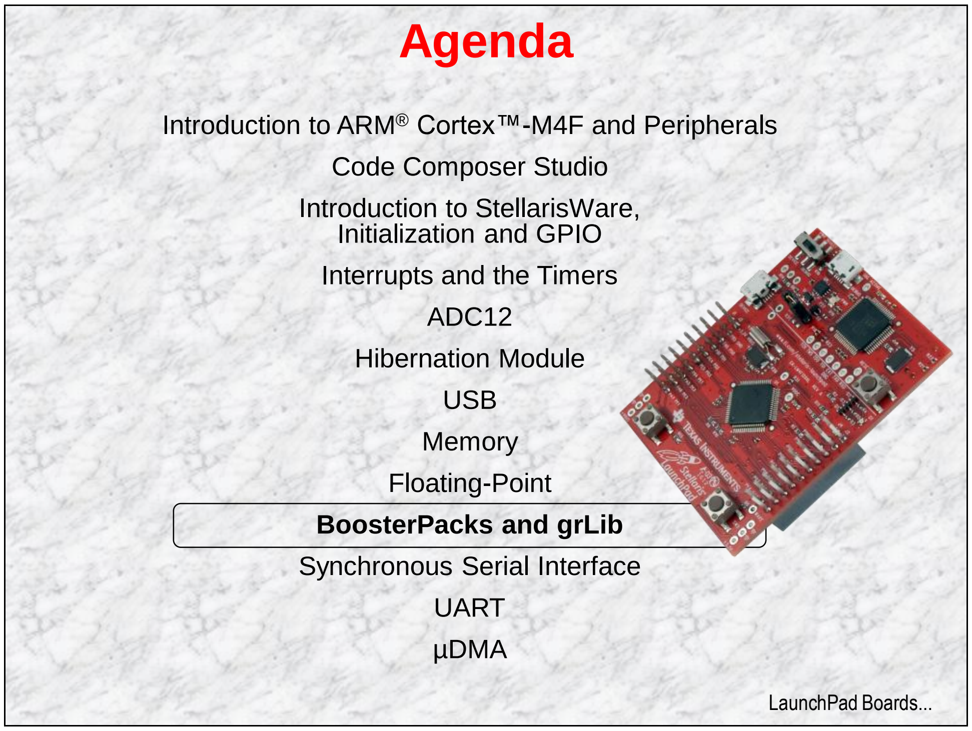
Introduction

This chapter will take a look at the currently available BoosterPacks for the LaunchPad board. We’ll take a closer look at the Kentec Display LCD TouchScreen BoosterPack and then dive into the StellarisWare graphics library.



Agenda

Introduction to ARM® Cortex™-M4F and Peripherals
Code Composer Studio
Introduction to StellarisWare, Initialization and GPIO
Interrupts and the Timers
ADC12
Hibernation Module
USB
Memory
Floating-Point
BoosterPacks and grLib
Synchronous Serial Interface
UART
µDMA

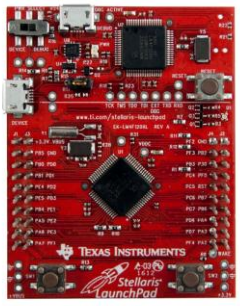
LaunchPad Boards...

Chapter Topics

LaunchPad Boards and BoosterPacks

TI LaunchPad Boards

MSP430
$9.99US



Stellaris
$12.99US

C2000 Piccolo
$17.00US

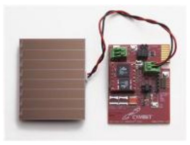
BoosterPack Connectors...

BoosterPack Connectors

- **Original Format (MSP430)**
  - VCC and Ground
  - 14 GPIO
  - Emulator Reset and Test
  - Crystal inputs or 2 more GPIO
- **XL Format (Stellaris/C2000) is a superset of the original, adding two rows of pins with:**
  - USB V_{BUS} and Ground
  - 18 additional GPIO

Available Boosterpacks...

Some of the Available BoosterPacks


Solar Energy Harvesting

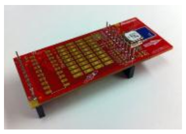
RF Module w/ LCD

Olimex 8x8 LED Matrix

TMP006 IR Temperature Sensor

Universal Energy Harvesting

Inductive Charging


Sub-1GHz RF Wireless

C5000 Audio Capacitive Touch

Capacitive Touch

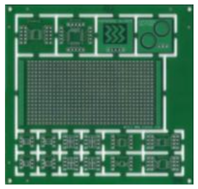
Proto Board

TPL0401 SPI Digital Pot.

TPL0501 SPI Digital Pot.

Available Boosterpacks...

Some of the Available BoosterPacks

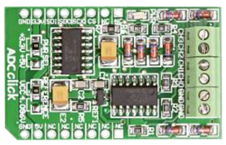

Proto board

ZigBee Networking

OLED Display

LCD Controller Development Package

MOD Board Adapter


Click Board Adapter

Kentec LCD Display...

See http://e2e.ti.com/group/msp430launchpad/b/boosterpacks/default.aspx for a list of TI boosterpacks.

Solar Energy Harvesting:
http://e2e.ti.com/group/msp430launchpad/b/boosterpacks/archive/2011/06/08/cymbet-enerchip-cc-solar-energy-harvesting-evaluation-kit-cbc-eval-10.aspx

Universal Energy Harvesting:
http://e2e.ti.com/group/msp430launchpad/b/boosterpacks/archive/2011/06/08/cymbet-enerchip-ep-universal-energy-harvesting-evaluation-kit-cbc-eval-09.aspx

Capacitive Touch:
http://e2e.ti.com/group/msp430launchpad/b/boosterpacks/archive/2011/04/17/430boost_2d00_sense1.aspx

RF Module w/ LCD:
http://e2e.ti.com/group/msp430launchpad/b/boosterpacks/archive/2011/07/13/golden-ic-rf-module-with-lcd-boosterpack.aspx

Inductive Charging:
http://e2e.ti.com/group/msp430launchpad/b/boosterpacks/archive/2011/06/08/cymbet-enerchip-ep-universal-energy-harvesting-evaluation-kit-cbc-eval-11.aspx

Proto Board:
http://joesbytes.com/10-ti-msp430-launchpad-mini-proto-board.html

Olimex 8x8 LED Matrix:
http://e2e.ti.com/group/msp430launchpad/b/boosterpacks/archive/2011/09/07/8x8-led-matrix-boosterpack-from-olimex.aspx

Sub-1GHz Wireless:
http://e2e.ti.com/group/msp430launchpad/b/boosterpacks/archive/2011/12/01/texas-instruments-sub-1ghz-rf-wireless-boosterpack-430boost-cc110l.aspx

TPL0401 SPI Digital Potentiometer:
http://e2e.ti.com/group/msp430launchpad/b/boosterpacks/archive/2011/11/18/texas-instruments-tpl0401-based-i2c-digital-potentiometer-tpl0401evm.aspx

TMP006 IR Temperature Sensor:
http://www.ti.com/tool/430boost-tmp006

C5000 Audio Capacitive Touch:
http://e2e.ti.com/group/msp430launchpad/b/boosterpacks/archive/2012/03/27/texas-instruments-c5000-audio-capacitive-touch-boosterpack-430boost-c55audio1.aspx

TPL0501 SPI Digital Potentiometer:
http://e2e.ti.com/group/msp430launchpad/b/boosterpacks/archive/2011/11/18/texas-instruments-tpl0501-based-spi-digital-potentiometer-tpl0501evm.aspx

Proto Board:
http://store-ovhh2.mybigcommerce.com/ti-booster-packs/

LCD Controller Development Package:
http://www.epson.jp/device/semicon_e/product/lcd_controllers/index.htm

ZigBee Networking:
http://www.anaren.com/

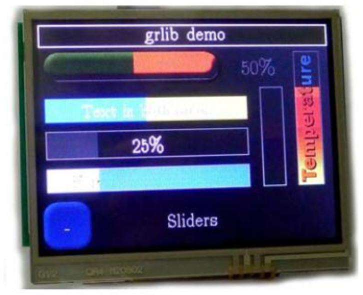
MOD Board adapter:
https://www.olimex.com/dev/index.html

OLED Display:
http://www.kentecdisplay.com/plus/view.php?aid=74

Click Board Adapter:
http://www.mikroe.com/eng/categories/view/102/click-boards/

KenTec TouchSceen TFT LCD

KenTec TouchScreen TFT LCD Display

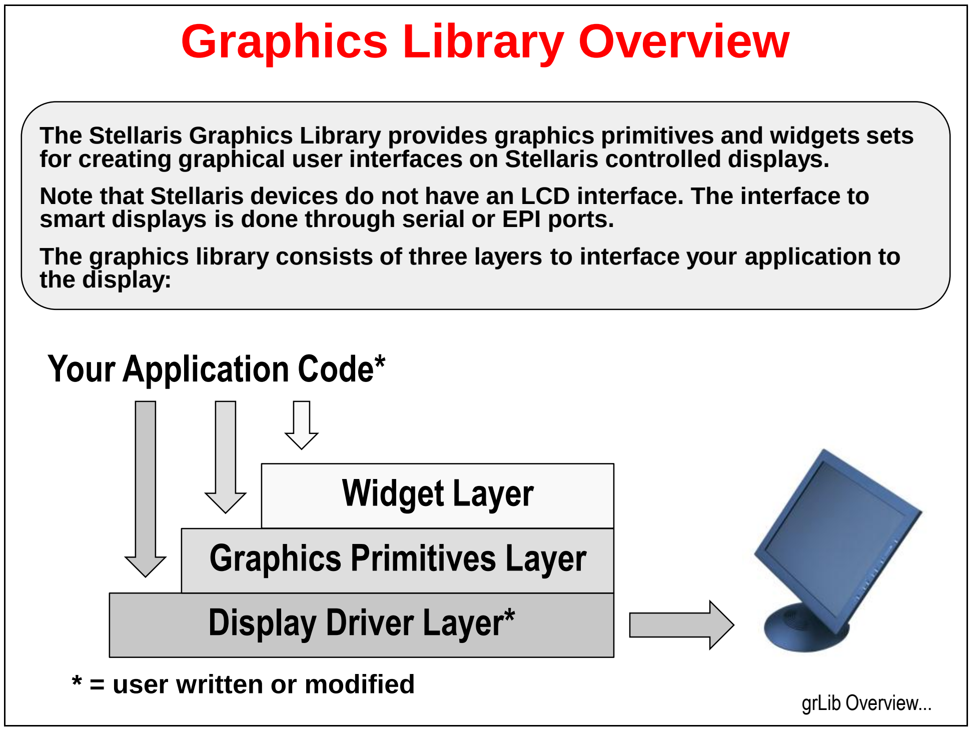


- Part# EB-LM4F120-L35
- Designed for XL BoosterPack pinout
- 3.5" QVGA TFT 320x240x16 color LCD with LED backlight
- Driver circuit and connector are compatible with 4.3", 5", 7" & 9"displays
- Resistive Touch Overlay

grLib Overview...

For more information go to: http://www.kentecdisplay.com/

Graphics Library

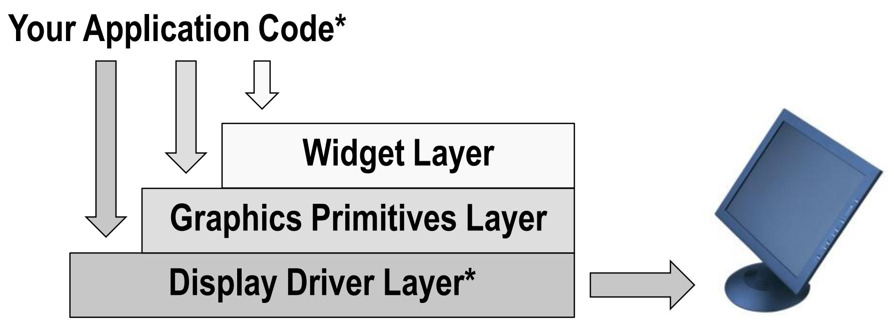


Graphics Library Overview

The Stellaris Graphics Library provides graphics primitives and widgets sets for creating graphical user interfaces on Stellaris controlled displays.

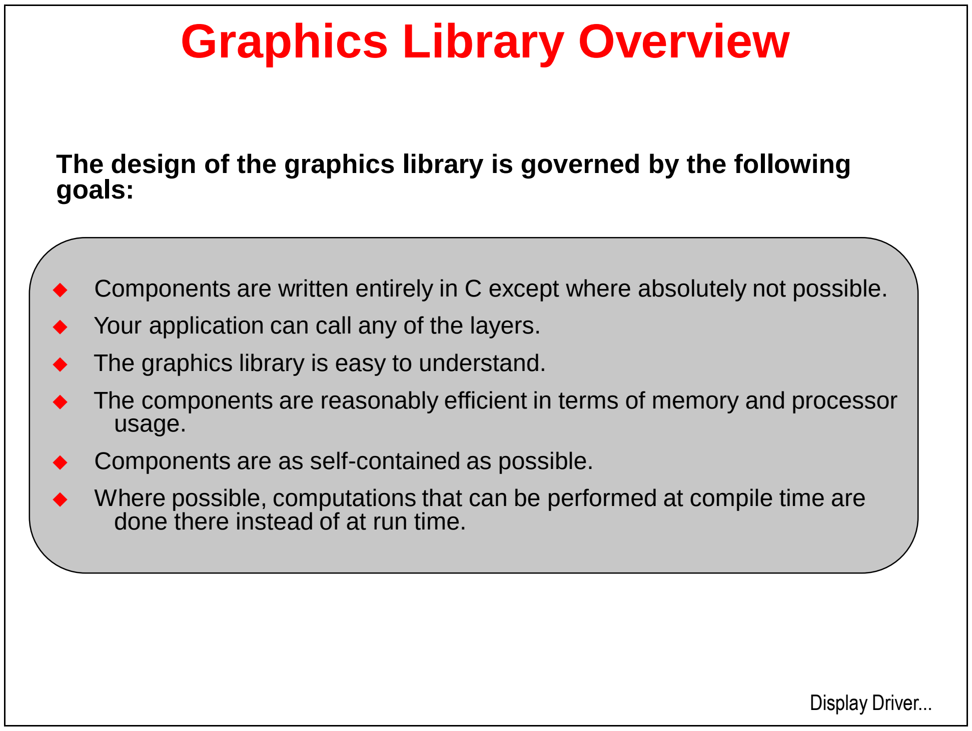
Note that Stellaris devices do not have an LCD interface. The interface to smart displays is done through serial or EPI ports.

The graphics library consists of three layers to interface your application to the display:



* = user written or modified

grLib Overview...



Graphics Library Overview

The design of the graphics library is governed by the following goals:

- Components are written entirely in C except where absolutely not possible.
- Your application can call any of the layers.
- The graphics library is easy to understand.
- The components are reasonably efficient in terms of memory and processor usage.
- Components are as self-contained as possible.
- Where possible, computations that can be performed at compile time are done there instead of at run time.

Display Driver...

Display Driver

Low level interface to the display hardware

Routines for display-dependent operations like:

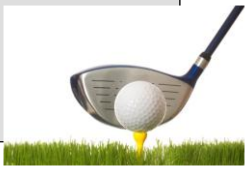
- Initialization
- Backlight control
- Contrast
- Translation of 24-bit RGB values to screen dependent color map

Drawing routines for the graphics library like:

- Flush
- Line drawing
- Pixel drawing
- Rectangle drawing

User-modified Hardware Dependent Code

- Connectivity of the smart display to the LM4F
- Changes to the existing code to match your display (like color depth and size)



Graphics Primitives...

This document: http://www.ti.com/lit/an/spma039/spma039.pdf has suggestions for modifying the display driver to connect to your display.

Graphics Primitives

Low level drawing support for:

- Lines, circles, text and bitmap images
- Support for off-screen buffering
- Foreground and background drawing contexts
- Color is represented as a 24-bit RGB value (8-bits per color)
  - ~150 pre-defined colors are provided
- 153 pre-defined fonts based on the Computer Modern typeface
- Support for Asian and Cyrillic languages

Widgets...

Widget Framework

- Widgets are graphic elements that provide user control elements

- Widgets combine the graphical and touch screen elements on-screen with a parent/child hierarchy so that objects appear in front or behind each other correctly

Canvas – a simple drawing surface with no user interaction

Checkbox – select/unselect

Container – a visual element to group on-screen widgets

Push Button – an on-screen button that can be pressed to perform an action

Radio Button – selections that form a group; like low, medium and high

Slider – vertical or horizontal to select a value from a predefined range

ListBox – selection from a list of options

Special Utilities...

Special Utilities

Utilities to produce graphics library compatible data structures

ftrasterize

- Uses the FreeType font rendering package to convert your font into a graphic library format.
- Supported fonts include: TrueType®, OpenType®, PostScript® Type 1 and Windows® FNT.

lmi-button

- Creates custom shaped buttons using a script plug-in for GIMP. Produces images for use by the pushbutton widget.

pnmtoc

- Converts a NetPBM image file into a graphics library compatible file.
- NetPBM image formats can be produced by: GIMP, NetPBM, ImageMajik and others.

mkstringtable

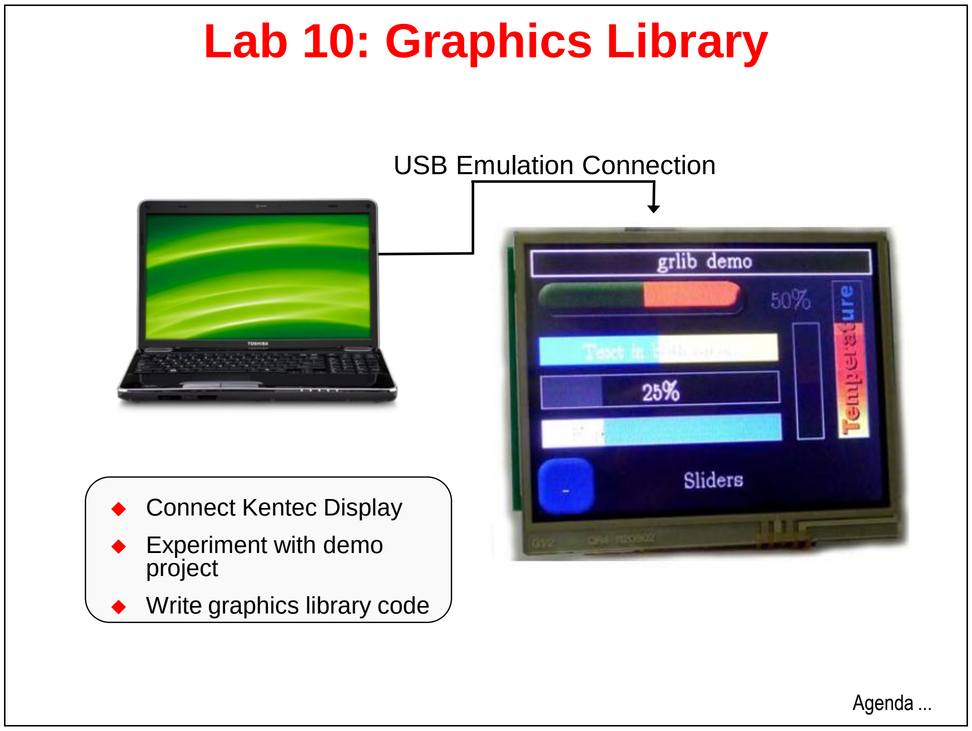
- Converts a comma separated file (.csv) into a table of strings usable by graphics library for pull down menus.

Lab...

Lab 10: Graphics Library

Objective

In this lab you will connect the KenTec display to your LaunchPad board. You will experiment with the example code and then write a program using the graphics library.



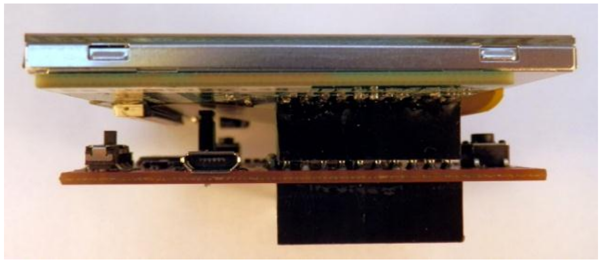
Lab 10: Graphics Library

- Connect Kentec Display
- Experiment with demo project
- Write graphics library code

Procedure

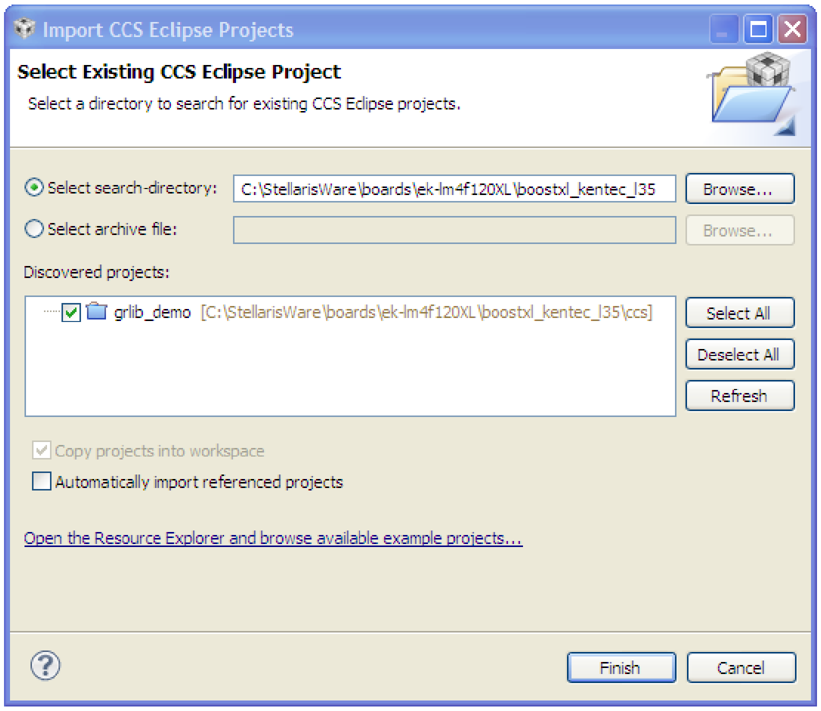
Connect the KenTec Display to your LaunchPad Board

1. Carefully connect the KenTec display to your LaunchPad board. Note the part numbers on the front of the LCD display. Those part numbers should be at the end of the LaunchPad board that has the two pushbuttons when oriented correctly. Make sure that all the BoosterPack pins are correctly engaged into the connectors on the bottom of the display.



Import Project

2. We're going to use the Kentec example project provided by the manufacturer. Maximize Code Composer and click Project → Import Existing CCS Eclipse Project. Make the settings shown below and click Finish. Note that this project will be automatically copied into your workspace.



Import CCS Eclipse Projects

Select Existing CCS Eclipse Project

Select a directory to search for existing CCS Eclipse projects.

(●) Select search-directory: C:\StellarisWare\boards\ek-lm4f120XL\boostxl_kentec_l35 [Browse...]

( ) Select archive file: [Browse...]

Discovered projects:

[✓] grlib_demo [C:\StellarisWare\boards\ek-lm4f120XL\boostxl_kentec_l35\ccs]

[✓] Copy projects into workspace

[ ] Automatically import referenced projects

Open the Resource Explorer and browse available example projects...

3. Expand the project in the Project Explorer pane, and then expand the **drivers** folder. The two files in this folder; `Kentec320x240x16_ssd2119_8bit.c` and `touch.c` are the driver files for the display and the touch overlay. Open the files and take a look around. Some of these files were derived from earlier StellarisWare examples, so you may see references to the DK-LM3S9B96 board.

   `Kentec320x240x16_ssd2119_8bit.c` contains the low level Display Driver interface to the LCD hardware, including the pin mapping, contrast controls and simple graphics primitives.

Build, Download and Run the Demo

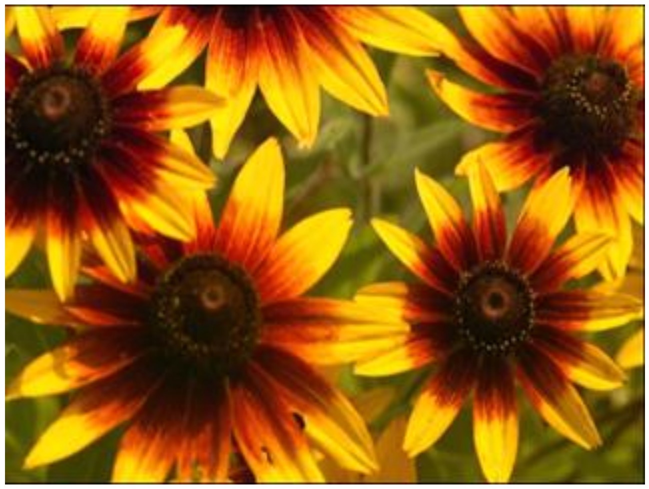
4. Make sure your board is connected to your computer, and then click the Debug button to build and download the program to the LM4F120H5QR device. The project should build and link without any warnings or errors.

5. Watch your LCD display and click the Resume button to run the demo program. Using the + and – buttons on-screen, navigate through the eight screens. Make sure to try out the checkboxes, push buttons, radio buttons and sliders. When you’re done experimenting, click Terminate on the CCS menu bar to return to the CCS Editing perspective.

Writing Our Own Code

6. The first task that our lab software will do is to display an image. So we need to create an image in a format that the graphics library can understand. If you have not done so already, download GIMP from www.gimp.org and install it on your PC. The steps below will go through the process of clipping the photo below and displaying it on the LCD display. If you prefer to use an existing image or photograph, or one taken from your smartphone camera now, simply adapt the steps below.

7. Make sure that this page of the workbook pdf is open for viewing and press PrtScn on your keyboard. This will copy the screen to your clipboard. The dimensions of the photo below approximate that of the 320x240 KenTec LCD.



8. Open GIMP (make sure it is version 2.8 or later) and click Edit → Paste. In the toolbox window, click the Rectangle Select tool, and select tightly around the border of the photo. Zoom in if that is easier for you. Click Image → Crop to selection. Click Image → Scale Image and make sure that the image size width/height is 320x240 and click Scale. You may need to click the "chain" symbol to the right of the pixel boxes to stop GIMP from preserving the wrong dimensions.

9. Convert the image to indexed mode by clicking Image → Mode → Indexed. Select Generate optimum palette and change the Maximum number of colors box to 16 (the color depth of the LCD). Click Convert.

10. Save the file by clicking File → Export… Name the image pic, change the save folder to `C:\StellarisWare\tools\bin` and select PNM image as the file type using the **+ Select File Type just above the Help button**. Click **Export**. When prompted, select **Raw** as the data formatting and click **Export**. Close GIMP.

11. Now that we have a source image file in PNM format, we can convert it to something that the graphics library can handle. We'll use the `pnmtoc` (PNM to C array) conversion utility to do the translation.

    Open a command prompt by clicking Start → Run. Type `cmd` in the window and click Open. The `pnmtoc` utility is in `C:\StellarisWare\tools\bin`. Type (Ctrl-V will not work) `cd C:\StellarisWare\tools\bin` in the command window, then press Enter to change the folder to that location.

    Finally, perform the conversion by typing `pnmtoc –c pic.pnm > pic.c` in the command window and hit Enter. When the process completes correctly, the cursor will simply drop to a new line. Close the DOS window.

12. Using Windows Explorer, find the CCS workspace in your My Documents folder. Open the folder and find the `grlib_demo` folder that was copied here when you imported this project. Copy pic.c from `C:\StellarisWare\tools\bin` to the `grlib_demo` folder.

    Look back in the expanded `grlib_demo` project in the CCS Project Explorer. If the `pic.c` file does not appear there, right-click on the project and select Refresh.

Modify pic.c

13. Open **`pic.c`** and add the following include to the very top of the file:

    `#include "grlib/grlib.h"`

Your **`pic.c`** file should look something like this (your data will vary greatly):

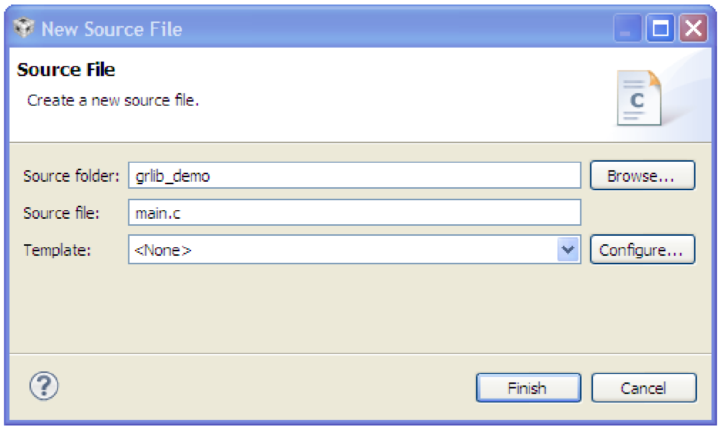
```
#include "grlib/grlib.h"

const unsigned char g_pucImage[] =
{
    IMAGE_FMT_4BPP_COMP,
    96, 0,
    64, 0,

    15,
    0x00, 0x02, 0x00,
    0x18, 0x1a, 0x19,
    0x28, 0x2a, 0x28,
    0x38, 0x3a, 0x38,
    0x44, 0x46, 0x44,
    0x54, 0x57, 0x55,
    0x62, 0x65, 0x63,
    0x72, 0x75, 0x73,
    0x81, 0x84, 0x82,
    0x93, 0x96, 0x94,
    0xa2, 0xa5, 0xa3,
    0xb3, 0xb6, 0xb4,
    0xc4, 0xc7, 0xc5,
    0xd7, 0xda, 0xd8,
    0xe8, 0xeb, 0xe9,
    0xf4, 0xf8, 0xf5,

    0xff, 0x07, 0x07, 0x07, 0x07, 0x07, 0x07, 0x07, 0x07, 0xff, 0x07, 0x07,
    0x07, 0x07, 0x07, 0x07, 0x07, 0x07, 0xff, 0x07, 0x07, 0x07, 0x07, 0x07,
    0x07, 0x07, 0x07, 0xfc, 0x07, 0x07, 0x07, 0x07, 0x07, 0x07, 0x03, 0x77,
    0x23, 0x77, 0x77, 0xe9, 0x77, 0x78, 0x70, 0x07, 0x07, 0xc1, 0x77, 0x2c,
    0x04, 0xde, 0xee, 0xee, 0xee, 0xe9, 0x3c, 0xee, 0xa1, 0x07, 0x07, 0x77,
    0x2c, 0x03, 0xcf, 0x00, 0xee, 0xee, 0xee, 0xef, 0xee, 0xef, 0xfe, 0xa0,
    0xf0, 0x07, 0x07, 0x77, 0x2c, 0x03, 0xcf, 0xee, 0xee, 0x4f, 0xee, 0xe9,
    0xee, 0xa0, 0x07, 0x07, 0x77, 0x2c, 0x04, 0x03, 0xcf, 0xee, 0xee, 0xee,
    0xe9, 0xee, 0x90, 0xf0, 0x07, 0x07, 0x77, 0x2c, 0x03, 0xcf, 0xee, 0xee,
    0x4f, 0xee, 0xe9, 0xee, 0x90, 0x07, 0x07, 0x77, 0x2c, 0x04, 0x03, 0xcf,

    many, many more lines of this data …

    0x77, 0x2c, 0x19, 0xfe, 0xee, 0xef, 0x03, 0xee, 0xee, 0xee, 0xee, 0xfb,
    0x20, 0x07, 0x07, 0xc1, 0x77, 0x2c, 0x05, 0xdf, 0xee, 0xee, 0xee, 0xe9,
    0x78, 0xf9, 0x07, 0x07, 0x77, 0x2d, 0x01, 0x8d, 0xee, 0x2f, 0xee, 0xee,
    0xe9, 0xf7, 0x07, 0x07, 0x77, 0x2e, 0x00, 0x39, 0xef, 0xee, 0xee, 0xee,
    0xee, 0xee, 0xf7, 0xf0, 0x07, 0x07, 0x77, 0x2e, 0x06, 0xdf, 0xee, 0xee,
    0x0f, 0xee, 0xee, 0xee, 0xf6, 0x07, 0x07, 0x77, 0x2f, 0x01, 0x7d, 0xfe,
    0xee, 0xee, 0xee, 0xee, 0xf7, 0x07, 0xe0, 0x07, 0x77, 0x2f, 0x17, 0xdf,
    0xee, 0xee, 0xee, 0x3c, 0xee, 0xf7, 0x07, 0x07, 0x77, 0x2f, 0x01, 0x7d,
    0x03, 0xee, 0xee, 0xee, 0xee, 0xf9, 0x10, 0x07, 0x07, 0xc0, 0x77, 0x2f,
    0x05, 0xad, 0xee, 0xfe, 0xee, 0xfc, 0x78, 0x20, 0x07, 0x07, 0x77, 0x2f,
    0x00, 0x27, 0x9d, 0x0f, 0xed, 0xee, 0xec, 0x40, 0x07, 0x07, 0x77, 0x2f,
    0x01, 0x00, 0x00, 0x28, 0x9a, 0xcc, 0xa9, 0x30, 0x07, 0xff, 0x07, 0x77,
    0x2f, 0x07, 0x07, 0x07, 0x07, 0x07, 0xc0, 0x07, 0x07,
};
```

Save your changes and close the **`pic.c`** editor pane. If you're having issues with this, you can find a `pic.c` file in the Lab10 folder.

Main.c

14. To speed things up, we're going to use the entire demo project as a template for our own `main()` code. But we can't have `grlib_demo.c` in the project since it already has a `main()`. In the Project Explorer, right-click on `grlib_demo.c` and select Resource Configurations → Exclude from Build… Click the Select All button to select both the Debug and Release configurations, and then click OK. In this manner we can keep the old file in the project, but it will not be used during the build process. This is a valuable technique when you are building multiple versions of a system that shares much of the code between them.

15. On the CCS menu bar, click File → New → Source File. Make the selections shown below and click Finish:


New Source File

Source File

Create a new source file.

Source folder: grlib_demo

Source file: main.c

Template: <None>

16. Open main.c for editing. Add (or copy/paste) the following lines to the top:

```
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/debug.h"
#include "driverlib/sysctl.h"
#include "grlib/grlib.h"
#include "drivers/Kentec320x240x16_ssd2119_8bit.h"
```

Pointer to the Image Array

17. The declaration of the image array needs to be made, as well as the declaration of two variables. The variables defined below are used for initializing the `Context` and `Rect` structures. `Context` is a definition of the screen such as the clipping region, default color and font. `Rect` is a simple structure for drawing rectangles. Look up these APIs in the Graphics Library users guide .

    Add a line for spacing and add the following lines after the includes:

```
extern const unsigned char g_pucImage[];
tContext sContext;
tRectangle sRect;
```

Driver Library Error Routine

18. The following code will be called if the driver library encounters an error.

    Leave a line for spacing and enter these line of codes after the lines above:

```
#ifdef DEBUG
void__error__(char *pcFilename, unsigned long ulLine)
{
}
#endif
```

Main()

19. The main() routine will be next. Leave a blank line for spacing and enter these lines of code after the lines above:

```
int main(void)
{

}
```

Initialization

20. Set the clocking to run at 50 MHz using the PLL (400MHz ÷ 2 ÷ 4). Leave a line for spacing, then insert this line as the first inside main():

```
SysCtlClockSet(SYSCTL_SYSDIV_4 | SYSCTL_USE_PLL | SYSCTL_OSC_MAIN | SYSCTL_XTAL_16MHZ);
```

    Initialize the display driver. Skip a line and insert this line after the last:

```
Kentec320x240x16_SSD2119Init();
```

    This next function initializes a drawing context, preparing it for use. The provided display driver will be used for all subsequent graphics operations, and the default clipping region will be set to the extent of the LCD screen. Insert this line after the last:

```
GrContextInit(&sContext, &g_sKentec320x240x16_SSD2119);
```

21. Let's add a call to a function that will clear the screen. We'll create that function in a moment. Add the following line after the last one:

```
ClrScreen();
```

22. The following function will create a rectangle that covers the entire screen, set the foreground color to black, and fill the rectangle by passing the structure `sRect` by reference. The top left corner of the LCD display is the point (0,0) and the bottom right corner is (319,239). Add the following code after the final closing brace of the program in main.c.

```
void ClrScreen()
{
   sRect.sXMin = 0;
   sRect.sYMin = 0;
   sRect.sXMax = 319;
   sRect.sYMax = 239;
   GrContextForegroundSet(&sContext, ClrBlack);
   GrRectFill(&sContext, &sRect);
   GrFlush(&sContext);
}
```

23. Declare the function at the top of your code right below your variable definitions:

```
void ClrScreen(void);
```

Displaying the Image

24. Display the image by passing the global image variable g_pucImage into GrImageDraw(...) and place the image on the screen by locating the top-left corner at (0,0) …we'll adjust this later if needed. Leave a line for spacing, then insert this line after the ClrScreen() call in main():

```
GrImageDraw(&sContext, g_pucImage, 0, 0);
```

25. The function call below flushes any cached drawing operations. For display drivers that draw into a local frame buffer before writing to the actual display, calling this function will cause the display to be updated to match the contents of the local frame buffer. Insert this line after the last:

```
GrFlush(&sContext);
```

26. We will be stepping through a series of displays in this lab, so we want to leave each display on the screen long enough to see it before it is erased. The delay below will give you a chance to appreciate your work. Leave a line for spacing, then insert this line after the last:

```
SysCtlDelay(SysCtlClockGet());
```

In previous labs we've simply passed a number to the `SysCtlDelay()` API call, but if you were to change the CPU clock speed, your delay time would change. `SysCtlClockGet()` will return the system clock speed and we can use that as our delay basis. Obviously, you could have your delay be twice, half, 1/5th or some other multiple of this.

27. Before we go any further, we'd like to take the code for a test run. With that in mind we're going to add the final code pieces now, and insert later lab code in front of this.

    LCD displays are not especially prone to burn in, but clearing the screen will mark a clear break between one step in the code and the next. This performs the same function as step 24 and also flushes the cache. Leave several lines for spacing and add this line below the last:

    ```
    ClrScreen();
    ```

28. Add a while loop to the end of the code to stop execution. Leave a line for spacing, then insert these line after the last:

    ```
    while(1)
    {
    }
    ```

    Don't forget that you can auto-correct the indentation if needed.

If you're having issues, you can find this code in `main1.txt` in the Lab10 folder.
Your code should look like this:

```
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/debug.h"
#include "driverlib/sysctl.h"
#include "grlib/grlib.h"
#include "drivers/Kentec320x240x16_ssd2119_8bit.h"

extern const unsigned char g_pucImage[];
tContext sContext;
tRectangle sRect;

void ClrScreen(void);

#ifdef DEBUG
void__error__(char *pcFilename, unsigned long ulLine)
{
}
#endif

int main(void)
{
   SysCtlClockSet(SYSCTL_SYSDIV_4|SYSCTL_USE_PLL|SYSCTL_OSC_MAIN|SYSCTL_XTAL_16MHZ);

   Kentec320x240x16_SSD2119Init();
   GrContextInit(&sContext, &g_sKentec320x240x16_SSD2119);
   ClrScreen();

   GrImageDraw(&sContext, g_pucImage, 0, 0);
   GrFlush(&sContext);

   SysCtlDelay(SysCtlClockGet());
   // Later lab steps go between here

   // and here
   ClrScreen();
   while(1)
   {
   }
}

void ClrScreen()
{
   sRect.sXMin = 0;
   sRect.sYMin = 0;
   sRect.sXMax = 319;
   sRect.sYMax = 239;
   GrContextForegroundSet(&sContext, ClrBlack);
   GrRectFill(&sContext, &sRect);
   GrFlush(&sContext);
}
```

Check the Build Options

29. Now would be a good time to check the build options that have been set in this demo code. You should know how to do this without explicit steps by now. Take a look in the Linker's File Search Path and note that the `.lib` file for the graphics library has been included.

    You might notice the use of two "new" path variables:

    - CG_TOOL_ROOT
    - SW_ROOT

    Take at look in the project properties under Resource → Linked Resources to see where these paths are defined.

Run the Code

30. Make sure `grlib_demo` is the active project. Compile and download your application by clicking the Debug button. Click the Resume button to run the program that was downloaded to the flash memory of your LM4F120H5QR. If your coding efforts were successful, you should see your image appear on the LCD display for a few seconds, then disappear.

    When you're finished, click the Terminate button to return to the CCS Edit perspective.

When you are including images in your projects, remember that they can be quite large in terms of memory space. This might possibly require a larger flash device, and increase your system cost.

Display Text On-Screen

31. Refer back to the code on page 10-20. In `main.c` in the area marked:

    ```
    // Later lab steps go between here

    // and here
    ```

    insert the following function call to clear the screen and flush the buffer:

    ```
    ClrScreen();
    ```

32. Next we'll display the text. Display text starting at (x,y) with the no background color. The third parameter (-1) simply tells the API function to send the entire string, rather than having to count the characters.

    `GrContextForegroundSet(...)`: Set the foreground for the text to be red.

    `GrContextFontSet(...)`: Set the font to be a max height of 30 pixels.

    `GrRectDraw(...)`: Put a white border around the screen.

    `GrFlush(...)`: And refresh the screen by matching the contents of the local frame buffer.

    Note the colors that are being used. If you'd like to try other colors, fonts or sizes, look in the back of the Graphics Library User's Guide. Add the following lines after the previous ones:

    ```
       sRect.sXMin = 1;
       sRect.sYMin = 1;
       sRect.sXMax = 318;
       sRect.sYMax = 238;
       GrContextForegroundSet(&sContext, ClrRed);
       GrContextFontSet(&sContext, &g_sFontCmss30b);
       GrStringDraw(&sContext, "Texas", -1, 110, 2, 0);
       GrStringDraw(&sContext, "Instruments", -1, 80, 32, 0);
       GrStringDraw(&sContext, "Graphics", -1, 100, 62, 0);
       GrStringDraw(&sContext, "Lab", -1, 135, 92, 0);
       GrContextForegroundSet(&sContext, ClrWhite);
       GrRectDraw(&sContext, &sRect);
       GrFlush(&sContext);
    ```

33. Add a delay so you can view your work.

    ```
    SysCtlDelay(SysCtlClockGet());
    ```

    Save your file.

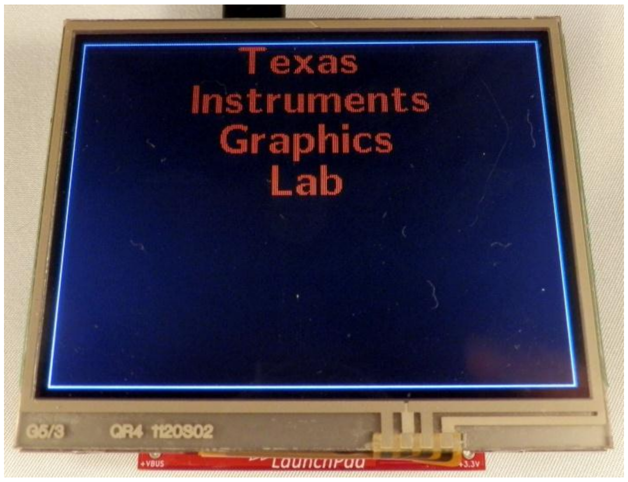
If you're having issues, you can find this code in `main2.txt` in the Lab10 folder.

Your added code should look like this:

```
    // Later lab steps go between here

    ClrScreen();

    sRect.sXMin = 1;
    sRect.sYMin = 1;
    sRect.sXMax = 318;
    sRect.sYMax = 238;
    GrContextForegroundSet(&sContext, ClrRed);
    GrContextFontSet(&sContext, &g_sFontCmss30b);
    GrStringDraw(&sContext, "Texas", -1, 110, 2, 0);
    GrStringDraw(&sContext, "Instruments", -1, 80, 32, 0);
    GrStringDraw(&sContext, "Graphics", -1, 100, 62, 0);
    GrStringDraw(&sContext, "Lab", -1, 135, 92, 0);
    GrContextForegroundSet(&sContext, ClrWhite);
    GrRectDraw(&sContext, &sRect);
    GrFlush(&sContext);

    SysCtlDelay(SysCtlClockGet());

    // and here
```

Build, Load and Test

34. Build, load and run your code. If your changes are correct, you should see the image again for a few seconds, followed by the on-screen text in a box for a few seconds. Then the display will blank out. Return to the CCS Edit perspective when you're done.



Drawing Shapes

35. Let's add a filled-in yellow circle. Make the foreground yellow and center the circle at (80,182) with a radius of 50. Add a line for spacing and then add these lines after the `SysCtlDelay()` added in step 33:

```
GrContextForegroundSet(&sContext, ClrYellow);
GrCircleFill(&sContext, 80, 182, 50);
```

36. Draw an empty green rectangle starting with the top left corner at (160,132) and finishing at the bottom right corner at (312,232). Add a line for spacing and add the following lines after the last ones:

```
sRect.sXMin = 160;
sRect.sYMin = 132;
sRect.sXMax = 312;
sRect.sYMax = 232;
GrContextForegroundSet(&sContext, ClrGreen);
GrRectDraw(&sContext, &sRect);
```

37. Add a short delay to appreciate your work. Add a line for spacing and add the following line after the last ones:

```
SysCtlDelay(SysCtlClockGet());
```

Save your work.

If you're having issues, you can find this code in `main3.txt` in the Lab10 folder.

Your added code should look like this:

```
// Later lab steps go between here

ClrScreen();

sRect.sXMin = 1;
sRect.sYMin = 1;
sRect.sXMax = 318;
sRect.sYMax = 238;
GrContextForegroundSet(&sContext, ClrRed);
GrContextFontSet(&sContext, &g_sFontCmss30b);
GrStringDraw(&sContext, "Texas", -1, 110, 2, 0);
GrStringDraw(&sContext, "Instruments", -1, 80, 32, 0);
GrStringDraw(&sContext, "Graphics", -1, 100, 62, 0);
GrStringDraw(&sContext, "Lab", -1, 135, 92, 0);
GrContextForegroundSet(&sContext, ClrWhite);
GrRectDraw(&sContext, &sRect);
GrFlush(&sContext);

SysCtlDelay(SysCtlClockGet());

GrContextForegroundSet(&sContext, ClrYellow);
GrCircleFill(&sContext, 80, 182, 50);

sRect.sXMin = 160;
sRect.sYMin = 132;
sRect.sXMax = 312;
sRect.sYMax = 232;
GrContextForegroundSet(&sContext, ClrGreen);
GrRectDraw(&sContext, &sRect);

SysCtlDelay(SysCtlClockGet());

// and here
```

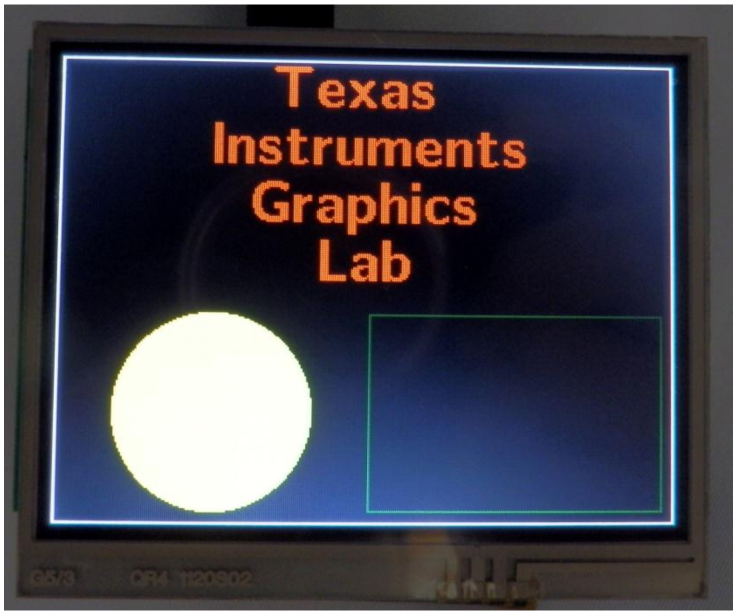
For reference, the final code should look like this:

```
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/debug.h"
#include "driverlib/sysctl.h"
#include "grlib/grlib.h"
#include "drivers/Kentec320x240x16_ssd2119_8bit.h"

extern const unsigned char g_pucImage[];
tContext sContext;
tRectangle sRect;

void ClrScreen(void);

#ifdef DEBUG
void__error__(char *pcFilename, unsigned long ulLine)
{
}
#endif

int main(void)
{
   SysCtlClockSet(SYSCTL_SYSDIV_4|SYSCTL_USE_PLL|SYSCTL_OSC_MAIN|SYSCTL_XTAL_16MHZ);

   Kentec320x240x16_SSD2119Init();
   GrContextInit(&sContext, &g_sKentec320x240x16_SSD2119);
   ClrScreen();

   GrImageDraw(&sContext, g_pucImage, 0, 0);
   GrFlush(&sContext);

   SysCtlDelay(SysCtlClockGet());
   // Later lab steps go between here

   ClrScreen();

   sRect.sXMin = 1;
   sRect.sYMin = 1;
   sRect.sXMax = 318;
   sRect.sYMax = 238;
   GrContextForegroundSet(&sContext, ClrRed);
   GrContextFontSet(&sContext, &g_sFontCmss30b);
   GrStringDraw(&sContext, "Texas", -1, 110, 2, 0);
   GrStringDraw(&sContext, "Instruments", -1, 80, 32, 0);
   GrStringDraw(&sContext, "Graphics", -1, 100, 62, 0);
   GrStringDraw(&sContext, "Lab", -1, 135, 92, 0);
   GrContextForegroundSet(&sContext, ClrWhite);
   GrRectDraw(&sContext, &sRect);
   GrFlush(&sContext);

   SysCtlDelay(SysCtlClockGet());

   GrContextForegroundSet(&sContext, ClrYellow);
   GrCircleFill(&sContext, 80, 182, 50);

   sRect.sXMin = 160;
   sRect.sYMin = 132;
   sRect.sXMax = 312;
   sRect.sYMax = 232;
   GrContextForegroundSet(&sContext, ClrGreen);
   GrRectDraw(&sContext, &sRect);

   SysCtlDelay(SysCtlClockGet());

   // and here
   ClrScreen();
   while(1)
   {
   }
}

void ClrScreen()
{
   sRect.sXMin = 0;
   sRect.sYMin = 0;
   sRect.sXMax = 319;
   sRect.sYMax = 239;
   GrContextForegroundSet(&sContext, ClrBlack);
   GrRectFill(&sContext, &sRect);
   GrFlush(&sContext);
}
```

This is the code in `main3.txt`.

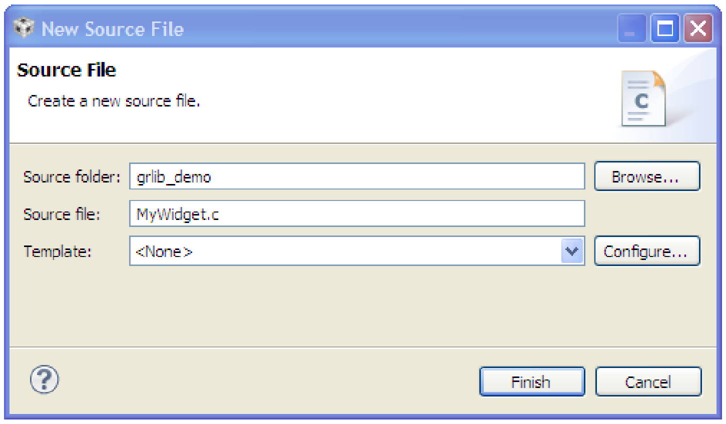
Build, Load and Test

38. Build, load and run your code to make sure that your changes work. Return to the CCS Edit perspective when you are done.



Widgets

39. Let's play with some widgets. In this case, we'll create a screen with a nice header and a large rectangular button that will toggle the red LED on and off. Modifying the existing code would be a little tedious, so we'll create a new file.
40. In the Project Explorer, right-click on `main.c` and select Resource Configurations → Exclude from Build… Click the Select All button to select both the Debug and Release configurations, and then click OK.
41. On the CCS menu bar, click File → New → Source File. Make the selections shown below and click Finish:



Source folder: grlib_demo
Source file: MyWidget.c
Template: <None>

42. Add the following support files to the top of `MyWidget.c`:

```
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/interrupt.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
#include "grlib/grlib.h"
#include "grlib/widget.h"
#include "grlib/canvas.h"
#include "grlib/pushbutton.h"
#include "drivers/Kentec320x240x16_ssd2119_8bit.h"
#include "drivers/touch.h"
```

43. The next two lines provide names for structures needed to create the background canvas and the button widget. Add a line for spacing, then add these lines below the last:

```
extern tCanvasWidget g_sBackground;
extern tPushButtonWidget g_sPushBtn;
```

44. When the button widget is pressed, a handler called OnButtonPress() will toggle the LED. Add a line for spacing, then add this prototype below the last:

```
void OnButtonPress(tWidget *pWidget);
```

45. Widgets are arranged on the screen in order of a parent-child relationship, where the parent is in the back. This relationship can extend multiple levels. In our example, we're going to have the background be the parent or root and the heading will be a child of the background. The button will be a child of the heading. Add a line for spacing and then add the following two global variables (one for the background and one for the button) below the last:

    ```
    Canvas(g_sHeading, &g_sBackground, 0, &g_sPushBtn,
           &g_sKentec320x240x16_SSD2119, 0, 0, 320, 23,
           (CANVAS_STYLE_FILL | CANVAS_STYLE_OUTLINE | CANVAS_STYLE_TEXT),
           ClrBlack, ClrWhite, ClrRed, g_pFontCm20, "LED Control", 0, 0);

    Canvas(g_sBackground, WIDGET_ROOT, 0, &g_sHeading,
           &g_sKentec320x240x16_SSD2119, 0, 23, 320, (240 - 23),
           CANVAS_STYLE_FILL, ClrBlack, 0, 0, 0, 0, 0, 0);
    ```

    Rather than re-print the parameter list for these declarations, refer to section 5.2.3.1 in the Stellaris Graphics Library User's Guide (SW-GRL-UG-xxxx.pdf). The short description is that there will be a black background. In front of that is a white rectangle at the top of the screen with "LED Control" inside it.

46. Next up is the definition for the rectangular button we're going to use. The button is functionally in front of the heading, but physically located below it (refer to the picture in step 50). It will be a red rectangle with a gray background and "Toggle red LED" inside it. When pressed it will fill with white and the handler named `OnButtonPress` will be called. Add a line for spacing and then add the following code below the last:

    ```
    RectangularButton(g_sPushBtn, &g_sHeading, 0, 0,
                      &g_sKentec320x240x16_SSD2119, 60, 60, 200, 40,
                      (PB_STYLE_OUTLINE | PB_STYLE_TEXT_OPAQUE | PB_STYLE_TEXT |
                       PB_STYLE_FILL), ClrGray, ClrWhite, ClrRed, ClrRed,
                       g_pFontCmss22b, "Toggle red LED", 0, 0, 0, 0, OnButtonPress);
    ```

    Refer to section 10.2.3.33 in the Stellaris Graphics Library User's Guide (spmu018n.pdf) for more detail.

47. The last detail before the actual code is a flag variable to indicate whether the LED is on or off. Add a line for spacing and then add the following code below the last:

    ```
    tBoolean g_RedLedOn = false;
    ```

48. When the button is pressed, a handler called OnButton Press() will be called. This handler uses the flag to switch between turning the red LED on or off. Add a line for spacing and then add the following code below the last:

```
void OnButtonPress(tWidget *pWidget)
{
    g_RedLedOn = !g_RedLedOn;

    if(g_RedLedOn)
    {
        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1, 0x02);
    }
    else
    {
        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1, 0x00);
    }
}
```

49. Lastly is the `main()` routine. The steps are: initialize the clock, initialize the GPIO, initialize the display, initialize the touchscreen, enable the touchscreen callback so that the routine indicated in the button structure will be called when it is pressed, add the background and paint it to the screen (parents first, followed by the children) and finally, loop while the widget polls for a button press. Add a line for spacing and then add the following code below the last:

```
int main(void)
{

    SysCt-
lClockSet(SYSCTL_SYSDIV_4|SYSCTL_USE_PLL|SYSCTL_OSC_MAIN|SYSCTL_XTAL_16MHZ);

    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
    GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3);
    GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3, 0x00);

    Kentec320x240x16_SSD2119Init();

    TouchScreenInit();

    TouchScreenCallbackSet(WidgetPointerMessage);

    WidgetAdd(WIDGET_ROOT, (tWidget *)&g_sBackground);

    WidgetPaint(WIDGET_ROOT);

    while(1)
    {
        WidgetMessageQueueProcess();
    }
}
```

Save your file.

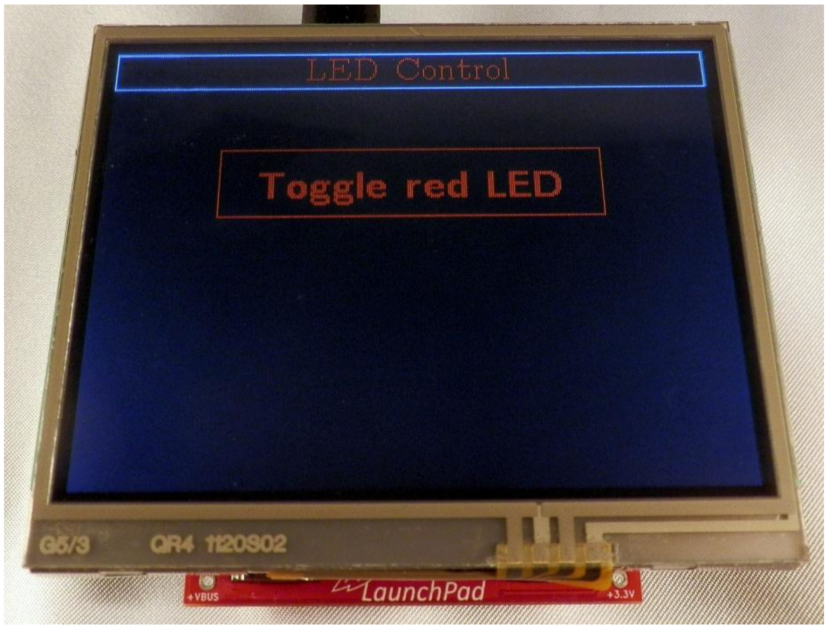
If you're having issues, you can find this code in `MyWidget.txt` in the Lab10 folder.

Your added code should look like the next page:

```
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/interrupt.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
#include "grlib/grlib.h"
#include "grlib/widget.h"
#include "grlib/canvas.h"
#include "grlib/pushbutton.h"
#include "drivers/Kentec320x240x16_ssd2119_8bit.h"
#include "drivers/touch.h"

extern tCanvasWidget g_sBackground;
extern tPushButtonWidget g_sPushBtn;

void OnButtonPress(tWidget *pWidget);

Canvas(g_sHeading, &g_sBackground, 0, &g_sPushBtn,
       &g_sKentec320x240x16_SSD2119, 0, 0, 320, 23,
       (CANVAS_STYLE_FILL | CANVAS_STYLE_OUTLINE | CANVAS_STYLE_TEXT),
       ClrBlack, ClrWhite, ClrRed, g_pFontCm20, "LED Control", 0, 0);

Canvas(g_sBackground, WIDGET_ROOT, 0, &g_sHeading,
       &g_sKentec320x240x16_SSD2119, 0, 23, 320, (240 - 23),
       CANVAS_STYLE_FILL, ClrBlack, 0, 0, 0, 0, 0, 0);

RectangularButton(g_sPushBtn, &g_sHeading, 0, 0,
                  &g_sKentec320x240x16_SSD2119, 60, 60, 200, 40,
                  (PB_STYLE_OUTLINE | PB_STYLE_TEXT_OPAQUE | PB_STYLE_TEXT |
                   PB_STYLE_FILL), ClrGray, ClrWhite, ClrRed, ClrRed,
                   g_pFontCmss22b, "Toggle red LED", 0, 0, 0, 0, OnButtonPress);

tBoolean g_RedLedOn = false;

void OnButtonPress(tWidget *pWidget)
{
    g_RedLedOn = !g_RedLedOn;

    if(g_RedLedOn)
    {
        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1, 0x02);
    }
    else
    {
        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1, 0x00);
    }
}
int main(void)
{

    SysCtlClockSet(SYSCTL_SYSDIV_4|SYSCTL_USE_PLL|SYSCTL_OSC_MAIN|SYSCTL_XTAL_16MHZ);

    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
    GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3);
    GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3, 0x00);

    Kentec320x240x16_SSD2119Init();

    TouchScreenInit();

    TouchScreenCallbackSet(WidgetPointerMessage);

    WidgetAdd(WIDGET_ROOT, (tWidget *)&g_sBackground);

    WidgetPaint(WIDGET_ROOT);

    while(1)
    {
        WidgetMessageQueueProcess();
    }
}
```

Build, Load and Test

50. Build, load and run your code to make sure that everything works. Press the rectangular button and the red LED on the LaunchPad will light, press it again and it will turn off.



51. Click the Terminate button to return to the CCS Edit perspective when you are done. Close all open lab projects and minimize Code Composer Studio.

52. If you want to reprogram the `qs-rgb` application that was originally on the LaunchPad board, the steps are in section two of this workshop.

53. Homework Ideas:

- Change the red background of the button so that it stays on when the LED is lit
- Add more buttons to control the green and blue LEDs.
- Use the Lab5 ADC code to display the measured temperature on the LCD in real time.
- Use the RTC to display the time of day on screen.
- Use the Lab6 Hibernation code to make the device sleep, and the backlight go off, after no screen touch for 10 seconds
- Use the Lab7 USB code to send data to the LCD and touch screen presses back to the PC.
- Use the Lab9 sine wave code to create a program that displays the sine wave data on the LCD screen.

STOP

You're done.