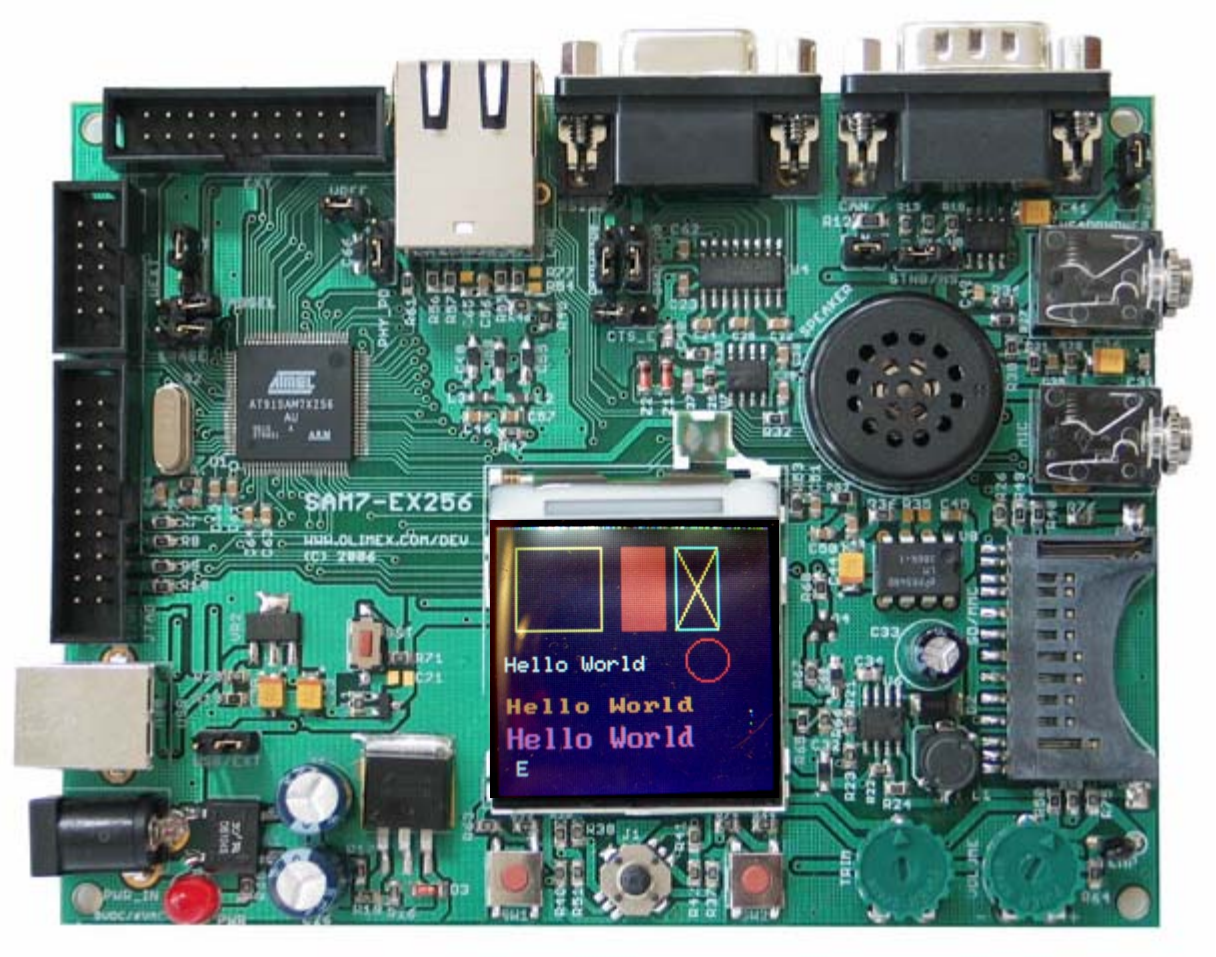
Nokia 6100 LCD Display Driver

Author: **James P. Lynch**



Introduction

There have been countless millions of Nokia cell phones sold world-wide and this economy of scale has made it possible for the hobbyist and experimenter to procure the LCD graphic display from these phones at a reasonable price. Sparkfun Electronics (www.sparkfun.com) sells a model 6100 for $19.95 (US). I've seen sources for this display on EBay for $7.99 (US) plus $10.00 shipping (from Hong Kong, so shipping is a bit slow). The Swedish web shop Jelu (www.jelu.se) has this display for about $20.00 (US) also (see photograph below).

Olimex uses these displays on their more sophisticated development boards, so this tutorial will be geared to the Olimex SAM7-EX256 board shown on the front cover.

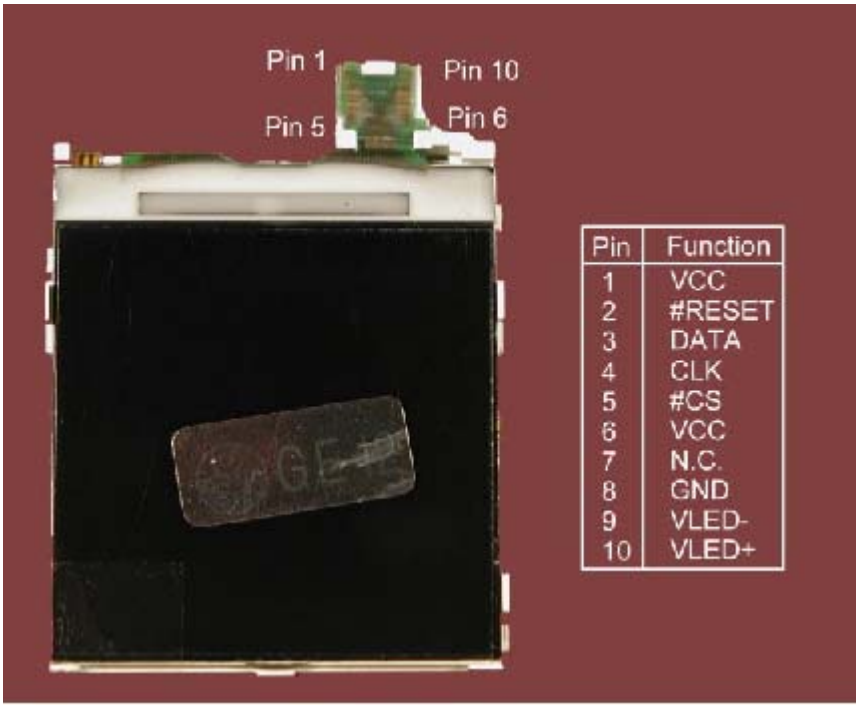


Figure 1. Nokia 6100 LCD Display (from Jelu web site)

The important specifications for this display are as follows:

132 x 132 pixels
12-bit color rendition (4 bits red, 4-bits green, 4-bits blue)
3.3 volts
9-bit SPI serial interface (clock/data signals)

The major irritant in using this display is identifying the graphics controller; there are two possibilities (Epson S1D15G00 or Philips PCF8833). The LCD display sold by the German Web Shop Jelu has a Leadis LDS176 controller but it is 100% compatible with the Philips PCF8833). So how do you tell which controller you have? Some message boards have suggested that the LCD display be disassembled and the controller chip measured with a digital caliper – well that's getting a bit extreme.

Here's what I know. The Olimex boards have both display controllers possible; if the LCD has a GE-12 sticker on it, it's a Philips PCF8833. If it has a GE-8 sticker, it's an Epson controller. The older Sparkfun 6100 displays were Epson, their web site indicates that the newer ones are an Epson clone. Sparkfun software examples sometimes refer to the Philips controller so the whole issue has become a bit murky. The trading companies in Honk Kong have no idea what is inside the displays they are selling. A Nokia 6100 display that I purchased from Hong Kong a couple of weeks ago had the Philips controller.

I was not happy with any of the driver software examples I had inspected; they all seemed to be “mash-ups” – collections of code snippets for both types of controllers mixed together. None of these examples matched exactly the Philips PCF8833 User Manual, which can be downloaded from this link.

http://www.nxp.com/acrobat_download/datasheets/PCF8833_1.pdf

So I set out to write a driver based solely on the LCD controller manufacturer’s manual. This is not to say that I didn’t have my own mysteries. I had to “invert” the entire display and “reverse” the RGB order to get the colors to work out properly.

To keep this tutorial simple, I will address only the **Philips PCF8833 LCD controller** that is on the Olimex boards that have the GE-12 sticker on the LCD. I will not address the issues of scrolling or partial displays (to conserve power) since these are rarely-used features.

I used the Olimex SAM7-EX256 evaluation board as the execution platform. This is an ARM7 board with many peripherals that is an excellent way to learn about the ARM architecture at a reasonable price ($120 from Sparkfun). I also used the YAGARTO/Eclipse platform as the cross-development system which is explained in great detail in my tutorial “**Using Open Source Tools for AT91SAM7 Cross Development**” which can be downloaded from the following link:

http://www.atmel.com/dyn/resources/prod_documents/atmel_tutorial_source.zip

Hardware connection issues are also not the subject of this tutorial; you can download the Olimex schematic for the SAM7-EX256 to see their design for a hardware interface to the Nokia 6100 LCD display.

LCD Display Orientation

The Nokia 6100 display has 132 x 132 pixels; each one with 12-bit color (4 bits RED, 4 bits GREEN and 4 bits BLUE). Practically speaking, you cannot see the first and last row and columns. The normal orientation is as follows:

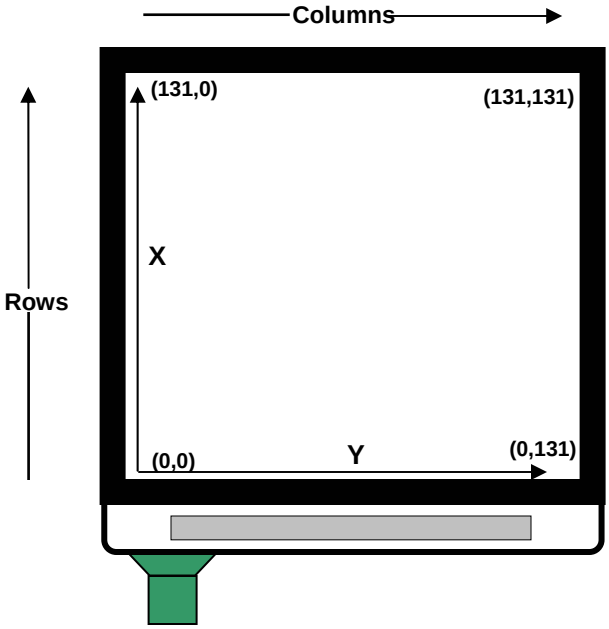


Figure 2. Default Orientation of Nokia 6100 LCD Display

That, of course, is upside-down on the Olimex SAM7-EX256 board if the silk-screen lettering is used as the up/down reference. So I set the “mirror x” and “mirror y” command to rotate the display 180 degrees, as shown below. This will be the orientation used in this tutorial (it is so easy to change back, if you desire).

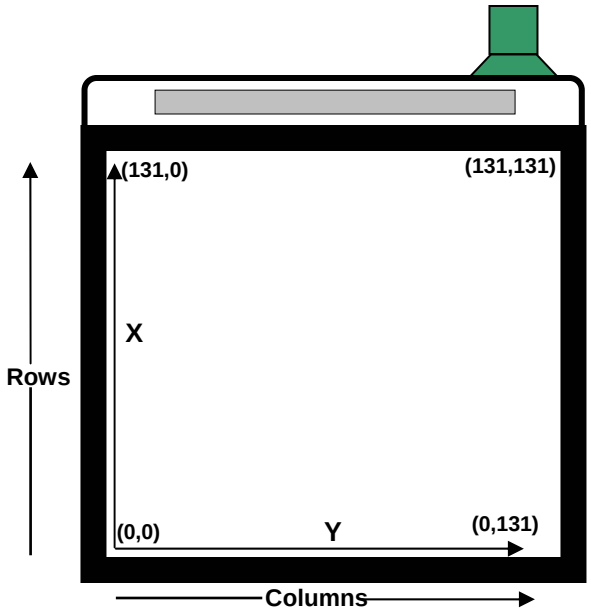


Figure 3. Tutorial Orientation of Nokia 6100 LCD Display

Communication with the Display

The Nokia 6100 uses a two-wire serial SPI interface (clock and data). The ARM7 microcontroller SPI peripheral generates the clock and data signals and the display acts solely as a slave device. Olimex elected to not implement the MISO0 signal that would allow the ARM microcontroller to read from the LCD display (you could read some identification codes, status, temperature data, etc). Therefore, the display is strictly **write-only**!

We send 9 bits to the display serially, the ninth bit indicates if a command byte or a data byte is being transmitted. Note in the timing diagram below from the Philips manual, the ninth bit (command or data) is clocked out first and is LOW to indicate a command byte or HIGH to indicate a data byte.

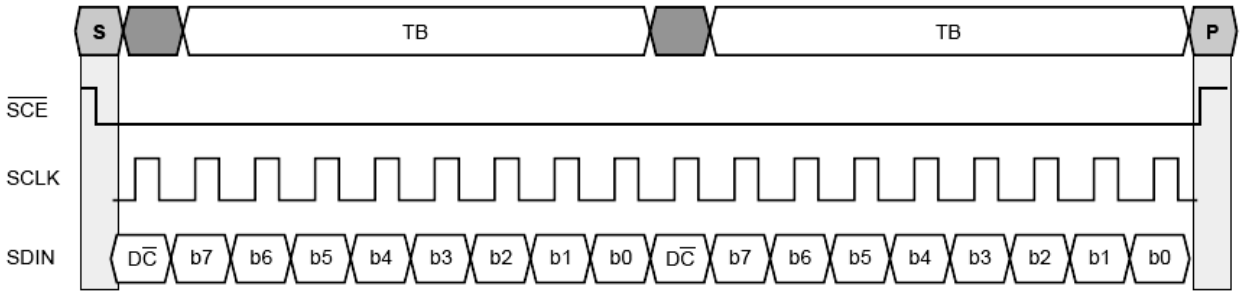


Figure 4. SPI serial interface sends commands and data bytes

How fast can this SPI interface be run? Since the PCF8833 data sheet specifies that the serial clock SCLK period be no less than 150 nsec, dividing the board’s master clock (48054841 Hz) by 8 gives a period of 166 nsec. Thus we can safely run the SPI interface at 6 MHz. I have run the SPI interface at 16 MHz and it still worked, but that is tempting fate.

The SAM7-EX256 board uses an ARM7 microprocessor; so commands or data are submitted to the SPI peripheral as unsigned integers (32 bits) wherein only the lower 9 bits are used.

For example, to send a command we clear bit 8 to specify this is a command transmission. The lowest 8 bits contain the desired PCF8833 command.

```
unsigned int      command;                              //  PCF8833 command byte

while ((pSPI->SPI_SR & AT91C_SPI_TXEMPTY) == 0);  // wait for the previous transfer to complete
command = (command & (~0x0100));                  // clear bit 8 - indicates a "command" byte
pSPI->SPI_TDR = command;                          // send the command
```

Likewise, to send a data byte we set bit 8 to specify that this is a data transmission. The lowest 8 bits contain the desired PCF8833 data byte.

```
unsigned int      data;                                 //  PCF8833 data byte

while ((pSPI->SPI_SR & AT91C_SPI_TXEMPTY) == 0);  // wait for the previous transfer to complete
data = (data | 0x0100);                           // set bit 8 - indicates a "data" byte
pSPI->SPI_TDR = data;                             // send the command
```

Both snippets have a “wait until TXEMPTY” to guarantee that a new command/data is not started before the previous one has completed. This is quite safe as you will never get stuck forever in that wait loop.

The LCD driver has three functions supporting the SPI interface to the LCD:

InitSpi( ) - sets up the SPI interface #1 to communicate with the LCD

WriteSpiCommand(command) - sends a command byte to the LCD

WriteSpiData(data) - sends a data byte to the LCD

Using these commands is quite simple; for example, to initialize the SPI interface and then set the contrast:

```
InitSpi( );                   // Initialize SPI interface to LCD
WriteSpiCommand(SETCON);      // Write contrast  (command 0x25)
WriteSpiData(0x30);           // contrast 0x30  (range is -63 to +63)
```

The hardware interface uses five I/O port pins; four bits from PIOA and one bit from PIOB, as shown in Table 1 and Figure 5 below.

PA2	LCD Reset (set to low to reset)
PA12	LCD chip select (set to low to select the LCD chip)
PA16	SPI0_MISO Master In - Slave Out (not used in LCD interface)
PA17	SPI0_MOSI Master Out - Slave In pin (Serial Data to LCD slave)
PA18	SPI0_SPCK Serial Clock (to LCD slave)
PB20	backlight control (normally PWM control, 1 = full on)

Table 1. I/O port bits used to support the SPI interface to the LCD Display

Note in Table 1 above that Olimex elected not to support the SPIO_MOSI – Slave In bit (PA16) which would have allowed the user to read from the display. The LED backlight needs a lot of current, so a boost converter is used for this purpose. The backlight can be turned on and off using PB20. It looks like you might be able to PWM the backlight, but I doubt anyone would want the backlight to be at half brightness.

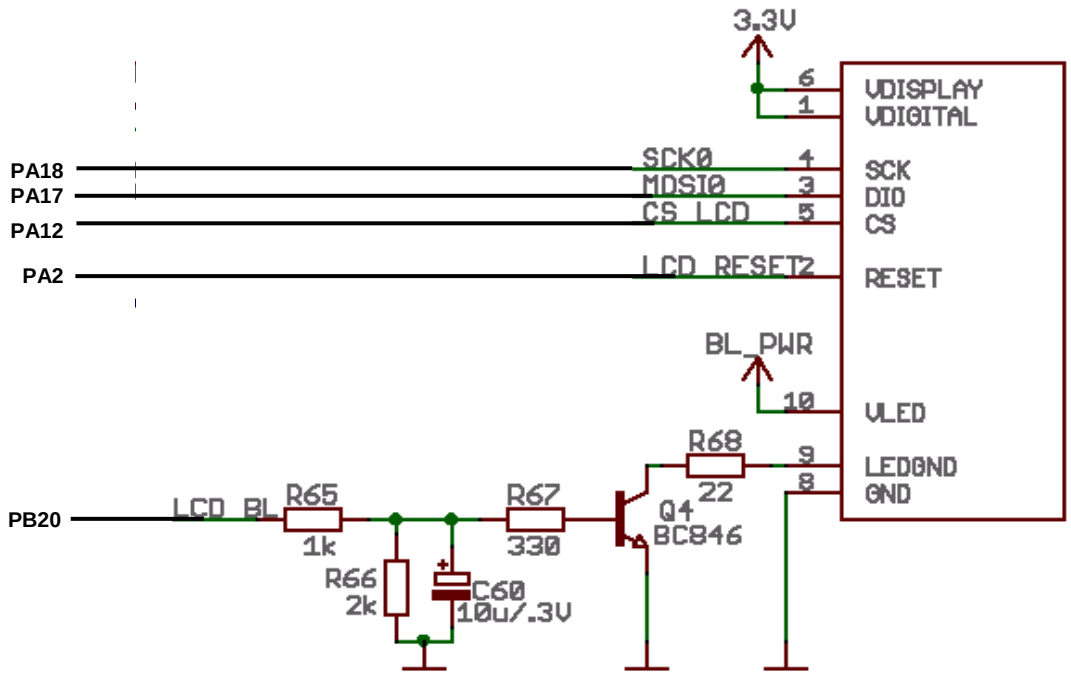


Figure 5. Hardware Interface to Nokia 6100 LCD Display

Addressing Pixel Memory

The Philips PCF8833 controller has a 17424 word memory (132 x 132), where each word is 12 bits (4-bit color each for red, green and blue). You address it by specifying the address of the desired pixel with the Page Address Set command (rows) and the Column Address Set command (columns).

The Page Address Set and Column Address Set command specify two things, the starting pixel and the ending pixel. This has the effect of creating a drawing box. This sounds overly complex, but it has a wrap-around and auto-increment feature that greatly simplifies writing character fonts and filling rectangles.

The pixel memory has 132 rows and 132 columns, as shown below in Figure 6

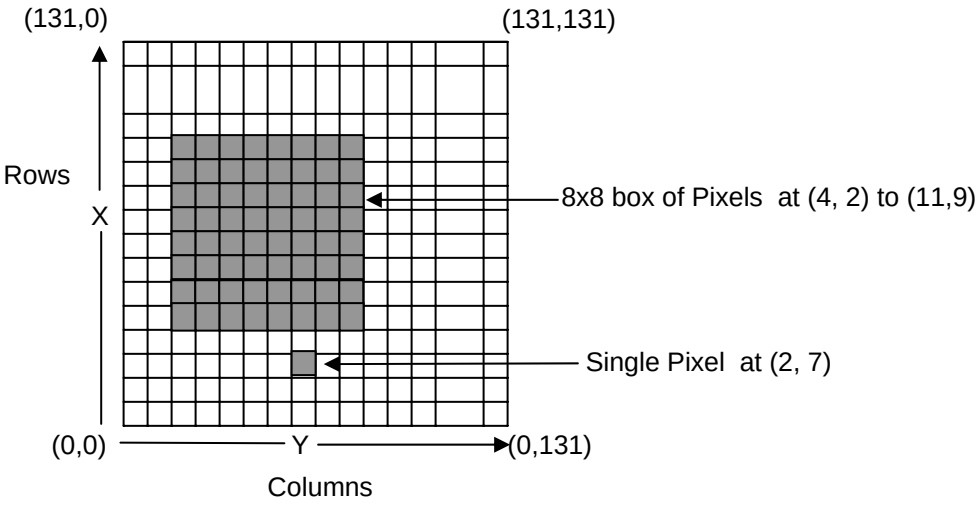


Figure 6. Philips PCF8833 Pixel Memory

To address a single pixel, just specify the same location for the starting pixel and the ending pixel on each axis. For example, to specify a single pixel at (2, 7) use the following sequence.

```
WriteSpiCommand(PASET);          // Row address set  (command 0x2B)
WriteSpiData(2);                 // starting x address
WriteSpiData(2);                 // ending x address (same as start)

WriteSpiCommand(CASET);          // Column address set  (command 0x2A)
WriteSpiData(7);                 // starting y address
WriteSpiData(7);                 // ending y address (same as start)
```

To address a rectangular area of pixels, just specify the starting location and the ending location on each axis, as shown below. For example, to define a drawing rectangle from (4, 3) to (11, 9) use the following sequence.

```
WriteSpiCommand(PASET);          // Row address set  (command 0x2B)
WriteSpiData(4);                 // starting x address
WriteSpiData(11);                // ending x address

WriteSpiCommand(CASET);          // Column address set  (command 0x2A)
WriteSpiData(3);                 // starting y address
WriteSpiData(9);                 // ending y address
```

Once the drawing boundaries have been established (either a single pixel or a rectangular group of pixels), any subsequent memory operations are confined to that boundary. For instance, if you try write more pixels than defined by the boundaries, the extra pixels are discarded by the controller.

12-Bit Color Data

The Philips PCF8833 LCD controller has three different ways to specify a pixel's color.

1. **12 bits per pixel** (native mode)

   Selection of the native 12 bits/pixel mode is accomplished by sending the Color Interface Pixel Format command (0x3A) followed by a single data byte containing the value 3.

   This encoding requires a Memory Write command and 1.5 subsequent data bytes to specify a single pixel. The bytes are packed so that two pixels will occupy three sequential bytes and the process repeats until the drawing boundaries are used up. Figure 7 illustrates the 12 bits/pixel encoding.

0	0	1	0	1	1	0	0	RAMWR command (memory write)
R	R	R	R	G	G	G	G	Data: Red and green from 1st pixel
B	B	B	B	R	R	R	R	Data: Blue from 1st pixel; red from 2nd pixel
G	G	G	G	B	B	B	B	Data: Green and blue from 2nd pixel

Figure 7. Color encoding for 12 bits / pixel - example illustrates sending 2 pixels

You might pose the question "What happens if I specify a single pixel with just two data bytes. Will the 4-bits of red information from the next pixel (usually set to zero) perturb the neighboring pixel? The answer is no since the PCF8833 controller writes to display RAM only when it gets a complete pixel. The straggler red bits from the next pixel wait for the completion of the remaining colors which will never come. Appearance of any command will cancel the previous memory operation and discard the unused

pixel information. To be safe, I added a NOP command in the LCDSetPixel( ) function to guarantee that the unused blue information from the next pixel is discarded.

Figure 8 demonstrates how to send a single pixel using 12-bit encoding. Note that the last 4 red bits from the next pixel will be ignored.

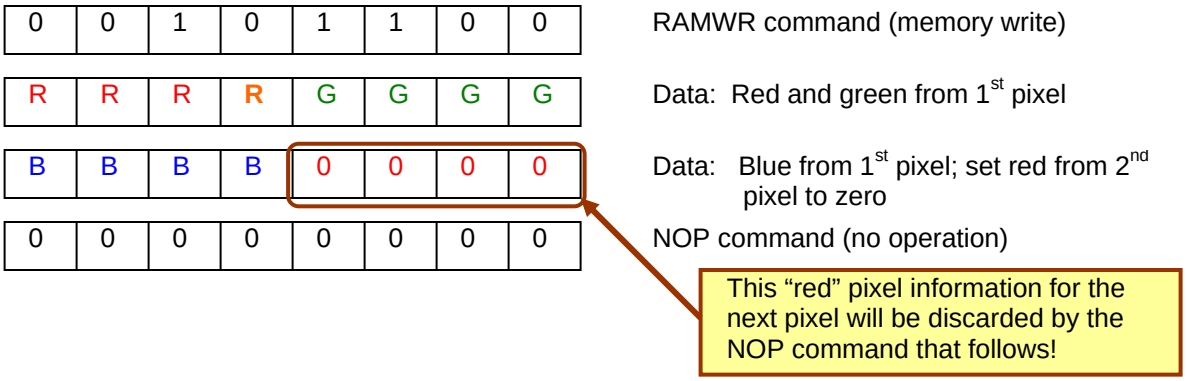


Figure 8. Color encoding for 12 bits / pixel - example illustrates sending 1pixel

2. **8 bits per pixel**

Selection of the reduced resolution 8 bits/pixel mode is accomplished by sending the Color Interface Pixel Format command (0x3A) followed by a single data byte containing the value 2.

This encoding requires a Memory Write command and a single subsequent data byte to specify a single pixel. The data byte contains all the color information for one pixel. The color information is encoded as 3 bits for red, 3 bits for green and 2 bits for blue, as shown in Figure 9 below

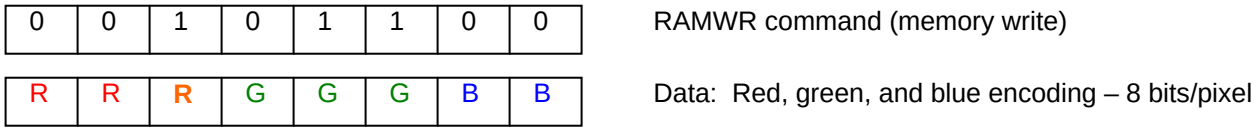


Figure 9. Color encoding for 8 bits – per - pixel

The important thing to note here is that this 8-bit color encoding will be converted to the 12-bit encoding by the Color Table that you set up in advance. This Color Set table will convert 3-bit RED to 4-bit RED, 3-bit GREEN to 4-bit GREEN and 2-bit BLUE to 4-bit BLUE. This is made possible by the specification of the 20 entry color table in the initialization step.

```
WriteSpiCommand(RGBSET);        // Define Color Table  (command 0x2D)
WriteSpiData(0);                // red 000 value
WriteSpiData(2);                // red 001 value
WriteSpiData(5);                // red 010 value
WriteSpiData(7);                // red 011 value
WriteSpiData(9);                // red 100 value
WriteSpiData(11);               // red 101 value
WriteSpiData(14);               // red 110 value
WriteSpiData(16);               // red 111 value
WriteSpiData(0);                // green 000 value
WriteSpiData(2);                // green 001 value
WriteSpiData(5);                // green 010 value
WriteSpiData(7);                // green 011 value
WriteSpiData(9);                // green 100 value
WriteSpiData(11);               // green 101 value
WriteSpiData(14);               // green 110 value
WriteSpiData(16);               // green 111 value
WriteSpiData(0);                // blue 000 value
WriteSpiData(6);                // blue 001 value
WriteSpiData(11);               // blue 010 value
WriteSpiData(15);               // blue 011 value
```

Consider the following points. The resolution of the Nokia 6100 display is 132 x 132 pixels, 12 bits/pixel. Since the 8 bits/pixel encoding is converted by the color table to 12 bits/pixel, there is no saving of display memory. The 8 bits/pixel encoding would use about 1/3 less data bytes to fill an area, so there would be a performance gain in terms of the number of bytes transferred. The 8 bits/pixel encoding would make a photograph look terrible. In the author's view, there's very little to be gained by using this mode in an ARM microcontroller environment. Therefore, I elected to not implement the color table and 8-bit encoding in this driver.

3. **16 bits per pixel**

Selection of 16 bits/pixel mode is accomplished by sending the Color Interface Pixel Format command (0x3A) followed by a single data byte containing the value 5.

This encoding requires a Memory Write command and two subsequent data bytes to specify a single pixel. The color information is encoded as 5 bits for RED, 6 bits for GREEN and 5 bits for BLUE, as shown in Figure 10 below

0	0	1	0	1	1	0	0	RAMWR command (memory write)
R	R	R	R	R	G	G	G	Data: Red (5 bits), Green (6 bits)
G	G	G	B	B	B	B	B	Data: Green (6 bits), Blue (5 bits)

Figure10. Color encoding for 16 bits – per - pixel

This pixel encoding is converted by the controller using a dithering technique to the 12-bit data for the pixel RAM. The net effect is to give 65k color variations. My view is that nobody is going to display the Mona Lisa on this tiny display, so 16-bit color encoding would be rarely used. I did not include support for it in the driver software, but you could easily add it if you desire.

Wrap-Around and Auto Increment

The wrap-around feature is the cornerstone of the controller's design and it amazes me how many people ignored it in drawing rectangles and character fonts. This feature allows you to efficiently draw a character or fill a box with just a simple loop – taking advantage of the wrap-around after writing the pixel in the last column and auto-incrementing to the next row. Remember how the pixel was addressed by defining a "drawing box"? If you are planning to draw an 8 x 8 character font, define the drawing box as 8 x 8 and do a simple loop on 64 successive pixels. The row and column addresses will automatically increment and wrap back when you come to the end of a row, as shown in Figure 11 below.

The rules for Auto-incrementing and Wrap-Around are as follows.

- Set the column and row address to the bottom left of the drawing box.
- Set up a loop to do all the pixels in the box. Specifically, since three data bytes will specify the color for two pixels, the loop will typically iterate over ½ the number of pixels
- Writing three memory bytes will illuminate two pixels (12-bit resolution). Each pixel written automatically advances the column address. When the "max" column address pixel is done, the column address wraps back to the column starting address **AND** the row address increments by one. Now keep writing memory bytes until the next row is illuminated and so on.

Figure 11 shows the traversal of the drawing box.

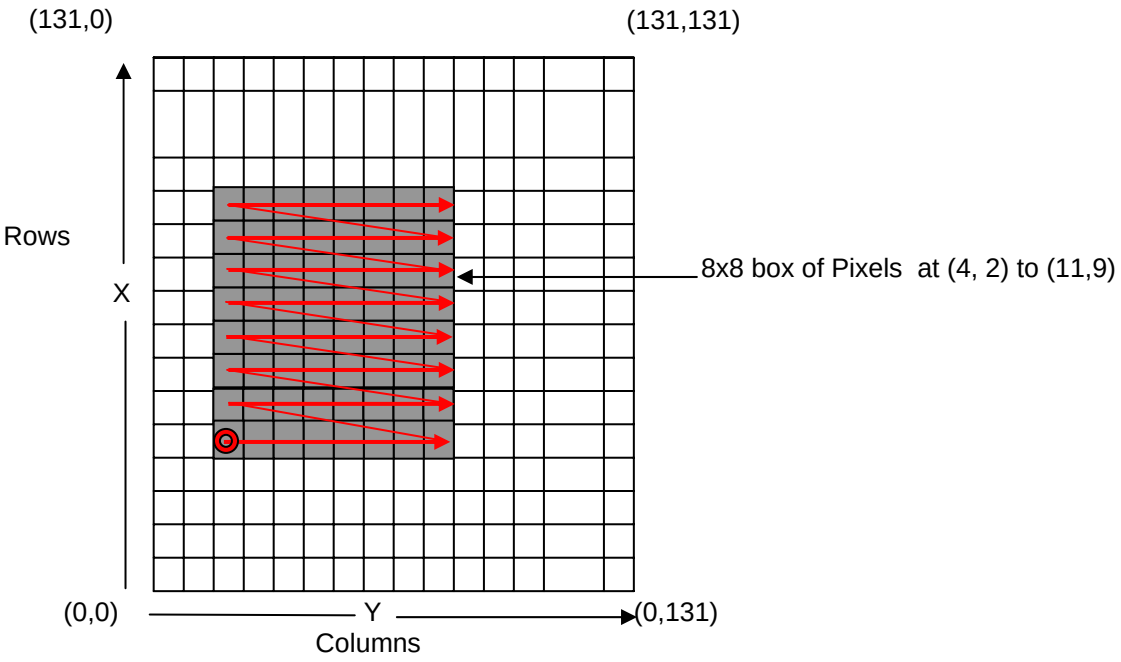


Figure 11. Drawing Box permits auto-increment and wrap-around.

To illustrate this technique, Figure 12 shows the code to fill an 8 x 8 box shown above. Note that we set the row and column address just once (pointing to the lower left corner). Then we do a single Memory Write command followed by three data bytes done 33 times. The grand total is 106 SPI transmissions.

Compare that to the implementation where you address each pixel, set Memory Write and feed two bytes of color data for each pixel. The grand total would be 576 SPI transmissions. The advantage gained using the auto-increment and wrap-around features is obvious.

```
// Row address set  (command 0x2B)
WriteSpiCommand(PASET);
WriteSpiData(4);
WriteSpiData(11);

// Column address set  (command 0x2A)
WriteSpiCommand(CASET);
WriteSpiData(2);
WriteSpiData(9);

// Write Memory  (command  0x2C)
WriteSpiCommand(RAMWR);

// loop on total number of pixels / 2
for (i = 0; i < ((((11 - 4 + 1) * (9 - 2 + 1)) / 2) + 1); i++) {
        // use the color value to output three data bytes covering two pixels
        WriteSpiData((color >> 4) & 0xFF);
        WriteSpiData(((color & 0xF) << 4) | ((color >> 8) & 0xF));
        WriteSpiData(color & 0xFF);
}
```

Add one to account for possible round-off error in the divide by 2

Figure 12. Code Snippet to Fill an 8 x 8 box

Code to use this technique to draw a character font is similar, but at each pixel you have to determine if the font calls for a foreground color or the background color.

Initializing the LCD Display

This was a surprise to me but the Philips PCF8833 does not quite boot into a “ready to display” mode after hardware reset. The following is the minimal commands/data needed to place it into 12-bit color mode.

First, we do a hardware reset with a simple manipulation of the port pin. Reset is asserted low on this controller.

```
// Hardware reset
LCD_RESET_LOW;
Delay(20000);
LCD_RESET_HIGH;
Delay(20000);
```

The controller boots into SLEEPIN mode, which keeps the booster circuits off. We need to exit sleep mode which will also turn on all the voltage booster circuits.

```
// Sleep out  (command 0x11)
WriteSpiCommand(SLEEPOUT);
```

This is still a mystery to me, but I had to invert the display and reverse the RGB setting to get the colors to work correctly in this particular display. If you have trouble, consider removing this command.

```
// Inversion on  (command 0x20)
WriteSpiCommand(INVON);              // seems to be required for this controller
```

For this driver, I elected to use the 12-bit color pixel format exclusively.

```
// Color Interface Pixel Format  (command 0x3A)
WriteSpiCommand(COLMOD);
WriteSpiData(0x03);                  // 0x03 = 12 bits-per-pixel
```

In setting up the memory access controller, I elected to use the “mirror x” and “mirror y” commands to reorient the x and y axes to agree with the silk screen lettering on the Olimex board. If you want the default orientation, send the data byte 0x08 instead. Finally, I had to reverse the RGB color setting to get the color information to work properly. You may want to experiment with this setting.

```
// Memory access controller  (command 0x36).
WriteSpiCommand(MADCTL);
WriteSpiData(0xC8);                  // 0xC0 = mirror x and y, reverse rgb
```

I found that setting the contrast varies from display to display. You may want to try several different contrast data values and observe the results on the display.

```
// Write contrast  (command 0x25)
WriteSpiCommand(SETCON);
WriteSpiData(0x30);                  // contrast 0x30
Delay(2000);
```

Now that the display is initialized properly, we can turn on the display and we’re ready to start displaying characters and graphics.

```
// Display On  (command 0x29)
WriteSpiCommand(DISPON);
```

Description of the Software Modules

A printout of the LCD driver code and a simple main program to test it follows directly below. A suitable Eclipse GNU project is also available in zip file format for those who have the Olimex board.

LCD.H

This include file contains the Philips commands and color specification codes.

```
#ifndef Lcd_h
#define Lcd_h
//  ********************************************************************************
//                    LCD Include File for Philips PCF8833 STN RGB- 132x132x3 Driver
//
//                    Taken from Philips data sheet Feb 14, 2003
//  ********************************************************************************

// Philips PCF8833 LCD controller command codes
#define NOP                     0x00    // nop
#define SWRESET                 0x01    // software reset
#define BSTROFF                 0x02    // booster voltage OFF
#define BSTRON                  0x03    // booster voltage ON
#define RDDIDIF                 0x04    // read display identification
#define RDDST                   0x09    // read display status
#define SLEEPIN                 0x10    // sleep in
#define SLEEPOUT                0x11    // sleep out
#define PTLON                   0x12    // partial display mode
#define NORON                   0x13    // display normal mode
#define INVOFF                  0x20    // inversion OFF
#define INVON                   0x21    // inversion ON
#define DALO                    0x22    // all pixel OFF
#define DAL                     0x23    // all pixel ON
#define SETCON                  0x25    // write contrast
#define DISPOFF                 0x28    // display OFF
#define DISPON                  0x29    // display ON
#define CASET                   0x2A    // column address set
#define PASET                   0x2B    // page address set
#define RAMWR                   0x2C    // memory write
#define RGBSET                  0x2D    // colour set
#define PTLAR                   0x30    // partial area
#define VSCRDEF                 0x33    // vertical scrolling definition
#define TEOFF                   0x34    // test mode
#define TEON                    0x35    // test mode
#define MADCTL                  0x36    // memory access control
#define SEP                     0x37    // vertical scrolling start address
#define IDMOFF                  0x38    // idle mode OFF
#define IDMON                   0x39    // idle mode ON
#define COLMOD                  0x3A    // interface pixel format
#define SETVOP                  0xB0    // set Vop
#define BRS                     0xB4    // bottom row swap
#define TRS                     0xB6    // top row swap
#define DISCTR                  0xB9    // display control
#define DOR                     0xBA    // data order
#define TCDFE                   0xBD    // enable/disable DF temperature compensation
#define TCVOPE                  0xBF    // enable/disable Vop temp comp
#define EC                      0xC0    // internal or external  oscillator
#define SETMUL                  0xC2    // set multiplication factor
#define TCVOPAB                 0xC3    // set TCVOP slopes A and B
#define TCVOPCD                 0xC4    // set TCVOP slopes c and d
#define TCDF                    0xC5    // set divider frequency
#define DF8COLOR                0xC6    // set divider frequency 8-color mode
#define SETBS                   0xC7    // set bias system
#define RDTEMP                  0xC8    // temperature read back
#define NLI                     0xC9    // n-line inversion
#define RDID1                   0xDA    // read ID1
#define RDID2                   0xDB    // read ID2
#define RDID3                   0xDC    // read ID3
```

```
// backlight control
#define BKLGHT_LCD_ON  1
#define BKLGHT_LCD_OFF 2

// Booleans
#define NOFILL 0
#define FILL           1

// 12-bit color definitions
#define WHITE          0xFFF
#define BLACK          0x000
#define RED            0xF00
#define GREEN          0x0F0
#define BLUE           0x00F
#define CYAN           0x0FF
#define MAGENTA        0xF0F
#define YELLOW         0xFF0
#define BROWN          0xB22
#define ORANGE         0xFA0
#define PINK           0xF6A

// Font sizes
#define SMALL  0
#define MEDIUM 1
#define LARGE  2

// hardware definitions
#define SPI_SR_TXEMPTY
#define LCD_RESET_LOW     pPIOA->PIO_CODR   = BIT2
#define LCD_RESET_HIGH    pPIOA->PIO_SODR   = BIT2

// mask definitions
#define   BIT0        0x00000001
#define   BIT1        0x00000002
#define   BIT2        0x00000004
#define   BIT3        0x00000008
#define   BIT4        0x00000010
#define   BIT5        0x00000020
#define   BIT6        0x00000040
#define   BIT7        0x00000080
#define   BIT8        0x00000100
#define   BIT9        0x00000200
#define   BIT10       0x00000400
#define   BIT11       0x00000800
#define   BIT12       0x00001000
#define   BIT13       0x00002000
#define   BIT14       0x00004000
#define   BIT15       0x00008000
#define   BIT16       0x00010000
#define   BIT17       0x00020000
#define   BIT18       0x00040000
#define   BIT19       0x00080000
#define   BIT20       0x00100000
#define   BIT21       0x00200000
#define   BIT22       0x00400000
#define   BIT23       0x00800000
#define   BIT24       0x01000000
#define   BIT25       0x02000000
#define   BIT26       0x04000000
#define   BIT27       0x08000000
#define   BIT28       0x10000000
#define   BIT29       0x20000000
#define   BIT30       0x40000000
#define   BIT31       0x80000000

#endif         // Lcd_h
```

LCD.C

The lcd.c module contains the SPI support code and a complete set of LCD graphics primitives. The line drawing and circle routines are derived from the famous Jack Bresenham algorithms from 1962. The rest of the graphics primitives are designed by the author. The font tables were adapted from the ARM example submitted to the Sparkfun web site by Jimbo.

```
//  ***********************************************************************************************
//                                          lcd.c
//
//            Nokia 6610 LCD Display Driver  (PCF8833 Controller)
//
//            Controller used for LCD Display is a Philips PCF8833 STN RGB-132x132x3 driver
//            Note:  Leadis LDS176 controller (GE-12) is 100% compatible with Philips PCF8833
//
//
//            Nokia 6610 has a 132 x 132 pixel matrix - each pixel has 12 bits of color information.
//
//                                        ____
//                                       |  |
//                                  _____|__|___
//                                 |  ===================== |
//             ^  (131,0)|-------------------------|(131,131)
//             |         |                         |
//             |         |                         |
//             |         |                         |
//             |         |                         |
//             |         |                         |
//             |     Rows|    Nokia 6610 Display   |    Note: In general, you can't view Row 0 or Row 131
//             |         |                         |          and Column 0 and Column 131.
//             X         |                         |
//             |         |                         |
//             |         |                         |
//             |         |                         |
//             |         |                         |
//             |         |-------------------------|
//                     (0,0)      Columns        (0,131)
//
//                       ---------- Y ------------>
//
//
//            132 x 132 pixel matrix has three methods to specify the color info
//
//            1.      12 bits per pixel
//                    requires command and 1.5 data bytes to specify a single pixel
//                    (3 data bytes can specify 2 pixels)
//
//            2.      8 bits per pixel
//                    requires one command and one data byte to specify the single pixel
//                    note: pixel data byte converted by RGB table to 12-bit format above
//
//            3.      16 bits per pixel
//                    requires one command and two data bytes to specify the single pixel
//                     note: pixel data bytes converted by dithering circuitry to 12-bit format above
//
//      THIS IMPLEMENTATION USES THE 12 BITS PER PIXEL METHOD!
//                                  ------------------------
//
//
//            Notes:  To orient the LCD display so that the origin (0, 0) is at the opposite
//                    corner from the connector (to match the orientation of the Olimex SAM7-EX256
//                    boards silk screen printing), I selected "mirror x" and "mirror y" setting
//                    in the MADCTL command (MY=1, MX=1).
//
//                    Also, while I can't yet explain it, I had to reverse the RGB orientation in the
//                    MADCTL command and invert the entire display using the INVON command to make
//                    0x0=black, 0xF= full red.
//
//
```

```
//
//
//      HARDWARE INTERFACE
//      ------------------
//
//      The Nokia 6610 display uses a SPI serial interface (9 bits)
//
//              PA2  = LCD Reset (set to low to reset)
//              PA12 = LCD chip select (set to low to select the LCD chip)
//              PA16 = SPI0_MISO Master In - Slave Out (not used in Olimex SAM7-EX256 LCD interface)
//              PA17 = SPI0_MOSI Master Out - Slave In pin (Serial Data to LCD slave)
//              PA18 = SPI0_SPCK  Serial Clock (to LCD slave)
//
//              SPI baud rate set to MCK/2 = 48054841/8 = 6006855 baud
//              (period = 166 nsec, OK since 150 nsec period is min for PCF8833)
//
//      The important thing to note is that you CANNOT read from the LCD!
//
//
//  Author:  James P Lynch  July 7, 2007
// ********************************************************************************************************************


// ************************************
//  Include Files
// ************************************
#include "at91sam7x256.h"
#include "lcd.h"
#include "bmp.h"


// ************************************
//  forward references
// ************************************
const unsigned char FONT6x8[97][8];
const unsigned char FONT8x8[97][8];
const unsigned char FONT8x16[97][16];
void InitLcd(void);
void Backlight(unsigned char state);
void WriteSpiCommand(unsigned int data);
void WriteSpiData(unsigned int data);
void InitLcd(void);
void LCDWrite130x130bmp(void);
void LCDClearScreen(void);
void LCDSetXY(int x, int y);
void LCDSetPixel(int  x, int  y, int  color);
void LCDSetLine(int x1, int y1, int x2, int y2, int color);
void LCDSetRect(int x0, int y0, int x1, int y1, unsigned char fill, int color);
void LCDSetCircle(int x0, int y0, int radius, int color);
void LCDPutChar(char c, int  x, int  y, int size, int fcolor, int bcolor);
void LCDPutString (char *lcd_string, const char *font_style, unsigned char x, unsigned char y,
                   unsigned char fcolor, unsigned char bcolor);
void Delay (unsigned long a);



// ************************************************
//  Pointers to AT91SAM7X256 peripheral data structures
// ************************************************
volatile       AT91PS_PIO    pPIOA   = AT91C_BASE_PIOA;
volatile       AT91PS_PIO    pPIOB   = AT91C_BASE_PIOB;
volatile       AT91PS_SPI    pSPI    = AT91C_BASE_SPI0;
volatile       AT91PS_PMC    pPMC    = AT91C_BASE_PMC;
volatile       AT91PS_PDC    pPDC    = AT91C_BASE_PDC_SPI0;
```

```
//  ********************************************************************************************
//                                        InitSpi( )
//
//     Sets up SPI channel 0 for communications to Nokia 6610 LCD Display
//
//     I/O ports used:       PA2  = LCD Reset (set to low to reset)
//                           PA12 = LCD chip select (set to low to select the LCD chip)
//                           PA16 = SPI0_MISO Master In - Slave Out (not used in LCD interface)
//                           PA17 = SPI0_MOSI Master Out - Slave In pin (Serial Data to LCD slave)
//                           PA18 = SPI0_SPCK  Serial Clock (to LCD slave)
//                           PB20 = backlight control (normally PWM control, 1 = full on)
//
//  Author:  Olimex, James P Lynch     July 7, 2007
//  ********************************************************************************************

void InitSpi(void) {

  // Pin PB20 used for LCD_BL  (backlight)
  pPIOB->PIO_OER    = BIT20;  // Configure PB20 as output
  pPIOB->PIO_SODR   = BIT20;  // Set PB20 to HIGH (backlight under PWM control - this will turn it full on)

  // Pin PA2 used for LCD_RESET
  pPIOA->PIO_OER    = BIT2;   // Configure PA2 as output
  pPIOA->PIO_SODR   = BIT2;   // Set PA2 to HIGH   (assert LCD Reset low then high to reset the controller)

  // Pin PA2 used for CS_LCD (chip select)
  pPIOA->PIO_OER    = BIT12;  // Configure PA12 as output
  pPIOA->PIO_SODR   = BIT12;  // Set PA12 to HIGH  (assert CS_LCD low to enable transmission)

  // Disable the following pins from PIO control (will be used instead by the SPI0 peripheral)
  //    BIT12 = PA12 -> SPI0_NPCS0 chip select
  //    BIT16 = PA16 -> SPI0_MISO Master In - Slave Out (not used in LCD interface)
  //    BIT17 = PA17 -> SPI0_MOSI Master Out - Slave In pin (Serial Data to LCD slave)
  //    BIT18 = PA18 -> SPI0_SPCK  Serial Clock (to LCD slave)
  pPIOA->PIO_PDR = BIT12 | BIT16 | BIT17 | BIT18;   // Peripheral A Disable Register (Disable PIO control)
  pPIOA->PIO_ASR = BIT12 | BIT16 | BIT17 | BIT18;   // Peripheral A Select Register (all 4 bits are in PIOA)
  pPIOA->PIO_BSR = 0;                               // Peripheral B Select Register (no bits in PIOB)

  //enable the SPI0 Peripheral clock
  pPMC->PMC_PCER = 1 << AT91C_ID_SPI0;

  // SPI Control Register  SPI_CR
  pSPI->SPI_CR = AT91C_SPI_SWRST | AT91C_SPI_SPIEN; //Software reset, SPI Enable  (0x81)
  pSPI->SPI_CR = AT91C_SPI_SPIEN;                   //SPI Enable (0x01)

  // SPI Mode Register  SPI_MR = 0xE0011
  pSPI->SPI_MR =
       (AT91C_SPI_DLYBCS & (0 << 24)) |             // Delay between chip selects (take default: 6 MCK
                                                    // periods)
       (AT91C_SPI_PCS & (0xE << 16))  |             // Peripheral Chip Select (selects SPI_NPCS0 or PA12)

       (AT91C_SPI_LLB & (0 << 7))     |             // Local Loopback Enabled (disabled)
       (AT91C_SPI_MODFDIS & (1 << 4)) |             // Mode Fault Detection (disabled)
       (AT91C_SPI_PCSDEC & (0 << 2))  |             // Chip Select Decode (chip selects connected directly
                                                    // to peripheral)
       (AT91C_SPI_PS & (0 << 1))      |             // Peripheral Select (fixed)
       (AT91C_SPI_MSTR & (1 << 0));                 // Master/Slave Mode  (Master)

   // SPI Chip Select Register SPI_CSR[0] = 0x01010311
   pSPI->SPI_CSR[0] =
       (AT91C_SPI_DLYBCT & (0x01 << 24))       |    // Delay between Consecutive Transfers (32 MCK periods)
       (AT91C_SPI_DLYBS & (0x01 << 16))        |    // Delay Before SPCK (1 MCK period)
       (AT91C_SPI_SCBR & (0x10 << 8))          |    // Serial Clock Baud Rate (baudrate = MCK/8 = 48054841/8
                                                    // = 6006855 baud
       (AT91C_SPI_BITS & (AT91C_SPI_BITS_9))   |    // Bits per Transfer (9 bits)
       (AT91C_SPI_CSAAT & (0x0 << 3))          |    // Chip Select Active After Transfer
       (AT91C_SPI_NCPHA & (0x0 << 1))          |    // Clock Phase (data captured on falling edge)
       (AT91C_SPI_CPOL & (0x01 << 0));              // Clock Polarity (inactive state is logic one)
}
```

```
//  ********************************************************************************
//                                          WriteSpiCommand.c
//
//     Writes 9-bit command to LCD display via SPI interface
//
//         Inputs:     data  -  Leadis LDS176 controller/driver command
//
//
//         Note:  clears bit 8 to indicate command transfer
//
//  Author:  Olimex, James P Lynch     July 7, 2007
//  ********************************************************************************
void WriteSpiCommand(volatile unsigned int command) {

        // wait for the previous transfer to complete
        while((pSPI->SPI_SR & AT91C_SPI_TXEMPTY) == 0);

        // clear bit 8 - indicates a "command"
        command = (command & ~0x0100);

        // send the command
        pSPI->SPI_TDR = command;
}


//  ********************************************************************************
//                                          WriteSpiData.c
//
//     Writes 9-bit command to LCD display via SPI interface
//
//         Inputs:     data  -  Leadis LDS176 controller/driver command
//
//
//         Note:  Sets bit 8 to indicate data transfer
//
//  Author:  Olimex, James P Lynch     July 7, 2007
//  ********************************************************************************

void WriteSpiData(volatile unsigned int data) {

        // wait for the transfer to complete
        while ((pSPI->SPI_SR & AT91C_SPI_TXEMPTY) == 0);

        // set bit 8, indicates "data"
        data = (data | 0x0100);

        // send the data
        pSPI->SPI_TDR = data;
}


//  ********************************************************************************
//                                          Backlight.c
//
//     Turns the backlight on and off
//
//         Inputs:     state  -  1 = backlight on
//                                  2 = backlight off
//
//
//  Author:  Olimex, James P Lynch     July 7, 2007
//  ********************************************************************************
void Backlight(unsigned char state) {

        if (state == 1)
                pPIOB->PIO_SODR   = BIT20;    // Set PB20 to HIGH
        else
                pPIOB->PIO_CODR   = BIT20;    // Set PB20 to LOW
}
```

```
//  ********************************************************************************
//                                        InitLcd.c
//
//     Initializes the Philips PCF8833 LCD Controller
//
//        Inputs:     none
//
//     Author:  James P Lynch     July 7, 2007
//  ********************************************************************************
void InitLcd(void) {

  // Hardware reset
  LCD_RESET_LOW;
  Delay(20000);
  LCD_RESET_HIGH;
  Delay(20000);

  // Sleep out  (command 0x11)
  WriteSpiCommand(SLEEPOUT);

  // Inversion on  (command 0x20)
  WriteSpiCommand(INVON);            // seems to be required for this controller

  // Color Interface Pixel Format  (command 0x3A)
  WriteSpiCommand(COLMOD);
  WriteSpiData(0x03);                // 0x03 = 12 bits-per-pixel

  // Memory access controler  (command 0x36)
  WriteSpiCommand(MADCTL);
  WriteSpiData(0xC8);                // 0xC0 = mirror x and y, reverse rgb

  // Write contrast  (command 0x25)
  WriteSpiCommand(SETCON);
  WriteSpiData(0x30);                // contrast 0x30
  Delay(2000);

  // Display On  (command 0x29)
  WriteSpiCommand(DISPON);
}


//  ********************************************************************************
//                                        LCDWrite130x130bmp.c
//
//     Writes the entire screen from a bmp file
//     Uses Olimex BmpToArray.exe utility
//
//        Inputs:     picture in bmp.h
//
//     Author:  Olimex, James P Lynch     July 7, 2007
//  ********************************************************************************
void LCDWrite130x130bmp(void) {

       long                  j;                    // loop counter

       // Memory access controler  (command 0x36)
       WriteSpiCommand(MADCTL);
       WriteSpiData(0x48);     // no mirror Y (temporary to satisfy Olimex bmptoarray utility)

       // Display OFF
       WriteSpiCommand(DISPOFF);

       // Column address set  (command 0x2A)
       WriteSpiCommand(CASET);
       WriteSpiData(0);
       WriteSpiData(131);

       // Page address set  (command 0x2B)
       WriteSpiCommand(PASET);
       WriteSpiData(0);
       WriteSpiData(131);

       // WRITE MEMORY
       WriteSpiCommand(RAMWR);
```

```
        For (j = 0; j < sizeof(bmp); j++) {
                WriteSpiData(bmp[j]);
        }

        // Memory access controler  (command 0x36)
        WriteSpiCommand(MADCTL);
        WriteSpiData(0xC8);     // restore to (mirror x and y, reverse rgb)

        // Display On
        WriteSpiCommand(DISPON);
}


//  ***************************************************************************
//                                      LCDClearScreen.c
//
//     Clears the LCD screen to single color (BLACK)
//
//        Inputs:     none
//
//     Author:  James P Lynch     July 7, 2007
//  ***************************************************************************
void LCDClearScreen(void) {

        long    i;                      // loop counter

        // Row address set  (command 0x2B)
        WriteSpiCommand(PASET);
        WriteSpiData(0);
        WriteSpiData(131);

        // Column address set  (command 0x2A)
        WriteSpiCommand(CASET);
        WriteSpiData(0);
        WriteSpiData(131);

        // set the display memory to BLACK
        WriteSpiCommand(RAMWR);
        For (i = 0; i < ((131 * 131) / 2); i++) {
                WriteSpiData(BLACK);
                WriteSpiData(BLACK);
                WriteSpiData(BLACK);
        }
}


//  ***************************************************************************
//                                      LCDSetXY.c
//
//     Sets the Row and Column addresses
//
//        Inputs:       x  =   row address (0 .. 131)
//                      y  =   column address  (0 .. 131)
//
//
//        Returns:   nothing
//
//     Author:  James P Lynch     July 7, 2007
//  ***************************************************************************
void LCDSetXY(int  x, int  y) {

        // Row address set  (command 0x2B)
        WriteSpiCommand(PASET);
        WriteSpiData(x);
        WriteSpiData(x);

        // Column address set  (command 0x2A)
        WriteSpiCommand(CASET);
        WriteSpiData(y);
        WriteSpiData(y);
}
```

```
//  ***************************************************************************************
//                                      LCDSetPixel.c
//
//     Lights a single pixel in the specified color at the specified x and y addresses
//
//     Inputs:   x     =   row address (0 .. 131)
//               y     =   column address  (0 .. 131)
//               color =   12-bit color value  rrrrggggbbbb
//                            rrrr = 1111 full red
//                                   :
//                                   0000 red is off
//
//                            gggg = 1111 full green
//                                   :
//                                   0000 green is off
//
//                            bbbb = 1111 full blue
//                                   :
//                                   0000 blue is off
//
//        Returns:   nothing
//
//             Note: see lcd.h for some sample color settings
//
//     Author:  James P Lynch     July 7, 2007
//  ***************************************************************************************
void LCDSetPixel(int  x, int  y, int  color) {

       LCDSetXY(x, y);
       WriteSpiCommand(RAMWR);
       WriteSpiData((unsigned char)((color >> 4) & 0xFFFF));
       WriteSpiData((unsigned char)(((color & 0x0F) << 4) | 0x00));
       WriteSpiCommand(NOP);
}


//  ***************************************************************************************************
//                                      LCDSetLine.c
//
//     Draws a line in the specified color from (x0,y0) to (x1,y1)
//
//     Inputs:   x     =   row address (0 .. 131)
//               y     =   column address  (0 .. 131)
//               color =   12-bit color value  rrrrggggbbbb
//                            rrrr = 1111 full red
//                                   :
//                                   0000 red is off
//
//                            gggg = 1111 full green
//                                   :
//                                   0000 green is off
//
//                            bbbb = 1111 full blue
//                                   :
//                                   0000 blue is off
//
//     Returns:   nothing
//
//     Note:  good write-up on this algorithm in Wikipedia (search for Bresenham's line algorithm)
//            see lcd.h for some sample color settings
//
//     Authors:       Dr. Leonard McMillan, Associate Professor UNC
//                    Jack Bresenham IBM, Winthrop University (Father of this algorithm, 1962)
//
//                    Note: taken verbatim from Professor McMillan's presentation:
//                          http://www.cs.unc.edu/~mcmillan/comp136/Lecture6/Lines.html
//
//  ***************************************************************************************************
void LCDSetLine(int x0, int y0, int x1, int y1, int color) {

        int dy = y1 - y0;
       int dx = x1 - x0;
       int stepx, stepy;
```

```
        if (dy < 0) { dy = -dy;  stepy = -1; } else { stepy = 1; }
        if (dx < 0) { dx = -dx;  stepx = -1; } else { stepx = 1; }
        dy <<= 1;                           // dy is now 2*dy
        dx <<= 1;                           // dx is now 2*dx

       LCDSetPixel(x0, y0, color);

        if (dx > dy) {
            int fraction = dy - (dx >> 1);   // same as 2*dy - dx
            while (x0 != x1) {
                if (fraction >= 0) {
                    y0 += stepy;
                    fraction -= dx;          // same as fraction -= 2*dx
                }
                x0 += stepx;
                fraction += dy;              // same as fraction -= 2*dy
                LCDSetPixel(x0, y0, color);
            }
        } else {
            int fraction = dx - (dy >> 1);
            while (y0 != y1) {
                if (fraction >= 0) {
                    x0 += stepx;
                    fraction -= dy;
                }
                y0 += stepy;
                fraction += dx;
                LCDSetPixel(x0, y0, color);
            }
        }
}


//  ***************************************************************************************
//                                        LCDSetRect.c
//
//      Draws a rectangle in the specified color from (x1,y1) to (x2,y2)
//      Rectangle can be filled with a color if desired
//
//      Inputs:   x     =   row address (0 .. 131)
//                y     =   column address  (0 .. 131)
//                fill  =   0=no fill, 1-fill entire rectangle
//                color =   12-bit color value for lines     rrrrggggbbbb
//                             rrrr = 1111 full red
//                                  :
//                             0000 red is off
//
//                             gggg = 1111 full green
//                                  :
//                             0000 green is off
//
//                             bbbb = 1111 full blue
//                                  :
//                             0000 blue is off
/       Returns:   nothing
//
//              Notes:
//
//              The best way to fill a rectangle is to take advantage of the "wrap-around" featute
//              built into the Philips PCF8833 controller. By defining a drawing box, the memory can
//              be simply filled by successive memory writes until all pixels have been illuminated.
//
//                     1.  Given the coordinates of two opposing corners (x0, y0) (x1, y1)
//                         calculate the minimums and maximums of the coordinates
//
//                             xmin = (x0 <= x1) ? x0 : x1;
//                             xmax = (x0 > x1) ? x0 : x1;
//                             ymin = (y0 <= y1) ? y0 : y1;
//                             ymax = (y0 > y1) ? y0 : y1;
//
```

```
//                      2. Now set up the drawing box to be the desired rectangle
//
//                              WriteSpiCommand(PASET);              // set the row boundaries
//                              WriteSpiData(xmin);
//                              WriteSpiData(xmax);
//                              WriteSpiCommand(CASET);              // set the column boundaries
//                              WriteSpiData(ymin);
//                              WriteSpiData(ymax);
//
//                      3. Calculate the number of pixels to be written divided by 2
//
//                              NumPixels = ((((xmax - xmin + 1) * (ymax - ymin + 1)) / 2) + 1)
//
//                              You may notice that I added one pixel to the formula.
//                              This covers the case where the number of pixels is odd and we
//                              would lose one pixel due to rounding error. In the case of
//                              odd pixels, the number of pixels is exact.
//                              in the case of even pixels, we have one more pixel than
//                              needed, but it cannot be displayed because it is outside
//                              the drawing box.
//
//                              We divide by 2 because two pixels are represented by three bytes.
//                              So we work through the rectangle two pixels at a time.
//
//                      4.  Now a simple memory write loop will fill the rectangle
//
//                              for (i = 0; i < ((((xmax - xmin + 1) * (ymax - ymin + 1)) / 2) + 1); i++) {
//                                      WriteSpiData((color >> 4) & 0xFF);
//                                      WriteSpiData(((color & 0xF) << 4) | ((color >> 8) & 0xF));
//                                      WriteSpiData(color & 0xFF);
//                              }
//
//              In the case of an unfilled rectangle, drawing four lines with the Bresenham line
//              drawing algorithm is reasonably efficient.
//
//      Author:  James P Lynch      July 7, 2007
//  ********************************************************************************************

void LCDSetRect(int x0, int y0, int x1, int y1, unsigned char fill, int color) {
        int     xmin, xmax, ymin, ymax;
        int             i;

        // check if the rectangle is to be filled
        if (fill == FILL) {

                // best way to create a filled rectangle is to define a drawing box
                // and loop two pixels at a time

                // calculate the min and max for x and y directions
                xmin = (x0 <= x1) ? x0 : x1;
                xmax = (x0 > x1) ? x0 : x1;
                ymin = (y0 <= y1) ? y0 : y1;
                ymax = (y0 > y1) ? y0 : y1;

                // specify the controller drawing box according to those limits
                // Row address set  (command 0x2B)
                WriteSpiCommand(PASET);
                WriteSpiData(xmin);
                WriteSpiData(xmax);

                // Column address set  (command 0x2A)
                WriteSpiCommand(CASET);
                WriteSpiData(ymin);
                WriteSpiData(ymax);

                // WRITE MEMORY
                WriteSpiCommand(RAMWR);

                // loop on total number of pixels / 2
                for (i = 0; i < ((((xmax - xmin + 1) * (ymax - ymin + 1)) / 2) + 1); i++) {

                        // use the color value to output three data bytes covering two pixels
                        WriteSpiData((color >> 4) & 0xFF);
                        WriteSpiData(((color & 0xF) << 4) | ((color >> 8) & 0xF));
                        WriteSpiData(color & 0xFF);
                }
```

```
        } else {

                // best way to draw un unfilled rectangle is to draw four lines
                LCDSetLine(x0, y0, x1, y0, color);
                LCDSetLine(x0, y1, x1, y1, color);
                LCDSetLine(x0, y0, x0, y1, color);
                LCDSetLine(x1, y0, x1, y1, color);
    }
}




//  ********************************************************************************
//                                          LCDSetCircle.c
//
//      Draws a line in the specified color at center (x0,y0) with radius
//
//      Inputs:   x0     =   row address (0 .. 131)
//                y0     =   column address  (0 .. 131)
//                radius =   radius in pixels
//                color  =   12-bit color value  rrrrggggbbbb
//
//      Returns:   nothing
//
//      Author:    Jack Bresenham IBM, Winthrop University (Father of this algorithm, 1962)
//
//                  Note: taken verbatim Wikipedia article on Bresenham's line algorithm
//                        http://www.wikipedia.org
//
//  ********************************************************************************

void LCDSetCircle(int x0, int y0, int radius, int color) {
        int f = 1 - radius;
        int ddF_x = 0;
        int ddF_y = -2 * radius;
        int x = 0;
        int y = radius;

        LCDSetPixel(x0, y0 + radius, color);
        LCDSetPixel(x0, y0 - radius, color);
        LCDSetPixel(x0 + radius, y0, color);
        LCDSetPixel(x0 - radius, y0, color);

        While (x < y) {
                if (f >= 0) {
                        y--;
                        ddF_y += 2;
                        f += ddF_y;
                }
                x++;
                ddF_x += 2;
                f += ddF_x + 1;
                LCDSetPixel(x0 + x, y0 + y, color);
                LCDSetPixel(x0 - x, y0 + y, color);
                LCDSetPixel(x0 + x, y0 - y, color);
                LCDSetPixel(x0 - x, y0 - y, color);
                LCDSetPixel(x0 + y, y0 + x, color);
                LCDSetPixel(x0 - y, y0 + x, color);
                LCDSetPixel(x0 + y, y0 - x, color);
                LCDSetPixel(x0 - y, y0 - x, color);
        }
}
```

```
//  ***************************************************************************
//                                      LCDPutChar.c
//
//     Draws an ASCII character at the specified (x,y) address and color
//
//        Inputs:       c       =   character to be displayed
//                      x       =   row address (0 .. 131)
//                      y       =   column address  (0 .. 131)
//                      size    =   font pitch (SMALL, MEDIUM, LARGE)
//                     fcolor  =   12-bit foreground color value         rrrrggggbbbb
//                     bcolor  =   12-bit background color value         rrrrggggbbbb
//
//
//        Returns:   nothing
//
//
//        Notes:  Here's an example to display "E" at address (20,20)
//
//                    LCDPutChar('E', 20, 20, MEDIUM, WHITE, BLACK);
//
//                                (27,20)        (27,27)
//                                   |              |
//                                   |              |
//                                ^  V              V
//                                :  _ # # # # # # #   0x7F
//                                :  _ _ # # _ _ _ #   0x31
//                                :  _ _ # # _ # _ _   0x34
//                                x  _ _ # # # # _ _   0x3C
//                                :  _ _ # # _ # _ _   0x34
//                                :  _ _ # # _ _ _ #   0x31
//                                :  _ # # # # # # #   0x7F
//                                :  _ _ _ _ _ _ _ _   0x00
//
//                                   ------y------->
//                                   ^              ^
//                                   |              |
//                                   |              |
//                                (20,20)       (20,27)
//
//
//     The most efficient way to display a character is to make use of the "wrap-around" feature
//     of the Philips PCF8833 LCD controller chip.
//
//     Assume that we position the character at (20, 20)  that's a  (row, col) specification.
//     With the row and column address set commands, you can specify an 8x8 box for the SMALL and MEDIUM
//     characters or a 16x8 box for the LARGE characters.
//
//                    WriteSpiCommand(PASET);              // set the row drawing limits
//                    WriteSpiData(20);                    //
//                    WriteSpiData(27);                    // limit rows to (20, 27)
//
//                    WriteSpiCommand(CASET);              // set the column drawing limits
//                    WriteSpiData(20);                    //
//                    WriteSpiData(27);                    // limit columns to (20,27)
//
//     When the algorithm completes col 27, the column address wraps back to 20
//     At the same time, the row address increases by one (this is done by the controller)
//
//     We walk through each row, two pixels at a time. The purpose is to create three
//     data bytes representing these two pixels in the following format (as specified by Philips
//     for RGB 4 : 4 : 4 format (see page 62 of PCF8833 controller manual).
//
//                    Data for pixel 0:  RRRRGGGGBBBB
//                    Data for Pixel 1:  RRRRGGGGBBBB
//
//                    WriteSpiCommand(RAMWR);              // start a memory write (96 data bytes to follow)
//
//                    WriteSpiData(RRRRGGGG);              // first pixel, red and green data
//                    WriteSpiData(BBBBRRRR);              // first pixel, blue data; second pixel, red data
//                    WriteSpiData(GGGGBBBB);              // second pixel, green and blue data
//                           :
//     and so on until all pixels displayed!
//                           :
//                    WriteSpiCommand(NOP);                // this will terminate the RAMWR command
//
//
//     Author:  James P Lynch    July 7, 2007
//  ***************************************************************************
```

```
void LCDPutChar(char c, int  x, int  y, int size, int fColor, int bColor) {

        extern const unsigned char FONT6x8[97][8];
        extern const unsigned char FONT8x8[97][8];
        extern const unsigned char FONT8x16[97][16];

        int                     i,j;
        unsigned int            nCols;
        unsigned int            nRows;
        unsigned int            nBytes;
        unsigned char           PixelRow;
        unsigned char           Mask;
        unsigned int            Word0;
        unsigned int            Word1;
        unsigned char           *pFont;

        unsigned char           *pChar;
        unsigned char           *FontTable[] = {(unsigned char *)FONT6x8,
                                                (unsigned char *)FONT8x8,
                                                (unsigned char *)FONT8x16};

        // get pointer to the beginning of the selected font table
        pFont = (unsigned char *)FontTable[size];

        // get the nColumns, nRows and nBytes
        nCols = *pFont;
        nRows = *(pFont + 1);
        nBytes = *(pFont + 2);

        // get pointer to the last byte of the desired character
        pChar = pFont + (nBytes * (c - 0x1F)) + nBytes - 1;

        // Row address set  (command 0x2B)
        WriteSpiCommand(PASET);
        WriteSpiData(x);
        WriteSpiData(x + nRows - 1);

        // Column address set  (command 0x2A)
        WriteSpiCommand(CASET);
        WriteSpiData(y);
        WriteSpiData(y + nCols - 1);

        // WRITE MEMORY
        WriteSpiCommand(RAMWR);

        // loop on each row, working backwards from the bottom to the top
        for (i = nRows - 1; i >= 0; i--) {

                // copy pixel row from font table and then decrement row
                PixelRow = *pChar--;

                // loop on each pixel in the row (left to right)
                // Note: we do two pixels each loop
                Mask = 0x80;
                for (j = 0; j < nCols; j += 2) {

                        // if pixel bit set, use foreground color; else use the background color
                        // now get the pixel color for two successive pixels
                        if ((PixelRow & Mask) == 0)
                                Word0 = bColor;
                        else
                                Word0 = fColor;
                        Mask = Mask >> 1;
                        if ((PixelRow & Mask) == 0)
                                Word1 = bColor;
                        else
                                Word1 = fColor;
                        Mask = Mask >> 1;

                        // use this information to output three data bytes
                        WriteSpiData((Word0 >> 4) & 0xFF);
                        WriteSpiData(((Word0 & 0xF) << 4) | ((Word1 >> 8) & 0xF));
                        WriteSpiData(Word1 & 0xFF);
                }
        }
        // terminate the Write Memory command
        WriteSpiCommand(NOP);
}
```

```
//  ***************************************************************************************************
//                                          LCDPutStr.c
//
//     Draws a null-terminates character string at the specified (x,y) address, size and color
//
//        Inputs:       pString =   pointer to character string to be displayed
//                      x       =   row address (0 .. 131)
//                      y       =   column address  (0 .. 131)
//                      Size    =   font pitch (SMALL, MEDIUM, LARGE)
//                      fColor  =   12-bit foreground color value   rrrrggggbbbb
//                      bColor  =   12-bit background color value   rrrrggggbbbb
//
//
//        Returns:   nothing
//
//  Notes:  Here's an example to display "Hello World!" at address (20,20)
//
//          LCDPutChar("Hello World!", 20, 20, LARGE, WHITE, BLACK);
//
//
//     Author:  James P Lynch    July 7, 2007
//  ***************************************************************************************************
void LCDPutStr(char *pString, int  x, int  y, int Size, int fColor, int bColor) {

       // loop until null-terminator is seen
       while (*pString != 0x00) {

              // draw the character
              LCDPutChar(*pString++, x, y, Size, fColor, bColor);

              // advance the y position
              if (Size == SMALL)
                     y = y + 6;
              else if (Size == MEDIUM)
                     y = y + 8;
              else
                     y = y + 8;

              // bail out if y exceeds 131
              if (y > 131) break;
       }
}


//  ***************************************************************************
//                                      Delay.c
//
//            Simple for loop delay
//
//            Inputs: a  -  loop count
//
//            Author:  James P Lynch  June 27, 2007
//  ***************************************************************************
void Delay (unsigned long a) {
       while (--a!=0);
}
```

```
//  ***************************************************************************
//            Font tables for Nokia 6610 LCD Display Driver  (PCF8833 Controller)
//
//            FONT6x8        -  SMALL font (mostly 5x7)
//            FONT8x8        -  MEDIUM font (8x8 characters, a bit thicker)
//            FONT8x16       -  LARGE font  (8x16 characters, thicker)
//
//            Note:  ASCII characters 0x00 through 0x1F are not included in these fonts.
//                   First row of each font contains the number of columns, the
//                   number of rows and the number of bytes per character.
//
//  Author:  Jimbo of Spark Fun, James P Lynch  July 7, 2007
//  ***************************************************************************
const unsigned char FONT6x8[97][8] = {

0x06,0x08,0x08,0x00,0x00,0x00,0x00,0x00,     //  columns, rows, num_bytes_per_char
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,     //     space   0x20
0x20,0x20,0x20,0x20,0x20,0x00,0x20,0x00,     //     !
0x50,0x50,0x50,0x00,0x00,0x00,0x00,0x00,     //     "
0x50,0x50,0xF8,0x50,0xF8,0x50,0x50,0x00,     //     #
0x20,0x78,0xA0,0x70,0x28,0xF0,0x20,0x00,     //     $
0xC0,0xC8,0x10,0x20,0x40,0x98,0x18,0x00,     //     %
0x40,0xA0,0xA0,0x40,0xA8,0x90,0x68,0x00,     //     &
0x30,0x30,0x20,0x40,0x00,0x00,0x00,0x00,     //     '
0x10,0x20,0x40,0x40,0x40,0x20,0x10,0x00,     //     (
0x40,0x20,0x10,0x10,0x10,0x20,0x40,0x00,     //     )
0x00,0x20,0xA8,0x70,0x70,0xA8,0x20,0x00,     //     *
0x00,0x20,0x20,0xF8,0x20,0x20,0x00,0x00,     //     +
0x00,0x00,0x00,0x00,0x30,0x30,0x20,0x40,     //     ,
0x00,0x00,0x00,0xF8,0x00,0x00,0x00,0x00,     //     -
0x00,0x00,0x00,0x00,0x00,0x30,0x30,0x00,     //     .
0x00,0x08,0x10,0x20,0x40,0x80,0x00,0x00,     //     / (forward slash)
0x70,0x88,0x88,0xA8,0x88,0x88,0x70,0x00,     //     0  0x30
0x20,0x60,0x20,0x20,0x20,0x20,0x70,0x00,     //     1
0x70,0x88,0x08,0x70,0x80,0x80,0xF8,0x00,     //     2
0xF8,0x08,0x10,0x30,0x08,0x88,0x70,0x00,     //     3
0x10,0x30,0x50,0x90,0xF8,0x10,0x10,0x00,     //     4
0xF8,0x80,0xF0,0x08,0x08,0x88,0x70,0x00,     //     5
0x38,0x40,0x80,0xF0,0x88,0x88,0x70,0x00,     //     6
0xF8,0x08,0x08,0x10,0x20,0x40,0x80,0x00,     //     7
0x70,0x88,0x88,0x70,0x88,0x88,0x70,0x00,     //     8
0x70,0x88,0x88,0x78,0x08,0x10,0xE0,0x00,     //     9
0x00,0x00,0x20,0x00,0x20,0x00,0x00,0x00,     //     :
0x00,0x00,0x20,0x00,0x20,0x20,0x40,0x00,     //     ;
0x08,0x10,0x20,0x40,0x20,0x10,0x08,0x00,     //     <
0x00,0x00,0xF8,0x00,0xF8,0x00,0x00,0x00,     //     =
0x40,0x20,0x10,0x08,0x10,0x20,0x40,0x00,     //     >
0x70,0x88,0x08,0x30,0x20,0x00,0x20,0x00,     //     ?
0x70,0x88,0xA8,0xB8,0xB0,0x80,0x78,0x00,     //     @  0x40
0x20,0x50,0x88,0x88,0xF8,0x88,0x88,0x00,     //     A
0xF0,0x88,0x88,0xF0,0x88,0x88,0xF0,0x00,     //     B
0x70,0x88,0x80,0x80,0x80,0x88,0x70,0x00,     //     C
0xF0,0x88,0x88,0x88,0x88,0x88,0xF0,0x00,     //     D
0xF8,0x80,0x80,0xF0,0x80,0x80,0xF8,0x00,     //     E
0xF8,0x80,0x80,0xF0,0x80,0x80,0x80,0x00,     //     F
0x78,0x88,0x80,0x80,0x98,0x88,0x78,0x00,     //     G
0x88,0x88,0x88,0xF8,0x88,0x88,0x88,0x00,     //     H
0x70,0x20,0x20,0x20,0x20,0x20,0x70,0x00,     //     I
0x38,0x10,0x10,0x10,0x10,0x90,0x60,0x00,     //     J
0x88,0x90,0xA0,0xC0,0xA0,0x90,0x88,0x00,     //     K
0x80,0x80,0x80,0x80,0x80,0x80,0xF8,0x00,     //     L
0x88,0xD8,0xA8,0xA8,0xA8,0x88,0x88,0x00,     //     M
0x88,0x88,0xC8,0xA8,0x98,0x88,0x88,0x00,     //     N
0x70,0x88,0x88,0x88,0x88,0x88,0x70,0x00,     //     O
0xF0,0x88,0x88,0xF0,0x80,0x80,0x80,0x00,     //     P  0x50
0x70,0x88,0x88,0x88,0xA8,0x90,0x68,0x00,     //     Q
0xF0,0x88,0x88,0xF0,0xA0,0x90,0x88,0x00,     //     R
0x70,0x88,0x80,0x70,0x08,0x88,0x70,0x00,     //     S
0xF8,0xA8,0x20,0x20,0x20,0x20,0x20,0x00,     //     T
0x88,0x88,0x88,0x88,0x88,0x88,0x70,0x00,     //     U
0x88,0x88,0x88,0x88,0x88,0x50,0x20,0x00,     //     V
0x88,0x88,0x88,0xA8,0xA8,0xA8,0x50,0x00,     //     W
0x88,0x88,0x50,0x20,0x50,0x88,0x88,0x00,     //     X
0x88,0x88,0x50,0x20,0x20,0x20,0x20,0x00,     //     Y
0xF8,0x08,0x10,0x70,0x40,0x80,0xF8,0x00,     //     Z
0x78,0x40,0x40,0x40,0x40,0x40,0x78,0x00,     //     [
0x00,0x80,0x40,0x20,0x10,0x08,0x00,0x00,     //     \  (back slash)
0x78,0x08,0x08,0x08,0x08,0x08,0x78,0x00,     //     ]
```

```
0x20,0x50,0x88,0x00,0x00,0x00,0x00,0x00,     //     ^
0x00,0x00,0x00,0x00,0x00,0x00,0xF8,0x00,     //     _
0x60,0x60,0x20,0x10,0x00,0x00,0x00,0x00,     //     `  0x60
0x00,0x00,0x60,0x10,0x70,0x90,0x78,0x00,     //     a
0x80,0x80,0xB0,0xC8,0x88,0xC8,0xB0,0x00,     //     b
0x00,0x00,0x70,0x88,0x80,0x88,0x70,0x00,     //     c
0x08,0x08,0x68,0x98,0x88,0x98,0x68,0x00,     //     d
0x00,0x00,0x70,0x88,0xF8,0x80,0x70,0x00,     //     e
0x10,0x28,0x20,0x70,0x20,0x20,0x20,0x00,     //     f
0x00,0x00,0x70,0x98,0x98,0x68,0x08,0x70,     //     g
0x80,0x80,0xB0,0xC8,0x88,0x88,0x88,0x00,     //     h
0x20,0x00,0x60,0x20,0x20,0x20,0x70,0x00,     //     i
0x10,0x00,0x10,0x10,0x10,0x90,0x60,0x00,     //     j
0x80,0x80,0x90,0xA0,0xC0,0xA0,0x90,0x00,     //     k
0x60,0x20,0x20,0x20,0x20,0x20,0x70,0x00,     //     l
0x00,0x00,0xD0,0xA8,0xA8,0xA8,0xA8,0x00,     //     m
0x00,0x00,0xB0,0xC8,0x88,0x88,0x88,0x00,     //     n
0x00,0x00,0x70,0x88,0x88,0x88,0x70,0x00,     //     o
0x00,0x00,0xB0,0xC8,0xC8,0xB0,0x80,0x80,     //     p  0x70
0x00,0x00,0x68,0x98,0x98,0x68,0x08,0x08,     //     q
0x00,0x00,0xB0,0xC8,0x80,0x80,0x80,0x00,     //     r
0x00,0x00,0x78,0x80,0x70,0x08,0xF0,0x00,     //     s
0x20,0x20,0xF8,0x20,0x20,0x28,0x10,0x00,     //     t
0x00,0x00,0x88,0x88,0x88,0x98,0x68,0x00,     //     u
0x00,0x00,0x88,0x88,0x88,0x50,0x20,0x00,     //     v
0x00,0x00,0x88,0x88,0xA8,0xA8,0x50,0x00,     //     w
0x00,0x00,0x88,0x50,0x20,0x50,0x88,0x00,     //     x
0x00,0x00,0x88,0x88,0x78,0x08,0x88,0x70,     //     y
0x00,0x00,0xF8,0x10,0x20,0x40,0xF8,0x00,     //     z
0x10,0x20,0x20,0x40,0x20,0x20,0x10,0x00,     //     {
0x20,0x20,0x20,0x00,0x20,0x20,0x20,0x00,     //     |
0x40,0x20,0x20,0x10,0x20,0x20,0x40,0x00,     //     }
0x40,0xA8,0x10,0x00,0x00,0x00,0x00,0x00,     //     ~
0x70,0xD8,0xD8,0x70,0x00,0x00,0x00,0x00};    //     DEL

const unsigned char FONT8x8[97][8] = {

0x08,0x08,0x08,0x00,0x00,0x00,0x00,0x00,     //  columns, rows, num_bytes_per_char
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,     //     space  0x20
0x30,0x78,0x78,0x30,0x30,0x00,0x30,0x00,     //     !
0x6C,0x6C,0x6C,0x00,0x00,0x00,0x00,0x00,     //     "
0x6C,0x6C,0xFE,0x6C,0xFE,0x6C,0x6C,0x00,     //     #
0x18,0x3E,0x60,0x3C,0x06,0x7C,0x18,0x00,     //     $
0x00,0x63,0x66,0x0C,0x18,0x33,0x63,0x00,     //     %
0x1C,0x36,0x1C,0x3B,0x6E,0x66,0x3B,0x00,     //     &
0x30,0x30,0x60,0x00,0x00,0x00,0x00,0x00,     //     '
0x0C,0x18,0x30,0x30,0x30,0x18,0x0C,0x00,     //     (
0x30,0x18,0x0C,0x0C,0x0C,0x18,0x30,0x00,     //     )
0x00,0x66,0x3C,0xFF,0x3C,0x66,0x00,0x00,     //     *
0x00,0x30,0x30,0xFC,0x30,0x30,0x00,0x00,     //     +
0x00,0x00,0x00,0x00,0x00,0x18,0x18,0x30,     //     ,
0x00,0x00,0x00,0x7E,0x00,0x00,0x00,0x00,     //     -
0x00,0x00,0x00,0x00,0x00,0x18,0x18,0x00,     //     .
0x03,0x06,0x0C,0x18,0x30,0x60,0x40,0x00,     //     / (forward slash)
0x3E,0x63,0x63,0x6B,0x63,0x63,0x3E,0x00,     //     0  0x30
0x18,0x38,0x58,0x18,0x18,0x18,0x7E,0x00,     //     1
0x3C,0x66,0x06,0x1C,0x30,0x66,0x7E,0x00,     //     2
0x3C,0x66,0x06,0x1C,0x06,0x66,0x3C,0x00,     //     3
0x0E,0x1E,0x36,0x66,0x7F,0x06,0x0F,0x00,     //     4
0x7E,0x60,0x7C,0x06,0x06,0x66,0x3C,0x00,     //     5
0x1C,0x30,0x60,0x7C,0x66,0x66,0x3C,0x00,     //     6
0x7E,0x66,0x06,0x0C,0x18,0x18,0x18,0x00,     //     7
0x3C,0x66,0x66,0x3C,0x66,0x66,0x3C,0x00,     //     8
0x3C,0x66,0x66,0x3E,0x06,0x0C,0x38,0x00,     //     9
0x00,0x18,0x18,0x00,0x00,0x18,0x18,0x00,     //     :
0x00,0x18,0x18,0x00,0x00,0x18,0x18,0x30,     //     ;
0x0C,0x18,0x30,0x60,0x30,0x18,0x0C,0x00,     //     <
0x00,0x00,0x7E,0x00,0x00,0x7E,0x00,0x00,     //     =
0x30,0x18,0x0C,0x06,0x0C,0x18,0x30,0x00,     //     >
0x3C,0x66,0x06,0x0C,0x18,0x00,0x18,0x00,     //     ?
0x3E,0x63,0x6F,0x69,0x6F,0x60,0x3E,0x00,     //     @  0x40
0x18,0x3C,0x66,0x66,0x7E,0x66,0x66,0x00,     //     A
0x7E,0x33,0x33,0x3E,0x33,0x33,0x7E,0x00,     //     B
0x1E,0x33,0x60,0x60,0x60,0x33,0x1E,0x00,     //     C
0x7C,0x36,0x33,0x33,0x33,0x36,0x7C,0x00,     //     D
0x7F,0x31,0x34,0x3C,0x34,0x31,0x7F,0x00,     //     E
0x7F,0x31,0x34,0x3C,0x34,0x30,0x78,0x00,     //     F
0x1E,0x33,0x60,0x60,0x67,0x33,0x1F,0x00,     //     G
```

```
0x66,0x66,0x66,0x7E,0x66,0x66,0x66,0x00,     //     H
0x3C,0x18,0x18,0x18,0x18,0x18,0x3C,0x00,     //     I
0x0F,0x06,0x06,0x06,0x66,0x66,0x3C,0x00,     //     J
0x73,0x33,0x36,0x3C,0x36,0x33,0x73,0x00,     //     K
0x78,0x30,0x30,0x30,0x31,0x33,0x7F,0x00,     //     L
0x63,0x77,0x7F,0x7F,0x6B,0x63,0x63,0x00,     //     M
0x63,0x73,0x7B,0x6F,0x67,0x63,0x63,0x00,     //     N
0x3E,0x63,0x63,0x63,0x63,0x63,0x3E,0x00,     //     O
0x7E,0x33,0x33,0x3E,0x30,0x30,0x78,0x00,     //     P  0x50
0x3C,0x66,0x66,0x66,0x6E,0x3C,0x0E,0x00,     //     Q
0x7E,0x33,0x33,0x3E,0x36,0x33,0x73,0x00,     //     R
0x3C,0x66,0x30,0x18,0x0C,0x66,0x3C,0x00,     //     S
0x7E,0x5A,0x18,0x18,0x18,0x18,0x3C,0x00,     //     T
0x66,0x66,0x66,0x66,0x66,0x66,0x7E,0x00,     //     U
0x66,0x66,0x66,0x66,0x66,0x3C,0x18,0x00,     //     V
0x63,0x63,0x63,0x6B,0x7F,0x77,0x63,0x00,     //     W
0x63,0x63,0x36,0x1C,0x1C,0x36,0x63,0x00,     //     X
0x66,0x66,0x66,0x3C,0x18,0x18,0x3C,0x00,     //     Y
0x7F,0x63,0x46,0x0C,0x19,0x33,0x7F,0x00,     //     Z
0x3C,0x30,0x30,0x30,0x30,0x30,0x3C,0x00,     //     [
0x60,0x30,0x18,0x0C,0x06,0x03,0x01,0x00,     //     \  (back slash)
0x3C,0x0C,0x0C,0x0C,0x0C,0x0C,0x3C,0x00,     //     ]
0x08,0x1C,0x36,0x63,0x00,0x00,0x00,0x00,     //     ^
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xFF,     //
0x18,0x18,0x0C,0x00,0x00,0x00,0x00,0x00,     //     `  0x60
0x00,0x00,0x3C,0x06,0x3E,0x66,0x3B,0x00,     //     a
0x70,0x30,0x3E,0x33,0x33,0x33,0x6E,0x00,     //     b
0x00,0x00,0x3C,0x66,0x60,0x66,0x3C,0x00,     //     c
0x0E,0x06,0x3E,0x66,0x66,0x66,0x3B,0x00,     //     d
0x00,0x00,0x3C,0x66,0x7E,0x60,0x3C,0x00,     //     e
0x1C,0x36,0x30,0x78,0x30,0x30,0x78,0x00,     //     f
0x00,0x00,0x3B,0x66,0x66,0x3E,0x06,0x7C,     //     g
0x70,0x30,0x36,0x3B,0x33,0x33,0x73,0x00,     //     h
0x18,0x00,0x38,0x18,0x18,0x18,0x3C,0x00,     //     i
0x06,0x00,0x06,0x06,0x06,0x66,0x66,0x3C,     //     j
0x70,0x30,0x33,0x36,0x3C,0x36,0x73,0x00,     //     k
0x38,0x18,0x18,0x18,0x18,0x18,0x3C,0x00,     //     l
0x00,0x00,0x66,0x7F,0x7F,0x6B,0x63,0x00,     //     m
0x00,0x00,0x7C,0x66,0x66,0x66,0x66,0x00,     //     n
0x00,0x00,0x3C,0x66,0x66,0x66,0x3C,0x00,     //     o
0x00,0x00,0x6E,0x33,0x33,0x3E,0x30,0x78,     //     p  0x70
0x00,0x00,0x3B,0x66,0x66,0x3E,0x06,0x0F,     //     q
0x00,0x00,0x6E,0x3B,0x33,0x30,0x78,0x00,     //     r
0x00,0x00,0x3E,0x60,0x3C,0x06,0x7C,0x00,     //     s
0x08,0x18,0x3E,0x18,0x18,0x1A,0x0C,0x00,     //     t
0x00,0x00,0x66,0x66,0x66,0x66,0x3B,0x00,     //     u
0x00,0x00,0x66,0x66,0x66,0x3C,0x18,0x00,     //     v
0x00,0x00,0x63,0x6B,0x7F,0x7F,0x36,0x00,     //     w
0x00,0x00,0x63,0x36,0x1C,0x36,0x63,0x00,     //     x
0x00,0x00,0x66,0x66,0x66,0x3E,0x06,0x7C,     //     y
0x00,0x00,0x7E,0x4C,0x18,0x32,0x7E,0x00,     //     z
0x0E,0x18,0x18,0x70,0x18,0x18,0x0E,0x00,     //     {
0x0C,0x0C,0x0C,0x00,0x0C,0x0C,0x0C,0x00,     //     |
0x70,0x18,0x18,0x0E,0x18,0x18,0x70,0x00,     //     }
0x3B,0x6E,0x00,0x00,0x00,0x00,0x00,0x00,     //     ~
0x1C,0x36,0x36,0x1C,0x00,0x00,0x00,0x00};    //     DEL


const unsigned char FONT8x16[97][16] = {

0x08,0x10,0x10,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,  //  columns, rows, nbytes
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,  //     space  0x20
0x00,0x00,0x18,0x3C,0x3C,0x3C,0x18,0x18,0x18,0x00,0x18,0x18,0x00,0x00,0x00,0x00,  //     !
0x00,0x63,0x63,0x63,0x22,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,  //     "
0x00,0x00,0x00,0x36,0x36,0x7F,0x36,0x36,0x36,0x7F,0x36,0x36,0x00,0x00,0x00,0x00,  //     #
0x0C,0x0C,0x3E,0x63,0x61,0x60,0x3E,0x03,0x03,0x43,0x63,0x3E,0x0C,0x0C,0x00,0x00,  //     $
0x00,0x00,0x00,0x00,0x00,0x61,0x63,0x06,0x0C,0x18,0x33,0x63,0x00,0x00,0x00,0x00,  //     %
0x00,0x00,0x00,0x00,0x1C,0x36,0x36,0x1C,0x3B,0x6E,0x66,0x66,0x3B,0x00,0x00,0x00,  //     &
0x00,0x30,0x30,0x30,0x60,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,  //     '
0x00,0x00,0x0C,0x18,0x18,0x30,0x30,0x30,0x30,0x18,0x18,0x0C,0x00,0x00,0x00,0x00,  //     (
0x00,0x00,0x18,0x0C,0x0C,0x06,0x06,0x06,0x06,0x0C,0x0C,0x18,0x00,0x00,0x00,0x00,  //     )
0x00,0x00,0x00,0x00,0x42,0x66,0x3C,0xFF,0x3C,0x66,0x42,0x00,0x00,0x00,0x00,0x00,  //     *
0x00,0x00,0x00,0x00,0x18,0x18,0x18,0xFF,0x18,0x18,0x18,0x00,0x00,0x00,0x00,0x00,  //     +
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x18,0x18,0x18,0x30,0x00,  //     ,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xFF,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,  //     -
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x18,0x18,0x00,0x00,0x00,0x00,  //     .
0x00,0x00,0x01,0x03,0x07,0x0E,0x1C,0x38,0x70,0xE0,0xC0,0x80,0x00,0x00,0x00,0x00,  //     / (forward slash)
0x00,0x00,0x3E,0x63,0x63,0x63,0x6B,0x6B,0x63,0x63,0x63,0x3E,0x00,0x00,0x00,0x00,  //     0  0x30
0x00,0x00,0x0C,0x1C,0x3C,0x0C,0x0C,0x0C,0x0C,0x0C,0x0C,0x3F,0x00,0x00,0x00,0x00,  //     1
```

```
0x00,0x00,0x3E,0x63,0x03,0x06,0x0C,0x18,0x30,0x61,0x63,0x7F,0x00,0x00,0x00,0x00,  //     2
0x00,0x00,0x3E,0x63,0x03,0x03,0x1E,0x03,0x03,0x03,0x63,0x3E,0x00,0x00,0x00,0x00,  //     3
0x00,0x00,0x06,0x0E,0x1E,0x36,0x66,0x66,0x7F,0x06,0x06,0x0F,0x00,0x00,0x00,0x00,  //     4
0x00,0x00,0x7F,0x60,0x60,0x60,0x7E,0x03,0x03,0x63,0x73,0x3E,0x00,0x00,0x00,0x00,  //     5
0x00,0x00,0x1C,0x30,0x60,0x60,0x7E,0x63,0x63,0x63,0x63,0x3E,0x00,0x00,0x00,0x00,  //     6
0x00,0x00,0x7F,0x63,0x03,0x06,0x06,0x0C,0x0C,0x18,0x18,0x18,0x00,0x00,0x00,0x00,  //     7
0x00,0x00,0x3E,0x63,0x63,0x63,0x3E,0x63,0x63,0x63,0x63,0x3E,0x00,0x00,0x00,0x00,  //     8
0x00,0x00,0x3E,0x63,0x63,0x63,0x63,0x3F,0x03,0x03,0x06,0x3C,0x00,0x00,0x00,0x00,  //     9
0x00,0x00,0x00,0x00,0x00,0x18,0x18,0x00,0x00,0x00,0x18,0x18,0x00,0x00,0x00,0x00,  //     :
0x00,0x00,0x00,0x00,0x00,0x18,0x18,0x00,0x00,0x00,0x18,0x18,0x18,0x30,0x00,0x00,  //     ;
0x00,0x00,0x00,0x06,0x0C,0x18,0x30,0x60,0x30,0x18,0x0C,0x06,0x00,0x00,0x00,0x00,  //     <
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x7E,0x00,0x00,0x7E,0x00,0x00,0x00,0x00,0x00,  //     =
0x00,0x00,0x00,0x00,0x60,0x30,0x18,0x0C,0x06,0x0C,0x18,0x30,0x60,0x00,0x00,0x00,  //     >
0x00,0x00,0x3E,0x63,0x63,0x06,0x0C,0x0C,0x0C,0x00,0x0C,0x0C,0x00,0x00,0x00,0x00,  //     ?
0x00,0x00,0x3E,0x63,0x63,0x6F,0x6B,0x6B,0x6E,0x60,0x60,0x3E,0x00,0x00,0x00,0x00,  //     @  0x40
0x00,0x00,0x08,0x1C,0x36,0x63,0x63,0x63,0x7F,0x63,0x63,0x63,0x00,0x00,0x00,0x00,  //     A
0x00,0x00,0x7E,0x33,0x33,0x33,0x3E,0x33,0x33,0x33,0x33,0x7E,0x00,0x00,0x00,0x00,  //     B
0x00,0x00,0x1E,0x33,0x61,0x60,0x60,0x60,0x60,0x61,0x33,0x1E,0x00,0x00,0x00,0x00,  //     C
0x00,0x00,0x7C,0x36,0x33,0x33,0x33,0x33,0x33,0x33,0x36,0x7C,0x00,0x00,0x00,0x00,  //     D
0x00,0x00,0x7F,0x33,0x31,0x34,0x3C,0x34,0x30,0x31,0x33,0x7F,0x00,0x00,0x00,0x00,  //     E
0x00,0x00,0x7F,0x33,0x31,0x34,0x3C,0x34,0x30,0x30,0x30,0x78,0x00,0x00,0x00,0x00,  //     F
0x00,0x00,0x1E,0x33,0x61,0x60,0x60,0x6F,0x63,0x63,0x37,0x1D,0x00,0x00,0x00,0x00,  //     G
0x00,0x00,0x63,0x63,0x63,0x63,0x7F,0x63,0x63,0x63,0x63,0x63,0x00,0x00,0x00,0x00,  //     H
0x00,0x00,0x3C,0x18,0x18,0x18,0x18,0x18,0x18,0x18,0x18,0x3C,0x00,0x00,0x00,0x00,  //     I
0x00,0x00,0x0F,0x06,0x06,0x06,0x06,0x06,0x06,0x66,0x66,0x3C,0x00,0x00,0x00,0x00,  //     J
0x00,0x00,0x73,0x33,0x36,0x36,0x3C,0x36,0x36,0x33,0x33,0x73,0x00,0x00,0x00,0x00,  //     K
0x00,0x00,0x78,0x30,0x30,0x30,0x30,0x30,0x30,0x31,0x33,0x7F,0x00,0x00,0x00,0x00,  //     L
0x00,0x00,0x63,0x77,0x7F,0x6B,0x63,0x63,0x63,0x63,0x63,0x63,0x00,0x00,0x00,0x00,  //     M
0x00,0x00,0x63,0x63,0x73,0x7B,0x7F,0x6F,0x67,0x63,0x63,0x63,0x00,0x00,0x00,0x00,  //     N
0x00,0x00,0x1C,0x36,0x63,0x63,0x63,0x63,0x63,0x63,0x36,0x1C,0x00,0x00,0x00,0x00,  //     O
0x00,0x00,0x7E,0x33,0x33,0x33,0x3E,0x30,0x30,0x30,0x30,0x78,0x00,0x00,0x00,0x00,  //     P  0x50
0x00,0x00,0x3E,0x63,0x63,0x63,0x63,0x63,0x63,0x6B,0x6F,0x3E,0x06,0x07,0x00,0x00,  //     Q
0x00,0x00,0x7E,0x33,0x33,0x33,0x3E,0x36,0x36,0x33,0x33,0x73,0x00,0x00,0x00,0x00,  //     R
0x00,0x00,0x3E,0x63,0x63,0x30,0x1C,0x06,0x03,0x63,0x63,0x3E,0x00,0x00,0x00,0x00,  //     S
0x00,0x00,0xFF,0xDB,0x99,0x18,0x18,0x18,0x18,0x18,0x18,0x3C,0x00,0x00,0x00,0x00,  //     T
0x00,0x00,0x63,0x63,0x63,0x63,0x63,0x63,0x63,0x63,0x63,0x3E,0x00,0x00,0x00,0x00,  //     U
0x00,0x00,0x63,0x63,0x63,0x63,0x63,0x63,0x63,0x36,0x1C,0x08,0x00,0x00,0x00,0x00,  //     V
0x00,0x00,0x63,0x63,0x63,0x63,0x63,0x6B,0x6B,0x7F,0x36,0x36,0x00,0x00,0x00,0x00,  //     W
0x00,0x00,0xC3,0xC3,0x66,0x3C,0x18,0x18,0x3C,0x66,0xC3,0xC3,0x00,0x00,0x00,0x00,  //     X
0x00,0x00,0xC3,0xC3,0xC3,0x66,0x3C,0x18,0x18,0x18,0x18,0x3C,0x00,0x00,0x00,0x00,  //     Y
0x00,0x00,0x7F,0x63,0x43,0x06,0x0C,0x18,0x30,0x61,0x63,0x7F,0x00,0x00,0x00,0x00,  //     Z
0x00,0x00,0x3C,0x30,0x30,0x30,0x30,0x30,0x30,0x30,0x30,0x3C,0x00,0x00,0x00,0x00,  //     [
0x00,0x00,0x80,0xC0,0xE0,0x70,0x38,0x1C,0x0E,0x07,0x03,0x01,0x00,0x00,0x00,0x00,  //     \  (back slash)
0x00,0x00,0x3C,0x0C,0x0C,0x0C,0x0C,0x0C,0x0C,0x0C,0x0C,0x3C,0x00,0x00,0x00,0x00,  //     ]
0x08,0x1C,0x36,0x63,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,  //     ^
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xFF,0x00,0x00,  //     _
0x18,0x18,0x0C,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,  //     `  0x60
0x00,0x00,0x00,0x00,0x00,0x3C,0x46,0x06,0x3E,0x66,0x66,0x3B,0x00,0x00,0x00,0x00,  //     a
0x00,0x00,0x70,0x30,0x30,0x3C,0x36,0x33,0x33,0x33,0x33,0x6E,0x00,0x00,0x00,0x00,  //     b
0x00,0x00,0x00,0x00,0x00,0x3E,0x63,0x60,0x60,0x60,0x63,0x3E,0x00,0x00,0x00,0x00,  //     c
0x00,0x00,0x0E,0x06,0x06,0x1E,0x36,0x66,0x66,0x66,0x66,0x3B,0x00,0x00,0x00,0x00,  //     d
0x00,0x00,0x00,0x00,0x00,0x3E,0x63,0x63,0x7E,0x60,0x63,0x3E,0x00,0x00,0x00,0x00,  //     e
0x00,0x00,0x1C,0x36,0x32,0x30,0x7C,0x30,0x30,0x30,0x30,0x78,0x00,0x00,0x00,0x00,  //     f
0x00,0x00,0x00,0x00,0x00,0x3B,0x66,0x66,0x66,0x66,0x3E,0x06,0x66,0x3C,0x00,0x00,  //     g
0x00,0x00,0x70,0x30,0x30,0x36,0x3B,0x33,0x33,0x33,0x33,0x73,0x00,0x00,0x00,0x00,  //     h
0x00,0x00,0x0C,0x0C,0x00,0x1C,0x0C,0x0C,0x0C,0x0C,0x0C,0x1E,0x00,0x00,0x00,0x00,  //     i
0x00,0x00,0x06,0x06,0x00,0x0E,0x06,0x06,0x06,0x06,0x06,0x66,0x66,0x3C,0x00,0x00,  //     j
0x00,0x00,0x70,0x30,0x30,0x33,0x33,0x36,0x3C,0x36,0x36,0x33,0x73,0x00,0x00,0x00,  //     k
0x00,0x00,0x1C,0x0C,0x0C,0x0C,0x0C,0x0C,0x0C,0x0C,0x0C,0x1E,0x00,0x00,0x00,0x00,  //     l
0x00,0x00,0x00,0x00,0x00,0x6E,0x7F,0x6B,0x6B,0x6B,0x6B,0x6B,0x00,0x00,0x00,0x00,  //     m
0x00,0x00,0x00,0x00,0x00,0x6E,0x33,0x33,0x33,0x33,0x33,0x33,0x00,0x00,0x00,0x00,  //     n
0x00,0x00,0x00,0x00,0x00,0x3E,0x63,0x63,0x63,0x63,0x63,0x3E,0x00,0x00,0x00,0x00,  //     o
0x00,0x00,0x00,0x00,0x00,0x6E,0x33,0x33,0x33,0x33,0x3E,0x30,0x30,0x78,0x00,0x00,  //     p  0x70
0x00,0x00,0x00,0x00,0x00,0x3B,0x66,0x66,0x66,0x66,0x3E,0x06,0x06,0x0F,0x00,0x00,  //     q
0x00,0x00,0x00,0x00,0x00,0x6E,0x3B,0x33,0x30,0x30,0x30,0x78,0x00,0x00,0x00,0x00,  //     r
0x00,0x00,0x00,0x00,0x00,0x3E,0x63,0x38,0x0E,0x03,0x63,0x3E,0x00,0x00,0x00,0x00,  //     s
0x00,0x00,0x08,0x18,0x18,0x7E,0x18,0x18,0x18,0x18,0x1B,0x0E,0x00,0x00,0x00,0x00,  //     t
0x00,0x00,0x00,0x00,0x00,0x66,0x66,0x66,0x66,0x66,0x66,0x3B,0x00,0x00,0x00,0x00,  //     u
0x00,0x00,0x00,0x00,0x00,0x63,0x63,0x36,0x36,0x1C,0x1C,0x08,0x00,0x00,0x00,0x00,  //     v
0x00,0x00,0x00,0x00,0x00,0x63,0x63,0x63,0x6B,0x6B,0x7F,0x36,0x00,0x00,0x00,0x00,  //     w




0x00,0x00,0x00,0x00,0x00,0x63,0x36,0x1C,0x1C,0x1C,0x36,0x63,0x00,0x00,0x00,0x00,  //     x
0x00,0x00,0x00,0x00,0x00,0x63,0x63,0x63,0x63,0x3F,0x03,0x06,0x3C,0x00,0x00,0x00,  //     y
0x00,0x00,0x00,0x00,0x00,0x7F,0x66,0x0C,0x18,0x30,0x63,0x7F,0x00,0x00,0x00,0x00,  //     z
```

```
0x00,0x00,0x0E,0x18,0x18,0x18,0x70,0x18,0x18,0x18,0x18,0x0E,0x00,0x00,0x00,0x00,  //     {
0x00,0x00,0x18,0x18,0x18,0x18,0x18,0x00,0x18,0x18,0x18,0x18,0x18,0x00,0x00,0x00,  //     |
0x00,0x00,0x70,0x18,0x18,0x18,0x0E,0x18,0x18,0x18,0x18,0x70,0x00,0x00,0x00,0x00,  //     }
0x00,0x00,0x3B,0x6E,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,  //     ~
0x00,0x70,0xD8,0xD8,0x70,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}; //     DEL
```

Sample Main Program Test Routine

The following is a simple main program that exercises every one of the LCD graphics primitives. After the tests have been completed, the main program falls into an endless blink loop. Since the SAM7-EX256 board has no user-programmable LED, the author added one as shown in the program's annotation.

```
//  ********************************************************************************************************
//                                        main.c
//
//     Nokia 6610 LCD demonstration program for Olimex SAM7-EX256 Evaluation Board
//
//     Performs a series of tests of the LCD driver.
//
//     When tests are complete, blinks LED4 (pin PA3) with an endless loop
//     PA3 is pin 1 on the EXT 20-pin connector (3.3v is pin 18)
//
//     The Olimex SAM7-EX256 board has no programmable LEDs.
//     Added a simple test LED from Radio Shack as shown below (attached to the 20-pin EXT connector.)
//
//                3.3 volts  |---------|           anode  |----|                PA3
//                EXT O------| 470 ohm |------------------|LED |-------------O EXT
//                Pin 18     |---------|                  |----| cathode        pin 1
//                                              Radio Shack Red LED
//                                              276-026 T-1 size (anode is the longer wire)
//
//          LED current:  I = E/R  = 3.3/470  =  .007 amps = 7 ma
//           Note: most PIO pins can drive 8 ma on the AT91SAM7X256, so we're OK
//
//
//
//   Author:  James P Lynch  July 7, 2007
//  ********************************************************************************************************

//  *****************************************************
//                Header Files
//  *****************************************************
#include "AT91SAM7X256.h"
#include "lcd.h"
#include "board.h"


//  *****************************************************
//                External References
//  *****************************************************
extern  void LowLevelInit(void);
```

```
int     main (void) {

        unsigned long  j;
        unsigned long  k;
        unsigned long  col;
        unsigned long  row;
        unsigned int   IdleCount = 0;
        int            TempColor[11] = { WHITE, BLACK, RED, GREEN, BLUE, CYAN, MAGENTA,
                                         YELLOW, BROWN, ORANGE, PINK };
        char           *TempChar[11] = { "White", "Black", "Red", "Green", "Blue", "Cyan",
                                          "Magenta", "Yellow", "Brown", "Orange", "Pink" };

        // Initialize the Atmel AT91SAM7S256 (watchdog, PLL clock, default interrupts, etc.)
        LowLevelInit();

        // Set up the LED (PA3)
        volatile AT91PS_PIO   pPIO = AT91C_BASE_PIOA;    // pointer to PIO data structure

        pPIO->PIO_PER = LED_MASK;             // PIO Enable Register - allow PIO to control pin PP3
        pPIO->PIO_OER = LED_MASK;             // PIO Output Enable Register - sets pin P3 to outputs
        pPIO->PIO_SODR = LED_MASK;            // PIO Set Output Data Register - turns off the LED

        // Initialize SPI interface to LCD
        InitSpi();

        // Init LCD
        InitLcd();

        // clear the screen
        LCDClearScreen();

        // ****************************************************************
        // *  color test - show boxes of different colors
        // ****************************************************************

        for (j = 0; j < 11; j++) {

                // draw a filled box
                LCDSetRect(120, 10, 25, 120, FILL, TempColor[j]);

                // label the color
                LCDPutStr("        ", 5, 40, LARGE, BLACK, BLACK);
                LCDPutStr(TempChar[j], 5, 40, LARGE, YELLOW, BLACK);

                // wait a bit
                Delay(2000000);
        }



        // ****************************************************************
        // *  character and line tests - draw lines, strings, etc.
        // ****************************************************************

        // clear the screen
        LCDClearScreen();

        // set a few pixels
        LCDSetPixel(30, 120, RED);
        LCDSetPixel(34, 120, GREEN);
        LCDSetPixel(38, 120, BLUE);
        LCDSetPixel(40, 120, WHITE);

        // draw some characters
        LCDPutChar('E', 10, 10, SMALL, WHITE, BLACK);

        // draw a string
        LCDPutStr("Hello World", 60, 5, SMALL, WHITE, BLACK);
        LCDPutStr("Hello World", 40, 5, MEDIUM, ORANGE, BLACK);
        LCDPutStr("Hello World", 20, 5, LARGE, PINK, BLACK);

        // draw a filled box
        LCDSetRect(120, 60, 80, 80, FILL, BROWN);
```

```
        // draw a empty box
        LCDSetRect(120, 85, 80, 105, NOFILL, CYAN);

        // draw some lines
        LCDSetLine(120, 10, 120, 50, YELLOW);
        LCDSetLine(120, 50, 80, 50, YELLOW);
        LCDSetLine(80, 50, 80, 10, YELLOW);
        LCDSetLine(80, 10, 120, 10, YELLOW);

        LCDSetLine(120, 85, 80, 105, YELLOW);
        LCDSetLine(80, 85, 120, 105, YELLOW);

        // draw a circle
        LCDSetCircle(65, 100, 10, RED);


        Delay(8000000);

        // ****************************************************************
        // *  bmp display test - display the Olimex photograph
        // ****************************************************************

        LCDClearScreen();

        LCDWrite130x130bmp();

        LCDPutStr("This guy is nuts", 115, 2, LARGE, BLACK, CYAN);

        // draw a filled box
        LCDSetRect(90, 70, 75, 120, FILL, YELLOW);

        LCDPutStr("HELP!", 80, 80, SMALL, BLACK, YELLOW);



        // ****************************
        // *  endless blink loop
        // ****************************
        while (1) {
                if  ((pPIO->PIO_ODSR & LED4) == LED4)        // read previous state of LED4
                        pPIO->PIO_CODR = LED4;               // turn LED4 (DS1) on
                else
                        pPIO->PIO_SODR = LED4;               // turn LED4 (DS1) off

                for (j = 1000000; j != 0; j-- );             // wait 1 second 1000000

                IdleCount++;                                 // count # of times through the idle loop
        }
}
```

When the main program runs, a series of color rectangles is displayed with the name of the color annotated at the bottom of the screen, as shown in Figure 13 below. The colors displayed are:

"White", "Black", "Red", "Green", "Blue", "Cyan", "Magenta", "Yellow", "Brown", "Orange", "Pink"

If you are curious as to how I developed my color values, I referred to this web site:

http://web.njit.edu/~kevin/rgb.txt.html

In this web site, **RGB to Color Name Mapping (Triplet and Hex),** there is a decimal color table where each color value is in the range 0 to 255. I simply used proportionality to convert these values to a range of 0 to 15. This may come in handy when you need to display the color Turquoise!

Figure 13. Filled Rectangle Test

Figure 14. Single Pixels, Rectangles. Lines and Characters

In Figure 14 above, the display shows various rectangles (filled and unfilled), lines and circles. The three font sizes are demonstrated and you can see some single pixel specifications on the far right.

In Figure 15 below, the Olimex BMP image has been displayed with a few overlays of text. Olimex has a free utility on their web site to convert pictures (.jpeg) into the 132 x 132 motif required by the Nokia 6100 LCD display. The text overlays demonstrate foreground and background color specification.

Figure 15. Display of a .bmp image with text

Conclusions

I set out to write a Nokia 6100 LCD Display Driver that was 100% related to the Philips PCF8833 Data Sheet. I generally succeeded but there is still the mystery of why the display needed to be inverted and the RGB setting had to be reversed. The subroutines contained herein are the most efficient for this particular controller.

If you need to port this to a different computational platform, then you need to modify the port pins used and rewrite the SPI routines to conform to the alternate microprocessor. I suspect most people could easily handle such details.

I would appreciate comments on this work and would be happy to accept any suggested improvements for inclusion in a future release.

About the Author

Jim Lynch lives in Grand Island, New York and is a software developer for Control Techniques, a subsidiary of Emerson Electric. He develops embedded software for the company's industrial drives (high power motor controllers) which are sold all over the world.

Mr. Lynch has previously worked for Mennen Medical, Calspan Corporation and the Boeing Company. He has a BSEE from Ohio University and a MSEE from State University of New York at Buffalo. Jim is a single Father and has two grown children who now live in Florida and Nevada. He has two brothers, one is a Viet Nam veteran in Hollywood, Florida and the other is the Bishop of St. Petersburg, also in Florida. Jim plays the guitar (search for lynchzilla on youtube.com), enjoys woodworking and hopes to write a book very soon that will teach students and hobbyists how to use these high-powered ARM microcontrollers. Lynch can be reached via e-mail at: lynch007@gmail.com