EE 445L - Lab 1: Fixed-point Output

Joshua Bryant jmb6357
James Morris jsm3288

September 12, 2014

1 Objective

The main objectives of this lab were as follows: to introduce and ourselves with the provided lab equipment, to familiarize ourselves with Keil μVision4 for the ARM Cortex M processor, to develop a set of useful fixed-point output routines that we can use in future labs, and to re-familiarize ourselves with fixed-point arithmetic and the differences in properties between fixed-point and floating-point.

2 Analysis and Discussion

1. It is good to design fixed.c to not be tied specifically to any low-level display routines to allow the code to remain portable. For instance, instead of calling the hardware specific function **ST7735_OutString()**, calling the generic **printf()** function allows whatever code calling our functions to specify what output is appropriate.

2. Guaranteeing that the decimal point is in the same physical position regardless of the number being printed is important for visual formatting on output screens. A standard format allows for easy reading of the information and for any form of formatted output where one may only want to modify the integer portion of the number and not the mantissa.

3. Fixed point numbers are useful if higher precision is needed in a number as well as less hardware complexity being required to operate on fixed point numbers. Floating point should be used if a larger range is required.

4. Binary fixed-point is preferable for scaling values by powers of 2 since this results in fast bit-shifts in the hardware. Binary fixed-point also has the benefit of being able to exactly represent fractional powers of two. Decimal fixed-point is useful for exactly representing fractional powers of 10 since these can only be approximated by binary fixed-point.

5. One example of an application for fixed-point would be real-time digital signal processing where using a microcontroller that supports floating point arithmetic would be too costly to be commercially viable.

6. Yes, it is possible to use floating-point on the ARM Cortex M4. It would require using an M4F processor and would require more complicated hardware in the chip used. Using floating point, in general, is also slower than using fixed-point arithmetic.

A
fixed.h

```
// filename ******** fixed.h **************
// Design specification for Lab 1
// Jonathan Valvano
// August 28, 2014

/* Do not post, solution to lab for the book
   "Embedded Systems: Real Time Interfacing to the Arm Cortex M",
   ISBN: 978-1463590154, Jonathan Valvano, copyright (c) 2014

 Copyright 2014 by Jonathan W. Valvano, valvano@mail.utexas.edu
 THIS SOFTWARE IS PROVIDED "AS IS".  NO WARRANTIES, WHETHER EXPRESS, IMPLIED
 OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF
 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE.
 VALVANO SHALL NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL,
 OR CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.
 For more information about my classes, my research, and my books, see
 http://users.ece.utexas.edu/~valvano/
 */

/*Altered by James Morris to include the typedefs used in the file.*/
#include <stdint.h>

/****************Fixed_uDecOut2s***************
 converts fixed point number to ASCII string
 format unsigned 32-bit with resolution 0.01
 range 0 to 999.99
 Input: unsigned 32-bit integer part of fixed point number
        greater than 99999 means invalid fixed-point number
 Output: null-terminated string exactly 6 characters plus null
 Examples
 12345 to "123.45"
 22100 to "221.00"
   102 to "  1.02"
    31 to "  0.31"
100000 to "***.**"    */
void Fixed_uDecOut2s(uint32_t n,  char *string);

 /****************Fixed_uDecOut2***************
 outputs the fixed-point value on the display
 format unsigned 32-bit with resolution 0.01
 range 0 to 999.99
 Input: unsigned 32-bit integer part of fixed point number
        greater than 99999 means invalid fixed-point number
 Output: none
 Examples
 12345 to "123.45"
 22100 to "221.00"
   102 to "  1.02"
    31 to "  0.31"
100000 to "***.**"    */
void Fixed_uDecOut2(uint32_t n);
```

```
 /****************Fixed_uDecOut3***************
 outputs the fixed-point value on the display
 format unsigned 32-bit with resolution 0.001
 range 0 to 99.999
 Input: unsigned 32-bit integer part of fixed point number
         greater than 99999 means invalid fixed-point number
 Output: none
 Examples
 12345 to "12.345"
 22100 to "22.100"
   102 to " 0.102"
    31 to " 0.031"
100000 to "**.***"    */
void Fixed_uDecOut3(uint32_t n);

/****************Fixed_sDecOut3s***************
 converts fixed point number to ASCII string
 format signed 32-bit with resolution 0.001
 range -9.999 to +9.999
 Input: signed 32-bit integer part of fixed point number
 Output: null-terminated string exactly 6 characters plus null
 Examples
  2345 to " 2.345"
 -8100 to "-8.100"
  -102 to "-0.102"
    31 to " 0.031"

 */
void Fixed_sDecOut3s(int32_t n, char *string);

/****************Fixed_sDecOut3s***************
 converts fixed point number to the display
 format signed 32-bit with resolution 0.001
 range -9.999 to +9.999
 Input: signed 32-bit integer part of fixed point number
 Output: none
 Output to display has exactly 6 characters
 Examples
  2345 to " 2.345"
 -8100 to "-8.100"
  -102 to "-0.102"
    31 to " 0.031"
 */
void Fixed_sDecOut3(int32_t n);



/**************Fixed_uBinOut8s***************
 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 Fixed_uBinOut8 function takes an unsigned 32-bit integer part
 of the binary fixed-point number and outputs the fixed-point value on the OLED.
```

```
 Input: unsigned 32-bit integer part of fixed point number
 Output: null-terminated string
Parameter output string
     0     "  0.00"
     2     "  0.01"
    64     "  0.25"
   100     "  0.39"
   50      "  1.95"
   512     "  2.00"
  5000     " 19.53"
 30000     "117.19"
255997     "999.99"
256000     "***.**"
*/
void Fixed_uBinOut8s(uint32_t n,  char *string);

/**************Fixed_uBinOut8***************
 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 Fixed_uBinOut8 function takes an unsigned 32-bit integer part
 of the binary fixed-point number and outputs the fixed-point value on the OLED.
 Input: unsigned 32-bit integer part of fixed point number
 Output: none
Parameter LCD 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 ***.**
*/
void Fixed_uBinOut8(uint32_t n);
```

B fixed.c

```
#include <stdio.h>
#include <math.h>
#include <stdint.h>
#include <stdlib.h>
#include <string.h>
#include "ST7735.h"
#include "fixed.h"

void Fixed_uDecOut2s(uint32_t n,  char *string){
int32_t i,j;
if(((j=(int)log10((double)n))<3)){
j = 2;
}
j += 1;
for(i = 5; i >= 5-j; i--){
if(i == 3){
string[i] = '.';
}
else{
string[i] = (n%10) + 0x30;
n = n/10;
}
}
}

void Fixed_uDecOut2(uint32_t n){
uint8_t *str;
str = (uint8_t*)malloc(sizeof(uint8_t)*7);
str = (uint8_t*)strcpy((char*)str, "      ");
if(n > 99999){
str = (uint8_t*)strcpy((char*)str, "***.**");
}
else{
Fixed_uDecOut2s(n, (char*)str);
}
printf((char*)str);
free(str);
}

void Fixed_uDecOut3(uint32_t n){
uint8_t *str;
int32_t i,j;
str = (uint8_t*)malloc(sizeof(uint8_t)*7);
str = (uint8_t*)strcpy((char*)str, "      ");
if(n > 99999){
str = (uint8_t*)strcpy((char*)str, "**.***");
}
else{
if((j=(int)log10((double)n))<4){
  j = 3;
  }
j++;
```

```
for(i = 5; i >= 5-j; i--){
if(i == 2){
str[i] = '.';
}
else{
str[i] = (n%10) + 48;
n = n/10;
}
}
}
printf((char*)str);
free(str);
}

void Fixed_sDecOut3s(int32_t n, char *string){
int32_t i, j;
if((j=(int)log10((double)n))<3){
j=3;
}
j++;
if(n < 0){
string[0] = '-';
}
for(i = 5; i >= 5-j; i--){
if(i == 2){
string[i] = '.';
}
else{
string[i] = (abs(n)%10) + 48;
n = n/10;
}
}
}

void Fixed_sDecOut3(int32_t n){
uint8_t *str;
str = (uint8_t*)malloc(sizeof(uint8_t)*7);
str = (uint8_t*)strcpy((char*)str, "      ");
if((n > 9999)||(n < -9999)){
str = (uint8_t*)strcpy((char*)str, " *.***");
}
else{
Fixed_sDecOut3s(n, (char*)str);
  }
printf((char*)str);
free(str);
}

void Fixed_uBinOut8s(uint32_t n,  char *string){
int32_t i,j;
n = ((100*n)+128)/256;
if((j=(int)log10((double)n))<3){
j=2;
}
```

```
j++;
for(i = 5; i >= 5-j; i--){
if(i == 3){
string[i] = '.';
}
else{
string[i] = (n%10) + 48;
n = n/10;
}
}
}

void Fixed_uBinOut8(uint32_t n){
uint8_t *str;
str = (uint8_t*)malloc(sizeof(uint8_t)*7);
str = (uint8_t*)strcpy((char*)str, "      ");
if(n >= 255998){
str = (uint8_t*)strcpy((char*)str, "***.**");
}
else{
Fixed_uBinOut8s(n, (char*)str);
}
printf((char*)str);
free(str);
}
```