CAB202 Tutorial 8
Display on the LCD Screen

Overview

Peripherals are the most important part of any computer system, and the Teensy boards are no different. The buttons and potentiometer are examples of input peripherals that allow you to provide information to the system. Conversely, the LCD screen and LEDs are output peripherals that allow you to receive information from the system. In last week's tutorial, you began interfacing the Atmega32u4 with two peripherals: the LEDs and the buttons. This week you will allow for debouncing for the buttons and expand into using the LCD screen (a much more complex peripheral). After completing this tutorial, you will be able to leverage the capabilities of peripherals to output advanced information to a user (e.g. characters and words) and respond to user input.

LCD Graphics

The Teensy board uses the Nokia5110-LCD Screen (the datasheet for this can be found in the Blackboard resources). This screen is made up of 48x84 pixels, where a single byte is used to group 8 adjacent column pixels together. Thus we have a total of 6 rows by 84 columns bytes of data, meaning a total of 504 bytes. These bytes are counted sequentially from left to right, top to bottom. Graphics functionality is included in the **`cab202_teensy`** library. An implementation of an LCD operations library is provided in `LCD.h` and `LCD.c`, with basic ncurses-like functionality provided in `graphics.h` and `graphics.c`. An ASCII font is provided in `ascii_font.h`.

Before pixel data can be written to the screen, the display must be initialised so that it can understand the commands provided to its input pins. This has been done for you in the `LCDInitialise()` function. If you wish to understand what exactly is happening in this initialization function, follow the comments and read the appropriate section of the data sheet. Your tutors can also help you with this.

Data is displayed on the LCD by pushing bytes to the display with a serial protocol that aligns with how the screen is initialised. The screen's internal electronics than process this serial data and turn on the appropriate pixel or pixels. An `LCDWrite()` function, which pushes a byte of data to the wherever the LCD cursor is located, has been provided for you. Consider this function for pushing the byte of data below to the screen (with the LCD write position set to 0,0):

```
0b11100111
```

Writing this byte would operate in the top-left hand corner of the screen on a single vertical line of pixels. This line is a single pixel wide and 8 pixels high. This write would turn on the first and last 3 pixels out of the 8. If the same byte of data was pushed to the LCD again, the same pixels would turn on in the adjacent vertical line of pixels (i.e. the same line of pixels would now be 2 pixels wide). This is because the LCD increments the cursor location to where the next byte of data needs to be placed. Although this behaviour is often desirable, it is not always the case. The location of the

cursor can be changed by issuing the appropriate command to the LCD screen. This is to be done in the `LCDPosition()` function.

Understanding the functions requires a sound knowledge of how the hardware operates. To gain the appropriate knowledge, you should read the LCD datasheet with a particular focus on the following sections:

- Sections 1, 2 and 3 on page 3: a nice summary of the LCD screen.
- Section 6 on page 5: explains the labels given to pins which are used in the remainder of the LCD datasheet.
- Sections 7.2, and 7.3 on page 6. Also section 7.7 on page 9.
- Section 8 introduction on page 11, as well as sections 8.1, 8.2 and 8.5, 8.6 on page 15.

The LCD functions have been expanded with graphics functions, that allow characters to be easily drawn at any pixel position on the screen by using a screen buffer that is displayed on the LCD screen using the `refresh()` function.

Setting up the Graphics Library

Download the following code resources from Topic 8 on Blackboard: **`cab202_teensy_v01.zip`**, and **`blank_screen.c`**. This zip folder contains multiple **`*.c`** and **`*.h`** files which you will compile into a library to use. This new library provides many tools for doing a number of different things with the Teensy. This week we will focus on the tools provided for accessing and writing to the LCD screen on the TeensyLCD. This library is similar to ncurses but has a lot less functionality. It's up to you to add the functionality that you require. You will need to compile this just as you did for the ZDK. Run **`make`** in the **`cab202_teeensy`** directory to create the library **`libcab202_teensy.a`**.

Now that you have a compiled version of the `cab202_teensy` library, you can now compile your own program that takes advantage of the functionality within this library. These commands should be familiar from last week's tutorial. The only difference is that you now must link against a library like we have been all semester. For example, if the file **`blank_screen.c`**, was in the same directory as your **`cab202_teensy`** folder, then the following commands would compile a **`blank_screen.hex`** executable:

```
avr-gcc –mmcu=atmega32u4 –Os –DF_CPU=8000000UL blank_screen.c
-o blank_screen.o –L./cab202_teensy –I./cab202_teensy
–lcab202_teensy –lm –std=gnu99

avr-objcopy –O ihex blank_screen.o blank_screen.hex
```

Note: ***when using avr-gcc if you want to use basic math functionality then you must link against the math library (-lm) when compiling!***

Assessable Exercises

NOTE: All questions are to be marked in your allocated prac session only!

1. Collect your Teensy in class (1 mark)

Yes, there's no catch here - attending your tutorial and collecting your Teensy will score you 1 mark.

2. Compile and run the spinning wheel program (1 mark)

Download the **`spinning_wheel.c`** source file from Blackboard. Compile this program, and run it on your Teensy. To successfully compile this program, you **must**:

1. Correctly specify the location of the `cab202_teensy` library with the **`-I`** and **`-L`** flags.
2. Link against both the `cab202_teensy` and `math` libraries with **`-lcab202_teensy`** and **`-lm`** respectively.
3. Correctly request the compiler to compile against the **`gnu99`** C standard.

Note: demonstrating this exercise to your tutor will award you the marks for exercises 1 as well

3. Modify the spinning wheel program to say hi and introduce yourself (1 mark)

Modify the source code used in the previous question. Make the following changes to the text on the screen:

- Replace the "`CAB202`" string with "`Hello <your tutor's name>`".
- Replace the "`Spinning wheel`" string with "`I'm <your student number>`".

Note: demonstrating this exercise to your tutor will award you the marks for exercises 1 and 2 as well

4. Turn those frowns upside down (1 mark)

The **`frowny.c`** source file on Blackboard repeatedly runs through a small loop. A sprite is positioned in the middle of the screen that starts as a smiley face. The image of the sprite then progresses from the smiley face to a frowny face. Reverse this transition, so that the sprite starts with a frowny face and progresses towards a smiley face.

Note: demonstrating this exercise to your tutor will award you the marks for exercises 1 and 2 as well

Non-assessable Exercises

NOTE: These exercises are not assessable. Assessable exercises are in the previous section.

5. Modify the spinning wheel program

Progressively make modifications to the **`spinning_wheel.c`** source file so that when compiled the program now:

- Moves the wheel from the right edge of the screen to the left edge of the screen.
- Has a wheel with 6 spokes.
- Has a wheel that moves and rotates twice as fast.
- Has a wheel that bounces from the left edge to the right edge indefinitely.

6. Modify the frowning program

Progressively make modifications to the **`frowny.c`** source file so that when compiled the program now:

- Bounces the sprite from the top to the bottom of the screen indefinitely.
- Bounces the sprite from the left of the screen to the right of the screen indefinitely. The emoticon should get happier as the sprite gets closer to the right edge, and sadder as it gets closer to the left edge.
- Randomly moves the sprite around the screen, with the emotion of the sprite still controlled by the same conditions as above.
- Add a second sprite to the program, the emotions of the sprites should depend on how far away the sprites are from each other (closer the happier).

Challenge Exercises

If you wish to extend your understanding, here are some other exercises you may wish to attempt:

- Move a sprite (e.g. ‘o’) diagonally across the screen, bouncing when it hits the edge of the screen.
- Move a sprite around in a square, or a circle.
- Create a picture.
- Create an animated picture.