ezLCD+103

An Overview of Research on the EarthLCD ezLCD+103 Touch Screen Display

Frank Pernice

ESE 476 Fall 2013

Statement of Work

Abstract

In current ESE 380 and ESE 381 classes, it is very common to develop projects using a generic 4x4 keypad and a 3-line display. While this is sufficient for most purposes, industry has been moving away from these interfaces and towards touch screen interfaces instead. Touch screens are pretty much the norm these days. They have largely replaced physical buttons and conventional displays in just about everything from cellular phones and automobiles to personal computers and even some refrigerators and other home appliances. While some of these applications of touch screen technology are more of a novelty rather than a practical application, it is nonetheless the trend among both consumer and commercial electronics.

Touch screen technology allows the product developer to easily customize a user interface for their product. They allow for ultimate flexibility; giving developers the option to enable, disable, or even remove buttons and other objects from the screen given the state of the system. This allows for simple and eloquent user interfaces which are sometimes more intuitive to use than, say, pressing a combination of generic buttons like *CTRL+ALT+Delete* to achieve some action. They allow for a user-interface concept often referred to as "fool-proof design." Fool-proof design is exactly what it sounds like; it's fool-proof. It is a design approach where certain buttons are disabled or removed from the screen when an option is not available. A perfect example of this principle is the back-button on your internet browser; when you open your browser for the first time, the back button is often disabled and grayed-out because there is no prior history in the browser session. Once you browse around the web a little bit, this button is enabled again and you are free to press it as many times as you wish *until* you reach the homepage again, after which it is once again disabled. This is a fundamental user interface design technique and should not be taken lightly in product development.

Touch screens, especially with color displays, allow the developer to make eye-appealing user interfaces. In product development, the user interface is everything to the consumer. One need not look further than the recent failure of RIM's Blackberry devices to see how important this concept really is. Other examples include the demise of search engines like Yahoo and Altavista (and the rise of Google) and the demise of MySpace (and the rise of Facebook). Consumers want easy-to-use user interfaces; they do not want to be bogged down by complex menus or numerous key combinations. In this regard, touch screen technology is a double-edged sword. They give the developer the freedom to create extremely complex UIs as well as simple and intuitive ones. UI design is more of an art than a science; but it is so crucial that there are often computer science classes dedicated to the topic and entire businesses which revolve around this subject.

Nonetheless, it is crucial that students interested in embedded systems design be introduced to user interface design and have the freedom to create a user interface on their own. By instructing students in both UI design and touch screen technology simultaneously, students will be exposed to the principles of a modern UI rather than learn on obsolete technology. The ezLCD+103 is an integrated solution to make this transition, simply put, "eZ". The C library I've developed for it should make the transition even easier.

Table of Contents

Hardware

Powering the Device and Logic Levels

The ezLCD+103 is a rather intriguing device when it comes to power and logic levels. The device is powered by a 5V source; however, its logic pins are **NOT** 5V tolerant and instead use 3.3V logic. Of course, the ATMega128 is a 5V logic device, as previously discussed in my EA DOGM128 report. In order to get around this requirement, I initially tried to use a voltage divider to step down the logic going to the ezLCD from the ATMega128. However, this proved to be an issue because of stray capacitances which distorted the logic levels. Again, since software development was of utmost importance, I personally purchased the BOB-12009 bi-directional logic-level converter from Sparkfun[1] to resolve the issue. This neat device consists of one BSS138 FET and a couple resistors as shown below in **figure 1** for each channel. The BOB-12009 has four channels and so the board consists of one logic level converter for each channel.

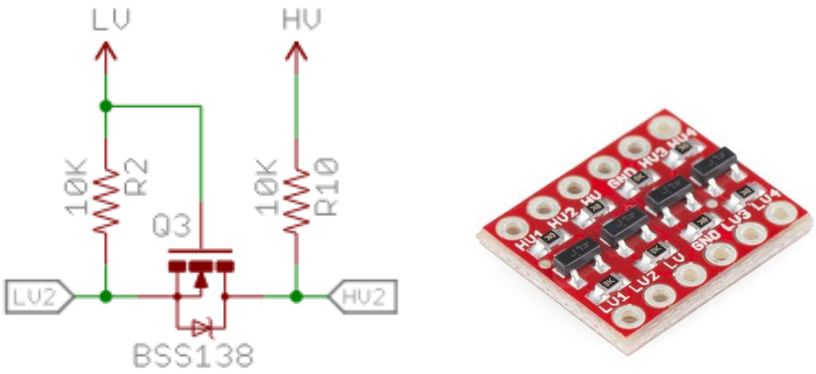


Figure 1: Bi-Directional Logic-Level Converter Schematic and the Sparkfun BOB-12009

Four channels were enough to convert all four SPI lines and so it was perfect for this display. Of course, since the ATMega128's minimum $V_{IH} = 0.6 \times V_{cc} = 3V$ when operating on a 5V supply, the MISO signal need not be stepped-up. Therefore, one of the channels on the BOB-12009 may be saved for another application. Oddly enough, the ON/OFF pin to the device **is** 5V tolerant, so this signal comes straight from the microcontroller as well.

Proprietary Connector

Out of the box, the ezLCD+103 is not breadboard compatible. In order to access its control pins, a **Hirose DF11-32DS** 32-pin header is needed. Due to the inability to find this connector with wires already crimped to it, I opted instead to (painfully) solder wires to the needed connections. Since the ezLCD uses SMD components, this is a slow and tedious process. Though this is approach was fine for prototyping and library development, it is highly suggested to buy Hirose connectors and have a breakout board fabricated for breadboard use.

Interfacing with the ezLCD+103

The ezLCD+103 offers a number of interfaces including I^2C, SPI, USB, RS232, and microSD. It is also capable of running standalone without an external microcontroller (but what's the fun in that?). Its dedicated GPU allows the master device to simply send commands and conserve resources. For simplicity, my library was developed to use SPI; however, it is written in a way that allows it to be changed rather easily. Since this is a touch screen, we obviously need to receive data from the SPI slave, so the MISO pin must be connected in this case. Unfortunately, the ezLCD+103 does not offer interrupt capability. That is, there are no logic lines between the master and the slave which would alert he master that an event occurred on the screen. In all cases (SPI, I^2C, or otherwise), the display only offers strict serial communication. Therefore, the screen must be polled by the microcontroller instead.

The timing diagram provided in the datasheet (also shown below in **figure 2**) indicates that this device has a clock polarity ($CPOL$) of 0 and a clock phase ($CPHA$) of 0 and that information is written to the screen most-significant-bit first. Data may be clocked in at a maximum of 4MHz; the library currently assumes a system clock of 16MHz.

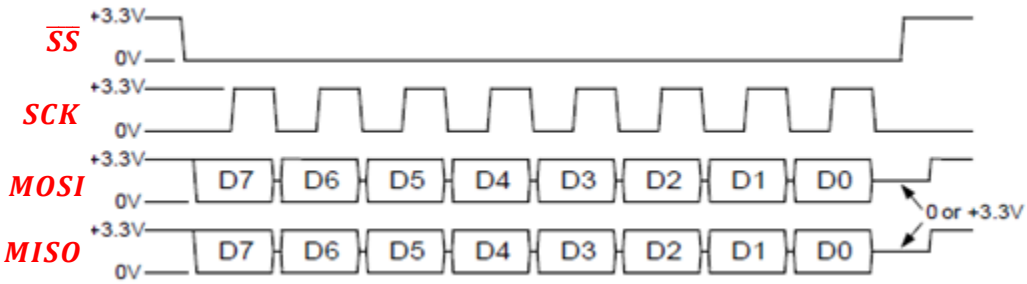


Figure 2: ezLCD+103 SPI Timing Diagram

Obviously, the ON/OFF signal should be set to logic 1 (ON) before sending data to the screen.

Circuitry Images and Schematic

The schematic provided in **figure 6** illustrates the interface between the ezLCD+103, the BOB-12009, and the ATMega128. For simplicity, the ATMega128 is not shown in the schematic. It should be noted that the ezLCD+103 includes a 3.3V source which may be used by external components. In this case, it is used for the BOB-12009 as shown. It should be noted that the included ezEarthIO Evaluation Board is not needed for this project. It essentially replaces our ATMega128 with a PIC microcontroller.

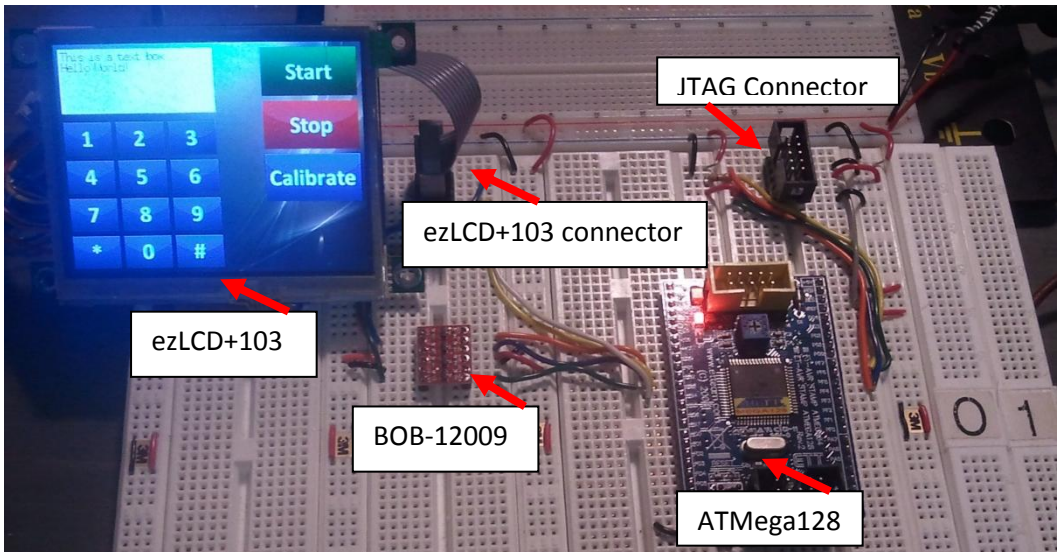


Figure 3: ezLCD+103, BOB-12009, and ATMega128

Figure 4: ezLCD+103 close-up (front)

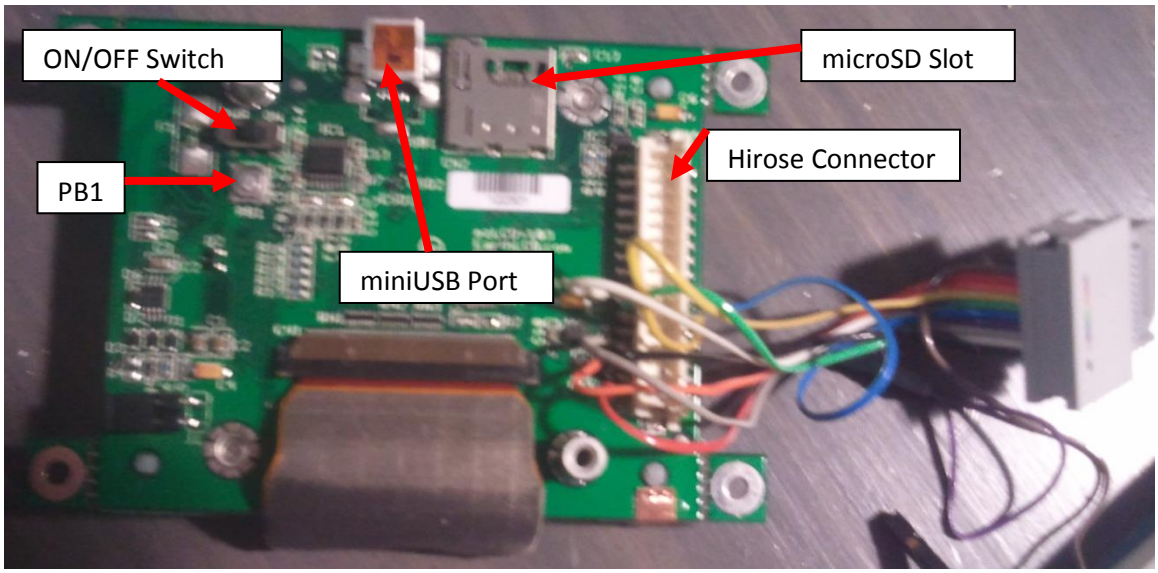


Figure 5: ezLCD+103 close-up (rear)

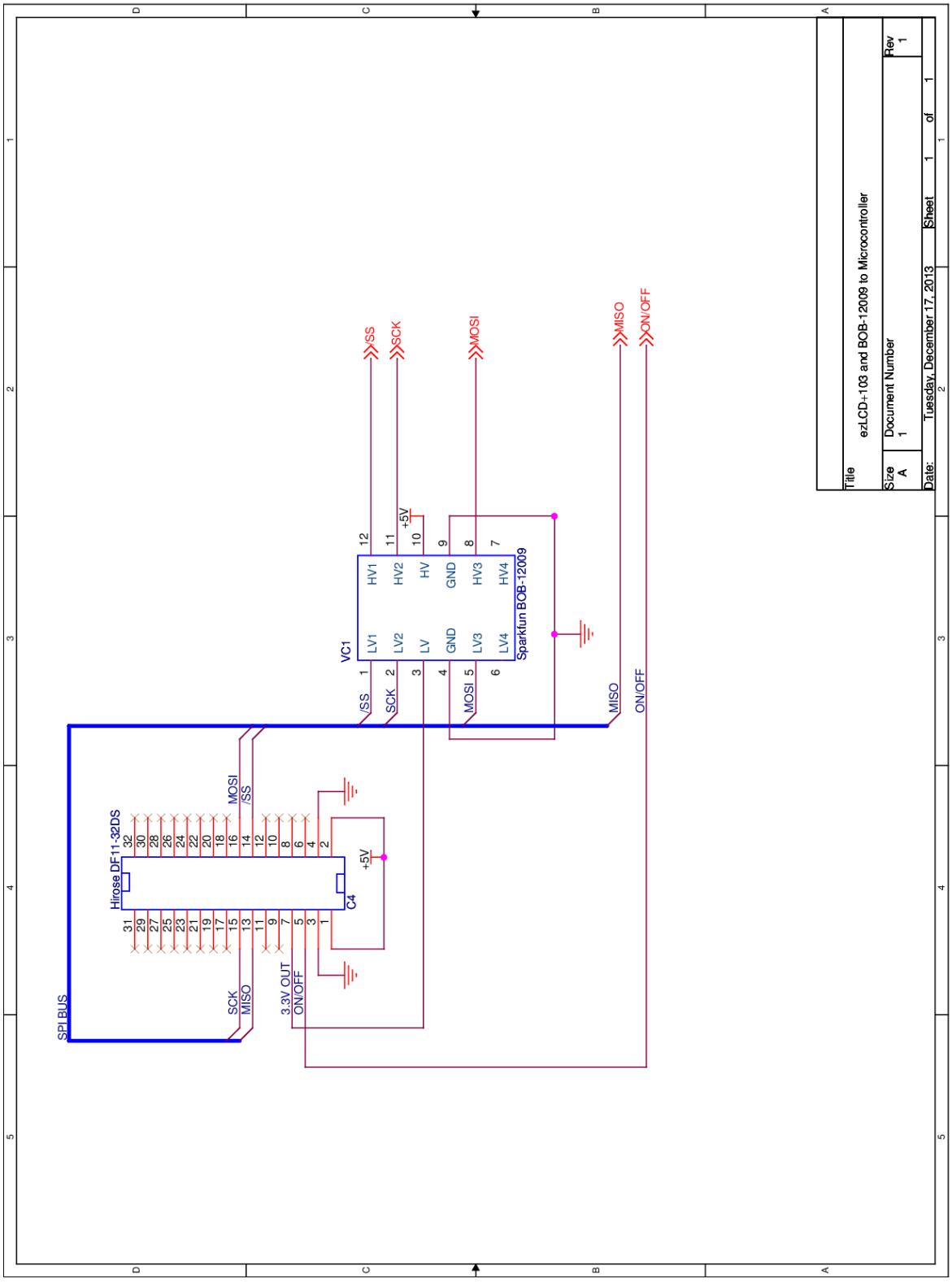


Figure 6: Schematic illustrating the ezLCD+103 and the BOB-12009. The microcontroller is off-page.

Software

Getting Started With the ezLCD+103

Firmware Update

Before doing anything with the ezLCD, it is important that the firmware be updated. The original firmware has issues with SPI communication which firmware update 2.4.2_103 patches. Upgrading the firmware is relatively easy and straight forward. Simply follow the steps below. For convenience, a pre-formatted memory card will be provided with this documentation. If using the pre-formatted card, the user should skip steps 1-3 below. Nonetheless, the steps are as follows.

1. Format a microSD card to either FAT-32, FAT-16, or FAT-12 formats.
    a. In Windows 7, open up “My Computer”
    b. Plug the microSD card into a memory card reader
    c. Locate the card under “My Computer” and right-click on it.
    d. After right-clicking on the card, click the “format” button. This will bring up the Windows format utility.
    e. Under the “File System” pull-down menu, choose any of the three formats mentioned above.
    f. If desired, give the card a name.
2. Copy the provided firmware file (.eze) to the root directory of the microSD card.
3. Eject and remove the microSD card from your PC.
4. Insert it into the ezLCD+103’s microSD slot shown above in **figure 5**.
5. Supply power to the ezLCD+103, but **DO NOT** turn it on yet.
    a. **NOTE:** Power may be supplied via the miniUSB port. One need not supply power through the Hirose connector. Simply connecting the display to your PC via USB will suffice. This allows for speedy mass-firmware upgrades.
6. Hold down the **PB1** button on the rear of the display (shown in **figure 5**)
7. Turn the display on using the **ON/OFF** switch and then release **PB1** once the firmware update screen loads.
8. Wait for the firmware to finish upgrading. This will take a few minutes.
9. Cycle the power with the **ON/OFF** switch and then disconnect power.
10. Finished!

Customizing the display

The ezLCD+103 is easy to customize with custom fonts, graphics, and buttons. Customization is done through a microSD card just like firmware upgrading. There are two files which are critical to this process: **UserConf.txt** and **UserRom.txt**. The first file is used for loading settings (such as default splash screen, brightness, communication protocols, etc) while the second one provides a list of files to be loaded to the display's ROM. Both of these files are discussed in detail in **section 9** of the ezLCD+103 datasheet. The important thing to take away from this section is that the order in which items are placed into **UserRom.txt** is important. For each file, per set of file types, an index is given based upon its placement in **UserRom.txt**. Again, for convenience, a microSD card containing pre-customized settings will be provided. If the user is using the provided library, it is strongly recommended not to touch the **UserConf.txt** file unless modifying miniscule settings such as the brightness or the splash screen. **Nonetheless, the procedure for customizing the display is the same as for loading firmware (except at step 2, the user will load the two aforementioned text files and the needed images and fonts to the card rather than a firmware file).**

One of the issues with building a library is that it is difficult to predict exactly how it will be used in the future. The same goes for display customization. The pre-customized microSD card contains a number of buttons (a full number pad; a calibrate button; start and stop buttons; back, next, and continue buttons; and a settings button). These should hopefully be sufficient for the near future. However, should they not be sufficient, new buttons may be easily added. One must simply load new bitmap images to the microSD card and add their path to the **UserRom.txt** file as discussed in **Section 9 of the datasheet**. In order to maintain uniformity among button styling, I have used an online button-maker[2] to generate the button images. URLs which link to my pre-configured buttons are listed at the end of this document. In this way, the same color scheme and styling can be maintained as new buttons are needed for new projects.

As new buttons, images, and fonts are added or removed from the display's ROM, it is important to modify the library's user configuration header file (**EZLCD_103_user_config.h) ** accordingly. This file provides a convenient place to store information about images and fonts (such as their indices in the **UserRom.txt** file and their sizes). This file should be modified accordingly as new things are loaded to the display's ROM. This file also contains pin configurations which may be modified as per the user's needs.

The ezLCD+103 Driver

The main goal of this research was to develop a library that would make using this screen as easy as possible. For brevity, the contents of the library itself will not be discussed in this paper since the library makes heavy use of Doxygen for documentation. This paper would best be classified as a novel if it were to discuss every single library function. And still, the library does not contain all of the functionality of the ezLCD since some of these functions were deemed to either be redundant and/or not useful for the purposes of ESE 381. However, since the ezLCD+103 is a command driven system, adding these functions is merely a matter of a few lines of code. Instead, this paper will fine-tune some of the more complex concepts which should be understood before diving into the library itself; namely images, fonts, and buttons.

Dealing with Images and Fonts

As previously mentioned images and fonts are loaded onto the ezLCD+103 via a microSD card. As **section 9.1** of the ezLCD+103 datasheet points out, each file listed in **UserRom.txt** is given an index based upon the order in which it is listed. This index is not absolute, but relative to the file types themselves. Suppose we listed 4 bitmap fonts, 5 true-type fonts (TTF), and 10 bitmap images in the **UserRom.txt** file. The bitmap fonts would be given indices of 0-3, the TTFs would be given indices of 0-4, and the bitmap images would be given indices of 0-9 respectively.

After the images, fonts, and **UserRom.txt** file have been programmed into the display itself, these indices are used to select them and paint them to the screen. For simplicity, it is easiest to define these parameters in the library's user configuration header file so that they can be referred to by name.

For instance, the following code snippet renders the background image over the default splash screen.

```
--------------- user_config.h --------------------------

#define BACKGROUND_1_INDEX                              1

---------------- main ----------------------------------

 ezLCD_set_xy(0,0);

 ezLCD_put_picture_rom(BACKGROUND_1_INDEX);
```

Buttons

Now that images have been discussed, we may now dive into the most important concept of a touch screen: buttons. Buttons are made up of several key components.

1. A unique ID number which will distinguish it from the other buttons on the screen
2. Up to three bitmap images which represent different button states. A button may share the same image for all states (however, that makes it difficult for the user to determine if they've actually pressed a button and therefore goes against the principle of "fool-proof" design).
    a. 1 image which depicts the button in its 'up' state
    b. 1 image which depicts the button in its 'down' state
    c. 1 image which depicts the button in its 'disabled' state

    The examples below show the button state images for the "Calibrate" button which is located on the pre-configured customization microSD card.

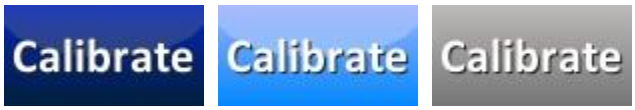


Figure 7: Up, down, and disabled button state images for the "Calibrate" button

3. A defined touch zone. A touch zone is the part of the button image which is sensitive to touch. For simplicity, it is best to make the touch zone and the images the same size. However, should the user decide otherwise, the Doxygen page for the library and the ezLCD+ External Commands Manual discuss this in further detail. The touch zone is specified by a width and a height in pixels.
4. XY coordinate of the top-left corner.

Once the button images have been loaded into the ROM and the ID and touch zone have been defined, the user can then begin to code.

Before any buttons can be placed on the screen, a touch protocol must be chosen. A touch protocol defines how the screen reacts to touch. There are three such protocols for the ezLCD+ series. They are referred to as ezButton, cuButton, and calibratedXY. Of those three, only ezButton and cuButton deal with buttons directly and are discussed below; the calibratedXY protocol deals with coordinates as its name implies. Because the ezLCD+ devices do not have a signal which may be used to trigger an interrupt on the SPI master side, we **MUST** poll the screen.

1. ezButton
    a. Using this protocol, the screen will broadcast data about buttons serially only when an event occurs. By event, we mean whether or not a button has been pressed or released. This is only done once.
    b. This is the recommended protocol since it is the easiest and most intuitive to use.
2. cuButton
    a. This protocol is similar to ezButton, except that data is broadcast anywhere from 5 to 20 times per second. The button states are slightly different in this mode. Instead, the screen broadcasts the value 0xFF continuously until a button is pressed. When a button is pressed, it will broadcast its ID. So it can be stated that when using this protocol, we have two states: button pressed and no buttons pressed.

Once we have decided upon a protocol and notified the screen of our choice, we may then begin to place buttons on the screen and begin polling for button presses. The following code snippet (taken from the library's *ezLCD_wait_for_event()* function) illustrates just one way to poll the buttons. It is left as an exercise to the students to develop others as needed. The function only works for the ezButton protocol as we will soon see.

```
uint8_t ezLCD_wait_for_event()
{
  uint8_t released = 0; /* Button Released flag */

  int8_t button = -1;   /* Button ID can NEVER be negative, but it can be zero
                         * so we initialize it to -1 to indicate that no button
                         * has been pressed yet. */


  /* Loop until the button has been released */
  while(button == -1 || released == 0)
  {
    /* Send nop command to display, we only care about the data being sent back
    from the ezLCD.*/
    button = ezLCD_transfer_data(0);

    if((button & 0xC0)>>6 == 2) /* If button state is UP */

    /* This if statement will only be entered AFTER the below "else if" has been
     * executed since the ezButton does not send any data about buttons which
     * have never been pressed. */
    {
      released = 1;           /*set the released flag to 1 so we may exit loop*/
      /* Return the button image to its UP position */
      ezLCD_set_button_state(button & 0x3F, EZLCD_BUTTON_UP);
    }
    else if((button & 0xC0)>>6 == 1) /* Else if button state is DOWN */
    {
      /* Set the button image to its DOWN position */
      ezLCD_set_button_state(button & 0x3F, EZLCD_BUTTON_DOWN);
    }
  }
  return button & 0x3F; /*Return the ID of the pressed button, making sure to
                         * mask out the now unneeded status bits.
                         */
}
```

To understand why this code works, we must look at the packet which the ezButton protocol returns over SPI.

The packet is illustrated below:

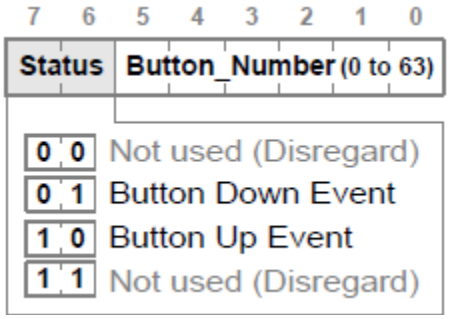


Figure 8: ezButton data packet

As noted earlier, the ezButton has two states, up and down. The two most-significant bits of the packet represent the state and the rest of the bits indicate the button ID (or button number). So in the code above, what we are doing is determining what state we are in based on the 2 MSBs and then returning the button ID when we are finished toggling the button. The code above results in the button going into its down state (and therefore displaying the light-blue image in the case of the calibrate button in **figure 7**) when held down and then going back into its up state (dark-blue) when released.

The cuButton has a similar packet which is discussed in greater detail on the Doxygen page.

Fonts

The ezLCD+ series supports 2 kinds of fonts; proprietary ezLCD bitmap fonts and true-type fonts (.ttf). Each of these will be discussed in some detail below.

Bitmap Fonts

The .ezf files stored on the customization microSD card consist of simple ASCII fonts. Each character is essentially a bitmap image and it cannot be scaled up or down; font size and color are ignored. There is little difference between these fonts and the bitmap images used for buttons, splash screens, backgrounds, etc. They are quick to render, but cannot be customized.

True Type Fonts (TTF)

True Type Fonts are a font standard co-developed by Microsoft and Apple. Unlike the bitmap fonts, these fonts are not ASCII based, but Unicode instead. Unicode supports a much broader array of characters than ASCII (which is essentially limited to the English alphabet, the Arabic numeral system, and some miscellaneous characters and symbols). The Unicode standard allows for just about every character in every language to be represented. It can represent everything from Greek to Chinese and still has room for expansion.

While this system offers an excellent solution to the limited character set of ASCII, we must remember that we are working with an 8-bit microcontroller and can only represent 256 characters in one byte. In order to avoid having to send 2 bytes or more serially when printing a TTF character, the ezLCD offers another solution. It allows us to set a base character. This allows us to still use one-byte numbers to write Unicode characters. The base character is the character which is addressed by the number zero. We can change which character is addressed by 0 with the *ezLCD_set_ttf_unicode_base()* function. This function takes a 2-byte number, allowing us to choose from a large number of base characters. Once a base character is chosen, we can select from the base character and the 255 characters above it to print to the screen.

Suppose we choose the Greek letter Δ as a base character. $\Delta's$ Unicode representation is $0394_{16}\ or\ 916_{10}$. Adding 255 to 916 yields $1171_{10}\ or\ 0493_{16}$, which represents the Cyrillic letter **ghe (ғ)**. So now, with a base of Δ we can access every Unicode character from Δ to ғ using only one-byte numbers.

In another example, the strings shown below are both drawn using numbers from 0 to 256. However, the base is different and therefore the resulting character sequence is different.

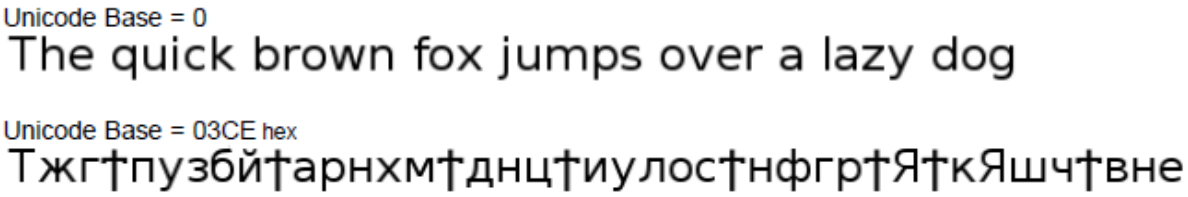


Figure 9: Differences in Unicode Bases

In addition to being able to represent large quantities of characters, TTFs are scalable and can be colored as well. They allow for a far greater amount of customization than their bitmap counterparts. The disadvantage is that they require some pre-processing (done by the ezLCD) before being rendered to the screen whereas the bitmaps do not. This takes some time so rendering TTFs may, in some cases be slow.

Library Overview

Portability

The ezLCD+103 library has been developed to maximize portability among different microcontrollers and compilers. This has been accomplished through extensive macro use and the addition of a user configuration header file. The user configuration file allows the user to change a number of settings such as the SPI port and pins, the target device, and SPI register names and bits. Additionally, as previously mentioned, it contains a list of font and image properties. This file has been pre-configured for the ATMega128 and the IAR compiler; however it may be modified as needed. The rest of the library should not need to be modified so long as this file is properly configured.

The library also make extensive use of typedef's to ensure proper bit-field widths across different microcontrollers. The types **uint8_t** and **uint16_t** define unsigned 8 and 16-bit numbers respectively while **int8_t** and **int16_ t** represent their signed equivalents. Strict ANSI C types were only used where it either made sense or was required (ie. putchar() accepts an ANSI C integer type as required by stdio.h and ezLCD_putchar() accepts a character because it is logical to do so).

File Hierarchy

As was the case with the EA DOGM128, the library's file hierarchy allows for parts of the library to be added or removed as needed. This is done by commenting out the undesired header files from the **ezLCD_103_driver.h** file and detaching them from the project. Unlike the EA DOGM128, this library is far more independent. With several exceptions, essentially any part of the library may be removed. This illustrated by the diagram in the Doxygen page.

Adding the library to your project is simple; merely add the library source and header files to your project and include **ezLCD_103_driver.h** at the top of your source code.

Using the Library Functions

The **main.c** file included with the library exemplifies how the library is used and should be used as a template for other projects. The series of figures below illustrates the result of running **main.c**. A large font size was chosen for photo quality only; it can be resized to something more practical in real applications.

1. Upon power-up, the Stony Brook University logo should appear as the default spash screen

Figure 10: Splash Screen

2. Momentarily, the main menu should appear. This consists of the text box and buttons shown below.

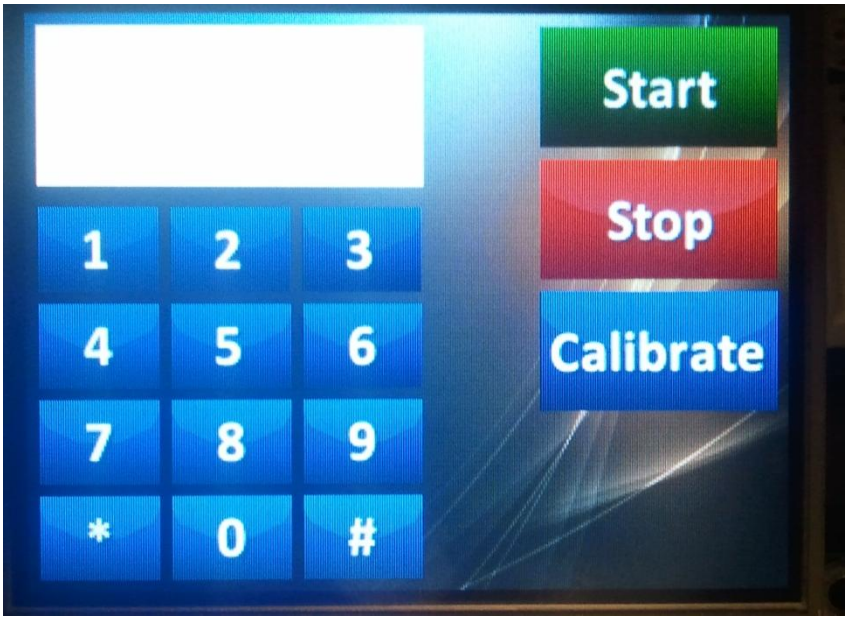


Figure 11: Main Menu

3. Buttons 0-9 will result in their respective number being printed to the white text box.

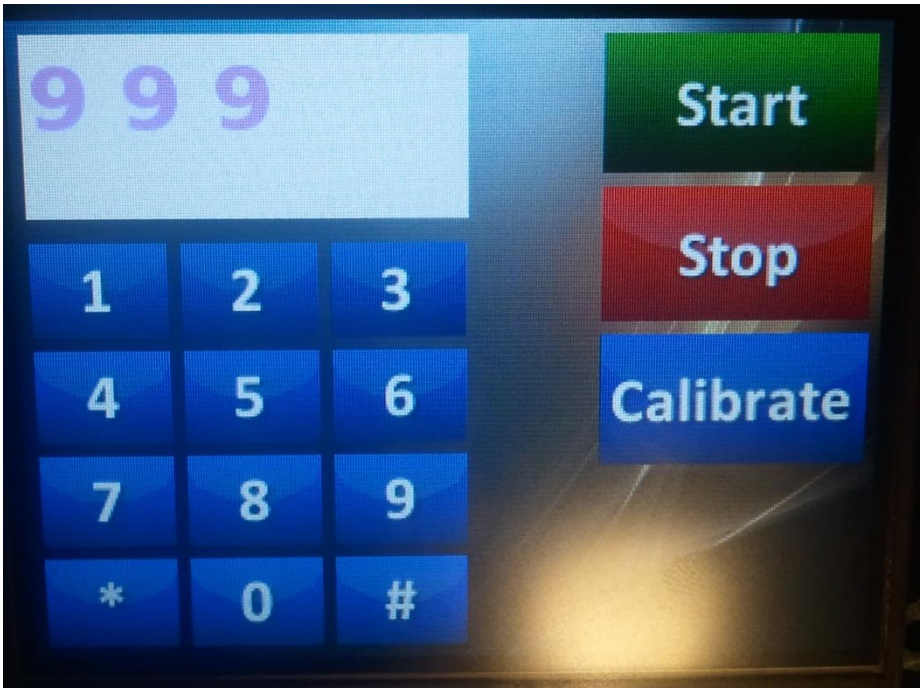


Figure 12: Number Buttons

4. The star and pound buttons illustrate the concept of Unicode base changing and font type changing. The pound button prints a capital letter Δ and the star button prints a using the Waltograph TTF.

Figure 13: Unicode Characters

5. Pressing the **Stop** button removes the keypad and the text box from the display, allowing the user to do something else with that area. Pressing **Start** will restore the number pad and text box back to the screen.

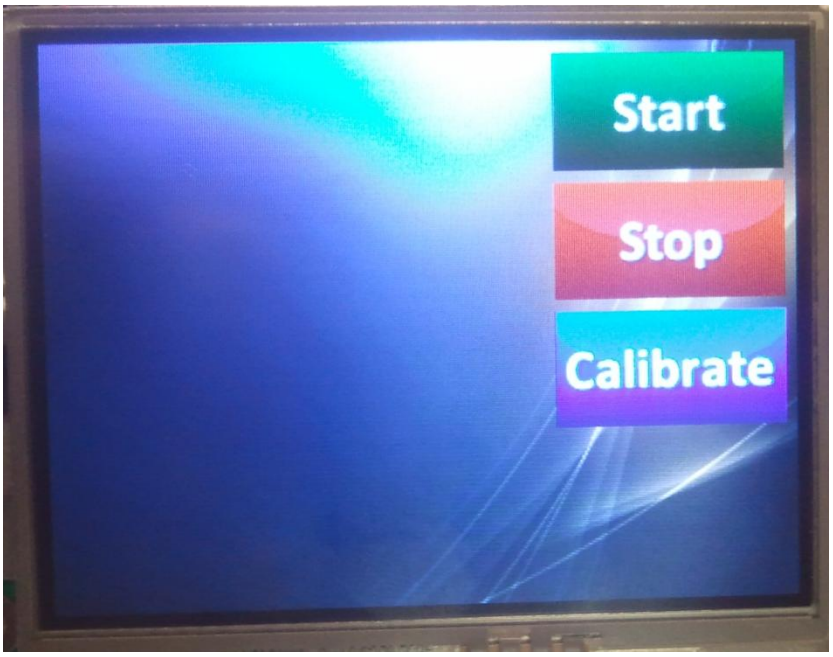
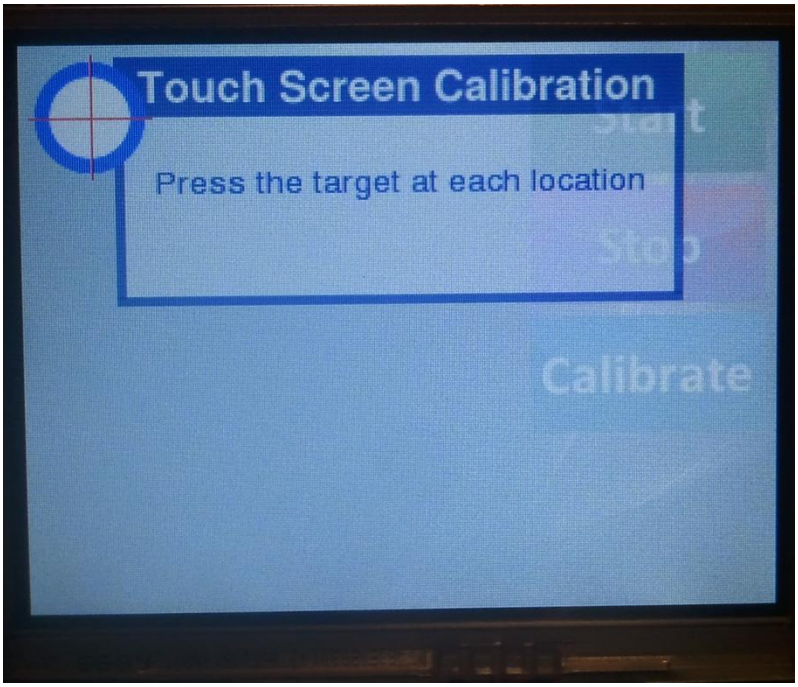


Figure 14: Resulting of pressing the Stop Button

6. The **Calibrate** button calls the calibration routine, which will wait until the screen has been calibrated to return to the previous screen.



Final Thoughts

- Since this display makes heavy use of very small microSD cards, a plan needs to be developed to prevent their loss in the lab.
    - The cards can be pre-configured to fit the project by the instructor and installed on all the displays. However, this does not allow for the students to be creative and design their own buttons, backgrounds, and other images.
    - A better solution would be to keep a **master configuration card** and make copies of it for each lab team. The students can modify these as they wish and the master copy can be used to restore them to their default settings should something go wrong. The displays themselves came with a large number of microSD cards and they should be sufficient for the entire class. The lab team would hand in the card with their breadboards at the end of every lab session so that they do not get lost.
        - Of course, this will also require that every lab station be equipped with a memory card reader.
- Image editing software should probably be installed on the lab computers. Freeware programs such as Gimp or Paint.net should be sufficient.

Conclusion

The ezLCD+103 is a very easy device to use; however, it is also very complex and contains many features which are beyond the scope of this paper (and hence, packed into the Doxygen page instead where they could be explained side-by-side with the code) and some others which have not been implemented yet in this library due to the fact that there wouldn't be a real use for them in ESE 381. For example, the ezLCD+ series feature a Lua interpreter. Lua is an open-source scripting language which may be used to run an entire program on the display itself. Lua scripts are loaded to the system ROM via the microSD card in the same way fonts and images are loaded. It can greatly simplify some operations (the microcontroller would just tell the ezLCD which Lua script to run). However, ESE 381 involves C programming and Lua is not a very popular language to begin with. So this feature was left out. Of course, other features of the 103 (over the 301 and 303) are more useful. The 103 has a powerful 32-bit AVR processor on board (compared with a less capable 16-bit processor) and plenty of memory. It is easily capable of running video! Of course, video may be overkill for ESE 381, but the fast processor and large memory should be able to compensate for the inefficient coding of new embedded C programmers. In addition, since the lab already has a large supply of these screens, there is no need to let them go to waste, especially since the "next generation" of displays do not offer anything new, but rather are aimed at cutting costs instead.

Links

1) https://www.sparkfun.com/products/12009
2) http://www.dabuttonfactory.com
    a. Blue Buttons
        i. http://dabuttonfactory.com/#t=Continue&f=Calibri-Bold&ts=24&tc=ffffff&tshs=1&tshc=222222&it=jpeg&c=0&bgt=gradient&bgc=0728b8&ebgc=011f3b&be=on&w=100&h=50
        ii. http://dabuttonfactory.com/#t=%23&f=Calibri-Bold&ts=24&tc=ffffff&tshs=1&tshc=222222&it=jpeg&c=0&bgt=gradient&bgc=9fb2fc&ebgc=0088ff&be=on&w=50&h=35
    b. Red Buttons
        i. http://dabuttonfactory.com/#t=Stop&f=Calibri-Bold&ts=24&tc=ffffff&tshs=1&tshc=222222&it=jpeg&c=0&bgt=gradient&bgc=e00000&ebgc=6e0404&be=on&w=100&h=50
        ii. http://dabuttonfactory.com/#t=Stop&f=Calibri-Bold&ts=24&tc=ffffff&tshs=1&tshc=222222&it=jpeg&c=0&bgt=gradient&bgc=e00000&ebgc=ff0000&be=on&w=100&h=50
    c. Green Buttons
        i. http://dabuttonfactory.com/#t=Start&f=Calibri-Bold&ts=24&tc=ffffff&tshs=1&tshc=222222&it=jpeg&c=0&bgt=gradient&bgc=09a803&ebgc=012b0c&be=on&w=100&h=50
        ii. http://dabuttonfactory.com/#t=Start&f=Calibri-Bold&ts=24&tc=ffffff&tshs=1&tshc=222222&it=jpeg&c=0&bgt=gradient&bgc=0fdb04&ebgc=00de38&be=on&w=100&h=50
    d. Disabled Buttons (ALL)
        i. http://dabuttonfactory.com/#t=%23&f=Calibri-Bold&ts=24&tc=ffffff&tshs=1&tshc=222222&it=jpeg&c=0&bgt=gradient&bgc=bdbdbd&ebgc=757575&be=on&w=50&h=35

References

1) Earth Computers Tech Inc., “ezLCD+ External Commands Manual” ezLCD+ Series User Guide, [Revised 2008].
2) Earth Computers Tech Inc., “ezLCD+103 Product Manual” ezLCD+103 datasheet, [Revised 2008].