# General Informations

## Exam Admission (Attestation)

Recap 🡪 Summary of a topic from previous week, Tips 🡪 Tips and Tricks for the next students, Functionality 🡪 Defined Robot + Remote Task (Snake Game in FS2017)

## Lab Material

**tinyK20** 🡪 Programmer/Debug Probe, USB & SWD Cable, used to program robot and remote, debugging interface

**K22 Zumo Robot** 🡪 V1 (2014) and V2 (2016), **K22FX512 (ARM Cortex-M4F**), 120 MHz, 512Kbyte of FLASH, 64 Kbyte of RAM, USB, 2 LED’s, 1 Buzzer, Reset + user button, 1:75 DC Motors, **Optical (V1) or Magnetic (V2) Quadrature Encoder**, IR Line Sensor, Arduino Headers

**K20 Remote** 🡪 included tinyK20, **K20DX128** (**ARM Cortex-M4)**, 50 MHz, 128 Kbyte FLASH, 16 Kbyte RAM, nRF24L01+ 2.4 GHz Transceiver, Nokia 64x48 BW **LCD**, **7 Buttons** (Joystick Buttons (4way + center), 2 side buttons), 260 mA LiPo Battery, ICharging: 195mA

# Build and Debug

## Eclipse Workspace

- Where Ecplise stores the ‘meta data’ (folder .metadata) 🡪 1. **‘global’ options** across projects, 2. **NEVER move/share meta data**

- Wsp (Workspace) could contain project folders, but it’s recommended to keep wsp and projects separate because it’s easier for the VCS (Version Control System) **Example: wsp** (C:\user\wsp\_kds) and **projects** (C:\user\projects)

|  |  |
| --- | --- |
| Workflow **make**  🡪 .mk file 🡪 Beinhaltet Regeln  für Abhängigkeiten zwischen den Files | Parallel Build Reducing Compile Time in Eclipse with Parallel Build. This is especially useful for host machines having multiple cores or CPU: such as each CPU then could do a compilation and balance the build load across all available CPUs to cut the build time. |
| Sharing Debug Configuration Normally the debug configurations are not stored in the project settings. If I zip that project or share it with a vcs, then the debug configurations are not shared.  **To share debug configurations** with my project, I need to enable ‘Shared file’ in the configuration: that way the configuration gets stored in a .launch file inside the project. |
| PE Code Generation In the processor expert project options you can say “don’t’ generate code before build automatically” to speed up PE projects |
| Debug without Build Uncheck the Option “Build (if required) before launching” to debug without build. This speeding up the debug/launch. |

# Version Control System (VCS)

## A good VCS

**Backup and Restore** (wiederherstellen), **Synchronization, Short- and Long-term undo, Track changes and ownership** (wer hat was wieso geändert), **Sandbox** (You can make temporary changes in an isolated area, test and work out the kinks before “checking in” your changes), **Branching and Merging** (A larger sandbox. You can **branch** (verzweigen) a copy of your code from the main trunk into a separate area and modify it in isolation (tracking changes separately). Later, you can **merge** (vereinigen, zusammenführen) your work back into the common area (main trunk).)

## Typical VCS

- Server(s) with data base(s)

- Clients connect to server

-locally or remote

- Single Server or distributed

|  |  |
| --- | --- |
| **‘Optimistic’ approach (e.g. GitHub)** | **‘Pessimistic’ approach** |
| Assumes rarely conflicts  Different clients can work on the same file concurrently | avoid conflics  Just one client can work on a file |
| **Centralized** | **Distributed** |
| typically single repository server, data is just on server, **local copy of current snapshot on client,** commit/compare when connected, Example: CVS, **SVN** (SubVersion) | Repository server (can be multiple), Data is on server and on client, **Local copy contains full repository history**, commit/compare even when not connected, Commits to local repo 🡪 then sync with server, Example: **GIT** |

## What to share and what not to share with VCS

|  |  |
| --- | --- |
| **Share:** 🡪 everything needed to build the project  - **.project** and **.cproject** (project files and build options  - source files (**\*.c,\*.h, etc.)** and source file folders  - **Project\_Settings** folder and files (linker files, startup files)  - **\*.launch** (contains launch/debugger settings)  - **ProcessorExpert.pe** (contains component settings) | **Not share:** 🡪 generated or derived (abgeleitete) resources  - generated documentation/log files  - Build output files **\*.o** (named as the build target)  - PE 🡪 **Documentation and Generated\_Code**  - **.ProcessorExpert.g\_c** and **.ProcessorExpert.g\_x** (contain information about the generated files) |

# Git

|  |  |  |
| --- | --- | --- |
| About Git - By Linus Torvalds  - **Index:** collection of added and file changes  - Staged and unstaged files can co-exist    (commit and ‘check in’ is the same) | Typical Git Workflow | Other Actions - **Discard/Revert**  🡪undo a local change  - **Delete**  🡪 deletes a file from the index/disk  **- Tag**  🡪 Mark files with a label, e.g. to check out all files with the same label |
| Ignoring files - .gitignore File ignores files and folders, wich shouldn’t be shared  - **Recommendation:** one .gitignore File per project  - **NOTE:** path in .gitignore File is relative to ignore file location | .gitignore Format - #: starts comment  - line item: ignore file/folder/pattern |  |

# Systems & Realtime

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Transforming Systems - Data processing quality  - Troughput (Durchsatz)  - Optimized system load  - Optimized Memory Usage | Reactive Systems - External events are driving system  - guaranteed response time  - Control loop,  - Realtime | | | Realtime - System Interaction with the environment  - System has to deal with the time constraints (Grenzen) of the real world (real time)  Not 🡪 as fast as possible  Instead 🡪 at the right time  **Realtime System Requirements:**  Correctness and External time conditions compliance (Zustimmung, Einhaltung)  **Examples:** Train system schedule computation (Abfahrtsplan),Railroad switch |
| Interactive Systems - short response time  - High system load  - Human-Machine Interaction (HMI) | | |
| Realtime for Computer Systems A computer is classified as Realtime if it can react on external events in the real world:  - With the correct result  - At the correct time  - Independent of current system load  - In a deterministic (foreseeable) way  **Claims:** Timeliness and Concurrency | | Timeliness (Rechtzeitigkeit) For all processing stages: **Input 🡪 Process 🡪 Output**  Categories: **absolute and relative** | Concurrency (Nebenläufigkeit) - Real world is concurrent 🡪 Problem: Computers are sequential | |

|  |  |  |
| --- | --- | --- |
| Reaction Time Realtime Systems require a defined absolute or relative reaction time  - **Interactive Systems** 🡪 seconds  - **Transforming and Reactive Systems** 🡪 milli- or microseconds  **System load defined with**  - Number of concurrent events/tasks  - Interval of events  - Reaction time for events  - Processing time for events | Hard Realtime - Incorrect if correct Result does not meet time conditions 🡪 outside the blue marked area, the data is | Soft Realtime Degradation, if correct result does not meet the time conditions 🡪 outside the blue area still ok, but nut very good |