Lab Overview

In this lab assignment, you will do the following:

- Add an LCD and a serial EEPROM to the hardware developed in Labs #1, #2 & #3.
- Write device drivers for the LCD and EEPROM.
- Use C pointers to access the LCD as a memory-mapped peripheral.
- Implement a bit-banged interface to the EEPROM.
- Write assembly and C programs to implement a user interface and perform user tasks.
- Write simple assembly and C programs to test EEPROM accesses.
- Gain experience in code integration.
- Continue learning how to use SDCC (and perhaps Code::Blocks or makefiles).

Students must work individually and develop their own original and unique hardware/software.

The signature due date for this lab assignment is **Friday, November 13, 2015 (Required Elements) and Wednesday, November 18, 2015 (Supplemental Elements)**.

The submission due date for this lab is **11:59pm Thursday, November 19, 2015**.

The cutoff date for this lab is **Wednesday, December 2, 2015**.

This lab is weighted as ~20% of your course grade.

Required elements are necessary in order to meet the requirements for the lab. Supplemental/challenge elements of the lab assignment may be completed by the student to qualify for a higher grade, but they do not have to be completed to successfully meet the requirements for the lab.

The highest possible grade an ECEN 5613 student can earn on this assignment without completing any of the supplemental elements is a 'B-'. ECEN 5613 students will have to complete the supplemental elements and attempt at least some of the challenges in order to compete for the highest grades. To avoid any late penalties, ECEN 5613 students must obtain a TA's signature on their work by the specified signature due dates for required and supplemental elements.

If you are up for an engineering challenge and want to learn more, then attempt the optional challenge(s). You do not need to complete optional elements in order to get a signature; however, completing optional elements with good results will help your work stand out. **Students must prioritize work on required elements and supplemental elements over challenge elements.**

All items on the signoff sheet must be completed to get a signature, but partial credit is given for incomplete labs. Note that receiving a signature on the signoff sheet does not mean that your work is eligible for any particular grade; it merely indicates that you have completed the work at an acceptable level.

NOTE: **The quality of your user interfaces and demo will impact your score on the lab.** Your goal should be to ensure that the user has a successful and positive experience with your software. Your code should handle error conditions gracefully (e.g. user input values outside the allowed range). Top scores are reserved for those students who submit outstanding implementations, including coding style.

Lab Details

1. Review the homework assignments associated with this lab.
2. Review the data sheets for the Optrex DMC 20434 LCD and the SED1278F (or Hitachi HD44780U) LCD controller. Review the document "SED1278F Technical Manual Errata".

   Refer to the LCD Guide ("Adding an LCD (with an HD44780 LCD controller) to your board") available on the course web site for further ideas and information on interfacing to the LCD module. It contains some very important notes, including one regarding errors in the LCD data sheet.
3. **[Required Element[1]]** Devise a way to securely mount the LCD **and** properly connect all of the data lines to your board. It may take you a little time to devise a good physical interface, so don't wait too long before getting started. Wire can be used to easily attach the LCD to your board without requiring any drilling (remember the previous warnings against drilling holes in the PCB).

   Data Lines: Most LCDs will have only 14 pins. LCDs with a backlight will have 16 pins, with two for the backlight. One option for connecting data lines is to use a 14-pin strip header or SIPP wire wrap socket. You may also attach the LCD through a ribbon cable to a 14- or 16-pin DIP socket on your board. A sturdy data line connection using a strip header can make it easy to mount the LCD.
4. **[Required Element[1]]** Design and implement your LCD circuit. Your LCD must be memory mapped in the address space reserved for peripherals (this is an example of memory mapped I/O), and will be accessible using the MOVX instruction (and via a pointer variable in C). The LCD contrast (V_{EE}) can sometimes be grounded, but you probably need to use a potentiometer or resistor divider to control the contrast so that you can see characters on the screen. The LCD in the parts kit has 14 pins/lines which must be connected.

   The eight data signals on the LCD must be connected to the data lines on Port 0 of the 8051. Ensure that the E signal on the LCD is high **only** when reading from or writing to the LCD.

NOTE: When asking the TA's and instructor for help with your program via e-mail, be sure to attach all relevant files to your e-mail. Attach the commented source files (.c, .h), and the .rst, .mem, and .map files. If your question involves external peripherals, you will also need to attach a PDF of your schematic.

5. **[Required Element[1]]** Implement an LCD device driver with the following C functions:

- ```
  // Name: lcdinit()
  // Description: Initializes the LCD (see Figure 25 on page 212
  //  of the HD44780U data sheet).
  void lcdinit()
  ```

- ```
  // Name: lcdbusywait()
  // Description: Polls the LCD busy flag. Function does not return
  //  until the LCD controller is ready to accept another command.
  void lcdbusywait()
  ```

- ```
  // Name: lcdgotoaddr()
  // Description: Sets the cursor to the specified LCD DDRAM address.
  //  Should call lcdbusywait().
  void lcdgotoaddr(unsigned char addr)
  ```

- ```
  // Name: lcdgotoxy()
  // Description: Sets the cursor to the LCD DDRAM address corresponding
  //  to the specified row and column. Location (0,0) is the top left
  //  corner of the LCD screen. Must call lcdgotoaddr().
  void lcdgotoxy(unsigned char row, unsigned char column)
  ```

- ```
  // Name: lcdputch()
  // Description: Writes the specified character to the current
  //  LCD cursor position. Should call lcdbusywait().
  void lcdputch(char cc)
  ```

- ```
  // Name: lcdputstr()
  // Description: Writes the specified null-terminated string to the LCD
  //  starting at the current LCD cursor position. Automatically wraps
  //  long strings to the next LCD line after the right edge of the
  //  display screen has been reached. Must call lcdputch().
  void lcdputstr(char *ss)
  ```

NOTE: I prefer you to write your own code for these routines. However, a variety of LCD routines and libraries suitable for SDCC are available on the web. You may use these libraries as long as your code contains clear documentation of how you obtained, utilized and/or modified them. Each of your code files must have a file header which identifies all authors of the code. (You already know this is the standing expectation in this class with regard to borrowed code.) **You must have a complete understanding of how all the code works**.

6. **[Required Element[1]]** Write a simple program that uses your LCD driver to prove that the six required functions are implemented correctly. Choose the sequence carefully so that it is easy for the TA to see that each function did its job correctly during the demonstration. This program is just test code and does not need to be completely robust, as long as it adequately demonstrates the functionality of each of the LCD functions above.

7. **[Required Element[1]]** Using a logic analyzer, prove that your LCD control signal timing is correct. Show the timing between the E, RS, R/W*, and data signals as measured at your LCD interface.

- **A logic analyzer screen capture or a simple hand sketch of these timing relationships and values must be turned in with your lab, along with your timing analysis. You may be able to use the floppy diskette from the tool kit for the bench top logic analyzer screen capture, if you have a PC with a floppy drive. You won't need a floppy disk if you use the LogicPort logic analyzers.**

You should also be able to prove that the LCD E control signal goes high only when the LCD is being accessed. You can verify this by running code which does not access the LCD and by triggering the logic analyzer on E going high. If E goes high during this test, then your implementation is incorrect. You may also be able to test this by using Paulmon2.

8. Read the EEPROM Guide "Adding an NM24C04 (or NM24C16) EEPROM to your board", available on the course web site. It has ideas and information on interfacing to the I²C EEPROM.

   Read the data sheet for the serial EEPROM included in your parts kit (e.g. Microchip 24LC16 or Fairchild-National Semiconductor NM24C16). You may also want to read Fairchild Application Note AN-794.

   [Optional, but recommended] Review Microchip app notes AN536, AN572, AN614 and AN709.

9. **[Required Element[1]]** Design and implement your EEPROM circuit. Your EEPROM should be connected to two unused port pins on Port 1 or Port 3. Note that since you are connecting to the EEPROM using port pins, the EEPROM does not consume any 8051 address space.

NOTE: In the next step, your EEPROM driver code may require use of specific port pins

10. **[Required Element[1]]** Implement an EEPROM I²C device driver with the ability to write and read a byte at any EEPROM I²C address using function calls from C. The underlying drivers may be in assembly if you wish, but the functions must be accessible from C. It does not matter what you name the functions. For example, you might implement the following two functions.

```
int eebytew(addr, databyte)  // write byte, returns status
int eebyter(addr)            // read byte, returns data or status
```

NOTE: A variety of I²C routines and libraries suitable for SDCC are available on the web – a quick web search would be beneficial. You may use these libraries as long as your code contains clear documentation of how you obtained, utilized and/or modified them. (You already know this is the standing expectation in this class with regard to borrowed code.)

11. **[Required Element[1]]** Verify that you can write data to and read data from the EEPROM using your I²C device driver and **verify the stored data is correct after cycling power**.

12. **[Required Element[1]]** Use a logic analyzer to prove that your byte write function sends the correct signals and has the correct I²C timing. Note that the LogicPort logic analyzer has an I²C interpreter that you might find useful as you debug your I²C bus transactions.

- **A simple hand sketch or a logic analyzer screen capture of these timing relationships and values must be turned in with your lab, along with your timing analysis. You may be able to use the floppy diskette from the tool kit for the benchtop logic analyzer screen capture, if you have a PC with a floppy drive. You won't need a floppy disk if you use the LogicPort logic analyzers.**

13. [Optional] Use the I²C triggering program on the Agilent 54622D oscilloscope to trigger on a write or read frame on the bus. Display SCL and SDA on the oscilloscope screen and verify that the transaction is for the address you intended. Verify that your rise and fall times fall within the limits given in the I²C specification. Alternatively, use a logic analyzer to trigger on a bus transaction.

14. [**Required Element**[1]]

NOTE: For this lab, all your code should be integrated – this will provide experience with integrating much functionality into a single program, and will also reduce signoff times since only a single program must be stored in the flash memory on your processor. Your demo and submission should be one well-integrated program, but the program can be modularized and consist of multiple code and header files.

Provide a well-designed menu on the PC terminal emulator screen which allows the user to:

- **Write Byte**: Enter an EEPROM address and a byte data value in hex. If the address and data are valid, store the data into the EEPROM. The program must allow any hex value from 0x00 to 0xFF to be programmed into **any** location in the EEPROM. Do not make the user type in "0x" before the address or data hex value.
- **Read Byte**: Enter an EEPROM address in hex. If the EEPROM address is valid, display on the PC screen in hex the contents of the EEPROM address, using the format "AAA: DD". Do not make the user type in "0x" before the address hex value.
- **LCD Display**: Enter an EEPROM address in hex and a row number 'Y' from the set {0,1,2,3}. If the EEPROM address is valid, display on the LCD display in hex the EEPROM address and the contents of the EEPROM address, using the format "AAA: DD", positioned on the LCD at (row,column) = (Y,0). Data from up to four EEPROM addresses can be seen on the LCD screen at any one time, depending on how many times the user has selected **LCD Display**. Do not make the user type in "0x" before the address hex value. This function must utilize the lcdgotoxy() device driver function.
- **Clear LCD**: Clear the LCD display.
- **Hex Dump**: Enter a start address and end address in hex. If the entered values are valid, read the contents of the EEPROM from the start address to the end address and display the data on the PC screen in hexadecimal, with a maximum of 16 bytes of data per line, in the following format:
  AAA: DD DD DD DD DD DD DD DD DD DD DD DD DD DD DD DD
  This format is similar to what you see when using the device programmer or when dumping memory contents using PAULMON2, where AAA is the starting address (in hex) for each block of 16 data values DD (in hex). The first memory cell in the EEPROM is address 0x000. You should be able to leverage code from Lab #3. Do not make the user type in "0x" before the address hex values.
- **DDRAM Dump**: Reads the entire contents of DDRAM and displays it in hex on the PC screen in a clean and logical format.
- **CGRAM Dump**: Reads the entire contents of CGRAM and displays it in hex on the PC screen in a clean and logical format.

The user must be able to execute menu items in any order (the program should not include any dependencies on the order in which a user selects menu items).

Your code should not ignore ACK's during I2C transactions.

[1] Required elements are necessary in order to meet the requirements for the lab. Supplemental, optional, and challenge elements of the lab assignment may be completed by the student to qualify for a higher grade, but they do not have to be completed to successfully meet the minimum requirements for the lab.

15. **[Supplemental Element[1]]:**

NOTE: The code for this element must be integrated into the previous C programs above. Your demo and submission should be one well-integrated program.

- In the bottom right corner of the LCD, continuously display the elapsed time since your program started running using the format "MM:SS.S", where MM is the number of elapsed minutes and SS.S represents the seconds to one-tenth of a second accuracy. For example, 5.1 seconds would be displayed as "00:05.1" and 64.3 seconds would be displayed as "01:04.3".
- Provide additional **Clock** menu options to stop the elapsed time clock, to restart the clock, and to reset the clock back to "00:00.0".

NOTE: Make sure that the cursor location is correctly stored before and restored after any ISRs.
NOTE: **If using SDCC, read the "interrupt" sections of the SDCC user manual carefully, and remember the correct use of 'volatile' and 'critical'.** Be careful when using variables from within the context of an ISR. This includes any functions that your ISR calls. Do not use printf/sprintf in an ISR (note that printf and sprintf share code).

NOTE: This supplemental element is an addition to the previous required element**. The required and supplemental code must be integrated together.** The elapsed timer must work correctly while simultaneously allowing all the menu options in the previous C program to work correctly.

NOTE: If you get this supplemental element signed off, don't turn in separate versions of code for both the required part and the supplemental part - just submit one integrated version.

16. **[Supplemental Element[1]]:**

NOTE: The code for this element must be integrated into the previous C programs above. Your demo and submission should be one well-integrated program.

Create Custom Character: Design and implement C routines which allow the creation of custom LCD characters using CGRAM. Implement the following function:

```
// Name: lcdcreatechar()
// Description: Function to create a 5x8 pixel custom character with
// character code ccode (0<=ccode<=7) using the row values given in
// the 8-byte array row_vals[].
void lcdcreatechar(unsigned char ccode, unsigned char row_vals[])
```

Provide a way for users to enter and display their own customer characters. A good custom character generation routine user interface should:

(a) accept values from the user representing the pixel pattern for each row of the custom character (the design can choose to allow either strings of bits or hex values representing each row of the character)

(b) display the current state to the user on the terminal screen after each row is entered

As part of your demo, show that you have created some fun logo using custom characters. For example, you could use several custom characters grouped together to create a pixel map of the CU logo.

17. **[Supplemental Element[1]]:**

NOTE: The code for this element must be integrated into the previous C programs above. Your demo and submission should be one well-integrated program.

- Read the PCF8574 I²C I/O expander data sheets and application notes available from the course web site. Integrate the chip into your embedded system, sharing the I2C bus with the EEPROM, and prove that you can configure some of its I/O pins to work as inputs and other pins to work simultaneously as outputs. Your parts kit already included a 16-pin wire wrap socket that could be used with the I2C expander chip. You can purchase another wire wrap socket if necessary.

  Provide a software interface that allows you to configure the pins individually as inputs or outputs, and also to check the status of the pins and to write to the pins that are outputs. Remember to use a bit mask in software when interacting with specific pins on the chip.

  To demonstrate your I/O expander, implement a stopwatch timer that displays total elapsed time and individual lap time. Configure a port pin on I/O expander as input using a pushbutton switch (or any other mechanism). Use the interrupt signal from the I/O expander to notify the processor of a button press. Make sure you choose appropriate triggering (level- or edge-) for the interrupt on the processor.

  When the pushbutton is pressed, the stopwatch starts counting total elapsed time and also lap time upwards from time 00:00:0. Then, after a button press, the lap time is stopped and displayed on the LCD, and the next lap starts from 00:00:0 on the next line. The stopwatch can time a maximum of 4 laps—thus if 4 laps are displayed, the program should not calculate or display any additional values unless the timer and laps are reset.

18. **[Supplemental Element[1]]:**

NOTE: The following routines must be integrated into the previous C programs above. Your demo and submission should be one well-integrated program.

- Modify your EEPROM I²C device driver to include a new function named `eereset()`:

```
// Name: eereset()
// Description: Performs a software reset of the I2C EEPROM using an
// algorithm that conforms to Microchip application note AN709.
void eereset()
```

  Use a logic analyzer to prove that eereset sends the correct sequence and has the correct I2C timing. Show the trace on the logic analyzer to the TA during signoff. You do **not** need to print/submit the trace with your report. Demonstrate that you understand the eereset code during your sign-off.

- Watchdog timers are an important feature in many embedded systems and allow the system to reset itself in the event that code execution strays from the desired path.

  Configure the Atmel hardware watchdog timer (WDT) to reset your system if a software upset occurs (e.g., a system hang condition is detected). Implement code that correctly services the WDT and prevents it from resetting the system when software execution is normal. Implement a solution that mimics a system hang condition and prove that the watchdog timer correctly resets your system in this case.

- **[Challenge]** Provide an option for the user to **Measure LCD DDRAM Search Times.** The user specifies a string to search for on the LCD screen. The program then reads the LCD DDRAM and if there is a match found for the string, the starting memory location of the string is returned and is displayed on the terminal. If there are multiple occurrences of the string, all the addresses must be returned and displayed on the terminal. During this search, an internal timer must be used to time the search and after the search is complete the time to search must be displayed on the terminal. The implementation must search the LCD controller DDRAM for the string (for this assignment, it is not acceptable to keep an SRAM copy of the LCD display and perform the search within the SRAM).

- **[Challenge]** Provide an option for the user to **Measure EEPROM Write Times.** Write a function that enables the user to measure with a logic analyzer or oscilloscope how long it takes to write data to the EEPROM in **byte write and page write modes**, including software overhead. One method is to use two GPIOs on the 8051 to help measure these times. Toggle a GPIO just before issuing a byte write command (including a STOP condition to force the EEPROM to commit the data). Use ACK polling to determine when the device has finished the write, and toggle the GPIO again. Use a logic analyzer or scope to measure the time the byte write took. Toggle a second GPIO just before sending a page write command (send a page of 16 bytes to the EEPROM and then commit the data), use ACK polling to determine the end of the write operation, and then toggle the second GPIO again. Compare both byte write and page write times to the data sheet write cycle timing specification, and determine how long it would take to write 1000 bytes of data to the EEPROM using the byte write and page write methods. Note: You'll want to treat this function as a critical section, and make sure to turn off interrupts while you are executing your write timing code. Be prepared to discuss how your measured times compare to calculated times, and how you might further reduce EEPROM write cycle time impact in an embedded system design.

- **[Challenge]** Implement the following **Timed Block Fill**:

  Enter a start address, end address, and a byte fill value in hex. If the entered values are valid, the EEPROM contents from the start address to end address are written with the fill value. Do not make the user type in "0x" before the address or fill hex values. To be considered valid, the end address must be greater than the start address, but no larger than then end of physical memory in the EEPROM.

  As soon as a valid set of input values have been entered, your code must call a function named eeprom_block_fill() from which all the EEPROM block fill write operations will be initiated. At the beginning of the eeprom_block_fill() function, an unused GPIO line must be toggled from low to high. At the end of the block fill operation (when ACK polling indicates that the last byte of data has been committed to non-volatile EEPROM storage), the same GPIO line must be toggled from high to low. This will provide a mechanism for measuring the time it takes for your software algorithm to perform the block fill operation. The time it takes for the algorithm to block fill with an instructor-supplied input value data set (consisting of start address, end address, and byte fill value) will be recorded. For this timing challenge, your processor must be running at an oscillator frequency of 11.0592MHz and X2 mode may be enabled.

Submission Questions

19. [**Required Element**[1]] As part of your submission, provide answers to the following:

    a) What operating system (including revision) did you use for your code development?

    b) What compiler (including revision) did you use?

    c) What exactly (include name/revision if appropriate) did you use to build your code (an IDE, make/makefile, or command line)?

    d) Did you install and use any other software tools to complete your lab assignment?

    e) Did you experience any problems with any of the software tools? If so, describe the problems.

NOTE: Make copies of your code, SPLD code, and schematic files and save them as an archive. You will need to submit the Lab #4 files electronically at the end of the semester.

Submission Instructions

Instructions: Print your name and sign the honor code pledge on the signoff sheet. Turn in a scan of the signed form, the items in the checklist below, and the answers to any applicable lab questions in order to receive credit for your work. No cover sheet please.

In addition to the items listed on the signoff checklist, be sure to review the lab for additional requirements for submission. **Submit all items electronically via Desire2Learn (D2L, https://learn.colorado.edu), to reduce paper usage.**

- ❑ Scan of signed and dated signoff sheet as the top sheet (No cover sheet please)
- ❑ Scan of timing diagrams and analyses for the LCD and EEPROM interfaces
- ❑ PDF of full copy of complete and accurate schematic of acceptable quality (all old/new components shown). Include programmable logic source code (e.g. .PLD file for the SPLD).
- ❑ Full copy of fully, neatly, clearly commented source code (including C and header files, and .RST, .MEM, and .MAP files). **Ensure your code is neat and easy to read, and that each file has header comments that identify the author and any leveraged code the file contains - there will be deductions if this information is not provided for all files.** If you submit a PDF of your code in order to improve its appearance, you must also submit the original source code files as well.
- ❑ Answers to submission questions regarding software environment and tools.

Make copies of your code, SPLD code, and schematic files and save them as an archive. You will need to submit all the lab files electronically at the end of the semester.

ECEN 5613 **Embedded System Design** **Fall 2015**
Lab #4 Signoff Sheet

You will need to obtain the signature of your TA on the following items in order to receive credit for your lab assignment. Signatures are due by **Friday, November 13, 2015 (Required Elements) and Wednesday, November 18, 2015 (Supplemental Elements)**. Labs completed late will receive grade reductions.

Print your name below, sign the honor code pledge, circle your course number, and then demonstrate your working hardware & firmware in order to obtain the necessary signatures. All items must be completed to get a signature, but partial credit **is** given for incomplete labs. Receiving a signature on this signoff sheet does not mean that your work is eligible for any particular grade; it merely indicates that you have completed the work at an acceptable level.

Student Name: _____

Honor Code Pledge: "On my honor, as a University of Colorado student, I have neither given nor received unauthorized assistance on this work. I have clearly acknowledged work that is not my own."

Student Signature: _____

Signoff Checklist

Required Elements

- ❑ Pins and signals labeled and decoupling capacitors present on board
- ❑ LCD functional, C code for basic LCD routines functional
- ❑ LCD control signal timing meets specifications (diagram)
- ❑ Serial EEPROM functional, contents present after power cycle
- ❑ C code for EEPROM functional, I^2C timing correct
- ❑ LCD Display, Clear, and Hex/DDRAM/CGRAM dumps _____

TA signature and date

Supplemental Elements (Qualifies student for higher grade.)

- ❑ Elapsed time display (accurate 1 second resolution)
- ❑ Elapsed time stop, restart, reset to "00:00.0":
- ❑ Support for custom LCD characters, fun logo
- ❑ Good integration with previous code, all functions work with no irregularities _____

Supplemental Elements (Qualifies student for higher grade.)

- ❑ PCF8574 I^2C I/O Expander
- ❑ EEPROM `eereset()` and WDT functional and correct _____

FOR TA/INSTRUCTOR USE ONLY Required Elements	Not Applicable	Poor/Not Complete	Meets Requirements	Exceeds Requirements	Outstanding
Schematics, SPLD code	☐	☐	☐	☐	☐
Hardware physical implementation	☐	☐	☐	☐	☐
Required Elements functionality	☐	☐	☐	☐	☐
Sign-off done without excessive retries		☐	☐		
Student understanding and skills	☐	☐	☐	☐	☐
Overall Demo Quality	☐	☐	☐	☐	☐

FOR TA/INSTRUCTOR USE ONLY Supplemental Elements	Not Applicable	Below Expectation	Meets Requirements	Exceeds Requirements	Outstanding
Supplemental Elements functionality	☐	☐	☐	☐	☐
Sign-off done without excessive retries		☐	☐		
Student understanding and skills	☐	☐	☐	☐	☐
Overall Demo Quality	☐	☐	☐	☐	☐

TA/Instructor Comments □ □ □ NOTE: This signoff sheet should be the top sheet of your submission.

- ❑ Optional Challenge: Measure LCD DDRAM search performance
- ❑ Optional Challenge: Measure EEPROM byte/page write times
- ❑ Optional Challenge: Measure EEPROM Block Fill performance