# Lab 1. Fixed-point Output

This laboratory assignment accompanies the book, Embedded Systems: Real-Time Interfacing to ARM Cortex M Microcontrollers, ISBN-13: 978-1463590154, by Jonathan W. Valvano, copyright © 2015.

**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 "Developing C Programs using Metrowerks" at http://users.ece.utexas.edu/~ryerraballi/CPrimer/
- Valvano Chapters 1, 2 and 3 from the book Embedded Systems: Real Time Interfacing,
- **Lab1.c**, **fixed.h**, **style.pdf**, **style_policy.pdf** and **c_and_h_files.pdf** guide

**Starter files**
- **ST7735_4C123** project

**Required Hardware**

| | |
|---|---|
| EK-TM4C123GXL, www.ti.com | $12.99 |
| 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 fixed-point and 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 create a **fixed.h** header file that contains the prototypes for public functions explaining how to use the functions. You will also create a **fixed.c** implementation file containing the actual C code for public and private functions. Private data structures should be included in this file. An important factor in modular design is to separate the policies of the interface (how to use the software is defined in the **fixed.h** file) from the mechanisms (how the programs are implemented, which is defined in the **fixed.c** file.) You should layer your driver on top of the existing ST7735 driver as shown in Figure 1.1. You are allowed to configure the driver to support **`printf`** and **`sprintf`**. However, if you use **`printf`**/**`sprintf`** the system will run slower and require more memory to operate. In the main program you will develop software to test your functions. The **Lab1.c** shows an example test program. You are allowed to modify Lab1.c to conform with the syntax of your functions. The driver files will be used in subsequent labs, whereas software in the main program will only be used in this lab to verify the software is operational. There are two long term usages of the main program. Sometimes we deliver the main program to our customer as an example of how our module can be used, illustrating the depth and complexity of our module. Secondly, the main program provides legal proof that we actually tested the software. A judge can subpoena files relating to testing to determine liability in a case where someone is hurt using a product containing our software.

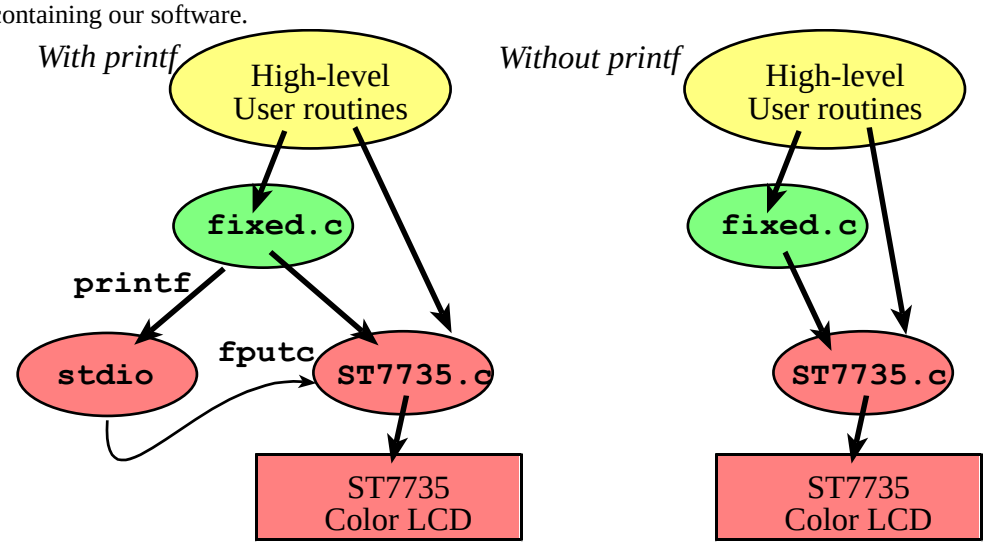


*Figure 1.1. If you use printf for character output, the call graph for the system includes stdio and there is a call to* ***fputc*** *from stdio back to your software. Without printf, the call graph.*

In order to use the power and flexibility of the **printf** function, we have implemented an output function called **fputc** inside the low-level ST7735 driver. In this lab, you can ignore the ***f** parameter. However, in future systems you could use it to allow the user to redirect output using **fprintf**. Implementing this function will direct all standard output to the ST7735 display. The driver will handle at special characters like **CR LF** and **TAB**. You can then add **#include <stdio.h>** and use **printf** to output to the display.

```
int fputc(int ch, FILE *f){ // Print a character to ST7735 LCD
  ST7735_OutChar(ch);
  return 1;
}
```

**Specifications**

You will develop and test these four functions

| | |
|---|---|
| **ST7735_sDecOut3** | Signed 32-bit decimal fixed-point Δ = 0.001 |
| **ST7735_uBinOut8** | Unsigned 32-bit binary fixed-point Δ = 1/256 |
| **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 -9.999 to +9.999. Any integer part outside the range of -9999 to +9999 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 |
|---|---|
| -100000 | *.*** |
| -10000 | *.*** |
| -9999 | -9.999 |
| -999 | -0.999 |
| -1 | -0.001 |
| 0 | 0.000 |
| 123 | 0.123 |
| 1234 | 1.234 |
| 9999 | 9.999 |
| 10000 | *.*** |

*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 1/256. The full-scale range is from 0 to 999.99. If the integer part is larger than 256000, it signifies an error. The **ST7735_uBinOut8** 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_uBinOut8** are illustrated in Table 1.2.

| Parameter | Display |
|---|---|
| 0 | 0.00 |
| 2 | 0.01 |
| 64 | 0.25 |
| 100 | 0.39 |
| 500 | 1.95 |
| 512 | 2.00 |
| 5000 | 19.53 |
| 30000 | 117.19 |
| 255997 | 999.99 |
| 256000 | ***.** |

*Table 1.2. Specification for the* **ST7735_uBinOut8** *function.*

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. Because of rounding your solution may be different by ±0.01 from the examples in Table 1.3.

The third and fourth functions you will implement will draw an X-Y scatter plot on the LCD. Before designing your X-Y plot software, run the **`main1`** software as part of the **ST7735_4C123** starter project. It creates a real-time (x,t) plot where the data are added to the LCD one by one in real time. 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 signed decimal fixed-point format with resolution of 0.001. The **`ST7735_XYplotInit`** function will configure the plot and clear the drawing area. Parameters:

| | |
|---|---|
| Title | ASCII string to label the plot |
| minX | smallest X data value allowed, Δ= 0.001 |
| maxX | largest X data value allowed, Δ= 0.001 |
| minY | smallest Y data value allowed, Δ= 0.001 |
| maxY | largest Y data value allowed, Δ= 0.001 |

The **`ST7735_XYplot`** function plots the X-Y scatter data. Parameters:

| | |
|---|---|
| num | number of points in the X, Y arrays |
| bufX | array of X data, Δ= 0.001 |
| bufY | array of Y data, Δ= 0.001 |

Feel free to adjust the specifications as will as long as it creates an X-Y scatter plot.

**Preparation (do this before your lab period) (do preparation by yourself if you do not have a partner yet)**
*Preparation is due before the lab period starts and must be uploaded to Canvas.* 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_uBinOut8`**
- **`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. You will find the prototypes for these four functions in the **fixed.h** file. You are allowed to modify the syntax of these functions as long as there is one decimal fixed-point output, one binary fixed-point output, and one pixel graphics plotting function.

If you do not have the board and the ST7735 LCD, you can perform this lab in simulation. Late preparations will not be accepted. 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 **fixed.c** to the Lab 1 project. "Syntax-error-free" hardcopy listings of your main, **fixed.h**, and **fixed.c** are 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 **fixed.h** are intended for the client (programmers that will use your functions.) The comments included in **fixed.c** are intended for the coworkers (programmers that will debug/modify your functions.) Your main program will be used to test the **fixed.c** functions. It is important for you to learn how to use the interactive features of the uVision4 debugger.

**Procedure (do this during your lab period)**

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 Lab 1.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`** and

**`fixed.h`**), your customer can use this file to understand how to use your package, its range of functions, and its limitations.

**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.

**2.** Experiment with the different features of uVision4 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 (**`fixed.h fixed.c main.c`**). 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.

**Deliverables (exact components of the lab report)**

A) Objectives (1/2 page maximum)
B) Hardware Design (none for this lab)
C) Software Design (upload **`fixed.h fixed.c main.c`** files to Canvas as instructed by your TA)
D) Measurement Data (none for this lab)
E) Analysis and Discussion (1 page maximum). In particular, answer these questions

1) In what way is it good design to minimize the number of arrows in the call graph for your system?

2) Why is it important for the decimal point to be in the exact same physical position independent of the number being displayed? Think about how this routine could be used with the **`ST7735_SetCursor`** command.

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

4) When should you use binary fixed-point over decimal fixed-point? When should you use decimal fixed-point over binary fixed-point?

5) 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).

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

**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.

```
// 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) You may find it useful read to the temperature measurement lab to get a feel for the context of the fixed-point routines you are developing. In particular, you should be able to use your Lab 1 functions in future labs without additional modification.

4) 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,   ST7735_BLUE);
```

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

```
ST7735_DrawPixel(i,   j,   ST7735_BLUE);
ST7735_DrawPixel(i+1, j,   ST7735_BLUE);
ST7735_DrawPixel(i,   j+1, ST7735_BLUE);
ST7735_DrawPixel(i+1, j+1, ST7735_BLUE);
```

6) To simulate, put LaunchPadDLL.dll in \Keil\ARM\BIN, and set the debug options