Generating PWM signals using Timers in the ATMega chip

This is a guide to control unhacked servos using Timer1 on the ATMega8 chip. For those of you using ATMega128 or different Timers, the techniques and ideas can easily be applied to your situation.

First a little information on servo control:

Servos have three input wires:
Red = Vcc (between 4.8V and 6V is standard - see your servo data sheet)
Black = Ground
White = PWM input

Servo control is done by sending a *pulse width modulation*or PWM signal to the PWM input pin. The servo compares that signal to the actual position of the servo and adjusts the servo accordingly. The internal circuitry of the servo expects a constant 50Hz PWM signal (a 50 Hz signal is one that repeats every 20 ms).

1/50 Hz = 20 ms

The signal you are going to give the servo is one that is high (5V) for 1-2ms and low (0V) for the remainer of the 20ms period. The duration of the high signal determines the position that the servo attempts to maintain. Note that the servo must continually receive this signal in order to maintain its position.

1.0ms = full left
1.5ms = middle
2.0ms = full right

Assuming you are using a servo that has 90 degrees of rotation:
full left = 0 degrees
middle = 45 degrees
full right = 90 degrees

Different servos have different ranges of rotation so your own full left, middle, and full right maybe correspond to different angles. You may also find that in order to achieve the full range of motion you need to send the servo high pulses longer that 2.0ms or shorter than 1.0ms. Don't be affraid to experiment to find what your servo is capable of.

Phase and Frequency Correct mode:

To generate a 50Hz signal with a high signal that varies between 1-2ms, we will use the ***Phase and Frequency Correct mode*** of the Timer on your Atmel ATMega chip.

In Phase and Frequency Correct mode the timer starts at zero, counts up to a user defined value called ICRn (*n is the timer number. In our example, we will use Timer1 and thus ICR1*), and then counts back down to zero. We want the counting up and down process to take 20ms in order to generate the 50Hz signal.

In order to determine the ICR1 value, you should know two things:

- System clock speed
- Timer clock speed

Hopefully the system clock speed is something you already know...
For those of you using the MDMicro Maveric boards, it will most likely be 16MHz.

The timer speed is determined by the system clock speed divided by a prescaler. The prescaler is set by the CSn2:0 bits which are located in the TCCRn register (where n is the timer number). For example, Timer1's speed is set by the CS1 bits which are located in TCCR1B register.

These images are taken from the ATMega8 data sheet:

Timer/Counter 1 Control Register B – TCCR1B

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	

The clock section bits are the last three bits in the TCCR1B register.

Table 40. Clock Select Bit Description

CS12	CS11	CS10	Description
0	0	0	No clock source. (Timer/Counter stopped)
0	0	1	$clk_{I/O}$/1 (No prescaling)
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on T1 pin. Clock on falling edge.
1	1	1	External clock source on T1 pin. Clock on rising edge.

clk_{IO} is your system clock. For example if your system clock was 16 MHz, your timer could be 16 MHz, 2 MHz, 250 KHz, 62.5 KHz, or 15.625 KHz depending on your prescaler.

To calculate how to generate a desired frequency, the ATMega data sheet provides this equation:

$$f_{OCnxPFCPWM} = \frac{f_{clk\_I/O}}{2 \cdot N \cdot TOP}$$

This equation tells you the relationship betwen system clock frequency($f_{clk\_I/O}$), prescaler (N = 1, 8, 64, 256, or 1024), ICR1(TOP) and the output PWM frequency ($f_{OCnxPFCPWM}$).

To get 50 Hz with a system clock frequency of 16 MHz, you would need to use the following TOP(ICR1) values for the following prescalers:

Prescaler N = 1 then TOP(ICR1) = 160000
Prescaler N = 8 then TOP(ICR1) = 20000
Prescaler N = 64 then TOP(ICR1) = 2500
Prescaler N = 256 then TOP(ICR1) = 625
Prescaler N = 1024 then TOP(ICR1) = 156.25

Note: You cannot use prescaler 1 or 1024 to generate a 50 Hz PWM with a 16 MHz:
Prescaler 1 cannot be used since 160000 too large to fit in TCR1.
TCR1 is a 16 bit register with a range from 0 to 65535.
Prescaler 1024 should not be used since you cannot put decimals into ICR1.

I would suggest prescaler 8 and set ICR1 to 20000 because this will allow you to change OCR1A between 1000 and 2000 to obtain 1 - 2 ms high pulses.

If you are using CodeVision, this the 2000 KHz timer.

What is OCRxn?

If you have been reading the ATMega data sheet about PWM generation, you may be wondering what OCRxn is and what is the difference between ICRn. First note that in OCRxn, the x defines the Timer number and n defines which servo you are controlling. Most timers can control multiply servos. For example on Timer1 you can set OCR1A, OCR1B, and sometimes OCR1C (read the data sheet to find out how many servos a particular timer can support).

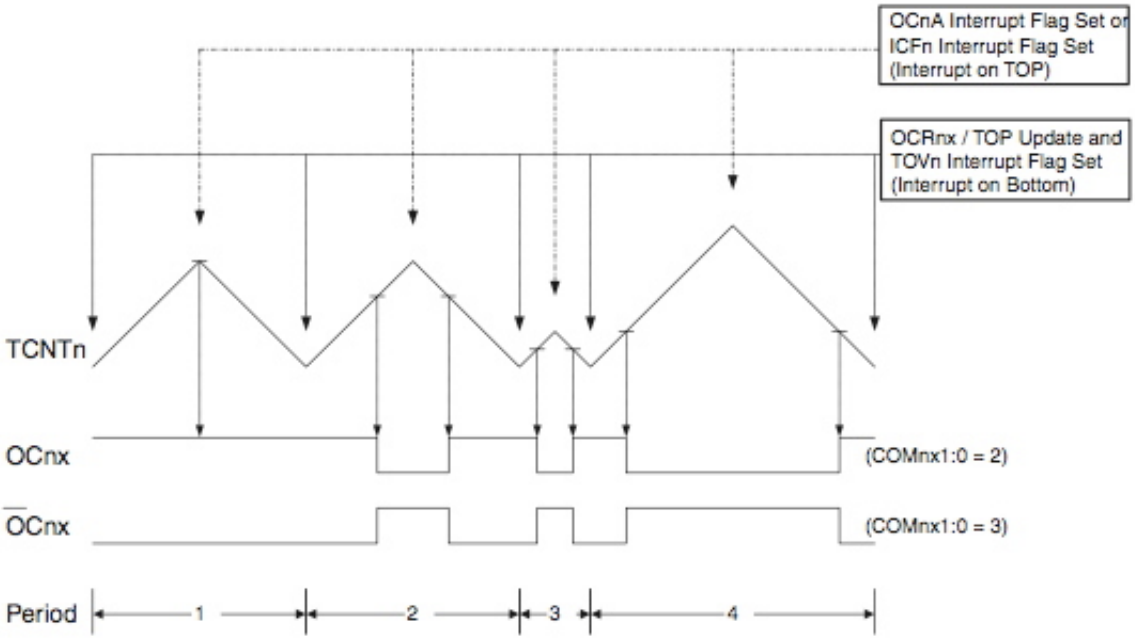
The simple explaination for OCRxn and ICRn is:

ICRn creates the 50 Hz PWM signal for the servo.
OCRxn controls the actually movement of the servo.

Once you set ICRn (ICR1 for Timer1), you won't change it again. However you will be constantly changing OCRxn (OCR1A for Timer1 servo A) to control the position of the servo.

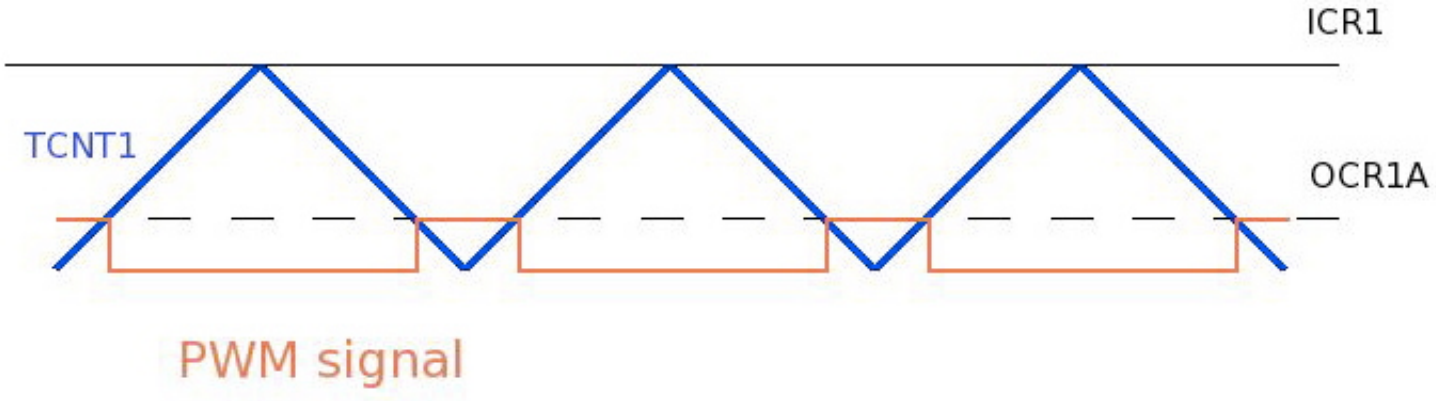
Here is diagram from the ATMega8 data sheet which shows how the OCRxn value controls the signal length. This diagram is a little confusing for our purpose because it shows ICRn changing and thus producing various frequency signals. We want a constant 50 Hz signal so we will never change ICRn after we set it.

Figure 40. Phase and Frequency Correct PWM Mode, Timing Diagram



(Page 94 in the ATMega8 data sheet)

Here is a diagram that shows how we will use OC1A to generate 1 - 2 ms high pulses to control servo position.



TCNT1 (Timer Counter 1) is the count value in Timer1.

It starts at zero, counts up to the value in ICR1 (Input Compare Register 1), and then counts back down to zero.

- When TCNT1 equals OCR1A (Output Compare Register 1A) while counting up, an ouput pin called OC1A (Output Compare 1A) goes high.
- When TCNT1 equals OCR1A while counting down, output pin OC1A goes low.
- *You can also think of the logic value on OC1A toggling every time TCNT1 = OCR1A.*

I know this can be a little confusing with all the similarily named registers and output pins. Here is a summary:

TCNT1 = Value of Timer1
ICR1 = Sets the upper limit to Timer1 (creates 50 Hz signal)
OCR1A = Sets when the PWM signal should toggle.
OC1A = Output pin where the PWM actually comes out of. You will have to look at your data sheet to see physically which pin it is.

To change the position of the servo, you would change OCR1A between 1000 and 2000.
Example C code:

```
OCR1A = 1500;
```

Or to have a **for** loop run through all the possible servo positions:

```
for(OCR1A=1000;OCR1A!=2000;OCR1A++)
  {
  delay_ms(1);
  }
```

I hope that helped you understand how to control a servo using an ATMega128 or ATMega8 chip.

Advantages to using the built-in PWM generation in the Timers are:

- Simpler code once you set all the registers correctly.
- Servo control takes no processing time.
- Your servo performance will not be affected by other interrupts or computationally heavy programs.

Disadvantages:

- You can only control a limited number of servos per timer. Usually between 2 and 3. However multiple timers have built-in PWM generation.
- No others that I can think of...

Troubleshooting:

If you are still unable to control a servo:

- Is the servo broken? Try using it on someone else's board that already has servos working.
- Also the servo should twitch when you plug it into power and ground. If not, something may be wrong with the servo itself.
- Did you find the correct OC1A output pin and set it up as an output pin in your code?
- A good idea would be to hook an oscilloscope up to your OC1A output pin to verify that you are getting a 50 Hz signal with the correct high and low pulses.
- If the signal is a different speed than you expected, are you sure about your system clock speed. Blown fuse bits will often set the ATMega's clock to an internal 1MHz clock.

[Guide to generating PWM Servo control code with CodeVision]

Last updated July 30, 2006. Please email me with any corrections or areas that need clarification.
Andrew Chambers (achamber at mil dot ufl dot edu)