Can Satellite (CanSat)

Design Manual

University Space Engineering Consortium – Japan (UNISEC)
November, 2011. Ver. 1.0.

Can Satellite (CanSat)

Dedicated to

All the Japanese People

In a grateful appreciation

Acknowledgements

We wish to express our sincere appreciation to the following institutes and individuals who have assisted me in the development of this text book: Professor Shinichi NAKASUKA of the University of Tokyo, Ms. Rei Kawashima, first secretary general of UNISEC Japan, all the staff members of Institute for Education on Space (IfES) at Wakayama University and Ms. Emiko Ando, secretary of CLTP office, All participants of first CanSat Leadership Training Program (CLTP1), Faculty of Engineering, Cairo University. King Fahd University of Petroleum and Minerals (KFUPM).

Preface

The concept of Can-sat was proposed by professor Robert Twiggs of Stanford University in 1999. Can-Sat is Soda can size satellite. Can-Sat is fundamental concept in teaching space engineering to undergraduate students. The Can-Sat provides an affordable way to acquire the students with the basic knowledge to many challenges in building a satellite. In Can-Sat projects, students will be able to design, build and test a small electronic payload that can fit inside a Soda can. The Can-Sat is launched and ejected from a rocket or a balloon. By the use of a parachute, the Can-Sat slowly descends back to earth performing its mission while transmitting telemetry. Post launch and recovery data acquisition will allow the students to analyze the cause of success and/or failure.

The present textbook is written especially to spread the knowledge of CanSat-based space education in a very systematic and interesting way. Up to our knowledge it is the first textbook of its kind. The technical contains of the textbook were a collections of lectures series given during the First CanSat Leader Training Program (CLTP1) which help in Japan from February 14 to March 18, 2011 at Wakayama University. Also from the experience gain from different participants attended this course. The book covers most of the background required to design a baseline CanSat for data gathering during its launching and descent and send it a ground station PC.

Table of Contents

CHAPTER 1: INTRODUCTION

"We will make the world that space activity is not something special. If half of the countries on earth has ability to develop satellite/spacecraft then it will be natural for human activity to go out of gravity of the Earth"

Rei Kawashima,
First Secretary General of UNISEC

1.1 MOTIVATION

In November 1998 at the University Space Systems Symposium (USSS) held in Hawaii, Prof. Bob Twiggs, Figure 1-1 (Stanford University Space Development Laboratory) suggested the so-called "CanSat" concept, whereby the entire satellite would adopt the size of a 350-ml can. This challenge launched the establishment of international workshops and competitions. During the workshops and competitions throughout the past decade, college students have designed and built CanSats; launched them by rockets, balloons and/or model aircrafts; and collected data during the descent of the CanSats by parachute, simulating the development of experiments in space. During these hands-on activities, the students have been working on student-led satellite development projects, realizing low-cost and short-term development programs [1].

Figure 1-1 Professor Bob Twiggs

CanSat is a small satellite analog, as shown in Figure 1-2. All of the components, such as sensors, actuators, and GPS, are housed inside a 350-ml can. CanSat provides an

affordable opportunity for educators and students to acquire basic knowledge of space engineering and to experience engineering challenges in building a satellite. The student will be able to design and build a small electronic payload that can fit inside a standard drink can (350 ml). The CanSats will be launched by a rocket or balloon and released in the air. Using a parachute, the CanSat will slowly descend back to the ground as it performs its designated mission (i.e., taking pictures and transmitting telemetry). By analyzing the data recorded by the CanSat, participants will investigate the reasons of its success and/or failure.

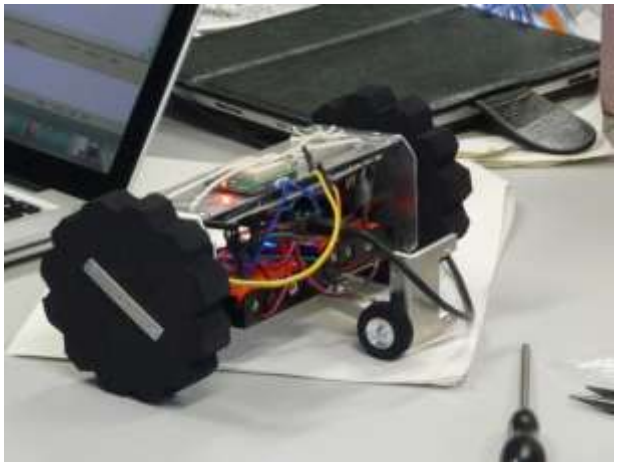


Figure 1-1 Rover-Type CanSat Developed by the CLTP participants in 2011 at Wakayama University.

The CanSat-based space engineering education challenges innovative students to get hands-on experience in a space related project during less than one year. As a space engineering project students will get experience from conceptual design, through integration and test, actual operation of the system. This will give the students experience of taking part of one whole project cycle within one year or less. One of the major advantages of the CanSat is the very low life cycle cost of the project. Thus, universities could involve more students to space related projects. The CanSat is small, non-orbiting and with limited complexity, but it is still like a "satellite" in terms of many of the challenges real satellites faces.

Each year an annual CanSat competition is arranged in USA. This annual competition allows teams from different universities and highschools to design and build a space-type system, according to the specifications released by the competition organizing committee, and then compete against each other at the end of two semesters to determine the winners. More information about the annual CanSat competition can be found in reference [1-2]. There are also several other CanSat competitions taking place in America, Europe and Asia. In USA there is also another CanSat competition called ARLISS, which has an international approach [1-3].

1.2 ORGANIZATION OF THE BOOK

The present textbook is written especially to spread the knowledge of CanSat-based space education in a very systematic and interesting way. Up to our knowledge it is the first textbook of its kind. The technical contains of the textbook were a collections of lectures series given during the First CanSat Leader Training Program (CLTP1) which help in Japan from February 14 to March 18, 2011 at Wakayama University. Also from the experience gain from different participants attended this course. The book covers most of the background required to design a baseline CanSat for data gathering during its launching and descent and send it a ground station PC. The organization of the textbook is as follow:

- Chapter 1 serves as introduction to the subject with the motivation for space engineering education.
- Chapter 2: summaries the basic mission type for CanSat
- Chapter 3 presents an introduction to the most important hardware components of the CanSat which is the microcontroller. The material is dedicated to certain type of microcontroller which is mbed NXP LPC1768 with its development online environment. This environment makes it easy to use such microcontroller and use other developers contributions from the mbed web site. Enormous example and illustrations are given in chapter 3 to acquire the students the required skills to write their own code.
- Chapter 4 introduces the basic concepts and protocols of the GPS (Global Positioning System). This sensor is quite new compared with the well know accelerometers, gyroscope, pressure and temperature.
- Chapter 5 deals with the interfaces of the mbed to different MEMS Transducers (MicroElectroMechanical Sensors) like the accelerometer, gyroscope, temperature, GPS, and pressure. Well test codes and developed class library for each of commercial transducers are presented in this chapter. At the end of the chapter, a program for system integration to read all the sensor and send then to a serially connected PC is developed.
- Chapter 6 introduces the basic concept of parachute analysis, simulation and design and a nonlinear model to predict the motion of real parachute during descending is developed and coded using MATLAB. Test example for a descending parachute from given altitude is simulated and the results were validated using lower order model.
- Chapter 7 presents the concept for the telemetry between the CanSat and Ground station PC. Commericall available RF comuunication modules were used.

Their interface with mbed and the needed firmware are developed. Ground station PC software was developed using MATLAB.

- Chapter 8 introduces the different technique and know-how for the mechanical construction of CanSat.
- Chapter 9 discusses some of the testing techniques for different MEMS sensors used before launching to ensure the reliability of the developed CanSat.
- Chapter 10 presents the technique to interpret and analyze the data acquired during CanSat mission.

CHAPTER 2: MISSION PROFILE AND SYSTEM DEVELOPMENT

2.1 INTRODUCTION

CanSat mission should address the objectives of the mission. Then the design and development should of the CanSat should produce CanSat that could achieve these objectives with high reliability and robustness. Basic mission like remote sensing (or imaging) or data acquisitions using different MEMS transducers usually performed for educational purposes and the beginner to the CanSat based-space engineering education.

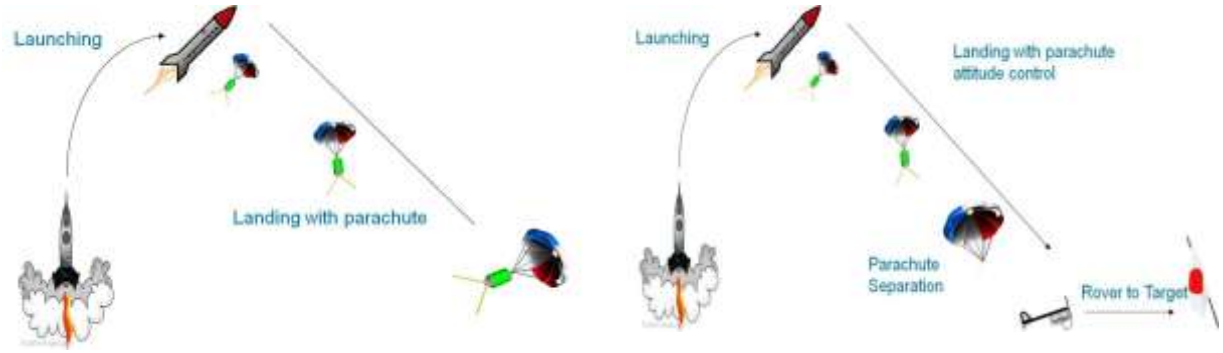
More advanced mission like launching a CanSat from a specific point and performa a fly back mission to return to another specific points usually done for advanced level training or CanSat competition.

2.2 MISSION TYPES

The mission Type can basically divide into two categories:

1. Data acquisition Mission (include imaging, sensors data acquisition)
2. Come-back or Fly back mission

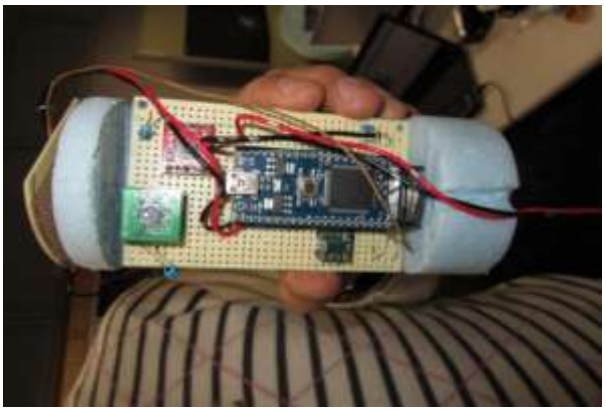
Figure 2-1 illustrates schematically the basic difference of between two missions



a) Data acquisition mission **b)** Fly back or come-back mission

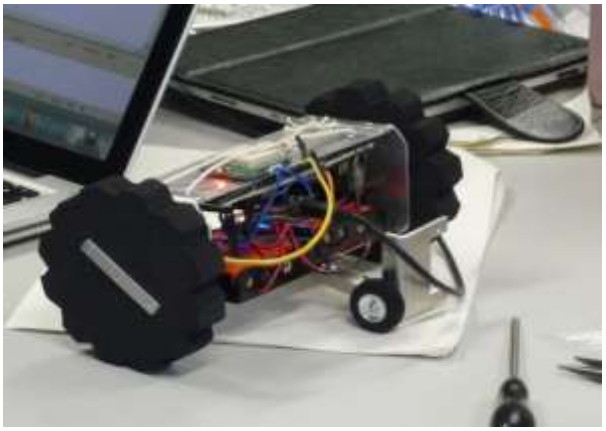
Figure 2-1 CanSat Mission types

Figure 2-2 shows three the different types of CanSat which was developed during the CLTP1.



a) Data Acquisition CanSat

b) Fly-back CanSat



c) Come-back CanSat

Figure 2-2 Photos show the different types of CanSat.

2.3 GENERALIZED HARDWARE ARCHITECTURE FOR THE CANSAT

Figure 2-3 shows the generalize hardware architecture for any type of CanSat. The microcontroller is the heart of the CanSat and should interface with the transducers and actuators as well as sending the data and receive commands from the ground station PC. During the data acquisition mission CanSat only read data from sensors, store and send it to the ground station PC. However, in come-back and fly-back mission the data should be process on-board the CanSat and decision should taken to deploy actuators for controls. The present textbook focuses on the basic type of CanSat which is data acquisition CanSat.

CHAPTER 3: INTRODUCTION TO MBED

3.1 INTRODUCTION

Microcontroller called Mbed was chosen as the main on-board computer of the Can-Sat which is known as an effective tool for rapid prototyping [3-1], shown in Figure 3-1. Mbed has its developing environment online and no special software installation is needed. There are numerous C++ libraries and classes available online for most of the COTS MEMS components. Appendix H summaries the most important C/C++ basic data types, operators and control statements

Figure 3-1 mbed Microcontroller

The package of the microcontroller shown in Figure 3-2 and contains an mbed Microcontroller and USB lead. The following requirement is needed to start working with the mbed.

- ·A computer running Windows, MacOS X or GNU/Linux with
  - a USB port,
  - an Internet connection, and
  - an active email address
- A web browser - Internet Explorer, Firefox, Chrome or Safari

Figure 3-2 mbed package

Figure 3-3 shows the pin layout of mbed.

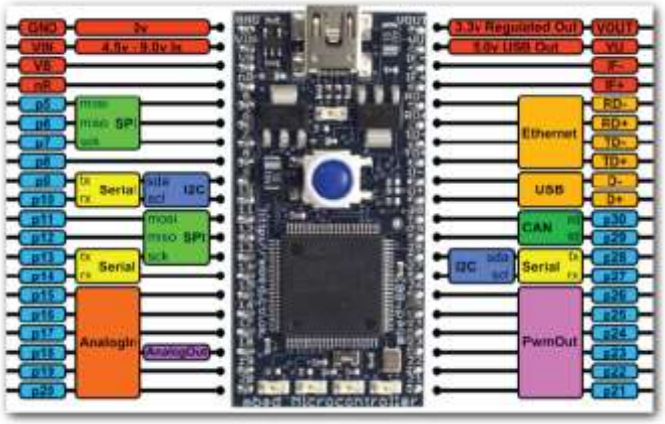


Figure 3-3 I/O pin layout and supported interfaces

The technical specifications of mbed NXP LPC1768 are

- The mbed NXP LPC1768 is a 32-bit ARM Cortex-M3 Core running at 96MHz, 512KB FLASH, 64KB RAM with a comprehensive set of peripherals and a built-in USB interface
- Designed specifically to make ARM microcontrollers easily accessible for rapid prototyping and experimentation
    - 0.1" DIP pins so you can use it in breadboard, stripboard and through-hole PCBs
    - Powered via USB or an external power source
    - Re-program it like a FLASH drive; it appears as a USB disk, and you just drag on your .bin files
    - Has on-board LEDs, and provides a serial port over the USB connection to allow for printf-style debugging
    - Lots of interfaces including Ethernet, USB Device and Host, CAN, SPI, I2C and other I/O
    -

More technical information about mbed NXP LPC1768 are presented in Appendix A. Figures 3-4 and 3-5 show how you can sign up your account after plunging the mbed to your PC.

Figure 3-4 Setting mbed

The setup instructions are as follows

- Connect your mbed Microcontroller to a PC
- Click the MBED.HTM link to get logged in
    - Click MBED.HTM to open it in your web browser
    - Choose "Signup", and create your mbed account - This will give you access to the Website, Tools, Libraries and Documentation, as shown in Figure 3-6.

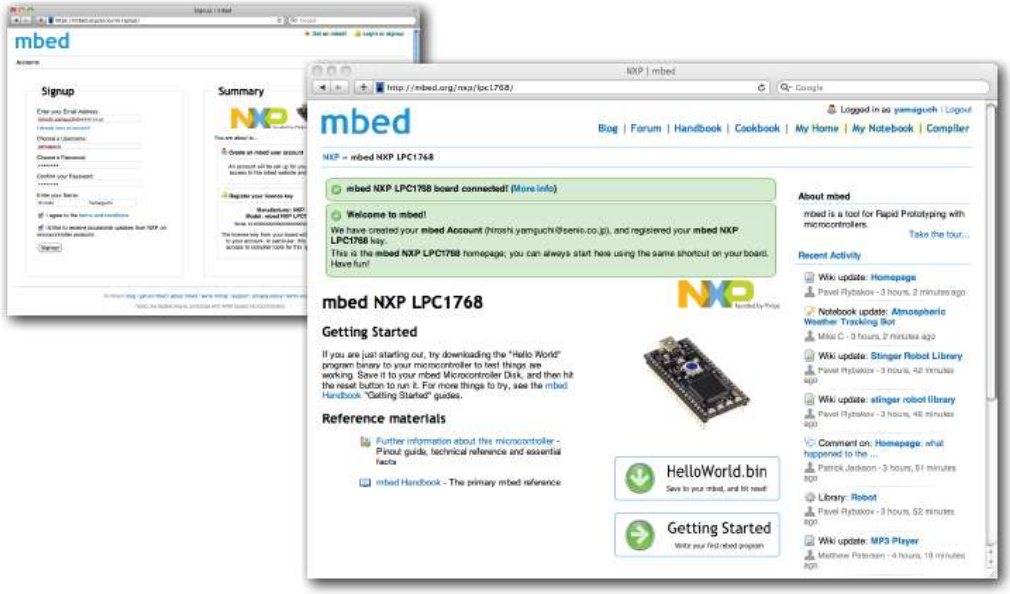
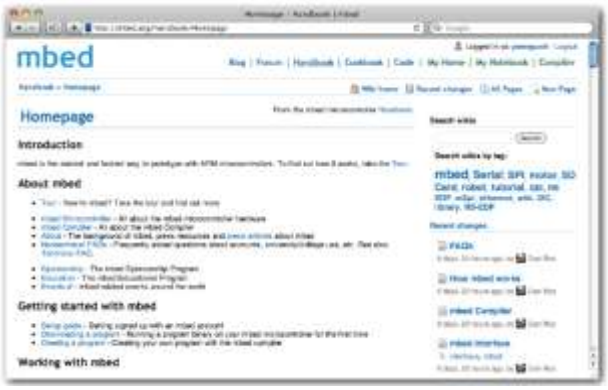


Figure 3-5 Creating online mbed account

a) mbed web site



b) Handbook

c) Forum

d) cookbook

e)

Figure 3-6 features of mbed web site

To downloading a simple “Hello World” program from the www.mbed.org web site to the mbed to the following:

- Go to the web page
  - http://mbed.org/handbook/Downloading-a-program
  - Click HelloWorld_LPC1768.bin to download the program binary
  - Save the binary to your mbed Microcontroller flash disk
- Press the reset button, as shown in Figure 3-7.
  - The newest program will be loaded and start running

Figure 3-7 mbed reset button and LEDs

To change the browser setting in IE8 to make the default download drive is med drive using the regedit application in windows, steps illustrated in Figures 3-7 to 3-13 should be done.

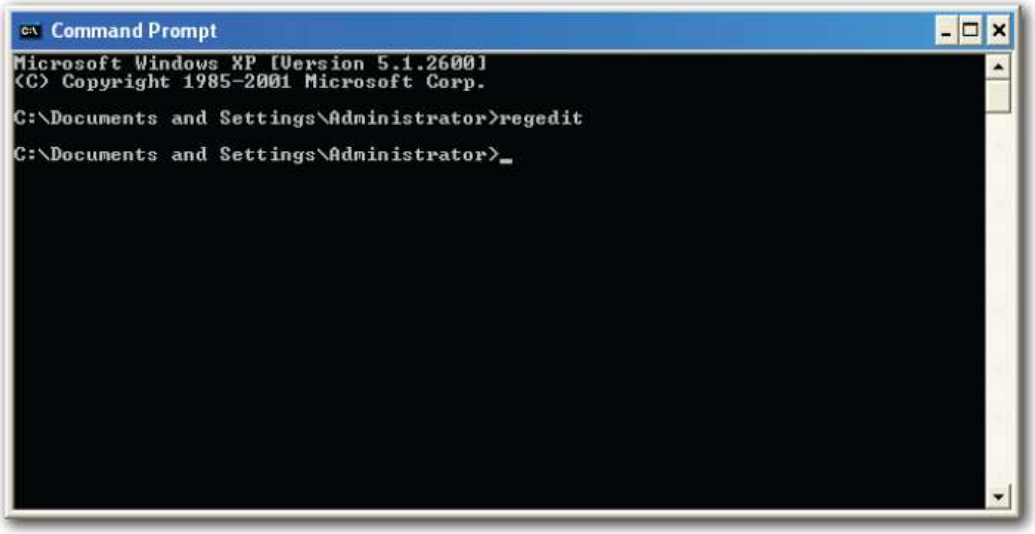


Figure 3-8 Accessing the Windows registry editor

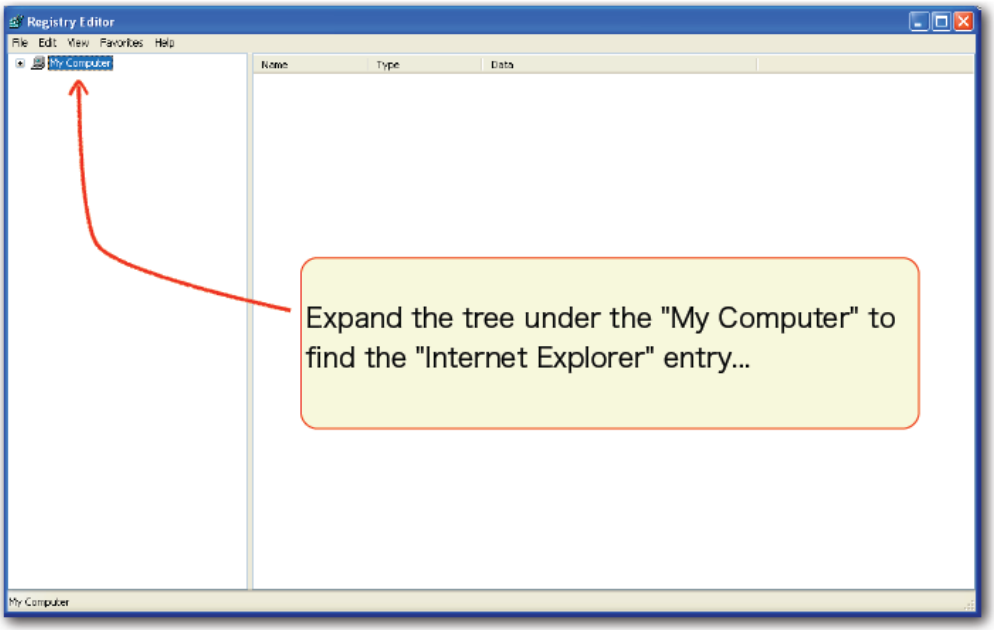


Figure 3-9 Windows registry editor

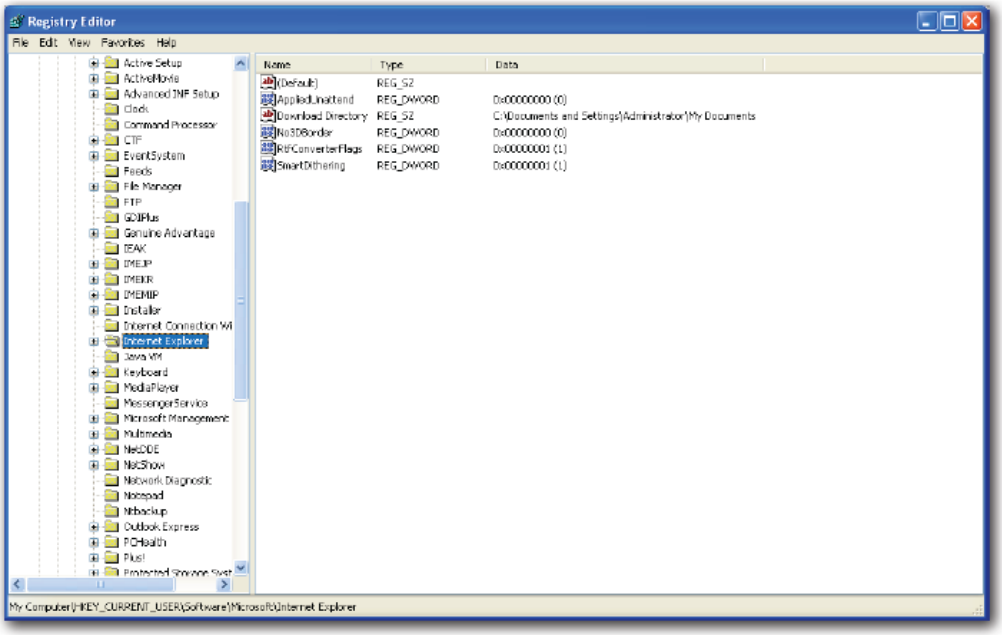


Figure 3-10 Access “Windows Internet Explorer” node

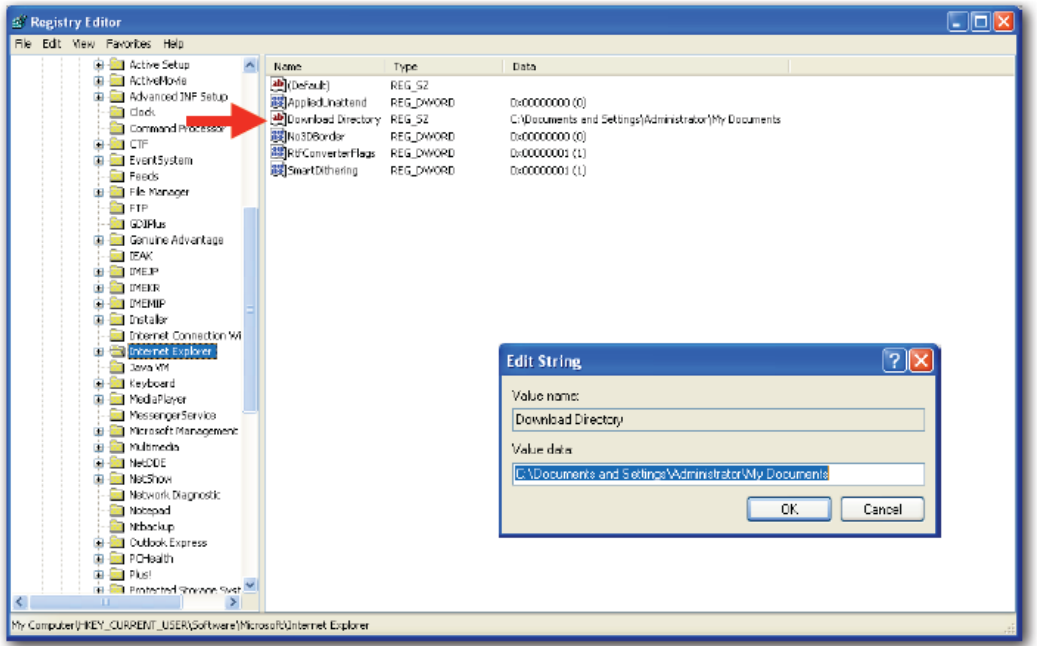


Figure 3-11 Changing the download directory

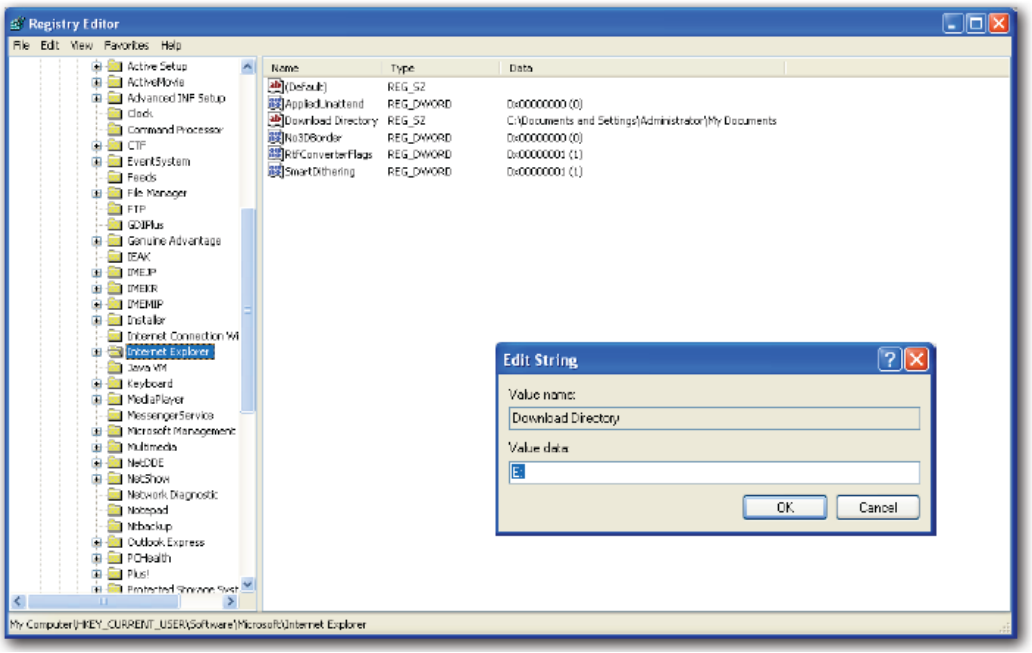


Figure 3-12 Changing the download directory

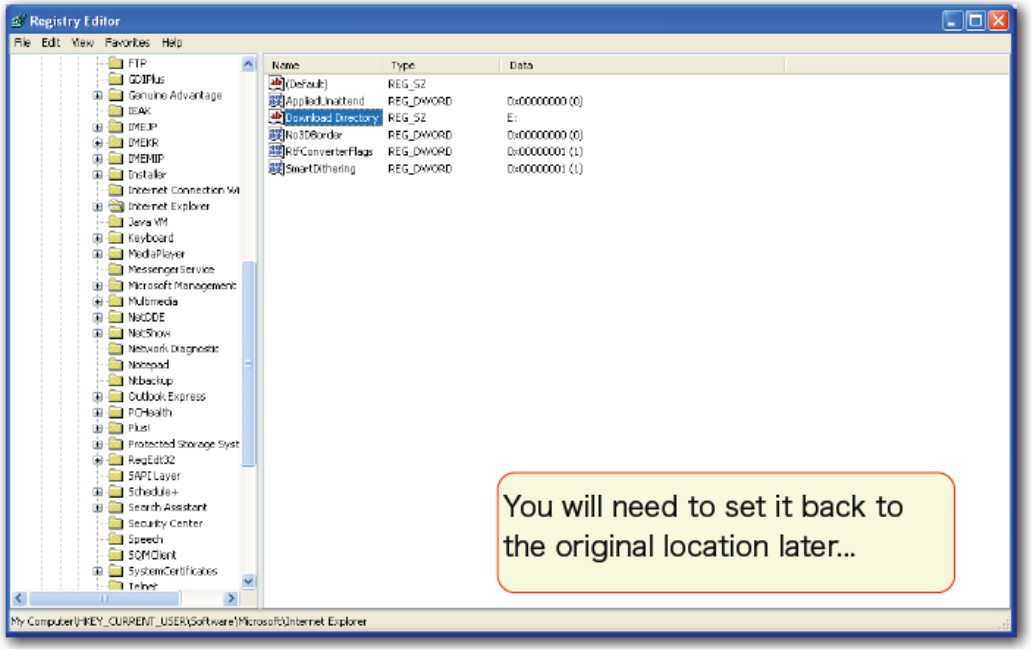


Figure 3-13 Changing the download directory

3.2 CREATE YOUR FIRST PROGRAM

To create your first program to the following steps:

- Open the mbed Compiler. Figure 3-14
- Create a new program
  - Right click "My Programs" to select "New Program..."
  - Enter name of the program, then click "OK"
  - New program will be created under "My Programs"
- View the default source code, Shown in Figure 3-15.
- Compile and download the program
  - Click "Compile" button in the tool bar
  - After a successful compilation, you will get a "Success" message
  - If there were errors, they will show up in the "Compiler Output" window
- Press the "reset" button

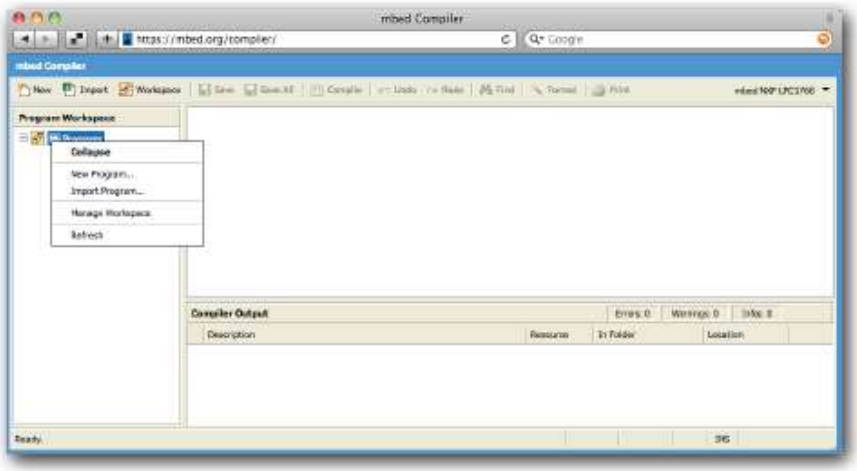


Figure 3-14 Creating your first program

```
#include "mbed.h"

DigitalOut myled(LED1);

int main() {
    while(1) {
        myled = 1;
        wait(0.2);
        myled = 0;
        wait(0.2);
    }
}
```

Figure 3-15 write and compile your program

3.3 PLAYING WITH THE BUILT-IN LEDS

The Built-in LEDs are shown in Figure 3-16 which are:

- LED1 to LED4
  - can be turn on and off from a program
- Status LED
  - turns on when powered
  - indicates the transmission status

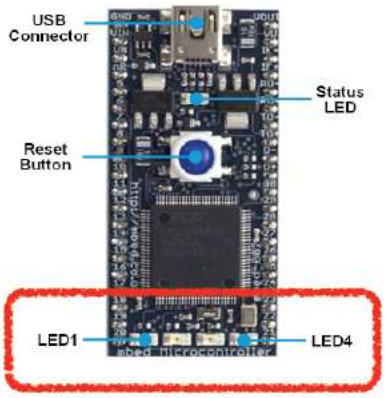


Figure 3-16 Built-in LEDs

The mbed default program is as follows:

```
#include "mbed.h"
DigitalOut myled(LED1);

int main() {
      while(1) {
      myled = 1;
      wait(0.2);
      myled = 0;
      wait(0.2);
      }
}
```

After compiling, reseat button must be pressed to upload the program to the microcontroller and the LED1 will blink. The following code is for blinking two LEDs

```
#include "mbed.h"
DigitalOut myled(LED1);
DigitalOut myled2(LED2);

int main() {
      while(1) {
      myled = 1;
      myled2 = 1;
      wait(0.2);
      myled = 0;
      myled2 = 0;
      wait(0.2);
      }
}
```

Using array can reduced the program length as in the following code

```
#include "mbed.h"
DigitalOut myleds[] = {LED1, LED2, LED3, LED4};

int main() {
     while(1) {
          myleds[0] = 1;
          wait(0.2);
          myleds[0] = 0;
          wait(0.2);
     }
}
```

The following code lights all LEDS

```
#include "mbed.h"
DigitalOut myleds[] = {LED1, LED2, LED3, LED4};

int main() {
     while(1) {
          myleds[0] = 1;
          myleds[1] = 1;
          myleds[2] = 1;
          myleds[3] = 1;
          wait(0.2);
          myleds[0] = 0;
          myleds[1] = 0;
          myleds[2] = 0;
          myleds[3] = 0;
          wait(0.2);
     }
}
```

Use a "for" loop

```
#include "mbed.h"
DigitalOut myleds[] = {LED1, LED2, LED3, LED4};

int main() {
     while(1) {
     for (int i = 0; i < 4; i++) {
     myleds[i] = 1;
     }
     wait(0.2);
     for (int i = 1; i < 4; i++) {
```

```
        myleds[i] = 0;
        }
        wait(0.2);
        }
}
```

Use a different "for" loop

```
#include "mbed.h"
DigitalOut myleds[] = {LED1, LED2, LED3, LED4};

int main() {
        for (int i = 0;; i = (i + 1) % 4) {
        myleds[i] = 1;
        wait(0.2);
        myleds[i] = 0;
        wait(0.2);
        }
}
```

To write a function, the following code give an example

```
#include "mbed.h"
DigitalOut myleds[] = {LED1, LED2, LED3, LED4};

void setMyLeds(bool arg) {
        for (int i = 0; i < 4; i++) {
        myleds[i] = arg;
        }
}

int main() {
        while(1) {
        setMyLeds(1);
        wait(0.2);
        setMyLeds(0);
        wait(0.2);
        }
}
```

The function argument, body and data types is described in Figure 3-17.

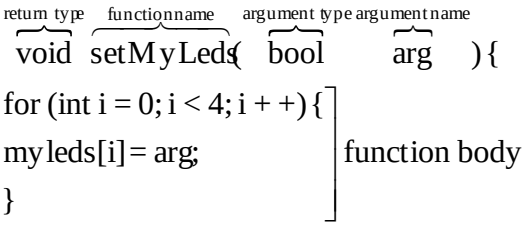


Figure 3-17 defining a function

To write another function is illustrated in the following code

```
#include "mbed.h"
DigitalOut myleds[] = {LED1, LED2, LED3, LED4};

void setMyLeds(bool arg0, bool arg1, bool arg2, bool arg3)
{
     myleds[0] = arg0;
     myleds[1] = arg1;
     myleds[2] = arg2;
     myleds[3] = arg3;
}
int main() {
     while(1) {
     setMyLeds(1, 1, 1, 1);
     wait(0.2);
     setMyLeds(0, 0, 0, 0);
     wait(0.2);
     }
}
```

Example 3-1: Count-down in binary

- Write a program to count down from 15 to 0
    - when the count reaches 0, flash all the LEDs, as shown in the following Figure 3-18.
- 0 to 15 in binary

decimal	binary	decimal	binary
0	● ● ● ●	8	○ ● ● ●
1	● ● ● ○	9	○ ● ● ○
2	● ● ○ ●	10	○ ● ○ ●
3	● ● ○ ○	11	○ ● ○ ○
4	● ○ ● ●	12	○ ○ ● ●
5	● ○ ● ○	13	○ ○ ● ○
6	● ○ ○ ●	14	○ ○ ○ ●
7	● ○ ○ ○	15	○ ○ ○ ○

Figure 3-18 Example 3-1

Solution:

```
#include "mbed.h"
DigitalOut myleds[] = {LED1, LED2, LED3, LED4};

void setMyLeds(bool arg0, bool arg1, bool arg2, bool arg3) {
     myleds[0] = arg0;
     myleds[1] = arg1;
     myleds[2] = arg2;
     myleds[3] = arg3;
}
int main() {
     for (int i = 15; i > 0; i--) {
          setMyLeds(i & 8, i & 4, i & 2, i & 1);
          wait(1.0);
     }
     for (bool on = false;; on = !on) {
          setMyLeds(on, on, on, on);
          wait(0.2);
     }
}
```

3.4 USING A TACT SWITCH

A tact switch, shown in Figure 3-19, is type of switch that is only on when the button is pressed. As soon as you release the button, the circuit is broken. It can also can be described as a type of switch which is only on when the button is pressed

- Legs 1 and 2 become connected when the button pressed

- Legs 2 and 3 (1 and 4) are always connected

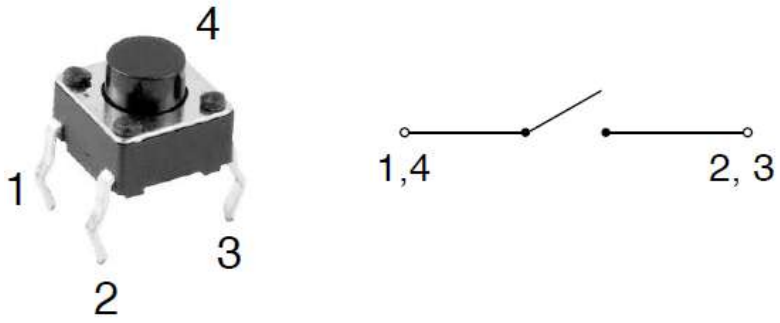


Figure 3-19 Tact or Tactile switch

Figure 3-20 illustrate how to connect the tact switch to mbed

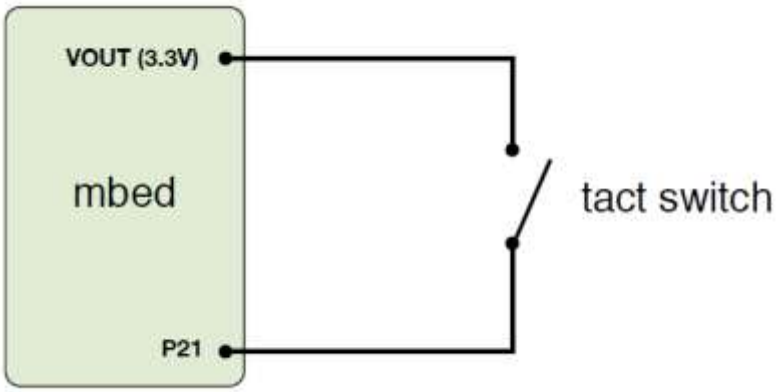


Figure 3-20 Connecting the tact switch

Using the breadboard illustrated in Figure 3-21. Connection of both the tact switch and mbed can be realized quickly, as shown in Figure 3-22. A breadboard (protoboard) is a construction base for prototyping of electronics. The term is commonly used to refer to solderless readboard (plugboard). Because the solderless breadboard does not require soldering, it is reusable. This makes it easy to use for creating temporary prototypes and experimenting with circuit design.

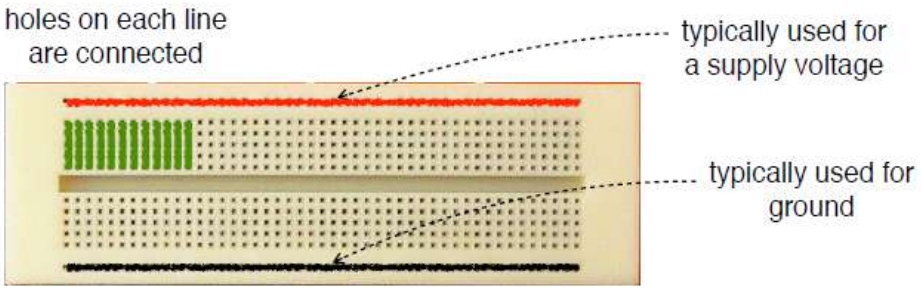


Figure 3-21 breadboard

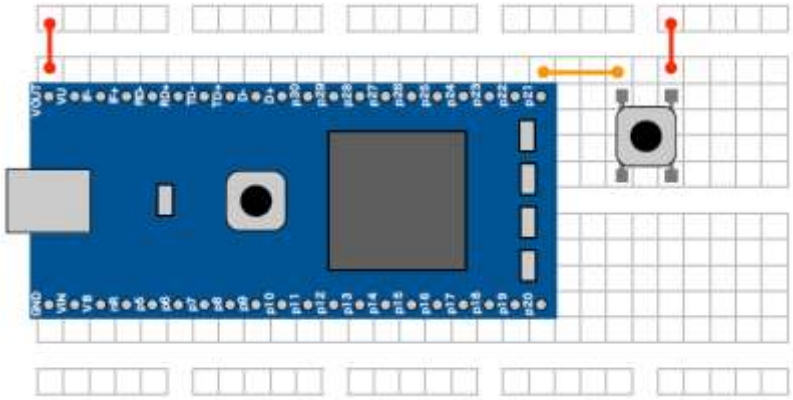


Figure 3-22 mbed and tact switch connected using the breadboard

Reading a switch input

- When the tact switch is pressed, the voltage of the p21 pin becomes the same voltage as the vout pin; when released, the voltage goes down to the ground level.
- Write a program to turn on the LED1 while the tact switch is pressed, as shown in Figure 3-23.

```
#include "mbed.h"
DigitalOut myled(LED1);
DigitalIn switchPressed(p21);

int main() {
     while (true) {
     myled = switchPressed;
     }
}
```

- A digital input, used for reading the state of a pin

- DigitalIn(PinName) creates a DigitalIn object, connected to the specified pin
- read() reads the input voltage, represented as 0 or 1 (int)
- read() is overloaded to the operator int()

```
myLed = switchPressed;

    is a shorthand of

myLed = switchPressed.read();
```

Figure 3-23 Digital input object

Pressing switch to reverse LED state is illustrated in the following code

```
#include "mbed.h"
DigitalOut myled(LED1);
DigitalIn switchPressed(p21);

int main() {
bool stateSW = false;
bool stateLED = false;
     while (true) {
          if (switchPressed != stateSW) {
               stateSW = !stateSW;
               if (stateSW) {
                    stateLED = !stateLED;
                    myled = stateLED;
               }
          }
     }
}
```

- A digital input, used to call a function on a raising or falling edge
- InterruptIn(PinName) creates an InterruptIn, connected to the specified pin
- rise(func) attaches a function to call when a rising edge occurs on the input
- fall(func) attaches a function to call when a falling edge occurs on the input

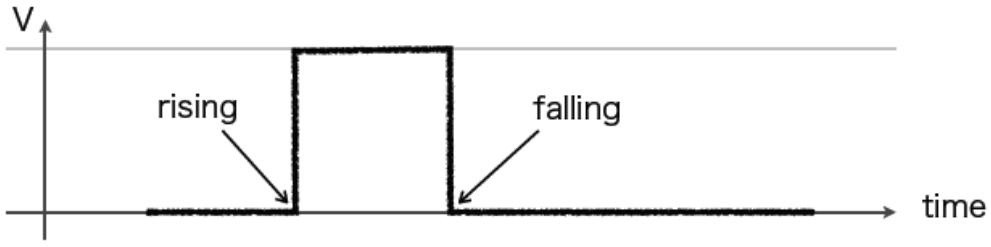


Figure 3-24 Digital input signal

Reading a switch through interrupt is illustaed in the following code

```
#include "mbed.h"
DigitalOut myled(LED2);
InterruptIn switchEvent(p21);

void switchPressed() {
     static bool stateLED;
     stateLED = !stateLED;
     myled = stateLED;
}

int main() {
switchEvent.rise(switchPressed);
     while (true) {
     wait(60);
     }
}
```

Reading a switch through interrupt is illustrated in the following code

```
#include "mbed.h"
DigitalOut myleds[] = {LED1, LED2, LED3, LED4};

InterruptIn switchEvent(p21);
     void switchReleased() {
     static int n = 0;
     myleds[(n + 3) % 4] = 0;
     myleds[n % 4] = 1;
     n++;
}

int main() {
switchEvent.fall(switchReleased);
     while (true) {
          wait(60);
     }
}
```

Contact bounce (also called *chatter*) is a common problem with mechanical switches and relays. Switch and relay contacts are usually made of springy metals that are forced into contact by an actuator. When the contacts strike together, their momentum and elasticity act together to cause bounce. The result is a rapidly pulsed electric current instead of a clean transition from zero to full current. The effect is usually unimportant in power circuits, but causes problems in some analogue and logic circuits that respond fast enough to misinterpret the on-off pulses as a data stream. The effects of contact bounce can be eliminated by modify the above code to be as follows

```
#include "mbed.h"
DigitalOut myleds[] = {LED1, LED2, LED3, LED4};
InterruptIn switchEvent(p21);
volatile bool interruptEnabled;

void switchReleased() {
     static int n = 0;
     if (interruptEnabled) {
          myleds[(n + 3) % 4] = 0;
          myleds[n % 4] = 1;
          n++;
          interruptEnabled = false;
     }
}
int main() {
     switchEvent.fall(switchReleased);
     while (true) {
          wait(0.1);
          interruptEnabled = true;
     }
}
```

3.5 COMMUNICATION WITH YOUR PC

To set a serial communication to the mbed and PC access mbed web site and download the mbed windows serial port driver as illustrated in Figure 3-25.

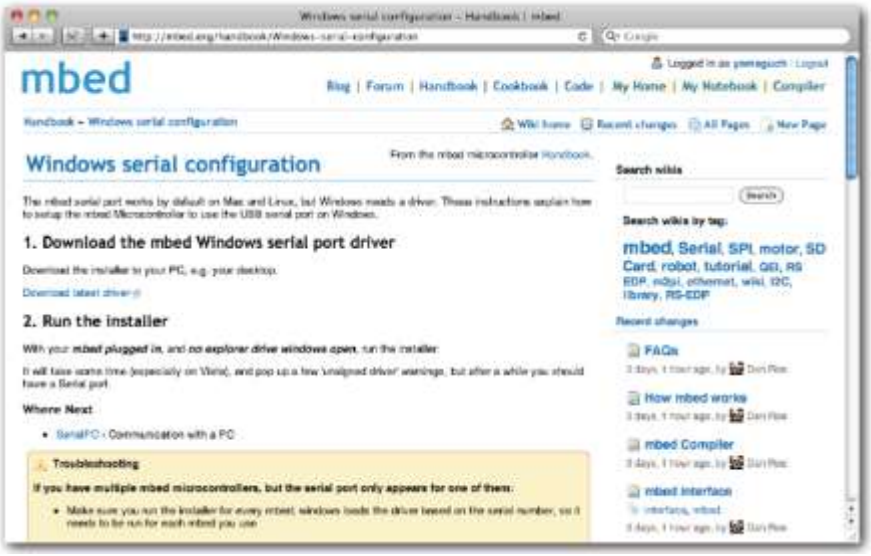


Figure 3-25 Setting up the windows serial communication

NOTE: If the installer reports the message ***"mbedWinSerial_nnnnn.exe is not a valid Win32 application".*** It is likely you are using Internet Explorer to download the installer

file, which sometimes seems to only download part of the installer application for an unknown reason. To solve this, download the installer with a different browser (Firefox, Chrome), and try again; this should solve the issue.

There is several applications for HyperTerminal. TeraTerm will be used in the following sections as shown in Figure 3-26 and Figure 3-27.

Terminal applications for different OS

- MacOS X
  - CoolTerm (download and install required)
  - screen command (available by default)
- Windows
  - CoolTerm (download and install required)
  - TeraTerm (download and install required)
- Linux
  - screen command (available by default)

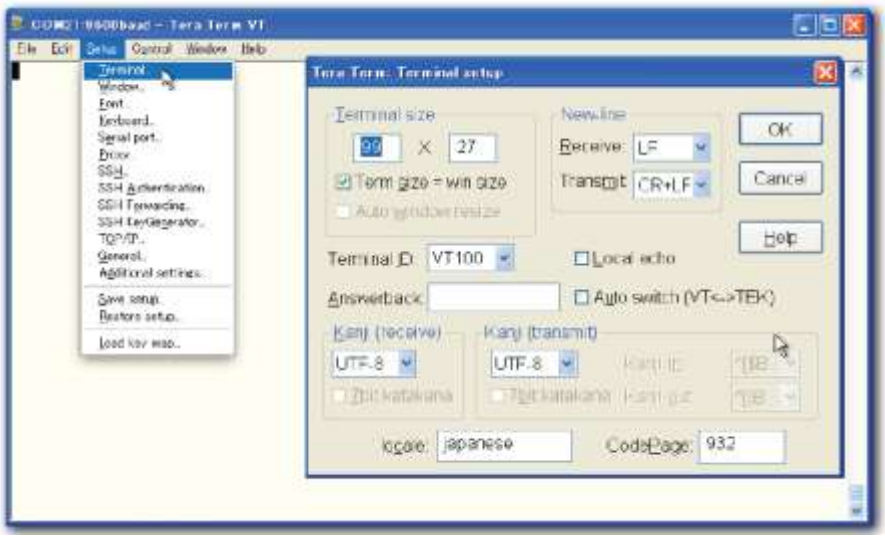


Figure 3-26 Setting up the Tera Term V1

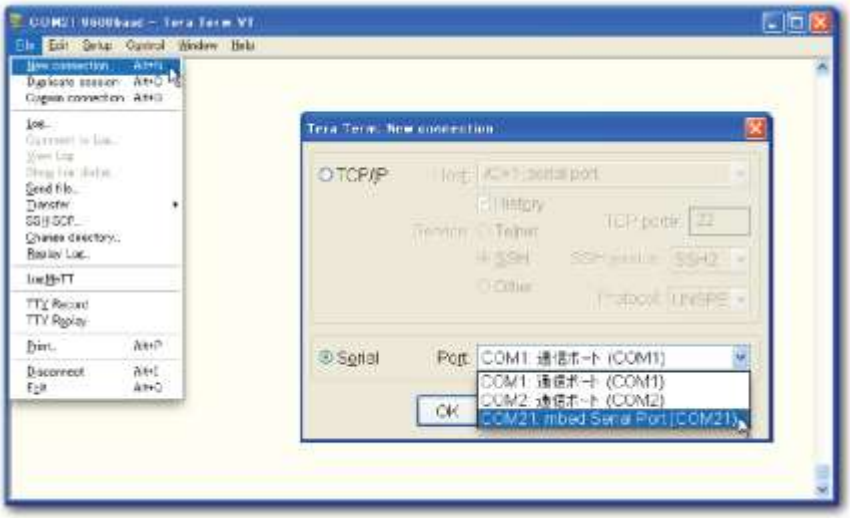


Figure 3-27 Setting up the Tera Term V1

Serial communication with a PC:

- The mbed Microcontroller can communicate with a host PC through a "USB Virtual Serial Port" over the same USB cable that is used for programming
- This enables you to:
    - Print out messages to a host PC terminal
    - Read input from the host PC keyboard
    - Communicate with applications running on the host PC

To run a "Hello World" program do the following steps:

- Create a HelloWorld program
- Connect your mbed to the PC with USB cable
- Compile and download it to your mbed
- Open the terminal application, Figure 3-28.
- Press the reset button

```
#include "mbed.h"
int main() {
     printf("Hello World!\n");
}
```

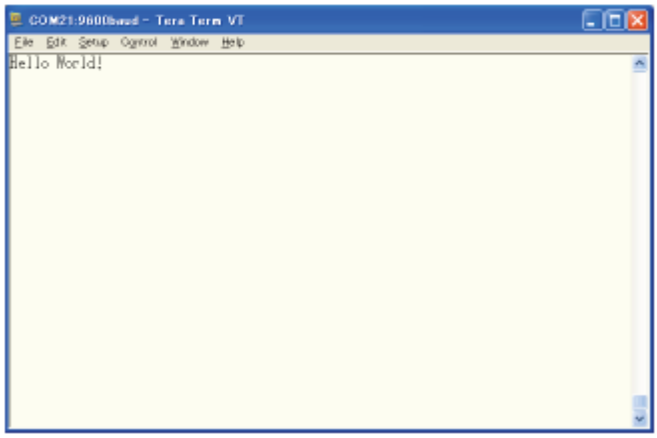


Figure 3-28 Output of the Hello World code in the HyperTerminal

To send character data to mbed do the following steps:

- Create an Echo program
- Connect your mbed to the PC with USB cable
- Compile and download it to your mbed
- Open the terminal application
- Press the reset button
- A message from the mbed appears
- Type any characters from your PC, Figure 3-29.

```
#include "mbed.h"
int main() {
     printf("Echoes back to the screen anything you type\n");
     while (true) {
          putchar(getchar());
     }
}
```

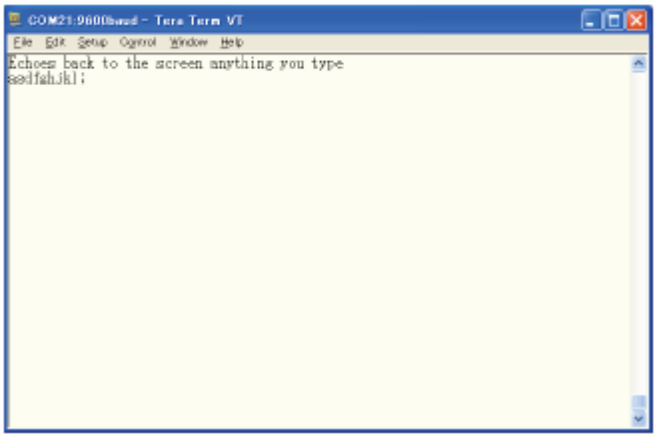


Figure 3-29 Output of the Echoes back code

3.6 ANALOG INPUT (CONNECTION WITH AN ACCELEROMETER)

In this section connection with accelerometer will be described. Specifications of the accelerometer employed here are presented in Table 3-1 and in Appendix B as well as Figure 3-30 and Figure 3-31.

Table 3-1 Freescale: 3-Axis Low-g Accelerometer

3mm x 5mm x 1.0mm LGA-14 Package
Low current consumption: 400 µA
Sleep mode: 3 µA
Low voltage operation: 2.2 V – 3.6 V
High sensitivity (800 mV/g @ 1.5g)
Selectable sensitivity (±1.5g, ±6g)
Fast turn-on time (0.5 ms Enable Response Time)
0g-detect for freefall protection
Robust design, high shocks survivability

Accelerometers are sensors and instruments for measuring, displaying and analysing acceleration and vibration. They can be used on a stand-alone basis, or in conjunction with a data acquisition system. Accelerometers are available in many forms. They can be raw sensing elements, packaged transducers, or a sensor system or instrument, incorporating features such as totalizing, local or remote display and data recording. Accelerometers can have from one to three axes of measurement, the multiple axes typically being orthogonal to each other. These devices work on many operating principles. The most common types of accelerometers are piezoelectric, capacitance, null-balance, strain gauge, resonance, piezoresistive and magnetic induction [3-2].

There are several physical processes that can be used to develop a sensor to measure acceleration. In applications that involve flight, such as aircraft and satellites, accelerometers are based on properties of rotating masses. In the industrial world, however, the most common design is based on a combination of Newton's law of mass acceleration and Hooke's law of spring action.

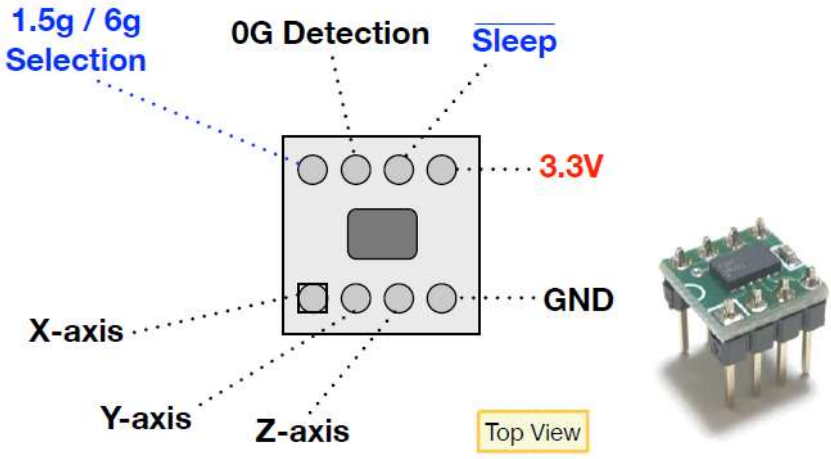
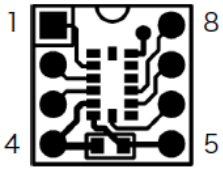


Figure 3-30 I/O of the Freescale: 3-Axis Low-g Accelerometer

Pin #	Symbol	Description
1	Xout	1.65V, when zero-g
2	Yout	1.65V, when zero-g
3	Zout	1.65V, when zero-g
4	Vss	GND
5	Vdd	Supply voltage (3.3V)
6	~Sleep	Sleep when LOW
7	0g-Detect	HIGH if 0g detected
8	g-Select	6g if HIGH, 1.5g otherwise



mode	sensitivity (mV/g)
1.5g	800
6g	206

Figure 3-31 Pin description

Figure 3-32 and Figure 3-33 show the connection of the accelerometer with mbed.

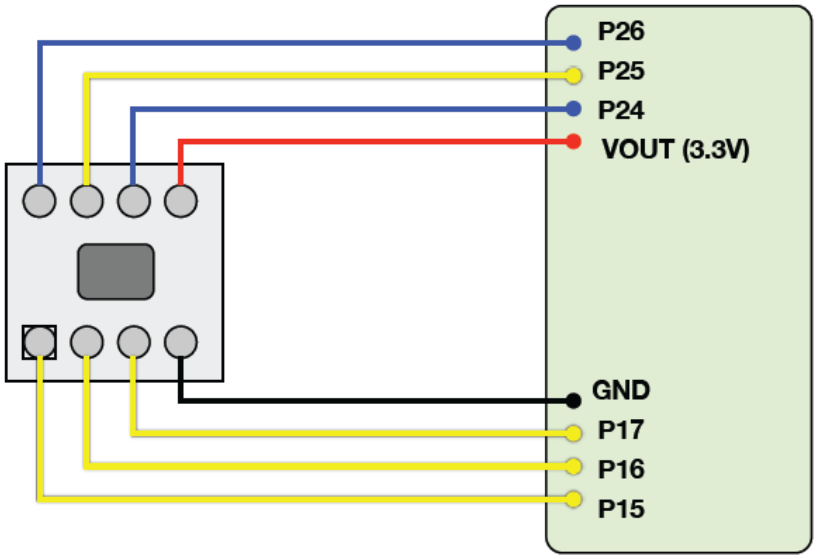


Figure 3-32 Schematic connection of accelerometer to mbed

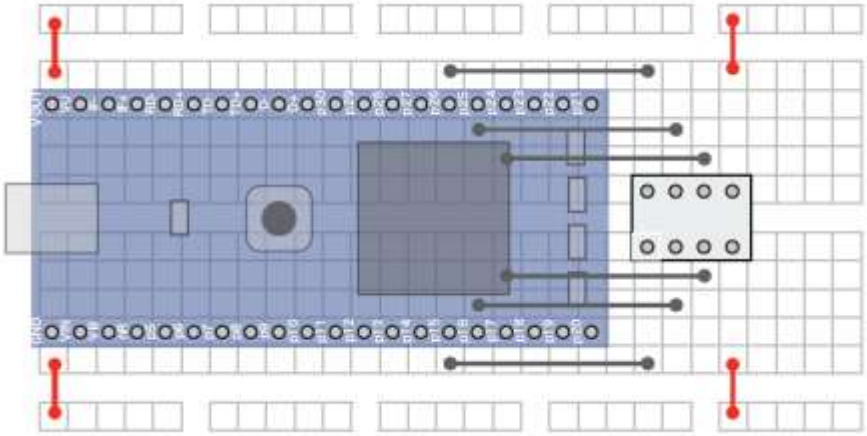


Figure 3-33 Connection of accelerometer to mbed on the breadboard

Write a program to read raw values

- Set the accelerometer to 6g mode
- Set the sleep mode OFF
- Read the raw value (normalized to [0, 1]) of xout
- Read the raw value (normalized to [0, 1]) of yout
- Read the raw value (normalized to [0, 1]) of zout
- Wait for 1 second
- Repeat them forever!

The code can be written as follows:

```
#include "mbed.h"
AnalogIn x(p15);
AnalogIn y(p16);
AnalogIn z(p17);
DigitalIn zeroGDetect(p25);
DigitalOut gSelect(p26);
DigitalOut sleep(p24);

int main() {
     gSelect = 1; // 6g mode
     sleep = 1; // do not sleep
     while (true) {
          printf("x = %5.3f, y = %5.3f, z = %5.3f\n", x.read(),
     y.read(), z.read());
          wait(1.0);
     }
}
```

AnalogIn can be described as follows:

- Analog input, used for reading the voltage on a pin
- AnalogIn(PinName) creates an AnalogIn, connected to the specified pin
- read() reads the input voltage, represented a float in the range between 0.0 and 1.0
- read() is overloaded to the operator float()

printf function formatting can be described as follows:

- Writes a format string
  printf(*format*, *arg1*, *arg2*, ...);
- Arguments
  - *format* : a template string that can contain embedded format tags, see Table 3-2.
  - *arg1*: the first variable substituted to the corresponding tag in the format string
  - *arg2*: the second variable substituted to the corresponding tag in the format string
  - *arg1, arg2*, ... are optional

Table 3-2 Printf embedded format tag examples

tag	variable type	converted to
`%d`	`int, short, long`	decimal string
`%f`	`float, double`	decimal floating point string
`%s`	`char` array terminated by a null character (`'\0'`)	string

Tags optionally contain `width.precision` after % character such as %5.3f, which converts a float to 5 digit floating decimal string whose fractional part contains 3 digits.

Compile, download and run it

- Compile the program
- If successfully compiled, it will automatically be downloaded to your mbed
- Open your terminal application.
- Press the reset button!

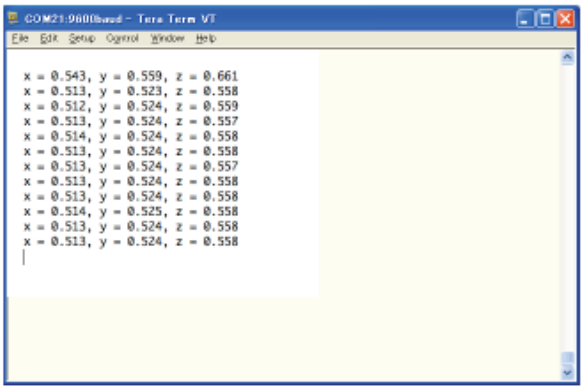


Figure 3-34 Accelerometer output to the HyperTerminal

The following program converts the readings to *g* values

```
#include "mbed.h"
AnalogIn x(p15);
AnalogIn y(p16);
AnalogIn z(p17);
DigitalIn zeroGDetect(p25);
DigitalOut gSelect(p26);
DigitalOut sleep(p24);
```

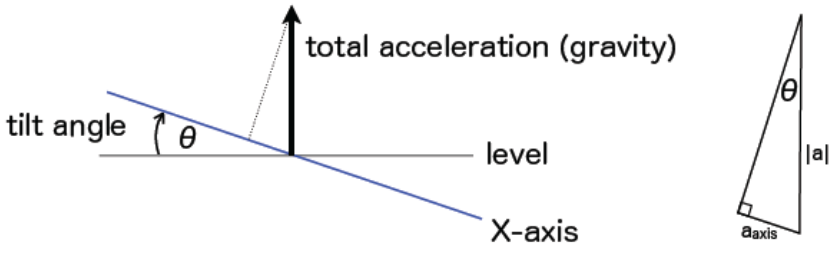
```
float toG(float value) {
     return ((value * 3.3) - 1.65) / 0.206;
}


int main() {
     gSelect = 1; // 6g mode
     sleep = 1; // do not sleep
          while (true) {
               float accelX = toG(x.read());
               float accelY = toG(y.read());
               float accelZ = toG(z.read());
               printf("x = %5.3f, y = %5.3f, z = %5.3f\n",
          accelX, accelY, accelZ);
               wait(1.0);
          }
}
```

Defining a function

float (return type) toG (function name) (float (argument type) value (argument name))

Computing the tilt (inclination) as shown in Figure 3-35.

- tilt angle θ= arcsin(a_{axis} / |a|)
  - |a| = total acceleration
  - a_{axis} = acceleration measured along the give axis




```
float accel = sqrt(accelX * accelX + accelY * accelY + accelZ * accelZ);
float tiltX = asin(accelX/accel);
```

Figure 3-35 Compute the inclination using accelerometer reading

The code can be written as follows:

```
#include "mbed.h"
AnalogIn x(p15);
AnalogIn y(p16);
AnalogIn z(p17);
DigitalIn zeroGDetect(p25);
DigitalOut gSelect(p26);
DigitalOut sleep(p24);
float toG(float value) {
     return ((value * 3.3) - 1.65) / 0.206;
}

int main() {
     gSelect = 1; // 6g mode
     sleep = 1; // do not sleep
          while (true) {
               float accelX = toG(x.read());
               float accelY = toG(y.read());
               float accelZ = toG(z.read());
               float accel = sqrt(accelX * accelX + accelY *
          accelY + accelZ * accelZ);
               float tiltX = asin(accelX/accel);
               float tiltY = asin(accelY/accel);
               float tiltZ = asin(accelZ/accel);
               printf("accelX = %5.3f, accelY = %5.3f, accelZ
          = %5.3f\n", accelX, accelY, accelZ);
               printf("tiltX = %5.3f, tiltY = %5.3f, tiltZ
          = %5.3f\n", tiltX, tiltY, tiltZ);
               wait(1.0);
     }
}
```

The commonly used mathematical function is listed in Table 3-3

Table 3-3 Commonly used mathematical functions are available

name	function
cos	cosine
sin	sine
tan	tangent
acos	arc cosine
asin	arc sine
atan	arc tangent
exp	exponential
log	natural logarithm
log10	common logarithm
pow	raise to power
sqrt	square root
ceil	round up value
floor	round down value
abs	absolute value

Zero G Detection

- ZeroG pin becomes HIGH, when all 3 axes are at 0g
  - Reading ZeroG pin is much faster than reading all 3 axes
  - Can use an interrupt on a "Zero G event" occurrence
- Enabling the application of free-fall protection

Reading 0G-Detect pin

```
#include "mbed.h"
DigitalOut leds[] = {LED1, LED2, LED3, LED4};
DigitalIn zeroGDetect(p25);
DigitalOut gSelect(p26);
DigitalOut sleep(p24);

int main() {
     gSelect = 1; // 6g mode
     sleep = 1; // do not sleep
     while (true) {
          if (zeroGDetect) { // zeroGDetect becomes 1 (true) if
     0G is detected
               for (int i = 0; i < 4; i++) {
                    leds[i] = 1;
               }
          }
          wait(0.1);
     }
}
```

Use an interrupt to detect 0G

```
#include "mbed.h"
DigitalOut leds[] = {LED1, LED2, LED3, LED4};
InterruptIn zeroGEvent(p25);
DigitalOut gSelect(p26);
DigitalOut sleep(p24);

void trigger() {
     for (int i = 0; i < 4; i++) {
          leds[i] = 1;
     }
}

int main() {
     gSelect = 1; // 6g mode
     sleep = 1; // do not sleep
```

```
        zeroGEvent.rise(trigger);
            while (true) {
                wait(60);
            }
}
```

Accelerometer calibration article is presented in Appendix C for further reading.

3.7 READ THE MBED API LIBRARY

An application programming interface (API) is a source code based specification intended to be used as an interface by software components to communicate with each other. An API may include specifications for routines, data structures, object classes, and variables. This section describe how to access API libraries in the mbed.or web site as shown in the Figures 3-36, 3-37, 3-38, and 3-39.

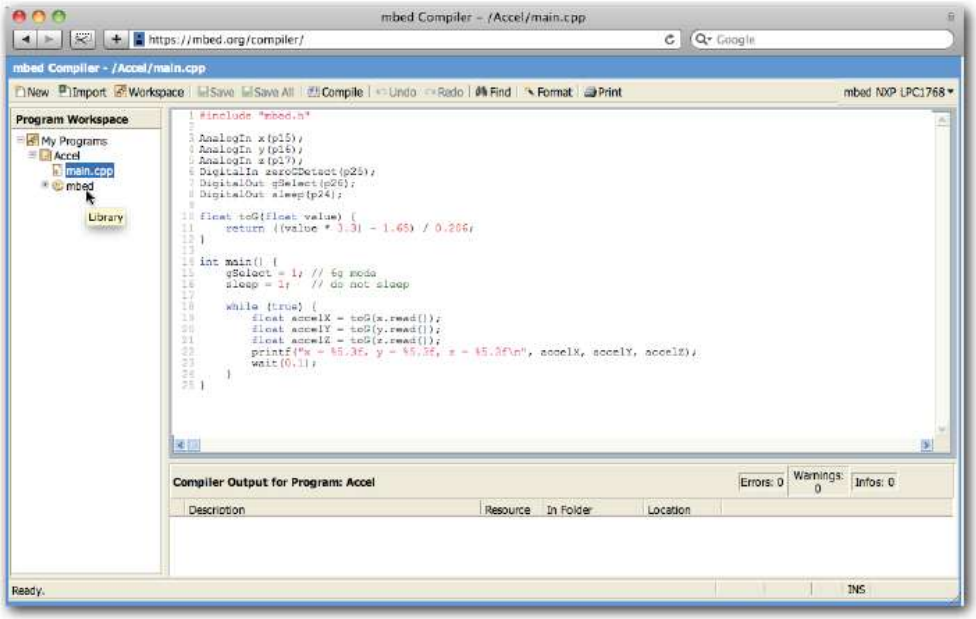


Figure 3-36 Accessing mbed API library

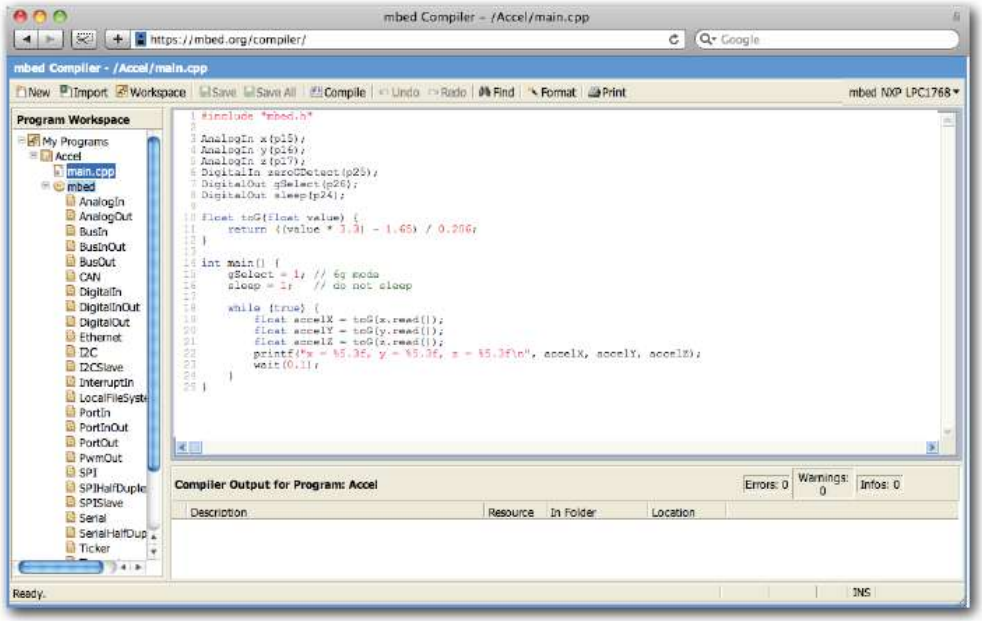


Figure 3-37 Accessing mbed API library

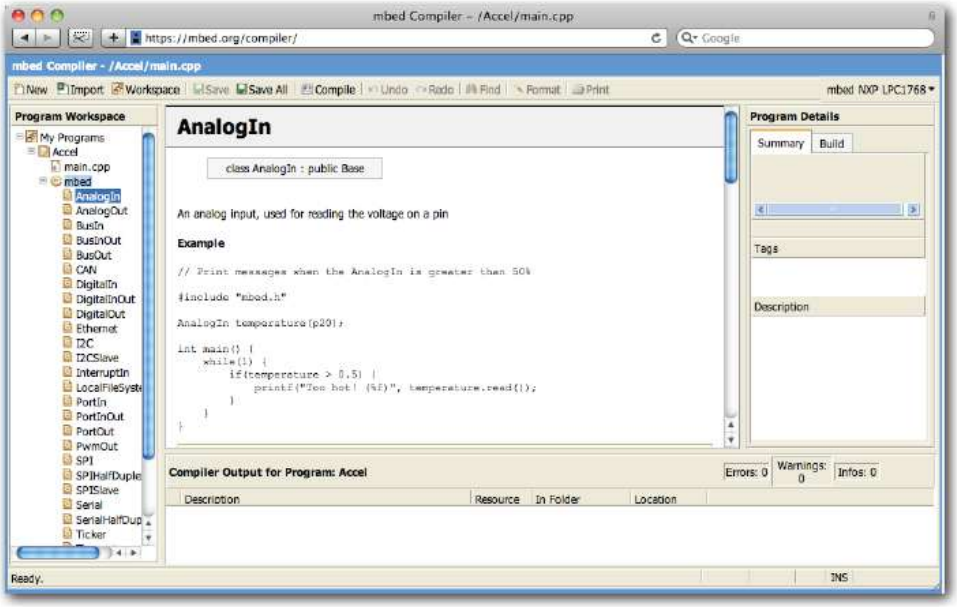


Figure 3-38 Accessing AnalogIn library

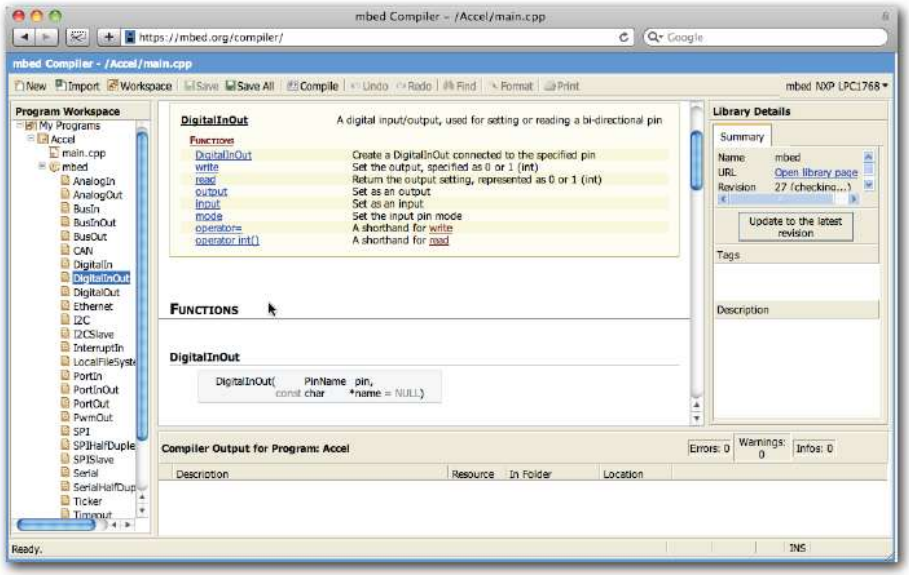


Figure 3-39 Accessing DigitalInOut library

3.8 WRITING A CLASS LIBRARY

This section illustrate how to creat API library for the accelerometer described in section 3.6.

Original code (accelerometer / tilt in 6g mode)

```
#include "mbed.h"
AnalogIn x(p15), y(p16), z(p17);
DigitalOut sleep(p24);
DigitalIn zeroGDetect(p25);
DigitalOut gSelect(p26);

float toG(float value) {
     return ((value * 3.3) - 1.65) / 0.206;
}

int main() {
     gSelect = 1; // 6g mode
     sleep = 1; // do not sleep
          while (true) {
               float accelX = toG(x);
               float accelY = toG(y);
               float accelZ = toG(z);
               float accel = sqrt(accelX * accelX + accelY *
          accelY + accelZ * accelZ);
               float tiltX = asin(accelX/accel);
```

```
            float tiltY = asin(accelY/accel);
            float tiltZ = asin(accelZ/accel);
            printf("accelX = %5.3f, accelY = %5.3f, accelZ
        = %5.3f\n", accelX, accelY, accelZ);
            printf("tiltX = %5.3f, tiltY = %5.3f, tiltZ
        = %5.3f\n", tiltX, tiltY, tiltZ);
            wait(0.2);
        }
}
```

A *class* is an expanded concept of a data structure: instead of holding only data, it can hold both data and functions.

An *object* is an instantiation of a class. In terms of variables, a class would be the type, and an object would be the variable.

Classes are generally declared using the keyword class, with the following format:

```
class class_name {
  access_specifier_1:
    member1;
  access_specifier_2:
    member2;
  ...
  } object_names;
```

Where class_name is a valid identifier for the class, object_names is an optional list of names for objects of this class. The body of the declaration can contain members, that can be either data or function declarations, and optionally access specifiers. a class can include a special function called constructor, which is automatically called whenever a new object of this class is created. This constructor function must have the same name as the class, and cannot have any return type; not even void.

The accelerometer class name, constructors, and function can be listed as follows:

- Name
    - Accelerometer
- Constructor
    - Accelerometer(pins...)
- Functions
    - float getAccel() // get total acceleration
    - float getAccelX(), getAccelY(), getAccelZ()
    - float getTiltX(), getTiltY(), getTiltZ()
    - void setScale(scale) // 1.5G or 6G mode
    - void setSleep(bool) // set sensor to sleep mode
    - bool detectedZeroG()

- void setZeroGDetectListener(func) // call func at 0g

Rewrite using the accelerometer class

```
#include "mbed.h"
#include "Accelerometer.h"
Accelerometer accel(p15, p16, p17, p24, p25, p26); // x, y, z,
sleep, 0g, g-select

int main() {
     accel.setScale(Accelerometer::SCALE_6G);
          while (true) {
               printf("accelX = %5.3f, accelY = %5.3f, accelZ
          = %5.3f\n",
          accel.getAccelX(), accel.getAccelY(),
     accel.getAccelZ());
          printf("tiltX = %5.3f, tiltY = %5.3f, tiltZ = %5.3f\n",
          accel.getTiltX(), accel.getTiltY(), accel.getTiltZ());
     wait(0.2);
          }
}
```

Where **Accelerometer.h** can be written as:

```
#ifndef ACCELEROMETER_H
#define ACCELEROMETER_H
#include "mbed.h"

class Accelerometer {
public:
     Accelerometer(PinName xoutPin, PinName youtPin,PinName
zoutPin,
     PinName sleepPin, PinName zeroGDetectPin, PinName
gSelectPin);
     enum Scale {SCALE_1_5G, SCALE_6G};
     float getAccel();
     float getAccelX();
     float getAccelY();
     float getAccelZ();
     float getTiltX();
     float getTiltY();
     float getTiltZ();
     void setScale(Scale scale);
     void setSleep(bool on);
     bool detectedZeroG();
     void setZeroGDetectListener(void (*func)(void));
```

```
     template<typename T> void setZeroGDetectListener(T* t, void
(T::*func)(void));
private:
     AnalogIn xout, yout, zout;
     DigitalIn zeroGDetect;
     DigitalOut sleep;
     DigitalOut gSelect;
     InterruptIn zeroG;
     float scale;
};
#endif
```

And Accelerometer.cpp can be written as:

```
#include "Accelerometer.h"
Accelerometer::Accelerometer(PinName xoutPin, PinName
youtPin,PinName zoutPin,PinName sleepPin, PinName zeroGDetectPin,
PinName gSelectPin) :
     xout(xoutPin), yout(youtPin), zout(zoutPin),
     zeroGDetect(zeroGDetectPin),
     sleep(sleepPin), gSelect(gSelectPin), zeroG(zeroGDetectPin)
{
     sleep = 1; // normal mode
     gSelect = 0; // 1.5G mode
     scale = 0.8;
}

float Accelerometer::getAccel() {
     float x = getAccelX();
     float y = getAccelY();
     float z = getAccelZ();
     return sqrt(x * x + y * y + z * z);
}

float Accelerometer::getAccelX() {
     return ((xout * 3.3) - 1.65) / scale;
}

float Accelerometer::getAccelY() {
     return ((yout * 3.3) - 1.65) / scale;
}

float Accelerometer::getAccelZ() {
     return ((zout * 3.3) - 1.65) / scale;
}

float Accelerometer::getTiltX() {
```

```
	float x = getAccelX();
	float y = getAccelY();
	float z = getAccelZ();
	float a = sqrt(x * x + y * y + z * z);
	return asin(x / a);
}

float Accelerometer::getTiltY() {
	float x = getAccelX();
	float y = getAccelY();
	float z = getAccelZ();
	float a = sqrt(x * x + y * y + z * z);
	return asin(y / a);
}
float Accelerometer::getTiltZ() {
	float x = getAccelX();
	float y = getAccelY();
	float z = getAccelZ();
	float a = sqrt(x * x + y * y + z * z);
	return asin(z / a);
}

void Accelerometer::setScale(Scale scale) {
	switch (scale) {
		case SCALE_1_5G:
			this->scale = 0.8;
			gSelect = 0;
			break;
		case SCALE_6G:
			this->scale = 0.206;
			gSelect = 1;
			break;
	}
}

void Accelerometer::setSleep(bool on) {
	sleep = !on;
}

bool Accelerometer::detectedZeroG() {
	return zeroGDetect;
}

void Accelerometer::setZeroGDetectListener(void (*func)(void)) {
	zeroG.rise(func);
}
```

```
template<typename T> void
Accelerometer::setZeroGDetectListener(T* t, void
(T::*func)(void)) {
     zeroG.rise(t, func);
}
```

The following figures will summarize the process of creating the API library

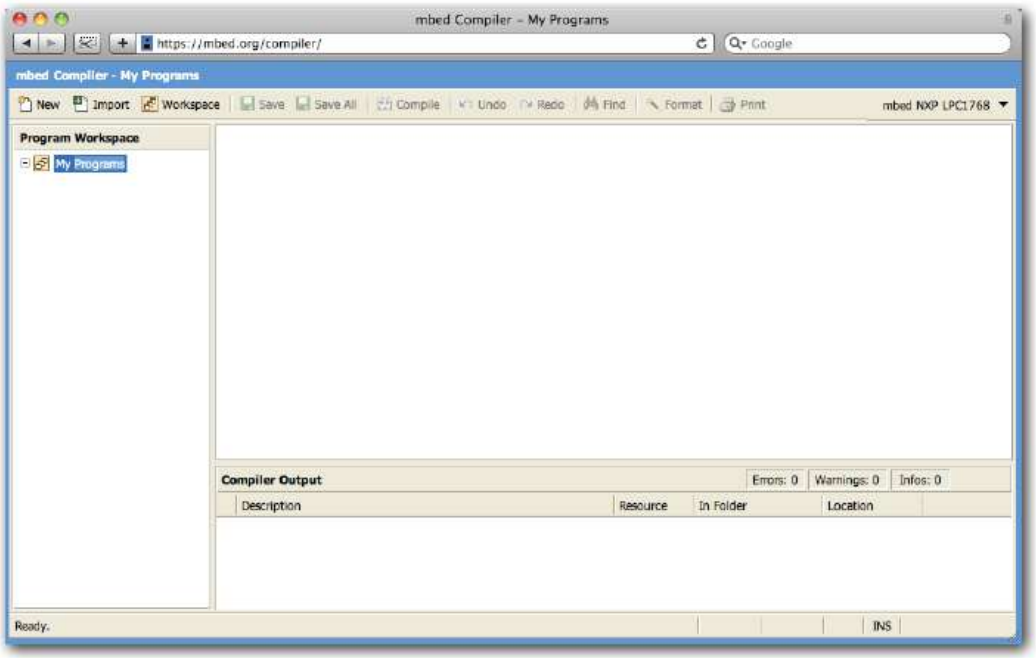


Figure 3-40 Open the complier window

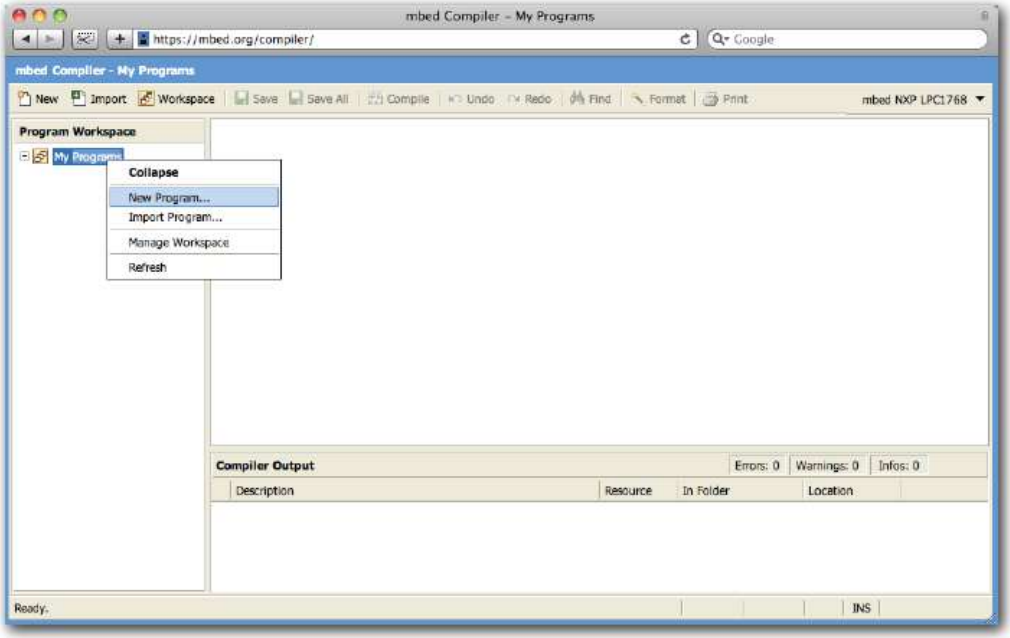


Figure 3-41 Create new program

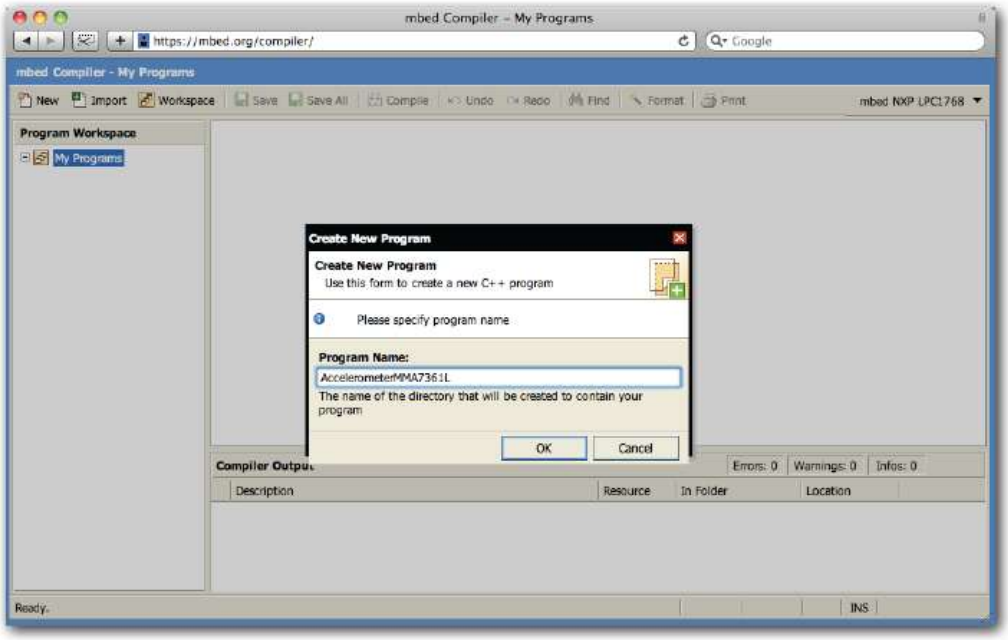


Figure 3-42 Enter the program name

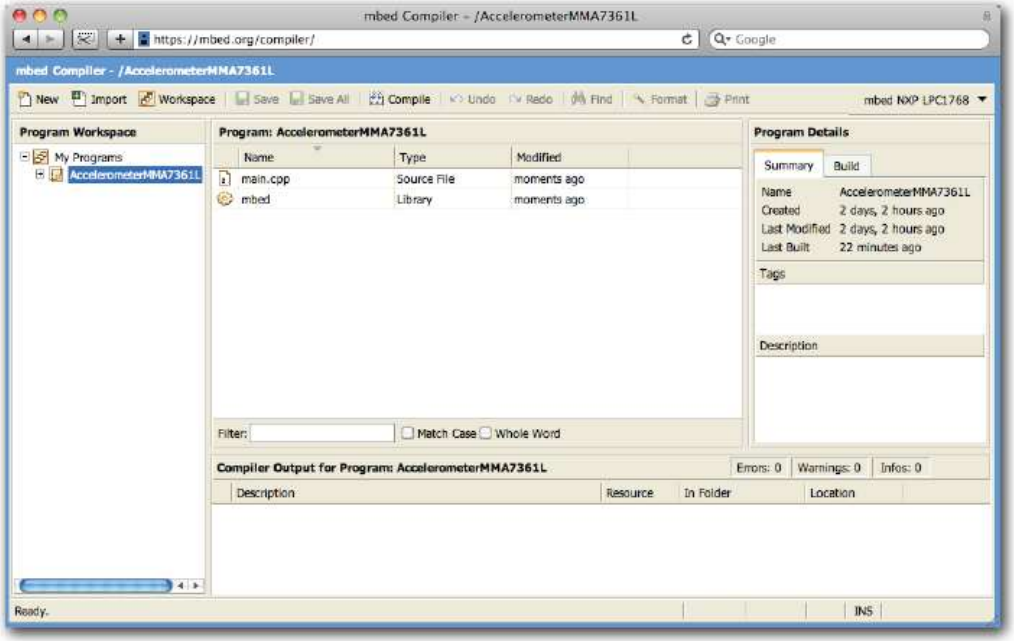


Figure 3-43 Program folder is created

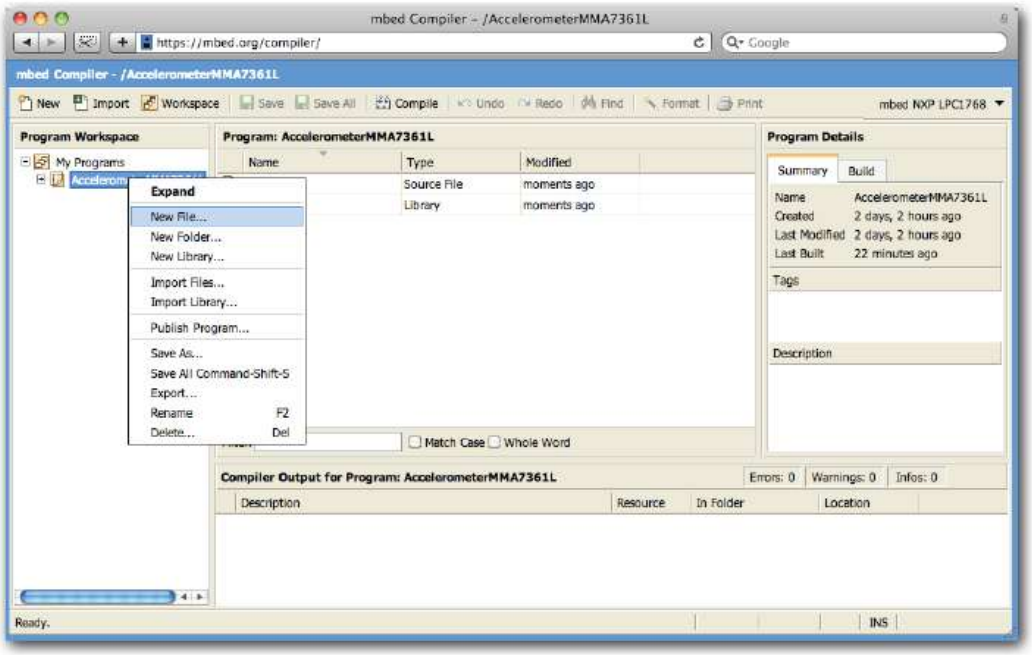


Figure 3-44 Create new file

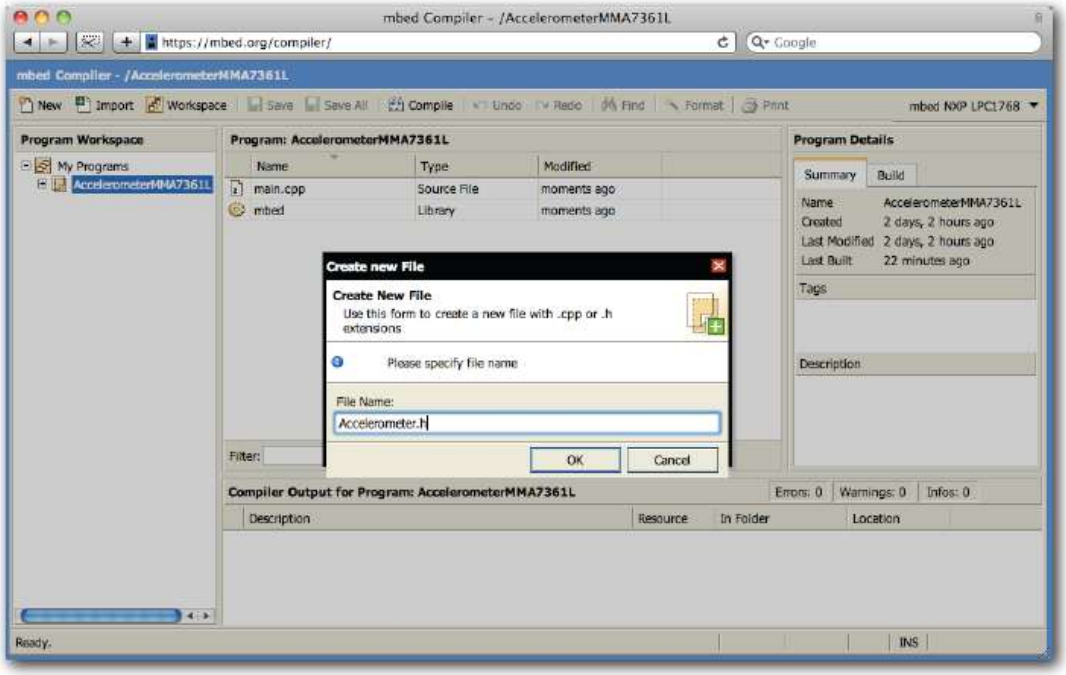


Figure 3-45 Enter the file name “accelerometer.h”

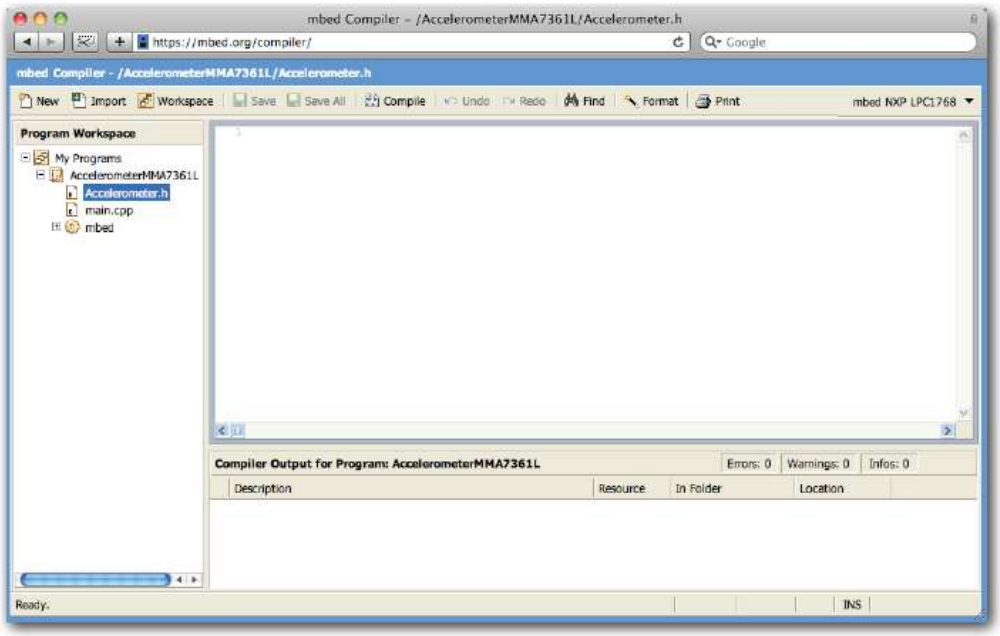


Figure 3-46 Empty file will appear

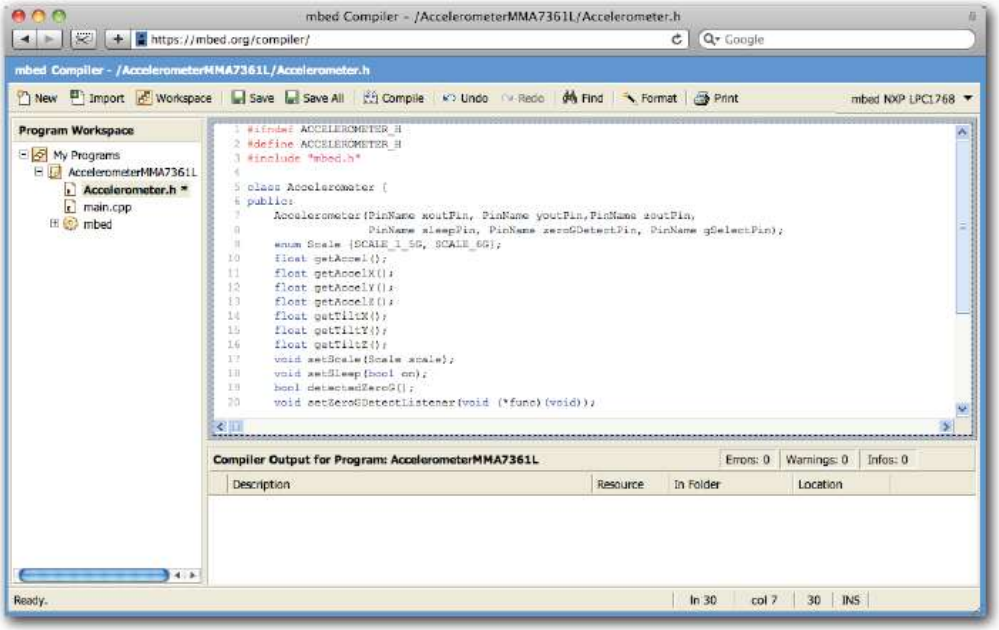


Figure 3-47 Write down the **"`Accelerometer.h`"**

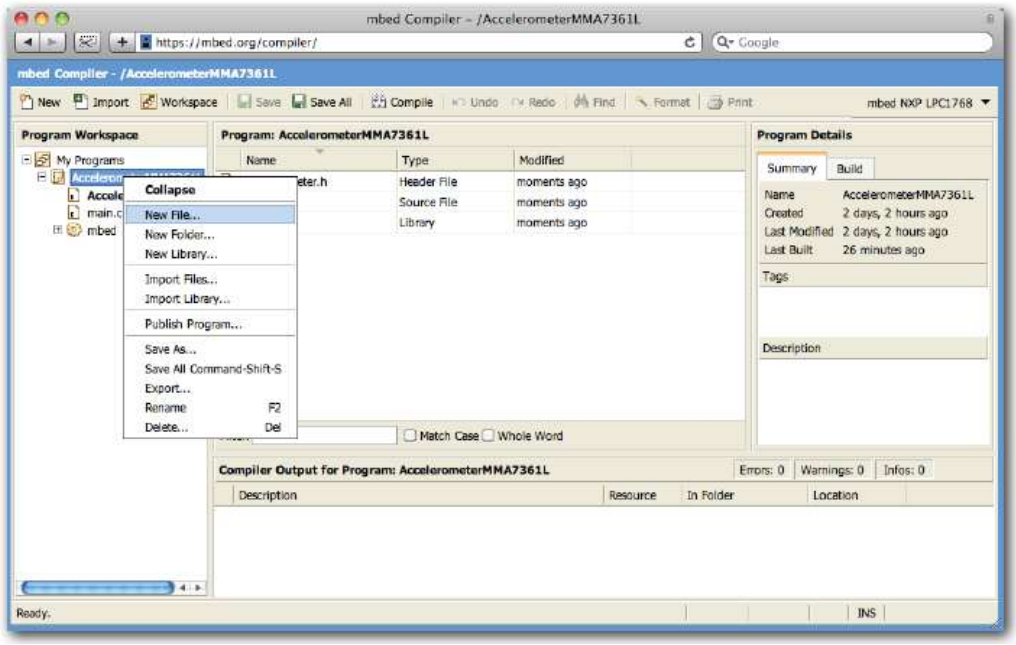


Figure 3-48 Create another new file

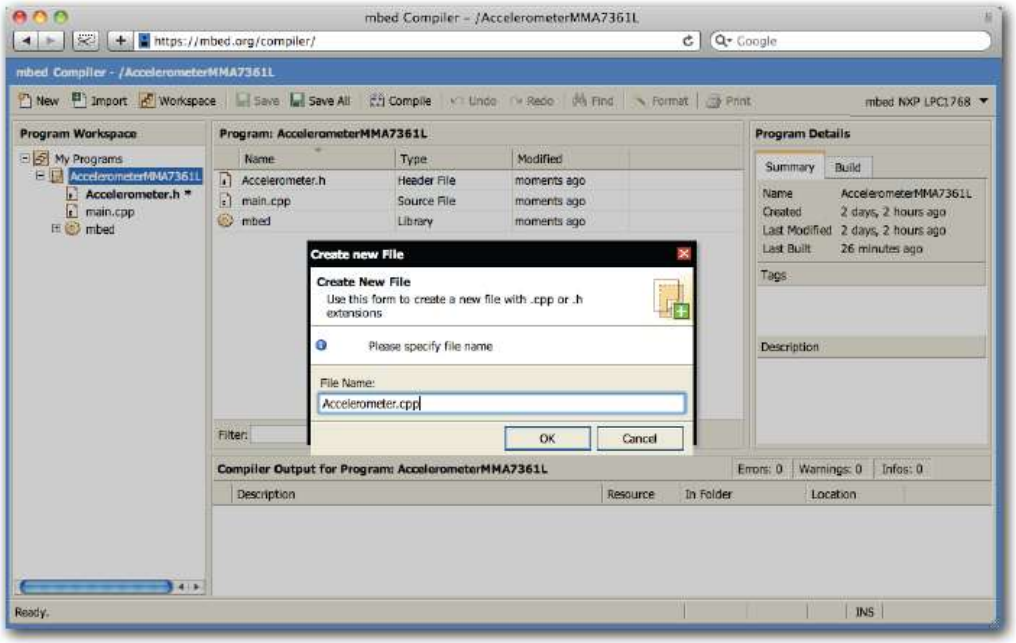


Figure 3-49 Enter the file name “accelerometer.cpp

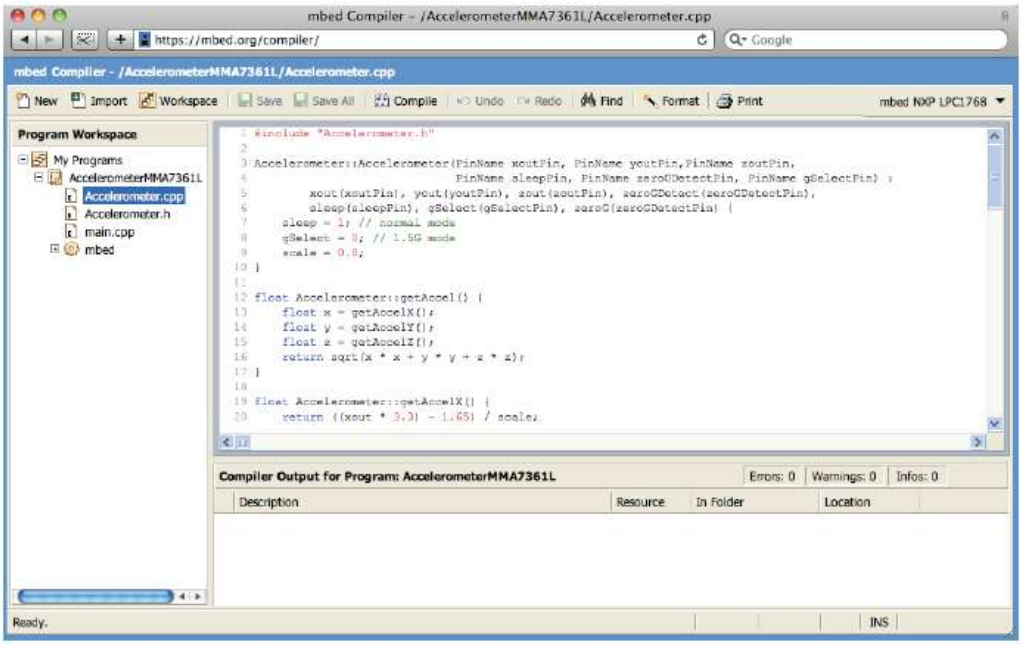


Figure 3-50 Write down the "**`Accelerometer.cpp`**"

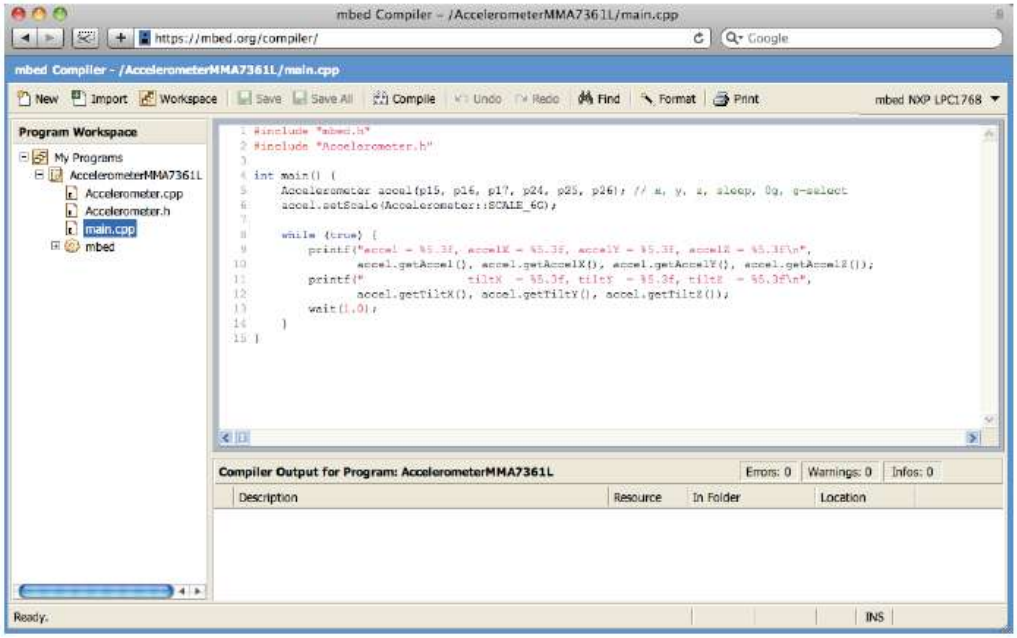


Figure 3-51 Write the main function as written above

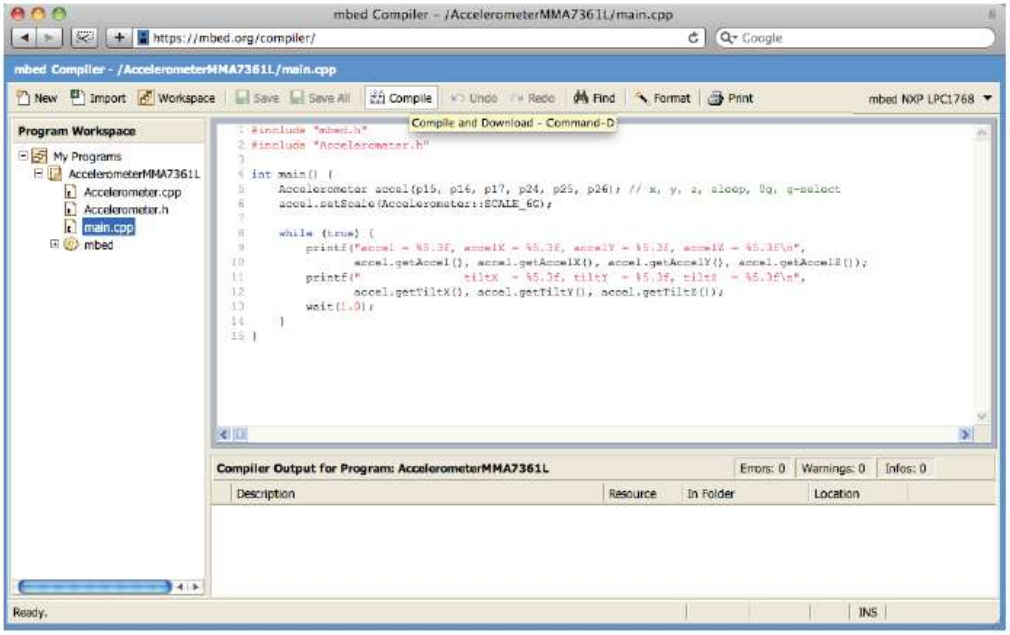


Figure 3-52 Compile the program

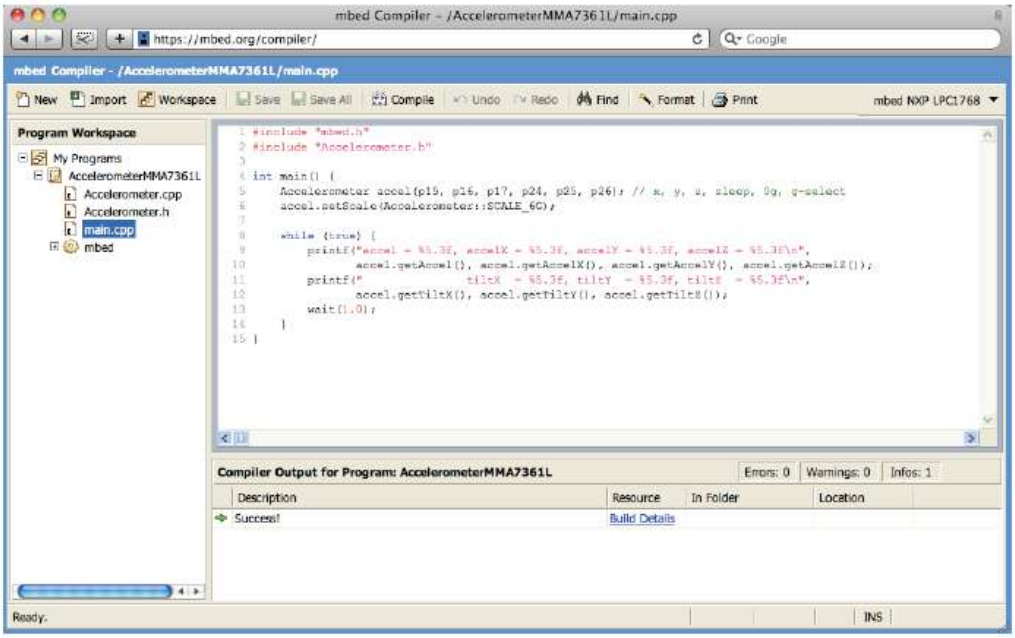


Figure 3-53 Successful compilation

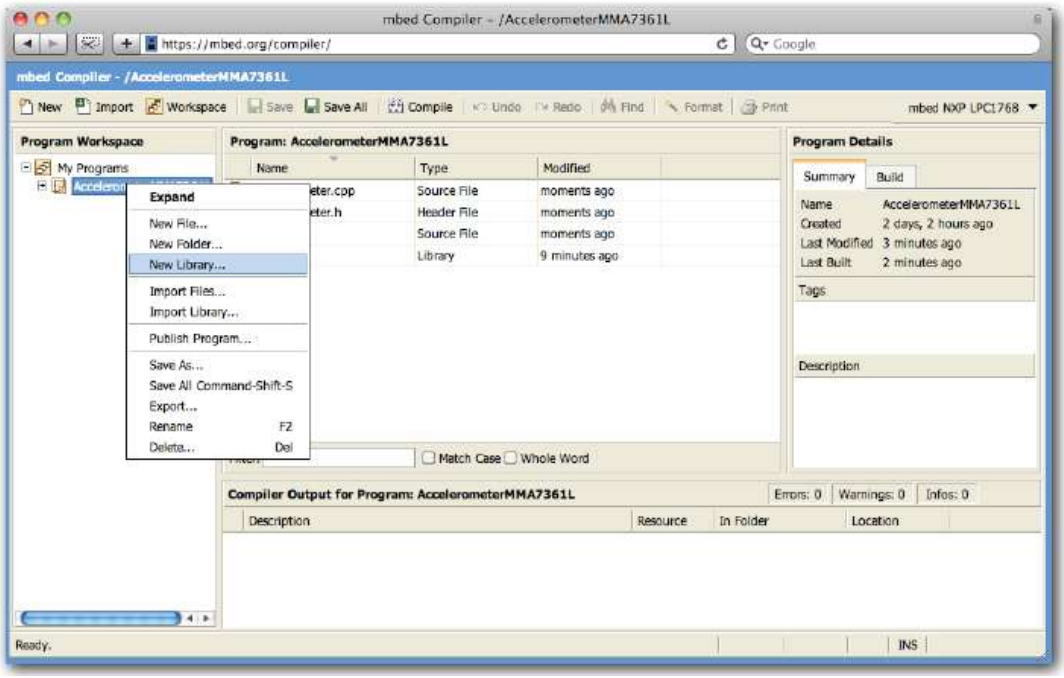


Figure 3-54 Create a new libraray

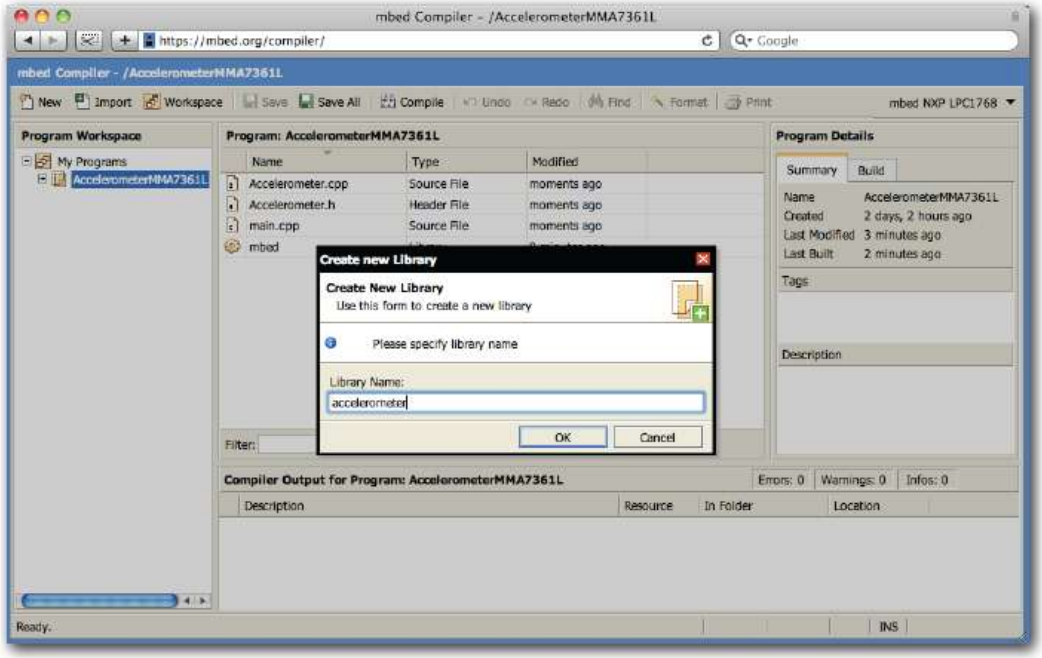


Figure 3-55 Enter the library name as “accelerometer”

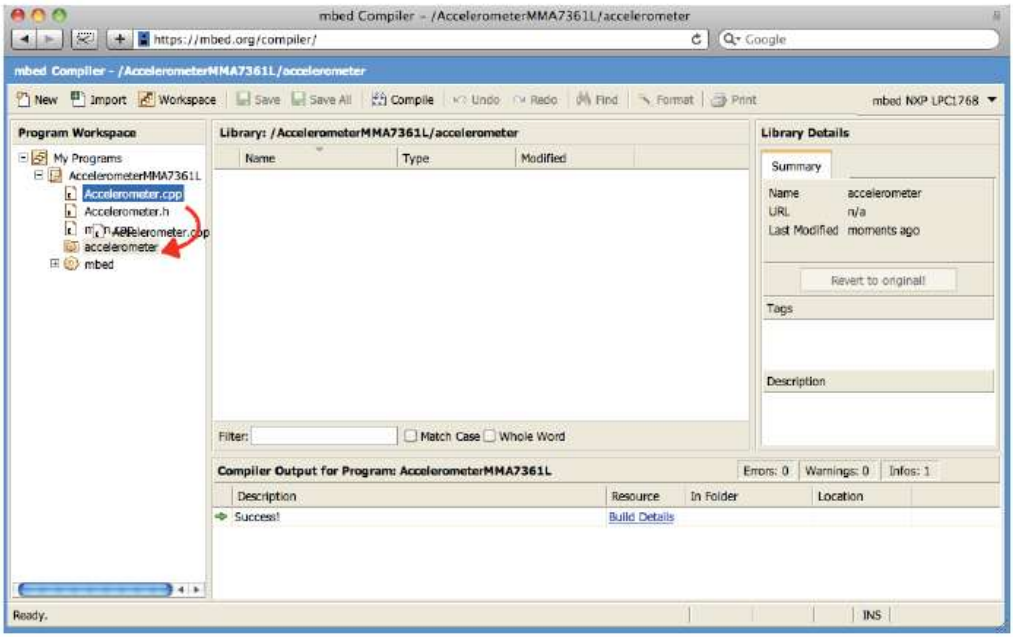


Figure 3-56 Drag the “accelerometer.cpp” to the library

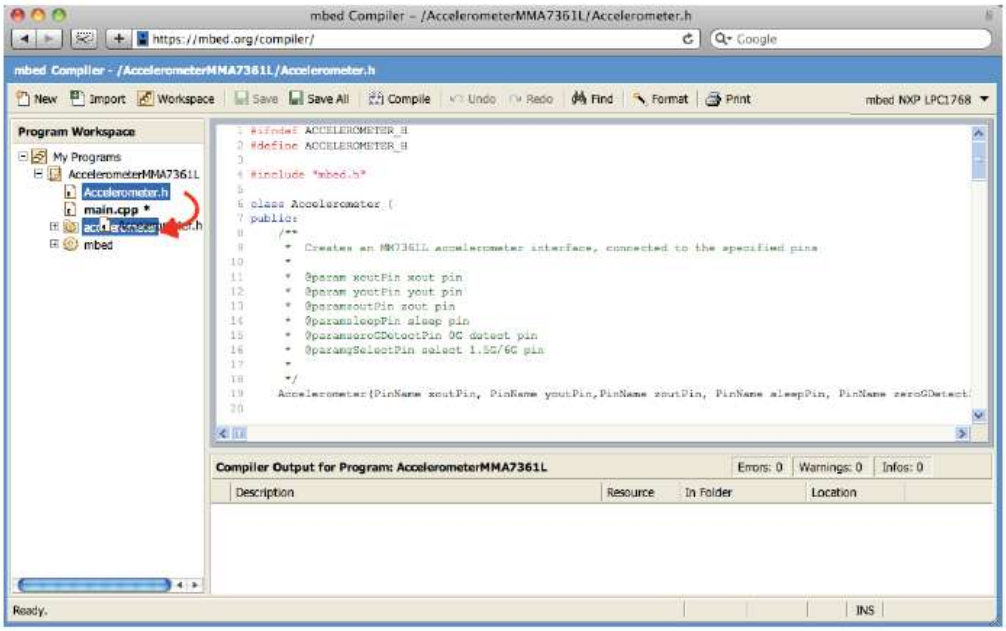


Figure 3-57 Drag the “accelerometer.h” to the library

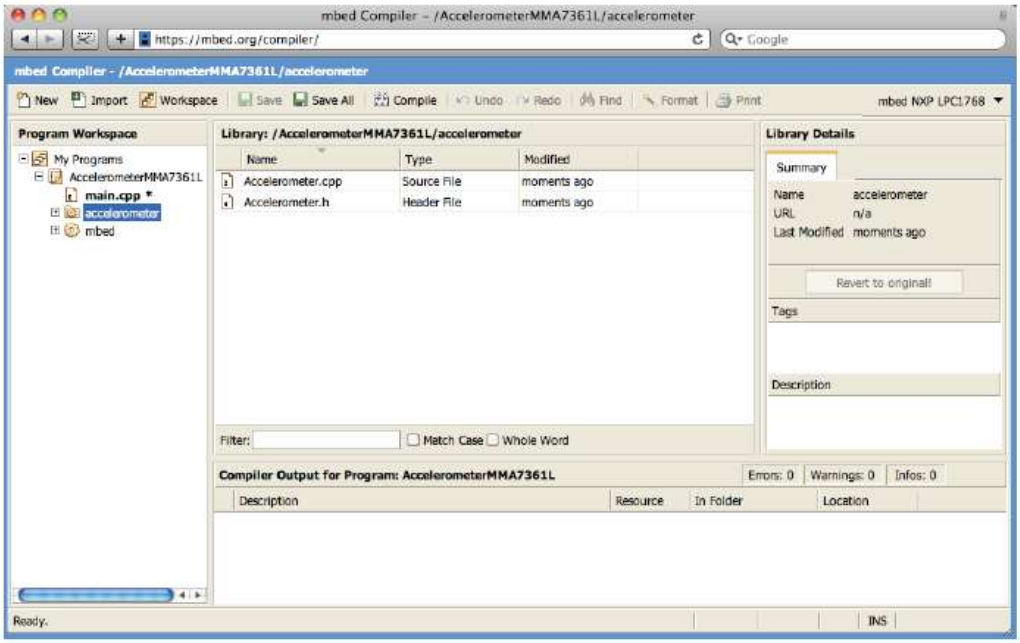


Figure 3-58 Library has been created

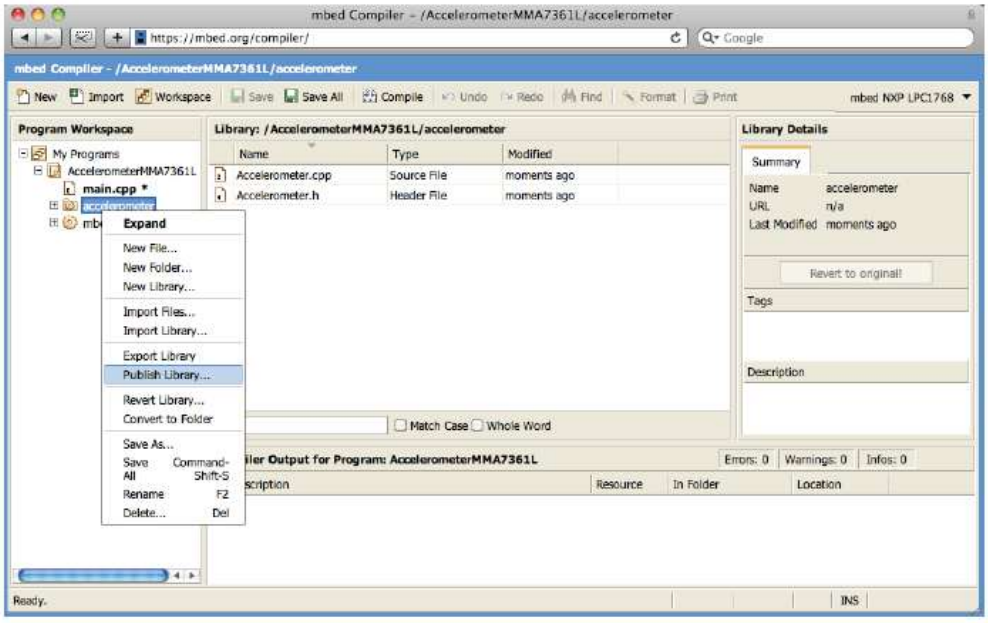


Figure 3-59 Publishing the library

3.9 USING THE MBED LOCAL FILE SYSTEM

A file system for accessing the local mbed USB disk drive allows programs to read and write files on the same disk drive that is used to program the mbed Microprocessor. Standard C file access functions are used to open, read and write. If

the microprocessor program makes an access to the local drive, it will be marked as "removed" on the host computer. This means it is no longer accessible from the host computer. The drive will only re-appear when the microcontroller program exits. Note that if the program does not exit, you will need to hold down the reset button on the mbed to be able to see the drive again.

The first example is to write "Hello World!" to a file is to write down a program to perform the following function;

- Create a local file system "local"
- Create & open a new file "out.txt" on the file system, see Figure 3-60.
- Write a message "Hello World!" to the file
- Close the file

The resulting code will be as follows;

```
#include "mbed.h"
LocalFileSystem local("local");

int main() {
     FILE *fp = fopen("/local/out.txt", "w");
     fprintf(fp, "Hello World!\n");
     fclose(fp);
}
```

To Run the program do the following steps:

- Eject (or unmount) the mbed USB disk from your computer
    - use "safely remove hardware" on Windows
    - do NOT unplug the USB cable
- Press the reset button
- The mbed USB disk will automatically be mounted, when the program finishes

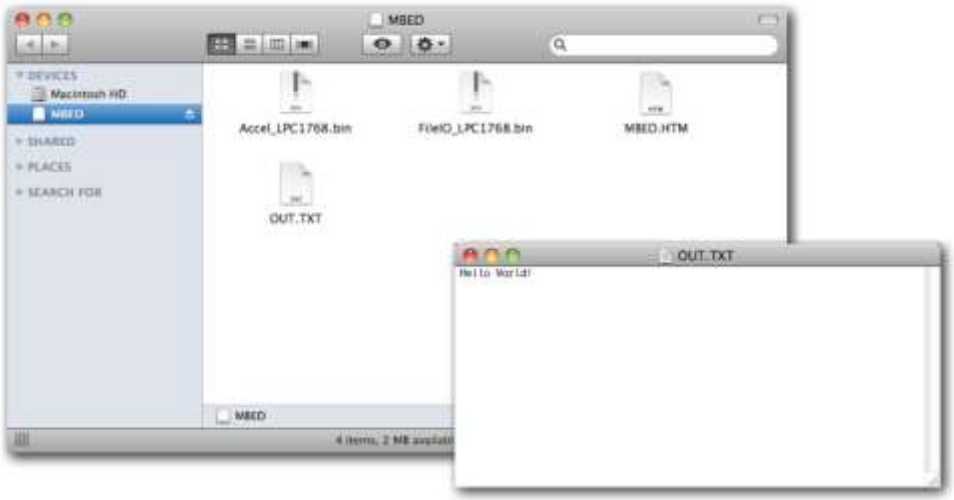


Figure 3-60 OUT.TXT appears on your mbed

Standard C file access functions are

- **FILE** - a file struct
- **fopen** - creates/opens a file struct and returns its pointer, or returns 0 if it fails, see Table 3-4
- **fgetc** - reads a character from the file
- **fputc** - writes a character to the file
- **fscanf** - reads input from the file and scans according to the format
- **fprintf** - writes a format string to the file
- **fclose** - completes the file I/O operations and releases the file struct

Table 3-4 fopen access modes (text mode)

mode string	description
"r"	Open a file for reading. The file must exist
"w"	Create an empty file for writing. If a file with the same name already exists its content is erased and the file is treated as a new empty file
"a"	Append to a file. Writing operations append data at the end of the file. The file is created if it does not exist
"r+"	Open a file for update both reading and writing. The file must exist.
"w+"	Create an empty file for both reading and writing. If a file with the same name already exists its content is erased and the file is treated as a new empty file
"a+"	Open a file for reading and appending. All writing operations are performed at the end of the file, protecting the previous content to be overwritten. You can reposition (fseek, rewind) the internal pointer to anywhere in the file for reading, but writing operations will move it back to the end of file. The file is created if it does not exist.

fopen will return 0 if error occurs

The following code shows how to read and write files

```
#include "mbed.h"

LocalFileSystem local("local");

int main() {
     FILE *fp = fopen("/local/out.txt", "r");
     FILE *fp2 = fopen("/local/out2.txt", "w");
     for (int c; (c = fgetc(fp)) != EOF; ) {
          fputc(c, fp2);
     }
     fclose(fp);
     fclose(fp2);
}
```

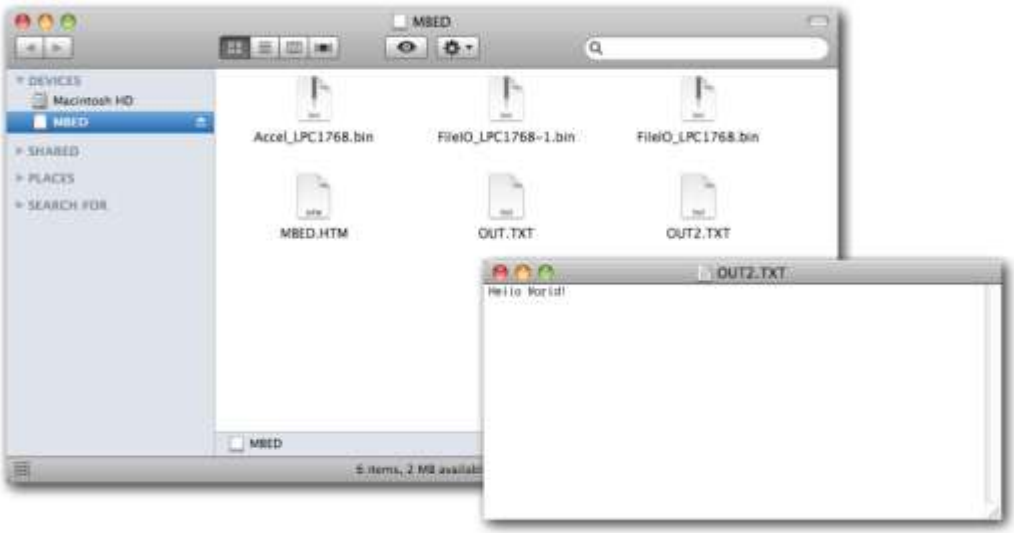


Figure 3-61 OUT2.TXT appears

To scan a data file, the following steps should be done

- Use a text editor to create a file "number.txt" on your mbed USB disk; The first line of the file should consist of your favorite number (positive integer)
- Read the file and get the number x
- Create a new file "sum.txt" and write a sentence (with appropriate substitutions to x and xxx); The sum of all the integers from 1 to x is xxx.

The code will be as follow

```
#include "mbed.h"

LocalFileSystem local("local");

int main() {
     FILE *fp = fopen("/local/number.txt", "r");
     FILE *fp2 = fopen("/local/sum.txt", "w");
     int x;
     fscanf(fp, "%d", &x);
          long sum = 0;
     for (int i = 1; i <= x; i++) {
          sum += i;
     }
     fprintf(fp2, "The sum of all the integers from 1 to %d
is %d\n", x, sum);
     fclose(fp);
     fclose(fp2);
}
```

3.10 USING THE REAL-TIME CLOCK

The mbed Microcontroller device has a built-in real-time clock (RTC) to provide the time to its applications. When powered, the time starts from 00:00:00 on Jan 1, 1970, which requires an adjustment to report a correct time. If properly powered with a backup battery, the RTC keeps running even when the mbed CPU itself is powered off.

To set the real time the time.h functions will be presented first. They are as follows;

- **time** - gets current time; returns the current timestamp, the number of seconds since January 1, 1970, 00:00:00 (the Epoch)
- **set_time** - Initializes and sets the time of the mbed RTC (real-time clock) to the time represented by the number of seconds from the
- Epoch
- **localtime** - converts a timestamp to a **tm** struct (time struct data)
- **strftime** - converts a **tm** struct to a human-readable string
- **mktime** - converts a **tm** struct to the corresponding timestamp

The following code setting the current time until December 1, 2011, time 00:00:00.

```
#include "mbed.h"

int main() {
     time_t seconds;
     char buf[32];
     seconds = (((2011 - 1970) * 365 + (2011 - 1968) / 4)
     + 31 + 28 + 31 + 30 + 31 + 30 + 31 + 31 + 30 + 31 + 30 ) *
24 * 3600;
     set_time(seconds);
     seconds = time(NULL);
     strftime(buf, sizeof(buf), "%x %X", localtime(&seconds));
     printf("mbed rtc = %s (timestamp = %d)\n", buf, seconds);
}
```

The setting equation can be explained as follows

$$\overbrace{(((2011-1970)*365+(2011-1968)/4)}^{\text{Days until the end of last years}}$$
$$+\underbrace{31+28+31+30+31+30+31+31+30+31+30}_{\text{Days fro Janaurayfirst until December 1, 2011, Time 00:00:00}})*24*3600;$$

The time format in the printf and strftime statements are defined in the following table.

Table 3-5 Time format specifiers

Specifier	Description
%S	Second (00-59)
%M	Minute (00-59)
%H	Hour (00-23)
%d	Day (01-31)
%m	Month (01-12)
%Y/%y	Year (2009/09)
%A/%a	Weekday Name (Monday/Mon)
%B/%b	Month Name (January/Jan)
%I	12 Hour Format (01-12)
%p	"AM" or "PM"
%X	Time (14:55:02)
%x	Date (08/23/01)

The following figures, Figure 3-62, shows the result on your terminal application

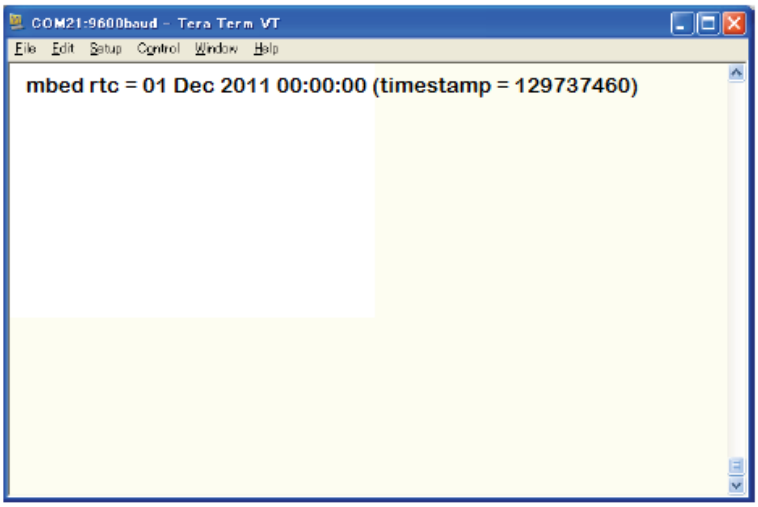


Figure 3-62 Time setting results in terminal application

The RTC keep running if connected to small 3V coin cell battery (e.g. CR1220). Use VB and GND pins to connect the cell as shown in the following figure.

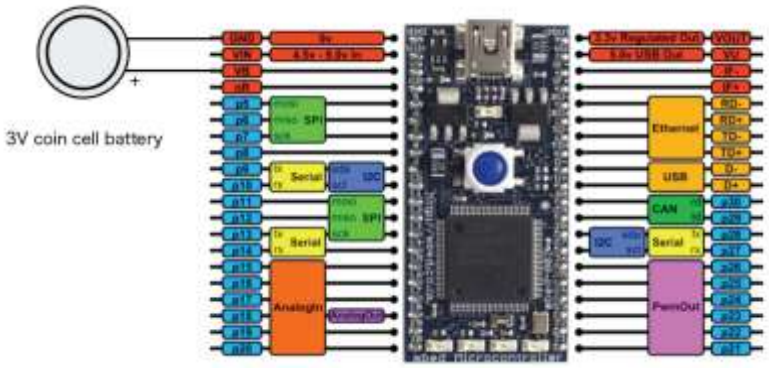


Figure 3-63 Backup Battery for RTC

To adjusting the RTC of your mbed use the next program to adjust the RTC and do the following procedures

- Open your terminal application with local echo mode on i.e. check the terminal configuration setting to switch the echo mode to on status
- Run the mbed program; It will ask you to type either t or T
  - type t for displaying current RTC time (5 seconds)
  - type T for adjusting the RTC: you are requested to enter the
- Correct date and time in **"yy/mm/dd hh:mm:ss"** format

```
#include "mbed.h"

void printTime(int n) {
     printf("\n");
     time_t prev = time(NULL);
     while (n > 0) {
          time_t seconds = time(NULL);
          if (prev != seconds) {
               char buf[32];
               strftime(buf, sizeof(buf), "%x %X",
          localtime(&seconds));
               printf("%s\n", buf);
               prev = seconds;
               n--;
          }
     }
}


void adjustTime() {
     int n1, n2, n3, n4, n5, n6;
     struct tm t;
     printf("\nEnter current time (yy/mm/dd hh:mm:ss)\n");
     scanf("%d/%d/%d %d:%d:%d", &n1, &n2, &n3, &n4, &n5, &n6);
```

```
      t.tm_sec = n6;
      t.tm_min = n5;
      t.tm_hour = n4;
      t.tm_mday = n3;
      t.tm_mon = n2 - 1;
      t.tm_year = n1 + 100;
      set_time(mktime(&t));
}

int main() {
      if (time(NULL) == (time_t) -1) {
            set_time(0);
      }
      printf("Adjust RTC - Enter t or T\n");

      while (true) {
            switch (getchar()) {
                  case 'T':
                        adjustTime();
                        break;
                  case 't':
                        printTime(5);
                        break;
            }
      }
}
```

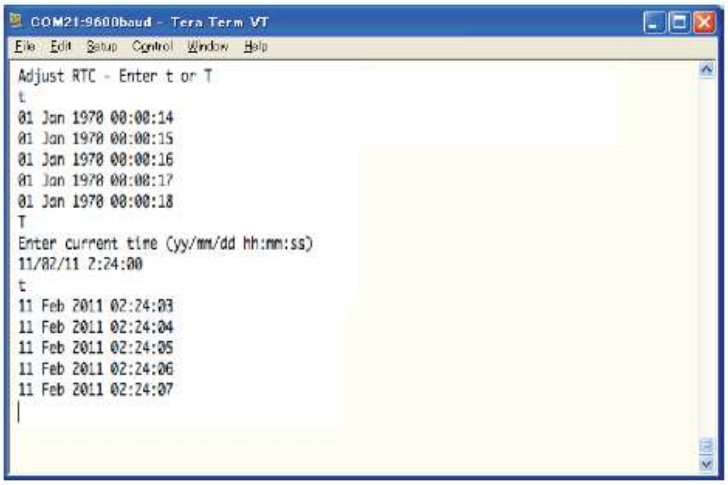


Figure 3-64 Adjusting the time

3.11 WRITING A DATA RECORDER

This section will describe how to write an acceleration data logger for the accelerometer described above. First the following things should be done:

- Choose 1.5G or 6G mode (or make it selectable)
- Periodically (e.g. every 0.1 second), record the measurement and time to a local file
- Use a tact switch to start/stop recording
- Use LEDs to show the status

The following hardware are required to perform this task:

- an accelerometer (MMA7361L)
- a tact switch
- a breadboard and jump wires
- a 9V battery (optional)
- MMA7361L interface library

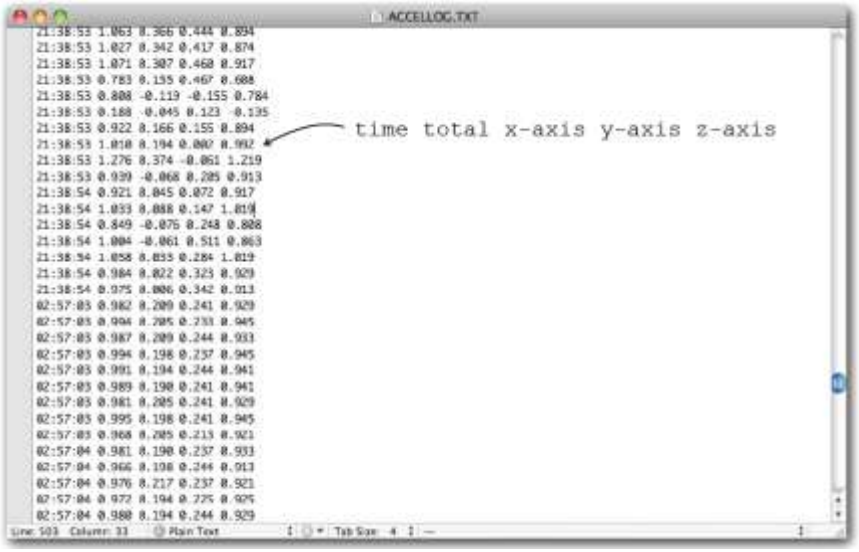


Figure 3-65 Logger output sample

Connection of the accelerometer is described in Figure 3-66 and Figure 3-67.

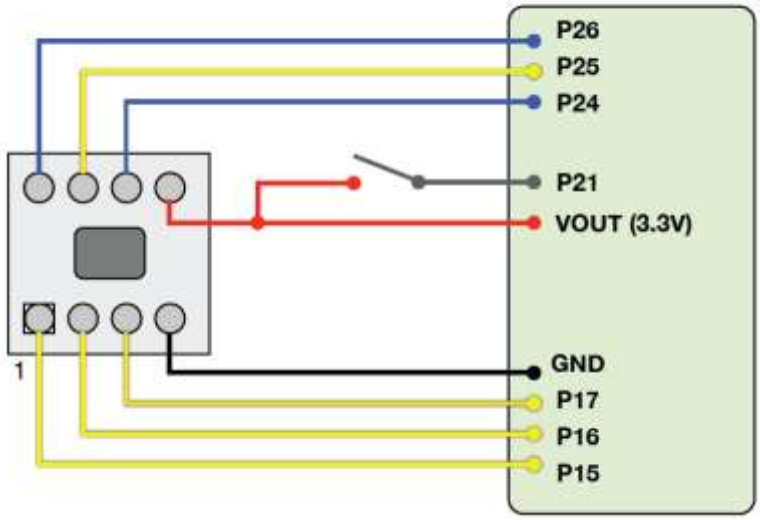


Figure 3-66 Schematic of wiring diagram

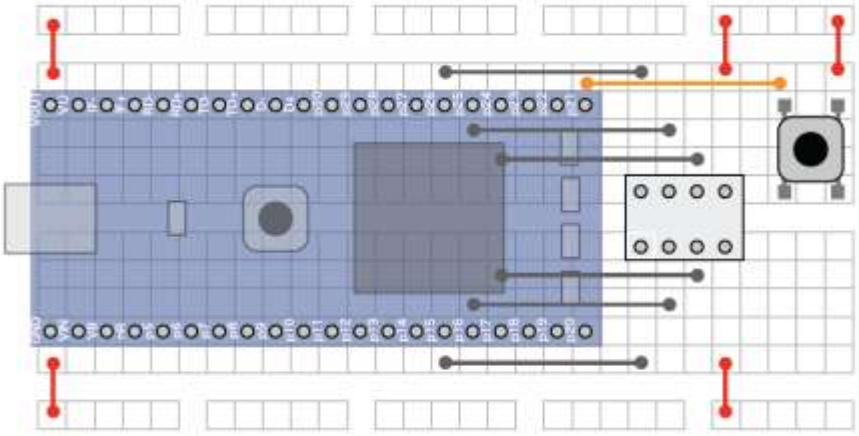


Figure 3-67 Circuiting on a breadboard

To import the accelerometer library follow the steps illustreated in the following figures

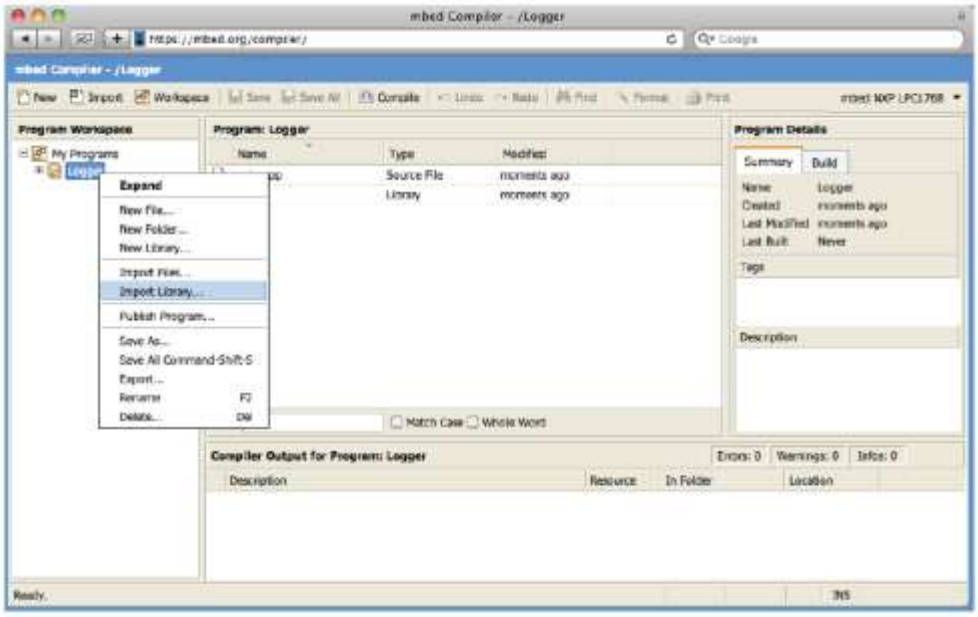


Figure 3-68 Import Library

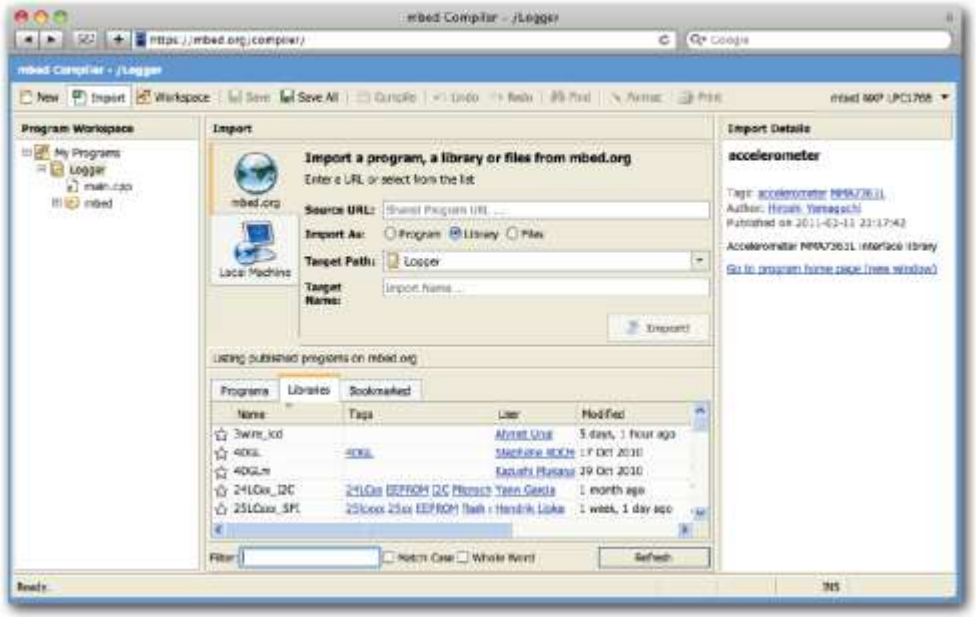


Figure 3-69 Select mbed.org tab

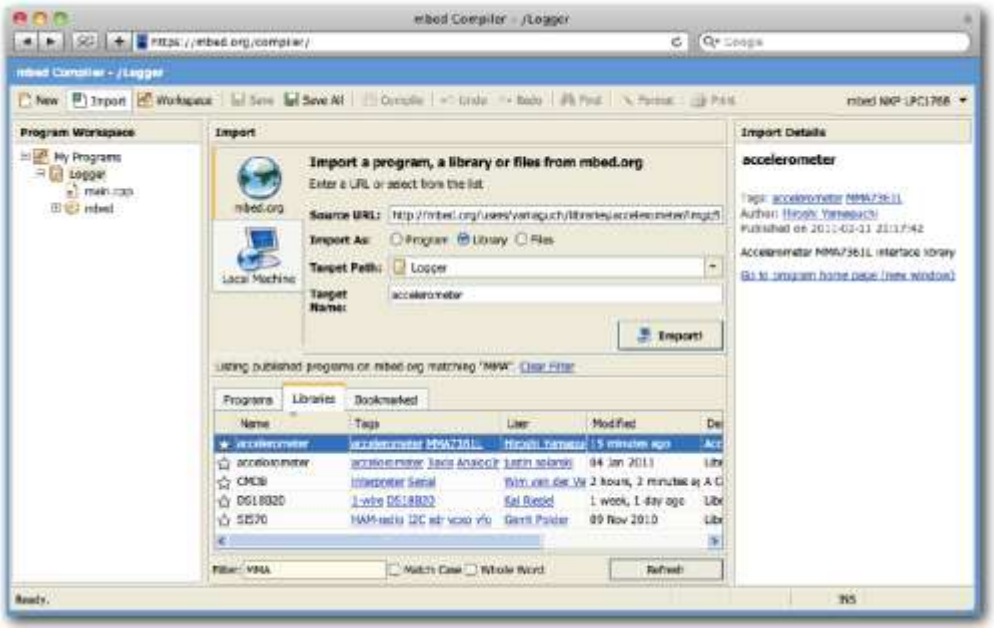


Figure 3-70 Find the library and press "Import!"

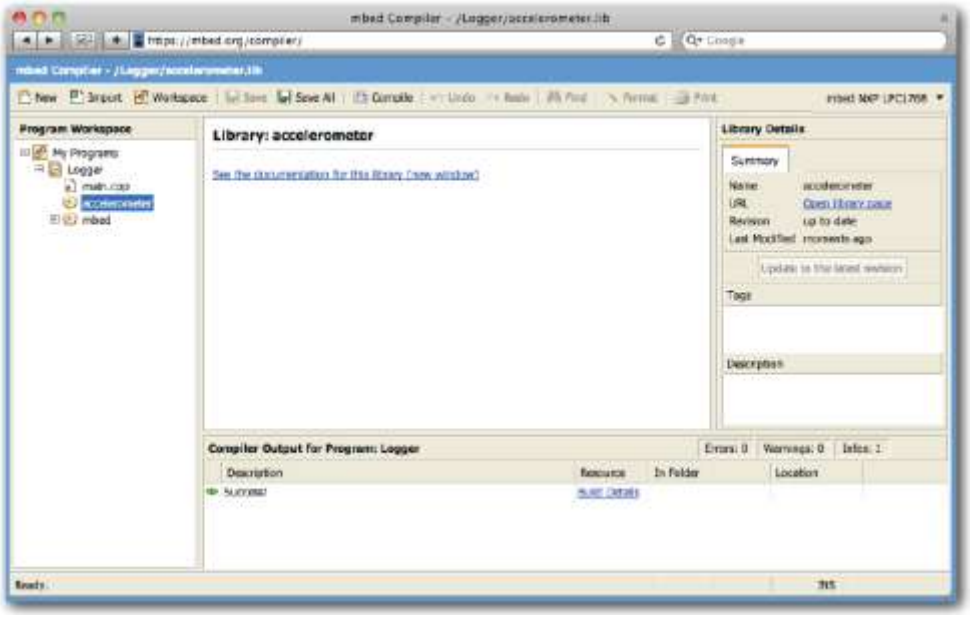


Figure 3-71 Library has been imported

Figure 3-72 Documentation about the library

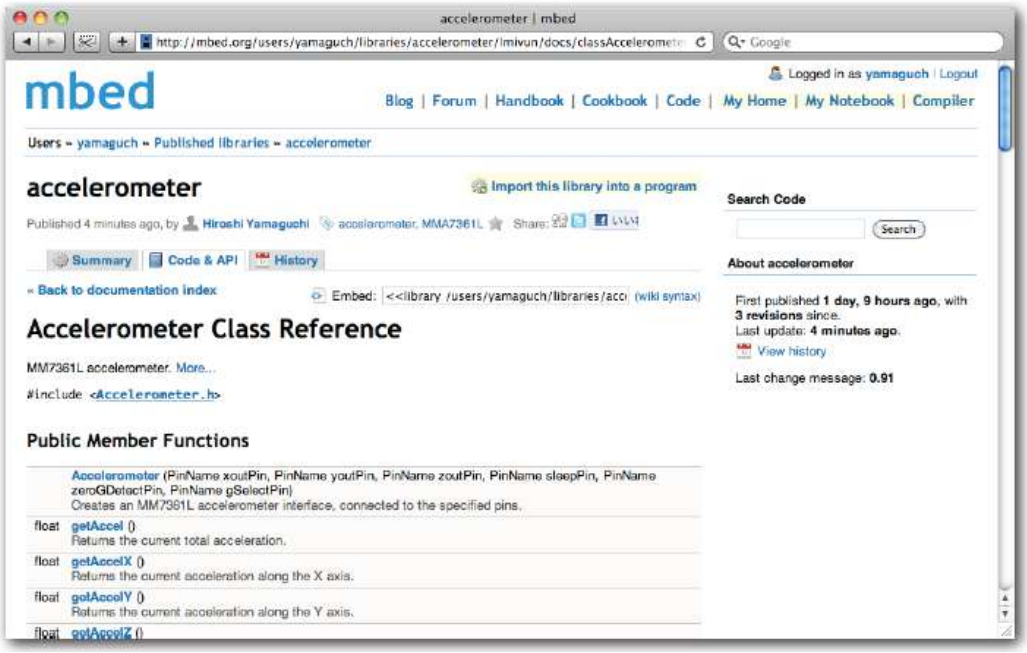


Figure 3-73 APIs of the library

The program for data logger can be written as follows:

```
#include "mbed.h"
#include "Accelerometer.h"
```

```
DigitalOut leds[] = {LED1, LED2, LED3, LED4};
DigitalIn swPressed(p21);
Accelerometer accel(p15, p16, p17, p24, p25, p26);
LocalFileSystem local("local");

void logAccel(FILE *fp, Accelerometer& accel) {
     char buf[32];
     time_t seconds = time(NULL);
     if (seconds == (time_t) -1) {
          set_time(0);
          seconds = 0;
     }
     float x = accel.getAccelX();
     float y = accel.getAccelY();
     float z = accel.getAccelZ();
     float a = sqrt(x * x + y * y + z * z);
     strftime(buf, sizeof(buf), "%X", localtime(&seconds));
     fprintf(fp, "%s %5.3f %5.3f %5.3f %5.3f\n", buf, a, x, y,
z);
}


int main() {
     FILE *fp = fopen("/local/accellog.txt", "a");

     accel.setScale(Accelerometer::SCALE_6G);

     for (bool on = true; !swPressed; on = !on) {
          leds[0] = on;
          wait(0.1);
     }

     for (int i = 0; i < 4; i++) {
          leds[i] = true;
     }

     wait(1.0);

     for (int i = 0; i < 4; i++) {
          leds[i] = false;
     }

     while (!swPressed) {
          leds[1] = true;
          logAccel(fp, accel);
          leds[1] = false;
          wait(0.1);
```

```
        }
        fclose(fp);
}
```

CHAPTER 4: INTRODUCTION TO GLOBAL POSITIONING SYSTEM (GPS)

4.1 INTRODUCTION

The Global Positioning System (GPS) is a satellite-based navigation system that was developed by the U.S. Department of Defense (DoD) in the early 1970s as the next generation replacement to the Transit system. Initially, GPS was developed as a military system to fulfill U.S. military needs. However, it was later made available to civilians, and is now a dual-use system that can be accessed by both military and civilian users [4-1].

GPS provides continuous positioning and timing information anywhere in the world under any weather conditions. Since GPS is a one-way-ranging (passive) system, it serves an unlimited number of users. That is, users can only receive the satellite signals. GPS consists, nominally, of a constellation of about 24 operational satellites [4-1].

GPS consists of three segments: space, control, and user, as shown in Figure 4-1. The space segment consists of about 24 satellites constellation. The control segment of the GPS system consists of a worldwide network of tracking stations. The user segment includes all military and civilian users. With a GPS receiver connected to a GPS antenna, a user can receive the GPS signals, which can be used to determine his or her position anywhere in the world. GPS is currently available to all users worldwide at no direct charge [4-1].

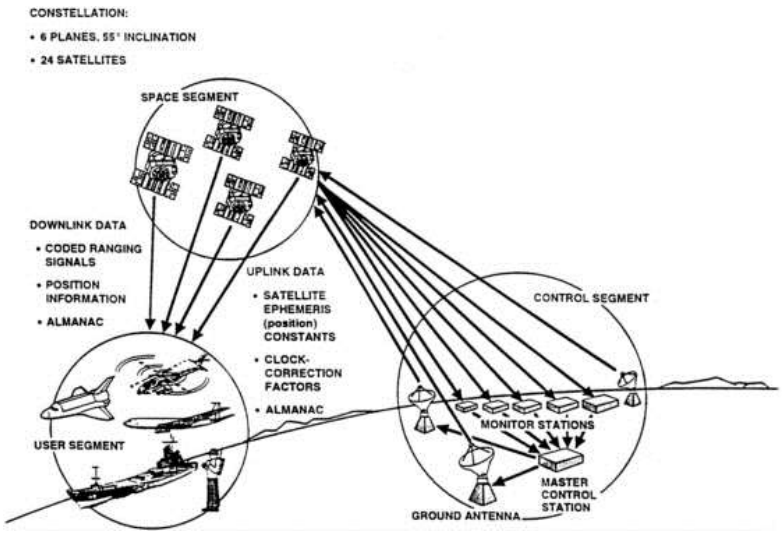


Figure 4-1 GPS Segments

The idea behind GPS is rather simple. If the distances from a point on the Earth (a GPS receiver) to three GPS satellites are known along with the satellite locations, then the location of the point (or receiver) can be determined by simply applying the well-known concept of

resection [4-1]. That is all! But how can we get the distances to the satellites as well as the satellite locations?.

Each GPS satellite continuously transmits a microwave radio signal composed of two carriers, two codes (or more for modernized satellites), and a navigation message. When a GPS receiver is switched on, it will pick up the GPS signal through the receiver antenna. Once the receiver acquires the GPS signal, it will process it using its built-in processing software. The partial outcome of the signal processing consists of the distances to the GPS satellites through the digital codes (known as the pseudo ranges) and the satellite coordinates through the navigation message. Theoretically, only three distances to three simultaneously tracked satellites are needed. In this case, the receiver would be located at the intersection of three spheres; each has a radius of one receiver-satellite distance and is centered on that particular satellite as shown in Figure 4-2.

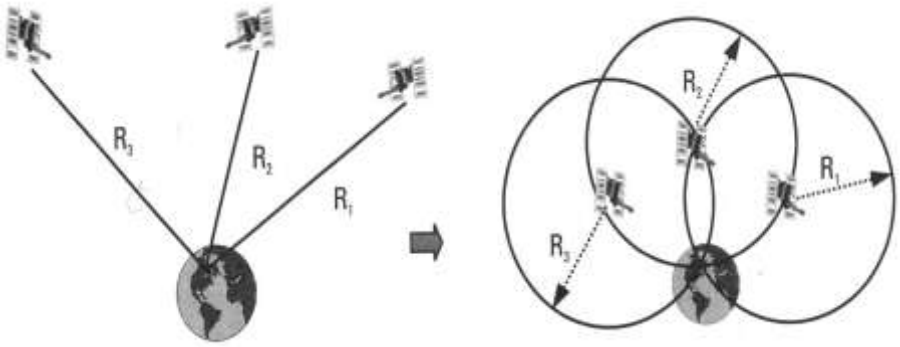


Figure 4-2 Basic idea of GPS positioning [4].

4.2 GPS STANDARD DATA FORMAT

Since individual GPS manufacturers have their own proprietary format for storing GPS measurements, it can be difficult to combine data from different receivers. A similar problem is encountered when interfacing various devices, including the GPS receiver. To overcome these limitations, a number of research groups and agencies have developed standard formats for various user needs. Brief description of one of these formats (NMEA 0183) will be presented in the subsequent section which is the format of the GPS (GT-723F) employed in this textbook, Specification sheet of the GPS receiver can be found in Appendix A.

4.3 NMEA 0183 FORMAT

NMEA was founded in 1957 by a group of electronics dealers to strengthen their relationships with electronic manufacturers. The NMEA 0183 standards are data streams in the ASCII format, transmitted at a rate of 4,800 bps, from a talker to a listener (one-way), where a talker is a device that sends data to other devices (e.g., a GPS receiver) and a listener is a device that receives data from another device (e.g., a laptop computer interfaced with the GPS

receiver). A high-speed version of NMEA 0183 is also available, which operates at a rate up to 38.4 K-baud.

The NMEA 0183 data streams may include information on position, datum, water depth, and other variables. The data is sent in the form of sentences, each starting with a dollar sign "$" and terminating with a carriage return-line feed "<CR><LF>"; the dollar sign "$" is followed by a five-character address field, which identifies the talker (the first two characters) and the format and data type (the last three characters). The data fields are separated by "," delimiter. A null field (i.e., a field with zero length) is used when the value of the field is unavailable or unreliable. The last field in any sentence follows a checksum delimiter character "*" and is a checksum field. The maximum total number of characters in any sentence is 82 (i.e., a maximum of 79 characters between the starting delimiter "$" and the terminating "<CR><LF>"). A number of these sentences are dedicated to GNSS (Global Navigation Satellite System), including GPS and GLONASS (the Russian GPS counterpart) systems, while the remaining sentences support other devices such echo sounders, gyroscopes, and others. Table 4-1 shows common GNSS-related NMEA sentences, which are defined in the current version of NMEA 0183.

Our discussion will be restricted to one sentence only, which is commonly used in practice-the GGA, or GPS fix data. This sentence represents the time, position, and solution-related information. Figure 4-3 shows the general structure of the GGA sentence, while Table 4-2 explains the terms of the sentence.

Table 4-1 Common GNSS-Related NEMA Sentences

NMEA Sentence	Description
AIM	GPS almanac data
GBS	GNSS satellite fault detection
GGA	GPS fix data
GMP	GNSS map projection fix data
GNS	GNSS fix data
GRS	GNSS range residuals
GSA	GNSS DOP and active satellites
GST	GNSS pseudorange error statistics
GSV	GNSS satellites in view

Figure 4-3 General Structure of a GGA sentence

Table 4-2 Explanation of GCA Sentence

Symbol	Explanation
$	Start of sentence delimiter
GP	Talker identifier (GPS in this case)
GCA	Data identifier (GPS fix data in this case)
,	Data field delimiter
hhmmss.ss	Time of position in UTC system
	(hoursminutesseconds.decimall. Decimal
IIII.II	digits are variable.
A	Latitude (degreesminutes.decimall. Decimal digits are variable.
yyyyy.yy	N/S (North or South)
a	Longitude (degreesminutes.decimal). Decimal digits are variable.
x	E/W (East or West)
	GPS quality indicator (0 = Fix not available or invalid)
	(1 = Point positioning with CIA-code)
	(2 = DGPS with CIA-code)
	(3 = Point Positioning with P-code)
	(4 = RTK with ambiguity parameters fixed to integer values)
	(5 = RTK with float ambiguity parameters)
	(6 = Estimated [dead reckoning] mode)
	(7 = Manula input model)
	(8 = Simulator mode)
xx	Number of satellites used in producing the solution
x.x	Horizontal Dilution of Precision (HOOP)
x.x	Altitude above geoid (Orthometric height)
M	Meters (units of Orthometric height)
x.x	Geoidal Height above the WGS84 ellipsoid (geoid-ellipsoid
M	separation) Meters (units of Geoidal Height)
x.x	Age of Differential GPS data in seconds (time since last RTCM message type 1 or 9 was received; null filed when Differential GPS mode is not used).
xxxx	Differential reference station 10 (range 0000-1023)
*	Checksum delimiter character
hh	Checksum field (last field in the sentence)
<CR><LF>	Sentence terminator

Most GPS receivers available on the market support the NMEA 0183 standards. However, not all receivers with the NMEA 0183 port output all the GPS-specific messages. In addition, some GPS receiver manufacturers may slightly change the standard format. However, they typically provide software to interpret the data sentence.

4.4 GPS DATUM

The fact that the topographic surface of the Earth is highly irregular makes it difficult for geodetic calculations to be performed. To overcome this problem, geodesists adopted a smooth mathematical surface, called the reference surface, to approximate the irregular shape of the Earth (more precisely to approximate the global mean sea level, the geoid) [4]. One such mathematical surface is the sphere, which has been widely used for low-accuracy positioning. For high-accuracy positioning, such as GPS positioning, however, the best mathematical surface to approximate the Earth and at the same time keep the calculations as simple as possible was found to be the biaxial ellipsoid, as shown if Figure 4-4. The biaxial reference ellipsoid, or simply the reference ellipsoid, is obtained by rotating an ellipse around its minor axis, b. Similar to the ellipse, the biaxial reference ellipsoid can be defined by the semiminor and semimajor axes (a, b) or the semi major axis and the flattening (a, f), where f = 1-(b / a).

An appropriately positioned reference ellipsoid is known as the geodetic datum. In other words, a geodetic datum is a mathematical surface, or a reference ellipsoid, with a well-defined origin (center) and orientation. For example, a geocentric geodetic datum is a geodetic datum with its origin coinciding with the center of the Earth. It is clear that there is an infinite number of geocentric geodetic datums with different orientations. Therefore, a geodetic datum is uniquely determined by specifying eight parameters: two parameters to define the dimension of the reference ellipsoid, three parameters to define the position of the origin, and three parameters to define the orientation of the three axes with respect to the Earth [4-1]. Table 4-3 shows some examples of three common reference systems and their associated ellipsoids.

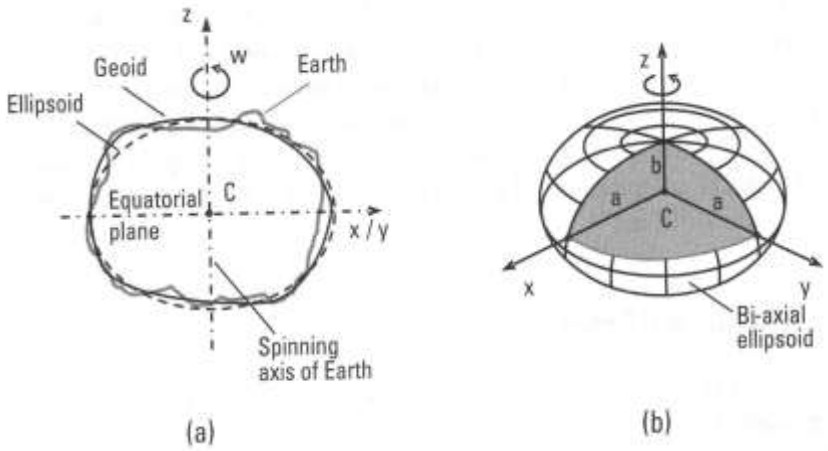


Figure 4-4 (a) Relationship between the physical surface of the Earth, the geoid, and the ellipsoid; (b) ellipsoidal parameters.

Table 4-3 Example of Reference System and Associated Ellipsoid [4-1]

Reference System	Ellipsoid	*a* (m)	*1/f*
WGS 84	WGS 84	6378137.0	298.257223563
NAD 83	GRS 80	6378137.0	298.257222101
NAD 27	Clarke 1866	6378206.4	294.9786982

In addition to the geodetic datum, the so-called vertical datum is used in practice as a reference surface to which the heights (elevations) of points are referred. Because the height of a point directly located on the vertical datum is zero, such a vertical reference surface is commonly known as the surface of zero height. The vertical datum is often selected to be the surface that best approximates the mean sea level on a global basis (the geoid), as shown in Figure 4-4.

4.5 GEODETIC COORDINATE SYSTEM

A coordinate system is defined as a set of rules for specifying the locations (also called coordinates) of points [4-1]. This usually involves specifying an origin of the coordinates as well as a set of reference lines (called axes) with known orientation. Figure 4-5 shows the case of a three-dimensional coordinate system that uses three reference axes (x, y, and z) that intersect at the origin (C) of the coordinate system. Coordinate systems may be classified as one-dimensional, two dimensional, or three-dimensional coordinate systems, according to the number of coordinates required to identify the location of a point. For example, a one-dimensional coordinate system is needed to identify the height of a point above the sea surface.

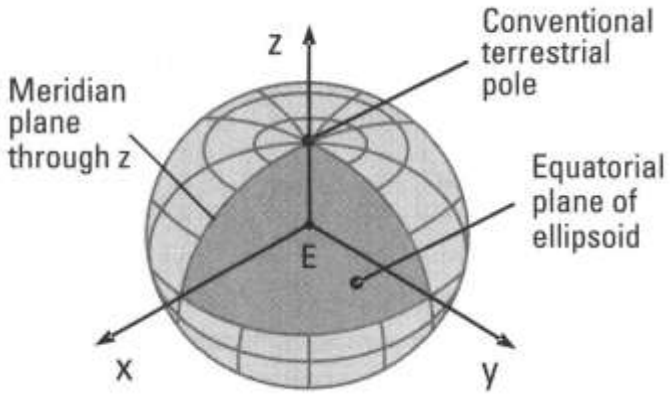


Figure 4-5 Three-dimensional coordinate system

Coordinate systems may also be classified according to the reference surface, the Orientation of the axes, and the origin. In the case of a three-dimensional geodetic (also known as geographic) coordinate system, the reference surface is selected to be the ellipsoid. The orientation of the axes and the origin are specified by two planes: the meridian plane through the polar or z-axis (a meridian is a plane that passes through the north and south poles) and the equatorial plane of the ellipsoid, as shown in Figure 4-5.

Of particular importance to GPS users is the three-dimensional geodetic coordinate system. In this system, the coordinates of a point are identified by the geodetic latitude *(ϕ),* the geodetic longitude (λ), and the height above the reference surface *(h).* Figure 4-6 shows these parameters. Geodetic coordinates *(ϕ, λ,* and *h)* can be easily transformed to Cartesian coordinates *(x, y,* and *z)* as shown in Figure 4-6 (b). To do this, the ellipsoidal parameters (a and f) must be known. It is also possible to transform the geodetic coordinates *(ϕ and λ)* into a rectangular grid coordinate (e.g., Northing and Easting) for mapping purposes. It is useful for some applications to express the coordinates of a point in a system known as the East-North-Up (ENU) system, which is related to the three-dimensional geodetic coordinate system discussed earlier, as shown in Figure 4-7. The ENU has its origin at the user's (i.e., point's) location; therefore, it is also known as the local-level system. The ENU system is a right-handed. It is possible to transform the ENU coordinates into geodetic coordinates, and vice versa.

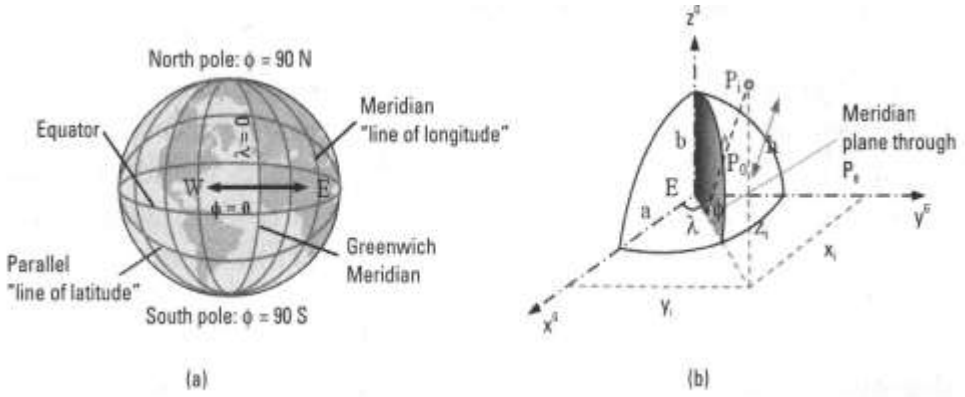


Figure 4-6 (a) Concept of geodetic coordinate; (b) geodetic and Cartesian coordinates.

4.5.1 CONVENTIONAL TERRESTRIAL REFERENCE SYSTEM

The Conventional Terrestrial Reference System (CTRS) is a three dimensional geocentric coordinate system (i.e., its origin coincides with the center of the Earth; as shown in Figure 4-5). The CTRS is rigidly tied to the Earth (i.e., it rotates with the Earth). It is therefore also known as the Earth-centered, Earth-fixed (ECEF) coordinate system. The orientation of the axes of the CTRS is defined as follows: The z-axis points toward the conventional terrestrial pole, which is defined as the average location of the pole during the period 1900-1905. The x-axis is defined by the intersection of the terrestrial equatorial plane and the meridianal plane that contains the mean location of the Greenwich observatory (known as the mean Greenwich meridian). It is clear from the definition of the x- and z-axes that the xz-plane contains the mean Greenwich meridian. The y-axis is selected to make the coordinate system right-handed (i.e., 90 degrees east of the x-axis, measured in the equatorial plane). The three axes intersect at the center of the Earth, as shown in Figure 4-5.

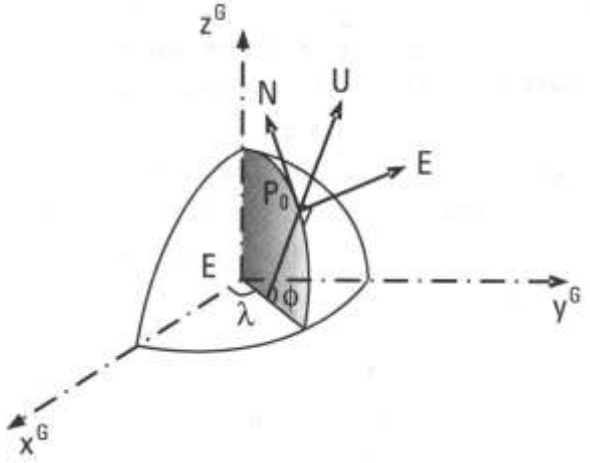


Figure 4-7 East-North-Up coordinate system

4.5.2 THE WGS 84 (WORLD GEODETIC SYSTEM 1984)

The WGS 84 is a three-dimensional, Earth-centered reference system developed by the former U.S. Defense Mapping Agency now incorporated into a new agency, NGA. WGS 84 is the official GPS reference system. In other words, a GPS user who employs the broadcast ephemeris in the solution process will obtain his or her coordinates in the WGS 84 system. The WGS 84 utilizes the CTRS combined with a reference ellipsoid that is identical, up to a very slight difference in flattening, with the ellipsoid of the Geodetic Reference System of 1980 (GRS 80); see Table 4-3. The latter was recommended by the International Association of Geodesy for use in geodetic applications.

4.6 CONVERTING FROM WGS84 TO NAVIGATION COORDINATES (ENU)

The transformation from ϕ,λ,h to ENU is a three stage process [4-2]:

1. Determine latitude, longitude and height of reference point, ($\phi;\lambda;h$).
2. Express small changes in latitude, longitude and height in Earth Centered Earth Fixed (ECEF) coordinates. GPS coordinates can be converted into ECEF coordinates using the following formulae:

$$\begin{aligned} x &= \left(\frac{a}{\chi}+h\right)\cos\phi\cos\lambda \\ y &= \left(\frac{a}{\chi}+h\right)\cos\phi\sin\lambda \\ z &= \left(\frac{a(1-e^2)}{\chi}+h\right)\sin\phi \end{aligned} \qquad (4\text{-}1)$$

Where

$$\chi = \sqrt{1 - e^2 \sin^2 \phi},$$

a and e^2 are the semi-major axis and the first numerical eccentricity of the Earth respectively. To convert small changes in latitude, longitude and height into ECEF coordinates we need to Taylor expand equation (4-1) about $\phi \to \phi + d\phi, \lambda \to \lambda + d\lambda$ and $h \to h + dh$.

$$\begin{aligned}
dx &= \left(\frac{-a\cos\lambda\sin\phi(1-e^2)}{\chi^3} - h\cos\lambda\sin\phi\right)d\phi - \left(\frac{a\sin\lambda\cos\phi}{\chi} + h\sin\lambda\cos\phi\right)d\lambda \\
&+ \cos\phi\cos\lambda dh + \left(\tfrac{1}{4}a\cos\phi\cos\lambda\left(-2-7e^2+9e^2\cos^2\phi\right) - \tfrac{1}{2}h\cos\lambda\cos\phi\right)d\phi^2 \\
&+ \left(\frac{a\sin\lambda\sin\phi(1-e^2)}{\chi^3} + h\sin\lambda\sin\phi\right)d\phi d\lambda - \cos\lambda\sin\phi dh d\phi \\
&+ \left(\frac{-a\cos\lambda\cos\phi}{2\chi} - \tfrac{1}{2}h\cos\lambda\cos\phi\right)d\lambda^2 - \sin\lambda\cos\phi dh d\lambda + \mathcal{O}(d\theta^3) + \mathcal{O}(dhd\theta^2) \\
dy &= \left(\frac{-a\sin\lambda\sin\phi(1-e^2)}{\chi^3} - h\sin\lambda\sin\phi\right)d\phi + \left(\frac{a\cos\lambda\cos\phi}{\chi} + h\cos\lambda\cos\phi\right)d\lambda \\
&+ \sin\phi\cos\lambda dh + \left(\tfrac{1}{4}a\cos\phi\sin\lambda\left(-2-7e^2+9e^2\cos^2\phi\right) - \tfrac{1}{2}h\sin\lambda\cos\phi\right)d\phi^2 \\
&+ \left(\frac{-a\cos\lambda\sin\phi(1-e^2)}{\chi^3} - h\cos\lambda\sin\phi\right)d\phi d\lambda - \sin\lambda\sin\phi dh d\phi \\
&+ \left(\frac{-a\sin\lambda\cos\phi}{2\chi} - \tfrac{1}{2}h\sin\lambda\cos\phi\right)d\lambda^2 + \cos\lambda\cos\phi dh d\lambda + \mathcal{O}(d\theta^3) + \mathcal{O}(dhd\theta^2) \\
dz &= \left(\frac{a(1-e^2)\cos\phi}{\chi^3} + h\cos\phi)\right)d\phi + \sin\phi dh + \cos\phi dh d\phi \\
&+ \left(\tfrac{1}{4}a\sin\phi\left(-2-e^2+9e^2\cos^2\phi\right) - \tfrac{1}{2}h\sin\phi\right)d\phi^2 + \mathcal{O}(dhd\phi^2),
\end{aligned} \tag{4-2}$$

where $d\theta$ is either $d\phi$ or $d\lambda$.

3. By means of a rotation, displacements in ECEF coordinates are transformed to ENU coordinates. The ECEF coordinates (*dx; dy; dz*) are orientated in such a way that from the centre of the Earth the $\hat{z}$ points in the direction of true north, $\hat{x}$ points in the direction of the prime meridian and the direction of $\hat{y}$ is 90° from the prime meridian, see Figure 4-7. The orientation of ENU coordinates is determined by rotating the ECEF Coordinates; firstly about the $\hat{z}$ axis by λ degrees and then the new $\hat{y}$ axis by ϕ degrees.

$$\begin{pmatrix} de \\ dn \\ du \end{pmatrix} = \begin{pmatrix} -\sin\lambda & \cos\lambda & 0 \\ -\sin\phi\cos\lambda & -\sin\phi\sin\lambda & \cos\phi \\ \cos\phi\cos\lambda & \cos\phi\sin\lambda & \sin\phi \end{pmatrix} \begin{pmatrix} dx \\ dy \\ dz \end{pmatrix} \tag{4-3}$$

Substituting equation (4-2) into equation (4-3), and ignoring terms of $O(d\theta^3)$ and $O(dhd\theta^2)$ and higher, we get

$$
\begin{aligned}
de &= \left(\frac{a}{\chi}+h\right)\cos\phi\,d\lambda - \left(\frac{a(1-e^2)}{\chi^3}+h\right)\sin\phi\,d\phi\,d\lambda + \cos\phi\,d\lambda\,dh \\
dn &= \left(\frac{a(1-e^2)}{\chi^3}+h\right)d\phi + \tfrac{3}{2}a\cos\phi\sin\phi e^2 d\phi^2 + dh\,d\phi \\
&+ \tfrac{1}{2}\sin\phi\cos\phi\left(\frac{a}{\chi}+h\right)d\lambda^2 \qquad (4\text{-}4) \\
du &= dh - \tfrac{1}{2}a\left(1-\tfrac{3}{2}e^2\cos\phi+\tfrac{1}{2}e^2+\frac{h}{a}\right)d\phi^2 \\
&- \tfrac{1}{2}\left(\frac{a\cos^2\phi}{\chi}-h\cos^2\phi\right)d\lambda^2 \quad .
\end{aligned}
$$

Note: The coordinates (ϕ;λ;h) are ellipsoidal (WGS84), not spheroidal (geocentric). The difference is most easily explained by Figure 4-8.

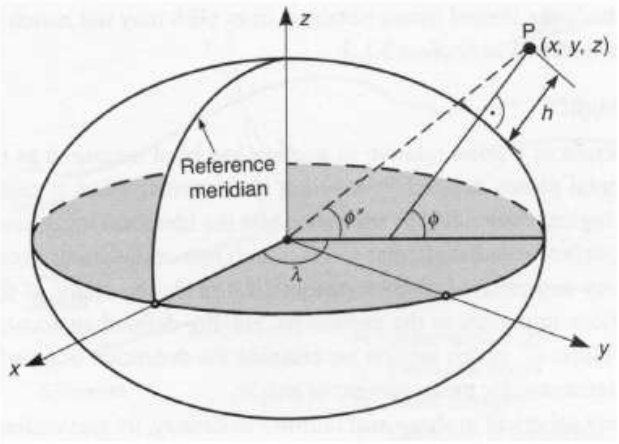


Figure 4-8 Ellipsoidal and spheroidal coordinates

4.7 MATLAB PROGRAM CONVERTING FROM WGS84 TO NAVIGATION COORDINATES (ENU)

Armed with equation (4-4) we are now in a position to write a Matlab program. The code for our program dllh2denu.m is given below. The routine dllh2denu is simple to use, it requires two inputs; the first is a vector indicating the location of the reference point in latitude, longitude in degrees and height in meters. The first input is a 1 x 3 vector of the form [4-2]:

$$\text{llh0} = [\phi;\ \lambda;\ \text{h}] :$$

The second input is a n x 3 matrix indicating the location of the points of interest (e.g. the location of the CanSat). The second input is of the form:

$$\text{llh} = \begin{bmatrix} \phi_1 & \lambda_1 & h_1 \\ . & . & . \\ . & . & . \\ . & . & . \\ \phi_n & \lambda_n & h_n \end{bmatrix}$$

The resulting output is a n x 3 matrix with the first, second, and third columns, representing the east, north and up displacement respectively in meters.

$$\text{denu} = \begin{bmatrix} e_1 & n_1 & u_1 \\ . & . & . \\ . & . & . \\ . & . & . \\ e_n & n_n & u_n \end{bmatrix}$$

As an example suppose the location of the reference point is

$$[\phi = 39^\circ, \lambda = -132^\circ, h = 0]$$

and three points of interested are located at

ϕ	λ	h
$39^\circ + 0.5^\circ$	-132°	0
$39^\circ + 0.5^\circ$	$-132^\circ + 0.5^\circ$	0
$39^\circ + 0.5^\circ$	$-132^\circ + 0.5^\circ$	60000

To determine the location of these points in ENU coordinates we would run dllh2denu in the following way:

```
>> format short g
>> llh0 = [39, -132, 0]

llh0 =

    39  -132     0

>> llh = [39 + 0.5, -132, 0;
39 + 0.5, -132+0.5, 0;
39 + 0.5, -132+0.5, 60000]

llh =

         39.5         -132            0
         39.5       -131.5            0
         39.5       -131.5        60000


>> denu = dllh2denu(llh0,llh)

denu =

            0        55510       -242.2
        43008        55629      -389.07
        43415        56153        59611
```

m-file dllh2denu.m

```
function denu = dllh2denu(llh0,llh)
% CONSTANTS
%---------
a = 6378137;
b = 6356752.3142;
e2 = 1 - (b/a)^2;
% Location of reference point in radians
%---------------------------------------
phi = llh0(1)*pi/180;
lam = llh0(2)*pi/180;
h = llh0(3)
% Location of data points in radians
%-----------------------------------
dphi= llh(:,1)*pi/180 - phi;
dlam= llh(:,2)*pi/180 - lam;
dh = llh(:,3) - h;
```

```
% Some useful definitions
%------------------------
tmp1 = sqrt(1-e2*sin(phi)^2);
cl = cos(lam);
sl = sin(lam);
cp = cos(phi);
sp = sin(phi);
% Transformations
%----------------
de = (a/tmp1+h)*cp*dlam - (a*(1-e2)/(tmp1^3)+h)*sp.*dphi.*dlam
+cp.*dlam.*dh;
dn = (a*(1-e2)/tmp1^3 + h)*dphi + 1.5*cp*sp*a*e2*dphi.^2 +
sp^2.*dh.*dphi + ...
0.5*sp*cp*(a/tmp1 +h)*dlam.^2;
du = dh - 0.5*(a-1.5*a*e2*cp^2+0.5*a*e2+h)*dphi.^2 - ...
0.5*cp^2*(a/tmp1 -h)*dlam.^2;
denu = [de, dn, du];
```

CHAPTER 5: CANSAT HARDWARE AND FIRMWARE DEVELOPMENT

5.1 INTRODUCTION

The mission considered for the present CanSat under consideration is a data gathering mission as presented in Chapter 2. This chapter will present the inference between mbed and the different transducers connect to it. The MEMS transducer considered in this mission is listed in Table 5-1. The appendices written in the last column of the table it is for the full technical details of the corresponding transducer.

Table 5-1. Basic hardware for the Can-Sat Development

Component	Model	Note
2-Axis Gyro	LPR530AL	Appendix F
3 Axis Accelerometer	MMA7361LC	Appendix B
Pressure Sensor	DSP 1000	Appendix D
GPS Module	GT-723F	Appendix E

5.2 SYSTEM ARCHITECTURE

Figure 5-1 shows the different interfaces between different hardware and the microcontroller. In this chapter hardware implementation and firmware developments is presented only for GPS, accelerometer, gyroscope, and pressure and temperature sensor.

5.3 GPS

Chapter 4 provides a complete presentation to the GPS systems the GPS employed here is GT-723F, shown in Figure 5-2. Only hardware implementation and firmware class for the present GPS will be presented. Figure 5-3 shows the GPS-mbed circuit diagram. The power source for GPS is taken from the mbed VU-5V output pin. General purpose LI-PO 7.4V, 1300 mAh Li-Po Power pack is connected to the circuit with the required regulator to provide 5 V and high current for servo and other on-board electronics where the mbed output current is not enough. The jumper JP1 in the circuit serve as hardware trigger for starting of the data gathering and storage from different

transducer. Software trigger can also be employed using accelerometer reading as the value of acceleration goes to high during a rocket launch.

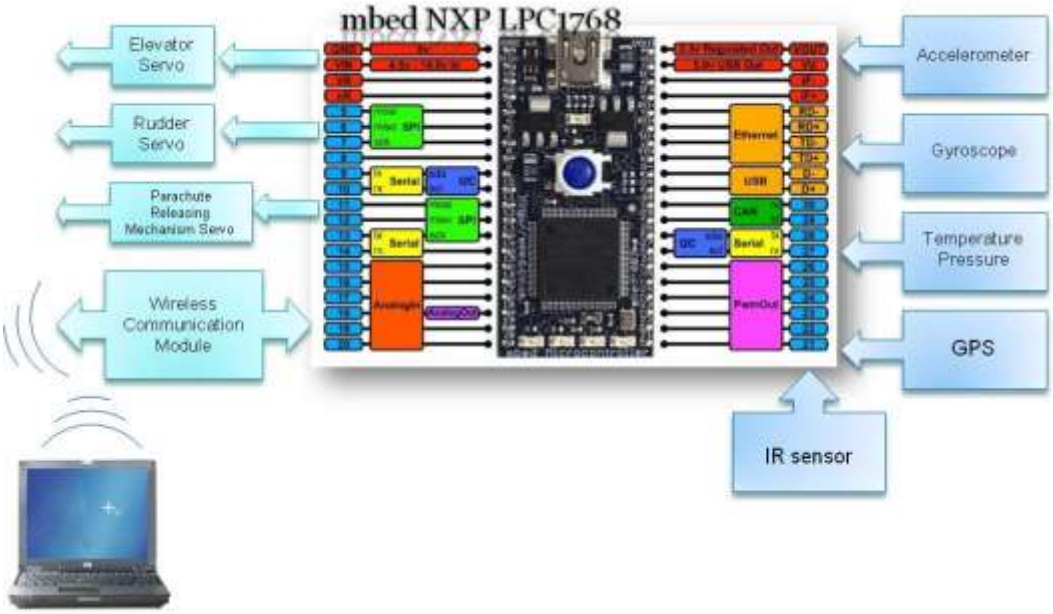


Figure 5-1 Hardware architecture of the CanSat

Figure 5-2 GPS GT-723F

The main program of mbed to interface with GPS is listed below along with the ccp and ***h*** file needed to create the GPS class library as described in section 3.8 of chapter 3.

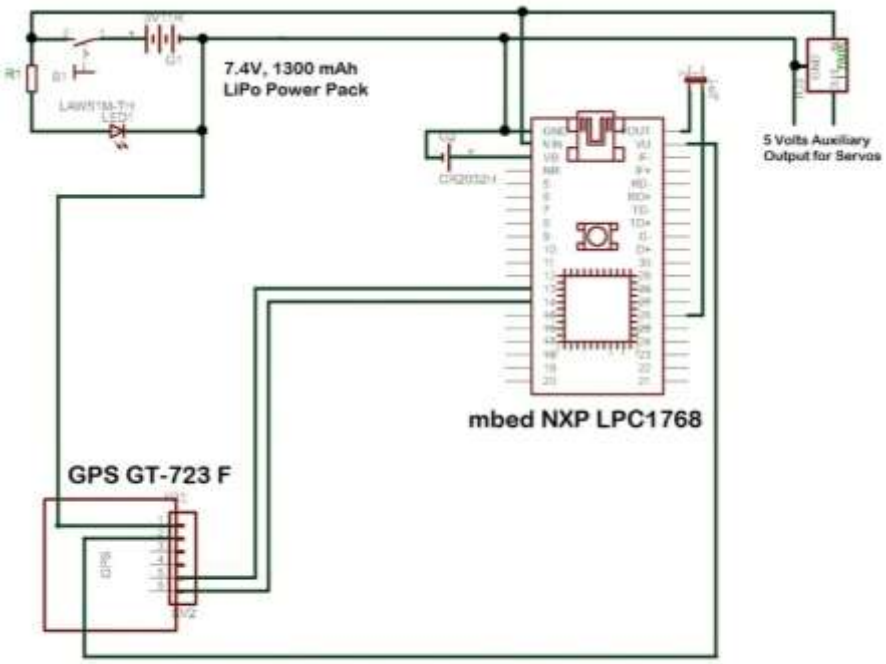


Figure 5-3 GPS circuit diagram

```
// Define the Libraries used in the program

#include "mbed.h"
#include "math.h"
#include "myGPS.h"

float temp;

// Define the Input and Output pins
// --------------------------------
// Serial port for PC access
// -------------------------
Serial pc(USBTX, USBRX);
//
// GPS pin assignment
// ------------------
myGPS GPS(p13,p14);

int No_Sat;
float Direction_GPS;
float Speed_GPS;
float Latitude;
float Longitude;
float Altitude;
float T_Angle;


Ticker Send_data;
```

```
int main()
{   pc.baud(9600);
    GPS.pc = &pc;

    Send_data.attach(&Print_data,1);

    while(1)
    {

    // endless loop for continuous reading of GPS data

    }
}

void Print_data()
{
        GPS.update();
        No_Sat=GPS.Satnum;
        Direction_GPS=GPS.Course;
        Speed_GPS=GPS.Speed;
        Latitude=GPS.Sec_Lat;
        Longitude=GPS.Sec_Lon;
        Altitude=GPS.Altitude;
        T_Angle=GPS.Theta;

        //RTC access
        time_t seconds=time(NULL);
        char buf[32];
        strftime(buf, sizeof(buf), "%X", localtime(&seconds));



        //Print to PC
        pc.printf("Time:, %s ,",buf);
        pc.printf("GPS :, %2d, %.2f, %.2f, %.2f, %.2f, %.2f,
",No_Sat,GPS.Course, GPS.Speed , GPS.Sec_Lat, GPS.Sec_Lon,GPS.Altitude);


}
```

The CPP code

```
include "myGPS.h"
void myGPS::setTarget(float a,  float b)
{
    T_Lat = a;
    T_Lon = b;
}
void myGPS::update()
{
    read_nmea();
    chop_message(nmea1);
    present_array();
```

```
    chop_message(nmea2);
    present_array();
}
myGPS::myGPS(PinName tx,PinName rx) :ser(tx,rx)
 {
    T_Lon = T_Lat= 0;
 }

 //--------------------read nmea from GPS unit--------------------
 void myGPS::read_nmea(void) {

//    char nmeabuff[100];
//    for(int i = 0; i< 5;i++)

    do{
     myGPS::ser.scanf("%s",nmea1);
    }while(!((nmea1[2]=='P') and (nmea1[3]=='G') and (nmea1[4]=='G') and
(nmea1[5]=='A')));
   // pc->printf("%s\r\n",nmea1);

     do{
     myGPS::ser.scanf("%s",nmea2);
    }while(!((nmea2[2]=='P') and (nmea2[3]=='R') and (nmea2[4]=='M') and
(nmea2[5]=='C')));
   // pc->printf("%s\r\n",nmea2);

 }

 //---------------- chop the nmea message separated by comma's-----------
 int myGPS::chop_message(char nmea[]) {
     for (int k=0; k<MAX_STR; k++) {          // just to make sure thet the
char array is set to 0x00
         for (int l=0; l<STR_LEN; l++) {
             gp[k][l]= 0x00;
         };
     }
     int strcnt=0;                            // string counter
     int strpos=0;                            // position inside the string
    for (int k=0; k < MESS_LEN; k++) {
         if (nmea[k] == '*') {                // detect end of message is
found
             gp[strcnt][strpos]= 0x00;        // close the string with 0x00
             return 0;                        // the work is done, end of this
function
        }
         if (nmea[k] == 0x2c) {              // detect the comma
             if (strpos == 0) {
                 gp[strcnt][0]= 'E';         // comma at position zero, string
must be empty
                 gp[strcnt][1]= 'm';         // comma at position zero, string
must be empty
                 gp[strcnt][2]= 'p';         // comma at position zero, string
must be empty
```

```
                gp[strcnt][3]= 't';        // comma at position zero, string
must be empty
                gp[strcnt][4]= 'y';        // comma at position zero, string
must be empty
                gp[strcnt][5]= 0x00;       // comma at position zero, string
must be empty
            } else {
                gp[strcnt][strpos]= 0x00;   // end the previous string
             }
             strcnt += 1;                   // increment to the next string
             strpos =0;                     // start at position zero
         } else {
             gp[strcnt][strpos]= nmea[k];    // add char to string
             strpos += 1;
         }
     }
     return 0;
 }

 // --------------- Dump only $GPGGA on the screen-----------------
 void myGPS::present_array() {
 // --------------- Dump only $GPRMC on the screen-----------------
        if ((gp[0][2]=='P') and (gp[0][3]=='R') and (gp[0][4]=='M') and
(gp[0][5]=='C')) {
             myGPS::Speed=(gp[7][0]-0x30)*100+(gp[7][1]-0x30)*10+(gp[7][2]-
0x30)+(gp[7][4]-0x30)/10.0; //10.0 giveg the floating point decimal
             myGPS::Course=(gp[8][0]-0x30)*100+(gp[8][1]-0x30)*10+(gp[8][2]-
0x30)+(gp[8][4]-0x30)/10.0;

      }

    if ((gp[0][2]=='P') and (gp[0][3]=='G') and (gp[0][4]=='G') and
(gp[0][5]=='A')) {
              int Lat_deg=(gp[2][0]-0x30)*10+(gp[2][1]-0x30);
             int Lat_min=(gp[2][2]-0x30)*10+(gp[2][3]-0x30);
             float Lat_sec=((((gp[2][5]-0x30)*1000+(gp[2][6]-
0x30)*100+(gp[2][7]-0x30)*10+(gp[2][8]-0x30)))/1000.000);
             myGPS::Sec_Lat=Lat_deg*60*60+Lat_min*60+Lat_sec;

             int Lon_deg=(gp[4][0]-0x30)*100+(gp[4][1]-0x30)*10+gp[4][2]-
0x30;
             int Lon_min=(gp[4][3]-0x30)*10+(gp[4][4]-0x30);
             float Lon_sec=((((gp[4][6]-0x30)*1000+(gp[4][7]-
0x30)*100+(gp[4][8]-0x30)*10+(gp[4][9]-0x30)))/1000.000);
             myGPS::Sec_Lon=Lon_deg*60*60+Lon_min*60+Lon_sec;
#define PI 3.141592
             myGPS::Theta=(PI-atan2((Sec_Lon-T_Lon),(Sec_Lat-T_Lat)))*180/PI;
            myGPS::Satnum = (gp[7][0]-0x30)*10+(gp[7][1]-0x30);
           myGPS:: Altitude = (gp[9][0]-0x30)*10000+(gp[9][1]-
0x30)*1000+(gp[9][2]-0x30)*100+(gp[9][3]-0x30)*10+(gp[9][4]-0x30)+(gp[9][6]-
0x30)*0.1;

     }
}
```

The h code

```
include "mbed.h"

#define MESS_LEN    100     // maximum message length
#define MAX_STR     20
#define STR_LEN     40

class myGPS
{
public:
    myGPS(PinName tx,PinName rx);   // when you define the instance, you can
access this function;
    //~myGPS();                       // when you deleted th

    float Sec_Lon;
    float Sec_Lat;
    float Course;
    float Speed;
    float Theta;
    int Satnum;
    float Altitude;

    Serial ser;

    void setTarget(float, float);
    void update();
    Serial *pc;

private:
    float T_Lon,T_Lat;
    char nmea1[100];
    char nmea2[100];

 void read_nmea(void) ;
 int chop_message(char []);
  void present_array() ;
   char gp[MAX_STR][STR_LEN];
};
```

5.4 ACCELEROMETER

Accelerometers, shown in Figure 5-4, are sensors and instruments for measuring, displaying and analyzing acceleration and vibration. They can be used on a stand-alone basis, or in conjunction with a data acquisition system. Accelerometers are available in many forms. They can be raw sensing elements, packaged transducers, or a sensor system or instrument, incorporating features such as totalizing, local or remote display and data recording. Accelerometers can have from one to three axes of measurement, the multiple axes typically being orthogonal to each other. These devices work on many

operating principles. The most common types of accelerometers are piezoelectric, capacitance, null-balance, strain gauge, resonance, piezoresistive and magnetic induction [3-2].

There are several physical processes that can be used to develop a sensor to measure acceleration. In applications that involve flight, such as aircraft and satellites, accelerometers are based on properties of rotating masses. In the industrial world, however, the most common design is based on a combination of Newton's law of mass acceleration and Hooke's law of spring action.

The circuit diagram is shown in Figure 5-5 and the main, CPP and h codes are listed below

Figure 5-4 Accelerometer MMA7361LC

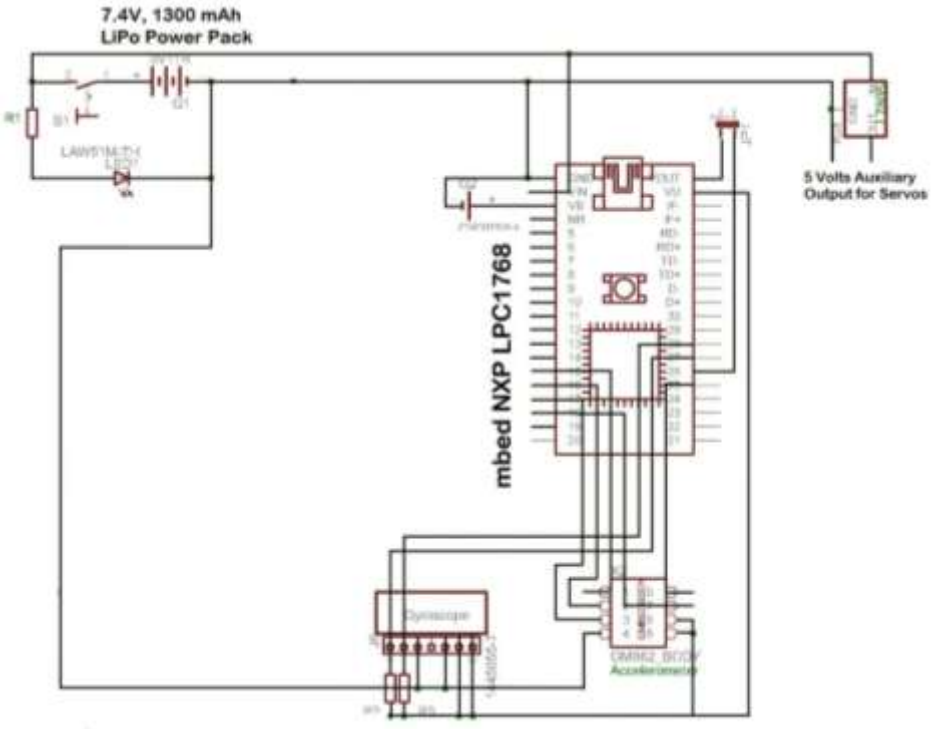


Figure 5-5 Accelerometer and gyroscope circuit diagram

```
// Define the Libraries used in the program
#include "mbed.h"
#include "math.h"
#include "myAccelero.h"

// Define the Input and Output pins

// Serial port for PC access
Serial pc(USBTX, USBRX);


float temp;

// Accelerometer pin assignment
AnalogIn x(p15);
AnalogIn y(p16);
AnalogIn z(p17);
DigitalIn Zero_g(p18);

//
myAccelero accel(p15,p16,p17); //x,y,z,0g
 float X_g;
 float Y_g;
 float Z_g;

Ticker myticker;
Ticker Send_data;

void update()
{
    X_g=accel.getAccelX();
    Y_g=accel.getAccelY();
    Z_g=accel.getAccelZ();

 }

void Print_data()
{
        //RTC access
        time_t seconds=time(NULL);
        char buf[32];
        strftime(buf, sizeof(buf), "%X", localtime(&seconds));

        //Print to PC
        pc.printf("Time:, %s ,",buf);
        pc.printf("Accel:, %.2f, %.2f, %.2f, ",X_g,Y_g,Z_g);
}

int main()
{   pc.baud(9600);

    accel.setScale(myAccelero::SCALE_6G);
    Send_data.attach(&Print_data,1);
```

```
   // Endless loop
   while(1)
    {

    }

}
```

The cpp code

```
#include "mbed.h"
#include "myAccelero.h"

myAccelero::myAccelero(PinName xoutPin,PinName youtPin,PinName zoutPin):
xout(xoutPin),yout(youtPin),zout(zoutPin){/*,ZeroGDetect(ZeroGDetectPin),Zero
G(ZeroGDetectPin)*/
    scale=0.8;
    }

float myAccelero::getAccel(){
    float x=getAccelX();
    float y=getAccelY();
    float z=getAccelZ();
    return sqrt(x*x+y*y+z*z);

}

float myAccelero::getAccelX(){
    return((xout*3.3)-1.65)/0.206;//-0.12;
}
float myAccelero::getAccelY(){
    return((yout*3.3)-1.65)/0.206;//-0.11;
}
float myAccelero::getAccelZ(){
    return((zout*3.3)-1.65)/0.206;//+0.04;
}

float myAccelero::getTiltX(){
    float x=getAccelX();
    float y=getAccelY();
    float z=getAccelZ();
    float a=sqrt(x*x+y*y+z*z);
    return asin(x/a)*180/ 3.141;
}

float myAccelero::getTiltY(){
    float x=getAccelX();
    float y=getAccelY();
    float z=getAccelZ();
    float a=sqrt(x*x+y*y+z*z);
    return asin(y/a)*180/ 3.14;
}

float myAccelero::getTiltZ(){
```

```
    float x=getAccelX();
    float y=getAccelY();
    float z=getAccelZ();
    float a=sqrt(x*x+y*y+z*z);
    return asin(z/a)*180/ 3.14;
}

void myAccelero::setScale(Scale scale){
    switch (scale){
        case SCALE_1_5G:
            this->scale=0.8;
            break;
        case SCALE_6G:
            this->scale=0.206;
            break;

}
}

/*bool myAccelero::detectedZeroG(){
    return zeroGDetect;
}

void myAccelero::setZeroGDetectListner(T* t, void(T::*func)(void)){
    zeroG.rise(func);
}

template<typename T>void myAccelero::setzeroGDetectListner(T*
t,void(T::*func)(void)){
    zeroG.rise(t,func);
}
*/
```

The h code

```
#ifndef myAccelero_H
#define myAccelero_H
#include "mbed.h"
class myAccelero{
public:
    myAccelero(PinName xoutPin,PinName youtPin,PinName zoutPin);
    enum Scale{SCALE_1_5G,SCALE_6G};
    float getAccel();
    float getAccelX();
    float getAccelY();
    float getAccelZ();
    float getTiltX();
    float getTiltY();
    float getTiltZ();
    //bool detectedZeroG();
    //void setZeroGDetectListner(void(*func)(void));
    //template<typename T>void setZeroGDetectListner(T* t,
void(T::*func)(void));
    void setScale(Scale scale);
```

```
private:
    AnalogIn xout,yout,zout;
    //DigitalIn ZeroGdetect;
    //InterruptIn zeroG;
    float scale;
};
#endif
```

5.5 GYROSCOPE

Gyroscopes, shown in Figure 5-6, are used in various applications to sense either the angle turned through by a vehicle or structure (displacement gyroscopes) or, more commonly, its angular rate of turn about some defined axis (rate gyroscopes). The sensors are used in a variety of roles such as:

- Flight path stabilization
- Autopilot feedback
- Sensor or platform stabilization
- Navigation.

Figure 5-6 Gyroscope LPR530AL

The circuit diagram is shown in Figure 5-5 and the main, CPP and h codes are listed below

```
// Define the Libraries used in the program
#include "mbed.h"
#include "math.h"
#include "myGyro.h"
// Define the Input and Output pins

// Serial port for PC access
Serial pc(USBTX, USBRX);

float temp;

// Gyro sensor pin assignment
myGyro Gyro(p28,p27);
```

```
Ticker myticker;
Ticker Send_data;

void update()
{
    Gyro.update();
}

void Print_data()
{
        //RTC access
        time_t seconds=time(NULL);
        char buf[32];
        strftime(buf, sizeof(buf), "%X", localtime(&seconds));
        //Print to PC
        pc.printf("Time:, %s ,",buf);
        pc.printf("Gyro:, %.2f, %.2f, %.2f,%.2f, ",Gyro.x,Gyro.y,Gyro.z);
}

int main()
{   pc.baud(9600);
    Gyro.pc = &pc;
    Gyro.dt = 0.2;
    Gyro.initialize();// Automatic initialization of gyro at the bigining,
Warning "Keep the gyro at rest at the beginning"

    myticker.attach(&update,Gyro.dt);
    Send_data.attach(&Print_data,1);
    // Endless loop
    while(1)
    {

    }

}
```

The cpp code

```
#include "myGyro.h"

myGyro::myGyro(PinName a,PinName b) : gyro(a,
b),omega(0,0,0),Xaxi(1,0,0),Yaxi(0,1,0),Zaxi(0,0,1)
{
//omega(0,0,0),Xaxi(1,0,0),Yaxi(0,1,0),Zaxi(0,0,1)
}
myGyro::~myGyro()
{
    ;
}


void myGyro::initialize()
```

```
{
     x0 = gyro.getGyroX();
     y0 = gyro.getGyroY();
     z0 = gyro.getGyroZ();


    for(int i = 0 ; i < 99; i++)
    {
        x0 += gyro.getGyroX();
        y0 += gyro.getGyroY();
        z0 += gyro.getGyroZ();
     }
       x0/=100;
       y0/=100;
       z0/=100;
       //pc->printf("avarage gyro values: %d %d %d\r\n",x0,y0,z0);
}

void myGyro::update()
{
    int xn,yn,zn;
    xn=gyro.getGyroX();
    yn=gyro.getGyroY();
    zn=gyro.getGyroZ();

    x=(xn-x0)/14.375;
    y=(yn-y0)/14.375;
    z=(zn-z0)/14.375;
   // pc->printf("gyro raw data : %d %d %d\r\n",xn,yn,zn);
    myGyro::transform();
}

void myGyro::transform()
{
   omega.set( x,y,z );
   float o = sqrt(omega%omega);

   omega = omega*(1./ (o==0?1:o));      // if o == 0 then 1 else o
   o *= 2.;

   Xaxi =(Xaxi-omega*(Xaxi%omega))*cos(toRad(o)*dt)
            +(omega*Xaxi)*sin(toRad(o)*dt)
            +omega*(omega%Xaxi);
    Xaxi = Xaxi*(1./sqrt(Xaxi%Xaxi));

   Yaxi =(Yaxi-omega*(Yaxi%omega))*cos(toRad(o)*dt)
            +(omega*Yaxi)*sin(toRad(o)*dt)
            +omega*(omega%Yaxi);
    Yaxi = Yaxi*(1./sqrt(Yaxi%Yaxi));

    Zaxi = Xaxi*Yaxi;
 }
```

The h code

```
#include "mbed.h"
#include "math.h"


#define toRad(a) (a/180*3.1415)
#define toDeg(a) (a*180/3.1415)
class myGyro
{
public:
    myGyro(PinName,PinName);        // constructor
    ~myGyro();                      // destructor

    void update();                    // update gyro data
    void transform();
    void initialize();

    class vector
    {
    public:
        void set(float x,float y, float z)
        {
            vector::x=x;
            vector::y=y;
            vector::z=z;
        }
        vector(float x,float y, float z)
        {
            vector::x=x;
            vector::y=y;
            vector::z=z;
        }
        vector operator +(const vector& V)
        {
            return vector(x+V.x,y+V.y,z+V.z);
        }
        vector operator -(const vector& V)
        {
            return vector(x-V.x,y-V.y,z-V.z);
        }
        vector operator -()
        {
            return vector(-x,-y,-z);
        }
        vector operator *(const vector& V)
        {
            return vector(y*V.z-z*V.y,z*V.x-x*V.z,x*V.y-y*V.x);
        }
        vector operator *(const float& A)
        {
            return vector(x*A,y*A,z*A);
        }
        float operator %(const vector& V)
        {
```

```
            return x*V.x+y*V.y+z*V.z;
        }
        float x,y,z;
    };


    vector Xaxi;
    vector Yaxi;
    vector Zaxi;

    float x,y,z;
    float dt;
    Serial *pc;
private:
    int x0,y0,z0;
    ITG3200 gyro;

    vector omega;
};
```

5.6 PRESSURE AND TEMPERATURE TRANSDUCERS

Barometric pressure, shown in Figure 5-7, are invariably used for height measurement in aircraft. As supplementary navigation aids, they are widely used for restricting the growth of errors in the vertical channel of an inertial navigation system.

Figure 5-7 MEMS pressure sensor

The altitude measurement based on pressure is a relative measurement, i.e. it compares pressures at different places. This can be done with one pressure sensor and, in that case, is also sensitive to barometric pressure changes. Where accurate altitude information is needed over longer times, barometric reference pressure is often used.

Assuming a constant temperature gradient of dT/dH in accordance with the 1976 US Standard Atmosphere, the altitude H as a function of pressure P is obtained:

$$H = T_0/(-dT/dH) * [1 - (P/P_0)^{(-dT/dH)*R/g}] \qquad (5\text{-}1)$$

Inserting $dT/dH = -6.5°C/km$, $T_0 = 288.15K$ (+15°C), $P_0 = 101325Pa$, $g = 9.82m/s^2$ and $R = 287.052m^2/s^2/K$ one gets:

$$H = 44.33km * [1 - (P/101325Pa)^{0.19}] \qquad (5\text{-}2)$$

The circuit diagram is shown in Figure 5-8 and the main is listed below. Online SCP1000.lib class from mbed.org web site can be used for the present temperatue and pressure transducer.

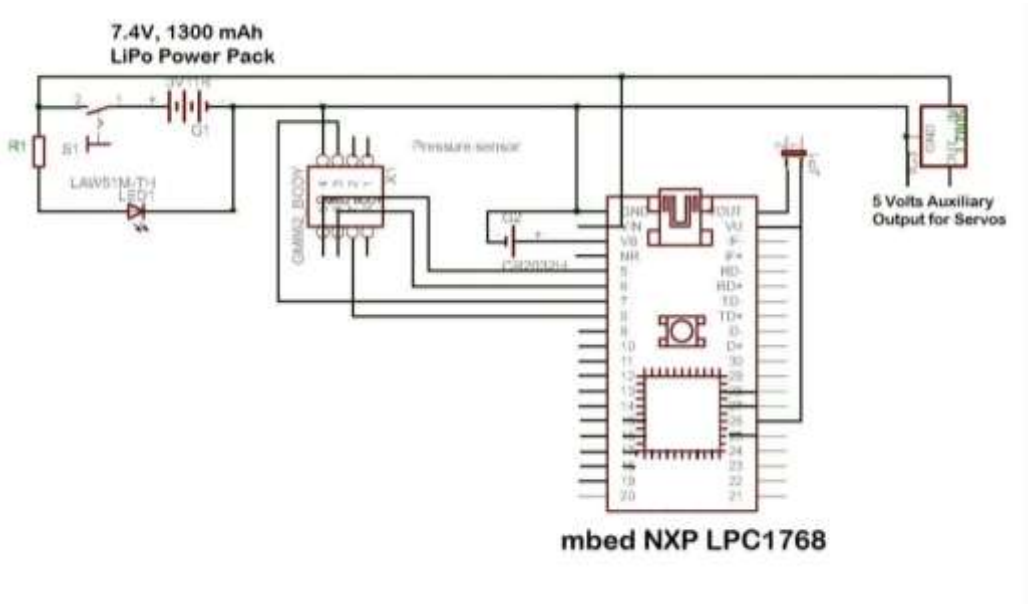


Figure 5-8 Pressure and Temperature circuit diagram

```
// Define the Libraries used in the program
#include "mbed.h"
#include "math.h"

// Define the Input and Output pins

// Serial port for PC access
Serial pc(USBTX, USBRX);

// Pressur sensor pin assignment
```

```
SCP1000 scp1000(p5,p6,p7,p8);

float temp;
int press;


Ticker myticker;
Ticker Send_data;

void update()
{
    temp=scp1000.readTemperature();
    press=scp1000.readPressure();
}

void Print_data()
{

        //RTC access
        time_t seconds=time(NULL);
        char buf[32];
        strftime(buf, sizeof(buf), "%X", localtime(&seconds));

        temp=scp1000.readTemperature();
        press=scp1000.readPressure();

        //Print to PC
        pc.printf("Time:, %s ,",buf);
        pc.printf("Temp:, %.2f, Press:, %d\r\n",temp,press);

}

int main()
{   pc.baud(9600);

    Send_data.attach(&Print_data,1);

    while(1)
    {

    }
}
```

5.7 SYSTEM INTEGRATION

Figure 5-9 present the circuits diagram for all the MEMS sensors described above. The main program is listed below to print the reading through the PC serial port through the HyperTerminal. The code will be extended to include sending the data to the ground station PC in Chapter 7 section 7.5.

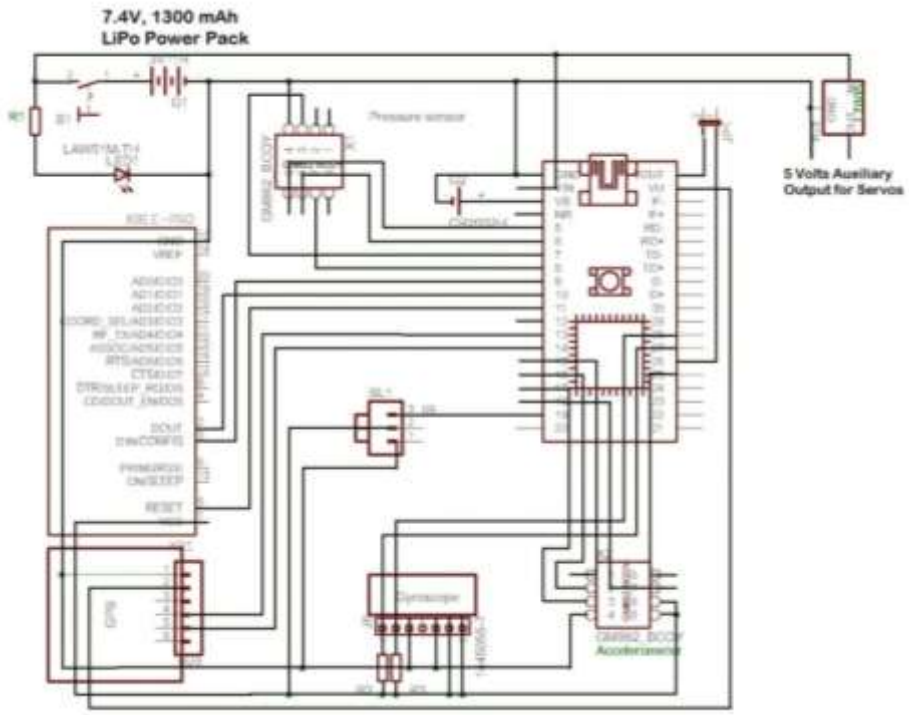


Figure 5-9 Complete circuit Diagram

```
// Define the Libraries used in the program
#include "mbed.h"
#include "SCP1000.h"
#include "math.h"
#include "myGPS.h"
#include "myGyro.h"
#include "myAccelero.h"
// Define the Input and Output pins

// Serial port for PC access
Serial pc(USBTX, USBRX);

// Pressur sensor pin assignment
SCP1000 scp1000(p5,p6,p7,p8);
float temp;
int press;
// Gyreo sensor pin assignment
//ITG3200 Gyro(p28,p27);
myGyro Gyro(p28,p27);

// Accelerometer pin assignment
AnalogIn x(p15);
AnalogIn y(p16);
AnalogIn z(p17);
DigitalIn Zero_g(p18);

//GPS pin assignment
```

```
myGPS GPS(p13,p14);
//int i;
int No_Sat;
float Direction_GPS;
float Speed_GPS;
float Latitude;
float Longitude;
float Altitude;
float T_Angle;

// Motor variables
float propeller=0.0;
float rudder=0.5;
float elevator=0.5;
float para_sw=0.5;


//
myAccelero accel(p15,p16,p17); //x,y,z,0g
 float X_g;
 float Y_g;
 float Z_g;

Ticker myticker;
Ticker Send_data;

void update()
{
    Gyro.update();
    X_g=accel.getAccelX();
    Y_g=accel.getAccelY();
    Z_g=accel.getAccelZ();
    temp=scp1000.readTemperature();
    press=scp1000.readPressure();


 }

void Print_data()
{
        GPS.update();
        No_Sat=GPS.Satnum;
        Direction_GPS=GPS.Course;
        Speed_GPS=GPS.Speed;
        Latitude=GPS.Sec_Lat;
        Longitude=GPS.Sec_Lon;
        Altitude=GPS.Altitude;
        T_Angle=GPS.Theta;


        //RTC access
        time_t seconds=time(NULL);
        char buf[32];
        strftime(buf, sizeof(buf), "%X", localtime(&seconds));
```

```
        temp=scp1000.readTemperature();
        press=scp1000.readPressure();
       // wait(0.01);

        //Print to PC
        pc.printf("Time:, %s ,",buf);
        pc.printf("GPS :, %2d, %.2f, %.2f, %.2f, %.2f, %.2f,
",No_Sat,GPS.Course, GPS.Speed , GPS.Sec_Lat, GPS.Sec_Lon,GPS.Altitude);
        pc.printf("Gyro:, %.2f, %.2f, %.2f,%.2f, ",Gyro.x,Gyro.y,Gyro.z);
        pc.printf("Accel:, %.2f, %.2f, %.2f, ",X_g,Y_g,Z_g);
        pc.printf("Motor:, %.2f,%.2f,%.2f,%.2f,
",propeller,rudder,elevator,para_sw);
        pc.printf("Temp:, %.2f, Press:, %d\r\n",temp,press);
}
```

CHAPTER 6: PARACHUTE DESIGN

6.1 INTRODUCTION

Parachute is a crucial element during the CanSat mission. Achieving both the desired descent rate and stable decent are key parameters in parachute design, See Figure 6-1. This chapter will introduce the basic theory and technique for designed a parachute for a CanSat mission.

Figure 6-1 stable descending of parachute with CanSat during the Japanese high school CanSat Championship "CanSat Koshien" in 2011.

6.2 FORCES ACTING ON THE PARACHUTE

Forces and moments acting on a parachute canopy may be presented in several ways. The two most frequently used methods, as shown in Figure 6-2, are with forces oriented to the axis of flight and with forces oriented to the axis of the parachute canopy. The tangential force, T: and the normal force, N, are calculated as follows

$$T = C_T S q \quad (6\text{-}1)$$

and

$$N = C_N S q \quad (6\text{-}2)$$

Where
CT = tangential force coefficient, dimensionless
CN = normal force coefficient, dimensionless

The resultant force, R, and the moment, M, are shown in the. The airflow fixed system is preferred for aerodynamic performance calculations, and the parachute fixed system for wing stress calculations. The aerodynamic coefficients C_L> C_D, C_T, C_N, and C_M can easily be determined in wind-tunnel measurements.

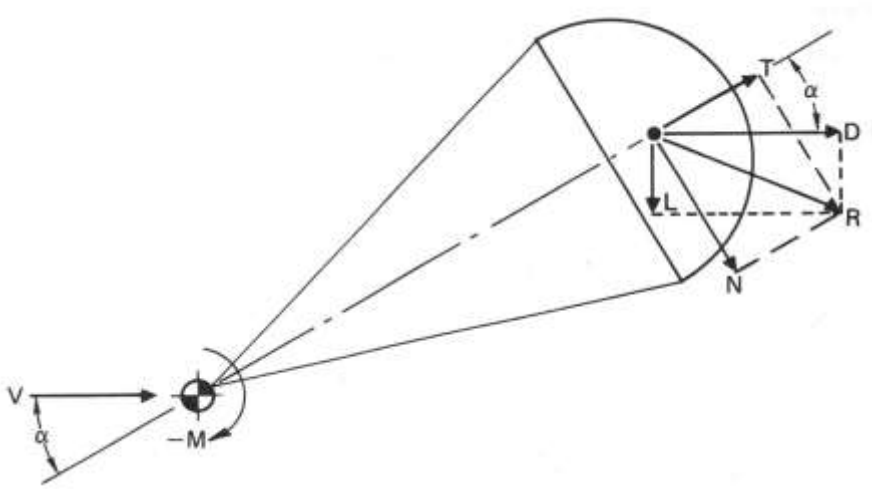


Figure 6-2 Forces Acting on a Parachute [6-1].

Wind-tunnel installations frequently measure normal and tangential force instead of lift and drag. If α, T, and N are known, the drag, D, can be calculated

$$D = T \cos \alpha + N \sin \alpha \qquad (6\text{-}3)$$

For a parachute with an angle of attack, α equal to zero, the drag force and the tangential force are synonymous. Figure 6-3 shows the coefficients C_T, C_D, and C_M versus angle of attack for stable and unstable parachutes.

The coefficient presentation shows two interesting facts. The slope of the moment coefficient curve, $dC_M/d\alpha$ for the unstable parachute is positive between -25 degrees and +25 degrees; this is, by definition, destabilizing. This parachute will oscillate approximately ± 25 degrees. The slope of the moment coefficient, $dC_M/d\alpha$, for the stable parachute is negative over the total angle of attack; this is, by definition, stabilizing. The steeper the negative $dC_M/d\alpha$ slope, the greater is the stabilizing tendency of the parachute, and the better is its damping capability against unstabilizing forces such as sudden gusts of wind.

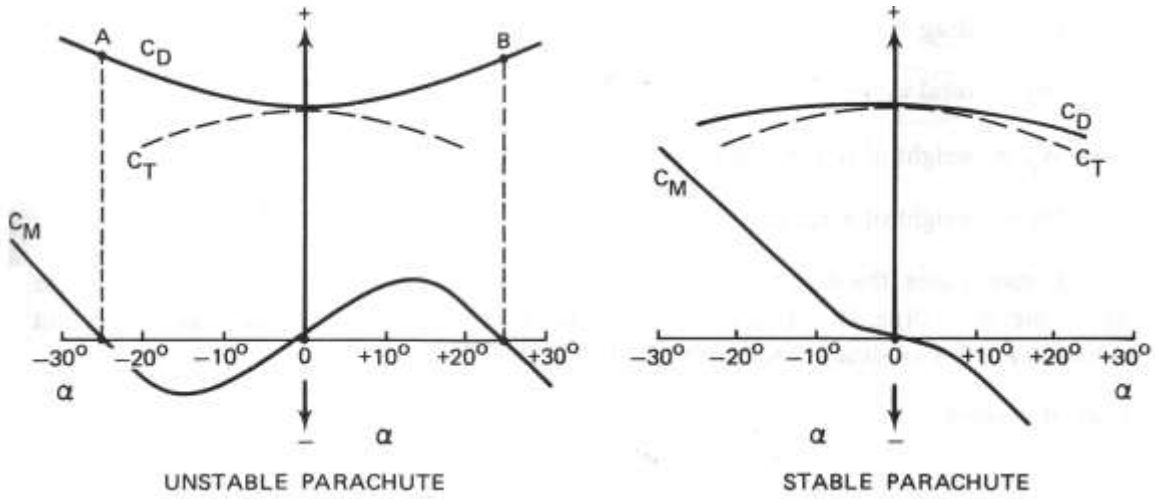


Figure 6-3 Coefficients C_D, C_T, and C_M Versus Angle of Attack, α, for Stable and Unstable Parachute [6-1].

6.3 EQUILIBRIUM OF FORCES IN STEADY DESCENT

A stable parachute in unaccelerated descent has an equilibrium between the total drag of the parachute and the load, D_T, and the weight of the load and the parachute assembly, W_T (Figure 6-4). For steady descent

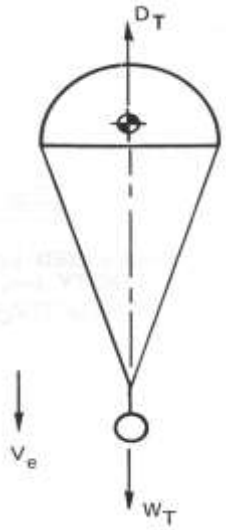


Figure 6-4 Forces Acting on a Parachute in Steady Descent

$$D_T = W_T \text{ or } D_p + D_L = W_p + W_L$$

Where

D_T = total drag, N

D_p = drag of parachute,N
D_L = drag of load, N
W_T= total weight, N
W_p= weight of parachute, N
W_L= weight of load, N

In most cases, the drag of the load can be neglected in relation to the large drag of the parachute. With drag, $D = (C_D S)_P \rho/2 v^2$ and $D_T = W_T$, and solving for v, the important equation for rate of descent, v_e, is obtained.

$$V_e = \sqrt{\frac{2W}{SC_D\rho}} \qquad (6\text{-}4)$$

Or in parachute terminology for rate of descent at sea level

$$V_{eo} = \sqrt{\frac{2W}{S_o C_{Do} \rho_o}} \qquad (6\text{-}5)$$

And rate of descent at any altitude

$$V_e = \sqrt{\frac{2W}{S_o C_{Do} \rho_o}} \frac{1}{\sqrt{\rho / \rho_o}} \qquad (6\text{-}6)$$

Where ρ_o is the density at the sea level

In the above equation for the rate of descent, Ve

WT = weight of load and parachute assembly, N
So = canopy surface area, S^2
C_{DO} = parachute drag coefficient related to S_o
ρ = air density at a specific altitude in kg/m^3

During descent from altitude, the parachute system is constantly decelerated because of the increasing air density. This can be ignored for slowly descending main parachutes.

6.4 PARACHUTE CHARACTERISTICS AND PERFORMANCE

Parachute is a crucial element of any CanSat mission. Their performance characteristics must be known and considered in calculating the descent rate. In the early 19208, the circular, flat parachute manufactured from solid cloth was the primary parachute used for the rescue of aviators, for sport jumping, and for airdrop of light loads. In the 19305, the military began using parachutes for the airdrop of troops and cargo and for the landing

deceleration of aircraft. Beginning in the 194Os, parachutes were used for recovery of unmanned aircraft, missiles, ordnance devices, and, later, recovery of manned and unmanned spacecraft.

Constructed Shape: The Plan and Profile columns define the constructed diameter and the cross section of the parachute canopy.

D_c, the constructed diameter of the canopy, can be obtained from the drawing of the specific parachute.

Nominal Diameter: **Do**, the nominal diameter of the parachute, can be calculated from the total canopy surface area, S_o, including the area of the vent and all other openings:

$$D_o = \sqrt{4S_o / \pi} = 1.1284\sqrt{S_o} \qquad (6\text{-}7)$$

Inflated Shape: D_p, the projected diameter of the inflated parachute canopy, is best calculated from the projected or inflated canopy area, S_p. as measured in wind-tunnel tests. The projected diameter of a parachute varies with parachute type, canopy porosity, suspension-line length, velocity, and canopy design. A large projected diameter, D_p, will generally result in a large drag coefficient, C_{Do}. The ratio of projected diameter to nominal diameter, D_p/D_o, is an indication of the drag effectiveness of the parachute design; the larger the projected diameter in relation to the normal diameter, the larger the drag coefficient.

Drag Coefficient: C_{Do} is the drag coefficient related to the total canopy surface area, S_o, The drag coefficient indicates how effectively a parachute canopy produces drag with a minimum of cloth area, thereby minimizing weight and volume.

Opening-Force Coefficient. C_x, the opening-force coefficient, indicates the difference between the instantaneous opening force and the steady drag force at constant speed. This condition, called the infinite mass case, is obtained in wind-tunnel tests and in parachute applications with high canopy loadings, $W/(C_dS)_p$. as exemplified by aircraft deceleration parachutes and first-stage drogue chutes.

Average Angle of Oscillation. The angle of oscillation is measured in wind-tunnel tests or during free-flight tests. Oscillation on most solid textile parachutes varies with size and descent velocity. Large parachutes oscillate less than small parachutes. Unstable parachutes tend to descend in an oscillating mode at rates of descent in excess of 7.62 m/s, glide at descent General Application. The general application column in Table 6-1 through 6-5 indicates the primary use of the particular parachute type. In descent rates below 4.572 m/s, and mix glide and oscillation during medium rates of descent.

General Application. The general application column in Table 6-1 through 6-5 indicates the primary use of the particular parachute type.

S_w in Table 6-4 is the wetted upper canopy surface area on gliding parachutes.

S_w/S_o in Table 6-4 is the ratio of the upper surface area to total cloth area, including all ribs and stabilizer panels.

Table 6-1 Solid Textile Parachute

TYPE	CONSTRUCTED SHAPE		$\frac{D_c}{D_o}$	INFLATED SHAPE $\frac{D_p}{D_o}$	DRAG COEF C_{D_o} RANGE	OPENING FORCE COEF C_X (LINE MASS)	AVERAGE ANGLE OF OSCILLATION, DEGREES	GENERAL APPLICATION
	PLAN	PROFILE						
FLAT CIRCULAR			1.00	0.67 TO 0.70	0.75 TO 0.80	~1.7	±10 TO ±40	DESCENT, OBSOLETE
CONICAL			0.93 TO 0.95	0.70	0.75 TO 0.90	~1.8	±10 TO ±30	DESCENT, M < 0.5
BICONICAL			0.90 TO 0.95	0.70	0.75 TO 0.92	~1.8	±10 TO ±30	DESCENT, M < 0.5
TRICONICAL POLYCONICAL			0.90 TO 0.95	0.70	0.80 TO 0.98	~1.8	±10 TO ±20	DESCENT, M < 0.5
EXTENDED SKIRT 10% FLAT			0.86	0.66 TO 0.70	0.78 TO 0.87	~1.4	±10 TO ±15	DESCENT, M < 0.5
EXTENDED SKIRT 14.3% FULL			0.81 TO 0.85	0.66 TO 0.70	0.75 TO 0.90	~1.4	±10 TO ±15	DESCENT, M < 0.5
HEMISPHERICAL			0.71	0.66	0.62 TO 0.77	~1.6	±10 TO ±15	DESCENT, M < 0.5, OBSOLETE
GUIDE SURFACE (RIBBED)			0.63	0.62	0.28 TO 0.42	~1.2	0 TO ±2	STABILIZATION, DROGUE, 0.1 < M < 1.5
GUIDE SURFACE (RIBLESS)			0.66	0.63	0.30 TO 0.34	~1.4	0 TO ±3	PILOT, DROGUE, 0.1 < M < 1.5
ANNULAR			1.04	0.94	0.85 TO 0.95	~1.4	<±6	DESCENT, M < 0.5
CROSS			1.15 TO 1.19	0.66 TO 0.72	0.60 TO 0.85	1.1 TO 1.2	0 TO ±3	DESCENT, DECELERATION

Table 6-2 Slotted Parachute

TYPE	CONSTRUCTED SHAPE PLAN	CONSTRUCTED SHAPE PROFILE	$\frac{D_c}{D_o}$	INFLATED SHAPE $\frac{D_p}{D_o}$	DRAG COEF C_{D_o} RANGE	OPENING FORCE COEF C_X (INF MASS)	AVERAGE ANGLE OF OSCILLATION, DEGREES	GENERAL APPLICATION
FLAT (FIST) RIBBON			1.00	0.67	0.45 TO 0.50	~1.05	0 TO ±3	DROGUE, DESCENT, DECELERATION, OBSOLETE
CONICAL RIBBON			0.95 TO 0.97	0.70	0.50 TO 0.55	~1.05	0 TO ±3	DESCENT, DECELERATION, 0.1 < M < 2.0
CONICAL RIBBON (VARIED POROSITY)			0.97	0.70	0.55 TO 0.60	1.05 TO 1.30	0 TO ±3	DROGUE, DESCENT, DECELERATION, 0.1 < M < 2.0
RIBBON [1] (HEMISFLO)			0.62	0.62	0.30 [1] TO 0.46	1.00 TO 1.30	±2	SUPERSONIC, DROGUE, 1.0 < M < 3.0
RINGSLOT			1.00	0.67 TO 0.70	0.56 TO 0.65	~1.05	0 TO ±5	EXTRACTION, DECELERATION, 0.1 < M < 0.9
RINGSAIL			0.84	0.69	0.75 TO 0.85	~1.10	±5 TO ±10	DESCENT, M < 0.5
DISC-GAP-BAND			0.73	0.65	0.52 TO 0.58	~1.30	±10 TO ±15	DESCENT, M < 0.5

[1] FOR SUPERSONIC APPLICATION, SEE SECTION 5.8.

Table 6-3 Rotating Parachute

TYPE	CONSTRUCTED SHAPE PLAN	CONSTRUCTED SHAPE PROFILE	$\frac{D_c}{D_o}$	INFLATED SHAPE $\frac{D_p}{D_o}$	DRAG COEF C_{D_o} RANGE	OPENING FORCE COEF C_X (INF MASS)	AVERAGE ANGLE OF OSCILLATION, DEGREES	GENERAL APPLICATION
ROTAFOIL			1.0	~0.90	0.85 TO 0.99	1.05	0 TO ±2	DROGUE, D_o < 7
VORTEX RING			1.9	N/A	1.5 TO 1.8	1.1 TO 1.2	0 TO ±2	DESCENT, SMALL D_o
SANDIA RFD			1.0	~0.9	1.25	1.1	0 TO ±2	DROGUE

Table 6-4 Maneuverable (Gliding) Parachute

TYPE	CONSTRUCTED SHAPE PLAN	CONSTRUCTED SHAPE PROFILE	AREA RATIO S_w/S_o	AERODYNAMIC FORCE COEF C_R RANGE	GLIDE RATIO $(L/D)_{MAX}$	GENERAL APPLICATION
TOJO, TU SLOTS, ETC			1.0	0.85 TO 0.90	0.5 TO 0.7	DESCENT
LeMOIGNE (PARACOMMANDER)			1.0	0.90 TO 1.00	1.1	DESCENT
PARAWING (SINGLE KEEL)			1.0	0.90 TO 1.10	2.0 TO 2.5	DESCENT
PARAWING (TWIN KEEL)			1.0	1.00 TO 1.10	2.8[1] TO 3.0	DESCENT
PARAFOIL			0.27	0.75 TO 0.85	2.8[1] TO 3.5	DESCENT
SAILWING			0.80 TO 0.90	N/A	2.8[1] TO 3.5	DESCENT
VOLPLANE			0.60	N/A	2.0[1] TO 3.0	DESCENT

[1] GLIDE RATIO IS AFFECTED BY ASPECT RATIO, AR, AND CANOPY LOADING, W/S_o

Table 6-5 Balloon Type decelerator

TYPE	CONSTRUCTED SHAPE PLAN	CONSTRUCTED SHAPE PROFILE	D_c/D_o	INFLATED SHAPE D_p/D_o	DRAG COEF C_{D_p} RANGE	OPENING FORCE COEF C_X (INF MASS)	AVERAGE ANGLE OF OSCILLATION, DEGREES	GENERAL APPLICATION
BALLUTE			0.51	0.51	0.51[1] TO 1.20	~1.05	<±1	STABILIZATION, DROGUE, 0.8 < M < 4

6.5 PARACHUTE SIMULATION DURING DESCENDING

The partial differential equation which governs the parachute descending motion can be written as:

$$\frac{dv}{dt} = g - \frac{C_D}{m} v \qquad (6\text{-}7)$$

$$\frac{dy}{dt} = v$$

Where
m = mass of parachutist
c = drag coefficient
g = gravitational acceleration

The results of the simulation are shown in Figure 6-5. The parachute is dropped from 500 meters altitude and has a mass of 1Kg and CD of 0.75 after opening. The chute opens after 3 second in this simulation. The terminal (descent) velocity is 5.211 which in an excellent agreement with the value computed using equation 6-4 which is 5.21176 m/sec. The total descending time for this CanSat is about 100 seconds.

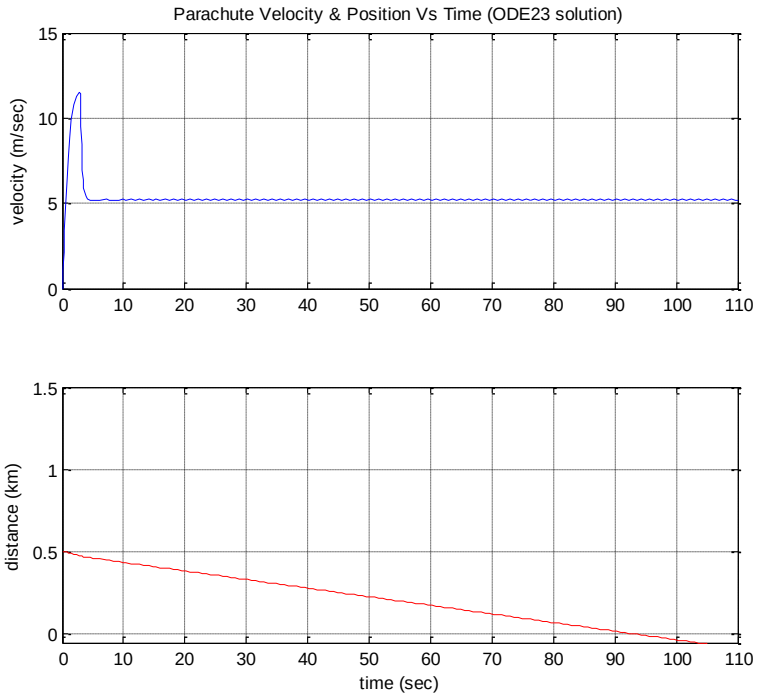


Figure 6-5 Parachute velocity and position versus time; parachute mass = 1kg, Cd after opening = 0.75; parachute opening after 3 sec.

The MATLAB program and its associated function are listed below.

```
%
%    PCHUT.M    Simple Simulation of a Falling Parachutist
%
```

```
%   In this Matlab file a discontinuous change in the drag coefficient is used to
%   simulate the opening of the parachute at some user-defined time after the
%   initiation of free fall.
%
%   However, in this case, the numerical integration of the first order ODEs is
%   performed with a standard ODE solver built into Matlab - the ODE23 routine.
%   ODE23 uses a higher order approximation for the derivative terms and it uses
%   a technique referred to as "adaptive step control" to adjust the integration
%   interval during the calculation to guarantee that a desired error criterion is
%   satisfied.  This makes life much easier, because now we do not have to
%   explicitly discretize the original ODEs and we do not have to worry about
%   picking a suitable integration step that gives good accuracy.  Instead, we
%   simply give ODE23 information about the original ODEs, specify the desired
%   error limit and let it go from there.
%
%   In this case the defining eqns. are:
%   dv/dt = g - (0.5 Rho A Cd/m)v^2     and      dy/dt = v
%   where    m = mass of parachutist
%            Cd = drag coefficient
%            g = gravitational acceleration
%            Rho = air density
%            A   = reference area
%
%   Note that these can be written as
%      dz(1)/dt = z(2)     and      dz(2)/dt = g - (c/m)z(2)
%   where   z(1) = y    and    z(2) = v
%
%   This second form, which is referred to as standard state form, is the
%   one used by the ODE23 routine.  These equations are included in the ODE
%   function file PCHUTA.M.
%
%   File prepared by J. R. White, UMass-Lowell (original Sept. 1997)
%
%
%   getting started
      clear all,   close all,   nfig = 0;
%
%   set properties
      global m g c1 c2 topen
      m = 0.25;        % mass of parachutist (kg)
      d = 1;           % parachute diameter (m)
      A = 0.25*pi*d*d; % parachute reference area (m^2)
      Rho = 1.225      % air density (kg/m^3)
      Cd  = 0.15       % Drag coefficient w/o chute
      c1 = 0.5*Rho*Cd*A;       % nominal drag coefficient with no chute  (kg/sec)
      c2 = 5*c1;       % nominal drag coefficient with chute open  (kg/sec)
      g  = 9.8;        % acceleration due to gravity (m/s^2)
      pos = 0.5;       % initial height (km)
      tol = 1.0e-3;    % max relative error allowed in solution
%
%   set time interval of interest, initial conditions, and time to open chute
      to = 0;   tf = 600.0;  % initial and final times (sec)
      zo = [0 0]';  % initial position (m) and velocity (m/s) in vector form
      disp('  Initially you are at a height of 500 m above ground.')
      topen = input('  Input time for parachute to open (< 600 sec) : ');
%
%   call ODE23 to solve problem (original ODEs are in function file PCHUT3A.M)
      options  = odeset('RelTol',tol);
      [t,z] = ode23('pchut4a',[to tf],zo,options);
%
%   plot velocity and position vs time
      nfig = nfig+1;   figure(nfig)
      subplot(2,1,1), plot(t,z(:,2),'b-'), grid, axis([to tf 0 10]);
```

```
      title('CanSat Velocity & Position Vs Time (ODE23 solution)')
      ylabel('velocity (m/sec)')
      subplot(2,1,2), plot(t,pos-z(:,1)/1000,'r-'), grid
      axis([to tf -pos/10 pos+1]);
      ylabel('distance (km)'),xlabel('time (sec)')
%
%   end of simulation
%
```

The function PCHUTA.M is listed below

```
%
%   PCHUTA.M   Function for ODE to Evaluate Falling Parachutist Equations
%                      (called from ODE23 in PCHUT.M )
%
%   The defining eqns are (in standard state form):
%         dz(1)/dt = z(2)
%         dz(2)/dt = g - (c/m)z(2)
%
%   where the value of c is time dependent (piecewise constant)
%
%   File prepared by J. R. White, UMass-Lowell (original Sept. 1997)
%
      function zp = pchuta(t,z)
      global m g c1 c2 topen
      if t < topen   c = c1;   else    c = c2;   end
      zp = zeros(length(z),1);
      zp(1) = z(2);
      zp(2) = g - (c/m)*z(2)*z(2);
%
%   end of function
%
```

CHAPTER 7: CANSAT LAUNCHERS

6.1 INTRODCUTION

Usually Can-Sat can be launched using one of the following methods:

1. Model rocket
2. RC model airplane
3. Balloon
4. High raise building

The advantages and disadvantage of each launching option will be disscusse and techniques to design Water rocket laucher and ballon for launching will be given

…. To be contined

CHAPTER 8: GROUND STATION DEVELOPMENT

7. 1 INTRODUCTION

This chapter describes how you can communicate the CanSat to a ground station and send command to the CanSat as well as receive the collected data from it. MATLAB development environment used to develop the ground station GUI software. A commercial ly available RF communication module called XBEE-PRO having a range of 1500 meters (line of sight) is presented. The configuration procedures to for the RF communication module are presented in a step by step way.

7.2 GROUND STATION SYSTEM ARCHITECTURE

Figure 7-1 shows the basic elements to RF communication between the CanSat and ground station PC. The communication cycle consists of:

1. CanSat with XBEE-PRO module connected with mbed as will be described later (full hardware specification of XBEE-PRO is presented in Appendix G)
2. Computer
3. XBEE-Explorer unit
4. XBEE-PRO module connected to the computer through the XBEE-Explorer and USB-Cable

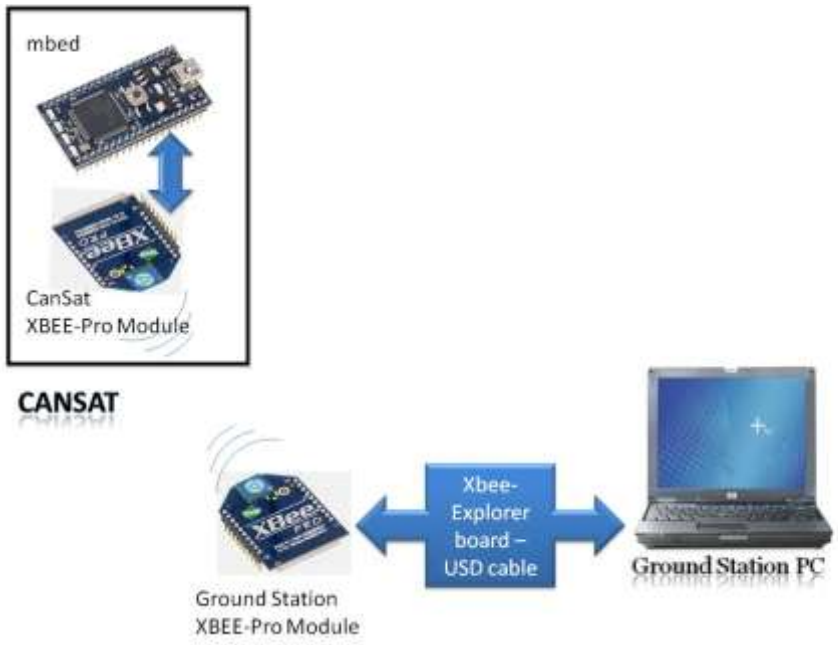


Figure 7-1 Ground Station System Architecture

The two XBEE-PRO modules are similar and before connection to both the CanSat and PC they must be paired first and this will be described the next section. Hardware and firmware development for the XBEE-PRO with mbed will be presented. Finally the software for the ground station PC using MATLAB PC will be presented.

7.3 PARING THE RF-COMMUNICATION MODULES (XBEE-PRO)

Radio module / radio modem is used to transmit the status/position of a CanSat to a ground control station.

XBee-Pro modules from Digi International can be used in two modes: In one mode they represent a mesh network. This is however not used by most users. In the mostly used "wireless serial link" mode, they forward data on their serial interface via wireless to a paired module.

This allows sending serial data wireless. Make sure to configure your mbed with the same settings as your ground control station PC and the Xbee-Pro modules.

7.3.1 RECOMMENDED SETTINGS

For the 2.4 GHz v2.0/v2.5 modules:

- Baud: 9600
- Parity: N (none)
- Data bits: 8
- Stop bits: 1
- Abbreviation for this mode: 57600 8N1

7.3.2 INSTALLATION

The simplest way to install the Xbee-Pro modules is to use X-CTU (XCTU) software [7-1]. This is a windows software from the manufacturer of the XBee-Pro modules and its very simple to use. Then just connect theUSB XBee Explorer, described above, to your windows computer. Now you can flash your XBee-Pro modules. If the program asks you to reset the XBee-Pro, you have to connect the RST pin to the GND pin of the XBee Explorer. You can make this with tweezers.

7.3.3 SETTING UP A SERIAL CONNECTION

For setting up a serial connection with the XBee-Pro you have to flash one module with the coordinator AT firmware and the other with the router or end-device AT firmware.

7.3.4 COMM SETTINGS

1. Choose the first available USB serial port (e.g. COM15)
2. Leave all the default options, set the speed to 9600 baud
3. Click on Test/Query
4. You should get the modem firmware revision

7.3.5 FLASHING THE CANSAT MODULE

1. Put in the first module in the USB XBee Explorer. This will be the module for the end device.
2. Start the X-CTU program and go to the modem configuration.
3. Select the function set ZNET 2.5 ROUTER/END DEVICE AT. The Mesh XBee modules (XBP24-DM) have slightly differently named settings.
4. Set the pan id to 123 (or another number, must be the same as the pan id of the coordinator). The PAN ID setting for XBP24-DM is named “Modem VID”.
5. Set the Node Identifier to “CANSAT” or another name.
6. Set the Baudrate you want to use. The ground station and the CANSAT module should have the same Baudrate. Also you need to know what Baudrate the current attached XBee-Pro has. 9600 is the maximum Baudrate setting for bidirectional transfers to work correctly. At a higher Baudrate setting, transmission can only be done in one direction.
7. Then write the firmware to the module

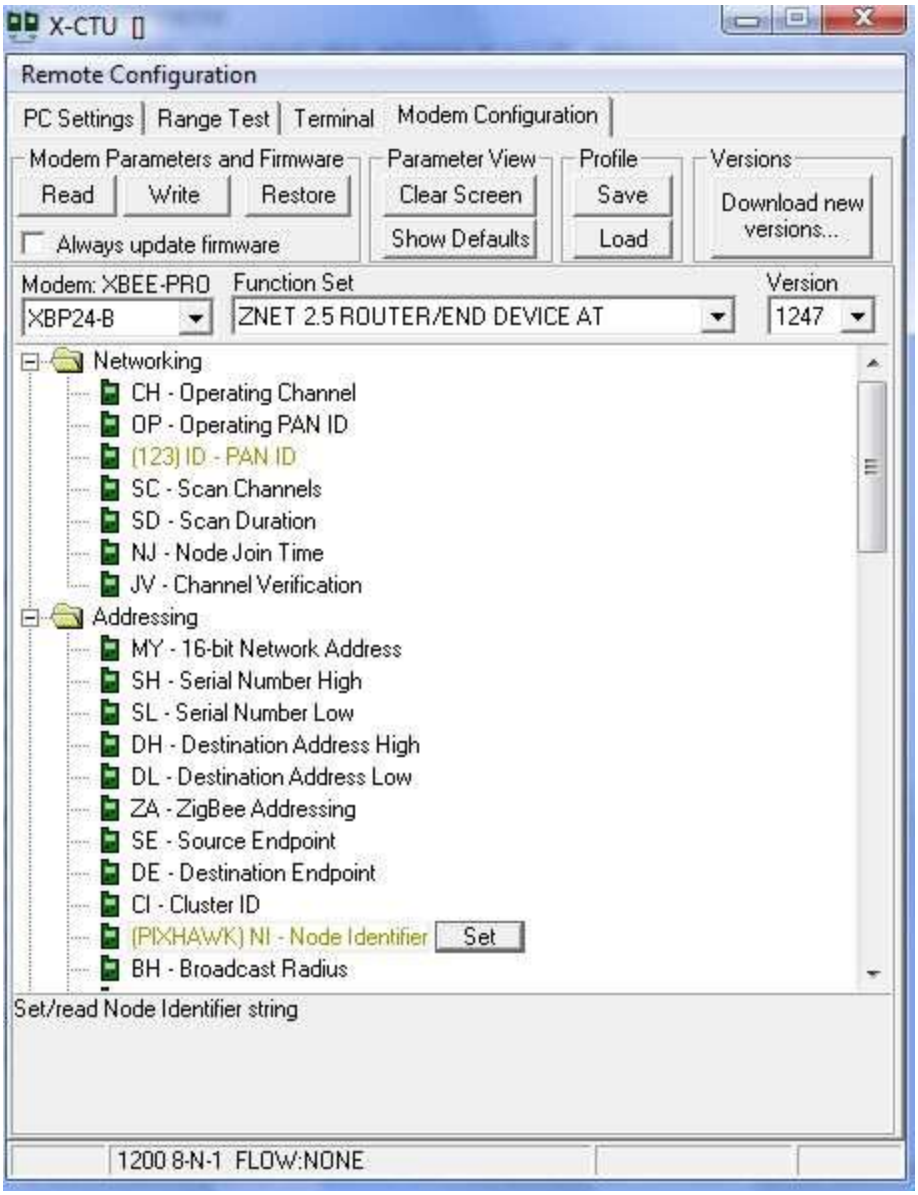


Figure 7-2 Flashing the CanSat Module

7.3.6 FLASHING THE GROUND STATION PC MODULE

1. Now put the groundstation module in the USB XBee Explorer.
2. Select the function set ZNET 2.5 COORDINATOR DEVICE AT. The Mesh XBee-Pro modules (XBP24-DM) have slightly differently named settings.
3. Set the pan id to 123 (or another number, must be the same as the pan id of the end device). The PAN ID setting for XBP24-DM is named “Modem VID”.
4. Set the Destination Address Low (DL) to FFFF.
5. Set the Node Identifier to “GROUNDSTATION” or another name.
6. Set the Baudrate you want to use. The ground station and the MAV module should have the same baudrate. Also you need to know what Baudrate the current attached XBee-Pro has. 9600 is the maximum Baudrate setting for

bidirectional transfers to work correctly. At a higher Baudrate setting, transmission can only be done in one direction.

7. Then write the firmware to the module.

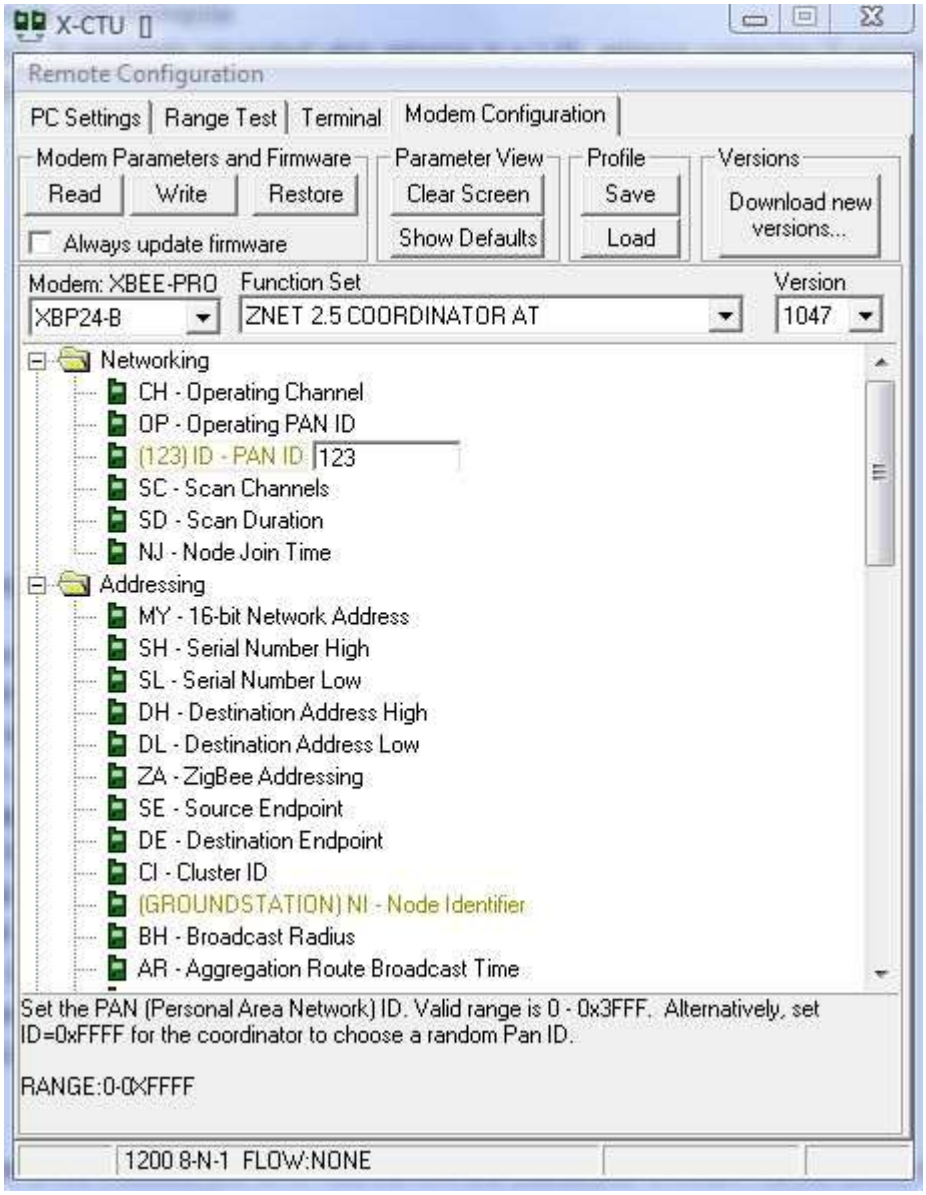


Figure 7-3 Flashing Ground Station PC Module

7.3.7 CONNECTING THE XBEE MODULE TO YOUR MAV

Now you have to connect the CANSAT module to your board. You just need four wires. One for ground, one for high level, one for RX and TX. Connect the DIN pin from the XBee-Pro module to the TX pin of your board and connect the DOUT pin with the RX pin.

7.4 INTERFACING MBED WITH XBEE-PRO

Figure 7.4 shows the wire connection of mbed to XBEE-PRO

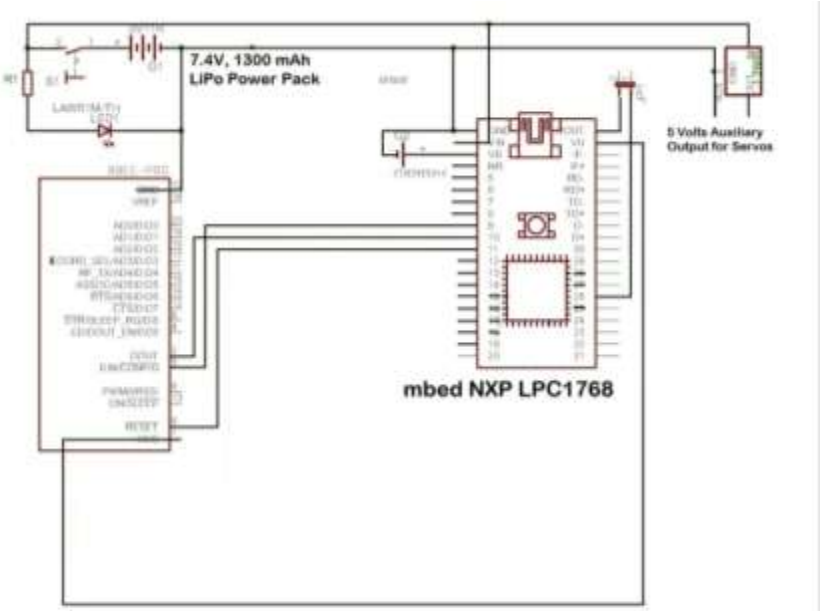


Figure 7-4 Wire Connection between mbed and XBEE-PRO

The following is a simple mbed code to send the word “Hello World” to ground station PC and the local PC connected to mbed, for development, upon pressing the reset button of the mbed. HyperTerminal should be open on the ground station PC and the port connected to the XBEE-Explore should be assigned, Figure 7.5 shows the output upon pressing the reset button on the ground station PC.

```
// Define the Libraries used in the program
#include "mbed.h"

// Define the Input and Output pins

// Serial port for PC access
Serial pc(USBTX, USBRX);
```

```
// Xbee Tranceiver pin Assignment
Serial xbee(p9,p10);
DigitalOut rst1(p11);  // Digital reset for the XBee, 200ns for
reset
int main()
{   pc.baud(9600);
    xbee.baud(9600);

        //Print to PC
        pc.printf("Hello World\n");


       //MatLab output
        xbee.printf("Hello World");
}
```

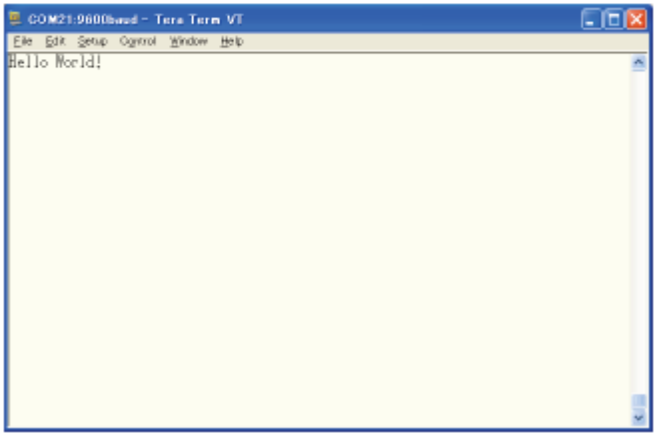


Figure 7-5 HyperTerminal application shows a “Hello World” written using another PC and send through the wireless module XBEE-PRO.

7.5 SYSTEM INTEGRATION

In the section a revised version of the main code written in section 5.7 in chapter 5 to include sending the data to the cround station PC through XBEE-PRO RF communication module is presented. The revised code will be as follows:

```
// Define the Libraries used in the program
#include "mbed.h"
#include "SCP1000.h"
#include "math.h"
#include "myGPS.h"
#include "myGyro.h"
#include "myAccelero.h"
// Define the Input and Output pins
```

```
// Serial port for PC access
Serial pc(USBTX, USBRX);

// Pressur sensor pin assignment
SCP1000 scp1000(p5,p6,p7,p8);
float temp;
int press;
// Gyreo sensor pin assignment
//ITG3200 Gyro(p28,p27);
myGyro Gyro(p28,p27);

// Accelerometer pin assignment
AnalogIn x(p15);
AnalogIn y(p16);
AnalogIn z(p17);
DigitalIn Zero_g(p18);

//GPS pin assignment
myGPS GPS(p13,p14);
//int i;
int No_Sat;
float Direction_GPS;
float Speed_GPS;
float Latitude;
float Longitude;
float Altitude;
float T_Angle;


// Xbee Tranceiver pin Assignment
Serial xbee(p9,p10);
DigitalOut rst1(p11);  // Digital reset for the XBee, 200ns for
reset



//
myAccelero accel(p15,p16,p17); //x,y,z,0g
 float X_g;
 float Y_g;
 float Z_g;

Ticker myticker;
Ticker Send_data;

void update()
{
    Gyro.update();
    X_g=accel.getAccelX();
    Y_g=accel.getAccelY();
    Z_g=accel.getAccelZ();
    temp=scp1000.readTemperature();
    press=scp1000.readPressure();
```

```
}

void Print_data()
{
        GPS.update();
        No_Sat=GPS.Satnum;
        Direction_GPS=GPS.Course;
        Speed_GPS=GPS.Speed;
        Latitude=GPS.Sec_Lat;
        Longitude=GPS.Sec_Lon;
        Altitude=GPS.Altitude;
        T_Angle=GPS.Theta;


        //RTC access
        time_t seconds=time(NULL);
        char buf[32];
        strftime(buf, sizeof(buf), "%X", localtime(&seconds));



        temp=scp1000.readTemperature();
        press=scp1000.readPressure();
       // wait(0.01);

        //Print to PC
        pc.printf("Time:, %s ,",buf);
        pc.printf("GPS :, %2d, %.2f, %.2f, %.2f, %.2f, %.2f,
",No_Sat,GPS.Course, GPS.Speed , GPS.Sec_Lat,
GPS.Sec_Lon,GPS.Altitude);
        pc.printf("Gyro:, %.2f, %.2f, %.2f,%.2f,
",Gyro.x,Gyro.y,Gyro.z);
        pc.printf("Accel:, %.2f, %.2f, %.2f, ",X_g,Y_g,Z_g);
        pc.printf("Temp:, %.2f, Press:, %d\r\n",temp,press);

       //MatLab output
        xbee.printf("%s, ",buf);
        xbee.printf("%2d, %.2f, %.2f, %.2f, %.2f, %.2f, ",No_Sat,
Direction_GPS, Speed_GPS ,Latitude,Longitude,Altitude);
        xbee.printf("%.2f, %.2f, %.2f, ",Gyro.x,Gyro.y,Gyro.z);
        xbee.printf("%.2f, %.2f, %.2f, ",X_g,Y_g,Z_g);
        xbee.printf("%.2f, %.2f\r\n",temp,press);


}

int main()
{   pc.baud(9600);
    xbee.baud(9600);
    GPS.pc = &pc;
    Gyro.pc = &pc;
    Gyro.dt = 0.2;
```

```
    Gyro.initialize();// Automatic initialisation of gyro at the
bigining,Warning "Keep the gyro at rest at the begining"

    accel.setScale(myAccelero::SCALE_6G);

    // xBee Initialisation
    rst1   =0;  //Set reset pin to 1
    wait(1);  //Wait at least one millisecond
    rst1   = 1;  //Set reset pin to 1
    wait(1);  //Wait another millisecond

    myticker.attach(&update,Gyro.dt);
    Send_data.attach(&Print_data,1);

    while(1)
    {

            wait(0.1);


    }
}
```

7.6 GROUND STATION SOFTWARE

Figure 7.6 shows an example of GUI software written using Matlab on the ground station PC. The program receives all the data from CanSat and plots it on real-time as well as save it in a log file. The program is written below.

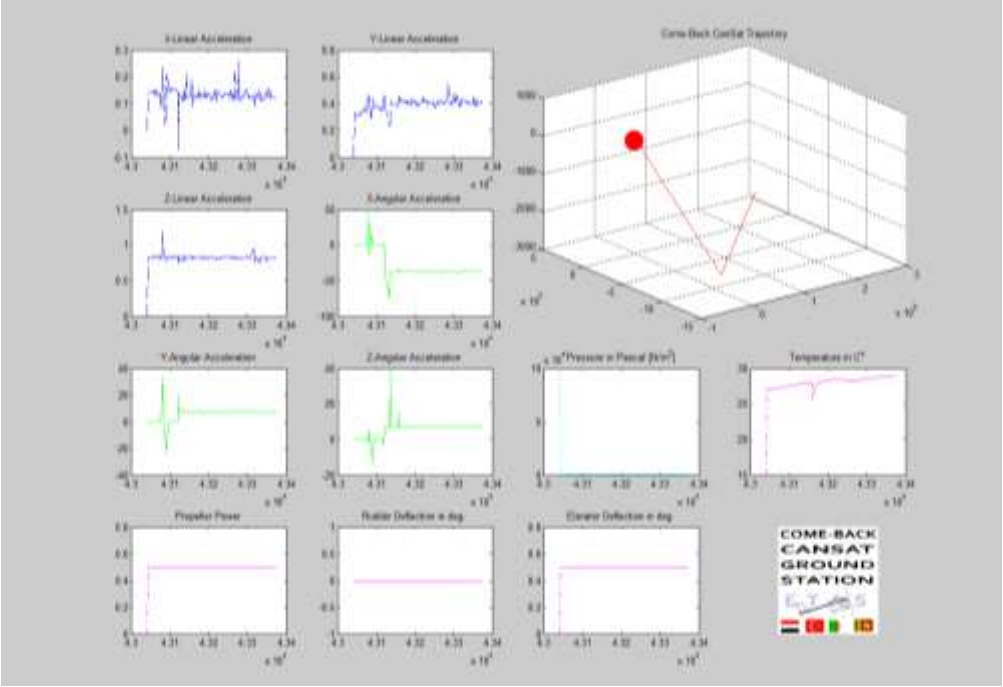


Figure 7-6 Matlab Ground Station GUI Software

```
% --------------------------------------------------------------------
%
% Ground Station Monitoring M-File for CanSat Leadership Training Program
% By: Mohammed Khalil Ibrahim
% On: 4th March 2011
%     Wakayama University, Wakayama, Japan
% --------------------------------------------------------------------
%
%
% First Configure the COM port to receive data from the XBEE and collect
% data
% --------------------------------------------------------------------
%
clear all;
global DATA;
global XBEE;
fp1 = fopen('gs_plot.log', 'w');

Rearth = 6367*1000; % Earth radius in meter [m]

% Initializing;
% ------------
time_old = 0;

% Accelerations
% -------------
AX_old = 0;
AY_old = 0;
AZ_old = 0;

% Angular Velcoity
```

```
% ----------------
WX_old = 0;
WY_old = 0;
WZ_old = 0;

% Pressure
% --------
P_old = 100000;

% Temperature
% -----------
T_old = 15;


% GPS data
% ---------
X_Target = 0;
Y_Target = 0;
Z_Target = 0;
LONG_TARGET = 10*pi/180;   % TARGET POSITION must be in rad.
LATI_TARGET = 10*pi/180;   % TARGET POSITION must be in rad.

X_old = 100;
Y_old = 100;
Z_old = 100;

% Controls
% --------
RDRC_old = 0;
ELVC_old = 0;
PROP_old = 0;
PARA_old = 0;

clc;
fprintf('Communication with CanSat is ready... \n');
A=input('Hit any key to start to start. Press 0 then Enter key to
exit.\n','s');
if strcmp(A,'0')
   break
end;

scrsz = get(0,'ScreenSize');
H = figure('Position',[50 50 scrsz(3)-100 scrsz(4)-150],'Name','Come-Back
CanSat Ground Station','NumberTitle','off');

while 1
      XBEE = 0;
      COMPORT = serial('COM10','BaudRate', 9600,'Parity', 'none','DataBits',8,
'StopBits',1);
      COMPORT.Terminator = 'CR/LF';
      COMPORT.BytesAvailableFcnMode = 'terminator';
      COMPORT.BytesAvailableFcn = @Xbee;
      COMPORT.timeout = 300;
      fopen(COMPORT);
      pause(0.6)
      fprintf('Data from the main program = %s \n',DATA);
      fclose(COMPORT);
```

```
      delete(COMPORT);

      LDATA= size(DATA);
      VAR_START = findstr(DATA, ',');
      NVAR = size(VAR_START);
      NVAR(2)

      % Assign the Varibale
      % hh:mm:ss, GPS:,No. of Sat,Course,Speed, Latitude, Longitude,Altitude,
Gyro:,Wx,Wy,Wz, Accel:, Ax,Ay,Az, Motor:,RDR,ELV,PROP,PARA


      if  NVAR(2) == 18
          TIME_STR = DATA(1:VAR_START(1)-1);
          NSAT_STR = DATA(VAR_START(1)+1:VAR_START(2)-1);
          SPED_STR = DATA(VAR_START(2)+1:VAR_START(3)-1);
          CORS_STR = DATA(VAR_START(3)+1:VAR_START(4)-1);
          LATI_STR = DATA(VAR_START(4)+1:VAR_START(5)-1);
          LONG_STR = DATA(VAR_START(5)+1:VAR_START(6)-1);
          ALTI_STR = DATA(VAR_START(6)+1:VAR_START(7)-1);
          WX_STR =   DATA(VAR_START(7)+1:VAR_START(8)-1);
          WY_STR =   DATA(VAR_START(8)+1:VAR_START(9)-1);
          WZ_STR =   DATA(VAR_START(9)+1:VAR_START(10)-1);
          AX_STR =   DATA(VAR_START(10)+1:VAR_START(11)-1);
          AY_STR =   DATA(VAR_START(11)+1:VAR_START(12)-1);
          AZ_STR =   DATA(VAR_START(12)+1:VAR_START(13)-1);
          RDRC_STR = DATA(VAR_START(13)+1:VAR_START(14)-1);
          ELVC_STR = DATA(VAR_START(14)+1:VAR_START(15)-1);
          PROP_STR = DATA(VAR_START(15)+1:VAR_START(16)-1);
          PARA_STR = DATA(VAR_START(16)+1:VAR_START(17)-1);
          T_STR    = DATA(VAR_START(17)+1:VAR_START(18)-1);
          P_STR    = DATA(VAR_START(18)+1:LDATA(2));

          LTIME = size(TIME_STR)
          if LTIME(2) > 8
            TIME_START = findstr(TIME_STR,':');
            HOURS_STR   = TIME_STR(1:TIME_START(1)-1);
            MINUTES_STR = TIME_STR(TIME_START(1)+1:TIME_START(2)-1);
            SECONDS_STR = TIME_STR(TIME_START(2)+1:LTIME(2));
            time_new =   str2num(SECONDS_STR) + 60 * str2num(MINUTES_STR) +
3600 * str2num(HOURS_STR); % Time in Seconds
          if time_old == 0
             time_old = time_new;
             fprintf('NEW TIME CALCULATED 1\n');
             continue;
             end
          end

          fprintf('New point will be drawn\n');
          time_new
          time_old

          %
          % Write the raw data to a FILE here
          % --------------------------------
```

```
            AX_new = str2num(AX_STR);
            AY_new = str2num(AY_STR);
            AZ_new = str2num(AZ_STR);

            WX_new = str2num(WX_STR);
            WY_new = str2num(WY_STR);
            WZ_new = str2num(WZ_STR);

            P_new  = str2num(P_STR);
            T_new  = str2num(T_STR);

            LATI = str2num(LATI_STR)*pi/(180*60*60);
            LONG = str2num(LONG_STR)*pi/(180*60*60);
            ALTI = str2num(ALTI_STR);

            X_new  = Rearth*(LONG - LONG_TARGET);% CALCULATION IS NEEDED
            Y_new  = Rearth*(LATI - LATI_TARGET);% CALCULATION IS NEEDED
            Z_new  = str2num(ALTI_STR);% CALCULATION IS NEEDED

            RDRC_new = str2num(RDRC_STR);
            ELVC_new = str2num(ELVC_STR);
            PROP_new = str2num(PROP_STR);
            PARA_new = str2num(PARA_STR);

        else
            time_new = time_old;
            AX_new = AX_old;
            AY_new = AY_old;
            AZ_new = AZ_old;

            WX_new = WX_old;
            WY_new = WY_old;
            WZ_new = WZ_old;

            P_new  = P_old;
            T_new  = T_old;

            X_new  = X_old;
            Y_new  = Y_old;
            Z_new  = Z_old;

            RDRC_new = RDRC_old;
            ELVC_new = ELVC_old;
            PROP_new = PROP_old;
            PARA_new = PARA_old;
        end


%
% Decode the data from the stream of received from the COM Port
% -------------------------------------------------------------
%

% Image
% -----
subplot(4,4,16);
rgb = imread('Ground_Station.jpg');
```

```
image(rgb);
axis off          % Remove axis ticks and numbers
axis image        % Set aspect ratio to obtain square pixels
hold on;

%
% Plot the data
% -------------
%
% Check that the time is 0

% GPS Data
% --------
subplot(4,4,[3 4 7 8]);
plot3([X_old X_new], [Y_old Y_new], [Z_old Z_new],'r-');
GRID ON
title('Come-Back CanSat Trajectory');
hold on;
plot3(X_Target, Y_Target,
Z_Target,'ro','MarkerSize',20,'MarkerFaceColor','r');
hold on;

% X-Acceleration
% --------------
subplot(4,4,1);
plot([time_old time_new], [AX_old AX_new],'b--');
title('X-Linear Acceleration');
hold on


% Y-Acceleration
% --------------
subplot(4,4,2);
plot([time_old time_new], [AY_old AY_new],'b--');
title('Y-Linear Acceleration');
hold on

% Z-Acceleration
% --------------
subplot(4,4,5);
plot([time_old time_new], [AZ_old AZ_new],'b--');
title('Z-Linear Acceleration');
hold on;

% Angular Acceleration in x-direction
% -----------------------------------
subplot(4,4,6);
plot([time_old time_new], [WX_old WX_new],'g-');
title('X-Angular Acceleration');
hold on

% Angular Acceleration in y-direction
% -----------------------------------
subplot(4,4,9);
plot([time_old time_new], [WY_old WY_new],'g-');
title('Y-Angular Acceleration');
hold on
```

```
% Angular Acceleration in z-direction
% ----------------------------------
subplot(4,4,10);
plot([time_old time_new], [WZ_old WZ_new],'g-');
title('Z-Angular Acceleration');
hold on;

% Pressure
% --------
subplot(4,4,11);
plot([time_old time_new], [P_old P_new],'c:');
title('Pressure in Pascal [N/{m^2}]');
hold on;

% Temperature
% -----------
subplot(4,4,12);
plot([time_old time_new], [T_old T_new],'m-.');
title('Temperature in {C^o}');
hold on;

% Propeller Power
% ---------------
subplot(4,4,13);
plot([time_old time_new], [PROP_old PROP_new],'m-.');
title('Propeller Power');
hold on;


% Rudder Deflection
% -----------------
subplot(4,4,14);
plot([time_old time_new], [RDRC_old RDRC_new],'m-.');
title('Rudder Deflection in deg.');
hold on;


% Elevator Deflection
% -----------------
subplot(4,4,15);
plot([time_old time_new], [ELVC_old ELVC_new],'m-.');
title('Elevator Deflection in deg.');
hold on;
pause(0.2);


% mtit('ETAS Come-Back Ground Station PC',
'fontsize',18,'xoff',0,'yoff',.03,'color',[0.5 0 0]);

%
% UPDATE
% ------
%
% Acceleration - Acceleromter
% ---------------------------
AX_old = AX_new;
```

```
AY_old = AY_new;
AZ_old = AZ_new ;

% Angular Velcoity; Gyro Data
% ---------------------------
WX_old = WX_new;
WY_old = WY_new;
WZ_old = WZ_new;


% Pressure
% --------
P_old = P_new;

% Temperature
% -----------
T_old = T_new;


% GPS data
% ---------
X_old = X_new;
Y_old = Y_new;
Z_old = Z_new;

% Time
% ----
time_old = time_new;

RDRC_old = RDRC_new;
ELVC_old = ELVC_new;
PROP_old = PROP_new;
PARA_old = PARA_new;

%
%  SAVE THE DATA FILE
%  ------------------


fprintf(fp1,'%f, %f , %f, %f, %f, %f, %f, %f, %f, %f, %f, %f, %f, %f, %f, %f\
n',time_new, X_new, Y_new, Z_new, AX_new, AY_new, AZ_new, WX_new, WY_new,
WZ_new, P_new, T_new, RDRC_new, ELVC_new, PROP_new, PARA_new );

%
% Close the serial port
%
end

fclose(fp1);
```

CHAPTER 9: MECHANICAL CONSTRUCTION

9.1 INTRODUCTION

The volume constraints for the any CanSat design are either 350 ml or 500 ml can with maximum mass of about 1 Kg. A hard- foam was used to fill and support the main PCB with microcontrollers and different MEMS sensors, batteries and Camera inside Can. The RF-communication module and GPS was fixed outside the can and guarded by soft foam. The final Can-Sat was tested several times with parachute had high descent rate which resulted in a shock when touching the ground and it survived the shock.

9.2 STRUCTURAL DESIGN

During mechanical construction of CanSat one must always take into consideration the mass budget and the size of the CanSat. In most of the competitions you need to launch your CanSat from 10-15 (?) cm diameter rockets. In addition to that the mechanical structure of the CanSat must provide an adequate space to contain the vulnerable electronic components. The design should protect it from shocks by absorbing some of the energy that is generated during launch, release and parachute deployment and landing.

The CanSat must have a mass of no more than 1000 g; considering its relatively low weight and the small distance it will possibly fall, a wide range of material could be considered as appropriate to fulfil the requirements, plastics and many different metals would be suitable. 350 ml or 500 ml volume can may be used or you can manufacture your own design cansat from different light-weight metal or composite materials such as glass fabric, carbon composit or alumininum. Aluminium or aluminium alloys provide the rigid structure desired while being light-weight, non-brittle commonly available and cheap.

Figure 9-1 Cansat made of carbon composite (Turkish Air Force Academy Team.

For a CanSat there is no need for thermal insulation if you are not going to send the CanSat to an altitude of more than 1000 m. The temperature will not drop more than 6-7 degrees compare to ground temperature. For higher altitudes, such as she temperature in the stratosphere (216 ◦K) is significantly lower than the temperature on ground. A CanSat that will be sent to such an altitude will be facing low temperature condition which may threaten sensors to work properly. In order for this CanSat to be successful, it must ensure that the electronic components (which are designed to operate at around sea level temperatures) are not damaged and operate as long as possible. This will be achieved by insulating the can to slow the temperature loss.

For the electronics, the PCB board should not be in direct contact with the outer surface fo the CanSat. Applying soft materials, such as foam or some kind of insulating material, on the edges of the frame holding the PCB boards, the shock will be more dispersed. The space remaining will contain sensors and the batteries. The sensors will need to be arranged so that they can still take data. The batteries can be firmly connected to foam and supported by CanSat walls preventing from moving.

For the effective use of the instrumentation, they need to be positioned where they can act correctly. This requires careful consideration of safety, operating temperatures and other environmental factors.

The pressure sensor has a minimum operating temperature of aronud 233°K . This sensor requires access to the outside environment to take true readings of atmospheric pressure. The temperature sensors will operate correctly until 233°K, then the readings are no longer reliable but the sensor will not be damaged. The radio transmitter has a minimum operating temperature of 253 ° K. If the temperature inside the probe is allowed to drop below ~253°K the signal will be affected or interrupted. In

this case the thermal insulator will play a major role, delaying cooling as much as possible.

After CanSat construction some simple temperature and shock resistance test can be performed on it. The results of these tests can then be used to determine whether the design was sufficient CanSat structure. CanSat also should be tested in windy, stormy and icy conditions.

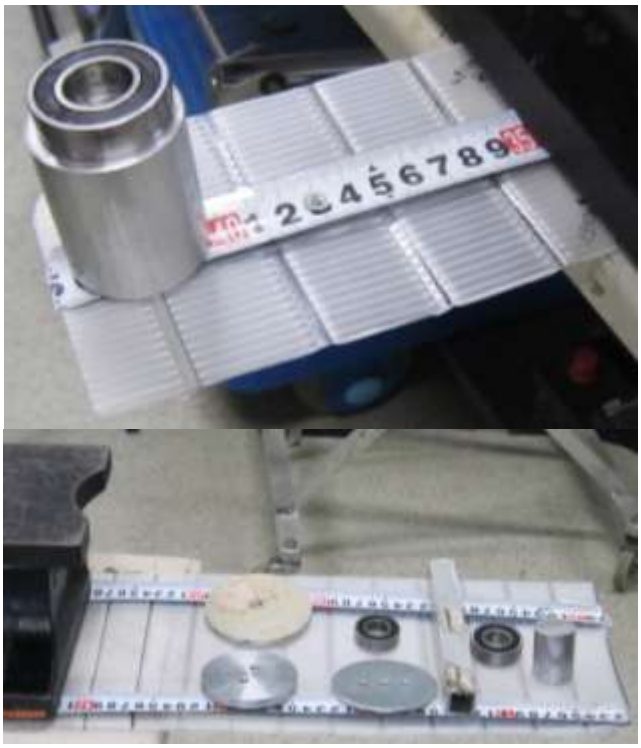


Figure 9-2 Wing Loading Test for a Fixed wing CanSat

For different types of CanSats you need appropriate designs always taking into account mass budget and size. Especially for come back competitons you may use paragliding, rover back or fixed wing types of CanSat. Paragliding CanSat requires direction and control mechanism. The direction in heavy weather may not support the forces of the paraglide properly and the cansat may stall or drifted away. For a rower back you should take into account the rower mechanism, obstacle avoidance system and control in 2D condition. However, for a fixed wing CanSat you must control the CanSat in 3D. Since you have limited space in the rocket your design must consider wing folding, tail and if there exist engine propeller. The concept of the design must aim to improve performance, weight, strenght and simplicity.

Figure 9-3. Rover back CanSat with paraglider inside a rocket.

For a Mechanical design you should include drawings of the structure and component layout and a list of materials and component selections. The mass budget shall include allocation of masses to the various subsystems and/or components in a tabular form.

Access to the electrical components is an important design consideration. During the development and testing phase of the CanSat, the battery, circuit boards and the transceiver will be removed and replaced several times. Easy access to these components will save a significant amount of time over the entire development and launch phase. In short, it is necessary to have a structure that is light, strong, versatile, and easy to disassemble.

Figure 9-4. Assembly in the workshop

CHAPTER 10: PRE-LAUNCHING TESTING

9.1 INTRDUCTION

…. To be continue

CHAPTER 11: DATA ANALYSIS

10.1 INTRODUCTION

To be continue **…**

REFERENCES

1-1 www.cltp.info

1-2 http://www.cansatcompetition.com/

1-3 http://www.arliss.org

3-1 mbed programming workshop for cansat, senio networks, inc., 2011

3-2 Pedro Castillo, Rogelio Lozano and Alejandro E. Dzul, “Modelling and Control of Mini-Flying Machines”, Springer-Verlag London Limited 2005

4-1 Ahmed El-Rabbany, “Introduction to GPS”, GNSS Technology and Applications Series, second Edition, Artech House, 2006.

4-2 S. P. Drake, “Converting GPS Coordinates ($\phi\lambda$h) to Navigation Coordinates (ENU)”, DSTO Electronics and Surveillance Research Laboratory, DSTO-TN-0432, 2002.

5-1 T.W. Knacke, Parachute Recovery System, Design Manual”, Parapublishing 1992.

7-1 http://www.digi.com

APPENDIX A MBED LPC LPC1768 SPECIFICATION SHEET

mbed NXP LPC1768
prototyping board

Rapid prototyping for the LPC1768 MCU

This board, which works with the groundbreaking mbed tool suite, lets you create a functioning prototype faster than ever. The tightly coupled combination of hardware and software makes it easy to explore designs quickly, so you can be more adventurous, more inventive, and more productive.

Features

- Convenient form-factor: 40-pin DIP, 0.1-inch pitch
- Drag-and-drop programming, with the board represented as a USB drive
- Best-in-class Cortex-M3 hardware
  - 100 MHz ARM with 64 KB of SRAM, 512 KB of Flash
  - Ethernet, USB OTG
  - SPI, I²C, UART, CAN
  - GPIO, PWM, ADC, DAC
- Easy-to-use online tools
  - Web-based C/C++ programming environment
  - Uses the ARM RealView compile engine
  - API-driven development using libraries with intuitive interfaces
  - Comprehensive help and online community

Benefits

- Get started right away, with nothing to install
- Get working fast, using high-level APIs
- Explore, test, and demonstrate ideas more effectively
- Write clean, compact code that's easy to modify
- Log in from anywhere, on Windows, Mac or Linux

The mbed NXP LPC1768 board lets you create prototypes without having to work with low-level microcontroller details, so you can experiment and iterate faster than ever.

Designers compose and compile embedded software using a browser-based IDE, then download it quickly and easily, using a simple drag-and-drop function, to the board's NXP Cortex-M3 microcontroller LPC1768.

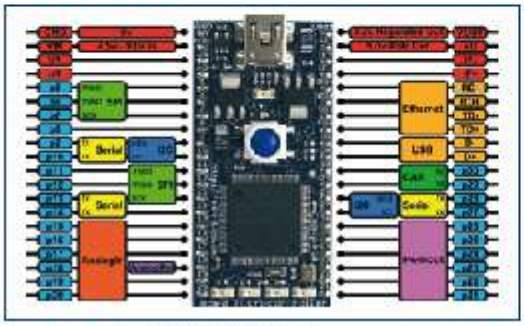
Engineers new to embedded applications can use the board to prototype real products incorporating microcontrollers, while experienced engineers can use it to be more productive in early stages of development. The mbed tools are designed to let you try out new ideas quickly, in much the same way that an architect uses a pencil and paper to sketch out concepts before turning to an advanced CAD program to implement a design.

NXP founded by Philips

Elegant simplicity

The mbed tool has been designed for the best trade-off between versatility and immediate connectivity. The LPC1768, housed in an LQFP package, is mounted on the mbed board, which uses a 40-pin DIP with a 0.1-inch pitch. The convenient form factor works seamlessly with solderless breadboards, stripboards, and PCBs.

There is no software to install – everything, even the compiler, is online. The compiler and libraries are completely modular, so they're easy to use, yet powerful enough to take on complex, real-world applications.



Pinout diagram of mbed NXP LPC1768 board

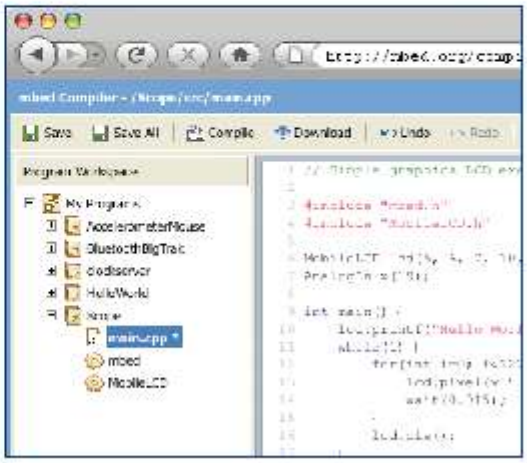
Hassle-free startup

Getting started is as simple as using a USB Flash drive. Simply connect the mbed NXP LPC1768 board to a Windows, Mac or Linux computer and it will appear as a USB drive. Follow the link on the board to connect to the mbed website, where you can sign up and begin designing. There are no drivers to install or setup programs to run. It's so easy, in fact, that you can have a "Hello World!" program running in as little as five minutes.

Online compiler

The mbed Compiler lets you write programs in C++ and then compile and download them to run on the mbed NXP LPC1768 microcontroller. There's no need to run an install or setup program, since the compiler runs online. Supported browsers include Internet Explorer, Firefox, Safari, or Chrome running on a Windows, Mac, or Linux PC. You can log in from anywhere and simply pick up where you left off. And, since you're working with a web-based tool, you can be confident that it's already configured and will stay up-to-date.

The compiler uses the ARM RealView compile engine, so it produces clean, efficient code that can be used free-of-charge, even in production. Existing ARM application code and middleware can be ported to the LPC1768 microcontroller, and the mbed tools can be used alongside other professional production-level tools, such as Keil MDK.



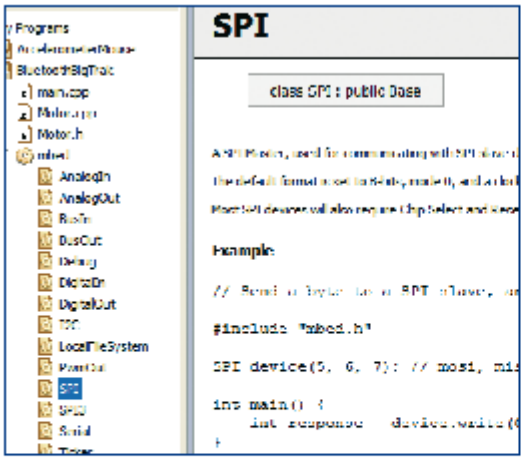
The mbed Compiler

Peripheral libraries

The mbed Library provides an API-driven approach to coding that eliminates much of the low-level work normally associated with MCU code development. You develop code using meaningful peripheral abstractions and API calls that are intuitive and already tested. That frees you up to experiment, without worrying about the implementation of the MCU core or its peripherals. You can work faster and be more creative, and can concentrate on exploring and testing the options for your design.

Rather than simply providing examples, mbed focuses on reusable library functionality, with clear interfaces and solid implementations. The core mbed Library supports the main LPC1768 peripherals, and the libraries already contributed by the mbed design community include USB, TCP/IP, and HTTP support. It's also possible to add third-party and open-source stacks.

The libraries comply with the ARM EABI and are built on the Cortex Microcontroller Software Interface Standard (CMSIS),

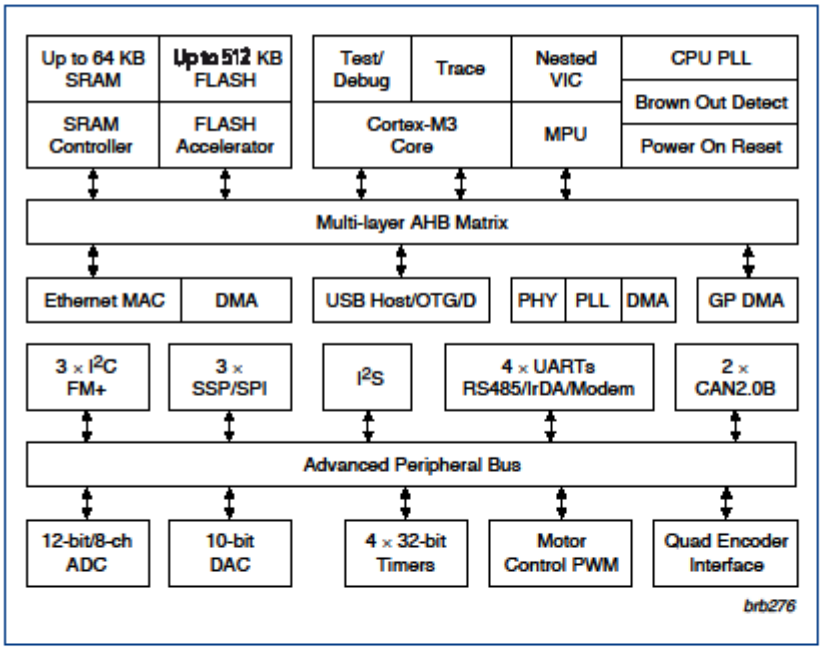


making it possible to migrate to other toolchains or implement custom code for peripheral interfaces.

LPC176x microcontrollers

The NXP microcontroller family LPC176x is a series of cost-effective, low-power Cortex-M3 devices that operate at up to 100 MHz. They feature best-in-class peripheral support, including Ethernet, USB 2.0 host/OTG/device, and CAN 2.0B. There are 512 KB of Flash memory and 64 KB of SRAM. The architecture uses a multi-layer AHB bus that allows high-bandwidth peripherals such as Ethernet and USB to run simultaneously, without impacting performance. The family is pin-compatible with NXP's 100-pin LPC236x series of ARM7-based microcontrollers.

The mbed Library



LPC1768 block diagram

LPC1768 features

ARM Cortex-M3 core	• 100 MHz operation • Nested Vectored Interrupt Controller for fast deterministic interrupts • Wakeup Interrupt Controller allows automatic wake from any priority interrupt • Memory Protection Unit • Four reduced-power modes: sleep, deep sleep, power-down and deep power-down
Memories	• 512 KB of Flash memory • 64 KB of SRAM
Serial peripherals	• 10/100 Ethernet MAC • USB 2.0 full-speed device/Host/ OTG controller with on-chip PHY • Four UARTs with fractional baud rate generation, RS-48, modem control, and IrDA • Two CAN 2.0B controllers • Three SSP/SPI controllers • Three I²C-bus interfaces with one supporting Fast Mode Plus (1-Mbit/s data rates) • I²S interface for digital audio
Analog peripherals	• 12-bit ADC with eight channels • 10-bit DAC
Other peripherals	• Ultra-low-power (< 1 uA) RTC • General-purpose DMA controller with eight channels • Up to 70 GPIO • Motor control PWM and Quadrature Encoder Interface to support three-phase motors • Four 32-bit general-purpose timers/counters
Package	• 100-pin LQFP (14 x 14 x 1.4 mm)

mbed

http://mbed.org

www.nxp.com

founded by PHILIPS

Date of release: September 2009
Document order number: 9397 750 16852
Printed in the Netherlands

APPENDIX B FREESCAL MMA7361LC ACCELEROMETER SPECIFICATION SHEETS

Freescale Semiconductor
Technical Data

Document Number: MMA7361LC
Rev 0, 03/2010

RoHS

±1.5g, ±6g Three Axis Low-g Micromachined Accelerometer

MMA7361LC

The MMA7361LC is a low power, low profile capacitive micromachined accelerometer featuring signal conditioning, a 1-pole low pass filter, temperature compensation, self test, 0g-Detect which detects linear freefall, and g-Select which allows for the selection between 2 sensitivities. Zero-g offset and sensitivity are factory set and require no external devices. The MMA7361LC includes a Sleep Mode that makes it ideal for handheld battery powered electronics.

MMA7361LC: XYZ AXIS ACCELEROMETER ±1.5g, ±6g

Features

- 3mm x 5mm x 1.0mm LGA-14 Package
- Low Current Consumption: 400 µA
- Sleep Mode: 3 µA
- Low Voltage Operation: 2.2 V – 3.6 V
- High Sensitivity (800 mV/g @ 1.5g)
- Selectable Sensitivity (±1.5g, ±6g)
- Fast Turn On Time (0.5 ms Enable Response Time)
- Self Test for Freefall Detect Diagnosis
- 0g-Detect for Freefall Protection
- Signal Conditioning with Low Pass Filter
- Robust Design, High Shocks Survivability
- RoHS Compliant
- Environmentally Preferred Product
- Low Cost

Bottom View

14 LEAD LGA CASE 1977-01

Typical Applications

- 3D Gaming: Tilt and Motion Sensing, Event Recorder
- HDD MP3 Player: Freefall Detection
- Laptop PC: Freefall Detection, Anti-Theft
- Cell Phone: Image Stability, Text Scroll, Motion Dialing, E-Compass
- Pedometer: Motion Sensing
- PDA: Text Scroll
- Navigation and Dead Reckoning: E-Compass Tilt Compensation
- Robotics: Motion Sensing

Top View

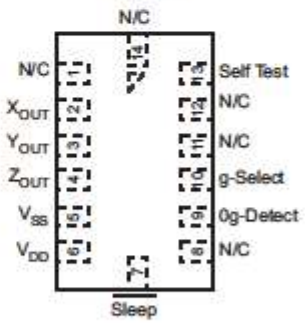


Figure 1. Pin Connections

ORDERING INFORMATION				
Part Number	Temperature Range	Package Drawing	Package	Shipping
MMA7361LCT	–40 to +85°C	1977-01	LGA-14	Tray
MMA7361LCR1	–40 to +85°C	1977-01	LGA-14	7" Tape & Reel
MMA7361LCR2	–40 to +85°C	1977-01	LGA-14	13" Tape & Reel

freescale semiconductor

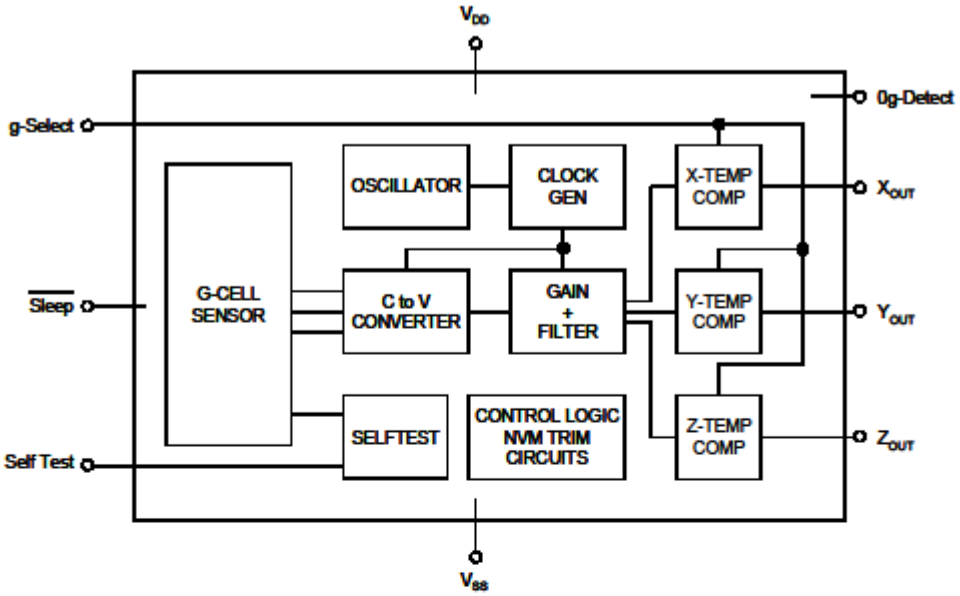


Figure 2. Simplified Accelerometer Functional Block Diagram

Table 1. Maximum Ratings
(Maximum ratings are the limits to which the device can be exposed without causing permanent damage.)

Rating	Symbol	Value	Unit
Maximum Acceleration (all axis)	g_{max}	±5000	g
Supply Voltage	V_{DD}	–0.3 to +3.6	V
Drop Test[1]	D_{drop}	1.8	m
Storage Temperature Range	T_{stg}	–40 to +125	°C

1. Dropped onto concrete surface from any axis.

ELECTRO STATIC DISCHARGE (ESD)

WARNING: This device is sensitive to electrostatic discharge.

Although the Freescale accelerometer contains internal 2000 V ESD protection circuitry, extra precaution must be taken by the user to protect the chip from ESD. A charge of over 2000 volts can accumulate on the human body or associated test equipment. A charge of this magnitude can alter the performance or cause failure of the chip. When handling the accelerometer, proper ESD precautions should be followed to avoid exposing the device to discharges which may be detrimental to its performance.

Table 2. Operating Characteristics
Unless otherwise noted: -40°C ≤ T_A ≤ 85°C, 2.2 V ≤ V_{DD} ≤ 3.6 V, Acceleration = 0g, Loaded output[1]

Characteristic	Symbol	Min	Typ	Max	Unit
Operating Range[2]					
Supply Voltage[3]	V_{DD}	2.2	3.3	3.6	V
Supply Current[4]	I_{DD}	—	400	600	µA
Supply Current at Sleep Mode[4]	I_{DD}	—	3	10	µA
Operating Temperature Range	T_A	-40	—	+85	°C
Acceleration Range, X-Axis, Y-Axis, Z-Axis					
g-Select: 0	g_{FS}	—	±1.5	—	g
g-Select: 1	g_{FS}	—	±6.0	—	g
Output Signal					
Zero-g (T_A = 25°C, V_{DD} = 3.3 V)[5], [6]	V_{OFF}				
XY		1.485	1.65	1.815	V
Z		1.32	1.65	1.815	v
Zero-g[4]	V_{OFF}, T_A	-2.0	±0.5	+2.0	mg/°C
Sensitivity (T_A = 25°C, V_{DD} = 3.3 V)					
1.5g	$S_{1.5g}$	740	800	860	mV/g
6g	S_{6g}	190.6	206	221.5	mV/g
Sensitivity[4]	S,T_A	-0.0075	±0.002	+0.0075	%/°C
Bandwidth Response					
XY	f_{-3dBXY}	—	400	—	Hz
Z	f_{-3dBZ}	—	300	—	Hz
Output Impedance	Z_O	—	32	—	kΩ
0g-Detect	$0g_{detect}$	-0.4	0	+0.4	g
Self Test					
Output Response					
X_{OUT}, Y_{OUT}	Δg_{STXY}	+0.05	-0.1	—	g
Z_{OUT}	Δg_{STZ}	+0.8	+1.0	+1.2	g
Input Low	V_{IL}	V_{SS}	—	0.3 V_{DD}	V
Input High	V_{IH}	0.7 V_{DD}	—	V_{DD}	V
Noise					
Power Spectral Density RMS (0.1 Hz – 1 kHz)[4]	n_{PSD}	—	350	—	$\mu g/\sqrt{Hz}$
Control Timing					
Power-Up Response Time[7]	$t_{RESPONSE}$	—	1.0	2.0	ms
Enable Response Time[8]	t_{ENABLE}	—	0.5	2.0	ms
Self Test Response Time[9]	t_{ST}	—	2.0	5.0	ms
Sensing Element Resonant Frequency					
XY	$f_{GCELLXY}$	—	6.0	—	kHz
Z	f_{GCELLZ}	—	3.4	—	kHz
Internal Sampling Frequency	f_{CLK}	—	11	—	kHz
Output Stage Performance					
Full-Scale Output Range (I_{OUT} = 3 µA)	V_{FSO}	V_{SS}+0.1	—	V_{DD}–0.1	V
Nonlinearity, X_{OUT}, Y_{OUT}, Z_{OUT}	NL_{OUT}	-1.0	—	+1.0	%FSO
Cross-Axis Sensitivity[10]	$V_{XY, XZ, YZ}$	-5.0	—	+5.0	%

1. For a loaded output, the measurements are observed after an RC filter consisting of an internal 32 kΩ resistor and an external 3.3 nF capacitor (recommended as a minimum to filter clock noise) on the analog output for each axis and a 0.1µF capacitor on V_{DD} - GND. The output sensor bandwidth is determined by the capacitor added on the output. f = 1/2π * (32 x 10^3) * C. C = 3.3 nF corresponds to BW = 1507 HZ, which is the minimum to filter out internal clock noise.
2. These limits define the range of operation for which the part will meet specification.
3. Within the supply range of 2.2 and 3.6 V, the device operates as a fully calibrated linear accelerometer. Beyond these supply limits the device may operate as a linear device but is not guaranteed to be in calibration.
4. This value is measured with g-Select in 1.5g mode.
5. The device can measure both + and – acceleration. With no input acceleration the output is at midsupply. For positive acceleration the output will increase above V_{DD}/2. For negative acceleration, the output will decrease below V_{DD}/2.
6. For optimal 0g offset performance, adhere to AN3484 and AN3447
7. The response time between 10% of full scale V_{DD} input voltage and 90% of the final operating output voltage.
8. The response time between 10% of full scale Sleep Mode input voltage and 90% of the final operating output voltage.
9. The response time between 10% of the full scale self test input voltage and 90% of the self test output voltage.
10. A measure of the device's ability to reject an acceleration applied 90° from the true axis of sensitivity.

PRINCIPLE OF OPERATION

The Freescale accelerometer is a surface-micromachined integrated-circuit accelerometer.

The device consists of a surface micromachined capacitive sensing cell (g-cell) and a signal conditioning ASIC contained in a single package. The sensing element is sealed hermetically at the wafer level using a bulk micromachined cap wafer.

The g-cell is a mechanical structure formed from semiconductor materials (polysilicon) using semiconductor processes (masking and etching). It can be modeled as a set of beams attached to a movable central mass that move between fixed beams. The movable beams can be deflected from their rest position by subjecting the system to an acceleration (Figure 3).

As the beams attached to the central mass move, the distance from them to the fixed beams on one side will increase by the same amount that the distance to the fixed beams on the other side decreases. The change in distance is a measure of acceleration.

The g-cell beams form two back-to-back capacitors (Figure 3). As the center beam moves with acceleration, the distance between the beams changes and each capacitor's value will change, ($C = A\varepsilon/D$). Where A is the area of the beam, ε is the dielectric constant, and D is the distance between the beams.

The ASIC uses switched capacitor techniques to measure the g-cell capacitors and extract the acceleration data from the difference between the two capacitors. The ASIC also signal conditions and filters (switched capacitor) the signal, providing a high level output voltage that is ratiometric and proportional to acceleration.

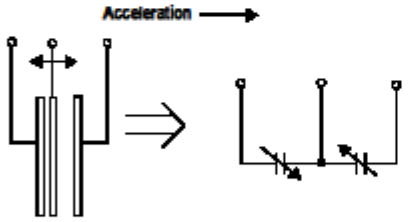


Figure 3. Simplified Transducer Physical Model

SPECIAL FEATURES

0g-Detect

The sensor offers a 0g-Detect feature that provides a logic high signal when all three axes are at 0g. This feature enables the application of Linear Freefall protection if the signal is connected to an interrupt pin or a poled I/O pin on a microcontroller.

Self Test

The sensor provides a self test feature that allows the verification of the mechanical and electrical integrity of the accelerometer at any time before or after installation. This feature is critical in applications such as hard disk drive protection where system integrity must be ensured over the life of the product. Customers can use self test to verify the solderability to confirm that the part was mounted to the PCB correctly. To use this feature to verify the 0g-Detect function, the accelerometer should be held upside down so that the Z-axis experiences -1g. When the self test function is initiated, an electrostatic force is applied to each axis to cause it to deflect. The X- and Y-axis are deflected slightly while the Z-axis is trimmed to deflect 1g. This procedure assures that both the mechanical (g-cell) and electronic sections of the accelerometer are functioning.

g-Select

The g-Select feature allows for the selection between two sensitivities. Depending on the logic input placed on pin 10, the device internal gain will be changed allowing it to function with a 1.5g or 6g sensitivity (Table 3). This feature is ideal when a product has applications requiring two different sensitivities for optimum performance. The sensitivity can be changed at anytime during the operation of the product. The g-Select pin can be left unconnected for applications requiring only a 1.5g sensitivity as the device has an internal pull-down to keep it at that sensitivity (800 mV/g).

Table 3. g-Select Pin Description

g-Select	g-Range	Sensitivity
0	1.5g	800 mV/g
1	6g	206 mV/g

Sleep Mode

The 3 axis accelerometer provides a Sleep Mode that is ideal for battery operated products. When Sleep Mode is active, the device outputs are turned off, providing significant reduction of operating current. A low input signal on pin 7 (Sleep Mode) will place the device in this mode and reduce the current to 3 µA typ. For lower power consumption, it is recommended to set g-Select to 1.5g mode. By placing a high input signal on pin 7, the device will resume to normal mode of operation.

Filtering

The 3 axis accelerometer contains an onboard single-pole switched capacitor filter. Because the filter is realized using switched capacitor techniques, there is no requirement for external passive components (resistors and capacitors) to set the cut-off frequency.

Ratiometricity

Ratiometricity simply means the output offset voltage and sensitivity will scale linearly with applied supply voltage. That is, as supply voltage is increased, the sensitivity and offset increase linearly; as supply voltage decreases, offset and sensitivity decrease linearly. This is a key feature when interfacing to a microcontroller or an A/D converter because it provides system level cancellation of supply induced errors in the analog to digital conversion process.

BASIC CONNECTIONS

Pin Descriptions

Top View

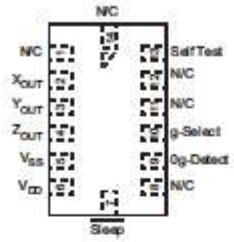


Figure 4. Pinout Description

Table 4. Pin Descriptions

Pin No.	Pin Name	Description
1	N/C	No internal connection Leave unconnected
2	X_{OUT}	X direction output voltage
3	Y_{OUT}	Y direction output voltage
4	Z_{OUT}	Z direction output voltage
5	V_{SS}	Power Supply Ground
6	V_{DD}	Power Supply Input
7	$\overline{\text{Sleep}}$	Logic input pin to enable product or Sleep Mode
8	NC	No internal connection Leave unconnected
9	0g-Detect	Linear Freefall digital logic output signal
10	g-Select	Logic input pin to select g level
11	N/C	Unused for factory trim Leave unconnected
12	N/C	Unused for factory trim Leave unconnected
13	Self Test	Input pin to initiate Self Test
14	N/C	Unused for factory trim Leave unconnected

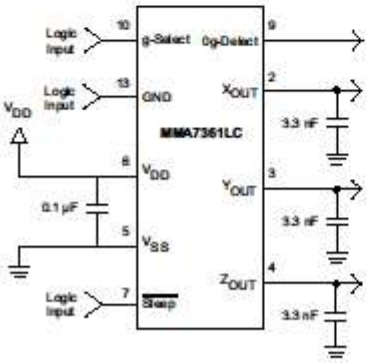


Figure 5. Accelerometer with Recommended Connection Diagram

PCB Layout

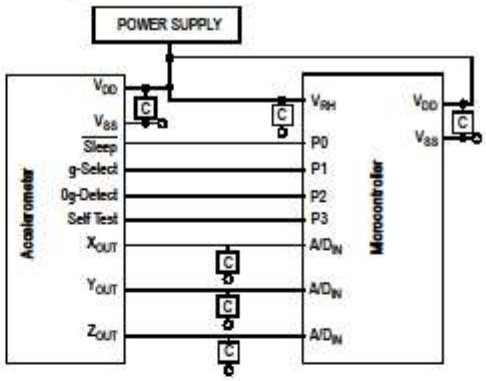
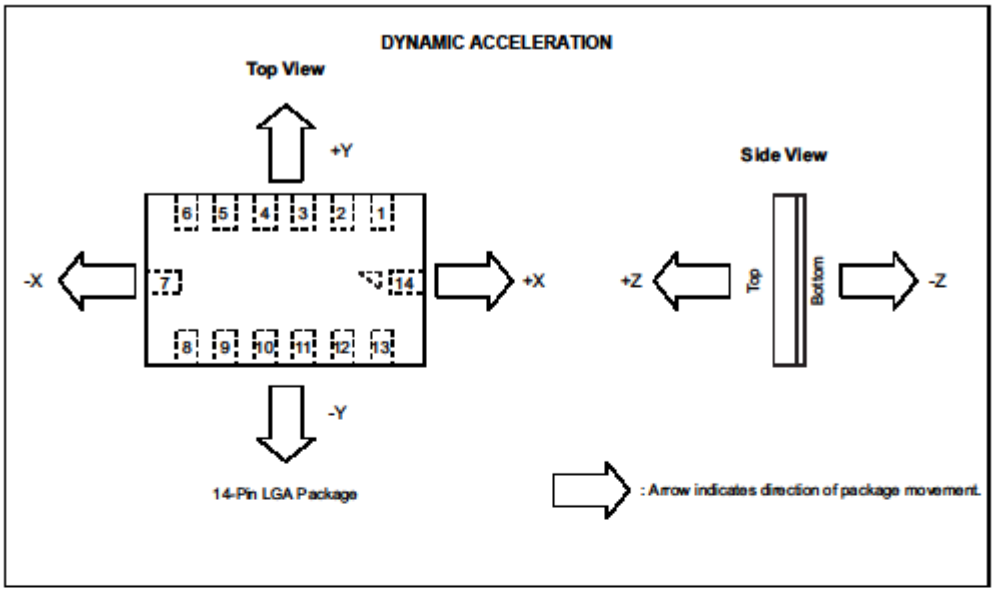
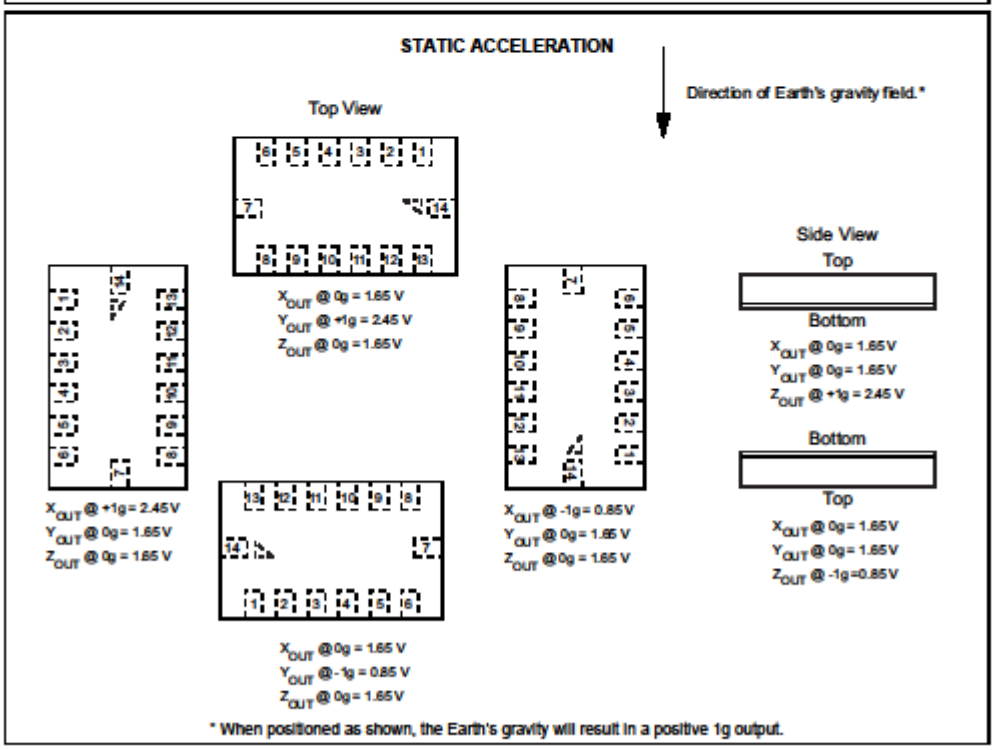


Figure 6. Recommended PCB Layout for Interfacing Accelerometer to Microcontroller

NOTES:

1. Use 0.1 µF capacitor on V_{DD} to decouple the power source.
2. Physical coupling distance of the accelerometer to the microcontroller should be minimal.
3. Place a ground plane beneath the accelerometer to reduce noise, the ground plane should be attached to all of the open ended terminals shown in Figure 6.
4. Use a 3.3 nF capacitor on the outputs of the accelerometer to minimize clock noise (from the switched capacitor filter circuit).
5. PCB layout of power and ground should not couple power supply noise.
6. Accelerometer and microcontroller should not be a high current path.
7. A/D sampling rate and any external power supply switching frequency should be selected such that they do not interfere with the internal accelerometer sampling frequency (11 kHz for the sampling frequency). This will prevent aliasing errors.
8. 10 MΩ or higher is recommended on X_{OUT}, Y_{OUT} and Z_{OUT} to prevent loss due to the voltage divider relationship between the internal 32 kΩ resistor and the measurement input impedance.


DYNAMIC ACCELERATION
Top View
+Y
-X
+X
-Y
14-Pin LGA Package
Side View
+Z
-Z
Top
Bottom
: Arrow indicates direction of package movement.


STATIC ACCELERATION
Direction of Earth's gravity field.*
Top View
X_{OUT} @ 0g = 1.65 V
Y_{OUT} @ +1g = 2.45 V
Z_{OUT} @ 0g = 1.65 V
X_{OUT} @ +1g = 2.45 V
Y_{OUT} @ 0g = 1.65 V
Z_{OUT} @ 0g = 1.65 V
X_{OUT} @ -1g = 0.85 V
Y_{OUT} @ 0g = 1.65 V
Z_{OUT} @ 0g = 1.65 V
X_{OUT} @ 0g = 1.65 V
Y_{OUT} @ -1g = 0.85 V
Z_{OUT} @ 0g = 1.65 V
Side View
Top
Bottom
X_{OUT} @ 0g = 1.65 V
Y_{OUT} @ 0g = 1.65 V
Z_{OUT} @ +1g = 2.45 V
Bottom
Top
X_{OUT} @ 0g = 1.65 V
Y_{OUT} @ 0g = 1.65 V
Z_{OUT} @ -1g = 0.85 V
* When positioned as shown, the Earth's gravity will result in a positive 1g output.

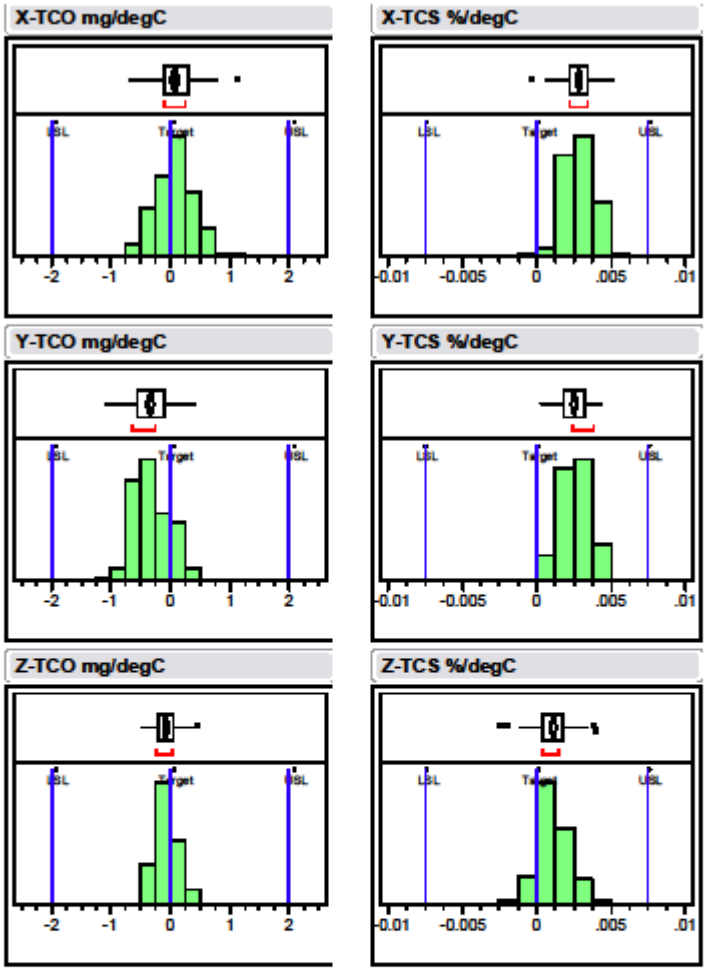


Figure 7. MMA7361LC Temperature Coefficient of Offset (TCO) and Temperature Coefficient of Sensitivity (TCS) Distribution Charts

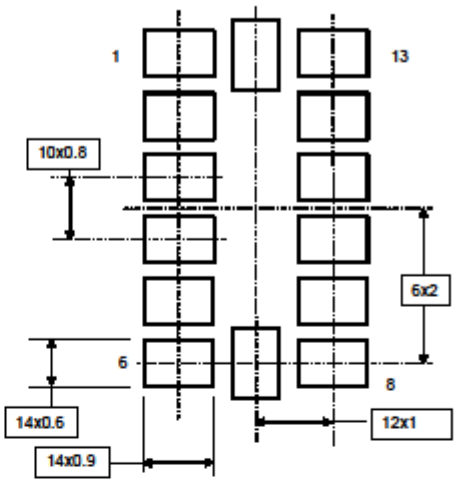
MINIMUM RECOMMENDED FOOTPRINT FOR SURFACE MOUNTED APPLICATIONS

PCB Mounting Recommendations

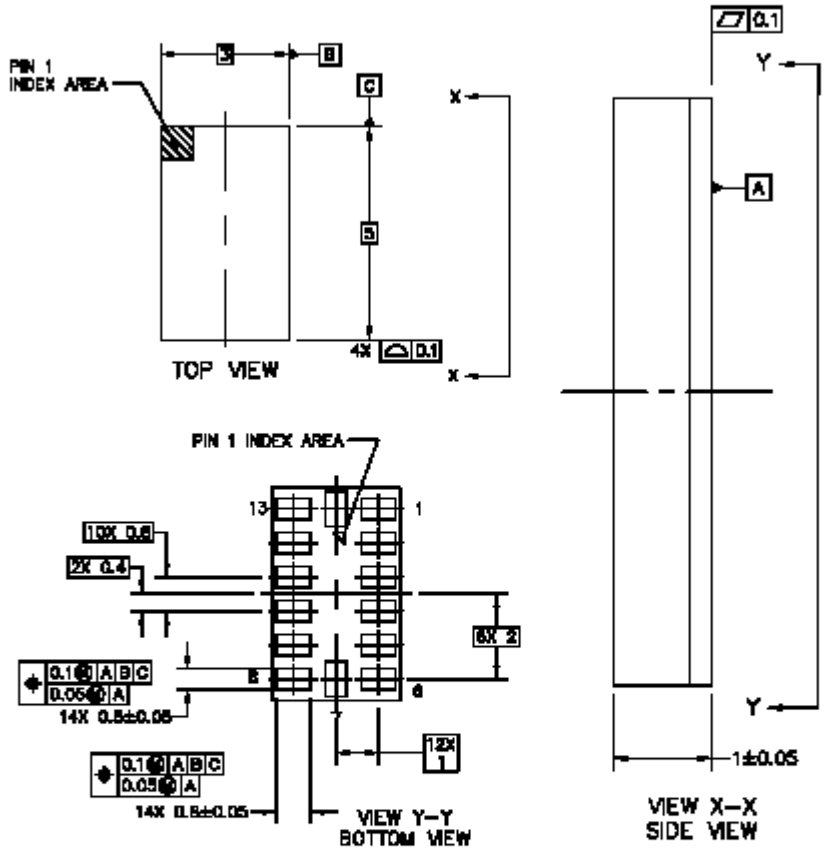
MEMS based sensors are sensitive to Printed Circuit Board (PCB) reflow processes. For optimal zero-g offset after PCB mounting, care must be taken to PCB layout and reflow conditions. Reference application note AN3484 for best practices to minimize the zero-g offset shift after PCB mounting.

Surface mount board layout is a critical portion of the total design. The footprint for the surface mount packages must be the correct size to ensure proper solder connection interface between the board and the package.

With the correct footprint, the packages will self-align when subjected to a solder reflow process. It is always recommended to design boards with a solder mask layer to avoid bridging and shorting between solder pads.



PACKAGE DIMENSIONS



© FREESCALE SEMICONDUCTOR, INC. ALL RIGHTS RESERVED.	MECHANICAL OUTLINE	PRINT VERSION NOT TO SCALE	
TITLE: LGA 14 I/O, 3 X 5 X 1 PITCH 0.8, SENSOR 1.0MM PKG	DOCUMENT NO: 98ASA10880D		REV: A
	CASE NUMBER: 1977–01		09 JAN 2008
	STANDARD: NON–JEDEC		

CASE 1977-01
ISSUE A
14-LEAD LGA

PACKAGE DIMENSIONS

NOTES:

1. ALL DIMENSIONS IN MILLIMETERS.
2. DIMENSIONING AND TOLERANCING PER ASME Y14.5M–1994.

© FREESCALE SEMICONDUCTOR, INC. ALL RIGHTS RESERVED.	MECHANICAL OUTLINE	PRINT VERSION NOT TO SCALE
TITLE: LGA 14 I/O, 3 X 5 X 1 PITCH 0.8, SENSOR 1.0MM PKG	DOCUMENT NO: 98ASA10801D	REV: A
	CASE NUMBER: 1977–01	09 JAN 2006
	STANDARD: NON–JEDEC	

CASE 1977-01
ISSUE A
14-LEAD LGA

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor, Inc.
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
+1-800-521-6274 or +1-480-768-2130
www.freescale.com/support

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
www.freescale.com/support

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

RoHS-compliant and/or Pb-free versions of Freescale products have the functionality and electrical characteristics of their non-RoHS-compliant and/or non-Pb-free counterparts. For further information, see http://www.freescale.com or contact your Freescale sales representative.

For information on Freescale's Environmental Products program, go to http://www.freescale.com/epp.

MMA7361LC
Rev. 0
03/2010

APPENDIX C CALIBRATION TECHNIQUE FOR ACCELEROMETERS

Freescale Semiconductor
Application Note

AN3447
Rev 0, 03/2007

Implementing Auto-Zero Calibration Technique for Accelerometers

by: Kimberly Tuck
Accelerometer Systems and Applications Engineering
Tempe, AZ

INTRODUCTION

This application note describes why it is important to implement an auto-zero calibration function into the MMA73x0L series accelerometers and how to implement it with a microcontroller containing an analog-to-digital converter. This auto-zero compensation technique is based on sampling the accelerometer voltage output value at 0g which is the zero reference. This reference value is known as the offset voltage. The expected offset values for the MMA73x0L series accelerometers is listed in Table 1 for 0g, +1g.

Table 1. Offset Values for MMA73x0L for X, Y and Z Directions

Device	Sensitivity	0g	-1g	+1g
MMA7360L	800mV/g	1.65V	0.85V	2.45V
MMA7340L	440mV/g	1.65V	1.21V	2.09V
MMA7330L	308mV/g	1.40V	1.092V	1.708V

OFFSET ERRORS AND WHY AUTO ZERO CALIBRATION IS IMPORTANT

Sources of offset errors can occur device to device based on offset variations from trim errors, mechanical stresses from the package and mounting, shifts due to temperature and due to aging. These variables can all change the offset. This can be very significant for many applications. The offset error alone can affect a tilt reading on a flat surface by as much as 12 degrees. That is an unacceptable error for this application.

Performing an auto-zero calibration technique using a microcontroller with an A/D converter can reduce these errors, but note that the error correction will be limited by the resolution of the A/D converter.

Figure 1 illustrates the transfer function of the accelerometer. This relation assumes linearity. The output is a voltage and the input is the acceleration measured in g's. The equation of the line is expressed as follows:

$$V_{OUT} = [(V_2 - V_1)/(g_2 - g_1)]^*g + V_{OFF} = S^*g + V_{OFF};$$

where V_2 and V_1 are two voltages and g_2 and g_1 are their corresponding input acceleration values. The slope of the line $S = (V_2-V_1)/(g_2-g_1)$ is the sensitivity of the accelerometer sensor. The y-intercept of the line is the voltage offset value V_{OFF}.

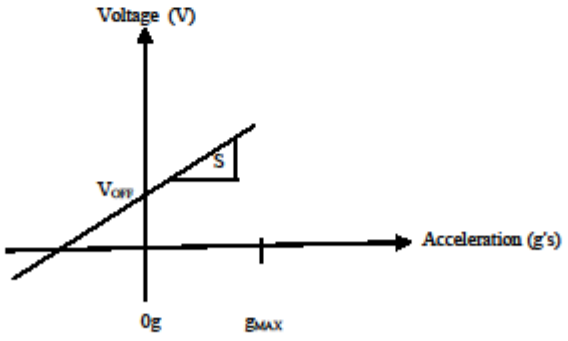


Figure 1. Transfer Function of the Accelerometer Sensor

A two point acceleration calibration can be performed to accurately determine the sensitivity and get rid of the offset calibration errors. This can be very expensive in high volume production due to extra time involved. Therefore, the sensitivity and offset data is given in the data sheet and a linear equation is used to determine the acceleration.

$$\text{Acceleration(g)} = (V_{OUT} - V_{OFF})/S$$

If an offset error is introduced due to device-to-device variation, mechanical stresses or offset shift due to temperature, those errors will show up as an error in the acceleration reading. As shown in Figure 2, if an offset error is introduced (ΔV_{OFF}) then there will be a corresponding error in the acceleration reading, Δg.

$$g + \Delta g = [V_{OUT} - (V_{OFF} + \Delta V_{OFF})]/S$$

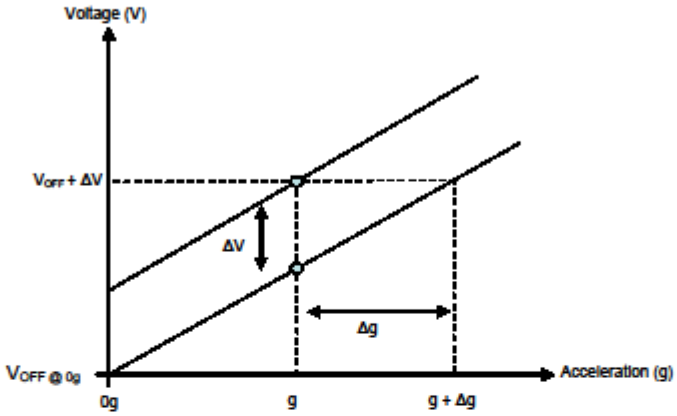


Figure 2. Effects of Offset Errors

OFFSET CALIBRATION ERRORS

The minimum expected and maximum offset values at 0g for the MMA73x0L devices are listed in Table 2.

Table 2. Offset Value Range for the MMA73XOL Accelerometer Series

Device Name	g-Range	Min. Offset	Expected Offset	Max. Offset
MMA7360L	1.5/6	1.485V	1.65V	1.815V
MMA7340L	3/12	1.485V	1.65V	1.815V
MMA7330L	4/16	1.316V	1.4V	1.484V

Even though the offset is laser trimmed, offset can shift due to packaging stresses, aging and external mechanical stresses due to mounting and orientation. This results in offset calibration error. Table 2 lists the minimum, maximum and expected offset ranges.

TEMPERATURE COEFFICIENT OF OFFSET ERRORS

The offset error due to temperature is due to the Temperature Coefficient of Offset (TCO). This parameter is the rate of change of the offset when the sensor is subject to temperature. It is defined as: TCO = ($\Delta V_{OFF}/\Delta T$) assuming it is linear. The MMA7360 has a TCO normalized with the span at 25°C of ±0.03% / °C.

TECHNIQUES FOR CALIBRATING THE OFFSET VOLTAGE

Manual 0g X, Y, Z Full Range Calibration

In order to find the 0g voltage output value of the accelerometer it is necessary to know that the device is sitting completely level. Although placing the device on the table may seem flat, this doesn't guarantee that the device is not experiencing a slight g force. The device may be experiencing more that 0g due to packaging or device shifts. One method used to get an accurate and reliable 0g reading is to rotate the device from +1g through -1g. The max value will be +1g and the minimum value will be -1g. Assuming that the sensitivity is symmetric from zero to positive and from zero to negative the sensitivity of the device can be calculated by dividing by 2. Knowing the sensitivity the 0g offset value can be calculated by adding the sensitivity to the minimum value or by subtracting the sensitivity from the max value. It is also a good idea to place the part level and check the 0g offset value. It should be very close. This method must be followed for all three axes. The drawback of this technique is that it is tedious.

Simple 0g X, Y, Z calibration

Another method would be to assume 0g on a level surface. The device would need to be turned 90 degrees once to go from 0g XY to 0g in Z. The 0g values would be recorded this way. This technique does not guarantee as much accuracy as the previous method.

Freefall Calibration

Another method to calibrate the device for 0g offset would be to record the offset of X, Y and Z while the device is in freefall. The possible downfall of this approach is the fact that the device may rotate while falling and put a force on the device. Also it may be inconvenient to recalibrate each time by putting the device in freefall. The benefit of this approach is that all three axes can be at 0g all at the same time.

Simple 0g X, 0g Y, +1g Z calibration

Another method that is extremely convenient would be to place the device on a flat surface so that X is at 0g, Y is at 0g and Z is at +1g. The values are recorded. The X and Y offset values would be fairly accurate, but the +1g value would have errors because this value would not be recorded at 0g. The known sensitivity would be subtracted from the +1g to calculate an assumed 0g offset value for Z. This is convenient in that the device would never need to be rotated or moved. The disadvantage of this approach is that it is the least accurate way to calibrate 0g offset of the above mentioned techniques.

IMPLEMENTATION OF AUTO-ZERO WITH A MICROCONTROLLER

When implementing auto-zero (a zero reference) 0g must be used. An auto-zero command can be automated by the system or it can be commanded manually. This may be somewhat dependant on the application. Note that if the auto-zero is to be performed only once and the offset correction data is stored in memory the TCO offset error and calibration error will not be corrected if the sensor later experiences a wide temperature range or later experiences an offset shift. It would be wise to perform an auto-zero calibration at the operating temperature to compensate for the TCO and auto calibrate as often as possible to dynamically compensate for system offset errors.

Auto-zero can be implemented easily when the integrated sensor is interfaced to a microcontroller. The auto-zero algorithm is listed below:

1. Sample the sensor output when a known zero reference is applied to the sensor (0g is the reference). Store the current 0g or +1g offset (depending on the technique used) as CZ_{OFF}.
2. Sample the sensor output at the current applied acceleration. Call this CA.
3. Subtract the stored offset correction, CZ_{OFF}, from CA. The acceleration being measured at the current reading is simply:

$$A_{MEAS} = [CA - CZ_{OFF}]/S$$

For the *Simple 0g X, 0g Y, +1g Z calibration* technique a slightly different calculation is required for the Z axis which has recorded the CZ_{OFF} value at +1g. The sensitivity 'S' must therefore be subtracted from CZ_{OFF} in the calculation.

For the Z Case:

$$A_{MEAS} = [CA - (CZ_{OFF} - S)]/S$$

Note that the equation is simply a straight line equation where S is the sensitivity of the accelerometer. The auto-zero algorithm is shown graphically in Figure 3.

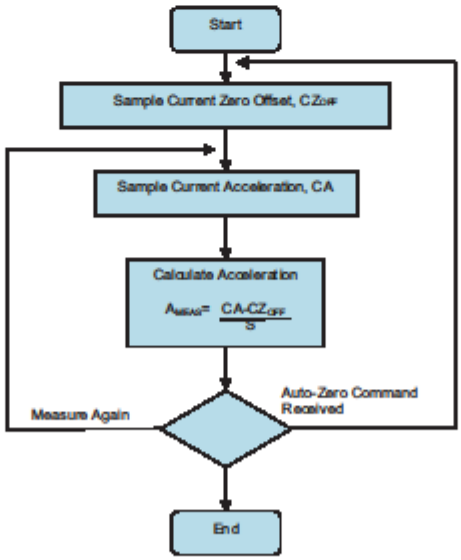


Figure 3. Flow Chart of the Auto-Zero Algorithm

Sample Code for: *Simple 0g X, 0g Y, +1g Z calibration method*

S = Sensitivity;

X0g_current = XCZ_{OFF};

Y0g_current = YCZ_{OFF};

Z1g_current = ZCZ_{OFF};

XCA = XCA_val;

YCA = YCA_val;

ZCA = ZCA_val;

AmeasX = (XCA-X0gcurrent)/S;

AmeasY = (YCA-Y0gcurrent)/S;

AmeasZ = (ZCA-(Z1g_current-S))/S;

Sample Code for: *all other 0g calibration methods*

S = Sensitivity;

X0g_current = XCZ_{OFF};

Y0g_current = YCZ_{OFF};

Z0g_current = ZCZ_{OFF};

XCA = XCA_val;

YCA = YCA_val;

ZCA = ZCA_val;

AmeasX = (XCA-X0gcurrent)/S;

AmeasY = (YCA-Y0gcurrent)/S;

AmeasZ = (ZCA-Z0g_current)/S;

A/D RESOLUTION ERROR

The auto-zero calibration technique can reduce the offset errors, but the A/D converter used in this process has a limited resolution and therefore introduces an error of its own.

Typically an 8 bit A/D converter is used, which cuts the 3.3V supply voltage on the MMA7360L into 255 steps, 12.9mV for each step. If a 10 bit A/D converter is used this would cut the 3.3V supply voltage into 1023 steps, 3.2mV for each step. A 12 bit A/D converter cuts the 3.3V supply voltage by 4095 steps, 0.8mV per step. Therefore the resolution is much better when a larger A/D converter is used, which reduces the error. Using an 8 bit A/D converter the voltage offset would be V_{OFF} + 0.129V = 1.779V. The V_{OFF} can be auto-zeroed, but the A/D converter resolution remains erroneous.

CONCLUSION

An auto-zero calibration technique is very important for minimizing offset errors. It is easily implemented into the accelerometer sensor system using a microcontroller with an A/D converter and a few lines of code. The resulting minimized offset errors of the system is limited only by the resolution of the A/D converter.

REFERENCES

1. AN1636, Implementing Auto Zero for Integrated Pressure Sensors, Ador Reodique, Freescale Semiconductor, Inc., Freescale Application Note.

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor, Inc.
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
+1-800-521-6274 or +1-480-768-2130
www.freescale.com/support

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
www.freescale.com/support

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

AN3447
Rev. 0
03/2007

APPENDIX D SCP 1000 ABSOLUTE PRESSURE SENSOR SPECIFICATION SHEETS

VTI TECHNOLOGIES

SCP1000 Series

Data Sheet

SCP1000 SERIES (120 kPa) ABSOLUTE PRESSURE SENSOR

Features

- 30 kPa - 120 kPa measuring range
- Single +2.4 ... 3.3 V supply
- Four measuring modes plus power down mode
- On-chip temperature measurement
- Fully calibrated and compensated component
- Standard digital output: SPI or I^2C (1
- Small package size with optional (2 sealing gasket. Diameter 6.1 mm, height 1.7 mm
- Proof pressure 2.0 MPa
- Pb-free solderable component & RoHS-compatible

Applications

- Barometric pressure measurement and altimeter applications
- Home weather stations
- Advanced medical applications
- Level gauging

Benefits

- 1.5 Pa (~10cm at sea level) resolution with < 10 µA current consumption
- Altimeter/barometer function can be realized with minimum use of MCU
- Chemically and mechanically robust package

Ordering Information

Product	Description
SCP1000-D01	30 kPa - 120 kPa, SPI interface
SCP1000-D11	30 kPa - 120 kPa, I^2C interface

Absolute Maximum Ratings

Parameter	Value	Unit
Supply voltage (Vdd)	-0.3 to +3.6	V
Voltage at input / output pins (3	-0.3 to (Vdd + 0.3)	V
ESD (Human Body Model)	±2.0	kV
ESD (Charged Device Mode)	±0.5	kV
Storage temperature	-30 ... +85	°C
Proof pressure	2.0	MPa

Note 1) I^2C compatible TWI (Two Wire Interface)
Note 2) VTI's I-seal gasket
Note 3) Referred to DVDD

Electrical Characteristics

Vdd = 2.7 V and ambient temperature unless otherwise specified

Parameter	Condition	Min.	Typ	Max.	Units
Supply voltage Vdd		2.4		3.3	V
Current Consumption	High resolution mode		25		µA
	High speed mode		25		µA
	Ultra low power mode		3.5		µA
	Power down		200	500	nA
Output Load	@ 500 kHz			50	pF
Digital Pins Input Capacitance				1.6	pF
SPI Clock Frequency				500	kHz
I^2C Clock Frequency				400	kHz
Data Transfer Time	@ 500 kHz		120		µs

Performance Characteristics[4]

Parameter	Condition	Min.	Typ	Max.	Units
Operating Pressure Range	Nominal	30		120	kPa
Operating Temperature		-20		+70	°C
Resolution [5]	High resolution mode		1.5	3	Pa
	High speed mode		3	6	Pa
Relative Pressure Accuracy [5]	600 hPa...1200 hPa +10 °C ... +40 °C	-50		+50	Pa
Absolute Pressure Accuracy [5]	600 hPa...1200 hPa +10 °C ... +40 °C	-150		+150	Pa
Absolute Pressure Accuracy [5]	300 hPa...1200 hPa -20 °C ... +70 °C	-200		+200	Pa
Long-term Stability	12 months		+100		Pa
Digital pressure output data word length				19	bits
Digital temperature output data word length				14	bits
Pressure Data Output Refresh Rate	High resolution mode		1.8		Hz
	High speed mode		9		Hz
Temp. Resolution			0.2	0.5	°C
Temp. Accuracy		-2	±1	+2	°C

Note 4 Soldered components. Detailed information about the effect of soldering and mounting can be found in SCP1000 Pressure Sensor Assembly Instructions (Technical Note 51)
Note 5 Typical = Median, Min. /Max. = 95% of the components within the population

Interface Options

Communication interface is pre-programmed in the factory and it can be either SPI (Serial Peripheral Interface) or I^2C (Inter-Integrated Circuit). The SPI interface is a full duplex 4 wire interface and the connection between the MCU and SCP1000 is achieved using MOSI, MISO, SCK and CSB. The I^2C interface is slave-only half duplex two-wire interface available on SDA and SCL pins.

Interface Options

The SCP1000 Pressure sensor has 4 normal operation modes plus power down mode.

Mode	Pressure measurement	Resolution	Output data refresh rate
High resolution	Continues	17 bits	1.8 Hz
High speed	Continues	15 bits	9 Hz
Ultra low power	Periodical	15 bits	Approximately 1 Hz
Low power	External trigger	17 bits or 15 bits	
Power Down	Activated through PD pin		

Electrical Connections

The SCP1000 pressure sensor supports two communication interfaces, SPI and I^2C. Appropriate communication interface is pre-programmed in the factory.

SCP1000- D01 (SPI Interface)

Pin #	Name	Function	Characteristics
1	ATST	Connect to analog ground	For factory use only
2	TRIG[6]	Trigger input, connect to GND if not used	Digital input
3	DRDY	Interrupt signal (data ready)	Digital output
4	CLK	Connect to digital GND	
5	DVDD	Digital supply voltage	Power line
6	DVSS	Digital ground	Power line
7	DVDDS	Digital supply filter	Digital power supply filter
8	PD[6]	Power down, connect to GND if not used	Input to force the chip in power down mode
9	SCK[6]	SPI clock input	Interface clock
10	MOSI[6]	SPI data input	Digital data input
11	MISO	SPI data output	Digital data output
12	CSB[6]	SPI chip select	Digital input
13	AVDD	Analog supply voltage	Power line
14	AVSS	Analog ground	Power line
15	AVSS	Analog ground	Power line
16	ATST	Connect to analog ground	For factory use only

Note 6 The MCU has to actively drive the signal in high and low states

SCP1000- D11 (I^2C Interface)

Pin #	Name	Function	Characteristics
1	ATST	Connect to analog ground	For factory use only
2	TRIG[6]	Trigger input, connect to GND if not used	Digital input
3	DRDY	Interrupt signal (data ready)	Digital output
4	CLK	Connect to digital GND	
5	DVDD	Digital supply voltage	Power line
6	DVSS	Digital ground	Power line
7	DVDDS	Digital supply filter	Digital power supply filter
8	PD[6]	Power down, connect to GND if not used	Input to force the chip in power down mode
9	SCL	I^2C serial clock	Interface clock
10	SDA	I^2C data	I2C data input/output
11	MISO	n/c	n/c
12	CSB	Connect to analog supply voltage	
13	AVDD	Analog supply voltage	Power line
14	AVSS	Analog ground	Power line
15	AVSS	Analog ground	Power line
16	ATST	Connect to analog ground	For factory use only

Note 6 The MCU has to actively drive the signal in high and low states

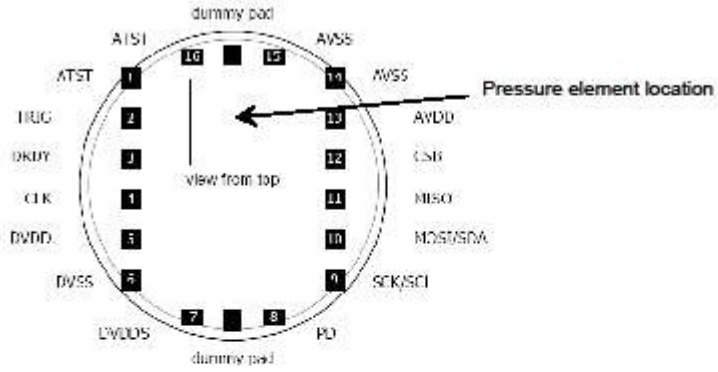
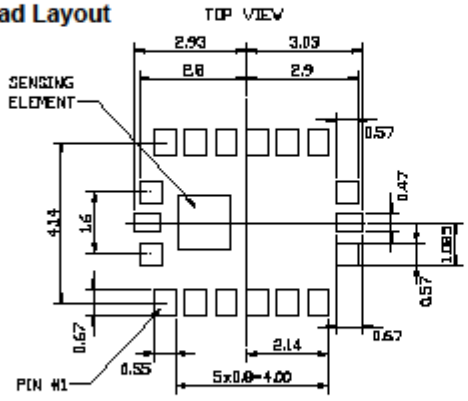


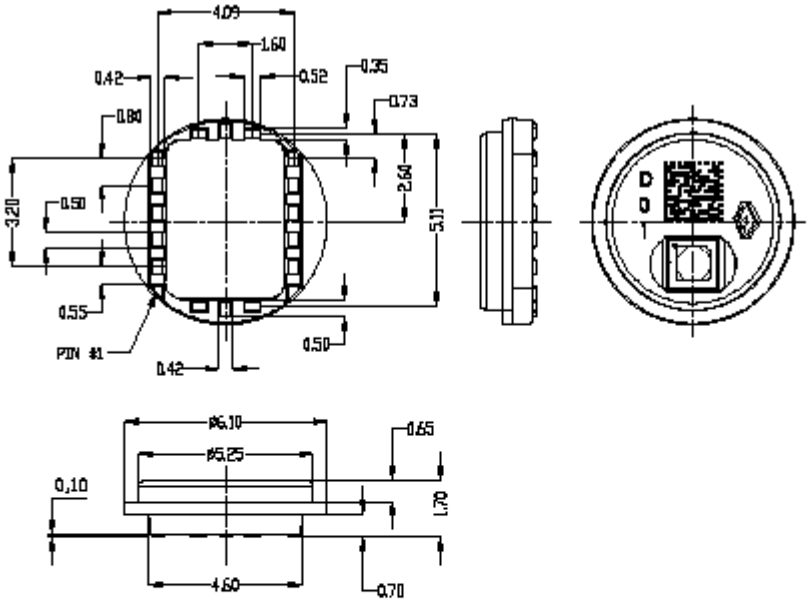
Figure 1 Pin arrangement of SCP1000

Note: In order to ensure the contact reliability, both pins in the pin pairs 1,16 and 14,15 (which are internally connected together) should be soldered.

PWB Pad Layout

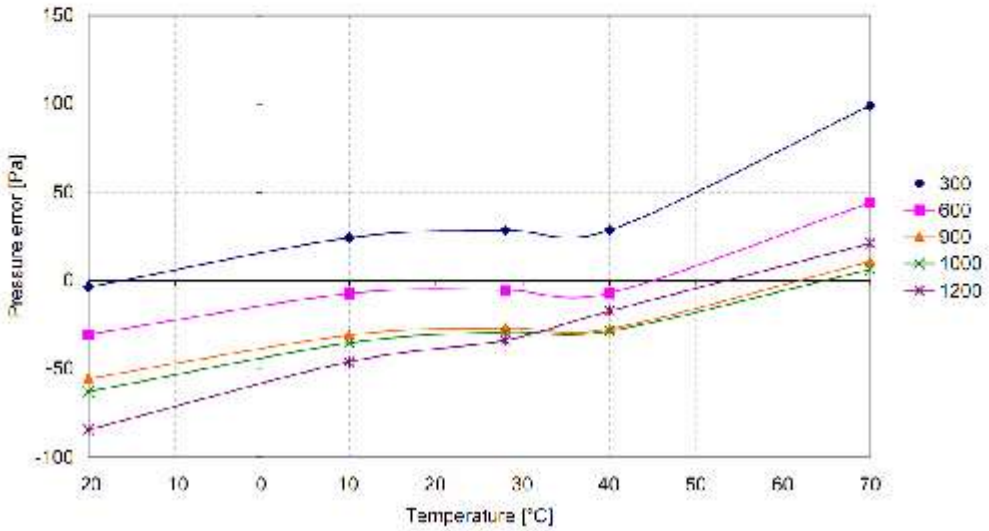


Dimensions

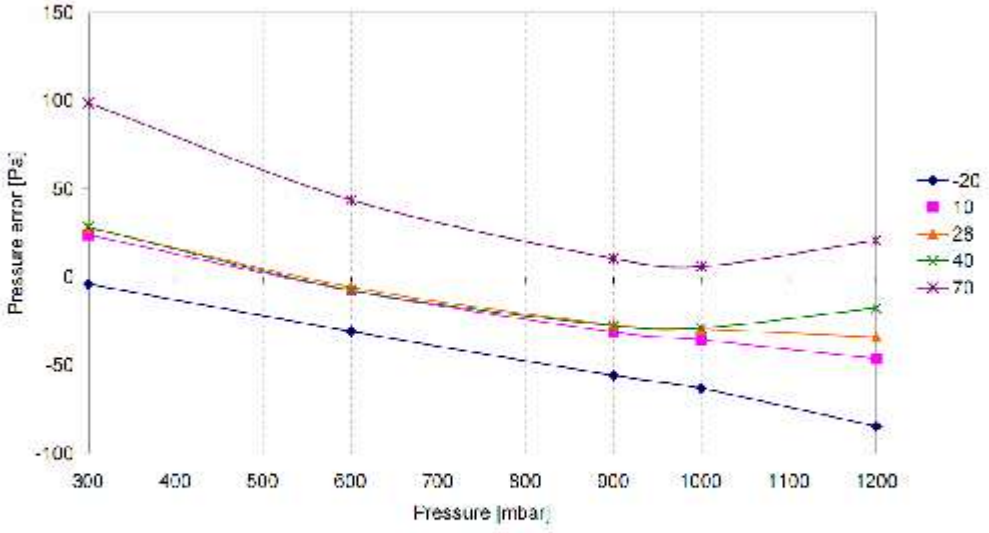


Typical Performance characteristics

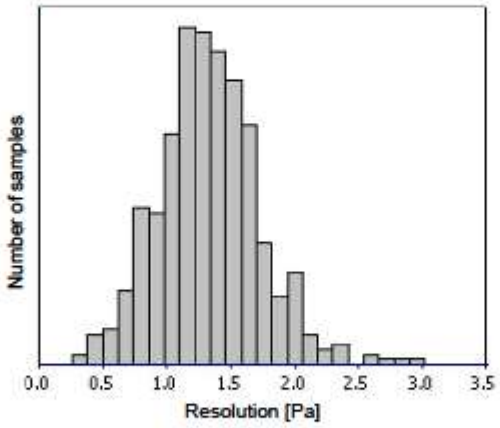
Pressure error over temperature range



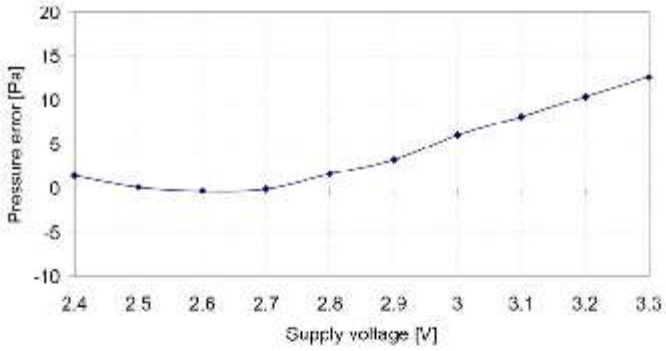
Pressure error over pressure range



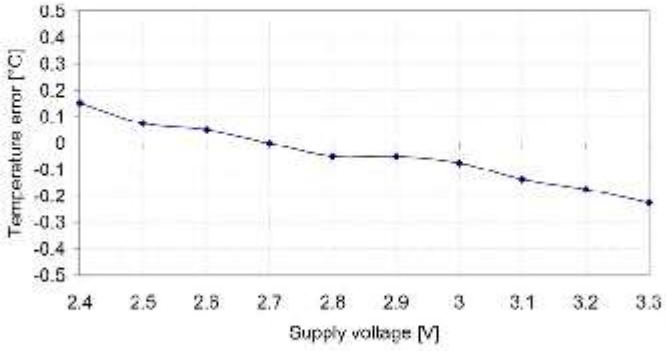
Resolution distribution in high resolution measurement mode [Pa], N=534



Pressure error over supply voltage range



Temperature error over supply voltage range



APPENDIX E GPS GT-723F SPECIFICATION SHEETS

CanMore™

GT-723F

Fast Acquisition Enhanced Sensitivity 65 Channel GPS Smart Antennae

The GT-723F is a compact all-in-one GPS module solution intended for a broad range of Original Equipment Manufacturer (OEM) products, where fast and easy system integration and minimal development risk is required.

The receiver continuously tracks all satellites in view and provides accurate satellite positioning data. The GT-723F is optimized for applications requiring high-performance, low cost, and maximum flexibility; suitable for a wide range of OEM configurations including handhelds, sensors, asset tracking, PDA-centric personal navigation system, and vehicle navigation products.

Its 65 parallel channels and Venus 6 search bins provide fast satellite signal acquisition and short startup time. Acquisition sensitivity of –155dBm and tracking sensitivity of –160dBm offers good navigation performance even in urban canyons having limited sky view..

Satellite-based augmentation systems, such as WAAS and EGNOS, are supported to yield improved accuracy. Besides it also supports A-GPS function and fixed in the short time.

RS232-level serial interface are provided on the interface connector. Supply voltage of 3.0V~6.0V is supported.

Users can modify NMEA sentences or Binary code by the extra flash memory.

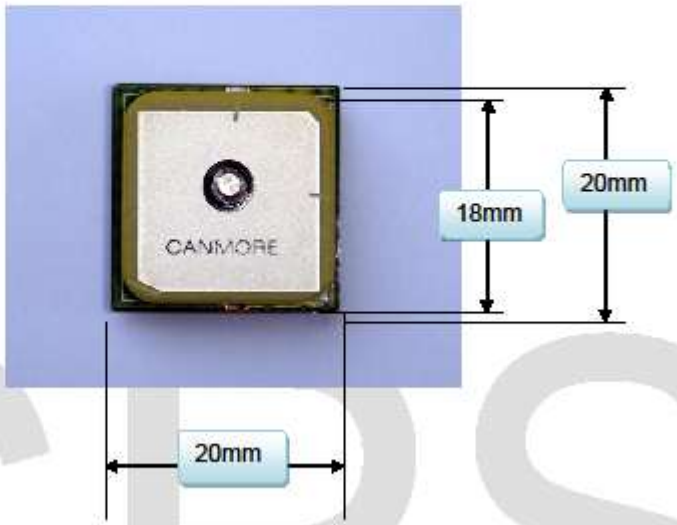
FEATURES

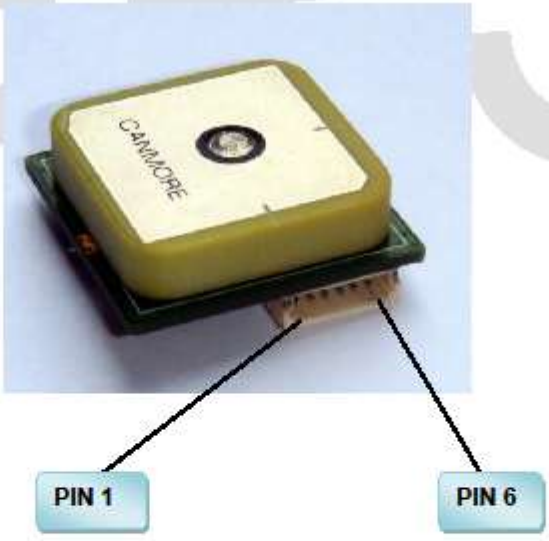
- Acquire and track 65 satellites simultaneously
- Venus 6 correlators
- Signal detection better than -160dBm
- Reacquisition sensitivity –155dBm
- Cold start < 30 seconds at –147dBm
- Hot start < 1sec under open sky
- 5m CEP accuracy
- SBAS (WAAS, EGNOS) support
- Support A-GPS function
- < 30mA
- Built-in passive antenna
- Flash-memory based
- Smallest Dimension: 20 x 20 x 8.8 mm

TECHNICAL SPECIFICATIONS

Receiver Type	65 parallel channels, L1 C/A code
Accuracy	Position 2.5m CEP Velocity 0.1m/sec Time 250ns
Startup Time (average)	< 1sec hot start < 30sec cold start
Signal Reacquisition	1s
Sensitivity	-155dBm acquisition -160dBm tracking
A-GPS	Support PromptFix® AGPS
Update Rate	1Hz standard (5Hz/10Hz special order)
Dynamics	4G (39.2m/sec2)
Operational Limits	Altitude < 18,000m or velocity < 515m/s (COCOM limit, either may be exceeded but not both)
Serial Interface	3.3V RS232 level
Protocol	NMEA-0183 V3.01 GPGGA, GPGLL, GPGSA, GPGSV, GPRMC, GPVTG, GPZDA ,9600 baud, 8, N, 1
Datum	Default WGS-84 User definable
Input Voltage	3.0V~6.0V
Input Current	~50mA acquisition ~30mA tracking
Dimension	20.0mm L x 20.0mm W x 8.8mm H
Weight:	5g
Operating Temperature	-40ºC ~ +85ºC
Humidity	5% ~ 95%

Mechanical Dimension





GT-720F Lateral View

PINOUT DESCRIPTION

Pin Number	Signal Name	Description
1	Ground	Power and signal ground
2	Power	3.0V ~ 5.0V DC input
3	Serial Data In 2	Asynchronous serial input at RS-232 level, to input command message
4	Serial Data Out 2	Asynchronous serial output at RS-232 level, to output NMEA message
5	Serial Data In 1	Asynchronous serial input at TTL level, to input command message
6	Serial Data Out 2	Asynchronous serial output at TTL level, to output NMEA message

NMEA Messages

The serial interface protocol is based on the National Marine Electronics Association's NMEA 0183 ASCII interface specification. This standard is fully define in "NMEA 0183, Version 3.01" The standard may be obtained from NMEA, www.nmea.org

GGA - GPS FIX DATA
Time, position and position-fix related data (number of satellites in use, HDOP, etc.).

Format:
$GPGGA,<1>,<2>,<3>,<4>,<5>,<6>,<7>,<8>,<9>,M,<10>,M,<11>,<12>,*<13><CR><LF>

Example:
$GPGGA,104549.04,2447.2038,N,12100.4990,E,1,06,01.7,00078.8,M,0016.3,M,,*5C<CR><LF>

Field	Example	Description
1	104549.04	UTC time in hhmmss.ss format, 000000.00 ~ 235959.99
2	2447.2038	Latitude in ddmm.mmmm format Leading zeros transmitted
3	N	Latitude hemisphere indicator, 'N' = North, 'S' = South
4	12100.4990	Longitude in dddmm.mmmm format Leading zeros transmitted
5	E	Longitude hemisphere indicator, 'E' = East, 'W' = West
6	1	Position fix quality indicator 0: position fix unavailable 1: valid position fix, SPS mode 2: valid position fix, differential GPS mode
7	06	Number of satellites in use, 00 ~ 12
8	01.7	Horizontal dilution of precision, 00.0 ~ 99.9
9	00078.8	Antenna height above/below mean sea level, -9999.9 ~ 17999.9
10	0016.3	Geoidal height, -999.9 ~ 9999.9
11		Age of DGPS data since last valid RTCM transmission in xxx format (seconds) NULL when DGPS not used
12		Differential reference station ID, 0000 ~ 1023 NULL when DGPS not used
13	5C	Checksum

Note: The checksum field starts with a '*' and consists of 2 characters representing a hex number. The checksum is the exclusive OR of all characters between '$' and '*'.

GLL - LATITUDE AND LONGITUDE, WITH TIME OF POSITION FIX AND STATUS
Latitude and longitude of current position, time, and status.

Format:
$GPGLL,<1>,<2>,<3>,<4>,<5>,<6>,<7>*<8><CR><LF>

Example:
$GPGLL,2447.2073,N,12100.5022,E,104548.04,A,A*65<CR><LF>

Field	Example	Description
1	2447.2073	Latitude in ddmm.mmmm format Leading zeros transmitted
2	N	Latitude hemisphere indicator, 'N' = North, 'S' = South
3	12100.5022	Longitude in dddmm.mmmm format Leading zeros transmitted
4	E	Longitude hemisphere indicator, 'E' = East, 'W' = West
5	104548.04	UTC time in hhmmss.ss format, 000000.00 ~ 235959.99
6	A	Status, 'A' = valid position, 'V' = navigation receiver warning
7	A	Mode indicator 'N' = Data invalid 'A' = Autonomous 'D' = Differential 'E' = Estimated
8	65	Checksum

GSA - GPS DOP AND ACTIVE SATELLITES
GPS receiver operating mode, satellites used for navigation, and DOP values.

Format:
$GPGSA,<1>,<2>,<3>,<3>,<3>,<3>,<3>,<3>,<3>,<3>,<3>,<3>,<3>,<3>,<3>,<4>,<5>,<6>*<7><CR><LF>

Example:
$GPGSA,A,3,26,21,,,09,17,,,,,,,10.8,02.1,10.6*07<CR><LF>

Field	Example	Description
1	A	Mode, 'M' = Manual, 'A' = Automatic
2	3	Fix type, 1 = not available, 2 = 2D fix, 3 = 3D fix
3	26,21,,,09,17,,,,,,	PRN number, 01 to 32, of satellite used in solution, up to 12 transmitted
4	10.8	Position dilution of precision, 00.0 to 99.9
5	02.1	Horizontal dilution of precision, 00.0 to 99.9
6	10.6	Vertical dilution of precision, 00.0 to 99.9
7	07	Checksum

GSV - GPS SATELLITE IN VIEW
Number of satellites in view, PRN number, elevation angle, azimuth angle, and C/No. Only up to four satellite details are transmitted per message. Additional satellite in view information is sent in subsequent GSV messages.

Format:
$GPGSV,<1>,<2>,<3>,<4>,<5>,<6>,<7>,...,<4>,<5>,<6>,<7> *<8><CR><LF>

Example:
$GPGSV,2,1,08,26,50,016,40,09,50,173,39,21,43,316,38,17,41,144,42*7C<CR><LF>
$GPGSV,2,2,08,29,38,029,37,10,27,082,32,18,22,309,24,24,09,145,*7B<CR><LF>

Field	Example	Description
1	2	Total number of GSV messages to be transmitted
2	1	Number of current GSV message
3	08	Total number of satellites in view, 00 ~ 12
4	26	Satellite PRN number, GPS: 01 ~ 32, SBAS: 33 ~ 64 (33 = PRN120)
5	50	Satellite elevation number, 00 ~ 90 degrees
6	016	Satellite azimuth angle, 000 ~ 359 degrees
7	40	C/No, 00 ~ 99 dB Null when not tracking
8	7C	Checksum

RMC - RECOMMANDED MINIMUM SPECIFIC GPS/TRANSIT DATA
Time, date, position, course and speed data.

Format:
$GPRMC,<1>,<2>,<3>,<4>,<5>,<6>,<7>,<8>,<9>,<10>,<11>,<12>*<13><CR><LF>

Example:
$GPRMC,104549.04,A,2447.2038,N,12100.4990,E,016.0,221.0,250304,003.3,W,A*22<CR><LF>

Field	Example	Description
1	104549.04	UTC time in hhmmss.ss format, 000000.00 ~ 235959.99
2	A	Status, 'V' = navigation receiver warning, 'A' = valid position
3	2447.2038	Latitude in dddmm.mmmm format Leading zeros transmitted
4	N	Latitude hemisphere indicator, 'N' = North, 'S' = South
5	12100.4990	Longitude in dddmm.mmmm format Leading zeros transmitted
6	E	Longitude hemisphere indicator, 'E' = East, 'W' = West
7	016.0	Speed over ground, 000.0 ~ 999.9 knots
8	221.0	Course over ground, 000.0 ~ 359.9 degrees
9	250304	UTC date of position fix, ddmmyy format
10	003.3	Magnetic variation, 000.0 ~ 180.0 degrees
11	W	Magnetic variation direction, 'E' = East, 'W' = West
12	A	Mode indicator 'N' = Data invalid 'A' = Autonomous 'D' = Differential 'E' = Estimated
13	22	Checksum

VTG - COURSE OVER GROUND AND GROUND SPEED
Velocity is given as course over ground (COG) and speed over ground (SOG).

Format:
GPVTG,<1>,T,<2>,M,<3>,N,<4>,K,<5>*<6><CR><LF>

Example:
$GPVTG,221.0,T,224.3,M,016.0,N,0029.6,K,A*1F<CR><LF>

Field	Example	Description
1	221.0	True course over ground, 000.0 ~ 359.9 degrees
2	224.3	Magnetic course over ground, 000.0 ~ 359.9 degrees
3	016.0	Speed over ground, 000.0 ~ 999.9 knots
4	0029.6	Speed over ground, 0000.0 ~ 1800.0 kilometers per hour
5	A	Mode indicator 'N' = Data invalid 'A' = Autonomous 'D' = Differential 'E' = Estimated
6	1F	Checksum

ZDA TIME AND DATE

Format:
$GPZDA,<1>,<2>,<3>,<4>,<5>,<6>*<7><CR><LF>

Example:
$GPZDA,104548.04,25,03,2004,,*6C<CR><LF>

Field	Example	Description
1	104548.04	UTC time in hhmmss.ss format, 000000.00 ~ 235959.99
2	25	UTC time: day (01 ... 31)
3	03	UTC time: month (01 ... 12)
4	2004	UTC time: year (4 digit year)
5		Local zone hour Not being output by the receiver (NULL)
6		Local zone minutes Not being output by the receiver (NULL)
7	6C	Checksum

Binary Messages

See *Binary Message Protocol User's Guide* for detailed descriptions.

CanMore Electronics Co., LTD.

No. 40, Chenggonh 5th St., Jhubei City Hsinchu County, 302, Taiwan

Phone +886 3 6586046
Fax +886 3 6583940
Email: sales@canmore.com.tw
Website:http:// www.canmore.com.tw
http://canmorecorp.trustpass.alibaba.com/

APPENDIX F GYROSCOPE SPECIFICATION SHEET

LPR530AL

MEMS motion sensor: dual axis pitch and roll ±300°/s analog gyroscope

Preliminary data

Features

- 2.7 V to 3.6 V single-supply operation
- Wide operating temperature range (-40 °C to +85 °C)
- High stability overtemperature
- Analog absolute angular-rate output
- Two separate outputs for each axis (1x and 4x amplified)
- Integrated low-pass filters
- Low power consumption
- Embedded power-down
- Embedded self-test
- High shock and vibration survivability
- ECOPACK® RoHS and "Green" compliant (see *Section 5*)

Applications

- Pointing devices, remote and game controllers
- Motion control with user interface
- GPS navigation systems
- Industrial and robotics

Description

The LPR530AL is a low-power dual-axis micromachined gyroscope capable of measuring angular rate along pitch and roll axes.

It provides excellent temperature stability and high resolution over an extended operating temperature range (-40 °C to +85 °C).

LGA-16L (5x5x1.5mm)

The LPR530AL has a full scale of ±300 °/s and is capable of detecting rates with a -3 dB bandwidth up to 140 Hz.

The gyroscope is the combination of one actuator and one accelerometer integrated in a single micromachined structure.

It includes a sensing element composed by single driving mass, kept in continuous oscillating movement and able to react when an angular rate is applied based on the Coriolis principle.

A CMOS IC provides the measured angular rate to the external world through an analog output voltage, allowing high level of integration and production trimming to better match sensing element characteristics.

ST's gyroscope family leverages on robust and mature manufacturing process already used for the production of micromachined accelerometers.

ST is already in the field with several hundreds million sensors with excellent acceptance from the market in terms of quality, reliability and performance.

LPR530AL is provided in plastic land grid array (LGA) package. Several years ago ST pioneered successfully the usage of this package for accelerometers. Today ST has the widest manufacturing capability and strongest expertise in the world for production of sensor in plastic LGA package.

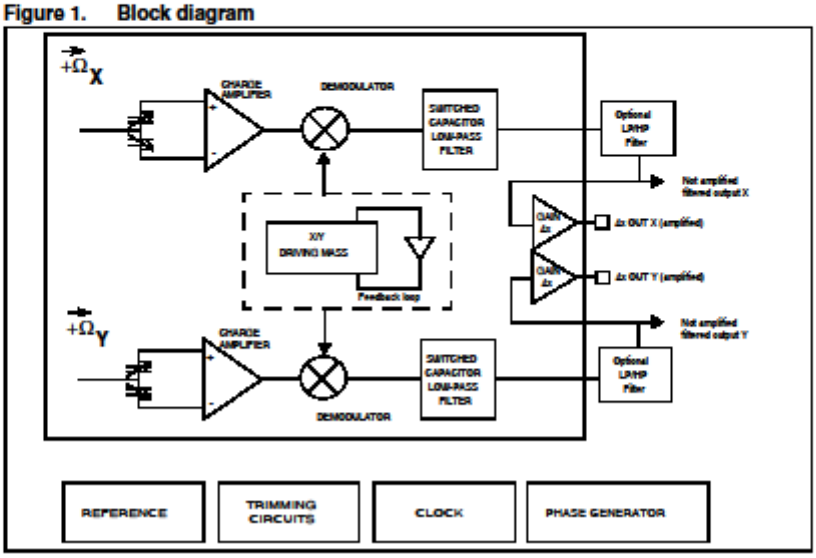
Table 1. Device summary

Order code	Temperature range (°C)	Package	Packing
LPR530AL	-40 to +85	LGA-16 (5x5x1.5)	Tray
LPR530ALTR	-40 to +85	LGA-16 (5x5x1.5)	Tape and reel

Contents

1 Block diagram and pin description

Figure 1. Block diagram



1.1 Pin description

Figure 2. Pin connection

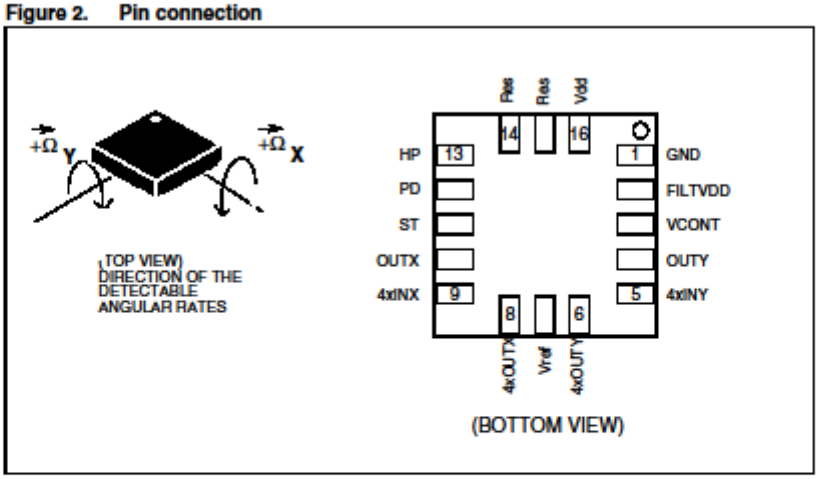


Table 2. Pin description

Pin #	Pin name	Analog function
1	GND	0V supply voltage
2	FILTVDD	PLL filter connection pin #2
3	VCONT	PLL filter connection pin #1
4	OUTY	Not amplified output
5	4xINY	Input of 4x amplifier
6	4xOUTY	Y rate signal output voltage (amplified)
7	Vref	Reference voltage
8	4xOUTX	X rate signal output voltage (amplified)
9	4xINX	Input of 4x amplifier
10	OUTX	Not amplified output
11	ST	Self-test (logic 0: normal mode; logic 1: self-test)
12	PD	Power-down (logic 0: normal mode; logic 1: power-down mode)
13	HP	High pass filter reset (logic 0: normal operation mode; logic1: external high pass filter is reset)
14,15	Res	Reserved. Connect to Vdd
16	Vdd	Power supply

2 Mechanical and electrical specifications

2.1 Mechanical characteristics

Table 3. Mechanical characteristics @ Vdd = 3 V, T = 25 °C unless otherwise noted(1)

Symbol	Parameter	Test condition	Min.	Typ.(2)	Max.	Unit
FSA	Measurement range	4x OUT (amplified)		±300		°/s
FS		OUT (not amplified)		±1200		°/s
SoA	Sensitivity(3)	4x OUT (amplified)		3.33		mV/ °/s
So		OUT (not amplified)		0.83		mV/ °/s
SoDr	Sensitivity change vs temperature	Delta from 25°C		0.03		%/°C
Voff	Zero-rate level(3)			1.23		V
Vref	Reference voltage			1.23		V
OffDr	Zero-rate level change Vs temperature	Delta from 25°C		0.05		°/s/°C
NL	Non linearity	Best fit straight line		±1		% FS
BW	Bandwidth(4)			140		Hz
Rn	Rate noise density			0.035		°/s / $\sqrt{Hz}$
Top	Operating temperature range		-40		+85	°C

1. The product is factory calibrated at 3 V. The operational power supply range is specified in *Table 4*.
2. Typical specifications are not guaranteed
3. Sensitivity and zero-rate level are not ratiometric to supply voltage
4. The product is capable of measuring angular rates extending from DC to the selected BW.

2.2 Electrical characteristics

Table 4. Electrical characteristics @ Vdd =3 V, T=25 °C unless otherwise noted(1)

Symbol	Parameter	Test condition	Min.	Typ.(2)	Max.	Unit
Vdd	Supply voltage		2.7	3	3.6	V
Idd	Supply current	PD pin connected to GND		6.8		mA
IddPdn	Supply current in power-down mode	PD pin connected to Vdd		1	5	µA
VST	Self-test input	Logic 0 level	0		0.2*Vdd	V
		Logic 1 level	0.8*Vdd		Vdd	
VPD	Power-down input	Logic 0 level	0		0.2*Vdd	V
		Logic 1 level	0.8*Vdd		Vdd	
Top	Operating temperature range		-40		+85	°C

1. The product is factory calibrated at 3 V
2. Typical specifications are not guaranteed

2.3 Absolute maximum ratings

Stresses above those listed as "Absolute maximum ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device under these conditions is not implied. Exposure to maximum rating conditions for extended periods may affect device reliability.

Table 5. Absolute maximum ratings

Symbol	Ratings	Maximum value	Unit
Vdd	Supply voltage	-0.3 to 6	V
Vin	Input voltage on any control pin (PD, ST)	-0.3 to Vdd +0.3	V
T_{STG}	Storage temperature range	-40 to +125	°C
A	Acceleration	3000 *g* for 0.5 ms	
		10000 *g* for 0.1 ms	
ESD	Electrostatic discharge protection	2 (HBM)	kV

This is a mechanical shock sensitive device, improper handling can cause permanent damage to the part

This is an ESD sensitive device, improper handling can cause permanent damage to the part

3 Terminology

3.1 Sensitivity

An angular rate gyroscope is a device that produces a positive-going output voltage for counterclockwise rotation around the sensitive axis considered. Sensitivity describes the gain of the sensor and can be determined by applying a defined angular velocity to it. This value changes very little over temperature and time.

3.2 Zero-rate level

Zero-rate level describes the actual output signal if there is no angular rate present. The zero-rate level of precise MEMS sensors is, to some extent, a result of stress to the sensor and therefore zero-rate level can slightly change after mounting the sensor onto a printed circuit board or after exposing it to extensive mechanical stress. This value changes very little over temperature and time.

3.3 Self-test

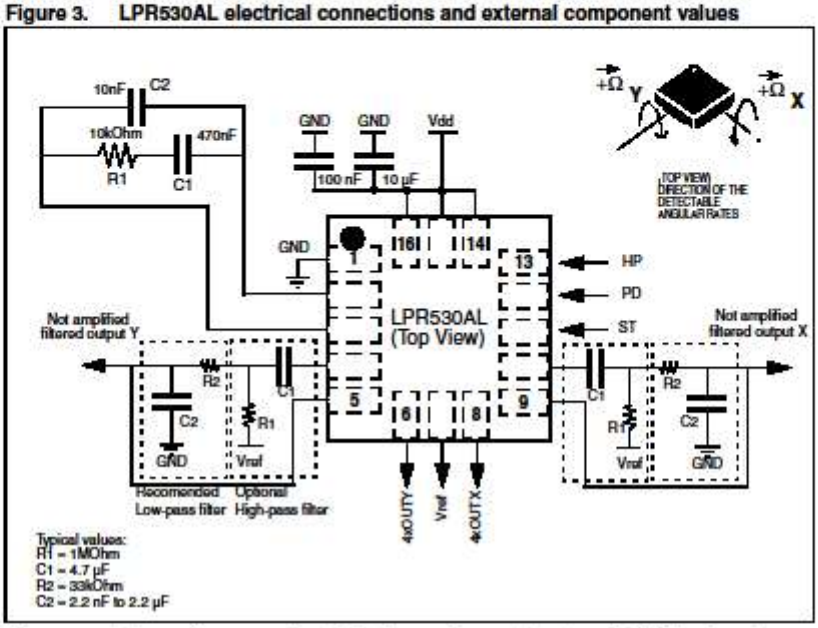
Self-test allows testing of the mechanical and electrical part of the sensor, allowing the seismic mass to be moved by means of an electrostatic test-force. The self-test function is off when the ST pin is connected to GND. When the ST pin is tied to Vdd, an actuation force is applied to the sensor, emulating a definite Coriolis force. In this case the sensor output will exhibit a voltage change in its DC level which is also dependent on the supply voltage. When ST is active, the device output level is given by the algebraic sum of the signals produced by the velocity acting on the sensor and by the electrostatic test-force. If the output signals change within the amplitude specified in *Table 3*, then the mechanical element is working properly and the parameters of the interface chip are within the defined specifications.

3.4 High pass filter reset (HP)

The LPR530AL provides the possibility to reset the optional external high pass filter by applying a high logic value to the HP pad. This procedure ensures faster response, especially during overload conditions. Moreover, this operation is suggested each time the device is powered.

4 Application hints

Figure 3. LPR530AL electrical connections and external component values



Power supply decoupling capacitors (100 nF ceramic or polyester + 10 µF Aluminum) should be placed as near as possible to the device (common design practice).

The LPR530AL allows band limiting the output rate response through the use of an external low pass filter (suggested) and/or high pass filter (optional) in addition to the embedded low pass filter (f_t = 140 Hz).

4xOUTX and 4xOUTY are respectively OUTX and OUTY amplified outputs lines, internally buffered to ensure low output impedance.

If external high pass or low pass filtering is not applied it is mandatory to short-circuit respectively pad 4 to pad 5 and pad 9 to pad 10 when amplified outputs are used.

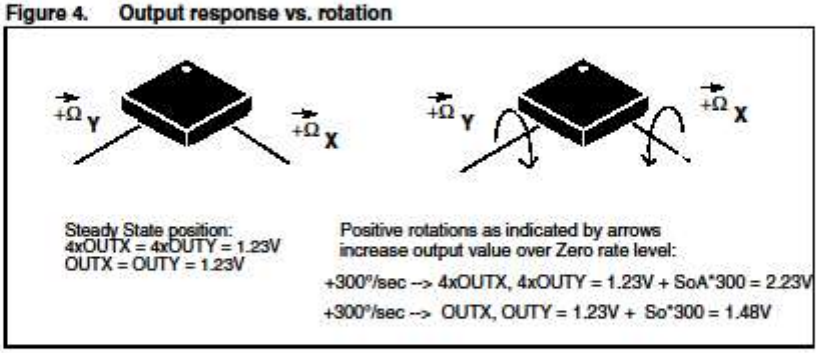
When only non-amplified outputs are used (OUTX/Y), it is suggested to set pads 5 and 9 to fixed reference voltages (GND/Vref).

When high pass filter is applied to not amplified output (OUTx), it is recommended to buffer the line before entering ADC for performance optimization.

The LPR530AL IC includes a PLL (Phase Locked Loop) circuit to synchronize driving and sensing interfaces. Capacitors and resistors must be added at **FILTVDD** and **VCONT** pins (as shown in *Figure 3*) to implement a low-pass filter.

4.1 Output response vs. rotation

Figure 4. Output response vs. rotation



4.2 Soldering information

The LGA package is compliant with the ECOPACK®, RoHS and "Green" standard.
It is qualified for soldering heat resistance according to JEDEC J-STD-020C.

Leave "pin 1 indicator" unconnected during soldering.

Land pattern and soldering recommendations are available at www.st.com.

5 Package information

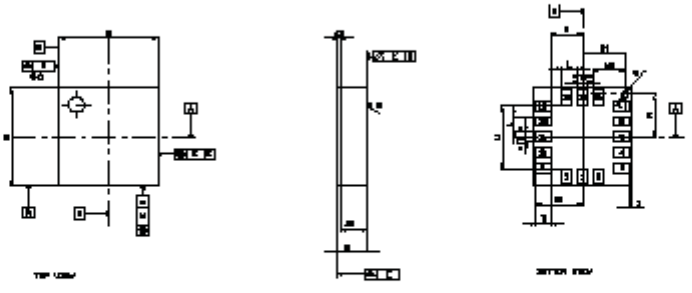
In order to meet environmental requirements, ST offers these devices in different grades of ECOPACK® packages, depending on their level of environmental compliance. ECOPACK® specifications, grade definitions and product status are available at: www.st.com. ECOPACK is an ST trademark.

Figure 5. LGA-16: mechanical data and package dimensions

Dimensions						
Ref.	mm			Inch		
	Min.	Typ.	Max.	Min.	Typ.	Max.
A1	1.46	1.5	1.6	0.057	0.059	0.063
A2			1.33			0.052
A3	0.16	0.2	0.24	0.006	0.008	0.009
C		0.3			0.012	
D1	4.85	5	5.15	0.191	0.197	0.203
E1	4.85	5	5.15	0.191	0.197	0.203
L		0.8			0.031	
L1		3.2			0.126	
M		1.6			0.062	
M1	2.15	2.175	2.20	0.085	0.086	0.087
M2		1.625			0.064	
N		2.175			0.086	
N1		2.4			0.094	
T1		0.8			0.031	
T2	0.475	0.5	0.525	0.019	0.020	0.021
R	1.2		1.6	0.047		0.063
S		0.1			0.004	
h		0.15			0.006	
k		0.05			0.002	
j		0.1			0.004	

Outline and mechanical data

LGA-16 (5x5x1.5mm)
Land Grid Array Package



6 Revision history

Table 6. Document revision history

Date	Revision	Changes
04-Jun-2009	1	Initial release
06-Jul-2009	2	Small text changes to improve readability. Updated *Table 4*

APPENDIX G XBEE SPECIFICATION SHEET

1. XBee®/XBee-PRO® RF Modules

The XBee and XBee-PRO RF Modules were engineered to meet IEEE 802.15.4 standards and support the unique needs of low-cost, low-power wireless sensor networks. The modules require minimal power and provide reliable delivery of data between devices.

The modules operate within the ISM 2.4 GHz frequency band and are pin-for-pin compatible with each other.

Key Features

Long Range Data Integrity

XBee

- Indoor/Urban: up to 100' (30 m)
- Outdoor line-of-sight: up to 300' (90 m)
- Transmit Power: 1 mW (0 dBm)
- Receiver Sensitivity: -92 dBm

XBee-PRO

- Indoor/Urban: up to 300' (90 m), 200' (60 m) for International variant
- Outdoor line-of-sight: up to 1 mile (1600 m), 2500' (750 m) for International variant
- Transmit Power: 63mW (18dBm), 10mW (10dBm) for International variant
- Receiver Sensitivity: -100 dBm

RF Data Rate: 250,000 bps

Advanced Networking & Security

Retries and Acknowledgements

DSSS (Direct Sequence Spread Spectrum)

Each direct sequence channels has over 65,000 unique network addresses available

Source/Destination Addressing

Unicast & Broadcast Communications

Point-to-point, point-to-multipoint and peer-to-peer topologies supported

Low Power

XBee

- TX Peak Current: 45 mA (@3.3 V)
- RX Current: 50 mA (@3.3 V)
- Power-down Current: < 10 µA

XBee-PRO

- TX Peak Current: 250mA (150mA for international variant)
- TX Peak Current (RPSMA module only): 340mA (180mA for international variant
- RX Current: 55 mA (@3.3 V)
- Power-down Current: < 10 µA

ADC and I/O line support

Analog-to-digital conversion, Digital I/O

I/O Line Passing

Easy-to-Use

No configuration necessary for out-of box RF communications

Free X-CTU Software (Testing and configuration software)

AT and API Command Modes for configuring module parameters

Extensive command set

Small form factor

Worldwide Acceptance

FCC Approval (USA) Refer to Appendix A [p64] for FCC Requirements. Systems that contain XBee®/XBee-PRO® RF Modules inherit Digi Certifications.

ISM (Industrial, Scientific & Medical) **2.4 GHz frequency band**

Manufactured under **ISO 9001:2000** registered standards

XBee®/XBee-PRO® RF Modules are optimized for use in the United States, Canada, Australia, Japan, and Europe. Contact Digi for complete list of government agency approvals.

Specifications

Table 1-01. Specifications of the XBee®/XBee-PRO® RF Modules

Specification	XBee	XBee-PRO
Performance		
Indoor/Urban Range	Up to 100 ft (30 m)	Up to 300 ft. (90 m), up to 200 ft (60 m) International variant
Outdoor RF line-of-sight Range	Up to 300 ft (90 m)	Up to 1 mile (1600 m), up to 2500 ft (750 m) international variant
Transmit Power Output (software selectable)	1mW (0 dBm)	63mW (18dBm)* 10mW (10 dBm) for International variant
RF Data Rate	250,000 bps	250,000 bps
Serial Interface Data Rate (software selectable)	1200 bps - 250 kbps (non-standard baud rates also supported)	1200 bps - 250 kbps (non-standard baud rates also supported)
Receiver Sensitivity	-92 dBm (1% packet error rate)	-100 dBm (1% packet error rate)
Power Requirements		
Supply Voltage	2.8 – 3.4 V	2.8 – 3.4 V
Transmit Current (typical)	45mA (@ 3.3 V)	250mA (@3.3 V) (150mA for international variant) RPSMA module only: 340mA (@3.3 V) (180mA for international variant)
Idle / Receive Current (typical)	50mA (@ 3.3 V)	55mA (@ 3.3 V)
Power-down Current	< 10 µA	< 10 µA
General		
Operating Frequency	ISM 2.4 GHz	ISM 2.4 GHz
Dimensions	0.960" x 1.087" (2.438cm x 2.761cm)	0.960" x 1.297" (2.438cm x 3.294cm)
Operating Temperature	-40 to 85º C (industrial)	-40 to 85º C (industrial)
Antenna Options	Integrated Whip, Chip or U.FL Connector, RPSMA Connector	Integrated Whip, Chip or U.FL Connector, RPSMA Connector
Networking & Security		
Supported Network Topologies	Point-to-point, Point-to-multipoint & Peer-to-peer	
Number of Channels (software selectable)	16 Direct Sequence Channels	12 Direct Sequence Channels
Addressing Options	PAN ID, Channel and Addresses	PAN ID, Channel and Addresses
Agency Approvals		
United States (FCC Part 15.247)	OUR-XBEE	OUR-XBEEPRO
Industry Canada (IC)	4214A XBEE	4214A XBEEPRO
Europe (CE)	ETSI	ETSI (Max. 10 dBm transmit power output)*
Japan	R201WW07215214	R201WW08215111 (Max. 10 dBm transmit power output)*
Australia	C-Tick	C-Tick

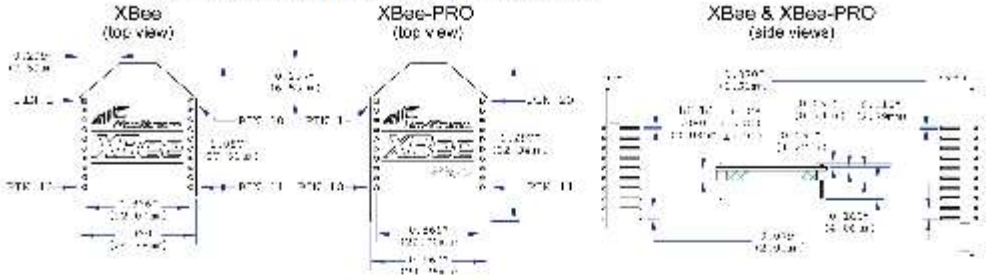
* See Appendix A for region-specific certification requirements.

Antenna Options: The ranges specified are typical when using the integrated Whip (1.5 dBi) and Dipole (2.1 dBi) antennas. The Chip antenna option provides advantages in its form factor; however, it typically yields shorter range than the Whip and Dipole antenna options when transmitting outdoors.For more information, refer to the "XBee Antennas" Knowledgebase Article located on Digi's Support Web site

Mechanical Drawings

Figure 1-01. Mechanical drawings of the XBee®/XBee-PRO® RF Modules (antenna options not shown)

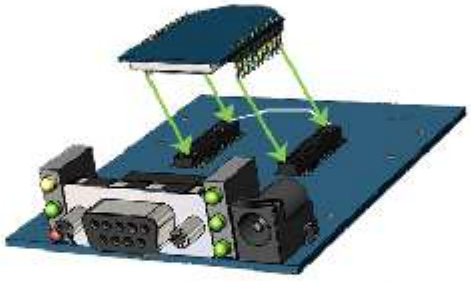
The XBee and XBee-PRO RF Modules are pin-for-pin compatible.



Mounting Considerations

The XBee®/XBee-PRO® RF Module was designed to mount into a receptacle (socket) and therefore does not require any soldering when mounting it to a board. The XBee Development Kits contain RS-232 and USB interface boards which use two 20-pin receptacles to receive modules.

Figure 1-02. XBee Module Mounting to an RS-232 Interface Board.



The receptacles used on Digi development boards are manufactured by Century Interconnect. Several other manufacturers provide comparable mounting solutions; however, Digi currently uses the following receptacles:

- Through-hole single-row receptacles -
  Samtec P/N: MMS-110-01-L-SV (or equivalent)
- Surface-mount double-row receptacles -
  Century Interconnect P/N: CPRMSL20-D-0-1 (or equivalent)
- Surface-mount single-row receptacles -
  Samtec P/N: SMM-110-02-SM-S

Digi also recommends printing an outline of the module on the board to indicate the orientation the module should be mounted.

Pin Signals

Figure 1-03. XBee®/XBee-PRO® RF Module Pin Numbers
(top sides shown - shields on bottom)

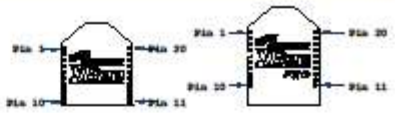


Table 1-02. Pin Assignments for the XBee and XBee-PRO Modules
(Low-asserted signals are distinguished with a horizontal line above signal name.)

Pin #	Name	Direction	Description
1	VCC	-	Power supply
2	DOUT	Output	UART Data Out
3	DIN / $\overline{\text{CONFIG}}$	Input	UART Data In
4	DIO8*	Output	Digital Output 8
5	$\overline{\text{RESET}}$	Input	Module Reset (reset pulse must be at least 200 ns)
6	PWM0 / RSSI	Output	PWM Output 0 / RX Signal Strength Indicator
7	PWM1	Output	PWM Output 1
8	[reserved]	-	Do not connect
9	$\overline{\text{DTR}}$ / SLEEP_RQ / DI8	Input	Pin Sleep Control Line or Digital Input 8
10	GND	-	Ground
11	AD4 / DIO4	Either	Analog Input 4 or Digital I/O 4
12	$\overline{\text{CTS}}$ / DIO7	Either	Clear-to-Send Flow Control or Digital I/O 7
13	ON / $\overline{\text{SLEEP}}$	Output	Module Status Indicator
14	VREF	Input	Voltage Reference for A/D Inputs
15	Associate / AD5 / DIO5	Either	Associated Indicator, Analog Input 5 or Digital I/O 5
16	$\overline{\text{RTS}}$ / AD6 / DIO6	Either	Request-to-Send Flow Control, Analog Input 6 or Digital I/O 6
17	AD3 / DIO3	Either	Analog Input 3 or Digital I/O 3
18	AD2 / DIO2	Either	Analog Input 2 or Digital I/O 2
19	AD1 / DIO1	Either	Analog Input 1 or Digital I/O 1
20	AD0 / DIO0	Either	Analog Input 0 or Digital I/O 0

* Function is not supported at the time of this release

Design Notes:

- Minimum connections: VCC, GND, DOUT & DIN
- Minimum connections for updating firmware: VCC, GND, DIN, DOUT, RTS & DTR
- Signal Direction is specified with respect to the module
- Module includes a 50k Ω pull-up resistor attached to $\overline{\text{RESET}}$
- Several of the input pull-ups can be configured using the PR command
- Unused pins should be left disconnected

Electrical Characteristics

Table 1-03. DC Characteristics (VCC = 2.8 - 3.4 VDC)

Symbol	Characteristic	Condition	Min	Typical	Max	Unit
V_{IL}	Input Low Voltage	All Digital Inputs	-	-	0.35 * VCC	V
V_{IH}	Input High Voltage	All Digital Inputs	0.7 * VCC	-	-	V
V_{OL}	Output Low Voltage	I_{OL} = 2 mA, VCC >= 2.7 V	-	-	0.5	V
V_{OH}	Output High Voltage	I_{OH} = -2 mA, VCC >= 2.7 V	VCC - 0.5	-	-	V
I_{IN}	Input Leakage Current	V_{IN} = VCC or GND, all inputs, per pin	-	0.025	1	µA
I_{OZ}	High Impedance Leakage Current	V_{IN} = VCC or GND, all I/O High-Z, per pin	-	0.025	1	µA
TX	Transmit Current	VCC = 3.3 V	-	45 (XBee) / 215, 140 (PRO, Int)	-	mA
RX	Receive Current	VCC = 3.3 V	-	50 (XBee) / 55 (PRO)	-	mA
PWR-DWN	Power-down Current	SM parameter = 1	-	< 10	-	µA

Table 1-04. ADC Characteristics (Operating)

Symbol	Characteristic	Condition	Min	Typical	Max	Unit
V_{REFH}	VREF - Analog-to-Digital converter reference range		2.08	-	V_{DDAD}*	V
I_{REF}	VREF - Reference Supply Current	Enabled	-	200	-	µA
		Disabled or Sleep Mode	-	< 0.01	0.02	µA
V_{INDC}	Analog Input Voltage[1]		V_{SSAD} - 0.3	-	V_{DDAD} + 0.3	V

1. Maximum electrical operating range, not valid conversion range.

* V_{DDAD} is connected to VCC.

Table 1-05. ADC Timing/Performance Characteristics[1]

Symbol	Characteristic	Condition	Min	Typical	Max	Unit
R_{AS}	Source Impedance at Input[2]		-	-	10	kΩ
V_{AIN}	Analog Input Voltage[3]		V_{REFL}		V_{REFH}	V
RES	Ideal Resolution (1 LSB)[4]	2.08V ≤ V_{DDAD} ≤ 3.6V	2.031	-	3.516	mV
DNL	Differential Non-linearity[5]		-	±0.5	±1.0	LSB
INL	Integral Non-linearity[6]		-	±0.5	±1.0	LSB
E_{ZS}	Zero-scale Error[7]		-	±0.4	±1.0	LSB
F_{FS}	Full-scale Error[8]		-	±0.4	±1.0	LSB
E_{IL}	Input Leakage Error[9]		-	±0.05	±5.0	LSB
E_{TU}	Total Unadjusted Error[10]		-	±1.1	±2.5	LSB

1. All ACCURACY numbers are based on processor and system being in WAIT state (very little activity and no IO switching) and that adequate low-pass filtering is present on analog input pins (filter with 0.01 µF to 0.1 µF capacitor between analog input and VREFL). Failure to observe these guidelines may result in system or microcontroller noise causing accuracy errors which will vary based on board layout and the type and magnitude of the activity.
Data transmission and reception during data conversion may cause some degradation of these specifications, depending on the number and timing of packets. It is advisable to test the ADCs in your installation if best accuracy is required.
2. R_{AS} is the real portion of the impedance of the network driving the analog input pin. Values greater than this amount may not fully charge the input circuitry of the ATD resulting in accuracy error.
3. Analog input must be between V_{REFL} and V_{REFH} for valid conversion. Values greater than V_{REFH} will convert to $3FF.
4. The resolution is the ideal step size or 1LSB = (V_{REFH}-V_{REFL})/1024
5. Differential non-linearity is the difference between the current code width and the ideal code width (1LSB). The current code width is the difference in the transition voltages to and from the current code.
6. Integral non-linearity is the difference between the transition voltage to the current code and the adjusted ideal transition voltage for the current code. The adjusted ideal transition voltage is (Current Code-1/2)*(1/(((V_{REFH}+E_{FS})-(V_{REFL}+E_{ZS}))).
7. Zero-scale error is the difference between the transition to the first valid code and the ideal transition to that code. The Ideal transition voltage to a given code is (Code-1/2)*(1/(V_{REFH}-V_{REFL})).
8. Full-scale error is the difference between the transition to the last valid code and the ideal transition to that code. The ideal transition voltage to a given code is (Code-1/2)*(1/(V_{REFH}-V_{REFL})).
9. Input leakage error is error due to input leakage across the real portion of the impedance of the network driving the analog pin. Reducing the impedance of the network reduces this error.

10. Total unadjusted error is the difference between the transition voltage to the current code and the ideal straight-line transfer function. This measure of error includes inherent quantization error (1/2LSB) and circuit error (differential, integral, zero-scale, and full-scale) error. The specified value of E_{TU} assumes zero E_{IL} (no leakage or zero real source impedance).

APPENDIX H C/C++ BASIC DATA TYPES, OPERATORS AND CONTROL STATEMENTS

Built-in data types

- Character - represents an alphabet, number or symbol character
  - char
- Integer - represents an signed or unsigned integer
  - signed char, short, int, long
  - unsigned char, unsigned short, unsigned int, unsigned long
- Floating point number - in IEEE single or double precision format
  - float
  - double
- Boolean - true or false
  - bool

Arithmetic operations

Basic mathematical opeprators

- Addition (**+**)
- Subtraction (**-**)
- Multiplication (*****)
- Division (**/**)
- Modulus (**%**)

Short-hand notation to perform a mathematical operation and an assignment at the same time

+=, **-=**, ***=**, **/=**, **%=**

- Add 4 to a variable x and assign x to the result

x += 4;

Increment and decrement operators

- Increment operator (**++**)
  i++ is a short hand for **i = i + 1** or **i += 1**
- Decrement operator (**--**)
  i-- is a short hand for **i = i - 1** or **i -= 1**

Bitwise operators

- Bitwise operators perform Boolean algebra on the corresponding bits in the arguments to produce the result

- Bitwise-and (**&**)

```
3 & 5 produces 1
2 & 4 produces 0
```

- Bitwise-or (**|**)

```
3 | 5 produces 7
2 | 4 produces 6
```

- Bitwise-xor (**^**)

```
3 ^ 5 produces 6
2 ^ 4 produces 6
```

- Bitwise operators can be combined with **=** like **&=**, **|=** and **^=**

Shift operators

- Left shift operator(**<<**)produces the operand to the left of the operator shifted to the left by the number of bits specified after the operator

```
1 << 1 produces 2
1 << 5 produces 32
7 << 3 produces 56
```

- Right shift operator(**>>**)produces the operand to the left of the operator shifted to the right by the number of bits specified after the operator

```
1 >> 1 produces 0
5 >> 1 produces 2
7 >> 2 produces 1
```

Relational operators

Relational operators establish a relationship between the values of the operands. They produce a Boolean **`true`** if the relationship is true, and **`false`** if the relationship is false.

- Less than (**<**)
- Less than or equal to (**<=**)
- Greater than (**>**)
- Greater than or equal to (**>=**)
- equivalent (**==**)

- not equivalent (**!=**)

Logical operators

- The binary logical operators produce a true or false based on the logical relationship of its arguments

  - Logical AND (**&&**)
  - Logical OR (**||**)

- The unary logical not operator will take a Boolean and produce its negation

  - Logical NOT (**!**)

- An expression is **`true`** if it has a non-zero value, and **`false`** if it has a value of zero

- If you print a **`bool`**, you'll typically see a '1' for **`true`** and '0' for **`False`** Integer and bool conversion

Integer and bool conversion

- Integer to bool conversion

  - Non zero integers are converted to **`true`**
  - 0 is converted to **`false`**

- bool to Integer conversion

  - **`true`** is converted to 1
  - **`false`** is conveted to 0

if-else, while, do-while statements

if (*expression*)
    statement

if (*expression*)
    statement
else
    statement

while (*expression*)
    statement

```
do
        statement
while (expression);
```

Example:

```
if (n == -1)
        printf("error\n");

if (n == 0)
        n = 1;
else
        n = n + 1;

while (n < 100)
        n = n * n;

do
        n = n * n;
while (n < 100);
```

for statement

```
for (initialization; conditional; step)
        statement

for (int i = 0; i < 100; i++)
        printf("square of %d = %d\n", i, i * i);
```

switch statement

```
switch (selector) {
        case value1:
                statement;
                break;
        case value2:
                statement;
                break;
        ...
        default:
                statement;
}
```

Example:

```
switch (response) {
	case 'y':
		printf("Yes\n");
		break;
	case 'n':
		printf("No\n");
		break;
	case 'c':
		printf("Cancel\n");
		break;
	default:
		printf("Invalid\n");
}
```

CLTP Office

c/o University Space Engineering Consortium (UNISEC)

Central Yayoi 2Fl., 2-3-2 Yayoi, Bunkyo-ku, Tokyo, 113-0032, Japan

Tel :+81-3-5800-6645

Fax:+81-3-3868-2208

E-mail: secretariat@cltp.info

www.cltp.info

ISBN-978-4-906837-00-7

C3053