



# TinyOS

## Sensor Network Programming



# Lecture Overview

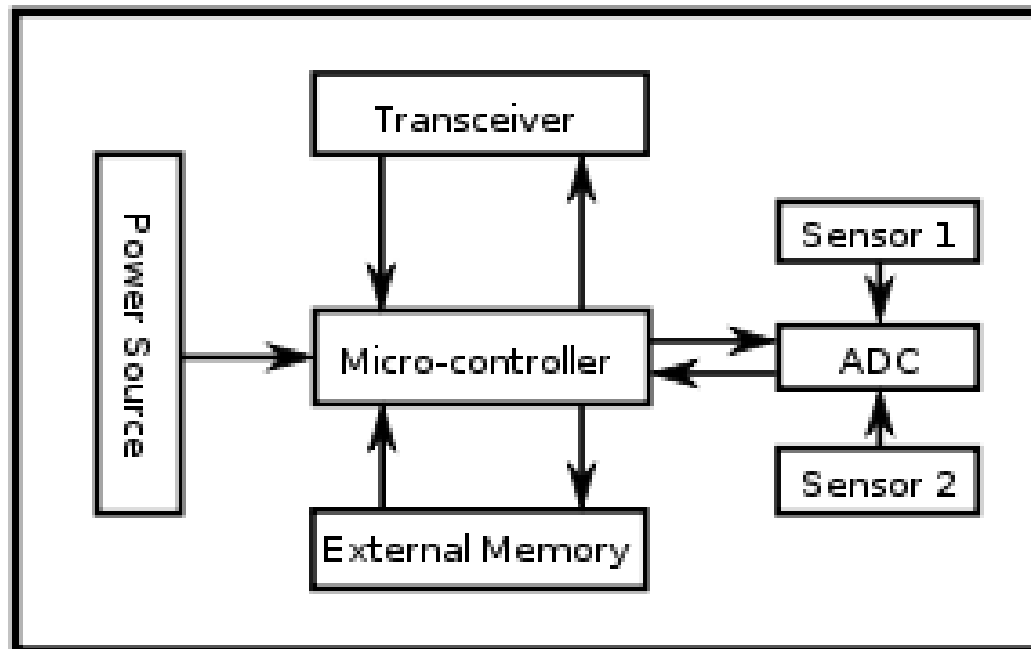
- 1. Hardware Primer
- 2. Introduction to TinyOS.
- 3. Programming TinyOS.
- 4. Hands on section.



# Sensor node(mote):

1. Node in a wireless sensor network
2. Capable of performing some processing
3. Gathers information from sensors
4. Communicates with other connected nodes in the network.

# Architecture of sensor node:





# Sensor node:

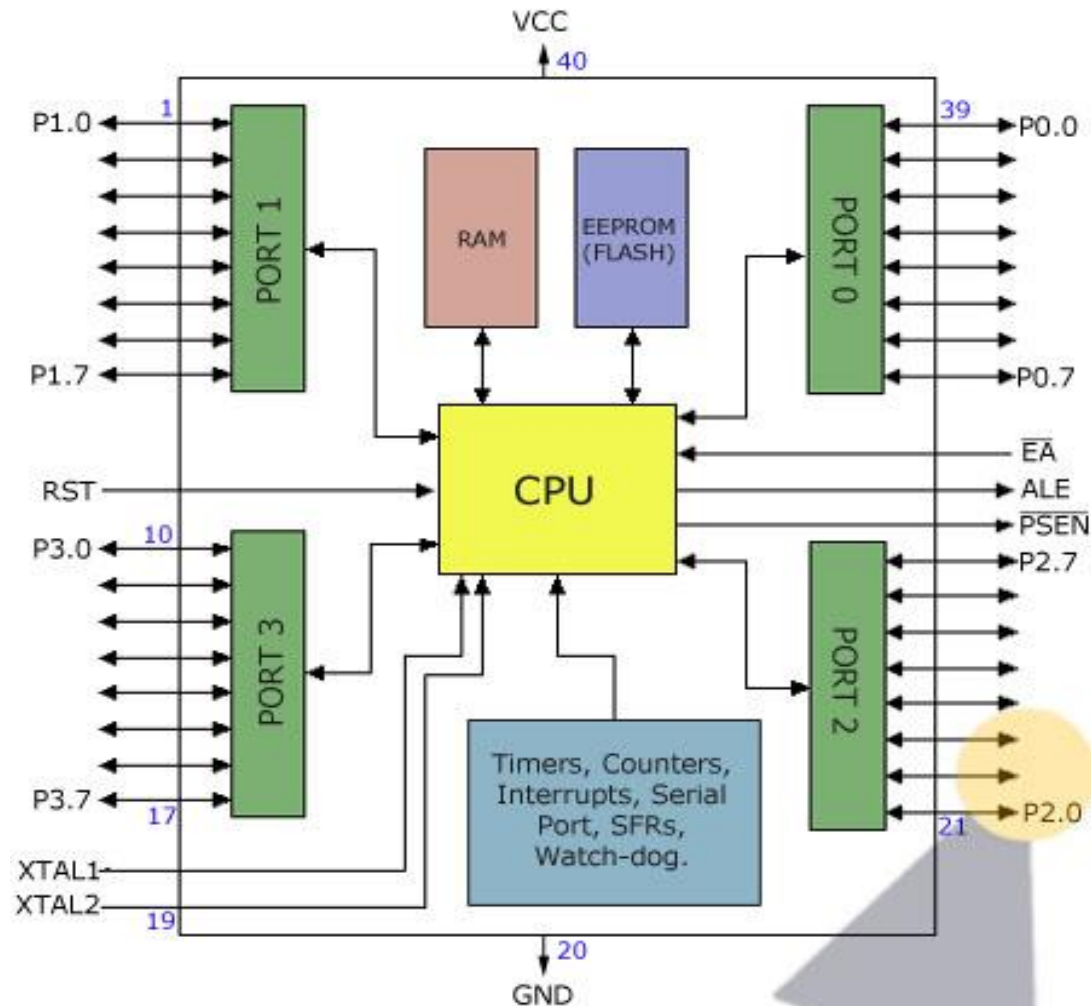
Two main parts

1. Microcontroller
2. Transceiver

IITH Mote specifications:

<http://www.iith.ac.in/~raji/downloads/IITH-mote-webpage.pdf>

# Basic controller architecture





# Purpose of controller

Functions of a mote:

- Collecting data from various sensors
- Process data and extract useful information
- Transmitter controlling
- Local storage maintenance



# Purpose of controller

## Data collections:

- Collecting data from various sensors, simultaneously.
- Data collected from individual sensors should have to be maintained properly.
- Sequential sampling (Reduces data rate).
- Adaptive sampling where one can adapt sampling rate based on some classification.





# Purpose of controller

## Data processing:

- Some applications require on board processing of the collected data.
- Most of the adaptive sampling algorithms use on board processing due to less delay.

## Transceiver control:

- Controller can force transceiver into sleep mode when it is not needed.
- Can wake up transmitter, when there is some data to be transmitted.



# Purpose of controller

## Local storage maintenance:

- If the gateway is not in the range, then the data can be stored on to the local storage.
- When the gateway comes into vicinity, it can transmit the stored data and free up the local storage.

## Power gating:

- Some of the functional blocks which are not necessary at present can be switched off to conserve power and can only be turned on when needed.



# How to select a controller?

## Things to keep in mind

- Power consumption
- Processing required
- Mode of communication (Baseband and RF processing)
- Priority of application (Medical or Pollution data)



# Simple controller example

## ATMEGA128 uC

- 8 bit architecture
- 8 channel ADC (10 bit resolution)
- TWI
- 2 UART interfaces
- SPI interface (To interface additional memory)
- Can run TinyOS & Contiki.

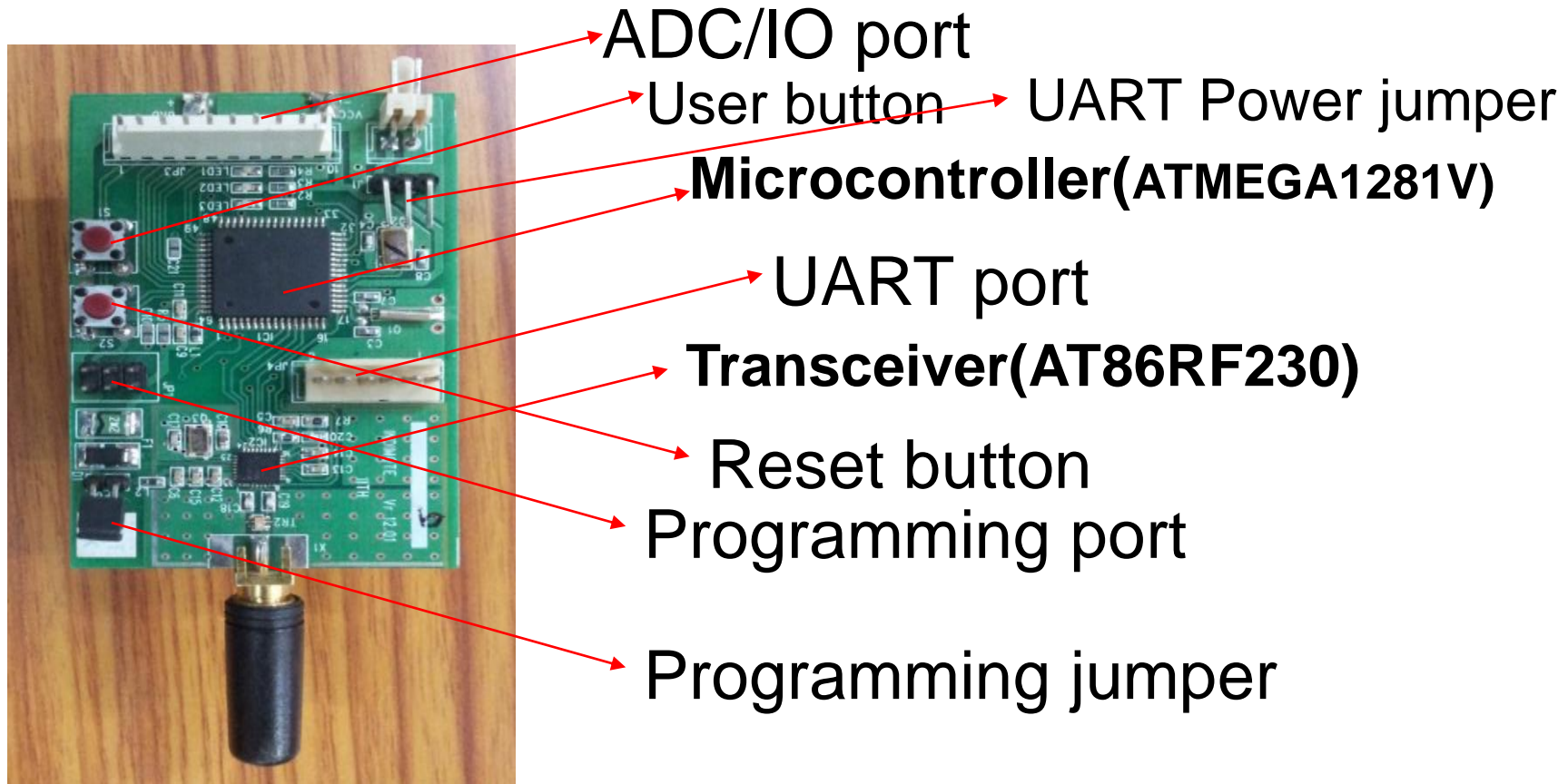


# Transceiver

AT86RF230:

Low Power 2.4 GHz Transceiver for ZigBee,  
IEEE 802.15.4, 6LoWPAN, ISM  
Applications.

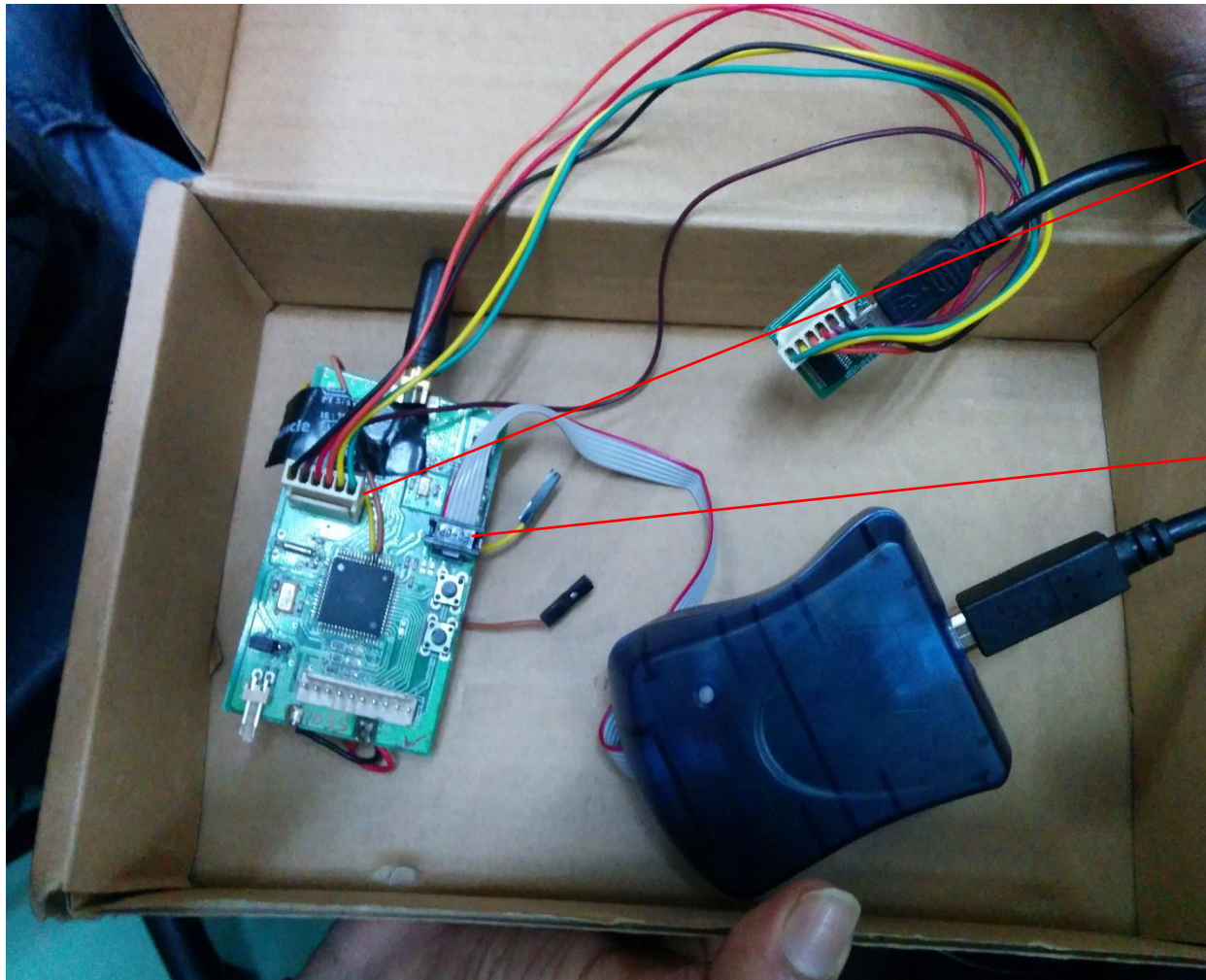
# IITH Mote(sensor node):



IITH Mote specifications:

<http://www.iith.ac.in/~raji/downloads/IITH-mote-webpage.pdf>

# Hardware setup to programming










UART port con.

Programmer  
con.



# UC Berkeley Family of Motes

Mote Type Year	<i>WeC</i> 1998 	<i>René</i> 1999 	<i>René 2</i> 2000	<i>Dot</i> 2000 	<i>Mica</i> 2001 	<i>Mica2Dot</i> 2002 	<i>Mica 2</i> 2002 	<i>Telos</i> 2004 
Microcontroller								
Type	AT90LS8535		ATmega163		ATmega128		TI MSP430	
Program memory (KB)	8		16		128		60	
RAM (KB)	0.5		1		4		2	
Active Power (mW)	15		15		8		33	
Sleep Power ( $\mu$ W)	45		45		75		75	
Wakeup Time ( $\mu$ s)	1000		36		180		180	
Nonvolatile storage								
Chip	24LC256				AT45DB041B			ST M24M01S
Connection type	I <sup>2</sup> C				SPI			I <sup>2</sup> C
Size (KB)	32				512			128
Communication								
Radio	TR1000				TR1000	CC1000		CC2420
Data rate (kbps)	10				40	38.4		250
Modulation type	OOK				ASK	FSK		O-QPSK
Receive Power (mW)	9				12	29		38
Transmit Power at 0dBm (mW)	36				36	42		35
Power Consumption								
Minimum Operation (V)	2.7		2.7		2.7		1.8	
Total Active Power (mW)	24				27	44	89	41
Programming and Sensor Interface								
Expansion	none	51-pin	51-pin	none	51-pin	19-pin	51-pin	10-pin
Communication	IEEE 1284 (programming) and RS232 (requires additional hardware)							USB
Integrated Sensors	no	no	no	yes	no	no	no	yes





# Lecture Overview

- 1. Hardware Primer
- 2. Introduction to TinyOS
- 3. Programming TinyOS
- 4. Hands on section.

# What is TinyOS?

- An operation system
- An open-source development environment
- Not an operation system for general purpose, it is designed for wireless embedded sensor network.
  - Official website: <http://www.tinyos.net/>
- Programming language: NesC (an extension of C)
- It features a **component-based** architecture.
- Supported platforms include Linux, Windows 2000/XP with Cygwin.

# Install TinyOS

1. Install Ubuntu 12.04/13.04/14.04 or any higher versions.
2. Enable root user.
3. Switch to root user to install TinyOS.
4. Open terminal (Ctrl+Alt+T).

# Installation procedure:

1. `gedit /etc/apt/source.list`

Add this at end of the file

`deb http://hinrg.cs.jhu.edu/tinyos hardy main`

2. `apt-get update`

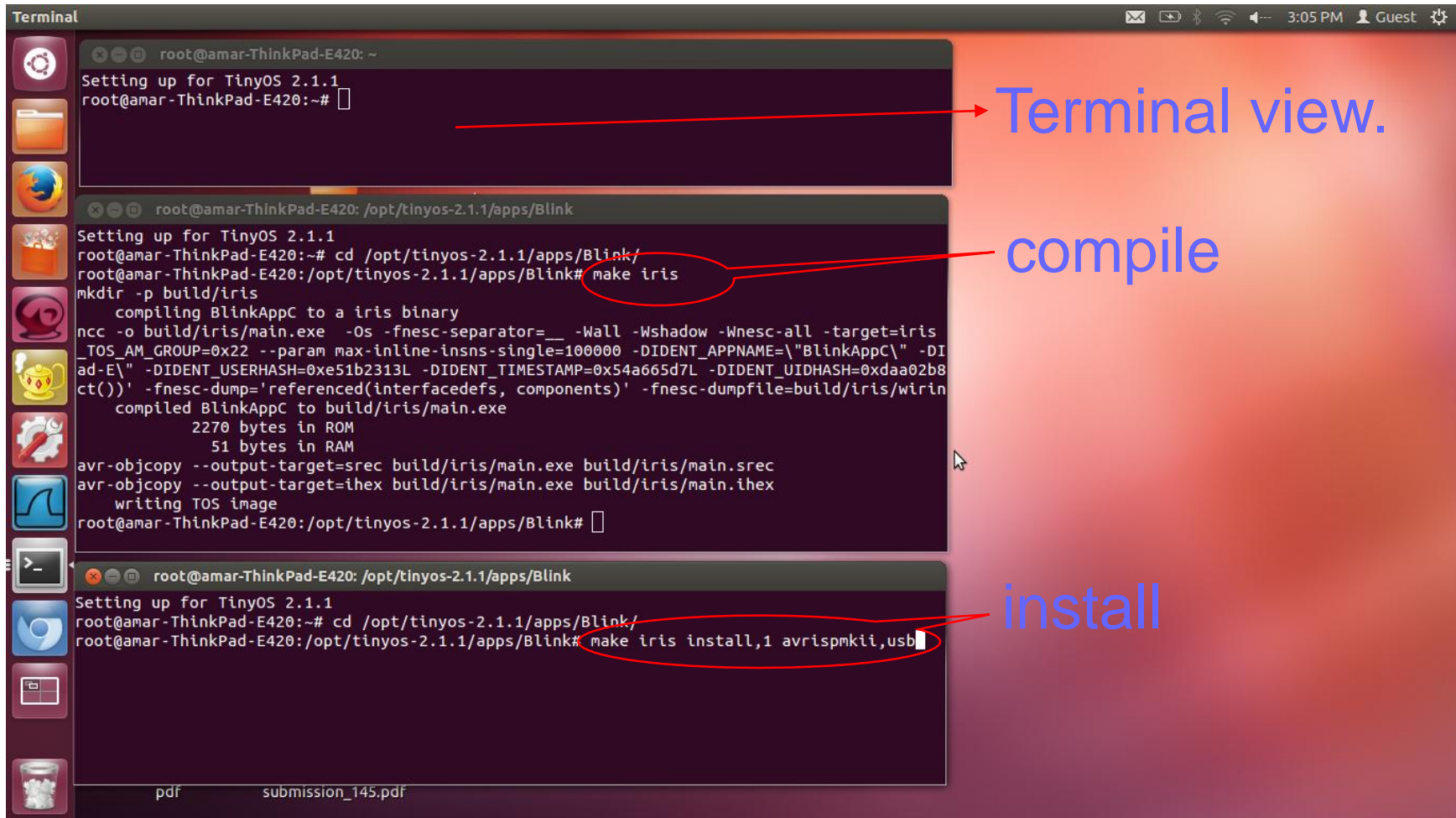
3. `apt-get install tinynos-2.1.1`

4. `gedit ~/.bashrc`

Add this at end of file

```
#Sourcing the tinynos environment variable
setup script source /opt/tinynos-
2.1.1/tinynos.sh
```

# Compile and install program



The image shows a terminal window with three stacked sessions. The first session shows the initial setup for TinyOS 2.1.1. The second session shows the compilation of the 'Blink' application using 'make iris', with a red circle highlighting the command. The third session shows the installation of the compiled program using 'make iris install,1 avrispnkii,usb', with a red circle highlighting the command. Red arrows point from the text labels 'Terminal view.', 'compile', and 'install' to their respective parts in the terminal output.

```
root@amar-ThinkPad-E420: ~  
Setting up for TinyOS 2.1.1  
root@amar-ThinkPad-E420:~#  
  
root@amar-ThinkPad-E420: /opt/tinyos-2.1.1/apps/Blink  
Setting up for TinyOS 2.1.1  
root@amar-ThinkPad-E420:~# cd /opt/tinyos-2.1.1/apps/Blink/  
root@amar-ThinkPad-E420:/opt/tinyos-2.1.1/apps/Blink# make iris  
mkdir -p build/iris  
compiling BlinkAppC to a iris binary  
ncc -o build/iris/main.exe -Os -fnesc-separator=__ -Wall -Wshadow -Wnesc-all -target=iris  
_TOS_AM_GROUP=0x22 --param max-inline-insns-single=100000 -DIDENT_APPNAME=\"BlinkAppC\" -DI  
ad-E\" -DIDENT_USERHASH=0xe51b2313L -DIDENT_TIMESTAMP=0x54a665d7L -DIDENT_UIDHASH=0xdaa02b8  
ct())' -fnesc-dump='referenced(interfacedefs, components)' -fnesc-dumpfile=build/iris/wirin  
compiled BlinkAppC to build/iris/main.exe  
2270 bytes in ROM  
51 bytes in RAM  
avr-objcopy --output-target=srec build/iris/main.exe build/iris/main.srec  
avr-objcopy --output-target=ihex build/iris/main.exe build/iris/main.ihex  
writing TOS image  
root@amar-ThinkPad-E420:/opt/tinyos-2.1.1/apps/Blink#  
  
root@amar-ThinkPad-E420: /opt/tinyos-2.1.1/apps/Blink  
Setting up for TinyOS 2.1.1  
root@amar-ThinkPad-E420:~# cd /opt/tinyos-2.1.1/apps/Blink/  
root@amar-ThinkPad-E420:/opt/tinyos-2.1.1/apps/Blink# make iris install,1 avrispnkii,usb
```

pdf submission\_145.pdf

# Program installation view on terminal

```
root@amar-ThinkPad-E420: /opt/tinyos-2.1.1/apps/Blink
avrdude: verifying ...
avrdude: 1 bytes of hfuse verified
avrdude: reading input file "0xff"
avrdude: writing efuse (1 bytes):
Writing | ##### | 100% 0.00s
avrdude: 1 bytes of efuse written
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% 0.00s
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 (2270 bytes):
Writing | ##### | 100% 0.66s
avrdude: 2270 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 2270 bytes
avrdude: reading on-chip flash data:
Reading | ##### | 100% 0.65s
avrdude: verifying ...
avrdude: 2270 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
root@amar-ThinkPad-E420: /opt/tinyos-2.1.1/apps/Blink#
```



# Lecture Overview

- 1. Hardware Primer
- 2. Introduction to TinyOS
- 3. Programming TinyOS
- 4. Hands on section.

# Program files

Every application needs 4 files

1. Make file (Makefile)
2. Configuration file (SensorAppC.nc)
3. Module file (SensorC.nc)
4. Header file (Sensor.h) (if application needs)

Sensor is application name.

check example application in TinyOS

**cd /opt/tinyos-2.1.1/apps/** (path)

**cd /opt/tinyos-2.1.1/apps/tutorials** (path)

To develop application gedit or eclips IDE can be used

<https://www.youtube.com/watch?v=IO5spZwKwRQ>



# Editors for writing a application

Gedit:

Create a folder with your application name.

Open terminal

**Cntrl+Alt+T**

Open a document by using gedit command

And save with your application name.

**gedit documentname**

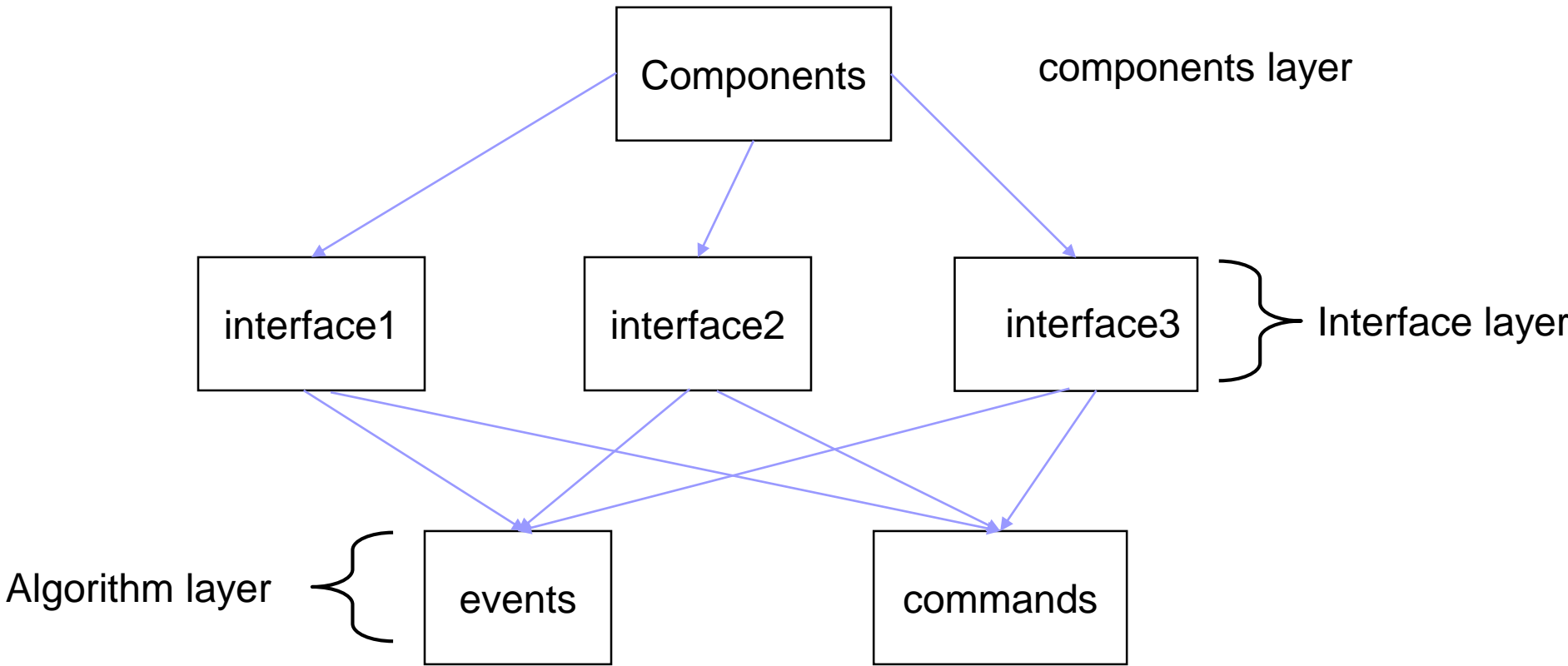
create 4 files with mentioned extension in one folder.



# How to write a application

Programming structure:

1. Search interfaces required for your application.
2. Search components which provides those interfaces.
3. Use commands and events which will be provided by interfaces to develop algorithms.





# How to write application

Makefile: compiler can compile program.

```
“COMPONENT= SensorAppC  
include $(MAKERULES)”
```

# Configuration file(SensorAppC.nc):

File contains components which provides and uses interfaces.

1. Initialization of components.
2. Wiring of components with interfaces.

Components example:

MainC, LedsC, TimerMilliC.

<http://www.tinyos.net/tinyos-2.1.0/doc/nescdoc/micaz/>

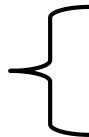


# Module file(SensorC.nc):

1. File contains Interfaces initialization and using interfaces.
2. Interfaces contains commands and events.
3. Commands and events are used to develop algorithm.

# Component Syntax - Configuration

Component  
Selection



```
configuration SensorAppC
{
}
implementation
{
  components MainC, SensorC, LedsC;
  components new TimerMilliC() as Timer0;
  components new TimerMilliC() as Timer1;
  components new TimerMilliC() as Timer2;
  SensorC -> SensorC.Boot;
```

Wiring the  
Components  
together



```
  SensorC.Timer0 -> Timer0;
  SensorC.Timer1 -> Timer1;
  SensorC.Timer2 -> Timer2;
  SensorC.Leds -> LedsC;
}
```

# Module syntax:

```
#include "Timer.h"
```

```
module SensorC()
```

```
{  
  uses interface Timer<TMilli> as Timer0;  
  uses interface Timer<TMilli> as Timer1;  
  uses interface Timer<TMilli> as Timer2;  
  uses interface Leds;  
  uses interface Boot;
```

interface X as Y

= interface X as X

```
}
```

```
implementation
```

```
{  
  event void Boot.booted()  
  {  
    call Timer0.startPeriodic( 250 );  
    call Timer1.startPeriodic( 500 );  
    call Timer2.startPeriodic( 1000 );
```

} commands

```
}
```

```
  event void Timer0.fired()
```

```
  {  
    call Leds.led0Toggle();
```

Event

```
  }
```

```
  event void Timer1.fired()
```

```
  {  
    call Leds.led1Toggle();
```

```
  }
```

```
  event void Timer2.fired()
```

```
  {  
    call Leds.led2Toggle();
```

```
  }
```

```
}
```





# Lecture Overview

- 1. Hardware Primer
- 2. Introduction to TinyOS
- 3. Programming TinyOS
- 4. Hands on section



# Try new applications

# Further Reading

- Go through the on-line tutorial:
  - <http://www.tinyos.net/tinyos-1.x/doc/tutorial/index.html>
- Search the help archive:
  - <http://www.tinyos.net/search.html>
- NesC language reference manual:
  - <http://www.tinyos.net/tinyos-1.x/doc/nesc/ref.pdf>



Thank you.