



Application Note

Interfacing a Graphic LCD with a Z8 Encore! XP[®] 64K Series Flash MCU

AN019602–1207

Abstract

This application note discusses an interface between Zilog's Z8 Encore! XP[®] 64K Series Flash MCU and a Crystalfontz (CFAG240128D-FMI-T) Graphics LCD Module. The source code file associated with this application note, AN0196-SC01.zip, provides device-driver APIs to display text and monochrome BMP images on the Crystalfontz (CFAG240128D-FMI-T) Graphics LCD. This source code file also provides a utility to convert the BMP images to HEX values, which in turn facilitates the display of BMP images.

► **Note:** *The source code file associated with this application note, AN0196-SC01.zip, is available for download at www.zilog.com.*

Family Overview

Z8 Encore! XP[®] Flash Microcontrollers

Zilog's Z8 Encore! XP[®] products are based on new Zilog's eZ8 CPU and introduce Flash memory to Zilog's extensive line of 8-bit microcontrollers. Flash memory in-circuit programming capability allows faster development time and program changes in the field. The high-performance register-to-register based architecture of the eZ8 core maintains backward compatibility with Zilog's popular Z8[®] MCU. Z8 Encore! XP MCUs combine a 20 MHz core with Flash memory, linear-register SRAM, and an extensive array of on-chip peripherals. These peripherals make the Z8 Encore! XP MCU suitable for various applications including motor control, security systems, home appliances, personal electronic devices, and sensors.

Discussion

This section describes the general construction theory of a Graphics LCD, the specific details of the Sanyo LC7981 controller, and the Crystalfontz CFAG240128D-FMI-T Display Module.

A General-Purpose Graphics LCD Module

A general-purpose Graphics LCD module consists of the following:

- A display substrate device
- An LCD driver controller
- Row and segment drivers
- An onboard bias power supply
- An EL or CCFL backlight
- An optional RAM buffer

Figure 1 displays the arrangement of an LCD module. The components of this Graphics LCD Module are described below:

- Graphics Display Substrate Panel
- Graphics Display Driver Controller
- Row and Segment Drivers
- Onboard Bias Power Supply
- Graphics RAM Buffer
- Backlight
- Driver Controller and Graphics LCD Module
- Registers and Memory Space
- User RAM Area
- Busy Flag

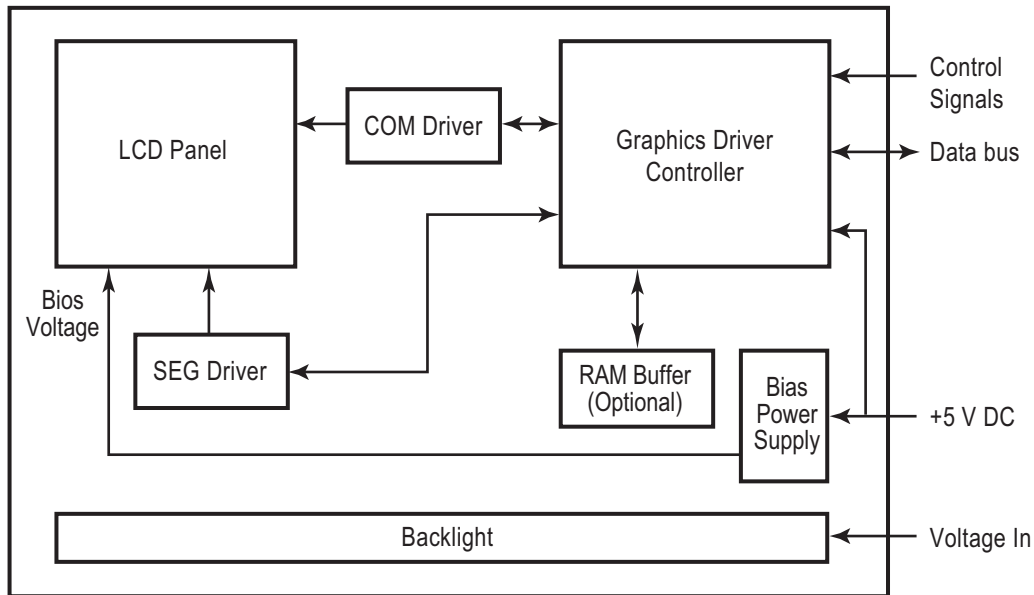


Figure 1. Construction of a Graphics LCD Module

Graphics Display Substrate Panel

The graphics display panel substrate defines the resolution of the LCD. For example, a panel featuring 240 horizontal dots and 128 vertical dots (the maximum size of an image) yields a resolution of $240 \times 128 = 30,720$ total dots. The panel consists of liquid crystals, which are turned ON (enabled) to display a dot (also called a pixel).

This combination of dots (some of them switched ON and others switched OFF) creates the effect of a visual image.

- **Note:** *As graphic LCDs are of transmissive type, the angle of light reflected from the enabled dots determines the viewability of the image. Usually, the vertical viewing angle for Graphic LCDs is more than the horizontal angle. This aspect must be taken into account when considering placement of an LCD in an application.*

Figure 2 displays the positioning of a character on a 240 x 128 LCD panel. The horizontal pitch (Hp) and vertical pitch (Vp) define the size of a character and the total number of characters that can be placed on the screen.

In the graphic panel (see Figure 2), the Hp and Vp values are each set to 8 pixels. Thus, for a 240 x 128 display, the total number of characters in a line is $240 \div 8$, for a total of 30 characters. Additionally, a total number of $128 \div 8 = 16$ rows can be displayed, for a total display of $30 \times 16 = 480$ characters. This calculation can be extended to create a custom character map or an image map on the LCD panel.

Display duty is used to define the scanning frequency of the LCD panel and is the inverse of the number of rows in the panel. In the above example, the display duty is $1/128$, which must be configured at the time the LCD driver controller is initialized.

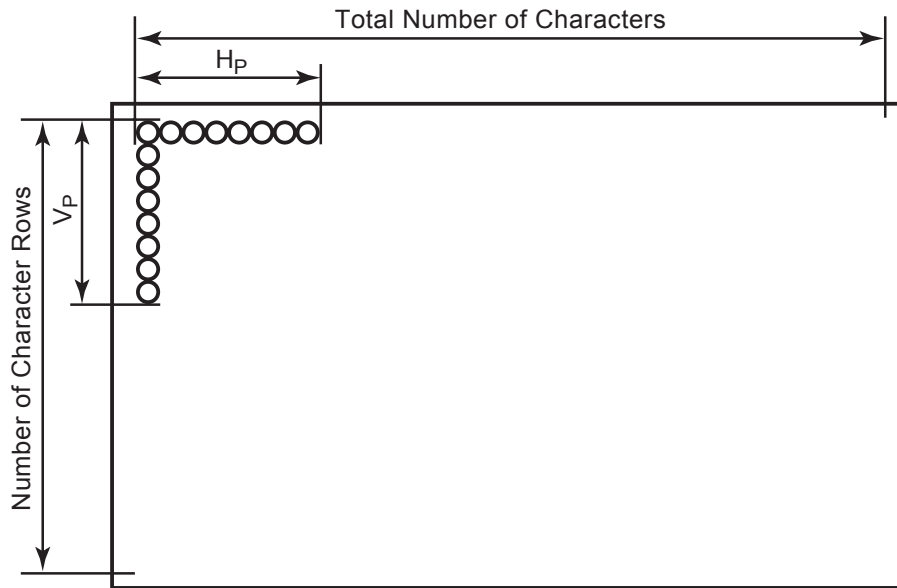


Figure 2. Character Positioning on an LCD Panel

Graphics Display Driver Controller

The graphics display driver is the main controller driving the LCD panel. It employs various drive voltages at different frequencies as specified by the external microcontroller.

In effect, the display driver acts as an interface between the LCD panel and the microcontroller, interpreting various commands and accordingly illuminating the pixels on the display. To support its operation, various other peripheral components are also required, like row/segment driver ICs and memory chips.

Row and Segment Drivers

Row driver ICs, also known as common drivers, are required to drive the display panel. These drivers scan and energize the horizontal rows, one at a time. Segment driver ICs energize the vertical columns in a sequential manner. The interaction point of the horizontal scan and the vertical scan results in an illuminated pixel.

Onboard Bias Power Supply

The liquid crystal display panel requires varying levels of negative voltages at specified frequencies to operate. A bias power supply and a negative voltage generator is built on board to supply this AC voltage.

Graphics RAM Buffer

To draw an image on the LCD panel, the graphics driver controller accepts input from the external microcontroller and illuminates the specified pixels. To accelerate this process, an onboard memory buffer is helpful for collecting all the data, then scanning the display to draw the image.

Backlight

Graphics display panels are of transmissive type and depend on ambient light conditions to be viewed properly, as they reflect light off the pixels. A backlight is used to compensate for variations in ambient light conditions and to provide display consistency.

Backlights can be Electroluminescent (EL) or Common Cathode Fluorescent Display (CCFL) type. To operate, these two types of backlights require proper driving voltages from external inverters. The input voltage to a CCFL backlight is in the range of 200 V to 250 V AC and is provided using a DC-to-AC inverter.

Driver Controller and Graphics LCD Module

Figure 1 on page 2 displays the CrystalFontz CFAG240128D-FMI-T Module and includes the following features:

- 240x128-pixel STN-type graphic display
- Onboard Sanyo LC7981 graphics controller
- Three-number LC7940 segment driver for driving 240 pixels
- Two-number LC7942 common driver for driving 128 pixels
- Onboard 4 KB SRAM memory buffer
- Built-in negative voltage generator for bias supply
- Built-in temperature compensation circuit
- White CCFL backlight

Registers and Memory Space

The available registers and memory spaces of the LC7981 graphics driver controller are described below.

Character Generator ROM (CGROM)—The LC7981 graphics driver controller features a CGROM with capacity of 7360 bits containing a total of 192 characters. These ready-made font characters are ASCII-compatible and can be utilized by the user program to display the character strings.

Registers—The LC7981 controller features five registers for setting up the display and to send and receive data bytes, as follows:

- Instruction register
- Data Input register
- Data Output register
- Dot register
- Mode Control register

The Instruction register is used to configure the type of display and to provide operational instructions. The Data Input and Output registers are meant for writing and reading pixel data from the module.

The Dot register is used to provide information such as the character pitch of fonts, the number of vertical dots, and more. This register is not directly accessible to the user program; it receives data from the Data Input register. The Mode Control register is used to change the status of the display and to turn the cursor ON and OFF.

User RAM Area

The CFAG240128 Graphics Module contains a 4 KB SRAM chip (6264) to store data bytes to be displayed. This memory buffer is available via the Data Input register. If additional memory buffer is required, the external microcontroller must be equipped with its buffer space, as the display module does not provide additional RAM expansion. The RAM map of a module describes its addressable locations where data can be read from or written to by the microcontroller. This data can be stored in the optional RAM buffer or directly displayed on the graphics panel.

Busy Flag

While sending instructions for reading or writing to the LC7981 driver controller, you must poll the busy flag to check the status of the controller. A value of 1 indicates that the LC7981 is busy. The user program must wait until the graphics driver controller is available. A cleared busy flag (value = 0) indicates that the controller is ready to receive the next instruction.

Alternatively, you can incorporate a sufficient delay between two instructions. The amount of delay must be determined by trial and error.

Developing the Application with Z8 Encore! XP MCU

The hardware for this application consists of the Crystalfontz CFAG240128D-FMI-T graphics display module and the Z8F642 MCU Development Kit (Z8F64200100KIT) for the Z8F6423 microcontroller. Figure 3 displays a block diagram of the arrangement. For a detailed schematic diagram, see Appendix A—Schematics on page 10.

The software consists of routines to initialize the display module and to read or write data to it. For the LC7981 driver controller and the Z8F6423 microcontroller, these routines are detailed in their respective flowcharts (see Appendix B—Flowcharts on page 11).

In the implementation, Port F is used as the data bus and some Port G pins are used as the control

bus. However, the application code is flexible which allows you to choose any available port for the data and control buses by modifying the following statements in the GLCD_API.h header file:

```
#define DATAaddr PFADDR
// Define data port as Port F
#define DATActl PFCTL
#define DATAout PFOUT
#define SIGaddr PGADDR
// Define control signal port as Port G
#define SIGctl PGCTL
#define SIGout PGOUT
```

Additionally, the timing of the control signals that read and write data must be clearly interpreted and followed for proper operation of the software driver. Control signal timing is specific to the type of graphics controller driver used in an LCD module. For the LC7981 driver controller timing values are provided in a data sheet (for a list of references, see References on page 9).

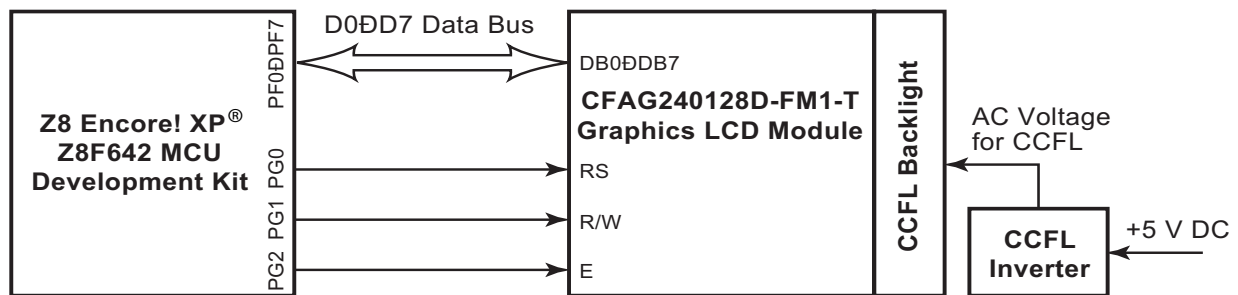


Figure 3. Hardware Connection Between Graphics LCD Module and Z8F642 Development Kit

The source code file associated with this Application Note, AN0196-SC01.zip, provides APIs to be used with the Graphics LCD. For a list of the Graphic LCD API, see Appendix C—Graphics LCD Driver APIs on page 12.

Displaying Bitmap Files on a Graphic LCD

This section discusses the process of generating BMP files using Microsoft Paint, converting BMP files to an array of HEX data files, and integrating the HEX data files to the Z8 Encore! XP project.

Generating BMP Files using Microsoft Paint

The bitmap image (see [Figure 4](#)) is an example to demonstrate the display on the 240 x 128 Graphic LCD. Bitmap images can be created using the **Microsoft Paint** application.

Follow the steps below to create a bitmap image:

1. Open the **Microsoft Paint** application and click on the **Image** menu.
2. From the **Image** menu, select **Attributes**. In the **Attributes** window, enter 240 for the **Width** and 128 for the **Height**.
3. Click **Pixels** radio button under the **Units** field of the **Attribute** window.
4. Click **Black and White** radio button under the **Color** field of the **Attribute** window.
5. Click **OK** to select the above attributes.
6. Create the required image in the area provided for creating images.
7. Click **Image** menu and select the **Flip/Rotate** option. In the **Flip and Rotate** window, click the **Flip Vertical** radio button and then click **OK**.
8. Save the file as a Monochrome BMP file.

The width of the image corresponds to the number of pixels in a row and the height of the image corresponds to the number of pixels in a column. The number of pixels must be set within the display area of the graphic LCD.

The number of pixels must not exceed the dimensions 240 X 128. Moreover, the number of pixels in a row must be divisible by 8 as the display on every row is done byte wise, and one byte of data corresponds to eight pixels. [Figure 4](#) displays a BMP image created using the above steps.



Figure 4. Snapshot of a BMP Image Displayed on a Graphic LCD

After implementing step 7 and step 8, the image displayed by [Figure 5](#) is inverted and saved as `Zilog.bmp`. [Figure 5](#) displays this inverted image.



Figure 5. Snapshot of an Inverted BMP Image Displayed on a Graphic LCD

Converting the BMP Image to HEX Value

To display a given BMP image on a graphic LCD, it is necessary to convert the BMP image to a stream of HEX values. A utility file, `Z_bmp2ohex.exe`, enables you to convert the BMP image to a stream of HEX values, is provided in the source code files associated with this application note. This is a command line utility that reads data from the BMP file. This removes the BMP header (which is 62 bytes) and stores the data into a two dimensional array of constant characters. This utility also creates a file, `filename_bmp.inc` and stores the hex data in that file.

The `Z_bmp_to_hex.exe` file supports the following types of command line arguments:

1. `Z_bmp_to_hex <filename.ext>`

Only `filename.bmp` is entered as the command line argument. The output file is created in the working directory itself.

2. `Z_bmp_to_hex <output directory> <filename.ext>`

The first command line argument is the directory in which the output file has to be stored. The second command line argument is `filename.bmp`.

Follow the steps below to convert the BMP image to HEX values:

1. Download the compressed project file (zip file); extract the `Z_bmp_to_hex.exe` file from the project file, and copy the same to an appropriate directory.
2. Copy the BMP files created previously, to the same directory which contains the `Z_bmp_to_hex.exe` utility file.
3. Go to the command prompt and select the path, where the utility is stored. Then execute the `Z_bmp_to_hex` utility by providing appropriate arguments as displayed in [Figure 6](#).

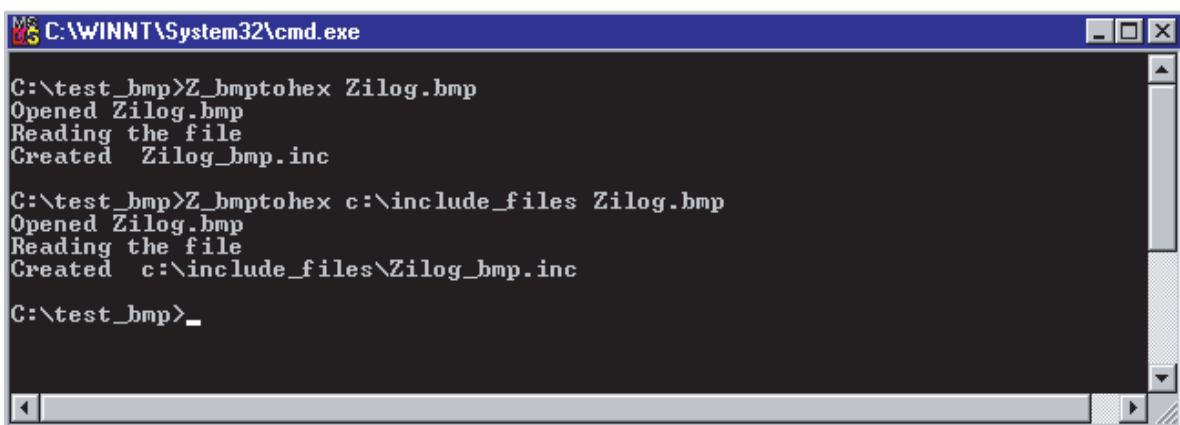
[Figure 6](#) displays the `Z_bmp_to_hex` file utility and the `Zilog.bmp` files stored in the path

`C:\test_temp`. In the first case, the output file `Zilog.bmp.inc` file is stored in the working directory. In the second case, the output file is stored in the `include_files` directory present in the C drive.

Integrating BMP HEX Data to Z8 Encore! XP® Project

The hex values representing the BMP images must be stored in the Flash memory of the Z8 Encore! XP microcontroller. Follow the steps below to store hex values in Flash memory:

1. Copy the `Zilog.bmp.inc` file to the `GLCD_zap.pro` working directory.
2. Open the `GLCD_test.c` file, and include the `Zilog.bmp.inc` statement (`#include Zilog.bmp.inc`) in the beginning of the file.
3. Then add the data type declaration `extern const unsigned char Zilog_bmp_data [128] [30]` to the `GLCD_test.c` file. Since the content of the `Zilog.bmp.inc` file is an array (`const unsigned char Zilog_bmp_data [128] [30]`), it is necessary to add the data type declaration in the `GLCD_test.c` file.
4. The `disply_bmp` API is provided to display the contents of the above two-dimensional array on the graphic LCD.



```

C:\WINNT\System32\cmd.exe
C:\test_bmp>Z_bmp_to_hex Zilog.bmp
Opened Zilog.bmp
Reading the file
Created Zilog.bmp.inc

C:\test_bmp>Z_bmp_to_hex c:\include_files Zilog.bmp
Opened Zilog.bmp
Reading the file
Created c:\include_files\Zilog.bmp.inc

C:\test_bmp>_

```

Figure 6. A Snap Shot of the Command Prompt Indicating the Appropriate Arguments

Testing

The complete source code for the Graphics LCD Driver APIs and the `z_bmptohex.exe` utility, developed for the application described in this document has been tested and demonstrated, and is available for download at www.zilog.com in the file `AN0196-SC01.zip`. The testing of the driver APIs is performed using the hardware described earlier and running the sample application on the Z8F642 Development Kit.

Equipments Used

The following equipment are used for testing:

- Z8F642X Development Kit (Z8F64200100KIT)
- 240x128 Graphics LCD Module from Crystalfontz Inc.(CFAG240128D-FMI-T).

Procedure

Follow the steps below to test the graphics LCD software:

1. Download and extract the `AN0196-SC01.zip` file.
2. Open the `Project Source Code` folder within the `AN0196-SC01.zip` file (`AN0196-SC01.zip` also contains the utility `z_bmptohex.exe` which converts BMP images to HEX values).
3. Setup the Z8 Encore! XP[®] Z8F642 MCU Development Kit and the Graphics LCD Module as per the schematics provided in [Appendix A—Schematics](#) on page 10.
4. Open the project file `GLCD_zap.pro` in ZDS II-IDE for Z8 Encore! XP. If there is any change in the port assignment for interfacing the Graphics LCD Module, as mentioned in [Developing the Application with Z8 Encore! XP MCU](#) on page 5.

5. Build the project using ZDS II-IDE. All the files in the `AN0196-SC01.zip` file are used for testing.
6. Flash the executable binary project into the Z8 Encore! XP Z8F642 MCU Development Kit using ZDS II-IDE.
7. Reset the microcontroller. Initially, the LCD is set to the character mode due to which characters are displayed on the LCD. Then, the LCD changes to the graphics mode in which lines and rectangular images are displayed.

The displaying of the three BMP images at fixed intervals also happens in a similar manner. The display in graphic mode continues and the same pictures are repeatedly displayed in a loop.

Summary

This application note describes a method to interfacing Crystalfontz (CFAG240128D-FMI-T) Graphics LCD Module with Z8F642x microcontroller. The APIs provided with this application note can be used directly. The device driver APIs and the `z_bmptohex.exe` utility make it convenient for the BMP images to be displayed on the graphics LCD, therefore reducing the product development time.

References

The documents referenced in this application note are listed below:

- Sanyo LC7981—Sanyo Data Sheet: http://service.semic.sanyo.co.jp/semi/ds_pdf_e/LC7981.pdf
- Graphics LCD Module—CFAG240128D Series Graphic LCD Modules: <http://www.crystalfontz.com/products/240128d/index.html#CFAG240128DFMIT>

Other related documents are listed below:

- eZ8 CPU User Manual (UM0128)
- Z8 Encore! XP[®] 64K Series Flash Microcontrollers Product Specification (PS0199)
- Z8 Encore! XP[®] 64K Series Flash Microcontrollers Development Kit (UM0151)
- Z8 Encore! XP[®] Development Kit Quick Start Guide (QS0028)
- Zilog Developer Studio II—Z8 Encore![®] User Manual (UM0130)

Appendix A—Schematics

Figure 7 displays layout of the Graphics LCD Module and the Z8F6423 MCU.

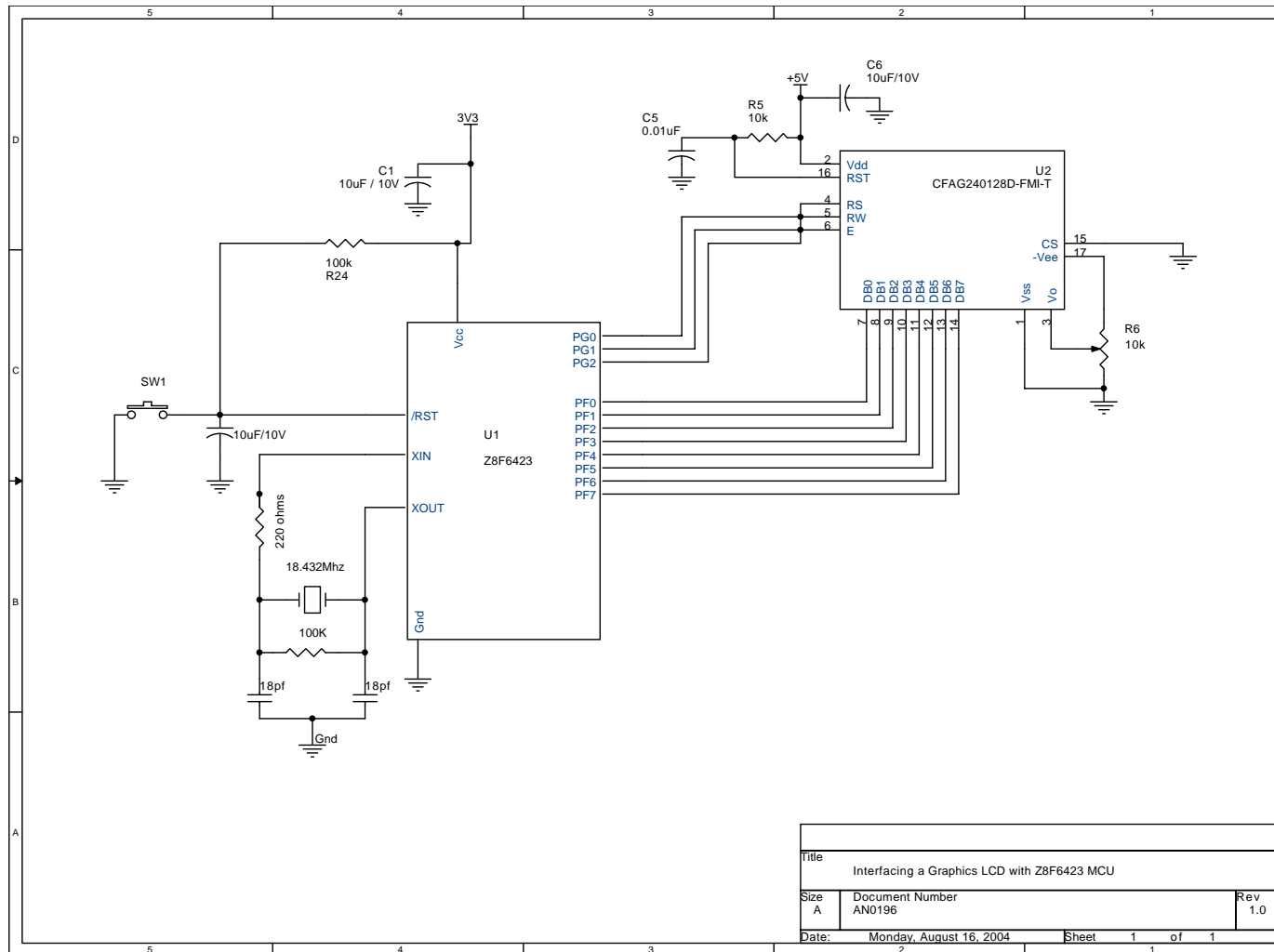


Figure 7. Schematic Diagram of Graphics LCD Module and Z8F6423 MCU

Appendix B—Flowcharts

Figure 8 displays the flow for setting up and initializing the LC7981 graphics controller on the CFAG240128D-FMI-T module in conjunction with the Z8F6423 MCU.

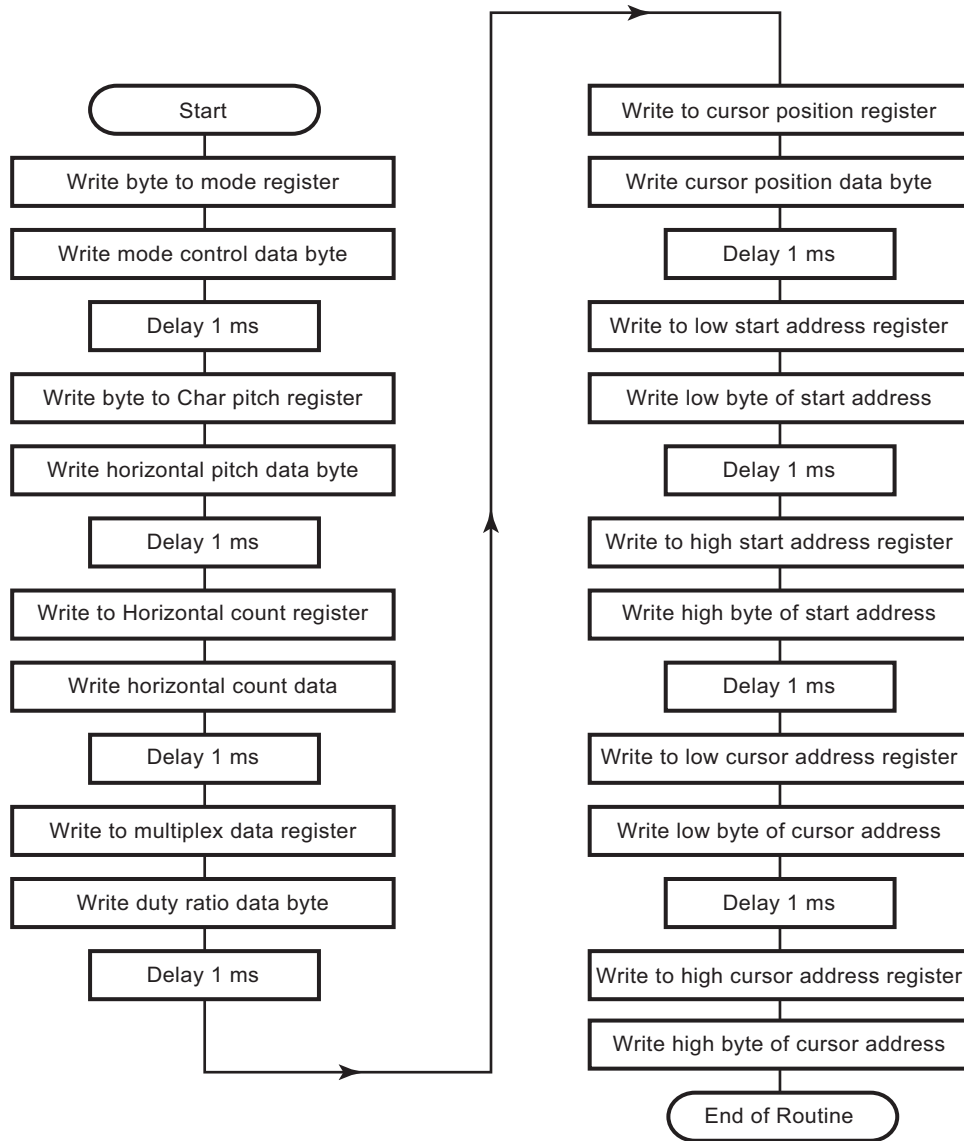


Figure 8. Initializing the HD44780 Controller on a 16x2 LCD Module

Appendix C—Graphics LCD Driver APIs

The software implementation of the Graphics LCD driver using the Z8F6423 microcontroller involves the development of driver APIs to initialize, read, and write to the display module. [Table 1](#) list the Graphics LCD Driver APIs for quick reference.

Table 1. File Manipulation Routines

ZInit_LC7981	Sets up and initializes the graphic driver controller.
ZLCDon	Creates a file.
ZLCDoff	Turns off the module.
ZLCDhome	Sets the cursor to the home position.
ZLCDprints	Prints a character string.
ZLCDprintch	Prints a single character.
ZDrawlines	Draws slanting lines.
ZDrawrectangle	Draws a rectangle.
ZPixelon	Illuminates a specified pixel.
ZPixeloff	Turns off a specified pixel.
ZSetcursor	Places the cursor at a specified location.
ZClearchar	Clears the screen characters.
ZCleargraf	Clears the LCD panel.
ZBlinkon	Blinks the cursor at its current location.
ZBlinkoff	Turns off the cursor's blink mode.
Zdisply_bmp	Displays BMP images on the graphic LCD.
Zswap_bits	Swaps bits.

ZInit_LC7981

Synopsis

Initialization

Description

The ZInit_LC7981 API sets up and initializes the Graphic driver controller LC7981 for proper operation of the LCD module. This API must be called on microcontroller powerup, before calling other APIs.

Argument(s)

Specify Character or Graphics mode

Return Value(s)

Success or failure of initialization

Example

```
ZInit_LC7981(0x01);      // Initialize in Character mode
ZInit_LC7981(0x02);      // Initialize in Graphics mode
```

The flowchart in [Appendix B—Flowcharts](#) on page 11 shows the initialization process of the graphics driver controller for the CFAG240128D-FMI-T LCD module.

ZLCDOn

Synopsis

Graphics display On

Description

The ZLCDOn API turns on the LCD module for display. In this state, it consumes maximum power for operation.

Argument(s)

None

Return Value(s)

None

Example

```
ZLCDOn ( );
```


ZLCDoff

Synopsis

Graphics display Off

Description

The `ZLCDoff` API turns off the module. The module conserves power in this mode.

Argument(s)

None

Return Value(s)

None

Example

```
ZLCDoff ();
```

ZLCDhome

Synopsis

Home Cursor

Description

The ZLCDhome API brings the cursor position to the first location of the display.

Argument(s)

None

Return Value(s)

None

Example

```
ZLCDhome ( ) ;
```

ZLCDprints

Synopsis

Prints a string

Description

The ZLCDprints API prints a character string in a specified location.

Argument(s)

X and Y coordinates of the location and the pointer to the string.

X is the row number and Y is the column position on the display screen.

Return Value(s)

None

Example

```
static unsigned char mystring[50] = "This is a character
string";
ZLCDprints(2,5,mystring); // Print 'mystring' starting
                          // at 5th column of 2nd row
```

ZLCDprintch

Synopsis

Prints a character

Description

The ZLCDprintch API prints a single character in a specified location.

Argument(s)

X and Y coordinates of the location and the ASCII character.

X is the row number and Y is the column position on the display screen.

Return Value(s)

None

Example

```
ZLCDprintch(2,5,'Z');    // Print 'Z' at 5th column, 2nd row
```

ZDrawlines

Synopsis

Draws straight lines

Description

The `ZDrawlines` API draws slanting lines on the screen.

Argument(s)

None

Return Value(s)

None

Example

```
ZDrawlines();
```

ZDrawrectangle

Synopsis

Draws a rectangle

Description

The ZDrawrectangle API draws a rectangle with specified length and width at a specified location on the screen. All dimensions are in pixels.

Argument(s)

X and Y coordinates of the top left corner of the rectangle,

Length and breadth of the rectangle in pixels.

Return Value(s)

None

Example

```
ZDrawrectangle(51,108,24,24); // Draw a square around center  
                        // of screen with 24-pixel lengths
```


ZPixelon

Synopsis

Turns on a pixel

Description

The ZPixelon API illuminates a specified pixel on the screen.

Argument(s)

X and Y coordinates of the pixel.

X is the horizontal location and Y is the vertical location of a pixel on the display screen.

Return Value(s)

None

Example

```
ZPixelon(119,63); // Turn on the pixel at the center of the  
screen
```

ZPixeloff

Synopsis

Turns off a pixel

Description

The `ZPixeloff` API turns off a specified pixel on the screen.

Argument(s)

X and Y coordinates of the pixel.

X is the horizontal location and Y is the vertical location of the pixel on the display screen.

Return Value(s)

None

Example

```
ZPixeloff(119,63); // Turn off the pixel at the center of the  
screen
```

ZSetcursor

Synopsis

Place cursor

Description

The `ZSetcursor` API is used to place cursor at a specified location on the screen.

Argument(s)

X and Y coordinates of the cursor by row and column, respectively.

X is the row number and Y is the column location of the cursor on the display screen.

Return Value(s)

None

Example

```
ZSetcursor(2,5); // Place cursor at 5th column of 2nd row.
```

ZClearchar

Synopsis

Clears character screen

Description

The ZClearchar API clears all characters on the screen.

Argument(s)

None

Return Value(s)

None

Example

```
ZClearchar();
```

ZCleargraf

Synopsis

Clears LCD panel

Description

The ZCleargraf API clears all graphics on the screen.

Argument(s)

None

Return Value(s)

None

Example

```
ZCleargraf ();
```

ZBlinkon

Synopsis

Blinks cursor

Description

The ZBlinkon API is used to blink the cursor at its current location.

Argument(s)

None

Return Value(s)

None

Example

```
ZBlinkon();
```


ZBlinkoff

Synopsis

Turns off cursor blink mode

Description

The ZBlinkoff API is used to turn off the blink mode of the cursor.

Argument(s)

None

Return Value(s)

None

Example

```
ZBlinkoff();
```

Zdisply_bmp

Synopsis

Displays BMP images on Graphic LCD.

Description

The `Zdisply_bmp` API is used to display BMP images that are available as two-dimensional arrays.

Argument(s)

```
const unsigned char *array;  
unsigned char no_rows;  
unsigned char no_column;
```

Return Value(s)

None

Example

```
Zdisply_bmp (A, B, C);
```

where,

A is the name of the `Const unsigned char array` data generated during conversion of BMP image to hex data.

B is the number of rows in that array.

C is the number of columns in that array.

Zswap_bits

Synopsis

Swaps a byte.

Description

The `Zswap_bits` API is used to swap a byte (b0 to b7, b1 to b6, b2 to b5, and b3 to b4).

Argument(s)

unsigned `data_byte`;

Return Value(s)

unsigned char;

Example

```
Zswap_bits (data_byte);
```



Warning: DO NOT USE IN LIFE SUPPORT

LIFE SUPPORT POLICY

ZILOG'S PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS WITHOUT THE EXPRESS PRIOR WRITTEN APPROVAL OF THE PRESIDENT AND GENERAL COUNSEL OF ZILOG CORPORATION.

As used herein

Life support devices or systems are devices which (a) are intended for surgical implant into the body, or (b) support or sustain life and whose failure to perform when properly used in accordance with instructions for use provided in the labeling can be reasonably expected to result in a significant injury to the user. A critical component is any component in a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system or to affect its safety or effectiveness.

Document Disclaimer

©2007 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. The information contained within this document has been verified according to the general principles of electrical and mechanical engineering.

Z8, Z8 Encore!, and Z8 Encore! XP are registered trademarks of Zilog, Inc. All other product or service names are the property of their respective owners.