Application Note AN2343

LCD Driver Based on the HT1621 Controller

Author: Andrew Smetana
Associated Project: Yes
Associated Part Family: All
PSoC Designer Version: 4.2 SP3
Associated Application Notes: AN2228

Abstract

This Application Note describes implementation of a liquid crystal display (LCD) driver based on the widely available HOLTEK HT1621 LCD Controller. Methods and algorithms of display control are described and an API library is provided. The LCD used in this example was a customer's custom part. The proposed algorithms can be easily adapted to any custom LCD panel connected to the controller.

Introduction

LCDs are widely used as data display devices in embedded systems. Among the features that have made LCDs popular are low price, low power dissipation, lightweight, durability, reliability, and broad support by dedicated ICs for communication with microcontrollers.

A good example of an LCD driver is the Hitachi character LCD driver HD44780, the industry standard. This dot-matrix LCD controller is supported by PSoC APIs. It is useful in applications that permit alphanumeric data display. However, specialized displays are often needed. Specialized displays keep end- product prices low, simplify the interface between the microcontroller and LCD driver, and decrease weight and size parameters. Examples of specialized displays include clocks, calculators, telephones, and home and industrial appliances. This implementation is based on one of these dedicated drivers, the HT1621.

This Application Note addresses the proposed implementation in two parts:

- General Description
- LCD Driver Implementation

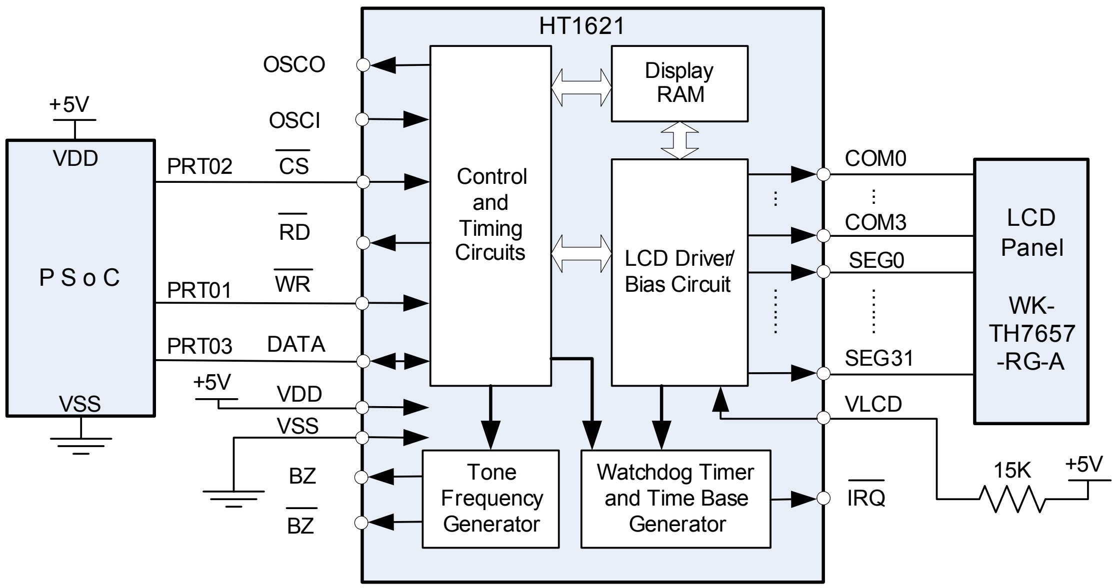
General Description

The HT1621 driver is a 128-segment (32x4), multi-functional LCD driver with memory mapping. The software configuration feature of HT1621 makes it suitable for many LCD applications, including LCD modules and display subsystems. Only three or four connections are required for interfacing between the host controller and the HT1621.

A structural schematic of the display system is shown in Figure 1. This structure requires few external components and uses only three interface connections. The system consists of the PSoC, HT1621 controller, and an LCD panel. The HT1621, besides its primary function as an LCD controller, has peripherals including the watchdog timer, time base generator and the tone frequency generator. For more information about these features, refer to the HT1621 data sheet. Note that the controller has an on-chip RC oscillator (256 kHz) for controlling the LCD and peripherals. This General Description focuses on the components and functions of the HT1621 driver that relate directly to the LCD:

- Display Memory RAM
- LCD Driver in HT1621
- Command Format
- Interfacing with HT1621

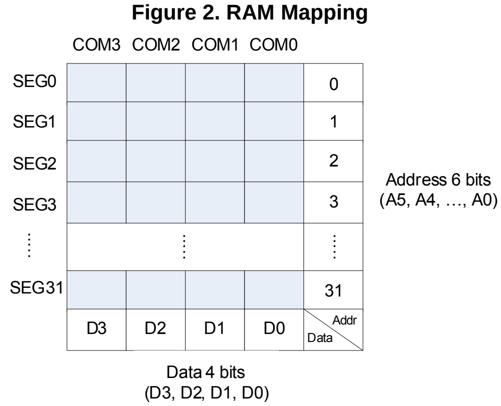
Figure 1. Structural Schematic of the Display System



Display Memory RAM

The static display memory (RAM) is organized in a 32x4-bit matrix, which stores the displayed data. The contents of the RAM are directly mapped to the contents of the LCD driver. Data in the RAM can be accessed by READ, WRITE, and READ-MODIFY-WRITE commands. Note that in the schematic for the proposed LCD system, one-way communication is implemented (from PSoC to HT1621). The correspondence between RAM and the LCD pattern is shown in Figure 2.

Figure 2. RAM Mapping



LCD Driver in HT1621

The HT1621 is a 128 (32x4)-segment LCD driver. It can be configured as 1/2 or 1/3 bias, and is designed with two, three or four segments in common with the LCD driver. These configuration features make the HT1621 suitable for multiple LCD applications. The LCD driving clock is derived from the system clock. The signal frequency of the display control is 256 Hz. Some LCD commands are described in Table 1.

In the table, the bold font of 100 indicates the command code ID. The X in the code signifies any bit in a corresponding position. If successive commands are issued, the command code ID (except in the first command) is omitted. The LCD OFF command turns the LCD display off by disabling the LCD bias generator. The BIAS and COM commands relate to the LCD panel. LCD commands make the HT1621 compatible with most types of LCD panels.

Table 1. LCD Commands

Name	Command Code	Function
LCD OFF	**100** 00000010X	Turn off LCD outputs
LCD ON	**100** 00000011X	Turn on LCD outputs
BIAS and COM	**100** 0010abXcX	c=0: 1/2 bias option c=1: 1/3 bias option ab=00: 2 commons option ab=01: 3 commons option ab=10: 4 commons option

Command Format

The HT1621 can be configured by using the software settings. There are two instruction modes:

- **Command mode** – For configuration of HT1621 resources
- **Data mode** – For transmission of display data to the LCD. Data mode includes READ, WRITE and READ-MODIFY-WRITE operations.

These modes are summarized in Table 2.

Table 2. HT1621 Instruction Mode Parameters

Operation	Mode	ID
READ	Data	110
WRITE	Data	101
READ-MODIFY-WRITE	Data	101
COMMAND	Command	100

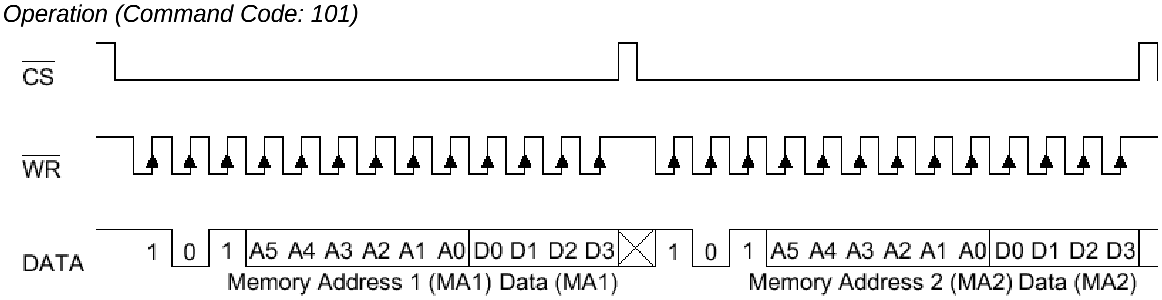
The instruction mode ID must be issued before the data or command is sent.

Interfacing with HT1621

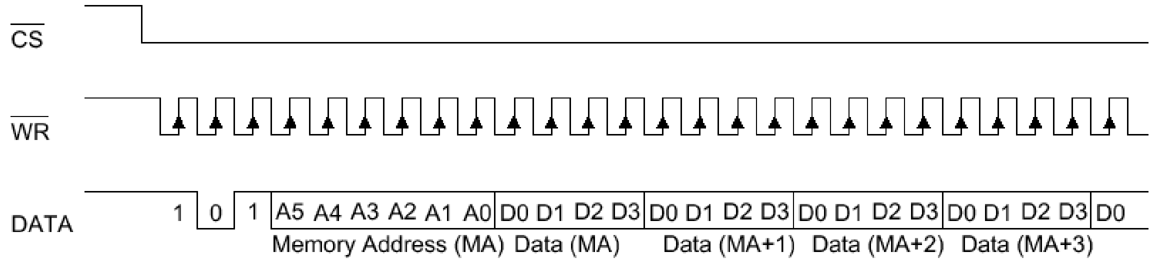
Only four lines are required to interface with the HT1621: CS, DATA, RD, and WR. The CS line is used to initialize the serial interface circuit and to terminate the connection between the host controller and the HT1621. The DATA line is the serial data input/output (IO) line. Data to be read or written, or commands to be written, must be passed through the DATA line. The RD line is the READ clock input, and the WR line is the WRITE clock input. The data, address or command on the DATA line are all clocked into the HT1621 on the rising edge of the WR signal. In this proposed driver implementation, only three lines are used: CS, WR, and DATA. In this way, simplex communication is used to control the LCD. A time diagram of a WRITE operation is shown in Figure 3.

Figure 3. Time Diagram of a WRITE Operation

WRITE Operation (Command Code: 101)



WRITE Mode (Successive Address Writing)



LCD Driver Implementation

This project uses a custom LCD as an output device. The HOLTEK HT1621 driver forms an image on the LCD. In this way, the LCD and its driver create a data display system.

The following subsections guide you through key aspects of the LCD driver implementation:

- Algorithm of Logical Access to the LCD Segments / Symbols
- Digits and Segment Highlight
- Display APIs

The code example in the following section describes how logical access to the individual LCD segment is organized; that is, how to switch on or switch off a segment). The code also describes how logical access to whole symbols (for example, digits) is organized.

Algorithm of Logical Access to the LCD Segments / Symbols

Analyzing the layout of the display segments, it can be observed that their organization has no defined regularity. Also, to access a given segment, it is necessary to know two of its coordinates [segment, common]. Therefore, it is necessary to work out a standard method by which each segment can be accessed. That is, to turn on a certain segment of a display, it is necessary to write “1” in the HT1621 video memory at a given address [segment, common], as shown in Figure 2. To standardize access to a set of disordered display segments, they must be enumerated in an easily mapped order. For the example shown in Figure 4, the *LCD_segments[102]* array is declared, which contains information about each segment, but already in a known order. In this case, the array elements are addresses of the display segments, as shown in Figure 5.

Convenient access to any display segment is thus organized according to indexes in the *LCD_segments* array. This array, like other functions and variables used by the LCD module, is located in the *lcd.c* file, and definitions of constants are in the corresponding *lcd.h* file. The indexes of individual segments are defined in the following view (from *lcd.h*):

```
#define LCD_DIGIT_INDEX          0          // next digit + 7
#define LCD_ERROR_INDEX          70         // ERROR
#define LCD_PC_INDEX             71         // PC
#define LCD_CALC_INDEX           72         // CALC
#define LCD_MUSIC_INDEX          73         // MUSIC
#define LCD_VOLUME_INDEX         74         // VOLUME
#define LCD_NUM_LOCK_INDEX       75         // NUM LOCK
#define LCD_MINUS_INDEX          76         // MINUS
#define LCD_CAPS_LOCK_INDE       77         // CAPS LOCK
#define LCD_SCROLL_LOCK_INDEX    78         // SCROLL LOCK
#define LCD_FUNCTION_INDEX       79         // FUNCTION
#define LCD_BATTERY_INDEX        80         // BATTERY
#define LCD_PM_INDEX             81         // PM
#define LCD_AM_INDEX             82         // AM
#define LCD_POINT_INDEX          83         // next point + 1
#define LCD_COMMA_INDEX          93         // next comma + 1
#define LCD_COLON_INDEX          100        // next + 1
```

As can be seen in Figure 4, at first, the digital segments are enumerated in the determined order in the array in series. This standard method of storing enables easy access to any digital position on the display by adding a digital position multiplied by 7 to the *LCD_DIGIT_INDEX* constant. Access to the points and commas can be arranged in a similar way.

Digits and Segment Highlight

Consider how indicating a digit in the single position is implemented. Digits on the screen are presented as a set of 7-character segments. It is necessary to define the template of each digit in order to achieve visualization. Screen images of some digits and the corresponding digits' templates in memory are shown in Figure 7.

Figure 4. LCD Segment Enumeration

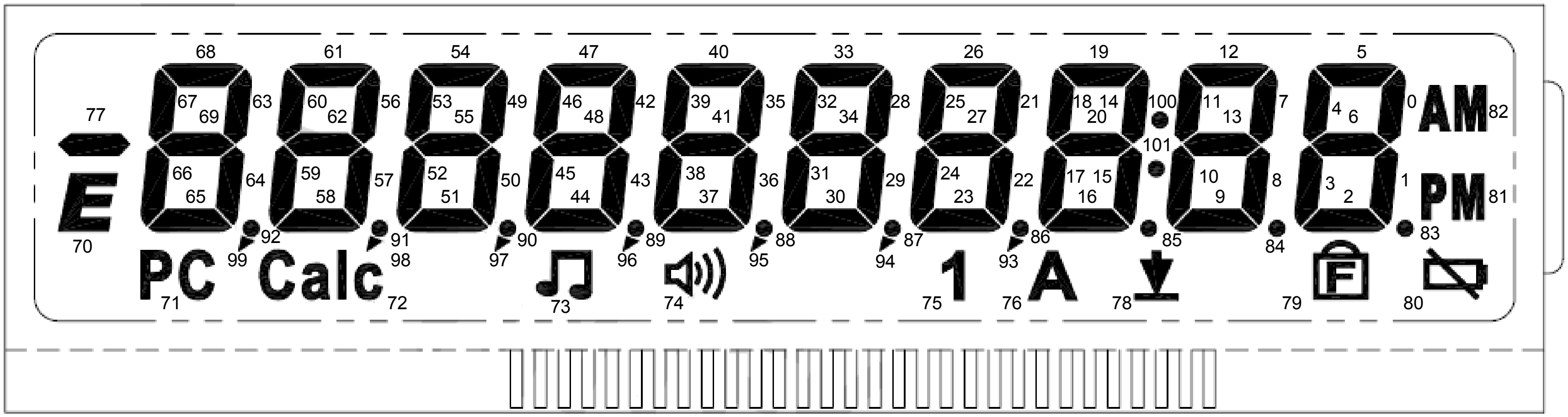


Figure 5. LCD Segment Presentation in the *LCD_segments[102]* Array

	Segment No.					Common No.	
0	S4	S3	S2	S1	S0	C1	C0
7	6	5	4	3	2	1	0

Figure 6. Digital Segment Enumeration

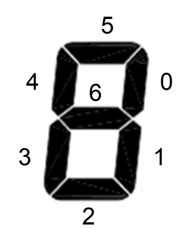
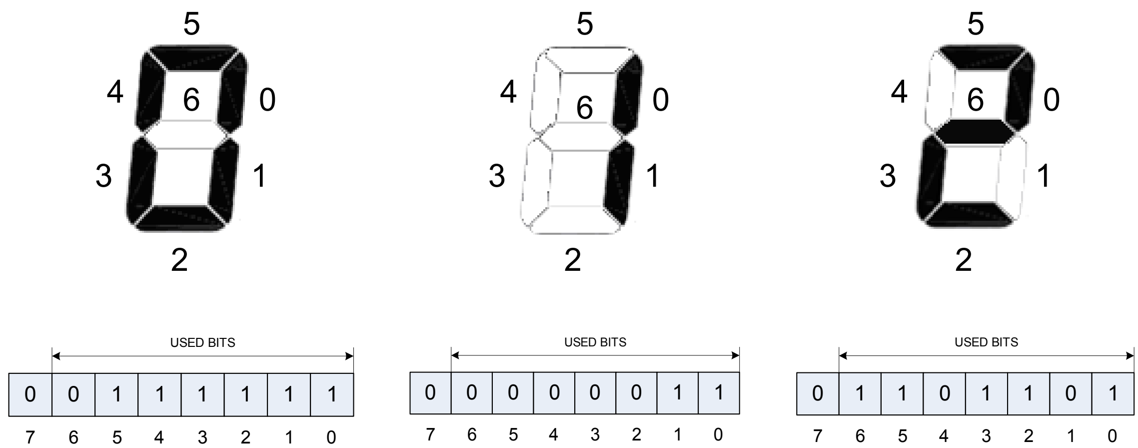


Figure 7. Images of Digits and Their Templates



A template of each digit is stored in the *LCD_digit_segments[12]* array. In addition to the 10 digits, this array contains the codes of special symbols: SPACE – for turning off a digital position, and MINUS – for drawing the minus sign (−).

To explain the process of any *digit* highlight in *i* – position: Digits are enumerated from right to left, from 0 to 9 on the LCD. First, you must obtain the digit template from the *LCD_digit_segments[12]* array, using the following sequence:

digit_template = LCD_digit_segment[digit] (1)

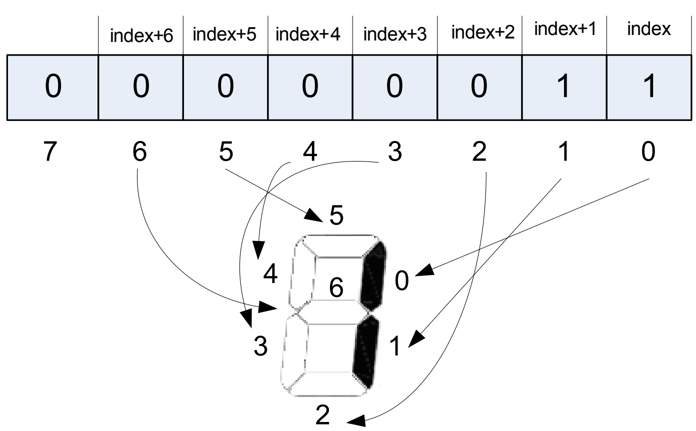
Calculate the index of the first segment in *i*-digit position in the *LCD_segments* table:

*index = LCD_DIGIT_INDEX + 7*i* (2)

The elements of the *LCD_segments* array, in combination with the indexes (*index..index+7*) ,yield the addresses of the working segments in *i*-digit position. Also, in the loop, each bit of a digit's template is examined and it is determined whether a segment must be turned on or off in the corresponding *LCD_segments[index+j]* element (*j=0..6*) in the display. This process is shown in Figure 8.

The element *LCD_segment[index+j]* determines the physical address (coordinate) of a segment in video memory.

Figure 8. Correspondence Between the Template of a Digit and the Template's Segment Indexes in the LCD_segments[102] Array



To provide convenient access to the video memory of the HT1621 display driver, a local copy of the driver's memory was created in the RAM of the PSoC device and declared as *BYTE LCD_memory[16]*. Also, a unidirectional communication interface requires this kind of LCD RAM usage. Each bit in this memory is responsible for the state (1 - on, 0 - off) of one segment in the LCD. After changes are made to the local copy of the display's video memory, it is copied to the memory of HT1621, and the screen image is automatically updated. Access to an individual segment in the *LCD_memory* array must be executed on the basis of a corresponding segment address [segment, common]. The video memory of HT1621 is organized in a 32x4-bit matrix, so 16 bytes of device RAM is enough to save a local copy of the video memory.

Display of the remaining LCD elements is simpler, because they occupy only one segment. For example, consider the output of a PC segment on the LCD:

```
Addr = LCD_segments[LCD_PC_INDEX];
Segment = Addr >> 2;
// calculating
Common = Addr & 0x03;
// physical address of segment
```

By means of this physical address [segment, common], the corresponding bit in local video memory is modified. Next, the RAM video memory is copied to controller of the display.

Display APIs

After access is gained to the segments of the display, images are created on the LCD by means of the following functions:

- *void LCD_show_digit(char pos, char digit)* – Displays a digit in a given position
- *void LCD_show_minus(BOOL state)* – Shows/hides the minus (-) segment for a calculator's display, depending on the *state* parameter
- *void LCD_show_colon(BOOL state)* – Shows/hides a colon (:), depending on the *state* parameter
- *void LCD_show_point(char pos,BOOL state)* – Shows/hides a point in the given position, depending on the *state* parameter
- *void LCD_show_segment(char segment_index, BOOL state)* – Shows/hides a given segment, depending on the *state* parameter. The segment is defined by the index in the *LCD_segments* array. Segment indexes definitions are presented in the *lcd.h* code example earlier in this Application Note.

In addition to the above functions, other LCD APIs are implemented. They control the state of the special symbols on the screen (Calc, PC, AM, PM, and so on).

The preceding bulleted list of functions works only with the local copy of video memory. The functions do not form images on the actual LCD. To enable images on the actual LCD, subroutines were written that implement the physical interface between PSoC and the HT1621 driver. These subroutines are implemented fully in the firmware, without use of PSoC hardware resources. The subroutines that enable images on the actual LCD are the following:

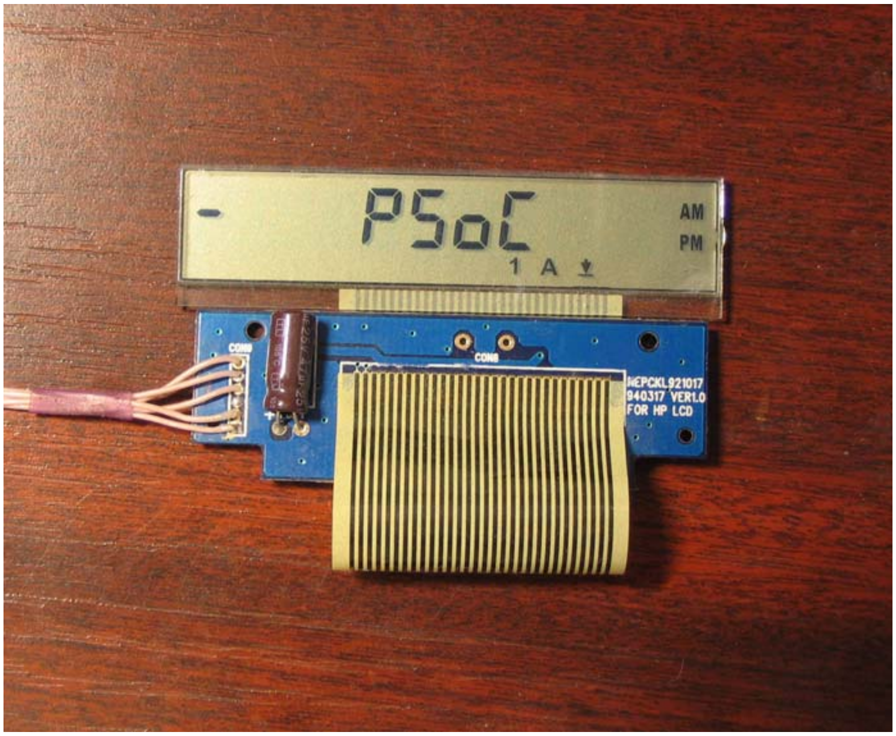
- *void LCM_Initial(void)* – Performs initializations actions to start the HT1621 driver. This subroutine executes three functions: turns on the built-in oscillator, sets the 1/3 bias and 1/4 duty LCD properties, and turns on the built-in bias voltage generator to highlight the LCD panel.
- *void SendLcdCommand(BYTE command)* – Sends a command to the HT1621 driver
- *void LcdWrite(void);* – Copies the local video memory to the HT1621 driver

The LCD driver and its corresponding library occupy about 1.6K of Flash memory in the PSoC small memory model and 1.8K in the large memory model. RAM use is the same for both models and requires 16 bytes of static memory. To save memory, a bidirectional driver interface must be implemented.

Conclusion

This Application Note has described a display system built on a PSoC, HT1621 controller and LCD panel. The interface between the device and the HT1621 driver has been described, as has access to the LCD. Using the proposed material, a customer-selected LCD panel can be applied, and a corresponding API library can be developed with a minimum of effort. Step-by-step implementation of the system has been detailed. An inexpensive, reliable, and durable display system is the result.

LCD Photograph



About the Author

Name: Andrew Smetana

Title: Electrical Engineer

Background: Andrew Smetana received a Master of Science degree in 2004 from National University “Lvivska Polytechnika“ (Ukraine). His interests include various aspects of embedded systems development.

Contact: smetana@ukrwest.net

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709
Phone: 408-943-2600
Fax: 408-943-4730
http://www.cypress.com