

Department of Computer Science
Southern Illinois University Carbondale



Exposure to Sensor Motes



CS 441 – WIRELESS & MOBILE COMPUTING

Instructor – Dr. Kemal Akkaya
kemal@cs.siu.edu




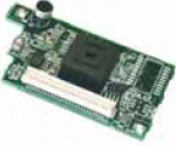
Section 1: Introduction

This guide is intended to describe the step-by-step process of setting up the MEMSIC IRIS motes, running simple programs for hardware verification, and to run a few simple programs to get an understanding of the tools used in the process.

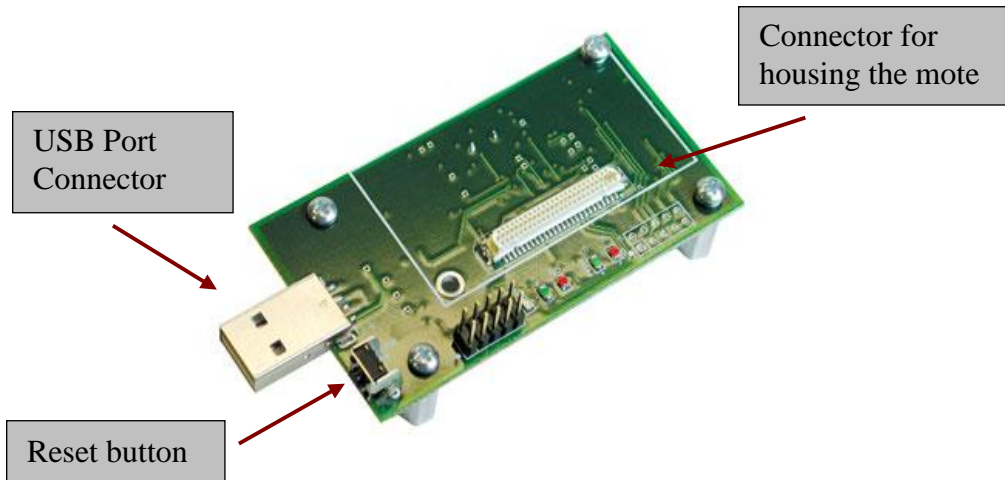
It is assumed the students have a basic understanding of Linux and electronic hardware components. The students will work in groups of two.

Section 2: An Overview of the Hardware Used

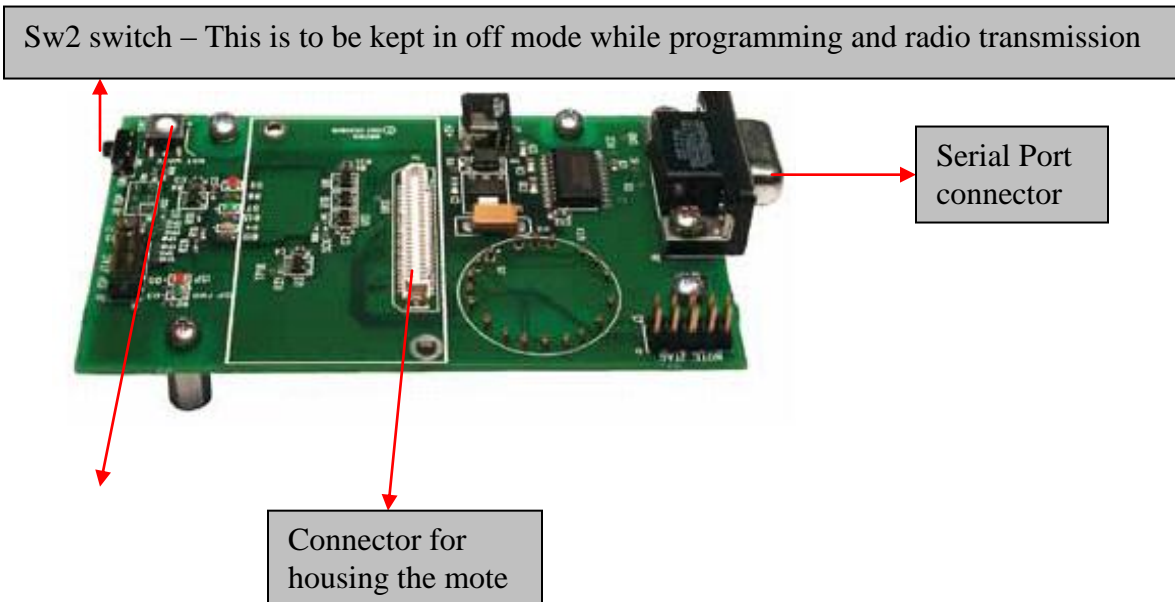
List of hardware components that will be used in the labs.

#	Component	Purpose	Image
1	Programming Board – MIB520CB	Used for programming the motes and for housing the base mote for wireless communication. USB based connection. Referred to as gateway.	
2	Programming Board – MIB510	Used for programming the motes and for housing the base mote for wireless communication. Non-USB based connection. Referred to as gateway.	
3	IRIS Mote - XM2110CB	This includes a processor that runs the TinyOS. It is capable of radio transmission and reception. It has a 51-pin connector for housing the sensor. Uses 2.4 GHz IEEE 802.15.4 standard. Referred to as wireless board.	
4	Sensing Unit – MTS300	Has the capability to sense data and transmit it using the processor/radio module. It is attached to the motes via the 51 pin. Referred to as sensor board.	

1. **MIB520:** The Programming (Interface) board is connected to the PC by a serial port via USB.



2. **MIB510:** The Programming (Interface) board is connected to the PC by a serial port.



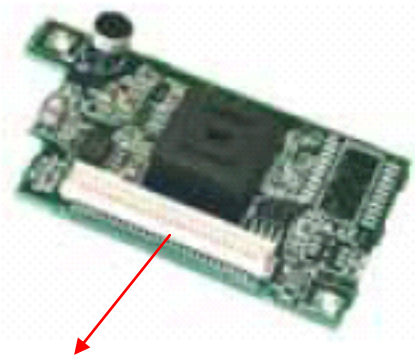
3. **IRIS:** The Motes are connected to the interface board for programming. They house the sensors for sensing and transmission of data.

ON/OFF switch. It is very *important* to remove batteries and to keep this switch in off mode while programming



51-pin connector for connecting to the interface board or for housing the sensors

- 4. **MTS300:** The sensor modules are connected to the motes for sensing and data transmission



51-pin connector for connecting to the motes

Section 3: TinyOS Operating System & Installation

TinyOS is an open source operating system targeted for the embedded wireless networks. It uses a component-based architecture. The TinyOS operating system is written using the nesC which is a structured component based language.

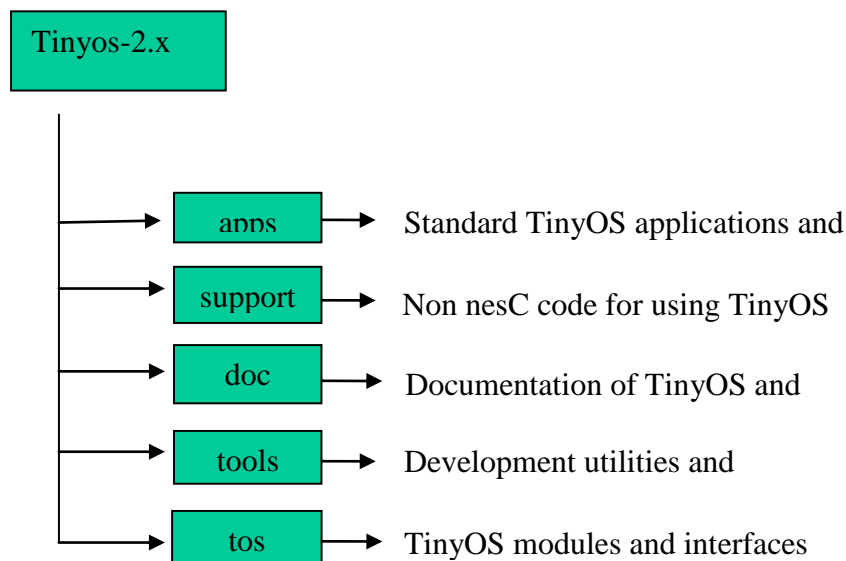
For further information please visit - <http://www.tinyos.net/>

For more information about nesC please refer <http://nesc.sourceforge.net/>

Installing TinyOS on your computer:

We will go through the document provided by Crossbow - Chapter 2 for installation instructions. This is available on the class website under Hands-On Labs section.

The directory structure of tinyos-2.x is as shown below.



Most of the applications that will be described in this document can be found in the apps directory.

Installation Steps

1. Download VMWare Player 2.0 for Windows from [here](#).
2. Download the XubunTOS virtual machine archive file from [here](#).
3. Install the VMWare Player by launching the .exe and following the onscreen instructions.

4. Extract XubuntuTOS and any archives inside it to the directory of your choice.
5. Run VMWare Player and double click the open option, navigate to where you extracted XubuntuTOS and open the Xubuntos 2.1.vmx file.
6. In the virtual machine find /usr/bin/motelist and open the file with a text editor replace all instances of “6001” with “6010” and save the file.

Connecting the MIB520CB

1. Plug in the MIB520 with the attached mote via a USB port. Make sure that the switch on the mote is in the off position and the batteries are removed when programming it.

(WARNING! Keeping the power button on or failure to remove the batteries when writing the flash could render the motes unusable).

2. Click the Devices option at the top of the Vmplayer window and navigate to the future devices usb composite device option, hover over it and click connect.
3. Open a cmd shell and run motelist take note of the USB device the mote is connected to.

Section 4: Running a Simple Application: Blink

Blink is a basic application that starts a 1Hz timer and toggles the LEDs every time it fires. It is a very simple program that is little more than a demonstration of TinyOS programming.

This binary has to be uploaded into the flash in the mote. This is done using the programming (interface) board as described in the [section 1](#).

Verify that you have:

- Removed the batteries from the mote that is to be programmed (whose hardware verification is to be done) and ensure that the power switch is turned off. (*Keeping the power button on or failure to remove the batteries when writing the flash could render the motes unusable*).
- Connected the female part of the 51-pin connector on the mote to the male part on MIB520 located on the top of the board.
- Now the setup is ready for the flash memory on the mote to be programmed.
- **Running Blink**
 1. Open a cmd shell and change the directory to:
/opt/tinyos-2.1.0/apps/Blink
 2. Run the command “make iris install,1 mib520,[usb port connected]”
where the usb port is in the form of /dev/ttyUSB0 or similar.
- When the flash is being written the red LED next to the power LED (green LED) will light up.

If everything goes smoothly the following output will be seen:

```
Terminal - xubuntu@xubuntu-tinyos: /opt/tinyos-2.1.0/apps/Blink
File Edit View Terminal Go Help
avrdude: verifying efuse memory against 0xff:
avrdude: load data efuse data from input file 0xff:
avrdude: input file 0xff contains 1 bytes
avrdude: reading on-chip efuse data:
Reading | ##### | 100% 0.02s
avrdude: verifying ...
avrdude: 1 bytes of efuse verified
avrdude: reading input file "build/iris/main.srec.out-1"
avrdude: input file build/iris/main.srec.out-1 auto detected as Motorola S-Record
avrdude: writing flash (2268 bytes):
Writing | ##### | 100% 0.73s
avrdude: 2268 bytes of flash written
avrdude: verifying flash memory against build/iris/main.srec.out-1:
avrdude: load data flash data from input file build/iris/main.srec.out-1:
avrdude: input file build/iris/main.srec.out-1 auto detected as Motorola S-Record
avrdude: input file build/iris/main.srec.out-1 contains 2268 bytes
avrdude: reading on-chip flash data:
Reading | ##### | 100% 0.53s
avrdude: verifying ...
avrdude: 2268 bytes of flash verified
avrdude: safemode: Fuses OK
avrdude done. Thank you.
rm -f build/iris/main.exe.out-1 build/iris/main.srec.out-1
```

- The above output demonstrates that the board and the mote are functioning properly.
- **Now if we observe the interface board the lights will be blinking in order.**
- As the next step, you can detach the mote and then install batteries. Now turn the power switch to ON. You should be able to see the lights blinking again.

If the verifications cannot be done, the user may use the following tips to solve different problems:

1. Check whether the switches on programming board and motes are on proper setting.
2. When housing the motes on the board, check whether the motes are firmly connected on board.
3. Use antennas with remote motes.
4. Hit reset button on programming board.
5. Verify proper permissions on the USB port, `sudo /bin/chmod 666 [usb port such as:/dev/ttyUSB0]`
6. Check the radio frequency setting.

Section 5: Lab-2: Testing Radio Communication

This experiment uses two motes that communicate with each other.

The applications used here are:

- **RadioCountToLeds**

RadioCountToLeds maintains a 4Hz10 counter, broadcasting its value in an AM packet every time it gets updated. A RadioCountToLeds node that hears a counter displays the bottom three bits on its LEDs. This application is a useful test to show how basic AM communication and timers work.

This will be programmed onto two separate motes.

and

- 1. **RadioSenseToLeds**

RadioSenseToLeds samples a platform's default sensor at 4Hz and broadcasts this value in an AM packet. A RadioSenseToLeds node that hears a broadcast displays the bottom three bits of the value it has received. This application is a useful test to show that basic AM communication, timers, and the default sensor work.

Instructions:

- Now take the mote ONE and connect it to the Interface Board to load RadioCountToLeds.
- Run the following commands to compile the program
 1. `cd /opt/tinyos-2.1.0/apps/RadioCountToLeds`
 2. `export CLASSPATH=./opt/tinyos-2.1.0/support/sdk/java/tinyos.jar`
 2. `make iris install,1 mib520,[usb port connected]`
where the usb port is in the form of `/dev/ttyUSB0` or similar.

The screen output should be as show below.

```
File Edit View Terminal Go Help
avrdude: verifying efuse memory against 0xff:
avrdude: load data efuse data from input file 0xff:
avrdude: input file 0xff contains 1 bytes
avrdude: reading on-chip efuse data:

Reading | ##### | 100% 0.02s

avrdude: verifying ...
avrdude: 1 bytes of efuse verified
avrdude: reading input file "build/iris/main.srec.out-1"
avrdude: input file build/iris/main.srec.out-1 auto detected as Motorola S-Record
avrdude: writing flash (10648 bytes):

Writing | ##### | 100% 3.35s

avrdude: 10648 bytes of flash written
avrdude: verifying flash memory against build/iris/main.srec.out-1:
avrdude: load data flash data from input file build/iris/main.srec.out-1:
avrdude: input file build/iris/main.srec.out-1 auto detected as Motorola S-Record
avrdude: input file build/iris/main.srec.out-1 contains 10648 bytes
avrdude: reading on-chip flash data:

Reading | ##### | 100% 2.45s

avrdude: verifying ...
avrdude: 10648 bytes of flash verified

avrdude: safemode: Fuses OK

avrdude done. Thank you.

rm -f build/iris/main.exe.out-1 build/iris/main.srec.out-1
xubuntu@xubuntu-tinyos:/opt/tinyos-2.1.0/apps/RadioCountToLeds$
```

- The LEDs on the board will not yet light up.
- Now power off the interface board and take the mote ONE out. Put the batteries into the mote and turn the power switch to ON position.
- Repeat the process and install RadioCountToLeds on the second mote. The LEDs on both motes will start blinking.
- Repeat the process by removing mote TWO.
- Put the batteries into mote TWO and turn the power switch to the on position.
- Both motes should begin blinking again. Notice if you switch off one mote both stop blinking.
- Repeat the preceding steps but install RadioSenseToLeds instead. Both motes should light up in varying orders. This is because instead of using a counter the node is using its default sensor, which currently gives the voltage on the Iris platform. We will be making use of this sensor in the next part of the lab.

Section 6: Lab-3: MultihopOscilloscope - A simple application using MultiHop Routing Protocol

MultihopOscilloscope is an example application that uses MultiHop ad-hoc routing. It is designed to be used in conjunction with the Oscilloscope java tool. A node with the MultihopOscilloscope process installed will periodically sample its default sensor and broadcast a message every few readings. The base-station can then receive the readings from multiple nodes in the Ad-Hoc network.

Tools:

SerialForwarder - net.tinyos.sf.SerialForwarder

The SerialForwarder tool reads packet data from a serial port and forwards it over the wireless connection.

Oscilloscope.class

This class processes sensor data from MultihopOscilloscope programmed nodes via a serial forwarder and displays the readings via a graph. The controls at the bottom of the screen allow you to zoom in or out on the X axis, change the range of the Y axis, and clear all received data. You can change the color used to display a mote by clicking on its color in the mote table.

Instructions:

Three (at least two motes are necessary) motes are used to demonstrate this application.

The three motes used are:

MOTE 0: This node acts as a base mote that serves as a collection root and receives packets from other motes in the ad-hoc network.

MOTE 1 and MOTE 2: They form a part of the ad-hoc network and send data packets to MOTE 0.

The following commands will navigate to the applications directory and add the tinyos.jar file to the CLASSPATH for compilation.

- `cd /opt/tinyos-2.1.0/apps/MultihopOscilloscope`
- `export CLASSPATH=./opt/tinyos-2.1.0/support/sdk/java/tinyos.jar`

Now this program needs to be installed on all the motes that will be part of the ad-hoc network. The following commands do this.

Take the base mote “MOTE 0” and connect it to the Interface board.

- `make iris install,0 MIB520,/dev/ttyUSB0`

This will install the application on MOTE 0’s flash and set it as the base station.

Similarly for the MOTEs 1 and 2 use the following commands respectively

- `make iris install,1 MIB520,/dev/ttyUSB0`
- `make iris install,2 MIB520,/dev/ttyUSB0`

Now put the MOTE 0 back on to the interface board and power up the rest of the motes by connecting the batteries and turning the power switch ON.

Compilation and execution of the java tools

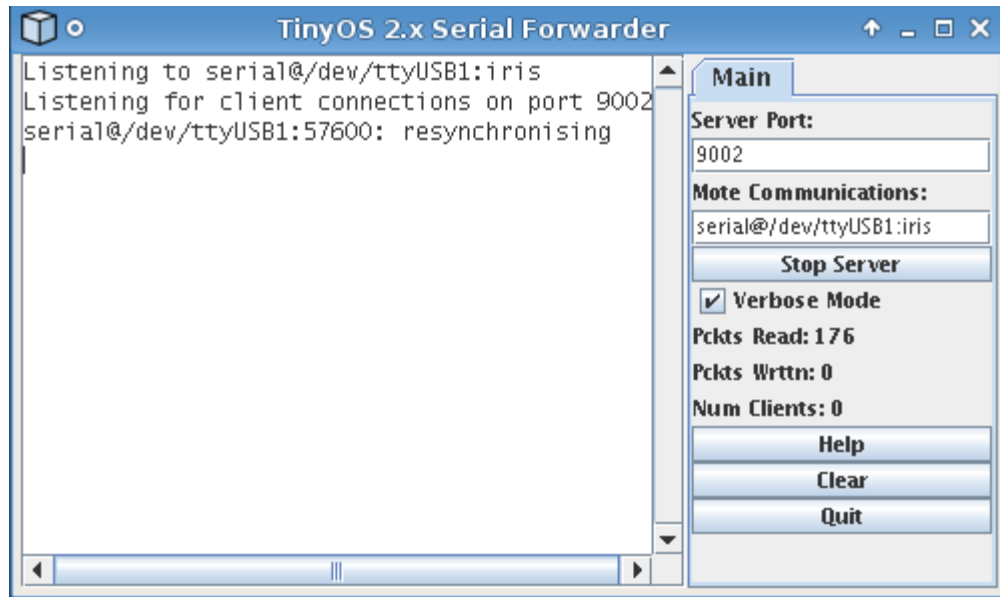
The following java tools have to be invoked now to generate packets for transmission and to visualize the network:

SerialForwarder

First we will invoke the SerialForwarder.

- `cd opt/tinyos-2.1.0/apps/MultihopOscilloscope/java`
- `export CLASSPATH=./opt/tinyos-2.1.0/support/sdk/java/tinyos.jar`
- `make`
- `java net.tinyos.sf.SerialForwarder -comm serial@/dev/ttyUSB1:iris`

This command brings up the following GUI



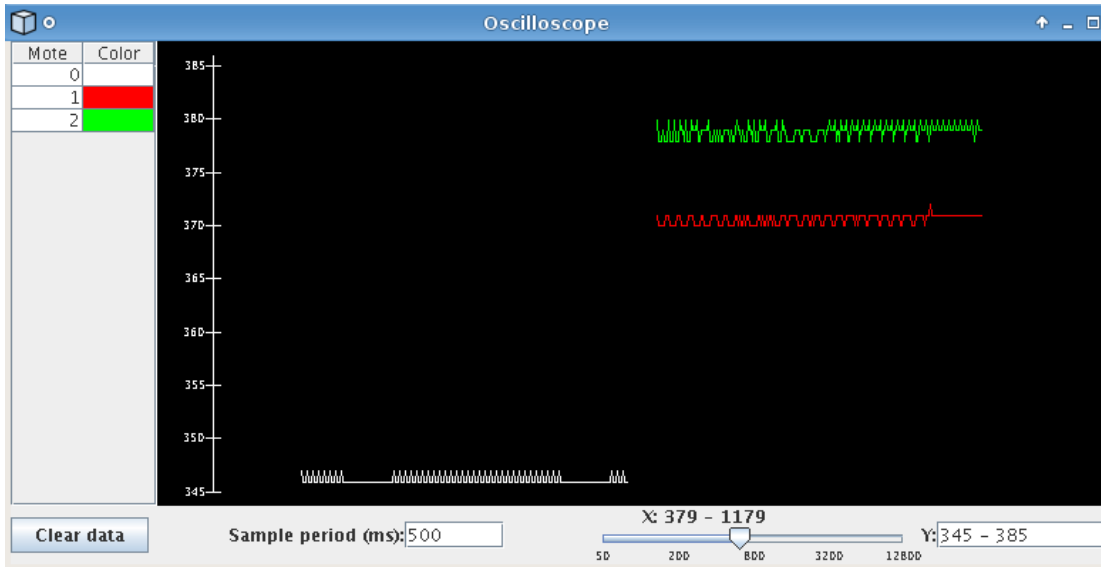
If the installation of the application on the MOTES went fine the “*Pckts Read:*” will start to increment. This keeps increasing till the “*Stop Server*” button is pressed.

Visualization tool

We now invoke the visualization tool which should be opened in a new terminal window.

7. `cd /opt/tinyos-2.1.0/apps/MultihopOscilloscope`
8. `java Oscilloscope`

This brings up a screen as shown below.



There are three MOTEs communicating as shown in the figure above.

This demonstrates the simple MultiHop ad-hoc routing protocol.