Programming Embedded Systems

Lecture 1
Introduction to the course
Monday Jan 17, 2011

Philipp Rümmer
Uppsala University
Philipp.Ruemmer@it.uu.se

Lecture outline

- Organisation
  - Teachers
  - Lectures, exercises, labs, project
- Topics + focus of the course
- Recap of the C language

About myself
(course instructor)

- Started at UU 12 days ago
- Diploma/M.Sc. from Karlsruhe University, 2004
- PhD from Chalmers, Gothenburg, 2008
- Main background: formal methods, verification

About Othmane Rezine
(assistant)

- PhD student in verification group
- Will do exercises + labs

Main structure of the course

Organisation of the course

Part 1
period 3, week 2-11
14 lectures (±), 6 assignments, 2 labs (3hp)

Two main topics:
use of operating systems for embedded systems
programming languages for embedded systems

Part 2
period 4, week 12-22
Embedded systems project (4hp)

Exam (3hp)

Lectures

- Mondays: 2 hours
- Wednesdays: 1 or 2 hours

Sometimes shared with assignment discussions

Assignments

- 6 assignments, solved by students individually
- Practice material from lectures
- Graded pass/fail
- ≥4 have to be handed in + passed
- Given out on Wednesdays:
  Jan 19, 26; Feb 2, 9, 16, 23
  handed in Wednesday following week before the lecture

http://www.philipp.ruemmer.org
Philipp.Ruemmer@it.uu.se

http://www.it.uu.se/katalog/othre279
othmane.rezine@it.uu.se

http://www.it.uu.se/edu/course/homepage/pins/vt11
Labs

- Done in groups of 2 people
- Small embedded systems projects
- No real embedded hardware → use of simulators
- We will give lab supervision during 1 lab slot per week (probably Fridays)

More infos will be on course page http://www.it.uu.se/edu/course/homepage/pins/vt11

Labs (2)

- **Lab 1:** use of embedded operating systems  
  **Start:** end of week 3 (this week)  
  **Deadline:** Feb 11, week 6
- **Lab 2:** embedded programming languages  
  **Start:** week 6  
  **Deadline:** Mar 4, week 9

More infos will be on course page http://www.it.uu.se/edu/course/homepage/pins/vt11

Project (period 4)

- Larger groups (3-4 people)
- Use of actual “embedded” hardware
- Details made available in due time

Course topics

Recapitulation: embedded systems

- System fulfilling particular task in a larger context
  - control systems in automotive, avionics, railway, process automation, communication, ...
  - mobile devices, ubiquitous devices
- Often with **real-time constraints**
- System: **hardware + software**

Course location: hardware/software co-design

Embedded systems require hardware and software to be designed simultaneously:

- **Abstract system specification/model**
- **Co-design**
- **System**
- **Hardware design**
- **Software design**

This course

Course covering (more) co-design: Microcontroller Programming, Lars Ericsson

Course location: considered hardware

- **8-bit micro-controllers** (e.g., 8051, AVR, ≤1KiB RAM)
- **larger micro-controllers** (e.g., ARM, PIC32, ≤1MiB RAM)
- **general-purpose processors** (e.g., x86, PowerPC)

Digital electronics design with VHDL

This course

Course covering: Microcontroller Programming, Lars Ericsson

Course location: development process

Image showing the development process with various stages such as project definition, detailed design, implementation, verification and validation, and operation and maintenance.
Course location: software architectures

- no operating system, simple control loop
- dedicated RTOS (e.g., LynxOS, VxWorks, Windows CE)
- POSIX 1003.1b (standard for real-time OSs)
- generic OS extended for RT (e.g., RT-Linux)
- generic OS (e.g., Unix, Windows)

This course

Operating systems courses

Course focus 1: embedded operating systems

- OS simplifies development of systems:
  - Multi-tasking, scheduling, task pre-emption, deadlines
  - Synchronisation, shared resources
  - Drivers for communication, periphery
  - Interrupt handling
- Large variety of OSs common for embedded systems
  - e.g., LynxOS, VxWorks, Windows CE, RT-Linux, FreeRTOS, ECOS, OSE, QNX, Integrity, ...

Main OS considered in course: FreeRTOS

- Small industrial OS, open-source (GPL)
- C API
- Satisfies hard real-time requirements
- Pre-emptive/cooperative multi-tasking, co-routines
- Fixed-priority scheduler
- Platforms: ARM, x86, Freescale, ...

http://www.freertos.org/

Related course topics

- Interrupt handling
- Accessing ports, devices like sensors, actuators, buses
- Memory management
- Synchronisation, inter-task communication
- Use of development boards
- Further OSs: RT-Linux, POSIX 1003.1b

Selection criteria for embedded operating systems

- Hardware requirements: Runtime overhead, memory, hardware platforms
- Provided features e.g., synchronisation, priority inversion prevention, real-time capabilities
- Guaranteed latencies Task switching Interrupt handling
- Provided schedulers: Static (table driven, priority driven) Dynamic (planning based, best effort) Fault tolerant scheduling

FreeRTOS (2)

- Will be introduced in lectures, used for assignments + labs (via simulators)

Related course topics (2)

- Requirements, safety properties
- Correctness: simulation, testing, debugging, verification
- Fault tolerance, redundancy
- Determinism, predictability
Course focus 2: embedded programming lang.

- Language in 1\textsuperscript{st} part of the course: C
- General trend, however: use of increasingly high-level languages
- Course will give an overview of different common paradigms

High-level imperative lang.

- Real-time Java, Ada 95
- High-level heap model
- Scoped memory (garbage collectors are difficult in real-time systems)
- Built-in real-time primitives

Synchronous languages

- Lustre, Esterel, Signal
- Execution governed by a global clock, static scheduling
- Determinism is guaranteed (even in presence of concurrency)
- Sometimes also used for modelling/prototyping

Graphical languages

- Matlab/Simulink, SCADE/Lustre
- Data-flow: programs as graphs of data-processing blocks

Graphical languages (2)

- State charts: graphical design of program states
- More details in course “Model-based design of embedded software,” Bengt Jonsson

Prevention of runtime errors

- Techniques to prevent null-pointer dereferences, arithmetic over-/under-flows, etc.
- Static analysis, model checking, theorem proving
- SPARK/Ada: “high-integrity” version of Ada; programs include assertions + specs

Further information

- Course page: http://www.it.uu.se/edu/course/homepage/pins/vt11
- There will be a web forum for questions (as soon as we found out how to set it up on studentportalen ...)
  - Always check the forum before sending us an email!
Next lecture

- Wednesday, Jan 19, 13:15, Pol_1245
- Intro to fixed-priority scheduling
- Intro + tutorial to FreeRTOS

Rest of this lecture

- Questionnaire
- Recap of C programming