Entwicklungsumgebung

Echtzeitsysteme 2 – Vorlesung/Übung

Fabian Scheler
Michael Stilkerich
Wolfgang Schröder-Preikschat

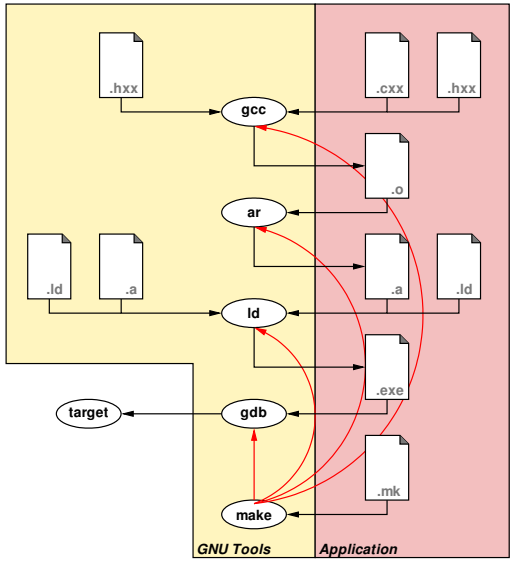
Lehrstuhl für Informatik IV
Verteilte Systeme und Betriebssysteme
Friedrich-Alexander Universität Erlangen-Nürnberg

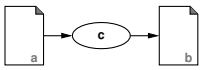
http://www4.cs.fau.de/~{scheler,mike,wosch}
{scheler,mike,wosch}@cs.fau.de

Übersicht

- GNU Tools
- ProOSEK
- KESO
- SMC

Entwicklungsumgebung - Überblick





Werkzeug c liest Datei a und schreibt Datei b

Werkzeug a kontrolliert/aktiviert Werkzeug b

Datei a bindet Datei b ein

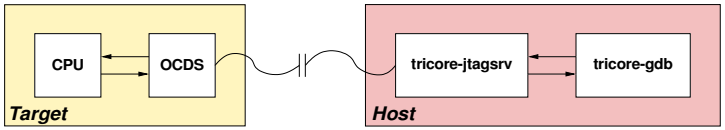
GNU Tools – GCC & LD

- 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

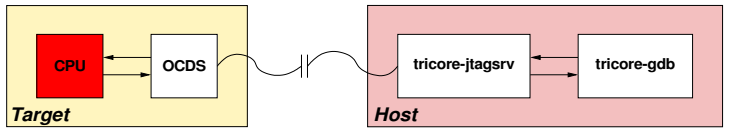
`-meabi` `-mcpu=tc1796`	Auswahl der TriCore-Architektur – es existieren verschiedene Instruktionssätze
`-mcpu8` `-mcpu10`	Workarounds für Silicon-Bugs

GNU Tools – GDB

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

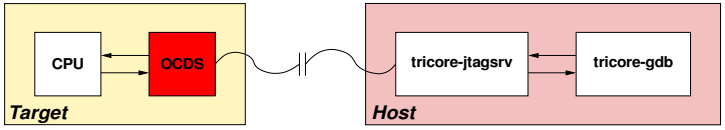


GNU Tools – GDB



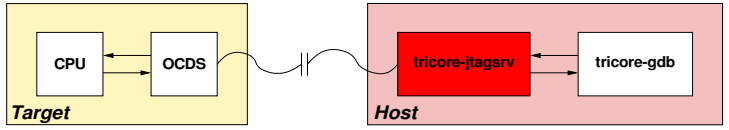
- CPU
  - TriCore CPU Core
  - Peripheral Control Processor (PCP)
  - DMA

GNU Tools – GDB



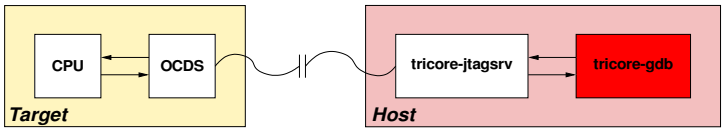
- On-Chip Debug Support
  - Level 1 – low-cost debugging
  - Level 2 – tracing (CPU, PCP, DMA)
  - Level 3 – TC1796 emulation device
  - HW(≤ 4)/SW Breakpoints (program/data)
  - RW memory + registers
  - besteht aus
    - OCDS System Control Unit (OSCU)
    - JTAG Debug Interface (JDI)
    - Multi Core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

GNU Tools – GDB



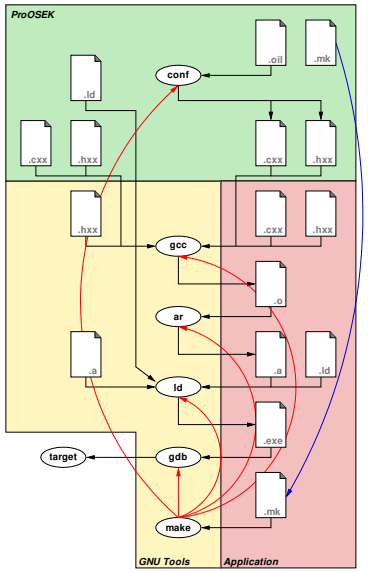
- tricore-jtagsrv
  - Proxy zwischen JTAG und GDB
  - Target ↔ tricore-jtagsrv: JTAG via Parallelport
  - tricore-jtagsrv ↔ GDB: TCP/IP
  - Aufruf: `tricore-jtagsrv -i <init_file> <port>`
    - Programm: `/proj/i4ezs/tools/gnutricore/bin`
    - Initialisierungsdaten: `/proj/i4ezs/tools/gnutricore/tricore/jtag_targets`

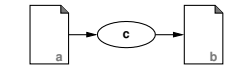
GNU Tools – GDB



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

Entwicklungsumgebung - Überblick





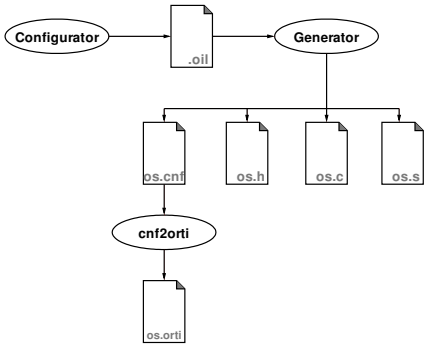
Werkzeug c liest Datei a und schreibt Datei b

Werkzeug a kontrolliert/aktiviert Werkzeug b

Datei a bindet Datei b ein

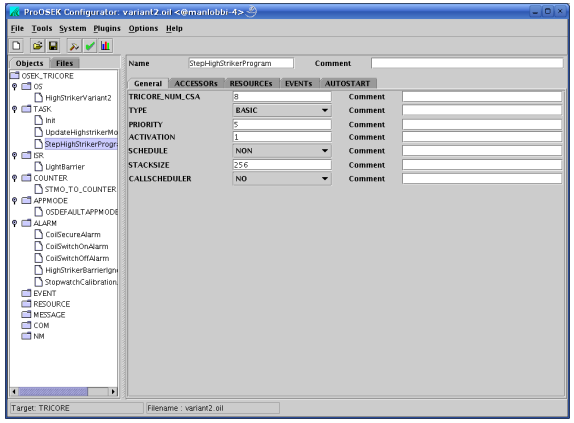
ProOSEK

- 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: `/local/proosek`
- Workflow



ProOSEK – Configurator

- grafisches Konfigurationswerkzeug
- Ergebnis: Systemkonfiguration (OIL-Datei)
- Aufruf: `/local/proosek/bin/proosek`



ProOSEK – Generator

- Übersetzer: OIL-Datei → Implementierung
- Ergebnis: System von Header- und Implementierungs-dateien von ProOSEK
  - os.h: Schnittstelle des OSEK-Laufzeitsystems
    - OSEK-API
    - konfigurationsspezifische Konstanten ...
  - os.c: Implementierung des OSEK-Laufzeitsystems
    - OSEK-API ...
  - os.s: Implementierung des OSEK-Laufzeitsystems
    - Vektortabelle
    - Kontextwechsel ...
  - os.cnf: Beschreibung der Konfiguration
    - Stacks ...
- Aufruf:

```
/local/proosek/bin/proosek -o <dir> <oil>
```


ProOSEK – cnf2orti

- Übersetzer: os.cnf → os.orti
- OSEK runtime information (ORTI)
  - OSEK-Unterstüzung für den Debugger
  - Welcher Task läuft gerade?
  - Welche Zustände haben die Tasks gerade?
  - Welche Alarme sind aktiv?
  - Welche Resource sind gerade belegt?
- wird von gängigen Debuggern ausgewertet
  - Lauterbach Trace32
  - Hitex Tanto
  - leider nicht: GDB
- Aufruf:

```
/local/proosek/bin/proosek orti <cnf> <orti>
```


make

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

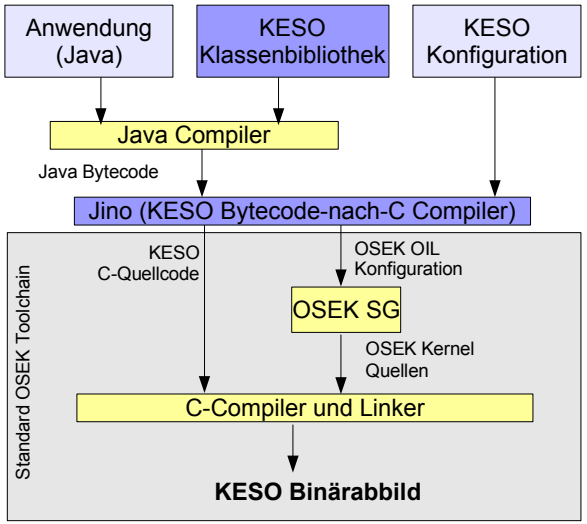
```
...
#include $(OSEK_BASE)/make/host.Linux
...
```


make – erforderliche Variablen

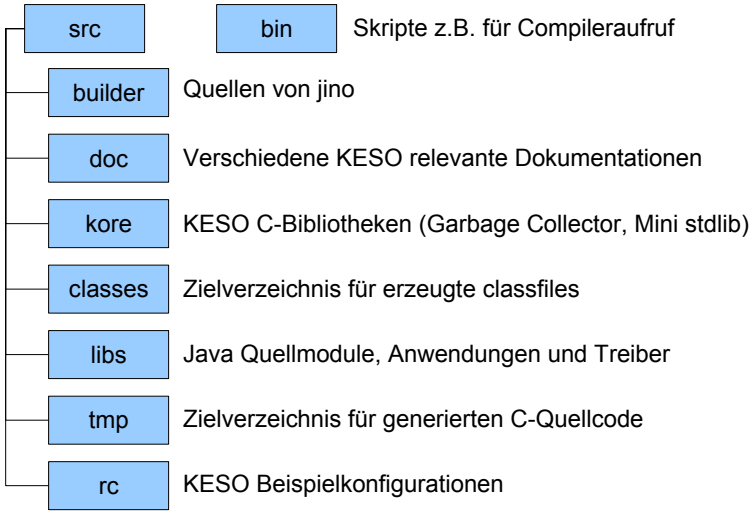
- folgende Variablen müssen definiert sein:

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, die benutzerdefinierte Bibliotheken enthalten	
LIBS_USER	Benutzerdefinierte Bibliotheken	

KESO Toolchain



KESO Verzeichnisstruktur



KESO - Umgebungsvariablen

- folgende Variablen müssen definiert sein:

Variable	Beschreibung	Wert
KESOROOTPATH	Wurzel des KESO Baumes	
JINOFLAGS	Flags für Jino (optional)	
KESOSRCPATH	Pfad zum src Verzeichnis im KESO Baum	${KESOROOTPATH}/src
JDK	Installationsverzeichnis des Java DK >= 1.4	
JDKTOOLS	Pfad zur tools.jar des JDK	${JDK}/lib/tools.jar
KESORC	Pfad zur Konfigurationsdatei, relativ zu ${KESOSRCPATH}	

- Setzen z.B. durch Aufruf (in ${KESOROOTPATH}) von
  `source bin/setup.bash`
- KESORC muss in jedem Fall manuell gesetzt werden

KESO Beispielübersetzungslauf

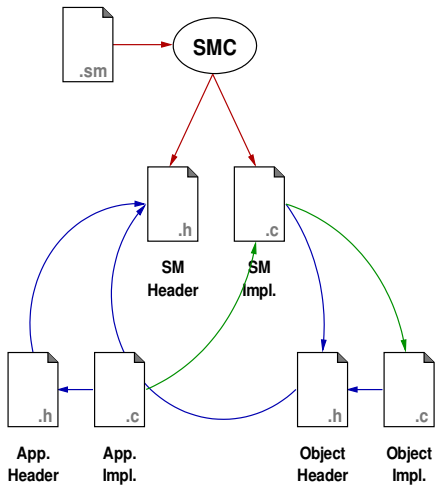
- Übersetzung der bestehenden Robertino Anwendung

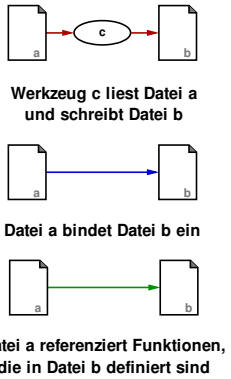
```
$ ls
bin    josek  siwihaas     src
$ source bin/setup.bash
$ cd src
$ export KESORC=rc/kesorc.mdsa
$ make
( ... Jino output ... )
$ cd tmp && ls
Keso_drive0  Keso_drive1  Keso_drive2  ( ... )
$ cd Keso_drive0 && make
( ... Make output ... )
keso.elf  :
section      size      addr
.text        6924         0
.data         222   8388704
.bss           61   8388926
Total       59069
```

SMC – State Machine Compiler

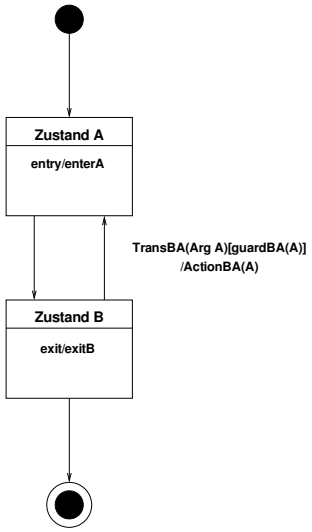
- übersetzt eine textuelle Beschreibung eines Zustandsautomaten in Quellcode
  - mögliche Programmiersprachen: Java, C++, **C**, Lua, Perl, **Dot**, ...
- orientiert sich an UML Statecharts
- Zustandsautomat modelliert Verhalten eines Objekts
- Kopplung zwischen Anwendung und Zustandsautomat
  - Methoden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

SMC - Workflow





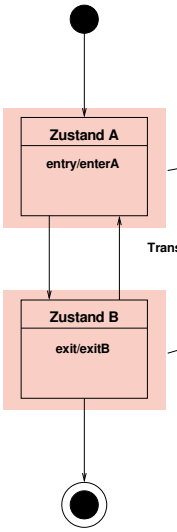
SMC - Syntax




```
%class TheObject
%header TheObject.h

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

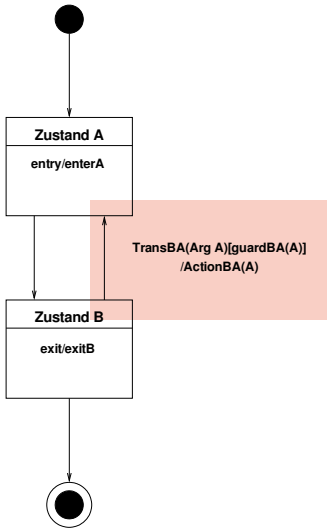

SMC - Syntax




```
%class TheObject
%header TheObject.h

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

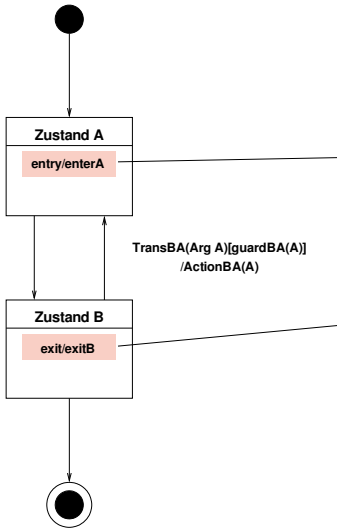
SMC - Syntax




```
%class TheObject
%header TheObject.h

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

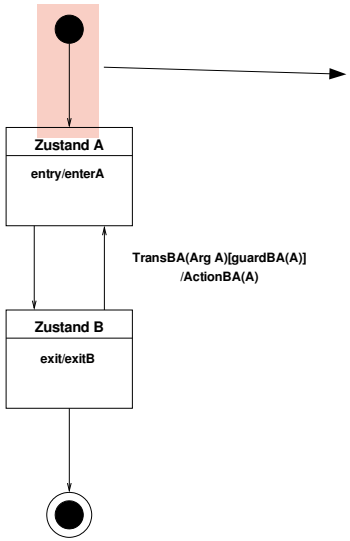

SMC - Syntax




```
%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`

Aufgabe

- Implementierung eines binären Zählers
  - zyklisches Hochzählen
  - benutzergesteuertes Hochzählen
- Unter Verwendung
  - der blossen GNU Toolchain
  - ProOSEK
- Ausgabe des Zählerstands über LEDs
- LEDs liegen an den Pins 0.5 – 0.12
- Taster liegt an Pin 1.0