WHITE PAPER

TEXAS INSTRUMENTS

The Stellaris® Graphics Library Makes Short Order of Assembling a Dynamic HMI

Jean Anne Booth
General Manager, Stellaris® ARM® Cortex™-M MCUs
Sue Cozart
Applications Engineer

Introduction

A well-designed human-machine interface (HMI) can be a major selling point for an embedded electronic system. From medical instruments and consumer devices to industrial controls, a nice display with intuitive controls can combine style with functionality to make it easy for a user to set, monitor, and operate the equipment.

By making use of Texas Instruments' versatile Stellaris Graphics Library, implementing a human-machine interface (HMI) can be a breeze for the software programmer without requiring a detailed understanding of graphics rendering or touchscreen feedback. This paper describes and illustrates the capabilities of the Stellaris Graphics Library.

Once it is decided that a graphical user interface (GUI) will be used to monitor operation of a piece of equipment and be integrated into the electronic controls for operating that equipment, the human-machine interface must be programmed to run on the underlying processor. Graphical elements must be presented on the display screen and associated with control components. A more natural interface results when the display is designed well and the controls are tightly integrated.

Figure 1. *A 2.8" QVGA 16-bit color LCD resistive touch panel and the Stellaris Graphics Library comprises the human-machine interface on the Stellaris MDL-IDM28 Intelligent Display Module*

The Stellaris Graphics Library provides a ready-made set of graphics elements and integrated controls that can be called as functions with parameters unique to each use. With this library, the programmer can focus on the operation and control of the equipment being designed without getting distracted by graphics implementation details. The Graphics Library provides a wide variety of graphical elements including the supporting structure for the presentation and operation of the elements.

The Stellaris Graphics Library is an exclusive, royalty-free software package developed for the Texas Instruments' Stellaris family of 32-bit ARM® Cortex™-M3-based microcontrollers (MCUs). It is readily available from the Texas Instruments' web site and is included with Stellaris microcontroller development and display module reference design kits that have color displays, along with the Stellaris Peripheral Driver Library, fully documented and with C code source. The reference design kits make good use of the Graphics Library and are a great starting point for any application's HMI.

Check Out the Reference Library

The Stellaris Graphics Library is optimized to serve a variety of applications. Like most programs, hand-written assembly code designed for a specific application might run faster, consume less memory, and be more precise, but it might take five times as long to write, as much more time to debug, and then be difficult to change. Considering that the libraries can be licensed at no charge and used without paying royalties for Stellaris microcontrollers, 99% of programmers should be quite pleased with the capabilities of Graphics Library. A quick look at the Stellaris Graphics Library will demonstrate these capabilities.

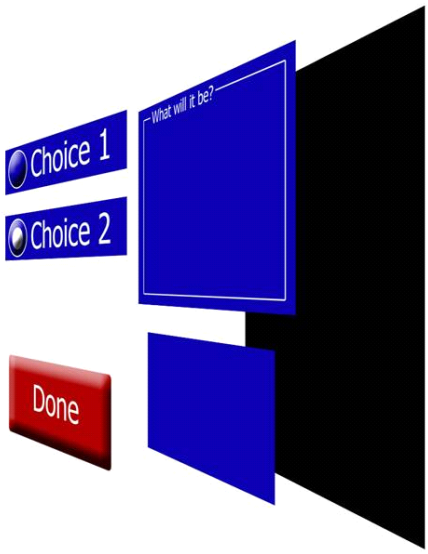


Figure 2. *A stack of virtual planes determine what is seen on the display and the feedback actions*

The Graphics Library builds from layers of display planes, starting with a base background (shown in Figure 2). The background simply establishes the default values of each pixel that are not otherwise set, and can be a solid color or any image that is provided. The nature of the other layers is determined by the programmer according to the desire for the interface. The layers establish a sequence for painting the eventual image on the display as well as (in the opposite direction) an order for the handling of inputs from the user.

Draw Many Shapes

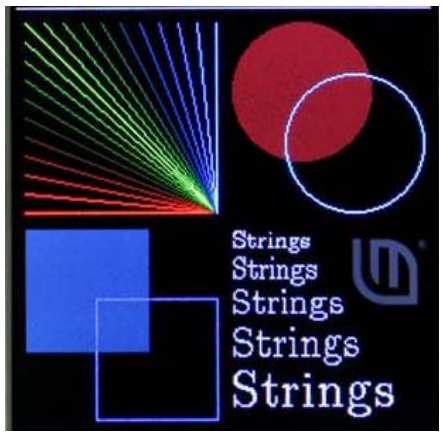


Figure 3. *Lower level graphics elements provide basic drawing operations*

There are a number of basic drawing operations available in the Graphics Library. These operations facilitate drawing points, lines, rectangles, and circles on the display. Shapes can be either filled or outlined. Text can be created in many sizes and bitmapped images can be displayed. Colors can be chosen from a wide selection. Clipping rectangle support allows more intricate graphics to be easily designed, such as rounded rectangle corners. Additionally, functions are available even at a low level to detect intersection and overlap of different graphics elements placed in the display. Many of the lower level graphics elements are shown in Figure 3 on page 2 and will be evident as the basis of the more sophisticated structures that are detailed throughout this document.

One of the goals of the Graphics Library is to offer both useful drawing elements and efficient processor performance and code density as well. Therefore, there is a uniform structure to the elements, compression techniques are used, and common features are found across many elements. Many characteristics are set up in a graphics context that helps give a common look to individual elements while minimizing memory requirements.

Widgets Add Life to the HMI

At the heart of a screen-based human-machine interface, especially one based on a touchscreen, are widgets. Widgets are the graphic elements presented on a screen that are tightly coupled with specific actions undertaken by the software, triggered by a designated touch on a touchscreen or an associated input device such as a physical button, switch, or sensor. One such example might be a graphic that looks like an upward-pointing arrow which, when touched, increases the speed of a motor.

A widget can perform output functions, such as displaying speed, level, or some other value, representing them graphically or in text. The graphic can change color or shape according to the value displayed, providing additional information to the operator such as alerts or extreme conditions. But the widget's most interesting feature is the input associated with it, especially when the input comes from the same screen on which the output is displayed; or at least it appears to come from the same place. The graphic's appearance might change just to provide the feedback that it was touched, such as by changing color. But more importantly, the widget can initiate a particular action in the software when that input is confirmed.

The Stellaris Graphics Library offers a variety of widgets to the programmer including push buttons, checkboxes, radio buttons, sliders, and list boxes. A canvas and a container provide backdrops, information, and isolation for more active widgets. Widgets can have parent, sibling, or child relationships with other widgets which determine hierarchy and processing sequences. Proper placement of widget graphics on the display using the aforementioned planes can visually demonstrate those relationships to the user.

Widgets
Canvas
Container
Push Button
Image Button
Checkbox
Radio Button
List Box
Slider

Like any good program, there is an established structure for working with widgets and graphical elements, including organization and specific procedures to follow to ensure the widgets behave as intended. Those procedures are described in the accompanying Graphics Library documentation and are only mentioned here as appropriate.

A Canvas to Paint On

A Canvas Widget is similar to a drawing surface. Like many graphic elements, it is rectangular and its dimensions and location on the display are specified by numerous parameters given by the programmer. The canvas can be outlined and/or filled with selected colors. It can display an image or contain text. Figure 4 shows some examples of canvases. Various canvases might be placed on a display. To aid in the clean appearance of the user interface, graphical elements and widgets can be constructed in a buffer memory which can later be loaded to the screen in a single operation. A number of screen buffers can be prepared and sent in sequence to the screen to simulate brief animation, within the limits of the system.

Figure 4. *A canvas gives drawing objects a home*

Put Things that Belong Together in a Container

A number of widgets can be placed on a Container Widget which primarily provides a visual grouping. The Container Widget can have a label and an outline to help visually show the one-of-many or any-of-many groupings. The top box in the middle plane of Figure 2 on page 2 illustrates a container widget with a label of “What will it be?” The Container Widget helps reinforce associations or interdependencies among widgets, although this is only through visual context. Figure 5 shows a variety of containers.

Figure 5. *Associated graphic objects and widgets can be placed in a container*

Scribble an Input

A freehand input capture tool can make an unusual contribution to an HMI. On its own it might just capture the initials of the service man or the signature for a credit card purchase with the help of a stylus. It might be used as an overlay to an image or other graphic elements and show user-provided connections or touches. In general, it can provide an electronic piece of paper that can be offered to the display and with the hand-made inputs stored in memory for later retrieval. Its use is limited only by the imagination of the system designer. There is no Scribble Widget, but adding a few elements to a Canvas Widget can make one. A simple form of a scribble pad is shown in Figure 6. It translates pressure on the touchscreen into painting of the associated pixels with a selected color, leaving a trace on the screen where it was touched. The application code can be called every

Figure 6. *Freehand input capture from the Scribble Application Example resembles handwriting*

time pressure on the screen changes so position on the screen can be marked. The screen can be cleared when desired or captured with a button push. “Ink color” can first be selected using button methods described next. While not formalized as a widget, StellarisWare has this scribble functionality implemented in a software example which serves as a good example of extending the functionality offered by the Stellaris Graphics Library.

Buttons Can Be Ornamental and Initiate Action

The Stellaris Graphics Library offers a number of button widgets in familiar styles that mimic the functionality of various types of physical push buttons. For example push buttons can be used to increase the speed of a motor, enter the digit “3,” and start an organized shutdown sequence. Buttons and switches are often used to perform such common actions as:

- Momentary connection (or disconnection) while depressed
- Toggle (change states) between ON and OFF with each press-and-release action
- Toggle between two connections
- Select one of many options
- Select any of many options
- Accept options as indicated and continue
- Initiate an action

Ultimately, all of these actions can be boiled down to just one function in software: initiate an action. The program code can then perform these and many other actions according to the present state and the needs of the system. Some buttons have an option for a combination momentary and toggle effect, repeatedly toggling as long as the button is pressed. This can be handy for ramping up a motor speed by 800 rpm rather than the 100 rpm a single push might normally give.

The concept of pressing a button in an electronic HMI is borrowed from a physical push button, although on a touchscreen, the physical action is more of just touching the screen (with some allowance for de-bouncing). One of the benefits of touchscreens is there is no mechanical button to wear out.

On the other hand, mechanical movement provides a tactile and possibly audible feedback that lets the operator know the button was pushed. For touchscreens, a separate audio output might be suggested, which can be more meaningful in that it can indicate that the software has begun to take action. The sound from some mechanical switches might not correlate to an electrical connection actually being made, and certainly doesn’t confirm that an operational action has been initiated. A touchscreen can also change the appearance of a virtual button depending on its state, although the finger might obscure the change and confirmation requires continuing to look at the screen.

Technically, in the Stellaris Graphics Library, pushing and releasing a button triggers a call to the application code. The type of button widget that is used affects some characteristics of button behavior as described in the next few sections, but the application code really determines the action that is taken with each button push.

The description of the Push Button Widget applies, for the most part, to other button types with variations noted under the other button descriptions.

Push a Button

The Push Button Widget is designed to replicate the intent of a physical push button: turning something ON or OFF, changing from one mode to another, or signaling that it is time to do something. Push buttons can be rectangular or circular or with clever use of images they can take on realistic appearances of any shape, with rounded edges, a three-dimensional look, and shadowing. The button can have text and/or an image centered as if on the object. Two sets of colors and images can be associated with each push button so the appearance of the button can be changed as it goes from being pressed to being released. The button can change from red to green when it is turned ON, or the image atop it can brighten as if backlit when in the ON position. Examples of the styles of push buttons possible are illustrated in Figure 7 as well as in later figures.

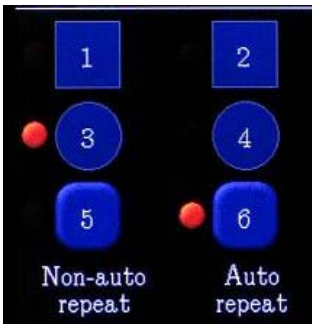


Figure 7. *Push buttons of many styles are possible*

The button makes use of an auto-repeat function that, when held down continuously, toggles from ON to OFF periodically after an initial delay. This makes “bumping up” a counter easy with a single button. Some form of feedback to the user is handy when using this feature so the user realizes the repeated action is taking place.

Of course, how the software routine for the operation of the button responds is left to the discretion of the programmer. The software routine might further interact with the graphical display, lighting what looks like an LED pilot light (as is done in Figure 7), presenting speed or other data readouts, or showing a completely different image. The routine might even transition to a completely different look on the screen, moving to the next stage of the process, the next menu, or a better configuration for the new state of the machine. As mentioned, some form of audio, such as clicks, tones, horns, or even mechanical vibration can add realism to a virtual touchscreen button and greatly improve the effectiveness of the human-machine interface being designed. Speech synthesis might make sense for some applications.

Choose Just One Station with Radio Buttons

Old car radios are the inspiration for the Radio Button Widget. What teenager in the 1970s could survive twenty minutes in the front seat of a car without changing the radio station ten times? Pushbuttons located below the radio dial gave the teenager a chance to check three stations before deciding on just one. Add a second teenager to the same car and an argument over which station to listen to can be assured. All technology comes with pros and cons.

A radio or television is only tuned to one station at a time and no combination of stations makes sense. Thus, radio buttons imply utilization for a one-of-many selection of mutually-exclusive choices. While options for low, medium, and high

might tempt someone to push both low and medium to get something in between, the use of radio buttons is designed to make that not possible.

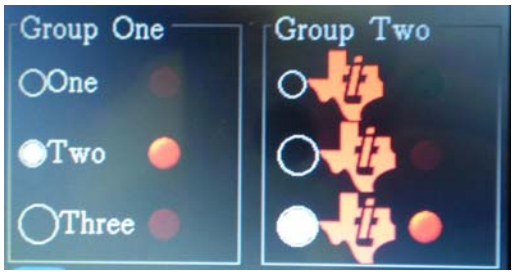


Figure 8. *Only one radio button can be selected at a time*

If one Radio Button Widget is used, then at least one other should also be associated with it. As many radio buttons as needed can be tied together, limited by the practicality of what fits nicely on the display screen. Radio buttons appear different than push buttons in that radio buttons look much simpler, like small circles, with text or an image label beside the button rather than on it. A radio button shows that it has been pushed by filling the button circle with its given color and style, but displaying only an empty circle (outline) when it is not selected. No specific change in the label reflects the state of the button. Figure 8 shows two sets of radio buttons, each with a different look on separate container widgets. The illusion of illuminating an LED to the right is performed by the application code.

Radio buttons must link with the other buttons in the group, and this is done through a parent-and-sibling relationship established through widget parameters. All radio buttons that are part of the same one-of-many selection group must be siblings; that is, children of the same parent. An input that sets any radio button triggers all other linked sibling radio buttons to clear themselves. Each radio button can run a routine upon being selected (set) or cleared. An auto-repeat operation would have no practical use in a one-of-many selection and, therefore, is not included for radio buttons.

Radio buttons are ideal for picking an operating mode, selecting a single option from many possibilities, and setting discrete values. Radio Button Widgets visibly enhanced on a Container Widget create a familiar selection mechanism for a display.

Check Out the Box

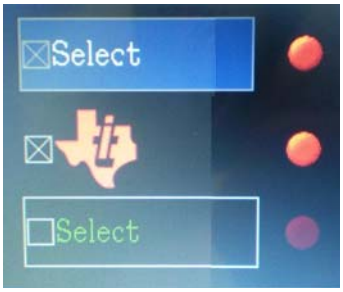


Figure 9. *The checkbox gives a simple way of selecting options*

A checkbox is primarily used to select an option or a number of options, although it can also directly initiate an action. The Checkbox Widget consists of a small square (or box) that displays an “X” in it when it is selected (“on”) or appears empty when not selected. There is an area next to the box for a label. The label can be either text or images for checkboxes with different backgrounds and outlines possible. Examples of checkboxes are shown in Figure 9. When checkboxes are used for any-of-many choices, the call triggered to the application program often simply logs the change and returns back to waiting for further user input. Frequently, a separate button must be pushed to tell the application that the final selection has been made

("Done," "OK," or "Submit") and to move on. Figure 2 on page 2 demonstrates this method.

Sophisticated Tools Come Easy

Some widgets in the Stellaris Graphics Library are not simply nice graphical elements but rather sophisticated in their operation. They are powerful HMI design features because they pack a lot of information and functionality in a small space yet behave in a familiar manner. Although the implementation can be complicated, TI has taken care of the underlying code; the programmer only has to provide parameters that give the graphic elements the desired look and that point to the application routines to run when a qualified input is received. There are even functions provided in the library that help the programmer construct lists, convert images, and otherwise make the best use of the graphic operations. The library has special features that makes it easy to design a single HMI that displays different words according to the specific language that has been selected, which is handy for equipment which can be used internationally.

Choose from a Long Laundry List

It takes a certain amount of space to describe each possible choice in an option list and space is often at a premium on a display. When there is a long list of items to choose from, a scrolling list can save a lot of space or at least clean up a display, with the slight downside that all the items are not visible at once.

The ListBox Widget in the Stellaris Graphics Library might be considered a selector tool to make one-of-many choices. The idea should be familiar since it is used in numerous computer menus, web pages, electronic forms, database entries, and drawing programs. This is not a "drop-down" form of menu or list, but an always-visible window that shows a small portion of the entire list of possible choices. The user can scroll up and down the list looking for the desired item which is then "clicked" to select it. A dragging motion beyond either end of the list initiates the scrolling. The list can be as long as needed and be organized in any way the programmer wishes, because each item links to the next item in the list. The list can be somewhat dynamic, allowing entries be added and removed as the application runs and conditions change. Ultimately, the ListBox Widget initiates an action in the application program when an item in the list is selected.

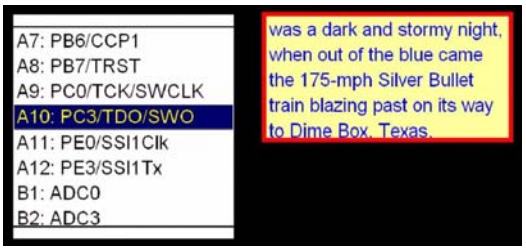


Figure 10. *List boxes select one-of-many (left) and scrolling text boxes (right)*

List boxes can also be used as a sort of scrolling text output window. With limited space, long dialogs are difficult to display on a fairly static screen. However, because a list box presents a window containing sequential text, a little creativity can turn a list box into a moving newspaper column for instructions, procedures, explanations, or about anything else long strings of text can convey. Being dynamic, when configured correctly, a list box can display the progress of a program, recurring data readouts, and real-time instructions based on current

conditions. Because there is a fair amount of control over the layout, color, and presentation of a list box, the user might never realize this text window is just a repurposed list box. Figure 10 on page 8 show the two uses of a list box including the scrolling list (on the left) and the scrolling text in a box (on the right).

Sliders Fit It All in One Tight Space

One of the most unique and useful widgets in a human-machine interface is the slider. Resembling the linear volume controls seen on a mixing board or master control panel in a sound studio, the virtual slider has all of the qualities of such physical controls but offers additional flexibility. A slider can be a combination input and output device, giving a real, live visual indicator of the current output and then with a touch or two on the display, the user can change settings as needed.

One of the characteristics of the slider that makes it such a valuable HMI feature is that the current setting is shown in relation to the lowest and the highest possible settings. The button or handle is slid closer to the maximum to increase the associated value (volume, speed, temperature…), or slid back to the minimum end to decrease it, both being natural motions. In some senses, the slider gives an analog appearance, although we all know its actions are backed by digital interpretations. To increase the speed, for instance, a person might just slide the lever up a little or to the right; precise numbers might not be important, just “more” or “less.” This “fuzzy logic” gives a good human aspect to the human-machine interface.

The Slider Widget included in the Stellaris Graphics Library is one of the more configurable controls. Sliders can be oriented either horizontally or vertically, with the maximum end of the scale to the right or top. Along with the usual outline and fill colors for the widget, there is one additional color. The normal foreground color fills only the lower part of the slider that is proportional to the current value compared to the range of the specified minimum to maximum (see Figure 11 on page 9). A background color fills the balance of the graphic to show the remaining range represented, resembling the look of an old-fashioned mercury thermometer.

Figure 11. *The slider is a superb multi-purpose graphical HMI device*

To enhance the fill color distinction of the slider position, text and/or an image can be associated with the foreground. Different text, color, and an additional image can be associated with the background as well, further highlighting the differences. As the slider is moved, the text can transition from the background text to the foreground text or reveal the foreground image while covering more of the background image, and vice versa.

Of course, the slider is excellent for touchscreen HMIs. The Slider Widget correlates a touch on the screen with the current slider position and both end-points and then determines the new value within the specified range. The slider image is updated to reflect the new value which is also included with an appropriate trigger to the application code. It is easy to tell the widget to go to the minimum or maximum value by running off the end of the slider. The Slider Widget also accepts inputs from the application code to keep its displayed image current with data that is changing due to reasons besides touchscreen input.

Because the slider is dynamic, analog, and programmable, it is useful for the programmer to consider displaying elsewhere readouts of, say, digital representations of the end-points and current set point, and maybe an actual reading (it can take some time for the motor speed to change from 2500 rpm to 3000 rpm or the temperature to rise from 25°C to 30°C, so the set value and the actual might differ). Also, it might be desirable to expand, contract, raise, or lower the displayed range of the slider as one “zooms in” (or up). Changing colors can also visually indicate changes or extremes of the range. By itself, and enhanced with additional displays, the slider offers many opportunities to both display readings from the electronic equipment in a readily-comprehended manner, and to take input from the user for making changes, all using the same graphical display. Other slider examples can be seen elsewhere in this document and documentation for Stellaris MCU reference design kits.

Putting It All Together

A number of characteristics are common across many graphical elements and widgets. In fact this commonality is good, establishing a look, style, and feel for the human-machine interface when the different elements are combined, contributing to the intuitive nature of a good HMI.

Letters, Colors, and Parameters Build a Style

The Stellaris Graphics Library makes use of a set of standard text fonts and sizes as well as some standard colors for all of the drawing objects. However, the designer is free to customize the given sets as well.

Most of the fonts available are a proportional font called Computer Modern which has been pre-converted to bitmaps for use on the desired display. Font sizes from 12 to 48 are available, with a normal, bold, italic, small caps, and san serif version available at each even point size. A basic fixed-spacing 6x8 font is also provided. Examples of the provided typefaces are shown in Figure 12. When used with most widgets, a choice of opaque or transparent text can be made in case the background obscures the text characters.

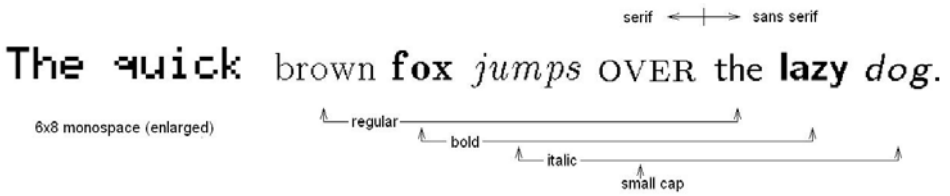


Figure 12. *Many typefaces are available for ASCII characters*

A broad variety of colors are available in the Stellaris Graphics Library with black and shades of gray for the simplest displays to essentially any combination of red-green-blue (RGB) intensities. Figure 13 shows a few of the colors from the standard predefined palette including basic colors such as blue and red, but also a variety of advanced colors are also available including Papaya Whip, Gainsboro, and Peru. Most graphics objects can have distinct foreground, text, and background colors. While the graphics software utilizes full 24-bit color, each system's device drivers translate colors according to the display hardware being used.

Figure 13. *Samples of predefined colors available for drawing objects*

Most of the widgets have a number of other parameters as well that add pizzazz or flexibility to them. Shapes, borders, line widths, and backgrounds can emphasize objects as needed. These widgets are programmable, so application code can give the graphics a more dynamic character. Colors, text, sizes, and styles can be changed by the application when the state of the widget changes (button pressed or button released) to aid the operator in recognizing the widget's state.

Customize to Your HMI's Content

The Graphics Library even has some special functions that allow designers to easily convert company colors from RGB, favorite typefaces, established icons and corporate logos to a format that can be used with the Graphics Library functions.

The graphics primitives, widgets, framework, examples, and uses described here are only the beginning. Not all the available functions and options have been detailed here. Still, the possibilities are virtually endless. The Stellaris Graphics Library should accommodate the needs of most graphically-oriented touchscreen human-machine interfaces. Great flexibility is available simply by changing the established parameters for the different functions.

For the more discerning and adventuresome, the source code for the libraries is provided and can easily be modified and expanded to fit any need. Simple changes or major reworks of the fundamental C code can be readily made. The Graphics

Library is intended to be easy to use and can be viewed as a starting point for more application-specific widgets the designer creates. Do remember, however, that the simplest and most intuitive interface is the most effective, so be cautious to avoid creeping complexity.

Combine Widgets to Form an HMI

An application's HMI is most likely to be built from a collection of graphical elements and widgets like those the Stellaris Graphics Library supports. The HMI might involve a sequence of screens that guide the user down an initialization path. Some screens might be presented to monitor the equipment when it is running. Still other screens can be displayed for changing ongoing operating settings. During maintenance, a series of screens might be used to display status and operational details of the equipment or to pull up error logs. Each of these screens might use a combination of push buttons, radio buttons, list boxes, sliders, pictorial images, text instructions, and other graphical elements borrowed from the Graphics Library.

The Stellaris Graphics Library supports any number of its graphics elements and widgets to be displayed, as screen space permits. Canvases and containers provide visual support to groupings while layering of planes can enforce structure. Illustrations made up of line drawings can be displayed with appropriate buttons. Images taken from more-formal documentation can be easily converted for widget use. Scales can be added to sliders, just by adding some lines and text beside the slider.

However, usability should be balanced with simplicity, keeping each screen as clean yet complete as possible for the most effective HMI. Figure 14 on page 13 shows some combinations of widgets and graphical elements used for the touchscreen interface on some Stellaris MCU reference design kits. The last two images show how one might replicate an oscilloscope, a familiar instrument in any electronics laboratory.

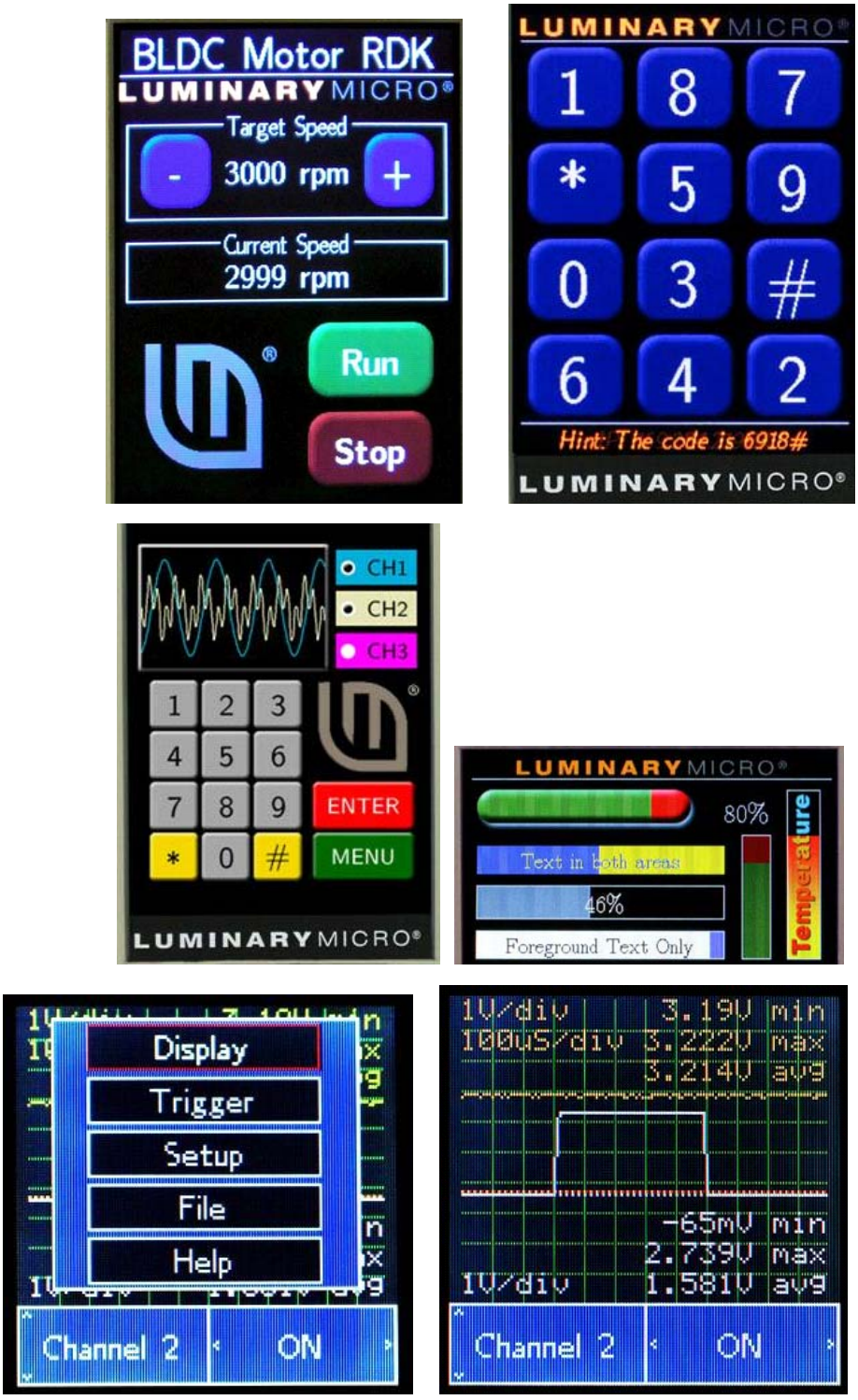


Figure 14. *HMIs used in Stellaris reference design kits demonstrate good use of multiple widgets, text, and graphic elements*

Widgets are Highly Configurable

The widgets of the Stellaris Graphics Library lay a foundation for a sophisticated human-machine interface. The library is well documented, examples of its components are illustrated in this paper, and the widgets can be seen in action in numerous Stellaris MCU development kits and reference design kits. The widgets in the library are highly configurable as-is, with numerous variables that the programmer sets so the widget presents the graphic images precisely as the programmer wishes. To give better insight into the flexibility of the widgets as they come in the Graphics Library, Figure 15 on page 15 shows many variable names used by the Slider Widget and indicates the characteristic widget that is affected by each. Other widgets have similar parameters that define their associated graphic.

Library Makes Easy Touchscreen Controls

A modest human-machine interface implemented on a touchscreen can make an effective monitoring station and control center for very small or fairly substantial electronic systems. When using a Stellaris ARM-Cortex-M3-based microcontroller to run the system, some well-defined graphics primitives and applications widgets from the Stellaris Graphics Library can be leveraged to quickly and easily construct a visually appealing display and control center by merely specifying a few parameters (see Figure 15 on page 15 for examples).

A number of HMI examples have been shown in this paper, with others included in various Stellaris development kits, which demonstrate the library's potential. With this little knowledge and a few ideas about the requirements of the application, while keeping in mind the importance of simplicity, a system designer should be able to quickly assemble a useful HMI. For the more ambitious designer, complete C source code is available so the library's off-the-shelf capabilities can be expanded and customized to fit the needs of essentially any application.

Additional References

In addition to this document, the following documents and software are also available for download at www.ti.com/stellaris:

- *StellarisWare Graphics Library User's Guide*
- Stellaris Graphics Library Standalone Package
- StellarisWare Driver Library Standalone Package
- *Access Your Embedded Controller with Ease through a Web Server White Paper*

Texas Instruments • 108 Wild Basin, Suite 350 • Austin, TX 78746
http://www.ti.com/stellaris

Widget Style Example

The following example shows two different instances of the slider widget and describes how the values set in the instance data structure (tSliderWidget) affect the appearance of the control. All slider widgets consist of two areas:

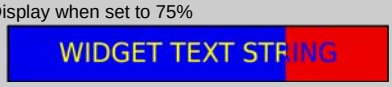
- The “foreground” area is on the left of a horizontal slider or the bottom of a vertical slider. Its size represents the current slider value.
- The “background” area is on the right of a horizontal slider or the top of a vertical slider and occupies the (100 - value)% of the area of the slider.

Text-Based Slider

A text-based slider allows you to draw a single text string in the center of the widget with an optional background. Text and background colors are chosen independently for the two portions of the slider, “foreground” and “background.”

Display when set to 25%

Display when set to 75%



Foreground Area Background Area

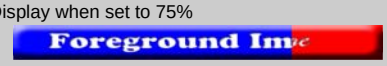
Field	Value	Note
ulStyle	SL_STYLE_OUTLINE	Draw a 1 pixel border around the widget using ulOutlineColor.
	SL_STYLE_FILL	Fill the “foreground” area of the widget with ulFillColor.
	SL_STYLE_BACKG_FILL	Fill the “background” area of the widget with ulBackgroundFillColor.
	SL_STYLE_TEXT	Draw the text string pointed to by pcText in color ulTextColor over the “foreground” area of the widget
	SL_STYLE_BACKG_TEXT	Draw the text string pointed to by pcText in color ulBackgroundTextColor over the “background” area of the widget
ulFillColor	ClrBlue	Set the color of the “foreground” area of the widget to blue.
ulBackgroundFillColor	ClrRed	Set the color of the “background” area of the widget to red.
ulOutlineColor	ClrBlack	Set the widget outline color to black.
ulTextColor	ClrYellow	Draw “foreground” text in yellow.
ulBackgroundTextColor	ClrBlue	Draw “background” text in blue.
pFont	A pointer to your favorite grlib font	This is the font that will be used to draw the text string.
pcText	A pointer to “WIDGET TEXT STRING”	This points to the string which will be drawn, centered, within the widget.
pucImage	NULL	Ignored since SL_STYLE_IMG is not specified.
pucBackgroundImage	NULL	Ignored since SL_STYLE_BACKG_IMG is not specified.

Image-Based Slider

An image-based slider offers greater display flexibility than a text-based widget at the cost of increased memory usage. In this case, the “foreground” and “background” portions of the display are taken from two images stored in the application. These must be the same size and cover the full area of the slider. The images shown here were created using Adobe Photoshop, saved as GIF images then converted to the correct format for the Stellaris Graphics Library using the open-source “gif2pnm” tool followed by the “pnmtoc” tool included in StellarisWare.

Display when set to 25%

Display when set to 75%



Foreground Area Background Area

Field	Value	Note
ulStyle	SL_STYLE_IMG	Draw the “foreground” area of the widget using the image pointed to by pucImage.
	SL_STYLE_BACKG_IMG	Draw the “background” area of the widget using the image pointed to by pucBackgroundImage.
ulFillColor	0	Ignored since SL_STYLE_FILL is not specified.
ulBackgroundFillColor	0	Ignored since SL_STYLE_BACKG_FILL is not specified
ulOutlineColor	0	Ignored since SL_STYLE_OUTLINE is not specified
ulTextColor	0	Ignored since SL_STYLE_TEXT is not specified.
ulBackgroundTextColor	0	Ignored since SL_STYLE_BACKG_TEXT is not specified.
pFont	NULL	Ignored since neither SL_STYLE_TEXT nor SL_STYLE_BACKG_TEXT is specified.
pcText	NULL	Ignored since neither SL_STYLE_TEXT nor SL_STYLE_BACKG_TEXT is specified.
pucImage	A pointer to the data for the foreground image.	
pucBackgroundImage	A pointer to the data for the background image.	

Incorporating the text into the slider images provides you the option of using any font and effect at your disposal but ties the image to one specific use (described by the text). In some circumstances it is desirable to use a “blank” image and draw the text dynamically. This allows the same background and foreground images to be used for multiple sliders, each of which will still show different text. To achieve this, add SL_STYLE_TEXT and SL_STYLE_BACKG_TEXT to the ulStyle value and set ulTextColor, ulBackgroundTextColor, pFont, and pcText appropriately.

Figure 15. *A glimpse into the Stellaris Graphics Library: a widget style example*

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

TI products are not authorized for use in safety-critical applications (such as life support) where a failure of the TI product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of TI products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by TI. Further, Buyers must fully indemnify TI and its representatives against any damages arising out of the use of TI products in such safety-critical applications.

TI products are neither designed nor intended for use in military/aerospace applications or environments unless the TI products are specifically designated by TI as military-grade or "enhanced plastic." Only products designated by TI as military-grade meet military specifications. Buyers acknowledge and agree that any such use of TI products which TI has not designated as military-grade is solely at the Buyer's risk, and that they are solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI products are neither designed nor intended for use in automotive applications or environments unless the specific TI products are designated by TI as compliant with ISO/TS 16949 requirements. Buyers acknowledge and agree that, if they use any non-designated products in automotive applications, TI will not be responsible for any failure to meet such requirements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products		**Applications**	
Audio	www.ti.com/audio	Communications and Telecom	www.ti.com/communications
Amplifiers	amplifier.ti.com	Computers and Peripherals	www.ti.com/computers
Data Converters	dataconverter.ti.com	Consumer Electronics	www.ti.com/consumer-apps
DLP® Products	www.dlp.com	Energy and Lighting	www.ti.com/energy
DSP	dsp.ti.com	Industrial	www.ti.com/industrial
Clocks and Timers	www.ti.com/clocks	Medical	www.ti.com/medical
Interface	interface.ti.com	Security	www.ti.com/security
Logic	logic.ti.com	Space, Avionics and Defense	www.ti.com/space-avionics-defense
Power Mgmt	power.ti.com	Transportation and Automotive	www.ti.com/automotive
Microcontrollers	microcontroller.ti.com	Video and Imaging	www.ti.com/video
RFID	www.ti-rfid.com	Wireless	www.ti.com/wireless-apps
RF/IF and ZigBee® Solutions	www.ti.com/lprf		
	TI E2E Community Home Page		e2e.ti.com

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265