Lab 4e LCD Alarm Clock

This laboratory assignment accompanies the book, Embedded Microcomputer Systems: Real Time Interfacing, by Jonathan W. Valvano, published by Brooks-Cole, copyright © 2000.

Goals
- Optimize an existing hardware/software interface between a LCD display and a microcomputer,
- Design a hardware/software interface between three switches and a microcomputer,
- Design a hardware/software driver for generating single tones on a speaker,
- Implement a digital alarm clock.

Review
- Valvano Chapter 3 on Basic Handshake Mechanisms,
- Valvano Section 8.3 on LCD fundamentals,
- Valvano Section 2.7 on device drivers.
- **`hd44780.pdf`** and **`LCD.pdf`** data sheets

Starter files
- **`RTI`** and **`LCD`** projects

Background

Microprocessor controlled LCD displays are widely used having replaced most of their LED counterparts, because of their low power and flexible display graphics. This experiment will illustrate how a handshaked parallel port of the microcomputer will be used to output to the LCD display. The hardware for the display uses an industry standard HD44780 controller. The low-level software initializes and outputs to the HD44780 controller.

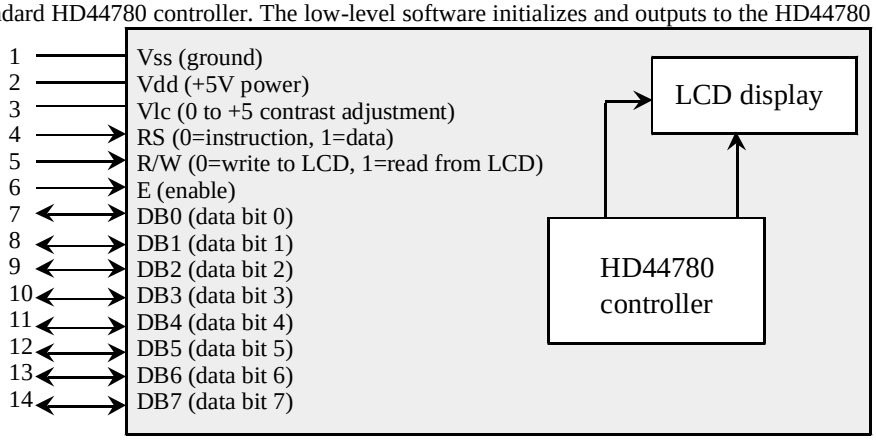

Figure 4.1. 1 by 16 LCD display.

There are four types of access cycles to the HD44780 depending on RS and R/W

RS	R/W	Cycle
0	0	Write to Instruction Register
0	1	Read Busy Flag (bit 7)
1	0	Write data from µP to the HD44780
1	1	Read data from HD44780 to the µP

Two types of synchronization can be used, blind cycle and gadfly. Most operations require 40 µs to complete while some require 1.64 ms. The example implementation shown in the **`LCD`** project uses **`TCNT`** to create the blind cycle wait. YOU ARE REQUIRED TO DEVELOP SOFTWARE THAT READS THE BUSY FLAG. A gadfly interface provides feedback to detect a faulty interface, but has the problem of creating a software crash if the LCD never finishes. You will implement the best interface utilizing both gadfly and blind cycle, so that the software can return with an error code if a display operation does not finish on time (due to a broken wire or damaged display.) Your low-level functions must perform both gadfly (wait for the busy flag to be clear) and blind cycle. For each operation you will return with the error code equal to success if the busy becomes clear, but return with a failure if the busy does not become clear after waiting the appropriate amount of time.

The paragraph discusses issues involved in developing software that will be sold. Because the software for embedded systems is much smaller than software for a general-purpose computer, it is customary to provide

sell copyrighted source files (e.g., **LCD.H** and **LCD.C**) that the user can compile with their application. In a large software system, one typically sells the header file together with precompiled object code. In our embedded system, the compiler will perform the linking. You are encouraged to modify/extend this example, and define/develop/test your own format. Normally, we group the device driver software into four categories. We will use interrupts to interface the I/O devices in the later labs. Interrupt service routines would be classified as protected support software, not directly callable by the user.

1. Data structures: global, protected (accessed only by the device driver, not the user.)

OpenFlag boolean that is true if the display port is open
initially false, set to true by **LCD_Open**
static storage (or dynamically created at bootstrap time, i.e., when loaded into memory)

2. Initialization routines (used/called by the user)

LCD_Open Initialization of display port
Sets **OpenFlag** to true
Initialize hardware, other data structures
Returns an error code if unsuccessful
hardware non-existent, already open, out of memory, hardware failure, illegal parameter
Input Parameters(**display**, **cursor**, **move**, **size**)
Output Parameter(none)
Typical calling sequence

```
if(!LCD_Open(LCDINC+LCDNOSHIFT,LCDNOCURSOR+LCDNOBLINK,
             LCDNOSCROLL+LCDLEFT,LCD2LINE+LCD7DOT)) error();
```

3. Regular I/O calls (called by user to perform I/O)

LCD_Clear clear the LCD display
Returns an error code if unsuccessful. E.g., device not open, hardware failure (happens when a wire is loose)
Input Parameter(none)
Output Parameter(error code)
Typical calling sequence

```
if(LCD_Clear()) error();
```

LCD_OutChar Output an ASCII character to the LCD port
Returns an error code if unsuccessful. E.g., device not open, hardware failure (happens when a wire is loose)
Input Parameter(ASCII character)
Output Parameter(error code)
Typical calling sequence

```
if(LCD_OutChar(letter)) error();
```

LCD_OutString Output a NULL-terminated ASCII string to the LCD port
Returns an error code if unsuccessful. E.g., device not open, hardware failure (happens when a wire is loose)
Input Parameter(pointer to ASCII string)
Output Parameter(error code)
Typical calling sequence

```
if(LCD_OutString("Hello world")) error();
```

4. Support software (protected/`static`, not directly accessible by the user).

outCsr sends one command code to the LCD control/status
Input Parameter(command is 8-bit function to execute)
Output Parameter(none}

wait fixed time delay
Input Parameter(time in microseconds to wait)
Output Parameter(none}

You will write the first main program in order to test and evaluate the device driver software and hardware interface. For a device this simple you should be able to test all modes and all characters. Purposefully cause errors and see if the software gives the appropriate response. For example, the LCD project contains a main program that is used to test the features of the **LCD.H**, **LCD.C** device driver.

Jonathan W. Valvano

The second main program will implement a simple digital alarm clock. You can connect individual push-button switches (see Figure 4.2) to input pins of the 6812, which the user can use to set the current time and the alarm time. The LCD display will be used to display the current time. You are free to implement whatever features you wish, but there must be a way to set the time. The system maintains three global variables **`hour`**, **`minute`**, **`second`**. No SCI input is allowed, and the time parameters must be maintained using the RTI interrupts. In particular, the RTI interrupt service routine should increment `second` once a second, increment `minute` once a minute, and increment `hour` once an hour. The foreground (main) will output to the LCD display, and interact will the operator via the switch inputs. If the correct sequence of switches is pushed, the main program can initialize the values of `hour`, `minute`, `second`.

Figure 4.2. S.P.S.T. momentary, normally open. 0.48" square x 0.18" body. Plunger stands 0.3" above body. Four PC leads on 0.2" x 0.5" centers. CAT# MPB-127, http://www.allelectronics.com.

You can interface a speaker using a NPN transistor like PN2222 to an output port. The resistor controls the loudness of the sound. Please make it quiet, try 1 kΩ. The maximum I_{CE} of the transistor must be larger than 5V/32Ω. The diode is used as a snubber to remove back EMF when the transistor switches off. If you toggle the output pin in the background ISR, then sound will be generated. See Figure 4.3.

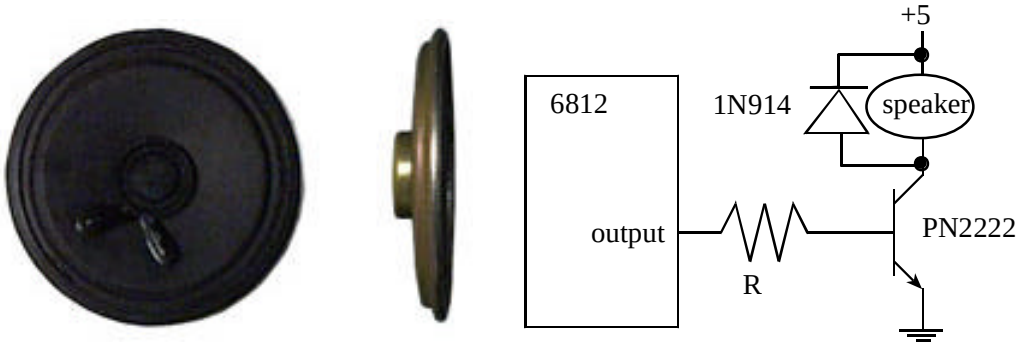

Figure 4.3. 2.25" 32 OHM SPEAKER. 0.38" thick. CAT# SK-232, http://www.allelectronics.com.

Preparation

Show the required hardware connections. Label all hardware chips, pin numbers, and resistor values. Modify the existing low-level LCD device driver to implement the combined blind-cycle/gadfly synchronization. You must have a separate **`LCD.H`** and **`LCD.C`** files to simplify the reuse of these routines. Write one main program that tests all features of the driver.

Write a second main program that implements the digital alarm clock. The interrupt service routine used to maintain time must run as fast as possible. This means you must perform all LCD I/O from the main program. You must be careful not to let the LCD show an intermediate time of 1:00:00 as the time rolls over from 1:59:59 to 2:00:00. You must also be careful not to disable interrupts too long (more than one RTI interrupt period), because a time error will result if any RTI interrupts are skipped. You will have to do some 32-bit math to maintain the exact time. For example, if the RTI interrupts at 30.517Hz, then interrupts are requested at exactly 32768 µs. One method is to add 32768 to a 32-bit **`counter`**. When the **`counter`** exceeds 1 million, increment **`second`** and subtract 1 million from the **`counter`**.

Procedure

You should look at the +5 V voltage versus time signal on a scope when power is first turned on to determine if the LCD "power on reset" circuit will be properly activated. The LCD data sheet specifies it needs from 0.1 ms to 10 ms rise time from 0.2 V to 4.5 V to generate the power on reset. Connect the LCD to your microcomputer.

Use the scope to verify the sharpness of the digital inputs/outputs. If needed, adjust the contrast potentiometer for the best looking display. Test the device driver software and two main programs in small pieces.

Deliverables (exact components of the lab report)

A) Objectives (1/2 page maximum)

B) Hardware Design

LCD interface, showing all external components

speaker interface, showing all external components

C) Software Design (no software printout in the report)

A call-graph illustrating the modularity of the hardware/software components of the alarm clock

D) Measurement Data

Plot the LCD supply voltage versus time as the system is powered up

Plot the speaker voltage versus time during an alarm sound

E) Analysis and Discussion (1 page maximum)

Checkout

Using the first main program, you should be able to demonstrate all the "cool" features of your LCD display system. Next, download the digital alarm program. Demonstrate that your digital alarm clock is stand-alone by turning the power off then on. The digital alarm clock should run (the time will naturally have to be reprogrammed) without downloading the software each time.

Your software files will be copied onto the TA's zip drive during checkout.

Hints

1) Make sure the 14 wires are securely attached to your board.

2) One way to test for the first call to open is to test the direction register. After reset, the direction registers are usually zero, after a call to open, some direction register bits will be one.

3) Observe the starter files in the `LCD` project. These C language routines only output to Port H/J. Notice that they do not perform any input (either status or data), therefore they leave `DDRJ=0xFF`, `DDRH=0xFF`. Because you have to include inputs, then you must toggle `DDRH`, so that `PORTH` is an output for writes and an input for reads.

4) Although many LCD displays use the same HD44780 controller, the displays come in various sizes ranging from 1 row by 16 columns up to 4 rows by 40 columns.