TI-RSLK MAX

Texas Instruments Robotics System Learning Kit

Texas Instruments

Module 11

Lab 11: Interfacing graphical displays

Lab: Interfacing graphical displays

11.0 Objectives

The purpose of this lab is to interface a display module to the microcontroller and develop a set of software routines to assist in debugging of the robot.

1. You will connect a display module to the microcontroller.
2. You will use a serial protocol to communicate with the display.
3. You will implement a set of software functions for the display.

Good to Know: Even though the display doesn't actually contribute to the input-calculate-output loop required to solve the robot challenge, it will be extremely useful when debugging when the robot is running untethered in its world.

11.1 Getting Started

11.1.1 Software Starter Projects

There are three options for Lab 11. Pick one of these projects depending on your hardware:

Lab11_LCD (starter project for this lab using Nokia 5110 LCD)
Lab11_OLED (starter project for this lab using SSD1306 OLED)
Lab11_UART (starter project for this lab using UART, no LCD nor OLED)

11.1.2 Student Resources (in datasheets directory)

Nokia5110.pdf (data sheet for the LCD)
SSD1306.pdf (data sheet for the OLED)
TI-RSLK MAX construction guide document

11.1.3 Reading Materials

Chapter 11, "Embedded Systems: Introduction to Robotics"

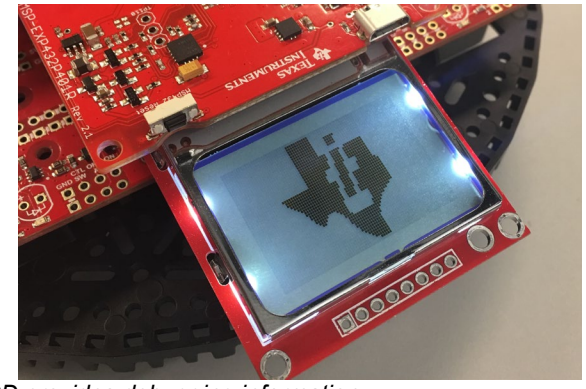

Figure 1. LCD provides debugging information.

11.1.4 Components needed for this lab

All components needed for this lab are included in the TI-RSLK Max kit (TIRSLK-EVM), you will need to purchase LCD or OLED displays (optional). Headers are included in your kit. This lab will require disassembly of your robot and some soldering. Batteries are also needed to power up the robot.

Quantity	Description	Manufacturer	Mfg P/N
1	TI-RSLK MAX robot kit	TI	TIRSLK-EVM
1	LCD display* or OLED display*	Sparkfun/Adafruit Adafruit	10168/338 SSD1306/938

11.1.5 Lab equipment needed

Oscilloscope (two channels with at least 10MHz sampling)
Or Logic Analyzer (4 channels with at least 10MHz sampling)

11.2 System Design Requirements

The overall goal of this lab is to interface a display to the microcontroller and use it to output characters and numbers. The Nokia 5110 is a monochrome display that is 84 pixels wide by 48 pixels high. The Adafruit SSD1306 is an OLED display that is 128 pixels wide by 64 pixels high. You can use either display for this lab. If you have neither display, you can send character information to the PC using the UART serial channel, provided by the LaunchPad.

Each character on the LCD and OLED is defined by a 5 pixels wide by 8 pixels high image. You can see the font table as a constant array called **ASCII** inside **Nokia.c** or **SSD1306.c** file. For example, the letter '7' is defined as

```
{0x01, 0x71, 0x09, 0x05, 0x03}
```

This 40-bit value creates the image shown in Figure 2 on the display. The 0x01 is the first column and 0x03 is the last column, with bit 0 on top.

Each character has bit 7 clear to make a space between lines. The function **Nokia5110_OutChar** will place a blank vertical line in front of and one blank line after each character. This means each character requires a 7 by 8 pixel area to print. Since the LCD display is 84 wide by 48 high, this font size allows for 84/7= 12 characters on each line, and allows 48/8=6 lines.

For the OLED, each character also has bit 7 clear to make a space between lines. However, the function **SSD1306_OutChar** will place only one blank line after each character. This means each character requires a 6 by 8 pixel area to print. Since the OLED display is 128 wide by 64 high, this allows for 128/6= 21 characters on each line, and allows 64/8=8 lines.

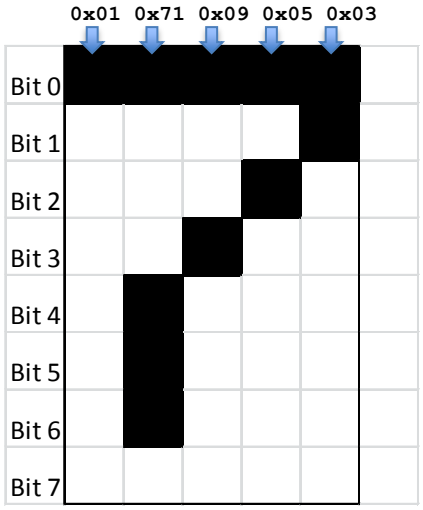

Figure 2. Pixel image to create the number 7. Bit 7 is clear for all characters.

On the low level, you are required to write one routine that sends an 8-bit command to the display (**commandwrite**) and a second routine that sends 8-bit data to the display (**datawrite**). These routines use busy-wait synchronization with the SPI synchronous serial interface. These are private functions, so prototypes are not available in the **Nokia5110.h** or **SSD1306.h** header files.

On the high level, you are required to write one routine that outputs a string (**OutString**) and a second routine to output an unsigned 16-bit decimal number (**OutUDec**).

More specific instructions can be found in the **Nokia5110.h Nokia5110.c SSD1306.h** and **SSD1306.c** files. These files are located in the **inc** folder. This means after you complete this lab, you can use these functions in the remaining labs.

11.3 Experiment set-up

You will implement this lab using the TI-RSLK MAX robot that you build in lab 5 that is connected to the MSP432 LaunchPad. Figure 3 shows the TI-RSLK chassis board supports three different displays. You will need to remove the chassis board from the robot to solder the LCD. See the TI-RSLK MAX construction guide document that contains the instructions.

Warning: Please ensure the +5V jumper on the MSP432 LaunchPad is disconnected or removed. Not removing this jumper will cause permanent damage to the LaunchPad and the TI-RSLK chassis board.

However you are wiring up an LCD ignore the pin numbers and match the pin names. Three LCD headers are provided on the TI-RSLK MAX chassis board, see Figure 3. Each interface uses the following five MSP432 pins.

MSP432 P9.7 is SPI USC3SIMO, also called LCDMOSI (data)
MSP432 P9.6 is a GPIO, also called LCDDC (data/command)
MSP432 P9.5 is SPI UCA3CLK, also called LCDSCLK (clock)
MSP432 P9.4 is SPI UCA3STE, also called LCDCS (select)
MSP432 P9.3 is a GPIO, also called LCDRST (reset)

Pins P9.3 and P9.6 are set to be regular digital outputs. DC=1 means data, and DC=0 means command. The reset pin is used to initialize the LCD hardware. Notice the displays do not run off power from the USB cable. Battery power on the TI-RSLK MAX robot must be active to operate the LCD.

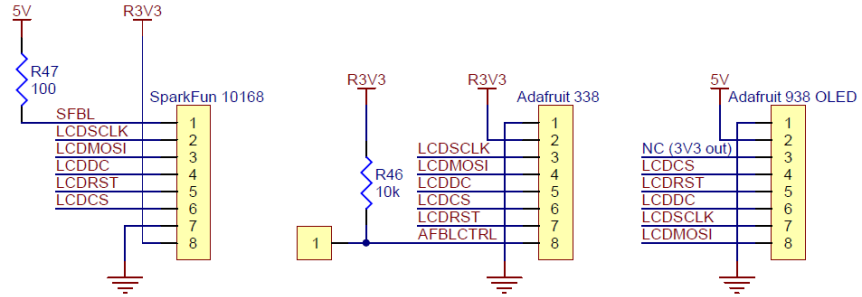

Figure 3. TI-RSLK MAX chassis board interface.

Lab: Interfacing graphical displays

11.4 System Development Plan

11.4.1 commandwrite (LCD or OLED options only)

To send one 8-bit command, your **commandwrite** function should perform the following four steps, even though 1) and 4) are the same.

1) Wait for the SPI to be idle (let previous output finish), **UCBUSY**
2) Set DC for command (0)
3) Write 8-bit command to the SPI data register (**TXBUF**), starts SPI
4) Wait for the SPI to be idle (let this transmission finish), **UCBUSY**

Look up in the Nokia5110/SSD1306 and MSP432 data sheets what the expected waveforms should look like when **commandwrite** is called in the program. Use an oscilloscope or logic analyzer to verify the waveforms are as expected. You can use a simple program like the following to test this low-level function.

```
void Testcommandwrite(void){
  while(1){
    commandwrite(0x21);
  }
}
```

11.4.2 datawrite (LCD or OLED options only)

To send one 8-bit data, your **datawrite** function should perform the following three steps. Skipping step 4) when outputting data makes it run much faster.

1) Wait for the SPI to be idle (let previous output finish), **UCBUSY**
2) Set DC for data (1)
3) Write 8-bit data to the SPI data register (**TXBUF**), starts SPI

11.4.3 OutChar (UART only)

To send one 8-bit data, your **UART0_OutChar** function should perform the following two steps.

1) Wait for the UART to be idle (let previous output finish), **TXIFG**
2) Write 8-bit data to the UART data register (**TXBUF**), starts UART

11.4.4 OutString (all three options)

There is a midlevel function (**Nokia5110_OutChar SSD1306_OutChar** or **UART0_OutChar**) that outputs one character to the output device. You will use this function to implement a function called **OutString** that outputs a null-terminated string.

11.4.4 OutUDec (all three options)

Similarly, you will use the **Nokia5110_OutChar SSD1306_OutChar** or **UART0_OutChar** routine to implement a function called **OutUDec** that outputs a 16-bit unsigned number to the device. One of the specifications for this function is that the image is created right justified that fills exactly 5 characters. This functionality allows you to output numbers on the device that are easy to read.

More specifically, for numbers 0 to 9, you will output 4 spaces and the one digit. For numbers 10 to 99, you will output 3 spaces followed by two digits. For numbers 100 to 999, you will output 2 spaces followed by three digits. For numbers 1000 to 9999, you will output 1 space followed by four digits. For numbers 10000 to 65535, you will output all five digits.

To illustrate how this function could be used, consider the example where the display contains debug data continuously updated during a robot run, You could execute this code once at the beginning

```
Nokia5110_SetCursor(0,2);  // left, third row
Nokia5110_OutString("D= ");
Nokia5110_OutUDec(0);
Nokia5110_OutString(" mm");
```

Then, when you wish to update the LCD with a new distance value, you can just output the 5 characters of the new value. This method will make a pretty display that will not flicker as the numbers change.

```
Nokia5110_SetCursor(3,2);  // spot for number
Nokia5110_OutUDec(myDistance);
```

Use a debugging profile to measure how long it takes to execute **Nokia5110_OutUDec.** Knowing that the SPI clock is 4 MHz, explain why this measurement is reasonable.

Lab: Interfacing graphical displays

11.5 Troubleshooting

The LCD does not display characters:

- Check all the connections between LaunchPad and LCD.
- Make sure RESET is high.
- Run a simple main program that calls **commandwrite** over and over with the same command. Use a logic analyzer or scope to verify all five signals from LaunchPad to LCD are proper. Section 12.1 of the data sheet describes the expected SPI protocol.

Back light does not operate:

- Check the ground.
- Verify you have pin 1 properly identified and haven't wired it backwards.

11.6 Things to think about

In this section, we list thought questions to consider after completing this lab. These questions are meant to test your understanding of the concepts in this lab.

- What does it mean that this interface is serial? Why is serial important?
- What does it mean that this interface is synchronous? Why is synchronous important?
- What is the purpose of each of these three files: **Nokia5110.h**, **Nokia5110.c**, and **Lab11_LCDmain.c**?
- How long does it take to execute **Nokia5110_OutUDec**? Is it appropriate to call this function during an ISR, or is it better to always perform LCD output in the main program?

11.7 Additional challenges

In this section, we list additional activities you could do to further explore the concepts of this module. You could extend the system or propose something completely different. For example,

- Add an output function for 16-bit signed numbers.
- Add an output function for signed 32-bit numbers
- Interface a different LCD, such as the ST7735R
- Create a set of functions that plots data versus time on the LCD

11.8 Which modules are next?

Modules 1-11 have introduced the basics of the microcontroller. The next set of modules allow for more complex functionality for the robot.

Module 12) interface the motors to the robot.
Module 13) write software to adjust power to the motors.
Module 14) use interrupts to detect collisions in real time
Module 15) interface IR distance sensors to measure distance to the wall.
Module 16) interface tachometers to measure wheel speed.

11.9 Things you should have learned

In this section, we review the important concepts you should have learned in this module:

- Understand busy-wait and know why busy-wait was used for this interface and not interrupts
- Synchronous serial protocol: how it works and why it is important
- The concept of minimally intrusive debugging

ti.com/rslk

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale (www.ti.com/legal/termsofsale.html) or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

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