This laboratory assignment accompanies the book, [*Embedded Systems: Real-Time Interfacing to ARM Cortex M Microcontrollers, ISBN-13: 978-1463590154*](https://www.amazon.com/Embedded-Systems-Real-Time-Interfacing-Microcontrollers/dp/1463590156), by Jonathan W. Valvano, copyright © 2021.

## Goals

• To introduce the lab equipment

• To familiarize yourself with Keil uVision4 for the ARM Cortex M processor

• To develop a set of useful fixed-point output routines.

## Review

• “How to program…” section located at the beginning of this laboratory manual,

• Read "Embedded Software in C for an ARM Cortex M"  [here](http://users.ece.utexas.edu/~valvano/embed/toc1.htm)

• Valvano Chapters 1, 2 and 3 from the book Embedded Systems: Real Time Interfacing,

• **Lab1.c**, **style.pdf**, **style\_policy.pdf** and **c\_and\_h\_files.pdf** guide

## Starter files

• **ST7735\_4C123** project – located in ValvanoWare

• [http://users.ece.utexas.edu/%7Evalvano/EE445L/downloads.htm](http://users.ece.utexas.edu/~valvano/EE445L/downloads.htm)

• [http://users.ece.utexas.edu/%7Evalvano/EE445L/downloads.htm#Keil](http://users.ece.utexas.edu/~valvano/EE445L/downloads.htm" \l "Keil)

• <http://users.ece.utexas.edu/~valvano/arm/ValvanoWareTM4C123v5.zip>

**Team Size: 1 (each student performs Lab 1 on their own)**

## Required Hardware

EK-TM4C123GXL [http://www.ti.com](http://www.ti.com/tool/ek-tm4c123gxl?keyMatch=tm4c123g&tisearch=Search-EN-Everything)  $16.99

Sitronix ST7735 Color LCD <http://www.adafruit.com/products/358> $19.99

## Background

The objectives of this lab are to introduce the TM4C123 programming environment and to develop a set of useful graphic routines that will be used in the subsequent labs. A **software module** is a set of related functions that implement a complete task. In particular, you will write software that plots data on the LCD graphics screen.

## Specifications

You will develop and test these four functions

**ST7735\_sDecOut3** Signed 32-bit decimal fixed-point Δ = 0.001

**ST7735\_uBinOut5** Unsigned 32-bit binary fixed-point Δ = 1/32

**ST7735\_XYplotInit** Specify the X and Y axes for an x-y scatter plot

**ST7735\_XYplot** Plot an array of (x,y) data

The first format you will handle is signed 32-bit decimal fixed-point with a resolution of 0.001. The full-scale range is from -99.999 to +99.999. Any integer part outside the range of -99999 to +99999 signifies an error. The **ST7735\_sDecOut3** function takes a signed 32-bit integer part of the fixed-point number and outputs the fixed-point value on the display. The specifications of **ST7735\_sDecOut3** are illustrated in Table 1.1. In order to make the display output pretty it is required that all output commands produce exactly 6 characters. This way the decimal point is always drawn in the exact same location, independent of the number being displayed.

|  |  |
| --- | --- |
| Parameter | Display |
| **0** | **0.000** |
| **4** | **0.004** |
| **-5** | **-0.005** |
| **78** | **0.078** |
| **-1254** | **-1.254** |
| **9999** | **9.999** |
| **-10000** | **-10.000** |
| **12345** | **12.345** |
| **-56789** | **-56.789** |
| **-99999** | **-99.999** |
| **99999** | **99.999** |
| **100000** | **\*\*.\*\*\*** |
| **-100000** | **-\*\*.\*\*\*** |

*Table 1.1. Specification for the* **ST7735\_sDecOut3** *function.*

The second format you will handle is unsigned 32-bit binary fixed-point with a resolution of 1/32. The full-scale range is from 0 to 999.99. If the integer part is larger than 31999, it signifies an error. The **ST7735\_uBinOut5** function takes an unsigned 32-bit integer part of the binary fixed-point number and outputs the fixed-point value on the display. The specifications of **ST7735\_uBinOut5** are illustrated in Table 1.2. It is ok if the least significant digit is off a little due to rounding

|  |  |
| --- | --- |
| Parameter | Display |
| **0** | **0.00** |
| **1** | **0.03** |
| **5** | **0.16** |
| **100** | **3.13** |
| **127** | **3.97** |
| **252** | **7.88** |
| **535** | **16.72** |
| **2560** | **80.00** |
| **6092** | **190.38** |
| **13000** | **406.25** |
| **16383** | **511.97** |
| **17283** | **540.09** |
| **31999** | **999.97** |
| **32000** | **\*\*\*.\*\*** |

*Table 1.2. Specification for the* **ST7735\_uBinOut5** *function.*

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*ST7735\_XYplotInit\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Specify the X and Y axes for an x-y scatter plot

Draw the title and clear the plot area

Inputs: title ASCII string to label the plot, null-termination

minX smallest X data value allowed, integer

maxX largest X data value allowed, integer

minY smallest Y data value allowed, integer

maxY largest Y data value allowed, integer

bcolor 16-bit color of the background

Outputs: none

assumes minX < maxX, and minY < maxY

Example Plot X = 0 to 1500, Y = 0 to 127

\*/

void ST7735\_XYplotInit(char \*title, uint32\_t minX, uint32\_t maxX,

uint32\_t minY, uint32\_t maxY);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*ST7735\_XYplot\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Plot an array of (x,y) data

Inputs: num number of data points in the two arrays

bufX array of 32-bit integer data

bufY array of 32-bit integer data

color 16-bit color of the data points

Outputs: none

assumes ST7735\_XYplotInit has been previously called

neglect any points outside the minX maxY minY maxY bounds

\*/

void ST7735\_XYplot(uint32\_t num, uint32\_t bufX[],

uint32\_t bufY[], uint16\_t color);

You should call other LCD graphics programs within the LCD graphics module. Your functions will draw an X-Y scatter plot on the LCD. Before designing your X-Y plot software, run the **ST7735\_4C123** starter project and review the existing functions available within the **ST7735.c** module. Your X-Y scatter plot will be given all the data at the same time. All data for this plotting feature will use 32-bit unsigned binary fixed-point format with resolution of 1/32. The **ST7735\_XYplotInit** function will configure the plot and clear the drawing area. The **ST7735\_XYplot** function plots the X-Y scatter data. You are allowed to modify the syntax of these functions as long as it is useful for pixel graphics plotting.

## Preparation (do this before your lab period)

*Preparation is due before the lab period starts and must be shown to the TA at the start of Lab.* In general, preparation involves designing all hardware and software needed for that lab. More specifically you need to type all software into the computer to the point at which the software will compile. For the hardware, you need to gather all the parts needed to build the circuits. In Lab 1, preparation is writing the C code implementation for these four functions:

* **ST7735\_sDecOut3**
* **ST7735\_uBinOut5**
* **ST7735\_XYplotInit**
* **ST7735\_XYplot**

You should design the four functions, type the C code into Keil, and compile the project. As part of preparation, you do not need to run or debug the software functions. Show the preparation to the TA at the start of lab.

Rather than creating a new project, we suggest you make a copy of the existing ST7735 project and rename it Lab 1. You will need to **add your code into an appropriate place within the project**. “Syntax-error-free” software is required as preparation. Doing the preparation before lab allows you to debug with the TA present in the lab. The proper approach to EE445L is to design and edit before lab, and then you can build your hardware and debug your software during lab. Document clearly the operation of the routines. The comments included in **header files** are intended for the client (programmers that will use your functions.) The comments included in **code files** are intended for the coworkers (programmers that will debug/modify your functions.) Your main program will be used to test the functions. It is important for you to learn how to use the interactive features of the uVision5 debugger.

You are free to develop a main program that tests your function. You may implement this test in whatever style you wish. You may create your own or edit the starter **Lab1.c**. This main program has three important roles. First, you will use it to test all the features of your program. Second, a judge in a lawsuit can subpoena this file. In a legal sense, this file documents to the world the extent to which you verified the correctness of your program. When one of your programs fails in the marketplace, and you get sued for damages, your degree of liability depends on whether you took all the usual and necessary steps to test your software, and the error was unfortunate but unforeseeable, or whether you rushed the product to market without the appropriate amount of testing and the error was a foreseeable consequence of your greed and incompetence. Third, if you were to sell your software package (**fixed.c, fixed.h,** **ST7735.c** and **ST7735.h**), your customer can use this file to understand how to use your package, its range of functions, and its limitations.

## Procedure (do this during your lab period)

If you do not have the board and the ST7735 LCD, you can perform this lab in simulation. Copy this file

<https://www.dropbox.com/s/azig59lbopm1nmb/LaunchPadDLL.dll?dl=1> into this in \Keil\ARM\BIN folder.

**0.** If you bought your board used, ask the TA how to run the tester project.

**1.** Run these example projects as a review of EE319K:

PeriodicSysTickInts\_4C123 (review periodic interrupts)

EdgeInterrupt\_4C123 (review input interrupts and notice the extra counts from switch bounce)

ADCSWTrigger\_4C123 (review periodic timer interrupts and busy-wait ADC)

UARTInts\_4C123 (review FIFO queues, input/output interrupts)

The UART projects produce output that can be seen on PuTTY. The SysTick and ADC projects will toggle the LEDs. The EdgeInterrupt example counts the number of times you push the switch connected to PF4.

**Using PeriodicSysTickInts, EdgeInterrupt see how TExaSdisplay can be used to perform hardware debugging with a logic analyzer. Using ADCSWTrigger see how TExaSdisplay can be used to perform hardware debugging with an oscilloscope.**

**2.** Experiment with the different features of uVision5 and its debugger. Familiarize yourself with the various options and features available in the editor/assembler/terminal. Edit, compile, download, and run your project working through all aspects of software development. In particular:

• learn how to remove all tabs (Edit->Advanced->UntabifySelection);

• learn how to comment and uncomment large sections of code;

• know how to compile, download, and debug a project;

• in the debugger run with and without View->PeriodicWindowUpdate;

• in the debugger observe assembly listings visualizing the correlation between C and assembly;

• set breakpoints, add global variables to watch window, observe I/O device registers in debugger;

**3.** Debug your software modules. If you are having a lot of trouble debugging, we suggest you debug the software on the simulator. This is the only lab that will run on the simulator, so it is not necessary to set up simulation for this class.

**4.** Use SysTick to profile execution speed of **ST7735\_uBinOut5**. More specifically, run this code segment and look at **ElapsedTime** and **ElapsedTime2** (ECE319K Lab 8 review). On the simulator, I got a **ST7735\_uBinOut** measurement time of 1733us and a **ST7735\_OutString** measurement time of 1702us. On the real board, my times were 5308us and 5277us respectively.

uint32\_t startTime,stopTime; // in 12.5ns

uint32\_t ElapsedTime,ElapsedTime2; // in usec

int main(void){

PLL\_Init(Bus80MHz); // Bus clock is 80 MHz

ST7735\_InitR(INITR\_REDTAB);

NVIC\_ST\_RELOAD\_R = 0x00FFFFFF; // maximum reload value

NVIC\_ST\_CURRENT\_R = 0; // any write to current clears it

NVIC\_ST\_CTRL\_R = 5;

startTime = NVIC\_ST\_CURRENT\_R;

ST7735\_uBinOut5(16383); // output 511.97

stopTime = NVIC\_ST\_CURRENT\_R;

ElapsedTime = ((startTime-stopTime)&0x0FFFFFF)/80; // usec

startTime = NVIC\_ST\_CURRENT\_R;

ST7735\_OutString("511.97"); // output 511.97

stopTime = NVIC\_ST\_CURRENT\_R;

ElapsedTime2 = ((startTime-stopTime)&0x0FFFFFF)/80; // usec

while(1){}

}

## Deliverables (exact components of the lab report)

A) Objectives (1/2 page maximum)

B) Software Design (upload one file with your header, code, and test components to Canvas as instructed by your TA)

C) Analysis and Discussion (1 page maximum). In particular, answer these questions

1) When should you use fixed-point over floating point? When should you use floating-point over fixed-point?

2) Give an example application (not mentioned in the book) for fixed-point. Describe the problem and choose an appropriate fixed-point format. (no software implementation required).

3) Can we use floating point on the ARM Cortex M4? If so, what is the cost?

4) Compare your **ElapsedTime** and **ElapsedTime2** measurements with mine. Were your times approximately the same, or significantly different from mine? Do you think execution is *I/O bound* (speed limited by the output rate to the LCD) or *CPU bound* (speed limited by software algorithm and CortexM running at 80 MHz)? Why?

**Fun activity.** Before you answer the *CPU* or *I/O Bound* question, go around the lab room, and exchange measurement data with other groups.

Text

Description automatically generated Text

Description automatically generated

A picture containing screenshot

Description automatically generated A screenshot of a cell phone

Description automatically generated

**10 points Extra credit** Perform an empirical study to evaluate four implementations on the Cortex M4. Two implements use fixed-point, two use floating-point, two are written in assembly, and two are written in C. For each implementation measure the total execution time. Make conclusions about implementing arithmetic operations on the Cortex M4. We recommend you use the example project with **Float** in its name to do this part.

// version 1: C floating point

// run with compiler options selected for floating-point hardware

volatile float T; // temperature in C

volatile uint32\_t N; // 12-bit ADC value

void Test1(void) {

for(N=0; N<4096; N++){

T = 10.0 + 0.009768 \* N;

}

}

// version 2: C fixed-point

volatile uint32\_t T; // temperature in 0.01 C

volatile uint32\_t N; // 12-bit ADC value

void Test2(void){

for(N=0; N<4096; N++){

T = 1000+ (125\*N+64)>>7;

}

}

; Version 3 assembly floating point

; run with floating-point hardware active

AREA DATA, ALIGN=2

T SPACE 4

N SPACE 4

AREA |.text|, CODE, READONLY, ALIGN=2

THUMB

Test3

MOV R0,#0

LDR R1,=N ;pointer to N

LDR R2,=T ;pointer to T

VLDR.F32 S1,=0.009768

VLDR.F32 S2,=10

loop3 STR R0,[R1] ; N is volatile

VMOV.F32 S0,R0

VCVT.F32.U32 S0,S0 ; S0 has N

VMUL.F32 S0,S0,S1 ; N\*0.09768

VADD.F32 S0,S0,S2 ; 10+N\*0.0968

VSTR.F32 S0,[R2] ; T=10+N\*0.0968

ADD R0,R0,#1

CMP R0,#4096

BNE loop3

BX LR

; version 4, assembly fixed point

AREA DATA, ALIGN=2

T SPACE 4

N SPACE 4

AREA |.text|, CODE, READONLY, ALIGN=2

THUMB

Test4 PUSH {R4,R5,R6,LR}

MOV R0,#0

LDR R1,=N ;pointer to N

LDR R2,=T ;pointer to T

MOV R3,#125

MOV R4,#64

MOV R5,#1000

loop4 STR R0,[R1] ; N is volatile

MUL R6,R0,R3 ; N\*125

ADD R6,R6,R4 ; N\*125+64

LSR R6,R6,#7 ; (N\*125+64)/128

ADD R6,R6,R5 ; 1000+(N\*125+64)/128

STR R6,[R2] ; T = 1000+(N\*125+64)/128

ADD R0,R0,#1

CMP R0,#4096

BNE loop4

POP {R4,R5,R6,PC}

## Checkout (show this to the TA)

You should be able to demonstrate correct operation of each routine:

* Show the TA you know how to observe global variables and I/O ports using the debugger;
* Demonstrate to the TA you know how to observe assembly language code;
* Verify proper input/output parameters when calling a function;
* Verify the proper handling of illegal formats;
* Demonstrate your software does not crash.

Read the **style.pdf**, **style\_policy.pdf** and **c\_and\_h\_files.pdf** from the lab manual website, and be prepared to answer a few questions.

## Hints

1) Do not create a new project. Start with a project you know works, make a copy of the project and, then make small changes. It is good practice to make a small change and test it. Once you have some new code that works, make a back-up, so that when you add something that doesn’t work, you can go back to a previous working version and try a new approach. Please add documentation that makes it easier to change and use in the future. Your job is to organize these routines to facilitate subsequent laboratories.

2) It is also good practice to look at the assembly language created by the compiler to verify the appropriate function. Analyzing the assembly listing files is an excellent way to double-check if your software will perform the intended function. This is especially true when overflow, dropout, and execution speed are important. We have not found any bugs with this compiler. Most reported compiler bugs (my program doesn’t do what I want) turn out to be programmer errors or misunderstanding about the C language. However, if you think you’ve found a bug, email the source and assembly listing to the TA explaining where the bug is.

3) Note: because the plot will scale the (x,y) data onto the pixel coordinates of the LCD, the actual units of X, Y need not be 0.001 as long as all the x-resolutions are the same and all the y-resolutions are the same. In particular, in the following equations, the units of all the x terms cancel and the units of all the y terms cancel. In this code, the plotting area is a square on the bottom (0,32) to (127,159)

// i goes from 0 to 127

// x=MaxX maps to i=0

// x=MaxX maps to i=127

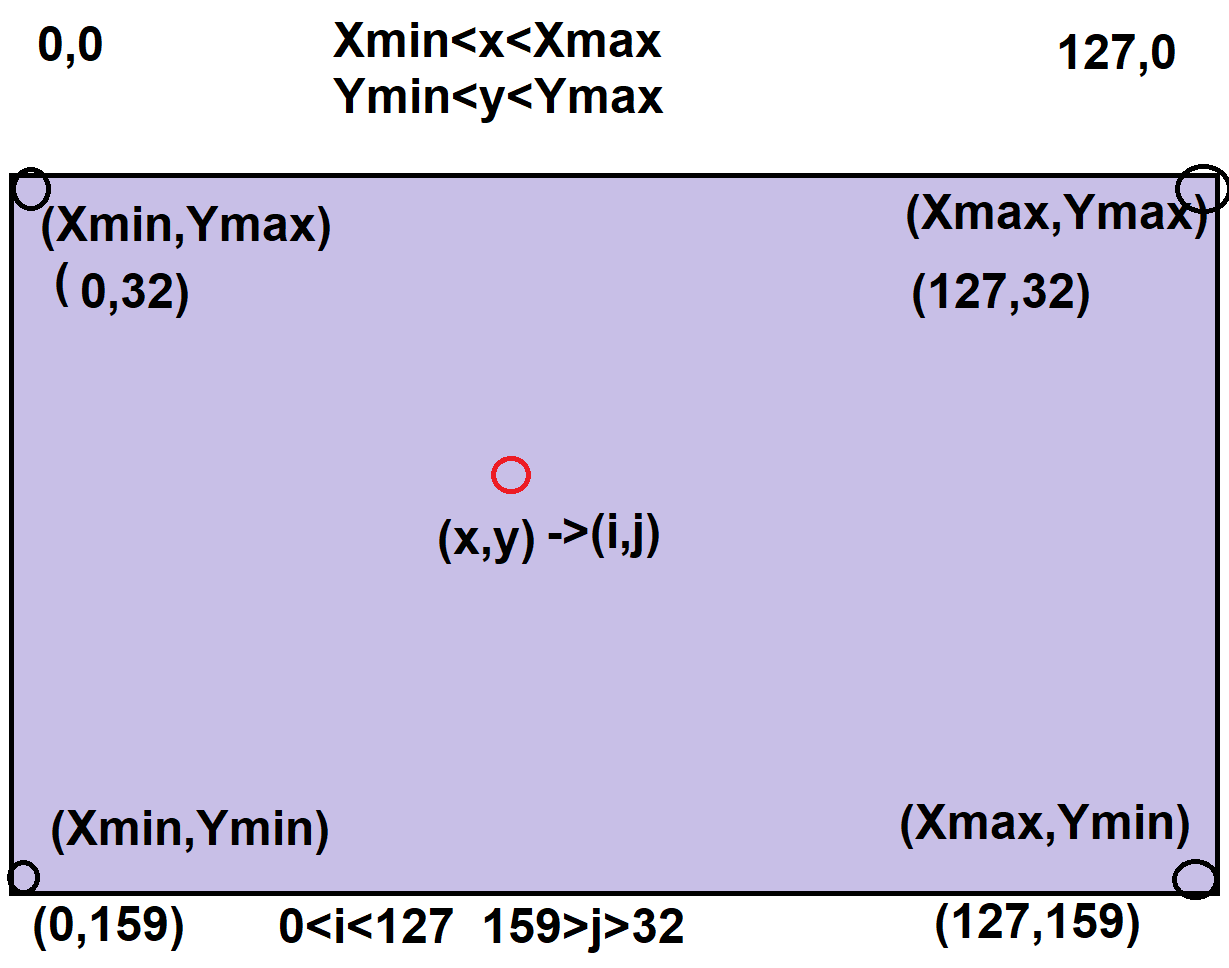
i = (127\*(x-MinX))/(MaxX-MinX);

// y=MaxY maps to j=32

// y=MinY maps to j=159

j = 32+(127\*(MaxY-y))/(MaxY-MinY);

ST7735\_DrawPixel(i, j, color);



4) If you want a bigger dot, plot four pixels at each point

ST7735\_DrawPixel(i, j, color);

ST7735\_DrawPixel(i+1, j, color);

ST7735\_DrawPixel(i, j+1, color);

ST7735\_DrawPixel(i+1, j+1, color);

5) To simulate, put LaunchPadDLL.dll in \Keil\ARM\BIN, and set the debug options (as shown on the next page)

A screenshot of a social media post

Description automatically generated