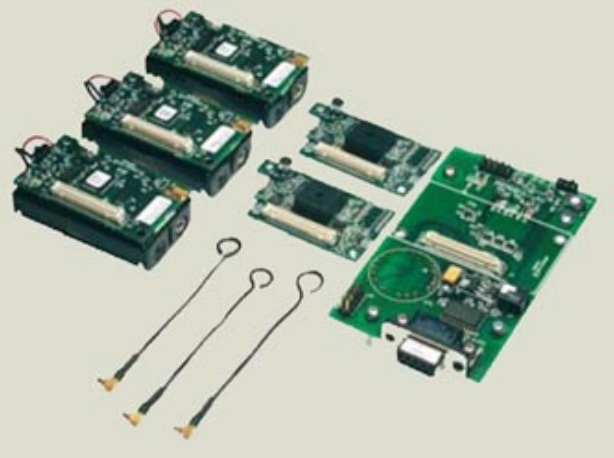
Beginner's Guide to Crossbow Motes



The Crossbow processor/radio boards, more commonly known as motes, allow multiple sensors distributed over a wide area to wirelessly transmit their data back to a base station attached to a computer. The motes run the operating system TinyOS, which handles power, radio transmission and networking transparent to the user. The network formed is adhoc, meaning the motes figure out how to form the most efficient network by themselves. The network also supports multihopping, allowing a mote out of range of the base station to pass its information from mote to mote until the data reaches the base station. With these tasks in place, you can concentrate on building sensing applications.

INTRODUCTION

The purpose of this tutorial is to get you up and running programming motes. This tutorial will tell you what to buy, how to install all the necessary software, how to work the software and how to program the mote with the equivalent of a "hello world" application. The code for the program will then be explained so that you can become familiar with the programming language and start programming your own applications.

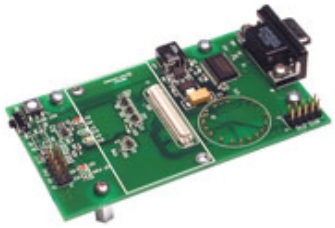
The rest of the tutorial is presented as follows:

NOTE: This tutorial is written for the MICA2 motes running TinyOS v1.1.8. Crossbow is constantly upgrading their hardware and software. The contents of this tutorial may not be directly applicable to current Crossbow releases.

MATERIALS

Crossbow offers two options: buy the motes in a kit or buy parts individually. The MICA2 Basic Kit (MOTE-KIT4x0) is a good starter. It contains everything you need to learn the basics and ascend the learning curve of mote development. As you become more comfortable with the mote you can expand upon this kit by purchasing individual components. If you wish to compose your own kit, the bare minimum you will need is: a "gateway" to connect the mote to the computer (used for programming and communication), 2 motes and a sensor board. The following parts are recommended:

ALL MATERIALS AVAILABLE FROM CROSSBOW



MIB510

QTY: 1 COST: $95.00

Serial "gateway" used to program and communicate with the motes.

NOTE: If your computer does not have a serial port, you will also need to buy a serial to USB adapter.

MPR400

QTY: 2 COST: $150.00 each

More commonly known as the MICA2 Mote. The 900 MHz motes offer greater bandwidth, which means more available channels. If you want to have several mote networks operating at once, this is the way to go. The 433 MHz motes get better range. Best for outdoor applications.

MTS300

QTY: 1 COST: $120.00

Basic sensor board that plugs into the mote. Contains light sensor, temperature sensor, microphone and sounder. More advanced sensor boards have accelerometers, magnetometers and GPS. Crossbow also sells Data Acquisition (DAQ) boards for custom sensor applications.

NOTE:It is also strongly recommended that you have a computer running Windows 2000 or XP. The computer must have a serial port or USB port with serial adapter. You must have administrator privileges and 1 GB of free space for installation.

BACKGROUND

The term "motes" refers to a general class of technologies aimed at having small, robust and versatile sensors that are easily deployable over a wide area. Such sensor networks could be distributed in factories to monitor manufacturing conditions, spread over fields to log environmental conditions for agriculture, or mixed into concrete to actively measure building stresses and vibrations. Crossbow refers to its mote/sensor line of products as "Smart Dust", suggesting that one day the technology will reach the nanoscale.

The motes sold by Crossbow were originally developed at the University of California Berkeley. The MICA2 motes are based on the ATmega128L AVR microprocessor. The motes run an operating system called TinyOS.

TinyOS is an event driven operating system that handles the power consumption and radio networking. It is built to let the user focus on writing applications to acquire and react to sensor data. The driving goal behind TinyOS is to minimize power consumption. As such, it has been built around a "hurry up and sleep" methodology. The majority of the time the operating system maintains the mote in low power or "sleep" mode. Commands are only executed when an event occurs, and then they are carried out as quickly as possible so that the processor can go back to sleep. Such an architecture greatly reduces power consumption and extends battery life.

The motes are programmed in the nesC language. nesC is a dialect of C that implements a "wiring" concept. Applications are formed out of components - the basic building blocks of nesC code. Components are like lists of events and functions. These events and functions are called the interfaces of the component. For instance, a component may contain an LED interface, a TIMER interface, or an ADC interface. Applications are formed by "wiring" together the interfaces of different components. This will be discussed in greater detail later.

All of the code written for the motes is compiled in Cygwin. Cygwin is a linux emulator. Much like running "command" in windows provides you with a DOS environment, running Cygwin provides you a linux environment. After writing your code in a text editor, you will start up Cygwin, navigate to the directory where your code is located, and compile it.

INSTALLATION

Installing the software for mote development is a multiple step process that, if done incorrectly, can lead to major headaches later down the road. It is imperative that you follow these instructions precisely. These instructions assume the following:

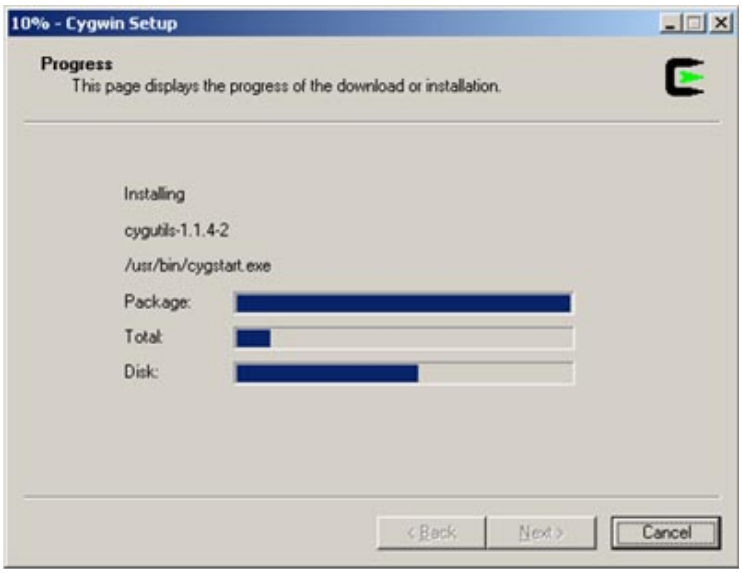
- You are running Windows 2000 or XP.
- You have never installed TinyOS before.
- You do not have any versions of java Software Development Kit (SDK) previously installed.

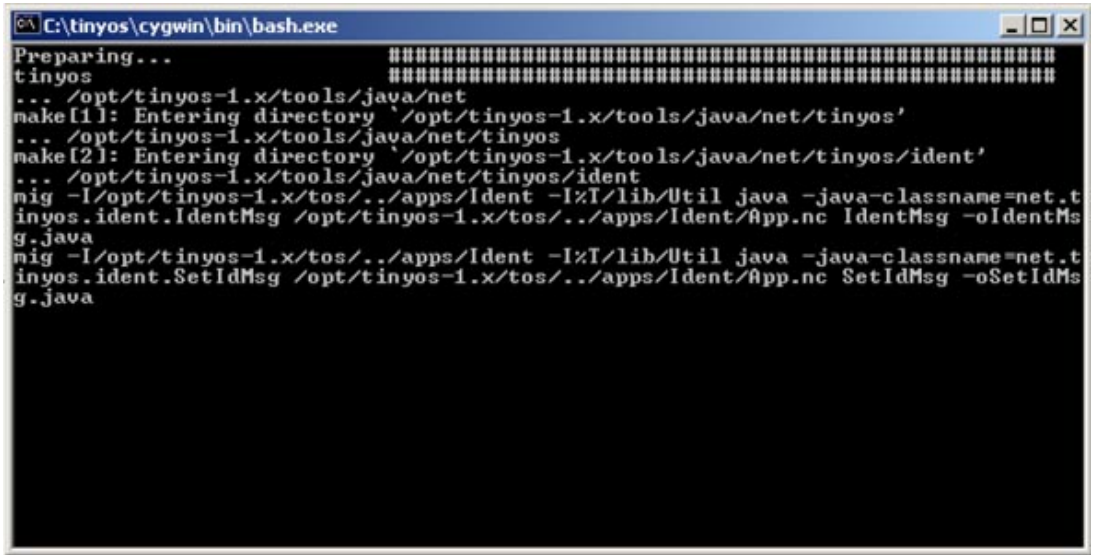
If you're unsure if you have previous versions of TinyOS or java SDK installed, you can still proceed with the installation. The installation wont affect the operation of any existing programs. You'll just end up with errors when trying to compile programs or run crossbow software.

The installation will proceed as follows: Install TinyOS v1.1.0 and cygwin, upgrade to latest release of TinyOS, install Crossbow's supported applications.

INSTALL TinyOS v1.1.0

1. You should have received a CD from Crossbow. Place this in the CD drive. If a window automatically pops up, close it.
2. Navigate to the root directory of the CD drive.
3. In the folder "TinyOS Install", run the file "tinyos-1.1.0-1is.exe".
4. In the "Setup Type" window, select "Complete" and use the default destination folder. Click on "Next".
5. In the "Java License Agreement" window, click "Yes" then "Next".
6. Follow the intstructions and choose all the defaults. This part of the installation can take as long as 45 minutes.





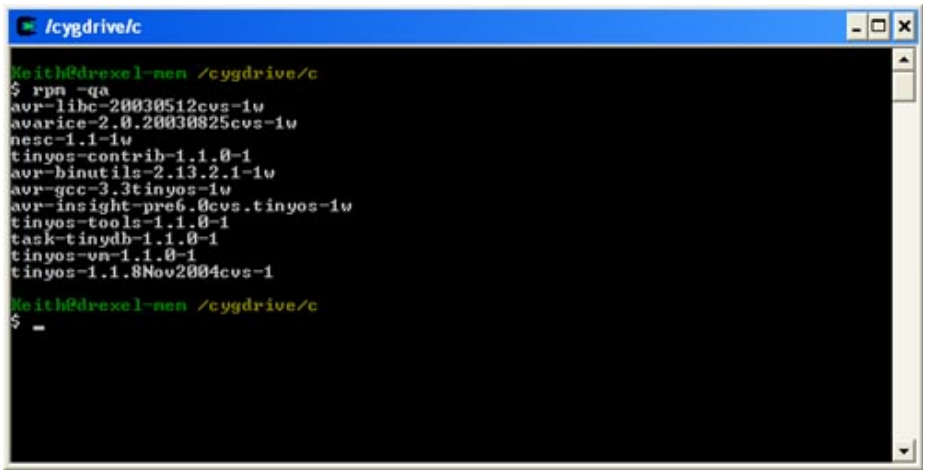
If everything goes well, you should see the two windows above, followed by a message box informing you that the installation is done. When the installation is complete, you should see a Cygwin icon on your desktop. Double-click this to open a new Cygwin window. In this window type: `toscheck`. After completion, the last line should read "toscheck completed without errors."

UPGRADE TinyOS

The version of TinyOS that you just installed is the last major release of the software. Major releases of TinyOS are rigorously tested and gaurunteed to work. However, the operating system is constantly evolving, and every 6 - 8 weeks the latest "snapshot" of code is released. Crossbow recommends that you always update TinyOS using the latest RPM file. To update TinyOS using the latest RPM file, do the following:

1. Navigate to the "C:\tinyos\cygwin\opt" directory.
2. Create a copy of the "tinyos-1.x" folder in this directory. Rename the copy "tinyos-1.1.0". This folder contains the existing version of TinyOS. It is good practice to always create a copy of the existing version of TinyOS before upgrading.
3. From the root directory of the CD drive, navigate to the folder "TinyOS Updates".

4. Copy the file "tinyos-1.1.8Nov2004cvs-1.cygwin.noarch.rpm" to the directory "C:\tinyos\cygwin\tmp\". **NOTE: This file name will be different depending on what the latest release of TinyOS is.**
5. Close any open Cygwin windows, then open a new Cygwin window.
6. In Cygwin, navigate to the "tmp" directory. To do this, type: `cd /tmp`.
7. To begin the upgrade, type:
   `rpm --nodeps --force --ignoreos -Uvh tinyos-1.1.8Nov2004cvs-1.cygwin.noarch.rpm`
8. After the installation completes, you can verify it by typing: `rpm -qa`. You should see output like that shown below.



INSTALL CROSSBOW APPS

While the installation of TinyOS does provide you with some sample code, Crossbow will not support any of it. On the CD, Crossbow provides sample code that they have written and tested. To install it, do the following:

1. From the "TinyOS Updates" folder on the CD, copy the file "xbow.tgz" to the directory "C:\tinyos\cygwin\tmp".
2. In Cygwin, navigate to the "tmp" directory.
3. Type the following:

```
gunzip xbow.tgz

cd /opt

tar -xvf /tmp/xbow.tar -C /opt/tinyos-1.x/contrib
```

This is the end of the installation. There is one last step you should take to make navigating Cygwin easier:

1. In windows, navigate to the "C:\tinyos\cygwin\etc" directory.
2. Open the file "profile" in Wordpad.
3. At the bottom of the file, type the following:

```
alias cdjava="cd /opt/tinyos-1.x/tools/java"
alias cdxapps="cd /opt/tinyos-1.x/contrib/xbow/apps"
alias cdxbow="cd /opt/tinyos-1.x/contrib/xbow"
```

4. Save and close the file.

These aliases are like shortcuts in Cygwin. They will allow you to change directories using one short word as opposed to one big long command. You are now ready to begin programming!

PROGRAMMING

In this section you will program a mote with the equivalent of a "hello world" program. You wont actually be writing any code. You will install code already developed by Crossbow onto the mote.

Take out your MIB510 serial gateway (the programming board) and plug in its power supply and serial cable. **NOTE: Remove the batteries before plugging in the mote.** The MIB510 is made to be powered from either the AC adapter or from the mote. Inserting the mote with batteries while the MIB510 is plugged into the AC adapter can fry the gateway. Take one of the MICA2 motes and plug it into the MIB510. You must also ensure that the mote is completely seated in the socket. If the mote is only half-way plugged in, when you try to program the mote you will receive errors. Even worse, the fuses for the AVR could become improperly programmed, preventing you from reprogramming the mote.

1. Close all open Cygwin windows and start a new Cygwin window.
2. Type:

   ```
   cdxapps

   cd blink

   make mica2
   ```

   This compiles the "Blink" program.
3. Now type:

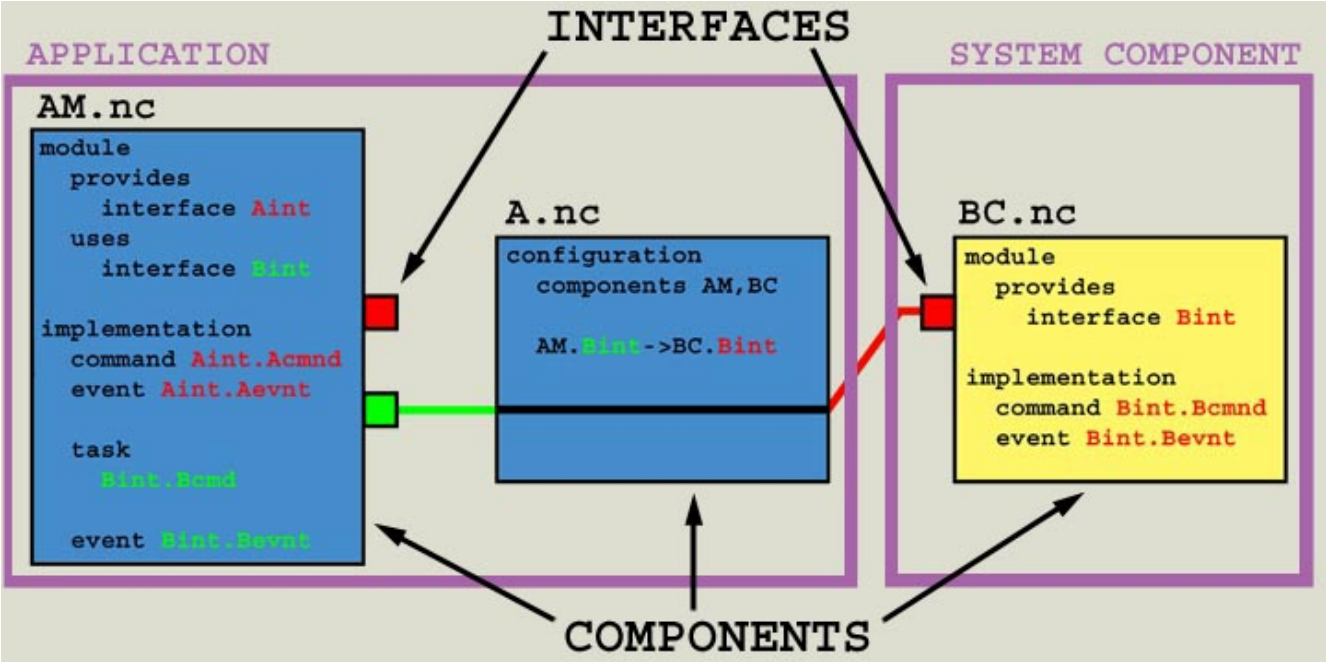
   ```
   make mica2 reinstall mib510,COM#
   ```

   Where "#" is the number of the com port that the MIB510 is plugged into.
4. Remove the mote from the programming board, install the batteries, and flip the switch on the side to the "ON" position.

You should see the red LED on the mote start to blink at a steady rate. Congratulations, you just programmed your first mote!

THE CODE

Before looking at any code, lets go over some of the basic concepts and rules for programming in nesC. nesC was constructed to be modular, allowing the user to snap together pieces of code. The diagram below represents a typical nesC application that references standard system files and contains some of its own code. This diagram will be used to explain the programming philosophy.



As mentioned previously, the primary building blocks of nesC code are **components**. Components are linked to other components through their **interfaces**. Interfaces are the only means of data transfer in or out of a component. Interfaces are really linking commands and events provided by one component to another component that wishes to use them. The interface for the component is defined in a module. The commands and events passed by interfaces are defined in implementations. Components are wired together through a configuration.

Looking at the diagram, module BC.nc **provides** the interface Bint. The implementation for the module defines a command Bint.Bcmnd and an event Bint.Bevnt. Lets say we were programming module AM.nc and we wanted to use the command Bint.Bcmnd. To gain access to the command, in the decleration of our module we would say it **uses** the interface Bint. We would now write a seperate configuration file A.nc which wires the interface of module AM.nc to the interface of module BC.nc. Module AM.nc now has access to all of the commands provided by BC.nc. A typical implementation for module AM.nc might define commands or events provided by AM.nc. The implementation might also contain a **task** where it uses the command from BC.nc. However, a fundamental rule of nesC states that any module that uses a command from another module must address all of the events provided by the other module. Hence, AM.nc must also contain code to address the event Bint.Bevnt.

The concepts presented here are still abstract. Lets see how this is used in real code. As an example, we'll analyze the Blink application that you just programmed onto the mote. The full code is presented in the links below. Clicking the links will bring up the code for each file in a new window.

Blink.nc - Blink configuration file (wires blink to outside files)
BlinkM.nc - Blink module (where the actual code is)
SingleTimer.nc - Outside file used by blink

The code also uses the files LedsC.nc and Main.nc. These files a standard system files and are not specific to this application. For the source code for these files, look in the directory where TinyOS was installed (likely located in "c:/tinyos/cygwin/opt/tinyos-1.x/tos/system").

Let's examine the configuration file first. Looking at the following code:

```
implementation {
  components Main, BlinkM, SingleTimer, LedsC;
  Main.StdControl -> SingleTimer.StdControl;
  Main.StdControl -> BlinkM.StdControl;
  BlinkM.Timer -> SingleTimer.Timer;
  BlinkM.Leds -> LedsC;
}
```

As can be seen, the application "Blink" uses the files "Main", "BlinkM", "SingleTimer" and "LedsC". The standard control interface from main is wired to the standard control interface for the timer and for the blink module. The timer and leds interfaces in the blink module are wired to the singletimer component and the ledsc component, respectively. This allows the blink module to call commands from singletimer and ledsc. This is the upper level programming. This sets up the blink module with all of the functions it will need. Now lets look at the blink module.

```
module BlinkM {
  provides {
     interface StdControl;
  }
  uses {
     interface Timer;
     interface Leds;
  }
}
```

This first portion of code sets up the interface for the module. This interface is required to implement the wiring in the configuration file. As stated previously, the blink module provides stdcontrol and uses timer and leds. Since blink uses the timer and leds components, it must handle all of the events from these components. We'll see where this comes in later.

```
implementation {
  command result_t StdControl.init() {
     call Leds.init();
     return SUCCESS;
  }

  command result_t StdControl.start() {
     return call Timer.start(TIMER_REPEAT, 1000);
  }
  command result_t StdControl.stop() {
     return call Timer.stop();
  }
```

This is the implementaion, where all of the actual code is written. The standard control interface provides the "init", "start" and "stop" commands. These commands are used by the "main" component. Main is a component that is executed first by any TinyOS application. All applications must reference "main" in their configuration.

When an application first starts, the init command is issued. Here you can see that when the blink module init command is called, the blink module issues the init command to leds. This command will always return SUCCESS.

Immediately after the init command is called, the start command is called. The blink module's start and stop commands are used to start and stop the timer. The timer is set to repeat every 1000 msec.

```
  event result_t Timer.fired(){
     call Leds.redToggle();
     return SUCCESS;
  }
}
```

Finally, the blink module must handle the "fired" event provided by the timer. This event is convinient for executing operations at specific intervals. In this case, the "fired" event is used to turn the red LED on and off.

FINAL WORDS

After reviewing this tutorial you should be able to program a mote and understand simple nesC code. If you have questions about this tutorial you can email me at **Keithicus@drexel.edu**.