Echtzeitsysteme

Anhang: Maschinennahes Programmieren in Ada

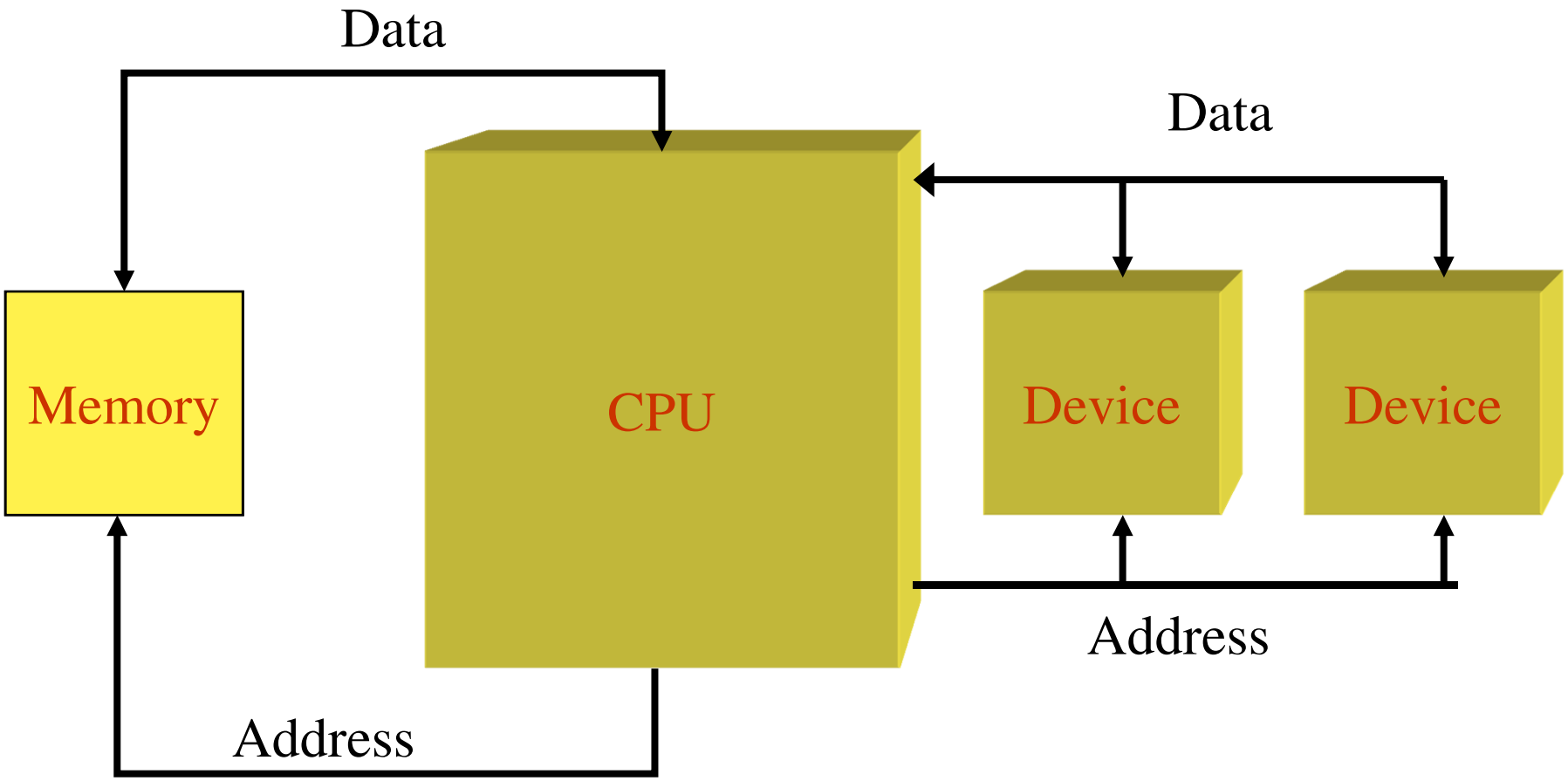
Prof. Dr. Roland Dietrich

Anforderungen

- Typische für Echtzeitsysteme: Zugriff auf spezielle Geräte
  - realisiert durch spezielle Ein-/Ausgabe-Operationen
- Realisierung der Geräte-Ein-/Ausgabe
  - Durch spezielle Geräteregister (***port mapped I/O***), vgl. S. A-3
    - Zugriff auf Geräte erfolgt über einen speziellen Bus (unabhängig vom Zugriff auf den Speicher)
  - Durch spezielle Speicheradressen (***memory mapped I/O***), vgl. S. A-4
    - Zugriff auf Geräte erfolgt über denselben Bus wie auf den Speicher
- Steuerung der Geräte-Ein-/Ausgabe
  - Status-gesteuert
    - Ein Programm kann den Status eines Geräts feststellen
    - Ein Programm kann Aktionen auf dem Gerät veranlassen (Befehle)
    - Ein Programm kann Daten vom Gerät lesen und ins Gerät schreiben
  - Interrupt-gesteuert
    - Ein Gerät ist in der Lage einen Interrupt auszulösen
    - Das Programm reagiert durch Ausführen eines Interrupt-Handlers

Anforderungen

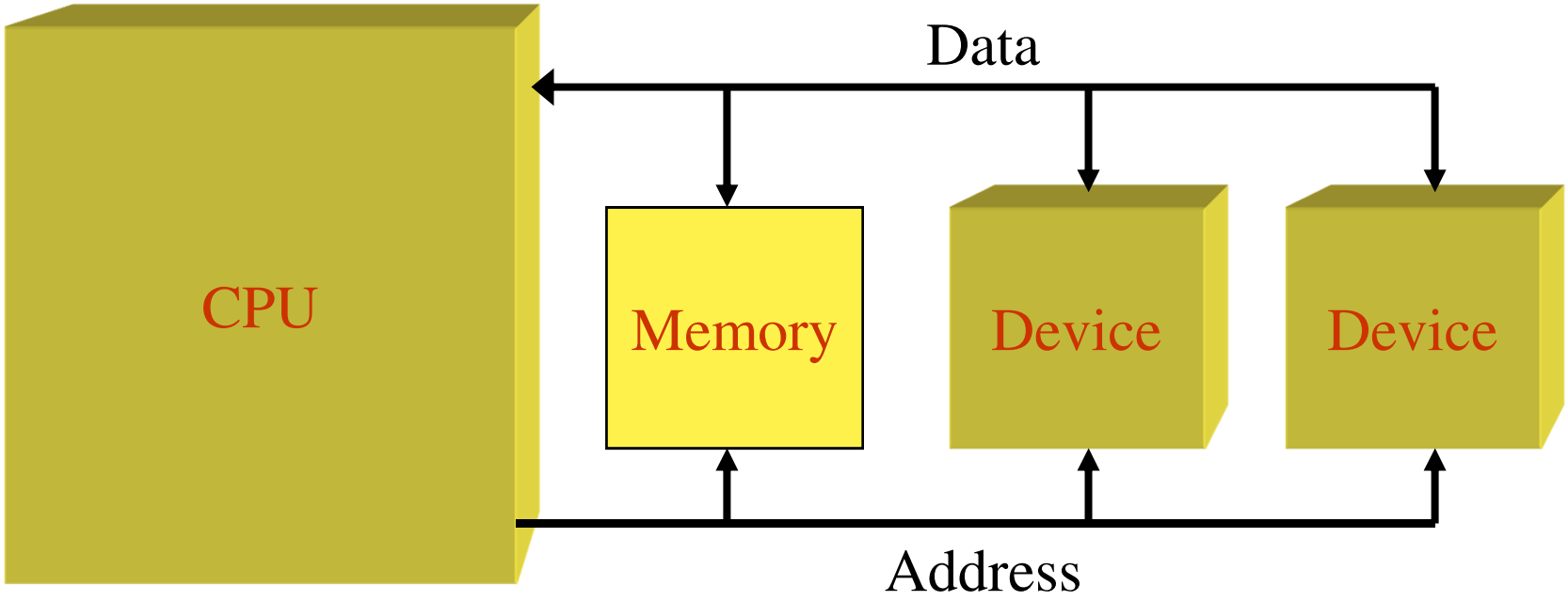
- ***Port mapped I/O:*** separate Busse für Speicher und Geräte



[Quelle Burns und Wellings 2005, Abb. 14.1]

Anforderungen

- ***Memory mapped I/O***: Gemeinsamer Bus für Speicher und Geräte



[Quelle Burns und Wellings 2005, Abb. 14.2]

Beispiel-E-/A-System

- In Anlehnung an Motorola 68000 Prozessoren
  (Siehe [Burns & Wellings, Kap.14.1.4])
  - E-/A-Register sind im Speicher abgebildet (memory mapped I/O)
  - Steuerungs- und Statusregister (***control & status register***, ***csr***)
    - enthalten alle Informationen über den Gerätestatus
    - ermöglichen freigeben und sperren von Interrupts
    - Bedeutung der Bits:

Bits:	**Bedeutung:**	
15 - 12	Errors	*gesetzt bei Gerätefehlern*
11	Busy	*gesetzt, wenn das Gerät aktiv ist*
10 - 8	Unit select	*wenn mehrere Geräte verwaltet werden*
7	Done/ready	*E-/A-fertig oder Gerät bereit*
6	Interrupt enable	*gesetzt, wenn Interrupts freigegeben sind*
5 - 3	reserved	*Gebrauch nicht festgelegt*
2 - 1	Device function	*zeigen die benötigte Gerätefunktion an*
0	Device enable	*gesetzt, wenn das Gerät freigegeben ist*

Beispiel-E-/A-System

- In Anlehnung an Motorola 68000 Prozessoren
  - Datenpuffer-Register dienen als Zwischenspeicher für Daten vom und zum Gerät (***data buffer register***, **dbr**)
    - Bedeutung der Bits (die Daten sind hier Zeichen):

Bits:	**Bedeutung:**
15 - 8	nicht genutzt
7 - 0	Daten

  - Ein Gerät kann mehrere *csr* und *dsr* haben
  - Bei einem Interrupt
    - Speichert der Prozessor den aktuellen Befehlszähler (*Program counter, PC*) und den Prozessor-Status (*Processor status word, PSW*) auf dem Stack
    - der neue PC und PSW werden aus einem sog. **Interrupt-Vektor** geladen
      - erstes Datenwort = Adresse der Interrupt-Service-Routine
      - zweites Datenwort = PSW einschließlich Priorität der Interrupt-Service-Routine

Anforderungen

- Anforderungen an Höhere Programmiersprachen (z.B. Ada) für maschinennahe Programmierung
  - Modularisierung und Kapselung
    - Maschinenabhängige Programmteile sind in der Regel nicht portabel und sollten im Code isoliert werden
      - Ada: Packages und geschützte Typen
  - Geeignete Repräsentation von Geräte-Registern (lesender/schreibender Zugriff), z.B.:
    - Programm-Variable
    - Objekt (in objektorientierten Sprachen)
    - Kommunikationskanal
  - Eine geeignete Repräsentation für Interrupts, z.B.
    - Prozeduraufruf
    - Start einer sporadischen Task
    - Asynchrone Benachrichtigung (vgl. 5-58ff)
    - Bedingungssynchronisation mit gemeinsamer Variable
    - Nachrichtenbasierte Synchronisation: Interrupt = inhaltslose Nachricht

Geräteregister in Ada

- Ada bietet Möglichkeiten, die Implementierung von Datentypen zu beeinflussen: ***representation aspects***
- Beispiele:
  - **Attribute** von Datentypen und -Objekten:
    - Größe (*size*) von Objekten in Bits,
    - Ausrichtung von Objekten im Speicher (*alignement*),
    - Maximaler Speicherplatz für Tasks
    - Adressen von Objekten
  - Werte für Aufzählungskonstanten
  - Record- (Struktur-) Komponenten:
    - Offset
    - Länge (in Bits)

Geräteregister in Ada

- Beispiel-E-/A-System in Ada (Siehe [Burns & Wellings, Kap.14.3])
  - Aufzählungstypen für Codes (Fehler, Funktionen, Einheiten)

```
type Error_T is (None, Read_Error, Write_Error,
                 Power_Fail, Other);
type Function_T is (Read, Write, Seek);
type Unit_T is new Integer range 0 .. 7;
```

  - Record-Typ für Steuerungs- und Statusregister (*csr*), vgl. S. A-5

```
type Csr_T is record
   Errors   : Error_T;
   Busy     : Boolean;
   Unit     : Unit_T;
   Done     : Boolean;
   Ienable  : Boolean;
   Dfun     : Function_T;
   Denable  : Boolean;
end record;
```

Geräteregister in Ada

- Beispiel-E-/A-System in Ada
  - Konkrete Codes für Gerätefunktionen (z.B.):

    ```
    01=READ, 10=WRITE, 11=SEEK
    ```

  - Aufzählungstypen für Codes mit Festen Werten:

```
type Function_T is (Read, Write, Seek);
for Function_T use (Read => 1, Write => 2, Seek => 3 };

type Error_T is (None, Read_Error, Write_Error,
                  Power_Fail, Other);
for Error_T use (None => 0, Read_Error => 1,
     Write_Error => 2, Power_Fail => 3, Other => 4);
-- note, this is in fact the default assignment
type Unit_T is new Integer range 0 .. 7;
```

Geräteregister in Ada

- Beispiel-E-/A-System in Ada
  - Festlegung der Speicherstruktur des Record-Typs für csr (vgl. S. A-5)

```
Word : constant := 2; -- number of storage units in a word
Bits_In_Word : constant := 16; -- bits in word

for Csr_T use record
  Denable  at 0*Word range 0..0; -- at word 0 bit 0
  Dfun     at 0*Word range 1..2;
  Ienable  at 0*Word range 6..6;
  Done     at 0*Word range 7..7;
  Unit     at 0*Word range 8 .. 10;
  Busy     at 0*Word range 11 .. 11;
  Errors   at 0*Word range 12 .. 15;
end record;

for Csr_T'Size use Bits_In_Word; -- the size of object of Csr type
for Csr_T'Alignment use Word; -- object should be word aligned
for Csr_T'Bit_order use Low_Order_First;
  -- first bit is least significant bit of byte
```

Geräteregister in Ada

- Beispiel-E-/A-System in Ada
  - Festelgung von Speicheradressen für Register

```
csr : Csr_T;
for csr'Address use
System.Storage_Elements.To_Address(8#177566#);
```

  - Setzen eines Registers

```
csr := (Denable => True, Dfun => Read,
        Ienable => True, Done => False,
        Unit => 4, Errors => None);
```

  - Lesen des Registers

```
if csr.Errors = Read_Error then
  raise Disk_Error;
end if;
```

  - Zu beachten:
    - `csr` ist eigentlich eine Menge von gemeinsamen Variablen
      - Gleichzeitiger Zugriff vom Gerätesteuerungsprogramm und vom Gerät
    - `csr` sollte im Rahmen eines geschützen Objekts implementiert werden

Interrupts in Ada

- Das **Unterbrechungs-Modell** von Ada
  - Ein **Interrupt** repräsentiert eine Klasse von Ereignissen, die durch die Systemhardware entdeckt wird
  - Das **Auftreten** (***occurence***) eines Interrupts besteht aus zwei Vorgängen:
    - Der **Erzeugung** (***generation***) des Interrupts
      - Das Ereignis in der Hardware, die den Interrupt für das Programm verfügbar macht
    - Der **Auslieferung** (***delivery***) des Interrupts
      - Die Aktion, die die Unterbrechungs-Behandlungsroutine (***Interrupt-Handler***) aufruft
  - Zwischen Erzeugung und Auslieferung ist ein Interrupt "**hängend**" (**pending**)
  - Die **Latenzzeit (*latency*)** eines Interrupts ist der Zeitraum zwischen Erzeugung und Auslieferung
  - Die Unterbrechungs-Behandlungsroutine wir ein mal pro Auslieferung ausgeführt

Interrupts in Ada

- Das Unterbrechungs-Modell von Ada
  - Solange eine Unterbrechung behandelt wird, sind alle Interrupts aus derselben Quelle **blockiert**
    - Es ist Geräte-abhängig, ob blockierte Interrupts hängend bleiben oder verloren gehen
  - Es gibt **reservierte Interrupts** (z.B. Uhr-Interrupts um die delay-Anweisung zu implementieren)
    - Reservierte Interrupts werden vom Laufzeitsystem behandelt
    - Der Programmiere darf keine Behandlungsroutinen für reservierte Interrupts definieren
  - Jeder Interrupt hat einen eindeutigen, implementierungsabhängigen Bezeichner
    - z.B. die Adresse des Interrupt-Vektors

Interrupts in Ada

- Unterbrechungsbehandlung durch **geschützte Prozeduren**
  - `pragma Attach_Handler(Handler_Name, Expression);`
    - steht in der Spezifikation oder im Rupf eines geschützten Objekts (in einem Package*, library level*)
    - weist dem Interrupt mit dem Bezeichner, der sich als Wert des Expression ergibt, die Prozedur `Handler_Name` des Objekts als Handler zu
      - Die Zuweisung erfolgt bei Erzeugung des geschützten Objekts
    - Löst die Ausnahme `Program_Error` aus, falls
      - bei ein Objekt erzeugt wird und der Interrupt reserviert ist
      - wenn der Interrupt bereits einen Handler hat
      - wenn eine Prioritätsobergrenze nicht im zulässigen Wertebereich ist
  - `pragma Interrupt_Handler(Handler_Name);`
    - steht in der Spezifikation eines geschützten Objekts (in einem Package)
    - ermöglicht die dynamische Zuteilung der parameterlosen Prozedur `Handler_Name` des Objekts als Interrupt-Handler für einen oder mehrere Interrupts
      - Mit Funktionen des Packages `Ada.Interrupts` (s. [Beispiele zum Anhang])
    - Die Objekte müssen in einem Package erzeugt werden (*library level*)

Interrupts in Ada

- **Beispiel:** Zugriff auf einen Analog-/Digital-Konverter (ADC)

  [Burns & Wellings 2009, Kap. 14.3.3]

  – 16 Bit Register zum Auslesen der Ergebnisse: Adresse 8#150000#

  – 16 Bit Register zum Steuern: Adresse 8#150002#

    - Bedeutung der Steuerungsbits:

Bit	*Name*	*Bedeutung*
0	A/D Start	Set to 1 to start a conversion
6	Interrupt/Enable/Disable	Set to 1 to enable the device
7	Done	Set to 1 when conversion complete
8-13	Channel	Required input channel out of 64
15	Error	Set if device malfunctions

  – Es wird davon ausgegangen, dass der Bezeichner `Adc` als eine Interrupt-Id registriert ist im Package `Ada.Interrupts.Name` (s. [Beispiele zum Anhang])

  – Ada-Code: siehe [Beispiele zum Anhang]

Literatur

[Burns & Wellings 2009] Alan Burns, Andy Wellings: *Real-Time Systems and Programming Languages. Ada, Real-Time Java and C/Real-Time POSIX*. Addison Wesley, 2009.