

Micrium, Inc.

Using a GUI in an Embedded System

Application Note

AN-5000

Christian E. Legare

Christian.Legare@Micrium.com

Jean J. Labrosse

Jean.Labrosse@Micrium.com

www.Micrium.com

1.00 Introduction

Introducing a new product requires the designer to think about the product differentiators. Designing a user-friendly product, considering all other features are equivalent, will help increase the product acceptance and sales. A good User Interface is definitively one of these differentiators. In many instances, a Graphical User Interface (GUI) is the best approach. For examples, consider these popular products:

- Cell phones
- Digital Cameras
- Industrial Controls
- GPS Instrumentation
- Military / F.L.I.R. Applications
- Internet Appliances
- Wireless Devices
- Copiers
- MP3 Players
- Medical Devices
- Handheld Computers
- Printers
- Set-top Boxes
- Web Browsers

Users expect a product with a graphical display that looks familiar – similar to the de facto standard Microsoft Windows Graphical User Interface, as well as color displays because the information content is higher.

As a developer, you quickly realize that these expectations have a strong impact on the cost of the product.

In addition, you might want to make your product as universal as possible by adding multi-lingual support – mix Latin, Cyrillic, Han, Katakana, Hiragana etc. in a single application with JIS and UNICODE formats. The language support on the display also has a high toll on an embedded system. Character sets are generally stored in memory. Embedded systems often have scarce memory resources. The use of a GUI will allow you to use very large character sets in memory-limited systems.

An embedded system has limitations that a PC desktop does not have. These limitations are often in conflict with the user expectations of a “Windows like” interface, because this type of interface is resource hungry.

The two major constraints that a designer has to work with when considering implementing a GUI are:

a. Code space (ROM) / Data space (RAM) Requirements:

Most embedded systems have low ROM/RAM. High performance combined with low resource usage has always been a major design consideration. Depending on which modules are being used, even single-chip systems with less than 64Kb ROM and 2Kb RAM (Yes K, this is not a typo) can be supported by an embedded GUI. The actual performance and resource usage depends on many factors (CPU, compiler, memory model, optimization, configuration, interface to LCD controller, etc.).

b. Low CPU overhead:

Most of the drawing is done by the embedded CPU which does not run as fast as current desktop computers. One is lucky if he can have a 32-bit CPU running at 100 MHz, 20 to 100 times slower than desktops. In fact, embedded GUIs should run on 8/16/32-bit CPUs.

Other hardware issues are the choice of LCD displays and LCD controllers. A good combination of display and controller will reduce the load on the CPU and ROM/RAM and at the same time provide a high quality GUI. Also, products can be designed using either ‘Black and White’ (B&W), ‘Gray Scales’ (GS) or ‘Color’ displays.

Any product requires some form of feedback from the user. When a GUI is used, the user feedback is reflected on the display. An embedded GUI needs to support one of the following input devices:

Input device	GUI support
A keyboard with limited number of keys	Soft Keys supports "membrane keys" placed around the perimeter of the display screen
A Touch screen or light pen	The GUI will provide assistance by interacting directly with input hardware, maximizing throughput.
A tracking device (mouse, track pad, joystick, track ball or track point, etc...)	In this case the GUI will help by providing support for software cursors.

Table 1 – Input devices

2.00 How do you select a GUI?

When designing a product with a GUI, you may be tempted to write your own from scratch. This is generally a mistake because of the time involved.

C++ is becoming more and more popular as embedded CPU resources increase, but C++ has not established itself yet, as the language of choice in the embedded world. C is still the predominant language used in embedded systems.

Companies are working towards code re-use and are thus protecting their investment. An embedded GUI needs to be portable to different processors. A designer would like to find a library that conforms to the C standard and that is highly portable by abstracting all hardware-dependent functionality. A library strictly following the ANSI C Standard would assure this portability.

It's unusual for a desktop GUI to be provided in source form. For an embedded system, it's almost mandatory since it protects the end user from the GUI provider. It's not sufficient to provide source code - the code must be clean, consistent and well documented.

Over 50% of the embedded systems designed today have some sort of real-time operating system, whether commercial or designed in-house. The GUI must be designed to work with just about any of them while at the same time allow multiple tasks to access the display as needed. For the other 50%, the GUI should also be able to run in standalone mode - the entire application running in one big loop. Interrupts must be used for real time parts of the software since no real time kernel is used.

The architecture of an embedded GUI can be represented by the diagram of Figure 1. The selection of your GUI must be based on the availability of each modules and the quality of their implementation.

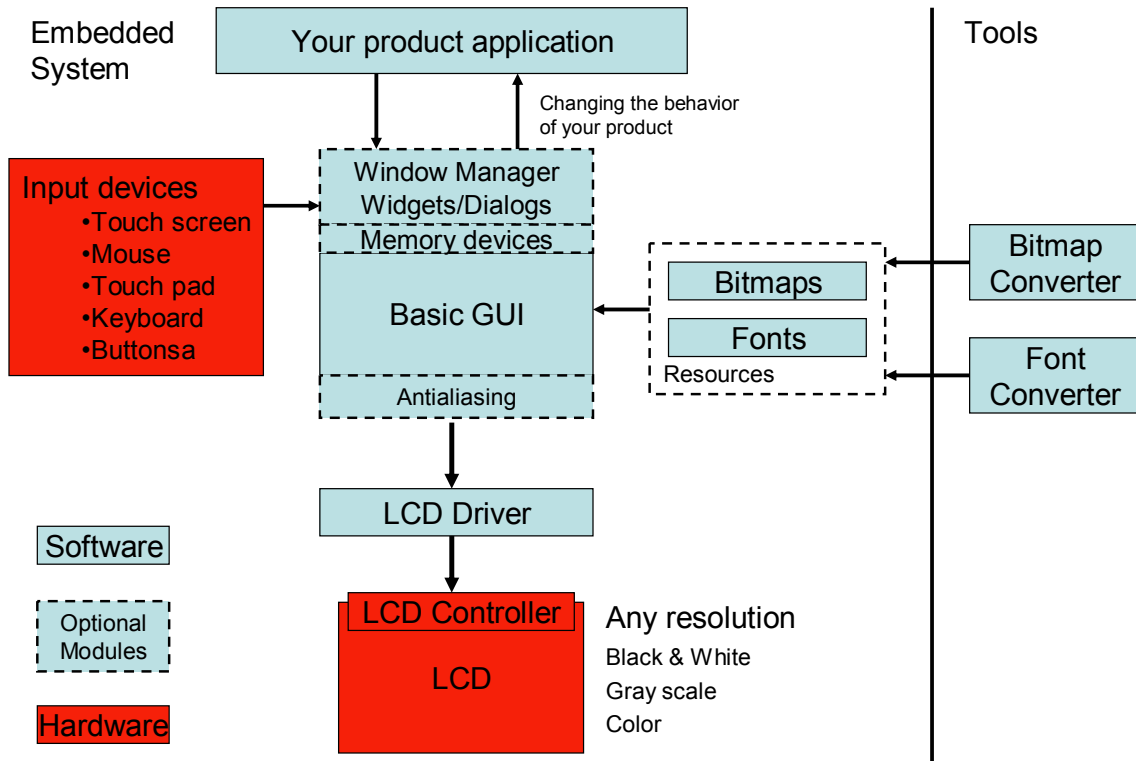


Figure 1 – Embedded System

The examples in the following sections are taken from μ C/GUI, a commercial portable embedded Graphical User Interface software package.

3.00 Window Management Support?

One decision that has a major influence on your GUI implementation is the choice to use or not a Window Management module. The selection criteria can be summarized as:

- User interface look and feel.
 - Code and data space requirement
 - Complexity or simplicity
1. Because your user is expecting a look and feel he is familiar with, you may want to provide him with the versatility of a Window Management module.
 2. If you choose to include a Window Management module, you have to be ready to pay the price in memory space required. Using windows, widgets (controls) and dialogs requires more resources. A typical example with μ C/GUI:

Without Window Manager:

- RAM: 100 bytes
- Stack: 500 bytes
- ROM: 10-25 KB (depending on the functionality used)

With Window Manager, widgets (controls) and dialogs:

- RAM: 2KB and more as you add windows (typically 6KB)
- Stack: 1.2KB
- ROM: 30KB and more depending on the number of windows and different widgets used.

Note that ROM requirements will increase if your application also uses many fonts.

3. You would use a Window management module as soon as your application gets more complex, for example displaying alarms on top of other information and having to redraw the display.

As a last note, the use of a Window Management module generally does not have a significant impact on the CPU load.

3.01 Without Window Management

If you cannot afford the extra resources needed by windowing, the GUI still need to offer a minimum set of functions to provide an embellished product such as:

- Text and Number display
- Multiple fonts
- Lines, polygons, arcs, circles, ellipses
- Boxes

3.01.01 Text and Number display

Only a few routines are required to allow you to write any text, using any available font, at any location on the display. In μ C/GUI, some of these routines are `GUI_DispChar()`, `GUI_DispString()`, `GUI_DispStringAt()` and `GUI_GotoXY()`.

Instead of using standard C library functions, embedded GUIs generally come with more efficient built-in functions to display values anywhere on the screen. In addition, most of these display functions do not require the use of time consuming floating-point library and are thus optimized for both speed and size.

3.01.02 Multiple fonts

An embedded GUI generally only needs to support a few fonts. Because memory is scarce in an embedded system, these fonts are usually coded as bitmaps. The fonts are stored in either C files, object files or libraries. The font files are linked with your application but the font declarations are contained in header files.

In general, 2 types of fonts are required: monospaced bitmap fonts and proportional bitmap fonts. In high-end system, you will also find proportional bitmap fonts with antialiasing (described later) information. To be as universal as possible the GUI should support ASCII, ISO 8859-1 and Unicode.

It is generally recommended to compile all available fonts and link them as library modules or putting all of the font object files in a library which you can link with your application. This gives the opportunity to the linker to keep only the fonts which are needed by your application.

3.01.03 Lines, Polygons, Arcs, Circles, Ellipses

As a minimum, your GUI library needs to include 2-D graphic routines to draw points, lines, polygons, arcs, circles and ellipses which should be sufficient for a lot of applications. Since these routines are called frequently in most applications, they must be optimized for speed. For example, the horizontal and vertical line functions do not require the use of single-dot routines.

3.01.04 Bitmaps

Bitmaps are important in any GUI application. Everybody wants to have its logo on the display. For example, bitmaps which can be used with μ C/GUI are normally defined as `GUI_BITMAP` structures in C. The structures -- or rather the picture data which is referenced by these structures -- can be quite large. For images that you plan to re-use (i.e. a company logo) it is very efficient to define them as `GUI_BITMAP` structures that can be used directly.

On the other hand, for application that continuously references new images, such as bitmaps downloaded by the user, the GUI should also provide a mean to download and display these images. μ C/GUI provides two functions for these situations:

- For bitmaps defined at compile time:
`GUI_DrawBitmap()`
- For bitmaps downloaded at run time:
`GUI_BmpDraw()`

It is time-consuming and inefficient to generate these bitmaps manually, especially if you are dealing with sizable images and multiple shades of gray or colors. For this reason, μ C/GUI comes with a Bitmap Converter.

The μ C/GUI Bitmap Converter can convert any bitmap into C. It supports palette conversion for different LCDs. For efficiency, bitmaps may also be saved without palette data and in compressed form.

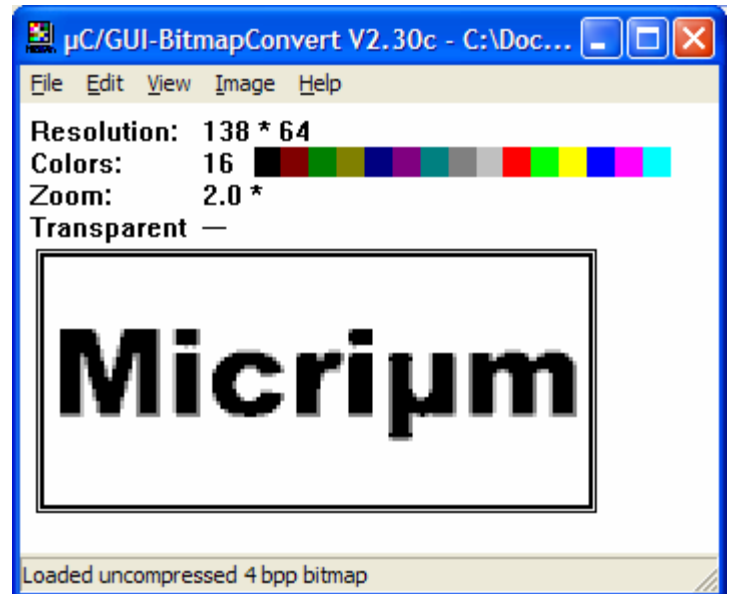


Figure 2 – Bitmap Converter example

3.02 To Window

Window management is generally built using some of the functionality described in the previous section.

3.02.01 Window Manager

A minimum Window Management option should include a set of routines which allow you to easily create, move, resize, and otherwise manipulate any number of windows. The GUI should also provide lower-level support by managing the layering of windows on the display and by alerting your application to display changes that affect its windows.

3.02.02 Widgets

Widgets are windows with object-type properties; they are called controls in the Microsoft Windows world and make up the elements of the user interface. They can react automatically to certain events; for example, a button can appear in a different state if it is pressed. Widgets need to be created, have properties which may be changed at any time during their existence and are then typically deleted when they are no longer needed.

Once a widget is created, it is treated just like any other window; the Window Manager module ensures that it is properly displayed (and redrawn) whenever necessary. Widgets are not required when writing an application or a user interface, but they can make programming much easier.

Below are examples of some of the widgets supported by μ C/GUI.






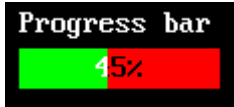


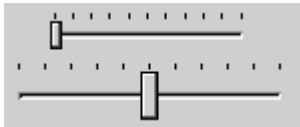
Name	Description	Sample
BUTTON	Button which can be pressed. Text or bitmaps may be displayed on a button.	
CHECKBOX	Check box which may be checked or unchecked.	
EDIT	Single-line edit field which prompts the user to type a number or text.	
FRAMEWIN	Frame window. Creates the typical GUI look.	
LISTBOX	Listbox which highlights items as they are selected by the user.	
PROGBAR	Progress bar used for visualization.	
RADIOBUTTON	Radio button which may be selected. Only one button may be selected at a time.	
SCROLLBAR	Scrollbar which may be horizontal or vertical.	
SLIDER	Slider bar used for changing values.	

Table 2 – Widgets example

3.02.03 Dialogs

A dialog box (or dialog) is normally a window that is used to request input from the user. It may contain multiple widgets, requesting information from the user through various selections, or it may take the form of a message box which simply provides information (such as a note or warning to the user) and an "OK" button.

3.02.04 Flicker-free drawing by first writing to memory

In embedded systems the display frequency can be quite high. For example, if you display the update of an RPM value. The screen is updated as drawing operations are executed, which gives it a flickering appearance as the various updates are made. Instead of updating the LCD controller chip with multiple drawing operations, memory can be used to create a virtual display. All drawing operations are first performed directly to memory and then, the final image is written to the LCD controller once.

This difference can be seen in the example below which illustrates a sequence of drawing operations both with and without the use of a virtual display called a `memory device`. The drawing is shown in Figure 3.

```

/*
*****
*                               With Virtual Display
*****
*/
static void _MeterExample(void)
{
    int          i;                /* loop counter                */
    DRAWCONTEXT  DrawContext;      /* Structure containing data for drawing */

    while (1) {
        for (i = 0; i < 220; i++) {
            _UpdateParameters(&DrawContext, i);    /* Calc needle position etc. */
            GUI_MEMDEV_Draw(NULL, &_Draw, &DrawContext, 0, 0); /* Do the drawing */
            GUI_Delay(20);                          /* Give user time to see result */
        }
    }
}

/*
*****
*                               Without Virtual Display
*****
*/
static void _DemoBandingMemdev (void)
{
    int          i;                /* loop counter                */
    DRAWCONTEXT  DrawContext;      /* Structure containing data for drawing */

    while (1) {
        for (i = 0; i < 220; i++) {
            _UpdateParameters(&DrawContext, i);    /* Calc needle position etc. */
            _Draw(&DrawContext);                  /* Do the drawing */
            GUI_Delay(20);                          /* Give user time to see result */
        }
    }
}

```

In fact, if there is not enough memory to buffer the entire drawing area, μ C/GUI slices the drawing area in bands and automatically calls the drawing routines multiple times.



Figure 3.

3.02.05 Antialiasing

Antialiasing smoothes curves and diagonal lines by "blending" the background color with that of the foreground. The higher the number of shades used between background and foreground colors, the better the antialiasing result (and the longer the computation time).

4.00 Project development considerations

Designing an embedded GUI is a complex and time consuming task. It is already difficult to develop the user interface for your product and, in the early stages of the development cycle, it can be difficult or impossible to do application development on the intended target platform simply because the hardware is unstable or unavailable.

4.01 Simulation

µC/GUI allows you to completely develop the GUI portion of your product on a standard Microsoft Windows platform. You can create, test, and debug your entire user interface using familiar and mature Windows development tools. In other words, instead of using the actual LCD driver, the PC simulates the LCD of your product (see Figure 4). This lets you get a head start on your project and thus greatly reduces development time. Once you have the actual hardware, you simply recompile the exact same code and run it on your target. The difference lies only in the selection of the LCD driver.

A key benefit to simulation is to allow you to quickly create product prototypes for use by management and marketing.

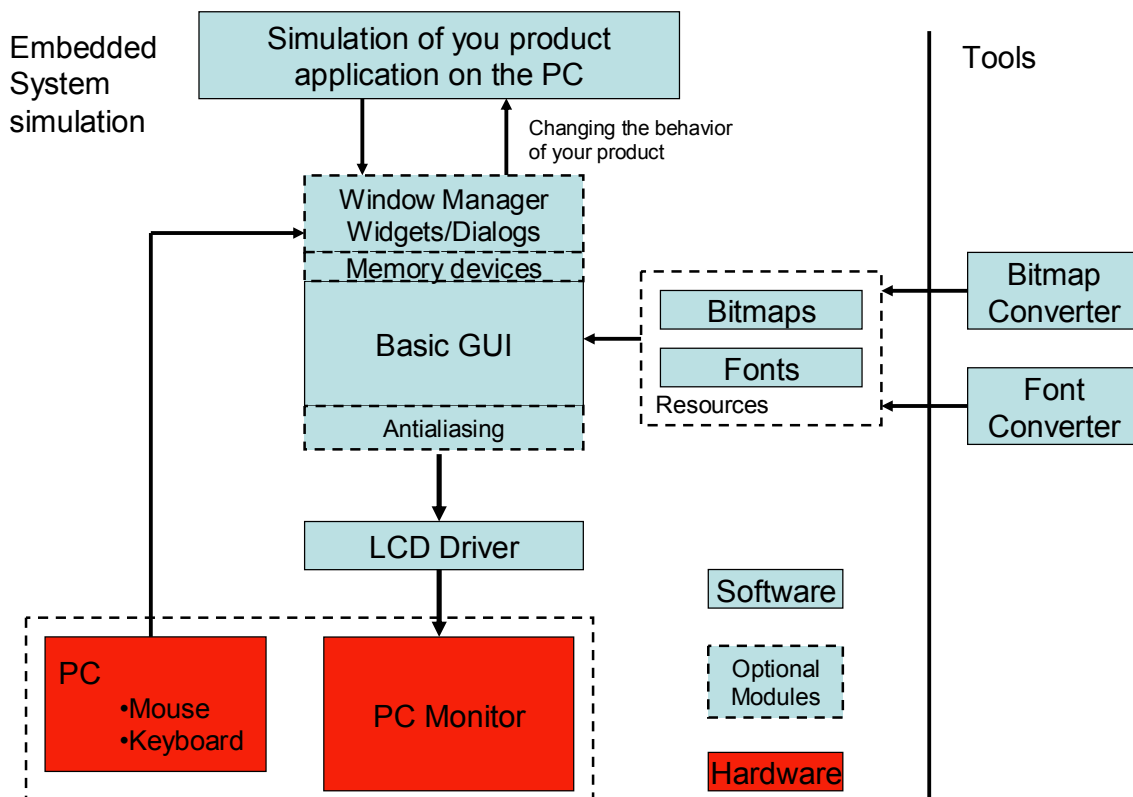


Figure 4 – Embedded System simulation

μ C/GUI even allows you to create a bitmap (a picture) of your product and test the actual product behaviour directly on your PC. The use of an optional background bitmap allows you to build a virtual prototype of your system as shown in Figure 5. Buttons on your product may be simulated. Used in conjunction with your display it makes the prototype even closer to the real thing.

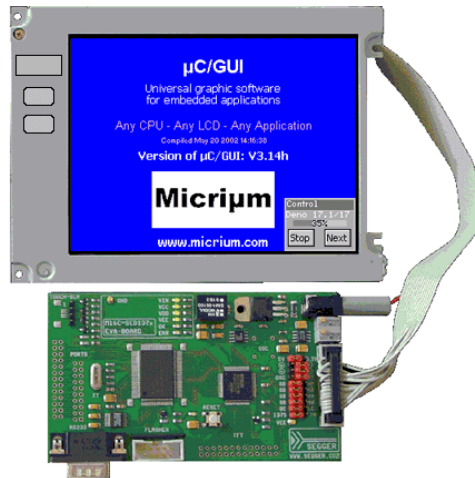


Figure 5 - Example of vending machine display simulated using a bitmap.

5.00 Conclusion

Developing a full featured GUI packages requires many man-years of development. As with any other software development effort, you may underestimate the task at hand and be tempted to develop your own package. We recommend that you obtain a commercially available product and concentrate on own value added features.

Commercial products provide proven reliability and portability. However, such products must be compatible and work with commercially available Real Time Operating Systems (RTOS). Specifically, OS-specific dependencies should be encapsulated and well documented. GUI vendors should also provide you with plenty of example code that runs on a number of embedded CPUs.

For an embedded product, you should select a GUI that is written in ANSI C that supports many LCD controllers.

The benefits to your customers of using a good user interface far outweigh the price of a GUI library.

References

μ C/OS-II, The Real-Time Kernel, 2nd Edition

Jean J. Labrosse

CMP Technical Books, 2002

ISBN 1-57820-103-9

Contacts

Micrium Technologies Corporation

949 Crestview Circle

Weston, FL 33327

954-217-2036

954-217-2037 (FAX)

e-mail: Christian.Legare@Micrium.com

WEB: www.Micrium.com

Micrium, Inc.

949 Crestview Circle

Weston, FL 33327

954-217-2036

954-217-2037 (FAX)

e-mail: Jean.Labrosse@Micrium.com

WEB: www.Micrium.com

CMP Books, Inc.

1601 W. 23rd St., Suite 200

Lawrence, KS 66046-9950

(785) 841-1631

(785) 841-2624 (FAX)

WEB: <http://www.rdbooks.com>

e-mail: rdorders@rdbooks.com

Disclaimer

Silicon Labs intends to provide customers with the latest, accurate, and in-depth documentation of all peripherals and modules available for system and software implementers using or intending to use the Silicon Labs products. Characterization data, available modules and peripherals, memory sizes and memory addresses refer to each specific device, and "Typical" parameters provided can and do vary in different applications. Application examples described herein are for illustrative purposes only. Silicon Labs reserves the right to make changes without further notice and limitation to product information, specifications, and descriptions herein, and does not give warranties as to the accuracy or completeness of the included information. Silicon Labs shall have no liability for the consequences of use of the information supplied herein. This document does not imply or express copyright licenses granted hereunder to design or fabricate any integrated circuits. The products are not designed or authorized to be used within any Life Support System without the specific written consent of Silicon Labs. A "Life Support System" is any product or system intended to support or sustain life and/or health, which, if it fails, can be reasonably expected to result in significant personal injury or death. Silicon Labs products are not designed or authorized for military applications. Silicon Labs products shall under no circumstances be used in weapons of mass destruction including (but not limited to) nuclear, biological or chemical weapons, or missiles capable of delivering such weapons.

Trademark Information

Silicon Laboratories Inc.®, Silicon Laboratories®, Silicon Labs®, SiLabs® and the Silicon Labs logo®, Bluegiga®, Bluegiga Logo®, Clockbuilder®, CMEMS®, DSPLL®, EFM®, EFM32®, EFR®, Ember®, Energy Micro, Energy Micro logo and combinations thereof, "the world's most energy friendly microcontrollers", Ember®, EZLink®, EZRadio®, EZRadioPRO®, Gecko®, ISOModem®, Precision32®, ProSLIC®, Simplicity Studio®, SiPHY®, Telegesis, the Telegesis Logo®, USBXpress® and others are trademarks or registered trademarks of Silicon Labs. ARM, CORTEX, Cortex-M3 and THUMB are trademarks or registered trademarks of ARM Holdings. Keil is a registered trademark of ARM Limited. All other products or brand names mentioned herein are trademarks of their respective holders.



Silicon Laboratories Inc.
400 West Cesar Chavez
Austin, TX 78701
USA

<http://www.silabs.com>