WSN-Projects: Lecture-1

Programming Motes using Low-level Languages

Organization

- Course website
  - http://cone.informatik.uni-freiburg.de/teaching/teamprojekt/bsprojects-WSN-w08

- Five lectures weeks
  1. Programming Motes using Low-Level Languages
  2. Programming Motes with TinyOS and NesC
  3. JVM and TakaTuka Introduction

  4. **Efficient JVM Research (part-1)**
  5. **Efficient JVM Research (part-2)**

- Team projects
  - Group of two

Grading

- One quiz 10%
- Team project 90%
  - Documentation 10%
    - Code comments
    - Final Report if required
  - Final presentation 10%
  - Your codes, its contribution to TakaTuka and its functionality 70%

Who Should Take This Course?

- A good programmer
- Someone looking for Bachelor thesis in next semester
- Someone likes to have publications/research

Quiz # 1

- Lets start with a quiz! ☺

Introduction

- Wireless sensor network (WSN)
  - WSN is a wireless network consists of tiny nodes (called motes) with sensing capabilities
  - Each mote usually has small computation power, communication radio, sensors and battery
- Ad-Hoc network Vs WSN
  - Nodes of Ad-Hoc Network are usually more powerful (e.g. PC)
  - Ad-Hoc network used Standard software
  - Goals are different
    - Ad-Hoc: Goal is to use existing devices and create a network for special cases
    - WSN: Goal is to be part of environment

WSN Devices and Software

- Available motes in department
  - Scatterweb-motes (Free University Berlin) – C
  - Crossbow-mica2 – TinyOS/NesC
  - Crossbow-TelosB – TinyOS/NesC
  - Sun-Spot – Java
    - Not really a typical mote
  - Sentilla Perk – Java
    - Not open source software
- Mica2
  - 128 KB of Flash
  - 4 KB of RAM

Objectives

- TakaTuka objectives
  - Develop an open source JVM for motes
  - Optimize Java Binaries (Jar files)
    - Currently we have by 90% smaller Java binaries
  - Make Java use less RAM
  - Make Java Run Faster
    - On PC and on motes

- Objective of this course
  - Projects and Bachelor theses for improving TakaTuka-JVM

LL-Programming of Motes:
Pro and Con

- Advantages
  - Every mote hardware support a low-level program
  - Low level programs are fast

- Disadvantages
  - Not portable
    - Have to change a program depending upon a hardware
  - Difficult to debug, program and extend

- Why to know LL-programming?
  - Concepts at LL languages are important
  - To develop high level languages, JVM and tools

Microcontroller families

- Three leading microcontrollers Families
  - Amtel AVR – 8 bit RISC
  - Texas Instruments (TI) MSP430 – 16 Bit
  - ARM – 32 bit RISC

- Mica2 – AVR ATmega128
- Sun-Spot – ARM9
- TelosB and Sentilla motes – MSP430

AVR ATmega 128

- See data sheet pages 2-11
- Pin and port for digital I/O
  - All Ports are bi-directional
  - Each port has 8 pins
- Input port
  - External connection can set the port-x values
  - Microcontroller can get the port-x values
- Output port
  - Microcontroller can set the port-x values
  - External connection can get the port-x values

AVR ATmega 128

- Direction of port -- Input/Output
  - Use Data Direction Register DDRx
    - DDRx = 1 --- Output Portx
    - DDRx = 0 --- Input Portx

LL Programming of Motes: Assembly

- ATmega128 data sheet page 12-14
  - Instruction set Summary
  - Flash (Programming memory) Vs Main memory speed difference?

- Write assembly program
  - Write 5 on PortA
  - PortA is connected with three LED
  - The LEDs will write two

LL Programming of Motes: Assembly

```
.include "m128def.inc"

ldi r16, 0xFF
out DDRB, r16

ldi r16, 0x05
out PORTA, r16

emptyloop:
rjmp emptyloop
```

LL Programming of Motes: C

- Assembly
  - Difficult to program
  - Longer programs need to be written
- AVRLibC
  - C library to be used with GCC on AVR microcontrollers
  - Open source and freely available

```
#include <avr/io.h>

int main (void) {
   DDRA  = 0xff;
   PORTA = 0x05;
   while(1) {
     /* "leere" Schleife*/;
   }
   /* wird nie erreicht */
   return 0;
}
```

LL Programming of Motes: Interrupts

- Interrupts
  - An interrupt is an asynchronous signal from hardware indicating the need for attention (e.g. data received, reset button pressed etc).
- Handling interrupts
  - The PC is jump to specific location associated with the Interrupts
    - For example reset interrupt location is 0x0000
  - The code at interrupt jump location should
    - save the registers values
    - Perform a task
    - Restore old register values
  - Return to previous task
- Read AVRLibc chapter-6 for details

Lab # 1

- Install AVRLibC
- Install AVR-GCC
- Install UISP
- Compile and transfer HelloWorld (LEDs) program written in C to a mote

The End