字符 LCD 数据手册 LCD V 1.60

Copyright © 2012 Cypress Semiconductor Corporation. All Rights Reserved.

资源	PSoC® 模块			API 存储器 （字节）		引脚 （每个外部 I/O）
	数字	模拟 CT	模拟 SC	闪存	RAM	
CY8C29/27/26/25/24/22/21xxx, CY8C23x33, CY7C603xx/64215, CYWUSB6953, CY8C20x34, CY8CLED02/04/08/16, CY8C21x45, CY8C22x45, CY8C28x45, CY8CPLC20, CY8CLED16P01, CY8C28xxx, CY8C21x12						
启用条形图	0	0	0	646	0	一个端口中共 7 个
禁用条形图	0	0	0	434	0	一个端口中的 7 个

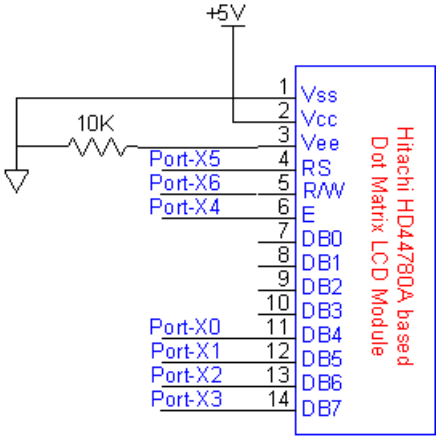
如需一个或多个使用此用户模块且完全配置的功能性示范项目，请转到 www.cypress.com/psocexampleprojects.

功能和概述

- 使用行业标准 Hitachi HD44780 LCD 显示屏驱动芯片协议
- 仅需要 7 个 I/O 引脚
- 提供用于显示 RAM 或 ROM 字符串的子程序
- 提供用于显示数字的子程序
- 提供用于显示水平和垂直条形图的子程序
- 使用单一 I/O 端口

字符 LCD 用户模块是一套子程序库，可向普通两行或四行 LCD 模块写入文本字符串和格式化的数字。 使用 LCD 模块的字符图形功能，可支持垂直和水平条形图。 此模块专为行业标准 Hitachi HD44780 两行 16 字符 LCD 显示屏驱动芯片开发，亦适用于很多其他四行显示屏。 此库使用 4 位接口模式以减少所需 I/O 引脚的数量。

Figure 1. LCD 连接到 PSoC 接口图



Note 对某些显示屏而言，将显示屏上的信号线 DB0~3 分别连接一个 10K 的下拉电阻到地，会有更好的效果

功能描述

LCD 用户模块使用单一 I/O 端口，连接至行业标准的 Hitachi HD44780A LCD 控制器。 此种类型的显示屏接口简单，由 8 个数据位、读 / 写 (R/W) 位、寄存器选择位 “RS” 和信号使能位 “E” 组成。 为减少所需的引脚数，特采用 4 位接口模式。 如下 LCD 至 PSoC 框图以及图表描述了 4 位接口连接方式。

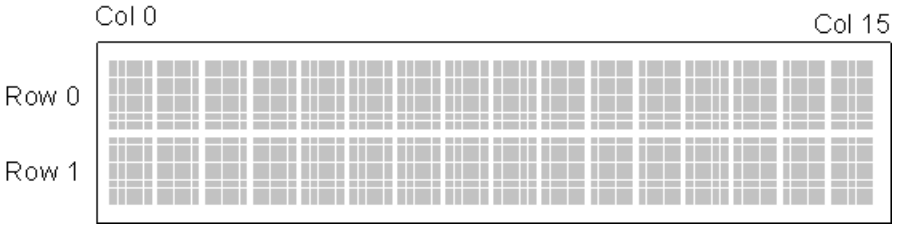
在某些显示屏上，DB0、DB1、DB2 和 DB3 可能需要通过 10K 电阻下拉至 V_{SS}。 将这些信号下拉以确保进入 4 位模式。

Table 1. LCD 至 PSoC 对应连接

PSoC 引脚	LCD 引脚	说明
Port-X0	DB4	数据位 0
Port-X1	DB5	数据位 1
Port-X2	DB6	数据位 2
Port-X3	DB7	数据位 3
Port-X4	E	启用 LCD
Port-X5	RS	寄存器选择
Port-X6	R/W	读 / 非写

光标定位函数可将光标置于任何位置。 对于两行 16 字符显示屏，左上角为位置 (0,0)，右下角为位置 (1,15)。 请参见下图。

Figure 2. 光标位置



低阶指令用于向显示屏的数据和控制寄存器写入数据。 请参阅 LCD 制造商数据手册，了解具体功能和字体信息。

放置

LCD 用户模块仅使用一个端口中的 7 个 I/O 引脚，而不使用任何数字或模拟模块。 不存在放置限制。 单一项目中可放置多个 LCD 模块。

参数和资源

LCDPort

选择一个 PSoC I/O 端口用于连接 LCD 显示屏。

Note 应使用影子寄存器控制 LCD 端口的 Px[7]，以实现其他目的。 请参考影子寄存器数据手册，了解更多详情。

条形图

选择是否启用条形图功能。 若禁用，则不会生成条形图代码，从而节省 ROM 空间。

应用程序编程接口

应用程序编程接口 (API) 子程序作为用户模块中的一部分，设计人员可以使用这些 API 实现更高级别的功能。 本节指定每个函数的接口，以及 “ 引用 ” 文件所提供的相关常量。

Note

在这里，如同所有用户模块 API 中的一样，A 和 X 寄存器的值可以通过调用 API 函数来更改。 函数调用前会自动保留 A 和 X 的值，以便于在调用后需要返回原始 A 和 X 的值， 选择此 “ 寄存器易失 ” 规则是为了提高效率，自 PSoC Designer 1.0 版起已强制使用此规则。 C 编译器自动遵循此要求。 汇编语言程序员也必须确保其代码遵守这一策略。 虽然一些用户模块 API 函数可以保留 A 和 X 不变，但是无法保证它们将来也会如此。

对于大型储存器模块驱动，保存 CUR_PP、IDX_PP、MVR_PP 以及 MVW_PP 寄存器中的所有值也是调用程序的职责。 尽管部分寄存器现在可能不可修改，但是在将来的新版本中可能会改变。

以下是用于 LCD 用户模块的 API 编程子程序：

基本字符 LCD 函数

LCD_Start

说明：

初始化 LCD 以使用多行 4 位接口。 此函数应当在调用任何其他 LCD 函数之前调用。

C 语言原型：

```
void  LCD_Start(void);
```

汇编：

```
lcall  LCD_Start
```

参数：

None

返回值：

None

副作用：

A 和 X 寄存器可以由此函数的本次执行或以后执行而被修改。 在大内存模式下 (CY8C29xxx)，所有 RAM 页面指针寄存器也会出现这种状况。 如果需要，调用函数负责通过调用 fastcall16 函数保留值。 当前，仅修改 CUR_PP 页面指针寄存器。

LCD_Init

说明：

初始化 LCD 以使用多行 4 位接口。 此函数应当在调用任何其他 LCD 函数之前调用。

C 语言原型：

```
void  LCD_Init(void);
```

汇编：

```
lcall  LCD_Init
```

参数：

None

返回值：

None

副作用：

A 和 X 寄存器可以在此函数的本次执行中或以后执行中被修改。 在大内存模式下 (CY8C29xxx)，所有 RAM 页面指针寄存器也会出现这种状况。 如果需要，调用函数负责通过调用 fastcall16 函数保留值。 当前，仅修改 CUR_PP 页面指针寄存器。

LCD_Position

说明：

将光标移动到参数指定的位置。 左上方字符为行 0，列 0。 对于两行 16 字符显示屏，右下方字符为行 1，列 15。

C 语言原型：

```
void  LCD_Position(BYTE bRow, BYTE bCol);
```

汇编：

```
mov   A,01h               ; Load Row
mov   X,02h               ; Load Column
lcall  LCD_Position
```

参数：

bRow: 放置光标的行号。 0 代表第一行。

bCol: 放置光标的列号。 0 代表第一 （最左边的）列。

返回值：

None

副作用：

A 和 X 寄存器可以由此函数的本次执行或以后执行而被修改。 在大内存模式下 (CY8C29xxx)，所有 RAM 页面指针寄存器也会出现这种状况。 如果需要，调用函数负责通过调用 fastcall16 函数保留值。 当前，仅修改 CUR_PP 页面指针寄存器。

字符串打印函数

LCD_PrString

说明：

将 RAM 中以 null 结尾的字符串打印至 LCD 中光标所在位置。

C 语言原型：

```
void  LCD_PrString(CHAR * sRamString);
```

汇编：

```
mov   A,>sRamString   ; Load MSB part of pointer to RAM-based null
                      ; terminated string.
mov   X,<sRamString   ; Load LSB part of pointer to RAM-based null
                      ; terminated string.
lcall  LCD_PrString    ; Call function to display string at current
                      ; LCD cursor position.
```

参数：

sRamString: 指向 RAM 中以 null 结尾的字符串的指针。

返回值：

None

副作用：

A 和 X 寄存器可以在此函数的本次执行中或以后执行中修改。 在大内存模式下 (CY8C29xxx)，所有 RAM 页面指针寄存器也会出现这种状况。 如果需要，调用函数负责通过调用 fastcall16 函数保留值。 目前修改了 CUR_PP 和 IDX_PP 页面指针寄存器。

LCD_PrCString

说明：

将 ROM 中以 null 结尾的字符串打印至 LCD 中光标所在位置。

C 语言原型：

```
void  LCD_PrCString(const char * sRomString);
```

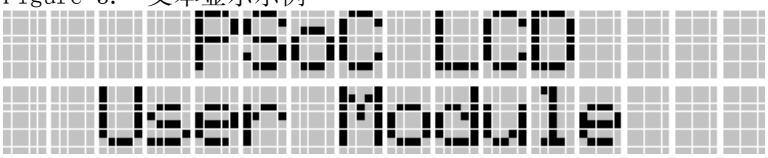
汇编：

```
mov   A,>sRomString   ; Load MSB part of pointer to ROM-based null
                      ; terminated string.
mov   X,<sRomString   ; Load LSB part of pointer to ROM-based null
                      ; terminated string.
lcall  LCD_PrCString   ; Call function to display string at current
                      ; LCD cursor position.
```

字符串打印代码示例：

```
char str[ ] = "User Module";     // Define "RAM" based string
LCD_Start();                     // Initialize LCD hardware
LCD_Position(0,4);              // Position cursor @ row 0, col 4
LCD_PrCString("PsoC LCD");       // Print a constant "ROM" string
LCD_Position(1,2);               // Position cursor @ row 1, col 2
LCD_PrString(str);               // Print "RAM" based string.
```

Figure 3. 文本显示示例



参数：

sRomString: 指向 ROM 中以 null 结尾的字符串的指针。

返回值：

None

副作用：

A 和 X 寄存器可以在此函数的本次执行中或以后执行中被修改。 在大内存模式下 (CY8C29xxx)，所有 RAM 页面指针寄存器也会出现这种状况。 如果需要，调用函数负责通过调用 fastcall16 函数保留值。 当前，仅修改 CUR_PP 页面指针寄存器。

数字打印函数

LCD_PrHexByte

说明：

在 LCD 光标的位置用双字符十六进制字符串的形式显示一个字节。

C 语言原型：

```
void  LCD_PrHexByte(BYTE bValue);
```

汇编：

```
mov   A, [bValue]          ; Load byte to be printed
lcall  LCD_PrHexByte        ; Call function
```

参数：

bValue： 要显示为双字符十六进制字符串的一个 8 位值。

返回值：

None

副作用：

A 和 X 寄存器可以由此函数的本次执行或以后执行而被修改。 在大内存模式下 (CY8C29xxx)，所有 RAM 页面指针寄存器也会出现这种状况。 如果需要，调用函数负责通过调用 fastcall16 函数保留值。 当前，仅修改 CUR_PP 页面指针寄存器。

LCD_PrHexInt

说明：

在 LCD 中光标的当前位置以四字符十六进制字符串的形式打印整数。

C 语言原型：

```
void  LCD_PrHexInt(INT iValue);
```

汇编：

```
mov   A, [iValue+1]    ; Load LSB byte to be printed
mov   X, [iValue]      ; Load MSB byte to be printed
lcall  LCD_PrHexInt     ; Call function
```

参数：

iValue： 要显示为四字符十六进制字符串的一个 16 位值。

返回值：

None

副作用：

A 和 X 寄存器可以由此函数的本次执行或以后执行而被修改。 在大内存模式下 (CY8C29xxx)，所有 RAM 页面指针寄存器也会出现这种状况。 如果需要，调用函数负责通过调用 fastcall16 函数保留值。 当前，仅修改 CUR_PP 页面指针寄存器。

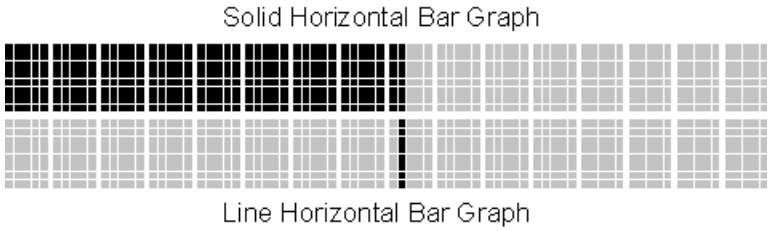
垂直条形图函数

每个显示屏字符由 5 个水平像素 乘以 8 个垂直像素组成。 水平条形图在单个字符中显示一组垂直线，每一条线由 1 个水平像素乘以 8 个垂直像素组成，即一个像素列。 每一个字符可显示 1 至 5 个垂直像素列，5 个像素列即可显示整个字符。

显示屏的左起第一个像素列编号为 1，最后一个像素列编号为 N * 5，其中 N 为字符数。 一个 16 字符显示屏可能有 80 个像素列，编号从 1 到 80。

实体条形图在一组指定的持续字符内显示 1 至 N 的像素列。 线性条形图只显示指定的像素列。 如下为这两种类型的水平条形图示例：

Figure 4. 条形图类型



LCD_InitBG

说明：

初始化 LCD 以显示指定类型的水平条形图。 此函数应当在调用 LCD_DrawBG() 之前调用。必须指定条形图的类型。 本函数不绘制条形图，但通过加载自定义字符 RAM 中的数据去显示指定类型条形图。 要在两种水平条形图类型之间进行转换，必须调用此子程序。

LCD_SOLID 和 LCD_LINE 被定义为输入常量。

C 语言原型：

```
void  LCD_InitBG(BYTE bBGType);
```

汇编：

```
mov   A, LCD_SOLID_BG
lcall  LCD_InitBG
```

参数：

BYTE bBGType： 条形图的类型，可为以下两类之一：

```
LCD_SOLID_BG
LCD_LINE_BG
```

返回值：

None

副作用：

A 和 X 寄存器可以由此函数的本次执行或以后执行而被修改。 在大内存模式下 (CY8C29xxx)，所有 RAM 页面指针寄存器也会出现这种状况。 如果需要，可通过调用 fastcall16 函数来保留其值。 当前，仅修改 CUR_PP 页面指针寄存器。

LCD_DrawBG

说明：

从字符位置 (bRow,bCol) 开始绘制水平条形图，字符长度为 “bLen”，图形止于 “bPixelColEnd” 列。

C 语言原型：

```
void LCD_DrawBG(BYTE bRow, BYTE bCol, BYTE bLen, BYTE bPixelColEnd);
```

汇编：

Note 当使用大型储存器模块时，调用 LCD_DrawBG 时，应在函数名称前加下划线，即调用 _LCD_DrawBG.

小型储存器模块汇编：

```
mov   A,19h           ; Set bPixelColEnd = 25
push  A
mov   A,06h           ; Set bLen = 6 pixel columns
push  A
mov   A,03h           ; Set bCol = 3
push  A
mov   X,SP            ; Setup data pointer (X)
dec   X
mov   A,01h           ; Set bRow = 1 -> the second line
lcall LCD_DrawBG
add   SP,-3           ; Restore the stack
```

大型储存器模块汇编：

```
mov   A,19h           ; Set bPixelRowEnd = 25
push  A
mov   A,06h           ; Set bLen = 6 pixel columns
push  A
mov   A,03h           ; Set bCol = 3
push  A
mov   A,01h           ; Set bRow = 1
push  A
lcall _LCD_DrawBG
add   SP,-4           ; Restore the stack
```

参数：

bRow： 定义开始字符行 - 范围为 0 至行数减 1。

bCol： 定义开始字符列 - 范围为 0 至字符列数减 1。

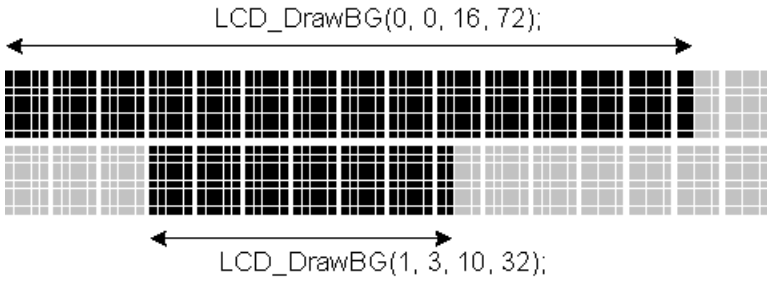
bLen： 定义包含全部字符的条形图长度。

bPixelColEnd： 定义从哪个像素列开始绘制图形。

Note 实体条形图从字符 (bRow,bCol) 的第一个像素列开始绘制所有像素列，一直到 bPixelColEnd 指定的像素列为止。 线性条形库用以图绘制定义好的特殊字符

线性条形图中 bLen=1，bPixelColEnd 的范围为 1 至 5。

Figure 5. 水平条形图示例



返回值：

None

副作用：

A 和 X 寄存器可以由此函数的本次执行或以后执行而被修改。 在大内存模式下 (CY8C29xxx)，所有 RAM 页面指针寄存器也会出现这种状况。 如果需要，调用函数负责通过调用 fastcall16 函数保留值。 当前，仅修改 CUR_PP 页面指针寄存器。

垂直条形图函数

每个显示屏字符由 5 个水平像素乘以 8 个垂直像素组成。 垂直条形图在单个字符中显示一组水平线，每一条线由 1 个垂直像素乘以 5 个水平像素组成，即一个像素行。 每个字符可显示 1 至 8 个水平像素行，8 个像素行即可显示完整字符。

从字符底部开始，第一个像素行编号为 1，最后一行编号为 8。 两行结合在一起可生成 16 个像素行的垂直条形图。

LCD_InitVBG

说明：

初始化 LCD 以显示垂直条形图。 在调用 LCD_DrawVGB() 之前，应当先调用本函数。本函数使用绘制垂直条形图所需的数据初始化自定义字符 RAM。

C 语言原型：

```
void  LCD_InitVBG(void);
```

汇编：

```
lcall  LCD_InitVBG
```

参数：

None

返回值：

None

副作用：

A 和 X 寄存器可以由此函数的本次执行或以后执行而被修改。 在大内存模式下 (CY8C29xxx)，所有 RAM 页面指针寄存器也会出现这种状况。 如果需要，调用函数负责通过调用 fastcall16 函数保留值。 当前，仅修改 CUR_PP 页面指针寄存器。

LCD_DrawVBG

说明：

从字符位置 (bRow,bCol) 的第一个像素行开始绘制垂直条形图，字符高度为 bHeight，止于指定的垂直像素行 bPixelRowEnd。

C 语言原型：

```
void  LCD_DrawVBG(BYTE bRow, BYTE bCol, BYTE bHeight, BYTE bPixelRowEnd);
```

汇编：

Note 当使用大型储存器模块时，调用 LCD_DrawVBG 时，应在函数名称前加下划线，即调用 _LCD_DrawVBG.

小型储存器模块汇编：

```
mov   A,0Ah           ; Set bPixelRowEnd = 10
push  A
mov   A,02h           ; Set bHeight = 2 columns
push  A
mov   A,03h           ; Set bCol = 3
push  A
mov   X,SP            ; Setup data pointer (X)
dec   X
mov   A,01h           ; Set bRow = 1 -> the second line
lcall LCD_DrawVBG
add   SP,-3           ; Restore the stack
```

大型储存器模块汇编：

```
mov   A,0Ah           ; Set bPixelRowEnd = 10
push  A
mov   A,02h           ; Set bHeight = 2 columns
push  A
mov   A,03h           ; Set bCol = 3
push  A
mov   A,01h           ; Set bRow = 1
push  A
lcall _LCD_DrawVBG
add   SP, -4          ; Restore the stack
```

参数：

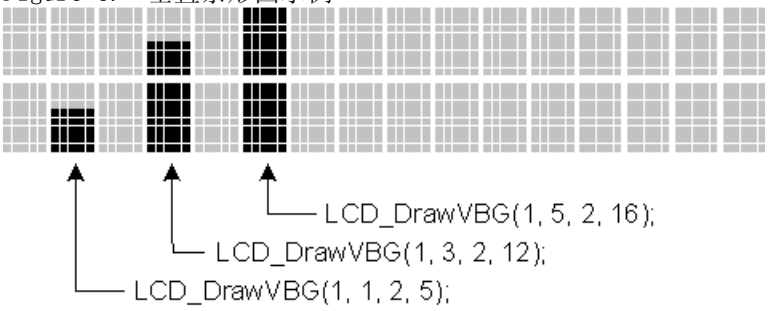
bRow： 定义开始字符行 - 范围为 0 至行数减 1。

bCol： 定义开始字符列 - 范围为 0 至字符列数减 1。

bHeight： 定义包含全部字符的垂直条形图的高度。

bPixelColEnd： 定义从哪个垂直像素行开始绘制图形。

Figure 6. 垂直条形图示例



返回值：

None

副作用：

A 和 X 寄存器可以由此函数的本次执行或以后执行而被修改。 在大内存模式下 (CY8C29xxx)，所有 RAM 页面指针寄存器也会出现这种状况。 如果需要，调用函数负责通过调用 fastcall16 函数保留值。 当前，仅修改 CUR_PP 页面指针寄存器。

可选函数

LCD_Control

说明：

向 LCD 控制寄存器中写入字节。 要了解具体的 LCD 有效命令，请查看相应的 LCD 数据手册。

C 语言原型：

```
void  LCD_Control(BYTE bCmd);
```

汇编：

```
mov   A,03h          ; Load data to be written to Control register.
lcall  LCD_Control    ; Call function
```

参数：

bCmd： 发送给控制寄存器的命令字节值。

返回值：

None

副作用：

A 和 X 寄存器可以由此函数的本次执行或以后执行而被修改。 在大内存模式下 (CY8C29xxx)，所有 RAM 页面指针寄存器也会出现这种状况。 如果需要，调用函数负责通过调用 fastcall16 函数保留值。 当前，仅修改 CUR_PP 页面指针寄存器。

LCD_WriteData

说明：

向 LCD 数据寄存器写入字节或字符。

C 语言原型：

```
void  LCD_WriteData(BYTE bData);
```

汇编：

```
mov   A,03h            ; Load data to be written to Data register
lcall  LCD_WriteData    ; Call function
```

参数：

bData： 发送给数据寄存器的数据字节值。

返回值：

None

副作用：

A 和 X 寄存器可以由此函数的本次执行或以后执行而被修改。 在大内存模式下 (CY8C29xxx)，所有 RAM 页面指针寄存器也会出现这种状况。 如果需要，调用函数负责通过调用 fastcall16 函数保留值。 当前，仅修改 CUR_PP 页面指针寄存器。

LCD_Delay50uTimes

说明：

延迟 “bTimes”，延迟时间为 50 µs 的倍数。 延迟循环与 CPU 时钟无关。

C 语言原型：

```
void  LCD_Delay50uTimes(BYTE bTimes);
```

汇编：

```
mov   A,03h             ; Load delay time (example 3 would be 150uSec).
lcall  LCD_Delay50uTimes ; Call function
```

参数：

bTimes： 延迟 50 µS 的次数。

返回值：

None

副作用：

A 和 X 寄存器可以由此函数的本次执行或以后执行而被修改。 在大内存模式下 (CY8C29xxx)，所有 RAM 页面指针寄存器也会出现这种状况。 如果需要，通过调用 fastcall16 函数来保留其值。

LCD_Delay50u

说明：

延迟 50 µs。 此函数与时钟无关。

C 语言原型：

```
void  LCD_Delay50u(void);
```

汇编：

```
lcall  LCD_Delay50u      ; Call function
```

参数：

None

返回值：

None

副作用：

A 和 X 寄存器可以由此函数的本次执行或以后执行而被修改。 在大内存模式下 (CY8C29xxx)，所有 RAM 页面指针寄存器也会出现这种状况。 如果需要，通过调用 fastcall16 函数来保留其值。

固件源代码示例

以下为在 LCD 上打显示符串的简单汇编语言和 C 语言示例：

```
;;-------------------------------------------------------------------
;; Sample asm LCD Code
;;
;;  Print the string "PSoC LCD" on the top row starting at the 6th
;;  location on an LCD.
;;-------------------------------------------------------------------

include "m8c.inc"       ; part specific constants and macros
include "PSoCAPI.inc"   ; PSoC API definitions for all User Modules
export _main

area text (ROM, REL)

_main:
   call   LCD_Start       ; Initialize LCD
   mov    A,00h          ; Set cursor position at row = 0
   mov    X,05h           ; col = 5
   call   LCD_Position
   mov    A,>THE_STR      ; Load pointer to ROM string
   mov    X,<THE_STR
   call   LCD_PrCString   ; Print constant "ROM" string

loop:
   jmp    loop

.LITERAL
THE_STR:
DS "PSoC LCD"
DB  00h                   ; String should always be null terminated
.ENDLITERAL
```

以下为使用 C 语言编写的示例项目：

```
//-------------------------------------------------------------------
// Sample C code for LCD
//
//  Print the string "PSoC LCD" on the top row starting at the 6th
//  location on an LCD.
//
//-------------------------------------------------------------------
#include <m8c.h>        // part specific constants and macros
#include "PSoCAPI.h"    // PSoC API definitions for all User Modules
```

```
void main(void)
{
   char theStr[] = "PSoC LCD";    // Define RAM string
   LCD_Start();                   // Initialize LCD
   LCD_Position(0,5);             // Place LCD cursor at row 0, col 5.
   LCD_PrString(theStr);          // Print "PSoC LCD" on the LCD
}
```

版本历史记录

版本	创作者	说明
1.5	DHA	添加了版本历史
1.60	DHA	修正了创建可用端口清单的机制。
1.60.b	DHA	本数据手册中包括如下更新： 1. 更新了 LCD_DrawBG 和 LCD_DrawVBG. 的示例代码。 2. 补充了有关用于 LCD 端口控制的影子寄存器的更多信息。

Note PSoC Designer 5.1 在所有用户模块数据手册中都引入了 “ 版本历史 ”。 本数据表详细介绍了当前和先前用户模块版本之间的区别。

Copyright © 2012 Cypress Semiconductor Corporation. The information contained herein is subject to change without notice. Cypress Semiconductor Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in a Cypress product. Nor does it convey or imply any license under patent or other rights. Cypress products are not warranted nor intended to be used for medical, life support, life saving, critical control or safety applications, unless pursuant to an express written agreement with Cypress. Furthermore, Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress products in life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

PSoC Designer™ and Programmable System-on-Chip™ are trademarks and PSoC® is a registered trademark of Cypress Semiconductor Corp. All other trademarks or registered trademarks referenced herein are property of the respective corporations.

Any Source Code (software and/or firmware) is owned by Cypress Semiconductor Corporation (Cypress) and is protected by and subject to worldwide patent protection (United States and foreign), United States copyright laws and international treaty provisions. Cypress hereby grants to licensee a personal, non-exclusive, non-transferable license to copy, use, modify, create derivative works of, and compile the Cypress Source Code and derivative works for the sole purpose of creating custom software and or firmware in support of licensee product to be used only in conjunction with a Cypress integrated circuit as specified in the applicable agreement. Any reproduction, modification, translation, compilation, or representation of this Source Code except as specified above is prohibited without the express written permission of Cypress.

Disclaimer: CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Cypress reserves the right to make changes without further notice to the materials described herein. Cypress does not assume any liability arising out of the application or use of any product or circuit described herein. Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress' product in a life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

Use may be limited by and subject to the applicable Cypress software license agreement.