**Introduction**

**Pulse Width Modulation (PWM)** is a technique by which the width of a pulse is varied while keeping the frequency constant.

Why do we need to do this? Let’s take an example of controlling DC motor speed, more the Pulse width more the speed. Also, there are applications like controlling light intensity by PWM.

A period of a pulse consists of an **ON** cycle (5V) and an **OFF** cycle (0V). The fraction for which the signal is ON over a period is known as the **duty cycle**.

Duty Cycle (In %) =

E.g. Consider a pulse with a period of 10ms which remains ON (high) for 2ms.The duty cycle of this pulse will be

D = 2ms / 10ms = 20%

Through the PWM technique, we can control the power delivered to the load by using the ON-OFF signal.

Pulse Width Modulated signals with different duty cycle are shown below

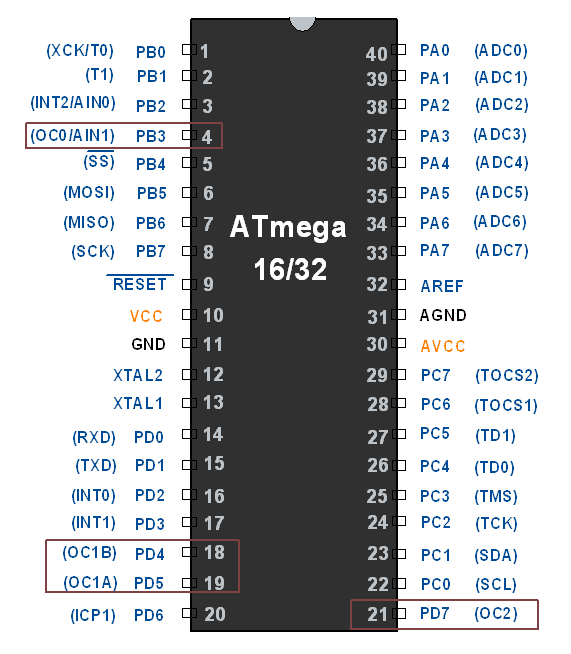
**PWM Duty Cycle Waveforms**

**AVR ATmega PWM**

ATmega has an inbuilt PWM unit. As we know, ATmega has 3 Timers T0, T1, and T2 which can be used for PWM generation. Mainly there are two modes in PWM.

1. Fast PWM
2. Phase correct PWM

We need to configure the Timer Register for generating PWM. PWM output will be generated on the corresponding Timer’s output compare pin (OCx).

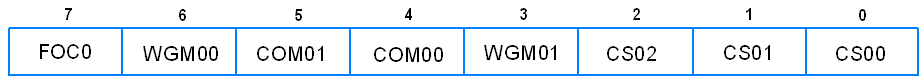


**PWM out Pins on AVR ATmega16/32**

**Configuring Timer0 for PWM generation**

It is simple to configure PWM mode in Timer. We just need to set some bits in the TCCR0 register.

**TCCR0: Timer Counter Control Register 0**

****

**Bit 7- FOC0:**Force compare match

Write only bit, which can be used while generating a wave. Writing 1 to this bit will force the wave generator to act as if a compare match has occurred.

**Bit 6, 3 - WGM00, WGM01**: Waveform Generation Mode

| **WGM00** | **WGM01** | **Timer0 mode selection bit** |
| --- | --- | --- |
| 0 | 0 | Normal |
| 0 | 1 | CTC (Clear timer on Compare Match) |
| **1** | **0** | **PWM, Phase correct** |
| **1** | **1** | **Fast PWM** |

**Bit 5:4 - COM01:00:**

1. When WGM00: WGM01= 11 i.e. **Fast PWM.**Compare Output Mode

waveform generator on OC0 pin

| **COM01** | **COM00** | **Mode Name** | **Description** |
| --- | --- | --- | --- |
| 0 | 0 | Disconnected | The normal port operation, OC0 disconnected |
| 0 | 1 | Reserved | Reserved |
| **1** | **0** | **Non-inverted** | Clear OC0 on compare match, set OC0 at TOP |
| **1** | **1** | **Inverted PWM** | Set OC0 on compare match, clear OC0 at TOP |

         2. When WGM00: WGM01= 10 i.e. **Phase correct PWM.**Compare Output Mode

waveform generator on OC0 pin

| **COM01** | **COM00** | **Description** |
| --- | --- | --- |
| 0 | 0 | The normal port operation, OC0 disconnected |
| 0 | 1 | Reserved |
| **1** | **0** | Clear OC0 on compare match when up-counting, set OC0 on compare match when down-counting |
| **1** | **1** | Set OC0 on compare match when up-counting, Clear OC0 on compare match when down-counting |

**Bit 2:0 - CS02:CS00:**Clock Source Select

These bits are used to select a clock source. When CS02: CS00 = 000, then timer is stopped. As it gets a value between 001 to 101, it gets a clock source and starts as the timer.

| **CS02** | **CS01** | **CS00** | **Description** |
| --- | --- | --- | --- |
| **0** | **0** | **0** | No clock source (Timer / Counter stopped) |
| **0** | **0** | **1** | clk (no pre-scaling) |
| **0** | **1** | **0** | clk / 8 |
| **0** | **1** | **1** | clk / 64 |
| **1** | **0** | **0** | clk / 256 |
| **1** | **0** | **1** | clk / 1024 |
| **1** | **1** | **0** | External clock source on T0 pin. clock on falling edge |
| **1** | **1** | **1** | External clock source on T0 pin. clock on rising edge. |

**Fast PWM mode**

To set Fast PWM mode, we have to set WGM00: 01= 11. To generate a PWM waveform on the OC0 pin, we need to set COM01:00= 10 or 11.

COM01:00= 10 will generate Noninverting PWM output waveform and COM01:00= 11 will generate Inverting PWM output waveform. See fig.

void PWM\_init()

{

/\*set fast PWM mode with non-inverted output\*/

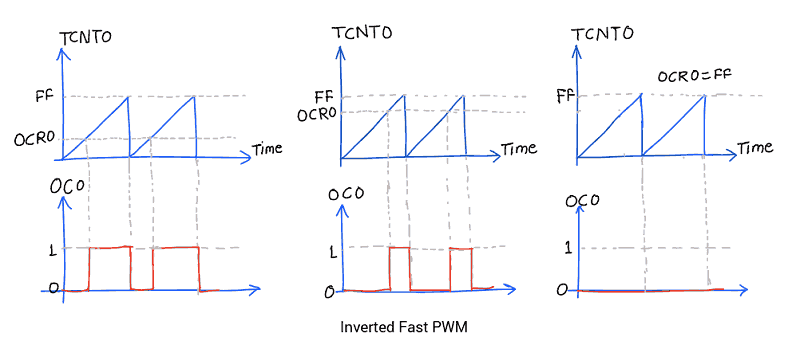
TCCR0 = (1<<WGM00) | (1<<WGM01) | (1<<COM01) | (1<<CS00);

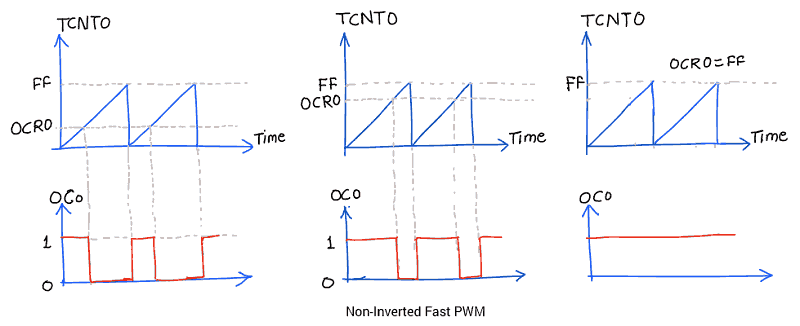
DDRB|=(1<<PB3); /\*set OC0 pin as output\*/

}

**Setting Duty cycle:** we have to load value in the OCR0 register to set the duty cycle.

255 value for 100% duty cycle and 0 for 0% duty cycle. Accordingly, if we load value 127 in OCR0, the Duty cycle will be 50%.

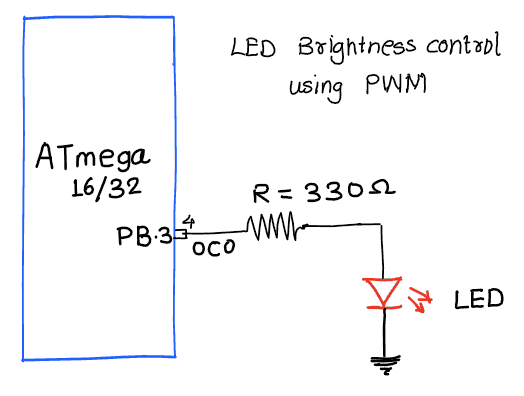




The advantage of using PWM mode in AVR is that it is an inbuilt hardware unit for waveform generation and once we set the PWM mode and duty cycle, this unit starts generating PWM and the controller can do other work.

**Example**

 Control LED brightness using Fast PWM.



/\*

AVR ATmega16 PWM to control LED brightness

http://www.electronicwings.com

\*/

#define F\_CPU 8000000UL

#include "avr/io.h"

#include <util/delay.h>

void PWM\_init()

{

/\*set fast PWM mode with non-inverted output\*/

TCCR0 = (1<<WGM00) | (1<<WGM01) | (1<<COM01) | (1<<CS00);

DDRB|=(1<<PB3); /\*set OC0 pin as output\*/

}

int main ()

{

unsigned char duty;

PWM\_init();

while (1)

{

for(duty=0; duty<255; duty++)

{

OCR0=duty; /\*increase the LED light intensity\*/

\_delay\_ms(8);

}

for(duty=255; duty>1; duty--)

{

OCR0=duty; /\*decrease the LED light intensity\*/

\_delay\_ms(8);

}

}

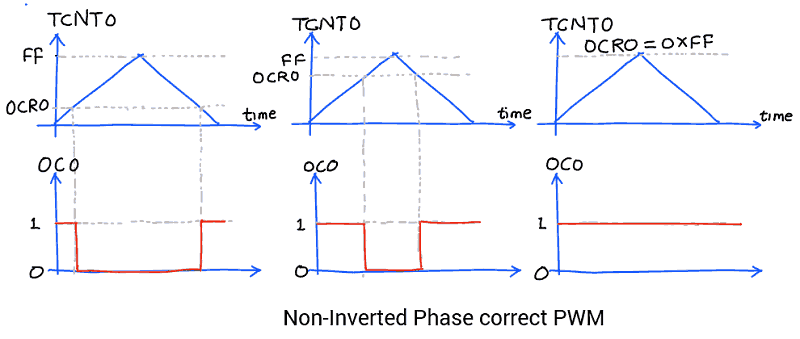
}

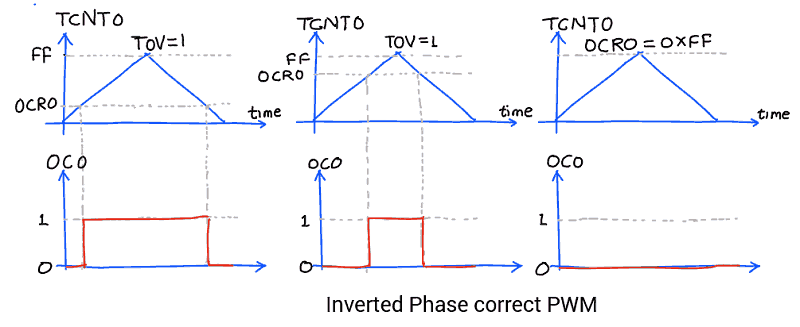
**Phase correct PWM mode**

To set Phase correct PWM, we just have to set the TCCRO register as follows.

We can set the output waveform as inverted or non-inverted. See fig.

TCCR0 = (1<<WGM00) | (1<<COM01) | (1<<CS00);





Similarly, we can set PWM output on the other three OCx pins using Timer1 and Timer2.

PWM output is somehow close to the Analog output. We can use it as analog output for generating sine wave, audio signals, etc. it is also referred to as DDS.

Graphical LCD 128x64 interfacing with AVR ATmega16/ATmega32.

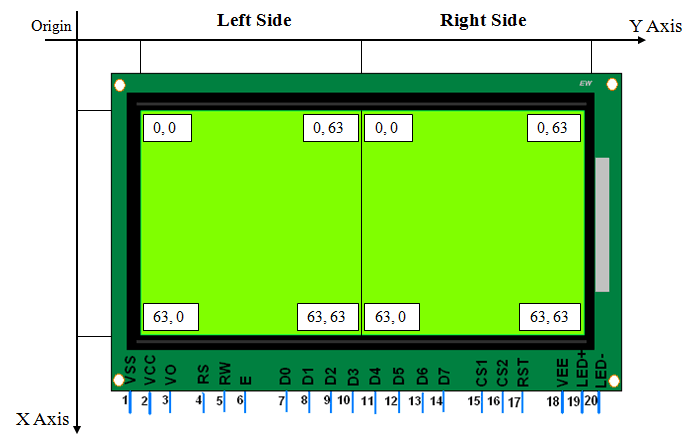
**Introduction**

GLCD is a display device that can be used in embedded systems for displaying data and/or images/custom characters.

* Basically, 128x64 Graphical LCD is a matrix of pixels.
* Each pixel is accessed by their X and Y address.
* We can simply visualize any pixel by making its value HIGH (1).

Hence, we can make any graphical design pixel by pixel using GLCD.

To get familiar with GLCD pins and their functions refer to [GLCD 128x64](http://electronicwings.com/sensors-modules/glcd-128x64).



**Basic Structure of GLCD 128x64 Displays**

**Programming GLCD**

Let's program the AVR ATmega16 microcontroller to print text character on GLCD JHD12864E.

**Initialization**

To initialize the display, we need to do the below steps,

* Send Display OFF command i.e. 0x3E
* Send Y address e.g. here 0x40 (Start address).
* Send X address (Page) e.g. here 0xB8 (Page0).
* Send Z address (Start line) e.g. here 0xC0 (from 0th line).
* Now send Display ON command i.e. 0x3F

**GLCD\_Init function**

Input arguments: It has no input arguments.

Return type: It does not return any data type.

void GLCD\_Init() /\* GLCD initialize function \*/

{

Data\_Port\_Dir = 0xFF;

Command\_Port\_Dir = 0xFF;

/\* Select both left & right half of display & Keep reset pin high \*/

Command\_Port |= (1 << CS1) | (1 << CS2) | (1 << RST);

\_delay\_ms(20);

GLCD\_Command(0x3E); /\* Display OFF \*/

GLCD\_Command(0x40); /\* Set Y address (column=0) \*/

GLCD\_Command(0xB8); /\* Set x address (page=0) \*/

GLCD\_Command(0xC0); /\* Set z address (start line=0) \*/

GLCD\_Command(0x3F); /\* Display ON \*/

}

**Command Write**

To write a command do the below steps

* Send command on data pins.
* Make RS = 0 (Command Register) and RW = 0 (Write Operation).
* Make High to Low transition on Enable pin of min. 1us period.

**GLCD\_Command function**

Input arguments: It has an input argument of Command.

Return type: It does not return any data type.

void GLCD\_Command(char Command) /\* GLCD command function \*/

{

Data\_Port = Command; /\* Copy command on data pin \*/

Command\_Port &= ~(1 << RS); /\* Make RS LOW for command register\*/

Command\_Port &= ~(1 << RW); /\* Make RW LOW for write operation \*/

Command\_Port |= (1 << EN); /\* HIGH-LOW transition on Enable \*/

\_delay\_us(5);

Command\_Port &= ~(1 << EN);

\_delay\_us(5);

}

**Data Write**

To write data do the below commands

* Send Data on data pins.
* Make RS = 1 (Data Register) and RW = 0 (Write Operation).
* Make High to Low transition on Enable pin of min 1 us period.

**GLCD\_Data function**

Input arguments: It has input argument Data.

Return type: It does not return any data type.

void GLCD\_Data(char Data) /\* GLCD data function \*/

{

Data\_Port = Data; /\* Copy data on data pin \*/

Command\_Port |= (1 << RS); /\* Make RS HIGH for data register \*/

Command\_Port &= ~(1 << RW); /\* Make RW LOW for write operation \*/

Command\_Port |= (1 << EN); /\* HIGH-LOW transition on Enable \*/

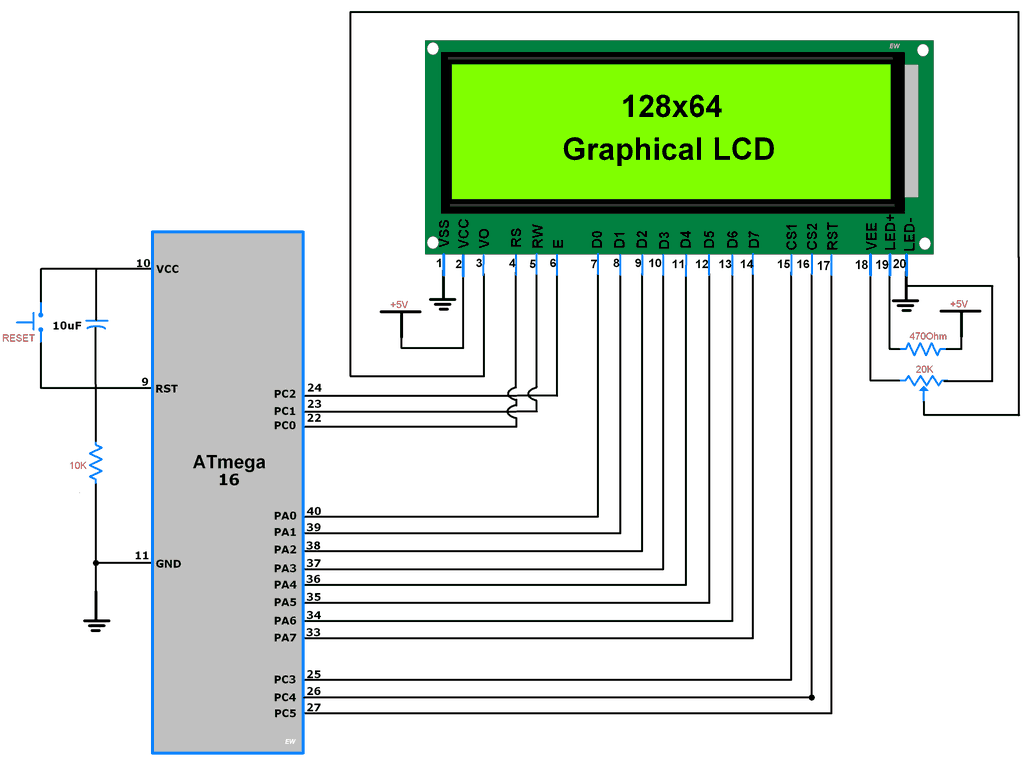
\_delay\_us(5);

Command\_Port &= ~(1 << EN);

\_delay\_us(5);

}

**Interfacing Diagram**



**Interfacing GLCD With ATmega 16**

**Program for Text Print**

/\*

\* ATmega\_GLCD\_TextFont

\* http://electronicwings.com

\*/

#define F\_CPU 8000000UL /\* Define CPU clock Freq 8MHz \*/

#include <avr/io.h> /\* Include AVR std. library file \*/

#include <util/delay.h> /\* Include delay header file \*/

#include <stdio.h> /\* Include std i/o library file \*/

#include "Font\_Header.h"

#define Data\_Port PORTA /\* Define data port for GLCD \*/

#define Command\_Port PORTC /\* Define command port for GLCD \*/

#define Data\_Port\_Dir DDRA /\* Define data port for GLCD \*/

#define Command\_Port\_Dir DDRC /\* Define command port for GLCD \*/

#define RS PC0 /\* Define control pins \*/

#define RW PC1

#define EN PC2

#define CS1 PC3

#define CS2 PC4

#define RST PC5

#define TotalPage 8

void GLCD\_Command(char Command) /\* GLCD command function \*/

{

Data\_Port = Command; /\* Copy command on data pin \*/

Command\_Port &= ~(1 << RS); /\* Make RS LOW for command register\*/

Command\_Port &= ~(1 << RW); /\* Make RW LOW for write operation \*/

Command\_Port |= (1 << EN); /\* HIGH-LOW transition on Enable \*/

\_delay\_us(5);

Command\_Port &= ~(1 << EN);

\_delay\_us(5);

}

void GLCD\_Data(char Data) /\* GLCD data function \*/

{

Data\_Port = Data; /\* Copy data on data pin \*/

Command\_Port |= (1 << RS); /\* Make RS HIGH for data register \*/

Command\_Port &= ~(1 << RW); /\* Make RW LOW for write operation \*/

Command\_Port |= (1 << EN); /\* HIGH-LOW transition on Enable \*/

\_delay\_us(5);

Command\_Port &= ~(1 << EN);

\_delay\_us(5);

}

void GLCD\_Init() /\* GLCD initialize function \*/

{

Data\_Port\_Dir = 0xFF;

Command\_Port\_Dir = 0xFF;

/\* Select both left & right half of display & Keep reset pin high \*/

Command\_Port |= (1 << CS1) | (1 << CS2) | (1 << RST);

\_delay\_ms(20);

GLCD\_Command(0x3E); /\* Display OFF \*/

GLCD\_Command(0x40); /\* Set Y address (column=0) \*/

GLCD\_Command(0xB8); /\* Set x address (page=0) \*/

GLCD\_Command(0xC0); /\* Set z address (start line=0) \*/

GLCD\_Command(0x3F); /\* Display ON \*/

}

void GLCD\_ClearAll() /\* GLCD all display clear function \*/

{

int i,j;

/\* Select both left & right half of display \*/

Command\_Port |= (1 << CS1) | (1 << CS2);

for(i = 0; i < TotalPage; i++)

{

GLCD\_Command((0xB8) + i);/\* Increment page \*/

for(j = 0; j < 64; j++)

{

GLCD\_Data(0); /\* Write zeros to all 64 column \*/

}

}

GLCD\_Command(0x40); /\* Set Y address (column=0) \*/

GLCD\_Command(0xB8); /\* Set x address (page=0) \*/

}

void GLCD\_String(char page\_no, char \*str)/\* GLCD string write function \*/

{

unsigned int i, column;

unsigned int Page = ((0xB8) + page\_no);

unsigned int Y\_address = 0;

float Page\_inc = 0.5;

Command\_Port |= (1 << CS1); /\* Select Left half of display \*/

Command\_Port &= ~(1 << CS2);

GLCD\_Command(Page);

for(i = 0; str[i] != 0; i++) /\* Print char in string till null \*/

{

if (Y\_address > (1024-(((page\_no)\*128)+FontWidth)))

break;

if (str[i]!=32)

{

for (column=1; column<=FontWidth; column++)

{

if ((Y\_address+column)==(128\*((int)(Page\_inc+0.5))))

{

if (column == FontWidth)

break;

GLCD\_Command(0x40);

Y\_address = Y\_address + column;

Command\_Port ^= (1 << CS1);

Command\_Port ^= (1 << CS2);

GLCD\_Command(Page + Page\_inc);

Page\_inc = Page\_inc + 0.5;

}

}

}

if (Y\_address>(1024-(((page\_no)\*128)+FontWidth)))

break;

if((font[((str[i]-32)\*FontWidth)+4])==0 || str[i]==32)

{

for(column=0; column<FontWidth; column++)

{

GLCD\_Data(font[str[i]-32][column]);

if((Y\_address+1)%64==0)

{

Command\_Port ^= (1 << CS1);

Command\_Port ^= (1 << CS2);

GLCD\_Command((Page+Page\_inc));

Page\_inc = Page\_inc + 0.5;

}

Y\_address++;

}

}

else

{

for(column=0; column<FontWidth; column++)

{

GLCD\_Data(font[str[i]-32][column]);

if((Y\_address+1)%64==0)

{

Command\_Port ^= (1 << CS1);

Command\_Port ^= (1 << CS2);

GLCD\_Command((Page+Page\_inc));

Page\_inc = Page\_inc + 0.5;

}

Y\_address++;

}

GLCD\_Data(0);

Y\_address++;

if((Y\_address)%64 == 0)

{

Command\_Port ^= (1 << CS1);

Command\_Port ^= (1 << CS2);

GLCD\_Command((Page+Page\_inc));

Page\_inc = Page\_inc + 0.5;

}

}

}

GLCD\_Command(0x40); /\* Set Y address (column=0) \*/

}

int main(void)

{

GLCD\_Init(); /\* Initialize GLCD \*/

GLCD\_ClearAll(); /\* Clear all GLCD display \*/

GLCD\_String(0,"Atmel"); /\* Print String on 0th page of display \*/

while(1);

}

**Output Image**

