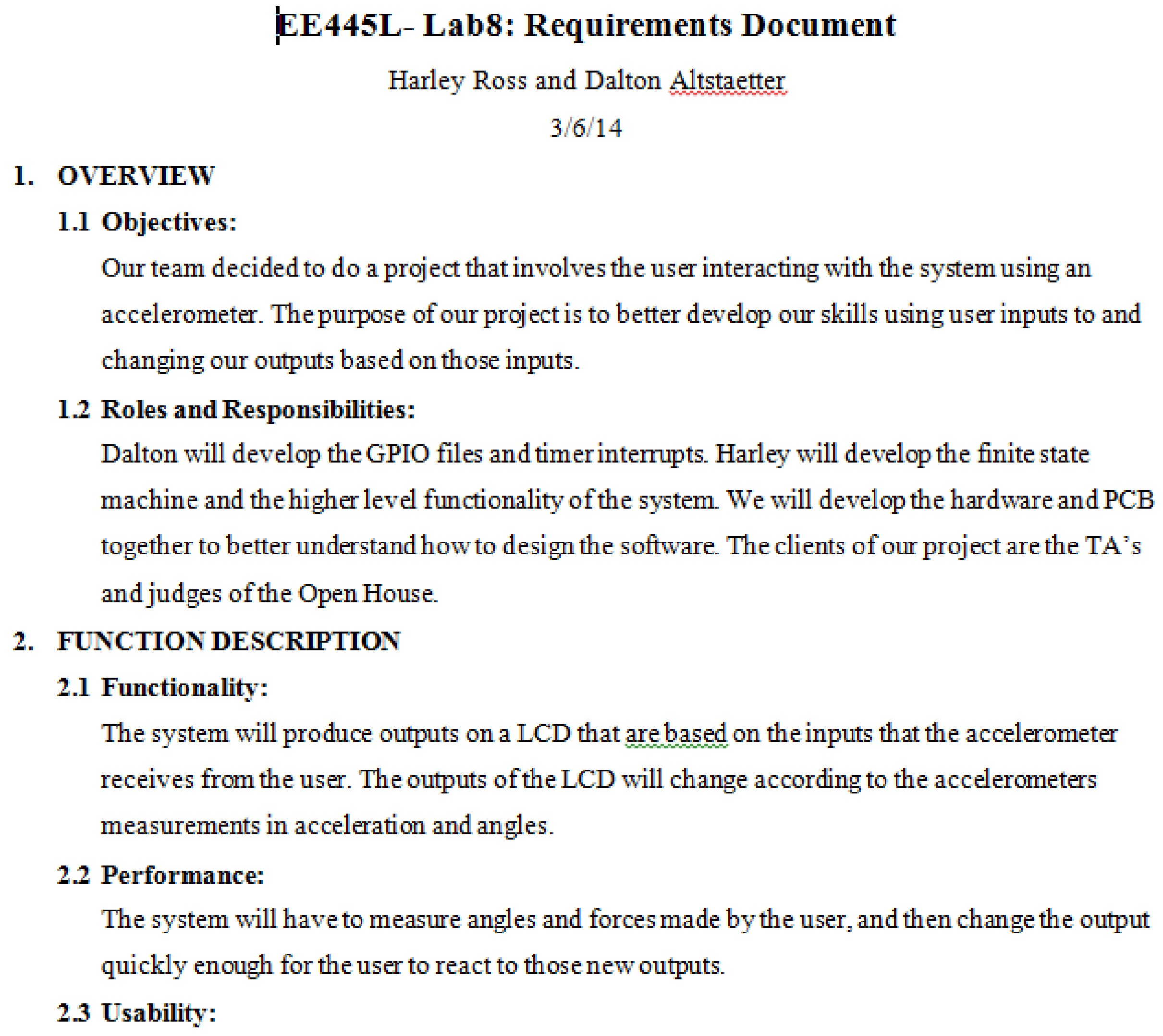
EE445L – Lab8: Software Drivers for an Embedded System

Harley Ross and Dalton Altstaetter

3/31/14

OBJECTIVES



EE445L- Lab8: Requirements Document

Harley Ross and Dalton Altstaetter

3/6/14

1. **OVERVIEW**

1.1 Objectives:

Our team decided to do a project that involves the user interacting with the system using an accelerometer. The purpose of our project is to better develop our skills using user inputs to and changing our outputs based on those inputs.

1.2 Roles and Responsibilities:

Dalton will develop the GPIO files and timer interrupts. Harley will develop the finite state machine and the higher level functionality of the system. We will develop the hardware and PCB together to better understand how to design the software. The clients of our project are the TA's and judges of the Open House.

2. **FUNCTION DESCRIPTION**

2.1 Functionality:

The system will produce outputs on a LCD that are based on the inputs that the accelerometer receives from the user. The outputs of the LCD will change according to the accelerometers measurements in acceleration and angles.

2.2 Performance:

The system will have to measure angles and forces made by the user, and then change the output quickly enough for the user to react to those new outputs.

2.3 Usability:

The output interface will be a LCD screen. The inputs will be the measurements read from the accelerometer. The inputs and outputs for our design will vary based on the cost of each.

3. DELIVERABLES

3.1 Reports:

The reports we will write are the Schematic, PCB Layout, Measurement Data, Testing Procedures, Testing Data, and Analysis and Discussion.

3.2 Outcomes:

3.2.1 Objectives

1-page requirements document

3.2.2 Hardware Design

Regular circuit diagram (SCH file)

PCB layout and three printouts (top, bottom and combined)

3.2.3 Software Design

Include the requirements document (Preparation a)

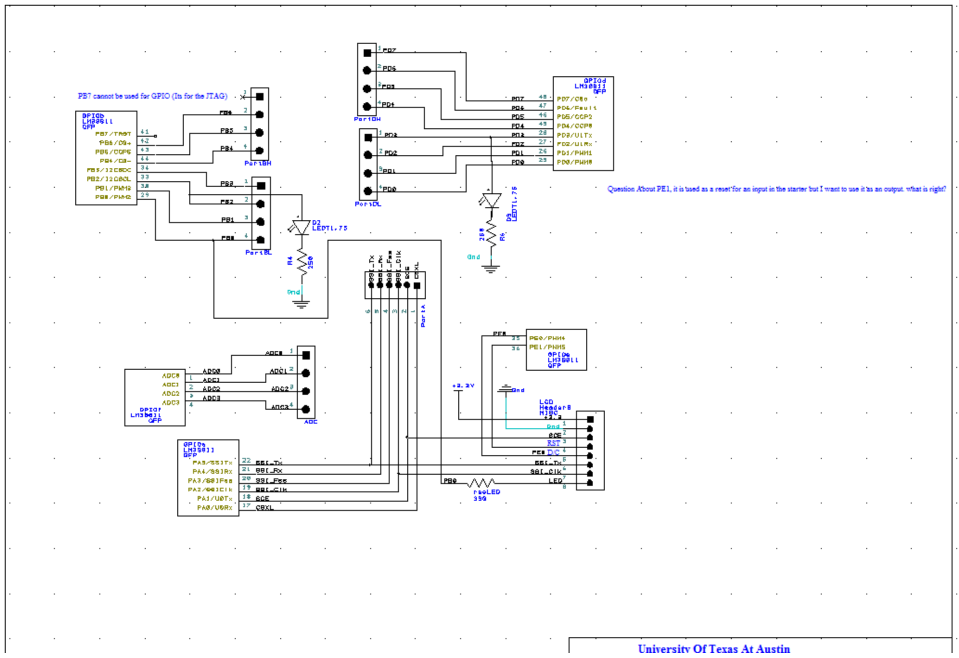
3.2.4 Measurement Data

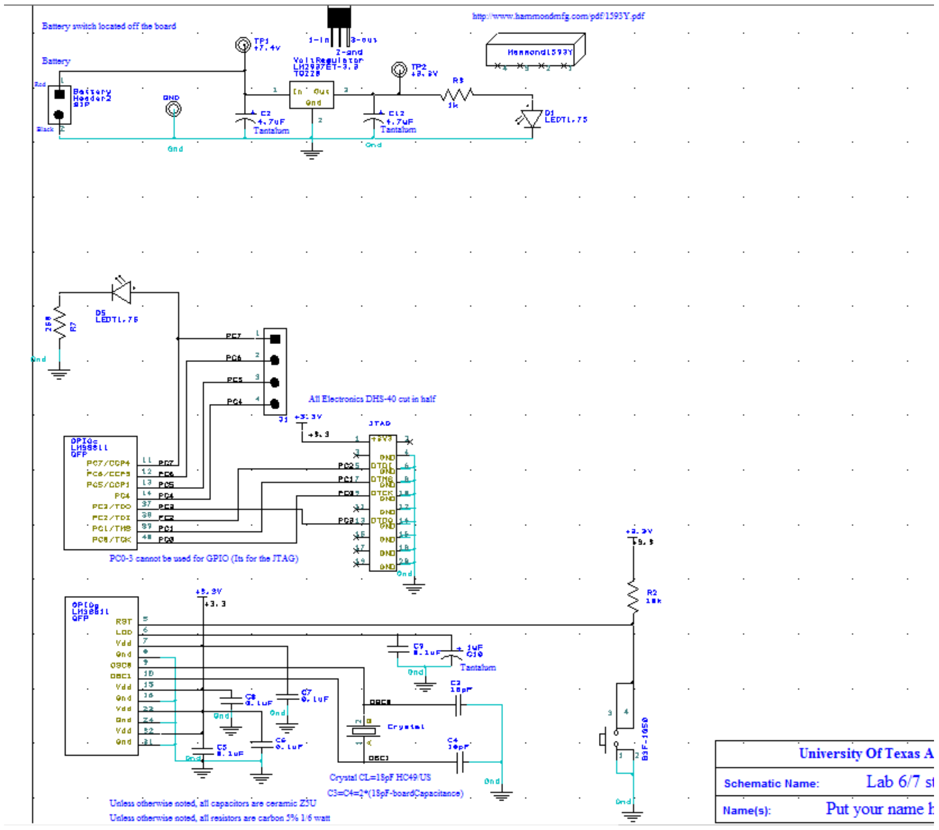
Give the estimated current (Procedure d)

Give the estimated cost (Procedure e)

3.2.5 Analysis and Discussion (none)

HARWARE DESIGN
SCH




Battery switch located off the board
http://www.hammondmfg.com/pdf/1593Y.pdf
Battery
Red
Black
Battery Header2 SIP
GND
TP1 +7.4V
1-In
2-Gnd
3-Out
Voltage Regulator LM2937ET-3.3 TO220
TP2 +3.3V
Hammond1593Y
C2 4.7uF Tantalum
C12 4.7uF Tantalum
R3 1k
D1 LEDT1.75
Gnd
D5 LEDT1.75
R7 250
All Electronics DHS-40 cut in half
JTAG
GPIOc LM3S811 QFP
PC0-3 cannot be used for GPIO (Its for the JTAG)
+3.3V
R2 10k
GPIOg LM3S811 QFP
Crystal
Crystal CL=18pF HC49/US
C3=C4=2*(18pF-board)Capacitance)
Unless otherwise noted, all capacitors are ceramic Z5U
Unless otherwise noted, all resistors are carbon 5% 1/6 watt
B3F-1050
University Of Texas A
Schematic Name:
Lab 6/7 st
Name(s):
Put your name h

SOFTWARE DESIGN

Accelerometer

```
// LSM9DSO.c
#include "LSM9DSO.h"
#include <stdio.h>
#include "inc/hw_ssi.h"
#include "inc/hw_memmap.h"
#include "inc/hw_sysctl.h"
#include "inc/hw_types.h"
#include "driverlib/debug.h"
#include "driverlib/gpio.h"
#include "driverlib/ssi.h"
#include "driverlib/sysctl.h"
#include "DAC.h"
#include "lm3s1968.h"
#include "fixed.h"
#include <stdlib.h>

#define SIZE                        6
#define HIGH                        1
#define LOW                         0
#define NEWLINE                     0xD
#define TAB                                0x9

void Delay(unsigned long ulCount);
void Fixed_sDecOut22s(unsigned long n);
void GyroStuff(unsigned long* ulDataTx, unsigned long* ulDataRx);
void AccelStuff(unsigned long* ulDataTx, unsigned long* ulDataRx);
void SPIread(enum sensor type, unsigned long* ulDataTx, unsigned long* ulDataRx);

extern unsigned long DataX[100];
extern unsigned long DataY[100];
extern unsigned long DataZ[100];
extern int angleX;

static enum gyro_scale gScale;
static enum accel_scale aScale;
//static enum mag_scale mScale;


/* gRes, aRes, and mRes store the current resolution for each sensor.
// Units of these values would be DPS (or g's or Gs's) per ADC tick.
// This value is calculated as (sensor scale) / (2^15).
// it is scaled by 1,000,000 bc of integer division
*/
unsigned long gRes, aRes, mRes;

int currentAngleX = 0;
int currentAngleY = 0;
int currentAngleZ = 0;

long XLangleX = 0;
long XLangleY = 0;
long XLangleZ = 0;

// create a low pass filter to reduce the jitter
```

```c
long filtGx[6] = {0};
long filtGy[6] = {0};
long filtGz[6] = {0};

long filtAx[6] = {0};
long filtAy[6] = {0};
long filtAz[6] = {0};

/* We'll store the gyro, accel, and magnetometer readings in a series of
// public class variables. Each sensor gets three variables -- one for each
// axis. Call readGyro(void), readAccel(void), and readMag(void) first, before using
// these variables!
// These values are the RAW signed 16-bit readings from the sensors.
*/
short gx, gy, gz; // x, y, and z axis readings of the gyroscope
short ax, ay, az; // x, y, and z axis readings of the accelerometer
short mx, my, mz; // x, y, and z axis readings of the magnetometer

// xmAddress and gAddress store the SPI chip select pin
// for each sensor.
unsigned char xmAddress, gAddress;

extern void Delay(unsigned long ulCount);

// need it in radians
static long atan2(short y, short x)
{
       static long angle123=0;
       long lx;
       long ly;
       long swap = 0;;
       if(y > x)
       {
              swap = x;
              x = y;
              y = swap;
       }

       lx = (long)x;
       ly = (long)y;

       ly = (ly+50)/100;
       lx = (lx+50)/100;

       angle123 = (3*lx*lx-ly*ly+(3*lx*lx/2))/(3*lx*lx)*ly;
       angle123 += ((((((ly*ly+2)/5)*ly+lx/2)/lx)*ly+lx/2)/lx)*ly+lx/2)/(lx*lx*lx);
       if(angle123 > 1000)
       {      return 90000; }
       angle123 *= 572;
       if(swap)
       {
              angle123 = 90000-angle123;
       }


/*     if(y==0)
//     {
//            if(x>=0)
//              {
//                      return 0;
//              }
//              return 180;
//      }
//      else if(x == 0)
//      {
//              if(y>0)
//              {
//                      return 90;
//              }
//              return -90;
//      }
//      else if(x>0 && y>0)
//      {
//              angle123 = 3667*lx*ly;
//              angle123 /= (64*lx*lx+17*ly*ly);
//              //angle = (3667*lx*ly)/(64*lx*lx+17*ly*ly);
//              return angle123;
//      }
//      else if(x>0 && y<0)
//      {
//              angle123 = 3667*lx*ly;
//              angle123 /= (64*lx*lx+17*ly*ly);
////            angle = (3667*lx*ly)/(64*lx*lx+17*ly*ly);
//              angle123 += 360;
//              return angle123;
//      }
//      else if(x<0 && y>0)
//      {
//              angle123 = 3667*abs(lx)*ly;
//              angle123 /= (64*lx*lx+17*ly*ly);
////            angle = (3667*abs(lx)*ly)/(64*abs(lx)*abs(lx)+17*ly*ly);
//              angle123 = 180 - angle123;
//              return angle123;
//      }
//      else
//      {
//              lx = abs(lx);
//              ly = abs(ly);
//              angle123 = (3667*lx*ly)/(64*lx*lx+17*ly*ly);
//              angle123 += 180;
//              return angle123;
//      }
*/
return angle123;
}

unsigned short begin(enum gyro_scale gScl, enum accel_scale aScl, enum mag_scale mScl,
enum gyro_odr gODR, enum accel_odr aODR, enum mag_odr mODR)
{
        //unsigned char gTest;
        //unsigned char xmTest;
        //unsigned long* testPtr;
        /* Store the given scales in class variables. These scale variables
        // are used throughout to calculate the actual g's, DPS,and Gs's.
        //mScale = mScl;
        */
```

```
	gScale = gScl;
	aScale = aScl;


	/* Once we have the scale values, we can calculate the resolution
	// of each sensor. That's what these functions are for. One for each sensor
	//calcmRes(); // Calculate Gs / ADC tick, stored in mRes variable
	*/
	calcgRes(); // Calculate DPS / ADC tick, stored in gRes variable
	calcaRes(); // Calculate g / ADC tick, stored in aRes variable

	// Now, initialize our hardware interface.
//initSPI();  // Initialize SPI

	/* To verify communication, we can read from the WHO_AM_I register of
	// each device. Store those in a variable so we can return them.
  gTest = gReadByte(WHO_AM_I_G);   // Read the gyro WHO_AM_I
  xmTest = xmReadByte(WHO_AM_I_XM);	// Read the accel/mag WHO_AM_I
	*/

	// Gyro initialization stuff:
initGyro();   // This will "turn on" the gyro. Setting up interrupts, etc.
/*
//  setGyroODR(gODR); // Set the gyro output data rate and bandwidth.
//	setGyroScale(gScale); // Set the gyro range
*/
	// Accelerometer initialization stuff:
initAccel(); // "Turn on" all axes of the accel. Set up interrupts, etc.
/*
	setAccelODR(aODR); // Set the accel data rate.
//	setAccelScale(aScale); // Set the accel range.
*/

	/* Magnetometer initialization stuff:
	//initMag(); // "Turn on" all axes of the mag. Set up interrupts, etc.
	//setMagODR(mODR); // Set the magnetometer output data rate.
	//setMagScale(mScale); // Set the magnetometer's range.
	*/

	// Once everything is initialized, return the WHO_AM_I registers we read:

return 0;// (xmTest << 8) | gTest;
}

// http://www.school-for-champions.com/algebra/square_root_approx.htm
unsigned long sqrt(unsigned long arg)
{
	//sqrt(arg) ~ 0/5*(arg/guess + guess)
	unsigned long error;
  unsigned long appValue;
	unsigned long guess = 12;
	error = 10;
	for(; error > 2; )
	{
		appValue = (arg+guess*guess+guess)/(guess*2);
		error = abs(appValue - guess);
		guess = appValue;
	}
	return appValue;
}

void chipSelectPin(enum sensor type, unsigned char value)
{
/*	if (type == GYRO)
//	{
//		if(value)
//		{
//			GPIO_PORTG_DATA_R |= 0x04;
//		}
//		else
//		{
//			GPIO_PORTG_DATA_R &= ~0x04;
//		}
//	}
//
//	if (type == XM)
//	{
//		if(value)
//		{
//			GPIO_PORTG_DATA_R |= 0x01;
//		}
//		else
//		{
//			GPIO_PORTG_DATA_R &= ~0x01;
//		}
//	}
*/
	if (type == GYRO)
	{
		if(value)
		{
			GPIO_PORTB_DATA_R |= 0x04;
		}
		else
		{
			GPIO_PORTB_DATA_R &= ~0x04;
		}
	}

	if (type == XM)
	{
		if(value)
		{
			GPIO_PORTB_DATA_R |= 0x01;
		}
		else
		{
			GPIO_PORTB_DATA_R &= ~0x01;
		}
	}
}
/*#define ACCELEROMETER_SENSITIVITY 8192.0
//#define GYROSCOPE_SENSITIVITY 65.536
//
//#define M_PI 3.14159265359
//
```

```
//#define dt 0.01                                            // 10 ms sample rate!
//
//void ComplementaryFilter(short accData[3], short gyrData[3], float *pitch, float *roll)
//{
//    float pitchAcc, rollAcc;
//
//    // Integrate the gyroscope data -> int(angularSpeed) = angle
//    *pitch += ((float)gyrData[0] / GYROSCOPE_SENSITIVITY) * dt; // Angle around the X-
axis
//    *roll -= ((float)gyrData[1] / GYROSCOPE_SENSITIVITY) * dt;    // Angle around the
Y-axis
//
//    // Compensate for drift with accelerometer data if !bullshit
//    // Sensitivity = -2 to 2 G at 16Bit -> 2G = 32768 && 0.5G = 8192
//    int forceMagnitudeApprox = abs(accData[0]) + abs(accData[1]) + abs(accData[2]);
//    if (forceMagnitudeApprox > 8192 && forceMagnitudeApprox < 32768)
//    {
//     // Turning around the X axis results in a vector on the Y-axis
//        pitchAcc = atan2((float)accData[1], (float)accData[2]) * 180 / M_PI;
//        *pitch = *pitch * 0.98 + pitchAcc * 0.02;
//
//     // Turning around the Y axis results in a vector on the X-axis
//        rollAcc = atan2((float)accData[0], (float)accData[2]) * 180 / M_PI;
//        *roll = *roll * 0.98 + rollAcc * 0.02;
//    }
//}
*/


void SPIsingleByte(enum sensor type, unsigned long ulDataTx, unsigned long* ulDataRx)
{
       chipSelectPin(type,LOW);
       SSIDataPut(SSI1_BASE, ulDataTx);

       // Wait until SSI0 is done transferring all the data in the transmit FIFO.
       while(SSIBusy(SSI1_BASE))
       {}

       SSIDataGet(SSI1_BASE, ulDataRx);

       *ulDataRx &= 0x000000FF;
       chipSelectPin(type,HIGH);
}
unsigned long* GetData(enum sensor type, unsigned char numBytes, unsigned long*
ulDataTx,unsigned long* ulDataRx)
{
       unsigned int i;
       //static unsigned long ulDataRx[SIZE];
       /*unsigned long ulDataTx[SIZE] =
{0x00008F00,0x000009200,0x0000A000,0x0000A400,0x0000A500,0x0000A600};
       // ^^ XL important addresses
       //unsigned long ulDataTx[SIZE] =
{0x0000A800,0x00000A900,0x0000AA00,0x0000AB00,0x0000AC00,0x0000AD00};
       // ^^ Gyro XYZ data addresses
       //unsigned long ulDataTx[SIZE] =
{0x00008F00,0x00000A000,0x0000A100,0x0000A200,0x0000A300,0x0000A400};
       // ^^ Gyro important addresses
       */
```

```
	// clear debug Rx buffer
	for(i=0;i<numBytes;i++)
	{
		ulDataRx[i] = 0;
	}

	chipSelectPin(type,LOW);
	while(SSIDataGetNonBlocking(SSI1_BASE, &ulDataRx[i]))
	{
	}
	chipSelectPin(type,HIGH);
	Delay(10);

	for(i=0;i<numBytes;i++)
	{
		SPIsingleByte(type,ulDataTx[i],&ulDataRx[i]);

		if(i < numBytes)
		{
			printf("Tx: 0x%04X",(unsigned int)ulDataTx[i]); // prints 4 hex 
digits with leading zeros
			printf("%c", TAB);
			printf("Rx: 0x%02X",(unsigned int)ulDataRx[i]); // prints 2 hex 
digits with leading zeros
			printf("%c", NEWLINE);
			Delay(4000);		// delay ~1 sec at 12 MHz
		}
	}
	return  &ulDataRx[0];
}
void lowPassFilterData(enum sensor type, long dataX, long dataY, long dataZ)
{
	if(type == GYRO)
	{
		filtGx[5] = filtGx[4];
		filtGx[4] = filtGx[3];
		filtGx[3] = filtGx[2];
		filtGx[2] = filtGx[1];
		filtGx[1] = filtGx[0];
		filtGx[0] = dataX;

		filtGy[5] = filtGy[4];
		filtGy[4] = filtGy[3];
		filtGy[3] = filtGy[2];
		filtGy[2] = filtGy[1];
		filtGy[1] = filtGy[0];
		filtGy[0] = dataY;

		filtGz[5] = filtGz[4];
		filtGz[4] = filtGz[3];
		filtGz[3] = filtGz[2];
		filtGz[2] = filtGz[1];
		filtGz[1] = filtGz[0];
		filtGz[0] = dataZ;
	}
	else
	{
```

```
              filtAx[5] = filtAx[4];
              filtAx[4] = filtAx[3];
              filtAx[3] = filtAx[2];
              filtAx[2] = filtAx[1];
              filtAx[1] = filtAx[0];
              filtAx[0] = dataX;

              filtAy[5] = filtAy[4];
              filtAy[4] = filtAy[3];
              filtAy[3] = filtAy[2];
              filtAy[2] = filtAy[1];
              filtAy[1] = filtAy[0];
              filtAy[0] = dataY;

              filtAz[5] = filtAz[4];
              filtAz[4] = filtAz[3];
              filtAz[3] = filtAz[2];
              filtAz[2] = filtAz[1];
              filtAz[1] = filtAz[0];
              filtAz[0] = dataZ;

       }
}

void GetDataXYZ(enum sensor type)
{
/*     static int j = 0;
//     static int tempX = 0;
//     static int sum = 0;
//     int runSumX;
//     int runSumY;
//     int runSumZ;
       */
       unsigned int i;
       short tempAx,tempAy,tempAz;
       unsigned long ulDataRx[SIZE];
       unsigned long ulDataTx[SIZE] =
{0x0000A800,0x00000A900,0x0000AA00,0x0000AB00,0x0000AC00,0x0000AD00};

       SPIread(type, ulDataTx, ulDataRx);

       if(type == GYRO)
       {
              GyroStuff(ulDataTx,ulDataRx);
       }
       else
       {
              AccelStuff(ulDataTx, ulDataRx);
       }
}
/////////////
void GyroStuff(unsigned long* ulDataTx, unsigned long* ulDataRx)
{
              gx = (ulDataRx[0]&0xFF) + ((ulDataRx[1]&0xFF) << 8);
              gy = (ulDataRx[2]&0xFF) + ((ulDataRx[3]&0xFF) << 8);
              gz = (ulDataRx[4]&0xFF) + ((ulDataRx[5]&0xFF) << 8);

//            printf("%d%c",gx+545,NEWLINE);
//            printf("%d%c",gy-392,NEWLINE);
//            printf("%d%c%c",gz-4900,NEWLINE,NEWLINE);

              //gx = calcGyro(gx)+4095; // x offset data for 245DPS
              //gy = calcGyro(gy)-3010; // y offset data for 245DPS
              //gz = calcGyro(gz)+29376;//-12213; // z offset data for 245DPS

              gx = calcGyro(gx+600);
              //Fixed_sDecOut22s((unsigned long) gx); // x offset data for 245DPS
              gy = calcGyro(gy-392);
              //Fixed_sDecOut22s((unsigned long) gy); // y offset data for 245DPS
              gz = calcGyro(gz-4980);
              //Fixed_sDecOut22s((unsigned long) gz);//-12213; // z offset data for
245DPS


//            Fixed
//            Fixed_sDecOut22s((unsigned long) gx);
//            Fixed_sDecOut22s((unsigned long) gy);
//            Fixed_sDecOut22s((unsigned long) gz);
//            printf("%c",NEWLINE);printf("%c",NEWLINE);

              // http://www.hobbytronics.co.uk/accelerometer-gyro creates a good filter
              // output filter data
              lowPassFilterData(GYRO,(long)gx,(long)gy,(long)gz);
                     gx = (filtGx[0]+filtGx[1]+filtGx[2]+filtGx[3])/4;
                     gy = (filtGy[0]+filtGy[1]+filtGy[2]+filtGy[3])/4;
                     gz = (filtGz[0]+filtGz[1]+filtGz[2]+filtGz[3])/4;

//                   gx =
(filtGx[0]+filtGx[1]+filtGx[2]+filtGx[3]+filtGx[4]+filtGx[5])/6;
//                   gy =
(filtGy[0]+filtGy[1]+filtGy[2]+filtGy[3]+filtGy[4]+filtGy[5])/6;
//                   gz =
(filtGz[0]+filtGz[1]+filtGz[2]+filtGz[3]+filtGz[4]+filtGz[5])/6;

              if(filtGx[0] - filtGx[1] < 150)
                     gx = 0;
              if(filtGy[0] - filtGy[1] < 150)
                     gy = 0;
              if(filtGz[0] - filtGz[1] < 150)
                     gz = 0;
//            if(filtGx[1] - filtGx[0] < 150)
//                   gx = 0;
//            if(filtGy[1] - filtGy[0] < 150)
//                   gy = 0;
//            if(filtGz[1] - filtGz[0] < 150)
//                   gz = 0;

//            Fixed_sDecOut22s((unsigned long) gx);
//            Fixed_sDecOut22s((unsigned long) gy);
//            Fixed_sDecOut22s((unsigned long) gz);
//            printf("%c",NEWLINE);
//
         //if (gyroRate >= rotationThreshold || gyroRate <= -rotationThreshold)
       if (gx <= 24500 || gx >= -24500)
       {
    currentAngleX += gx;
```

```
  }
       if (gy <= 24500 || gy >= -24500)
       {
    currentAngleY += gy;
  }
       if (gz <= 24500 || gz >= -24500)
       {
    currentAngleZ += gz;
  }

       //Keep our angle between 0-359 degrees
  if (currentAngleX < 0)
    currentAngleX += 36000;
  else if (currentAngleX > 35900)
    currentAngleX -= 36000;
       //Keep our angle between 0-359 degrees
  if (currentAngleY < 0)
    currentAngleY += 36000;
  else if (currentAngleY > 35900)
    currentAngleY -= 36000;
       //Keep our angle between 0-359 degrees
  if (currentAngleZ < 0)
    currentAngleZ += 36000;
  else if (currentAngleZ > 35900)
    currentAngleZ -= 36000;


//     Fixed_sDecOut22s((unsigned long) gx);
       printf("%c",TAB);
//     Fixed_uDecOut2((unsigned long) currentAngleX);  printf("%c",NEWLINE);
//     Fixed_sDecOut22s((unsigned long) gy);
       printf("%c",TAB);
//     Fixed_uDecOut2((unsigned long) currentAngleY);  printf("%c",NEWLINE);
//  Fixed_sDecOut22s((unsigned long) gz);                            printf("%c",TAB);
//     Fixed_uDecOut2((unsigned long) currentAngleZ);  printf("%c",NEWLINE);


       //            Fixed_sDecOut22s((unsigned long) currentAngleX);
//            Fixed_sDecOut22s((unsigned long) currentAngleY);
//            Fixed_sDecOut22s((unsigned long) currentAngleZ);
              printf("%c",NEWLINE);

/*            if(j < 100)
//            {
//                   DataX[j] = gx;
//                   DataY[j] = gy;
//                   DataZ[j] = gz;
//                   j++;
//            }
//            if( j == 100)
//            {
//                   j = 0;
//                   runSumX=0;
//                   runSumY=0;
//                   runSumZ=0;
//                   for(j=0; j < 100; j++)
//                   {
//                          runSumX += DataX[j];
//                          runSumY += DataY[j];
//                          runSumZ += DataZ[j];
//
//                          if(j == 98)
//                          {
//                                 j++;j--;
//                          }
//                   }
//                   while(1)
//                   {
//                          printf("%d%c",runSumX/100,NEWLINE);
//                          printf("%d%c",runSumY/100,NEWLINE);
//                          printf("%d%c",runSumZ/100,NEWLINE);
//                          while(1)
//                          {}
//                   }
//            }

//            printf("Gx: %d%c",gx,NEWLINE);
//            printf("Gy: %d%c",gy,NEWLINE);
//            printf("Gz: %d%c",gz,NEWLINE);
*/




}






long findAngleMeas(long y, long x)
{
       unsigned long magnitude;
       unsigned long xDegree;
       unsigned long yDegree;

       long xDegreeSigned;
       long yDegreeSigned;

       long xTemp;
       long yTemp;
       const unsigned long MAX = 16384;

       if(x > MAX)
              {x = MAX;}
       if(y > MAX)
              {y = MAX;}
       if( x < 0)
              {x = 0;}
       if(y < 0)
              {y = 0;}

       xTemp = atan2(y,x);
       printf("AngleX: %u%c",xTemp,NEWLINE);
```

```
/*      xTemp = x;
//      yTemp = y;
//
//
//      //xDegree = atan2(
//
//
//      if(xTemp > MAX)
//      {
//              xTemp = MAX;
//      }
//      else if(xTemp < -MAX)
//      {
//              xTemp = -MAX;
//      }
//      if(yTemp > MAX)
//      {
//              yTemp = MAX;
//      }
//      else if(yTemp < -MAX)
//      {
//              yTemp = -MAX;
//      }
//
//      // put in the region of positive integers 0-32678
//      xTemp += MAX;
//      yTemp += MAX;
//      // zero -> -90 deg
//      // 2*MAX = 32768 -> 90 degrees
//      xDegree = (((((xTemp*100*355+37)/75)*179+56)/113)*24)/32768;
//      xDegreeSigned = (long)(xDegree-9000);
//
//      yDegree = (((((yTemp*100*355+37)/75)*179+56)/113)*24)/32768;
//      xDegreeSigned = (long)(yDegree-9000);
//
//      if (x < 0 && y > 0)
//      {
//
//
//      }
//      magnitude     = sqrt(x*x+y*y);
//
//
*/
return xTemp;
}

void AccelStuff(unsigned long* ulDataTx, unsigned long* ulDataRx)
{
       short tempAx,tempAy,tempAz;
       long lTempAx,lTempAy,lTempAz,mag;

       tempAx = (ulDataRx[0]&0xFF) + ((ulDataRx[1]&0xFF) << 8)+2030;
       tempAy = (ulDataRx[2]&0xFF) + ((ulDataRx[3]&0xFF) << 8);
       tempAz = (ulDataRx[4]&0xFF) + ((ulDataRx[5]&0xFF) << 8);

       lTempAx = abs((long)tempAx);
       lTempAy = abs((long)tempAy);
```

```
        lTempAz = abs((long)tempAz);

        mag = 4*sqrt((lTempAy*lTempAy+8)/16+(lTempAz*lTempAz+8)/16);
        XLangleX = findAngleMeas(lTempAx,mag);

        mag = 4*sqrt((lTempAx*lTempAx+8)/16+(lTempAz*lTempAz+8)/16);
        XLangleY = findAngleMeas(lTempAy,mag);

        mag = 4*sqrt((lTempAx*lTempAx+8)/16+(lTempAy*lTempAy+8)/16);
        XLangleZ = findAngleMeas(lTempAz,mag);

        // This wont work for  the +/- 16g range
        ax = calcAccel(tempAx/((aScale+1)));
        ay = calcAccel(tempAy/((aScale+1)));////(aScale));
        az = calcAccel(tempAz/((aScale+1)));///(aScale));

        //ax = (2*ax + 21)/ 2; // calibration offset
        ay = (ay*100)/104-2;

//      mag = sqrt(ay*ay+az*az);
//      findAngleMeas(ax,mag);
//
//      mag = sqrt(ax*ax+az*az);
//      findAngleMeas(ay,mag);
//
//      mag = sqrt(ax*ax+ay*ay);
//      findAngleMeas(az,mag);

//      printf("%d%c",tempAx,NEWLINE);
//      printf("%d%c",tempAy,NEWLINE);
//      printf("%d%c",tempAz,NEWLINE);

// Filter accel data
/////////////////////////////////////////////////////////////
              lowPassFilterData(XM,(long)ax,(long)ay,(long)az);
              ax = (filtAx[0]+filtAx[1]+filtAx[2]+filtAx[3])/4;
              ay = (filtAy[0]+filtAy[1]+filtAy[2]+filtAy[3])/4;
              az = (filtAz[0]+filtAz[1]+filtAz[2]+filtAz[3])/4;

//                    ax =
(filtAx[0]+filtAx[1]+filtAx[2]+filtAx[3]+filtAx[4]+filtAx[5])/6;
//                    ay =
(filtAy[0]+filtAy[1]+filtAy[2]+filtAy[3]+filtAy[4]+filtAy[5])/6;
//                    az =
(filtAz[0]+filtAz[1]+filtAz[2]+filtAz[3]+filtAz[4]+filtAz[5])/6;
/////////////////////////////////////////////////////////////

//            Fixed_sDecOut3((long) ax);
//            Fixed_sDecOut3((long) ay);
//            Fixed_sDecOut3((long) az);

        Fixed_sDecOut22s((long) ax);
printf("%d",ax);printf("%c",NEWLINE);//printf("%d%c",atan2(ax,ay),NEWLINE);
        Fixed_sDecOut22s((long) ay);
printf("%d",ay);printf("%c",NEWLINE);//printf("%d%c",atan2(ay,az),NEWLINE);
        Fixed_sDecOut22s((long) az);
printf("%d",az);printf("%c",NEWLINE);//printf("%d%c",atan2(ax,az),NEWLINE);
        Delay(250000);
```

```
//
//		Fixed_uDecOut2((unsigned long) abs(ax/100));
//		Fixed_uDecOut2((unsigned long) abs(ay/100));
//		Fixed_uDecOut2((unsigned long) abs(az/100));

}
signed long calcGyro(short gyro)
{
	signed long scaledGyroData;
	//char sign = 0;
	if(gyro < 0)
	{
		//sign = -1;
		scaledGyroData = (gRes*(long)(~gyro+1) + 5000)/10000;
		scaledGyroData = ~scaledGyroData + 1;
	}
	else
	{
		scaledGyroData = (gRes*(long)gyro + 5000)/10000;
	}
	// Return the gyro raw reading times our pre-calculated DPS/(ADC tick):
	return scaledGyroData;
}

signed long calcAccel(short accel)
{
	signed long scaledAccelData;
	if(accel < 0)
	{
		scaledAccelData = (aRes * (long)(~accel+1) + 50000)/100000;
		scaledAccelData = ~scaledAccelData + 1;
	}
	else
	{ scaledAccelData = (aRes * (long)accel + 50000)/100000; }
	// Return the accel raw reading times our pre-calculated g's / (ADC tick):
	return scaledAccelData;
}

void initGyro(void)
{
	unsigned long ulRead;
	unsigned long ulSend =
0x00000A000;//,0x0000A100,0x0000A200,0x0000A300,0x0000A400};
	/* CTRL_REG1_G sets output data rate, bandwidth, power-down and enables
	// Bits[7:0]: DR1 DR0 BW1 BW0 PD Zen Xen Yen
	// DR[1:0] - Output data rate selection
	// 00=95Hz, 01=190Hz, 10=380Hz, 11=760Hz
	// BW[1:0] - Bandwidth selection (sets cutoff frequency)
	// Value depends on ODR. See datasheet table 21.
	// PD - Power down enable (0=power down mode, 1=normal or sleep mode)
	// Zen, Xen, Yen - Axis enable (o=disabled, 1=enabled)
	// gWriteByte(CTRL_REG1_G, 0x0F); // Normal mode, enable all axes
	*/
DAC_Out(GYRO, 0x4F,CTRL_REG1_G,0);
//	GetData(GYRO,1,&ulSend,&ulRead);

	/* CTRL_REG2_G sets up the HPF
	// Bits[7:0]: 0 0 HPM1 HPM0 HPCF3 HPCF2 HPCF1 HPCF0
	// HPM[1:0] - High pass filter mode selection
	// 00=normal (reset reading HP_RESET_FILTER, 01=ref signal for filtering,
	// 10=normal, 11=autoreset on interrupt
	// HPCF[3:0] - High pass filter cutoff frequency
	// Value depends on data rate. See datasheet table 26.
	gWriteByte(CTRL_REG2_G, 0x00); // Normal mode, high cutoff frequency
	*/
	DAC_Out(GYRO, 0x00, CTRL_REG2_G, 0);

	/* CTRL_REG3_G sets up interrupt and DRDY_G pins
	// Bits[7:0]: I1_IINT1 I1_BOOT H_LACTIVE PP_OD I2_DRDY I2_WTM I2_ORUN I2_EMPTY
	// I1_INT1 - Interrupt enable on INT_G pin (0=disable, 1=enable)
	// I1_BOOT - Boot status available on INT_G (0=disable, 1=enable)
	// H_LACTIVE - Interrupt active configuration on INT_G (0:high, 1:low)
	// PP_OD - Push-pull/open-drain (0=push-pull, 1=open-drain)
	// I2_DRDY - Data ready on DRDY_G (0=disable, 1=enable)
	// I2_WTM - FIFO watermark interrupt on DRDY_G (0=disable 1=enable)
	// I2_ORUN - FIFO overrun interrupt on DRDY_G (0=disable 1=enable)
	// I2_EMPTY - FIFO empty interrupt on DRDY_G (0=disable 1=enable)
	// Int1 enabled (pp, active low), data read on DRDY_G:
	gWriteByte(CTRL_REG3_G, 0x88);
	*/

	/* CTRL_REG4_G sets the scale, update mode
	// Bits[7:0] - BDU BLE FS1 FS0 - ST1 ST0 SIM
	// BDU - Block data update (0=continuous, 1=output not updated until read
	// BLE - Big/little endian (0=data LSB @ lower address, 1=LSB @ higher add)
	// FS[1:0] - Full-scale selection
	// 00=245dps, 01=500dps, 10=2000dps, 11=2000dps
	// ST[1:0] - Self-test enable
	// 00=disabled, 01=st 0 (x+, y-, z-), 10=undefined, 11=st 1 (x-, y+, z+)
	// SIM - SPI serial interface mode select
	// 0=4 wire, 1=3 wire
  gWriteByte(CTRL_REG4_G, 0x00); // Set scale to 245 dps
	*/
DAC_Out(GYRO, 0x90, CTRL_REG4_G, 0); //BDU & 500 DPS

	/* CTRL_REG5_G sets up the FIFO, HPF, and INT1
	// Bits[7:0] - BOOT FIFO_EN - HPen INT1_Sel1 INT1_Sel0 Out_Sel1 Out_Sel0
	// BOOT - Reboot memory content (0=normal, 1=reboot)
	// FIFO_EN - FIFO enable (0=disable, 1=enable)
	// HPen - HPF enable (0=disable, 1=enable)
	// INT1_Sel[1:0] - Int 1 selection configuration
	// Out_Sel[1:0] - Out selection configuration
	gWriteByte(CTRL_REG5_G, 0x00);

	// Temporary !!! For testing !!! Remove !!! Or make useful !!!
	configGyroInt(0x2A, 0, 0, 0, 0); // Trigger interrupt when above 0 DPS...
	*/
}


void initAccel()
{
	/* CTRL_REG0_XM (0x1F) (Default value: 0x00)
	// Bits (7-0): BOOT FIFO_EN WTM_EN 0 0 HP_CLICK HPIS1 HPIS2
	// BOOT - Reboot memory content (0: normal, 1: reboot)
	// FIFO_EN - Fifo enable (0: disable, 1: enable)
	// WTM_EN - FIFO watermark enable (0: disable, 1: enable)
	// HP_CLICK - HPF enabled for click (0: filter bypassed, 1: enabled)
	// HPIS1 - HPF enabled for interrupt generator 1 (0: bypassed, 1: enabled)
	// HPIS2 - HPF enabled for interrupt generator 2 (0: bypassed, 1 enabled)
  //xmWriteByte(CTRL_REG0_XM, 0x00);
	*/
  DAC_Out(XM, 0x00, CTRL_REG0_XM,0);

	/* CTRL_REG1_XM (0x20) (Default value: 0x07)
	// Bits (7-0): AODR3 AODR2 AODR1 AODR0 BDU AZEN AYEN AXEN
	// AODR[3:0] - select the acceleration data rate:
	// 0000=power down, 0001=3.125Hz, 0010=6.25Hz, 0011=12.5Hz,
	// 0100=25Hz, 0101=50Hz, 0110=100Hz, 0111=200Hz, 1000=400Hz,
	// 1001=800Hz, 1010=1600Hz, (remaining combinations undefined).
	// BDU - block data update for accel AND mag
	// 0: Continuous update
	// 1: Output registers aren't updated until MSB and LSB have been read.
	// AZEN, AYEN, and AXEN - Acceleration x/y/z-axis enabled.
	// 0: Axis disabled, 1: Axis enabled
  // xmWriteByte(CTRL_REG1_XM, 0x67); // 100Hz data rate, x/y/z all enabled
	// Serial.println(xmReadByte(CTRL_REG1_XM));
  */
  DAC_Out(XM, 0x6F, CTRL_REG1_XM,0);

	/* CTRL_REG2_XM (0x21) (Default value: 0x00)
	// Bits (7-0): ABW1 ABW0 AFS2 AFS1 AFS0 AST1 AST0 SIM
	// ABW[1:0] - Accelerometer anti-alias filter bandwidth
	// 00=773Hz, 01=194Hz, 10=362Hz, 11=50Hz
	// AFS[2:0] - Accel full-scale selection
	// 000=+/-2g, 001=+/-4g, 010=+/-6g, 011=+/-8g, 100=+/-16g
	// AST[1:0] - Accel self-test enable
	// 00=normal (no self-test), 01=positive st, 10=negative st, 11=not allowed
	// SIM - SPI mode selection
	// 0=4-wire, 1=3-wire
  // xmWriteByte(CTRL_REG2_XM, 0x00); // Set scale to 2g
*/
  DAC_Out(XM, 0x00,CTRL_REG2_XM,0);
	/* CTRL_REG3_XM is used to set interrupt generators on INT1_XM
	// Bits (7-0): P1_BOOT P1_TAP P1_INT1 P1_INT2 P1_INTM P1_DRDYA P1_DRDYM P1_EMPTY

	// Accelerometer data ready on INT1_XM (0x04)
  // xmWriteByte(CTRL_REG3_XM, 0x04);
	*/
}


// from the book, Sec. 7-5 pg 372
// send the 16-bit data to the SSI, return a reply
void DAC_Out(enum sensor type, unsigned char data, unsigned char subAddress, unsigned
char csPin)
{
	unsigned short TxBytes = 0;
	TxBytes |= (0x00FF & data);
	TxBytes &= SINGLEWRITE; // sets 2 MSB to 0
	TxBytes |= ((0x003F & subAddress) << 8); // shift 6-bit address bits into place,
i.e. bit13 to bit8;
```

```
        // first write the address then the data
        /////////////////////////////////////
        // I need to add a specifier for the chip select bit
        // to differentiate among gyro, accel, & magnetometer
        /////////////////////////////////////
        //GPIO_PORTG_DATA_R &= ~0x04;// Open communication
        chipSelectPin(type,LOW);
        // write the MS_bit first
        // If write, bit 0 (MS) should be 0
        // If single write, bit 1 should be 0
        while(SSI1_SR_TNF == 0)
        {}; // wait until room in FIFO
        SSI1_DR_R = TxBytes;  // data out
        Delay(60); // for some reason it needs this delay of >= 48, it doesn't get
                                    // stored in the XL/G if not

        //GPIO_PORTG_DATA_R |= 0x04; // Close communication
        chipSelectPin(type,HIGH);
}

void initSPI()
{
        /*
        // Now set up the SPI port to talk to the LSM9DS0
  */
  SysCtlPeripheralEnable(SYSCTL_PERIPH_SSI1);
  SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);
  GPIOPinConfigure(GPIO_PE0_SSI1CLK);
  GPIOPinConfigure(GPIO_PE1_SSI1FSS);
  GPIOPinConfigure(GPIO_PE2_SSI1RX);
  GPIOPinConfigure(GPIO_PE3_SSI1TX);
  GPIOPinTypeSSI(GPIO_PORTE_BASE,GPIO_PIN_3|GPIO_PIN_2|GPIO_PIN_1|GPIO_PIN_0);

  /* LSM9DS0 SPI Specs
  // Max SPI Clock : 10 MHz
  // Data Order : MSB transmitted first
  // Clock Polarity: high when idle => SPO = 1;
  // Clock Phase : sample on rising edge => SPH = 1
  //
  // We read and write 8 bits to the LSM9DS0 but we need the Stellaris to drive the
clock.
        // We send 16 bits, MSB are specifiers(R/W,Inc,Address respectively), LSB is data
(send MSbit 1st)
        // R/W - bit 15, Read = 1, Write = 0;
        // Inc - bit 14, (Auto Increment Address) Inc = 1
        // Address - bit 13 to bit 8, MOSI address to Tx data to
        // data - bit 7 to bit 0, data to store at said address
        // When we do a write, all 16 bits are written.
  // When we do a read, we take the command byte and shift left 8 bits, writing 0's as
the last eight bits.
  // Since this mode is full duplex, we'll get 16 bits back but mask off the top 8,
leaving only the read data we are
  */
  SSIConfigSetExpClk(SSI1_BASE, SysCtlClockGet(), SSI_FRF_MOTO_MODE_3, SSI_MODE_MASTER,
5000000, 16);
  SSIEnable(SSI1_BASE);
}
```

```
void calcgRes()
{
	/* Possible gyro scales (and their register bit settings) are:
	// 245 DPS (00), 500 DPS (01), 2000 DPS (10). Here's a bit of an algorithm
	// to calculate DPS/(ADC tick) based on that 2-bit value:
	// look at http://electronics.stackexchange.com/questions/39024/how-do-i-get-gyro-
sensor-data-l3g4200d-into-degrees-sec

	*/
	switch (gScale)
	{
		case G_SCALE_245DPS:
		gRes = (245*1000000 + 16384)/32768;
		break;
		case G_SCALE_500DPS:
		gRes = (500*1000000 + 16384)/32768;
		break;
		case G_SCALE_2000DPS:
		gRes = (2000*1000000 + 16384)/32768;
		break;
	}
}

void calcaRes()
{
	/* Possible accelerometer scales (and their register bit settings) are:
	// 2 g (000), 4g (001), 6g (010) 8g (011), 16g (100). Here's a bit of an
	// algorithm to calculate g/(ADC tick) based on that 3-bit value:
	*/
	aRes = aScale == A_SCALE_16G ? (16.0*1000000) / 32768.0 :
	((((unsigned long) aScale + 1.0)*10000000) * 2.0) / 32768.0;
	printf("%caRes: %lu %c",NEWLINE, aRes,NEWLINE);
}

void SPIread(enum sensor type, unsigned long* ulDataTx, unsigned long* ulDataRx)
{
	unsigned char i;
		// Clear Rx Buffer
	for(i=0;i<SIZE;i++)
	{
		ulDataRx[i] = 0;
	}
	/* Read any residual data from the SSI port.  This makes sure the receive
	// FIFOs are empty, so we don't read any unwanted junk.  This is done here
	// because the SPI SSI mode is full-duplex, which allows you to send and
	// receive at the same time.  The SSIDataGetNonBlocking function returns
	// "true" when data was returned, and "false" when no data was returned.
	// The "non-blocking" function checks if there is any data in the receive
	// FIFO and does not "hang" if there isn't.
	*/
	chipSelectPin(type,LOW);
	while(SSIDataGetNonBlocking(SSI1_BASE, &ulDataRx[i]))
	{
	}
	chipSelectPin(type,HIGH);
	Delay(1000);
	/* Send the data using the "blocking" put function.  This function
	// will wait until there is room in the send FIFO before returning.
```

```
        // This allows you to assure that all the data you send makes it into
        // the send FIFO.
        */

        // debuggin on logic analyzer to calculate
        // dT for the Complimentary filter
        if(type == GYRO)
                {       GPIO_PORTG_DATA_R ^= 0x01;}
        for(i=0;i<SIZE;i++)
        {
                /*
                // Send the data using the "blocking" put function.  This function
                // will wait until there is room in the send FIFO before returning.
                // This allows you to assure that all the data you send makes it into
                // the send FIFO.
                */
                chipSelectPin(type,LOW);
                SSIDataPut(SSI1_BASE, ulDataTx[i]);
                /*
                // Wait until SSI1 is done transferring all the data in the transmit FIFO.
                */
                while(SSIBusy(SSI1_BASE))
                {}
                /*
                // Receive the data using the "blocking" Get function. This function
                // will wait until there is data in the receive FIFO before returning.
                */
                SSIDataGet(SSI1_BASE, &ulDataRx[i]);
                /*
                // Since we are using 8-bit data, mask off the MSB that was read full
                // duplex while we were sending our command byte.
                */
                ulDataRx[i] &= 0x000000FF;
                /*
                // Display the data that SSI1 received.
                // The datasheet says this value should be 4.
                */
                chipSelectPin(type,HIGH);
        }
}
```

Nokia LCD

```
// OLEDTestMain.c
// Runs on LM3S1968
// Test Output.c by sending various characters and strings to
// the OLED display and verifying that the output is correct.
// Daniel Valvano
// July 28, 2011

/* This example accompanies the book
   "Embedded Systems: Real Time Interfacing to the Arm Cortex M3",
   ISBN: 978-1463590154, Jonathan Valvano, copyright (c) 2011
   Section 3.4.5

 Copyright 2011 by Jonathan W. Valvano, valvano@mail.utexas.edu
    You may use, edit, run or distribute this file
    as long as the above copyright notice remains
 THIS SOFTWARE IS PROVIDED "AS IS".  NO WARRANTIES, WHETHER EXPRESS, IMPLIED
```

```c
 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/
 */

#include <stdio.h>
#include "driverlib/gpio.h"
#include "Output.h"
#include "Pll.h"
#include "lm3s1968.h"
#include "sysctl.h"
#include "Dac.h"

// image of a longhorn
const char Longhorn[] = {
  0x08, 0x08, 0x08, 0x08, 0x08, 0x18, 0x18, 0x18, 0x38, 0x30, 0x30, 0x30, 0x70, 0xF0,
  0xE0, 0xC0, 0xC0, 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x80, 0xC0, 0xC0, 0xE0, 0xE0, 0xF0, 0x70,
  0x70, 0x30, 0x30, 0x18, 0x18, 0x18, 0x18, 0x08, 0x08, 0x08, 0x08, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x01, 0x03, 0x03, 0x03, 0x07, 0x0F, 0x0E, 0x0C, 0x1C, 0x38, 0x38, 0xB8, 0xF8, 0xF0,
  0xF0, 0xF8, 0xF8, 0xF8, 0xF8, 0xF8, 0xF8, 0xFC, 0xFC, 0xF8, 0xF8, 0xF8, 0xF8, 0xF8,
  0xF8, 0xF8, 0xF8, 0xF8, 0xF8, 0xF8, 0xF8, 0xF8, 0xF0, 0xE0, 0xE0, 0xF0, 0xF0, 0xF0,
  0xF0, 0x78, 0x38, 0x3C, 0x1C, 0x1F, 0x0F, 0x07, 0x03, 0x03, 0x01, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x06, 0x0F, 0x0F, 0x0F, 0x0F,
  0x0F, 0x07, 0x07, 0x07, 0x1F, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
  0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0x1F, 0x0F, 0x1F, 0x1F, 0x1F, 0x1F,
  0x1F, 0x1F, 0x1E, 0x0E, 0x04, 0x00, 0x00, 0x00, 0x0F, 0x05, 0x0B, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x01, 0x07, 0x1F, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
  0xFF, 0xFF, 0xFF, 0xFF, 0x1F, 0x07, 0x03, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x3C, 0x7F, 0x7F, 0xFF, 0xFF, 0xFF, 0xFF,
  0xFF, 0xFF, 0x7F, 0x3F, 0x28, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
};

// image of a longhorn loves 319k
const char Longhorn2[] = {
  0x08, 0x08, 0x08, 0x08, 0x08, 0x18, 0x18, 0x18, 0x38, 0x30, 0x30, 0x30, 0x70, 0xF0,
  0xE0, 0xC0, 0xC0, 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x80, 0xC0, 0xC0, 0xE0, 0xE0, 0xF0, 0x70,
  0x70, 0x30, 0x30, 0x18, 0x18, 0x18, 0x18, 0x08, 0x08, 0x08, 0x08, 0x00, 0x00, 0x00,
  0xF0, 0x08, 0x04, 0x24, 0x24, 0xE4, 0x24, 0x24, 0x04, 0x04, 0x04, 0x04, 0x04, 0x08,
  0xF1, 0x03, 0x03, 0x03, 0x07, 0x0F, 0x0E, 0x0C, 0x1C, 0x38, 0x38, 0xB8, 0xF8, 0xF0,
  0xF0, 0xF8, 0xF8, 0xF8, 0xF8, 0xF8, 0xF8, 0xFC, 0xFC, 0xF8, 0xF8, 0xF8, 0xF8, 0xF8,
  0xF8, 0xF8, 0xF8, 0xF8, 0xF8, 0xF8, 0xF8, 0xF8, 0xF0, 0xE0, 0xE0, 0xF0, 0xF0, 0xF0,
  0xF0, 0x78, 0x38, 0x3C, 0x1C, 0x1F, 0x0F, 0x07, 0x03, 0x03, 0x01, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0xFF, 0x00, 0x00, 0x04, 0x04, 0x07, 0x84, 0x44, 0x40, 0x80, 0x40, 0x40, 0x80, 0x00,
  0x00, 0x01, 0x02, 0xFC, 0x00, 0x00, 0x00, 0x00, 0x00, 0x06, 0x0F, 0x0F, 0x0F, 0x0F,
  0x0F, 0x07, 0x07, 0x07, 0x1F, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
  0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0x1F, 0x0F, 0x1F, 0x1F, 0x1F, 0x1F,
  0x1F, 0x1F, 0x1E, 0x0E, 0x04, 0x00, 0x00, 0x00, 0x0F, 0x05, 0x0B, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0xFF, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01, 0x02, 0x84, 0x88, 0x84, 0x02, 0x01, 0x00,
  0x00, 0x80, 0x00, 0x00, 0x01, 0x82, 0x84, 0x84, 0x84, 0x04, 0x04, 0x84, 0x04, 0x04,
  0x84, 0x08, 0x10, 0xE0, 0x00, 0x01, 0x07, 0x1F, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
  0xFF, 0xFF, 0xFF, 0xFF, 0x1F, 0x07, 0x03, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x0F, 0x10, 0x20, 0x40, 0x80, 0x80, 0x80, 0x80, 0x88, 0x88, 0x8A, 0x85, 0x80, 0x80,
  0x89, 0x8F, 0x88, 0x80, 0x80, 0x83, 0x82, 0x82, 0x8F, 0x80, 0x80, 0x8F, 0x82, 0x85,
  0x88, 0x80, 0x80, 0x8F, 0x50, 0x20, 0x00, 0x3C, 0x7F, 0x7F, 0xFF, 0xFF, 0xFF, 0xFF,
  0xFF, 0xFF, 0x7F, 0x3F, 0x28, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
};


const char smiley16bit [] = {
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00,
        0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01, 0xC0, 0x40, 0x00,
0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x01, 0xC0, 0xE0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x01, 0xC0, 0xE0,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01, 0xC0, 0xE0, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00,
        0x00, 0x01, 0xC0, 0xE0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01, 0xC0,
0xE0, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x01, 0xC0, 0xE0, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x01,
        0xC0, 0x70, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01, 0xC0, 0x70, 0x00,
0x00, 0x00, 0x00,
        0x00, 0x00, 0x10, 0x01, 0xC0, 0x70, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x38,
0x01, 0xC0, 0x70,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x38, 0x01, 0xC0, 0x70, 0x02, 0x00, 0x00,
0x00, 0x00, 0x00,
        0x38, 0x01, 0xC0, 0x70, 0x07, 0x00, 0x00, 0x00, 0x00, 0x00, 0x38, 0x01, 0xC0,
0x70, 0x0F, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x3C, 0x01, 0xE0, 0x70, 0x1E, 0x00, 0x00, 0x00, 0x00,
0x00, 0x1E, 0x00,
        0xE0, 0x70, 0x3C, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0F, 0x00, 0x40, 0x70, 0x78,
0x00, 0x00, 0x00,
        0x00, 0x00, 0x07, 0x80, 0x00, 0x70, 0xF8, 0x00, 0x00, 0x00, 0x00, 0x00, 0x03,
0xC0, 0x00, 0x21,
        0xF0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xE0, 0x00, 0x07, 0xC0, 0x00, 0x00,
0x00, 0x00, 0x00,
        0x00, 0xF8, 0x00, 0x1F, 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x7E, 0x00,
0x3E, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x1F, 0xFF, 0xFC, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x0F,
        0xFF, 0xF0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01, 0xFF, 0xC0, 0x00,
0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
};


const char smiley16bit0 [] = {
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x80,
0xC0, 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x80, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0xFF, 0xFF, 0xFF, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x3F, 0xFF, 0xFF,
0xC0,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x3E, 0x7F, 0xFE, 0xE0, 0xC0,
0x80,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x3F, 0x7F, 0xFF, 0x60, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0xFF, 0xFF, 0xFF, 0x00, 0x00, 0x00, 0x00, 0x00, 0x80, 0xC0, 0xE0, 0xF0, 0x78, 0x3C,
0x18,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x01, 0x03, 0x03, 0x0F, 0x1E, 0x1C, 0x38, 0x78, 0x70, 0x70, 0xE0, 0xE0, 0xE0, 0xE0,
0xE0,
0xE0, 0xE0, 0xE0, 0xE0, 0xE0, 0xE1, 0x73, 0x79, 0x38, 0x3C, 0x1C, 0x0E, 0x0F, 0x07, 0x03,
0x03,
0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
};

const char HelloworldMono [] = {
```

```
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xC0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xC0,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xE0, 0x00, 0x00, 0x00, 0xE0, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFF, 0x04, 0x04, 0x04,
0x04,
0x04, 0x04, 0xFF, 0x00, 0x00, 0x7C, 0x92, 0x92, 0x92, 0x92, 0x9C, 0x00, 0x00, 0xFF, 0x00,
0x00,
0x00, 0xFF, 0x00, 0x00, 0x00, 0x7C, 0x82, 0x82, 0x82, 0x82, 0x7C, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x0E, 0xF0, 0x00, 0x80, 0x70, 0x0E, 0x70, 0x80, 0x00, 0xF0, 0x0E, 0x00,
0x00,
0x00, 0xE0, 0x10, 0x10, 0x10, 0x10, 0xE0, 0x00, 0x00, 0xF0, 0x20, 0x10, 0x10, 0x00, 0xFF,
0x00,
0x00, 0x00, 0xE0, 0x10, 0x10, 0x10, 0x20, 0xFF, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0xFF,
0x00, 0x00, 0x00, 0x00, 0xFF, 0x00, 0x00, 0x00, 0x00, 0xFF, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x07, 0x01, 0x00, 0x00, 0x00, 0x01,
0x07,
0x00, 0x00, 0x00, 0x00, 0x00, 0x03, 0x04, 0x04, 0x04, 0x04, 0x03, 0x00, 0x00, 0x07, 0x00,
0x00,
0x00, 0x00, 0x07, 0x00, 0x00, 0x00, 0x03, 0x04, 0x04, 0x04, 0x02, 0x07, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x06, 0x00, 0x00, 0x00, 0x00, 0x06, 0x00, 0x00, 0x00, 0x00, 0x06, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
};

const char Course1 [] = {
       0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFE, 0x02,
0x02,
0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02,
0x02,
0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02,
0x02,
0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02,
0x02,
0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02,
0x02,
0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x82, 0x42, 0x22, 0xF2, 0x02, 0xFE,
0x00,
0x00, 0xFF, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x1C, 0x22, 0x41, 0x41, 0x41, 0x22, 0x1C,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x18, 0x24, 0x42, 0x81, 0x00, 0x00,
0x00,
0xFF, 0x00, 0xFF, 0x00, 0x00, 0x7F, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40,
0x40,
0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40,
0x40,
0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40,
0x40,
0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40,
0x40,
0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40,
0x40,
0x40, 0x41, 0x42, 0x44, 0x4F, 0x40, 0x7F, 0x00,
};

const char Course2 [] = {
0x00, 0xFE, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02,
0x02,
0xC2, 0x22, 0x12, 0x12, 0x12, 0x22, 0xC2, 0x02, 0x02, 0xFE, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0xFF, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x01, 0x02, 0x04, 0x04, 0x04, 0x02, 0x01, 0x00, 0x00, 0xFF, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFF, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0xFF, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFF, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x03, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02,
0x02,
0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02,
0x02,
0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02,
0x02,
0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x82, 0x42, 0x22, 0xF2, 0x02, 0xFE,
0x00,
0x00, 0xFF, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x18, 0x24, 0x42, 0x81, 0x00, 0x00,
0x00,
0xFF, 0x00, 0xFF, 0x00, 0x00, 0x7F, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40,
0x40,
0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40,
0x40,
0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40,
0x40,
0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40,
0x40,
0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40,
0x40,
0x40, 0x41, 0x42, 0x44, 0x4F, 0x40, 0x7F, 0x00,
};

const char Course3 [] = {
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x80, 0x40, 0x20, 0x10, 0x08, 0x04,
0x02,
0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02,
0x02,
0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02,
0x02,
0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02,
0x02,
0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x82, 0x42,
0x22,
0xF2, 0x02, 0xFE, 0x00, 0x00, 0x80, 0x40, 0x20, 0x10, 0x08, 0x04, 0x02, 0x01, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x18, 0x24,
0x42,
0x81, 0x00, 0x00, 0x00, 0xFF, 0x00, 0xFF, 0x00, 0x00, 0xFF, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x80, 0x40, 0x40, 0x40, 0x40, 0x40,
0x40,
0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40,
0x40,
0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40,
0x40,
0x40, 0x40, 0x40, 0x40, 0x40, 0x41, 0x42, 0x44, 0x4F, 0x40, 0x7F, 0x00, 0x00, 0xFF, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x80, 0x40, 0x20, 0x10, 0x08, 0x04, 0x02, 0x01, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0xFF, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xE0, 0x10, 0x08, 0x08, 0x08, 0x10,
0xE0,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFF, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x7F, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x41,
0x42,
0x42, 0x42, 0x41, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x7F, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
};

const char Course4 [] = {
0x00, 0xFE, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02,
0x02,
0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02,
0x02,
0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02,
0x02,
0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02,
0x02,
0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02,
0x02,
0x02, 0x02, 0xFE, 0x00, 0x00, 0xFF, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x1C, 0x22, 0x41,
0x41,
0x41, 0x22, 0x1C, 0x00, 0x00, 0x00, 0xFF, 0x00, 0x00, 0xFF, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0xC0, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40,
0x40,
0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40,
0x40,
0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40,
0x40,
0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x7F, 0x00, 0x00, 0xFF, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x03, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02,
0x02,
0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02,
0x02,
0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02,
0x02,
0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x82, 0x42, 0x22, 0xF2, 0x02, 0xFE,
0x00,
0x00, 0xFF, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x18, 0x24, 0x42, 0x81, 0x00, 0x00,
0x00,
0xFF, 0x00, 0xFF, 0x00, 0x00, 0x7F, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40,
0x40,
0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40,
0x40,
0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40,
0x40,
0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40,
0x40,
0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40,
0x40,
0x40, 0x41, 0x42, 0x44, 0x4F, 0x40, 0x7F, 0x00,
};

const char YouWin [] = {
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x04, 0x10, 0x00, 0x80, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x20, 0x00, 0x10, 0x10, 0x10, 0x20, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x02, 0x20, 0x00, 0x00,
0x00,
0x00, 0x00, 0x0E, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x02, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x20, 0x00, 0x00, 0x10, 0x10, 0x20, 0x00, 0x00, 0x00, 0x00, 0x00, 0x03, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x1F, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x08, 0x10, 0x10, 0x10, 0x00, 0x08, 0x00,
0x00,
0x00, 0x00, 0x00, 0x10, 0x10, 0x00, 0x00, 0x08, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x08, 0x08, 0x00, 0x00, 0x00, 0x00, 0x01, 0x08, 0x08, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x18, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
};




void DisableInterrupts(void); // Disable interrupts
void EnableInterrupts(void);  // Enable interrupts
long StartCritical (void);    // previous I bit, disable interrupts
void EndCritical(long sr);    // restore I bit to previous value
void WaitForInterrupt(void);  // low power mode

unsigned char array[10];
```

```
// delay function for testing from sysctl.c
// which delays 3*ulCount cycles
#ifdef __TI_COMPILER_VERSION__
       //Code Composer Studio Code
       void Delay(unsigned long ulCount){
       __asm (        "    subs    r0, #1\n"
                      "    bne     Delay\n"
                      "    bx      lr\n");
}

#else
       //Keil uVision Code
       __asm void
       Delay(unsigned long ulCount)
       {
    subs    r0, #1
    bne     Delay
    bx      lr
       }

#endif

void PortG_Init(void)
{
       volatile unsigned long delay;
       SYSCTL_RCGC2_R |= SYSCTL_RCGC2_GPIOG;

       delay = SYSCTL_RCGC2_R;

       GPIO_PORTG_DIR_R |= GPIO_PIN_2;
       GPIO_PORTG_DEN_R |= GPIO_PIN_2;
       GPIO_PORTG_AFSEL_R &= ~GPIO_PIN_2;
       GPIO_PORTG_DATA_R |= GPIO_PIN_2;
       Delay(100);
       GPIO_PORTG_DATA_R &= ~GPIO_PIN_2;
}

int main(void)
{
       int level = 0;
       unsigned short count = 0;
       DisableInterrupts();
       PLL_Init();
  SysCtlClockSet(SYSCTL_SYSDIV_4 | SYSCTL_USE_PLL | SYSCTL_OSC_MAIN |
                 SYSCTL_XTAL_8MHZ); // 50 MHz Clock
       PortG_Init();
 // Output_Init();
 // Output_Color(15);
 // printf("Hello, world.");
 // printf("%c", NEWLINE);
 // Delay(4000000);           // delay ~1 sec at 12 MHz

       Nokia_LCD_InitSSI1();
       Delay(10000000);

       for(count=0; count<5; count=count+1)
       {
            //Nokia5110_DrawFullImage(HelloworldMono);
    //Nokia5110_DrawFullImage(Longhorn);
    Delay(16666667);                    // delay ~1 sec at 50 MHz
    //Nokia5110_DrawFullImage(Longhorn2);
    Delay(16666667);                    // delay ~1 sec at 50 MHz
  }
  count = 0;
  Nokia5110_Clear();
  //Nokia5110_OutString("************* LCD Test *************Letter: Num:------- ---- ");


  //****************LCD levels**************************************************
  //***************************************************************************
  //***************************************************************************
  //I'll put this into it's own file later
  while(level <= 3) {

       //course 1
       if(level == 0) {
              Nokia5110_DrawFullImage(Course1); //print course1
       }

       //course 2
       else if(level == 1) {
              Nokia5110_DrawFullImage(Course2); //print course2
       }

       //course 3
       else if(level == 2) {
              Nokia5110_DrawFullImage(Course3); //print course3
       }

       //course 4
       else {
              Nokia5119_DrawFullImage(Course4); //print course4
       }

       //print ball

       //ball movement

              x = xVector; //some calculation for distance ball will travel in x
direction
              y = yVector; //some calculation for distance ball will travel in y
direction

       while(x != 0 || y != 0) {

              //move in x direction
              if(x > 0) {
                     //detect wall or hole

                     //hits wall
                     if(/*wall*/) {
                            //reverse direction
                     }

                     //hits hole
```

```
                   else if(/*hole*/) {
                          level++;
                          Nokia5110_Clear();
                          break;
                   }

                   //clear path
                   else {
                          //print ball in x direction
                          //clear where ball used to be
                          x--;
                   }
            }

            //move in y direction
            if(y > 0) {
                   //detect wall or hole

                   //hits wall
                   if(/*wall*/) {
                          //reverse direction
                   }

                   //hits hole
                   else if(/*hole*/) {
                          level++;
                          Nokia5110_Clear();
                          break;
                   }

                   //clear path
                   else {
                          //print ball in y direction
                          //clear where ball used to be
                          y--;
                   }
            }

     }


}

Nokia5110_Clear();
Nokia5110_DrawFullImage(YouWin);

//***********end of level code*****************************************************
//*********************************************************************************
//*********************************************************************************




     while(1)
     {
            GPIO_PORTG_DATA_R ^= GPIO_PIN_2;
            Delay(4000000);
     }
```

```
//  printf("Hello, world.");
//  printf("%c", NEWLINE);
//  Delay(4000000);           // delay ~1 sec at 12 MHz
//  Output_Color(8);
//  printf("A really long string should go to the next line.\r");
//  printf("Oxxx(::::::::::::::>%c", NEWLINE);
//  Delay(4000000);           // delay ~1 sec at 12 MHz
//  Output_Color(15);
//  printf("Color Table:%c", NEWLINE);
//  Delay(4000000);           // delay ~1 sec at 12 MHz
//  Output_Color(8);
//  printf("<::::::::::::::)xxxO%c", NEWLINE);
//  for(i=15; i>=1; i=i-2){
//    Delay(4000000);         // delay ~1 sec at 12 MHz
//    Output_Color(i);
//    printf("Color: %u%c", i, TAB);
//    Output_Color(i-1);
//    printf("Color: %u%c", i-1, NEWLINE);
//  }
//  Delay(4000000);           // delay ~1 sec at 12 MHz
//  Output_Clear();
}
```

MEASUREMENT DATA
Estimated Current: 10 mA