ESE 381 Embedded Microprocessor Systems Design II

Spring 2016, K. Short
February 1, 2016 8:12 am

MODULE # 1 Keypad and LCD Display

To be performed the weeks starting Feb. 7th and Feb. 14th.

Prerequisite Reading

1. Atmel AVR240 *4 x 4 Keypad - Wake-up on Keypress* application note.
2. Interrupts, pages 165 - 172 of IAR Embedded Workbench C-SPY Debugging Guide.
3. Power Management and Sleep Modes, pages 63 through 68 ATmega128A data sheet (9/15).
4. EA DOG- M Alphanumeric LCD Module Data Sheet.

References

1. Sitronix ST7036 Dot Matrix Controller Driver Data Sheet.

Overview

You will be exposed to many hardware and software concepts in this module. You are not expected to have a complete understanding of them all at this time. The concepts that are new for this course will be discussed in detail as the course progresses. It will be made clear to you in lecture at what time during the course you are expected to have a thorough understanding of which specific concepts. The software for this module's projects is provided in this module's folder on Blackboard.

This module includes the interfacing of a 4 x 4 keypad and a 3-line, 16 character per line, alphanumeric LCD display to an ATmega128 (on an ET-AVR board). These two kinds of I/O devices are commonly used in many embedded system designs.

You have two weeks to complete this module. All signoffs must be completed by the second of the two weeks. However, as a matter of pacing, you are expected to complete the keypad portion of the module the first week and the LCD display portion the second week.

Keypad

The connection of a 4 x 4 keypad to an Atmel AT90S1200 AVR microcontroller and the assembly language software to scan the keypad is described in Atmel application note AVR240. Understanding this application note will make you familiar with the hardware and the scan algorithm. However, instead of scanning the keypad using an assembly language program, you will use a C program.

LCD Module

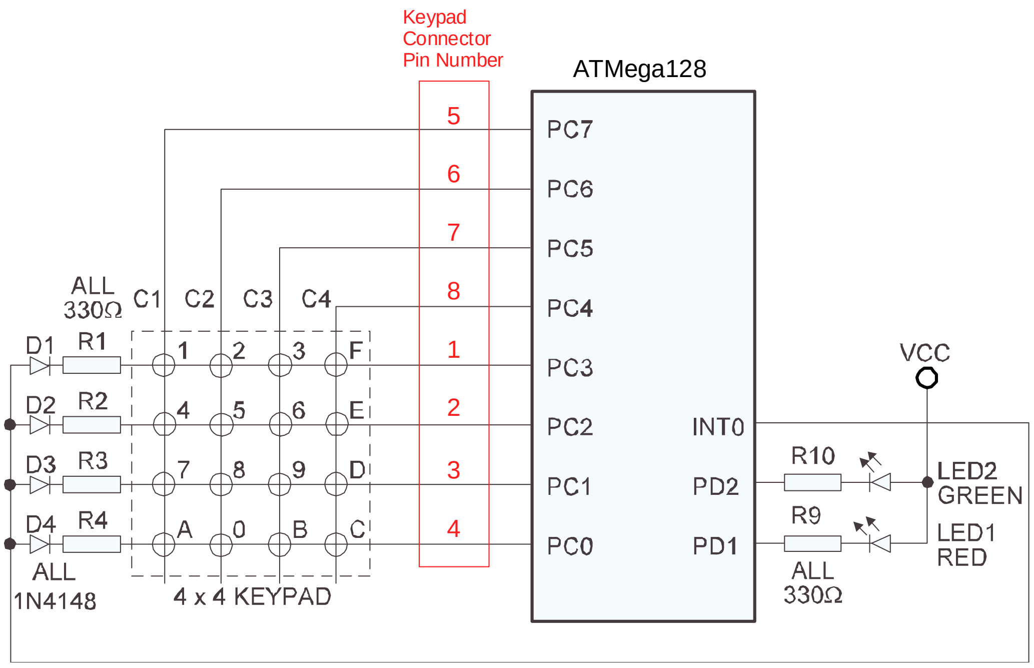
Liquid Crystal Displays (LCDs) provide a low cost alphanumeric display capability. LCDs are available as complete modules that accept ASCII representation of the data to be displayed. Many of these modules use similar display controller ICs that support a 4/8-bit parallel interface (e.g. HD44780). More recent modules use controllers (e.g. ST7036) that support either a 4/8-bit parallel interface or a SPI serial interface. The result is that the interface and operation of many different LCD modules are similar because they use the same or similar controllers.

The display module we will use is an Electronic Assembly (EA) DOGM163W-A. This is a 3 line, 16 characters per line display.

Design Task
Project Keypad
The keypad's columns are connected to the high nibble of PortC and its rows are connected to the low nibble, as shown in the figure that follows. A keypress generates an interrupt at INT0. Pins PD1 and PD2 drive LEDs used by the test program.

Create an IAR Embedded Workbench project named `keypad` in a workspace named `mod1`. Add the file `keyscan.c` to the project. This C program scans the keypad and indicates the key code received by flashing one of two LEDs. The basic scan algorithm is similar to that in application note AVR240.

There are three things you must modify in file `keyscan.c`. First, this code is written to use PortB instead of PortC. You must modify the code to use PortC. Second the values for the constant array `tbl` used in the table lookup may need to be changed to return key values corresponding to those of the figure that follows. Third, this code assumes that the ATmega128 is run at 1 MHz. The ATmega128 on the ET-AVR board runs at 16 MHz. You must modify the code so that the delays are the same at the faster clock rate.

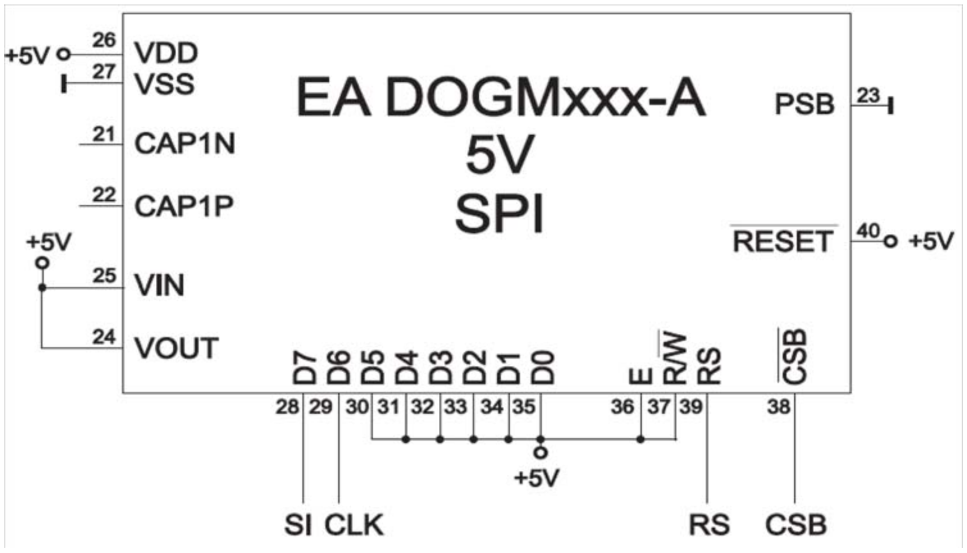


Try to simulate this program prior to your laboratory session. There are a number of issues that must be resolved for a successful simulation, including the program's delays and its interrupt. Reference 2 of the prerequisite reading discusses simulating interrupts in C-SPY.

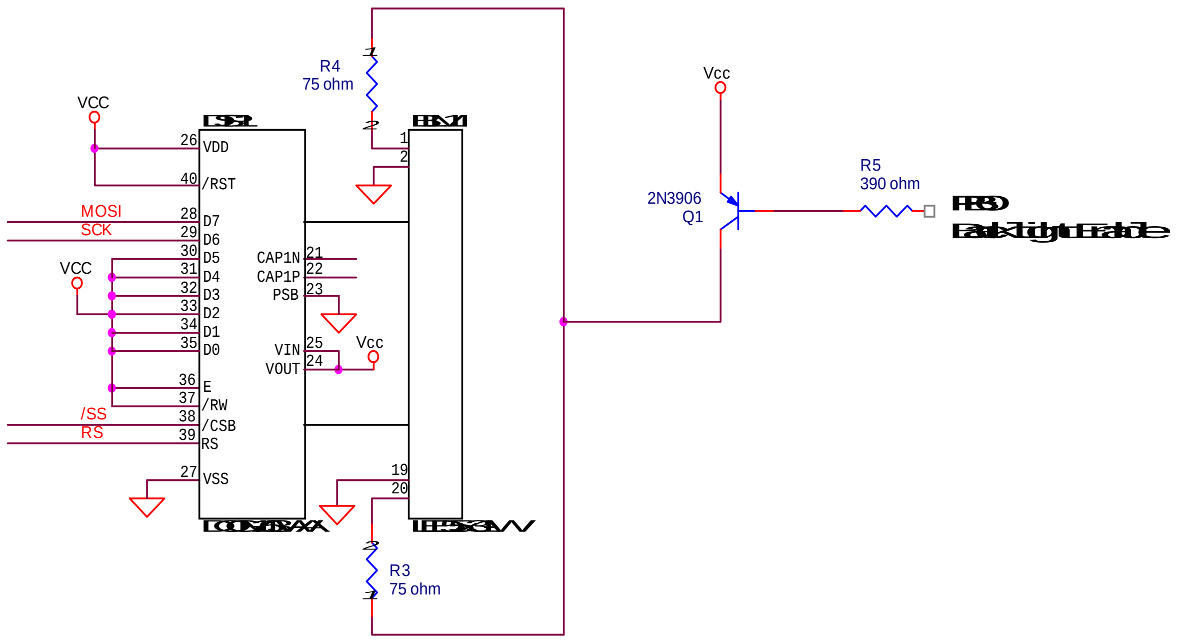
Note:
When simulating interrupts, it appears that you have to exit the ISR to remove the "condition" that caused the interrupt. For keyscan.c, the interrupt is configured for low level. If the while statements, at the end of the ISR, that check for a key being released are compiled in the simulated code, the simulation hangs waiting for the key to be released. Since the pin that is being tested is INT0 it appears that the simulator does not "return this pin to 1" until the ISR returns. The #ifndef debug preprocessor directives eliminate this code (as well as the __delay_cycles intrinsic function calls) from a compilation if debug is defined.

Your prelab must include a schematic for your system. One page of the schematic must show the ATmega128's wiring. Use the ET-AVR Stamp ATMega128 schematic to obtain the basic information. This page must also include the JTAG interface connector. A second page of your schematic must show the keypad interface. Your prelab must also contain a listing of your modified code for `keyscan.c`.

Project Test Display
The display module uses the Sitronix ST7036 Dot Matrix Controller Driver. The DOGM163W-A module can be operated at a supply voltage of either 3.3 V or 5.0 V. Connections to the module, when operated at 5.0 V and using its SPI, are shown in the following block diagram from the DOG Series data sheet.

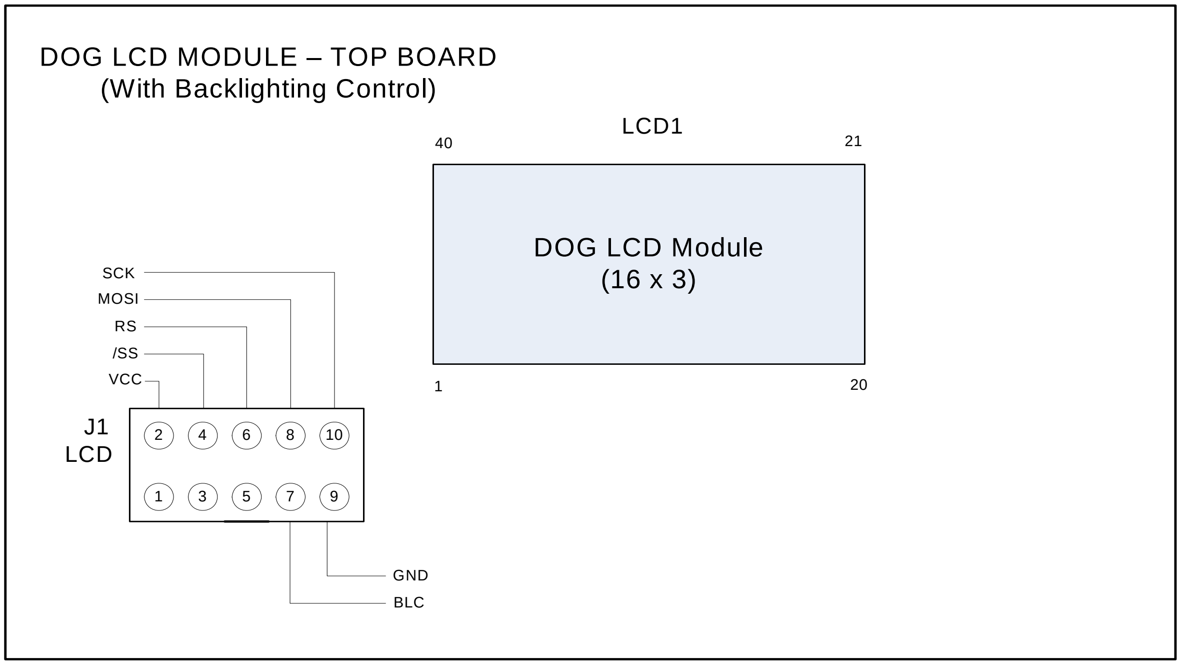


For higher contrast, the DOGM163W-A can be used with a backlight. LED backlights of various colors are available. We will use a white backlight (EA LED55X31-W). You will be provided a prewired display that uses a DOGM163W-A LCD Module and an EA LED55X31-W LED backlight. A schematic of the circuit on the prewired display board is shown below.



The connections to the display board are brought out to a 10 pin header, J1 on the figure that follows. A flat cable is used to connect the display board to a 10-pin header that you must put on your breadboard and wire to the ATmega128's SPI.

The layout of the display board is shown below.



The ST7036 Dot Matrix Controller Driver on the DOG LCD Module contains a microcontroller that controls what is displayed on the LCD display. Commands and data are sent over the SPI interface from the ATmega128 to the DOG LCD Module. The microcontroller on the ST7036 executes these commands as its instructions.

The initialization and display update subroutines provided use PB0 (/SS) to select the DOG module for operation. PB2 (MOSI) is used to provide the serial data to the display, which is clocked in by PB1 (SCK). The other signals to the LCD module are PB4 (RS) for the register select signal. The BLC input to the LCD Module controls blanking (intensity). For now, you must hardwire BLC to ground to leave the display fully ON.

Operation of the DOG LCD Module requires a power on initialization sequence. This can be done by a software subroutine. Subroutine `init_lcd_dog` is provided for this purpose.

After initialization, you could directly send commands to the ST7036 to control the display of data along with the data to be displayed. Using this approach, your program would deal directly with the ST7036 on a character by character basis.

An alternate approach is to write a driver subroutine for the ST7036. An example of such a subroutine uses three 16-byte memory buffers (arrays) in the ATmega16’s SRAM. One buffer is provided for each line of the LCD display. Each location in a buffer corresponds to a character position on the corresponding line of the LCD display. The first location in a buffer corresponds to the leftmost character position in the corresponding line. Characters are represented in a buffer by their corresponding ASCII codes.

The subroutine `update_lcd_dog` handles all of the details required to display data in the SRAM buffers on the lines of the DOG LCD Module. Your application program has only to write the ASCII code for each character you want displayed into the corresponding location in the SRAM buffers and then call the update display subroutine.

Subroutine `update_lcd_dog` displays the contents of three memory arrays, `dsp_buff_1`, `dsp_buff_2`, `dsp_buff_3` on the LCD display. The first memory array's character codes define the characters to be displayed on the top row of the LCD display, the second memory array defines the characters to be displayed on the second row, and the third memory array defines the characters to be displayed on the bottom row. Code in this file also allocates the memory for the three arrays.

Subroutines `init_lcd_dog` and `update_lcd_dog` are written in AVR assembler and provided in the file `lcd_dog_asm_driver_inc`. A test program `lcd_dog_test_128.asm` is also provided. After you have wired in the DOG LCD Module you can test it using the test program provided. However, since this assembly language code uses AVR's directives, you must create an AVR Studio project name `lcd_dog_test_128` to run this test program.

Project Kpd2dsp
Create an IAR Embedded Workbench project named `kpd2dsp` and add it to the existing workspace `mod1`. Add the files `keyscan_isr.c`, `kpd2dsp.c`, `lcd.h`, `lcd_ext.c`, `and` `lcd_dog_iar_driver.asm` to the project. This mixed language project simply displays, on the LCD, the key code of a key when it is pressed.

The file `keyscan_isr.c` is a modification of the ISR from `keyscan.c`. It removes the code to flash the LEDs and simply returns the key code in a variable. The file `kpd2dsp.c` contains the main function and displays on the LCD the value of key pressed.

The header file `lcd.h` declares the functions and arrays in `lcd_dog_iar_driver.asm` so that they can be referenced by a C program. These two files together allow a C program to use the display. The code in `lcd_dog_iar_driver.asm` is equivalent to that in `lcd_dog_asm_driver.inc`, except it is written in IAR's AVR assembly language. Both assembly languages have the same instruction mnemonics, but different directives.

The file `lcd_ext.c` contains two functions that make it easier for a C program to use the LCD display. The function `clear_dsp()` clears the display buffer arrays. When followed by calling the function `update_dsp()`, the display is blanked.

The function `putchar()` puts a single character, passed to it as an argument, into the display buffer at the position corresponding to the value of variable `index`. Variable index is declared in the same file. This `putchar` function replaces the standard `putchar` function, so a `printf` statement will print to the LCD. Using `printf` makes it much easier to use the LCD display in a C program. However, use of the `printf` function causes the compiler to link in the `printf` function code from the standard library and this significantly increases the program's size.

The program in the file `kpd2dsp.c` prints the characters "Key pressed = *n*", on the LCD. The value *n* is the key value expressed as decimal integer. If you press a new key, the value of *n* changes.

In the laboratory, after you have demonstrated that this program works, you must demonstrate a modified version of this program. The modified version starts by blanking the display. When the first key is pressed, its value appears is the leftmost position of line 1 of the LCD. However, now the key's value must be displayed as a single character. So, when a key with the value 15 is pressed it will appear on the display as F.

When a key is pressed subsequently, its value is displayed on the LCD in the next character position to the right. When the first row is filled, the next key value appears in the leftmost position of the second row. When the second row is filled, the subsequent characters appear on the third row. When the third row is filled, the next key value appears in the leftmost position of the first row. You must determine how to modify this program to accomplish this result. **The modified program is not required to be submitted as part of your prelab. But, you must submit its source file at the end of the second week of this module and demonstrate its operation.**

Laboratory Activity

Remove the switches and bargraph LED used in Module 0 from you breadboard.

Project Keypad

Wire the keypad to the ATmega128. Have a TA check your wiring before applying power. Change the ATmega128's configuration to run at 16MHz using its external crystal. You will be shown how to do this in the laboratory. Measure the voltage at the INT0 pin of the ATmega128 while pressing any key on the keypad. Next measure the voltage at INT0 with no keys pressed. Record the measured values. Are these values compatible with the ATmega128's logic level requirements?

Be sure to comment out the `#define debug` statement before compiling this program for emulation. Emulate project `keypad` using JTAGICE3. Create a table listing each key's scan code and its returned key value. After verifying the operation of your keypad and its interface, have a TA confirm that the keypad operates correctly. Demonstrate to the TA that you know how to place two breakpoints in the program. One to display the scan code (called `keycode` in the program) and one to display the keyvalue returned (called `num` in the program). Obtain the TA's signature.

Project Display Test

Add a 10-pin header to connect to the SPI pins of the ATmega128. A flat cable will be used to connect from this header to the DOG LCD Module. Create the project `lcd_dog_test_128` in

AVR's Studio. Using Studio, configure the ATmega128 to run at 8MHz using its internal RC oscillator. You will be shown how to do this in the laboratory. Load and run the program. Demonstrate to the TA that your display program and hardware operate properly. Obtain the TA's signature. Using Studio, change ATmega128's configuration to run at 16MHz using its external crystal. Determine whether the display operates properly at this frequency. Record the results in your laboratory report.

Project Kpd2dsp
In IAR Embedded Workbench build the project `kpd2dsp`. Emulate `kpd2dsp` using JTAGICE3. Have a TA verify that your keypad and LCD display operate properly together and obtain the TA's signature.

Load your modified version of `kpd2dsp` and debug it. When it is working properly, have a TA verify its operation and obtain the TA's signature.

Leave the hardware that you have wired on the breadboard intact. This hardware will be used again in later laboratories.