Interfacing a Hitachi HD44780 to a Silicon Laboratories C8051F120

Table of Contents

Introduction:

This document is intended to explain the basics of interfacing a Hitachi HD44780 LCD Controller with the Silicon Laboratories C8051F120 microcontrollers and to provide sample code in the form of a C header file for the 8051. All code for this document was developed and tested with SDCC 2.7.0. All tables, diagrams, and charts from the Hitachi data sheets unless credited otherwise.

Notes on the LCD panel document:

These are some notes to clarify the LCD screen manual.

This was developed for use in Microprocessor Systems, which is taught at Rensselaer Polytechnic Institute. The LCD display is used in one of the lab experiments conducted by the students.

This document covers the basic operation of the screen with one exception. There is no coverage of how to use the character ram. This is a memory block into which custom characters may be stored and then displayed on the screen. This is beyond what is used in class and so this material was omitted from the manual. People interested in this should refer to the Hitachi data sheets on the HD44780.

Email: steved@ecse.rpi.edu
Snail mail: Steve Dombrowski, JEC 6th floor, RPI, Troy, NY 12180.

Hardware Operation:

The hardware in the HD44780 is mostly transparent to the programmer. As a result many of the features do not need an in depth explanation. Those readers interested in more detailed information should refer to the Hitachi Data Sheet for the HD44780. For this project the Optrex DMC-16204 Display Module (DigiKey part number 73-1033ND) was used. This incorporates the HD44780 as the on board LCD Screen controller.

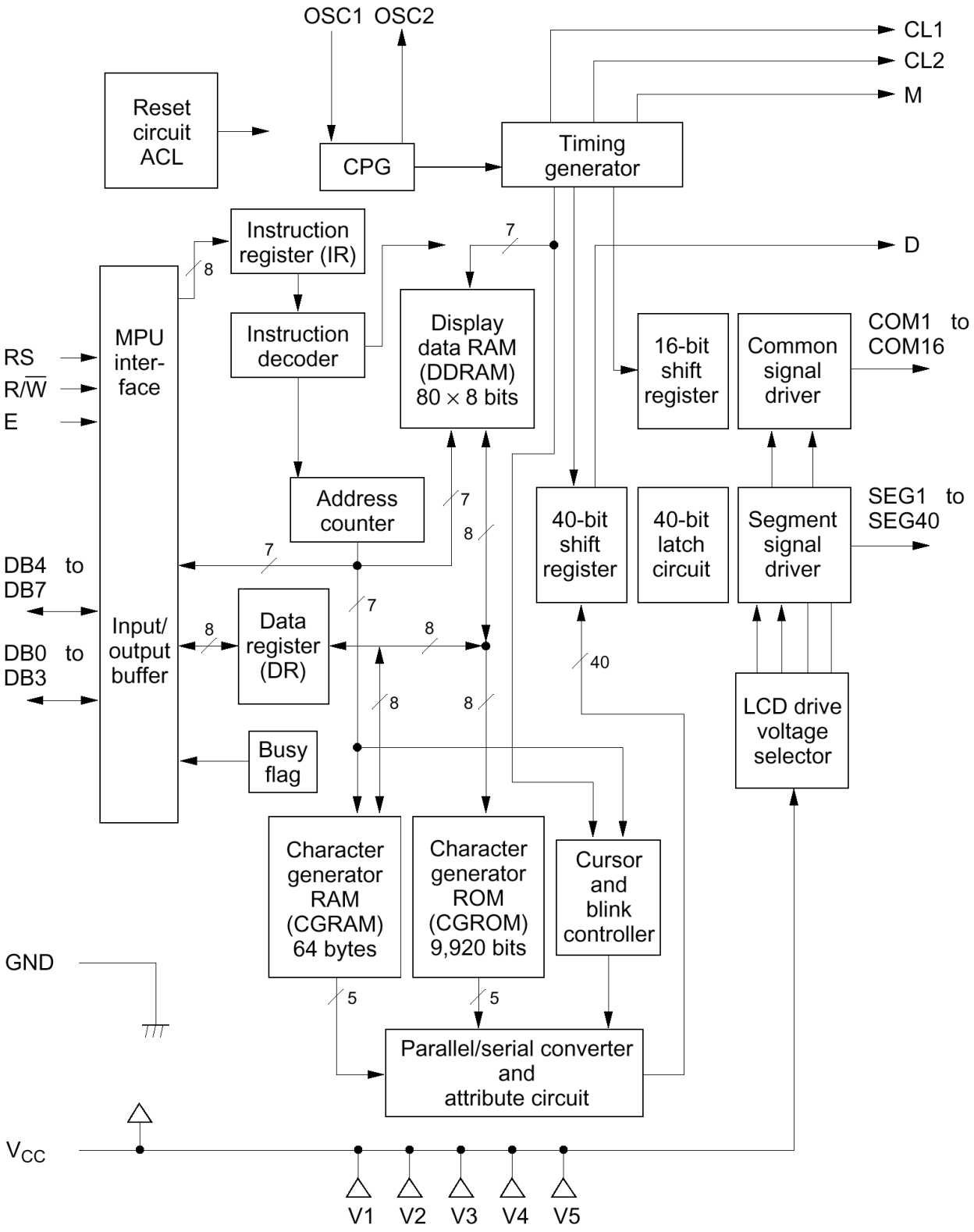
The Optrex DMC-16204 Display Module has 14 connections between itself and the microprocessor. On the Optrex DNC-16204, pin 2 provides power and pin 1 is connected to ground. Pin 3 controls the brightness level of the screen and is connected to the wiper of a 10 kOhm potentiometer. Pin 4 is the register select of the LCD screen. This is used to select between the instruction register or address counter of the HD44780. When the input to the pin is low the instruction register is active and the data register is active when the input is high. Pin 5 is the Read/Write select. When the input to the pin is high, the HD44780 is in read mode, when the input is low it is set up for a write. Pin 6 is the LCD enable. This is used to clock data and instructions into the HD44780. Pins 7 to 14 are the data pins. Pin 14 also doubles as the Busy Flag for the LCD screen. While many LCD screens use this order for the pins, the exact pin configuration may vary by part type and manufacturer. Be sure to refer to the LCD documentation before using this code to ensure that they are compatible. Schematics for wiring the LCD screen to the C8051 are included in appendix A.

The basic operation of the screen is controlled by the state of the Register Select (RS) and the Read/Write (R/W) pins. These operations are summarized in Table 1.

Table 1: Register Selection

RS	R/$\overline{W}$	Operation
0	0	IR write as an internal operation (display clear, etc.)
0	1	Read busy flag (DB7) and address counter (DB0 to DB6)
1	0	DR write as an internal operation (DR to DDRAM or CGRAM)
1	1	DR read as an internal operation (DDRAM or CGRAM to DR)

Figure 1: HD44780U Block Diagram



Memory:
The HD44780 provides an 80x8 bit Display Data RAM (DDRAM). This is used to store the data that is being displayed on the screen. This allows the HD44780 to store up to 40 characters per line. It is important to note that the DMC-16204 will display only 16 characters per line. The extra memory here can be used to store characters that may then be shifted onto the screen. All data to be displayed must be stored in the form of an 8 bit ASCII code character.

Figure 2: 1 Line Display

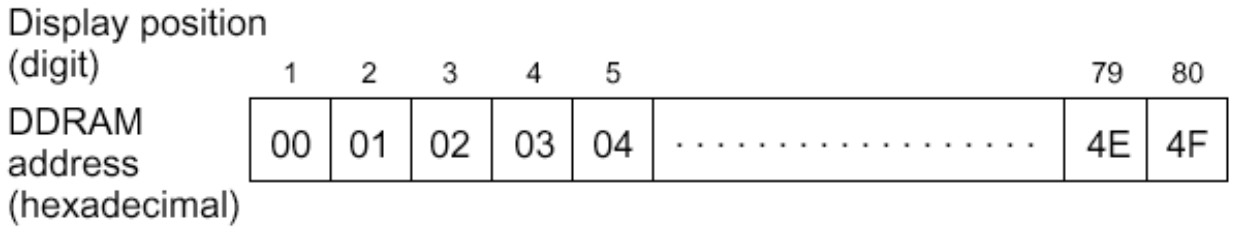
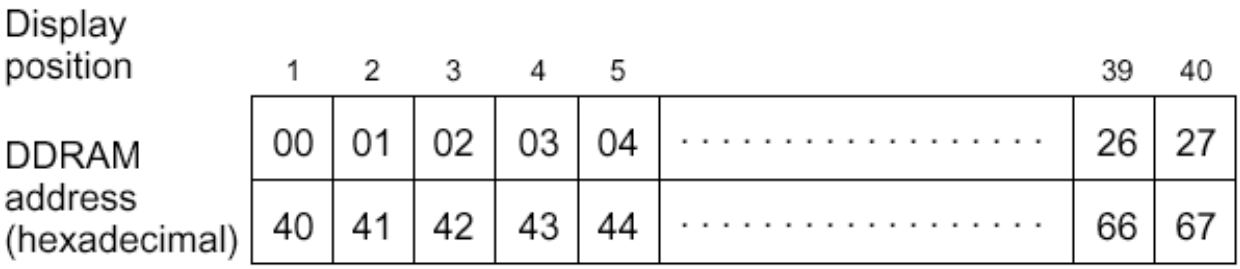


Figure 3: 2 Line Display



Instructions:
The HD44780 has a number of different instructions that it can execute. These instructions are listed in the following table:

Table 2: Instructions

Instruction	Code RS	R/$\overline{W}$	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Description	Execution Time (max) (when f_{cp} or f_{OSC} is 270 kHz)
Clear display	0	0	0	0	0	0	0	0	0	1	Clears entire display and sets DDRAM address 0 in address counter.	
Return home	0	0	0	0	0	0	0	0	1	—	Sets DDRAM address 0 in address counter. Also returns display from being shifted to original position. DDRAM contents remain unchanged.	1.52 ms
Entry mode set	0	0	0	0	0	0	0	1	I/D	S	Sets cursor move direction and specifies display shift. These operations are performed during data write and read.	37 μs
Display on/off control	0	0	0	0	0	0	1	D	C	B	Sets entire display (D) on/off, cursor on/off (C), and blinking of cursor position character (B).	37 μs
Cursor or display shift	0	0	0	0	0	1	S/C	R/L	—	—	Moves cursor and shifts display without changing DDRAM contents.	37 μs
Function set	0	0	0	0	1	DL	N	F	—	—	Sets interface data length (DL), number of display lines (N), and character font (F).	37 μs
Set CGRAM address	0	0	0	1	ACG	ACG	ACG	ACG	ACG	ACG	Sets CGRAM address. CGRAM data is sent and received after this setting.	37 μs
Set DDRAM address	0	0	1	ADD	ADD	ADD	ADD	ADD	ADD	ADD	Sets DDRAM address. DDRAM data is sent and received after this setting.	37 μs
Read busy flag & address	0	1	BF	AC	AC	AC	AC	AC	AC	AC	Reads busy flag (BF) indicating internal operation is being performed and reads address counter contents.	0 μs
Write data to CG or DDRAM	1	0	Write data								Writes data into DDRAM or CGRAM.	37 μs t_{ADD} = 4 μs*
Read data from CG or DDRAM	1	1	Read data								Reads data from DDRAM or CGRAM.	37 μs t_{ADD} = 4 μs*
	I/D = 1: Increment I/D = 0: Decrement S = 1: Accompanies display shift S/C = 1: Display shift S/C = 0: Cursor move R/L = 1: Shift to the right R/L = 0: Shift to the left DL = 1: 8 bits, DL = 0: 4 bits N = 1: 2 lines, N = 0: 1 line F = 1: 5 × 10 dots, F = 0: 5 × 8 dots BF = 1: Internally operating BF = 0: Instructions acceptable										DDRAM: Display data RAM CGRAM: Character generator RAM ACG: CGRAM address ADD: DDRAM address (corresponds to cursor address) AC: Address counter used for both DD and CGRAM addresses	Execution time changes when frequency changes Example: When f_{cp} or f_{OSC} is 250 kHz, $37\ \mu s \times \frac{270}{250} = 40\ \mu s$

Note: — indicates no effect.

* After execution of the CGRAM/DDRAM data write or read instruction, the RAM address counter is incremented or decremented by 1. The RAM address counter is updated after the busy flag turns off. In Figure 10, t_{ADD} is the time elapsed after the busy flag turns off until the address counter is updated.

It is important to note that the HD44780 can only execute one instruction at a time. Before sending an instruction to the display, the busy flag must to be read. If the busy flag is zero, then the instruction can be sent to the display, otherwise the instruction must be held by the microprocessor until the current instruction has completed execution and the busy flag is cleared.

Instruction Descriptions:

Clear Display
This instruction writes a 0x20 to all locations in the DDRAM. It also sets the DDRAM address to zero and unshifts the display, if it had been shifted. It also sets the display to increment mode.

Return Home
The DDRAM Address is set to zero and the display is unshifted, if it had been shifted.

Entry Mode Set
This instruction has two parameters, which it controls. The first is I/D. If this bit is high, the display increments the DDRAM address by one every time a character is written to the screen. If it is low, then the display address will be decremented by one every time a character is written. The second parameter is S. When S is high, the display shifts after a character is written to the screen. It will shift to the right if I/D = 0 or to the left if I/D=1. When S is low, the display does not shift when a character is written.

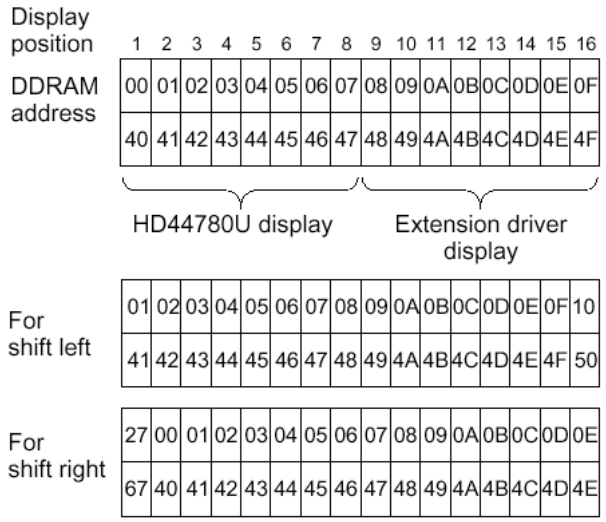
Display Control On/Off
This instruction has 3 parameters that the user can set. The first is D. This turns the display on when it is high and off when it is low. The second parameter is C. This displays the cursor when it is high and turns the cursor off when it is low. The last parameter is B. When this is high the character indicated by the cursor will blink. When it is low the display will not blink.

Cursor or Display Shift
This instruction shifts either the cursor or display by 1 character, without modifying the data stored in the DDRAM. The direction of the shift is determined by the value in the R/L bit. Both lines shift simultaneously. The shifting type and direction are summarized in the following table:

Table 3: Shift Functions

S/C	R/L	
0	0	Shifts the cursor position to the left. (AC is decremented by one.)
0	1	Shifts the cursor position to the right. (AC is incremented by one.)
1	0	Shifts the entire display to the left. The cursor follows the display shift.
1	1	Shifts the entire display to the right. The cursor follows the display shift.

Figure 4: 2 Line by 16 Character Display



Function Set

This instruction is used to initialize the display and what format the display will be using. This is done only during the initialization process and it may not be changed later in the program. DL is the data length of the interface. For this program, DL is always high, since the only the 8 bit interface is used. N is the number of display lines and F is the font size.

Table 4: Function Set

N	F	No. of Display Lines	Character Font	Duty Factor	Remarks
0	0	1	5 × 8 dots	1/8	
0	1	1	5 × 10 dots	1/11	
1	*	2	5 × 8 dots	1/16	Cannot display two lines for 5 × 10 dot character font

Note: * Indicates don't care.

Set DDRAM Address

This sets the DDRAM to the address included in the instruction. When the display is in single line mode the addresses range from 0x00 to 0x4F. In 2 line mode, the instructions range from 0x00 to 0x27 for the first line and from 0x40 to 0x67 for the second line.

Read Busy Flag

This instruction sends the state of the Busy Flag to the microcontroller. This appears on bit 7 and is used to determine if the LCD screen controller is still executing an instruction. If the bit is high, then there is an instruction executing that must be completed before another instruction can be written to the LCD screen controller

Write Data to DDRAM

This instruction writes an 8-bit pattern to the DDRAM.

Software Operation:

The code to control the LCD screen was developed as a C header file for the C8051F120. The C header files were written to provide an easy interface to the LCD screen. The two files are similar with the only differences being the ports used by the screen and the delay cycles used. The assembly code is much smaller then the C code and is also in many ways far more flexible. The code for these files is included in the Appendices.

C Code:
The C code was written as a header file that could be included in any program that interfaces with an LCD screen. For the C8051, the header file uses Port 2 for writing data and Port 1.5 to P1.7 for the control signals.

The header file contains six functions to control the LCD screen. These are OpenXLCD, SetDDRamAddr, BusyXLCD, WriteCmdXLCD, WriteDataXLCD, and WriteBuffer. These functions provide all the basic features needed to display data on the screen and to position the cursor.

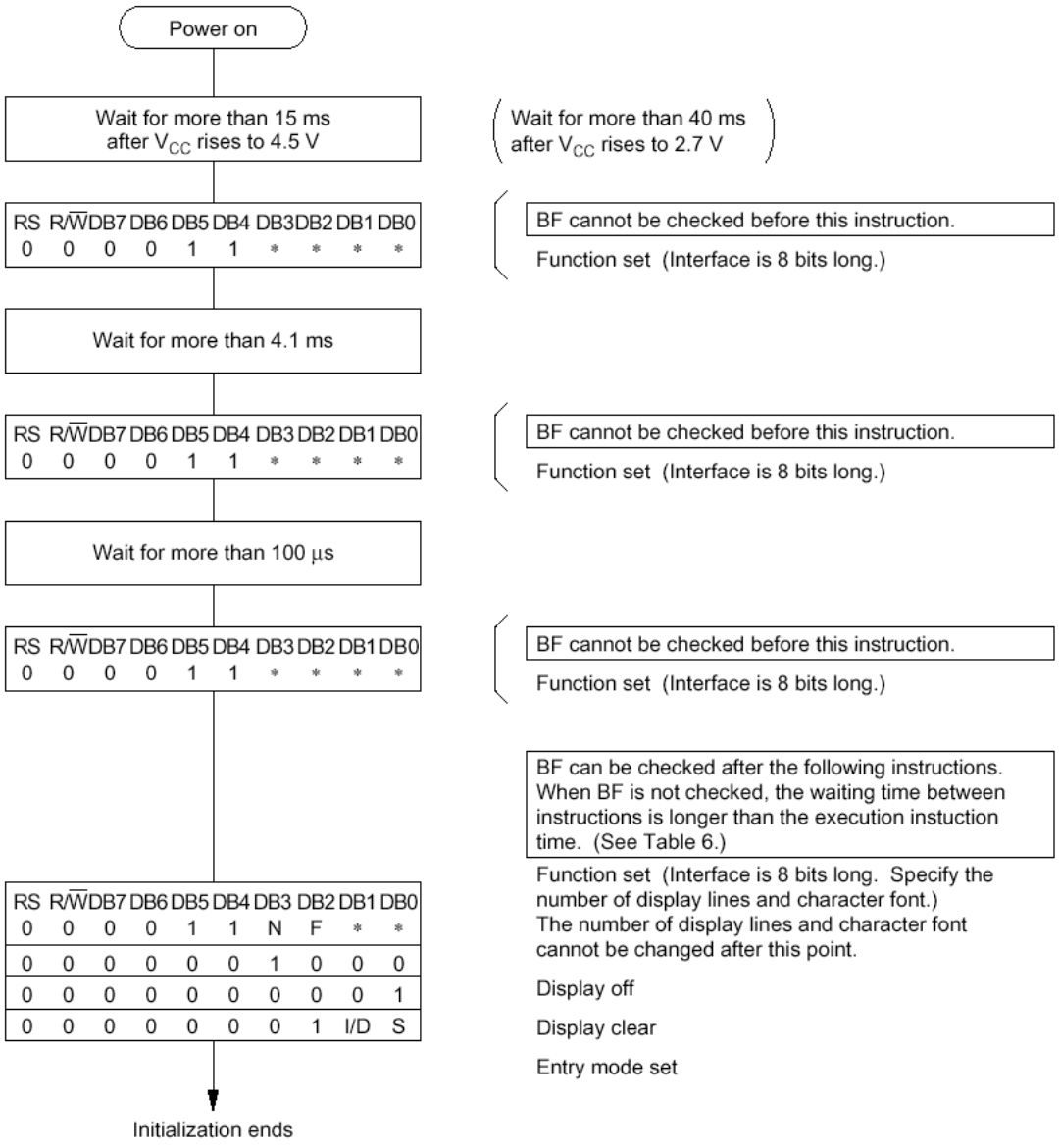
lcd_init:
This function executes all the initialization routines required by the HD44780 before it can be used. This routine sets the controller for 8-bit data entry but doesn't initialize the number of display lines and the character font of the LCD screen. This is done separately by passing the desired type of display to the controller when after lcd_init is called. The choices available are:

Screen Display	**Value**
5x8 single line	0x30
5x8 double line	0x3F
5x10 single line	0x34

These values may only be changed on startup using the function lcd_cmd. They cannot be changed after the LCD screen has been initialized. The initialization routine ends by turning on the display and cursor, clearing the entire display and setting the DDRAM address to 0.

It is important to note that this function must be customized for the processor on which it is running. This is because there are several delay loops that are executed by this routine. These are all time dependant and were designed around the microprocessor's clock, 49.76 MHz for the C8051F120. If this code is ported to other processors then these delay loops will need to be rewritten to take into account the clock frequency of the processor you are using.

Figure 5: 8 Bit Interface



lcd_cmd:
This function is used to write commands to the LCD screen. The command to be issued is passed in as a parameter of the function. There are several different commands, which can be issued to the LCD controller. These are for clearing the display, resetting the DDRAM address to 0, turning the display and cursor on or off, and shifting the display and cursor. The basic format of each of these is summarized in Table 2 above.

lcd_dat:
This function is used to write data to the LCD screen. The data to be written is passed in as a parameter of this function. This is very similar in operation to the lcd_cmd routine. The data to be displayed must be written to the display as an ASCII character. The character set stored in the controller is listed in Table 5 below.

Table 5: Character Set

Note: The user can specify any pattern for character-generator RAM.

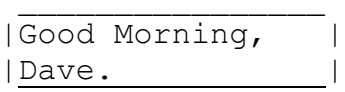
lcd_goto:
This function sets the address of the cursor. This is a special case of the lcd_cmd function. The function is almost identical to lcd_cmd, the only difference being that the value passed into the function is logically ORed with 0x80, which places a 1 in the leading location. This is used to signify that an address is being sent to the display as opposed to a character or other command.

lcd_busy_wait:
This function is used to check the busy flag of the LCD screen. When the busy flag is high, the LCD controller is still processing a previous command and cannot accept any new instructions. As a result, the busy flag must be checked before each attempt to write a command or data to the LCD screen. This step is incorporated in the lcd_cmd, lcd_dat and lcd_goto functions, as they all call the lcd_busy_wait instruction as their first operation, and wait until the flag is cleared before proceeding to issue a new instruction.

lcd_puts:
This command is used to write a string of characters, stored in a buffer, to the LCD screen. This function contains a while loop that simply makes repeated calls to the lcd_dat function until the entire buffer has been transmitted. The only limitation on the use of the buffer is that only data can be sent to the screen. Instructions must be sent separately by calling lcd_cmd.

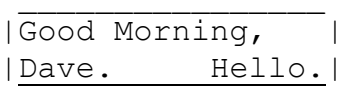
Test Code:
The C program LCDTEST8051.c is included with this document. The program executes the same set of operations and is used to exercise the LCD Screen to ensure that it is working properly by issuing a variety of instructions to the LCD screen.

The code initializes turns the display, turns it on, and then prints:

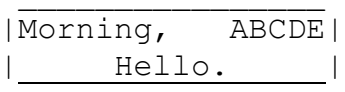


The message is held for about 5 s. The code then blinks the display once (off, then on again).

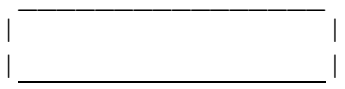
After another short delay, the cursor moves to address 0xCA and prints "Hello" at this location.



Following yet another delay, "ABCDE" is written to Address 0x41 and the display is shifted left 5 times (with delays so that they may be observed).



The last instruction clears the display after another delay.



Assembly Code:
The assembly code for the LCD screen consists of two subroutines that can be included with any program to provide an interface to the LCD controller to the SiLabs C8051. It is a good idea when using these programs to allow for a large stack, since all parameters are passed via the stack. The subroutines that are used are Writelcd, which writes the data to the LCD screen and Initlcd, which executes all the screen initialization routines.

Writelcd:
This subroutine handles all data and instruction writes to the LCD screen. The data to be written is passed to the subroutine by the A register and the RS and R/W states are passed by the B register. In the subroutine, these values are then pushed onto the stack. The subroutine then checks the busy flag. Once the busy flag is clear the data is popped off the stack and written to Port 6. Then the control pin states are then popped off the stack and written to Port 7. Several

clock cycles later the enable is pulsed and the data is written to the HD44780. The control pins are then cleared and the subroutine returns control to the calling function.

Initlcd:
This subroutine functions identically to the lcd_init function in the C code. The screen type is passed to the subroutine in the A register and is then pushed onto the stack. It is popped off the stack later in the subroutine so that it can be written to Port 7. The user does not need to specify any other information when using this subroutine.

Test Code:
This is very simple code that shows how the Writelcd and Initlcd subroutines are used in a program. This is not, however, intended in any way as a diagnostic tool to check the functioning of the screen. If there is a question as to whether or not the screen is bad, it is recommended that the sample C code be run, as it provides a far more comprehensive test of the functions and features of the HD44780.

Function differences between the C code and Assembly code routines

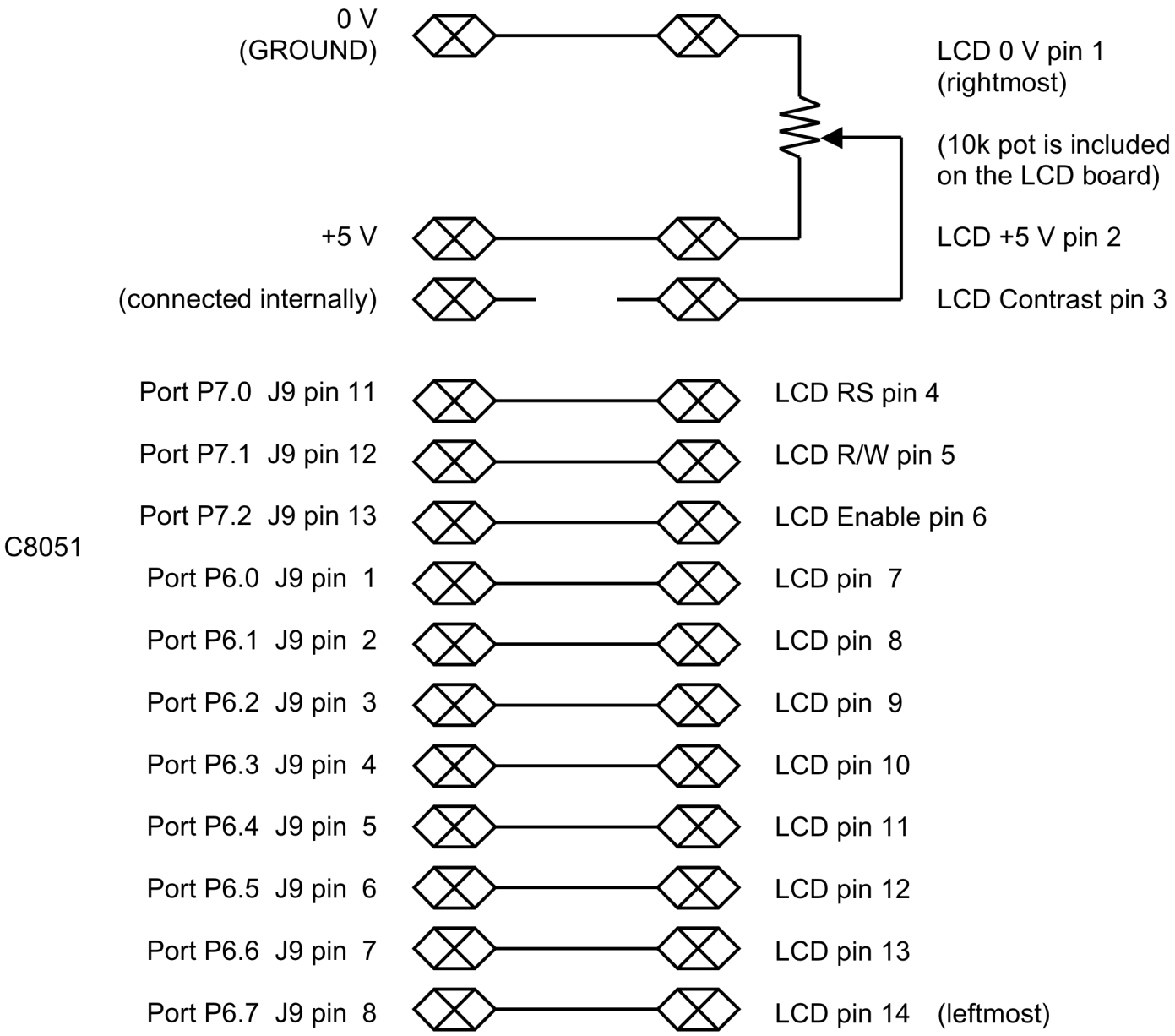
There are very few actual differences between the C header file and the assembly code subroutines. The differences between them are summarized briefly below.

1) The assembly code version reduces the number of different operations need by the programmer as it combines the lcd_cmd, lcd_dat, lcd_goto, and lcd_busy_wait functions into a single subroutine. This can be done since the only difference between the first 3 instructions is the state of the RS and R/W pins. As the programmer must pass the state of the control pins to the subroutine from the main code, the functions can be combined into a single subroutine. In addition as there is only one subroutine, the lcd_busy_wait functions can be included in that subroutine since it is not called by any other subroutine.

2) The C header file provides a function to write a string of ASCII characters to the screen. This function was omitted from the assembly code subroutine. This was left as an exercise for the students. To write a string of data or a sequence of data and instructions, the Writelcd subroutine must be slightly modified.

The programmer simply pushes the data or instructions, and the control signals for them, onto the stack in reverse execution order. The new Writelcd subroutine must receive the number of instructions stored on the stack and continue processing them until all the instructions have been transmitted. This has an advantage over the C header lcd_puts function in that it can handle both instructions and data since the control signals are passed along with the data to be written to the screen.

Appendix A: Wiring Connections



C8051 LCD Display Wiring Connections

Be sure to add a 10k pull-up on P7.2 (Enable) to +3.3 V (or +5 V) if 8051 is configured for open-drain on P7.

Appendix B: LCD.h, LCD.c and LCDTEST8051.c

LCD.h

```
//-- Filename: LCD.h
//-- Header File for LCD display routines in LCD.c
//-- Updated for SDCC C compiler and C8051F120
//-- RPK Oct 2007

#include <C8051F120.h>

//-----------------------------------------------------------------------------
// Global Defines
//-----------------------------------------------------------------------------

#define LCD_DAT_PORT  P6         // LCD is in 8 bit mode
#define LCD_CTRL_PORT P7         // 3 control pins on P7
#define RS_MASK       0x01       // for assessing LCD_CTRL_PORT

#define RW_MASK       0x02
#define E_MASK        0x04

//-----------------------------------------------------------------------------
// Global MACROS
//-----------------------------------------------------------------------------
// original delays [1]
#define pulse_E();\
       small_delay(5);\
       LCD_CTRL_PORT = LCD_CTRL_PORT | E_MASK;\
       small_delay(5);\
       LCD_CTRL_PORT = LCD_CTRL_PORT & ~E_MASK;\

//-- function prototypes ------------------------------------------------------

void lcd_init       (void);      // initialize the LCD to 8 bit mode
void lcd_busy_wait  (void);      // wait until the LCD is no longer busy
char lcs_dat        (char c);    // write data to the LCD controller
//char putchar      (char c);    // replaces standard function and uses LCD
void lcd_puts       (char string[]);// send string to LCD at current cursor location
void lcd_cmd        (char cmd);  // write a command to the LCD controller
void lcd_clear      (void);      // clear display
void lcd_goto       (char addr); // move to address addr
void lcd_home       (void);      // home cursor
//void lcd_move_cursor (char dist);// moves cursor forward or back by dist
void lcd_cursor     (bit on);    // 1 displays cursor, 0 hides it
void small_delay    (char d);    // 8 bit,  about 0.34us per count @22.1MHz
void large_delay    (char d);    // 16 bit, about 82us   per count @22.1MHz
void huge_delay     (char d);    // 24 bit, about 22ms   per count @22.1MHz

//-----------------------------------------------------------------------------
```

LCD.c

```
//-- Filename: LCD.c
//-- LCD Functions Implementation
//-- Updated for SDCC C compiler and C8051F120
//-- RPK Oct 2007
//
// P7 is used for the control signals, P7.2 = E, P7.1 = RW, P7.0 = RS, output only
// P6 is used for data: P6.7 is read to get the status of the LCD
// P6 must be configured as bidirectional (open-drain) and set to FF (or at least 80)
// before reading the status of P6.7

#include <LCD.h>

//----------------------------- LCD related Functions -------------------------------
//#pragma OPTIMIZE (7)


//-------------------------------------------------------------------------------------
// lcd_init
//-------------------------------------------------------------------------------------
//
void lcd_init(void)
{
       char SFRPAGE_SAVE;

       SFRPAGE_SAVE = SFRPAGE;                 // Save Current SFR page
       SFRPAGE = CONFIG_PAGE;

       LCD_CTRL_PORT = LCD_CTRL_PORT & ~RS_MASK;      // RS = 0
       LCD_CTRL_PORT = LCD_CTRL_PORT & ~RW_MASK;      // RW = 0
       LCD_CTRL_PORT = LCD_CTRL_PORT & ~E_MASK;       //  E = 0
       large_delay(200);                       // 16ms delay

       LCD_DAT_PORT = 0x38;                    // set 8-bit mode
       small_delay(1);            //RPK
       pulse_E();
       large_delay(50);                        // 4.1ms delay      [50]

       LCD_DAT_PORT = 0x38;                    // set 8-bit mode
       small_delay(1);            //RPK
       pulse_E();
       large_delay(3);                         // 1.5ms delay      [2]

       LCD_DAT_PORT = 0x38;                    // set 8-bit mode
       small_delay(1);            //RPK
       pulse_E();
       large_delay(3);                         // 1.5ms delay      [2]

       lcd_cmd(0x06);                          // curser moves right
       lcd_clear();
       lcd_cmd(0x0E);                          // display and curser on

       SFRPAGE = SFRPAGE_SAVE;                 // Restore SFR page
}
//#pragma OPTIMIZE (9)


//-------------------------------------------------------------------------------------
// lcd_busy_wait
//-------------------------------------------------------------------------------------
//
// wait for the busy bit to drop
//
void lcd_busy_wait(void)
{
```

```
	char SFRPAGE_SAVE;

	SFRPAGE_SAVE = SFRPAGE;		// Save Current SFR page
	SFRPAGE = CONFIG_PAGE;

	LCD_DAT_PORT = 0xFF;		// Set to FF to enable input on P6
	LCD_CTRL_PORT = LCD_CTRL_PORT & ~RS_MASK;	// RS = 0
	LCD_CTRL_PORT = LCD_CTRL_PORT | RW_MASK;	// RW = 1
	small_delay(3);		// [1 was original delay value]
	LCD_CTRL_PORT = LCD_CTRL_PORT | E_MASK;	//  E = 1
	//TB_GREEN_LED = 1;
	do
	{					// wait for busy flag to drop
		small_delay(2);	// [1 was original delay value]
	} while ((LCD_DAT_PORT & 0x80) != 0);
	//TB_GREEN_LED = 0;

	SFRPAGE = SFRPAGE_SAVE;			// Restore SFR page
}


//-----------------------------------------------------------------------------
// lcd_dat (putchar)
//-----------------------------------------------------------------------------
//
// write a character to the LCD screen
//
char lcd_dat(char dat)
//char putchar(char dat)
{
	char SFRPAGE_SAVE;

	SFRPAGE_SAVE = SFRPAGE;			// Save Current SFR page
	SFRPAGE = CONFIG_PAGE;

	lcd_busy_wait();
	LCD_CTRL_PORT = LCD_CTRL_PORT | RS_MASK;	// RS = 1
	LCD_CTRL_PORT = LCD_CTRL_PORT & ~RW_MASK;	// RW = 0
	LCD_DAT_PORT = dat;
	small_delay(1);		//RPK
	pulse_E();

	SFRPAGE = SFRPAGE_SAVE;			// Restore SFR page

	return 1;
}


//-----------------------------------------------------------------------------
// lcd_puts
//-----------------------------------------------------------------------------
//
// write a string to the LCD screen
//
void lcd_puts(char string[])
{
	int i;
	i=0;
	while(string[i])
	{
		lcd_dat(string[i]);
		i++;
	}
}


//-----------------------------------------------------------------------------
// lcd_cmd
```

```
//-------------------------------------------------------------------------------------
//
// write a command to the LCD controller
//
void lcd_cmd(char cmd)
{
       char SFRPAGE_SAVE;

       SFRPAGE_SAVE = SFRPAGE;                         // Save Current SFR page
       SFRPAGE = CONFIG_PAGE;

       lcd_busy_wait();
       LCD_CTRL_PORT = LCD_CTRL_PORT & ~RS_MASK;      // RS = 0
       LCD_CTRL_PORT = LCD_CTRL_PORT & ~RW_MASK;      // RW = 0
       LCD_DAT_PORT = cmd;
       small_delay(1);            //RPK
       pulse_E();

       SFRPAGE = SFRPAGE_SAVE;                         // Restore SFR page
}


//-------------------------------------------------------------------------------------
// lcd_clear
//-------------------------------------------------------------------------------------
void lcd_clear(void)
{
       lcd_cmd(0x01);      //-- clear LCD display
       lcd_cmd(0x80);      //-- cursor go to 0x00

}


//-------------------------------------------------------------------------------------
// lcd_goto
//-------------------------------------------------------------------------------------
void lcd_goto(char addr)
{
       lcd_cmd(addr | 0x80);
}


//-------------------------------------------------------------------------------------
// lcd_home
//-------------------------------------------------------------------------------------
void lcd_home(void)
{
       lcd_cmd(0x80);      // cursor to 0 (home)
}


//-------------------------------------------------------------------------------------
// lcd_cursor
//-------------------------------------------------------------------------------------
void lcd_cursor(bit on)    // 1 displays cursor, 0 hides it
{
       if (on)
              lcd_cmd(0x0E);
       else
              lcd_cmd(0x0C);
}


//-------------------------------------------------------------------------------------
// delay routines
//-------------------------------------------------------------------------------------
//
// delay using spin wait
```

```
//
void small_delay(char d)
{
	while (d--);
}


void large_delay(char d)
{
	while (d--)
		small_delay(255);
}


void huge_delay(char d)
{
	while (d--)
		large_delay(255);
}
```

LCDTEST8051.c

```
// Filename: LCDTEST8051.c
// Basic test program for the Hitachi HD 44780
// interfaced to the C8951F120
//-- Updated for SDCC C compiler and C8051F120
//-- RPK Oct 2007
//
// An extra 10k pull-up resistor may be needed on the E control line P7.2
// to the +3.3V supply on the P8051F120 board if P7 is configured as open-drain.

//-----------------------------------------------------------------------------
// Includes
//-----------------------------------------------------------------------------
#include <LCD.c>

//-----------------------------------------------------------------------------
// Function Prototypes
//-----------------------------------------------------------------------------
void main(void);
void SYSCLK_INIT(void);

//-----------------------------------------------------------------------------
// MAIN Routine
//-----------------------------------------------------------------------------
void main()
{
       unsigned int i, j;
       char buffer[]="Hello.";

       WDTCN = 0xDE;              // Disable the watchdog timer
       WDTCN = 0xAD;              // note: = "DEAD"!

       SYSCLK_INIT();             // Initialize the oscillator

       lcd_init();                // initialize the LCD screen
       lcd_cmd(0x3F);             // set display to 2 lines 5x8
                                  // (0x30=1 line 5x8, 0x34=1 line 5x10)
       lcd_cmd(0x0C);             // turn on display and cursor
       lcd_cmd(0x01);             // clear display

       lcd_goto(0);               // set cursor address to 0
       lcd_dat('G');              // write "Good Morning    "
       lcd_dat('o');              //       "Dave            "
       lcd_dat('o');
       lcd_dat('d');
       lcd_dat(' ');
       lcd_dat('M');
       lcd_dat('o');
       lcd_dat('r');
       lcd_dat('n');
       lcd_dat('i');
       lcd_dat('n');
       lcd_dat('g');
       lcd_dat(',');

       lcd_goto(0x40);            // set cursor address to 0x40=64
       lcd_dat('D');
       lcd_dat('a');
       lcd_dat('v');
       lcd_dat('e');
       lcd_dat('.');

       for(i=0; i<200; i++)       // long pause for display
              for(j=0; j<50000; j++);
```

```
	lcd_cmd(0x08);			// turn off display

	for(i=0; i<40; i++)		// 1 sec. pause for display
		for(j=0; j<50000; j++);

	lcd_cmd(0x0C);			// turn on display and cursor

	for(i=0; i<40; i++)		// 1 sec. pause for display
		for(j=0; j<50000; j++);

	lcd_goto(0xCA);			// set cursor address to 74 (=0x4A)

	lcd_puts(&buffer);		// write buffer to screen

	for(i=0; i<200; i++)		// long pause for display
		for(j=0; j<50000; j++);

	lcd_goto(0x10);			// go to address 16 (=0x10)
	lcd_dat(0x41);			// write ABCDE
	lcd_dat(0x42);
	lcd_dat(0x43);
	lcd_dat(0x44);
	lcd_dat(0x45);

	lcd_cmd(0x18);			// shift display left 5 times
	for(i=0; i<15; i++)		// brief pause for display
		for(j=0; j<50000; j++);
	lcd_cmd(0x18);
	for(i=0; i<15; i++)		// brief pause for display
		for(j=0; j<50000; j++);
	lcd_cmd(0x18);
	for(i=0; i<15; i++)		// brief pause for display
		for(j=0; j<50000; j++);
	lcd_cmd(0x18);
	for(i=0; i<15; i++)		// brief pause for display
		for(j=0; j<50000; j++);
	lcd_cmd(0x18);

	for(i=0;i<200;i++)		// long pause for display
		for(j=0; j<50000; j++);

	lcd_cmd(0x01);			// clear display
}

//-----------------------------------------------------------------------------
// SYSCLK_Init
//-----------------------------------------------------------------------------
//
// Initialize the system clock to use a 22.11845MHz crystal as its clock source
//
void SYSCLK_INIT(void)
{
	int i;
	char SFRPAGE_SAVE;

	SFRPAGE_SAVE = SFRPAGE;		// Save Current SFR page
	SFRPAGE = CONFIG_PAGE;

	OSCXCN    = 0x67;
	for (i = 0; i < 3000; i++);	// Wait 1ms for initialization
	while (!(OSCXCN & 0x80));
	CLKSEL    = 0x01;

	XBR0=0x04;
	XBR1=0x80;
	XBR2=0x40;			// Enable Crossbar with Weak Pullups

	P7MDOUT=0x07;			// Set E, RW, RS controls to push-pull
```

```
    P6MDOUT=0x00;                  // P6 must be open-drain to be bidirectional:
                                   // used for both read & write operations

    SFRPAGE = SFRPAGE_SAVE;        // Restore SFR page
}
```