EE445L - Lab 1: Fixed-point Output

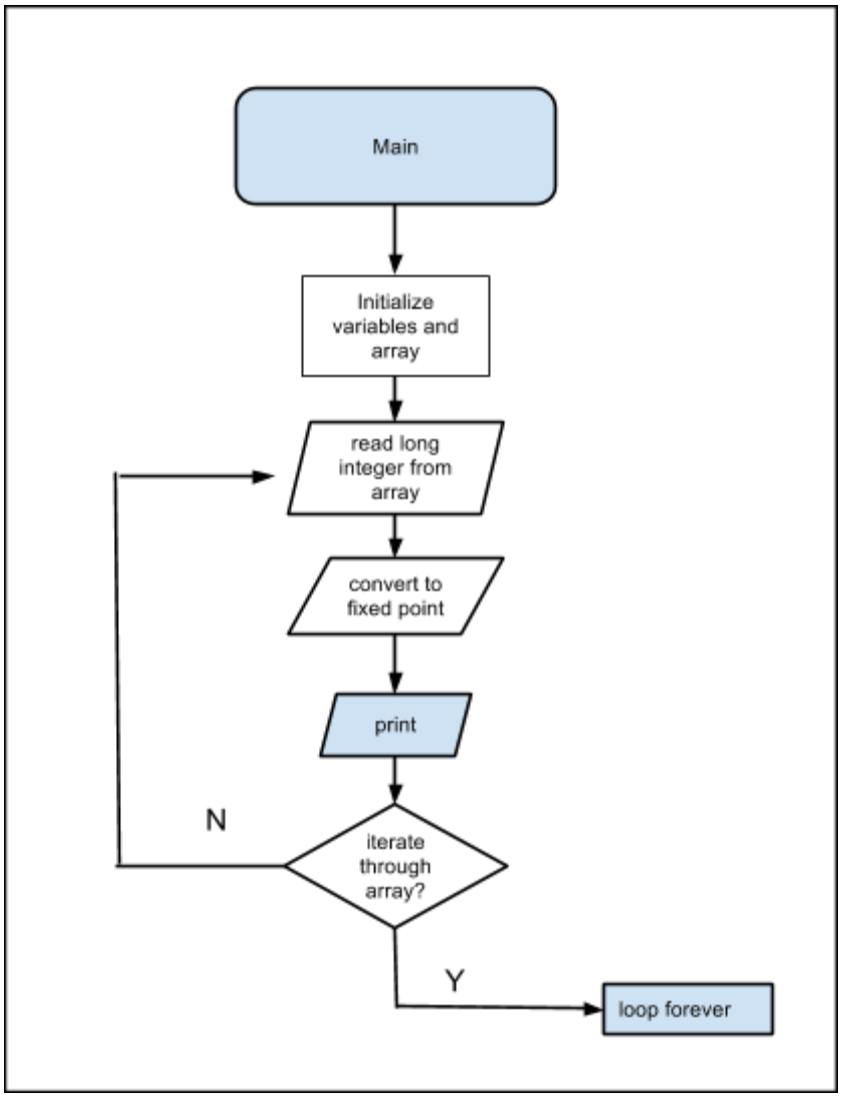
Duc Tran and Brandon Wong

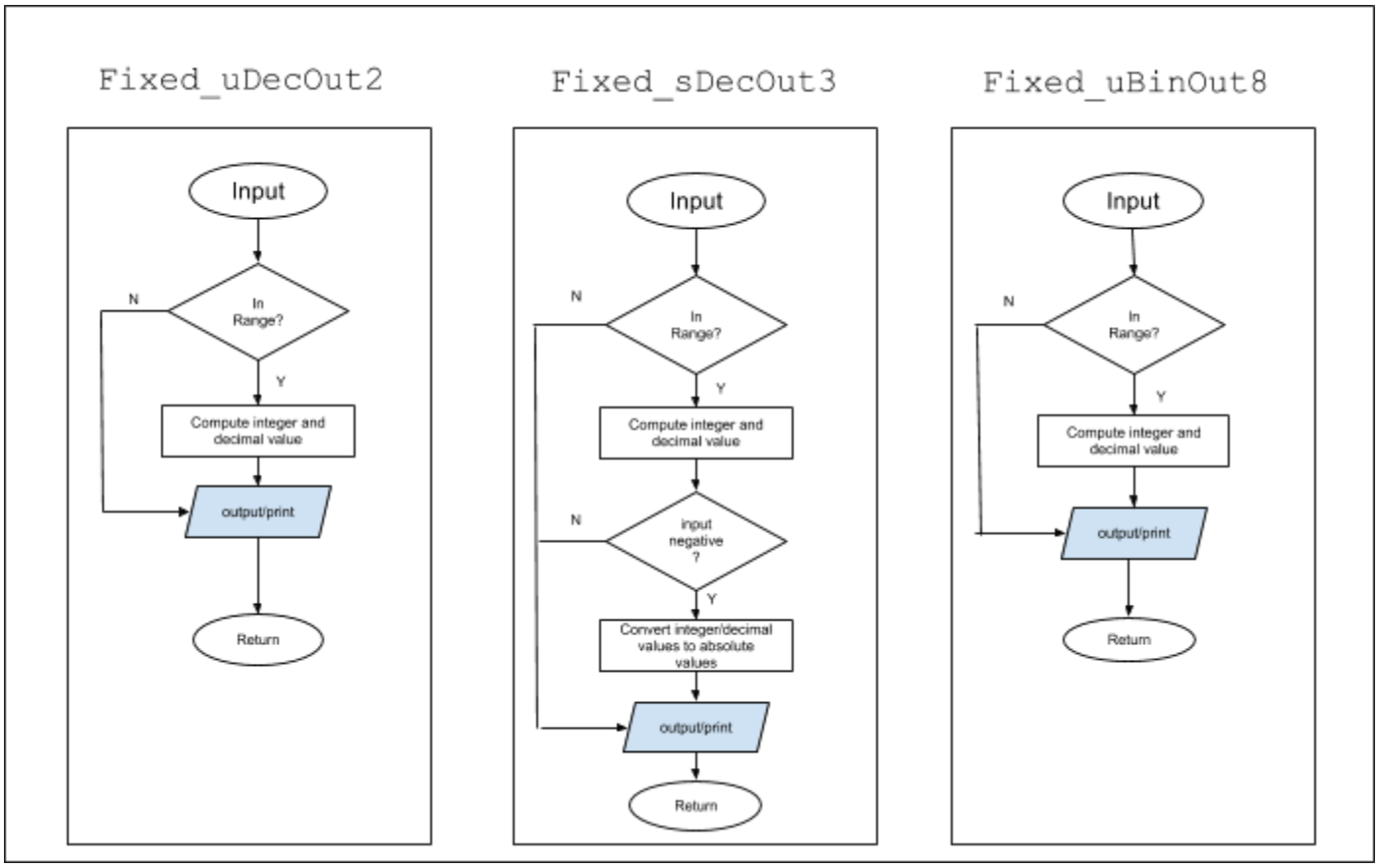
9/13/2013

1.0 OBJECTIVES

Our main objective for this lab is to familiarize ourselves with the LM3S1968 programming environment as well as the Keil uVision4 software. Our task was to complete the fixed-point routine from the provided starter files so that we can use these functions for future labs. After completing the functions, we learned the use of the debugger and mounting the program onto the board.

2.0 SOFTWARE DESIGN





3.0 ANALYSIS AND DISCUSSION

1. We do not know how the OLED display is written. We are only provided with the drivers of the display so we can output our values onto the display using the functions that were provided by the designers. This is a good design because it encapsulates information between our code for fixed points and the writers for the OLED display.

2. One of the constraint for the lab is that the subroutine must output six characters with a fixed amount of digit for decimal and the integer. If we move the decimal point, the resolution changes. For example if the number is “ 1.01”, and if we decide to shift the decimal point to the left then the number become “ 1.010”. Although the two have the same value but “ 1.01” have a resolution of .01 while “ 1.010” have a resolution of .001.

3. Fixed point is practical for numbers that have consistent fraction values and decimal values. If the smallest fraction you need to represent is a value of a hundredth, then we can fix two points for the fraction and the rest for the decimal. Fixed point is also much easier to read. However fixed point

integers are limited in the range of numbers that it can display. When you want to display a wide range of numbers, floating point will be better.

4. To represent fractions, decimal fixed point provides a much wider resolution than binary fixed point. This means that binary fixed point have difficulties in representing certain fractions. However, binary fixed point numbers are much quicker in evaluation because it's ability to logically shift left or right rather than division or multiplication for decimal fixed point.

5. Currency would be an application where you can apply fixed point because two points would represent the change, and the rest of the points would represent the dollar amount.

6. We can use floating point on Arm Cortex M3 but because it doesn't have any hardware for floating point application, speed of calculation will be slow. The Pentium w/MMX one of the processors known for adding single precision floating point. So floating point arithmetic operations would be faster.

4.0 CODE

```
//Filename: Fixed.c
//Author: Duc Tran, Brandon Wong
//Initial Creation Date: September 4, 2013
//Description:
//Lab Number: W 2-3:30
//TA : Omar & Mahesh
//Date of last revision :September 9, 2013
//Hardware Configuration : NONE
#include <stdio.h>
#include "fixed.h"


/****************Fixed_uDecOut2s***************
 Description: 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(unsigned long n, char *string){
          unsigned int integer = 0;
          unsigned int decimal = 0;

          if (n >  99999){
    sprintf(string,"***.**");
            return;
  }

          integer = n/100;
          decimal = n%100;
          sprintf(string,"%3u.%.2u",integer,decimal);
  return;
}

 /****************Fixed_uDecOut2***************
 outputs the fixed-point value on the OLED
 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(unsigned long n){
          unsigned int integer = 0;
```

```
        unsigned int decimal = 0;

        if (n >  99999){
   printf("***.**");
          return;
  }

        integer = n/100;
        decimal = n%100;
        printf("%3u.%.2u",integer,decimal);
 return;
}

/****************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(long n, char *string){
        int integer = 0;
        int decimal = 0;

        //sprinf(answer,%d,n)
        if(n > 9999 || n <-9999){
                sprintf(string,"**.***");
                return;
        }
        integer = n/1000;
        decimal = n%1000;
```

```
        if(n<0){
                decimal = decimal * (-1);
                integer = integer * (-1);
                sprintf(string,"-%i.%.3i",integer,decimal);
        }
        else{
                sprintf(string,"%2i.%.3i",integer,decimal);
        }
        return;
}
/****************Fixed_sDecOut3***************
 converts fixed point number to OLED
 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
 OLED has exactly 6 characters
 Examples
  2345 to " 2.345"
 -8100 to "-8.100"
  -102 to "-0.102"
    31 to " 0.031"
 */
void Fixed_sDecOut3(long n){
  int integer = 0;
        int decimal = 0;

        //sprinf(answer,%d,n)
        if(n > 9999 || n < -9999){
                printf("**.***");
                return;
        }
        integer = n/1000;
        decimal = n%1000;
        if(n<0){
                decimal = decimal * (-1);
                integer = integer * (-1);
```

```
                printf("-%i.%.3i",integer,decimal);
        }
        else{
                printf("%2i.%.3i",integer,decimal);
        }
        return;
}
/**************Fixed_uBinOut8s***************
 Description:  Convert unsigned 32-bit integer to a fix point value, and output to char string
 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(unsigned long n,  char *string){
  //fixed point number = I * res
        unsigned long decimal = 0;
        unsigned long integer = 0;
        if(n >= 256000){
                sprintf(string,"***.**");
                return;
        }
        decimal = (((n%256)*100/256));
        integer = n/256;
        sprintf(string,"%3lu.%.2lu",integer,decimal);
        return;
}
```

```
/**************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 OLED 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(unsigned long n){
        //fixed point number = I * res
        unsigned long decimal = 0;
        unsigned long integer = 0;
        if(n >= 256000){
                printf("***.**");
                return;
        }
        decimal = ((n%256)*100/256);
        integer = n/256;
        printf("%3lu.%.2lu",integer,decimal);
        return;
}

//////////////////////////////////////////

//Filename: Lab1.c
```

```
//Author: Duc Tran, Brandon Wong
//Initial Creation Date: September 4, 2013
//Description:
//Lab Number: W 2-3:30
//TA : Omar & Mahesh
//Date of last revision :September 12, 2013
//Hardware Configuration : NONE
#include <stdio.h>
#include "fixed.h"
#include "Output.h"

// const will place these structures in ROM
const struct outTestCase{       // used to test routines
  unsigned long InNumber;       // test input number
  char OutBuffer[10];           // Output String
};
const struct outTestCase2{        // used to test routines
  long InNumber;                // test input number
  char OutBuffer[10];           // Output String
};
typedef const struct outTestCase outTestCaseType;
typedef const struct outTestCase2 outTestCaseType2;
outTestCaseType outTests8[16]={
{     0,  "  0.00" }, //      0/256 = 0.00
{     4,  "  0.01" }, //      4/256 = 0.01
{    10,  "  0.03" }, //     10/256 = 0.03
{   200,  "  0.78" }, //    200/256 = 0.78
{   254,  "  0.99" }, //    254/256 = 0.99
{   505,  "  1.97" }, //    505/256 = 1.97
{  1070,  "  4.17" }, //   1070/256 = 4.17
{  5120,  " 20.00" }, //   5120/256 = 20.00
{ 12184,  " 47.59" }, //  12184/256 = 47.59
{ 26000,  "101.56" }, //  26000/256 = 101.56
{ 32767,  "127.99" }, //  32767/256 = 127.99
{ 32768,  "128.00" }, //  32768/256 = 128
{ 34567,  "135.02" }, //  34567/256 = 135.02
{123456,  "482.25" }, // 123456/256 = 482.25
```

```
{255998,  "999.99" }, // 255998/256 = 999.99
{256000,  "***.**" }  // error
};

outTestCaseType outTests2[16]={
{      0, "  0.00" }, //      0/100 = 0.00
{      4, "  0.04" }, //      4/100 = 0.04
{     10, "  0.10" }, //     10/100 = 0.10
{    200,  "  2.00" }, //    200/100 = 2.00
{    254,  "  2.54" }, //    254/100 = 2.54
{    505,  "  5.05" }, //    505/100 = 5.05
{   1070,  " 10.70" }, //   1070/100 = 10.70
{   5120,  " 51.20" }, //   5120/100 = 51.20
{  12184,  "121.84" }, //  12184/100 = 121.84
{  26000,  "260.00" }, //  26000/100 = 260.00
{  32767,  "327.67" }, //  32767/100 = 327.67
{  32768,  "327.68" }, //  32768/100 = 327.68
{  34567,  "345.67" }, //  34567/100 = 345.67
{  9999,  " 99.99" }, //    9999/100 = 99.99
{  99999,  "999.99" }, //  99999/100 = 999.99
{100000,  "***.**" }  // error
};

outTestCaseType2 outTests3[16]={
{      0, " 0.000" }, //      0/1000 = 0.0
{     -4, "-0.004" }, //      4/1000 = 0.004
{    10,  " 0.010" }, //     10/1000 = 0.010
{    200,  " 0.200" }, //    200/1000 = 0.200
{    254,  " 0.254" }, //    254/1000 = 0.254
{    505,  " 0.505" }, //    505/1000 = 0.505
{  -1070,  "-1.070" }, //   1070/1000 = 1.070
{   5120,  " 5.120" }, //   5120/1000 = 5.120
{ -12184,  "**.***" }, //  -12184/1000 = error
{      -4, "-0.004" }, //  -4/1000 = -0.004
{ -32767,  "**.***" }, //  32767/1000 = error
{  32768,  "**.***" }, //  32768/1000 = error
{  34567,  "**.***" }, //  34567/1000 = error
```

```
{ 9999,  " 9.999" }, // 9999/1000 = 9.999
{-9999,  "-9.999" }, // -9999/1000 = -9.999
{-7888,  "-7.888" }  // -7888/1000 = -7.888
};

unsigned int Errors,AnError;
char Buffer[10];
void main(void){ // possible main program that tests your functions
  unsigned int i;
  unsigned int inc;
  Output_Init();
  Errors = 0;
  printf("Begin Fixed_uBinOut8\r\n");
  for(i=0; i<16; i++){
    Fixed_uBinOut8(outTests8[i].InNumber);
    printf("\r\n");
    for(inc =0;inc < 1000000;inc++) {
        //do nothing
    }
  }
  printf("Finished Fixed_uBinOut8\r\n");
  printf("Begin Fixed_uDecOut2\r\n");
  for(i=0; i<16; i++){
    Fixed_uDecOut2(outTests2[i].InNumber);
    printf("\r\n");
    for(inc =0;inc < 1000000;inc++) {
        //do nothing
    }
  }
  printf("Finished Fixed_uDecOut2\r\n");
  printf("Begin Fixed_sDecOut3\r\n");
  for(i=0; i<16; i++){
    Fixed_sDecOut3(outTests3[i].InNumber);
    printf("\r\n");
    for(inc =0;inc < 1000000;inc++) {
        //do nothing
    }
```

```
  }
  printf("Finished Fixed_sDecOut3\r\n");
  for(;;) {} /* wait forever*/
}

    printf("\r\n");
    for(inc =0;inc < 1000000;inc++) {
         //do nothing
    }
  }
  printf("Finished Fixed_sDecOut3\r\n");
  for(;;) {} /* wait forever*/
}
```