ES Software Engineering

Lecture 8
Tools for an embedded system developer

In the previous lecture

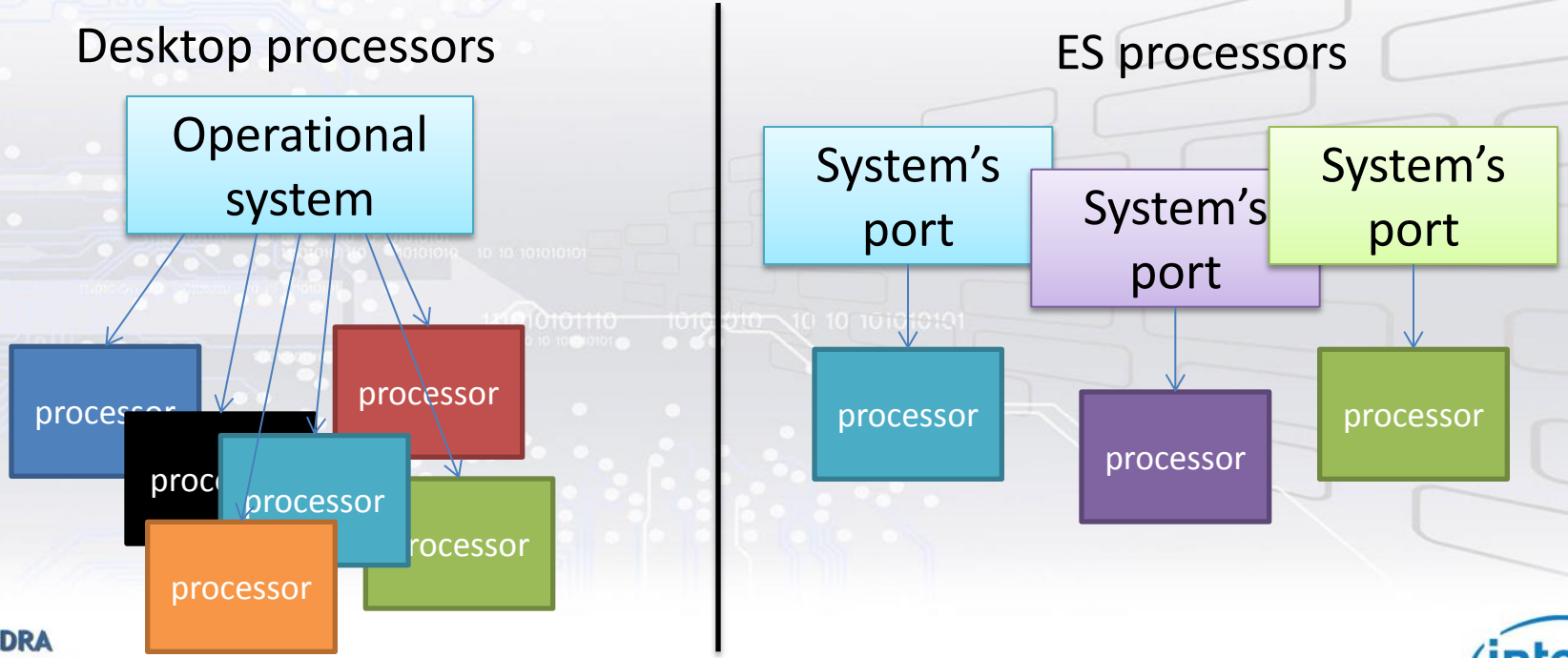
1. System & data architecture design
2. Agile programming
3. Open Source
4. Software Licences

Plan of the lecture

1. Porting an embedded system
2. Version control systems
3. Hardware emulators
4. Embedded Systems market in Poland

Porting of a system

Software vs. hardware



Porting of a system

I. Preparing a system port should be started with familiarizing oneself with:

1. Hardware platform documentation (i.e. mainly the processor documentation)
2. Operating system documentation

Preparing the port is about adapting the system architecture (software) to the hardware architecture (CPU).

II. The first practical steps are:

1. Compiling a chosen system port (made available to a different hardware platform)
2. Configuring a hardware emulator – adapting it to the processor architecture to which the port is being prepared

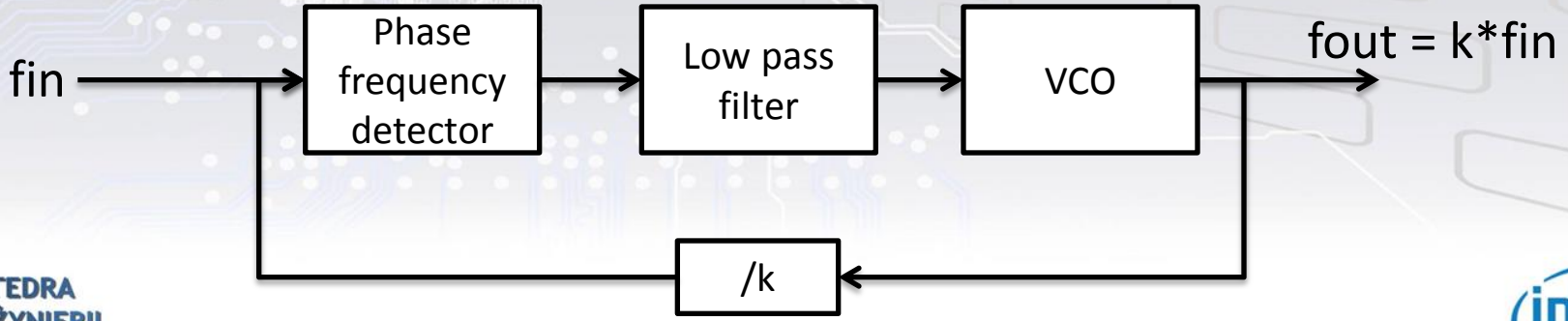
Porting of a system

III. Controlling clocks

Modern processors for embedded systems (e.g. ARM) have multiple clocks responsible for the operation of processor modules (memory management units, peripheral circuits).

The first communication with the ported system (e.g. via UART) requires correctly configured clocks and a PLL loop through appropriate entries in registers (see documentation).

PLL loop configuration:



Porting of a system

IV. Preparing assembly code to MMU and macros for handling the Cache.

Using MMU significantly speeds up the processor.

Assembly code and macros can usually be borrowed from system ports, for processors of a similar architecture but an older version. This code requires only minor modifications for adapting it to a new architecture.

The main objective of a developer is to define an offset and a number of memory pages which make it possible to map a virtual memory basing on the available physical memory resources. This requires handling the TLB mechanism.

Porting of a system

V. Handling interrupts

It goes down to handling a mechanism for calling system functions depending on register contents, which are set by an application initiating interrupts.

Developer must become familiar with the mechanism for initiating interrupts by the operating system and the interrupt handling mechanism in the processor.

VI. Handling the watchdog

The idea of a watchdog is that it sets the counter to a specific value, which is then decremented when measured by an independent processor clock. After reaching a certain minimum value – a specified operation is performed (interrupt initiation, processor reset etc.).

The developer's task is to determine the initial/final value, to define the clock and actions. As a rule, each of these steps is about setting appropriate entries in registries during a system startup.

Porting of a system

VI. Implementation of drivers

The final port of an embedded system should have drivers available for typical communication protocols (I2C, SPI, Ethernet etc.). Each processor has a set of communication devices compliant with these protocols, but handling them in the system requires implementing drivers basing on the documentation of processor transceivers and a methodology of defining drivers planned for the operating system.

It is one of the most time-consuming and difficult tasks.

Porting of a system

VII. Running tests

Many operating systems have a set of tests which, started in a permanent cycle, makes it possible to continuously control the operation of the port and the coherence of the system during subsequent modifications. The tests verify the correct operation of drivers, macros for memory handling, register entries compliance etc. Running all available tests or defining own tests is a separate project task.

Version control systems

Working on large projects requires using **version control systems.** Their task is to track source code changes and linking code modified by many developers at different moments.

Version control systems:

1. Local – makes it possible to write data on a local computer

- RCS
- GNU Source Code Control System
- SCCS

Version control systems

2. Distributed – based on the P2P architecture
- Git
- GNU Arch
- svk
- Bazaar
- BitKeeper

3. Centralized – based on the client-server architecture
- CVS
- GNU CSSC
- Supervision – SVN
- Visual Studio Team Foundation Server

Version control systems

Functionalities of version control systems:

RCS -> CVS -> SVN

RCS operates on single files of a local computer, and the history of the changes made is stored in a file named *plik.v*

The system, because of being used on a local machine (i.e. by a single user) does not offer any additional functionalities such as controlling changes made by multiple users.

Version control systems

Functionalities of version control systems:

RCS -> **CVS** -> SVN

- CVS was implemented as an extension of the RCS functionality, which is why it uses the same history format (plik.v)
- It allows multiple users to work on one file
- Supports creating branches
- Features a mechanism for resolving conflicts

Limitations:

- Metadata is not versioned
- When working on multiple files – approving changes is not a separate operation
- It does not support renaming files in the repository

Version control systems

Functionalities of version control systems:

RCS -> CVS -> **SVN**

- Features a history of directory and file name changes, file locations in directories and properties of files and directories
- Changes in several files/folders are separate operations – in case of breaking a network connection – changes are not saved unless all changes are saved
- The functionality of user authentication and authorization, data compression, using the HTTP protocol
- Additional server (regardless of HTTP) – e.g. as a separate daemon
- Branches are defined as copies, which occupy little space in the permanent storage
- Size of data being transferred depends on changes made and not the size of the file

Hardware emulators

Emulator – computer program executed in a single computer system and duplicating the operation of another computer system.

Emulator parts:

- CPU Emulator – the most complicated part of the program
- Memory emulator
- Input output devices emulator

The main problem of emulators is the operating speed:

- Adjusting emulation speed to the required minimal speed of operating applications
- High-frequency processors emulation made in a computer system running at a slightly greater frequency

Hardware emulators

GDB Emulator – text emulator reading binary files

- monitoring and modifying internal variables
- ability to call functions regardless of the programs
- remote mode
- reversible debugging
- GUI – Xxgdb, KDbg, GDBtk
- compatible with Dev-C++, Qt Creator, Lazarus, Eclipse, VisualStudio

Hardware emulators

QEMU Emulator

- Running multiple operating systems
- Is fast
- Supports multiple CPU architectures
- Two operating modes: user and system
- Network card emulation
- Handling snapshots
- Works with a VNC (possibility of remote operation)
- Virtual server
- Supports USB tablets
- Dedicated for Linux/UNIX systems – partial support for MS Windows
- No accelerators such as VMware or VirtualBox

Project management tools

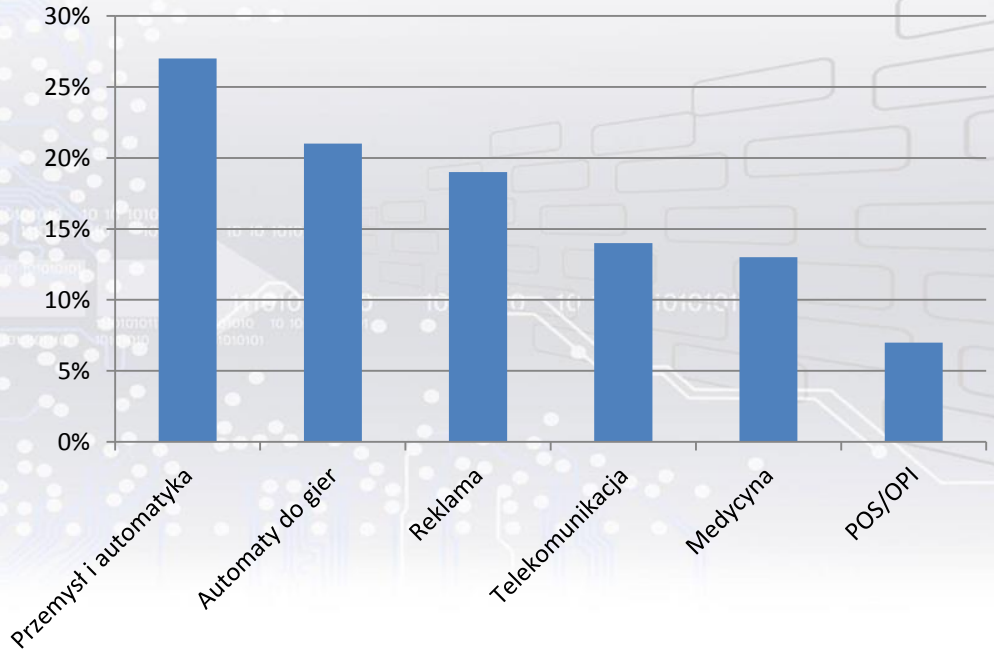
Project management tools on the basis of Redmine

Redmine – *free and open source tool used for managing multiple projects*

- Calendar, charts, repository, diff viewer, wikis, forums
- Visual representations
- Compatible with version control systems (SVN, CVS, Git etc.)
- Role-based access control
- E-mail notifications
- Fields for issues, time-entries, projects, users
- Multiple databases

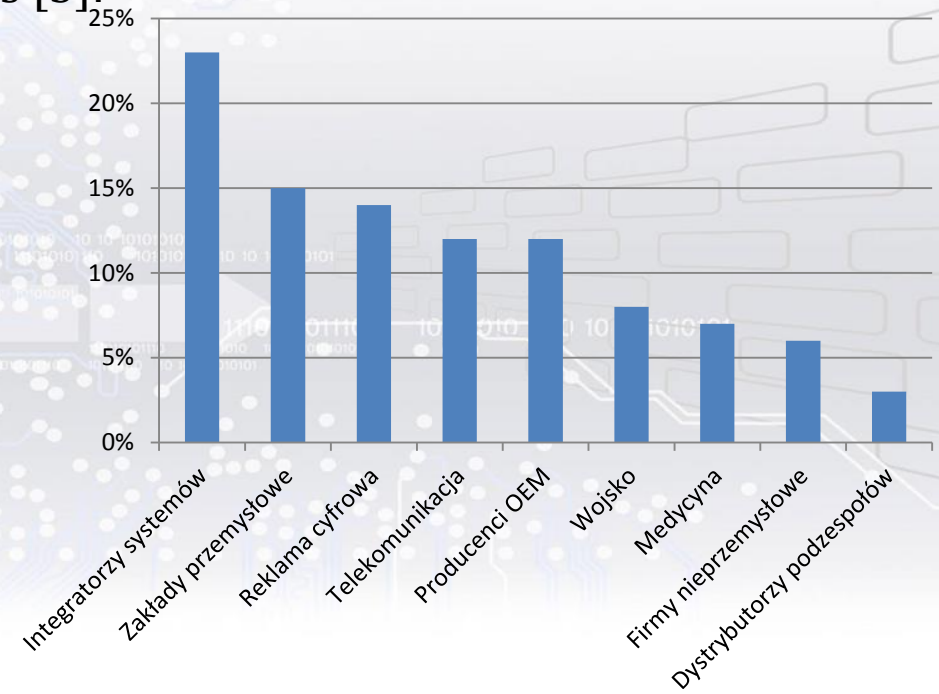
ES market in Poland

The most important applications using a single board computers in Poland [5]:



ES market in Poland

Overview of industries which are the largest customers of computers for embedded systems [5]:



Bibliography

[1] https://en.wikipedia.org/wiki/Version_control

[2] https://en.wikipedia.org/wiki/Redmine

[3] https://en.wikipedia.org/wiki/GNU_Debugger

[4] https://en.wikipedia.org/wiki/QEMU

[5] http://elektronikab2b.pl/raporty/9585-polski-rynek-komputerow-do-systemow-embedded#.V22Q7_mLSM8