LIQUID CRYSTAL DISPLAY (LCD)

LCD Modules can present textual information to user. It's like a cheap "monitor" that you can hook in all of your gadgets.

They come in various types. The most popular one is 16x2 LCD Module. It has 2 rows and 16 columns.

A 16X2 LCD MODULE

HOW TO CONNECT LCD TO THE MCU

In order to connect LCD to the MCU, you have to first make physical connections between the pins of LCD and MCU. In order to connect the LCD, you have to use one PORT of the MCU completely for this purpose. Suppose you chose a particular PORT. The LCD module must be connected to the port bits as follows:

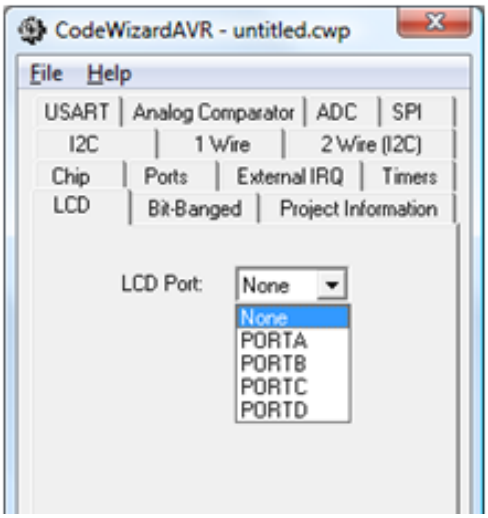
[LCD]	[MCU]
1 GND-	GND
2 +5V-	VCC
3 VLC-	LCD HEADER Vo
4 RS -	bit 0 of that PORT
5 RD -	bit 1
6 EN -	bit 2
11 D4 -	bit 4
12 D5 -	bit 5
13 D6 -	bit 6
14 D7 -	bit 7

Leave other pins of the LCD open, i.e. do not connect them anywhere.
These connections that you have to make will also appear when you are using CVAVR Wizard.

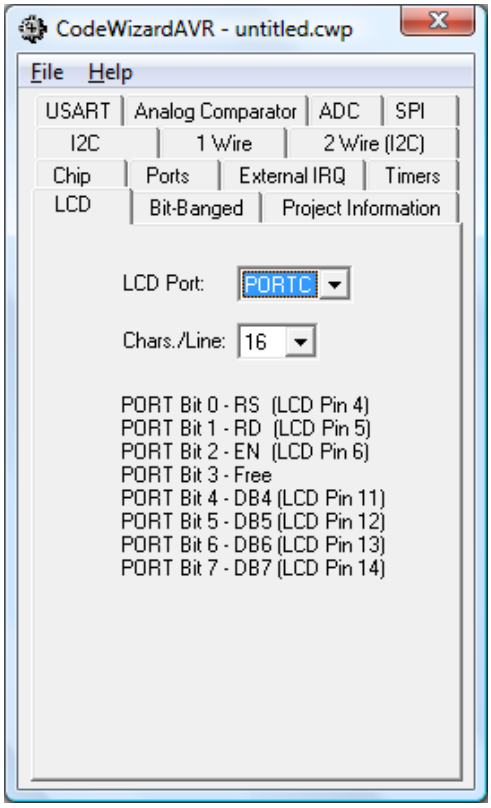
USING CVAVR TO CONTROL AN LCD

First, open the CVAVR Wizard in front of you as usual.

Select your chip and clock frequency. Then click on LCD tab. Select the PORT at which you want to make LCD connections.



Set chars/line text box to 16, because you will be using 16*2 LCD. All the connections of the LCD other than power connections will be displayed in the wizard. Make other settings and then select generate save, save and exit from file menu. To save the file, give the file name.



Once this is done, you are ready to use LCD display form your program. You can now use inbuilt functions to write to the LCD screen.

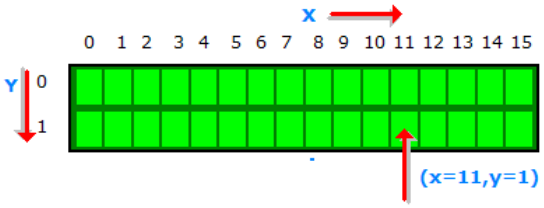
PROGRAMMING THE LCD

We mainly call the following functions while using a LCD

1. `lcd_clear();` : This will clear the display and bring the cursor back to (0,0)
2. `lcd_gotoxy(int,int);` : This will take the cursor to the desired row and column.

   For example the following will take the cursor to (11,1) i.e. 12th column of second line.

   ```
   lcd_gotoxy(11,1);
   ```



   Now anything you write to the LCD will be printed at (11,1).

3. `lcd_putsf(String);` : To print the String at the current position of the cursor. Using this function, you can output a constant String as :

   For example, `lcd_putsf("0003");` will yield

4. `lcd_putchar(char);` : To print a character at the current position of the cursor. We may also specify the ASCII value of the character.

A String in C is an array of characters. To declare a String, you will use

```
char[<max size of the string>] <name>;
```

For example, `char[25] a;`

Here, we have used 25 as a safe limit to our number of characters in the string.

String can be assigned as:

```
A = "Hello";
```

Or, when initializing, as:

```
char[] a = "Hello World";
```

5. `lcd_puts(String);` : This is generally used to print a variable String. (Note: requires you to include the file "stdio.h" in your program)

This function is different from `lcd_putsf();`

Now suppose that you have to display a variable on the LCD screen, for example, a number. What will you do?

Well, `lcd_puts();` comes to the rescue. What you will do is first convert the number into a string and then use it in the function `lcd_puts();`

You will have do the first part using a first defined in the file "stdio.h", so first include this file as follows:

```
#include <stdio.h>
```

Now declare a String variable in which you will hold the converted value. Now use the function:

```
void sprintf(String1, String2, <list of variables>);
```

Where String1 is the String variable where you declared above. The String2 gives the format of the output String. The following example makes it clear:

If we wish to display

"Seconds = i"

Where i is a variable (Note: we are printing the value of i), then the command would be

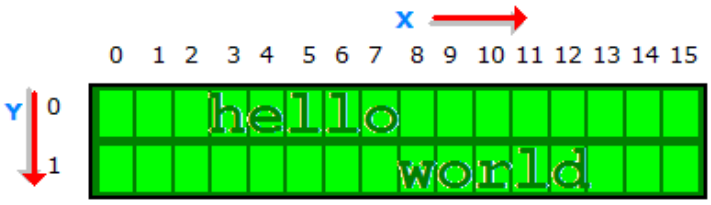
```
char[20] s;

sprintf(s,"seconds=%d",i);

lcd_puts(s);
```

"Seconds = " would be the constant part or the textual part and %d specifies that there has to be an integer at that place, which is i.

SAMPLE PROGRAM:

```
#include  <lcd.h>

lcd_clear();

lcd_goto(3,0);

lcd_putsf("hello");

lcd_goto(8,1);

lcd_putsf("world");
```

OUTPUT:

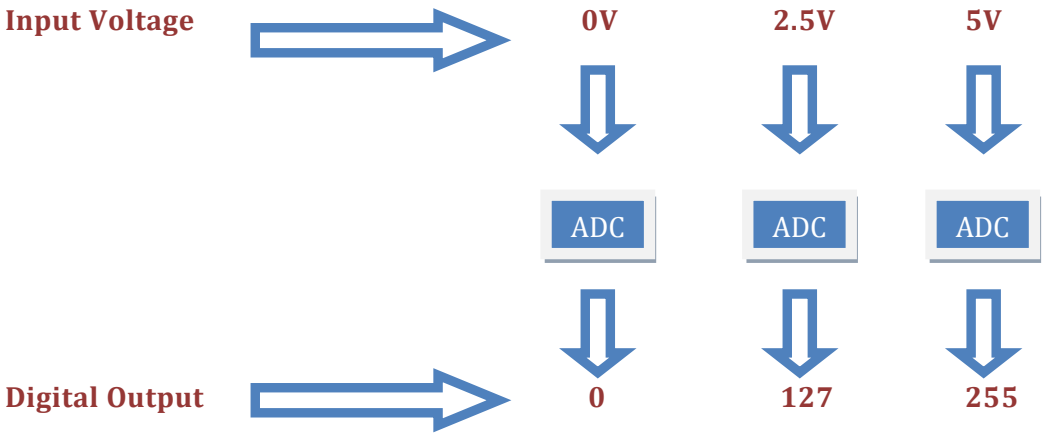


ADC

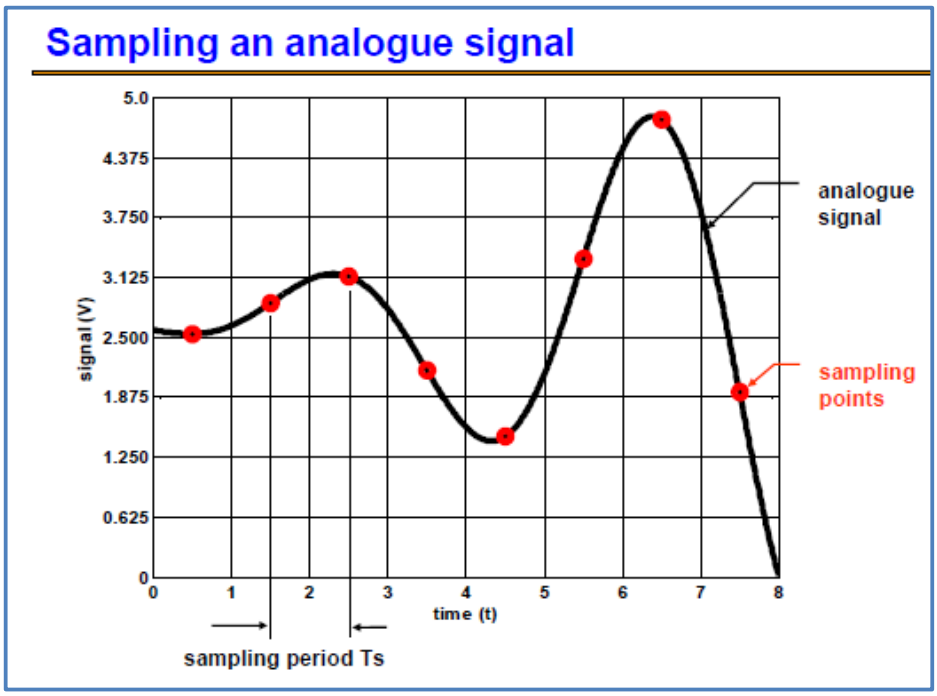
INTRODUCTION

Most of the physical quantities around us are continuous. By continuous, we mean that the quantity can take any value between two extremes. For example, the atmospheric temperature can take any value within a certain range. If an electrical quantity is made to vary directly in proportion to this value then what we have is an Analogue Signal. Now we have brought a physical quantity into the electrical domain. The electrical quantity in most cases is voltage. To bring this quantity into digital domain we have to convert this into digital form. For this an ADC or analogue to digital converter is needed. Most modern MCU including AVRs have an ADC on chip.

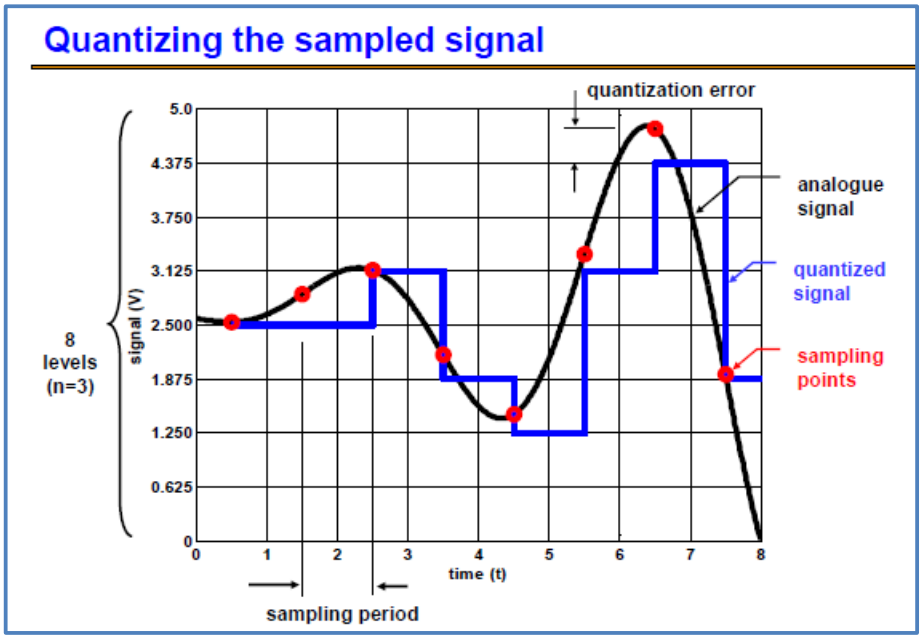
An ADC converts an input voltage into a number. An ADC has a resolution. A 8 bit ADC has a range of 0-255. (2^8=256) The ADC also has a Reference Voltage (ARef). When the input voltage is GND the output is 0 and when the input voltage is equal to ARef the output is 255. So the input range is 0 to ARef and the output range is 0 to 255.



You can see that any analogue signal is not perfectly converted – a factor that affects the output quality is the "sampling rate". The ADC cannot continuously read the input signal and change its output – it does so in certain time intervals. The frequency at which it samples the input is called its sampling rate.



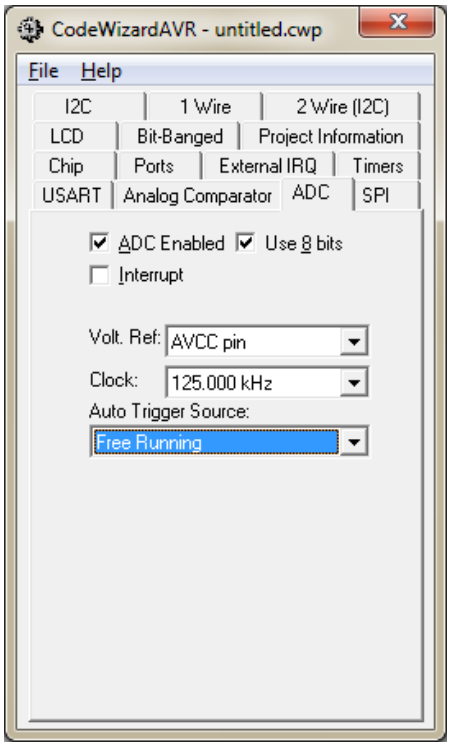
Also, the value of the analogue signal read is not stored perfectly - for example, a voltage of 2.501 would be read as 2.5V i.e. 127 in digital format. Thus, the signal is "quantized", or, in other terms, there is certain graininess to the digital signal that you obtain. How accurate your digital signal is depends on your resolution – higher resolutions will make your digital signal more accurate.



Now that you know the basics of ADC, let us see how we can use the inbuilt ADC of AVR MCUs. The ADC is shared with PORTA, that is, you can the PORTA terminals for your ADC requirements.

The ADC can be run in two modes – single conversion and free running. In single conversion the ADC does the conversion and then stops, while in free mode it is continuously converting. It does a conversion and then starts the next conversion immediately after that.

You can set various parameters for the ADC in the ADC tab of CodeVisionAVR's Configuration Wizard:



- Check "ADC Enabled" to initialize the ADC.
- To use an 8 bit ADC, check "Use 8 bits". Else, the ADC will have a resolution of 10 bits.
- You can also generate an interrupt every time the ADC completes a sampling cycle. To do so, Check "Interrupt".
- "Volt. Ref." will let you set the value of ARef. You can select any of the 3 options.
- "Clock" will let you select the sampling rate.
- "Auto Trigger Source" will let you select the source for triggering the ADC conversion. Select "Free Running" to convert continuously or "Analogue comparator" to compare once at the start of the execution of your code. There are also other events available for triggering the ADC conversion.
- Click on File -> Generate, Save and Exit to generate your program.

IMPLEMENTING ADC IN YOUR CODE

It is fairly easy to obtain an ADC value. All you need is this function:

```
read_adc(x);
```

Where ‘x’ is the pin number at which you are receiving the analogue output. This function returns an integer value. A sample implementation would be like:

```
a = read_adc(5);
```