Application Note

Interfacing LCD Modules to the Z8 MCU

AN003201-Z8X0500
This publication is subject to replacement by a later edition. To determine whether a later edition exists, or to request copies of publications, contact:

ZiLOG Worldwide Headquarters
910 E. Hamilton Avenue
Campbell, CA 95008
Telephone: 408.558.8500
Fax: 408.558.8300
www.ZiLOG.com

Windows is a registered trademark of Microsoft Corporation.

Information Integrity
The information contained within this document has been verified according to the general principles of electrical and mechanical engineering. Any applicable source code illustrated in the document was either written by an authorized ZILOG employee or licensed consultant. Permission to use these codes in any form, besides the intended application, must be approved through a license agreement between both parties. ZILOG will not be responsible for any code(s) used beyond the intended application. Contact the local ZILOG Sales Office to obtain necessary license agreements.

Document Disclaimer
© 2000 by ZILOG, Inc. All rights reserved. Information in this publication concerning the devices, applications, or technology described is intended to suggest possible uses and may be superseded. ZILOG, INC. DOES NOT ASSUME LIABILITY FOR OR PROVIDE A REPRESENTATION OF ACCURACY OF THE INFORMATION, DEVICES, OR TECHNOLOGY DESCRIBED IN THIS DOCUMENT. ZILOG ALSO DOES NOT ASSUME LIABILITY FOR INTELLECTUAL PROPERTY INFRINGEMENT RELATED IN ANY MANNER TO USE OF INFORMATION, DEVICES, OR TECHNOLOGY DESCRIBED HEREIN OR OTHERWISE. Except with the express written approval ZILOG, use of information, devices, or technology as critical components of life support systems is not authorized. No licenses or other rights are conveyed, implicitly or otherwise, by this document under any intellectual property rights.
Table of Contents

General Overview .............................................................. 1
Discussion ........................................................................... 1
  LCD Module Basics ......................................................... 1
  Z8 Microcontroller ......................................................... 7
  Connecting the LCD Module ............................................. 7
  Serial Interface ................................................................ 8
  Firmware ........................................................................ 8
  Operational Results ......................................................... 13
Summary ............................................................................. 14
Technical Support ............................................................... 14
  Assembling the Application Code .................................... 14
  Source Code ................................................................... 15
Test Procedure .................................................................... 26
  Equipment Used ............................................................. 26
  General Test Setup and Execution .................................. 27
  Test Results .................................................................... 27
References ........................................................................ 28
Appendix ........................................................................... 28

Acknowledgements

Project Lead Engineer
John D. Conder

Application and Support Engineer
John D. Conder

System and Code Development
John D. Conder
Interfacing LCD Modules to the Z8 MCU

There is an increasing demand to interface Liquid Crystal Display (LCD) modules to low-end microcontrollers. Unfortunately, little information is offered to address real-world applications and to help the design engineer understand how to make LCD modules work.

General Overview

This Application Note provides a detailed example of creating a simple serial interface to an LCD module, using a Z8 microcontroller. The application allows text messages typed on a computer keyboard to be directly displayed on the LCD module when the circuit is connected to the computer serial port. The messages are up to 16 characters long, and are terminated by the Enter key. If more than 16 characters are entered, complete lines of 16 characters are displayed until the Enter key terminates the message. The serial interface has the following configuration:

- 4800 baud
- 8 data bits
- 1 start bit, 1 stop bit
- No parity

This Application Note utilizes the Z86E08 microcontroller and the Hyundai Electronics Industries (HEI) HC16102 LCD module. However, the code is applicable to any processor in the Z8 family. Also, because the HC16102 module is based on the Hitachi HD44780 controller, the code is applicable to any other modules which utilize this device to control the LCD.

Discussion

LCD Module Basics

Figure 1 contains a block diagram of the HC16102 module. The HC16102 module contains a 1-line by 16-character display, a Hitachi HD44780 display controller, and an LED backlight. The HD44780 divides the 16 characters into two lines of eight characters each, even though all characters appear physically on the same line. The controller has an on-board character generator in ROM capable of displaying 192 ASCII characters, along with eight user-programmable characters. All characters are displayed in a 5x7 font. The module is also capable of configuring
the data bus for either an 8-bit or a 4-bit interface. This Application Note utilizes the 8-bit option. Table 1 indicates the module’s pin configuration.

Figure 1. LCD Module Block Diagram

<table>
<thead>
<tr>
<th>Pin</th>
<th>Symbol</th>
<th>Signal Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$V_{SS}$</td>
<td>GND</td>
</tr>
<tr>
<td>2</td>
<td>$V_{DD}$</td>
<td>Power Supply</td>
</tr>
<tr>
<td>3</td>
<td>$V_O$</td>
<td>LCD Driver Supply Voltage</td>
</tr>
<tr>
<td>4</td>
<td>RS</td>
<td>Register Select: 0 = Instruction, 1 = Data</td>
</tr>
<tr>
<td>5</td>
<td>R/W</td>
<td>Read/Write: 0 = MPU to LCM, 1 = LCM to MPU</td>
</tr>
<tr>
<td>6</td>
<td>E</td>
<td>Enable: (active high)</td>
</tr>
<tr>
<td>7 to 14</td>
<td>DB0 to DB7</td>
<td>Data Bus</td>
</tr>
<tr>
<td>15</td>
<td>A</td>
<td>Anode of LED Backlight</td>
</tr>
<tr>
<td>16</td>
<td>K</td>
<td>Cathode of LED Backlight</td>
</tr>
</tbody>
</table>
The module is configured and controlled by the microcontroller via the instruction set listed in the HC16102 Instruction Set in Table 2. (Table 2 uses a number of abbreviations that are defined at the bottom of each page of the table.) Each instruction has a maximum execution time. Upon issuing an instruction, the microcontroller waits for at least the maximum execution time before issuing another instruction. Most execution times are at least 40 µsec.

The process of writing an instruction to the module is very straightforward. Essentially, the RS, R/W, and DB7:0 signals are set to the proper levels and then the E signal is pulsed. On the falling edge of the E signal, the instruction is accepted and processed by the HD44780. The minimum pulse width for the E signal is 230 ns with a minimum cycle time of 500 ns.

### Table 2. HC16102 Instruction Set

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Code</th>
<th>Description</th>
<th>Execution Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clear Display</td>
<td>000000001</td>
<td>Clears entire display and sets DD RAM address 0 in address counter.</td>
<td>1.64ms</td>
</tr>
<tr>
<td>Return Home</td>
<td>00000001x</td>
<td>Sets DD RAM address 0 in address counter. Also returns display from being shifted to original position. DD RAM contents remain unchanged.</td>
<td>1.64ms</td>
</tr>
</tbody>
</table>

**Notes:**
- **I/D:** 1 = Increment, 0 = Decrement
- **S:** 1 = Accompanies display shift
- **S/C:** 1 = Display Shift, 0 = Cursor move
- **R/L:** 1 = Shift to the right, 0 = Shift to the left
- **DL:** 1 = 8-bit interface, 0 = 4-bit interface
- **N:** 1 = 2 lines, 0 = 1 line
- **F:** 1 = 5x10 dot array, 0 = 5x7 dot array
- **BF:** 1 = Busy, 0 = Instructions acceptable
- **x:** Don’t care
- **ACG:** CG RAM address
- **ADD:** DD RAM address (corresponds to cursor address)
- **AC:** Address counter used for both DD and CG RAM
- **DD RAM:** Display data RAM
- **CG RAM:** Character generator RAM
### Table 2. HC16102 Instruction Set (Continued)

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Code</th>
<th>Description</th>
<th>Execution Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entry Mode Set</td>
<td>0 0 0 0 0 0 1 I/D S</td>
<td>Sets cursor move direction and specifies shift of display. These operations are performed during data write and read.</td>
<td>40µs</td>
</tr>
<tr>
<td>Display On/Off</td>
<td>0 0 0 0 0 1 D C B</td>
<td>Sets On/Off of entire display (D), cursor On/Off (C), and blinking of cursor position character (B).</td>
<td>40µs</td>
</tr>
<tr>
<td>Cursor or Display Shift</td>
<td>0 0 0 0 1 S/C R/L x x</td>
<td>Moves cursor &amp; shifts display without changing DD RAM contents.</td>
<td>40µs</td>
</tr>
<tr>
<td>Function Set</td>
<td>0 0 0 1 DL N F x x</td>
<td>Sets interface data length (DL), number of display lines (N), and character font (F).</td>
<td>40µs</td>
</tr>
<tr>
<td>Set CG RAM Address</td>
<td>0 0 1 ACG</td>
<td>Sets CG RAM address. CG RAM data is sent and received after this setting.</td>
<td>40µs</td>
</tr>
</tbody>
</table>

Notes:
- **I/D**: 1 = Increment, 0 = Decrement
- **S**: 1 = Accompanies display shift
- **S/C**: 1 = Display Shift, 0 = Cursor move
- **R/L**: 1 = Shift to the right, 0 = Shift to the left
- **DL**: 1 = 8-bit interface, 0 = 4-bit interface
- **N**: 1 = 2 lines, 0 = 1 line
- **F**: 1 = 5x10 dot array, 0 = 5x7 dot array
- **BF**: 1 = Busy, 0 = Instructions acceptable
- **x**: Don’t care
- **ACG**: CG RAM address
- **ADD**: DD RAM address (corresponds to cursor address)
- **AC**: Address counter used for both DD and CG RAM
- **DD RAM**: Display data RAM
- **CG RAM**: Character generator RAM
Message data must be initialized before it is written to the module. Initialization is done either internally by the module’s reset circuit or externally by instructions from the microcontroller. For the module’s internal reset circuit to successfully complete initialization, the \( V_{dd} \) signal must transition smoothly to 4.5V within 0.1 milliseconds and 10.0 milliseconds. If this condition is not guaranteed, the module must be initialized by the microcontroller. For this Application Note, the initialization is performed externally by the microcontroller. The flowchart in Figure 2 illus-

Table 2. HC16102 Instruction Set (Continued)

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Code</th>
<th>Description</th>
<th>Execution Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set DD RAM Address</td>
<td>0 0 1 ADD</td>
<td>Sets DD RAM address. DD RAM data is sent and received after this setting.</td>
<td>40µs</td>
</tr>
<tr>
<td>Read Busy Flag</td>
<td>0 1 BF AC</td>
<td>Reads busy flag (BF) indicating internal operation is being performed and reads address counter contents (AC).</td>
<td>0µs</td>
</tr>
<tr>
<td>Write Data to CG or DD RAM</td>
<td>1 0 DATA</td>
<td>Writes data into DD RAM or CG RAM.</td>
<td>40µs</td>
</tr>
<tr>
<td>Read Data from CG or DD RAM</td>
<td>1 1 DATA</td>
<td>Reads data from DD RAM or CG RAM.</td>
<td>40µs</td>
</tr>
</tbody>
</table>

Notes:

- I/D: 1 = Increment, 0 = Decrement
- S: 1 = Accompanies display shift
- S/C: 1 = Display Shift, 0 = Cursor move
- R/L: 1 = Shift to the right, 0 = Shift to the left
- DL: 1 = 8-bit interface, 0 = 4-bit interface
- N: 1 = 2 lines, 0 = 1 line
- F: 1 = 5x10 dot array, 0 = 5x7 dot array
- BF: 1 = Busy, 0 = Instructions acceptable
- x: Don’t care
- ACG: CG RAM address
- ADD: DD RAM address (corresponds to cursor address)
- AC: Address counter used for both DD and CG RAM
- DD RAM: Display data RAM
- CG RAM: Character generator RAM
istrates the initialization sequence. Note that the sequence is for 8-bit mode. A slightly different sequence exists for 4-bit mode. However, the 4-bit mode sequence is beyond the scope of this application and therefore not included in this Application Note.

Figure 2. LCD Module Initialization Sequence
Z8 Microcontroller

The microcontroller selected for this application is the Z86E08. It provides a minimal configuration for the application. As such, there are very few resources remaining for interfacing to other circuitry. For applications that require additional connections, other Z8 microcontrollers, such as the 28-pin Z86E3x series or the 40-pin Z86E4x series may be substituted with minimal impact to the firmware.

The Z86E08 is an 18-pin device with 14 available I/O pins. The I/O pins are arranged into two 3-bit ports (Port 0 and Port 3) and one 8-bit port (Port 2). Port 0 (P02–P00) is a dedicated output port. Port 2 (P27–P20) is a bidirectional port with each pin independently configurable as input or output. Port 3 (P33–P31) is a dedicated input port.

The Z86E08 is capable of operating at a crystal frequency of up to 12 MHz. 8 MHz was chosen for this application because it simplifies the system timing. Other frequencies may be selected. However, other frequencies require a reevaluation of the serial interface timing as well as the timing of the firmware delay loops that are used in the LCD Module interface.

Connecting the LCD Module

The Appendix contains the schematic for this application. As illustrated in the schematic, the LCD Module data bus, DB7:0, is connected directly to Port2 of the Z8 microcontroller. Port2 was chosen because it is the only bidirectional port available on the Z86E08. It is also the only 8-bit wide port available. For other Z8 applications, other ports may be used. The LCD Module control signals, E, R/W, and RS, are connected to Port0 of the Z8 because they are strictly module inputs. These signals could be connected to any Port0 pin. The required connection is used because the E signal is toggled most frequently. Therefore, it is assigned to the least significant Port0 pin, P00.

The module provides two pins for the LED backlight, pin 15 for the anode (A) and pin 16 for the cathode (K). The anode is connected to VCC by a 10-Ohm resistor, R2, and the cathode is connected to Gnd via a push-button switch, SW1. The LED has a forward voltage of approximately 4.1 V and a forward current of 110 mA. The resistor is provided to limit the current to the 110mA requirement when the button is pressed.

Contrast for the LCD is controlled by the voltage applied at pin 3, VO, of the LCD Module. The contrast is adjusted by connecting the wiper leg of potentiometer R3 to VO and connecting the other legs to VCC and Gnd. A 10-kΩ potentiometer is used in this application, but any value from 10K to 20K is acceptable.
Serial Interface

The serial interface for this application functions only to receive data from a host source. Therefore, the serial port connection is limited to the receive data pin (DB9 connector pin 3) and the Gnd pin (DB9 connector pin 5). Traditionally, an RS-232 buffer device is used to isolate the processor from the receive pin of the connector. However, for this application, the buffer device is eliminated and a simple analog circuit replaces it. This circuit consists of two diodes, D1 and D2, and resistor R1. Diode D1 limits the voltage on the processor pin to approximately 0.7V above $V_{CC}$, while diode D2 limits the voltage to approximately 0.7V below Gnd. This limitation protects the processor pin from the damaging voltages present when connected to an RS-232 driver. Resistor R1 limits the current into or out of the circuit to just a few milliamps, thereby protecting the diodes and processor pin from potential damage due to excessive current.

One of the key differences in this serial interface is that it requires the processor to invert the sense of the serial data transitions. This inversion is traditionally performed by the RS-232 buffer, which has been eliminated. From the processor’s point of view, the start bit is logic level 1 instead of the traditional level 0, all the data is inverted, and the stop bit is logic level 0.

Firmware

The majority of the functions for this application reside in the Z8 firmware. This firmware consists of three major routines: a main processing loop, a serial input routine, and a display write routine. Figure 3 contains a high-level flowchart for the main processing loop.
Figure 3. Main Processing Loop Flowchart

Power On

Initialize Z8 ports and registers
Initialize LCD module

Write ">Ready" to the display

Reconfigure Timers and Interrupts
Reset the message buffer pointer

Overrun flag set?

Yes

Add 17th character to message buffer
Clear the Overrun flag

No

Character received?

Yes

Yes

Is it 10hex ("CR")?

No

No

Fill missing characters with " "
Write the message to the display

Is it 13hex ("LF")?

No

Yes

Set the Overrun flag

Is it the 17th character?

No

Add character to the message buffer

No

Frame error?

Yes

Abort current message
Write ">Frame Error!!" to display

No

No
The basic operation of the main processing loop is as follows. On power-up, the Z8 performs an initialization sequence and then enters into a main loop waiting for an indication that a new character has been received via the serial routine. When a new character is received, the character is examined to determine the next action. If the character is 10h (ASCII carriage return), the message is considered terminated. The message buffer is padded with as many spaces as required to complete the 16-character message and the message is written to the LCD Module. The Z8 reenters the main loop waiting for new characters. The 13h character (ASCII line feed) is ignored completely because it is part of the message termination. If any other character is received, it is written to the message buffer and the buffer pointers are incremented. If the number of characters exceeds 16, without receiving character 10h, the message buffer is automatically written to the LCD Module and message buffering is restarted.

Figure 4 contains the high-level flowchart for the serial input routine. Because the Z86E08 doesn’t contain a UART, a firmware routine provides the receive portion of the UART function. Essentially, the firmware routine detects the rising edge of the start bit, samples it again in the middle to validate it, and then repetitively samples at bit-time intervals to obtain the data and stop bits.
Figure 4. Serial Input Routine Flowchart

IRQ3

Wait for 1/2 a bit time (104us)
Resample the P31 data pin

Still logic 1?

Yes

Mask off IRQ3 & enable IRQ4 (T0)
Clear the serial word buffer
Initialize bit counter

Halt

No

IRQ4

Data bit = 1, rotate into word buffer

P31 = "0"?

Yes

No stop bit - set error flag

No

Data bit = 0, rotate into word buffer

Decrement bit counter

Bit count = 0?

Yes

Halt

No

IRQ4

Stop bit OK - set char available flag

P31 = "0"?

Yes

No stop bit - set frame error flag

No

Disable T0, re-enable IRQ3 only

Return
By connecting the receive data signal to P31 of the Z86E08, the rising edge of the start bit is used to generate an interrupt, IRQ3, to the processor whenever a start bit occurs. The bit-time delay is created by configuring the internal T0 timer for a time delay of 208 µs. This operation results in an effective baud rate of 4808. When the start bit is detected and validated, the IRQ3 interrupt is masked off and the T0 interrupt, IRQ4, is enabled. The processor is halted between data bits, and resumes processing when the IRQ4 interrupt occurs. If the start bit validation fails, the IRQ3 interrupt is considered to be noise and is ignored. If the stop bit is not present, a frame error is declared by setting the frame error flag. If the word is completed successfully, a flag is set to denote that a new character is available.

Figure 5 contains the high level flowchart for the write display routine. This routine expects a 16-character ASCII message to be in the message buffer. The routine performs 16 consecutive writes to the LCD Module, beginning with the first word in the buffer and writing the next sequential word on each pass. Because the HD44780 LCD controller is configured as two 8-character lines, the 16 character write is broken into two passes of eight characters each. Between the two passes, the display controller must be instructed to switch from line 1 to line 2.
Operational Results

This application is designed to connect to a regulated +5V power supply. The supply regulation must be within +/- 0.5V, which is the operational limit of the LCD module. Connection to any other power source requires additional circuitry to provide this level of supply voltage and regulation.

When power is applied to the unit, it responds by displaying >Ready on the LCD Module. This display is an indication that the unit is ready to receive messages.
The unit displays any message it receives. Note that the LCD module uses the character space greater than 07Fh as Asian characters and other symbols. Messages containing these codes produce interesting results. Consult the data sheet for the HC16102 or the HD44780 for a complete listing of the character map.

Operating the unit without connection to a serial port can sometimes display the >Frame Error!!! message. This message displays because noise is coupled into the unconnected data line. Depending on the amount of noise and its intensity, the display may appear to flash the message. An optional capacitor, C3, is included (see Appendix schematic) to prevent the noise from creating false message starts. C3 has no other purpose and the capacitor can be eliminated from designs that are not required to function while disconnected from the host.

Summary

This application provides an effective and reusable demonstration of applying the basic LCD Module instructions to control an LCD display. A simple serial interface is demonstrated for sending messages to the display. Because the LCD Module utilizes a common Hitachi HD44780 controller, the firmware has wide applicability.

The application code readily fits into the 2-KB program space of the Z86E08, using only 510 bytes. There is a large amount of space available for creating special effects or adding precoded messages. Also, if additional I/O pins are required, the code readily transfers to higher pin-count microcontrollers in the Z8 family.

Technical Support

Assembling the Application Code

Any Z8 assembler may be used, but the ZiLOG Developer Studio (ZDS) is recommended. This integrated suite of software tools allows for program file handling, editing, real-time emulation and debugging when used with the appropriate emulator. Future versions of ZDS incorporate a C-Compiler, simulator and trace buffer. See ZiLOG’s web-page at www.zilog.com for news and free downloads of ZDS.

Place the .ASM file and .INC file in their own sub-directory. Invoke ZDS and select a new project from the file menu. Under Target Selection, select Family. Under Master Select, select Z8. Under Project Target, select Z86E08. Select the appropriate emulator type to be used. Browse to fill in the project name by clicking on the ... key. Select the sub-directory containing the .ASM and .INC files, name the project, (the extension is added for you), click Save and the first ZDS screen reappears with the project name, path, and file extension filled in. If everything is acceptable, click OK.
Click on the **Project** tab and select **Add to Project**. Then select **Files**. Double click on the **LCM_Interface.asm** file. This file and the **.INC** file are now displayed in the project window. Click on the **Build** tab and select **Build**. The **Output** window displays the assembly results. The standard assembler and linker settings produce listing and hex files, along with the ZDS files, in the same sub-directory. Save the project and files by clicking on the **File** tab and selecting these options. The ZDS Project File is included, and when the ZDS is installed, allows you to skip the above steps for program assembly.

Program the OTP by selecting the OTP option with the hex code installed. Never install the OTP until access to it is required, either for blank checking, verification, or programming. Insert a blank Z86E08 into the OTP socket and click on the program OTP selection. Differences exist between earlier GUIs and the ZDS, so take the time to read and understand the operation of the SW in use. Pad unused memory locations with **FFh** before programming. If padding is not consistently done, differences occur in the check sum.

**Source Code**

This application uses the following source files:

- **LCM_Interface.asm**
- **RegDef.inc**

Instead of displaying each file separately, they are shown exactly in the order and location they are `<included>` in the main source, **LCM_Interface.asm**. This is similar to the way the output listing file (**LCM_Interface.lst**) is generated.

**************************************************************************
*       Module Name: Z8 based Serial Interface to a LCD Module
*       Copyright: ZiLOG Inc.
*       Date: 09/24/99
*       Created by: John D. Conder
*       Modified by:
*       Description: This module contains the code for using the
*                   Z86E08 microcontroller to create a 4800 baud RS232
*                   serial interface to a Hyundai HC16102 LCD module.
*                   The module has a 16-character by 1-line display
*                   format. The controller will display a 0 to 16
*                   character message received via the serial port.
*                   Messages are terminated by the "Enter" key (ASCII
*                   code sequence 13hex, 10hex). The display is not
*                   updated until either the "Enter" is received or
*                   16 characters are received. The serial interface
*                   is fixed at 8 data bits, no parity, 1 start bit
*                   and 1 stop bit.

**************************************************************************
### Application Note

**Interfacing LCD Modules to the Z8 MCU**

---

```assembly
实施方案

; Include the register and constant definitions
#include "RegDef.inc"

;>Title: RegDef.inc
;=DATE: September 24 1999
;=PURPOSE: Register and constant definitions for the=
;=LCD Module interface app note
;=FILE TYPE: .included header file
;=ASSEMBLER: ZiLOG ZDS/ZMASM
;=PROGRAMMER: John Conder

;BIT DEFINITIONS

;Port 0
;pins na--------na 13 12 11 Function Polarity I/O
;bits [7 6 5 4 3 2 1 0]
;
; | LCM Enable | hi-true | 0
; | LCM R/W | bipolar | 0
; | LCM Reg Sel | bipolar | 0

P0_E_Hi .equ 00000001b ; LCM E control bit = 1
P0_RW_Hi .equ 00000010b ; LCM RW control bit = 1
P0_RS_Hi .equ 00000100b ; LCM RS control bit = 1

P0_Init .equ 00000000b ; Reg 000 - Port0 Data Init

;Port 1
;pins na-----------na (Port 1 nonexistent on Z86E08)
;bits [7 6 5 4 3 2 1 0]
;
P01M_Init .equ 00000100b ; Reg 0F8 - Port0&1 Mode Init
; xxxxxx00b ; P00-P03 Mode=Outputs
; xxxxxxxxxb ; 1=Reserved
; 000000xxb ; 0=Reserved
```
Interfacing LCD Modules to the Z8 MCU

Port 2
pins 4 3 2 1 18 17 16 15 Function Polarity I/O
bits [7 6 5 4 3 2 1 0]
<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>DB0</td>
<td>bipolar</td>
<td>I/O</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DB1</td>
<td>bipolar</td>
<td>I/O</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DB2</td>
<td>bipolar</td>
<td>I/O</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DB3</td>
<td>bipolar</td>
<td>I/O</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DB4</td>
<td>bipolar</td>
<td>I/O</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DB5</td>
<td>bipolar</td>
<td>I/O</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DB6</td>
<td>bipolar</td>
<td>I/O</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DB7</td>
<td>bipolar</td>
<td>I/O</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

P2_Init .equ 00000000b ; Reg 002 - Port2 Data Init
P2_LCM8bits .equ 00111000b ; LCM 8bit function set
P2_DOnNoC .equ 00001100b ; LCM display on, no cursor
P2_AIncNoS .equ 00000110b ; LCM addr incr, no shift
P2_DspClr .equ 00000001b ; LCM display clear
P2_CurHome .equ 00000010b ; LCM cursor home
P2_CGRam .equ 01000000b ; LCM set CG Ram
P2_DDRam1 .equ 10000000b ; LCM DD Ram line 1
P2_DDRam2 .equ 11000000b ; LCM DD Ram line 2

Port 3
pins na-------na 10 9 8 na Function Polarity I/O
bits [7 6 5 4 3 2 1 0]
<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Unused</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Serial Input</td>
<td>bipolar</td>
<td>I</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unused</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

P3_RxBit .equ 00000100b ; Serial Input data bit
P3_Init .equ 00000000b ; Reg 003 - Port3 Data Init
P3M_Init .equ 0000001b ; Reg 0F7 - Port3 Mode
| xxxxxxx1b | 1=Port2 as Push-Pull |
| xxxxxxx0x | 0=P33-P31 as Digital Mode |
| 000000xb | 0=Reserved |

Timer definitions
TMR_Init .equ 00000000b ; Reg 0F1 - Timer Mode
| xxxxxxx0 | 1=Load T0 |
| xxxxxxx0x | 1=Enable T0 Count |
| xxxxx0xx | 1=Load T1 |
| xxxxx0xxx | 1=Enable T1 Count |
| xx00xxxx | 00=Tin Mode: external clock |
| 00xxxxxx | 00=Reserved |
TMR_RxEnab .equ 00000011b ; Enable Serial Receive

Timer0 (serial input)
T0_Init .equ 208 ; Reg 0F4 - 208uS (4808bps)
PRE0_Init .equ 00000101b ; Reg 0F5 - T0 Prescaler
; xxxxxxx1 ; 1=Modulo n
; xxxxxxx0x ; 0=Reserved
; 000001xx ; Modulo value (lus @ 8MHz)

; Timer1 (unused)
T1_Init .equ 208 ; Reg 0F2 - 208us (4808bps)
PRE1_Init .equ 00000111b ; Reg 0F3 - T1 Prescaler
; xxxxxxx1 ; 1=Modulo-n
; xxxxxxx1x ; 1=Internal Clock Source
; 000001xx ; Modulo value (lus @ 8MHz)

; Interrupt definitions

IPR_Init .equ 00100011b ; Reg 0F9 - Interrupt Priority
; xxx00xx1 ; Group priority C>A>B
; xxxxxxx1x ; Group C 1=IRQ4>IRQ1
; xxxxx0xx ; Group B 0=IRQ2>IRQ0
; xx1xxxxx ; Group A 1=IRQ3>IRQ5
; 00xxxxxx ; Reserved-Must be 0

IMR_Init .equ 00000000b ; Reg 0FB - Interrupt Mask
; xx000000 ; 0=IRQ5-IRQ0 disabled
; x0xxxxxx ; 0=Reserved-Must be 0
; 0xxxxxxx ; 0=Global Interrupt disable
IMR_RxEnab .equ 00001000b ; Enable IRQ3(P32) for serial detect
IMR_RxLoop .equ 00010000b ; Enable IRQ4(T0) for serial loop

IRQ_Init .equ 00000000b ; Reg 0FA - Interrupt Request
; xx000000 ; 0=Clear request bits 5-0
; 00xxxxxx ; 0=Reserved-Must be 0
IRQ_T0bit .equ 00010000b ; T0 interrupt bit

; System Definitions

RegBot .equ 004h ; Bottom register
RegTop .equ 07Fh ; Top register
StackTop .equ RegTop+1 ; Top of Stack

AppRP .equ 010h ; System Register Pointer
MsgBufRP .equ 020h ; Message Buffer Pointer

Wt1MCnst .equ 01Fh ; WaitLoop1 msbyte count constant
Wt1LCnst .equ 0FFh ; WaitLoop1 lsbyte count constant
Wt2Const .equ 01Fh ; WaitLoop2 count constant

; System Flags

RxWrdAvail .equ 00000001b ; Serial input word available
RxFrameErr .equ 00000010b ; Serial input frame error
RxOverrun .equ 00000100b ; Display line greater than 16
**************************************************************************
*       Global variables
********************************************************************************
DEFINE REGDATA, SPACE=RFILE

; RAM MAP
SEGMENT REGDATA

; Register Bank0
DS 16 ; Bank0 Space

; Register Bank1 - AppRP (010-01Fhex)
FlagReg DS 1 ; R0 - System Flag register
RxBitNum DS 1 ; R1 - Current bit
RxChrNum DS 1 ; R2 - Current character
RxWrdBuf DS 1 ; R3 - Serial Word Buffer
Unused DS 7 ; R4-R10

IRQ3Gnr1 DS 1 ; R11 - Interrupt general reg
Genral0 DS 1 ; R12 - Main general reg0
Genral1 DS 1 ; R13 - Main general reg1
Genral2 DS 1 ; R14 - Main general reg2
Genral3 DS 1 ; R15 - Main general reg3

; Register Bank2 - SerialRP (020-02F hex)
DS 16 ; Message Buffer space

; END RAM MAP

********************************************************************************
*       Global function declarations
********************************************************************************
; none

********************************************************************************
*       Interrupt Vectors
********************************************************************************
SEGMENT code

vector reset = Main
vector irq0 = IRQ0
vector irq1 = IRQ1
vector irq2 = IRQ2
vector irq3 = IRQ3
vector irq4 = TMR0
vector irq5 = TMR1

********************************************************************************
*       Z8 based Serial Interface to a LCD Module
*
**
This is the main section of the program. It is essentially a loop that cycles each time a complete message is received. The message is considered complete when the "CR" + "LF" characters are received. If the message exceeds 16 characters, the characters are displayed in lines of 16 characters with each new line overwriting the previous one. The receipt of individual characters is denoted by activation of the character available flag.

**************************************************************************
Main:
  di ; Disable interrupts
 ; Init Ports
   ld P2M,#P2M_Init ; Init Port2 Mode
   ld P2,#P2_Init ; Init Port2 Data
   ld P3M,#P3M_Init ; Init Port3 Mode
   ld P3,#P3_Init ; Init Port3 Data
   ld P01M,#P01M_Init ; Init Port0&1 Mode
   ld P0,#P0_Init ; Init Port0 Data
 ; Init Timer subsystem
   ld T0,#T0_Init ; Init T0
   ld PRE0,#PRE0_Init ; Init PRE0
   ld T1,#T1_Init ; Init T1
   ld PRE1,#PRE1_Init ; Init PRE1
   ld TMR,#TMR_Init ; Init TMR
 ; Init Interrupt subsystem
   ld IPR,#IPR_Init ; Init Interrupt Priority
   ld IMR,#IMR_Init ; Init Interrupt Mask
   ld IRQ,#IRQ_Init ; Clear any IRQ prior to ei
   ld RP,#AppRP ; Initialize the register pointer
 ; Clear Register Banks for debug clarity
   ld RegBot,#RegBot+1 ;
   clr @RegBot ;
   inc RegBot ;
   cp RegBot,#RegTop+1 ;
   jr ule,ClrRam ;
    ld SPL,#StackTop ; Initialize the stack pointer
   ld P2M,#P2M_Write ; Port2 Mode = output
call LCM_Wait1 ; Wait for LCM to stabilize
call LCM_Init ; Initialize the LCD Module
   ld R12,#>MsgInit ; Create a message pointer
   ld R13,#<MsgInit ;
call MsgBufLoad ; Load the message buffer
call DisplayMsg ; Write message to the LCD module
Msg_Loop:
 ; Reconfigure to recieve new input string and wait for characters
di
   ld TMR,#TMR_Init ; Disable Timers
   ld IMR,#IMR_RXEnab ; Enable IRQ3 start bit detect
   ld IRQ,#IRQ_Init ; Clear all pending interrupts
   ld R2,#MsgBufRP ; Initialize buffer location
tm R0,#RxOverrun ; Check for overrun situation
jr z,ClrBuffer ; If no overrun, clear the buffer
ld @R2,R3 ; Save the overrun character
inc R2 ; Move character pointer
and R0,#~RxOverrun ; Clear the Over-run flag
ClrBuffer:
clr R3 ; Clear the word buffer
ei
Chr_Wait:
tm R0,#RxWrdAvail ; Valid character received?
jr nz,Rcvd_Char ; If so, jump
tm R0,#RxFrameErr ; Frame error received?
jr z,Chr_Wait ; If not, jump & continue waiting

; Frame error detected - abort the input string - display error message
di
ld TMR,#TMR_Init ; Disable Timers
ld IMR,#IMR_Init ; Mask off all interrupts
ld IRQ,#IRQ_Init ; Clear any pending interrupts
ld R12,#>MsgFrErr ; Create a message pointer
ld R13,#<MsgFrErr ;
call MsgBufLoad ; Load the message buffer
call DisplayMsg ; Write message to the LCD module
and R0,#~RxFrameErr ; Clear the frame error flag
jr Msg_Loop ; Return to main message loop

Rcvd_Char:
; Complete character's been recieved - check for end message or
; buffer overflow - otherwise, save char & continue looking for more.
cp R3,,'#r' ; End of message: "CR"?
jr eq,EndOfMsg ; If so, jump to display it
cp R3,,'#n' ; else, "LF" character?
jr eq,IgnoreLF ; If so, jump to ignore it
cp R2,#MsgBufRP+10h ; else, check for overrun
jr uge,MsgOvrRun ; If overrun, jump to display
inc R2 ; Move character pointer
and R0,#~RxWrdAvail ; Clear the char available flag
jr Chr_Wait ; Return to wait for more

IgnoreLF:
; End of input message - ignore the line feed character
and R0,#~RxWrdAvail ; Clear char available flag
jr Chr_Wait ; Return to wait for more

MsgOvrRun:
; More than 16 characters have been recieved - display partial message
or R0,#RxOverrun ; Set the Over-run flag

EndOfMsg:
; End of input message - blank fill empty buffer space, if any, and
; write the message to the LCD module.
di
and R0,#~RxWrdAvail ; Clear char available flag
ld TMR,#TMR_Init ; Disable Timers
ld IMR,#IMR_Init ; Mask all interrupts
ld IRQ,#IRQ_Init ; Clear any pending interrupts

BlankFill:
cp R2,#MsgBufRP+10h ; Check for end of buffer
jr uge,MsgBufFull ; If at end, jump
ld @R2,'# ' ; else, save a ' ' character
inc R2 ; Move character pointer
jr BlankFill ; jump to continue filling

MsgBufFull:
call DisplayMsg ; Write message to the LCD module
jr Msg_Loop ; Return to wait for more

**************************************************************************
* Function Name: LCM_Init
*
* Returns: Nothing
* Entry values: Register Pointer set to AppRP
* Description: This routine prepares the LCD Module for message display.
* Notes: This routine has 2 entry points. The first one (LCM_Init) perform intialization and reset, the second one (LCM_Reset) performs reset only.
**************************************************************************

LCM_Init:
and P0,#~P0_RS_Hi ; Force RS control line low
ld P2,#P2_Init ; Zero the data bus
or P0,#P0_E_Hi ; Force E control line hi
and P0,#~P0_E_Hi ; Force E control line low
call LCM_Wait1 ; Wait for approx 20ms
ld P2,#P2_LCM8bits ; Function set instruction - 8bits
or P0,#P0_E_Hi ; Force E control line hi
and P0,#~P0_E_Hi ; Force E control line low
call LCM_Wait1 ; Wait for approx 20ms
or P0,#P0_E_Hi ; Force E control line hi
and P0,#~P0_E_Hi ; Force E control line low
ld P2,#P2_DOnNoC ; Turn on display - no cursor
or P0,#P0_E_Hi ; Force E control line hi
and P0,#~P0_E_Hi ; Force E control line low
call LCM_Wait1 ; Wait for approx 20ms
ld P2,#P2_AIncNoS ; Address increment, no shift
or P0,#P0_E_Hi ; Force E control line hi
and P0,#~P0_E_Hi ; Force E control line low
call LCM_Wait1 ; Wait for approx 20ms

LCM_Reset:
ld P2,#P2_DspClr ; Clear the display
or P0,#P0_E_Hi ; Force E control line hi
and P0,#~P0_E_Hi ; Force E control line low
call LCM_Wait1 ; Wait for approx 20ms
ld P2,#P2_CurHome ; Send the cursor home
or P0,#P0_E_Hi ; Force E control line hi
and P0,#~P0_E_Hi ; Force E control line low
call LCM_Wait1 ; Wait for approx 20ms
ld P2,#P2_CGRam ; Set the CG Ram
or P0,#P0_E_Hi ; Force E control line hi
and P0,#~P0_E_Hi ; Force E control line low
call LCM_Wait2 ; Wait for approx 80us
ld P2,#P2_DDRam1 ; Set the DD Ram
or P0,#P0_E_Hi ; Force E control line hi
and P0,#~P0_E_Hi ; Force E control line low
call LCM_Wait2 ; Wait for approx 80us
ret

**************************************************************************
* Function Name: DisplayMsg
*
* Returns: Nothing
* Entry values: Register Pointer set to AppRP
* Description: This routine writes the contents of the message buffer into the LCD module for display.
* Notes:
**************************************************************************
DisplayMsg:
call LCM_Reset ; Reset LCM for line1
ld R13,#MsgBufRP ; Initialize line1 char pointer
call LCM_Write ; Write line1 data to the LCM
and P0,#~P0_RS_Hi ; Force RS control line to 0
ld P2,#P2_DDRam2 ; Load line2 starting address
or P0,#P0_E_Hi ; Force E control line to 1
and P0,#~P0_E_Hi ; Force E control line to 0
call LCM_Wait2 ; Wait for approx 80us
ld R13,#MsgBufRP+8 ; Initialize line2 char pointer
call LCM_Write ; Write line2 data to the LCM
ret

**************************************************************************
* Function Name: LCM_Write
*
* Returns: Nothing
* Entry values: Register Pointer set to AppRP
* R13 loaded with addr of 1st character of the line
* Description: This routine loads the 8 characters of the display line into the module.
* Notes:
**************************************************************************
LCM_Write:
or P0,#P0_RS_Hi ; Force RS control line to 1
ld R12,#008h ; Initialize loop counter
LCM_WrLoop:
ld P2,@R13 ; Place character on data bus
or P0,#P0_E_Hi ; Force E control line to 1
and P0,#~P0_E_Hi ; Force E control line to 0
call LCM_Wait2 ; Wait for LCM processing
inc R13 ; Move the character pointer
djnz R12,LCM_WrLoop ; Check for end of loop
and P0,#~P0_RS_Hi ; Force RS control line to 0
ret
**Function Name:** LCM_Wait1

* Returns: Nothing
* Entry values: None
* Description: This routine creates a delay of approximately 20ms
* Notes:

```asm
LCM_Wait1:
  ld R14,#Wt1MCnst ; Initialize upper byte of count
Wait1_Lp2:
  ld R15,#Wt1LCnst ; Initialize lower byte of count
Wait1_Lp1:
  djnz R15,Wait1_Lp1 ; Decrement ls byte count till 0
  djnz R14,Wait1_Lp2 ; Decrement ms byte count till 0
ret
```

**Function Name:** LCM_Wait2

* Returns: Nothing
* Entry values: None
* Description: This routine creates a delay of approximately 80us
* Notes:

```asm
LCM_Wait2:
  ld R14,#Wt2Const ; Initialize count value
Wait2_Lp:
  djnz R14,Wait2_Lp ; Decrement count till 0
ret
```

**Function Name:** MsgBufLoad

* Returns: Message buffer loaded with 16 character message
* Entry values: Register Pointer set to AppRP
  * R12 contains msbyte of message start addr
  * R13 contains lsbyte of message start addr
* Description: This routine loads the message buffer with an internally generated 16 byte message.
* Notes:

```asm
MsgBufLoad:
  ld R2,#MsgBufRP ; Initialize Comm Buffer Location
  ld R14,#010h ; Initialize loop counter
MsgBufLoop:
  ldci @R2,@RR12 ; Load Character into buffer
  djnz R14,MsgBufLoop ; Test for end of message
  dec R2
ret
```
IRQ3 Interrupt Service

This routine performs the RS232 input function at 4800 baud.
Format: 8 bits data – LSB first, no parity, 1 start, 1 stop
Note: All bits inverted since there's no inverting input buffer
Procedure: Rising Start bit edge causes IRQ3 service.
After a half bittime input is sampled again to validate
the Start bit. Then IRQ5 is enabled and T0 is setup for bittime
delay in continous mode.

**************************************************************************
IRQ3:
; Setup half bit-time and wait to validate start bit
ld R11,#024h ; Half bit-time =~ 104us
StrtBitWait:
djnz R11,StrtBitWait ; Wait for center of start bit
tm P3,#P3_RxBit ; Take Sample on P32: RX=0?
Jr nz,StrtValid ; If nonzero, Start bit is valid!
Iret ; else, ignore it

StrtValid:
ld TMR,#TMR_RxEnab ; Load & enable T0
ld IMR,#IMR_RxLoop ; Enable IRQ4 (T0) only
ei ; Reenable interrupts
clr R3 ; Clear the word buffer
ld R1,#008h ; Load the number of data bits
Rcv_Loop:
nop ; Clear pipeline
halt ; Wait to sample data
tm P3,#P3_RxBit ; RX=0?
jr z,Rcvd0 ; If zero, then jump
rcf ; else, reset carry (data=0)
jr Rcvd1 ;
Rcvd0:
scf ; Set carry (data=1)
Rcvd1:
rrc R3 ; Carry into MSB, LSB into carry
dec R1 ; Decrement bit counter
Jr nz,Rcv_Loop ; If not 0, jump to continue loop
nop ; else, wait for stop bit
halt ;
tm P3,#P3_RxBit ; Test Stop Bit
jr z,FrameOK ; If stop bit=0 – OK, jump
or R0,#RxFrameErr ; else, set frame error flag
jr RxExit ;

FrameOK:
or R0,#RxWrdAvail ; Set data available flag
RxExit:
di
ld IRQ,#IRQ_Init ; Clear interrupts
ld TMR,#TMR_Init ; Disable timers
ld IMR,#IMR_RxEnab ; Reenable IRQ3 only
iret
**********************************************************************
* Timer 0 Interrupt Service  
* This timer is used to create the bit time for the 4800 baud xfer.  
* There is no processing involved.  
**********************************************************************
TMR0:  
   iret

**********************************************************************
* Unused Interrupt Service  
**********************************************************************

; Empty IRQ's defined earlier so that the processor will have a 16 bit  
; address in memory to jump to and return from in the case of a stray  
; or glitch interrupt.

IRQ0:  
IRQ1:  
IRQ2:  
TMR1:  
   iret

**********************************************************************
* System messages  
**********************************************************************

MsgBlank:  .ASCII "               " ; Blank display message  
MsgInit:   .ASCII ">Ready          " ; Initialization message  
MsgFrErr:  .ASCII ">Frame Error!!! " ; Frame error message

**********************************************************************

; End of main program.

End

Test Procedure

Equipment Used
Testing the application requires the following items:

• Target application board built according to the schematic in the Appendix  
• 5V, 1A bench supply (for application power)
Windows 95/98/NT-based PC with ZDS 2.11 or higher installed
Z86CCP01ZEM (CCP Emulator)
Z86CCP00ZAC (Emulator Accessory Pack)
8V @ 0.8 A power supply (for emulator power)

A DOS or Windows terminal program, such as HyperTerminal, running on the COM port of your choice, is also required to exercise the application's RS-232 interface.

General Test Setup and Execution
Exercise the application by either burning an OTP (stand-alone) or running the application from the emulator.

If using an emulator, at least two free serial ports are required on your PC. One is for the emulator and the other for the application's RS-232 interface. Follow the instructions for Assembling the Application Code as described in the previous section.

To send messages to the application, configure the terminal program as follows:
- Direct connection to the com port where the application is connected
- 4800 baud, no parity, 8 data bits, 1 stop bit, and 1start bit
- No flow control
- Echo locally typed characters to the computer screen.

Test Results
When power is applied to the application, the application immediately responds with the >Ready message. To read the display, it may be necessary to adjust the LCD contrast by changing the setting of potentiometer R3. Also, press switch SW1 to demonstrate the module backlight.

Using the terminal program, demonstrate the display of messages entered from the keyboard. Note that the standard ASCII character set (7-bit) is supported. Messages containing ASCII characters greater than 07Fh result in the display of Asian or symbol characters. A message may contain from 0 to 16 characters and is terminated by the Enter key. If more than 16 characters are entered as a message, they display in sets of 16 characters until the Enter terminates the message.

Note: If the HyperTerminal program is used as the terminal software, be aware that some versions contain a bug. If any character is pressed repeatedly, the third occurrence and then every other occurrence of the character is corrupted. The
corruption is manifested as having the most significant bit of the character set to 1 (for example, \texttt{031h} is corrupted into \texttt{0B1h}). This value includes the \texttt{Enter} key.

**References**

2. The Z8 Application Note HandbookDB97Z8X01, ZiLOG Corporation, 1996.

**Appendix**

**Figure 6. LCM Interface Schematic**