Lab 7. LCD Device Driver for the Sitronix ST7735 (Sp 2023)

Videos see end of eBook chapter 7 for Lab videos

http://users.ece.utexas.edu/%7Evalvano/Volume1/IntroToEmbSys/Ch7_LocalVariables.htm#7_6

Preparation

Read all of ebook Chapter 7 (skip 7.3) or textbook sections 7.1, 7.2, 7.3, 7.4, 7.5, 7.6, and 7.7
Read the data sheet for the LCD display
1-8-tft-display.pdf (in datasheets tab)
Lab7 starter project from the original installer

Purpose

This lab has these major objectives:

1. to interface an LCD interface that can be used to display information on the embedded system;
2. to learn how to design implement and test a device driver using busy-wait synchronization;
3. to learn how to allocate and use local variables on the stack;
4. to use fixed-point numbers to store non-integer values.

The LaunchPad.DLL will simulate the Sitronix ST7735. Lab 7 has an automatic grader but your solution working on the real board with the LCD should be your ultimate goal.

Figure 7.1. Sitronix ST7735R

When buying a ST7735R, make sure it has a PCB with 0.1in header and not just the display. Make sure it is 160 by 128 pixels.

EE312H/EE319H students must use **recursion** for **OutDec**.

A short soldering guide can be found at http://users.ece.utexas.edu/~valvano/Volume1/SolderingGuide.pdf

System Requirements

In this lab you will create an embedded system that:

- Interfaces to the ST7735R LCD
- Displays numbers in decimal format
- Displays fixed point numbers in decimal format

Procedure

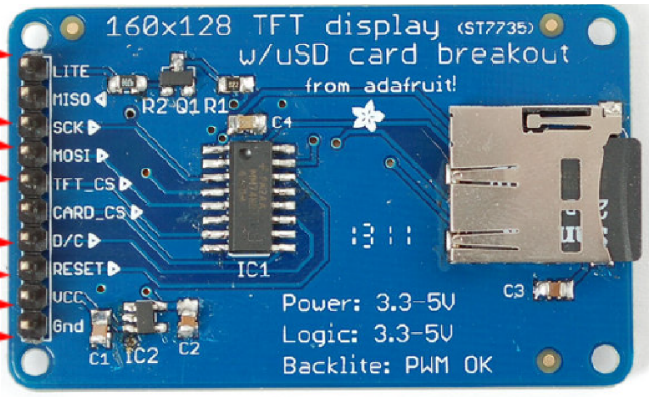
The basic approach to this lab will be to first develop and debug your system using the simulator. During this phase of the project you will use the debugger to observe your software operation. After the software is debugged, you will run your software on the real TM4C123. There are many functions to write in this lab, so it is important to develop the device driver in small pieces. One technique you might find useful is **desk checking**. Basically, you hand-execute your functions with a specific input parameter. For example, using just a pencil and paper think about the sequential steps that will occur when `Dec2String` or `Fix2String` processes the input 187. Later, while you are debugging the actual functions on the simulator, you can single step the program and compare the actual data with your expected data.

For LED heartbeats, you can use PF1 PF2 PF3 LED on the LaunchPad, For switch input you can use PF0 or PF4.

Part a - Understand Hardware

In this lab you will interface a Sitronix ST7735 LCD to the TM4C123 as shown in Program 7.1. There are many ST7735R LCDs. When wiring your LCD, look at the pin names and not the pin numbers

```
// pin 10 Backlight    +3.3 V
// pin 9  MISO         unconnected
// pin 8  SCK          PA2 (SSI0Clk)
// pin 7  MOSI         PA5 (SSI0Tx)
// pin 6  TFT_CS       PA3 (SSI0Fss)
// pin 5  CARD_CS      unconnected
// pin 4  D/C          PA6 (GPIO)
// pin 3  RESET        PA7 (GPIO)
// pin 2  VCC          +3.3 V
// pin 1  Gnd          ground
```



Program 7.1. Interface connections for the Sitronix ST7735.

D/C stands for data/command; you will make **D/C** high to send data and low to send a command. You should skim and understand the [ST77735 Data Sheet](). This Data sheet will be extremely useful for getting your driver up and running.

If your LCD looks different, then see the header file ST7735.h for how to make the connections. See Figure 7.2

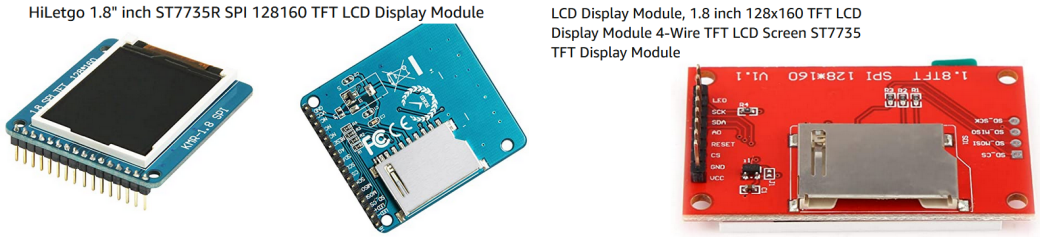


Figure 7.2. Alternative versions of the Sitronix ST7735 (all these are fine).

This lab will use "busy-wait" synchronization, which means before the software issues an output command to the LCD, it will wait until the display is not busy. In particular, the software will wait for the previous LCD command to complete. For the 10ms wait function needed in **IO.c**, we suggest you use the cycle-counting approach to blind wait (like Lab 2) instead of SysTick (like Lab 5) because you will need SysTick periodic interrupts for Labs 8, 9, and 10.

Make sure **char** data type is unsigned. Unclick "Plain Char is Signed" box, so **char** contains values 0 to 255.

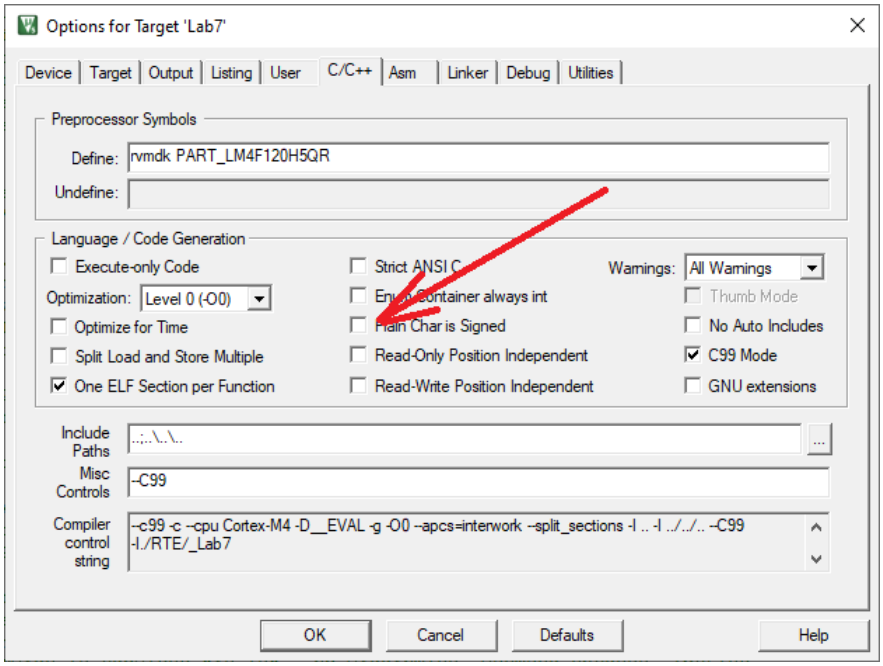


Figure 7.3. Compile options to use extended ASCII.

Part b - Write LCD Driver

First, write the description of the driver. Since this driver will be developed in assembly, your descriptions are placed in the comments before each subroutine. In Labs 8, 9, 10, we will call these driver functions from C, so we also place the function prototypes for the public functions in the header file. It is during the design phase of a project that this information is specified. The second component of a device driver is the implementation of the functions that perform the I/O. If the driver were being developed in C, then the implementations would have been placed in the corresponding code file. When developing a driver in assembly, the implementations are the instructions and comments placed inside the body of the subroutines.

In addition to public functions, a device driver can also have private functions. This interface will require a private function that outputs to commands to the LCD (notice that private functions do not include **`ST7735_`** in their names).In this lab, you are required to develop and test seven public functions (notice that public functions include **`ST7735_`** or **`IO_`** in their names). The third component is a main program that can used to test these functions. We have given you this main program, which you can find in the **Lab7Main.c** file. Please use the starter project which connects all these files.

In the **IO.c** file you will implement and test three functions to handle a user switch and heartbeat LED.

```
//------------IO_Init------------
// Initialize GPIO Port for a switch and an LED
// Input: none
// Output: none
void IO_Init(void)

//------------IO_HeartBeat------------
// Toggle the output state of the  LED.
// Input: none
// Output: none
void IO_HeartBeat(void)
```

```
//------------IO_Touch------------
// wait for release; delay for 20ms; and then wait for press
// Input: none
// Output: none
void IO_Touch(void)
```

In the **BusyWait.s** file you will implement and test two functions to communicate directly with the Sitronix ST7735 or LCD. You will not write the initialization ritual, because it is given in the **ST7735.c** file. However you will write these two functions that output to the LCD. Your **`SPIOutCommand`** function will be used to output 8-bit commands to the LCD, and your **`SPIOutData`** function will be used to output 8-bit data to the LCD. You do not need to create local variables on the stack for these two functions.

```
; This is a helper function that sends an 8-bit command to the LCD.
; Inputs: R0 = 32-bit command (number)
;     R1 = 32-bit SPI status register address
;     R2 = 32-bit SPI data register address
;     R3 = 32-bit GPIO port address for D/C
; Output: none
; Assumes: SSI0 and port A have already been initialized and enabled
SPIOutCommand
;1) Read SSI0_SR_R and check bit 4,
;2) If bit 4 is high, go to step 1 (wait for BUSY bit to be low)
;3) Clear D/C=PA6 to zero
;4) Write the command to SSI0_DR_R
;5) Read SSI0_SR_R and check bit 4,
;6) If bit 4 is high, go to step 5 (wait for BUSY bit to be low)

; This is a helper function that sends an 8-bit data to the LCD.
; Input: R0  8-bit data to transmit
;     R1 = 32-bit SPI status register address
;     R2 = 32-bit SPI data register address
;     R3 = 32-bit GPIO port address for D/C
; Output: none
; Assumes: SSI0 and port A have already been initialized and enabled
SPIOutData
;1) Read SSI0_SR_R and check bit 1,
;2) If bit 1 is low, go to step 1 (wait for TNF bit to be high)
;3) Set D/C=PA6 to one
;4) Write the 8-bit data to SSI0_DR_R
```

Question: *Why do these functions use R1-R3 and not simply access the registers directly?* ***Answer:*** *This allows the grader to perform extensive testing prior to using them for actual LCD I/O.*

Part c - Debug LCD Driver in Simulation

This lab is sufficiently complex that we require you to debug the two LCD.s functions on the simulator. If your **`SPIOutData`** and **`SPIOutCommand`** functions in **BusyWait.s** are correct, the main program will output the **welcome** message on LCD screen, as shown in Figure 7.4.

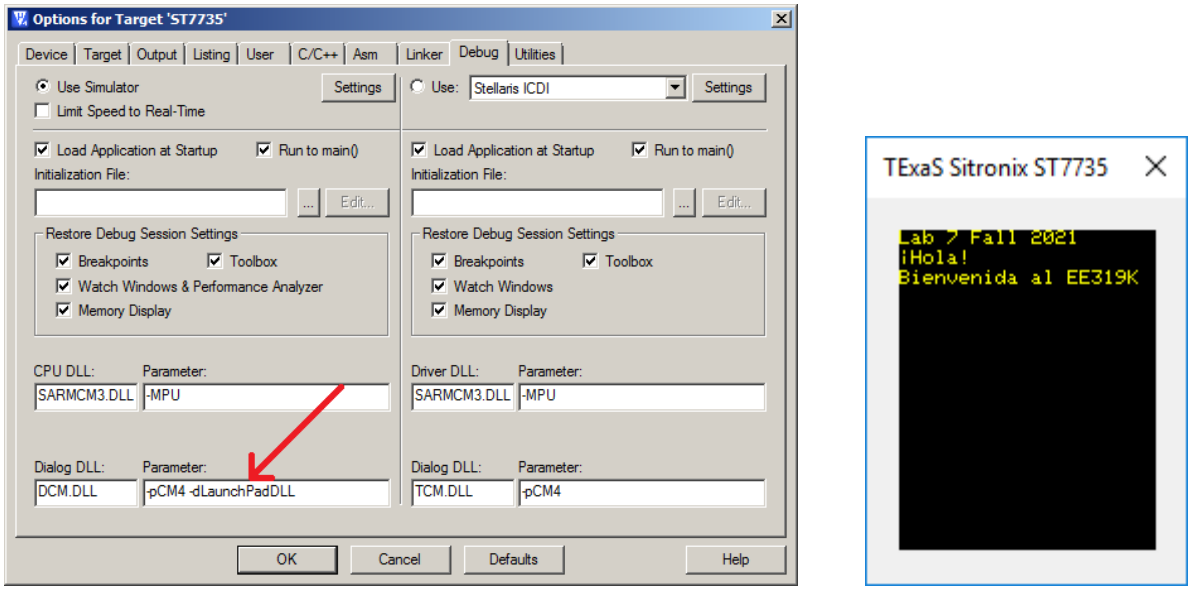


Figure 7.4. The simulation output showing LCD.s is operational. Your system should display Spring 2023.

Part d - Test Display Hardware

To test the hardware display, you can download and run the **ST7735_4C123** project in your EE319Kware folder.

Do not use the **ST7735_4C123** project to build your lab 7 solution, but it can be run to make sure your LCD is working and your wiring is correct.

Part e - Add decimal functions to LCD Driver

Implement the **Dec2String** and **Fix2String** functions (in **StringConversion.s**) which print a decimal number to the screen and a fixed point number to the screen, respectively. Both **`Dec2String`** and **`Fix2String`** functions **must use local variables on the stack with symbolic binding**. If you want a challenge, you should consider using recursion for OutDec. To observe the stack when assembly programs are running, set the memory view to observe the Stack Pointer (R13), and choose the type to be unsigned long. This will show you the top 16 elements on the stack.

Built on top of your **BusyWait.s** file is the C file **ST7735.c**, which implements 6 public functions that can be used to display characters and graphics on the display. *You should not modify the **ST7735.c** file*. One of the functions, **`ST7735_OutChar`**, outputs one ASCII character on the display. In your file **StringConversion.s**, you will implement and test two more display functions. Your **`Dec2String`** function will be used to output an unsigned 32-bit integer to the LCD, and your **`Fix2String`** function will be used to output an unsigned 32-bit fixed-point number to the LCD.

```
;----------------------Dec2String----------------------
; Convert a 32-bit number into unsigned decimal format
; String the string into the empty array add null-termination
; Input: R0 (call by value) 32-bit unsigned number
;     R1 pointer to empty array
; Output: none
; Invariables: This function must not permanently modify registers R4 to R11
Dec2String
```

You may implement **`Dec2String`** using **either iteration or recursion.** ECE319H students must use recursion. All ECE319K/ECE319H students must have at least one local variable, allocated on the stack, and use **symbolic binding** for the local. You can use SP or stack frame R11 for accessing the local variable. We do not recommend recursion for **Fix2String**.

Question: *Does* ***Dec2String*** *itself need to be recursive for ECE319H? There are two options for ECE319H, 1) make* ***Dec2String*** *recursive, or 2) write a helper function, called* ***OutDec****, that* ***Dec2String*** *calls. This second option involves implementing the following in assembly*

```
char *Pt; // global pointer
void Dec2String(uint32_t n, char *p){
  Pt = p;
  OutDec(n); // saves characters using the pointer Pt
  *Pt = 0;   // add null
}
```

Then, write the ***OutDec*** *function recursively. The local variables on the stack with symbolic binding should be included in* ***OutDec****,*

```
; ----------------------Fix2String----------------------
; Create characters for LCD display in fixed-point format
; unsigned decimal, resolution 0.001, range 0.000 to 9.999
; There needs to be a space after the fixed point number
;  but no space after *.***
; Inputs:  R0 is an unsigned 32-bit number
;          R1 pointer to empty array
; Outputs: none
; E.g., R0=0,  then create "0.000 "
;     R0=3,    then create "0.003 "
;     R0=89,   then create "0.089 "
;     R0=123,  then create "0.123 "
;     R0=9999, then create "9.999 "
;     R0>9999, then create "*.***"  ****no space here***
; Invariables: This function must not permanently modify registers R4 to R11
Fix2String
```

You must have at least one local variable, allocated on the stack, and use symbolic binding for the local.

Parameter	LCD display
0	0.000
1	0.001
999	0.999
1000	1.000
9999	9.999
10000 or more	*.***

Table 7.1. Specification for the **`Fix2String`** *function.*

An important factor in device driver design is to separate the interface (how to use the programs, which are defined in the comments placed at the top of each subroutine) from the mechanisms (how the programs are implemented, which are described in the comments placed within the body of the subroutine.)

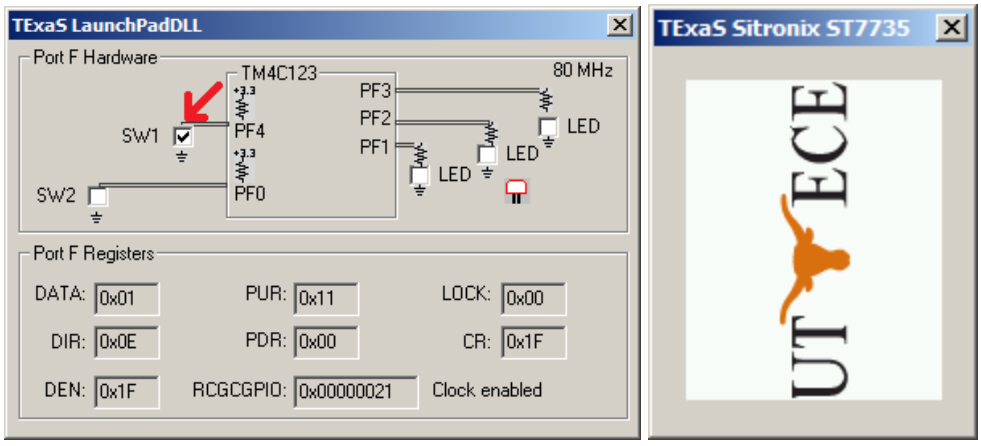
A main function that exercises your LCD driver's new decimal output facilities is provided in your project (**Lab7Main.c**). This software has two purposes. For the developer (you), it provides a means to test the driver functions. It should illustrate the full range of features available with the system. The second purpose of the main program is to give your client or customer (e.g., the TA) examples of how to use your driver. You can modify/comment this main function to help you debug your code. Your modification will not be part of your deliverable.

Part f - Debug Decimal Functions in Simulation

Debug your **`Fix2String`** and **`Dec2String`** functions in the simulator. *We recommend you test* ***`Dec2String`*** *and* ***`Fix2String`*** *prior to running the grader. Run something like this and single step through your code*.

```
char Test[20];
int main(void){
  Dec2String(123,Test);
  Fix2String(123,Test);
  while(1){
  }
}
```

When **StringConversion.s** is correct, the LCD will show both a decimal number and a fixed point number, see Figure 7.5. The **`TestData`** array in the **Lab7Main.s** file lists the test values for **StringConversion.s**.



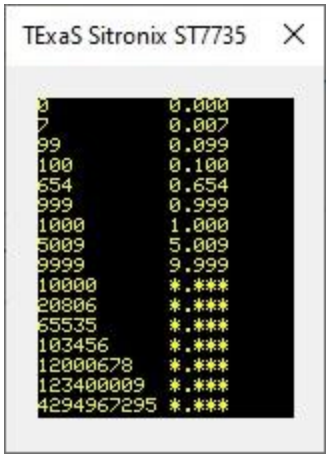


Figure 7.5. The simulation output showing StringConversion.s; press and release PF4 to advance output.

Part g - Test Decimal Functions in Hardware

After your functions are tested in simulation, you will then test them on the real board. See Figure 7.6.

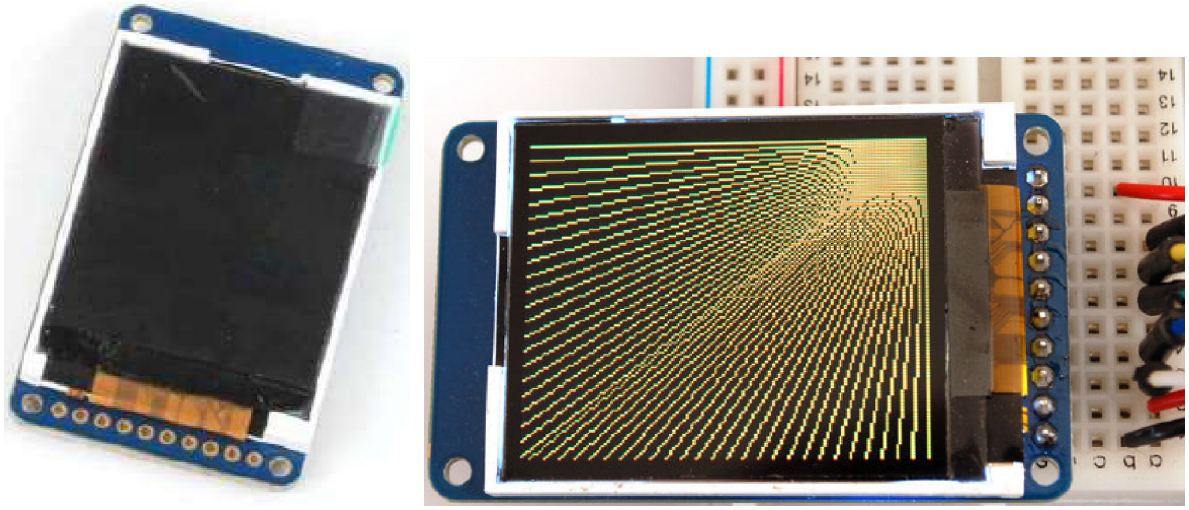


Figure 7.6. The Sitronix ST7735R is a low-cost LCD with 160x128 pixels.

Demonstration

(both partners must be present, and demonstration grades for partners may be different)

There are grading sheets for every lab so you know exactly how you will be evaluated. You will also be required to demonstrate the proper operation on the actual microcontroller. During demonstration to the TA, you will run your system in the debugger and show the binding, allocation/initialization, access and deallocation of the local variables. Each time a function is called, an **activation record** is created on the stack, which includes parameters passed on the stack (none in this lab), registered saved, and the local variables. You will be asked to observe the stack in the

debugger and identify the activation records created during the execution of **`Dec2String`**. TAs may ask you questions on LCD interfacing, and programming.

Do all these well in advance of your checkout

1. **Signup for a time with a TA. If you have a partner, then both must be present**
2. **Upload your software to canvas, make sure your names are on all your software**
3. **Upload your one pdf with deliverables to Canvas**

Do all these during the TA checkout meeting

1. **Have your one pdf with deliverables open on your computer so it can be shared**
2. **Have Keil Lab 7 open so TA can ask about your code**
3. **Start promptly, because we are on a schedule. If you have a partner, then both must be present**
4. **Demonstrate lab to TA**
5. **Answer questions from TA to determine your understanding**
6. **TA tells you your score (later the TA will upload scores to Canvas)**

Deliverables

1. Your name, professor, and EID.
2. All files that you have changed or added (**BusyWait.s, IO.c** and **StringConversion.s**) into a Lab7.c *and upload this Lab7.c file to Canvas*.
3. Optional Feedback : http://goo.gl/forms/rBsP9NTxSy

Hints

1. This will run in simulation mode, so you can test your low-level functions, which should output the **`"Lab 7, Welcome to 319K!"`** message on LCD screen.
2. It will be important to pass data into your functions in Register R0; this way you will be able to call your functions from C code later in Labs 8, 9, and 10. It is important to save and restore Registers R4-R11 if you use them.
3. You must use the cycle-counting approach to implement the blind waits (Lab 2) instead of SysTick (Lab 5) because you will need SysTick periodic interrupts for Labs 7, 8, and 9.
4. There is blind-cycle synchronization in ST7735.c using a call to the function **`Delay1ms`**

FAQ

1. **What does SSI0_SR_R do?**

   There is a serial interface module on the chip, and SSI0_SR_R is the status register for it. It can tell you whether data is currently being transmitted and received, and if the Transmit and Receive FIFO's are full or empty (to send and receive data with the serial module, you read/write information to/from hardware FIFO's and then the module will take care of sending the data for you).

2. **Where are we supposed to use 10ms Wait function?**

   Wait10ms will be used in IO_Touch to debounce your switches. You will write this private function inside IO.c. Use the simple while loop like Lab 3 and not the SysTick like Lab 5

3. **Does anyone know in what manner we are expected to connect the board to the LCD? Are we meant to solder directly to the LCD or are we meant to use the header? I'm not getting**

a very good connection between the LCD and the header so I don't know how I could go about using the header as my primary connection, but I also don't want to destroy the LCD if I should not be soldering directly to it.

You should solder the header on to the LCD so you can plug it in onto your breadboard. Please ask the instructor or TA if you are unsure.

4. **The starter files mention setting up a switch. What exactly is this switch used for?**

   The switch, which is switch SW1 and on Port F, is used to change through the initialization screens. The screen should change from "Lab7! ..." to the longhorn when the switch is pressed and released, and then to your code when pressed and released again.

5. **For OutDec and OutFix, are we supposed to put our final result back into R0 as the output? I am confused as to how we should be "returning" the string of ASCII values that we created in OutDec and OutFix.**

   R1 is a pointer to an empty string. Store the characters in that buffer (null terminated).

6. **Our simulation works perfectly but when we attempt to run our code on hardware the screen stays blank (white). The test program works perfectly fine though. Any suggestions?**

   Check the connections. If you didn't solder and try to use electrical tape or wrapping the wires, the connections are not secured, so your LCD screen isn't being powered. You need to solder the pins to the LCD module. If you used solder, make sure none of your solder clumps are touching. These create additional connections which interfere with the signals. Some other people have had issues with using the incorrect register in write command or write data. Check those two functions and ensure they are exactly what it should be--if the test program works, then these are the most likely function culprits.

7. **Can someone explain how Dec2String and Fix2String are used? I'm very confused..and where does the stack take place in this?**

   Look at ST7735_OutUDec in ST7735.c. ST7735_OutUDec calls your Dec2String, and then outputs the string to the LCD. You use the stack for local variables

8. **How do I fix this error?**

   \ST7735.axf: Error: L6238E: print.o(.text) contains invalid call from '~PRES8

   (The user did not require code to preserve 8-byte alignment of 8-byte data objects)' function to 'REQ8 (Code was permitted to depend on the 8-byte alignment of 8-byte data items)' function ST7735_OutChar.

   Working with C code requires you to have an even number of registers when working with the stack using the PUSH and POP instruction. There are two solutions to this: 1) Make

sure you are always PUSHing and POPing even number of registers or, 2) include a PRESERVE8 directive under your AREA CODE for the assembler to handle this for you.

Extra credit for Lab 7: you may do either but not both extra credits

Option 1: 5 points extra credit Study how the font table works in **ST7735.c**. Each ASCII character from 0 to 254 is encoded as a 5-wide by 7-tall pixel image. See **static const uint8_t Font[]**. Notice all the data have bit 7 clear; this will create a 1-pixel separation between characters on different rows of the display. Notice the font images are stored upside down (bit 0 is the top, and bit 6 is the bottom). Find a foreign language character that will be recognizable, displayed in 5 by 7 pixel format. Draw this character as a 5 by 7 image. Convert this image into 5 8-bit bytes (bit 7 must be clear), and replace the one of the lines of **Font[]** with the new image (the first line is n=0, and the last is n=254). You can test the new character by calling **ST7735_OutChar(n)**, where n is the line number in **Font[]** that your replaced. If you replace one of the lines 128 to 254, open the Keil options, go to C/C++ and uncheck *Plain Char is Signed* (so **char** will be unsigned).

Option 2: 5 points extra credit Study how **ST7735_DrawBitmap** works in **ST7735.c**. Draw an image with at least 3 colors, 6 or more pixels high, and more than 6 or more pixels wide. Look at the 16-bit colors in **ST7735.h**. Manually create the data structure to store the image. Do not use Paint and BMPconvert.exe to automatically create the data structure. For example, here is a 5 wide by 6 tall image. Notice the 2-d image is stored as a linear array. Notice the data is stored upside down. Notice each pixel is a 16-bit number encoded in 5-6-5 RGB format.

```
// test image2
// [red]   [yellow]  [green]   [blue]      [black]
// [red]   [yellow]  [green]   [blue]      [white]
// [red]   [yellow]  [green]   [blue]      [black]
// [red]   [yellow]  [green]   [blue]      [white]
// [red]   [yellow]  [green]   [blue]      [black]
// [black] [75%grey] [50%grey] [25%grey] [white]
const uint16_t Test2[] = {
  0x0000, 0x4208, 0x8410, 0xC618, 0xFFFF,
  0x001F, 0x07FF, 0x07E0, 0xF800, 0x0000,
  0x001F, 0x07FF, 0x07E0, 0xF800, 0xFFFF,
  0x001F, 0x07FF, 0x07E0, 0xF800, 0x0000,
  0x001F, 0x07FF, 0x07E0, 0xF800, 0xFFFF,
  0x001F, 0x07FF, 0x07E0, 0xF800, 0x0000
};
```

To output this image in the middle of the display, we can execute

ST7735_DrawBitmap(64,80,Test2,5,6);

For more information see the comments in **ST7735_DrawBitmap**.