TinyOS Lab Exercise
Introduction to TinyOS 2
TinyOS Lab Exercise in Ad Hoc and Sensor Networks

- **Sensor network programming in a nutshell**
  - Read ‘Getting started with TinyOS’ (at home)
  - Solve two **Lab-style** exercises on real hardware
    - Teams of two to three students are ideal
    - One lab working place is available in ETL F29
    - Reservation system on the course website
Wireless Sensor Nodes

- **Shockfish TinyNode**
  - Slow CPU
    - 8 MHz Texas Instruments MSP430 microcontroller
  - Little memory
    - 10 KByte RAM, 48 KByte ROM, 512 KByte external flash
  - Short-range radio
    - 868 MHz Xemics XE1205 ultra-low power wireless transceiver
  - Light sensor, temperature and humidity sensors
Exercise 1

- **Exchange of a sensor data**
  - Two sensor nodes are used for this task
  - One node periodically samples its light sensor and broadcasts the sensor reading over its radio
  - The other node listens for radio messages and signals if it is getting brighter or darker
    - Brighter → The green LED of the receiver is set
    - Darker → The red LED of the receiver is set
    - No significant change → The yellow LED is set

Light sensor

value = 520
Exercise 2

- Optical Communication using Morse Codes

```
A
on     off     on     off

A
on     off     on     off
```

To do
TinyOS

- TinyOS is an operating system for sensor nodes
  - Open source project with a strong academic background
  - Hardware drivers, libraries, tools, compiler

- TinyOS applications are written in nesC
  - C dialect with extra features
  - nesC compiler converts your application into plain C code

http://www.tinyos.net
NesC/TinyOS Programming Model

• Programs are built out of components

• Components use and provide interfaces

```c
interface Send {
    command error_t send(message_t* msg);
    event void sendDone(message_t* msg);
}
```

• Components are wired together by connecting interface users with interface providers
TinyOS Concurrency Model

• Tasks are executed sequentially by the TinyOS scheduler
  – only one task can be active at a time
  – Longer background processing jobs

  ```
  task void sendMessage() {
    call Send.send(...);
  }
  post sendMessage();
  ```

• Events (callbacks)
  – Short duration (hand off computation to tasks if necessary)

  ```
  event void Send.sendDone(...) {
    ... (do something)
  }
  ```
Split-Phase Operations in TinyOS

Java

```java
void sendMessage() {
    Message m = new Message();

    // send operation is blocking
    Send.send(m);

    // do next task
    nextTask();
}
```

TinyOS

```c
task void sendMessage() {
    message_t* msg = &msgBuffer;
    // command is non-blocking
    call Send.send(msg);
    // do something here
    ...
}

event void Send.sendDone(...) {
    // send completed, post next task
    post nextTask();
}
```
Final Remarks

- Code skeletons for both applications are provided on the lab PC. All software required during the lab is already pre-installed.

- The lab work place is in the ETL building (ETL F29). Keys must be fetched in our office ETZ G64.1 when your lab slot starts.

Register for your lab time slot on the course website