Projektarbeit im SS 2007

Software-Entwicklungswerkzeuge für einen konfigurierbaren Prozessor

Prof. Dr. Gundolf Kiefer

www.fh-augsburg.de/~kiefer/projektarbeit

Agenda

1. Die Aufgabe
2. Vorschläge zur Realisierung
3. Organisatorisches
4. Nächste Schritte

1. Die Aufgabe

- **Die Firma *HEkonPro** entwickelt einen neuen Prozessor für konfigurierbare Systeme.**

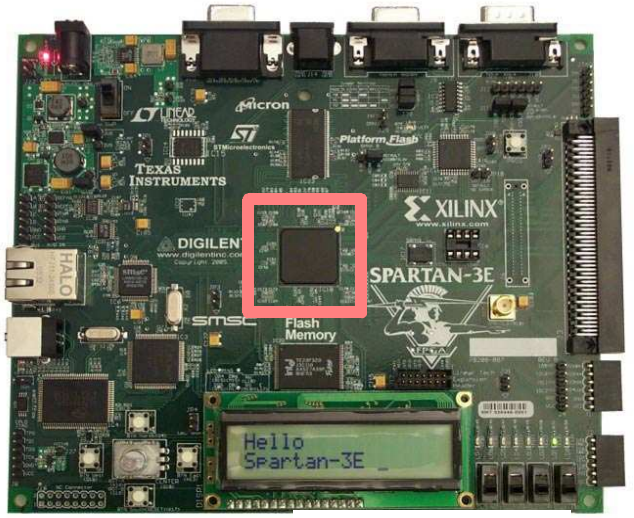


Bild: Xilinx/Digilent

- **Für die Erstellung der Software-Entwicklungs-Tools wird die Firma *SEkonPro** *(das sind Sie!)* beauftragt:**

  a) Assembler (Compiler)

  b) Simulator

  c) Grafische Oberfläche

**** Name frei erfunden***

Der Prozessor

- **Allgemeine Daten**
  - RISC-Architektur
  - 8 Register: r0, ..., r7
  - Wortbreite 16-Bit

- **Befehlssatz & -codierung**

Operation	Opcode	Bedeutung
ADD	00 000 ddd sss ttt --	Rddd <- Rsss + Rttt
SUB	00 001 ddd sss ttt --	Rddd <- Rsss - Rttt
SAL	00 010 ddd sss ttt --	Rddd <- Rsss << 1
SAR	00 011 ddd sss ttt --	Rddd <- Rsss >> 1
AND	00 100 ddd sss ttt --	Rddd <- Rsss & Rttt
OR	00 101 ddd sss ttt --	Rddd <- Rsss | Rttt
XOR	00 110 ddd sss ttt --	Rddd <- Rsss ^ Rttt
NOT	00 111 ddd sss ttt --	Rddd <- ~Rsss
LDIL	01 -00 ddd nnn nnn nn	Rddd[7:0] <- nnnnnnnn
LDIH	01 -01 ddd nnn nnn nn	Rddd[15:8] <- nnnnnnnn
LD	01 -10 ddd sss --- --	Rddd <- Mem[Rsss]
ST	01 -11 --- sss ttt --	Mem[Rsss] <- Rttt
JMP	10 000 --- sss --- --	PC <- Rsss
HALT	10 -01 --- --- --- --	Prozessor hält an
JZ	10 -10 --- sss ttt --	if (Rttt == 0) PC <- Rss
JNZ	10 -11 --- sss ttt --	if (Rttt != 0) PC <- Rss

a) Assembler: Beispiel-Programm

```
; Berechnung der ersten 8 Fibonacci-Zahlen

        .org 0x100              ; alles folgende ab Adresse 100
const:  .data 0, 1, 'a', 'bc', -5, 1000 ; jedes Element belegt 16 Bit
label:  .res 8                  ; 8 Worte reservieren

start:  ldil r1, 1              ; r1 := 1
        ldih r1, 0
        ldil r2, label & 255    ; r2 := label (Adresse)
        ldih r2, label >> 16
        ldil r3, 8              ; r3 := 8 (Schleifenzähler)
        ldih r3, 0
        ldil r4, loop & 255     ; r4 := loop (Sprungadresse)
        ldih r4, loop >> 16
        ldil r5, 0              ; r5 := 0 = fib(0)
        ldih r5, 0
        add  r6, r0, r1         ; r6 := 1 = fib(1)
loop:   st   [r2], r5
        add  r7, r5, r6         ; r7 := r5 + r6 = fib(n) + fib(n+1)
        add  r5, r6, r0         ; r5 := r6
        add  r6, r7, r0         ; r6 := r7
        sub  r3, r3, r1         ; r3 := r3 – 1, Schleifenzähler erniedrigen
        jnz  r3, r4             ; Sprung nach 'loop', falls r3 != 0
        halt                    ; Prozessor anhalten / Simulation stoppen

        .end                    ; Ende des Programmes
```

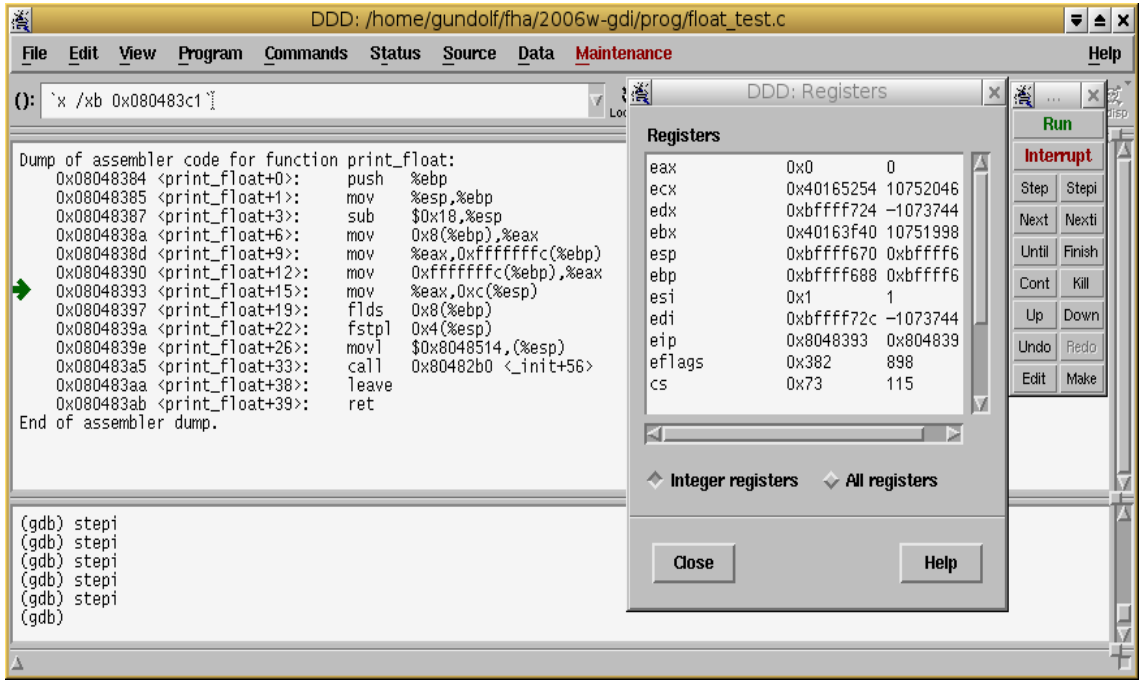
b) Simulator: Beispiel-Lauf

```
> load fibonacci.out
> mem 0x100-0x10f
0x0100: 0000 0001 ...
0x0108: ...
> break loop
Breakpoint set at 0x011a
> go start
Stopped at PC=0x11a
> step
Stopped at PC=0x11b
> registers
Registers: r0=0x0000 r1=0x0001 r2=...
> go
HALT instruction reached at 0x120
> stats
Simulated 36 instructions, 132 clock cycles
> quit
Bye!
```

c) Grafische Oberfläche (GUI)

- **Separate Anwendung**
  - ruft Kommandozeilen-Simulator (=Backend) im Hintergrund auf
- **Anzeigebereiche für**
  - Programmcode / Disassembler-Listing
  - Eingabeaufforderung

    -> Möglichkeit, beliebige Simulator-Kommandos einzugeben
  - Prozessor-Register
  - Speicher-Auszug
- **Menüpunkte / Knöpfe (nur) für die wichtigsten Kommandos**



Anforderungen des Kunden

- **Der Kunde will:**
  - **Hohe Code-Qualität**
    - stabil, sauber programmiert, klar strukturiert
    - portierbar, möglichst wenige Abhängigkeiten von externen Bibliotheken
    - benutzerfreundlich (insbesondere in Fehlersituationen!)
    - modularer Aufbau
  - **Möglichkeit für spätere Erweiterungen**
    - neue Prozessorbefehle
    - Simulation von Peripherie
    - Alternatives Prozessormodelle, z.B.
      - taktgenaue Simulation für Pipeline-/Superskalar-/...-Strukturen
      - Ansteuerung eines realen Prozessors (On-Chip-Debugging)
  - **Gute Dokumentation**

2. Vorschläge zur Realisierung

a) Assembler

- **Tool 'flex'**
  - Werkzeug zur Bau eines *Scanners*, der einen Programmtext in Tokenstrom zu übersetzt, z. B.

    "loop: ldih r2, mydat >> 16"

    -> *<Label: "loop"> <':'> <Opcode: OP_LDIH> <Reg: 2> <','> <Label: "mydat"> <'>>'> <Num: 16>*

  - Weitere Information: *'info flex'*
- **Tool 'bison'**
  - Werkzeug zum Bau eines *Parsers*, der einen Tokenstrom interpretiert und weiter verarbeitet.

Parser-Generator "*bison*"

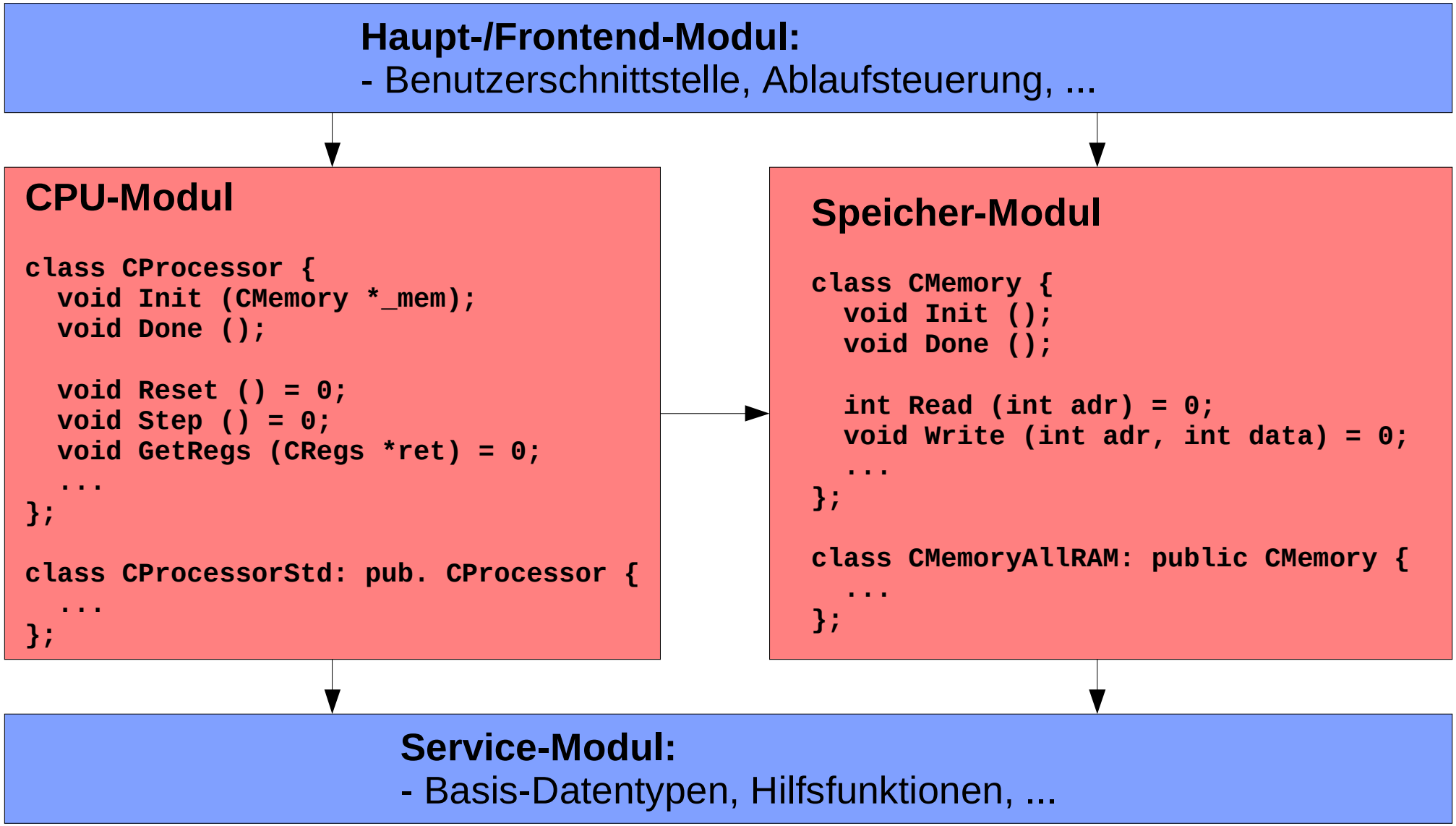
Beispiel: Rechner für arithmetische Ausdrücke

```
[...]

/* Bison declarations.  */
%token NUM
%left '-' '+'
%left '*' '/'

%% /* The grammar follows.  */
input:    /* empty */
        | input line
;
line:     '\n'
        | exp '\n'  { printf ("\t%.10g\n", $1); }
;
exp:      NUM                { $$ = $1;         }
        | exp '+' exp        { $$ = $1 + $3;    }
        | exp '-' exp        { $$ = $1 - $3;    }
        | exp '*' exp        { $$ = $1 * $3;    }
        | exp '/' exp        { $$ = $1 / $3;    }
        | '(' exp ')'        { $$ = $2;         }
;
%%
```

b) Simulator


Haupt-/Frontend-Modul:
- Benutzerschnittstelle, Ablaufsteuerung, ...
CPU-Modul
class CProcessor {
void Init (CMemory *_mem);
void Done ();
void Reset () = 0;
void Step () = 0;
void GetRegs (CRegs *ret) = 0;
...
};
class CProcessorStd: pub. CProcessor {
...
};
Speicher-Modul
class CMemory {
void Init ();
void Done ();
int Read (int adr) = 0;
void Write (int adr, int data) = 0;
...
};
class CMemoryAllRAM: public CMemory {
...
};
Service-Modul:
- Basis-Datentypen, Hilfsfunktionen, ...

c) Simulator-GUI

Tcl/Tk:

- Skript-Sprache: erlaubt das Erstellen von grafischen Anwendungen mit wenigen Zeilen Code, keine Compilation notwendig

- Beispiel
  - Code (Datei "hello.tcl"):

```
button .hello -text "Hello, world" -command {
    puts stdout "Hello, world"; destroy .
}
pack .hello
```

  - Starten: *wish hello.tcl*

- Weitere Informationen
  - http://www.tcl.tk/
  - http://www.tcl.tk/man/tcl8.4/
  - *'wish /usr/share/doc/tk8.4/examples/widget'*

Kommunikation mit dem Backend über Pipes

- Starten des Simuators als externer Prozess, dann Umleiten der Standard-Ein-/Ausgabe

Beispiel (asynchron):

- Erzeugen der Pipe und Starten des Simulators:

```
set backPipe [open "|sim_backend -i" r+]
fconfigure $backPipe -blocking no
fileevent $backPipe readable "HandleLine $backPipe"
```

- Absetzen von Kommandos:

```
puts $::backPipe "$cmd"
flush $::backPipe
```

- Reagieren auf Ausgaben des Backends

```
proc HandleLine backPipe {
    set chars [gets $::backPipe line]
    if { $chars < 0 } {
        catch { close $backPipe }
        ... # Fehlerbehandlung
    } else {
    ... # '$line' verarbeiten
}
```

3. Organisatorisches

3.1. Termine & Räume

- **Regelmäßige Treffen: Mo, 15:40 Uhr** in J101 (voraussichtl.)

- **Arbeiten: G209**
  - Suse-Linux
  - Installation von zusätzl. Software/Bibliotheken: F. Schöppler

- **Nächstes Treffen: 19. 3. '07**

3.2. Rollen im Team

Das Team

- organisiert sich selbst
- bestimmt **Projektleiter;** dieser vertritt das Team nach ...
  - ... innen:
    - Koordination der Teilprojekte
    - Zeitplan eingehalten? Falls nicht, Vereinbarungen treffen (Actions)
    - evtl. Helfen bei Engpässen
  - ... aussen:
    - Ansprechpartner für den Kunden

Der Betreuer (Gundolf Kiefer)**:**

a) Kunde

b) Berater

nicht: Projektleiter o.ä.

Aufgabenverteilung

- **Teilprojekte**
  - Assembler (ca. 2 Personen)
  - Simulator (ca. 2-3 Personen)
  - Simulator-Oberfläche (ca. 1-2 Personen)

- **Spezielle Zuständigkeiten, z.B.:**
  - **Projektleiter**
  - **Sommerfest-Organisator**
  - Tester
  - PR-Manager (z.B. Koordination der Projektpräsentation, Projekt-Webseite)
  - IT-Verantwortlicher (z.B. Installation von Software, Wartung SVN-Server)
  - **...**

3.3. Leistungsnachweis

- **Am Semesterende abzuliefern:**
  - **CD mit**
    - Quellcode
    - Programmbeispielen
    - Dokumentation
      - Installation, Übersetzung, Systemvoraussetzungen
      - Benutzerhandbuch
      - Programmierschnittstellen (APIs)
      - Dateiformate (z.B. Objekt-Dateien)
  - **Abschlusspräsentation**
    - *Zeit für Vorbereitung einplanen!*

4. Nächste Schritte

Bis zum nächsten Treffen:

a) Verteilung der Rollen/Aufgaben, Wahl des Projektleiters

b) Einarbeitung

- Welche Sprache(n)/Tools/Bibliotheken werden verwendet? (Vorschläge übernehmen oder eigene machen & begründen?)
- Einarbeitung in Sprache(n)/Tools/Bibliotheken

c) Pflichtenheft

- Welche Features sollen Ihre Tools unterstützen, welche nicht?
- Wo gibt es evtl. Probleme mit der Realisierung?

d) Zeitplan aufstellen

- Arbeitsschritt ≙ 1-2 Wochen
- Ergebnisorientiert (Was ist wann fertig?)
  z.B. "12.4.: Modul XY implementiert und getestet"

e) Zu a) – d) Kurzpräsentationen vorbereiten

*-> Agenda per E-Mail bis **1.3.** an <gundolf.kiefer@fh-augsburg.de>*

Einarbeitung & Pflichtenheft

Einarbeitung

Assembler-Team:

- Theorie
  - Franz Josef Schmitt: "Praxis des Compilerbaus", Hanser Verlag 1992, Kapitel 3-6
- Tools: flex, bison (yacc)

Simulator-Team:

- DLXsim als Beispiel
  - Download: *ftp://max.stanford.edu/pub/max/pub/hennessy-patterson.software/dlx.tar.Z*
  - Report: *http://heather.cs.ucdavis.edu/~matloff/DLX/Report.html*

GUI-Team:

- TclTk (s.o.)

Pflichtenheft

- Genaue Syntax, z.B. unterstützte Direktiven
- Inhalt der Objektdateien
- ...

- Benutzer-Kommandos
- APIs
- Inhalt der Objektdateien
- ...

- Aufbau Oberfläche, Menüs
- Schnittstelle zum Simulator
- evtl. Erweiterungsschnittstellen
- ...

Nächstes Treffen

- **Kurzvorträge (je ca. 5 Minuten)**
  1. Tools zum Compilerbau: flex und bison
  2. Simulation auf Befehlsebene
  3. Grafische Oberflächen mit Tcl/Tk
  4. Vorstellung des Projektteams (Rollen!) und des Zeitplans
  5. Leistungsumfang des *SEkonPro*-Assemblers
  6. Leistungsumfang des *SEkonPro*-Simulators
  7. Leistungsumfang des *SEkonPro*-Simulator-Frontends

- **Ziel: "Verkaufsgespräch"**
  - Kunden überzeugen, dass Sie die notwendige ***Kompetenz*** haben und ein ***hochwertiges*** Produkt ***pünktlich*** liefern können.