

Programmierung und Modellierung fehlertoleranter Echtzeitsysteme

Sommersemester 2008

- Prof. Dr.-Ing. habil. Armin Zimmermann
- Daniel Lüdtké
EN 251, 314-25324, dluedtké@cs.tu-berlin.de

Vorlesung: Do 12-14, MA041
Di 12-14, HL001 06.05., 12.05., 19.05., 27.05.

Übung: Nur zu Schwerpunkten im Praktikum
Di 12-14, HL001 – 22.04., 08.07.
Do 12-14, MA041 – 08.05., 05.06.

Rechnerzeiten: EN258, EN266 (Türcode: **2569**)
Di 14-18, betreut: 15-17 Uhr
Beginn: heute

Unterlagen: Verkauf in der ersten Rechnerzeit
1€ ADA-Unterlagen (wie in EES)
alles andere auf der Web-Seite

Anrechnung: alle Studiengänge 4 SWS, TNA

Weitere Informationen, kurzfristige Änderungen:
<http://pdv.cs.tu-berlin.de/PMFE-SS2008/>

Großübungen

- zu Schwerpunkten im Praktikum

22. 04.	Eisenbahn-API (Aufgabe 1 und 3)
08. 05.	TimeNET und Petri-Netze (Aufgabe 2)
05. 06.	Leistungsbewertung mit TimeNET und Ausfallüberwachung (Aufgaben 4 und 5)
08. 07.	Fragestunde zur Klausur

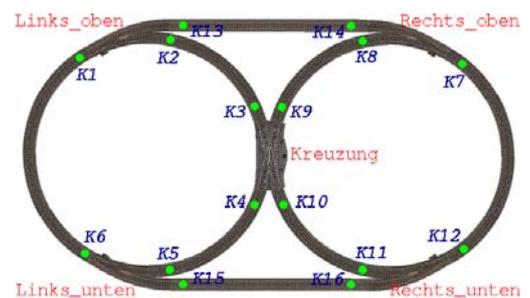
Klausur

- Mittwoch, 22. Juli 09-13 Uhr im EB 301
- Alle Unterlagen zugelassen (Kofferklausur)
- Voraussetzung:
alle Übungsaufgaben erfolgreich bearbeitet

Praktikum

- A ① **Messung der Lok-Fahrzeiten** (Ada)
Einarbeitung in Eisenbahn und API
Abgabetermin: 06. Mai 2008
 - A ② **Modellierung und Verifikation** (TimeNET)
Petri-Netze, Modell der Zugsteuerung
Abgabetermin: 20. Mai 2008
 - A ③ **Steuerprogramm** (Ada)
Fehlertolerante Steuerung von zwei Loks
Abgabetermin: 10. Juni 2008
 - A ④ **Leistungsbewertung** (TimeNET)
Stochastische Modellierung, optimale Parameter
Abgabetermin: 24. Juni 2008
 - A ⑤ **Redundante Steuerung** (Ada)
Ausfalltoleranz, hot standby
Abgabetermin: 08. Juli 2008
- erste Rechnerübung
 - verspätete Anmeldungen
 - Wechsel in andere Gruppe abstimmen
 - Verkauf der Unterlagen
 - Praktikumsräume
Entwicklungsrechner: EN258, EN259, EN266
Eisenbahnanlage: EN266a

Eisenbahnanlage

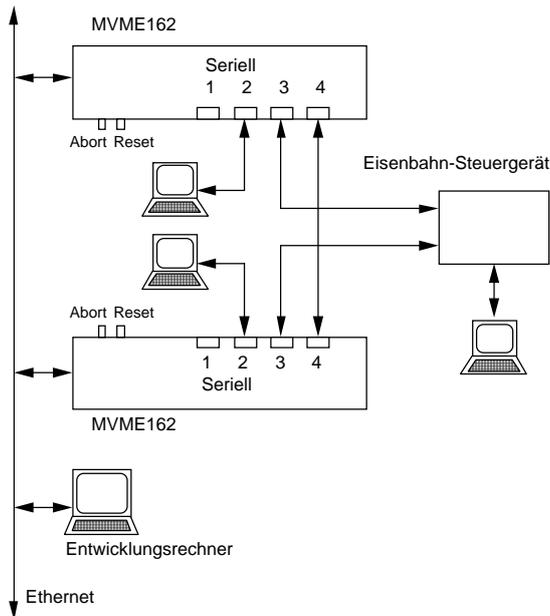


- Märklin Digital, Steuerung über Märklin-Interface
- Reed-Kontakte K1..K16 an Rückmeldemodul, Magnete an Loks

Arbeitsumgebung

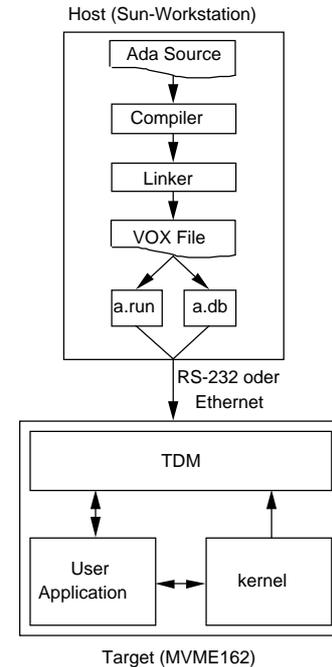
- Entwicklungsrechner SunRay
- Echtzeitrechner gold/zink, kupfer/zinn
 - MVME 162LX-263,
Prozessor Motorola 68040, 16 MB RAM, VME-Bus,
Schnittstellen: seriell, parallel, SCSI, Ethernet

Hardwarekonfiguration



Softwarekonfiguration

Ada-Cross-Compiler von VADS
(Verdix Ada Development System)



Auf dem Echtzeitrechner laufen:

- **etdm**
 - Ethernet Turbo Debug Monitor
 - verbindet Sun und MVME162 über Ethernet
 - wird nach dem Einschalten und nach Reset automatisch über Ethernet geladen und gestartet (Bootserver bolero)
- **krm.vox**
 - Laufzeitkern für Ada-Benutzerprogramme im Verdix Objects Executable (vox)-Format
 - wird beim Laden des Benutzerprogramms mittels **start** automatisch als erstes auf MVME162 geladen
 - wird nicht gestartet
- **bahn.vox** (Bsp.)
 - ausführbares Ada-Benutzerprogramm
 - wird mittels **start** geladen und gestartet

Softwareentwicklung

- alle Kommandos der Ada-Entwicklungsumgebung in `/home/pdv/pdv/vads/bin/` (unabhängig von Lehrveranstaltung)
- alle PMfE betreffenden Dateien stehen in `/home/pdv/pdv/lehre/pmfe/...`
- LV-spezifische Kommandos: **init**, **makelib**, **start**
- zu Beginn jeder Sitzung Umgebung initialisieren: `source /home/pdv/pdv/lehre/pmfe/init` (in jedem X-Terminal, automatisieren; für bash)
Achtung: nicht mit alten Einstellungen oder denen von Robotik vermischen, immer klar umschalten!
- zur Verfügung gestellte Pakete sind in der Ada-Library im Unterverzeichnis: **eisenbahn/pmfelib**
- Verzeichnis zur Programmentwicklung anlegen, z.B. : `~/pmfe/a1/`
in dieses Verzeichnis wechseln, dort **makelib** (nur auf bolero!) ausführen
⇒ Ada-Library wird erzeugt (jedes Verzeichnis kann in eine Ada-Library gewandelt werden)
- Ada-Programm schreiben
 - Namenskonvention: **name_s.a** : specification
name_b.a : body

- Pakete übersetzen: (richtige Reihnefolge!)
ada_name_s.a
ada_name_b.a
- Programm binden:
a.ld <Name Hauptprogramm> -o <Programmname>
Bsp: a.ld bahn_main -o bahn.vox
- wer möchte, benutzt a.make (s. Manual)
- Hilfe zu allen Kommandos:
 - a.help
 - Manual “Die Ada-Umgebung”
 - /home/pdv/pdv/vads/man in MANPATH-Variable aufnehmen
- start <Rechnername> <ausführbares Programm>
Shell-Script für Mehrbenutzerbetrieb im Praktikum
 - Vorprogramm erwartet Bestätigung durch Eingabe des Benutzernamens
 - nach Bestätigung lädt start Benutzerprogramm bahn.vox und startet dieses
 - startdr startet Programm gleichzeitig auf beiden Rechnern (Aufgabe 5)
 - queues zeigt Zustand der Warteschlangen an

Hinweise

- alle Ausgaben über Prozeduren von Text_IO gelangen zur Sun und werden in dem X-Terminal dargestellt, in dem start aufgerufen wurde
- Rechner nicht länger als 3 Minuten belegen
- Programmende
 - normales Ende:
Ausgabe auf der Sun (normally)
 - Ende mit Exception:
Ausgabe auf der Sun (Name der Exception)
 - Absturz: (kommt nicht vor!?)
Reset an MVME162 (reboot)

Vorgegebene Pakete

- In der Ada-Library
/home/pdv/pdv/lehre/pmfe/eisenbahn/pmfelib
befinden sich bereits folgende Pakete:
 - SERIAL_IO: Basisroutinen zur Ein- und Ausgabe über die 4 seriellen Schnittstellen
 - SCREEN: spezielle Routinen zur Ausgabe auf dem VT100-Terminals an der 2. seriellen Schnittstelle
 - Bahn_Defs: vordefinierte Typen für Züge, Weichen und Kontakte
 - Eisenbahn_Io: Schnittstelle zur Prozessperipherie, Routinen zur Kommunikation mit dem Eisenbahn-Steuergerät auf logischer Ebene
 - Terminal_Io: spezialisierte Ein- und Ausgaberroutinen für Benutzungsoberfläche der Eisenbahnsteuerung

Vorgegebenes Paket Bahn_Defs:

```

package Bahn_Defs is
  subtype T_Lok_Nr is Byte range 1..80;
    Loknummern von 1 bis 80 sind möglich.
  subtype T_Geschwindigkeit is Byte
    range 0..14;
    0: Halt, 1: langsam, 14: schnell
  type T_Weiche is (Links_Oben,
    Links_Unten, Kreuzung,
    Rechts_Oben, Rechts_Unten);
    Die Weichen sind so bezeichnet, wie sie von einem
    Betrachter vor der Anlage gesehen werden.
  type T_Richtung is (Gerade, Rund);
    Jede Weiche hat zwei mögliche Stellungen:
    Gerade: Nicht abbiegen bzw. Kreuzung überqueren,
    Rund: Abbiegen bzw. Kreuzung nicht überqueren
  type T_Kontakt is (K1, K2, K3, K4, K5,
    K6, K7, K8, K9, K10, K11, K12,
    K13, K14, K15, K16);
    Entspricht den auf der Anlage angegebenen
    Kontaktbezeichnungen.
  type T_Kontaktfeld is
    array (T_Kontakt) of Boolean;
    Jedes Kontaktfeld ist 2 Bytes groß
    (1 Bit/Kontakt = 2 Bytes)
end Bahn_Defs;
```

Vorgegebenes Paket **Eisenbahn_Io**:

```

package Eisenbahn_IO is
procedure Initialisieren;
    Initialisiert das Steuergerät und setzt alle 80
    möglichen Lokgeschwindigkeiten auf Null, sowie alle
    Weichen auf gerade. Dauert ca. 10 Sekunden!
procedure Initialisieren
    (Lok1, Lok2: in T_Lok_Nr);
    Initialisiert die Eisenbahnanlage. Die
    Fahrtgeschwindigkeiten der angegebenen Loks
    werden auf Null (Halt) gesetzt und alle Weichen
    werden gerade gestellt.
    Mögliche Werte für Lok1/2: 1 - 80
procedure Nothalt;
    Stoppt mit sofortiger Wirkung alle Züge;
    Wiederaufnahme des Fahrbetriebs ist nur durch
    erneutes Initialisieren möglich.
procedure Lok_Steuern (Lok: in T_Lok_Nr;
    Geschwindigkeit: in T_Geschwindigkeit);
    Setzt die Fahrtgeschwindigkeit der angegebenen
    Lok. Mögliche Werte für Lok 1 – 80
    Mögliche Werte für Geschwindigkeit: 0 (Halt), 1
    (langsam) - 14 (schnell)
procedure Weiche_Schalten
    (Nr: in T_Weiche;
    Richtung: in T_Richtung);

```

*Schaltet eine Weiche in die angegebene Richtung.
 Mögliche Werte für Nr: Links_Oben, ...
 Mögliche Werte für Richtung: Gerade, Rund*

```

procedure Kontaktstatus_abfragen
    (Aktiv : in Boolean;
    Kontakte: out T_Kontaktfeld;
    Empfangen: out Boolean);
    Liefert die seit dem letzten Auslesen überfahrenen
    Kontakte zurück. Mit Aktiv auf "true" wird der
    Auslesevorgang initiiert, danach blockiert die
    Funktion so lange, bis Kontaktaten empfangen
    wurden. Empfangen ist danach immer "true". Mit
    Aktiv auf "false" wird nichtblockierend auf den
    ankommenden Kontaktstatus gewartet. Empfangen
    gibt dabei an, ob Kontaktaten gelesen wurden. Der
    passive Rechner muss die Funktion mit Aktiv auf
    "false" aufrufen.
procedure Zeit_holen(Zeit : out Time);
    Gibt die aktuelle Systemzeit zurück.
procedure Schreiben (B :in Byte);
    Sendet ein Byte an den anderen Echtzeitrechner.
procedure Lesen (B : out Byte;
    Empfangen : out Boolean);
    Liest ein Byte nichtblockierend vom anderen
    Echtzeitrechner. Empfangen gibt dabei an, ob ein
    Byte gelesen wurden.
end Eisenbahn_IO;

```

Vorgegebenes Paket **Terminal_Io**:

- Ausgabe des aktuellen Zugstatus auf dem Terminal
- ASCII-„Grafik“ bereits vorgegeben

```

package Terminal_Io is
procedure Anhalten (Lok: T_Lok_Nr;
    Kontakt: T_Kontakt);
    Stellt eine angehaltene Lok auf dem gegebenen
    Kontakt dar. Lok: 1 – 80, Kontakt: K1 - K16
procedure Fahren (Lok: T_Lok_Nr;
    Kontakt: T_Kontakt);
    Stellt eine fahrende Lok auf dem gegebenen
    Kontakt dar. Lok: 1 – 80, Kontakt: K1 - K16
procedure Stellen (Nr: T_Weiche;
    Richtung: T_Richtung);
    Stellt die Richtung einer Weiche dar. Mögliche Werte
    für Nr: Links_Oben, Links_Unten, Kreuzung,
    Rechts_Oben, Rechts_Unten. Mögliche Werte für
    Richtung: Gerade, Rund
procedure Ein_Zeichen_Einlesen
    (Zeichen: in out CHARACTER);
    Gibt Prompt "Eingabe:" aus und liest ein Zeichen von
    der Tastatur.
procedure Ein_Zeichen_Nichtblock_Einlesen
    (Zeichen: in out CHARACTER;
    Empfangen : Boolean);

```

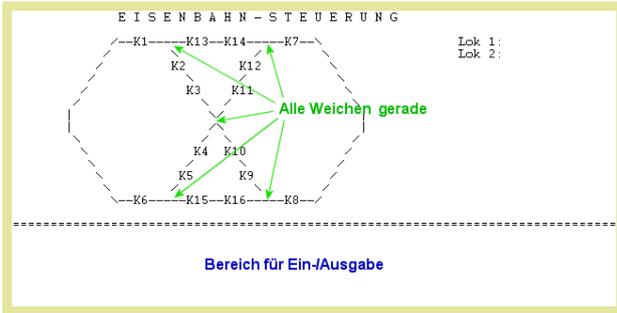
*Liest ein Zeichen nichtblockierend von der Tastatur.
 Wenn ein Zeichen gelesen wurde, liefert Empfangen
 true zurück.*

```

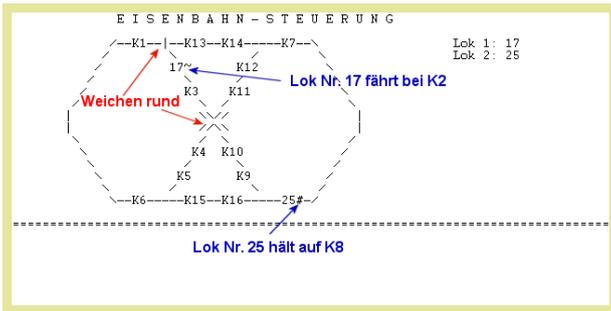
procedure Lok_Nr_Einlesen
    (Prompt: String; Lok: out T_Lok_Nr);
    Gibt Prompt aus und liest die Lok-Nr. ein
procedure Fehler (Text: String);
    Gibt "Fehler:" gefolgt von Text aus
procedure Anzeigen (Text: String);
    Gibt übergebenen Text aus.
procedure Init_Grafik(L1,L2 : T_Lok_Nr);
    Initialisiert den Bildschirm und zeichnet einen Gleis-
    plan. L1/L2 geben die verwendeten Loknummern an.
end Terminal_Io;

```

Benutzung von Terminal_IO



Zustand nach der Initialisierung



Beispiel während des Betriebs

Fahren auf der Anlage

- Zu Beginn: Anlage in mit Initialisierungsbefehl in Startzustand versetzen!
- Loks anhalten
- Weichen „gerade“ schalten
- Anlagenstrom einschalten

Warum?

- Loks speichern ihre Geschwindigkeit
- Weichen haben keinen definierten Ausgangszustand.
- „Alles-Halt“ vorsehen! Reset stellt nicht alles zurück.

Fallstricke VADS-Compiler ...

- **a.run (start)** im aktuellen Verzeichnis oder mit absoluten Pfaden benutzen
a.run kann keine relativen Pfade und keine Links auflösen !
- Daten im Puffer der seriellen Schnittstelle erzeugen manchmal Abbruch mit Meldung:
a.run_etdm : unexpected target signal: processor exception number ...
beim Start des Programms
Abhilfe: Programm nochmals starten
- Programm terminiert ohne Fehlermeldung (!) bei
 - nicht ausführbaren delay-Anweisungen
 - Verklemmungen
 Abhilfe:
V_XTasking.Set_Exit_Disabled (Value: Boolean);
- Compiler erzeugt Fehler in überwachten Ausdrücken der Form:

```
select
  accept bla do
    action1;
  end bla;
or
  when condition => action2;
or
  ...
end select;
```

wenn mit **condition** auf Aggregate (z. B. auf Komponenten von records) zugegriffen wird