TinyOS Tutorial

Greg Hackmann

CSE 521S Fall 2010

Washington University in St. Louis

Outline

- **Installing TinyOS and Building Your First App**
- Hardware Primer
- Basic nesC Syntax
- Advanced nesC Syntax
- Network Communication
- Sensor Data Acquisition
- Debugging Tricks and Techniques

TinyOS Installation

- TinyOS Documentation Wiki: http://docs.tinyos.net/
  - Various installation options listed under “Getting started” section

- Pre-compiled .rpm and .deb packages for Fedora and Ubuntu Linux users
  - **Ubuntu users: be sure to remove `brltty` package**
- All necessary drivers already included with Linux kernel

TinyOS Installation (cont.)

- OS X unofficially supported but works well
- Precompiled packages available at http://www.cse.wustl.edu/~gwh2/tinyos-2.1.1.dmg
  - Need to install pySerial (http://pyserial.sourceforge.net) and FTDI FT232R serial drivers (http://www.ftdichip.com/Drivers/VCP.htm) separately
- Can also compile using MacPorts: see TinyOS Wiki

TinyOS Installation (cont.)

- Windows installation uses Cygwin to emulate Linux software layer
  - Problematic under XP, refuses to work on some Vista/7 machines (updating Cygwin after the installation may help)
  - `gcc` is also very slow under Cygwin
- "Running a XubunTOS Virtual Machine Image in VMware Player" tutorial recommended instead
  - Or pick your favorite VM software and install Ubuntu yourself

TinyOS Directory Structure

- /opt/tinyos-2.1.1 ($TOSROOT)
  - apps
  - support
    - make
    - sdk
  - tools
  - tos

make System

- `$TOSROOT/support/make` includes lots of `Makefiles` to support the build process
- Create a simple stub `Makefile` in your app directory that points to main component

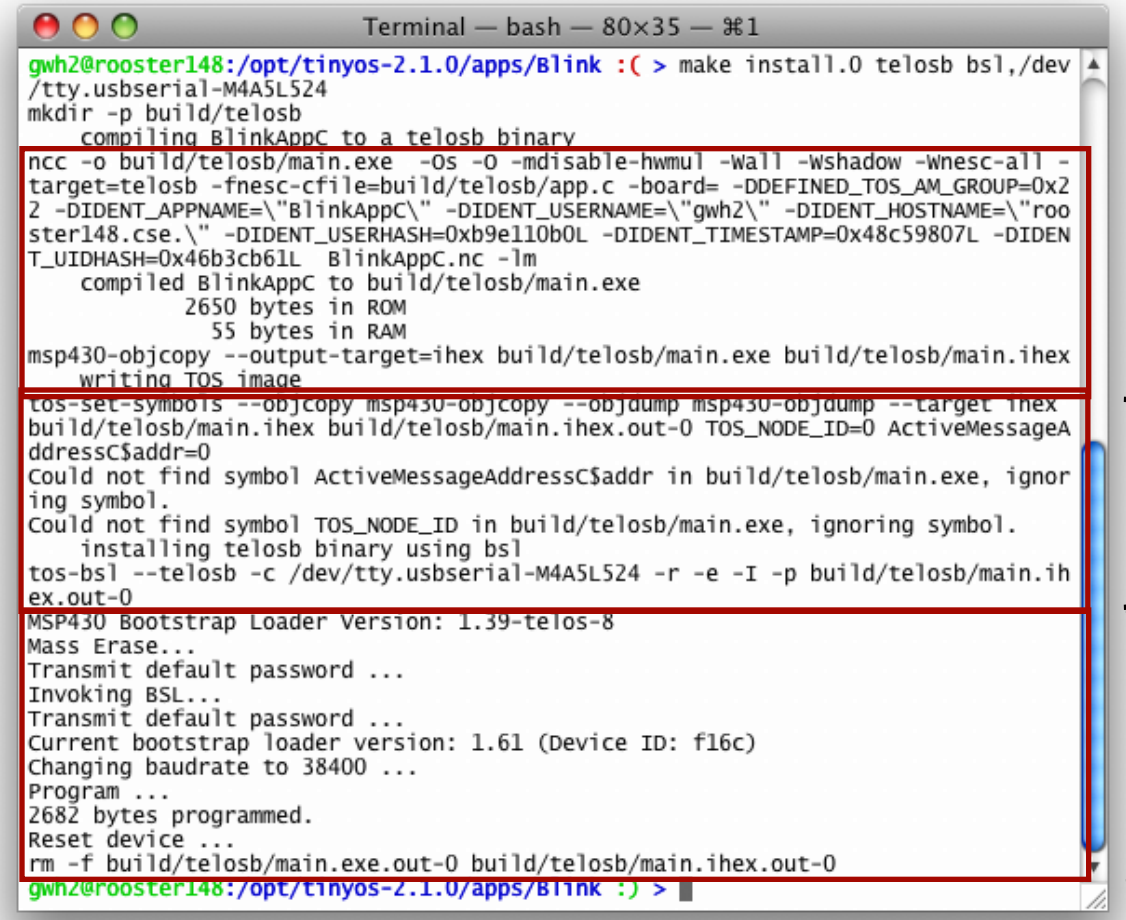
```
COMPONENT=[MainComponentC]
SENSORBOARD=[boardtype] # if needed
include $(MAKERULES)
```

- `make [platform]` in app directory
  - Builds but does not install program
  - `platform`: one of the platforms defined in `$TOSROOT/tos/platforms (mica2, micaz2, telosb)`

make System

- `make [re]install.[node ID] [platform] [programming options]`
  - node ID: `0 - 255`
  - programming options:
    - mica2/micaz: `mib510,/dev/ttyXYZ`
    - telosb: `bsl,/dev/ttyXYZ`
- `make clean`
- `make docs [platform]`
  - Generates HTML documentation in `$TOSROOT/doc/nesdoc/[platform]`

Build Stages



Preprocess .nc to .c, then compile .c to binary

Set AM address and node ID in binary

Program mote

How to Get Help

- TinyOS Documentation Wiki: http://docs.tinyos.net
- TinyOS Programming Manual: 139-page PDF intro to nesC and TinyOS 2.x: http://www.tinyos.net/tinyos-2.x/doc/pdf/tinyos-programming.pdf
- TinyOS Tutorials: short HTML lessons on using parts of TinyOS (sensors, radio, TOSSIM, etc.):http://docs.tinyos.net/index.php/TinyOS_Tutorials

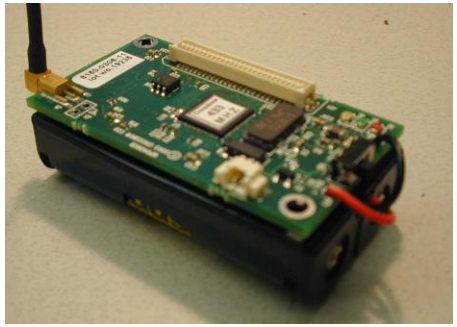
How to Get Help

- nesdoc: annotated API for all interfaces and components in TinyOS: http://docs.tinyos.net/index.php/Source_Code_Documentation
- TinyOS Enhancement Protocols (TEP): formal documentation for TinyOS features: http://docs.tinyos.net/index.php/TEPs

Outline

- Installing TinyOS and Building Your First App
- **Hardware Primer**
- Basic nesC Syntax
- Advanced nesC Syntax
- Network Communication
- Sensor Data Acquisition
- Debugging Tricks and Techniques

MICA2 Mote (MPR400CB)



128KB Instruction EEPROM

4KB Data EEPROM

Atmel ATmega128L µP 7.3827MHz (8 MIPS)

2 AA

512KB External Flash Memory

UART 2

Chipcon CC1000 radio, 38K or 19K baud, Manchester, 315, 433, or 900MHz

SPI bus

ADC 0-7
UART 1
I2C Bus

3 LEDs

Quantity: 56

51 pin I/O Connector

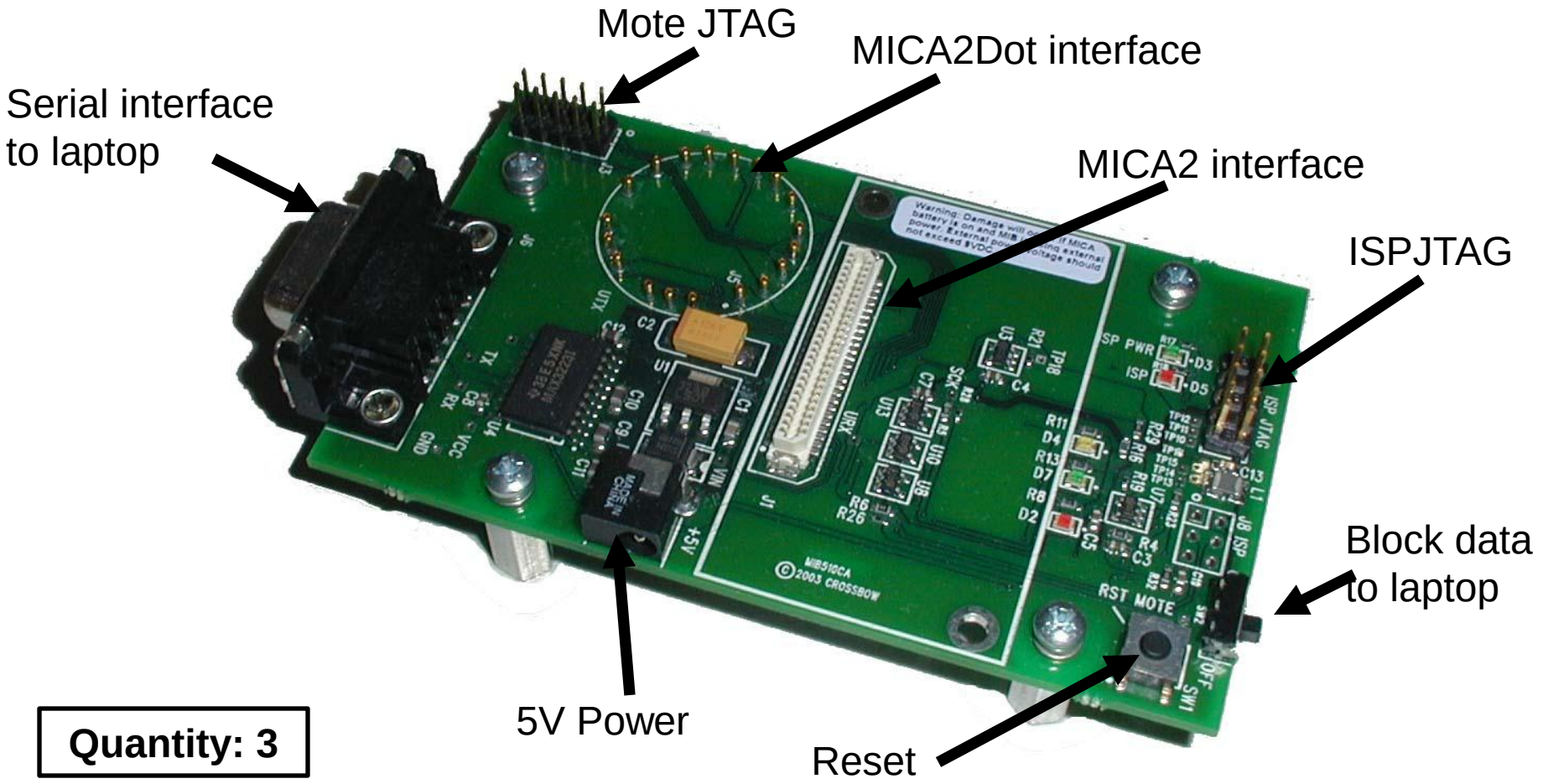
To Sensors, JTAG, and/or Programming Board

MPR2400 MICAz

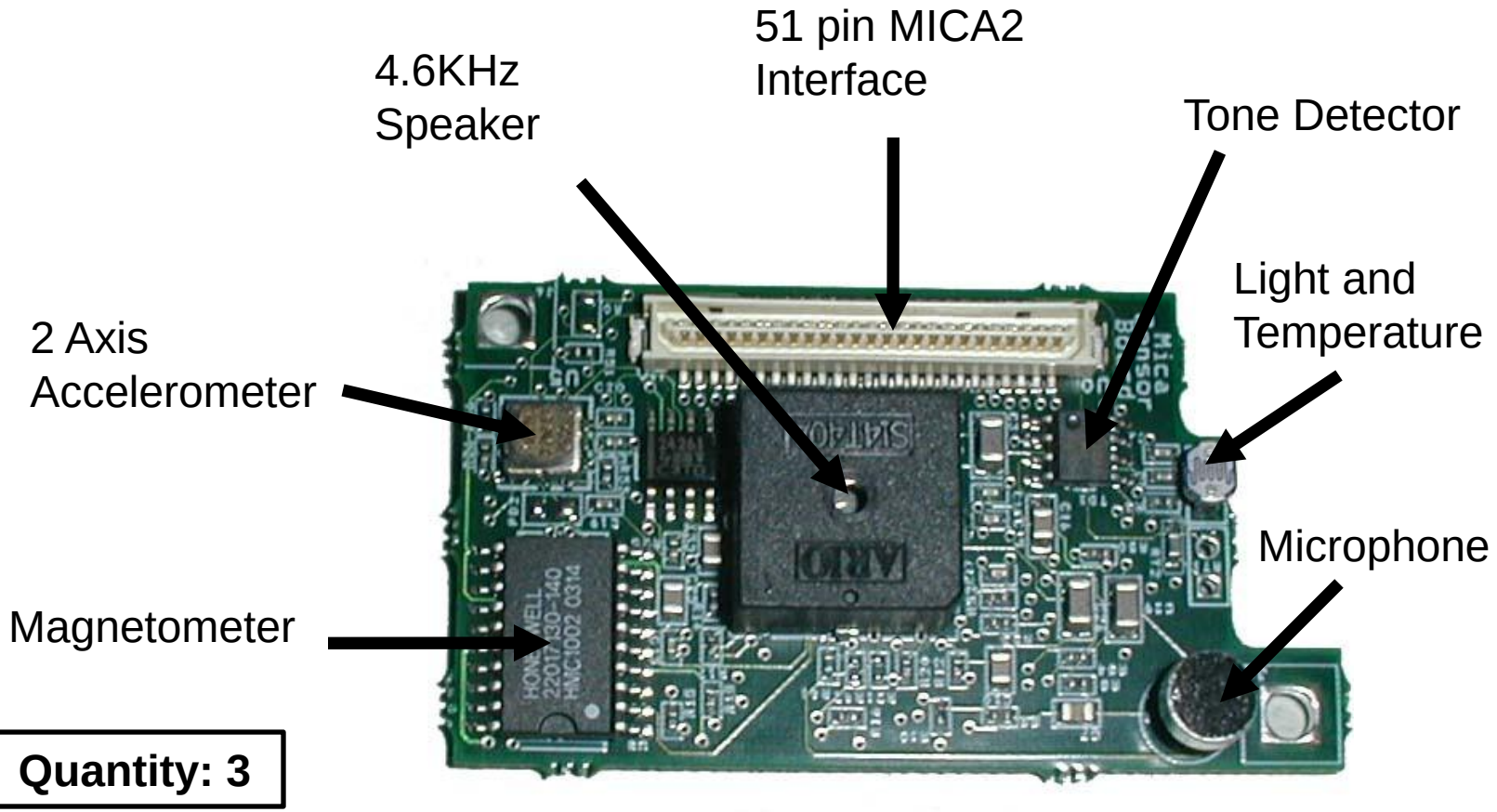
- Same as Mica2 except with IEEE 802.15.4 radio
  - 2.4GHz
  - 250kbps
- Quantity: 7

ZigBee™Alliance

Programming Board (MIB510)



MTS310CA Sensor Board

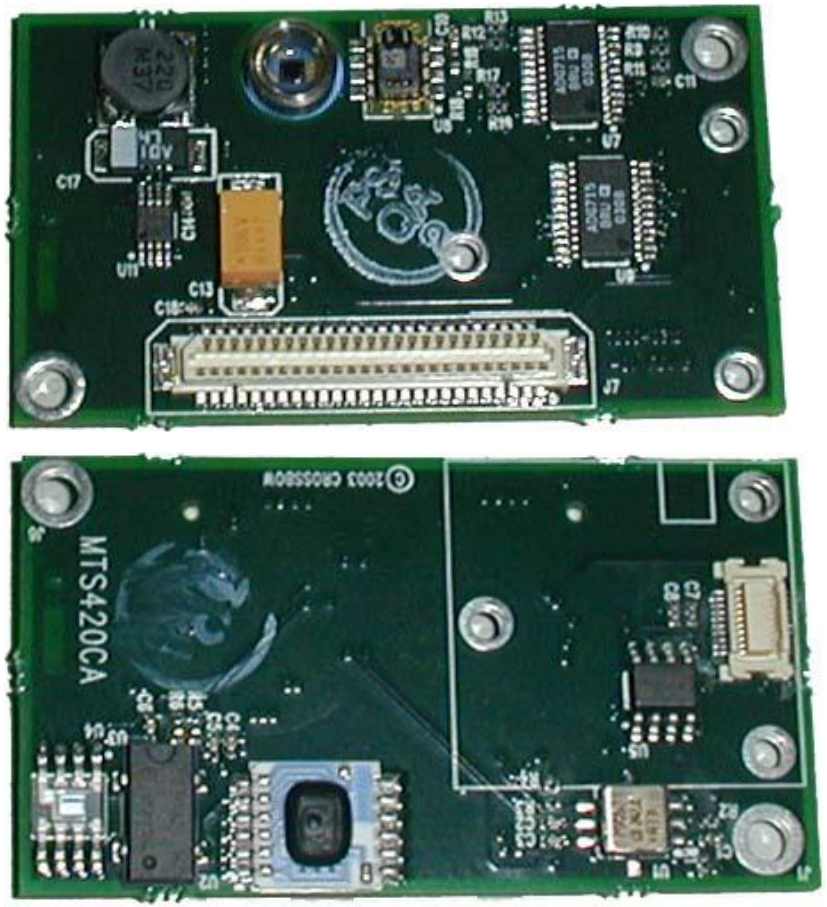


Quantity: 3

MTS420 Sensor Board

- GPS
- Accelerometer
- Light
- Temperature
- Humidity
- Barometric Pressure
- 2KB EEPROM Conf.

- Quantity: 1



Tmote Sky (aka TelosB)

- IEEE 802.15.4 Radio
  - 250kbps
- TI MSP430 microcontroller
  - 16MHz, 16 MIPS, 10kB RAM
- Integrated antenna & USB interface
- Low power utilization
  - 1.8mA/5.1µA vs. Mica 2's 8mA/15µA
- Quantity w/o on-board sensors: 6
- Quantity w/temperature, light, and humidity sensors: 20
- Quantity w/SBT80 sensor board: 4

NSLU2 Network Storage Link ("Slug")

- 266MHz Xscale CPU, 32MB SDRAM, 8MB flash, 1x Ethernet port
- Wired power
- No built-in radio, but 2x USB 2.0 ports for add-on 802.11/Bluetooth/mote interface

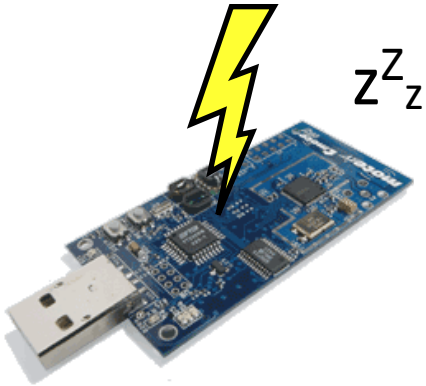
- Can be easily converted to an embedded Linux box with third-party firmware
  - Our testbed uses the OpenWrt distribution (http://openwrt.org)
- Quantity: 15

Outline

- Installing TinyOS and Building Your First App
- Hardware Primer
- **Basic nesC Syntax**
- Advanced nesC Syntax
- Network Communication
- Sensor Data Acquisition
- Debugging Tricks and Techniques

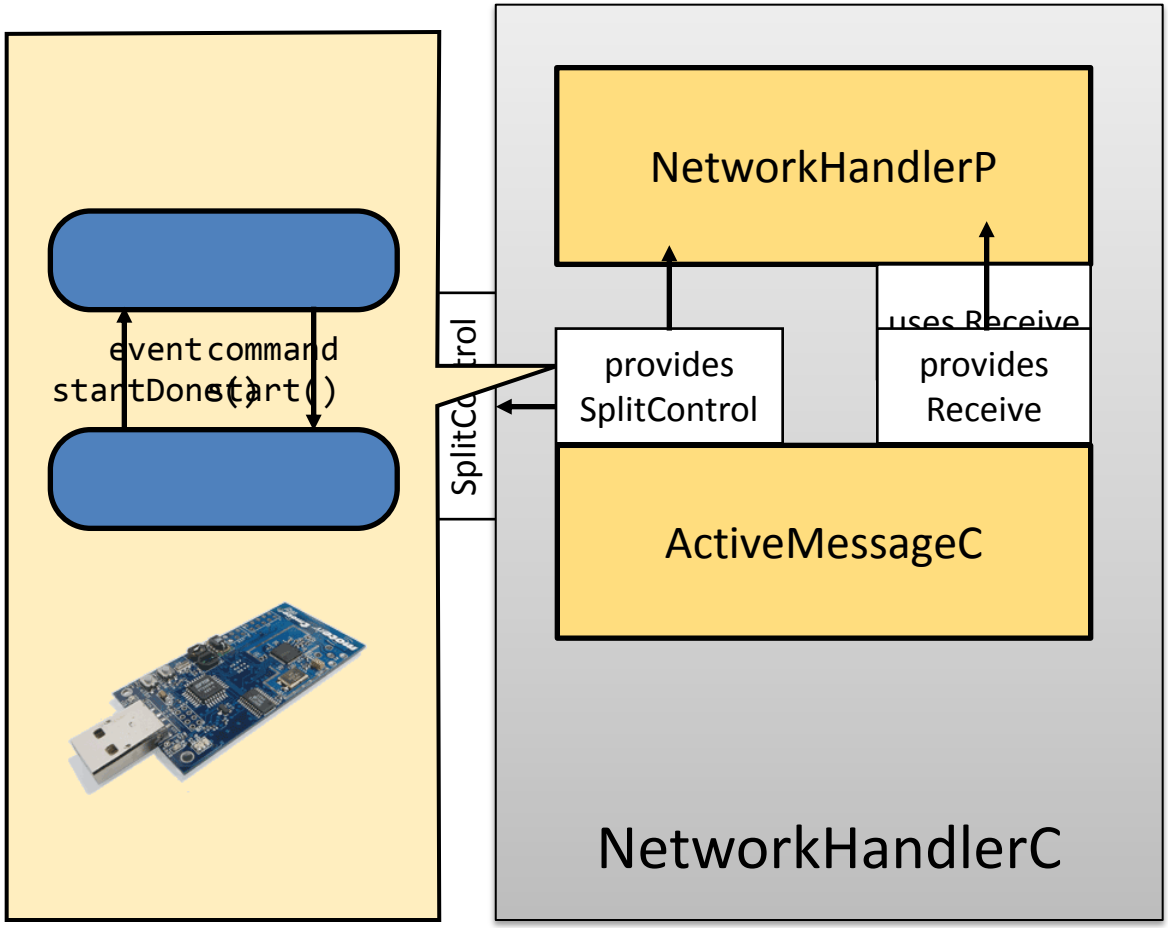
TinyOS Execution Model

- To save energy, node stays asleep most of the time
- Computation is kicked off by hardware interrupts
- Interrupts may schedule tasks to be executed at some time in the future
- TinyOS scheduler continues running until all tasks are cleared, then sends mote back to sleep

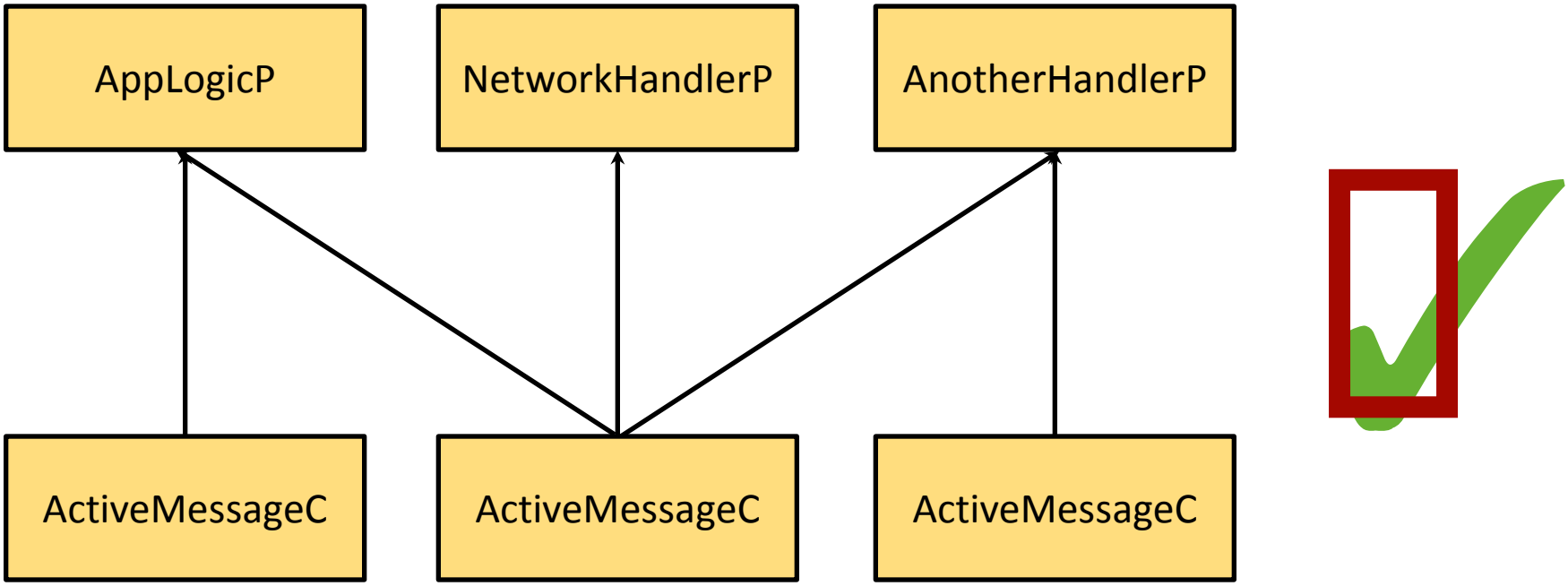


handlePacket | readSensor | sendResponse

TinyOS Component Model



Components != Objects



Interfaces

- List of exposed events and commands
- Like ordinary C function declarations, except with `event` or `command` in front

```
interface Receive {
  event message_t * Receive(message_t * msg, void * payload,
    uint8_t len);
  command void * getPayload(message_t * msg, uint8_t * len);
  command uint8_t payloadLength(message_t * msg);
}
```

Modules

- Modules `provide` the implementation of one or more interfaces
- They may consume (`use`) other interfaces to do so

```
module ExampleModuleP {
  provides interface SplitControl;
  uses interface Receive;
  uses interface Receive as OtherReceive;
}
implementation {
  ...
}
```

- "Rename" interfaces with the `as` keyword -- required if you are using/providing more than one of the same interface!

Modules

- `implementation` block may contain:
  - Variable declarations
  - Helper functions
  - Tasks
  - Event handlers
  - Command implementations

Modules: Variables and Functions

- Placed inside `implementation` block exactly like standard C declarations:

```
...
implementation {
  uint8_t localVariable;
  void increment(uint8_t amount);

  ...

  void increment(uint8_t amount) {
    localVariable += amount;
  }
}
```

Modules: Tasks

- Look a lot like functions, except:
  - Prefixed with `task`
  - Can't return anything or accept any parameters

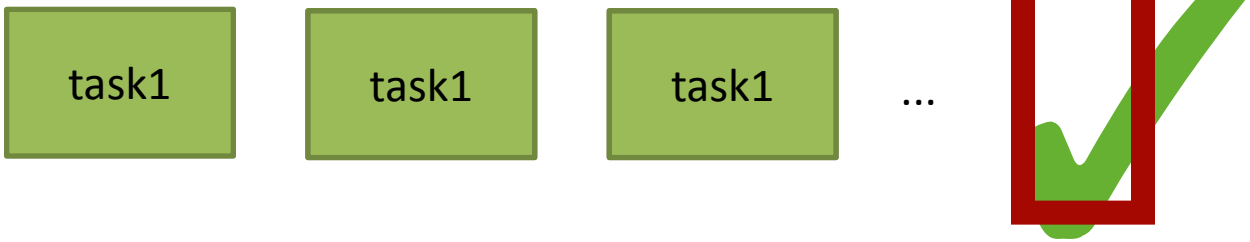
```
implementation {
  ...
  task void legalTask() {
    // OK
  }
  task bool illegalTask() {
    // Error: can't have a return value!
  }
  task void anotherIllegalTask(bool param1) {
    // Error: can't have parameters!
  }
}
```

Modules: Task Scheduling

- Tasks are scheduled using the `post` keyword

```
error_t retval;
retval = post handlePacket();
// retval == SUCCESS if task was scheduled, or E_FAIL if not
```

- TinyOS guarantees that task will *eventually* run
  - Default scheduling policy: FIFO



Modules: Commands and Events

- Commands and events also look like C functions, except:
  - they start with the keyword `command` or `event`
  - the "function" name is in the form `InterfaceName.CommandOrEventName`
- e.g.

```
implementation {
  command error_t SplitControl.start() {
     // Implements SplitControl's start() command
  }

  event message_t * Receive.receive(message_t * msg, void * payload,
    uint8_t len) {
     // Handles Receive's receive() event
  }
}
```

Modules: Commands and Events

- Commands are invoked using the `call` keyword:

```
call Leds.led0Toggle();
// Invoke the led0Toggle command on the Leds interface
```

- Event handlers are invoked using the `signal` keyword:

```
signal SplitControl.startDone();
// Invoke the startDone event handler on the SplitControl interface
```

Modules: Commands and Events

- A command, event handler, or function can call or signal *any* other command or event from *any* interface wired into the module:

```
module ExampleModuleP {
  uses interface Receive;
  uses interface Leds;
}
implementation {
  event message_t Receive.receive(message_t * msg, void * payload,
    uint8_t len) {
     // Just toggle the first LED
     call Leds.led0Toggle();
     return msg;
  }
  ...
}
```