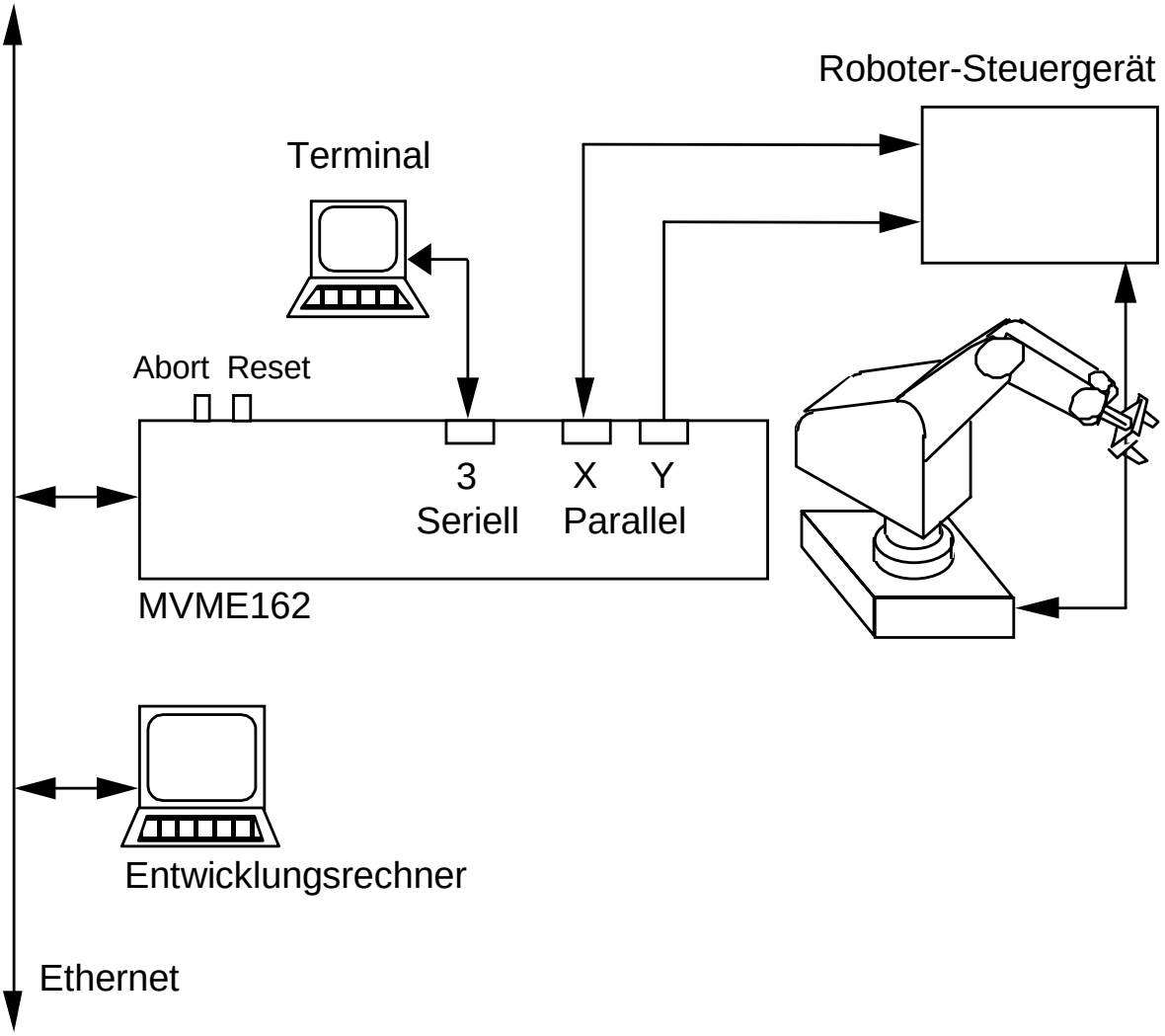
Arbeitsumgebung

- Entwicklungsrechner
  - Unix-System
  - Sun Ray 1 an SunSparc
    SunOS 5.8 (Solaris 2.8), X-Windows
  - im Prinzip jeder Rechner im Haus (möglicherweise jedoch Probleme bei anderen SunOS-Versionen)

- Echtzeitrechner
  - graue Kästen mit Metall-Namen (silber, blei, platin)
  - MVME 162LX-263,
    Prozessor Motorola 68040, 16 MB RAM, VME-Bus,
    Schnittstellen: seriell, parallel, SCSI, Ethernet
  - Roboter-Steuerrechner mit Ada-Laufzeitumgebung

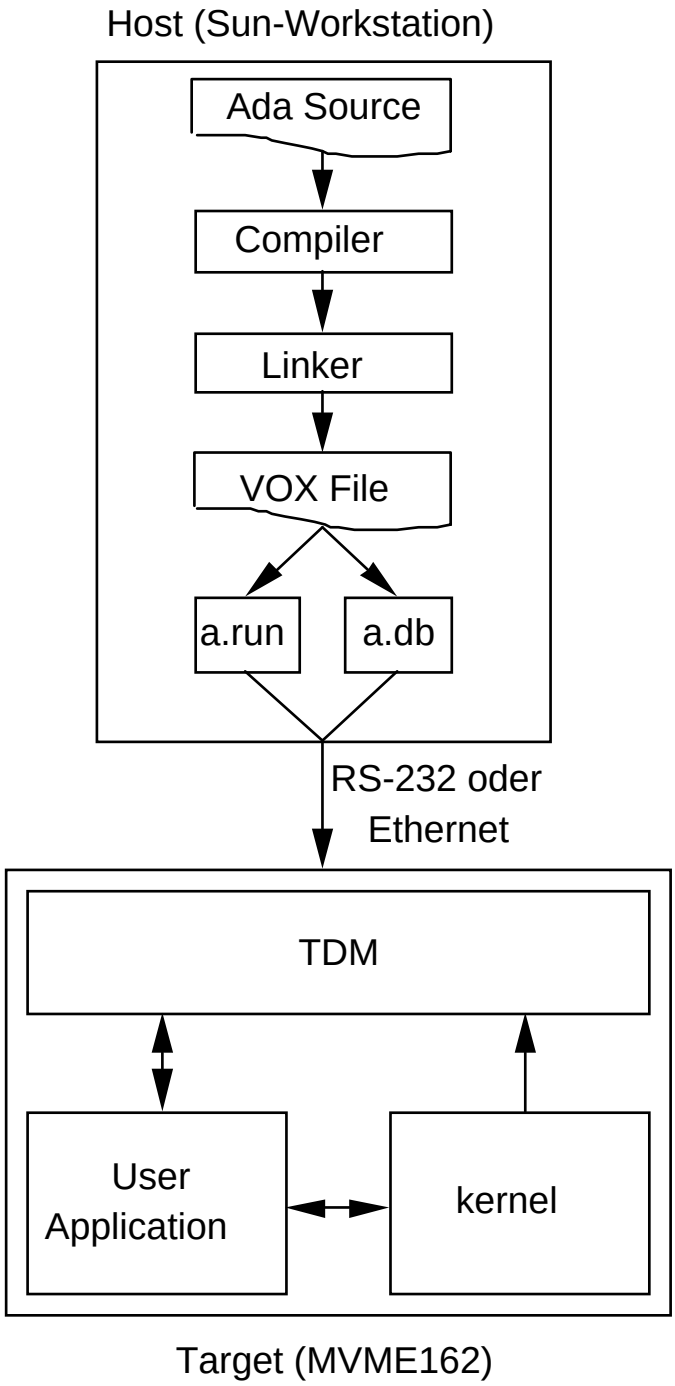
- Ada-Umgebung
  - Ada-Cross-Compiler:
    VADS (Verdix Ada Development System)
  - Informationen zur Ada-Programmentwicklung anschließend und in den Vorlesungsunterlagen aus „Eingebettete Echtzeitsysteme“



Echtzeitrechner MVME162

- 2 serielle Schnittstellen
  - Seriell_1 (Console): verboten
  - Seriell_3: Terminal ⇒ Bildschirmausgabe, Tastatureingabe
- 2 parallele Schnittstellen
  - Parallel_X: Robotersteuerung ⇒ Kontakte einlesen
  - Parallel_Y: Robotersteuerung ⇒ Motorsteuerung

Programme



TDM: Turbo Debug Monitor

- **`etdm`**
  - Ethernet Turbo Debug Monitor
  - verbindet Sun und MVME162 über Ethernet
  - wird nach dem Einschalten und nach Reset automatisch über Ethernet geladen und gestartet
- **`krn.vox`**
  - Laufzeitkern für Ada-Benutzerprogramme im Verdix Objects Excecutable-Format
  - wird beim Laden des Benutzerprogramms mittels **`start`** automatisch als erstes auf MVME162 geladen
  - wird nicht gestartet
- **`move.vox`** (Bsp.)
  - ausführbares Ada-Benutzerprogramm
  - wird mittels **`start`** geladen und gestartet
- **`start`** <Rechnername> <ausführbares Programm>
  - Shell-Script für Mehrbenutzerbetrieb im Praktikum
  - lädt `krn.vox`
  - lädt und startet Vorprogramm zur Identifikation des folgenden Programms (Mehrbenutzerbetrieb)
  - Vorprogramm zeigt Namen des Benutzers an, dessen Programm als nächstes zur Ausführung kommt
  - Vorprogramm erwartet Bestätigung durch Eingabe des Benutzernamens
  - nach Bestätigung lädt **`start`** Benutzerprogramm **`move.vox`** und startet dieses, falls MVME162 nicht von anderem Benutzer belegt

Programmentwicklung

- jedes Verzeichnis kann zu einer Ada-Library modifiziert werden
- alle Robotik betreffenden Dateien stehen in **`~pdv/lehre/robotik/...`** **`(/home/pdv/pdv/lehre/robotik/...)`**
- Vorgaben zu den Aufgaben stehen in **`~pdv/lehre/robotik/{adair,roblib,vorgaben}`**
  - Nesten des Roboters, Hand-Ansteuerung
  - Ausführen einzelner Gelenkschritte
  - Abfrage der Endabschalter
- alle Kommandos der Ada-Entwicklungsumgebung in **`~pdv/vads/bin/`** (unabhängig von Lehrveranstaltung)
- LV-spezifische Kommandos (**`makelib`**, **`start`**) in **`~pdv/lehre/robotik/`**
- zu Beginn jeder Sitzung Umgebung initialisieren: **`source ~pdv/lehre/robotik/init`**
  - in jedem X-Terminal, automatisieren
  - auf Shell achten (**`init.bash, init.tcsh`**)
- Verzeichnis zur Programmentwicklung anlegen, z.B. : **`mkdir ~/robotik/a1/`**
- auf **`pueblo`** einloggen (**`ssh pueblo`**)
  in das Verzeichnis wechseln (**`cd ...`**)
  dort **`makelib`** ausführen $\Rightarrow$ Ada-Library wird erzeugt
- Ada-Programm schreiben

- Namenskonvention: **`name_s.a`** : specification
  `name_b.a` : body
- alt: **`name_.a`** : specification
  `name.a` : body

- Pakete übersetzen:
  `ada name_s.a`
  `ada name_b.a`
- Programm binden:
  `a.ld` <Name Hauptprogramm> **`-o`** <Programmname>
  Bsp: **`a.ld robot_main -o move.vox`**
- wer möchte, benutzt **`a.make`** (s. Manual)
- Hilfe zu allen Kommandos:
  - **`a.help`**
  - Manual „Die Ada-Umgebung“ aus EES
  - **`/home/pdv/pdv/vads/man`** in MANPATH-Variable aufnehmen
- Programm auf Echtzeitrechner übertragen (falls frei):
  `start blei move.vox`

  „silber“, „blei“ und „platin“ sind Ausführungsrechner
- an den Robotern:
  Benutzernamen zur Bestätigung eingeben

Hinweise

- alle Ausgaben über Prozeduren von **`Text_Io`** gelangen zur Sun und werden in dem X-Terminal dargestellt, in dem **`start`** aufgerufen wurde

- bei der Ausgabe von Gleitkommawerten aufpassen
  → `CONSTRAINT_ERROR` möglich

- Rechner nicht länger als 5 Minuten belegen

Einschalten

1. Terminal ein
2. Echtzeitrechner ein
3. **`start`** ...
4. Not-Aus lösen
5. Zeitschaltuhr starten
6. Robotersteuerung ein

Programmende

- normales Ende:
  Ausgabe auf der Sun
- Ende mit Exception:
  Ausgabe auf der Sun (Name der Exception)
- Absturz: (kommt nicht vor !?)
  erst Abort-Knopf, dann Reset an MVME162 (reboot)

Fallstricke VADS-Compiler …

- **`a.run`** (**`start`**) im aktuellen Verzeichnis oder mit absoluten Pfaden benutzen:
  a.run kann keine relativen Pfade und keine Links auflösen!

- **`start`** von „anderen“ Executables (z. B. aus Verzeichnis **`muster/executable`**) mit Pfadangabe aus eigenem Verzeichnis mit Ada-Library heraus

- hängende Interrupts erzeugen 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

  Nur zur Information, keine Fragen:

  Abhilfe: V_XTasking.Set_Exit_Disabled(TRUE)

  Damit wird das Programm bei leerer „run“- oder „delay“-Warteschlange nicht beendet.

  Tritt vor allem bei Verwendung von Interrupt-Service-Routinen auf.

- Compiler erzeugt Fehler in überwachten Ausdrücken:

```
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.
Beispiel für Task, die nicht korrekt übersetzt wird:

```
type T_RESET is record
   FLAG : BOOLEAN := FALSE;
   ZEIT : TIME    := T0;
end record;

RESET : array(WEICHE) of T_RESET;
……
select
   accept go do
      -- some action
   end go;
or
   when RESET(LINKS_UNTEN).FLAG =>
      delay ...;
      action1;
or
   when RESET(LINKS_OBEN).FLAG =>
      delay ...;
      action2;
end select;
```

Vorgaben

- Paket **SERIAL_IO** für Bildschrimausgabe am Terminal ist Spezialisierung von **GSERIAL_IO** für die 3. serielle Schnittstelle (siehe Vorgaben)

```
generic t:Term_T;
package GSERIAL_IO is
  DEFAULT_FORE: constant Text_Io.Field:=2;
  DEFAULT_AFT : constant Text_Io.Field:=3;
  DEFAULT_EXP : constant Text_Io.Field:=0;
  procedure PutS(X: Byte);
  procedure PutS(X: Character);
  procedure PutS(X: String);
  procedure PutS(N: Integer;
    Width: in Text_Io.Field:=Integer'Width);
  procedure PutS(F: Float;
    Fore: in Text_Io.Field:=DEFAULT_FORE;
    Aft : in Text_Io.Field:=DEFAULT_AFT;
    Exp : in Text_Io.Field:=DEFAULT_EXP);
  procedure GetS(X: out Byte);
  procedure GetS(X: out Character);
  procedure GetS(X: out Integer);
  procedure GetS(X: out Float);
  procedure Get_ByteS(X:        out Byte;
                      Got_Byte: out BOOLEAN);
  procedure Put_LineS(X: String);
  procedure Get_LineS(X:    out String;
                      Last: out NATURAL);
  procedure New_LineS(X: POSITIVE:=1);
end GSERIAL_IO;
```

Achtung: Feld für Integer-Ausgabe nicht größer als 9 wählen, sonst wird eine Exception ausgelöst!

- Paket NEST_ROBOT enthält Gelenkkonstanten und die Prozedur zum Nesten einzelner Gelenke:

```
package NEST_ROBOT is

   type T_Robotergelenk is (KOERPERGELENK,
                            SCHULTERGELENK,
                            ELLBOGENGELENK,
                            HANDNEIGGELENK,
                            HANDDREHGELENK);

   EXCEPTION_NESTING_ERROR: exception;
   -- Fehler (Zeitueberschreitung) beim Nesten

   procedure Nest(GELENK: T_Robotergelenk);
   -- angegebenes Gelenk in Nullposition
   -- (Gelenkschrittnummer = 0) fahren

end NEST_ROBOT;
```

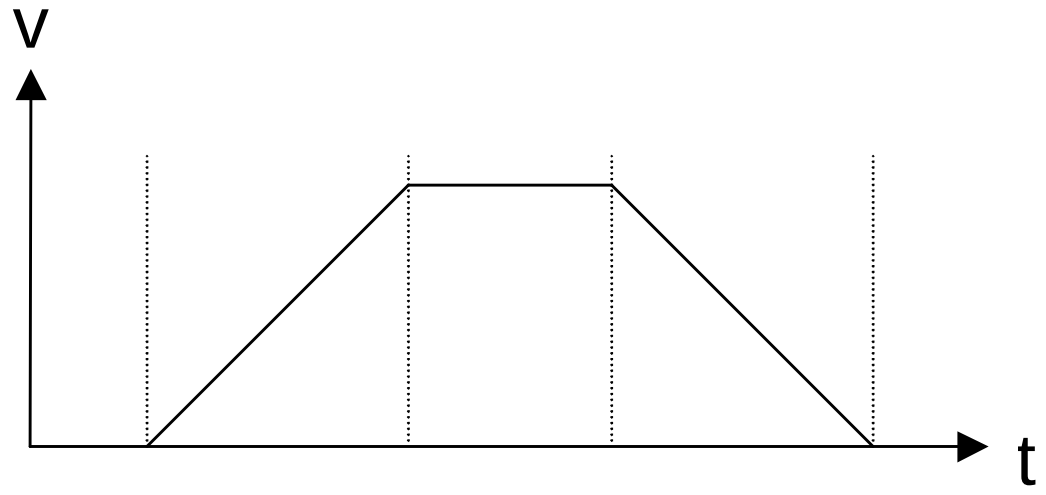
- Paket **GELENKE** enthält technische Konstanten, die Gelenkansteuerung und die Abfrage der Endabschalter:

```
package GELENKE is
   -- Gelenkschritt-Vektor
   -- (fuer ist_position und ziel_position)
   type t_rob_array is array (t_robotergelenk)
                             of integer;

   -- Gelenkwinkel-Vektor
   type t_point     is array (t_robotergelenk)
                             of float;

   -- Flag-Array, fuer die Endabschalter
   type t_schalter  is array (t_robotergelenk)
                             of boolean;

   -- mechanische Begrenzung des Roboters,
   -- Minimum
   min_position : constant t_rob_array :=
     (KOERPERGELENK  => -12000,
      SCHULTERGELENK => -5200,
      ELLBOGENGELENK =>  0,
      HANDNEIGGELENK =>  0,
      HANDDREHGELENK => -4800);

   -- Maximum
   max_position : constant t_rob_array :=
     (KOERPERGELENK  => 0,
      SCHULTERGELENK => 0,
      ELLBOGENGELENK => 3600,
      HANDNEIGGELENK => 4800,
      HANDDREHGELENK => 4800);
```

```
  -- Gelenkwinkel pro Gelenkschritt beim
  -- Roboter in Grad
  grad_pro_schritt : constant t_point :=
    (KOERPERGELENK  => 0.025,
     SCHULTERGELENK => 0.025,
     ELLBOGENGELENK => 0.025,
     HANDNEIGGELENK => 0.0375,
     HANDDREHGELENK => 0.0375);

  -- Einlesen der Endabschalter-Stellungen
  function endabschalter return t_schalter;

  -- ganzen Roboter nesten, Hand auf
  procedure nest_all
    (ist_position: out t_rob_array;
     ist_greifer : out t_greiferzustand);

  -- Kontrolle an Steuerrechner abgeben und
  -- Programm sicher terminieren
  procedure switch_off;

  -- alle Gelenke, deren ist_position von
  -- ziel_position abweicht, um je einen
  -- Schritt in die noetige Richtung bewegen,
  -- und die ist_position aktualisieren
  -- ACHTUNG: ES FINDET KEINE PRUEFUNG AUF
  -- DIE MECHANISCHEN GRENZEN UND KEINE
  -- ABFRAGE DER ENDABSCHALTER STATT
  -- DIES IST SACHE DES AUFRUFERS!!!
  procedure gelenkschritt
    (ist_position : in out t_rob_array;
     ziel_position: in     t_rob_array);
end GELENKE;
```

Hinweise zu Aufgabe 1

- Rampe mit konstanter Beschleunigung:



- 1. Variante: Verzögerungszeit *d* für einen Schritt an Position *s*, am Beispiel der positiven Rampe

$$d = \frac{1}{v}$$

$$v = at$$

$$s = \int v\,dt = \int at\,dt = \frac{at^2}{2} = \frac{v^2}{2a} \quad \Rightarrow \quad v^2 = 2as$$

$$d = \frac{1}{\sqrt{2as}}$$

- 2. Variante: *s(t)* direkt berechnen

- Auflösung der **`delay`**-Anweisung ist viel zu grob, daher:
  a) Verzögerung mit for-Schleife oder
  b) besser: Berechnung von *s* anhand der Rechner-Uhr:

  Paket **`calendar`** mit

```
type time is private;
function clock return time;
function seconds(date: time)
            return day_duration;
```

  Die Rechner-Uhr hat eine Auflösung von 10ms.

  Die Operatoren +, -, <, <=, > und >= sind entsprechend definiert.

  Beispiel:

```
t : time;
dt : float;
t := clock;
...
dt := float(seconds(clock - t));
```

- Vorschlag:

  Innerhalb einer Schleife Soll-Zustand der Gelenke bestimmen und fehlende Schritte sofort ausführen.

  Vorsicht: Das funktioniert nur, wenn so nur sehr wenige Schritte an den Steuerrechner übermittelt werden, da der Roboter sonst zu stark ruckt.

  → Begrenzung auf 10 Schritte!!!

Aufgabenabgabe

- Praktische Vorführung vor dem Betreuer
- Sofern nicht explizit angegeben, sind eigene Testszenarien zu überlegen, die die Fähigkeiten der entwickelten Programme geeignet demonstrieren
- Anwesenheit der gesamten Arbeitsgruppe
- Spätestens in der in der letzten betreuten Rechnerzeit der jeweiligen Abgabewoche, siehe WWW-Seite der Veranstaltung