

# Nokia LCD Library

## History

I found this LCD make for cellular phone Nokia 3310 by browsing the web at [www.jelu.se](http://www.jelu.se). There is many interesting things about this LCD: low price, easy to interface via SPI port, can support graphic and text and consume low current. The only thing that I did is writing code who gives you friendly functions to support this LCD. The last version of the library only send the bytes who had changed since the last update. It's considerably reduce the refresh time who can be very important in a batteries powered system.

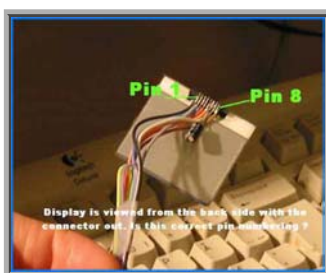
## Functions Features

- LcdInit(void)
- LcdClear(void)
- LcdUpdate(void)
- LcdGotoXY(byte x, byte y)
- LcdChr(LcdFontSize size, byte ch)
- LcdStr(LcdFontSize size, byte \*dataPtr)
- LcdPixel(byte x, byte y, LcdPixelMode mode)
- LcdLine(byte x1, byte y1, byte x2, byte y2)

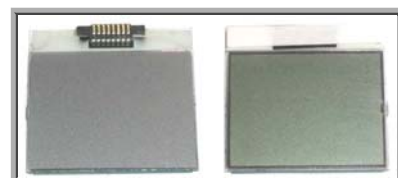
## Pictures

[Click to enlarge](#)

Pinout



Front/Back of the display



## Sources codes & Schematics

### -Schematic in GIF format

```
/*-----
```

```
Name      : NokiaLCD.h
```

```
Description : Header file for Nokia 84x48 graphic LCD driver.
```

```
Author     : 2003-03-08 - Louis Frigon.
```

```
History    : 2003-03-08 - First release.
```

```
-----*/
```

```
#ifndef _NOKIALCD_H_
```

```
#define _NOKIALCD_H_
```

```
/*-----
```

```
                General purpose constants
```

```
-----*/
```

```
#define NULL      0
```

```
#define FALSE     0
```

```
#define TRUE      1
```

```
#define LCD_X_RES    84
```

```
#define LCD_Y_RES    48
```

```
// Mega8 port B pinout for LCD.
```

```
#define LCD_DC_PIN    0x01 // PB0
```

```
#define LCD_CE_PIN    0x04 // PB2
```

```
#define SPI_MOSI_PIN  0x08 // PB3
```

```
#define LCD_RST_PIN   0x10 // PB4
```

```
#define SPI_CLK_PIN   0x20 // PB5
```

```
#define LCD_CACHE_SIZE      ((LCD_X_RES * LCD_Y_RES) / 8)
```

```
/*-----
```

*Type definitions*

```
-----*/
```

```
typedef char          bool;
```

```
typedef unsigned char    byte;
```

```
typedef unsigned int     word;
```

```
typedef enum
```

```
{
```

```
    LCD_CMD = 0,
```

```
    LCD_DATA = 1
```

```
} LcdCmdData;
```

```
typedef enum
```

```
{
```

```
    PIXEL_OFF = 0,
```

```
    PIXEL_ON = 1,
```

```
    PIXEL_XOR = 2
```

```
} LcdPixelMode;
```

```
typedef enum
```

```
{
```

```
    FONT_1X = 1,
```

```
    FONT_2X = 2
```

```
} LcdFontSize;
```

```
/*-----
```

*Public function prototypes*

```
-----*/
```

```
void LcdInit    ( void );
```

```
void LcdClear   ( void );
```

```
void LcdUpdate  ( void );
```

```
void LcdGotoXY  ( byte x, byte y );
```

```
void LcdChr    ( LcdFontSize size, byte ch );
void LcdStr    ( LcdFontSize size, byte *dataPtr );
void LcdPixel  ( byte x, byte y, LcdPixelMode mode );
void LcdLine   ( byte x1, byte y1, byte x2, byte y2, LcdPixelMode mode );

#endif // _NOKIALCD_H_

/*-----
                        End of file.
-----*/

/*-----

Name      : NokiaLCD.c

Description : This is a driver for the Nokia 84x48 graphic LCD.

Author     : 2003-03-08 - Sylvain Bissonnette

History    : 2003-02-08 - First release (v0.1) derived from Sylvain Bissonnette code base.
              2003-03-09 - v0.2, Louis Frigon: 2x fonts support.
              2003-03-20 - v0.3: Serialization optimized,
-----*/

#include <macros.h>
#include <iom8v.h>
#ifndef <stdio.h>

#include "NokiaLCD.h"

#define LCD_FIRMWARE_VERSION    0.3

/*-----

                        Private function prototypes
-----*/

// Function prototypes are mandatory otherwise the compiler generates unreliable code.

static void LcdSend ( byte data, LcdCmdData cd );
```

```
static void Delay ( void );
```

```
/*-----
```

*Character generator*

*This table defines the standard ASCII characters in a 5x7 dot format.*

```
-----*/
```

```
static const byte FontLookup [][][5] =
{
  { 0x00, 0x00, 0x00, 0x00, 0x00 }, // sp
  { 0x00, 0x00, 0x2f, 0x00, 0x00 }, // !
  { 0x00, 0x07, 0x00, 0x07, 0x00 }, // "
  { 0x14, 0x7f, 0x14, 0x7f, 0x14 }, // #
  { 0x24, 0x2a, 0x7f, 0x2a, 0x12 }, // $
  { 0xc4, 0xc8, 0x10, 0x26, 0x46 }, // %
  { 0x36, 0x49, 0x55, 0x22, 0x50 }, // &
  { 0x00, 0x05, 0x03, 0x00, 0x00 }, // '
  { 0x00, 0x1c, 0x22, 0x41, 0x00 }, // (
  { 0x00, 0x41, 0x22, 0x1c, 0x00 }, // )
  { 0x14, 0x08, 0x3E, 0x08, 0x14 }, // *
  { 0x08, 0x08, 0x3E, 0x08, 0x08 }, // +
  { 0x00, 0x00, 0x50, 0x30, 0x00 }, // ,
  { 0x10, 0x10, 0x10, 0x10, 0x10 }, // -
  { 0x00, 0x60, 0x60, 0x00, 0x00 }, // .
  { 0x20, 0x10, 0x08, 0x04, 0x02 }, // /
  { 0x3E, 0x51, 0x49, 0x45, 0x3E }, // 0
  { 0x00, 0x42, 0x7F, 0x40, 0x00 }, // 1
  { 0x42, 0x61, 0x51, 0x49, 0x46 }, // 2
  { 0x21, 0x41, 0x45, 0x4B, 0x31 }, // 3
  { 0x18, 0x14, 0x12, 0x7F, 0x10 }, // 4
  { 0x27, 0x45, 0x45, 0x45, 0x39 }, // 5
  { 0x3C, 0x4A, 0x49, 0x49, 0x30 }, // 6
  { 0x01, 0x71, 0x09, 0x05, 0x03 }, // 7
  { 0x36, 0x49, 0x49, 0x49, 0x36 }, // 8
  { 0x06, 0x49, 0x49, 0x29, 0x1E }, // 9
  { 0x00, 0x36, 0x36, 0x00, 0x00 }, // :
  { 0x00, 0x56, 0x36, 0x00, 0x00 }, // ;
  { 0x08, 0x14, 0x22, 0x41, 0x00 }, // <
  { 0x14, 0x14, 0x14, 0x14, 0x14 }, // =
```

```
{ 0x00, 0x41, 0x22, 0x14, 0x08 }, // >
{ 0x02, 0x01, 0x51, 0x09, 0x06 }, // ?
{ 0x32, 0x49, 0x59, 0x51, 0x3E }, // @
{ 0x7E, 0x11, 0x11, 0x11, 0x7E }, // A
{ 0x7F, 0x49, 0x49, 0x49, 0x36 }, // B
{ 0x3E, 0x41, 0x41, 0x41, 0x22 }, // C
{ 0x7F, 0x41, 0x41, 0x22, 0x1C }, // D
{ 0x7F, 0x49, 0x49, 0x49, 0x41 }, // E
{ 0x7F, 0x09, 0x09, 0x09, 0x01 }, // F
{ 0x3E, 0x41, 0x49, 0x49, 0x7A }, // G
{ 0x7F, 0x08, 0x08, 0x08, 0x7F }, // H
{ 0x00, 0x41, 0x7F, 0x41, 0x00 }, // I
{ 0x20, 0x40, 0x41, 0x3F, 0x01 }, // J
{ 0x7F, 0x08, 0x14, 0x22, 0x41 }, // K
{ 0x7F, 0x40, 0x40, 0x40, 0x40 }, // L
{ 0x7F, 0x02, 0x0C, 0x02, 0x7F }, // M
{ 0x7F, 0x04, 0x08, 0x10, 0x7F }, // N
{ 0x3E, 0x41, 0x41, 0x41, 0x3E }, // O
{ 0x7F, 0x09, 0x09, 0x09, 0x06 }, // P
{ 0x3E, 0x41, 0x51, 0x21, 0x5E }, // Q
{ 0x7F, 0x09, 0x19, 0x29, 0x46 }, // R
{ 0x46, 0x49, 0x49, 0x49, 0x31 }, // S
{ 0x01, 0x01, 0x7F, 0x01, 0x01 }, // T
{ 0x3F, 0x40, 0x40, 0x40, 0x3F }, // U
{ 0x1F, 0x20, 0x40, 0x20, 0x1F }, // V
{ 0x3F, 0x40, 0x38, 0x40, 0x3F }, // W
{ 0x63, 0x14, 0x08, 0x14, 0x63 }, // X
{ 0x07, 0x08, 0x70, 0x08, 0x07 }, // Y
{ 0x61, 0x51, 0x49, 0x45, 0x43 }, // Z
{ 0x00, 0x7F, 0x41, 0x41, 0x00 }, // [
{ 0x55, 0x2A, 0x55, 0x2A, 0x55 }, // 55
{ 0x00, 0x41, 0x41, 0x7F, 0x00 }, // ]
{ 0x04, 0x02, 0x01, 0x02, 0x04 }, // ^
{ 0x40, 0x40, 0x40, 0x40, 0x40 }, // _
{ 0x00, 0x01, 0x02, 0x04, 0x00 }, // '
{ 0x20, 0x54, 0x54, 0x54, 0x78 }, // a
{ 0x7F, 0x48, 0x44, 0x44, 0x38 }, // b
{ 0x38, 0x44, 0x44, 0x44, 0x20 }, // c
{ 0x38, 0x44, 0x44, 0x48, 0x7F }, // d
```

```

{ 0x38, 0x54, 0x54, 0x54, 0x18 }, //e
{ 0x08, 0x7E, 0x09, 0x01, 0x02 }, //f
{ 0x0C, 0x52, 0x52, 0x52, 0x3E }, //g
{ 0x7F, 0x08, 0x04, 0x04, 0x78 }, //h
{ 0x00, 0x44, 0x7D, 0x40, 0x00 }, //i
{ 0x20, 0x40, 0x44, 0x3D, 0x00 }, //j
{ 0x7F, 0x10, 0x28, 0x44, 0x00 }, //k
{ 0x00, 0x41, 0x7F, 0x40, 0x00 }, //l
{ 0x7C, 0x04, 0x18, 0x04, 0x78 }, //m
{ 0x7C, 0x08, 0x04, 0x04, 0x78 }, //n
{ 0x38, 0x44, 0x44, 0x44, 0x38 }, //o
{ 0x7C, 0x14, 0x14, 0x14, 0x08 }, //p
{ 0x08, 0x14, 0x14, 0x18, 0x7C }, //q
{ 0x7C, 0x08, 0x04, 0x04, 0x08 }, //r
{ 0x48, 0x54, 0x54, 0x54, 0x20 }, //s
{ 0x04, 0x3F, 0x44, 0x40, 0x20 }, //t
{ 0x3C, 0x40, 0x40, 0x20, 0x7C }, //u
{ 0x1C, 0x20, 0x40, 0x20, 0x1C }, //v
{ 0x3C, 0x40, 0x30, 0x40, 0x3C }, //w
{ 0x44, 0x28, 0x10, 0x28, 0x44 }, //x
{ 0x0C, 0x50, 0x50, 0x50, 0x3C }, //y
{ 0x44, 0x64, 0x54, 0x4C, 0x44 } //z
};

```

```

/*-----

```

*Global Variables*

```

-----*/

```

```

static byte LcdCache [ LCD_CACHE_SIZE ];

```

```

static int LcdCacheIdx;

```

```

static int LoWaterMark;

```

```

static int HiWaterMark;

```

```

static bool UpdateLcd;

```

```

/*-----

```

```

Name      : LcdInit

```

*Description : Performs MCU SPI & LCD controller initialization.*

*Argument(s) : None.*

*Return value : None.*

```
-----*/  
void LcdInit ( void )  
{  
    // Pull-up on reset pin.  
    PORTB |= LCD_RST_PIN;  
  
    // Set output bits on port B.  
    DDRB |= LCD_RST_PIN | LCD_DC_PIN | LCD_CE_PIN | SPI_MOSI_PIN | SPI_CLK_PIN;  
  
    Delay();  
  
    // Toggle display reset pin.  
    PORTB &= ~LCD_RST_PIN;  
    Delay();  
    PORTB |= LCD_RST_PIN;  
  
    // Enable SPI port: No interrupt, MSBit first, Master mode, CPOL->0, CPHA->0, Clk/4  
    SPCR = 0x50;  
  
    // Disable LCD controller  
    PORTB |= LCD_CE_PIN;  
  
    LcdSend(0x21, LCD_CMD); // LCD Extended Commands.  
    LcdSend(0xC8, LCD_CMD); // Set LCD Vop (Contrast).  
    LcdSend(0x06, LCD_CMD); // Set Temp coefficient.  
    LcdSend(0x13, LCD_CMD); // LCD bias mode 1:48.  
    LcdSend(0x20, LCD_CMD); // LCD Standard Commands, Horizontal addressing mode.  
    LcdSend(0x0C, LCD_CMD); // LCD in normal mode.  
  
    // Reset watermark pointers.  
    LoWaterMark = LCD_CACHE_SIZE;  
    HiWaterMark = 0;
```



```
LcdClear();  
LcdUpdate();  
}
```

```
/*-----
```

*Name* : *LcdContrast*

*Description* : *Set display contrast.*

*Argument(s)* : *contrast -> Contrast value from 0x00 to 0x7F.*

*Return value* : *None.*

*Notes* : *No change visible at ambient temperature.*

```
-----*/
```

```
void LcdContrast ( byte contrast )  
{  
    // LCD Extended Commands.  
    LcdSend( 0x21, LCD_CMD );  
  
    // Set LCD Vop (Contrast).  
    LcdSend( 0x80 | contrast, LCD_CMD );  
  
    // LCD Standard Commands, horizontal addressing mode.  
    LcdSend( 0x20, LCD_CMD );  
}
```

```
/*-----
```

*Name* : *LcdClear*

*Description* : *Clears the display. LcdUpdate must be called next.*

*Argument(s)* : *None.*

*Return value* : *None.*

```
-----*/  
void LcdClear ( void )  
{  
    int i;  
  
    for ( i = 0; i < LCD_CACHE_SIZE; i++)  
    {  
        LcdCache[i] = 0x00;  
    }  
  
    // Reset watermark pointers.  
    LoWaterMark = 0;  
    HiWaterMark = LCD_CACHE_SIZE - 1;  
  
    UpdateLcd = TRUE;  
}  
  
/*-----
```

*Name* : LcdGotoXY

*Description* : Sets cursor location to xy location corresponding to basic font size.

*Argument(s)* : x, y -> Coordinate for new cursor position. Range: 1,1 .. 14,6

*Return value* : None.

```
-----*/  
void LcdGotoXY ( byte x, byte y )  
{  
    LcdCacheIdx = (x - 1) * 6 + (y - 1) * 84;  
}  
  
/*-----
```

*Name* : LcdChr

*Description* : Displays a character at current cursor location and increment cursor location.

*Argument(s) : size -> Font size. See enum.*

*ch -> Character to write.*

*Return value : None.*

```
-----*/  
void LcdChr ( LcdFontSize size, byte ch )  
{  
    byte i, c;  
    byte b1, b2;  
    int tmpIdx;  
  
    if ( LcdCacheIdx < LoWaterMark )  
    {  
        // Update low marker.  
        LoWaterMark = LcdCacheIdx;  
    }  
  
    if ( (ch < 0x20) || (ch > 0x7b) )  
    {  
        // Convert to a printable character.  
        ch = 92;  
    }  
  
    if ( size == FONT_1X )  
    {  
        for ( i = 0; i < 5; i++ )  
        {  
            LcdCache[LcdCacheIdx++] = FontLookup[ch - 32][i] << 1;  
        }  
    }  
    else if ( size == FONT_2X )  
    {  
        tmpIdx = LcdCacheIdx - 84;  
  
        if ( tmpIdx < LoWaterMark )  
        {  
            LoWaterMark = tmpIdx;  
        }  
    }  
}
```

```
if ( tmpIdx < 0 ) return;

for ( i = 0; i < 5; i++ )
{
    c = FontLookup[ch - 32][i] << 1;
    b1 = (c & 0x01) * 3;
    b1 |= (c & 0x02) * 6;
    b1 |= (c & 0x04) * 12;
    b1 |= (c & 0x08) * 24;

    c >>= 4;
    b2 = (c & 0x01) * 3;
    b2 |= (c & 0x02) * 6;
    b2 |= (c & 0x04) * 12;
    b2 |= (c & 0x08) * 24;

    LcdCache[tmpIdx++] = b1;
    LcdCache[tmpIdx++] = b1;
    LcdCache[tmpIdx + 82] = b2;
    LcdCache[tmpIdx + 83] = b2;
}

// Update x cursor position.
LcdCacheIdx += 11;
}

if ( LcdCacheIdx > HiWaterMark )
{
    // Update high marker.
    HiWaterMark = LcdCacheIdx;
}

// Horizontal gap between characters.
LcdCache[LcdCacheIdx++] = 0x00;
}

/*-----
```

*Name* : *LcdStr*

*Description* : *Displays a character at current cursor location and increment cursor location according to font size.*

*Argument(s)* : *size -> Font size. See enum.*  
*dataPtr -> Pointer to null terminated ASCII string to display.*

*Return value* : *None.*

```

-----*/
void LcdStr ( LcdFontSize size, byte *dataPtr )
{
    while ( *dataPtr )
    {
        LcdChr( size, *dataPtr++ );
    }
}

/*-----

```

*Name* : *LcdPixel*

*Description* : *Displays a pixel at given absolute (x, y) location.*

*Argument(s)* : *x, y -> Absolute pixel coordinates*  
*mode -> Off, On or Xor. See enum.*

*Return value* : *None.*

```

-----*/
void LcdPixel ( byte x, byte y, LcdPixelMode mode )
{
    word index;
    byte offset;
    byte data;

    if ( x > LCD_X_RES ) return;
    if ( y > LCD_Y_RES ) return;

```

```
index = ((y / 8) * 84) + x;
offset = y - ((y / 8) * 8);

data = LcdCache[index];

if ( mode == PIXEL_OFF )
{
    data &= (~(0x01 << offset));
}
else if ( mode == PIXEL_ON )
{
    data |= (0x01 << offset);
}
else if ( mode == PIXEL_XOR )
{
    data ^= (0x01 << offset);
}

LcdCache[index] = data;

if ( index < LoWaterMark )
{
    // Update low marker.
    LoWaterMark = index;
}

if ( index > HiWaterMark )
{
    // Update high marker.
    HiWaterMark = index;
}
}

/*-----
```

*Name* : *LcdLine*

*Description* : *Draws a line between two points on the display.*

*Argument(s) : x1, y1 -> Absolute pixel coordinates for line origin.*

*x2, y2 -> Absolute pixel coordinates for line end.*

*mode -> Off, On or Xor. See enum.*

*Return value : None.*

-----\*/

```
void LcdLine ( byte x1, byte y1, byte x2, byte y2, LcdPixelMode mode )
```

```
{
```

```
    int dx, dy, stepx, stepy, fraction;
```

```
    dy = y2 - y1;
```

```
    dx = x2 - x1;
```

```
    if ( dy < 0 )
```

```
    {
```

```
        dy  = -dy;
```

```
        stepy = -1;
```

```
    }
```

```
    else
```

```
    {
```

```
        stepy = 1;
```

```
    }
```

```
    if ( dx < 0 )
```

```
    {
```

```
        dx  = -dx;
```

```
        stepx = -1;
```

```
    }
```

```
    else
```

```
    {
```

```
        stepx = 1;
```

```
    }
```

```
    dx <<= 1;
```

```
    dy <<= 1;
```

```
    LcdPixel( x1, y1, mode );
```

```
if ( dx > dy )
{
    fraction = dy - (dx >> 1);
    while ( x1 != x2 )
    {
        if ( fraction >= 0 )
        {
            y1 += stepy;
            fraction -= dx;
        }
        x1 += stepx;
        fraction += dy;
        LcdPixel( x1, y1, mode );
    }
}
else
{
    fraction = dx - (dy >> 1);
    while ( y1 != y2 )
    {
        if ( fraction >= 0 )
        {
            x1 += stepx;
            fraction -= dy;
        }
        y1 += stepy;
        fraction += dx;
        LcdPixel( x1, y1, mode );
    }
}

UpdateLcd = TRUE;
}
```

```
/*-----
```

*Name* : *LcdUpdate*



*Description* : Copies the LCD cache into the device RAM.

*Argument(s)* : None.

*Return value* : None.

```
-----*/  
void LcdUpdate ( void )  
{  
    int i;  
  
    if ( LoWaterMark < 0 )  
        LoWaterMark = 0;  
    else if ( LoWaterMark >= LCD_CACHE_SIZE )  
        LoWaterMark = LCD_CACHE_SIZE - 1;  
  
    if ( HiWaterMark < 0 )  
        HiWaterMark = 0;  
    else if ( HiWaterMark >= LCD_CACHE_SIZE )  
        HiWaterMark = LCD_CACHE_SIZE - 1;  
  
    // Set base address according to LoWaterMark.  
    LcdSend( 0x80 | (LoWaterMark % LCD_X_RES), LCD_CMD );  
    LcdSend( 0x40 | (LoWaterMark / LCD_X_RES), LCD_CMD );  
  
    // Serialize the video buffer.  
    for ( i = LoWaterMark; i <= HiWaterMark; i++ )  
    {  
        LcdSend( LcdCache[i], LCD_DATA );  
    }  
  
    // Reset watermark pointers.  
    LoWaterMark = LCD_CACHE_SIZE - 1;  
    HiWaterMark = 0;  
  
    UpdateLcd = FALSE;  
}  
  
/*-----
```

*Name* : *LcdSend*

*Description* : *Sends data to display controller.*

*Argument(s)* : *data -> Data to be sent*

*cd* -> *Command or data (see/use enum)*

*Return value* : *None.*

```

-----*/
static void LcdSend ( byte data, LcdCmdData cd )
{
    // Enable display controller (active low).
    PORTB &= ~LCD_CE_PIN;

    if ( cd == LCD_DATA )
    {
        PORTB |= LCD_DC_PIN;
    }
    else
    {
        PORTB &= ~LCD_DC_PIN;
    }

    // Send data to display controller.
    SPDR = data;

    // Wait until Tx register empty.
    while ( (SPSR & 0x80) != 0x80 );

    // Disable display controller.
    PORTB |= LCD_CE_PIN;
}

/*-----

```

*Name* : *Delay*

*Description : Uncalibrated delay for LCD init routine.*

*Argument(s) : None.*

*Return value : None.*

```
-----*/  
static void Delay ( void )  
{  
    int i;  
  
    for ( i = -32000; i < 32000; i++ );  
}  
  
/*-----  
                End of file.  
-----*/
```