Lab 7C. LCD Device Driver for the Sitronix ST7735

Preparation

Read Sections 7.1, 7.2, 7.3, 7.4, 7.5, 7.6, and 7.7
Read the data sheet for the LCD display (in project folder)
1-8-tft-display.pdf, ST7735R_V0.2.pdf
Lab7C starter project in SVN

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 use indexed addressing to access strings;
3. to learn how to design implement and test a device driver using busy-wait synchronization;
4. to learn how to allocate and use local variables on the stack;
5. to use fixed-point numbers to store non-integer values.

In Spring 2015, we switched from the monochrome Nokia to the color ST7735 because it will make the Lab 10 game more exciting. The LaunchPad.DLL will simulate the Sitronix ST7735, even though Lab 7C will not have an automatic grader.

Figure 7.1. Sitronix ST7735

System Requirements

In this lab you will create an embedded system that:

- Interfaces to the ST7735 LCD
- Displays numbers in decimal format
- Displays fixed point numbers in decimal format

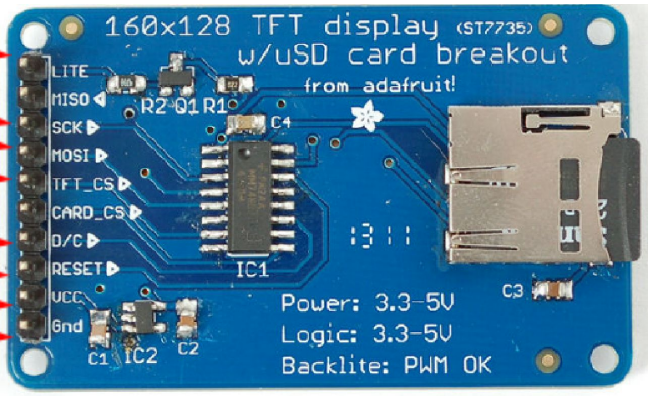
Procedure

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 **`LCD_OutDec`** or **`LCD_OutFix`** 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.

Part a - Understand Hardware

In this lab you will interface a Sitronix ST7735 LCD to the TM4C123 as shown in Program 7.1.

```
// 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.

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, 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.

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 **LCD.h**. 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, e.g., **LCD.C**. 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 **`LCD_`** in their names).In this lab, you are required to develop and test seven public functions (notice that public functions include **`LCD_`** 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.s** file. Please use the starter project which connects all these files.

In the **IO.s** 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
; Invariables: This function must not permanently modify registers R4 to R11

;------------IO_HeartBeat------------
; Toggle the output state of the LED.
; Input: none
; Output: none
; Invariables: This function must not permanently modify registers R4 to R11
IO_HeartBeat

;------------IO_Touch------------
; First, wait for the key to be released, and then
;        wait for the key to be touched
; Input: none
; Output: none
; This is a public function
; Invariables: This function must not permanently modify registers R4 to R11
```

In the **LCD.s** file you will implement and test two functions to communicate directly with the Sitronix ST7735 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 **`writecommand`** function will be used to output 8-bit commands to the LCD, and your **`writedata`** function will be used to output 8-bit data to the LCD.

```
; This is a helper function that sends an 8-bit command to the LCD.
; Input: R0  8-bit command to transmit
; Output: none
; Assumes: SSI0 and port A have already been initialized and enabled
writecommand
;1) Read SSI0_SR_R and check bit 4,
;2) If bit 4 is high, loop back 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, loop back 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
; Output: none
```

```
; Assumes: SSI0 and port A have already been initialized and enabled
writedata
;1) Read SSI0_SR_R and check bit 1,
;2) If bit 1 is low loop back 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
```

Part c - Debug LCD Driver in Simulation

This lab is sufficiently complex that we require you to debug the three LCD.s programs on the simulator. If your **`writedata`** and **`writecommand`** functions in **LCD.s** are correct, the main program will output the **`"Lab 7, welcome to 319K!"`** message on LCD screen, as shown in Figure 7.2.

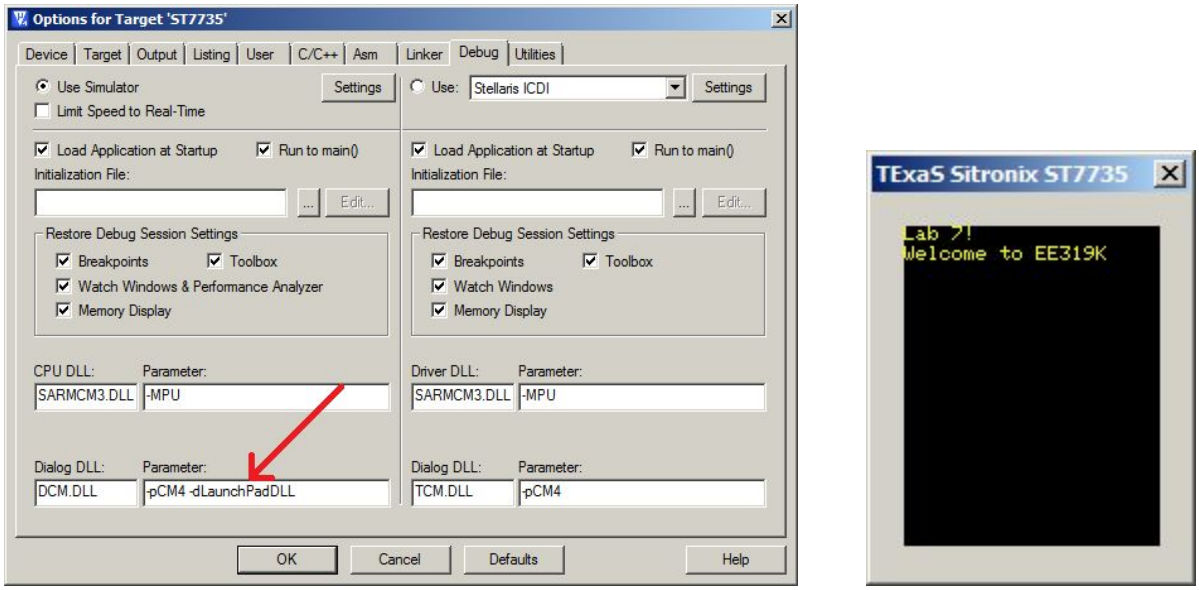


Figure 7.2. The simulation output showing LCD.s is operational.

Part d - Test Display Hardware

To test the hardware display, you can download and run the example **:**

http://users.ece.utexas.edu/~valvano/arm/ST7735_4C123.zip

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 **LCD_OutDec** and **LCD_OutFix** functions which print a decimal number to the screen and a fixed point number to the screen, respectively. Your **`LCD_OutDec`** and **`LCD_OutFix`** functions **must use local variables on the stack with binding**. To observe the stack when assembly programs are running, set the memory view to observe 0x200003c0, and the type to be unsigned long. This will showyou to top 16 elements on the stack.

Built on top of your **LCD.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 **print.s**, you will implement and test

two more display functions. Your **LCD_OutDec** function will be used to output an unsigned 32-bit integer to the LCD, and your **LCD_OutFix** function will be used to output an unsigned 32-bit fixed-point number to the LCD.

```
;-----------------------LCD_OutDec-----------------------
; Output a 32-bit number in unsigned decimal format
; Input: R0 (call by value) 32-bit unsigned number
; Output: none
; Invariables: This function must not permanently modify registers R4 to R11
```

LCD_OutDec (R0 is a 32-bit **number**)

You may implement this using either iteration or recursion. You must have at least one local variable, allocated on the stack, and use symbolic binding for the local. If you use recursion, the base case (n<10) requires one call to **ST7735_OutChar**. You can use SP or stack frame register for accessing the local variable.

```
; ----------------------LCD_OutFix----------------------
; Output characters to LCD display in fixed-point format
; unsigned decimal, resolution 0.001, range 0.000 to 9.999
; Inputs:  R0 is an unsigned 32-bit number
; Outputs: none
; E.g., R0=0,    then output "0.000 "
;       R0=3,    then output "0.003 "
;       R0=89,   then output "0.089 "
;       R0=123,  then output "0.123 "
;       R0=9999, then output "9.999 "
;       R0>9999, then output "*.*** "
; Invariables: This function must not permanently modify registers R4 to R11
```

LCD_OutFix (R0 is a 32-bit **number**)

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 **LCD_OutFix** *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 SVN folder. 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 **LCD_OutFix** and **LCD_OutDec** functions in the simulator. When **print.s** is correct, the LCD will show both a decimal number and a fixed point number, see Figure 7.3. The **`TestData`** array in the **Lab7Main.s** file lists the test values for **print.s**.

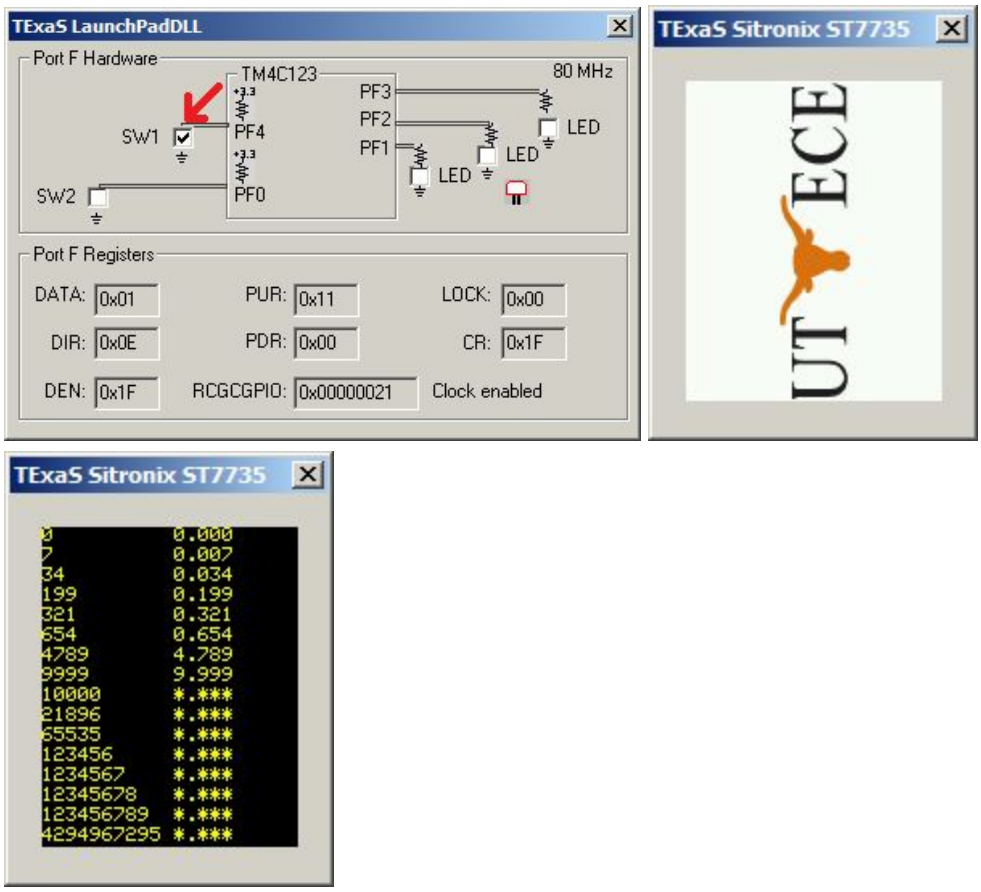


Figure 7.3. The simulation output showing print.s is operation; push PF4 to advance output.

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 7.4.

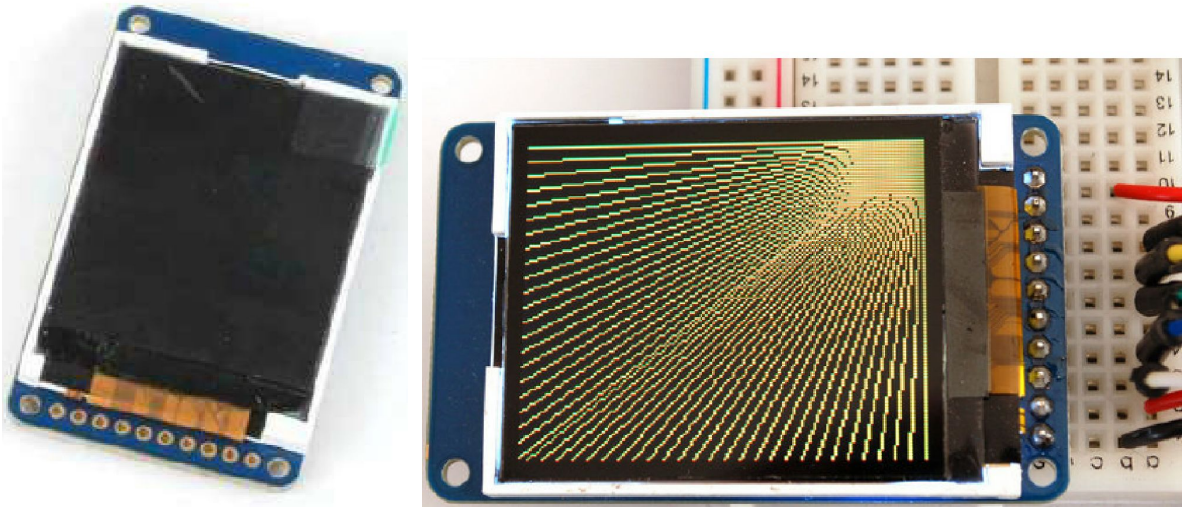


Figure 7.4. The Sitronix ST7735 is a low-cost LCD with 160x128 pixels.

Demonstration

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

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 **`LCD_OutDec`**. TAs may ask you questions on LCD interfacing, and programming. What is the difference between post and pre-increment modes of addressing? What does the D/C signal line on the LCD signify? What does busy-wait synchronization mean in the context of communication with the LCD? The TA will ask to see your implementation local variables and ask you to explain the four step process (binding, allocation, access and deallocation). You should be able to draw stack pictures. How does AAPCS apply to this lab? Why is AAPCS important?

Deliverables

1. Lab 7 grading sheet. You can print it yourself or pick up a copy in lab. You fill out the information at the top.
2. All source files that you have changed or added (**LCD.s, IO.s** and **print.s**) should be committed to SVN. Please do not commit other file types.
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. Because there is a blind cycle synchronization in ST7735.c, `Delay1ms` so please leave the compiler optimization on **Level 0** and do not click **Optimize for Time**

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?**

   delay10ms will be used in IO_Touch to debounce your switches. You can call the function with a BL instruction.

3. **Is there supposed to be a lab7main.s file in our lab7 folder? I seem to be missing it.**

   No, the "main" file for this lab is ST7735TestMain.c

4. **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.

5. **Is the third part (print.s) supposed to only appear for a small moment?**

   The output of your program is up to you, but the decimal value display should be long enough or repeat so that a TA can observe and evaluate your screen. You have initialized a switch, so ideally you could write a code that advances through your demonstration as you press it.

   You can keep the numbers stay on the screen until next time you press the switch by adding an extra IO_touch(); in the if statement in main():

```
//...
i++;
if(i==16){
      IO_Touch(); //<--- add this extra line here
       ST7735_FillScreen(0);
//...
```

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

The switch, which is switch one 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.

7. **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.**

   Rather than return a value you just need to call OutChar to display each integer to the screen (i.e. 123->'1' '2' '3'). For OutString to work with local variables on the stack, you would have to make sure they are null terminated, in sequential memory locations (which if you're using a double word aligned stack they likely won't be between function calls), and byte packed (meaning when you push, you 're not pushing a single character but four at a time since you're pushing a 32-bit word by default). This is assuming a recursive solution to OutDec though. If you create an iterative solution, then you would be able to allocate a char array on the stack as a local variable and output that value using OutString by following the other points listed above (although calling OutChar each time you convert a decimal place to ASCII would be less overhead than placing the value into an array and then calling OutString). It depends on your implementation however and is up to you!

8. **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. 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.

9. **Can someone explain how we can do this OutDec and OutFix part? I'm very confused..and where does the stack take place in this?**

   There are two functions, ST7735_OutChar and ST7735_OutString that you can use to output characters and strings on to the LCD screen in ST7735.c. To make use of these two functions, you need to somehow convert the 32bit input number into chars and pass them in as input parameters (think AAPCS standards). If you use recursive to implement OutDec and OutFix, stack will be your friend. Even if you don't, you will probably be using more registers than just R0 - R3 so you end up having to push/pop some registers onto/from the stack. We want you guys to become familiar with stack, stack frame, local variables etc. so please follow the instruction use local variables, symbolic binding when implementing these two functions.

10. **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.