CEG 3136 – Computer Architecture II
Lab3 – Hardware Interfacing – 7-Segment Displays/LCD

Objectives:

To introduce the interfacing of the Motorola 9S12DG256 through an implementation of multiplexed 7-segment display and interfacing to a Liquid Crystal Display (LCD).

PreLab:

- Marks are assigned for the pre-lab work.
    - Prepare a draft of the lab report according to instructions at the end of this document. The draft should be a Word document complete with cover page and sections of the report.
    - Design your system (hardware and software) according to the guidelines provided in this report – be sure to reflect your design in the draft lab report.
    - Code and correct your modules and integrate them into the CodeWarrior Alarm System project. Bring the updated project to the lab in a memory stick or on your laptop.
- Please show your prelab work (draft lab report and compiled CodeWarrior project) at the start of the lab session.

Equipment Used:

- Windows PC (with CodeWarrior installed)
- Dragon-12 Trainer

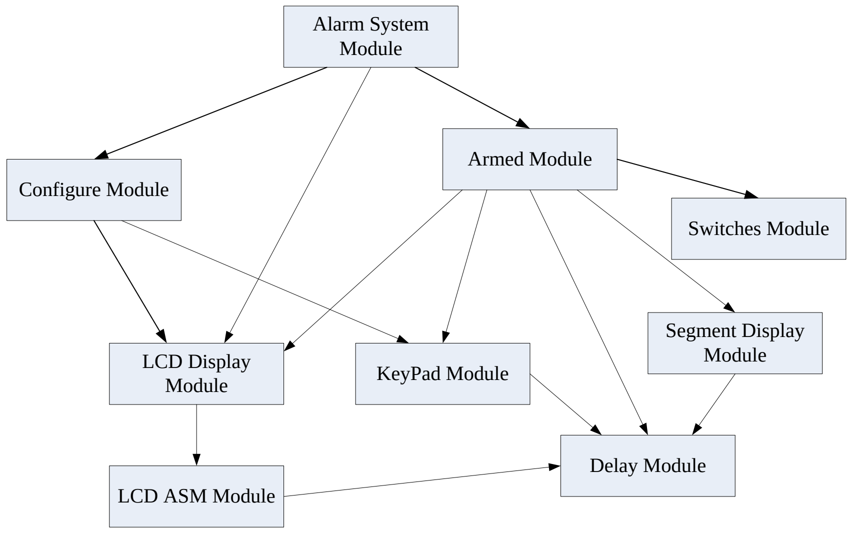
Description:

In this lab you will be integrating the 7-segment displays and the LCD into the alarm system project (a CodeWarrior C project). As well you will integrate the Keypad and Delay assembler modules from Lab 2 into the CodeWarrior C project. The 7-segment displays shall be used to show countdowns during arming and disarming of the alarm system; it is also used to show when the alarm is triggered. The LCD is used to display character strings such as the menu that appeared in the MiniIDE terminal. The keypad is used for input from the user. For details on how the 7-segment displays and the LCD are organized on Dragon-12, consult the Dragon-12 Manual, Section 4-3 and Section 4-5 of the Cady textbook. A provided PDF document contains an overview on interfacing to the LCD. Material from Tutorial 4 and Tutorial 5 can also be of help. In this lab, all interaction with the user is accomplished with the hardware on the Dragon-12 board.

Lab Software

Introduction

The CodeWarrior project is provided for the alarm system. It integrates the 7-segment displays and the LCD. The software modules defined for this project are shown below.



Alarm System Software Modules

You should already be familiar with the following modules that have been used in previous labs:

- Alarm System Module (files: alarm.c/alarm.h/alarmextern.h): This C module provides the same functions as the equivalent assembler module provided in the previous labs. The main difference is that all calls to Debug-12 routines to display strings on the Mini-IDE terminal have been changed to calls to the LCD Display module to display strings on the LCD Display.
- Config Module (files: config.c/config.h): This C module provides the same functions as the equivalent assembler module provided in previous labs but with calls to the LCD Display module to display strings on the LCD.
- Armed Module (files: armed.c/armed.h): Most of the logic in this C module is the same as the assembler module from the previous labs. Strings are printed to the LCD display. The module also uses to 7-segment display to print a down count (10, 9, 8, …) when arming the system and when disarming the system (when the front door is opened). The 7-segment display is also used to display the triggered alarm (flashes an A on the leftmost display). Note that these features are achieved by calling functions in the Segment Display module.
- Switches Module (files: switches.c/switches.h): This module is the C version of the assembler Switches module provided in Lab 2.
- LCD ASM Module (files: lcd.asm/lcd_asm.h): This is an assembler module that provides low-level functions for manipulating the LCD Display. This module is being provided to you. See the Design Section for details on the functions provided by the module.

You will be providing the following modules. The first two are assembler modules ported from Lab 2, while the other 2 are C modules that you must develop. The last page in this document provides guidelines for integrating assembler modules into the C project.

- Keypad Module (files: keypad.asm/keypad_asm.h): This module is the Keypad module from Lab 2. The header file *keypad_asm.h* defines function prototypes for the module (this file is provided) and the *keypad.asm* from Lab 2 must be added to the project.

- Delay Module (files: delay.asm, delay_asm.h): This module is the delay module from lab 2. The header file *delay_asm.h* defines function prototypes for the module (this file is provided) and the *delay.asm* file from lab 2 must be added to the project.
- LCD Display Module (files: lcdDisp.c/lcdDisp.h): This C module provides functions to display strings onto the LCD display. It makes use of the LCD ASM Module low level functions. The header file is provided and the template *lcdDisp.c* file needs to be completed.
- Segment Display Module (files: segDisp.c/segDisp.h): This C module provides functions to display characters on the 7-segment display. The header file is provided and the template *segDisp.c* file needs to be completed.

Software Design

1) Assembler Modules

The assembler modules from lab 2 are integrated into the C project by developing a header file that provides the function prototypes to the modules (these are provided) and updating the assembler files as described in the last page of this document.

KeyPad Module: The assembler subroutines in this module called as C functions. Recall that the module from Lab 2 contains one subroutine to initialize the hardware (i.e. Port A) connected to the keypad and a subroutine to detect a key pressed from the keypad (returns the ASCII code of the key pressed) and a subroutine that polls the keypad. See the file *keypad_asm.h*.

Delay Module: Contains the assembler subroutines for creating delays (the subroutines are called as C functions). The subroutines in this module will be called by other modules. You may need to update this module to meet the needs of the Segment Display Module. See the file *delay_asm.h*.

2) C Modules

The following modules are new and will be developped using the C programming language. The header files for the modules are provided. As well template C files are given.

Segment Display Module: The file *SegDisp.c* contains the code for the Segment Display Module. This module shall provide the following functions:
`void initDisp(void)`*:* Initialize the hardware (port B and port P) that is connected to the 7-segment displays. It should also initialize the displays to a blank (no segments are lit).
`void setCharDisplay(char, byte )`: A function that adds a character (identified by its ASCII code) for display. Reserve four bytes in memory to contain either the characters or codes for displaying the characters on corresponding displays. When the function is called, two arguments are provided, the character to display (first argument) and a display number (starting at 0) to indicate on which of the 4 7-segment display the character should appear.
`void segDisp(void)`: A function that updates the displays for a period of time (use 100 ms). This is to allow the calling routine to regain control periodically to perform other tasks, such as checking the keypad.
`void clearDisp(void)`: This function should clear the display.

LCD Display Module: This module makes use of the functions provided by the LCD ASM Module (see below). It provides the following functions to other modules for printing strings on the LCD Display.
`void initLCD(void)`: Initialises the LCD display (this is function that simply calls the `lcd_init()` function provided by the LCD ASM Module).

`void printLCDStr(char *, byte)`: This function prints a string on one of the two lines of the LCD display. The address of the string to be printed is passed in the first parameter while the second parameter is set to 0 or 1 to identify the first or second line respectively.

In the modules, define other functions as required. Be modular in your software design. In the subsequent lab, the Segment Display module will be redesigned so that polling is not required. If you are modular, you will be able to conserve more of your functions.

LCD ASM Module

The LCD ASM Module provides low level functions to manipulate the Liquid Crystal Display (LCD). See the document "Interfacing to a Liquid Crystal Display (LCD)" for an overview on how to interfacing to an LCD. A controller/driver simplifies the manipulation of the LCD. On the Dragon-12 card, the LCD is a two-line display with 16 characters on each line. Each character position is addressable according to the following table (hex addresses); each row represents a line on the LCD.

00	01	02	03	04	05	06	07	08	09	10	0A	0B	0C	0E	0F
40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F

Updating the display consists of first setting the "cursor" to a character position and then writing either a character at the position or a string starting at that position. The LCD ASM Module contains a number of low level functions (translated to assembler):

`void lcd_init(void)`: Initilalises the LCD display (and clears the display).
`void data8(char)`: Write an ASCII character at the current cursor position.
`void clear_lcd(void)`: Clears the display.
`void type_lcd(char*)`: Display the ASCII string at the current cursor position.
`void set_lcd_addr(char)`: Sets the address of the cursor on the display (see above table).
`void instr8(char)`: Write an instruction to the control register of the contrôler.

Here are a few hints to help you in your development:

1. To test the operation of the hardware, you can modify the parallel port registers. When in running the CodeWarrior debugger, you can modify and examine the contents of memory using the Debugger memory window; initialize the appropriate data registers of the ports of interest. You can use the same command to modify the data registers to check the control of the parallel ports affects the appearance of the 7-segment displays.
2. Share the preparation by sharing the modules between the two partners in the team. For example, one member can be responsible for the design and coding of the Segment Display module and the other member can be responsible for the "LCD Display" module and integrating the Delay and Keypad Modules into the CodeWarrior Project.
3. An additional Code Warrior project has been supplied to test the LCD Display module and Segment Display module (also uses the KeyPad and Delay Modules). It simply displays the characters on the 7-segment display as typed on the keypad starting from the leftmost display to the rightmost, rotating back to the leftmost display. Use it to debug the Segment display module (and interfacing of the KeyPad and Delay assembler modules). When done, simply copy and add the module files to the Alarm System project.

What-to-do LIST

PreLab related part:

- Draft the lab report; this should be a Word document that contains:
    - The Cover Page
    - Lab Objectives
    - Equipment and components used.
    - Software/Hardware Design
        1. Present a circuit that shows how the hardware used in the lab is connected to the various micro-controller ports. Show the keypad and displays.
        2. The software design (according to guidelines provided) of the LCD Display and Segment Display modules. Please be as complete as possible. It is not necessary to include pseudo-code, but describe all functions you plan to develop for your modules. Divide the set of functions into "Entry Points", functions called by external modules, and "Local Functions", functions used only withing the module. It is NOT necessary to include the designs for the KeyPad and Delay modules.
- Prepare the code for your system:
    - Create the C and header files for the LCD Display and Segment Display modules. They should be integrated in the CodeWarrior Alarm System project.
    - Create the header file and update the assembler files from lab 2 for the KeyPad and Delay modules, Integrate the files into the CodeWarrior Alarm System project.
    - When you arrive at the lab session, the CodeWarrior project should compile without any errors.
- You must show draft lab report and CodeWarrior project that compiles with no errors upon arrival at the lab to your TA (marks are assigned for this work). It is important to have this part of the lab ready since testing and getting the software to run will take time. It will not be possible to design and create the software at the lab session.

In the lab:

a. Load your software into the Dragon-12 card using Codewarrior. Run the alarm system and test all functions of the alarm system to ensure that both displays are operating properly.
b. Show to the TA your up-and-running system. He or she will record your success.

In your report:

a. Complete the design of the software modules you developed. Provide your complete CodeWarrior project files in a separate zipped file.

Adding ASM modules to a CodeWarrior C project

1) Create header file with C prototypes and other definitions. Example for keypad_asm.h:

```
/*----------------------
File: KeyPad_asm.h
Description:  Header file to use the KeyPad Module
-----------------------*/

#ifndef _KEYPAD_ASM_H
#define _KEYPAD_ASM_H

//C Prototypes to assembler subroutines
void initKeyPad(void);
Byte pollReadKey(void);
Byte readKey(void);

// Some Definitions
#define NOKEY 0  // See KeyPad.asm

#endif /* _KEYPAD_ASM_H */
```

2) Change header file for HCS12 register definitions (be sure to remove the inclusion of the files `sections.inc` and `reg9s12.inc`).

```
; Include header files
 NOLIST ; a space exists at the start of this line and the next 2 lines
 include "mc9s12dg256.inc"  ; Defines EQU's for Peripheral Ports
 LIST
```

3) Change section definitions (remove inclusion of "`sections.inc`" file from lab2). These sections are already defined in the CodeWarrior project. This means that you replace all SWITCH statements in the assembler file with a SECTION statement. For example "`    SWITCH global`" becomes "`.rodata SECTION`". Do note that there is not space at the beginning of these lines, for example, `.rodata` is place in the label field.
   a. Constant global data:
      ```
      .rodata SECTION ; place in constant data section (EEPROM)
      ```
   b. Code:
      ```
      .text SECTION   ; place in code section
      ```
   c. Variables:
      ```
      .data SECTION    ; place in data section (RAM)
      ```

   Notice that the `SECTION` assembler directive in CodeWarrior is different than the directive used in MiniIDE. It actually replaces the MiniIDE `SWITCH` directive (it is also used to define sections) and can be used to place subsequent code/data in the various sections defined by the CodeWarrior project. See the CodeWarrior documentation for more details on the `SECTION` assembler directive.

4) With the `XDEF` directive, define names of module subroutines as external symbols to be referenced by other modules:

```
; Define External Symbols
 XDEF initKeyPad, pollReadKey, readKey
```

5) With the `XREF` directive, define names of subroutines of other modules to be referenced by the current module

```
; External Symbols Referenced
 XREF delayms
```

6) The CodeWarrior Assembler is case sensitive. When you compile your project, any errors in labels (i.e. labels that have same characters but with different cases) will generate errors.
7) Some HCS12 instructions recognized by MiniIDE are not recognized by the CodeWarrior Assembler. For example, CodeWarrior does not recognize `ldb`, it must be `ldab`.