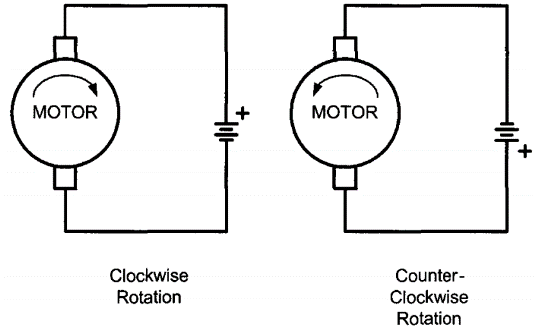
**DC motors**

A direct current (DC) motor is a widely used device that translates electrical pulses into mechanical movement. In the DC motor we have only (+) and (-) leads. Connecting them to a DC voltage source moves the motor in one direction. By reversing the polarity, the DC motor will move in the opposite direction. For example, the small fans used in many motherboards to cool the CPU are run by DC motors. When the leads are connected to the + and - voltage source, the DC motor moves. While a stepper motor moves in steps of 1 to 15 degrees. The maximum speed of a DC motor is indicated in rpm. The DC motor has two rpms: no-load and loaded. The manufacturer's datasheet gives the no-load rpm. The no-load rpm can be from a few thousand to tens of thousands. The rpm is reduced when moving a load and it decreases as the load is increased. DC motors also have voltage and current ratings. The nominal voltage is the voltage for that motor under normal conditions, and can be from 1 to 50 V, depending on the motor. As we increase the voltage, the rpm goes up. The current rating is the current consumption when the nominal voltage is applied with no load, and can be from 25 mA to a few Amps. As the load increases, the rpm is decreased, unless the current or voltage provided to the motor is increased, which in turn increases the torque. With a fixed voltage, as the load increases, the current (power) consumption of a DC motor is increased. If we overload the motor it will stall, and that can damage the motor due to the heat generated by high current consumption.

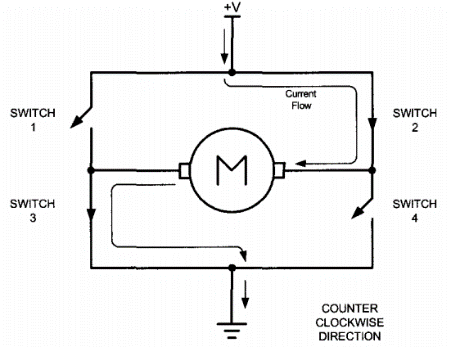
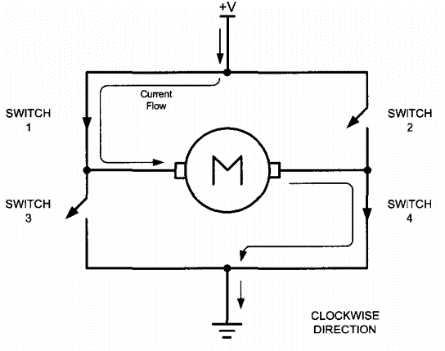
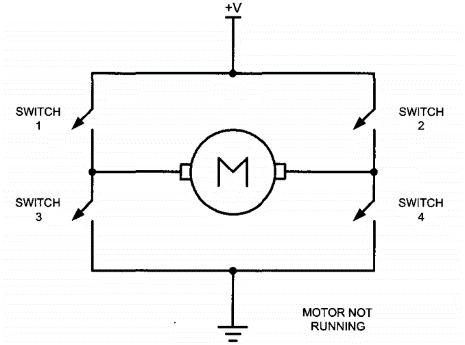
**Unidirectional control**

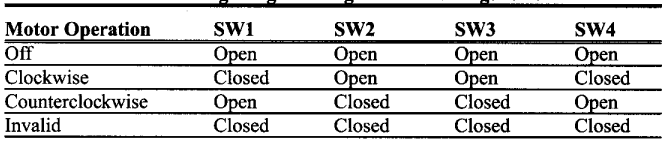
The Figure shows the DC motor rotation for clockwise (CW) and counterclockwise (CCW) rotations.



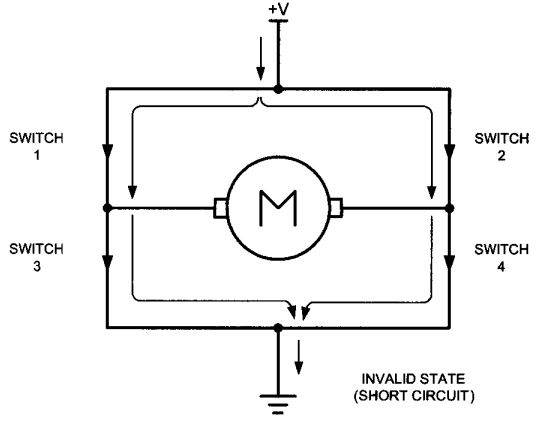
**Bidirectional control**

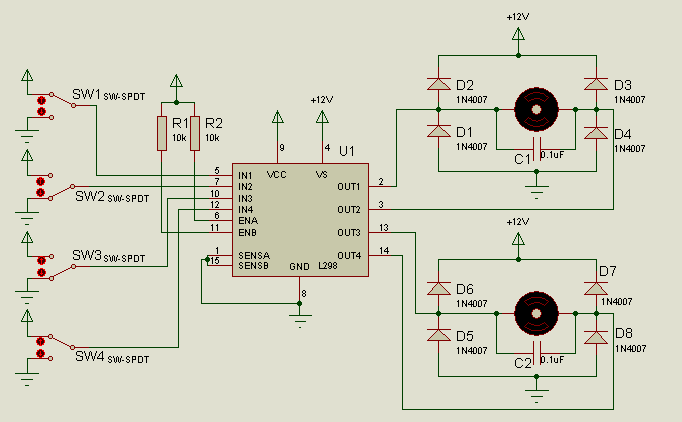
With the help of relays or some specially designed chips we can change the direction of the DC motor rotation. The figures show the basic concepts of H-bridge control of DC motors.





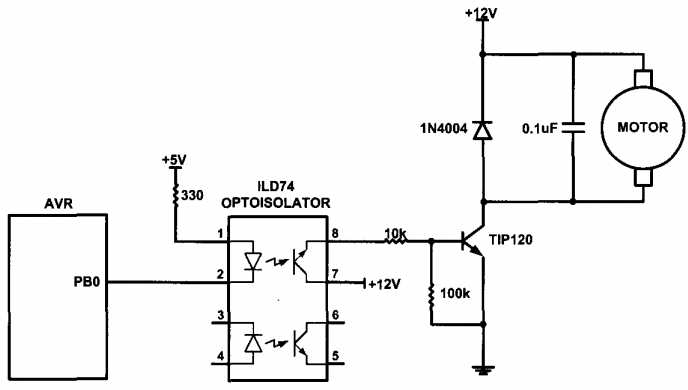
H-bridge control can be created using relays, transistors, or a single IC solution such as the L298. When using relays and transistors, you must ensure that invalid configurations do not occur.





DC motor control with opto-isolator

Opto-isolator is useful in many motor control applications. The figure shows the connections to a simple DC motor using an opto-isolator. Notice that the AVR is protected from EMI (Electromagnetic Compatibility (EMC / EMI)) created by motor brushes by using an opto-isolator and a separate power supply. These will reduce the possibility of damage to the control circuit. The separation of power supplies also allows the use of high-voltage motors. Notice that we use a decoupling capacitor across the motor; this helps reduce the EMI created by the motor. The motor is switched ON by clearing bit PB0.



**Pulse width modulation (PWM)**

The speed of the motor depends on three factors:

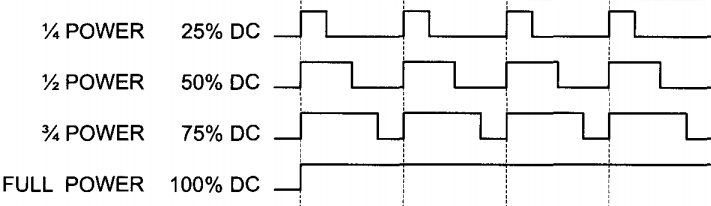
(a) Load

(b) Voltage

(c) current.

For a given fixed load we can maintain speed by using a method called pulse width modulation (PWM). By changing (modulating) the width of the pulse applied to the DC motor we can increase or decrease the amount of power provided to the motor, thereby increasing or decreasing the motor speed. Notice that, although the voltage has a fixed amplitude, it has a variable duty cycle. That means the wider the pulse, the higher the speed. PWM is so widely used in DC motor control that some microcontrollers come with the PWM circuitry embedded in the chip. In such microcontrollers all we have to do is load the proper registers with the values of the high and low portions of the desired pulse, and the rest is taken care of by the microcontroller. This allows the microcontroller to do other things. For microcontrollers without PWM circuitry, we must create the various duty cycle pulses using software, which prevents the microcontroller from doing other things. The ability to control the speed of the DC motor using PWM is one reason that DC motors are often preferred over AC motors. AC motor speed is dictated by the AC frequency of the voltage applied to the motor and the frequency is generally fixed. As a result, we cannot control the speed of the AC motor when the load is increased.

Figure: Pulse Width Modulation Comparisons



**Example01:** Write a program to monitor the status of a switch and perform the following

1. If S/W = 0, the DC motor moves with 80% duty cycle pulse.
2. If S/W = 1, the DC motor moves with 20% duty cycle pulse.

**PWM MODES IN 8-BIT TIMERS**

The ATmega16 comes with 3 timers, which can be used as wave generators. CPU itself to create the equivalent of PWM outputs. The advantage of using the built-in PWM feature of the AVR is that it gives us the option of programming the period and duty cycle, therefore relieving the CPU to do other important things.

**Frequency of the generated wave in Fast PWM mode**

In Fast PWM mode, the timer counts from 0 to top (0xFF in 8-bit counters) and then rolls over. So, the frequency of the generated wave is 1/256 of the frequency of timer clock. We know that the frequency of the timer clock can be selected using the prescaler. So, in 8-bit timers the frequency of the generated wave can be calculated as follows (N is determined by the prescaler):

**Duty cycle of the generated wave in Fast PWM mode**

The duty cycle of the generated mode can be determined using the OCR0 register. When COM01:00 = 10 (in non-inverted mode), the bigger OCR0 value results in a bigger duty cycle; When OCR0 = 255, the OC0 is 256 clocks out of 256 clocks, which means always high (duty cycle= 100%). Generally speaking, the OC0 is high, for a total of OCR0 + 1 clocks. So, the duty cycle can be calculated using the following formula in non-inverted mode:

Similarly, the duty cycle formula for inverted mode is as follows:

**Math01:** To generate a wave with duty cycle of 75% in non-inverted mode, calculate the OCR0

Solution:

75 = ((OCR0 + 1) / 256) x 100

Or, ((OCR0 + 1) / 256) = 75 / 100

Or, OCR0 + 1 = (75 x 256) / 100

Or, OCR0 + 1 = (75 x 256) / 100

Or, OCR0 + 1 = 192

Or, OCR0 = 191

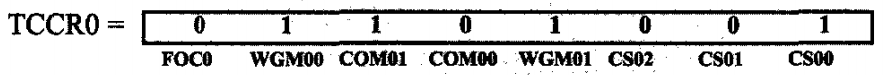
**Math02:** Find the value for TCCR0 Register to initialize Timer0 for Fast PWM mode, non-inverted PWM wave generator, and no prescaler.

Solution:

WGM01:00 = 11 = Fast PWM mode

CS02:00 = 001 = No prescaler

COM01:00 = 10 = Non-inverted PWM



**Loading values into the OCRx register in PWM modes**

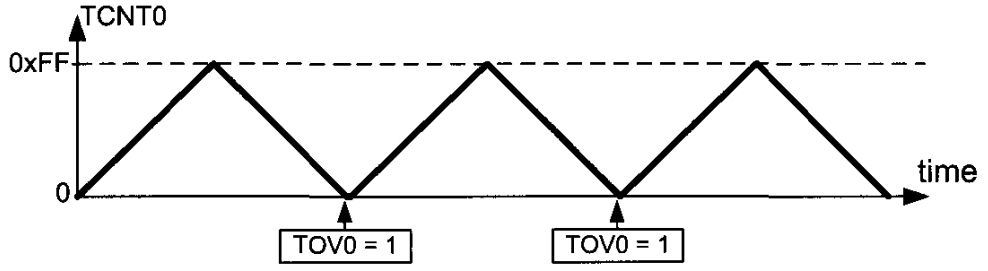
In the non-PWM modes (CTC mode and Normal mode), when we load a value into the OCR0 register, the value will be loaded instantly into the OCR0 register, but in the PWM modes (Fast PWM and Phase correct PWM), there is a buffer between us and the OCR0 register. When we read/write a value from/into the OCR0 we are dealing with the buffer. The contents of the buffer will be loaded into the OCR0 register only when the TCNT0 reaches to its topmost value. The top value is 0xFF in the 8-bit timers.

**Phase correct PWM mode programming of Timer0**

In the Phase correct PWM, the TCNT0 goes up and down like a yo-yo! First it counts up until it reaches the top value. Then it counts down until it reaches zero. The TOV0 flag is set whenever it reaches zero.

**Phase correct PWM mode**

In the figure you see the reaction of the waveform generator when compare match occurs in Phase correct PWM mode. When COM01:00 = 00 the OC0 pin operates as an I/O port. When COM01:00 = 10, the waveform generator clears the OC0 pin on compare match when counting up, and sets it on compare match when counting down. This mode is called non-inverted Phase correct PWM.



**Frequency of the generated wave in Phase correct PWM mode**

**Duty cycle of the generated wave in Phase correct PWM mode**

So, the duty cycle can be calculated using the following formula in non-inverted mode:

Similarly, the duty cycle formula for inverted mode is as follows:

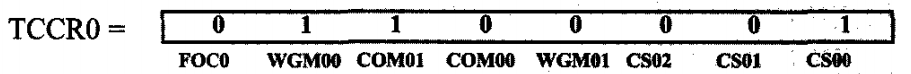
**Math03:** Find the value for TCCR0 for Phase correct PWM, non-inverted PWM wave generator, and no prescaler.

Solution:

WGM0 1:00 = 01 = Phase correct PWM mode

COM01:00 = 10 = Non-inverted PWM

CS02:00 = 001 = No prescaler



**Math03:** Assuming XTAL = 8 MHz, using non-inverted mode, write a program that generate a wave with frequency of 15,686 Hz and duty cycle of 75%

Solution:

We know that,

Or, 15686 = 8MHz / (510 x N)

Or, 510 x N x 15686 = 8MHz

Or, N = 1 (So, No prescaler)

Again we know,

Or, 75 = ((OCR0 + 1) x 100) / 256

Or, ((OCR0 + 1) x 100) = 75 x 256

Or, OCR0 + 1 = 19200 / 100

Or, OCR0 = 192 – 1

Or, OCR0 = 191

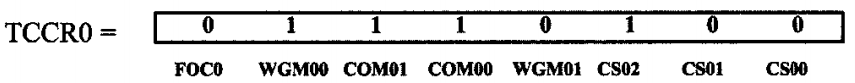
**Math04:** Find the value for TCCR0 for Phase correct PWM, inverted PWM wave generator, and prescaler = 256

Solution:

WGM01:00 = 01 = Phase correct PWM mode

COM01:00 = 11 = Inverted PWM

CS02:00 = 100 = Scale 256



**Math05:** Assuming XTAL = 8 MHz, using inverted mode, write a program that generates a wave with frequency of 61 Hz and duty cycle of 87.5%

Solution:

We know that,

Or, 61 = 8 MHz / (510 x N)

Or, 510 x N x 61 = 8 MHz

Or, N = 256 (Approx.) (So, 256 prescaler)

Again we know,

Or, (255 – OCR0) x 100 = Duty Cycle x 255

Or, 255 – OCR0 = (87.5 x 255) / 100

Or, 255 - OCR0 = 223

Or, OCR0 = 255 – 223

Or, OCR0 = 32 (Approx.)

**Difference between the waves generated by Phase correct PWM and** **FastPWM**

In Fast PWM, the phase of the wave is different for different duty cycles, while it remains unchanged in the Phase correct PWM. For driving motors, it is preferable to use Phase correct PWM rather than Fast PWM. In Fast PWM the frequency of the generated wave is twice that of the Phase correct mode. Thus, Fast PWM mode is preferable when we need to generate waves with high frequencies.

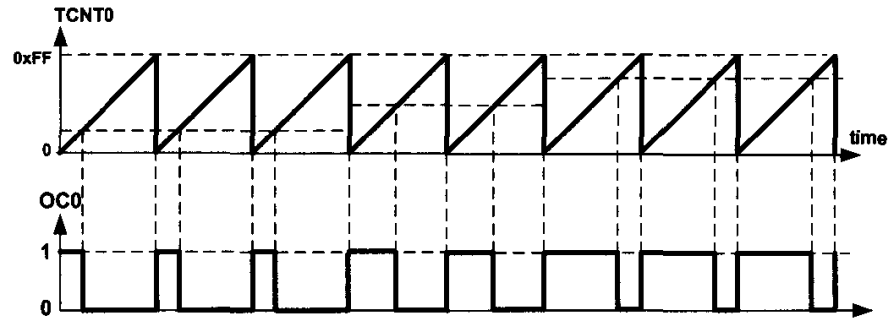


Figure: Fast PWM

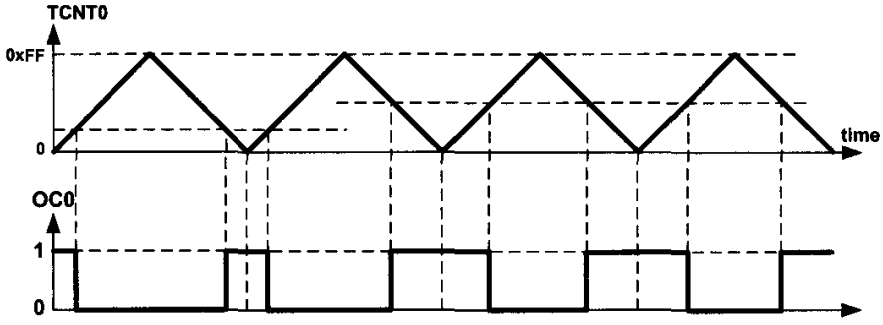


Figure: Phase Correct PWM

**Generating waves using Timer2**

Timer2 is an 8-bit timer. Therefore, it works similar to Timer0. The differences are register names, output port, and the prescaler values of TCCRn register.

**8-BIT PWM PROGRAMMING IN C**

**Example02:** Assuming XTAL = 8 MHz, using non-inverted mode, write a program that generates a wave with frequency of 31250 Hz and duty cycle of 75%

**Solution:**

We know that, for non-inverted mode

Or, 31250 = 8MHz / (256 x N)

Or, 256 x N x 31250 = 8MHz

Or, N = 1 (So, No prescaler)

Again we know that, for non-inverted mode

Or, 75 = ((OCR0 + 1) x 100) / 256

Or, ((OCR0 + 1) x 100) = 75 x 256

Or, OCR0 + 1 = 19200 / 100

Or, OCR0 = 192 – 1

Or, OCR0 = 191

**Example03:** Assuming XTAL = 8 MHz, using non-inverted mode, write a program that generates a wave with frequency of 3906.25 Hz and duty cycle of 37*.5%*

**Solution:**

We know that, for non-inverted mode

Or, 3906.25= 8MHz / (256 x N)

Or, 256 x N x 3906.25 = 8MHz

Or, N = 8 (So, 8 prescaler)

Again we know that, for non-inverted mode

Or, 37*.5* = ((OCR0 + 1) x 100) / 256

Or, ((OCR0 + 1) x 100) = 37*.5* x 256

Or, OCR0 + 1 = 96

Or, OCR0 = 96 – 1

Or, OCR0 = 95

**Example04:** Assuming XTAL = 8 MHz, inverted mode, write a program that generates a wave with frequency of 3906.25 Hz and duty cycle of 37*.5%*

**Solution:**

We know that, for inverted mode

Or, (255 – OCR0) x 100 = Duty Cycle x 256

Or, 255 – OCR0 = (37.5 x 256) / 100

Or, 255 - OCR0 = 96

Or, OCR0 = 255 – 96

Or, OCR0 = 159

Or, 3906.25= 8MHz / (256 x N)

Or, 256 x N x 3906.25 = 8MHz

Or, N = 8 (So, 8 prescaler)

**Example05:** Assuming XTAL = 8 MHz, using phase correct non-inverted mode, write a program that generates a wave with frequency of 15686 Hz and duty cycle of 75%

**Solution:**

15686 = 8MHz / (510 x N)

Or, N = 1 (No prescaler)

OCR0 = (255 x 75) / 100

Or, OCR0 = 191

**Example06:** Assuming XTAL = 8 MHz, using phase correct inverted mode, write a program that generates a wave with frequency of 61 Hz and duty cycle of 87.5%

**Solution:**

We know that, for inverted mode

61 = 8MHz / (510 x N)

Or, N = 256 (Approx.)

255 - OCR0 = (255 x 87.5) / 100

Or, OCR0 = 32