Entwicklungsumgebung

Echtzeitsysteme 2 – Vorlesung/Übung

Peter Ulbrich
Fabian Scheler
Wolfgang Schröder-Preikschat

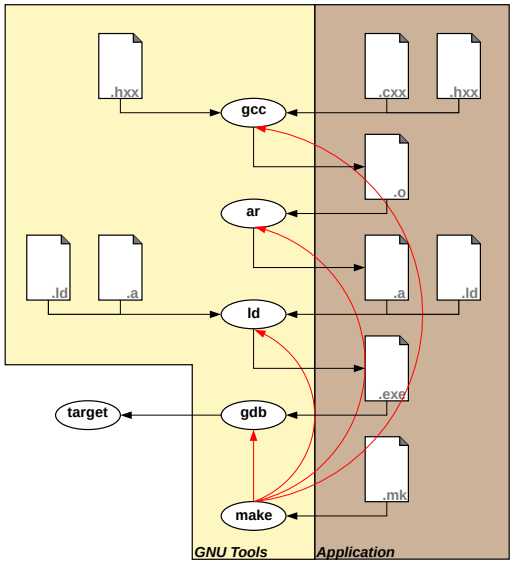
Lehrstuhl für Informatik 4
Verteilte Systeme und Betriebssysteme
Friedrich-Alexander Universität Erlangen-Nürnberg

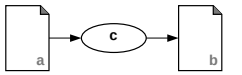
http://www4.cs.fau.de/~{scheler,ulbrich,wosch}
{ulbrich,scheler,wosch}@cs.fau.de

Übersicht

- GNU Tools
- ProOSEK
- nxtOSEK
- SMC
- Absint aiT

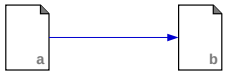
Entwicklungsumgebung – Überblick





Werkzeug c liest Datei a und schreibt Datei b

Werkzeug a kontrolliert/aktiviert Werkzeug b



Datei a bindet Datei b ein

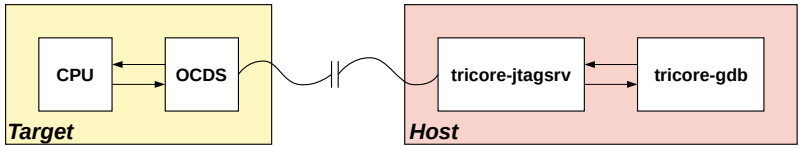
GNU Tools – GCC & LD

- weitere Infos:
  - Info-Seite GCC: `info gcc`
  - Info-Seite LD: `info ld`
  - TriCore-GCC User's Guide: `/proj/i4ezs/docs/compiler/gnutricore/usersguide.pdf`
- spezielle Flags für TriCore

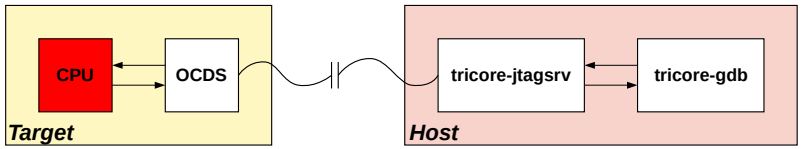
`-meabi` `-mcpu=tc1796`	Auswahl der TriCore-Architektur – es existieren verschiedene Instruktionssätze
`-mcpu8` `-mcpu10`	Workarounds für Silicon-Bugs

GNU Tools – GDB

- weitere Infos:
  - Info-Seite GDB: `info gdb`
  - TriCore-GCC User's Guide: `/proj/i4ezs/docs/compiler/gnutricore/usersguide.pdf`
- Funktionsweise

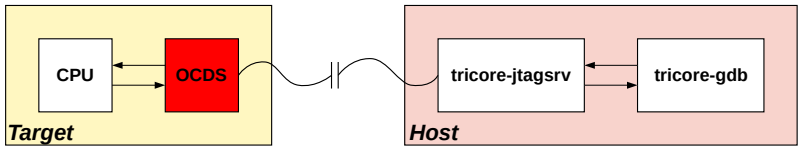


GNU Tools – GDB



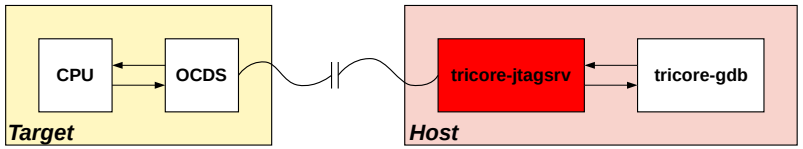
- CPU
  - TriCore CPU Core
  - Peripheral Control Processor (PCP)
  - DMA

GNU Tools – GDB



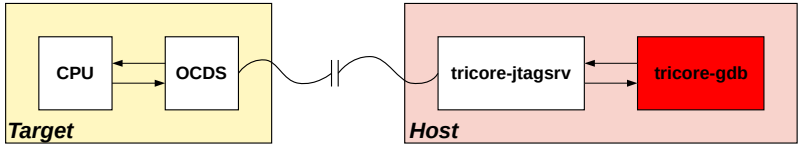
- On-Chip Debug Support
  - Level 1 – low-cost debugging
  - Level 2 – tracing (CPU, PCP, DMA)
  - Level 3 – TC1796 emulation device
  - HW(≤ 4)/SW Breakpoints (program/data)
  - RW memory + registers
  - besteht aus
    - OCDS System Control Unit (OSCU)
    - JTAG Debug Interface (JDI)
    - Multi Core Break Switch (MCBS)

GNU Tools – GDB



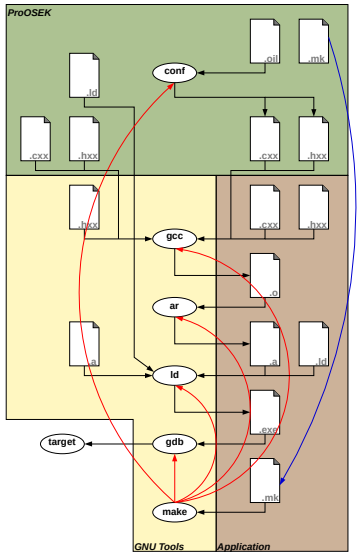
- tricore-jtagsrv
  - Proxy zwischen JTAG und GDB
  - Target ↔ tricore-jtagsrv: JTAG via Parallelport
  - tricore-jtagsrv ↔ GDB: TCP/IP
  - Aufruf: `tricore-jtagsrv -i <init_file> <port>`
    - Programm: `/proj/i4ezs/tools/gnutricore/bin`
    - Initialisierungsdaten: `/proj/i4ezs/tools/gnutricore/tricore/jtag_targets`

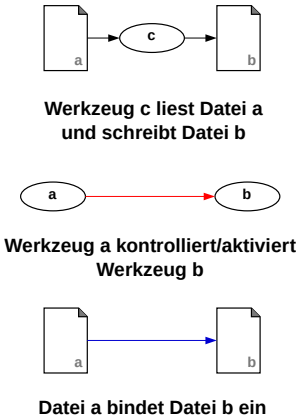
GNU Tools – GDB



- tricore-gdb
  - gdb für TriCore
  - Verbindung zum tricore-jtagsrv:
    `target tricore-remote <port>`

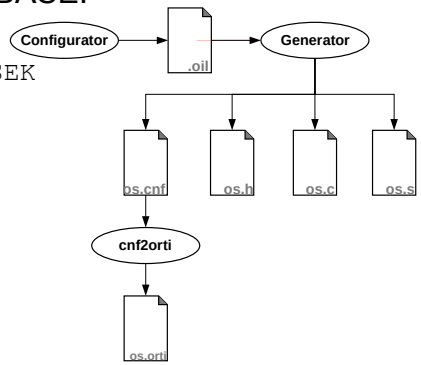
ProOSEK – Überblick





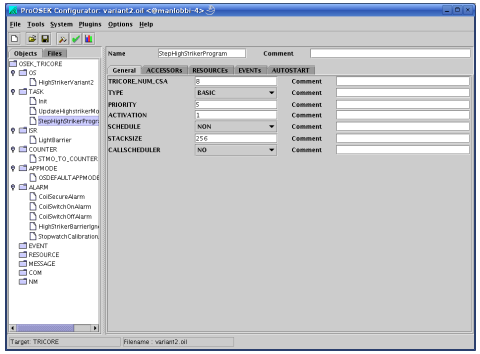
ProOSEK

- Implementierung des OSEK/OSEKtime Standards
  - Internet: `www.osek-vdx.org`
  - OSEK OS 2.2.3: `/proj/i4ezs/docs/os/os223.pdf`
  - Time-Triggered OS 1.0: `/proj/i4ezs/docs/os/ttos10.pdf`
- Umgebungsvariable OSEK_BASE:
  - CIP-Pool: `/local/proosek`
  - Labornetz: `/usr/local/ProOSEK`
- Workflow



ProOSEK – Configurator

- grafisches Konfigurationswerkzeug
- Ergebnis: Systemkonfiguration (OIL-Datei)
- Aufruf:
  - CIP-Pool: `/local/proosek/bin/proosek`
  - Labornetz: `/usr/local/ProOSEK/bin/conf`



ProOSEK – Generator

- Übersetzer: OIL-Datei → Implementierung
- Ergebnis: Header- und Implementierungsdateien
  - os.h: Schnittstelle des OSEK-Laufzeitsystems
    - OSEK-API
    - konfigurationsspezifische Konstanten ...
  - os.c, os.s: Implementierung des OSEK-Laufzeitsystems
    - Vektortabelle
    - Kontextwechsel ...
  - os.cnf: Beschreibung der Konfiguration
    - Stacks ...
- Aufruf:
  - CIP-Pool: `/local/proosek/bin/proosek -o <dir> <oil>`
  - Labornetz: `/usr/local/ProOSEK/bin/conf -o <dir> <oil>`

ProOSEK – cnf2orti

- Übersetzer: os.cnf → os.orti
- OSEK runtime information (ORTI)
  - OSEK-Unterstützung für den Debugger
  - Welcher Task läuft gerade?
  - Welche Zustände haben die Tasks gerade?
  - Welche Alarme sind aktiv?
  - Welche Ressourcen sind gerade belegt?
- wird von gängigen Debuggern ausgewertet
  - Lauterbach Trace32
  - Hitex Tanto
  - leider nicht: GDB
- Aufruf:
  - CIP-Pool: `/local/proosek/bin/proosek orti <cnf> <orti>`
  - Labornetz: `/usr/local/ProOSEK/bin/cnf2orti <cnf> <oil>`

make

- Info: `info make`
- ProOSEK stellt bereits make-Dateifragmente zur Verfügung:

```
...
include $(OSEK_BASE)/make/host.Linux
...
```

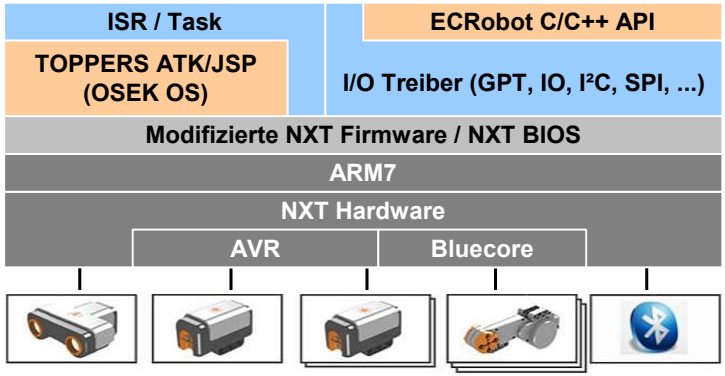
make – erforderliche Variablen

- folgende Variablen müssen definiert sein:

Variable	Beschreibung	Wert
OSEK_BASE	Installationsverzeichnis von ProOSEK	/local/proosek
BUILD_DIR	Verzeichnis, in dem alle generierten Dateien gespeichert werden	
OIL_FILE	OIL-Datei, die die Systemkonfiguration enthält	
FILE	Name des Binärabbilds, das erzeugt wird	
DEBUG	Übersetzerparameter, der die Erzeugung von Debug-Information veranlasst	-g
CPU	Welche CPU wird verwendet?	TRICORE
BOARD	Welches Board wird verwendet?	TriBoardTC1796
TOOL	Welche Toolchain wird verwendet?	gnu
TOOLPATH	Das Basisverzeichnis der Toolchain	/proj/i4ezs/tools/gnutricore
CC_INCLUDE_USER	Verzeichnis, das benutzerdefinierte Header enthält	
OBJS_USER	Benutzerdefinierte Übersetzungseinheiten	
CC_OPT_USER	Benutzerdefinierte Übersetzerparameter	
LINK_OPT_USER	Benutzerdefinierte Binderparameter	-gc-sections
LIBDIRS	Verzeichnisse mit benutzerdefinierten Bibliotheken	
LIBS_USER	Benutzerdefinierte Bibliotheken	

nxtOSEK - Übersicht

- Open Source OSEK Implementierung für den NXT
  - http://lejos-osek.sourceforge.net
  - TOPPERS/ATK OSEK API und TOPPERS/JSP Kern
  - ECRobot C/C++ API für leJOS NXJ Treiber



NXT Firmware-Anpassung

- Drei Möglichkeiten für nxtOSEK Anwendungen:

1. **John Hansen's Enhanced NXT-Firmware**
   - Beliebig viele Anwendungen im NXT-Flash
   - Größenbeschränkung pro Anwendung auf 64KB
2. NXT BIOS
   - Nur eine Anwendung im NXT-Flash
   - Maximale Größe 224KB
3. Direktes Booten aus dem RAM
   - Eine Anwendung im NXT-RAM (flüchtig!)
   - Maximale Größe 64KB, kein (langsamer) Flash Zugriff

→ NeXTTool ermöglich flashen und Firmware-Update

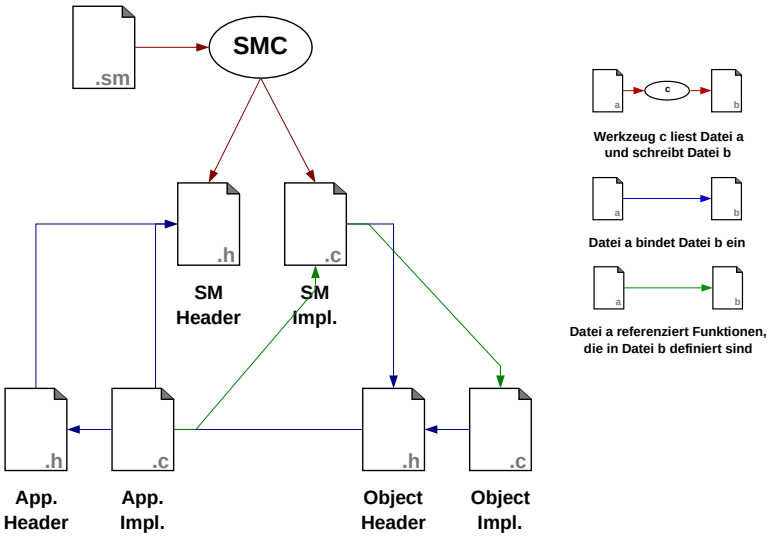
nxtOSEK – Werkzeugkette

- Verfügbar für Windows und **Linux**
- GNU MAKE
  - cygwin unter Windows
- OSEK OIL parser/code generator (Windows Binary)
  - Mittels Wine unter Linux
- GNU ARM Crosscompiler
- NeXTTool
  - Hochladen von Anwendung und Firmware per USB

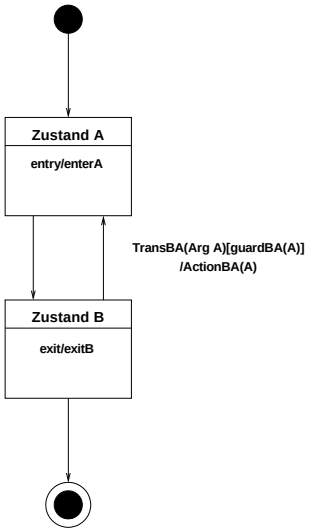
SMC – State Machine Compiler

- Übersetzer: Zustandsautomaten → Quellcode
  - mögliche Programmiersprachen: Java, C++, **C**, Lua, Perl, **Dot**, ...
- orientiert sich an UML Statecharts
- Zustandsautomat modelliert Verhalten eines Objekts
- Kopplung zwischen Anwendung und Zustandsautomat
  - Methoden bzw. Funktionsaufrufe
  - Anwendung → Transitionen des Zustandsautomaten
  - Zustandsautomat → Aktionen der Anwendung

SMC - Workflow



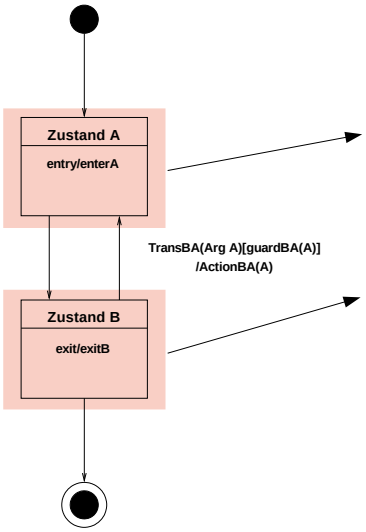
SMC - Syntax




```
%class TheObject
%header TheObject.h

%start SM::ZustandA
%map SM
%%
ZustandA
        Entry {
                enterA();
        }
{
        ...
}
ZustandB
        Exit {
                exitB();
        }
{
        TransBA(A: Arg)
                [guardBA(A)]
                ZustandA {
                        ActionBA(A);
                }
}
%%
```

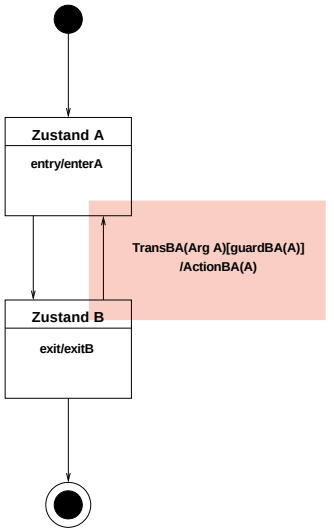

SMC - Syntax




```
%class TheObject
%header TheObject.h

%start SM::ZustandA
%map SM
%%
ZustandA
        Entry {
                enterA();
        }
{
        ...
}
ZustandB
        Exit {
                exitB();
        }
{
        TransBA(A: Arg)
                [guardBA(A)]
                ZustandA {
                        ActionBA(A);
                }
}
%%
```

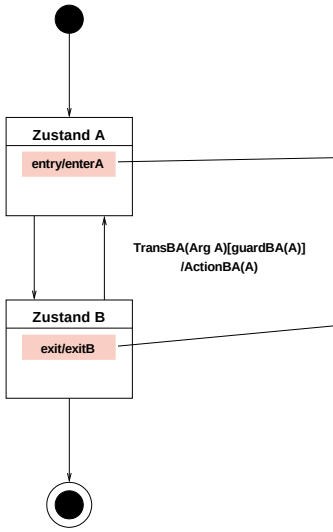

SMC - Syntax




```
%class TheObject
%header TheObject.h

%start SM::ZustandA
%map SM
%%
ZustandA
        Entry {
                enterA();
        }
{
        ...
}
ZustandB
        Exit {
                exitB();
        }
{
        TransBA(A: Arg)
                [guardBA(A)]
                ZustandA {
                        ActionBA(A);
                }
}
%%
```

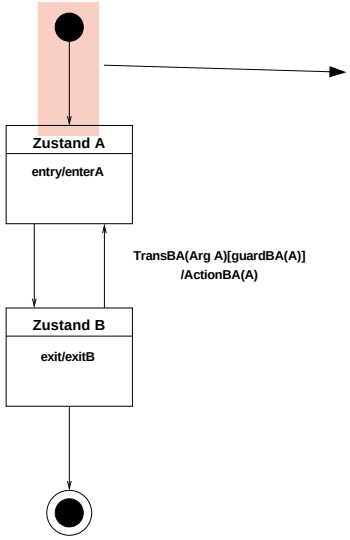
SMC - Syntax




```
%class TheObject
%header TheObject.h

%start SM::ZustandA
%map SM
%%
ZustandA
        Entry {
                enterA();
        }
{
        ...
}
ZustandB
        Exit {
                exitB();
        }
{
        TransBA(A: Arg)
                [guardBA(A)]
                ZustandA {
                        ActionBA(A);
                }
}
%%
```


SMC - Syntax




```
%class TheObject
%header TheObject.h

%start SM::ZustandA
%map SM
%%
ZustandA
        Entry {
                enterA();
        }
{
        ...
}
ZustandB
        Exit {
                exitB();
        }
{
        TransBA(A: Arg)
                [guardBA(A)]
                ZustandA {
                        ActionBA(A);
                }
}
%%
```


SMC – Programmierschnittstelle

- SMC modelliert das Verhalten von Objekten
  - objekt-basiertes Programmieren in C

SMC – Programmierschnittstelle

- SMC modelliert das Verhalten von Objekten
  - objekt-basiertes Programmieren in C

```
#include <TheObject.h>

struct TheObject theObject;
struct SMContext _fsm;

void TheObject_enterA(struct TheO
  ...
}
void TheObject_exitB(struct TheOb
  ...
}
void TheObject_ActionBA(struct TheObject *obj,Arg A) {
  ...
}

int guardBA(Arg a) {
  ...
}

...
```

Definition des Objekts

```
#include <SM_sm.h>
struct TheObject { ... };

void TheObject_enterA(...);
void TheObject_exitB(...);
void TheObject_ActionBA(...);

int guardBA(Arg a);
```

SMC – Programmierschnittstelle

- SMC modelliert das Verhalten von Objekten
  - objekt-basiertes Programmieren in C

```
#include <TheObject.h>

struct TheObject theObject;
struct SMContext _fsm;

void TheObject_enterA(struct TheObject *obj) {
  ...
}
void TheObject_exitB(struct TheObject *obj) {
  ...
}
void TheObject_ActionBA(struct TheObject *obj,Arg A) {
  ...
}

int guardBA(Arg a) {
  ...
}

...
```

das Objekt

SMC – Programmierschnittstelle

- SMC modelliert das Verhalten von Objekten
  - objekt-basiertes Programmieren in C

```
#include <TheObject.h>

struct TheObject theObject;
struct SMContext _fsm;

void TheObject_enterA(struct TheObject *obj) {
  ...
}
void TheObject_exitB(struct TheObject *obj) {
  ...
}
void TheObject_ActionBA(struct TheObject *obj,Arg A) {
  ...
}

int guardBA(Arg a) {
  ...
}

...
```

der Zustandsautomat

SMC – Programmierschnittstelle

- SMC modelliert das Verhalten von Objekten
  - objekt-basiertes Programmieren in C

```
#include <TheObject.h>

struct TheObject theObject;
struct SMContext _fsm;

void TheObject_enterA(struct TheObject *obj) {
  ...
}
void TheObject_exitB(struct TheObject *obj) {
  ...
}
void TheObject_ActionBA(struct TheObject *obj,Arg A) {
  ...
}

int guardBA(Arg a) {
  ...
}

...
```

Implementierung des Aktionen
- Operationen auf dem Objekt
- this-Zeiger wird explizit übergeben

SMC – Programmierschnittstelle

- SMC modelliert das Verhalten von Objekten
  - objekt-basiertes Programmieren in C

```
#include <TheObject.h>

struct TheObject theObject;
struct SMContext _fsm;

void TheObject_enterA(struct TheObject *obj) {
  ...
}
void TheObject_exitB(struct TheObject *obj) {
  ...
}
void TheObject_ActionBA(struct TheObject *obj,Arg A) {
  ...
}

int guardBA(Arg a) {
  ...
}

...
```

Guards sind globale Funktionen
- Rückgabewert ist ein Wahrheitswert

SMC – Verwendung

- SMC modelliert das Verhalten von Objekten
  - → objekt-basiertes Programmieren in C

```
#include <SM.h>

...

int main() {
  Arg B;

  /* initialisieren */
  SMContext_Init(&_fsm,&theObject);

  /* Transitionen aktivieren */
  SMContext_TransBA(&_fsm,B);

  return 0;
}
```

SMC – Advanced

- Default-Transitionen und -Zustände
- Verschachtelte Zustandsautomaten
  - → Abbildung auf push()/pop()
- alles weitere auf: `http://smc.sourceforge.net`
- Installation: `/proj/i4ezs/tools/smc`

WCET-Analyse: Absint aiT

- Statische Bestimmung der max. Ausführungszeit
- Pfad: `/local/ait_tricore`
- Dokumentation: `$Pfad/share/doc/ait_tricore`
- Verwendung: erstaunlich einfach ☺
  - aiT starten: `/local/ait_tricore/bin/aittricore`
  - Programm auswählen
    - Programm muss **vollständig gebunden** sein
    - Programm sollte im **internen Speicher** liegen
  - **Annotationen für Schleifengrenzen** hinzufügen
  - → Analyse mit `Ctrl-A` starten
- Beispiel: Annotationen für Schleifengrenzen

```
clock exactly 150 MHz;
compiler "tricore-gcc";

loop STOPWATCH_Init +1 loops max 3 begin;
```

Aufgabe

- Inbetriebnahme der Hardware
  - Laden/Ausführen/Debuggen von Programmen
  - Ausgabe eines Signal über einen I/O-Pin
- Unter Verwendung
  - der bloßen GNU Toolchain
  - ProOSEK / nxtOSEK
- Bestimmung der WCETs
  - für ausgewählte Funktionen/Prozeduren