Lab 3f LCD Alarm Clock

This laboratory assignment accompanies the book, Embedded Microcomputer Systems: Real Time Interfacing, Second edition, by Jonathan W. Valvano, published by Thomson, copyright © 2006.

Goals
- Optimize an existing hardware/software interface between a LCD display and a microcomputer,
- Design a hardware/software interface for three switches using key wakeup,
- Design a hardware/software driver for generating single tones on a speaker,
- Measure supply current necessary to run the embedded system,
- Implement a digital alarm clock.

Review
- Valvano Chapter 3 on Basic Handshake Mechanisms,
- Valvano Section 3.2.2 on key wakeup,
- Valvano Section 8.3 on LCD fundamentals,
- Valvano Section 2.7 on device drivers.
- http://users.ece.utexas.edu/~valvano/Datasheets/LCDOptrex.pdf
- http://users.ece.utexas.edu/~valvano/Datasheets/LCD.pdf
- http://users.ece.utexas.edu/~valvano/Datasheets/LCD100.pdf
- http://users.ece.utexas.edu/~valvano/Datasheets/LCDHD44780.pdf

Starter files
- **OC_DP512.zip** and **LCD_9S12.zip** projects, **Lab3f_Artist.sch** diagram

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. Even though our 9S12DP512 has many I/O pins, you will implement the 4-bit data interface.

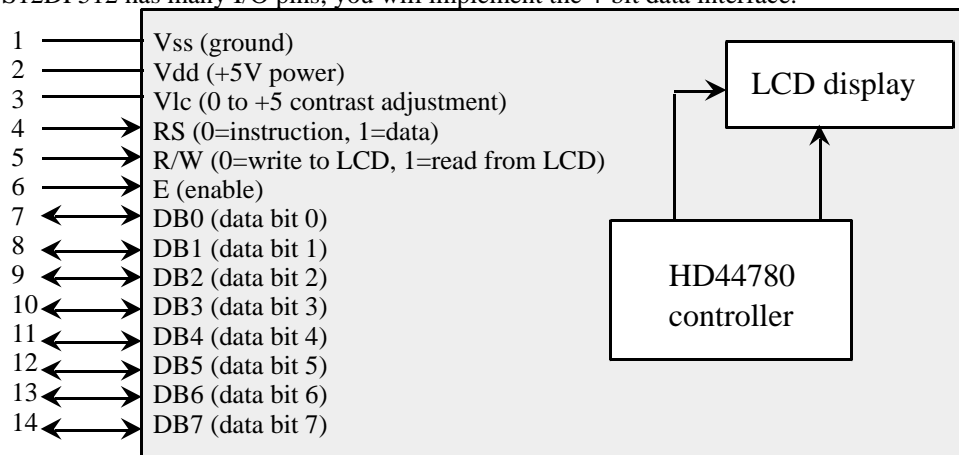| 1 | Vss (ground) |
2	Vdd (+5V power)
3	Vlc (0 to +5 contrast adjustment)
4	RS (0=instruction, 1=data)
5	R/W (0=write to LCD, 1=read from LCD)
6	E (enable)
7	DB0 (data bit 0)
8	DB1 (data bit 1)
9	DB2 (data bit 2)
10	DB3 (data bit 3)
11	DB4 (data bit 4)
12	DB5 (data bit 5)
13	DB6 (data bit 6)
14	DB7 (data bit 7)

Figure 3.1. Physically, the LCD is arranged 1 row by 16 columns, but logically (the way the software views it) the LCD is arranged in 2 rows by 8 columns.

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 could be used with the LCD, blind-cycle and busy-wait. Most operations require 40 µs to complete while some require 1.64 ms. The starter file shown in the **LCD** project uses **TCNT** to create the blind-cycle wait for all LCD operations. In your software, however, all commands executed during the initialization program will use *blind-cycle* synchronization. After your initialization program is executed, your subsequent LCD operations MUST use both *busy-wait* synchronization (that reads the **Busy Flag** over and over, then returns with a

TRUE if the LCD operation completes) and blind-cycle synchronization (that waits a reasonable amount of time, then returns with a FALSE if the LCD operation has not finished). A busy-wait interface by itself executes quickly, but has the problem of creating a software crash if the LCD never finishes. You will implement an excellent solution utilizing both busy-wait 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.) For each LCD operation you will return with the error code equal to TRUE if the busy becomes clear, but return with a FALSE if the busy does not become clear after waiting the appropriate amount of time.

The paragraph discusses issues involved in developing device driver that will be sold. Because the software for embedded systems is much smaller than software for a general-purpose computer, it is customary to sell copyrighted source files (e.g., **LCD.H** and **LCD.C**) that the user can compile into their application. In a large software system, one typically sells the header file together with precompiled object code. In an 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. Interrupt service routines would be classified as protected support software, not directly callable by the user.

1. Data structures: global, private (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**
    statically-allocated, but private in scope

2. Initialization routines (public functions called by the user)
LCD_Open   Initialization of display port
    Sets **OpenFlag** to true
    Initialize hardware, other data structures
    Uses blind cycle synchronization
    Takes 30 ms to finish, allowing time for the LCD to warm up
    Sets an internal code if unsuccessful
        already open, illegal parameter
    Input Parameters(none)
    Output Parameter(none)
    Typical calling sequence
            LCD_Open();
3. Regular I/O calls (public functions called by user to perform I/O)
LCD_Clear clear the LCD display
    Sets an internal code if unsuccessful. E.g., device not open, hardware failure (happens when a wire is loose)
    Input Parameter(none)
    Output Parameter(none)
    Typical calling sequence
            LCD_Clear();
LCD_OutChar Output an ASCII character to the LCD port
    Sets an internal code if unsuccessful. E.g., device not open, hardware failure (happens when a wire is loose)
    Input Parameter(ASCII character)
    Output Parameter(none)
    Typical calling sequence
            LCD_OutChar(0x65);
LCD_OutString Output a NULL-terminated ASCII string to the LCD port
    Sets an internal code if unsuccessful. E.g., device not open, hardware failure (happens when a wire is loose)
    Input Parameter(pointer to ASCII string)
    Output Parameter(none)
    Typical calling sequence
            LCD_OutString("Hello world");
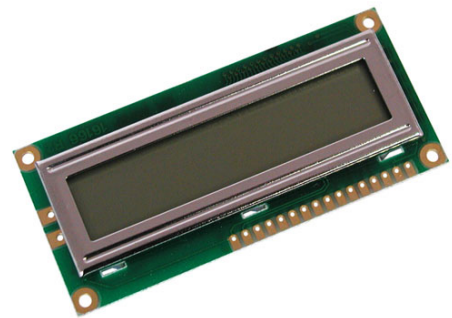**LCD_GoTo** Move the cursor to a particular row and column
    Sets an internal code if unsuccessful. E.g., device not open, hardware failure (happens when a wire is loose)
    Input Parameters (**row**, **column**) (feel free to make the first row/column be 0 or 1, as you wish)
    Output Parameter(none)
    Typical calling sequence
            LCD_GoTo(2,1); // second row, first column

Jonathan W. Valvano

LCD_ErrorCheck Check to see if the LCD driver has had any errors

Returns an error code if LCD has had any errors since initialization or since the last call to **ErrorCheck**

Input Parameter(none)

Output Parameter(error code)

Typical calling sequence

```
Err = ErrorCheck();
if(Err) Handle(Err);
```

4. Support software (private functions static, not directly accessible by the user).

readBusy reads the LCD status

Input Parameter(none)

Output Parameter(true if busy, false if not busy}

outCsrBlind sends one command code to the LCD control/status, using just 40 μs blind cycle wait

Input Parameter(command is 8-bit function to execute)

Output Parameter(none}

outCsr sends one command code to the LCD control/status, using both busy-wait and blind cycle

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. Consider what would happen if any of the LCD wires were disconnected. For example, the LCD project contains a main program that is used to test the features of the **LCD.H**, **LCD.C** device driver.

The second main program will implement a simple digital alarm clock. You can connect individual push-button switches (see Figure 3.2) to input pins of the 9S12, 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 output compare interrupts. In particular, the output compare 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 3.2. S.P.S.T. momentary, normally open.
http://users.ece.utexas.edu/~valvano/Datasheets/B3F-1059.pdf

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 speaker has inductance, so the 1N914 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 3.3. A grade of B can be achieved by satisfying these minimum specifications. At a minimum, your clock must be able to

1) display hours and minutes,
2) allow the operator to set the current time,
3) allow the operator to set the alarm time,
4) make a sound at the alarm time,
5) allow the operator to stop the sound.

Excellent grades are reserved for systems with extra features and are easy to operate. One example of extra features is using key wakeup interrupts, so the entire alarm clock runs in the background.

Jonathan W. Valvano

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/busy-wait 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.
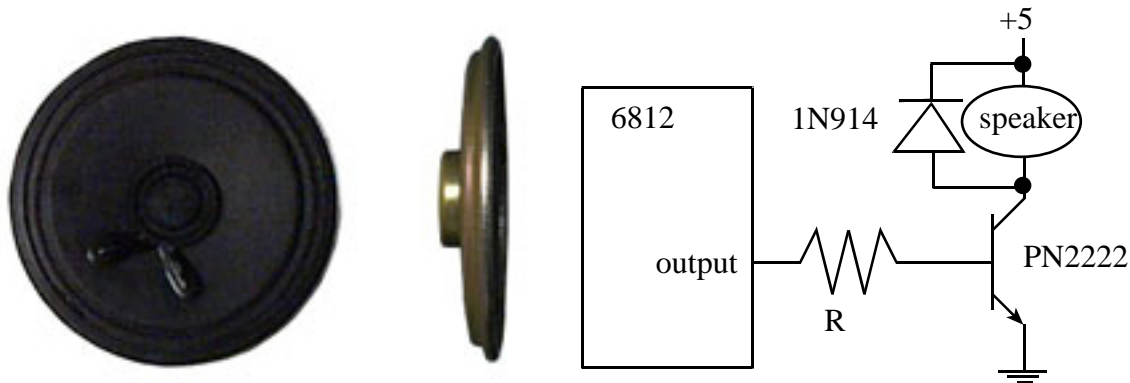

Figure 3.3. 2.25" 32 OHM SPEAKER. 0.38" thick. CAT# SK-232, http://www.allelectronics.com.

   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 output compare interrupt period), because a time error will result if any output compare interrupts are skipped. If you were to use the RTI interrupt, then you would 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**. However, with the output compare interrupts you will be using, getting incrementing the **second** variable exactly once a second is straight-forward. Figure 3.4 shows the data flow graph of the alarm clock.
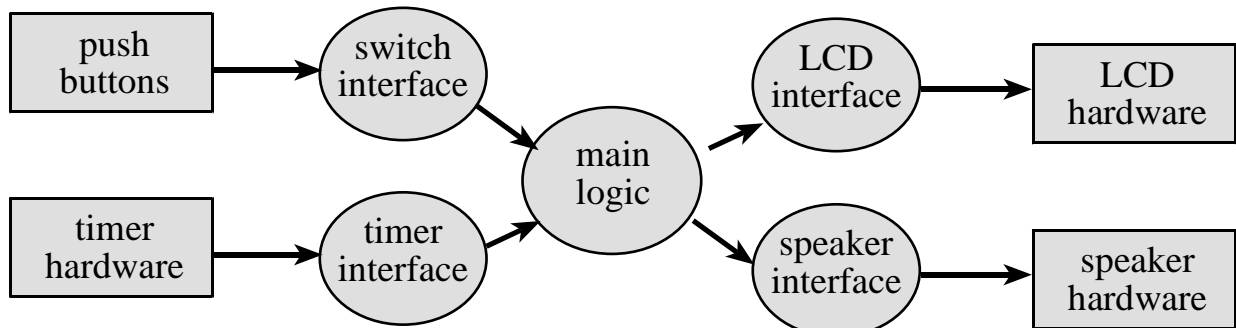

Figure 3.4. Data flows from the timer and the switches to the LCD and speaker.

Figure 3.5 shows a possible call graph of the system. Dividing the system into modules allows for concurrent development and eases the reuse of code.

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. Because the "power on reset" is typically not satisfied, most systems employ the software reset operation as specified in the data sheets for the HD44780-based LCD (see Figure 26 on page 47 of the data sheet LCDHD44780.pdf)
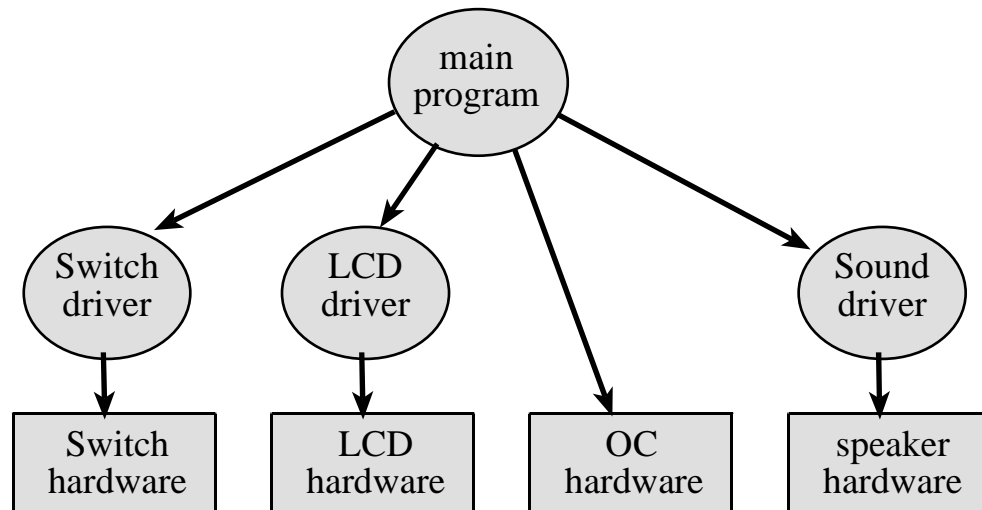
Jonathan W. Valvano

Figure 3.5. A call graph showing the four modules used by the alarm clock.

Remove the wall wart power cable and carefully power your alarm clock system using a lab power supply (not on the +5V pin on the protoboard but the 2-pin male header labeled J1-PWR on the board itself). Set the voltage to +6V and measure the required current to run the alarm clock. Take a measurement with and without the alarm sounding. Double check the positive and negative connections before turning it on. If you are at all unsure about this measurement ask your TA for help.

When reading and writing permanently-allocated variables in more than one thread, one must worry about critical sections. Recall that I/O ports, like PTT are considered permanently-allocated variables for the purposes of critical section analysis. In this lab there is one foreground thread, and two background threads (timer and key wakeup). Identify all the permanently allocated variables that have read or write access by more than one thread. Next, consider what would happen if the interrupt occurred between any two instructions of the main program. One common critical section is a display that reads 11:13:59, 11:13:59, 11:13:59, 11:13:59, **11:13:00,** 11:14:00, 11:14:00, 11:14:00 as the seconds field rolls over and the minute field increments. Look for critical sections, and if you find any remove them. Be ready to prove to your TA that your system has no critical sections.

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 (a hardcopy software printout is due at the time of demonstration)
         If you organized the system different than Figure 3.4 and 3.5, then draw its data flow and call graphs
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
         Measurements of current required to run the alarm clock, with and without the alarm
E) Analysis and Discussion (give short 1 or two sentence answers to these questions)
1) Give two ways to remove a critical section.
2) How long does it take to update the LCD with a new time?
3) What would be the disadvantage of updating the LCD in the background ISR?
4) Did you use the LCD clear function? If so, how could you have redesigned the LCD update to run much faster?
5) Assuming the system were battery powered, list three ways you could have saved power.

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.

Jonathan W. Valvano

A hardcopy printout of your software will be given to your TA, and graded for style at a later time.

Hints

0) Connecting or disconnecting wires on the protoboard while power is applied may damage your board.

1) Make sure the 14 wires are securely attached to your board. Some LCD displays may have 16 pins; if so, you should ignore pins 15 and 16.

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

3) Observe the starter files in the **LCD** project. These C language routines only output to **PTH**. Notice that they do not perform any input (either status or data), therefore they leave **DDRH=0x7F**. Because you have to include inputs, then you must toggle **DDRH**, so that **PTH** bits 3-0 are output for writes and an input for reads. The starter file does perform the proper software reset sequence as described on Figure 26 on page 47 of the data sheet LCDHD44780.pdf

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. The LCD display used in this lab is physically 1 row by 16 columns it is electrically configured as 2 rows by 8 columns.