Sensor Network Application Development
ZIGBEE CONCEPTS 0

Cruise Summerschool

Johannes Kepler University
November 5 - 7, 2007, Linz / Austria
Dipl.-Ing. Andreas Riener
riener@pervasive.jku.at

Overview

Structure of this lesson

ZigBee-Application development

- Introduction
- Interoperability of components (TinyOS, nesC, Tools, Makefiles)
- Example application
- Using the programming environment, adaptations

ZigBee development environment – What`s beeing installed?

TinyOS

- Open-source operating system designed for wireless embedded sensor networks
- Debugging functionality included
- Component-based architecture
- Enables rapid implementation with minimal code size
- Event-driven execution model enables fine-grained power management, power/battery saving capabilities, etc.

nesC

- (New) structured component-based language
- C-like syntax (extension of C-language)
- TinyOS operating system, libraries and applications are written in nesC
- Supports the TinyOS concurrency model
- Goal: build components that can be easily composed into complete, concurrent systems

ZigBee development environment – What`s beeing installed?

AVR-Tools

- Suite of software development tools for Atmel’s AVR processors (AVR...8-bit RISC microcontroller [µC], developed by ATMEL in 1996)
- Acronym AVR
  - "Advanced Virtual RISC"
  - Initials of the chip's designers:
    Alf-Egl Bogen and **V**egard Wollan, **R**ISC
  - Atmel says that the name AVR is not an acronym and does not stand for anything in particular...

Cygwin

- A linux-like environment for MS Windows
- User interface for compiling and downloading Mote applications
- Many open-source programs on Unix have been ported to Cygwin, including: X Window System, KDE, Gnome, Apache, TeX, and others

TinyOS programming architecture/concept

Application

- A TinyOS / nesC **application** consists of one or more **components**
- Are linked / wired together via **configurations** to get a **run-time executable**

Component

- Basic building blocks for nesC applications
- Two types: Modules & Configurations
- Can provide and use interfaces (bidirectionality in contrast to unidirectionality in common programming languages like JAVA)

Module

- A component that implements one or more interface
- Contains application code...

TinyOS programming architecture/concept

Configuration

- A **component** that wires **other components** together
- Idea: build **applications as set of modules, wiring together** by providing a configuration

Interface

- **Abstract definition** of the interaction of two components
- Concept is similar to JAVA interfaces
- NesC-interfaces are **bidirectional**

TinyOS programming architecture/concept

Components wired together as a **configuration** to create a Mote application

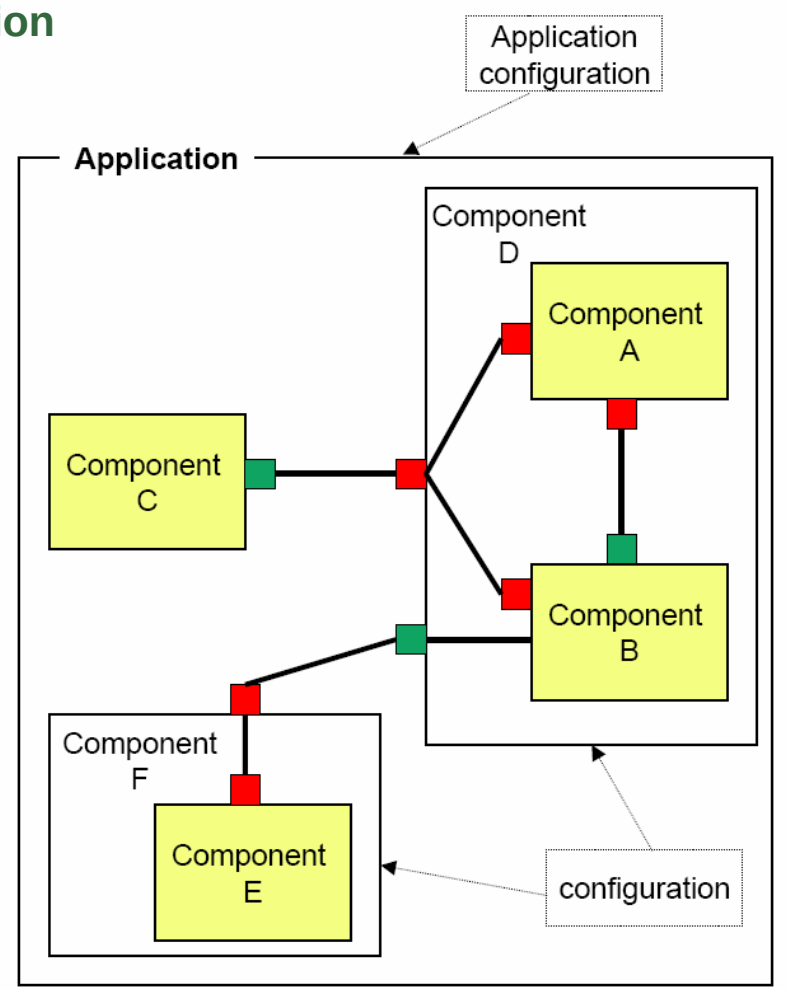
- Knowing how to wire components is sufficient to build an application

Link component interfaces

- **Provides**
- **Uses**

Two types of components

- **Module**
  component written with code
- **Configuration**
  wires components together



TinyOS Software Architecture

Somewhat object oriented

- Object abstraction
- Layered “inheritance” approach

In structure similar to Hardware Description Languages (HDL) like VHDL, Verilog, etc.

- Wired and bi-directional interfaces

Standard programming software in TinyOS is the UISP (Micro In-System Programmer)

- various arguments according to
  - programming hardware (mib510, eprb = mib600, etc.)
  - particular programming actions (erase, verify, re/install, etc.)
- Example: compile and install a program “DEMOAPP” on a zigbee mote connected via serial interface board (MIB510)

```
open a new cygwin window
$ CDXBOW                                  (alias to crossbow-home)
$ cd DEMOAPP
$ make micaz                              (compile for specific platform)
$ make micaz reinstall mib510,com1        (transfer via MIB510 serial
                                           programmer)
```

The Makefile "MakeXbowlocal”

Developed by Crossbow to provide a convenient way to change mote parameters

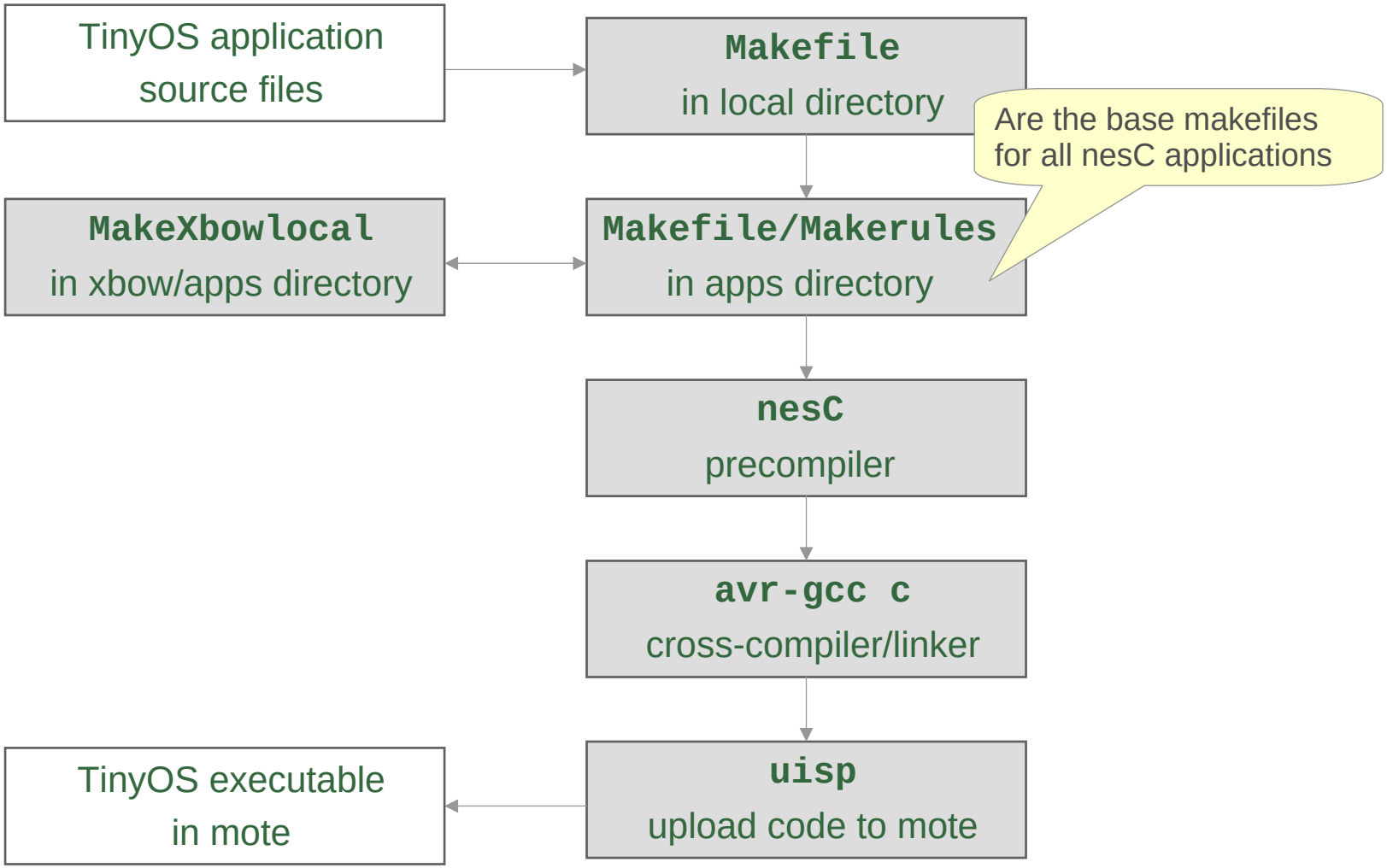
- local group ID // channel (RX/TX frequency) // transmission power // etc.
- resides in **`/tinyos-1.x/contrib/xbow/apps`**
- Used by adding a **`include`**-statement in the applications local makefile

```
include $(XBOWROOT)/../apps/MakeXbowlocal
```

File-Structure / Content (extract)

```
DEFAULT_PROGRAM=mib510  # default mote programmer
#DEFAULT_PROGRAM=eprb
MIB510=COM1             # default port for serial programmer
#EPRB=140.78.95.56
DEFAULT_LOCAL_GROUP=125 # default mote group id
CFLAGS = -DCC1K_DEFAULT_FREQ=RADIO_916BAND_CHANNEL_00
                        # set radio channel frequency
CFLAGS += -DRADIO_XMIT_POWER=0xFF
# set radio power; 0x00 least power, 0xFF max transmit power
CFLAGS +=-DCC2420_DEF_CHANNEL=26
# select zigbee channel; 15,20,25,26: no-overlap with 802.11
```

TinyOS – The "make" flow



The First TinyOS Application

Example application: "Blink"

- A simple test program "Blink" causes the red LED on the mote to turn on and off at 1Hz (example program can be found in /apps/Blink in the TinyOS tree

Components

- The application is composed of two **components**: a **module**, called *"BlinkM.nc"*, and a **configuration**, called *"Blink.nc"*

- *"Blink.nc"* is the configuration file for the Blink application and the source file that the nesC compiler uses to generate an executable file
  Note: All applications require a top-level configuration file, which is typically named after the application itself

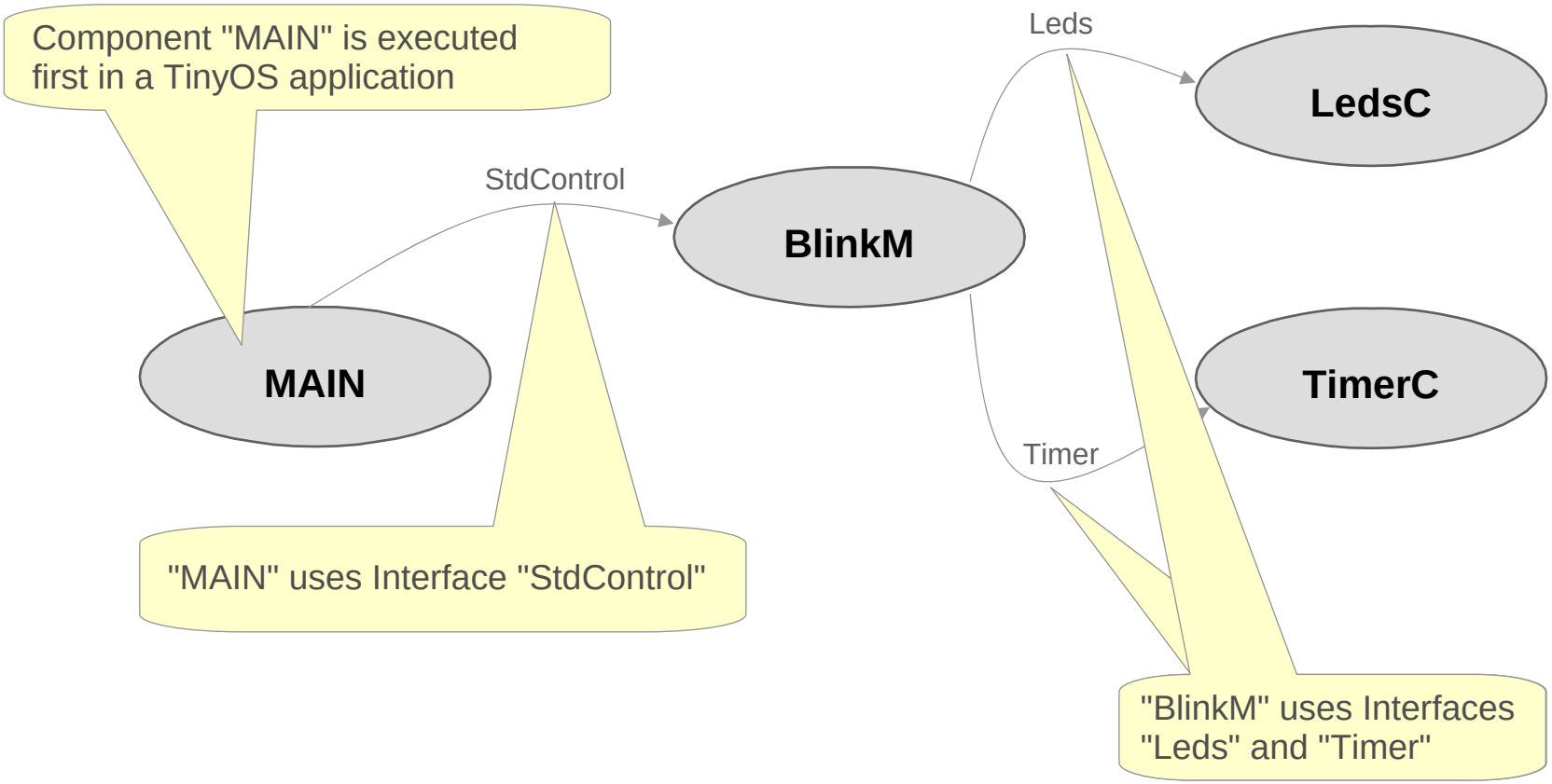
- *"BlinkM.nc"* actually provides the **implementation** of the Blink application. *"Blink.nc"* is used to wire the *"BlinkM.nc"* module to other components that the Blink application requires.

Distinction between modules and configurations allows to quickly "snap applications together"

More tutorials can be found online: http://www.tinyos.net/tinyos-1.x/doc/tutorial

Application architecture

Overview



Application Building Blocks

Configuration Blink.nc

- Used to assemble other components together (this is called **wiring)**
- Every nesC application is described by a **top-level configuration** that wires together the components inside
- The first 2 lines state that this is a configuration called "Blink". Within the empty braces here it is possible to specify uses and provides clauses, as with a module. **This is important to keep in mind: a configuration can use and provide interfaces!**
- The **components** line specifies the set of components that this configuration references (in this case Main, BlinkM, TimerC, and LedsC)

Blink.nc

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

Application Building Blocks

Interface StdControl.nc

- This is a common interface used to initialize and start TinyOS components
- Interfaces reside in directory **`/tos/interfaces/`**
- StdControl defines 3 commands: **init()**, **start()** and **stop()**

StdControl.nc

```
interface StdControl{
  command result_t init();
  command result_t start();
  command result_t stop();
}
```

Application Building Blocks

Interface Leds.nc

- Leds defines multiple commands, e.g. **init()**, **On()/Off()/Toggle()** for red, yellow and green LEDs, and **get()/set()** methods

Leds.nc

```
interface Leds {
  command result_t init();
  command result_t redOn();     //yellowOn, greenOn
  command result_t redOff();    //yellowOff, greenOff
  command result_t redToggle(); //yellowToggle, greenToggle
  command uint8_t get();
  command result_t set(uint8_t value);
}
```

Application Building Blocks

Interface Timer.nc

- Timer defines 3 commands: **start()**, **stop()** and **fired()**
- **start()** command is used to specify the type of timer (**TIMER_REPEAT** or **TIMER_ONE_SHOT**) and the interval at which the timer will expire (in milliseconds); repeating timer is stopped by the **stop()** command
- Applications receives an **fired()** event if the timer has expired

Timer.nc

```
interface Timer {
  command result_t start(char type, uint32_t interval);
  command result_t stop();
  event result_t fired();
}
```

Application Building Blocks

Module BlinkM.nc

BlinkM.nc

```
module BlinkM {
  provides {
    interface StdControl;
  }
  uses {
    interface Timer;
    interface Leds;
  }
} // implementation follows...
```

- BlinkM module provides the interface **StdControl** (this means: BlinkM implements the StdControl interface)
- BlinkM module uses two interfaces **Leds** and **Timer** (this means: BlinkM may call any command declared in the interfaces and must also implement any events declared in those interface)

Application Building Blocks

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

- Implementing the **Timer.fired()** event is necessary since BlinkM must implement any event from an interface it uses

Application: Compile and Execute

Cygwin environment

- Open a command window ("Start" - "Execute" "cmd")
- **`cd tinyos/cygwin/`**
- start cygwin.bat
- **`cd /opt/tinyos-1.x/apps/blink`**

Compile

- Compile for Zigbee Motes: **`make micaz`**
- Compile for PC simulation: **`make pc`**

Install

- Install on ZigBee Mote: **`make micaz install`**
- You can now test the program by unplugging the mote from the programming board and turning on the power switch...

- Simulate the functionality on your PC: **`build/pc/main.exe`**

  `> Usage: build/pc/main [-h|--help] [options] num_nodes_total`

  Simulate for one node: **`build/pc/main.exe 1`**

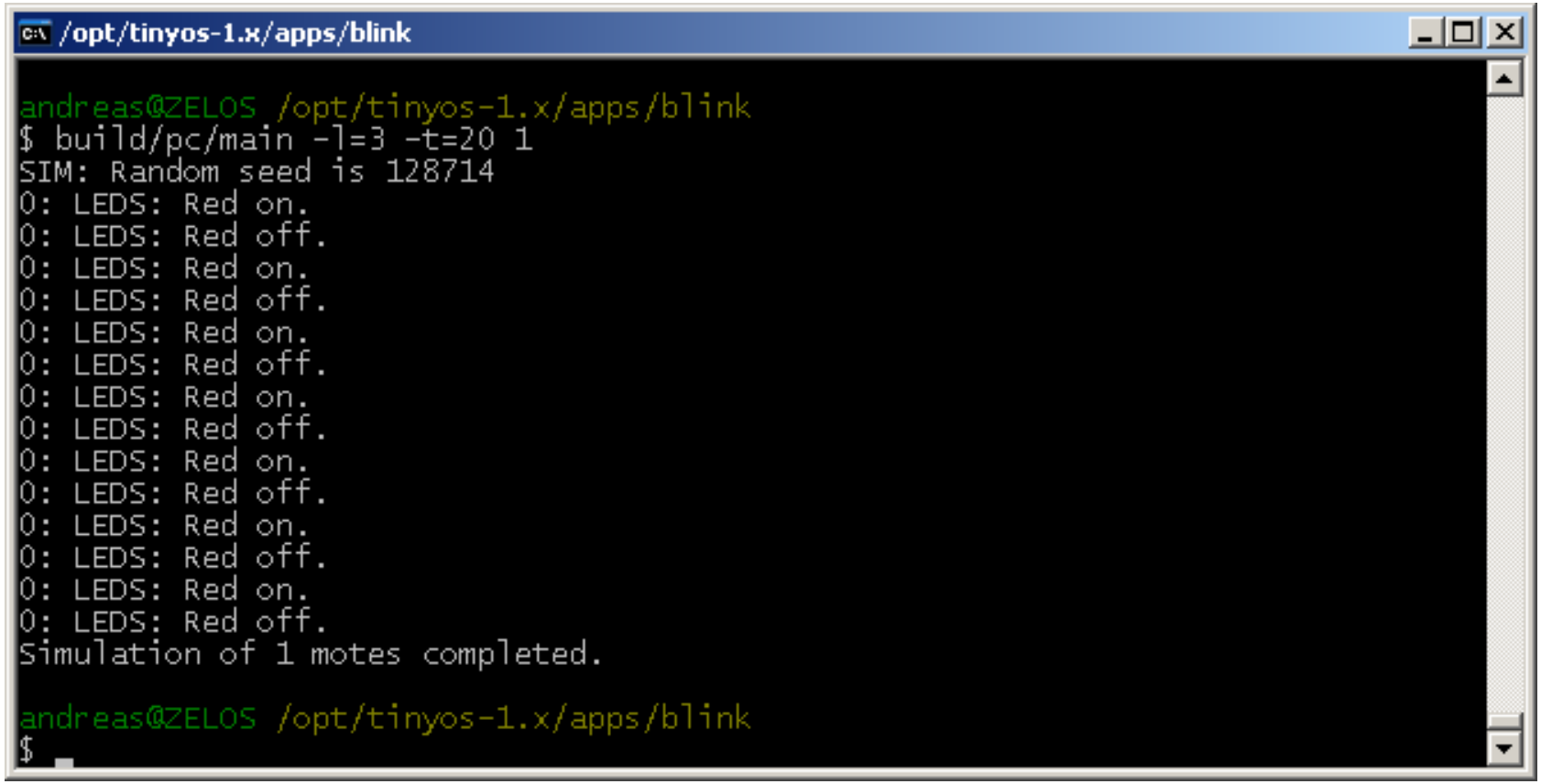
  `> export DBG=led`

  Only interesting debug messages from the LEDs are shown

Application: Simulation

Sample output

- **-l=3** -l=<scale> run simulation at <scale> times real time (fp constant)
- **-t=20** -t=<sec> run simulation for <sec> virtual seconds
- **1** numnodes number of nodes to simulate



TinyOS: Useful adaptations

Start Cygwin

- Run **`cygwin.bat`**

Configure alias

- Edit /etc/profile
  - **`vim /etc/profile`**
- Add following lines
  - **`alias CDTINYOS="cd /opt/tinyos-1.x"`**
  - **`alias CDJAVA="cd /opt/tinyos-1.x/tools/java"`**
  - **`alias CDXBOW="cd /opt/tinyos-1.x/contrib/xbow"`**
- Save and restart Cygwin
  - **`vim :x (save and exit)`**

Sensor Network Application Development
ZIGBEE CONCEPTS 0

Cruise Summerschool
Johannes Kepler University
November 5 - 7, 2007, Linz / Austria
Dipl.-Ing. Andreas Riener