest you to go through my previous posts on [Timers](#) and [PWM](#).

## Problem Statement

Let us take a problem statement. We need to generate a 50 Hz PWM signal having 45% duty cycle.

## Analysis

Given that

Frequency = 50 Hz

In other words, the time period, T

$T = T(\text{on}) + T(\text{off}) = 1/50 = 0.02 \text{ s} = 20 \text{ ms}$

Also, given that

Duty Cycle = 45%

Thus, solving according to equation given above, we get

$T(\text{on}) = 9 \text{ ms}$   
 $T(\text{off}) = 11 \text{ ms}$

Now, this can be achieved in two ways:

1. Use Timer in CTC Mode
2. Use Timer in PWM Mode

## Methodology – CTC Mode

Okay, so I won't be writing any code here (just the pseudo code). I assume that after reading my [previous posts](#), you are smart enough to write one yourself! We will discuss only the concepts.

Firstly, choose a suitable timer. For this application, we can choose any of the three timers available in ATMEGA32. Choose a suitable prescaler. Then set up the timer and proceed as usual. The catch lies here is that you need to update the compare value of OCRx register everytime. One such way is discussed in the pseudo code given below.

This is **analogous** to the traditional LED flasher, except the fact that the on and off times are different.

## Pseudo Code

```

1  #include <avr/io.h>
2  #include <avr/interrupt.h>
3
4  uint8_t count = 0;           // global counter
5
6  // initialize timer, interrupt and variable
7  void timerX_init()
8  {
9      // set up timerX with suitable prescaler and CTC mode
10     // initialize counter
11     // initialize compare value
12     // enable compare interrupt
13     // enable global interrupts

```

Email Address

Subscribe

[Like maxE on Facebook](#)



Sponsored Links

[i](#) [x](#)

```

15
16 // process the ISR that is fired
17 ISR (TIMERx_COMPA_vect)
18 {
19     // do whatever you want to do here
20     // say, increment the global counter
21     count++;
22
23     // check for the global counter
24     // if count == odd, delay required = 11 ms
25     // if count == even, delay required = 9 ms
26     // thus, the value of the OCRx should be constantly up
27     if (count % 2 == 0)
28         OCRx = 9999; // calculate and substitute appr
29     else
30         OCRx = 10999; // calculate and substitute appr
31 }
32
33 int main(void)
34 {
35     // initialize the output pin, say PC0
36     DDRC |= (1 << 0);
37
38     // initialize timerX
39     timerX_init();
40
41     // loop forever
42     while(1)
43     {
44         // do nothing
45     }
46 }
47

```

Now this is one method. And it's very inefficient. You can increase its efficiency by writing a better C code (syntax-wise), however the concept remains the same. If you have any other method/concept, you are most welcome to share it here! :)

UPDATE: One of the readers of maxEmbedded, "coolpales" has written [this](#) code, and it worked for him.

Please note that this code not tested yet! So, if any of you is trying it out, do post your results here, I would be happy to see them! :)

## Methodology – PWM Mode

Okay, so now lets learn about the PWM mode. The PWM Mode in AVR is hardware controlled. This means that everything, by *everything* I mean "everything", is done by the AVR CPU. All you need to do is to initialize and start the timer, and set the duty cycle! Cool, eh?! Let's learn how!

Here, I have used Timer0 of ATMEGA32 for demonstration. You can choose any other other timer or AVR microcontroller as well. Now let's have a look at the registers.

### TCCR0 – Timer/Counter0 Control Register

We have come across this register in my [Timer0 tutorial](#). Here, we will learn how to set appropriate bits to run the timer in PWM mode.

Bit	7	6	5	4	3	2	1	0	
	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	TCCR0
Read/Write	W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

TCCR0 Register

We will discuss only those bits which are of interest to us now.

- **Bit 6,3 – WGM01:0 – Waveform Generation Mode** – These bits can be set to either “00” or “01” depending upon the type of PWM you want to generate. Here’s the look up table.

Mode	WGM01 (CTC0)	WGM00 (PWM0)	Timer/Counter Mode of Operation	TOP	Update of OCR0	TOV0 Flag Set-on
0	0	0	Normal	0xFF	Immediate	MAX
1	0	1	PWM, Phase Correct	0xFF	TOP	BOTTOM
2	1	0	CTC	OCR0	Immediate	MAX
3	1	1	Fast PWM	0xFF	TOP	MAX

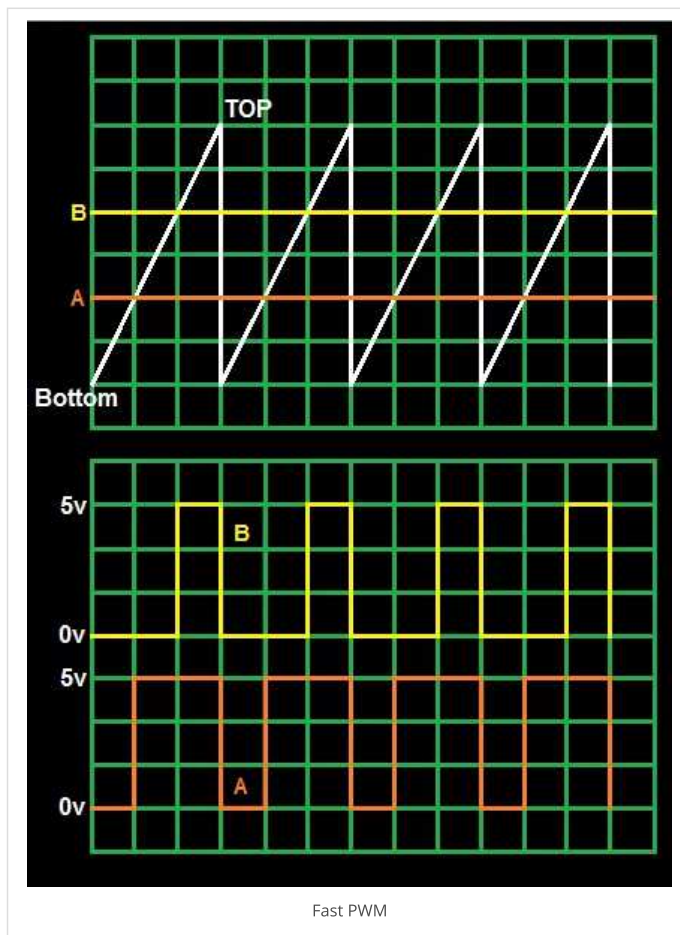
Waveform Generation Mode Bit Description

- **Bit 5,4 – COM01:0 – Compare Match Output Mode** – These bits are set in order to control the behavior of Output Compare pin (OC0, pin 4 in ATMEGA32) in accordance with the WGM01:0 bits. The following look up table determine the operations of OC0 pin for Fast PWM mode.

COM01	COM00	Description
0	0	Normal port operation, OC0 disconnected.
0	1	Reserved
1	0	Clear OC0 on compare match, set OC0 at TOP
1	1	Set OC0 on compare match, clear OC0 at TOP

Compare Output Mode, Fast PWM Mode

Now lets have a look at the Fast PWM waveforms. Detailed explanation can be found in my [previous tutorial](#).



Fast PWM

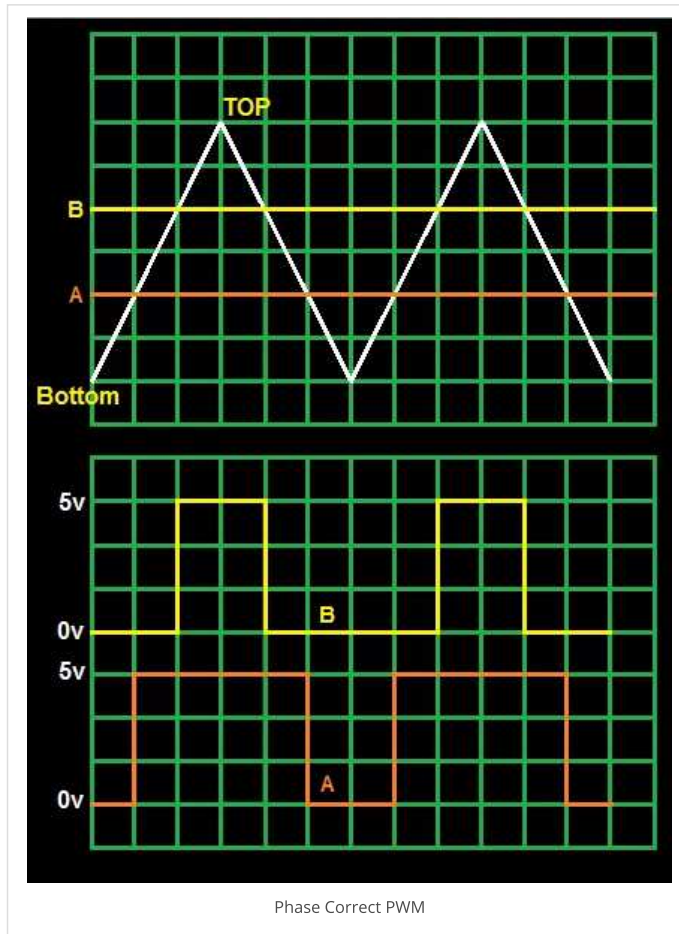
Now let me remind you that the AVR PWM is fully hardware controlled, which means that even the timer compare operation is done by the AVR CPU. All we need to do is to *tell* the CPU *what* to do once a match occurs. The COM01:0 pins come into play here. We see that by setting it to “10” or

determines whether the PWM is in inverted mode, or in non-inverted mode).

Similarly for Phase Correct PWM, the look up table and the waveforms go like this.

COM01	COM00	Description
0	0	Normal port operation, OC0 disconnected.
0	1	Reserved
1	0	Clear OC0 on compare match when up-counting. Set OC0 on compare match when downcounting.
1	1	Set OC0 on compare match when up-counting. Clear OC0 on compare match when downcounting.

Compare Output Mode, Phase Correct PWM Mode



Phase Correct PWM

Even here, setting COM01:0 to "10" or "11" determines the behavior of OC0 pin. As shown in the waveforms, there are two instances – one during up-counting, and other during down-counting. The behavior is clearly described in the look up table.

Please note that OC0 is an output pin. Thus, the effects of WGM and COM won't come into play unless the DDRx register is set properly. Refer [this](#) tutorial for more info.

- **Bit 2:0 – CS02:0 – Clock Select Bits** – These bits are already discussed in Timer0 tutorial.

## OCR0 – Output Compare Register

We have come across even this register in my [Timer0 tutorial](#). We use this register to store the compare value. But when we use Timer0 in PWM mode, the value stored in it acts as the duty cycle (obviously!). In the problem statement, it's given that the duty cycle is 45%, which means

$OCR0 = 45\% \text{ of } 255 = 114.75 = 115$

And that's it! Now we are ready to write a code for it! :)

## Edit: Note

The following code discusses how to create a PWM signal of a desired duty cycle. If you wish to change its frequency, you need to alter the TOP value, which can be done using the ICRx register (which is not supported by 8-bit timers). For 16-bit Timer1, it can be varied using ICR1A. I will discuss about this soon when we discuss about servo control.

## Code

So here goes the code. To learn about I/O port operations in AVR, view [this](#). To know about bit manipulations, view [this](#). To learn how to use AVR Studio 5, view [this](#). To learn how this code is structured, view the [previous TIMER0 post](#).

```

1  #include <avr/io.h>
2  #include <util/delay.h>
3  void pwm_init()
4  {
5      // initialize TCCR0 as per requirement, say as follows
6      TCCR0 |= (1<<WGM00)|(1<<COM01)|(1<<WGM01)|(1<<CS00);
7
8      // make sure to make OC0 pin (pin PB3 for atmega32) as
9      DDRB |= (1<<PB3);
10 }
11
12 void main()
13 {
14     uint8_t duty;
15     duty = 115;      // duty cycle = 45% of 255 = 114.75
16
17     // initialize timer in PWM mode
18     pwm_init();
19
20     // run forever
21     while(1)
22     {
23         OCR0 = duty;
24     }
25 }
```

## Problem Statement

So now, let's take another problem statement. This one is going to be a more of a practical stuff unlike the previous one!

Let's take the traditional LED flasher where we need to blink an LED at a particular frequency. But hey, wait, didn't we discuss it long back in [this](#) post (scroll down towards the end)? Hmm, so let's modify it so as to incorporate PWM. Unlike the traditional LED flasher (where LEDs are either ON or OFF), let's make it glow at the maximum brightness, and then slowly decrease its brightness till it reaches zero, and then again increase its brightness slowly till it becomes maximum.

## Analysis and Code

So how do we do it? Yes, you guessed it right! Decrease the duty cycle slowly from 255 to zero, and then increase it from zero to 255. Depending upon the duty cycle, the voltage applied to the LED varies, and thus the brightness. The following formula gives the relation between voltage and duty cycle.

$$V_{out} = \frac{\text{Duty Cycle}}{255} \times 5 \text{ volts}$$



So here goes the code. I won't explain it, you can decode it yourself. To learn about I/O port operations in AVR, view [this](#). To know about bit manipulations, view [this](#). To learn how to use AVR Studio 5, view [this](#). To learn how this code is structured, view the [previous TIMER0 post](#).

```

1  // program to change brightness of an LED
2  // demonstration of PWM
3
4  #include <avr/io.h>
5  #include <util/delay.h>
6
7  // initialize PWM
8  void pwm_init()
9  {
10     // initialize timer0 in PWM mode
11     TCCR0 |= (1<<WGM00)|(1<<COM01)|(1<<WGM01)|(1<<CS00);
12
13     // make sure to make OC0 pin (pin PB3 for atmega32) as
14     DDRB |= (1<<PB3);
15 }
16
17 void main()
18 {
19     uint8_t brightness;
20
21     // initialize timer0 in PWM mode
22     pwm_init();
23
24     // run forever
25     while(1)
26     {
27         // increasing brightness
28         for (brightness = 0; brightness < 255; ++brightness)
29         {
30             // set the brightness as duty cycle
31             OCR0 = brightness;
32
33             // delay so as to make the user "see" the chan
34             _delay_ms(10);
35         }
36
37         // decreasing brightness
38         for (brightness = 255; brightness > 0; --brightness)
39         {
40             // set the brightness as duty cycle
41             OCR0 = brightness;
42
43             // delay so as to make the user "see" the chan
44             _delay_ms(10);
45         }
46
47         // repeat this forever
48     }
49 }

```

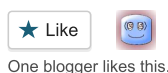
So here ends my another much awaited and long tutorial! Next up.. [Serial Communication](#)! See you around!! :)

And yeah, if you have any suggestions, doubts, constructive criticisms, etc, you are most welcome to drop a note below! **Subscribe to my blog or grab the RSS Feeds to stay updated!**

Loved it? Share it!



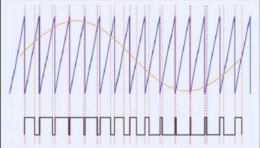
Like this:



Join our newsletter today for free.

[Subscribe Now](#)






AVR Timers - PWM Mode - Part I

August 7, 2011

32 Shares

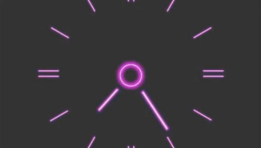
In "Atmel AVR"



AVR Timers - CTC Mode

July 14, 2011

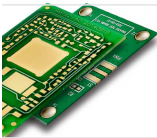
In "Atmel AVR"



Introduction to AVR Timers

June 22, 2011

In "Atmel AVR"



10pcs only \$10

China PCB——1-3 days shipping, Reg

Comments

er Comments

Srikar June 11, 2015

In the above LED Brightness code, OCR0 is given Brightness value. Is the Brightness value the MAX value and very 10ms the max is varied and hence we get varying wave even 10ms?  
Is this right?

REPLY



Srikar June 11, 2015

Am sorry that is not MAX, it is TOP\*.

REPLY



Suraj June 24, 2015

Hi Mayank,

Thank you for the wonderful tutorials. In fast PWM mode i could generate time periods from the range of nano seconds to seconds. But for an application in need to exactly fire 25 PWM pulses of some time period. If i give a variable inside the while loop and increment it for 25 times and then stop the timer, will i exactly get 25 pulses or is there any other way to count the number of pulses using the code or the interrupt functions.

Thank you in advance.





mridul kant September 22, 2015

how can i generate 50hz square clock pulse using timer,  
and how can i phase shifted clock pulse using ??  
please help  
thanks

[REPLY](#)



looka October 18, 2015

it worked with me as blinking LED !!

[REPLY](#)



Prashant Agarwal November 24, 2015

Thanks!

Keep Reading, Keep Sharing. :D

[REPLY](#)



deepakaagri November 30, 2015

Hello

i am using ATMEGA32U4 (arduino board ), i am facing problem with TIMER  
, i want to generate 1 Hz to 40 Hz signal in Two pins and change PWM  
signal with certain period of time .

example –

TIMER | PIN9 Freq. | PIN9 PWM | PIN10 Freq. | PIN10 PWM | Alternate  
Mode

T0 | 2Hz | 50% | 2Hz | 50% | 0

T1 | 3Hz | 50% | 15Hz | 20% | 0

T2 | 30Hz | 40% | 2Hz | 50% | 0

T3 | 2Hz | 50% | 2Hz | 50% | 1

#Alternate Mode activate 180 degree phase shift b/w PIN9 and PIN10 .

please help me /.

[REPLY](#)



Levi January 27, 2016

Hi there

In your article, you do a wonderful job of describing how to change the duty cycle of a PWM pulse, but there doesn't seem to be any mention of how to control the frequency of the pulse. If you find the time, could you possibly explain how to do this? It would be much appreciated

Thanks :)

[REPLY](#)



Levi January 27, 2016

Nevermind, I missed your edit in the article. Thanks anyway :)

[REPLY](#)



M Ahmed April 16, 2016

hi, i need to use timer-0 as overflow timer and as in PWM mode as the same time. is it possible logically. please let me know anyone if its right what i want to do.

[REPLY](#)



Max May 4, 2016

No, you can't use them in both the modes at the same time. Use different timers, or you'll have to generate PWM in software, which won't be effective.

[REPLY](#)



lolomolo May 2, 2016

cheers mate, thanks for the tutorials!

[REPLY](#)



Thanks for your wonderful tutorial. I want to control the speed of stepper motor with pwm and variable frequency (my stepper driver I297/I298). Is there any possibility to do this? I read the data sheet of atmega2560 but i didnt find about variable frequency its only about variable duty cycle and fixed frequency. could you give me some hint?

thanks alot!!!

[REPLY](#)

Max May 31, 2016

You can't generate variable frequency directly. You need to do some hack in the software and change frequency in every iteration in the software. Not sure if this'll help, but check this out:

<http://www.oxgadgets.com/2011/04/creating-a-variable-frequency-pwm-output-on-arduino-uno.html>

[REPLY](#)

amiteshsingh June 28, 2016

how to do pwm using ctc mode:

[https://raw.githubusercontent.com/amitesh-singh/amiduino/master/avr/avr\\_programming/pwm/usingctc/ctc.c](https://raw.githubusercontent.com/amitesh-singh/amiduino/master/avr/avr_programming/pwm/usingctc/ctc.c)

[REPLY](#)

Mayank July 5, 2016

PWM and CTC are two different modes of the timer. If you want to generate PWM signal using CTC, then you'll have to manually toggle the pin value at regular intervals (or interrupts) generated by the CTC.

[REPLY](#)

[← Older Comments](#)

## Leave a Reply

Enter your comment here...

## Copyright



maxEmbedded by [Mayank Prasad](#)  
is licensed under a [Creative Commons Attribution-NonCommercial-NoDerivs 3.0 Unported License](#).

© Copyright 2011-2014 by  
maxEmbedded. Some rights  
reserved.

## Tags

[accelerometer](#) [adapter](#) [adc](#) [assembly](#) [avr](#)  
[adc](#) [avr](#) [basics](#) [avr eeprom](#) [avr](#)  
[interrupts](#) [avr serial](#) [avr timers](#)  
[basics](#) [bootloader](#) [code gallery](#) [code](#)  
[optimization](#) [color spaces](#) [counters](#) [cv](#)  
[basics](#) [filter circuit](#) [i2c](#) [ir sensor](#) [matlab](#)  
[msp430](#) [basics](#) [opencv](#) [pcb design](#)  
[power supply](#) [printed circuit boards](#) [pwm](#)  
[raspberry pi](#) [rf communication](#) [rf](#)  
[module](#) [robotics](#) [robot locomotion](#) [rs232](#)  
[sbc](#) [basics](#) [sensors](#) [Serial](#) [spi](#) [ssd](#)  
[timers](#) [toolchain](#) [transformer](#) [uart](#) [usart](#)  
[voltage regulator](#) [wireless](#)

## Top Posts &amp; Pages

The ADC of the AVR

AVR Timers - TIMERO

The USART of the AVR

The SPI of the AVR

How to build your own  
Power Supply

AVR Timers - CTC Mode

AVR Timers - PWM Mode -  
Part I

maxEmbedded is a free and open source platform to share knowledge and learn about the concepts of robotics, embedded systems and computer vision. People from around the world who are enthusiastic about these topics and willing to support the open source community are invited to share their information, knowledge and expertise by means of written tutorials and videos on the website.