Multifunction Controller-less STN LCD Software Library and Reference Hardware

NXP mbed Design Challenge

Project Number: NXP3811

mbed LPC1768 Model

Abstract:

When an embedded device needs to convey much information to its user, the use of a graphic LCD is arguably the best way to do so. Unfortunately, in the world of rapid prototyping devices such as the mbed, integrating a graphic LCD is often a showstopper. Complexity and cost form two large barriers for the casual electronics developer. Quality software libraries are scarce and endless variations of LCD controller ICs make any attempt at a universal library a lofty and continuing task. Even with the incredible momentum of the open source movement, the unfortunate truth is that one of the most sought after capabilities of rapid prototyping environments remains one of the most poorly implemented. Some specialty companies, such as Matrix Orbital, aim to fill this need by providing easy to use displays. While they do a good job, they do so at a very high cost, with the most basic graphics LCDs starting at over $100USD.

The intent of this project is to provide one solution to both the problem of cost and the difficulty of implementation. With the relatively large memory and high performance of the LPC1768, it is possible to completely eliminate the need for an external LCD controller by implementing the control routine in software. The mbed communicates directly with the LCD driver ICs and handles refreshing the image at 60 or more hertz. While there are many different manufactures of STN LCD driver ICs, they tend have a de-facto standardized interface which made a working universal library an attainable goal. Extensive testing and optimization has been performed, resulting in very reasonable usage of processing time, which often times, is compensated by the dramatically reduced time to perform video writes. Using the setup pictured, with the mbed running at 96MHz, and the screen refreshing at 60Hz, 87% of the MCU cycles remained unused; typically, this figure is over 90% when using the library with a monochrome
display. The library supports monochrome and color LCDs with both 4 and 8 bit data paths and also includes experimental support for dual scan displays. This covers nearly any controller-less STN graphic LCD you might come across.

Because knowledge of the nature and workings of controlling LCDs is not very widespread, these types of LCDs are widely available in surplus and on eBay at fantastic price points. While researching this software, I purchased various quantities of nine different models, each costing as little as pennies for chip on glass in case quantities to at most $10USD at quantity one, for the lovely 6 inch model pictured in this abstract.

Of course, time is money, especially where rapid prototyping is concerned. One goal for this library is easier implementation than what is traditionally associated with a Graphic display. By extensively using the C/C++ pre-processor, an optimal balance between ease of configuration and performance has been achieved. Configuration is accomplished using simple and documented definitions in a single header file. It is possible to setup some screens with 5 minutes and a breadboard.

It seem that touch screen technology can be another point of difficulty for casual embedded developers. I see many questions related to set up, calibration, and controller choices. Thanks to the consistency of the mbed platform, this library completely eliminates the hassle associated with setting up a four-wire touch screen, by far the most common touch method on these types of display. Using the onboard ADC, a touch driver has been created, allowing for tight integration of touch and graphics. Setting up touch only involves plugging in the four wires, in any order, and making a single call to a calibrate function. The driver stores the calibration to the mbed file system and returns touch coordinates in terms of x and y pixel locations.

To ensure immediate productivity, a graphics primitive library has been included. In addition to basic shapes with filling and dithering, the library handles embedded bitmap images and fonts. A cross platform PC utility written using the Flex framework for Adobe Air provides a convenient way to encode fonts and images. Unique to this library, the utility allows for previewing fonts and graphics over Ethernet. It is no longer necessary to download and recompile just to find out an image or font does not look correct. The graphic pictured above was not programmed into the mbed flash; it was transferred instantly over a local network.

One unfortunate caveat about controller less-displays is that they often require you to bring your own bias voltage, typically around 20V. The library also includes a boost regulator class allowing the user to set up a simple low-power regulator using only a few external discreet components.

Finally, this project includes the details of a reference board designed to get a person up and running with almost any STN display very quickly. The boards includes a positive bias generated by the mbed PWM, a negative bias, a constant current source for LED backlights, a joystick, and an Ethernet port. The focus, however, is the novel approach used to accommodate the unpredictable pinouts of different LCDs.

Like any electronics solution, controlling an LCD with software is not without compromise. The focus of this project has been to seek novel ways to minimize the complexity and enhance the advantage of using controller less displays. I look forward to continuing this project for some time; however, it offers a working answer to graphics headaches today. It takes advantage of modern ARM resources and offers a solution that simply cannot be implemented in a practical way on most other rapid prototyping platforms. It enhances the attractiveness of the mbed platform and, most importantly, it just works.
Overview Block Diagram

External Passives → PWM → Boost Controller

LCD Driver IC

LCD Controller Logic

Screen Size Pinout → User Configuration

Frame Buffer

32-bit Chunks → 8-bit Chunks

User Program

Commands → Graphics Primitive Engine

Fonts, Images → Calibration Display

Screen Boundaries

Touch Events → Touch Driver

Analog Voltages → Digital Excitation

Ethernet

Touch Screen
STN LCD Controller Library: Abstract

**Code Snippet:** This is most of the “STN_UserConfig.h” file. These are the only options that need to be set to configure the controller.

```c
/**********************Screen Options**************************/

//The number of horizontal pixels in each line of the display
#define TOTAL_COLS 320

//The number of vertical pixels in each column of the display
#define TOTAL_ROWS 240

/*
 * this is the number of times per second the screen is refreshed.
 * a higher refresh rate results in a sharper image but uses more cpu resources
 */
#define REFRESH_RATE 60

/*
 * the pixel clock will be held high for zero to three
 * instruction cycles to ensure the minimum pulse time for driver IC
 * on the display is met. Start at three and move down until it
 * stops working. Many displays will work at zero.
 * This results in about an extra 1-2% cpu usage per clock.
 */
#define HOLD_CLOCK 0

/*
 * if this define is not commented, the display will be rotated 180
 * degrees this is only functional with non dual-scan monochrome
 */
#define DISPLAY180

/*
 * if your screen is color, uncomment this line.
 */
#define STN_COLOR

/*
 * if your screen is split into a top and bottom section (dual scan),
 * uncomment this line. This is used on large monochrome screens,
 * usually from many years ago. Support for this option is experimental.
 */
//#define DUAL_SCAN

/*
 * This is the number of pins your display uses for data. This must
 * be either 4 or 8.
 */
#define DATA_WIDTH 8
```
/* Screen Pinout */

The data pins are fixed. They should be connected according to this map:

8-bit: LPC1768 P0.4 - P0.11 -- the only place we have 8 bits in a row

D0     p30
D1     p29
D2     p8
D3     p7
D4     p6
D5     p5
D6     p28
D7     p27

4-bit: ALT_PINS

D0     p30  p6
D1     p29  p5
D2     p8   p28
D3     p7   p27

/******************** Screen Pinout *****************************/

edo

#if your data width is 4, this tells the compiler to use pins
p6, p5, p28, and p27
#
#define ALT_PINS

// the pin assigned to the frame sync function A.K.A. FLM, EIO - FLM
stands for first line marker
// -p24 on reference board
#define FLM_PIN p24

// the pin assigned to the line latch function A.K.A. LP
// -p25 on reference board
#define LP_PIN p25

/* the pin assigned to the clock pulse function A.K.A. CP, XCK.
This function can only be assigned to p19 through p26.
This needs to be assigned as an integer without the p in front.
-26 on reference board
*/
#define CP_PIN 26

/*
this is the frame inversion pin A.K.A. FR, FP, M Your display may not
have this If not, comment this line.
-p23 on reference board
*/
#define FR_PIN p23

/*
this pin turns the display on and off. You can hold this pin high if
you need the extra pin but be careful not to expose the display to an
extended DC bias. If you are not using this pin, comment it out.
-p22 on reference board
*/
#define DISPOFF_PIN p22
Reference Board Schematic Sheet 2: Power Conversion

Schematics are described in more detail in the full project report