
AN8000.13

Application Note

***Driving LCD Displays using XE8000
standard I/O ports***

Table of Contents

1	Introduction.....	3
1.1	Document structure.....	3
2	LCD Basics.....	4
2.1	LCD direct drive.....	5
2.2	LCD multiplex drive	6
2.3	Mixing multiplex and direct drive	7
3	Standard implementations	9
3.1	Driving a 4x8 segments LCD using standard microcontroller I/O ports.....	9
4	Application example.....	10
4.1	Introduction.....	10
4.2	Hardware	10
4.3	Defining the LCD pins	10
4.4	Software – Constant & Variables	11
4.5	Software - Functions	13
5	Annex A – Application source code.....	16
5.1	Main file	16
5.2	LCD Driver C file	17
5.3	LCD Driver Header file	22

1 INTRODUCTION

This application note intends to explain how to realize a simple LCD driver using the microcontrollers I/O ports.

The main goal of this document is to understand how to build a cheap LCD interface for small LCD displays for example a STD 4.5 digit TN display.

1.1 DOCUMENT STRUCTURE

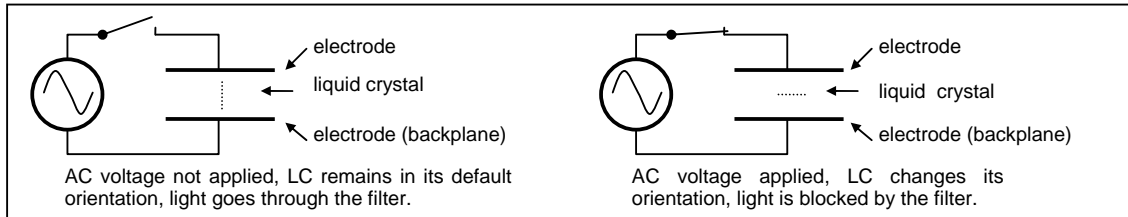
This document is separated in 5 sections

- **Introduction:** This section.
- **LCD Basics:** This section explains what is a LCD and different method to drive them.
- **Standard implementations:** This section explains how to implement the different driving methods described in the chapter "LCD basics".
- **Application example:** This section shows how to realize a LCD driver using an existing display.
- **Application source code:** Contains the source code of the application.

2 LCD BASICS

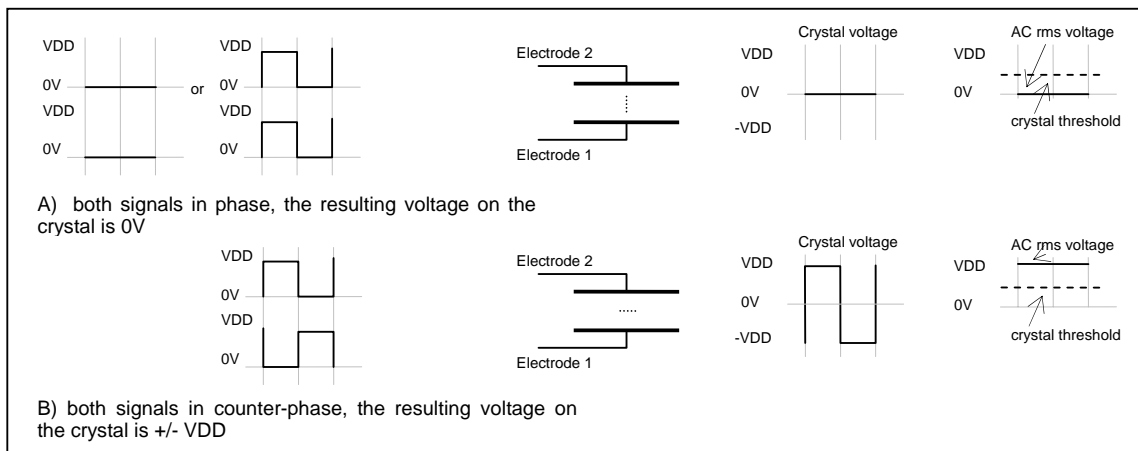
The Liquid Crystal Display (LCD) is a passive display technology. This means that this type of display does not emit light; instead, it uses the ambient light from the environment. By manipulating this light, it displays images using very little power. In addition LCDs can operate with relatively low voltage (2V and up). This has made LCDs the preferred technology whenever low power consumption and compact size are critical.

LCDs are composed of two electrodes with liquid crystals placed between them. By applying a voltage to the electrodes, the crystals change their orientation and modify the light polarization, which is revealed by a polarized filter placed on the surface of the display.



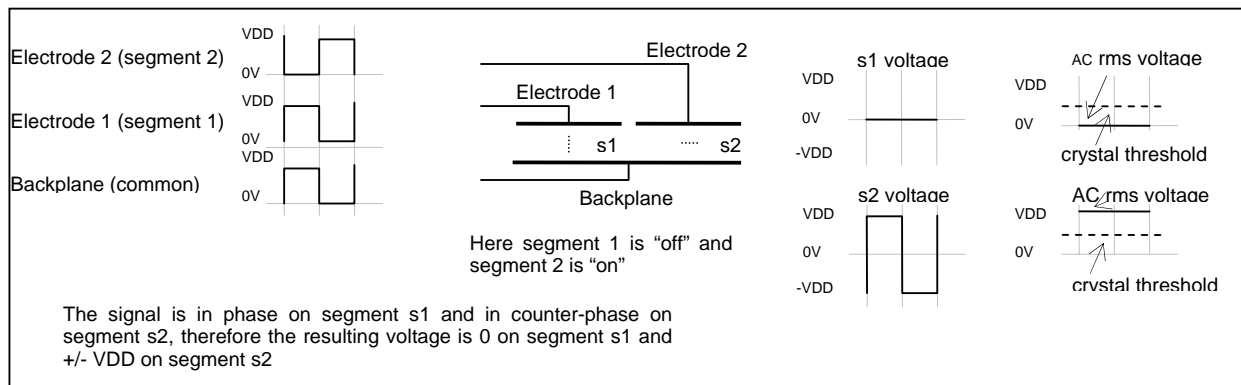
Liquid crystals are very sensitive to constant electric fields and DC voltages can cause an electrochemical reaction which destroys the liquid crystals irreversibly, therefore only DC-free AC-voltages should be applied.

As soon as the amplitude of the applied AC field is superior to a given threshold (dependant upon the LCD geometry and technology), the segment turns dark. If this amplitude is smaller than the same threshold, the segment remains transparent. If the amplitude is very near to the threshold voltage, the segment remains gray or glitches. Due to the specific characteristic described above, the way to drive a liquid crystal segment is as follows:



2.1 LCD DIRECT DRIVE

In multiple segments driving, one of the electrodes is common to all the segments, this electrode is called the backplane (see drawing below). The backplane sends a square signal and the other electrodes send the same signal with no phase shift for transparent segments or with a phase shift of 180° for dark segments.

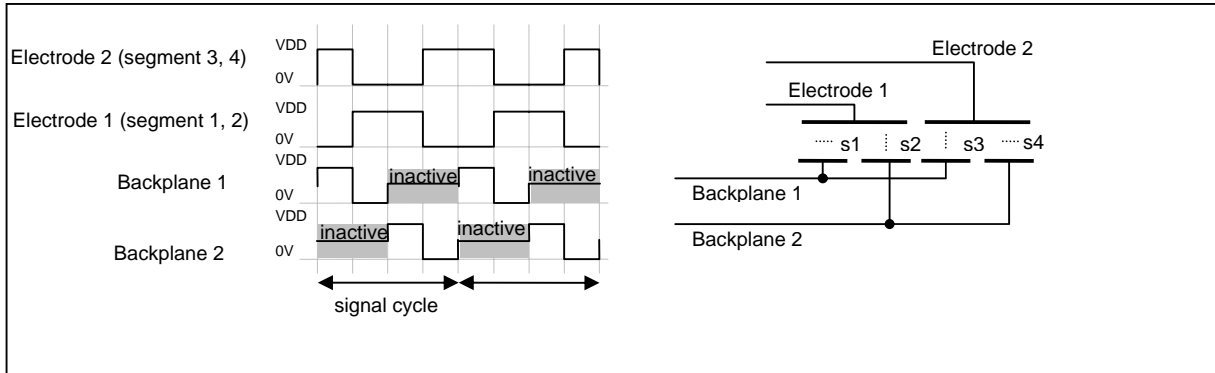


This way of driving LCD's segments is the simplest one and the one that offers the best contrast, since the resulting voltage on electrodes is VDD or 0V.

The drawback is that one needs one pin to drive each segment (plus one for the backplane), this may quickly lead to huge numbers of connections to drive simple LCD displays.

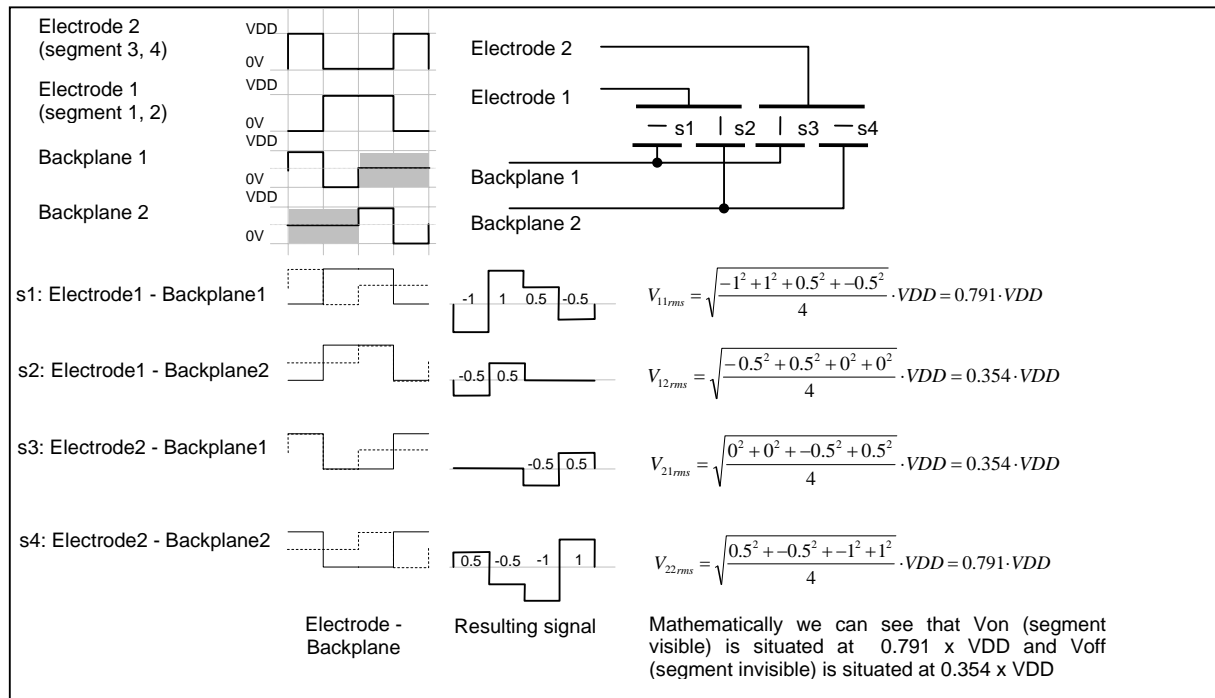
2.2 LCD MULTIPLEX DRIVE

In order to increase the number of segments one can increase the number of backplanes, this method enables one electrode to drive more than one segment. In the example below each electrode drives two segments (Multiplex by 2). Segments must also have an “inactive” phase when their electrode carries the signal for the segments on the other backplanes. This is done by adding intermediate signals onto each backplane. The AC voltage on one segment is then integrated over a complete cycle through all the backplanes.



This way of driving introduces another parameter, now the resulting voltages between Electrodes and backplanes are not 0V or VDD but values in between and the time to change all the segments is multiplied by the number of backplanes.

If we do some calculations we can have the resulting voltage for each segment. See figure below.



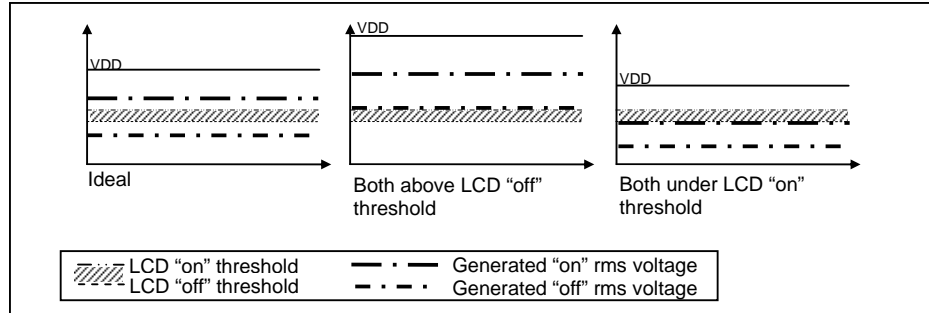
Multiplying the backplanes allows one to address more segments, the drawback of this technique is that the delta between Von and Voff is reduced; this implies that the contrast is also reduced.

LCD technology has fixed thresholds for Von and Voff, developers must take into account these thresholds and ensure that generated Voff and Von must respect these thresholds.

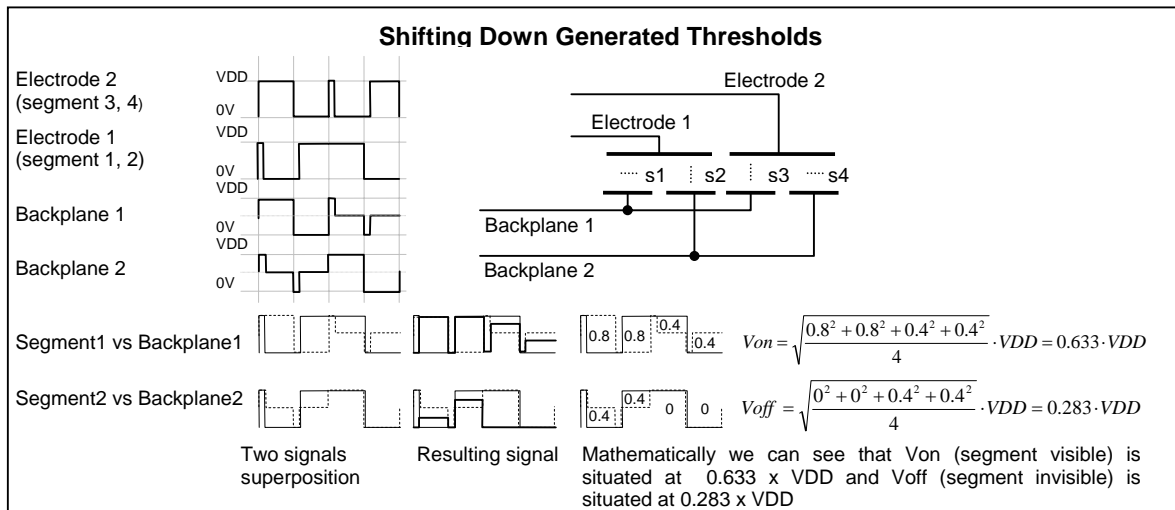
The standard manner to increase contrast when using multiple backplanes is to generate 3 levels of voltage instead of 2, but this technique is too complex to be implemented when using standard microcontroller I/O ports.

2.3 MIXING MULTIPLEX AND DIRECT DRIVE

The main problem encountered using multiplex drive is that both “on” and “off” generated voltages may be above or below the LCD thresholds. This is even more of an issue with bias $\frac{1}{2}$ and high multiplexing ratios where “on” and “off” generated voltages are not far from each other.

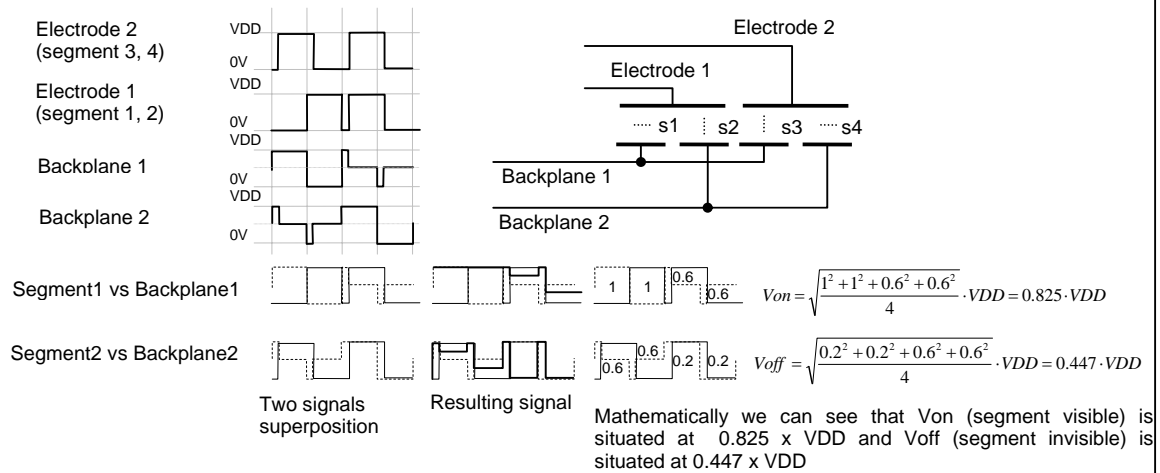


As the direct drive does not have this limitation (full signal is applied to the segment or no signal is applied), a simple way to shift the applied voltage is to mix multiplex and direct drive (see diagrams below). When using direct drive for a short period of time, all backplanes are connected together to form a single backplane and a uniform signal is applied to all electrodes, increasing or decreasing the mean RMS voltage applied to the whole LCD. By changing the ratio between the uniform direct drive signal and the addressed multiplexed signal, one moves the mean applied “on” and “off” levels.



As shown in figures above, direct drive waveforms are forced during a short period of time at the beginning of each pulse forcing them to 0V will shift down the average resulting values of generated thresholds and vice-versa.

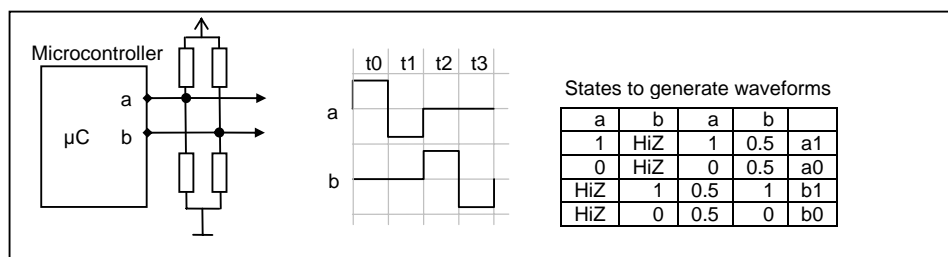
Shifting Up Generated Thresholds



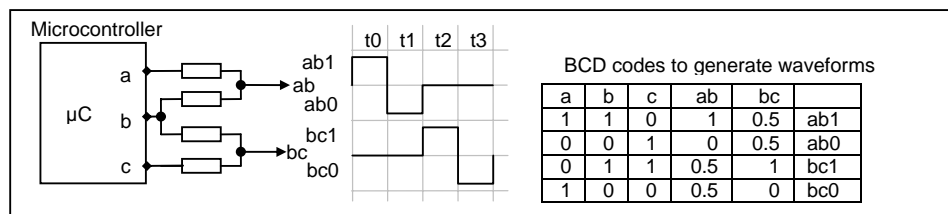
3 STANDARD IMPLEMENTATIONS

When using multiple backplanes, the signal on the backplanes becomes analog; this means when using microcontroller's standard I/O ports that external circuitry must be added in order to produce this analog waveform.

Following are examples on how to set up the circuitry in order to obtain multiplex by two waveforms.



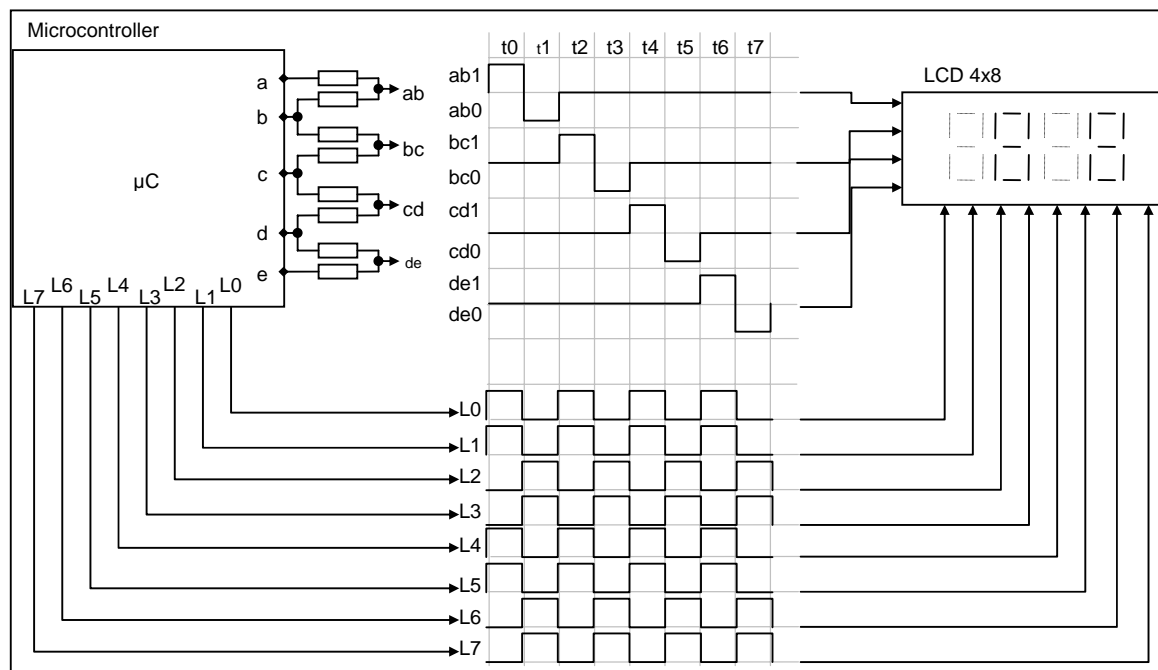
The figure above shows the most common way to create the analog waveforms playing with the High impedance capability of the I/O ports



The figure above shows an alternative for I/O ports that do not have the High impedance capability. Note that putting the I/O port as an input cannot be considered as a true HiZ, and in this case can lead to uncertain behavior since value of the input voltage is $V_{DD}/2$.

3.1 DRIVING A 4X8 SEGMENTS LCD USING STANDARD MICROCONTROLLER I/O PORTS

Application schematic



4 APPLICATION EXAMPLE

4.1 INTRODUCTION

In this chapter we will setup an example application using an LCD, based on the theory of the previous chapters.

The goal is to be able to develop an LCD driver with any simple LCD display.

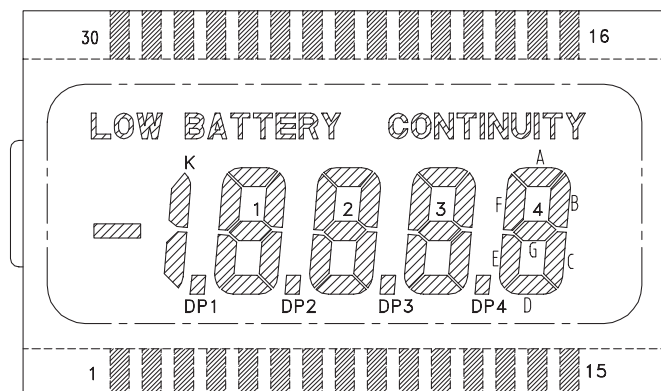
Knowing the pin assignment of the LCD will allow you to develop the corresponding driver.

4.2 HARDWARE

- 1 XE8000MP
- 1 XE8000EV101
- 1 Display LCD DISPLAYTECH 066-P2 MUX3 36 seg

4.2.1 LCD Hardware

The LCD hardware is described below:



Pin mapping:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16-30
1F	1A	1B	2F	2A	2B	3F	3A	3B	4F	4A	4B	COM1			NC
1E	1G	1C	2E	2G	2C	3E	3G	3C	4E	4G	4C		COM2		
DP1	1D	K	DP2	2D	-	DP3	3D	LOBAT	DP4	4D	CONT			COM3	

4.3 DEFINING THE LCD PINS

If we want to drive all the segments we have to use PC0 to PC7 and PB0 to PB3 for the signal driving and PB5 to PB7 to generate the common waveforms.

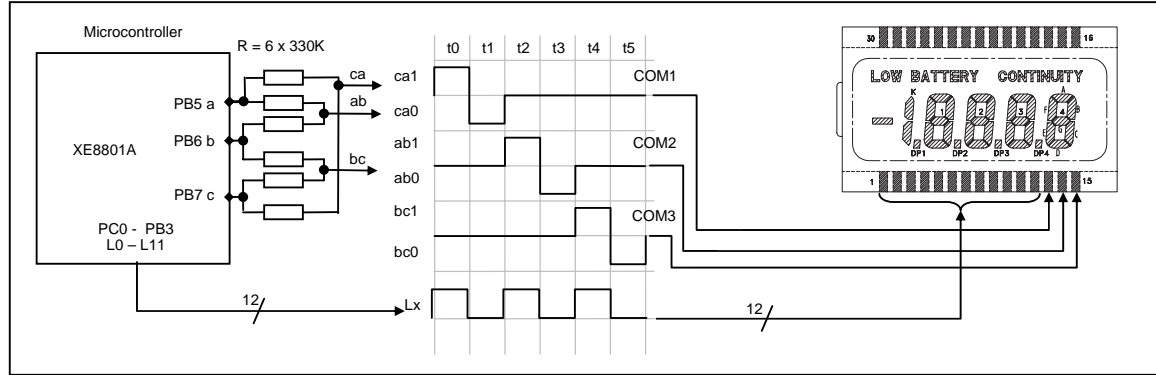
In this case, pins have been assigned as follows:

ports	PC 0	PC 1	PC 2	PC 3	PC 4	PC 5	PC 6	PC 7	PB 0	PB 1	PB 2	PB 3	PB 6-7	PB 6-5	PB 5-7	
LCD pins	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16-30
com1	1F	1A	1B	2F	2A	2B	3F	3A	3B	4F	4A	4B	COM 1			NC
com2	1E	1G	1C	2E	2G	2C	3E	3G	3C	4E	4G	4C		COM 2		
com3	DP1	1D	K	DP2	2D	-	DP3	3D	LO BAT	DP4	4D	CONT			COM 3	

4.3.1 Choosing The Common Waveform Generation Schematic

Since the XE8801A can only use analog mode (equivalent to Hi-Z) when using pairs of pins, the alternative chained resistor schematic as shown in standard implementations, has to be used. The chained resistor schematic is more complex to implement but allows the use of single pins rather than pairs, without using the analog mode

4.3.2 Application Schematic



4.4 SOFTWARE – CONSTANT & VARIABLES

Definitions of constants and variables

4.4.1 LCD Com Constant Table

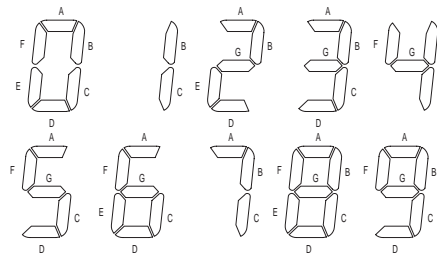
In order to generate the common wave forms we have to define an array containing six constants corresponding to the six possible states on the common lines: AB1, AB0, BC1, BC0, CA1 and CA0.

a	b	c	ab	bc	ca		PB7	PB6	PB5	PB4	PB3	PB2	PB1	PB0	hex
1	0	1	0.5	0.5	1	ca1	1	0	1	0	0	0	0	0	0xA0
0	1	0	0.5	0.5	0	ca0	0	1	0	0	0	0	0	0	0x40
1	1	0	1	0.5	0.5	ab1	0	1	1	0	0	0	0	0	0x60
0	0	1	0	0.5	0.5	ab0	1	0	0	0	0	0	0	0	0x80
0	1	1	0.5	1	0.5	bc1	1	1	0	0	0	0	0	0	0xC0
1	0	0	0.5	0	0.5	bc0	0	0	1	0	0	0	0	0	0x20

4.4.2 LCD Data Constant Table

To display numbers we need to have another constant table that contains the seven segments states for each number.

	a	b	c	d	e	f	g	hex
0	1	1	1	1	1	1	0	0x7E
1	0	1	1	0	0	0	0	0x30
2	1	1	0	1	1	0	1	0x6D
3	1	1	1	1	0	0	1	0x79
4	0	1	1	0	0	1	1	0x33
5	1	0	1	1	0	1	1	0x5B
6	1	0	1	1	1	1	1	0x5F
7	1	1	1	0	0	0	0	0x70
8	1	1	1	1	1	1	1	0x7F
9	1	1	1	0	0	1	1	0x73



4.4.3 LCD Lines Variable Table

This variable is a picture of the display and is used to refresh the LCD; one must modify this table in order to see the display changing at the next refresh.

This variable is the most important. It represents the actual LCD display state.

The table below represents the variable organization; we can see that every segment is represented for each phase of the LCD frame. (Note that "!" represent the Boolean operation NOT)

		1	2	3	4	5	6	7	8	9	10	11	12
		PC0	PC1	PC2	PC3	PC4	PC5	PC6	PC7	PB0	PB1	PB2	PB3
13	c1+	!1f	!1a	!1b	!2f	!2a	!2b	!3f	!3a	!3b	!4f	!4a	!4b
	c1-	1f	1a	1b	2f	2a	2b	3f	3a	3b	4f	4a	4b
14	c2+	!1e	!1g	!1c	!2e	!2g	!2c	!3e	!3g	!3c	!4e	!4g	!4c
	c2-	1e	1g	1c	2e	2g	2c	3e	3g	3c	4e	4g	4c
15	c3+	!DP1	!1d	!K	!DP2	!2d	!-	!DP3	!3d	!LOBAT	!DP4	!4d	!CONT
	c3-	DP1	1d	K	DP2	2d	-	DP3	3d	LOBAT	DP4	4d	CONT

Note: This table is initialized with all segments "off".

4.5 SOFTWARE - FUNCTIONS

4.5.1 Refresh Line

In order to display a value on the LCD we must implement a function that will modify the LCD_Lines[] variable. This function will use LCD_Data[] constants as a template to set the wanted segments of the wanted digit.

The special symbols such as DPx, LOBAT or CONT will be managed through another function described later in this document.

We will call this function Refresh_Line. This function will have 2 arguments. The first argument will be used as an index to the LCD_Data[] constant table (Digits template). The second argument corresponds to the digit to be displayed.

In order to simplify the code, the LCD_Data[] table has been rearranged in order to fit the LCD display physical layout.

The two tables below gives a representation of the Refresh_Line function behavior.

LCD_Data[]

	c	b	d	g	a	e	f	hex
0	1	1	1	0	1	1	1	0x77
1	1	1	0	0	0	0	0	0x03
2	0	1	1	1	1	1	0	0x5D
3	1	1	1	1	1	0	0	0x1F
4	1	1	0	1	0	0	1	0x2B
5	1	0	1	1	1	0	1	0x3E
6	1	0	1	1	1	1	1	0x7E
7	1	1	0	0	1	0	0	0x07
8	1	1	1	1	1	1	1	0x7F
9	1	1	0	1	1	0	1	0x2F

7
6
5
4
3
2
1

function call : Refresh_Line (9, 4)

- ① The function takes the number 9, segment f value in the table LCD_Data[]
- ② This value is inverted and placed at the fourth digit to enable the f segment (c1+ !4F)
- ③ Then the non inverted value is placed at the fourth digit to enable the f segment (c1- 4F)
4. Points 1 to 3 are repeated for all the segments of the fourth digit

LCD_Lines[]

	12	11	10	9	8	7	6	5	4	3	2	1
	PB3	PB2	PB1	PB0	PC7	PC6	PC5	PC4	PC3	PC2	PC1	PC0
c1+	0!4b	0!4a	0!4f	!3b	!3a	!3f	!2b	!2a	!2f	!1b	!1a	!1f
c1-	14b	14a	14f	3b	3a	3f	2b	2a	2f	1b	1a	1f
c2+	1!4c	0!4c	0!4e	!3c	!3g	!3e	!2c	!2g	!2e	!1c	!1g	!1e
c2-	04c	14c	14e	3c	3g	3e	2c	2g	2e	1c	1g	1e
c3+	!CONT	!4d	!DP4	!LOBAT	!3d	!DP3	!-	!2d	!DP2	!K	!1d	!DP1
c3-	CONT	04d	DP4	LOBAT	3d	DP3	-	2d	DP2	K	1d	DP1

4.5.2 Set Special

As special symbols can be considered as flags on the LCD display, this function allows all of them to be set individually.

We will call this function SetSpecial. This function will have 2 arguments. The first argument will be a predefined value corresponding to the symbol to be displayed (DPx, LOBAT...) . The second argument corresponds to the state of the symbol (0 = OFF, 1 = ON)

The table below gives a representation of the SetSpecial function behavior.

Function call: SetSpecial(DP1, 1)

- ① Place a 0 at the symbol DP1 (c3+ !DP)
- ② Place a 1 at the symbol DP1 (c3- DP)

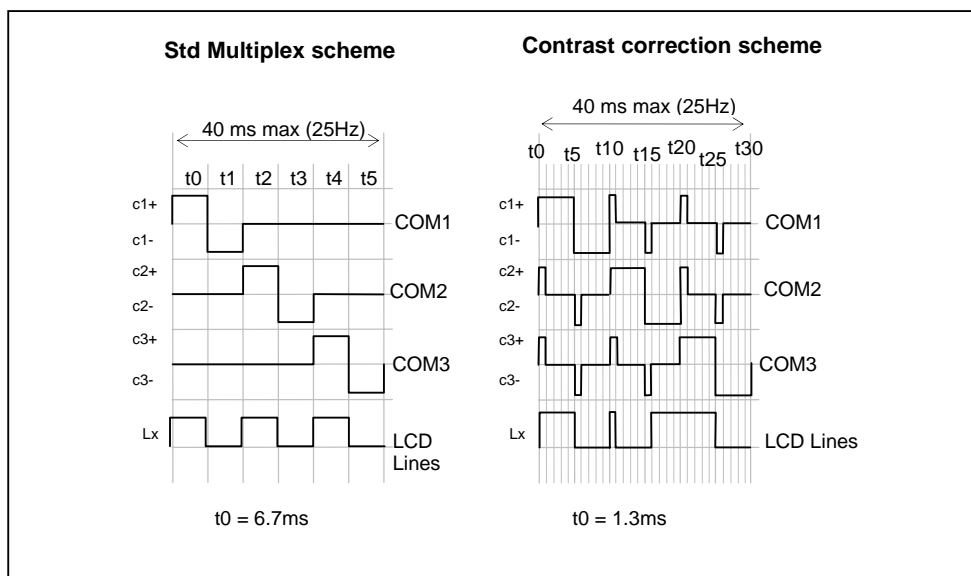
LCD_Lines[]

	12	11	10	9	8	7	6	5	4	3	2	1
	PB3	PB2	PB1	PB0	PC7	PC6	PC5	PC4	PC3	PC2	PC1	PC0
c1+	!4b	!4a	!4f	!3b	!3a	!3f	!2b	!2a	!2f	!1b	!1a	!1f
c1-	4b	4a	4f	3b	3a	3f	2b	2a	2f	1b	1a	1f
c2+	!4c	!4g	!4e	!3c	!3g	!3e	!2c	!2g	!2e	!1c	!1g	!1e
c2-	4c	4g	4e	3c	3g	3e	2c	2g	2e	1c	1g	1e
c3+	!CONT	!4d	!DP4	!LOBAT	!3d	!DP3	!-	!2d	!DP2	!K	!1d	0 DP1 ①
c3-	CONT	4d	DP4	LOBAT	3d	DP3	-	2d	DP2	K	1d	1 DP1 ②

4.5.3 LCD Refresh

In order to avoid the LCD display blinking the index of the LCD_Lines[x] array must be changed at least 150 times per second. This means a complete display refresh every 25 times per second (25Hz) which is the physical limit for the eye to not see the refresh.

Moreover, to implement the contrast correction as described in the chapter "Mixing multiplex and direct drive" we need to have a faster refresh rate, see the timing diagrams below.

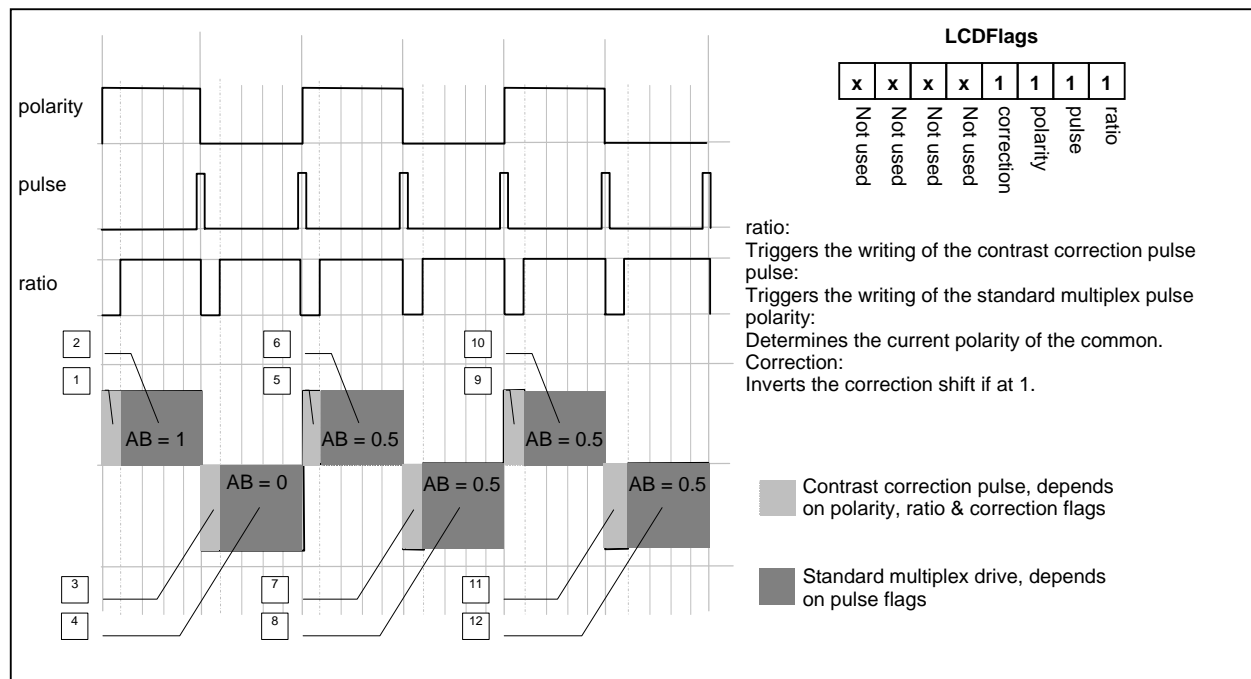


In this example we choose to have 6 steps to correct the contrasts, this implies a higher refresh frequency.

The function will be called LCDRefresh and will have to be called at least every 1.3 ms. This function does not need any arguments.

This function uses a variable that contains a set of flags, these flags will be used to generate the correct pulse widths for the contrast correction and the data display.

The diagram below shows the timings generated through these flags.



The table below shows the different states of the flags

state	rat	pul	pol	cor	AB =
0	0	0	1	0	
1	0	0	1	0	1
2	1	0	1	0	LCD_COM[0]
3	0	1/0	0	0	0
4	1	0	0	0	LCD_COM[1]
5	0	1/0	1	0	1
6	1	0	1	0	LCD_COM[2]
7	0	1/0	0	0	0
8	1	0	0	0	LCD_COM[3]
9	0	1/0	1	0	1
10	1	0	1	0	LCD_COM[4]
11	0	1/0	0	0	0
12	1	0	0	0	LCD_COM[5]

5 ANNEX A – APPLICATION SOURCE CODE

This LCD driver can be found annexed in a zip file, some other functions have been added in order to simplify the use of the LCD display.

Functions added:

void Number2LCD (_U16 Value);	This function takes a number from 0 to 9999 and displays it on the LCD
void InitLCD(void);	This function inits the hardware (counters ports etc..)
void AllChar(_U8 State);	This function clears/displays all the segments of the LCD
void LCDOff(void);	This function clears the LCD and disables all the peripherals

In the zip file you will find two projects, one is the standard template including the LCD driver below; the other is a basic clock application.

Below is the main file of the standard template showing the minimum code to write to see something on the LCD.

5.1 MAIN FILE

```

/*****
** File      : main.c
**
**
** Version   :
**
** Written by :
**
** Date      : XX-XX-XXXX
**
** Project   : -
**
*****/
** Changes   :
*****/
** Description : Main program
*****/
#include "Globals.h"
/*****
** Global variables declaration
*****/
_U8 Toggle = TRUE; // Global variable changed in irq 128Hz
/*****
** main : Main program function
*****/
** In      : -
** Out     : -
*****/
int main (void){
    _U16 i;

    InitMicro();           // Initializes the microcontroller (RC @ 1MHz)
    Monitor_Init();
    Monitor_SoftBreak;
    InitLCD();              // Initializes LCD

    RegIrqEnHig |= 0x40;    // Enables the 128Hz interrupt

    while(1){
        //displays numbers from 0 to 9999
        for (i = 0 ; 9999 > i; i++){
            Number2LCD (i); // display the current loop value

            do{
                asm("halt");// waits for the 1Hz interrupt
            }while(!Toggle);
            Toggle = FALSE;
        }
    }

    return 0;
}

void Handle_Irq_128Hz (void){
    Toggle = TRUE;
}

```

5.2 LCD DRIVER C FILE

```

/*****
** File      : 0x_LcdDriver.c
**
**
** Version   : V 1.0
**
** Written by : Gregoire Guye
**
** Date      : 21-02-2005
**
** Project   : -
**
**
** Changes   :
**
** Description : 0x LCD driver
**
**
***/
#include "0x_LcdDriver.h"

/*****
** Global variables declaration
**
***/
// Common Scheme
_U8 LCD_Com[] = {CA1, CA0, AB1, AB0, BC1, BC0};

// Display Scheme initialized at LCD off (all 0)
_U16 LCD_Lines[] = {0xFFFF, 0x0000, 0xFFFF, 0x0000, 0xFFFF, 0x0000 }; // Ports image

// Numbers from 0 to null predefined
// 0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17
// data (0,1,2,3,4,5,6,7,8,9,- , ,c,l,e,t,b,n)
_U8 LCD_Data[] = {0x77,0x60,0x3E,0x7C,0x69,0x5D,0x5F,0x64,0x7F,0x6D,0x08,0x00,0x17,
0x13,0x1F,0x1B,0x5B,0x4A};

volatile _U8 CountInt,CountIntStep = 1;
_U8 LCDFlags = 0x08;
_U8 CountIntTick;
_U8 CurrentLine = 0x00;
```



Application Note AN8000.13

Driving LCD Displays with I/Os

```

*****
** Refresh_Line : Transposes a number in a digit **
*****
** This routines transposes the values stored in the LCD Datable to the **
** table used to display values on LCD , in this case a 6 rows x 12 col **
** disposed as follow : **
**
**
** LCD_Lines[] **
**
** /-----\ **
** |PB3 . PB2 |PB1 . PB0 |PC7 . PC6 |PC5 . PC4 |PC3 . PC2 |PC1 . PC0 | **
** |!4B . !4A |!4F . !3B |!3A . !3F |!2B . !2A |!2F . !1B |!1A . !1F | **
** c1- |4B . 4A |4F . 3B |3A . 3F |2B . 2A |2F . 1B |1A . 1F | **
** c2+ |!4C . !4G |!4E . !3C |!3G . !3E |!2C . !2G |!2E . !1C |!1G . !1E | **
** c2- |4C . 4G |4E . 3C |3G . 3E |2C . 2G |2E . 1C |1G . 1E | **
** c3+ |CONT. !4D |!DP4. !LOBAT |!3D . !DP3 |!- . !2D |!DP2 . !K |!1D . !DP1 | **
** c3- |CONT. 4D |DP4. LOBAT |3D . DP3 | - . 2D |DP2 . K |1D . DP1 | **
** +-----+-----+-----+-----+-----+-----+ **
**
*****
** In : number (0-null), digit_no (0-3) **
** Out : - (LCD_lines modified) **
*****/
void Refresh_Line ( _U8 number, _U8 digit_no ){
    _U8 i, Lines_index, phase;
    _U16 mask, temp ,curr_num;

    Lines_index = 0; // initializes the line index
    phase = 0; // with phase = 0 first "1" on commons
    digit_no = (3 - digit_no);
    digit_no *= 3; // Transforms digit_no to column index (3 multiple)
    mask = 0x01; // Initialises the mask
    mask <= digit_no; // Place it to the right column
    curr_num = LCD_Data[number]; // Copy the Sline corresponding number
    for (i = 0; i < 14; i++){
        phase ^= 0x01; // Toggles the phase
        if(phase){ // Write the current val & its inverse successively
            temp = (~curr_num & 0x01); // Mask the current val (inverse) and save it in temp
        }else{
            temp = (curr_num & 0x01); // Mask the current val and save it in temp
            curr_num >= 1; // Seek the next current val to write
        }
        temp <= digit_no; // Places the value to its correct digit place
        LCD_Lines[Lines_index] &= ~mask; // Clears the previous value then write the new value
        LCD_Lines[Lines_index] |= (temp & mask);
        Lines_index++; // Writes next line
        if (i == 3){ // If the number of lines max -1 is reached
            mask <= 1; // Moves the mask by 1
            digit_no++; // Changes the line to write
            Lines_index = 0; // Re-initializes the line index
        }else if (i == 9){
            mask <= 1; // Moves the mask by 1
            digit_no++; // Changes the line to write
            Lines_index = 0; // Re-initializes the line index
        }
    }
}
}

```

```

/*****
** Display_LCD : Displays the current value of LCD_line table **
*****/
** In : - **
** Out : - **
*****/
void Display_LCD (void){
    _U8 IntPBVal, IntPCVal;

    if (CountIntTick == CountIntStep){
        LCDFlags |= 0x01; // Sets Ratio Flag to 1
    }
    else if (CountIntTick == 6){
        LCDFlags |= 0x03; // Sets Pulse &Ratio flags to 1
        CountIntTick = 0;
    }
    if(CountIntTick == 0 && ((LCDFlags & 0x03) == 0x03)){
    }else{
        CountIntTick++;
    }

    // Standard MUX3 drive section
    if((LCDFlags & 0x01) == 0x01){
        if((LCDFlags & 0x02) == 0x02){
            LCDFlags ^= 0x04; // toggles the polarity flag
            LCDFlags &= ~0x03; // Clears the Ratio & Pulses flags

            CurrentLine++;
        }else{
            // Save the current ports values
            IntPBVal = RegPBOut;
            IntPCVal = RegPCOut;

            // Clears the correct pins according to predefined masks
            IntPBVal &= ~(COMMON_MASK3 | LINE_MASK2);
            IntPCVal &= ~LINE_MASK1;

            // Applies the common pattern
            IntPBVal |= (LCD_Com[CurrentLine] & COMMON_MASK3);

            // Applies the data patterns
            IntPCVal |= (LCD_Lines[CurrentLine] & LINE_MASK1);
            IntPBVal |= ((LCD_Lines[CurrentLine] >> 8) & LINE_MASK2);

            // Update the display
            RegPBOut = IntPBVal;
            RegPCOut = IntPCVal;
        }
    }
    // End of Standard MUX3 drive section
    // Contrast adjustment section
    else{
        // Force the Commons to 0 or 1 during the period defined by the ratio
        if((LCDFlags & 0x04) == 0x04){
            RegPBOut |= COMMON_MASK3; // Forces Commons to 1
        }else{
            RegPBOut &= ~COMMON_MASK3; // Forces Commons to 0
        }

        // Force the Data to 0 or 1 during the period defined by the ratio
        if((LCDFlags & 0x04) == ((LCDFlags & 0x08)>> 1)){
            RegPCOut |= LINE_MASK1; // Forces Datas to 1 (PC0 - PC7)
            RegPBOut |= LINE_MASK2; // Forces Data to 1 (PB0 - PB3)
        }else{
            RegPCOut &= ~LINE_MASK1; // Forces Datas to 0 (PC0 - PC7)
            RegPBOut &= ~LINE_MASK2; // Forces Data to 0 (PB0 - PB3)
        }
    }
    // End of Contrast adjustment section
    if(CurrentLine == 6){ // End of the table = 6
        CurrentLine = 0;
    }
}

```

```

/*****
** SetSpecial : Sets the Special characters on LCD_Lines **
*****/
** In          : Special char to modify DP1, K, DP2, MINUS, **
**              DP3, LOBAT, DP4, CONT **
**              State 0 / 1 **
** Out         : - **
*****/
void SetSpecial(_U8 Schar, _U8 State){
    _U16 mask = 0x01;
    mask <<= (Schar);          // Shift the mask at the correct column
    if (State){
        LCD_Lines[4] &= ~mask;  // Unsets the corresponding bit
        LCD_Lines[5] |= mask;   // Set the corresponding bit (inverse)
    }else{
        LCD_Lines[4] |= mask;    // Sets the corresponding bit
        LCD_Lines[5] &= ~mask;   // Unsets the corresponding bit (inverse)
    }
}

/*****
** AllChar : Sets all the Characters on LCD_Lines **
*****/
** In          : State 0 / 1 **
** Out         : - **
*****/
void AllChar(_U8 State){
    if (State){
        LCD_Lines[0] &= ~0x7FFF;
        LCD_Lines[1] |= 0x7FFF;
        LCD_Lines[2] &= ~0x7FFF;
        LCD_Lines[3] |= 0x7FFF;
        LCD_Lines[4] &= ~0x7FFF;
        LCD_Lines[5] |= 0x7FFF;
    }
    else{
        LCD_Lines[0] |= 0x7FFF;
        LCD_Lines[1] &= ~0x7FFF;
        LCD_Lines[2] |= 0x7FFF;
        LCD_Lines[3] &= ~0x7FFF;
        LCD_Lines[4] |= 0x7FFF;
        LCD_Lines[5] &= ~0x7FFF;
    }
}

/*****
** Number2LCD : Transposes a 3 digits number on the LCD display **
*****/
** In : Value (0-9999) **
** Out : - **
*****/
void Number2LCD (_U16 Value){
    Refresh_SLine ((Value % 10), 0);
    Refresh_SLine (((Value / 10) % 10), 1);
    Refresh_SLine (((Value / 100) % 10), 2);
    Refresh_SLine (((Value / 1000) % 10), 3);
}

/*****
** InitLCD : Initialises the counters for the LCD **
*****/
** In          : - **
** Out         : - **
*****/
void InitLCD(void){
    RegPCOut      = 0x00;          // Set PC out at 0
    RegPCDir      = 0xFF;          // Sets PortC pins as outputs
    RegPBDIR      = 0xFF;          // Sets PortB pins as outputs
    RegPBOpen     = 0x00;          // Port B hasn't open drain outputs
    RegPBPullup   = 0xFF;          // No Pullups
    RegCntCtrlCk  |= 0x10;         // ck for counter C = 32KHz
    RegCntConfig1 &= ~0x40;        // Counter C down counting
    RegCntC       = 0x18;          // 1.3kHz
    RegIrqEnHig   |= 0x08;         // Authorize interrupt Counter C (240Hz)
    RegCntOn      |= 0x04;         // Starts the counter C
}

```

```

/*****
** LCDOff : Switches off the LCD Display
**
** In      : -
** Out     : -
**
*****/
void LCDOff(void){
    AllChar(0);           // Clears the LCD display
    RegPCOut   = 0x00;    // Clears PortC
    RegPBOut   &= ~0xEF;  // Clears PortB
    RegIrqEnHig &= ~0x08; // Disable interrupt Counter C (240Hz)
    RegCntOn   &= ~0x04; // Stops the counter C
}

/*****
** Interrupts Handling
**
*****/

/*****
** Handle_Irq_CntC : Handles the interruption Counter C that
**                   clocks the LCD display in order to have a
**                   total frame of 32Hz
**
** In      : -
** Out     : -
**
*****/
void Handle_Irq_CntC (void){

    Display_LCD ();
} //End Handle_Irq_CntC

```

5.3 LCD DRIVER HEADER FILE

```

/*****
** File      : 0x_LcdDriver.h
**
** Version   : V 1.0
**
** Written by : Gregoire Guye
**
** Date      : 21-02-2005
**
** Project   : -
**
** Changes   :
** Description : 0x LCD driver
*****/
#ifndef __O_X_LCD_DRIVER__
#define __O_X_LCD_DRIVER__

/*****
** Include files
*****/
#include "Globals.h"

/*****
** Global definitions
*****/
// Commons waveforms
#define AB1 0x60 // +----- F ----+
#define AB0 0x80 // |
#define BC1 0xC0 // |
#define BC0 0x20 // |
#define CA1 0xA0 // |
#define CA0 0x40 // |
// Digits

#define D1 3 // Digit1 Xxxx
#define D2 2 // Digit2 xxXx
#define D3 1 // Digit3 xxXx
#define D4 0 // Digit4 xxxX

// Special Chars

#define DP1 0 // Decimal point 1
#define K 2 // "I" Icon
#define DP2 3 // Decimal point 2
#define MINUS 5 // "-" Icon
#define DP3 6 // Decimal point 3
#define LOBAT 8 // low bat Icon
#define DP4 9 // Decimal point 4
#define CONT 11 // Continuity Icon

// Masks

#define COMMON_MASK3 0xE0 // (PB5 to PB7) Mux 3

#define LINE_MASK1 0xFF // (PC0 to PC7)
#define LINE_MASK2 0x0F // (PB0 to PB3)

// Special

#define _MINUS 10
#define _NULL 11
#define _C 12
#define _L 13
#define _E 14
#define _T 15
#define _B 16
#define _N 17

```

© Semtech 2006

All rights reserved. Reproduction in whole or in part is prohibited without the prior written consent of the copyright owner. The information presented in this document does not form part of any quotation or contract, is believed to be accurate and reliable and may be changed without notice. No liability will be accepted by the publisher for any consequence of its use. Publication thereof does not convey nor imply any license under patent or other industrial or intellectual property rights. Semtech. assumes no responsibility or liability whatsoever for any failure or unexpected operation resulting from misuse, neglect improper installation, repair or improper handling or unusual physical or electrical stress including, but not limited to, exposure to parameters beyond the specified maximum ratings or operation outside the specified range.

SEMTECH PRODUCTS ARE NOT DESIGNED, INTENDED, AUTHORIZED OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT APPLICATIONS, DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS. INCLUSION OF SEMTECH PRODUCTS IN SUCH APPLICATIONS IS UNDERSTOOD TO BE UNDERTAKEN SOLELY AT THE CUSTOMER'S OWN RISK. Should a customer purchase or use Semtech products for any such unauthorized application, the customer shall indemnify and hold Semtech and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs damages and attorney fees which could arise.

Contact Information

Semtech Corporation
Wireless and Sensing Products Division
200 Flynn Road, Camarillo, CA 93012
Phone (805) 498-2111 Fax : (805) 498-3804