http://extremeelectronics.co.in/avr-tutorials/timers-in-compare-mode-part-i

AVR Timers – An Introduction

Posted By Avinash On September 18th, 2008 01:32 PM. Under [AVR Tutorials]

Timers are standard features of almost every microcontroller. So it is very important to learn their use. Since an AVR microcontroller has very powerful and multifunctional timers, the topic of timer is somewhat "vast". Moreover there are many different timers on chip. So this section on timers will be multipart. I will be giving basic introduction first.

What is a timer ?

A timer in simplest term is a register. Timers generally have a resolution of 8 or 16 Bits. So a 8 bit timer is 8Bits wide so capable of holding value withing 0-255. But this register has a magical property ! Its value increases/decreases automatically at a predefined rate (supplied by user). This is the timer clock. And this operation does not need CPU's intervention.

Fig.: Basic Operation Of a Timer.

Since Timer works independently of CPU it can be used to measure time accurately. Timer upon certain conditions take some action automatically or inform CPU. One of the basic condition is the situation when timer OVERFLOWS i.e. its counted upto its maximum value (255 for 8 BIT timers) and rolled back to 0. In this situation timer can issue an interrupt and you must write an Interrupt Service Routine (ISR) to handle the event.

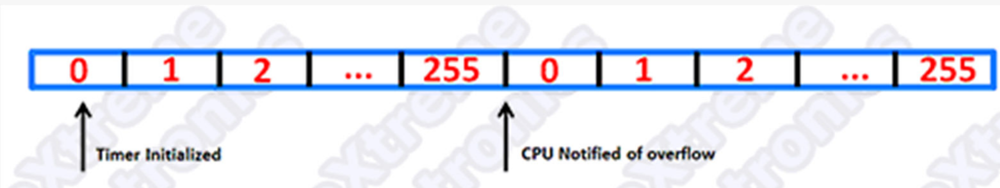


Fig.: Basic Operation Of a Timer.

Using The 8 BIT Timer (TIMER0)

The ATmega16 and ATmega32 has three different timers of which the simplest is TIMER0. Its resolution is 8 BIT i.e. it can count from 0 to 255.

Note:

Please read the "Internal Peripherals of AVRs" to have the basic knowledge of techniques used for using the OnChip peripherals(Like timer !)

The Prescaler

The Prescaler is a mechanism for generating clock for timer by the CPU clock. As you know that CPU has a clock source such as a external crystal of internal oscillator. Normally these have the frequency like 1 MHz,8 MHz, 12 MHz or 16MHz(MAX). The Prescaler is used to divide this clock frequency and produce a clock for TIMER. The Prescaler can be used to get the following clock for timer.

No Clock (Timer Stop).

No Prescaling (Clock = FCPU)

FCPU/8

FCPU/64

FCPU/256

FCPU/1024

Timer can also be externally clocked but I am leaving it for now for simplicity.

TIMER0 Registers.

As you may be knowing from the article "Internal Peripherals of AVRs" every peripheral is connected with CPU from a set of registers used to communicate with it. The registers of TIMERs are given below.

TCCR0 – Timer Counter Control Register. This will be used to configure the timer.

Bit	7	6	5	4	3	2	1	0
Name	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00
Initial Value	0	0	0	0	0	0	0	0

Fig.: TCCR0 – Timer Counter Control Register 0

As you can see there are 8 Bits in this register each used for certain purpose. For this tutorial I will only focus on the last three bits CS02 CS01 CS00 They are the CLOCK SELECT bits. They are used to set up the Prescaler for timer.

CS02	CS01	CS00	Description
0	0	0	Timer stoped
0	0	1	FCPU
0	1	0	FCPU/8
0	1	1	FCPU/64
1	0	0	FCPU/256
1	0	1	FCPU/1024
1	1	0	External Clock Source on PIN T0.Clock on falling edge
1	1	1	External Clock Source on PIN T0.Clock on rising edge

TCNT0 – Timer Counter 0

Bit	7	6	5	4	3	2	1	0
Name	TCNT0							
Initial Value	0	0	0	0	0	0	0	0

Timer Interrup Mask Register TIMSK

Bit	7	6	5	4	3	2	1	0
Name							OCIE0	TOIE0
Initial Value	0	0	0	0	0	0	0	0

This register is used to activate/deactivate interrupts related with timers.
This register controls the interrupts of all the three timers. The last two bits (BIT 1 and BIT 0) Controls the interrupts of TIMER0. TIMER0 has two interrupts but in this article I will tell you only about one(second one for next tutorial).
TOIE0 : This bit when set to “1” enables the OVERFLOW interrupt. Now time for some practical codes !!! We will set up timer to at a Prescaler of 1024 and our FCPU is 16MHz. We will increment a variable “count” at every interrupt(OVERFLOW) if count reaches 61 we will toggle PORTC0 which is connected to LED and reset “count= 0”. Clock input of TIMER0 = 16MHz/1024 = 15625 Hz Frequency of Overflow = 15625 /256 = 61.0352 Hz if we increment a variable “count” every Overflow when “count reach 61” approx one second has elapse.

Setting Up the TIMER0

```
  // Prescaler = FCPU/1024
   TCCR0|=(1<<CS02)|(1<<CS00);
 //Enable Overflow Interrupt Enable
   TIMSK|=(1<<TOIE0);
 //Initialize Counter
   TCNT0=0;
```

Now the timer is set and firing Overflow interrupts at 61.0352 Hz

The ISR

```
ISR(TIMER0_OVF_vect)
{
//This is the interrupt service routine for TIMER0 OVERFLOW Interrupt.
//CPU automatically call this when TIMER0 overflows.

//Increment our variable
```

```
    count++;
    if(count==61)
    {
    PORTC=~PORTC; //Invert the Value of PORTC
    count=0;
    }
}
```

Demo Program (AVR GCC)

Blink LED @ 0.5 Hz on PORTC[3,2,1,0]

```
#include <avr/io.h>
#include <avr/interrupt.h>

volatile uint8_t count;

void main()
{
   // Prescaler = FCPU/1024
   TCCR0|=(1<<CS02)|(1<<CS00);

   //Enable Overflow Interrupt Enable
   TIMSK|=(1<<TOIE0);

   //Initialize Counter
   TCNT0=0;

   //Initialize our varriable
   count=0;

   //Port C[3,2,1,0] as out put
   DDRC|=0x0F;

   //Enable Global Interrupts
   sei();

   //Infinite loop
   while(1);
}

ISR(TIMER0_OVF_vect)
```

```
{
   //This is the interrupt service routine for TIMER0 OVERFLOW Interrupt.
   //CPU automatically call this when TIMER0 overflows.

   //Increment our variable
   count++;
   if(count==61)
   {
      PORTC=~PORTC; //Invert the Value of PORTC
      count=0;
   }
}
```

Hardware

ATmega16 or ATmega32 running @ 16MHz. Connet LEDs using 330ohms resistors on PORTC[3,2,1,0]. If you are using xBoard you can connect four onboard LEDs to PORTC using four PIN Connectors.
Thats it for now meet in next tutorial. And please don't forget to post a comment, I am waiting for them !

Timers in Compare Mode – Part I

Posted By Avinash On October 24th, 2008 06:07 PM. Under AVR Tutorials

Hi Friends,

In last tutorials we discussed about the **basics of TIMERs of AVR**. In this tutorial we will go a step further and use the timer in ***compare mode*** .
In our first tutorial on timer we set the clock of the timer using a prescaler and then let the timer run and whenever it overflowed it informed us. This way we computed time. But this has its limitations we cannot compute time very accurately. To make it more accurate we can use the compare mode of the timer. In compare mode we load a register called Output Compare Register with a value of our choice and the timer will compare the current value of timer with that of Output Compare Register continuously and when they match the following things can be configured to happen.

1. A related Output Compare Pin can be made to go high,low or toggle automatically. This mode is ideal for generating square waves of different frequency.
2. It can be used to generate PWM signals used to implement a DAC digital to analog converter which can be used to control the speed of DC motors.
3. Simply generate an interrupt and call our handler.

On a compare match we can configure the timer to reset it self to 0. This is called CTC – Clear Timer on Compare match.

The compare feature is not present in the 8 bit TIMER0 of the ATmega8 so we will use the TIMER1 which is a 16 Bit timer. First we need to setup the timer's prescaler as described in the Timer0 tutorial.
Please
see this tutorial for a basic introduction of TIMERs.

The TIMER1 has two compare units so it has two output compare register OC1A and OC1B. The '1' in the name signifies that they are for timer '1'.

In this tutorial we will create a standard time base which will be useful for many projects requiring timing such as clocks,timers,stopwatches etc. For this we will configure the timer to generate an Compare match every millisecond and in the ISR we will increment a variable clock_millisecond. In this way we will have a accurate time base which we can use for computing time in seconds,minutes and hours.

References

For
learning about the basic idea of peripherals and their use with AVRs please see this tutorial.

For
learning about basics of timers see this.

AVR's Timers1 Registers

I will state the meaning of only those bits which are required for this tutorial. These bits are marked with a gray back ground in the table. For details about other bits please consult the datasheets.

Timer/Counter1 Control Register A (TCCR1A)

This register is used to configure the TIMER1. It has the following bits

Bit	7	6	5	4	3	2	1	0
Name	COM1A1	COM1A0	COM1B1	COM1B0	FOC1A	FOC1B	WGM11	WGM10
InitialValue	0	0	0	0	0	0	0	0

COM1A1 and COM1A0 -

This are used to configure the action for the event when the timer has detected a "match". As i told earlier the timer can be used to automatically set,clear or toggle the associated Output compare pin this feature can be configured from here. The table below shows the possible combinations.

COM1A1	COM1A0	Description
0	0	Normal Port Operation (The timer doesn't touches the PORT pins).
0	1	Toggle OC1A Pin on match
1	0	Clear OC1A on match – set level to low (GND)
1	1	Set OC1A on match – set level to High(Vcc)

The OC1A pin is the Pin15 on ATmega8 and Pin19 on ATmega16/32. As you may guess that we don't need any pin toggling or any thing for this project so we go for the first option i.e. **Normal Port Operation**

As I have told you that the TIMER1 has two compare unit, the COM1B1/COM1B0 are used exactly in same way but for the channel B.

WGM11 and WGM10 -

These combined with WGM12 and WGM13 found in TCCR1B are used for selecting proper mode of operation. WGM= Waveform Generation Mode.

Timer/Counter1
Control Register B (TCCR1B)

This register is also used for configuration. The Bits are.

Bit	7	6	5	4	3	2	1	0
Name	ICNC1	ICES1	-	WGM13	WGM12	CS12	CS11	CS10
InitialValue	0	0	0	0	0	0	0	0

The four bits

WGM13 – WGM12 – WGM11 – WGM10 are used to select the proper mode of operation.
Please refer to the datasheet for complete combinations that can be used. We
need the CTC mode i.e. clear timer on match so we set them as follows

WGM13=0

WGM12=1

WGM11=0

WGM10=0

This is the settings for CTC.

The CS12,CS11,CS10

These are used for selecting the prescalar value for generating clock for the
timer. I have already discussed them on TIMER0 tutorial.
I will select the prescalar division factor as 64. As the crystal we are using
is of 16MHz so dividing this by 64 we get the timer clock as

F(timer)=16000000/64 = 250000Hz

so timer will increment its value @ 250000Hz

The setting for this is

CS12	CS11	CS10
0	1	1

So the final code we write is

```
TCCR1B=(1<<WGM12)|(1<<CS11)|(1<<CS10);
```

TIMER
Counter 1 (TCNT1)

TCNT1H (high byte) TCNT1L(low byte). This is the 16 Bit counter

Output
Compare Register 1 A – OCR1A (OCR1AH,OCR1AL)

You load them with required value. As we need a time base of 1ms and our counter is running @ 250000Hz i.e. one increment take 1/250000 = 0.000004 Sec or 0.004 ms. So we need 1ms/0.004 = 250 increments for 1ms. Therefore we set OC1A=250. In this way when timer value is 250 we will get an interrupt and the frequency of occurrence is 1ms and we will use this for incrementing a variable clock_millisecond.

Output Compare Register 1 B – OCR1A (OCR1BH,OCR1BL)

Timer Counter Interrupt Mask (TIMSK)

This is the mask register used to selectively enable/disable interrupts. This register is related with the interrupts of timers (all timers TIMER0,TIMER1,TIMER2).

Bit	7	6	5	4	3	2	1	0
Name	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	-	TOIE0
InitialValue	0	0	0	0	0	0	0	0

Of all these bits 5,4,3,2 are for TIMER1 and we are interested in the OCIE1A which is Output Compare Interrupt Enable 1 A. To enable this interrupt we write

```
TIMSK|=(1<<OCIE1A);
```

After enabling the interrupt we also need to enable interrupt globally by using the function

```
sei();
```

This function is part of AVR-GCC interrupt system and enables the interrupt globally. Actually this translate in one machine code so there is no function call overhead.

So friends that's its for now ! The rest will be covered in latter tutorials. To get all the latest tutorials on your mailbox subscribe to my RSS feed via e-mail.

And don't forget to post your comment !!! What you think about them and what you will like to see here. Or simply post any doubt you have about this tutorial.

Goodbye, and have fun !

Timers in Compare Mode – Part II

Posted By Avinash On October 25th, 2008 12:27 PM. Under AVR Tutorials

Hello and welcome back to the discussion on the TIMERs in compare mode. In the last article we discussed the basics and the theory about using the timer in compare mode. Now its time to write some practical code and run it in real world. The project we are making is a simple time base which is very useful for other project requiring accurate computation of time like a digital clock or a timer that automatically switches devices at time set by user. You can use it for any project after understanding the basics.

We will have three global variable which will hold the millisecond, second and minutes of time elapsed. These variables are automatically updated by the compare match ISR. Look at the figure below to get an idea how this is implemented.

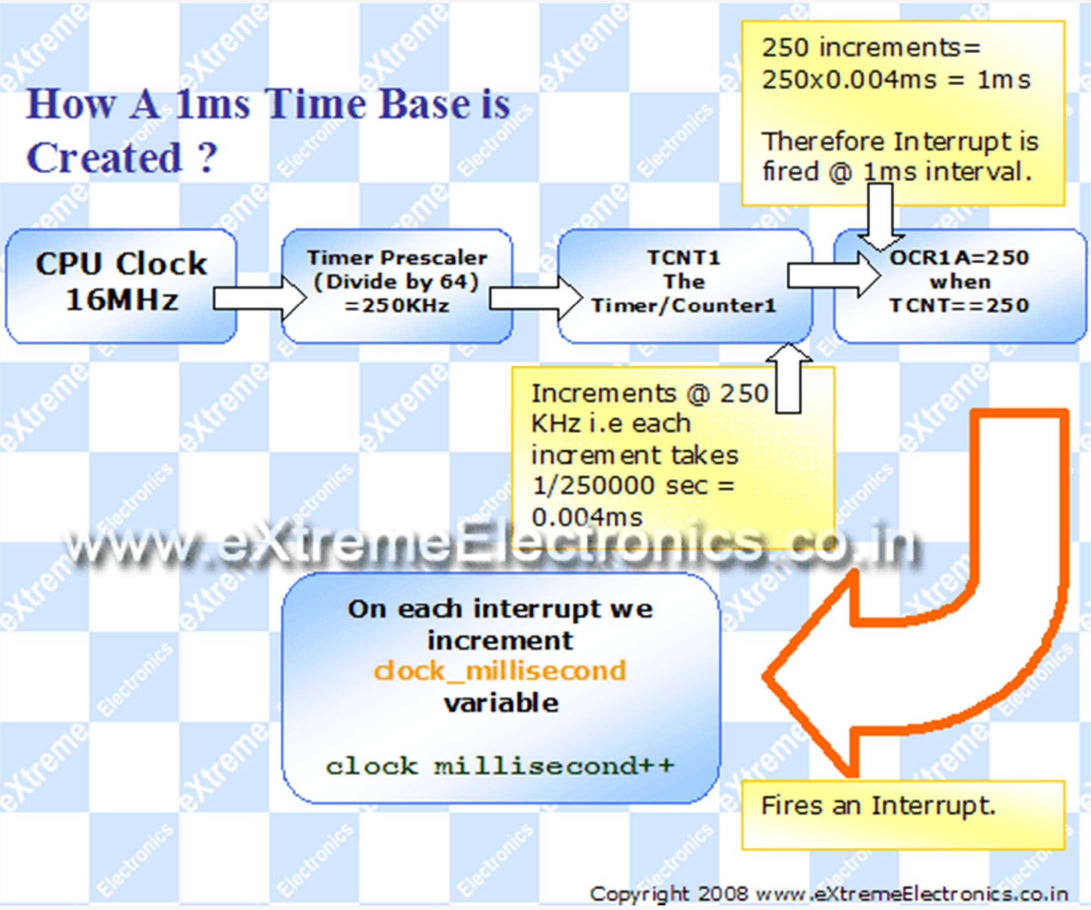


Fig – Using AVR Timer to generate 1ms Time base.

Complete Code

```
#include <avr/io.h>
#include <avr/interrupt.h>

#include "lcd.h"

//Global variable for the clock system
volatile unsigned int   clock_millisecond=0;
volatile unsigned char  clock_second=0;

volatile unsigned char  clock_minute=0;

main()
{
   //Initialize the LCD Subsystem
```

```
   InitLCD(LS_BLINK);
   //Clear the display
   LCDClear();

   //Set up the timer1 as described in the
   //tutorial

   TCCR1B=(1<<WGM12)|(1<<CS11)|(1<<CS10);
   OCR1A=250;

   //Enable the Output Compare A interrupt
   TIMSK|=(1<<OCIE1A);


   LCDWriteStringXY(0,0,"Time Base Demo");
   LCDWriteStringXY(0,1,"  :    (MM:SS)");

   //Enable interrupts globally

   sei();

   //Continuasly display the time
   while(1)
   {
      LCDWriteIntXY(0,1,clock_minute,2);
      LCDWriteIntXY(3,1,clock_second,2);
      _delay_loop_2(0);
   }

}


//The output compate interrupt handler
//We set up the timer in such a way that
//this ISR is called exactly at 1ms interval
ISR(TIMER1_COMPA_vect)
{
   clock_millisecond++;
   if(clock_millisecond==1000)
   {
      clock_second++;
      clock_millisecond=0;
```

```
        if(clock_second==60)
        {
            clock_minute++;
            clock_second=0;
        }
    }
}
```

Hardware

The hardware is ATmega8-16PU running at 16MHz with a LCD Connected to it.
I have used
xBoard MINI to make the prototype. You can also use your own ATmega8
board. To make a board in your own see this.
The output is displayed in a 16×2 character LCD module so please see the
LCD
interfacing tutorial for information about the connections and use.
I recommend you to first setup and test the LCD interfacing because it will
help you in many projects. If you have any problems setting it up please
post a comment here or use the forum.eXtremeElectronics.co.in

Fig – xBoard MINI can be used to prototype many projects easily!

Fig – The output of above program in 16×2 LCD module.

Goodbye for now. Meet you in next tutorials !!! And don't forget to subscribe to my feed via email to receive latest tutorials **direct in your mail box.**

PWM Signal Generation by Using AVR Timers.

Posted By Avinash On January 14th, 2009 10:33 AM. Under AVR Tutorials

In the last tutorial you saw how the PWM technique helps us generate analog signals from a microcontroller. In this tutorial we will see how PWM generation is implemented with microcontrollers.

Before you begin please see

- Introduction to PWM
- Introduction to AVR Timers

Generation of PWM signals is such a common need that all modern microcontrollers like AVR has dedicated hardware for that. The dedicated hardware eliminates the load of generation of PWM signal from software (thus frees the CPU). Its like asking the hardware to generate a PWM signal of a specific duty cycle and the task of CPU is over. The PWM hardware with start delivering the required signal from one of its PINs while the CPU can continue with other tasks.

In AVR microcontrolers PWM signals are generated by the TIMER units. (See AVR Timer Tutorials) . In this tutorial I will give you the basic idea of how PWM signals are generated by AVR timers. Their are two methods by which you can generate PWM from AVR TIMER0 (for ATmega16 and ATmega32 MCUs).

1. Fast PWM
2. Phase Correct PWM

Don't worry from their names they will become clear to you as we go on. First we will be considering the Fast PWM mode.

PWM Generation Fundas

We will use the simplest timer, TIMER0 for PWM generation.(Note TIMER0 of ATmega8 cannot be used for PWM generation, these are valid for ATmega16 and ATmega32).
In this part we won't be dealing with any code, we would just analyze the concepts.
So lets start!

We have a 8bit counter counting from 0-255 and the goes to 0 and so on. This can be shown on graph as

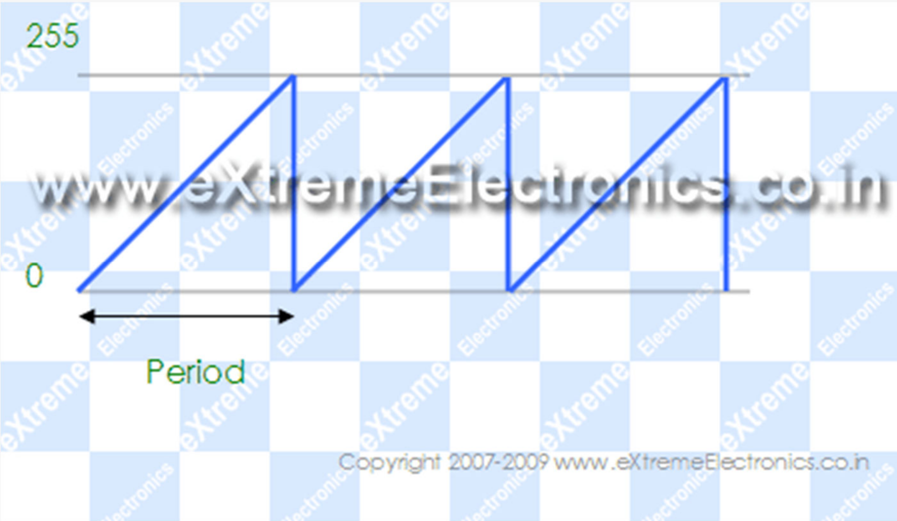


Fig. 1 – AVR Timer Count Sequence for Fast PWM.

The period depends upon the prescalar
settings. Now for PWM generation from this count sequence we have a new "friend" named OCR0 (Output Compare Register Zero , zero because its for TIMER0 and there are more of these for TIMER1 & TIMER2). We can store any value between 0-255 in OCR0, say we store 64 in OCR0 then it would appear in the graph as follows (the RED line).

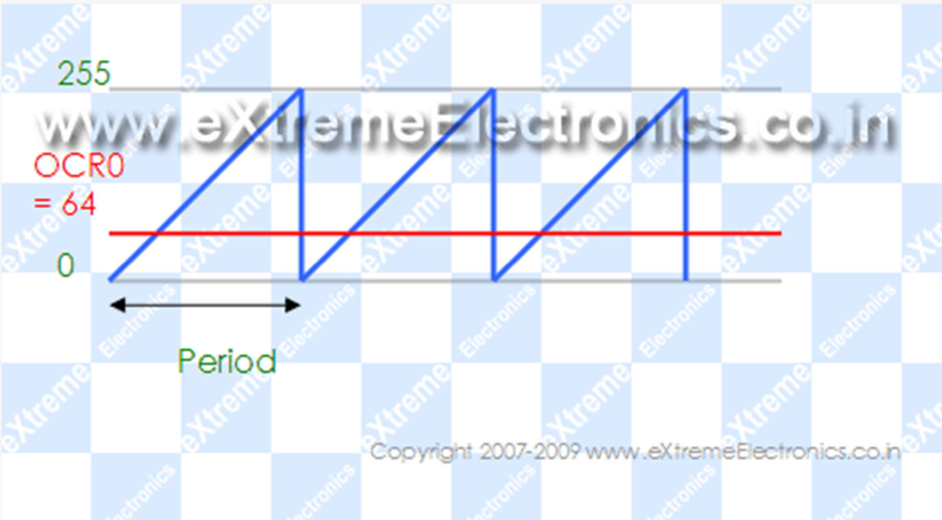


Fig. 2 – AVR Timer Count Sequence for Fast PWM with OCR0=64

So how does this Output Compare Register generates PWM? Well, the answer follows.

When the TIMER0 is configured for fast PWM mode,while up counting whenever the value of TIMER0 counter (TCNT0 register) matches OCR0 register an output PIN is pulled low (0) and when counting sequence begin again from 0 it is SET again (pulled high=VCC). This is shown in the figure 3. This PIN is named OC0 and you can find it in the PIN configuration of ATmega32.

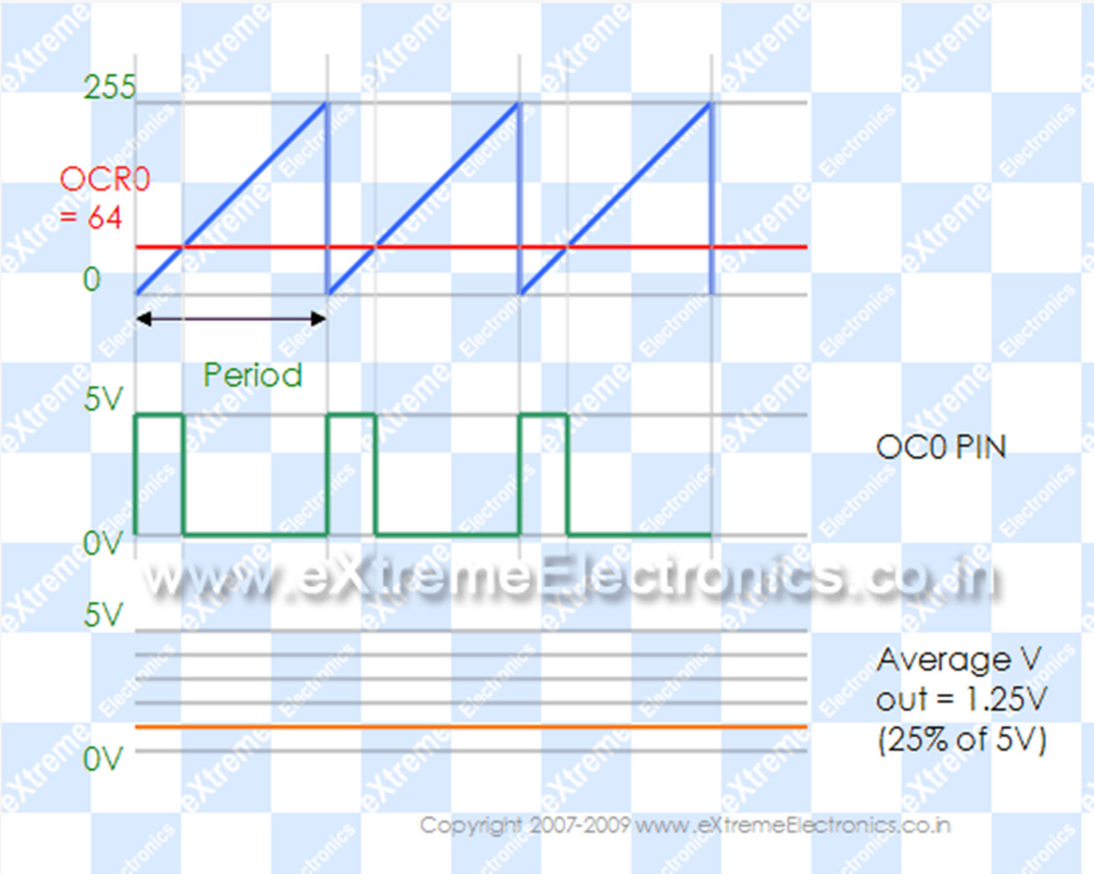


Fig. 3- AVR Timer Count Sequence for Fast PWM with OCR0=64

From the figure you can see that a wave of duty cycle of 64/256 = 25% is produced by setting OCR0=64. You can set OCR0 to any value and get a PWM of duty cycle of (OCR0 / 256). When you set it to 0 you get 0% dutycycle while setting it to 255 you get 100% duty cycle output. Thus by varying duty cycle you can get an analog voltage output from the OC0 PIN. The resolution of this PWM is 8BIT. Watch the animation below for a step by step explanation of PWM generation process.

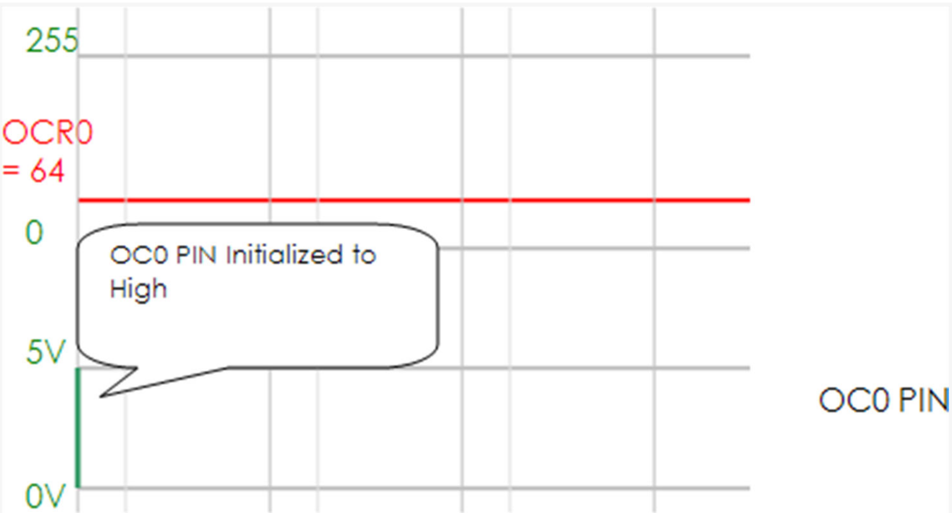


Fig. 4 – PWM Generation Process from AVR Timers.

One note about OCR0 is that it is double buffered. But what does than means?

It is just for your help. Double buffering means that you cannot directly write to OCR0 when ever you write to OCR0 you are actually writing to a buffer. The value of buffer is copied to actual OCR0 only during start of cycle (when TCNT0 wraps from 255 to 0). This nice feature prevents update of OCR0 in between the cycles. The new value of OCR0 comes into effect only on beginning of a new cycle even if you write to it in between a cycle.

In next tutorial we will see how to setup the TIMER0 in fast PWM mode, actually generate some PWM signals and use this to control the brightness of a LED.

Servo Motor Control by Using AVR ATmega32 Microcontroller

Posted By Avinash On June 7th, 2010 06:28 PM. Under AVR Tutorials

Servo motors are a type of electromechanical actuators that do not rotate continuously like DC/AC or stepper motors, rather they used to position and hold some object. They are used where continuous rotation is not required so they are not used to drive wheels (unless a servo is modified). In contrast they are used where something is needed to move to particular position and then stopped and hold there. Most common use is to position the **rudder** of aircrafts and boats etc. Servos can be used effectively here because the rudder do not need to move full 360 degrees nor they require continuous rotation like a wheel. The servo can be commanded to rotate to a particular angle (say 30) and then hold the rudder there. Servos also employs a feedback mechanism, so it can sense an error in its positioning and correct it. This is called**servomechanism**. So if the air flow exerts pressure on rudder and deflects it the servo will apply force in opposite direction and try to correct the error. Say if you ask servo to go and lock itself to 30 degrees and then try to rotate it with your hand, the servo will try hard and its best to overcome the force and keep servo locked in its specified angle.

Servos are also used to control the steering of RC cars, robotics arms and legs.

Their are many types of servos but here we will concentrate on small hobby servos. Hobby servo has motor and its control mechanism built into one unit. They have 3 wire connector. One is for positive supply other for ground and the last one for control signal. The image below shows a common hobby servo from **Futaba**, its **S3003**.

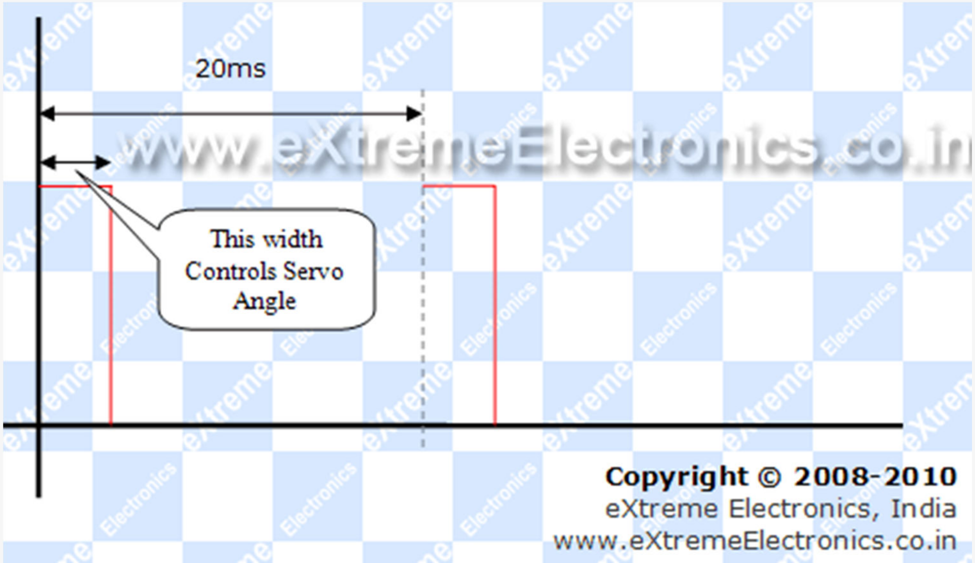
Futaba S3003

Futaba S3003 wiring.

1. RED -> Positive supply 4.8v to 6v
2. BLACK -> GND
3. WHITE -> Control Signal.

Controlling a Servo Motor.

Controlling a servo is easy by using a microcontroller, no external driver like h-bridge etc are required. Just a control signal is needed to be feed to the servo to position it in any specified angle. The frequency of the control signal is **50hz** (i.e. the period is 20ms) and the width of positive pulse controls the angle.



Servo Motor Control Signal.

For **Futaba 3003 servos,** I found out the following timings. The relation between the width of pulse and servo angle is given below. Note that these servos are only capable of rotating between 0 and 180 degrees.

- 0.388ms = 0 degree.
- 1.264ms = 90 degrees. (neutral position)
- 2.14ms = 180 degrees.

Controlling Servo Motors with AVR Microcontrollers.

You can use the AVR micro controllers PWM feature to control servo motors. In this way the PWM with automatically generate signals to lock servo and the CPU is free to do other tasks. To understand how you can setup and use PWM you need to have basic understanding of hardware timers and PWM modules in AVR. The following articles may be of great help.

- Introduction to PWM
- Generating PWM Signals by using AVR Timers.
- Introduction to AVR Timers.
- Timers in compare Mode.

Here we will use AVR Timer1 Module which is a 16bit timer and has two PWM channels(A and B). The CPU frequency is 16MHz, this frequency is the maximum frequency that most AVRs are capable of running. And so it is used in most development board like **Low Cost AVR Development Board** and **xBoards**.
We chose the prescaler as 64. So the timer will get 16MHz/64 = 250khz (4 uS period). We setup Timer Mode as Mode 14.

Timer Mode 14 features

- FAST PWM Mode
- TOP Value = ICR1

So the timer will count from 0 to ICR1(TOP Value). The formula for PWM frequency and calculation for TOP value is given below

$$f_{pwm} = \frac{f_{cpu}}{N(1+TOP)}$$

$$f_{pwm} = \frac{16000000Hz}{64(1+TOP)}$$

$$50\ Hz = \frac{16000000Hz}{64(1+TOP)}$$

$$TOP = 4999$$

So we set up ICR1A=4999, this gives us PWM period of 20ms (50 Hz). Compare Output Mode is set by correctly configuring bits COM1A1,COM1A0 (For PWM Channel A) and COM1B1,COM1B0(For PWM Channel B)

COM1A1 = 1 and COM1A0=0 (for PWM Channel A)

COM1B1 = 1 and COM1B0=0 (for PWM Channel B)

The above settings clears the OC1A (or OC1B) pin on Compare Match and SET (to high) at BOTTOM. The OC1A and OC1B pins are the PWM out pin in ATmega16/ATmega32 chips. This settings gives us NON inverted PWM output.

Now the duty cycle can be set by setting OCR1A and OCR1B registers. These two register controls the PWM high period. Since the period of timer is 4uS (remember 16Mhz divided by 64?) we can calculate values required for following servo angles.

- Servo Angle 0 degrees require pulse width of 0.388ms(388uS) so value of OCR1A = 388us/4us = 97
- Servo Angle 90 degrees require pulse width of 1.264ms(1264uS) so value of OCR1A = 1264us/4us = 316
- Servo Angle 180 degrees require pulse width of 2.140ms(2140uS) so value of OCR1A = 2140us/4us = 535

If you are using **Vigor VS-10 (High Torque Servo)** then you need the following values.

- Set OCR1A=180 for 0 degree.
- Set OCR1A=415 for 90 degree.
- Set OCR1A=650 for 180 degree.

In this way you can calculate the value of OCR1A(or OCR1B for second servo) for any angle required. We can see that the value of OCR1x varies from 97 to 535 for servo angle of 0 to 180 degrees.(180 to 650 in case of **Vigor VS-10**)

Complete AVR ATmega32 Code for Servo Motor Control Demo.

The demo program given below shows how to use servo motors with AVR microcontroller.
The job of the program is very simple, it starts by initializing the timer and
pwm. Then it locks the servo at 0 degree, then moves it to 90 degree and wait
for some time, then similarly goes to 135 degree and finally to 180 degrees.
The process is repeated as long as powered.

Some Parameter for proper working of program.

- LOW Fuse = 0xFF and HIGH Fuse = 0xC9
- Crystal Frequency = 16MHz.
- Servo Motor is branded **Futaba S3003**.
- MCU is ATmega32 or ATmega16.

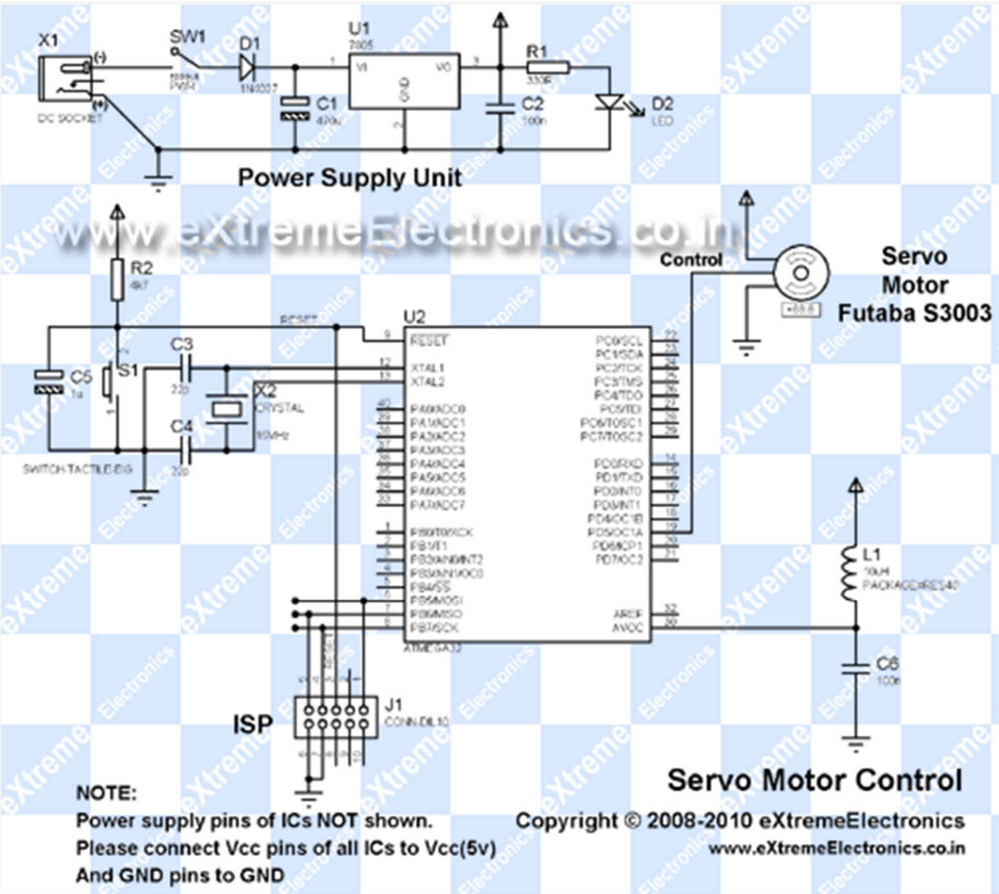
```
/******************************************************************************


Program to demonstrate the use servo motors with AVR Microcontrollers.


For More Details Visit: http://www.eXtremeElectronics.co.in


Copyright (c) 2008-2010
eXtreme Electronics, India


Servo Motor: Futaba s3003
Servo Control PIN (white): To OC1A PIN
Crystal: 16MHz
LOW Fuse: 0xFF
HIGH Fuse: 0xC9


Compiler:avr-gcc toolchain
Project Manager/IDE: AVR Studio


                                     NOTICE
                              --------
NO PART OF THIS WORK CAN BE COPIED, DISTRIBUTED OR PUBLISHED WITHOUT A
WRITTEN PERMISSION FROM EXTREME ELECTRONICS INDIA. THE LIBRARY, NOR ANY PART
OF IT CAN BE USED IN COMMERCIAL APPLICATIONS. IT IS INTENDED TO BE USED FOR


HOBBY, LEARNING AND EDUCATIONAL PURPOSE ONLY. IF YOU WANT TO USE THEM IN
COMMERCIAL APPLICATION PLEASE WRITE TO THE AUTHOR.



WRITTEN BY:
AVINASH GUPTA
me@avinashgupta.com
```

```
********************************************************************************/
#include <avr/io.h>

#include <util/delay.h>

//Simple Wait Function
void Wait()
{
   uint8_t i;
   for(i=0;i<50;i++)
   {
      _delay_loop_2(0);
      _delay_loop_2(0);
      _delay_loop_2(0);
   }

}

void main()
{
   //Configure TIMER1
   TCCR1A|=(1<<COM1A1)|(1<<COM1B1)|(1<<WGM11);        //NON Inverted PWM
   TCCR1B|=(1<<WGM13)|(1<<WGM12)|(1<<CS11)|(1<<CS10); //PRESCALER=64 MODE 14(FAST
PWM)

   ICR1=4999;  //fPWM=50Hz (Period = 20ms Standard).

   DDRD|=(1<<PD4)|(1<<PD5);   //PWM Pins as Out

   while(1)
   {

      OCR1A=97;   //0 degree
      Wait();

      OCR1A=316;  //90 degree
      Wait();

      OCR1A=425;  //135 degree

      Wait();

      OCR1A=535;  //180 degree
```

```
        Wait();
    }
}
```

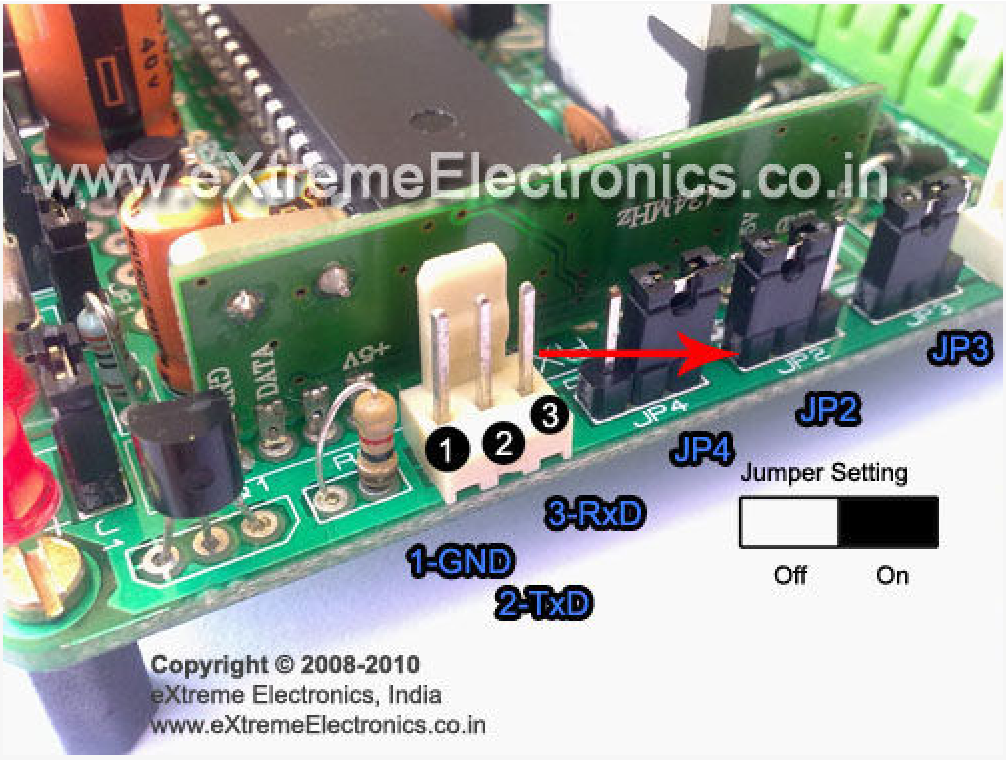
Hardware

You need a basic avr circuit with 5v regulated supply, 16 MHz crystal and proper reset circuit. All these are present in most **common development boards**.

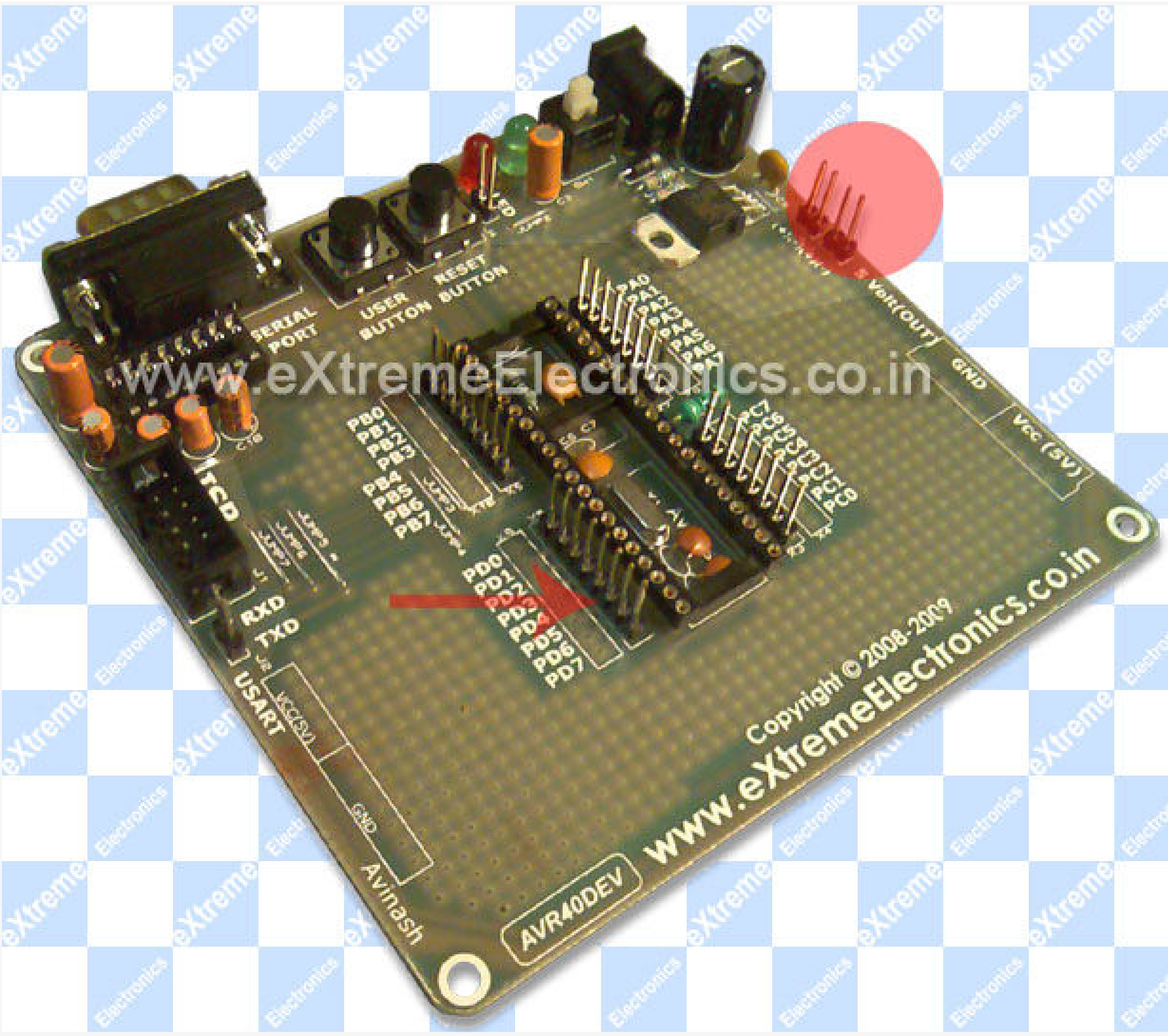


AVR ATmega32 Controlling a Futaba Servo Motor.

If you have development boards like **Low Cost AVR Development Board** or **xBoard v2.** Then you just need to add the servo motor. Connect RED wire of servo to 5v, BLACK wire to GND and the WHITE wire to OC1A PIN.



The RED Arrow Show the Position of OC1A pin on xBoard v2.0



The RED Arrow Show the Position of OC1A pin.

OC1A PIN is multiplexed with PD5 GPIO pin, that’s why the Low Cost AVR Development Board it is Marked as PD5. The RED Circle in above image is where you can find the 5v and GND supplies. So adding a servo in development board is very easy.

Downloads

- Complete AVR Studio Project For ATmega32.
- Complete AVR Studio Project For ATmega16(VS-10 Servo).
- HEX File for ATmega32.
- HEX File for ATmega16 (VS-10 Servo).
- HEX File for ATmega32 (VS-10 Servo).
- Only Servo.c file
- Complete Proteus VSM Simulation Project.

NOTE: HEX File can be uploaded to the boards by using a ISP Programmer like the **USB AVR Programmer v2.0**