« Back to home

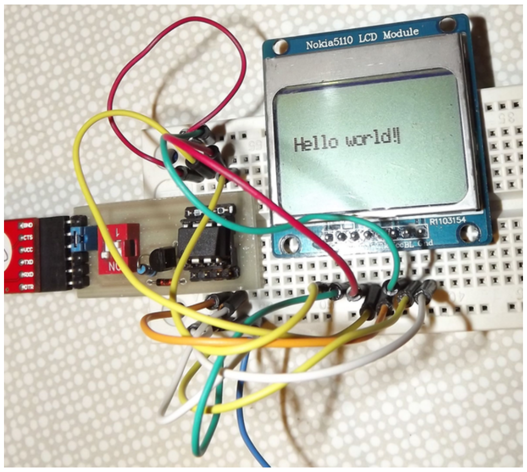
The Garage Lab

Electronics engineering, software development, IoT and home automation. What's not to love?

Nokia 5110 LCD Displays on the ATtiny

Posted on 1st May 2014 Tagged in prototyping, electronics, microcontrollers, attiny

To test the software SPI implementation in my ATtiny85 template project I wanted to use a component that I have had experience with before, was fairly simple to use and would be useful. A small Nokia LCD display fits the bill nicely.



The Garage Lab

Electronics engineering, software development, IoT and home automation. What's not to love?

If you are not familiar with these they are a 1.5" monochrome LCD with an 84 x 48 pixel resolution that were used in older Nokia phones such as the 3110 and 5110. These screens are widely available and very cheap (around $AU 3 on eBay). You can get them from SparkFun and AdaFruit as well.

There are a number of interface libraries available for the Arduino already but in this post I'm going to present a more low level interface. Most of the Arduino libraries I've seen use an off screen frame buffer to prepare the image before sending it to the display itself. This method consumes 504 bytes of RAM which is almost all of the available 512 bytes on an ATtiny so it's just not acceptable in our restricted environment. This library consumes no RAM and sends the data directly to the display - this imposes some limitations on what you can display and prevents any meaningful image composition (drawing text over images for example) but with a bit of forethought you can work around this to achieve nice looking and functional displays for your project.

> **NOTE**: These modules run at 3.3V so if you are using a 5V supply on your CPU you will need to run the inputs through a level converter first as well as provide a 3.3V power supply for module itself.

This code has already been integrated into the library, you can grab the repository from here and here is the documentation for the new functions. Some sample code to drive the routines can be found in this gist.

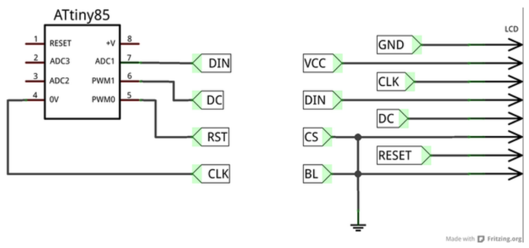
Connecting the Display

The display provides an SPI interface with most of the expected pins present (MOSI, SCK and CE - MISO is missing, you can't read data back from the device which is why

The Garage Lab

Electronics engineering, software development, IoT and home automation. What's not to love?

many libraries use the off screen frame-buffer). On top of that though there is a pin to select between command and data modes as well as a RESET pin which you will need to be able to manipulate during device initialisation. The next image shows how I wired everything up for this example.



NOTE: As pointed out by Dave in the comments the diagram above is incorrect, it shows the *CLK* signal wired to ground (Pin 4). It should actually be wired to pin 3 (which is PB4). I will update the diagram as soon as I can.

As this is the only SPI device in use I've wired the chip select line directly to ground so it is always selected. The backlight is pulled to ground as well, you can tie this to Vcc through a current limiting resistor if you like or drive it from a digital output or push button if you

The Garage Lab

Electronics engineering, software development, IoT and home automation. What's not to love?

want to be able to turn it on or off to match the lighting conditions. The remaining four pins are connected to the ATtiny.

Double check the pin assignments on the module before connecting it up, I have a few from different suppliers and the pin order is slightly different in two of them.

The Garage Lab

Electronics engineering, software development, IoT and home automation. What's not to love?

Communications

Before you do any communication with the device it needs to be reset. The datasheet is very specific on how this occurs; the RESET line must be LOW when power stabilises and must be pulled high within 100ms of power on - this means that the initialisation process must occur very early on in the program. Once the device has been reset you can start sending commands and data to it over the SPI bus.

The C/D pin is used to determine if the data being sent is a command (C/D is low) or data (C/D is high). The pin state must be sent prior to the first bit being sent. Bytes are sent to the device with the MSB coming first.

Sending Commands

The LCD supports 8 separate commands which are broken into a 'normal' and 'extended' command set. The majority of the 'extended' commands are to do with initialisation and device configuration, they are generally only sent immediately after the device is turned on in order to put it in a working state.

To send a single command all you need to do is set the C/D pin to LOW and then send out the command byte on MOSI clocked by the SCK pin. The MOSI value is read by the LCD on the rising edge of the SCK signal. The following simple function achieves this:

```
/** Send a command byte
 *
 * @param cmd the comman
 */
static void lcdCommand(u
  // Bring CD low
  PORTB &= ~(1 << LCD_CD
  // Send the data
  sspiOutMSB(LCD_SCK, LC
  }
```

The *LCD_CD*, *LCD_SCK* and *LCD_MOSI* constants define the pins that are being used for communication. To actually send data I'm using the sspiOutMSB() which is part of the template library I developed. This particular function

The Garage Lab

Electronics engineering, software development, IoT and home automation. What's not to love?

implements the SPI protocol in software allowing you to transfer data via SPI on arbitrary pins.

Sending data is similar, first the C/D pin must be set HIGH and then the data byte is sent MSB first. This variation looks like:

```
/** Send a data byte to
 *
 * @param data the data
 */
static void lcdData(uint
  // Bring CD high
  PORTB |= (1 << LCD_CD)
  // Send the data
  sspiOutMSB(LCD_SCK, LC
  }
```

These two functions are all that is needed to start developing the library.

Initialising the Device

The initialisation sequence was a little difficult to get working, it seems that some implementations of the chipset are a little picky about the order in which the commands are sent. Once that is resolved it easy enough to do - the code here has worked on a range of screens from multiple vendors without fail.

```
/** Initialise the LCD
 *
 * Sets up the pins requ
 * the actual chipset in
 * @ref hardware.h.
```

The Garage Lab

Electronics engineering, software development, IoT and home automation. What's not to love?

```
 */
void lcdInit() {
  // Set up the output p
  uint8_t val = (1 << LC
  PORTB &= ~val;
  DDRB |= val;
  // Do a hard reset on
  wait(10);
  PORTB |= (1 << LCD_RES
  // Initialise the LCD
  lcdCommand(0x21);  //
  lcdCommand(0xA1);  //
  lcdCommand(0x04);  //
  lcdCommand(0x14);  //
  lcdCommand(0x20);  //
  lcdCommand(0x0C);  //
  }
```

The Garage Lab

Electronics engineering, software development, IoT and home automation. What's not to love?

The first thing we do is initialise the IO pins that are going to used and ensure they are all set low. We then wait for 10ms (to make sure the voltage levels have well and truly settled) and raise the RESET pin high to reset the LCD.

Immediately after that we configure the LCD driver operation. The temperature coefficient and bias mode are the recommended values from the datasheet and can be left as they are. The LCD Vop value is essentially a contrast control (it controls the bias voltage to the liquid crystal). The command format is 01nnnnnn in binary where *nnnnnn* is a value between 0 and 64 inclusive. If this value is too high the entire display will be too dark and you won't be able to make out

The Garage Lab

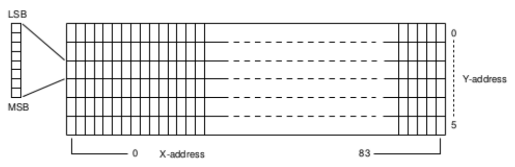
Electronics engineering, software development, IoT and home automation. What's not to love?

the image from the background, conversely if it is too low the display will appear washed out. The value used in the initialisation sequence above seems to provide good readability with sharp contrast - you can raise or lower it to suit your display if you need to - just make minor modifications (+/- 8 to start with) until you get a value that works for you.

The final command sets the addressing mode - this determines the order in which pixel data is sent to the device. Before we move on to that we need to look at how the pixels are stored in the displays memory.

Memory Layout

The pixels are stored in memory in 6 rows of 84 columns each. Each row is 8 pixels high (giving the vertical resolution of 48 pixels) and each column is a single byte setting the value of each of the 8 pixels.



To change the displayed pixels you send a byte to a specific row and column address - the bits in the byte set the value of each pixel in the row from the top (bit 0) to the bottom (bit 7). If you are used to the *normal* frame-buffer layout that goes from left to right, top to bottom this can be a little more difficult to get used to, calculate the correct offset to write a pixel using standard cartesian co-ordinates is certainly more complicated.

Writing Pixels

To write pixels to the display you need to set the row and column addresses as a command and then write data values to store in the LCD memory. The address will automatically increment as each data byte is received so you don't need to set the address for each byte. The following code shows a simple example of this:

```
/** Clear the screen
 *
 * Clear the entire disp
 *
 * @param invert if true
 *              with bl
 */
void lcdClear(bool inver
  uint8_t fill = invert?
  lcdCommand(0x80);
  lcdCommand(0x40);
  // Fill in the whole d
  for(uint16_t index = 0
    lcdData(fill);
}
```

The Garage Lab

Electronics engineering, software development, IoT and home automation. What's not to love?

In this case we are clearing the screen by setting all pixels off (or, if inverted, setting them on). The first two commands set the column address to 0 and then the Y address to 0. We then send 504 bytes to fill the entire memory. When the internal address counter reaches the end of a row it will automatically wrap to the next row so we don't have to reset the addresses every time we change rows.

The Garage Lab

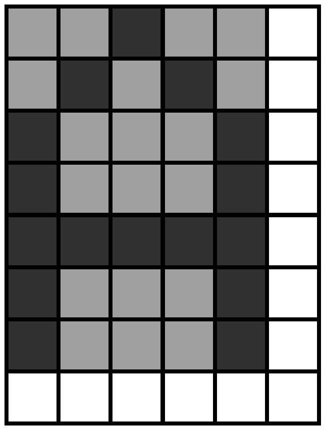
Electronics engineering, software development, IoT and home automation. What's not to love?

Simple Display Routines

I've kept the set of routines available very simple on purpose. As well as the screen clearing function (shown above) there is a function to clear a single row. For general output I've provided functions to display text (both plain and inverted) and small images. These functions are described below.

Drawing Text

The controller chip in the LCD does not have direct support for displaying text, you have to send the characters as bitmaps. I have included static data for a simple 5 x 7 pixel font in the library that is laid out in vertical strips so it is a quick process to render individual characters.



The font draws in a 5 x 7 pixel area but fills a 6 x 8 pixel square to provide spacing between characters both horizontally and vertically. At this resolution you can display up to 14 characters on each of the 6 lines of the display. I've provided functions for displaying single characters as well as NUL terminated strings from RAM or PROGMEM.

Characters cannot span rows but they can be positioned horizontally at any pixel location. Be aware that all pixels in the character cell will be modified when it is drawn, you cannot overlay graphics on an image without part of the image being erased.

The Garage Lab

Electronics engineering, software development, IoT and home automation. What's not to love?

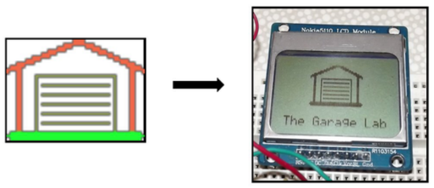
Drawing Images

To draw images efficiently it helps if they are stored in the same layout as the graphics memory in the module - that is, a sequence of 8 pixel vertical strips that can be sent to the display. I've added a function that will display images directly from PROGMEM. It expects a descriptor byte as the first data value which contains the width (in pixels) and height (in rows) of the image data that follows. This is packed into a single byte as *HHWWWWWW* where *HH* is one less than the height of the image (in 8 pixel rows) and *WWWWWW* is one less the width of the image in pixels. The bytes following the descriptor are vertical strips of 8 pixels each, *WWWWWW* + 1 for each row.

As with the character drawing functions the top of the image must be at the start of a row but it can be placed at any pixel position horizontally. If the image extends past the borders of the display it will simply be clipped. Once again, the routine will modify all pixels in the rectangle covered by the image erasing what was previously on the screen in that area.

The Garage Lab

Electronics engineering, software development, IoT and home automation. What's not to love?



Because manually generating the byte array for an ima would be a time consuming (not to mention annoying) process I wrote a simple Python script to convert arbitrary images into array definitions that can be embedded in your code. The script is called *lcdimage.py* and I've added it to the tools directory in the GitHub repository - simply run the script with a list of filenames and it will generate C code that you can paste into your project.

It uses the Python Imaging Library to read a wide range of image formats, for best results use a monochrome image or a PNG with transparency.

What Next?

These displays are cheap and easy to use - a good option to add a simple UI to your project. The downside is the number of pins they use - 4 out of the 6 available on the ATtiny. It might be possible to control the RESET pin on the

The Garage Lab

Electronics engineering, software development, IoT and home automation. What's not to love?

module with external circuitry rather than though an IO pin which would save another pin. The RESET line is not used after initialisation so you could reuse it as an analog input as long as the voltage being sampled doesn't fall below 0.7V - in one circuit I'm designing I'm using it to monitor a 6V battery pack through a voltage divider, in this case the device will power off before the voltage drops below 1.6V so it will not trigger a reset of the display.

You could use the single pin keypad I described earlier to provide simple menu navigation - the one pin serial interface could then be used to send menu selection data to a host machine (such as a Raspberry Pi without a display) to trigger some action.

Another idea is a simple Bluetooth enabled temperature sensor with display - the ATtiny has a built in temperature sensor so no external components are required for that, the battery could be monitored through the RESET line and the remaining two pins could be used to implement a serial interface to a HC05 Bluetooth module.

The Garage Lab

Electronics engineering, software development, IoT and home automation. What's not to love?

Overall it's another useful component that can be added to your toolbox. The code is now available in the template library, I'm looking forward to seeing what people come up with.

Share this article:

Comments **Community**

1 **Login**

Recommend **Share**

Sort by Best

Join the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

MustacheMtn • 7 months ago

Exactly what I was looking for and explained very well. Thank you.

Reply • Share ›

Dave • 3 years ago

I got it working. Very nice! I'm trying to debug some code on a tiny85 and w/o the serial output I was having a tough time getting information. There are only so many blinking LED that you can check! This should help me greatly. Thank you!

Reply • Share ›

The Garage Lab

Electronics engineering, software development, IoT and home automation. What's not to love?

Shane Gough Mod ➔ Dave
• 3 years ago

Hi Dave, glad it's working for you. If you are interested in serial communications on the ATtiny85 there is a software serial implementation in the library as well, it will use up less pins than the LCD interface and works nicely at 57600 baud.

Reply • Share ›

Dave ➔ Shane Gough
• 3 years ago

I'll give that a look, but I'm a little confused as to how to get that connection back to my computer. I'm using the AVR Pocket Programmer from sparkfun.com which is connected to the computer's USB. Would I somehow be

see more

Reply • Share ›

Shane Gough
Mod

The Garage Lab

Electronics engineering, software development, IoT and home automation. What's not to love?

Dave • 3 years ago

I'm glad you got it working and I apprecia the feedbacl thank you for that.

The serial interface would require a USB serial adapter on the PC side (I tend to use the little FTDI friend cables that are available from SparkFu AdaFruit and others). Sounds like the LCD is giving you the debuggi

The Garage Lab

Electronics engineering, software development, IoT and home automation. What's not to love?

The Garage Lab

Electronics engineering, software development, IoT and home automation. What's not to love?