

# TinyOS

Computer Network  
Programming  
Wenyuan Xu

## Lecture Overview

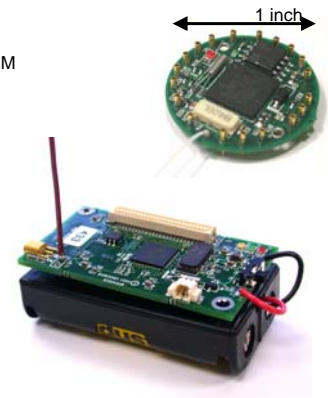
- 1. Hardware Primer
- 2. Introduction to TinyOS
- 3. Programming TinyOS
- 4. Network Communication

## UC Berkeley Family of Motes

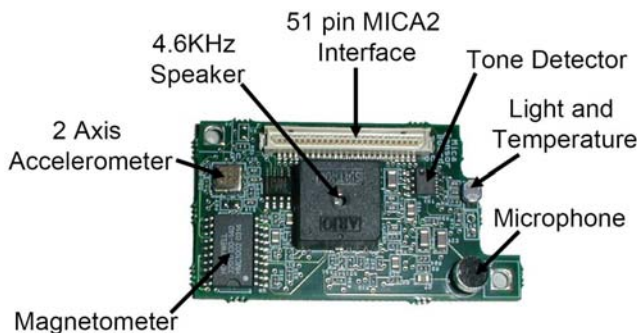
Mote Type Year	FitC 1998	Remi 1999	Remi 2 2000	Dot 2000	Mica 2001	Mica2Dot 2002	Mica 2 2002	Telos 2004
Microcontroller	AT90LS8535		ATmega163		ATmega128		TI MSP430	
Type								
Program memory (KB)	8		16		128		60	
RAM (KB)	0.5		1		4		2	
Active Power (mW)	15		15		8		33	
Sleep Power (µW)	45		45		75		75	
Wakeup Time (µs)	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 OdBm (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		59	
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

## Mica2 and Mica2Dot

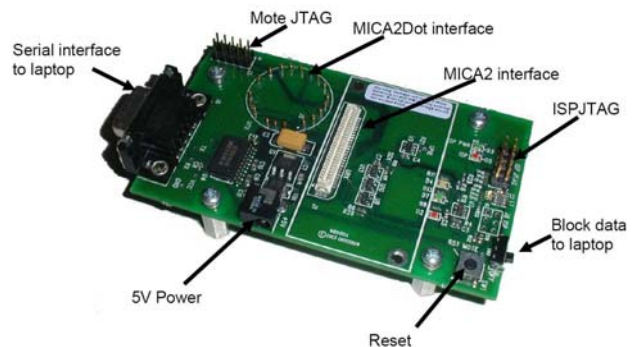
- ATmega128 CPU
  - Self-programming
  - 128KB Instruction EEPROM
  - 4KB Data EEPROM
- Chipcon CC1000
  - Manchester encoding
  - Tunable frequency
    - 315, 433 or 900MHz
  - 38K or 19K baud
- Lower power consumption
  - 2 AA batteries
- Expansion
  - 51 pin I/O Connector



## MTS300CA Sensor Board



## Programming Board (MIB510)



# Hardware Setup Overview



# Lecture Overview

- 1. Hardware Primer
- 2. Introduction to TinyOS
- 3. Programming TinyOS
- 4. Network Communication

# 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 and the 'make'

- Download
  - <http://www.tinyos.net/download.html>
- Directory Structure
  - /apps
    - /Blink
    - /Forwarder
  - /contrib
  - /doc
  - /tools
    - /java
  - /tos
    - /interfaces
    - /lib
    - /platform
      - /mica
      - /mica2
      - /mica2dot
    - /sensorboard
      - /micasb
    - /system
    - /types
- From within the application's directory:
  - **make (re)install.<node id> <platform>**
    - <node id> is an integer between 0 and 255
    - <platform> may be mica2, mica2dot, or all
    - Example: make install.0 mica2*
  - **make pc**
    - Generates an executable that can be run a pc for

# Build Tool Chain

- Convert NesC into C and compile to exec
- Modify exec with platform-specific options
- Set the mote ID
- Reprogram the mote

```

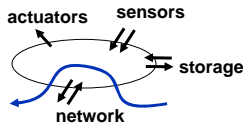
wenyuan@tinyos-laptop:~/opt/tinyos-1.x/apps/Blink$ ls
Blink.nc Blink.nc build Makefile README SingleTime.nc
wenyuan@tinyos-laptop:~/opt/tinyos-1.x/apps/Blink$ make mica2 install.0
compiling Blink to a mica2 binary
ncc -o build/mica2/main.exe -O3 -boardmicasb -target=mica2 -DCCIK_DEP_PAREQ=4330
02000 -Wall -wshadow -DEXP_TOS_IMPLGROUP=0x33 -wnesc-all -finline-limit=100000 -f
nesc-compile=build/mica2/app.c Blink.nc -lm
compiled Blink to build/mica2/main.exe
3626 bytes in ROM
49 bytes in RAM
avr-objcopy --output-target=srec build/mica2/main.exe build/mica2/main.srec
make mica2 reinstall.0 PROGRAMMER="STK" PROGRAMMER_FLAGS="" dprog=mb510 -dserial
s/dev/ttyUSB0 -dpart=ATmega128 -wr_fuse_eff "
make[1]: Entering directory /opt/tinyos-1.x/apps/Blink
installing mica2 binary
set-mote-id build/mica2/main.srec build/mica2/main.srec.0.out 'echo reinstall.0
[perl -pe 's/\.reinstall.//; s_$hex if /"0x/;';'
could not find symbol TOS_LOCAL_ADDRESS in build/mica2/main.exe, ignoring symbol
.
usps dprog=mb510 -dserial=/dev/ttyUSB0 -dpart=ATmega128 -wr_fuse_eff --eras
e --upload if=build/mica2/main.srec.0.out
Firmware Version: 2.1
stcwl AVR ATmega128 is found.
Uploading: flash
Fuse Extended Byte set to 0xff
make[1]: Leaving directory /opt/tinyos-1.x/apps/Blink
wenyuan@tinyos-laptop:~/opt/tinyos-1.x/apps/Blink$
    
```

# Lecture Overview

- 1. Hardware Primer
- 2. Introduction to TinyOS
- 3. Programming TinyOS
- 4. Network Communication

## Characteristics of Network Sensors

- Small physical size and low power consumption
- Concurrency-intensive operation
  - multiple flows, not wait-command-respond
- Limited Physical Parallelism and Controller Hierarchy
  - primitive direct-to-device interface
- Diversity in Design and Usage
  - application specific, not general purpose
  - huge device variation
  - => efficient modularity
  - => migration across HW/SW boundary
- Robust Operation
  - numerous, unattended, critical
  - => narrow interfaces



13

## A Operating System for Tiny Devices?

- Main Concept
  - HURRY UP AND SLEEP!!
    - Sleep as often as possible to save power
  - provide framework for concurrency and modularity
    - Commands, events, tasks
  - interleaving flows, events - never poll, never block
- Separation of construction and composition
- Programs are built out of components
  - Libraries and components are written in nesC.
  - Applications are too -- just additional components composed with the OS components
- Each component is specified by interfaces
  - Provides "hooks" for wiring components together
- Components are statically wired together based on their interfaces
  - Increases runtime efficiency

14

## Programming TinyOs

- There are two types of components in nesC:
  - Modules. It implements application code.
  - Configurations. It assemble other components together, called wiring
- A component does not care if another component is a module or configuration
- A component may be composed of other components via configurations
- A component provides and uses interfaces.
- A interface defines a logically related set of commands and events.
- Components implement the events they use and the commands they provide:

Component	Commands	Events
Use	Can call	Must implement
Provide	Must implement	Can signal

15

## Component Syntax - Module

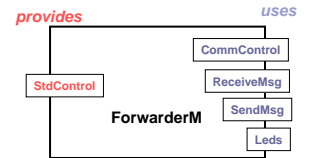
- A component specifies a set of interfaces by which it is connected to other components
  - provides a set of interfaces to others
  - uses a set of interfaces provided by others

module identifier specification module-implementation

```

module ForwarderM {
  provides {
    interface StdControl;
  }
  uses {
    interface StdControl as CommControl;
    interface ReceiveMsg;
    interface SendMsg;
    interface Leds;
  }
  implementation {
    ...// code implementing all provided commands
    and used events
  }
}
    
```

interface X as Y



= interface X as X

16

## Component Syntax - Configuration

```

configuration Forwarder {}
implementation
{
  components Main, LedsC;
  components GenericComm as Comm;
  components ForwarderM;

  Main.StdControl -> ForwarderM.StdControl;
  ForwarderM.CommControl -> Comm;
  ForwarderM.SendMsg -> Comm.SendMsg[AM_INTMSG];
  ForwarderM.ReceiveMsg -> Comm.ReceiveMsg[AM_INTMSG];
  ForwarderM.Leds -> LedsC;
}
    
```

configuration identifier specification configuration-implementation

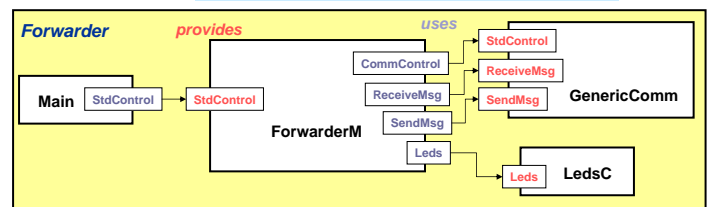
17

## Component Syntax - Configuration

```

configuration Forwarder {}
implementation
{
  components Main, LedsC;
  components GenericComm as Comm;
  components ForwarderM;

  Main.StdControl -> ForwarderM.StdControl;
  ForwarderM.CommControl -> Comm;
  ForwarderM.SendMsg -> Comm.SendMsg[AM_INTMSG];
  ForwarderM.ReceiveMsg -> Comm.ReceiveMsg[AM_INTMSG];
  ForwarderM.Leds -> LedsC;
}
    
```



## Interface Syntax- interface StdControl

- Look in <tos>/tos/interfaces/StdControl.nc

```
interface StdControl {
    // Initialize the component and its subcomponents.
    command result_t init();

    // Start the component and its subcomponents.
    command result_t start();

    // Stop the component and pertinent subcomponents
    command result_t stop();
}
```

- Multiple components may provide and use this interface
- Every component should provide this interface
  - This is good programming technique, it is not a language specification
  - To understand how an application works, start from reading the implementation of StdControl.init(), StdControl.start().

19

## Interface Syntax- interface SendMsg

- Look in <tos>/tos/interfaces/SendMsg.nc

```
includes AM; // includes AM.h located in <tos>\tos\types\

interface SendMsg {
    // send a message
    command result_t send(uint16_t address, uint8_t length,
        TOS_MsgPtr msg);

    // an event indicating the previous message was sent
    event result_t sendDone(TOS_MsgPtr msg, result_t success);
}
```

- Includes both command and event.
- Split the task of sending a message into two parts, send and sendDone.

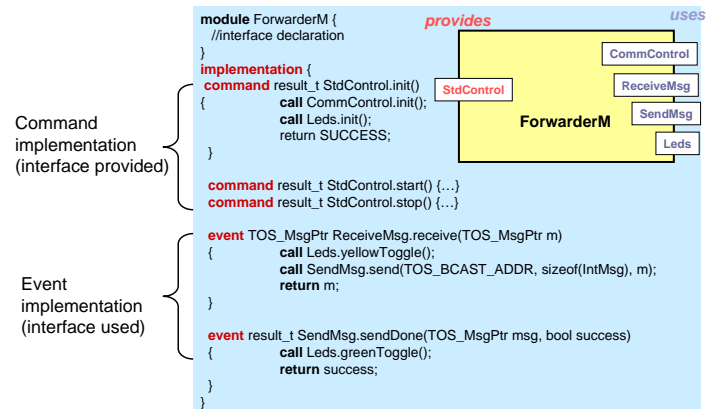
20

## Configuration Wires

- A configuration can bind an interface user to a provider using  $\rightarrow$  or  $\leftarrow$ 
  - User.interface  $\rightarrow$  Provider.interface
  - Provider.interface  $\leftarrow$  User.interface
- Bounce responsibilities using =
  - User1.interface = User2.interface
  - Provider1.interface = Provider2.interface
- The interface may be implicit if there is no ambiguity
  - e.g., User.interface  $\rightarrow$  Provider  $\leftrightarrow$  User.interface  $\rightarrow$  Provider.interface

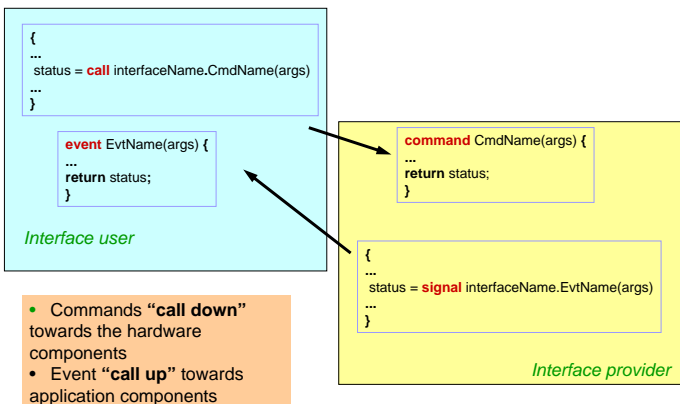
21

## Component implementation



22

## TinyOS Commands and Events



23

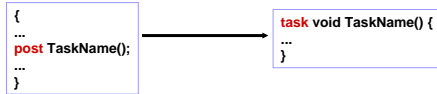
## TinyOs Concurrency Model

- TinyOS executes only one program consisting of a set of components.
- Two type concurrency:
  - Task
  - Hardware event handler
- Tasks:
  - Time flexible
  - Longer background processing jobs
  - Atomic with respect to other tasks (single threaded)
  - Preempted by event
- Hardware event handlers
  - Time critical
  - Shorter duration (hand off to task if need be)
  - Interrupts task and other hardware handler.
  - Last-in first-out semantics (no priority among events)
  - executed in response to a hardware interrupt

24

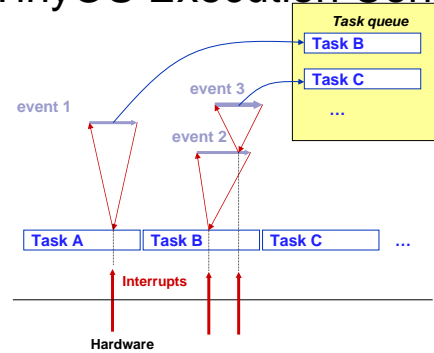
# Tasks

- Provide concurrency internal to a component
  - longer running operations
- Scheduling:
  - Currently simple FIFO scheduler
  - Bounded number of pending tasks
  - When idle, shuts down node except clock
- Uses non-blocking task queue data structure
- Simple event-driven structure + control over complete application/system graph
  - instead of complex task priorities and IPC



25

# TinyOS Execution Contexts



- Events generated by interrupts preempt tasks
- Tasks do not preempt tasks
- Both essential process state transitions

26

# Syntax summary

- Command:
  - Define: `command CmdName(args) {...}`
  - Call: `call interfaceName.commandName(arg);`
- Event:
  - Define: `event EvtName(args) {...}`
  - Signal: `signal interfaceName.EvtName(args)`
- Task:
  - Define: `task void TaskName(){...}`
  - Call: `post TaskName();`
- File name convention:
  - Filenames should be the name of the type contained within;
  - all nesC files have ".nc" as a suffix.
  - Example: 'genericComm' is defined in genericComm.nc
- How to read the code?
  - Find the top level application file, e.g. Forward.nc
  - Find the components used, find the component definition file.
  - Inside the component file, read from function: StdControl.init(), StdControl.start()

27

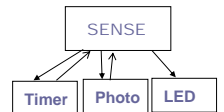
# Event-Driven Sensor Access Pattern

```

command result_t StdControl.start() {
    return call Timer.start(TIMER_REPEAT, 200);
}

event result_t Timer.fired() {
    return call sensor.getData();
}

event result_t sensor.dataReady(uint16_t data) {
    display(data)
    return SUCCESS;
}
    
```



- clock event handler initiates data collection
- sensor signals data ready event
- data event handler calls output command
- device sleeps or handles other activity while waiting
- conservative send/ack at component boundary

28

# Lecture Overview

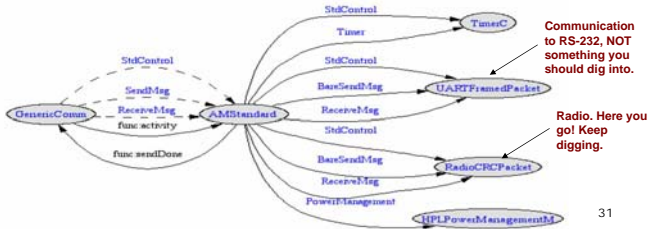
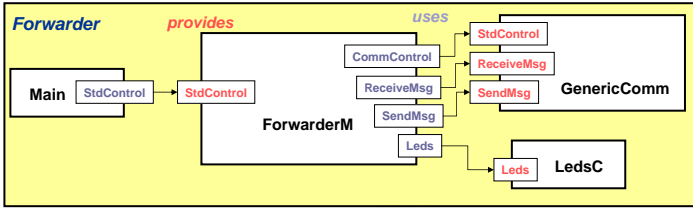
1. Hardware Primer
2. Introduction to TinyOS
3. Programming TinyOS
4. Network Communication

29

Do Your Homework!

30

# TinyOS



31

# 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>
- Getting started guide
  - [http://www.xbow.com/Support/Support\\_pdf\\_files/Getting\\_Started\\_Guide.pdf](http://www.xbow.com/Support/Support_pdf_files/Getting_Started_Guide.pdf)
- Hardware manual:
  - [http://www.xbow.com/Support/Support\\_pdf\\_files/MPR-M1B\\_Series\\_Users\\_Manual.pdf](http://www.xbow.com/Support/Support_pdf_files/MPR-M1B_Series_Users_Manual.pdf)

32