Posted by Max on Jul 14, 2011 in Atmel AVR, Microcontrollers | 85 comments

AVR Timers – CTC Mode

Hello friends! Welcome to the another tutorial on AVR Timers. Till now, we have covered the following topics in AVR Timers:

- Introduction to AVR Timers
- 8-bit TIMER0
- 16-bit TIMER1
- 8-bit TIMER2

The basic concepts of timers and its applications have been discussed in earlier posts. In this post, we will discuss about a special mode of operation – **Clear Timer on Compare (CTC) Mode**.

Search maxEmbedded

Search for:
Search

Popular Recent Random

MON 20 The ADC of the AVR
Posted by Max in Atmel AVR, Microcontrollers

TUE 06 RF Module Interfacing without Microcontrollers
Posted by Max in Electronics

CTC Mode

So till now, we have dealt with the basic concepts. We had two timer values with us – **Set Point** (SP) and **Process Value** (PV). In every iteration, we used to compare the process value with the set point. Once the process value becomes equal (or exceeds) the set point, the process value is reset. The following code snippet explains it:

```
1  max = 39999;   // max timer value set  <--- set point
2
3  // some code here
4  // ...
5  // ...
6  // ...
7
8  // TCNT1 <--- process value
9  if (TCNT1 >= max)   // process value compared with the se
10 {
11     TCNT1 = 0;      // process value is reset
12 }
13
14 // ...
```

Here, we have used the example of TIMER1. Since TIMER1 is a 16-bit timer, it can count upto a maximum of 65535. Here, what we desire is that the timer (process value) should reset as soon as its value becomes equal to (or greater than) the set point of 39999.

So basically, the CTC Mode implements the same thing, but unlike the above example, it implements it in hardware. Which means that we no longer need to worry about comparing the process value with the set point every time! This will not only avoid unnecessary wastage of cycles, but also ensure greater accuracy (i.e. no missed compares, no double increment, etc).

Hence, this comparison takes place in the hardware itself, inside the AVR CPU! Once the process value becomes equal to the set point, a flag in the status register is set and the timer is reset automatically! Thus goes the name – CTC – Clear Timer on Compare! Thus, all we need to do is to take care of the flag, which is much more faster to execute.

Let us analyze this CTC Mode in detail with the help of a problem statement.

Problem Statement

Let's take up a problem statement to understand this concept. We need to flash an LED every 100 ms. We have a crystal of XTAL 16 MHz.

Methodology – Using CTC Mode

Before proceeding any further, let's jot down the formula first. I also recommend you to read my TIMER0 tutorial in order to understand this better. I won't be revising the basic concepts here, just their application.

$$Timer\ Count = \frac{Required\ Delay}{Clock\ Time\ Period} - 1$$

Now, given XTAL = 16 MHz, with a prescaler of 64, the frequency of the clock pulse reduces to 250 kHz. With a Required Delay = 100 ms, we get the Timer

THU 16 LCD Interfacing with AVR
Posted by Max in Atmel AVR, Microcontrollers

FRI 24 AVR Timers – TIMER0
Posted by Max in Atmel AVR, Microcontrollers

FRI 10 I/O Port Operations in AVR
Posted by Max in Atmel AVR, Microcontrollers

Count to be equal to 24999. Up until now, we would have let the value of the timer increment, and check its value every iteration, whether it's equal to 24999 or not, and then reset the timer. Now, the same will be done in hardware! We won't check its value every time in software! We will simply check whether the flag bit is set or not, that's all. Confused, eh? Well, don't worry, just read on!

Okay, so now let me introduce you to the register bits which help you to implement this CTC Mode.

TCCR1A and TCCR1B Registers

The **Timer/Counter1 Control Register A** – TCCR1A Register is as follows:

Bit	7	6	5	4	3	2	1	0	
	COM1A1	COM1A0	COM1B1	COM1B0	FOC1A	FOC1B	WGM11	WGM10	TCCR1A
Read/Write	R/W	R/W	R/W	R/W	W	W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

TCCR1A Register

The **Timer/Counter1 Control Register B** – TCCR1B Register is as follows:

Bit	7	6	5	4	3	2	1	0	
	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	

TCCR1B Register

You are already aware of the **Clock Select Bits – CS12:0** in TCCR1B (if not, view the TIMER1 post, scroll down a bit and you will find it there). Hence, right now, we are concerned with the **Wave Generation Mode Bits – WGM13:0**. As you can see, these bits are spread across both the TCCR1 registers (A and B). Thus we need to be a bit careful while using them. Their selection is as follows:

Mode	WGM13	WGM12 (CTC1)	WGM11 (PWM11)	WGM10 (PWM10)	Timer/Counter Mode of Operation	TOP	Update of OCR1x	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	TOP	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	TOP	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	TOP	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	TOP	TOP
15	1	1	1	1	Fast PWM	OCR1A	TOP	TOP

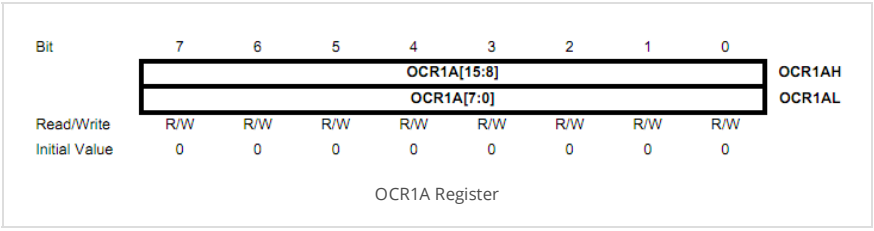
Wave Generation Mode Bit Description

We can see that there are two possible selections for CTC Mode. Practically, both are the same, except the fact that we store the timer compare value in different registers. Right now, let's move on with the first option (0100). Thus, the initialization of TCCR1A and TCCR1B is as follows.

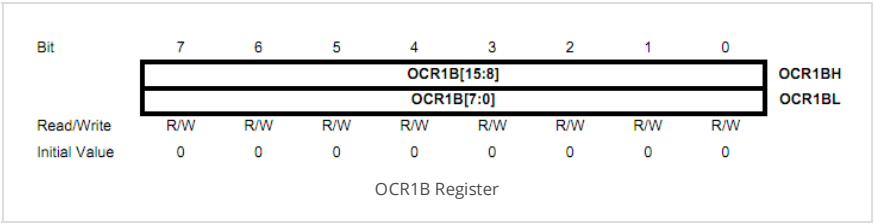
```
TCCR1A |= 0;
// not required since WGM11:0, both are zero (0)

TCCR1B |= (1 << WGM12)|(1 << CS11)|(1 << CS10);
// Mode = CTC, Prescaler = 64
```

OCR1A and OCR1B Registers

Now that we have set the CTC Mode, we must tell the AVR to reset the timer as soon as its value reaches such and such value. So, the question is, how do we set such and such values? The **Output Compare Register 1A** – OCR1A and the **Output Compare Register 1B** – OCR1B are utilized for this purpose.



OCR1A Register



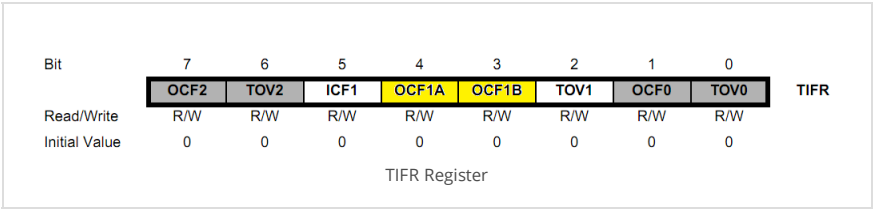
OCR1B Register

Since the compare value will be a 16-bit value (in between 0 and 65535), OCR1A and OCR1B are 16-bit registers. In ATMEGA16/32, there are two CTC channels – A and B. We can use any one of them or both. Let's use OCR1A.

```
OCR1A = 24999;   // timer compare value
```

TIFR Register

The **Timer/Counter Interrupt Flag Register** – TIFR is as follows. It is a common register to all the timers.



TIFR Register

We are interested in **Bit 4:3 – OCF1A:B – Timer/Counter1, Output Compare A/B Match Flag Bit**. This bit is set (one) by the AVR whenever a match occurs i.e. TCNT1 becomes equal to OCR1A (or OCR1B). It is cleared automatically whenever the corresponding Interrupt Service Routine (ISR) is executed. Alternatively, it can be cleared by writing '1' to it!

Code

Now that we are aware of the methodology and the registers, we can proceed to write the code for it. 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
3   // initialize timer, interrupt and variable
4   void timer1_init()
5   {
6       // set up timer with prescaler = 64 and CTC mode
7       TCCR1B |= (1 << WGM12)|(1 << CS11)|(1 << CS10);
8
9       // initialize counter
10      TCNT1 = 0;
11
12      // initialize compare value
13      OCR1A = 24999;
14  }
15
16  int main(void)
17  {
18      // connect led to pin PC0
19      DDRC |= (1 << 0);
20
21      // initialize timer
22      timer1_init();
23
24      // loop forever
25      while(1)
26      {
27          // check whether the flag bit is set
28          // if set, it means that there has been a compare
29          // and the timer has been cleared
30          // use this opportunity to toggle the led
31          if (TIFR & (1 << OCF1A)) // NOTE: '>=' used inste
32          {
33              PORTC ^= (1 << 0); // toggles the led
34          }
35
36          // wait! we are not done yet!
37          // clear the flag bit manually since there is no
38          // clear it by writing '1' to it (as per the data
39          TIFR |= (1 << OCF1A);
40
41          // yeah, now we are done!
42      }
43  }
```

So, now you have an LED flashing every 100 ms! **Now let's use another methodology to sort out the same problem statement.**

Methodology – Using Interrupts with CTC Mode

In the previous methodology, we simply used the CTC Mode of operation. We used to check every time for the flag bit (OCF1A). Now let's shift this responsibility to the AVR itself! Yes, now we do not need to check for the flag bit at all! The AVR will compare TCNT1 with OCR1A. Whenever a match occurs, it sets the flag bit OCF1A, and also fires an interrupt! We just need to attend to that interrupt, that's it! No other headache of comparing and stuffs!

There are three kinds of interrupts in AVR – overflow, compare and capture. We have already discussed the overflow interrupt in previous posts. For this case, we need to enable the compare interrupt. The following register is used to enable interrupts.

TIMSK Register

The **Timer/Counter Interrupt Mask Register**- TIMSK Register is as follows. It is a common register to all the timers. The greyed out bits correspond to other timers.

Bit	7	6	5	4	3	2	1	0	
	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0	TIMSK
Read/Write	R/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	

TIMSK Register

We have already come across TOIE1 bit. Now, the **Bit 4:3 – OCIE1A:B – Timer/Counter1, Output Compare A/B Match Interrupt Enable** bits are of our interest here. Enabling it ensures that an interrupt is fired whenever a match occurs. Since there are two CTC channels, we have two different bits OCIE1A and OCIE1B for them.

Thus, to summarize, whenever a match occurs (TCNT1 becomes equal to OCR1A = 24999), an interrupt is fired (as OCIE1A is set) and the OCF1A flag is set. Now since an interrupt is fired, we need an Interrupt Service Routine (ISR) to attend to the interrupt. Executing the ISR clears the OCF1A flag bit automatically and the timer value (TCNT1) is reset.

Interrupt Service Routine (ISR)

Now let's proceed to write an ISR for this. The ISR is defined as follows:

```
ISR (TIMER1_COMPA_vect)
{
    // toggle led here
    PORTB ^= (1 << 0);
}
```

Okay, that's all we need to do in the ISR. Now let's see how the actual code looks like.

Final Code

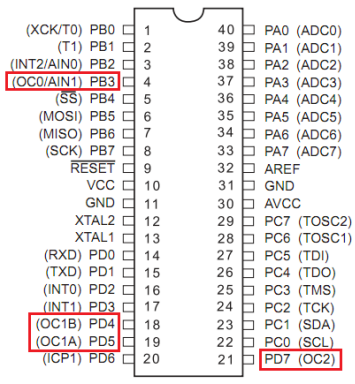
To learn about I/O port operations in AVR, view this. To know about bit manipulations, view this. To learn how to use AVR Studio 5, view this. To learn how this code is structured, view the previous TIMER0 post.

```
+ expand source
```

Hence, now we have seen how to implement the CTC mode using interrupts, reducing the code size, comparisons and processing time. But this is not over yet! We can reduce them further! **Let's take the same problem statement.**

Methodology – Using Hardware CTC Mode

Okay, so now, all of you have a look at the pin configuration of ATMEGA16/32. Can you see the pins PB3, PD4, PD5 and PD7? Their special functions are mentioned in the brackets (OC0, OC1A, OC1B and OC2). These are the Output Compare pins of TIMER0, TIMER1 and TIMER2 respectively. Before we learn how to use them, let's have another look at the TCCR1A



ATMEGA16/32 OC Pins

register.

TCCR1A Register

The **Timer/Counter1 Control Register A** – TCCR1A Register is as follows:

Bit	7	6	5	4	3	2	1	0	
	COM1A1	COM1A0	COM1B1	COM1B0	FOC1A	FOC1B	WGM11	WGM10	TCCR1A
Read/Write	R/W	R/W	R/W	R/W	W	W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

TCCR1A Register

Now time for us to concentrate on **Bit 7:6 – COM1A1:0** and **Bit 5:4 – COM1B1:0 – Compare Output Mode for Compare Unit A/B**. These bits control the behaviour of the Output Compare (OC) pins. The behaviour changes depending upon the following modes:

- Non-PWM mode (normal / CTC mode)
- Fast PWM mode
- Phase Correct / Phase & Frequency Correct PWM mode

Right now we are concerned only with the CTC mode. The following options hold good for non-PWM mode.

COM1A1/COM1B1	COM1A0/COM1B0	Description
0	0	Normal port operation, OC1A/OC1B disconnected.
0	1	Toggle OC1A/OC1B on compare match
1	0	Clear OC1A/OC1B on compare match (Set output to low level)
1	1	Set OC1A/OC1B on compare match (Set output to high level)

Compare Output Mode, non-PWM

Since we need to toggle the LED, we choose the second option (01). Well, that's all we need to do! No need to check any flag bit, no need to attend to any interrupts, nothing. Just set the timer to this mode and we are done! Whenever a compare match occurs, the OC1A pin is automatically toggled!

But we need to compromise on the hardware. Only PD5 or PD4 (OC1A or OC1B) can be controlled this way, which means that we should connect the LED to PD5 (since we are using channel A) instead of PC0 (which we had been using in all the examples till now).

Now let's see how the code looks like!

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.

```
+ expand source
```

Forcing Compare Match

Okay, so till now we have discussed almost all the concepts of CTC mode for TIMER1, except one – Forcing Compare Match. In the TCCR1A register,

till now we have ignored **Bit 3:2 – FOC1A:B – Force Output Compare for Compare Unit A/B**.

Bit	7	6	5	4	3	2	1	0	
	COM1A1	COM1A0	COM1B1	COM1B0	FOC1A	FOC1B	WGM11	WGM10	TCCR1A
Read/Write	R/W	R/W	R/W	R/W	W	W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

TCCR1A Register

If you see clearly, it's mentioned that these bits are write only bits. They are active only in non-PWM mode. Well, for ensuring compatibility with future devices, these bits must be set to zero (which they already are by default). Setting them to '1' will result in an immediate forced compare match and the effect will be reflected in the OC1A/OC1B pins. **The thing to be noted is that FOC1A/FOC1B will not generate any interrupt, nor will it clear the timer in CTC mode.**

CTC Mode – TIMER0

Till now we have explored CTC mode of TIMER1. In this section, I will describe the CTC mode of TIMER0 in brief. In other words, we will discuss about the registers only. We will not take a problem statement this time, because I assume that you have read whatever is written above and can use CTC mode of TIMER0 exactly in the same way! So let's get started.

TCCR0 Register

The **Timer/Counter0 Control Register**– TCCR10 Register is as follows:

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

- **Bit 6:3 – WGM01:0 – Wave Generation Mode** – Just like in TIMER1, we choose the type of wave mode from here as follows. Choose 10 for CTC mode.

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

Wave Generation Mode Bit Description

- **Bit 5:4 – COM01:0 – Compare Match Output Mode –** They control the behaviour of the OC0 (PB3) pin depending upon the WGM mode – non-PWM, Phase Correct PWM mode and Fast PWM mode. The selection options of non-PWM mode are as follows. Choose 01 to toggle the LED.

COM01	COM00	Description
0	0	Normal port operation, OC0 disconnected.
0	1	Toggle OC0 on compare match
1	0	Clear OC0 on compare match
1	1	Set OC0 on compare match

Compare Output Mode, non-PWM

- **Bit 7 – FOC0 – Force Output Compare –** This bit, when set to '1' forces an immediate compare match and affects the behaviour of OC0 pin. For ensuring compatibility with future devices, this bit must be set to '0'.

- **Bit 2:0 – CS02:0 – Clock Select Bits –** We are already familiar with these bits. View the TIMER0 post for more details.

OCR0 Register

The **Output Compare Register**– OCR0 Register is as follows:

Bit	7	6	5	4	3	2	1	0	
	OCR0[7:0]								OCR0
Read/Write	R/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	

OCR0 Register

The value to be compared (max 255) is stored in this register.

TIMSK Register

The **Timer/Counter Interrupt Mask**– TIMSK Register is as follows:

Bit	7	6	5	4	3	2	1	0	
	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0	TIMSK
Read/Write	R/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	

TIMSK Register

The **Bit 1 – OCIE0 – Timer/Counter0 Output Compare Match Interrupt Enable** enables the firing of interrupt whenever a compare match occurs.

TIFR Register

The **Timer/Counter Flag Register**– TIFR is as follows:

Bit	7	6	5	4	3	2	1	0	
	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0	TIFR
Read/Write	R/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	

TIFR Register

The **Bit 1 – OCF0 – Output Compare Flag 0** is set whenever a compare match occurs. It is cleared automatically whenever the corresponding ISR is executed. Alternatively it is cleared by writing '1' to it.

That's it! With this much information, I am sure that you can successfully generate a timer in CTC mode of TIMER0.

CTC Mode – TIMER2

Well, I leave this to you! It is exactly similar to the TIMER0 CTC Mode! Just go through the datasheet and you are through!!

Well, that's all for CTC Mode. I hope you enjoyed reading it! In the next post, we will discuss the different PWM modes and how to generate them.

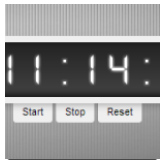
So till then, subscribe to my blog to stay updated! Alternatively, you can also grab the RSS Feeds for that!

Thank You!

Mayank Prasad
mayank.vitu@gmail.com

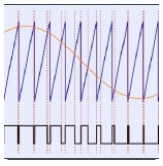
Related Posts

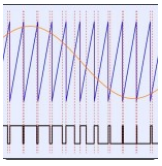
AVR Timers – TIMER0



AVR Timers – TIMER1

AVR Timers – TIMER2


AVR Timers – PWM Mode – Part I


AVR Timers – PWM Mode – Part II

Max

Max is the founder and admin of maxEmbedded. He describes himself as an 'embedded electronics freak' and an Apple/Linux fan. He likes to experiment, learn and share new things in this field. Furthermore, he likes to write stuffs on his website for techie newbies and loves to teach others. In his spare time, you will find him with volunteering somewhere!

More Posts - Website

Follow Me:

Loved it? Share it!

85 Comments


hemanth December 12, 2011

thanks a lot….each and every thing is given in clear manner

REPLY

Martin Matysiak February 4, 2012

Hey Mayank,

thank you very much for writing this and all the other articles in your blog! They just saved me a lot of time since I didn't have to search for the information I needed in the controller's datasheet and I highly appreciate that. Keep up the great work!

REPLY

Abdus Sami March 3, 2012

Great article, presented in a lovely manner.

REPLY

Mayank March 7, 2012

Thanks!

REPLY

serdar cetin March 14, 2012

Thank you very much about this artical.Atmel timers documents not clear.
Loves from Turkey
Serdar Cetin

REPLY

Jay March 21, 2012

Hi,
Thank for the tutorial. It was really helpful.
I just wanted to know if I can modify the value of OCR1A in ISR (using 16 bit Timer)?
Thank you
Jay

REPLY

Mayank March 22, 2012

Yes, why not?!

REPLY

Jay March 29, 2012

I am trying a output a pulse which is 130Hz in CTC mode. But the output is stuck at 122.3Hz and doesnt vary with changing the value of OCR1A.
Any suggestions?

(The code is very similar to the one with timer interrupt)
Thanks

Jay

REPLY

tepic May 15, 2012

Should:

TCCR1A |= 0; // not required since WGM11:0, both are zero (0)

Not read:

TCCR1A = 0;

?

REPLY

Mayank May 15, 2012

TCCR1A |= 0 implies that we are retaining the previous values if any. This is just to ensure that we don't accidently change the bits which we don't wish to change. However, if you want, you can also use TCCR1A = 0 during initialization. For more information about Bit Manipulations, view the Progamming 101 tutorial!

REPLY

tepic May 15, 2012

TCCR1A |= 0 changes no bits at all, retaining everything. I found I needed to clear WGM11 and WGM10. I guess a better suggestion should have been:

TCCR1A &= ~(1<<WGM11 | 1<<WGM10)

Having checked the other bits, for my purposes, TCCR1A=0 was a safe shortcut.

Great tutorial series though, I would not have go to this point without your guide. Thanks.

REPLY

Mayank May 15, 2012

Thanks, In that case go ahead!

REPLY

hemant June 23, 2012

Beauty..supreme understanding and effort on your side to make this stuff easy

REPLY

Mayank June 24, 2012

Thanks Hemant! Hope you enjoyed it

REPLY

Mwangi July 14, 2012

it’s a great article.simple language and everything covered

REPLY

mohammad ahmad July 22, 2012

iam very confused about putting one in the ocf1a ???
nw why we didn’t put TIFR |= (1 << OCF1A); inside the if scope instead of setting this bit out of the scope
i mean if the if condition not true at first we will put one and clear the bit and the second cycle i forced nw the condition to be true ???

if (TIFR & (1 <=’ used instead of ‘==’
{
PORTC ^= (1 << 0); // toggles the led
}
// wait! we are not done yet!
// clear the flag bit manually since there is no ISR to execute
// clear it by writing '1' to it (as per the datasheet)
TIFR |= (1 << OCF1A);

REPLY

Mohammad Ahmad August 3, 2012

what is the importance of existing two compare registers in timer 1 although the tcnt is cleard on comparing with only one of them ???

do you understand me ???

REPLY

Mayank August 4, 2012

TIMER1 has two physical channels, and hence there are two compare registers. Every time, TCNT1 is compared with either TCCR1A or TCCR1B. This can be used for fast switching between frequencies or maybe generate two different and unique frequencies at both the pins. I would suggest you to post this in avrfreaks.net You will get a better insight into this.

REPLY

Gopy August 3, 2012

well. good article. i have one suggestion from you. i have to measure time between pulse which i given to INT0. i am giving input pulse through INT0. i have to find time to measure accurate time between two pulse. i tried with over flow interrupt but its not correct. is there any way to do with CTC mode?

REPLY

Mayank August 4, 2012

I guess so! You can set some value of TCCR1A (say 20) and then divide the result by 20 to get the time between two pulses. Or else you can also set TCCR1A to 1. This would give you the exact time.

REPLY

shahi August 25, 2012

hello there I am in big trouble. I read your article but I am confused. I had a program and it's not working as it should. here is the code for atmega 32

everything is working except the seconds are counting after 4 seconds it counts 1 seconds. I was try to solve but I failed. Now I am asking for help. please help me. I have to submit this for my college project and my college teacher do not know how to use this program so I become hopeless. I know if you try you can help me. waiting for your reply.

```
// Code removed by the admin
```

REPLY

Mayank August 25, 2012

Hi Shahi
Such unexpected delays occur due to mismatch of the software frequency and hardware frequency.
1. What is the frequency your microcontroller is running at?
2. What is the frequency you have set in your software? View this to learn how to configure your software frequency.
3. If you are using an external crystal resonator, have you set the fuses of your microcontroller properly?

REPLY

rohits134 August 30, 2012

Reblogged this on itech and commented:
A nice tutorial to microcontrollers

REPLY

talal December 12, 2012

umm so im writing this code where im using external interrupt 1,the 16 bit counter as two b bit counters for pwm generation and timer 0 in ctc mode the problem is the interrupt routines seem to work fine as long as i donot use the pwm generation mode on timer 1 . as soon as i use timer 1 in pwm generation mode , the interrupt doesn't seem to work.
im using isis to simulate the code , any help you can offer ?

REPLY

Mayank December 14, 2012

Hi Talal,
I haven't used ISIS till now. Perhaps you should try posting your query to avrfreaks.net

REPLY

F_joel (@F_joel1) December 28, 2012

Hi Mayank. I'm trying to write a program that will count the pulse at the pin pb1 and display the frequency on a lcd. Can you help me with an example ? thanks

REPLY

Mayank December 28, 2012

Hi Joel,
You can do this easily using the Timers/Counters of AVR. Try reading my timer tutorials to get an idea about them.
What you can do is you can run a timer for one second, after which it fires an interrupt. In the interrupt service routine, read the value to count and clear it for the next cycle. This count will be equal to your frequency (since you are waiting only for one second), which you can display in your LCD.

REPLY

F_joel (@F_joel1) December 29, 2012

Hi Mayank. Thanks for the explanation, but could you write a simple program like that for me so that I can used it as example to write my own one. I've already tried many unsuccessfully. I just want to measure a frequency up to 20khz and display it through PortD on LCD.
Thanks.
this is what I've did ,but it isn't working

Code can be found here.

REPLY

Mayank December 29, 2012

Hi Joel,
The code you have written does not seem to be written in Atmel Studio 6, and so I may not be able to analyze it properly. But all the concepts have been very clearly described along with code examples in this post as well as my Timer1 post as well. Please take some time to read through them and try to implement the codes. The codes can also be found in my code gallery.

REPLY

kems2013 January 16, 2013

waho. your tutorials are the clearest i have ever seen

REPLY

Mayank January 19, 2013

Thanks!

REPLY

ismail January 22, 2013

effective tutorial..

REPLY

Mayank January 22, 2013

Thanks!

REPLY

Fuerb February 10, 2013

Hi Mayank,
thanks for putting the effort into writing this series of tutorials. Clearer than in your tutorial it is nowhere (for me).

REPLY

Mayank February 11, 2013

Thank you!

REPLY

Sigurdur February 11, 2013

Hi and thx for this awesome tutorial
I'm trying to blink the led on an Arduino uno 328P but no luck yet. I'm using Eclipse I might add.
I have to change the led to pin 9 (PB1) or 10 (PB2) to use timer1 right? I used DDRB = 0×02; to set the output to pin 9 which is the same as DDRB |= (1 << 1) right?
Another thing is that as soon as I add the ISR to the code I get a warning: warning: 'main' is normally a non-static function
Any ideas?

REPLY

Mayank February 12, 2013

Hi Sigurdur,
First of all may I ask you if you are using Arduino Uno, then why not program it using the Arduino software itself?

REPLY

Sigurdur February 12, 2013

Well That would be nice…I'm however taking a course which requires me to program the Arduino using this low level approach to get to know and understand better exactly how it works under the hood.

REPLY

Mayank February 12, 2013

Well, to be more precise, `DDRB = 0x02` is the same as `DDRB = (1<<1);` whereas `DDRB |= 0x02` is the same as `DDRB |= (1<<1);`
I hope you get me.
And as far as the warning is concerned, well, I don't have much idea about it. Upon searching the net, I found that people who got such warnings usually missed some brackets "}" somewhere in the code.

REPLY

Sigurdur February 12, 2013

Got it, I think
But does it matter which is used in this case that is?
It was a warning regarding the braces.

REPLY

Kunal March 7, 2013

really good , one of the best tuts tht i came acrossso far

REPLY

Mayank March 12, 2013

Thanks Kunal!

REPLY

Surendar Devasundaram March 21, 2013

Hey Mayank awesome tutorail and the way yo answer all the doubts is really cool

REPLY

Mayank March 26, 2013

Thank you Surendar!

REPLY

Luis March 27, 2013

Hi Mayank, I was surprised by your tutorials “max embedded” on AVR, are great, I would like you to help me in reviewing a code about timer0, my intention is to generate a 38 kHz frequency for an infrared LED and use the ATMEGA48 this is my code:
Code can be found here.
I hope you can help me thank you very much and success!

REPLY

Mayank April 3, 2013

Hi Luis,
There is only one register for Timer0, TCCR0 instead of TCCR0A and TCCR0B. The same holds good for OCR0A. There is no such register called so, only OCR0 exists. And you cannot assign a floating point value to OCR register. I would suggest you to go through my Timer0 tutorial. After that, try implementing it again, and get back here in case you want it reviewed again!

REPLY

Mayank April 3, 2013

However, these registers are valid in some AVR microcontrollers. Which microcontroller are you using?

REPLY

anurag March 30, 2013

awsm tutorial…
my all doubts are clear…
thnxx buddyy…

REPLY

Mayank April 3, 2013

Thanks Anurag!

REPLY

Hye Sang Ryu April 17, 2013

Hi!!
very good expaining..Thank you!
I'm second year of Electronics Engineering studying at Gangneung-Wonju National University ,Gangneung – in South Korea.!!
Nice too meet you!..

REPLY

Mayank April 27, 2013

Hi!
Greetings from India!
Pleasure to have you here!

REPLY

Nayan May 15, 2013

Hi Mayank sir, thank u….for ur great tutorial …I have learn a lots…

REPLY

timogruss May 21, 2013

Your timer tutorials are great… best I ever read on the web…
Thanks a lot!

REPLY

Mayank May 21, 2013

Thank you!

REPLY

Alex June 29, 2013

I have a problem……I read the above tutorial and wrote a code for producing a 38KHz signal to be used with a TSOP1738.Here is the code:
Code: http://pastebin.com/sDmPtz1H

What I should get is a constant (or flickering) output from the LED but what i'm getting are small pulses when ever i obstruct the IR led and reintroduce it back.
Is the timer part of the code correct??
Any solutions to the problem

REPLY

Mayank June 30, 2013

Hi Alex,

Your setup is working exactly as it should! From what I read here, TSOP1738 gives HIGH (+5V) when there is no IR source (i.e. you are not obstructing the IR LED), and it gives a LOW signal (0V) when there is a IR source centered at 38 kHz (i.e. you obstruct the

IR LED, thus the IR rays are emitted from the IR LED, bounces off your hand and reaches the TSOP sensor. I guess that's how you tried it).

Now coming to your code, the TIMER0 is configured perfectly. And in the code, you have connected your TSOP to `PA0`. Thus, by default when there is no IR source, then `PINA` should be `0x01`. Thus, `PINA & 0x01` becomes `0x01` and it goes to the `else` condition i.e. LED at `PB7` is turned OFF. Now when you obstruct the IR LED, the IR rays reach the TSOP by bouncing off your hand and `PINA` becomes `0x00`. Thus `PINA & 0x01` becomes `0x00` and the if condition is executed, where the LED at `PB7` is turned ON. This is exactly how you have described in your query above.

Is this what you were looking for?

REPLY

Alex July 2, 2013

yeah it happens exactly how you described above….
but when i'm giving a constant 38Khz signal (no obstacles in between) i should find the led on correct???….but in reality its off.
also using another mcu i generated a 38Khz signal using delay function (_delay_us(13)) and it worked perfectly fine (led on when tsop radiated, off otherwise).
so i though the timer of my first mcu is gone …..i flashed the timer code on the 2nd mcu and i'm still facing the same problem.

REPLY

Mayank July 2, 2013

You mean to say that your setup is reacting completely opposite – LED is ON with obstruction and OFF without obstruction?
Just to be clear, you are using using an IR LED to radiate IR at 38 kHz, which falls directly on a TSOP sensor, right?
Btw, are you sure that you are generating 38 kHz signal? Did you check it?

REPLY

George September 6, 2013

Hi again!!! In the first code example, in the while() loop, there is the following line: if (TIFR & (1 <=' used instead of '=='
I looked up for a '>=' but I did not find anything. Is this my mistake? Thank you.

REPLY

Yash September 7, 2013

Hello George
Its if(TIFR & (1**<<**OCF1A)).
This if statement is used to compare the status of the bit OCF1A bit with that in the TIFR Register (which is originally 0). If there is a match, then the if statement would return a '1', and the following conditional code would execute, else it would return a '0'.
In other words, in the above code, what we are doing is that we are setting the OCF1A bit to 1, and ANDing it with the TIFR register, which would subsequently compare the value we set just now to the OCF1A bit, with 0 (i.e. check for a status change in OCF1A).
I know this is a bit confusing to understand. It even to took me some time to figure this one out

REPLY

George September 6, 2013

if (TIFR & (1<=" used instead of "=="

REPLY

George September 6, 2013

I cannot understand. I copy the line but the blog does not accept it. I type the line with hands and still it does not appear correctly. Anyway, you may find the line in the first code example, in the main() and in the while(1) loop. Thanx!!!!

REPLY

George September 8, 2013

It will seem funny Mr Yash but this CTC page went clear quite immediately. But now, I will need your help definitely!!! I am a little bit confused with the following syntax.

a) What does this means? (1 << OCF1A);
b) What does this means? TIFR |= (1 << OCF1A); Why aren;t we just writing : TIFR |= 0×01; ?

Thank you in anticipation!!

REPLY

George September 8, 2013

DDRD |= (1 << 5); // ok I found it. Set the fifth bit of D Port equal to 1.

So, 1 << OCF1A means : we set OCF1A , or OCF1A = 0×01

REPLY

jishan December 12, 2013

sir i want used timer.if port A0 pin is high then LED will on & after 1min LED will automatically off.the input is IR sensor which is connected to A0 pin & LED is connected to D0 pin.please give me solution on that. THANKS…

REPLY

Mayank (Max) December 12, 2013

Hi Jishan,
Your problem is simple. Which microcontroller are you using? If the sensor is an analog one, you can use ADC. If it is a digital one, then simple GPIO operations would suffice. You can also read this on how to program digital IR sensors. Once you detect input, then simply start the timer and then wait for some event. The event

could be an interrupt or compare match or overflow or anything. Read my timer tutorials to know how to do that.

Best,
Max

REPLY

pablo December 12, 2013

Hi,

Could you please explain this line:

```
(TIFR1 & (1 << OCF1A))
```

I don't get it 😕

REPLY

Mayank (Max) December 12, 2013

Hi Pablo,
This is a condition which checks whether the `OCF1A` flag in the `TIFR` register is set(1) or not(0). And in case of AVR microcontrollers (most of them), there is only one `TIFR` register, and not `TIFR1`.

REPLY

pablo December 13, 2013

Thank you very much Mayank!

It's just strange to make if statements in that way (at least to me). My mind tells me it should be something like: register == 0b00000100 or register.bit == 1. But I guess your way is the only way, right?

Anyway, your blog is great! keep it up!

REPLY

Mayank (Max) December 13, 2013

Well, you could always use something like this: `if (TIFR & (1 << OCF1A) == true)`
The syntax of `if` statement checks for any `condition` inside the parenthesis `()`.

```
if (condition) {
// true
} else {
// false
}
```

Whenever you write any `condition` like, say, `count==20`, what the comparison results is in either `true` or `false`. This is because `if` can compare only `boolean` conditions.

So, whenever you write `TIFR & (1 << OCF1A)`, the result of this condition is either `0` (when OCF1A is not set) or some other value. The condition of `0` is considered as `false`, which means any other result will be taken as `true`. This means we do not explicitly need to use the `==` for comparison.

REPLY

pablo December 14, 2013

I know I know.

I program in a couple of languages, so I'm acquainted with the IF statment. It's just the way registers have to be mentioned that is getting a little work…

But thx for the explanation anyway

Paule December 13, 2013

Hello Mayank,

I found this page while trying to fix a nasty bug in my code. Worked on a

Atmega16A with two timers, both are polled for CTC flag (one inside an interrupt routine, one in main loop).

Bug had to do with access to TIFR register:
You should never ever do `TIFR |= (1 << OCF1A)`, because this will delete flags of other timers, if they are set (overflow / compare). Probably you do not intend this behaviour.

Better use `TIFR = (1 << OCF1A)` or, if you need to use interrupts, mask the appropriate bis.

Nevertheless, great article! So … thank you

Paule

REPLY

Mayank (Max) December 19, 2013

Hi Paule,
Sorry for the late reply. I didn't get the point you want to state here. As far as I know, we should always use OR to set the flag, since this will not modify other flags. The OR operation is used as bit mask in this case. See this for more information on bit manipulation.

Best,
Max

REPLY

hasham nisar February 19, 2014

MAYANK sir can you write a code in which timer 0 use as a CTC MoDE
?

REPLY

Mayank (Max) February 19, 2014

Nope, I can't. We don't write codes for people. We teach them concepts. I'm sure you are smart enough to write one yourself after reading this tutorial. Enjoy!

REPLY

hasham nisar February 20, 2014

ok Thanks sir i will try…

REPLY

Mayank (Max) February 21, 2014

Good luck with that. If you have trouble making sense out of any concept, feel free to post a comment. But please read the entire thing and try your best to understand it before putting it down here. Thanks.

REPLY

jyotendra sharma March 19, 2014

Hello Mayank sir,
Thanks for such a nice website. I have been following it for last 2 years (when I was in 2nd yr) to learn embedded and can't imagine starting any project without referring it once – its kind of notes copy for me.
You have added new page on embedded linux – can you bring on some more about porting linux to ARM processors. (Standalone processors – not some kind of kit.)

Please keep up the good work.

REPLY

Gonzalo April 13, 2014

Excelent guide! Thanks!! Actualy, the blog itself is great

REPLY

Max April 13, 2014

Thank you Gonzalo!

REPLY

VenkateshSai May 20, 2014

Aim : To dipaly ADC value every 1 sec and take appropriate action.
AVR : ATMEGA328p
Timer used : 16 bit only
Board : ARDUINO UNO

ISR shared var not incrementing,

REPLY

Max May 20, 2014

VenkateshSai,
It'd have been more helpful if you would have provided more information about your problem. How can you expect a response when you don't even tell properly what's wrong?!

REPLY

2tricky May 29, 2014

Thanks Max for your efforts with this tricky subject.

However may I pick your brains further – as I can't decide which way to go for my application…

My aim is to monitor a quadrature encoder at a point in its motion where it is quite slow still and insert extra pulses into the train of both encoder channels (A & B) whilst they are both low. The A+B+ 'edges' at the initial speed are about 1.25 ms apart. As this is my first direct port 'stuff' I've proven the idea (approximately) using the External Interrupts but whilst doing so, ran into a problem when milli() time stamping inside the ISR – far worse than would be expected from milli() not incrementing inside an ISR. So I wondered about using proper hardware time-stamps – but having crammed the subject so intensely, I can't see the wood for the

trees!

As I see it, an obvious approach is to time stamp from one edge to the next and when those stamps represent the correct pair, insert the new/shorter pulse after a calculated delay from the last channel to go low. Eventually the original pulse needs to 'carry on' after I've inserted about 100 extra pulses (guesstimate) and so my aim is to focus on adding one pulse for each in the original. May I ask, what approach would you gravitate to first?

Thanks a lot.

REPLY

Max June 22, 2014

Hi, sorry for the late response. Frankly speaking, your explanation didn't make much sense to me. I am afraid that I can't give you any suggestion on this. If you could describe it a little more and in simple sentences, I might. Sorry!

REPLY

Max April 1, 2014

Hey Jyotendra,
We're on it! It's next on our list! Till then, try to get your hands on one of them.

REPLY

Trackbacks/Pingbacks

AVR Timers – PWM Mode – Part I « maxEmbedded - [...] AVR Timers – CTC Mode [...]
AVR Timers – PWM Mode – Part II « maxEmbedded - [...] so I won't be writing any code here (just the pseudo code). I assume that after reading my previous …
Introduction to AVR Timers « maxEmbedded - [...] from normal operation, these three timers can be either operated in normal mode, CTC mode or PWM mode. We …
"Необычное" поведение режима CTC таймера1 | MyLinks - […] ISR(TIMER1_COMPA_vect) { digitalWrite(7,!digitalRead(7)); Cntr++; } Ссылки в помощь по теме: http://arduinodiy.wordpress.com/2012/02/28/timer-interrupts/ http://maxembedded.com/2011/07/14/avr-timers-ctc-mode/ […]

Post a Reply

Your email address will not be published. Required fields are marked *

Name *

Email *

Website

Comment

You may use these HTML tags and attributes: <a href="" title=""> <abbr title=""> <acronym title=""> <b> <blockquote cite=""> <cite> <code> <del datetime=""> <i> <q cite=""> <strike>

SUBMIT COMMENT

Notify me of follow-up comments by email.

Notify me of new posts by email.

Top Posts & Pages

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.