Control-Routing Funktionsbeschreibung

Schnittstelle zur Experimentkontrolle

April 1995 K. Huber, Strahlenzentrum Univ. Gießen
Version 11.Oct.2024

Inhaltsverzeichnis

1 Einleitung

Zur Datenerfassung und Experimentsteuerung existieren zwei verschiedene Rechner-Interfaces, das **Data-Routing** und das **Control-Routing**, so genannt nach den Aufgaben, zu denen sie im Wesentlichen eingesetzt werden:

- Zur schnellen Erfassung von großen Datenmengen wird vorzugsweise das **Data-Routing** verwendet. Es ermöglicht eine schnelle Dateneingabe von bis zu 8 Eingabekanälen. Eine Datenausgabe ist damit nicht möglich.
- Das **Control-Routing** hingegen wird hauptsächlich zur Steuerung des Experimentes eingesetzt. Es erlaubt die Ein- und Ausgabe von einzelnen Datenworten, wobei bis zu 8 Module mit je 8 Registern, also maximal 64 Register, adressierbar sind. Ferner ist eine Interrupt-Eingabe vorgesehen.

Die vorliegende Funktionsbeschreibung befasst sich ausschließlich mit dem **Control-Routing**. Es wird im folgenden oft abgekürzt als **Routing** bezeichnet.
Für das **Data-Routing** existiert eine eigene Beschreibung.

Das Kapitel **Aufbau und Funktion** gibt einen Überblick über den Aufbau des Control-Routing.

Das Kapitel **Standardkomponenten** enthält die Aufgabenbeschreibungen und Bedienungsanleitungen der Standardkarten des Routing.

Das Kapitel **Standard-Interface-Karten** enthält die Aufgabenbeschreibungen und Bedienungsanleitungen der Routing-Karten, die allgemein bei Experimenten zum Einsatz kommen.

Das Kapitel **Spezielle Interface-Karten** enthält eine Aufstellung der Routing-Karten, die für einzelne Experimente entwickelt wurden. Ihre Beschreibung findet man i.a. in den Unterlagen der Experimente, bei denen sie eingesetzt werden.

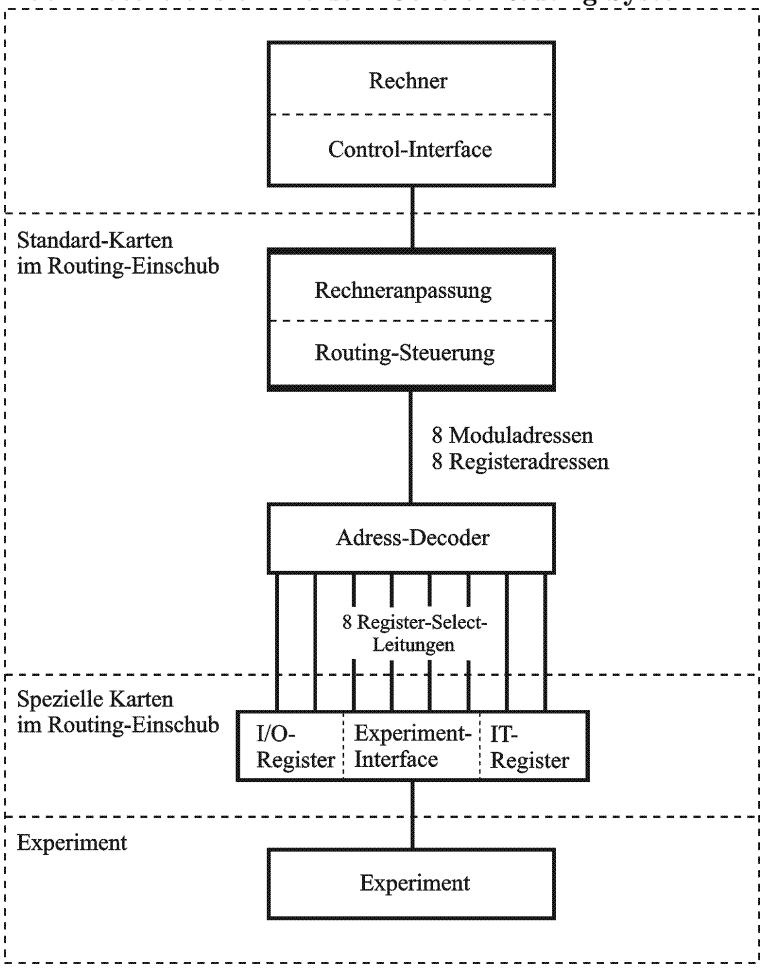
Das Kapitel **Technische Details** gibt technische Detail-Informationen über die Komponenten und deren Schnittstellen.

Im Kapitel **Oldies** sind überholte Beschreibungen gesammelt.

2 Aufbau und Funktion

2.1 Aufbau des Routing

Abb.: Datentransfer mit dem Control-Routing-System



- **Rechner**
  Das Control-Routing kann an jeden Rechner mit geeignetem I/O-Port angeschlossen werden. Realisiert wurden bisher Anschlüsse an TR86-Rechner, PDP11-Rechner, VME-Systeme und über USB an PCs.
  Aktuell ist das Control-Interface auf einem IP-Board (Industrie Pack) implementiert, so dass es an jeden Rechner angeschlossen werden kann, für den ein IP-Board-Carrier existiert (VME, PC usw.).
  Neu ist eine Rechneranpassung für USB-Anschluss, so dass das Routing jetzt auch über geeignete PCs gesteuert werden kann.
- **Rechneranpassung und Routing-Steuerung**
  Diese beiden Funktionen sind entweder auf einer gemeinsamen Karte oder auf zwei benachbarten Karten, mit der Steuerung links und der Anpassung rechts, untergebracht. Die Rechneranpassung umfasst die Sender und Empfänger der Signale über das Kabel

zum Rechner. Die Routing-Steuerung ist das logische Interface zwischen den rechnerseitigen Signalen und dem Routing-Bus.

- **Adress-Decoder**
  Zur Erleichterung des Anschlusses von Experiment-Interfaces wurde eine standardisierte Karte zur Entschlüsselung der binär verschlüsselten Modul- und Register-Adressen entwickelt. Die Adress-Decoder-Karte belegt eine Steckplatz abhängige Moduladresse und entschlüsselt die zugehörigen 8 Registeradressen. Über den P-Bus gibt sie 8 Register-Select-Signale an benachbarte Experiment-Interfaces weiter. Ihr Einsatz kann entfallen, wenn die Experiment-Interfaces die Entschlüsselung selber vornehmen.
- **Experiment-Interfaces**
  Eine Experiment-Interface-Karte enthält ein oder mehrere I/O-Register zur Ein- und Ausgabe der Daten. Bis zu 8 Register pro Karte können direkt adressiert werden und über Hilfsadressen natürlich weitere. Ferner kann sie vom Experiment ausgehende Interrupt-Wünsche in einem I/O-Register (IT-Register) speichern und dem Rechner eine Interrupt-Anforderung schicken.
  Es gibt bereits eine Reihen solcher Experiment-Interface-Karten (paralleles und serielles I/O, programmierbare Timer, Interrupt-Eingabe), die bei Bedarf möglicherweise direkt verwendet oder doch zumindest als Vorlage für Neuentwicklungen dienen können.

2.2 Überrahmen

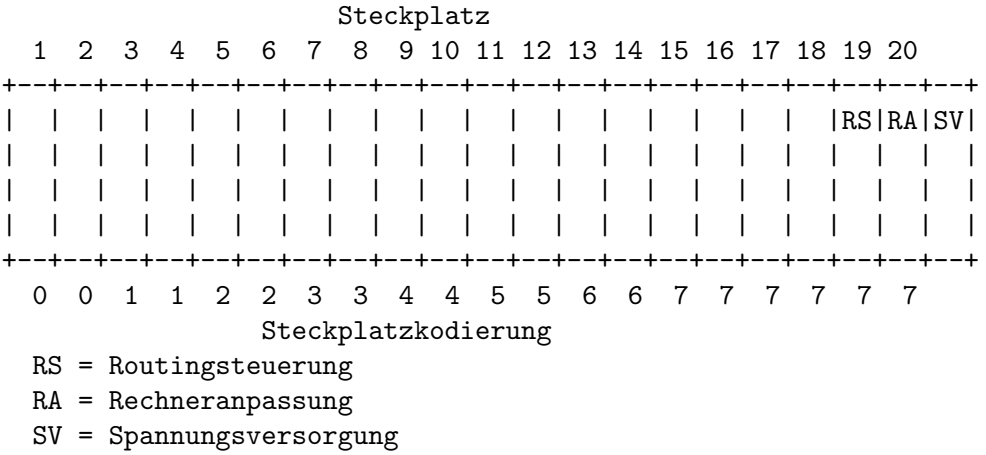
Das Control-Routing ist in einem 19"-Überrahmen untergebracht mit 20 Steckplätzen für Europakarten (Siehe [Abb Routing-Ueberrahmen], Seite 4.). Es wird der gleiche Überrahmen wie für das Data-Routing verwendet. In Ausnahmefällen können Data-Routing und Control-Routing im gleichen Überrahmen untergebracht werden, wenn die Bus-Verdrahtung in der Mitte durchtrennt wird und für die zweite Hälfte ein Bus-Abschluss nachgerüstet wird.
Die Routing Back-Plane enthält 42 allgemeine Bus-Leitungen für die Routing-Steuerung und 11 Privat-Bus-Leitungen zur Kommunikation benachbarter Karten.
Außer der Spannungsversorgung sind alle Schaltungskomponenten auf steckbaren Karten untergebracht.

Als Steckverbindung zum Routing-Bus werden 64-polige VG-Stecker verwendet. Die Anschlüsse der VG-Leisten sind im Überrahmen zum Teil als durchgehender Bus verdrahtet (allg. Bus: 1a,1c,...,21c) und zu einem anderen Teil bestehen Verbindungen zu benachbarten Steckerleisten (Privat-Bus: 22a/c,...,32a/c) (Siehe Abschnitt 7.1 [Routing-Bus], Seite 54.). Ferner ist für jede Leiste von links beginnend eine Steckplatzkodierung von 0-7 verdrahtet, wobei jedoch jeweils zwei benachbarte Leisten die gleiche Kodierung haben. Die vier überbleibenden Steckplätze am rechten Ende erhalten alle die Kodierung 7. Sie sind vorzugsweise für die Rechner-Anpassung und Routing-Steuerung zu benutzen, da der Bus-Abschluss sich am linken Ende des Überrahmens befindet. Am äußersten rechten Ende ist die Netzkarte mit Netzschalter und Betriebsspannungsanzeigen fest installiert.

Die Frontplattenbreite für eine Steckkarte ist üblicherweise 20 mm, es stehen jedoch auch 40 und 50 mm Frontplatten zur Verfügung.

Achtung:
Es wird dringend empfohlen, die Frontplatten der Steckkarten mit dem Überrahmen zu verschrauben zur Vermeidung von Betriebsstörungen. Bitte die Schrauben nicht gewaltsam anziehen, da dies zur Zerstörung der Gewinde im Überrahmen führt. Schrauben von min. 10mm Länge verwenden, sonst besteht ebenfalls die Gefahr der Zerstörung der Gewinde.

Abb.: Routing-Überrahmen



3 Standardkomponenten

3.1 Rechneranpassung und Routing-Steuerung

Diese beiden Funktionen sind entweder auf einer gemeinsamen Karte oder auf zwei benachbarten Karten, mit der Steuerung links und der Anpassung rechts, untergebracht.

3.1.1 Rechneranpassung

Die Rechneranpassungs-Karte enthält Sender, Empfänger und eventuell Pegelwandler für die Dialog- und Datenleitungen zwischen dem Rechner und dem Routing.

Bisher wurden drei Anpassungen an verschiedene Host-Rechner realisiert:

- Intel-PC - USB - Anpassung
  Mit der USB-Anpassung kann das Routing über einen Intel-PC mit VxWorks 7 kontrolliert werden.
  Rechneranpassung und Routing-Steuerung sind gemeinsam auf einer Europakarte untergebracht. Sie besitzt keine Bedienungselemente.
- VME - Industry Pack - Anpassung
  Noch aktuell ist die VME - Anpassung, die zusammen mit dem rechnerseitigen Interface eine Schnittstelle zu dem Industry Pack Standard darstellt. Für diesen IP-Standard werden Carrier-Boards für eine Anzahl weiterer Plattformen (z.B. PC) angeboten, so dass ein Anschluss des Control-Routing an einen anderen Rechnertyp ohne großen Aufwand möglich ist.
  Die Verbindung zum Rechner erfolgt über ein max. 40m langes 50-poliges Flachbandkabel. Kurze Kabellängen sind zu bevorzugen da die Länge in die Übertragungsgeschwindigkeit eingeht und kürzere Kabel störungssicherer sind.
  Rechneranpassung und Routing-Steuerung sind gemeinsam auf einer Europakarte untergebracht. Sie besitzt keine Bedienungselemente.
- PDP11 - DRV11-J - Anpassung
  Die PDP11-Anpassungen wurden inzwischen ausgemustert. Ihre Beschreibung finden Sie bei den 'Oldies'.

3.1.2 Routing-Steuerung

Die Routing-Steuerung ist das logische Interface zwischen den rechnerseitigen Signalen und dem Routing-Bus. Sie kontrolliert den Datentransfer mit den Registern der Experiment-Interfaces und sammelt deren Interruptanforderungen, um sie an den Rechner weiterzuleiten.

Die Control-Routing-Steuerung besitzt keine Bedienungselemente.

Siehe Abschnitt 2.1 [Aufbau des Routing], Seite 2.
Siehe Abschnitt 7.2 [Komponenten und Schnittstellen], Seite 59.

3.2 Address-Decoder

Zur Erleichterung des Anschlusses von Experiment-Interfaces wurde eine standardisierte Karte zur Entschlüsselung der binär verschlüsselten Modul- und Register-Adressen entwickelt. Die Address-Decoder-Karte belegt eine Steckplatz abhängige Moduladresse (3 Bits) und

entschlüsselt die zugehörigen Registeradressen (3 Bits). Über den P-Bus (VG-Stecker Pins 22c - 29c; Siehe Abschnitt 7.1 [Routing-Bus], Seite 54.) gibt sie 8 Register-Select-Signale an rechts nachfolgende Experiment-Interfaces weiter. Ihr Einsatz kann entfallen, wenn die Experiment-Interfaces die Entschlüsselung selber vornehmen.

Sobald das Signal "Address valid" anliegt, vergleicht die Address-Decoder-Karte ihre Steckplatzadresse PADR<0-2> mit der aktuellen Moduladresse MADR<0-2>. Bei Gleichheit aktiviert sie einen 3 zu 8 Demultiplexer, der aus der codierten Registeradresse RADR<0-2> 8 einzelne Register-Select-Signale erzeugt (Siehe Abschnitt 2.1 [Aufbau des Routing], Seite 2.), die sie über den P-Bus (Siehe Abschnitt 7.1 [Routing-Bus], Seite 54.) den benachbarten Experiment-Interfaces zur Verfügung stellt.

Die über den P-Bus angeschlossenen Experiment-Interfaces nutzen die Register-Select-Signale zusammen mit den Routing-Bus-Signalen "Enable Read", "Data Accepted" und "Data Available" zur Steuerung der Ein- und Ausgabe von Daten.

Es existieren zwei Versionen der Address-Decoder-Karte, die sich in den Bedienungselementen und Anzeigen und zum Teil auch in der Funktion unterscheiden:

Bedienungselemente:
alte Version:

```
Schalter: Run/Stop
    Run :  Die Address-Decodierung ist in Betrieb
    Stop:  Die Address-Decodierung ist außer Betrieb und alle
           Register-Select-Signale sind abgeschaltet (high).
```

neue Version: keine Bedienungselemente

Anzeigen:
alte Version:

```
LED : Select
      LED leuchtet: Address-Decoder-Karte ist durch Moduladresse selektiert
LEDs: 4 2 1
      Anzeige der Steckplatzadresse
      LED leuchtet: Bit der angegebenen Wertigkeit ist gesetzt
```

neue Version:

```
LEDs: 0 1 2 3 4 5 6 7
      Register select
      LED n leuchtet: Register n ist selektiert
LEDs: 4 2 1
      Anzeige der Steckplatzadresse
      LED leuchtet: Bit der angegebenen Wertigkeit ist gesetzt
```

Funktionelle Unterschiede:
alte Version:
Ursprünglich war geplant, dass das "Address_Error"-Signal (Routing-Bus VG-Stecker Pin 16c) von jedem Experiment-Interface gelöscht wird, dessen Register adressiert werden.

Tatsächlich wurde jedoch auf der alten Address-Decoder-Version "Address_Error" bereits gelöscht, wenn diese über die Moduladresse angesprochen wurde, und die Experiment-Interfaces haben möglicherweise das "Address_Error"-Signal nicht bedient. Damit kann jedoch die Software nicht feststellen, ob eine benötigte Experiment-Interface-Karte fehlt oder defekt ist. Durch Auftrennen der Verbindung nach Pin 16c auf dem Address-Decoder ist diese Panne zu beheben. Dann müssen gegebenenfalls aber die Experiment-Interfaces nachgerüstet werden (s.u.).

neue Version:
Die neue Address-Decoder Version überlässt das Löschen des "Address_Error"-Signals den Experiment-Interfaces, die dann zum Teil aber nachgerüstet werden müssen (teilweise bereits geschehen), falls die Software "Address_Error" abprüft.

Nachrüstung:
Als Nachrüstung genügt eine Diode vom Eingang der verwendeten "Register Select"-Signale zum Ausgang des "Address_Error"-Signals, z.B.:

```
22c ---|<|--- 16c
```

4 Standard-Interface-Karten

4.1 Serielle Ausgabe (Neues Design / EW 1-14)

Dieses PSO14 Control Routing Board ist ein neuentwickelter Parallel-Seriell-Wandler mit Ausgabe der seriellen Signale über Lichtleiter, der die G. Hoffmann Wandler zur Steuerung der Netzgeräte der neuen e-Kanone im Labor 017 ersetzt. Er kann mit unterschiedlicher Firmware programmiert werden um damit die alten Hofmann-Module (z.B. DAC-Karten) steuern zu können oder die neuentwickelten, seriellen AD5541CR oder LTC2758 DAC-Karten.

4.1.1 Serielle Protokolle

4.1.1.1 Original Hofmann Protokoll

Im Hofmann Protokoll wiederholen die PSO-Karten nach einer kurzen Pause beständig ihre Ausgabe. Nach dem Anschalten starten die Karten sofort mit der Ausgabe der zunächst undefinierten Information. Erhalten sie über das Control Routing neue Daten so wird die laufende Ausgabe abgebrochen und eine neue wird gestartet. Der serielle Empfänger erkennt den Abbruch und verwirft die unvollständigen Daten. Dies führt bei zu rasch aufeinander folgender Eingabe neuer Daten dazu, dass die serielle Ausgabe niemals fertig wird.

```
Die serielle Ausgabe wird mit 32.768kHz oder 2.4576 MHz getaktet.
Ein Bit ist 8 Takte lang.
Nach 16 Bits oder einem Abbruch folgt eine 8 Takte lange Pause.
Bit = 0:  1 1 0 0 0 0 0 0
Bit = 1:  1 1 1 1 1 0 0 0
Pause  :  0 0 0 0 0 0 0 0
LSB wird zuerst gesendet.
```

4.1.1.2 Modifizierte Hofmann Protokolle

Es stehen zwei leicht unterschiedliche Versionen zur Verfügung, mit denen versucht wird die Schwächen des Originals zu umgehen:

- Nach dem Powerup Reset erfolgt zunächst keine serielle Ausgabe um keine undefinierten Daten auszugegeben.
- Versionen:
  - *PSO2V1*
    Wie im Original werden die Ausgaben beständig wiederholt. Die erste laufende serielle Ausgabe nach der Eingabe neuer paralleler Daten wird jedoch nicht unterbrochen durch eine weitere Eingabe neuer Daten. Die neuen Daten gehen verloren. Nachfolgende Eingaben unterbrechen wie im Original Hofmann Protokoll eine laufende Ausgabe.

  - *PSO2V4*
    Die Daten werden mit jeder parallelen Eingabe nur einmal seriell ausgegeben. Die

laufende serielle Ausgabe wird nicht unterbrochen durch eine parallele Eingabe neuer Daten. Die neuen Daten gehen verloren.

```
Die serielle Ausgabe wird nur mit 2.4576 MHz getaktet.
Ein Bit ist 8 Takte lang.
Nach 16 Bits oder einem Abbruch folgt eine 8 Takte lange Pause.
Bit = 0:  1 1 0 0 0 0 0 0
Bit = 1:  1 1 1 1 1 0 0 0
Pause  :  0 0 0 0 0 0 0 0
LSB wird zuerst gesendet.
```

4.1.1.3 PSO16/32 Protokolle

Die PSO16/32 Protokolle übertragen 16 bzw. 32 Bit serielle Daten. Sie werden zum Steuern der neuen AD5541CR (16 Bit) und LTC2758 DAC-Karten (32 Bit) benötigt. Sie sind NICHT kompatibel mit vorhandenen Hofmann DAC-Karten.

- Nach dem Powerup Reset erfolgt zunächst keine serielle Ausgabe um keine undefinierten Daten auszugegeben.
- Die Daten werden mit jeder parallelen Eingabe nur einmal seriell ausgegeben.
- Zuerst wird ein 15 Bit langes festes Muster als Header gesendet, auf das sich die DAC-Karte synchronisiert. Dann folgen nahtlos die 16 bzw. 32 Daten-Bits. MSB wird zuerst gesendet.

  ```
  Bit-Takt:  2.5000 MHz
  Header  :  101010101010101
  Daten   :  xxxxxxxxxxxxxxx/xxxxxxxxxxxxxxx
  ```

- Die laufende serielle Ausgabe wird nicht unterbrochen durch eine parallele Eingabe neuer Daten. Die neuen Daten gehen verloren.

4.1.2 PSO2 Firmware

Mit der PSO2 Firmware können die beiden LWL-Sender unabhängig von einander programmiert und betrieben werden. Die PSO2 Firmware existiert in zwei Versionen (PSO2V1 und PSO2V4) mit den unterschiedlichen, modifizierten Hofmann Protokollen PSO2V1 bzw. PSO2V4. Jede dieser Versionen kann durch Steckbrücken für einen Hofmann kompatiblen Modus (MODE = old) oder einen AD5541CR DAC-Karten kompatiblen Modus (MODE = new) konfiguriert werden.

4.1.2.1 Bedienelemente (PSO2)

```
LED "Busy "  :  Anzeige einer Schreiboperation
LWL-Buchse 0 :  serieller Lichtleiter-Ausgang 0
LWL-Buchse 1 :  serieller Lichtleiter-Ausgang 1
Brücke Mode 0:  Stecker für Modus old/new Kanal 0
Brücke Mode 1:  Stecker für Modus old/new Kanal 1
TP Word 0    :  Triggersignal für Start einer seriellen Ausgabe 0
TP Bit 0     :  Triggersignal für Bits einer seriellen Ausgabe 0
TP Word 1    :  Triggersignal für Start einer seriellen Ausgabe 1
TP Bit 1     :  Triggersignal für Bits einer seriellen Ausgabe 1
```

4.1.2.2 Programmierung (PSO2)

Mit der PSO2 Firmware belegt die PSO14-Karte zwei Registeradressen des Address-Decoders mit doppelter Funktion:

```
Register 0    :  Statusabfrage Kanal 0 / Ausgabe Datenwort Kanal 0
Register 1    :  Statusabfrage Kanal 1 / Ausgabe Datenwort Kanal 1
```

Statusabfrage:
Beim Adressieren des Registers antwortet der zugeordnete Kanal nur mit einem Address-Accepted (AddAcc) wenn er nicht busy ist. Dieses Signal kann über den Routing-Status abgefragt werden:

```
Eingabe ((Routing Status & 0x0008) != 0) -> Kanal busy
```

Diese Statusabfrage ist nur bei Verwendung einer neuen Address-Decoder-Karte (EW 3/04) möglich da die alten (EW 21-88) ebenfalls dieses Bit bedienen (Siehe Abschnitt 3.2 [Address-Decoder], Seite 5.). Auf den alten gegebenenfalls die 16c-Verbindung zum VG-Stecker auftrennen. Für Interface-Karten, die AddAcc nicht bedienen führt dies aber zu einer Fehlermeldung falls die Software AddAcc abprüft!

Daten Ausgabe:
Die 16 Daten-Bits werden in dem zugeordneten Datenregister gespeichert und die Parallel-Seriell-Wandlung gestartet.

4.1.3 PSO32 Firmware

Mit der PSO32 Firmware können die beiden LWL-Sender unabhängig von einander für das PSO16 oder PSO32 Protokoll programmiert und betrieben werden zur Kontrolle der neuen AD5541CR (16 Bit) und LTC2758 DAC-Karten (32 Bit). Da geplant ist, die AD5541CR-Karten auf das 32 Bit Protokoll umzuprogrammieren, wird das 16 Bit Protokoll möglicherweise in Zukunft nicht mehr gebraucht...

4.1.3.1 Bedienelemente (PSO32)

```
LED "Busy "   :  Anzeige einer Schreiboperation
LWL-Buchse 0 :  serieller Lichtleiter-Ausgang 0
LWL-Buchse 1 :  serieller Lichtleiter-Ausgang 1
Brücke Mode 0:  unbenutzt
Brücke Mode 1:  unbenutzt
TP Word 0     :  Triggersignal für Start einer seriellen Ausgabe 0
TP Bit 0      :  Triggersignal für Bits einer seriellen Ausgabe 0
TP Word 1     :  Triggersignal für Start einer seriellen Ausgabe 1
TP Bit 1      :  Triggersignal für Bits einer seriellen Ausgabe 1
```

4.1.3.2 Programmierung (PSO32)

Mit der PSO32 Firmware belegt die PSO14-Karte zwei Registeradressen des Address-Decoders mit doppelter Funktion:

```
Register 0    : Statusabfrage Kanal 0 / Kommando Ausgabe
Register 1    : Statusabfrage Kanal 1 / Daten Ausgabe
```

Statusabfrage:
Beim Adressieren des Registers antwortet der zugeordnete Kanal nur mit einem Address-

Accepted (AddAcc) wenn er nicht busy ist. Dieses Signal kann über den Routing-Status abgefragt werden:

```
Eingabe ((Routing Status & 0x0008) != 0) -> Kanal busy
```

Diese Statusabfrage ist nur bei Verwendung einer neuen Address-Decoder-Karte (EW 3/04) möglich da die alten (EW 21-88) ebenfalls dieses Bit bedienen (Siehe Abschnitt 3.2 [Address-Decoder], Seite 5.). Auf den alten gegebenenfalls die 16c-Verbindung zum VG-Stecker auftrennen. Für Interface-Karten, die AddAcc nicht bedienen führt dies aber zu einer Fehlermeldung falls die Software AddAcc abprüft!

Kommando Ausgabe:

```
unteres Nippel -> Kanal 0, oberes Nippel -> Kanal 1
  Bit wenn gesetzt:
  0/4 1 -> nächste 16 Bit Datenausgabe auf low-Register
  1/5 1 -> nächste 16 Bit Datenausgabe auf high-Register
  2/6 0/1 -> 16/32 Bit Mode
  3/7 1 -> Start des Kanals 0/1
z.B. 0xAD ->
  Start Kanal 0, Mode32, nächstes Laden: low-Register
  Start Kanal 1, Mode16, nächstes Laden: high-Register
```

Daten Ausgabe:

Die 16 Daten-Bits werden gemäß der Kommando-Bits in den Datenregistern gespeichert. Die Daten bleiben auch nach einer seriellen Übertragung erhalten.

4.1.4 LWL-Planung

Die Leistung der LWL-Sender SFH756V muss an die Leitungslänge und die Empfindlichkeit der Empfänger SFH551V angepasst werden. Bisher wurden die Sender nahezu mit voller Leistung betrieben, was bei kurzen Leitungen (<2m) zu Übersteuerung der Empänger führt. Rechnungen mit Hilfe der Datenblätter und Application Notes und Tests zeigen, dass statt des bisher verwendeten Vorwiderstands von 82 Ohm eher 330 Ohm geeignet sind um bei Zimmertemperatur Leitungslängen von 1.5 m bis 20 m sicher abzudecken. Bei Tests mit den AD5541CR DAC-Boards waren die Grenzen ca. 82 Ohm bei 1.20 m und 2.5 kOhm bei 10 m.

```
Bezug: 1mW -> 0db

        Rechnung für max Leitungslänge (sicherer Betrieb nach Application Notes)
Fiber length                                  [m]   30,00   20,00   10,00    5,00
Detector power (min=-17db, max=-6db)         [db]  -17,00  -17,00  -17,00  -17,00
Fiber attenuation (0.22db/m)                 [db]    6,60    4,40    2,20    1,10
Additional fiber attanuation (0.5db/m)       [db]    1,50    1,50    1,50    1,50
Aging of fiber and known link disadvantages  [db]    1,00    1,00    1,00    1,00
Aging of transmitter and receiver            [db]    2,00    2,00    2,00    2,00
Range of emitter power due to temperature    [db]    3,00    3,00    3,00    3,00
Min emitter power (max 0db)                  [db]   -2,90   -5,10   -7,30   -8,40
LED current                                  [mA]   26,68   16,54   10,25    8,07
Resistor at 3V                             [kOhm]    0,112   0,181   0,29    0,372

     Rechnung für min Leitungslänge ohne Übersteuerung des Empfängers
Fiber length                                  [m]    3,00    2,00    1,50    1,00
Detector power (min=-17db, max=-6db)         [db]   -6,00   -6,00   -6,00   -6,00
Fiber attenuation (0.22db/m)                 [db]    0,66    0,44    0,33    0,22
```

Additional fiber attanuation (0.5db/m)	[db]	1,50	1,00	0,75	0,50
Aging of fiber and known link disadvantages	[db]	0,00	0,00	0,00	0,00
Aging of transmitter and receiver	[db]	0,00	0,00	0,00	0,00
Range of emitter power due to temperature	[db]	0,00	0,00	0,00	0,00
Angstfaktor	[db]	-3,00	-3,00	-3,00	-3,00
Max emitter power (max 0db)	[db]	-6,84	-7,56	-7,92	-8,28
LED current	[mA]	11,33	9,69	8,96	8,29
Resistor at 3V	[kOhm]	0,265	0,310	0,335	0,362

Rechnung für max Leitungslänge (sicherer Betrieb bei Zimmertemperatur)

Fiber length	[m]	30,00	20,00	10,00	5,00
Detector power (min=-17db, max=-6db)	[db]	-17,00	-17,00	-17,00	-17,00
Fiber attenuation (0.22db/m)	[db]	6,60	4,40	2,20	1,10
Additional fiber attanuation (0.5db/m)	[db]	1,50	1,50	1,50	1,50
Aging of fiber and known link disadvantages	[db]	1,00	1,00	1,00	1,00
Aging of transmitter and receiver	[db]	2,00	2,00	2,00	2,00
Range of emitter power due to temperature	[db]	0,00	0,00	0,00	0,00
Min emitter power (max 0db)	[db]	-5,90	-8,10	-10,30	-11,40
LED current	[mA]	13,90	8,62	5,34	4,21
Resistor at 3V	[kOhm]	0,216	0,348	0,562	0,713

4.1.5 Probleme und Lösungen

Störungen beim Abschalten der PSO-Karte
Im Testbetrieb zeigte sich, dass beim Abschalten des Überrahmens die PSO-Karte für einige ms periodische Signale sendet, die auf den DAC-Karten zu Zufallswerten führen. Eine genauere Untersuchung ergab, dass offenbar der 74LS245 als Treiber für die LWL-Sender diese Oszillationen erzeugt.

Wegen dieser Oszillationen beim Abschalten war die neue PSO-Karte zum Betrieb von Hofmann-DACs (Mode = old), obwohl dafür geplant, nicht geeignet. Deshalb wurde der LS245 probehalber ersetzt durch ein Piggy-Board mit BC556 Transistoren:

```
LS245 IN  --> BC556-Basis
LS245 OUT --> BC556-Emitter
LS245 VCC --> BC556-Collector
```

Diese Schaltung vermeidet die Oszillationen, hat aber flachere Flanken als ein LS245. Mit den neuen DAC-Boards ist das kein Problem. Aber die DAC-Karten in der Spannungsregelung von W. Arnold haben eine heftige, periodische Störung, die bei den LWL-Empfängern zu Einbrüchen in den Flanken und damit zu Übertragungsfehlern führt.

Die verschiedenen 245 Typen zeigen unterschiedliches Verhalten beim Abschalten:

```
Fairchild, Motorola Typen:
Typ         Oszillationen    Pause    Nachleuchten
LS245    120 kHz / 1.5 ms    8 ms      4 ms
ALS245   2-3 MHz / 200 us     --       4 ms
HC245     70 kHz / 2.3 ms     --      23 ms
HCT245    70 kHz /   3 ms     --      20 ms
Hingegen keine Oszillationen bei Texas Instrument Typen:
LS245, LS645, ALS245
```

Alle PSO-Karten sind jetzt mit SN74ALS245AN bestückt.

Die Oszillationen beim Abschalten waren der Anlass, das Übertragungs-Protokoll völlig zu überarbeiten. Nachdem zunächst versucht wurde, auch im Mode = new möglichst das

Hofmann-Protokoll zu verwenden, hat es nun mit dem ursprünglichen Hofmann-Protokoll keine Verwandschaft mehr:
Zuerst wird ein 15 Bit langes festes Muster (101010101010101) als Header gesendet, auf das sich die DAC-Karte synchronisiert. Dann folgen nahtlos die 16/32 Daten-Bits. Damit konnten die Oszillationen beim Abschalten sehr sicher erkannt werden da die Bit-Frequenz (2.5Mhz, 400ns) sich wesentlich von der Störung unterscheidet. Eine vollständige 16 Bit Übertragung braucht ca. 13us und ist damit ca. vier mal schneller als das Hofmann-Protokoll.

Danach zeigte sich, dass die PSO-Karte beim Abschalten noch zum Senden eines regulären Datenwortes angeregt wird mit der Info 0xffff, was bei den DACs zu 10V Ausgang führt.
Mit der PSO2Vx Firmware konnte dies nur dadurch verhindert werden, dass die PSO-Karte 0xffff niemals senden darf. Der letzte 0.15mV Schritt kann also nicht eingestellt werden.
Mit der PSO32 Firmware wird dies verhindert durch das Pruefen des MSB auf 0 im Kommando.

4.2 Serielle Ausgabe (G. Hoffmann Protokoll)

4.2.1 Aufgabe, Funktion

Diese Parallel-Seriell-Wandlerkarte wurde von G. Hoffmann entworfen. Sie wird bei verschiedenen Experimenten zur Steuerung eingesetzt: z. B. e-Kanonen-Netzgeräte, Messbereichs-Einstellungen usw..

Ein 16 Bit Datenwort wird seriell gewandelt und auf 50 Ohm Koax-Leitung ausgegeben. Die serielle Ausgabe wiederholt sich beständig. Sie wird abgebrochen und neu gestartet, wenn ein neues 16 Bit Datenwort ausgegeben wird. Dies schein aber nicht bei allen Karten reibungslos zu funktionieren (s.u.). Die Karten können, dem Problem angepasst, mit zwei unterschiedlichen internen Takten betrieben werden (32.768kHz, 2.4576MHz) und die Bits invertiert oder nicht invertiert senden.

Von der Parallel-Seriell-Wandlerkarte existieren zur Zeit mehrere unterschiedliche Layouts (Schaltungsunterlagen im 017-Ordner):

Layout 0 (20/88):
???

Layout 1 (26/88):
modifiziertes Layout 0, davon gibt es möglicherweise nur ein Exemplar;
interner Takt fest, nur mit einem Quarz bestückt;
Lötbrücken zum Umpolen der Ausgänge.

Layout 2 (26/88):
verbessertes Layout 1, häufiger im Einsatz;
interner Takt fest, nur mit einem Quarz bestückt;
manche Exemplare machen Probleme (s.h. 017-Ordner), wenn die Ausgabe eines Datenwortes schneller erfolgt als die Dauer des seriellen Signals (4.395ms bei 32.768kHz), sie senden überwiegend Schrott;
Lötbrücken zum Umpolen der Ausgänge.

Layout 3 (29.08.94; 13/96):
neues Layout;
zwei Quarze für 32.768kHz und 2.4576MHz interne Takte, wählbar über Steckbrücke;
Steckbrücke zum Umpolen der Ausgänge.

Die Serielle Ausgabe belegt eine Registeradresse des Address-Decoders (Siehe Abschnitt 3.2 [Address-Decoder], Seite 5.).

4.2.2 Bedienelemente

```
Layout 0:
???

Layout 1:
LED "Write"   :  Anzeige einer Schreiboperation
BNC-Buchsen   :  serielles Ausgangssignal, nicht kurzschlussfest
2 Brücken     :  Polarität des Ausgangssignals

Layout 2:
LED "Write"   :  Anzeige einer Schreiboperation
BNC-Buchsen   :  serielles Ausgangssignal, nicht kurzschlussfest
```

```
2 Brücken    :  Polarität des Ausgangssignals
```

Layout 3:

```
LED "Busy"   :  Anzeige einer Schreiboperation
LED "2MHz"   :  Anzeige 2.4576MHz Betrieb
LED "32kHz"  :  Anzeige 32.768kHz Betrieb
LED "Q"      :  Ausgangssignal normal
LED "Q\"     :  Ausgangssignal invertiert
BNC-Buchsen  :  serielles Ausgangssignal, nicht kurzschlussfest
Brücke "Q/Q\" :  Polarität des Ausgangssignals
Brücke "2.4576MHz/32.768kHz": Taktwahl
```

4.2.3 Programmierung

```
Register 0 out:       auszugebendes Datenwort (16 Bits)
```

4.2.4 Asynchrones, serielles Protokoll von G. Hoffmann

- es werden TTL-Pegel gesendet
- das niederwertigste Bit wird zuerst gesendet
- ein Bit ist 8 interne Takte lang
- Bit =0: 2 Takte "high"-Pegel, 6 Takte "low"-Pegel
- Bit =1: 5 Takte "high"-Pegel, 3 Takte "low"-Pegel
- Pause nach der Übertragung eines Datenwortes: 16 Takte "low"-Pegel

4.3 Serielle Eingabe (G. Hoffmann Protokoll)

4.3.1 Aufgabe, Funktion

Diese Seriell-Parallel-Wandlerkarte wurde von G. Hoffmann entworfen. Sie wird bei verschiedenen Experimenten zur Statuseingabe eingesetzt: z. B. Messbereichs-Einstellungen der Ionenstrom-Konverter usw..

Ein seriell über 50 Ohm Koax-Leitung eingegebenes Signal wird parallel in ein 16 Bit Datenwort gewandelt. Die Karten können, dem Problem angepasst, mit zwei unterschiedlichen internen Takten betrieben werden (32.768kHz, 2.4576MHz). Dazu muss der Quarz ausgetauscht werden.

Von der Seriell-Parallel-Wandlerkarte existieren zur Zeit mehrere unterschiedliche Layouts (11/88, 19/88, 3/89).

4.3.2 Bedienelemente

Layout 3/89:

```
LED "Read" :  Anzeige einer Leseoperation
BNC-Buchse :  serielles Eingangssignal
              (TTL, G. Hoffmann Protokoll, 50 Ohm Abschluss)
```

4.3.3 Programmierung

```
Register 0 in:       Eingabe des Datenwort (16 Bits)
```

4.4 Parallele Ausgabe (16 Bits)

4.4.1 Aufgabe, Funktion

Diese Karte für Parallele Ausgabe wurde von G. Hoffmann entworfen. Sie wird bei Scan-Experimenten in den Labors 017 und SI zur Ausgabe der aktuellen Kanalnummer eingesetzt.

Ein 16 Bit Datenwort wird parallel auf einen 20-poligen Flachbandstecker ausgegeben. Für diese Karte gibt es kein eigenes Layout sondern es wird ein modifiziertes 26/88-Layout des Parallel-Seriell-Wandlers verwendet (handverdrahtet). Ein Redesign wäre gelegentlich angebracht.

Die Parallele Ausgabe belegt eine Registeradresse des Address-Decoders (Siehe Abschnitt 3.2 [Address-Decoder], Seite 5.).

4.4.2 Bedienelemente

```
LED "Write"              :  Anzeige einer Schreiboperation

20-pol. Flachbandstecker:  parallele Ausgangssignale (16 Bits, TTL)
        Bit     Pin        Bit     Pin
         00      17         08      04
         01      15         09      06
         02      13         10      08
         03      11         11      10
         04      09         12      12
         05      07         13      14
         06      05         14      16
         07      03         15      18
     Ground     1,2     Ground    19,20
```

4.4.3 Programmierung

```
Register 0 out:       auszugebendes Datenwort (16 Bits)
```

4.5 Messintervall Timer

4.5.1 Aufgabe, Funktion

Diese Timer-Karte wurde von G. Hoffmann entworfen. Sie wird bei Scan-Experimenten in den Labors 017 und SI zur Programmierung der Mess- und Pausen-Intervalle der einzelnen Scan-Schritte eingesetzt.

- Es können Intervalle von 1 us bis ca. 1 Woche generiert werden.
- Als Takt wird entweder der interne 1 MHz Quarztakt des Routing oder ein externer Takt verwendet.
- Der Start des Zeitintervalls erfolgt wahlweise programmiert oder über ein externes Startsignal.
- Das Timer-Register muss nach jedem Ablauf eines Intervalls neu geladen werden!
- Wegen der nur 8 Bits langen Mantisse können die tatsächlichen Intervalllängen von den geplanten um bis zu 0.4% abweichen!

Zur Zeit existieren 2 handverdrahtete Karten, die noch im Einsatz sind, und einige Eagle-geroutete Karten.

Der Messintervall Timer belegt eine Registeradresse des Address-Decoders (Siehe Abschnitt 3.2 [Address-Decoder], Seite 5.).

4.5.2 Bedienelemente

```
LED "Write"             :  Anzeige einer Schreiboperation
LED "Intervall"         :  Anzeige Messintervall aktiv
BNC-Buchse "Intervall"  :  Messintervall Ausgang, TTL, pos. Logik
BNC-Buchse "Intervall\" :  Messintervall Ausgang, TTL, neg. Logik
BNC-Buchse "ext. Takt"  :  externer Takt Eingang, TTL
BNC-Buchse "ext. Start" :  externer Start Eingang, TTL, pos. Logik
```

4.5.3 Programmierung

```
Register 0 out:       Timer Datenwort (16 Bits)
        Bit 15        1 = externer Takt
        Bit 14        1 = nur externer Start
        Bits (12..8)  Exponent E = 0...31
        Bits (7..0)   Mantisse M = 0...255

Zeitintervall = M * 2^E * T
        T (Taktperiode) = 1 us für internen Takt

Berechnung von M und E für ein Intervall mit x Takten:
        ( ld(x) = ld(10) * lg(x) = 1 / lg(2) * lg(x) )
        E = MAX(0, INT( ld(x)) -7)
        M = INT(x / 2^E)        (max .8% Abweichung)
        M = INT(x / 2^E + 0.5)  (max .4% Abweichung)

Beispiele für internen 1 MHz Takt:
```

Intervall	M	E	Datenwort
1us (1 us)	1	0	0x0001
10us (10 us)	10	0	0x000a
100us (100 us)	100	0	0x0064
1ms (1 ms)	250	2	0x02fa
10ms (9.984 ms)	156	6	0x069c
100ms (99.84 ms)	195	9	0x09c3
1s (0.999424 s)	244	12	0x0cf4
10s (10.02701 s)	153	16	0x1099

Interner/externer Start:

Bit 14	ext Start	Write	Intervall-Start
0	H	ja	ja
0	L	ja	nein
0	L->H	--	ja
1	L	--	nein
1	H	--	nein
1	L->H	--	ja

4.6 Programmierbare Zeitbasis

4.6.1 Aufgabe, Funktion

Diese Zeitbasis-Karte wurde von G. Hoffmann entworfen. Sie wird in den Labors 017 und SI zur Registrierung der Messzeit pro Kanal eingesetzt.

Sie stellt den 2^n-fach untersetzten (n=0..15, programmierbar) internen 1 MHz Quarztakt am Ausgang zur Verfügung.

Es existieren verschiedene Layouts dieser Karte: 36/88, 2/89, 15/97

Die Programmierbare Zeitbasis belegt eine Registeradresse des Address-Decoders (Siehe Abschnitt 3.2 [Address-Decoder], Seite 5.).

4.6.2 Bedienelemente

```
LED "Write"       :  Anzeige einer Schreiboperation
LEDs "8 4 2 1"    :  Anzeige von "n" der 2^n-Taktuntersetzung
BNC-Buchse "Out"  :  untersetzter Zeittaktausgang, TTL
```

4.6.3 Programmierung

```
Register 0 out:       Zeitbasis Datenwort (16 Bits)
        Bits (3..0)   n=0..15 für Untersetzung 1/2^n des 1MHz Taktes
```

4.7 Interrupt Eingabe

4.7.1 Aufgabe, Funktion

Diese Karte wurde von G. Hoffmann entworfen. Sie wird in den Labors 017 und SI bei den Scan-Experimenten zur Synchronisierung des Experimentablaufs durch Interrupt eingesetzt.

Das Experiment startet die Karte mit dem Startimpuls (Start-Eingang). Gleichzeitig wird der Busy-In-Eingang mit der Summe aller relevanter Busy-Signale versorgt. Erst wenn das Summen-Busy verschwunden ist wird die Karte aktiv und gibt sowohl einen Interrupt-Wunsch an den Rechner ein, als auch ein Steuersignal (Read/Reset) an das Experiment aus.

Je nach Konfiguration (Steckbrücke J2) gibt die Karte nur einen Interrupt-Impuls an den Rechner oder sie speichert den Interrupt-Wunsch bis er durch eine Leseoperation gelöscht wird. Letztere Konfiguration wird benötigt, wenn mehrere Interrupt-Quellen existieren und durch Lesen der zugehörigen Register herausgefunden werden muss, welcher Interrupt aktiv war. Ein Interrupt steht an und wird gelöscht, wenn beim Lesen das Daten-Bit 0 gesetzt ist.

Es existieren verschiedene Layouts dieser Karte: 01.08.94, 11/97

Die Interrupt-Karte belegt eine Registeradresse des Address-Decoders (Siehe Abschnitt 3.2 [Address-Decoder], Seite 5.). Sie kann jedoch im IT-Puls-Modus ohne Address-Decoder betrieben werden.

4.7.2 Bedienelemente

```
LED "Select"           : Anzeige, dass die Karte adressiert ist
LED "Interrupt"        : Anzeige für einen anstehenden Interrupt
LED "Busy"             : Anzeige des Busy-Signals vom Experiment
BNC-Buchse "Start"     : Eingabe des Startsignals, TTL, pos. Logik
BNC-Buchse "Read/Reset": Ausgabe eines Steuersignals für das Experiment,
                         TTL, 100ns, pos. Logik
BNC-Buchse "Busy\ in"  : Eingabe des Busy-Signals vom Experiment,
                         TTL, neg. Logik
BNC-Buchse "Busy out"  : Ausgabe des Busy-Signals vom Experiment,
                         TTL, pos. Logik
Steckbrücke J2/1-2     : Die IT-Karte meldet mit einem IT-Puls einen
                         Interrupt bei der Routing-Steuerung an. Die
                         IT-Karte benötigt dann keine Address-Decoder-
                         Karte und der IT kann nicht gelesen werden.
Steckbrücke J2/2-3     : Die IT-Anmeldung wird auf der IT-Karte
                         gespeichert und solange an die Routing-Steuerung
                         gemeldet, bis er durch einen Read auf Register 0
                         (Bit 0) gelesen und gelöscht wird.
```

4.7.3 Programmierung

```
Register 0 in:  Lesen und Löschen des Interrupts
        Bit 0   1 = es steht ein Interrupt an
```

4.8 Programmierbare Totzeit

4.8.1 Aufgabe

Gelegentlich kann die exakte Totzeit einer Apparatur nicht bestimmt werden, so dass notwendige Korrekturrechnungen nicht möglich sind. In diesem Falle empfiehlt sich das Vorschalten einer gut bekannten künstlichen Totzeit, die über alle anderen Totzeiten der Apparatur dominiert, und deshalb als einzige in die Korrektur eingeht.

Manche Detektoren (z.B. Channeltrons) neigen dazu, nach einem registrierten Ereignis weitere Impulse abzugeben, durch die eine Messung erheblich verfälscht werden kann. Durch eine Totzeit können solche Nachimpulse unterdrückt werden. Dabei ist jedoch zu beachten, dass eine nicht-paralysierende Totzeit wie man sie z.B. mit einem Pulse-Strecher realisieren könnte in einem solchen Falle nicht ausreichend ist.

Beachten Sie dazu den Abschnitt ⟨undefined⟩ [Anwendungen], Seite ⟨undefined⟩.

Zur Untersuchung des Ereignisstromes auf Totzeiten, Nachimpulse usw. existiert eine Interface-Karte zum Messen der Abstandsverteilung von Impulsen, zu der es eine Anleitung in der Data-Routing-Funktionsbeschreibung gibt: HTML-Version (`http://www.strz.uni-giessen.de/ExpHelp/datarout/datarout.html`), PDF-Version (`http://www.strz.uni-giessen.de/ExpHelp/datarout/datarout.pdf`).

4.8.2 Funktion

Die Totzeitkarte kann sowohl über das Control-Routing als auch manuell (Jumper) programmiert werden in Schritten der Taktperiode. Für die zur Zeit zur Verfügung stehenden Logikbausteine sind Takte bis zu 40 MHz möglich. Standard ist 20 MHz. Es genügt nicht, den Quarzoszillator auszutauschen, der Logikbaustein muss auch umprogrammiert werden, da dieser sonst einen falschen Status meldet. Bei manueller Programmierung spielt dies keine Rolle, die Software würde aber u.U. eine falsche Totzeit einstellen.

Die Totzeitkarte liefert wahlweise eine paralysierende oder nicht-paralysierende Totzeit.

- *paralysierende Totzeit:*
  Ereignisse, die in die Totzeit eines Vorgängers fallen, gehen verloren und starten die Totzeit neu.
- *nicht-paralysierende Totzeit:*
  Ereignisse, die in die Totzeit eines Vorgängers fallen, gehen verloren, ohne Auswirkung auf die Totzeit.

Da die Totzeitkarte digital mit einem Quarztakt T arbeitet ist die Totzeit auf +-T/2 unscharf im Mittel jedoch recht genau (besser 10**-3) (Siehe Abschnitt 4.8.6.1 [Mittlere Totzeit], Seite 25.).

Die Totzeitkarte belegt drei Registeradressen des Address-Decoders (Siehe Abschnitt 3.2 [Address-Decoder], Seite 5.).

4.8.3 Bedienelemente

LED "Select":	leuchtet, wenn eines der drei Register adressiert wird
BNC-Buchse "In":	Signaleingang (TTL, pos. Logik)
BNC-Buchse "Out":	Signalausgang (TTL, pos. Logik)
BNC-Buchse "Lost":	Ausgang für Totzeitverluste (TTL, pos. Logik)

BNC-Buchse "Deadtime":	Ausgang für Totzeitsignal (TTL, pos. Logik)
manuelle Programmierung:	Siehe Abschnitt 4.8.4 [Programmierung], Seite 23.

4.8.4 Programmierung

Control-Routing Programmierung:
(dazu muss Jumper J15 entfernt werden!)

```
Register 0 out:       Command
        out 0x4000 -> paralysierende Totzeit
        out 0x0000 -> nicht-paralysierende Totzeit

Register 0 in:        Status
        Bit 14        1 = paralysierende Totzeit
        Bits (11..4)  Taktfrequenz [MHz]
        Bits (3..0)   Anzahl der Timer Bits (TBITS)

Register 1 out:       Ausgangs-Puls-Länge
        Bits (TBITS-1..0)  Länge Lp in Taktperioden [T]
             totale mittlere Pulslänge = (Lp + 2.5) * T

Register 2 out:       Länge der Totzeit
        Bits (TBITS-1..0)  Länge Ld in Taktperioden [T]
             totale mittlere Totzeitlänge = (Ld + 3.5) * T
```

Jumper Programmierung:
(x = Jumper set, - = Jumper not set)

```
J15        x    manuelle (Jumper) Programmierung
J14        x    paralysierende Totzeit
           -    nicht-paralysierende Totzeit
J(13..0)   x    Länge der Totzeit Ld in Taktperioden [T]
                (z.Z. nur J(9..0) belegt!!! ISP-Chip voll)
                    totale mittlere Totzeit = (Ld + 3.5) * T
                    totale mittlere Pulslänge = (Ld/2 + 2.5) * T
```

Pulslänge und Totzeit sind um +-0.5 T unscharf!
(Siehe Abschnitt 4.8.6.1 [Mittlere Totzeit], Seite 25.)

4.8.5 Messungen

Der programmierbare Mittelwert der Totzeit ist durch Signallaufzeiten fehlerbehaftet. Der genaue Wert kann mit Doppelpulsgenerator, Oszillograph und passendem Zeitnormal bestimmt werden:

Messung der tatsächlichen Totzeit mit einem Doppelpulser (Dez. 2001)
Das Scope wurde mit dem Quarz auf dem Deadtime-Board geeicht.

10 MHz Clock

```
Logikbaustein : ispLSI1032E-70
```

```
Clock         : 10 MHz
Doppelpulsrate: 15.3 kHz

          Doppelpulsabstand       (Out - In/2) / In/2
        1       300ns                    .000
        2       320ns                    .130
        3       340ns                    .316
        4       360ns                    .518
        5       380ns                    .718
        6       400ns                    .904
        7       420ns                   1.00

Fit (Origin) Werte 2 - 6 mit y = a0 + a1 * x
a) a0 = -3.0828 +- .00413
a1 = .01 fest
y = .5  =>  x = 358.3 +- .4
b) a0 = -2.9928 +- .03141
a1 = .00975 +- .00009
y = .5  =>  x = 358.3
```

Totzeit= n * T + 3.5 * T + 8.3ns; T = 100ns; n = prog. Zeit

20 MHz Clock

```
Logikbaustein : ispLSI1032E-70
Clock         : 20 MHz
Doppelpulsrate: 15.3 kHz

(unter 200ns löste der In-Zähler (U/D-Zähler) die Doppelpulse
nur noch teilweise auf, weshalb diese Werte zu hoch ausfallen müssen!)
          Doppelpulsabstand       (Out - In/2) / In/2
        1       150ns                    .000
        2       160ns                    .045
        3       170ns                    .166
        4       180ns                    .341
        5       190ns                    .572
        6       200ns                    .805
        7       210ns                   1.00

Fit (Origin) Werte 2 - 6 mit y = a0 + a1 * x
a) a0 = -3.2142 +- .01798
a1 = .02 fest
y = .5  =>  x = 185.7 +- .9
b) a0 = -3.081 +- .25353
a1 = .01926 +- .0014
y = .5  =>  x = 185.9
Fit (Origin) Werte 2 - 6 mit y = a0 + a1 * x + a2 * x**2
```

```
a) a0 = 3.359 +- 1.08274
a1 = -.05274 +- .01209
a2 = .0002 +- .00003
y = .5  =>  x = 187.4
```

Totzeit= n * T + 3.5 * T + 10.7ns; T = 50ns; n = prog. Zeit

4.8.6 Totzeit-Rechnungen

siehe auch Zählratenstatistik:
http://www.strz.uni-giessen.de/ExpHelp/statistik/statistik.html
http://www.strz.uni-giessen.de/ExpHelp/statistik/statistik.pdf

4.8.6.1 Mittlere Totzeit

Da die Totzeitkarte digital mit einem Quarztakt T arbeitet ist die Totzeit auf +-T/2 unscharf. Der Mittelwert liegt etwa bei 0.5T und ist Zählraten abhängig, da eine vorausgehende Totzeit immer mit einem Clock-Tick endet und der Abstand zweier aufeinanderfolgender Pulse eine R * exp(-Rt) -Verteilung hat.

Rechnung für das erste Intervall nach einer Totzeit:

```
Zeit:               t
Clock-Periode:      T
Ereignisrate:       R
Abstandsverteilung: D(t) = R * exp(-Rt)

Totzeitmittel:
        M = Int[0-T]((T - t) * D(t))dt / Int[0-T](D(t))dt
            Int[0-T](T * D(t))dt    = T * (1 - exp(-RT))
            Int[0-T](-t * D(t))dt   = -T / RT * (1 - (RT + 1) * exp(-RT))
            Int[0-T](D(t))dt        = 1 - exp(-RT)
        M = T * {1 - [1 - (RT + 1) * exp(-RT)] / [RT * (1 - exp(-RT))]}
RT -> 0: M -> 1/2 * T
RT -> unendl.: M -> 1 * T
RT << 1.0: M ~ (.5 + R*T/12) * T
FM = (M - .5*T)/.5*T = R*T/6
RT = .01: M = .500833 * T
RT = .1: M = .50833 * T
RT = 1.0: M = .5820 * T
```

Die Korrekturen sind vernachlässigbar;
z.B. 10MHz Clock; 1MHz Rate: 0.8ns Abweichung.

Da die Abstandsverteilung der Pulse eine exp(-Rt) - Verteilung ist, errechnet sich für M bei Eintreffen des nächstfolgenden Pulses in eine der nachfolgenden Clock-Perioden n der gleiche Wert:

```
D'(t) = R * exp(-R(t+nT)) = D(t) * exp(-RnT)
M = Int[0-T]((T - t) * D(t))dt / Int[0-T](D(t))dt
```

Anm. zur Abstandsverteilung von zufälligen (Poisson-verteilten) Ereignissen:
Man erhält sie als negative Änderung der Wahrscheinlichkeit, dass von 0 bis t kein Ereignis eintrifft: ???

```
D(t) = -d(exp(-Rt) / dt = R * exp(-Rt)
```

4.8.6.2 Korrekturformeln für nicht-paralysierende Totzeit

Die Totzeit-Korrekturformeln für konstante, nicht-paralysierende Totzeit sind:

```
Zeit:            t
Clock-Periode:   T
In-Rate:         R
Out-Rate:        r
Totzeitverluste: v
Totzeit:         Z

    v = r * R * Z
    R = r + v = r / (1 - r*Z)
    r = R / (1 + R*Z)
```

Für die unscharfe Totzeit, wie sie die Totzeitkarte liefert, ist eine Mittelung über die auftretenden Totzeiten nötig. Es werden gleichverteilte Totzeiten angenommen:

```
        r' = int[Z1,Z2](r)dZ / int[Z1,Z2]dZ
                int[Z1,Z2](r)dZ = ln(1 + R*Z2) - ln(1 + R*Z1)
                int[Z1,Z2]dZ = T
        r' = 1/T * ln[(1 + R*Z2)/(1 + R*Z1)]        (Z1 = Z-T/2; Z2 = Z+T/2)
           = 1/T * ln[(1 + R*Z + R*T/2) / (1 + R*Z - R*T/2)]
           = 1/T * ln[(1 + R*T/2/(1 + R*Z)) / (1 - R*T/2/(1 + R*Z))]
           = 1/T * [ln(1 + r*T/2) - ln(1 - r*T/2)]
rT << 1:    r' ~ r * (1 + (r*T)**2 /12)
            Fr'= |(r'-r)/r| = (r*T)**2 /12 = (R*T/(1 + R*Z))**2 /12
rT -> 0:    r' -> r

    R' = (exp(r*T) - 1) / [(Z + T/2) - (Z - T/2)*exp(r*T)]
       = (1 - exp(-r*T)) / ((Z + T/2)*exp(-r*T) - (Z - T/2))
rT << 1:    R' ~ R/(1 - (r*T)**2 /12)
            FR'= |(R'-R)/R| = (r*T)**2 /12 = (R*T/(1 + R*Z))**2 /12
rT -> 0:    R' -> R
```

Die Korrekturen sind vernachlässigbar klein;
z.B. 10MHz Clock; Rate R=1MHz; Totzeit Z=1us: Fr' = FR' = .02%

4.8.6.3 Korrekturformeln für paralysierende Totzeit

Die Totzeit-Korrekturformeln für konstante, paralysierende Totzeit sind:

```
Zeit:            t
Clock-Periode:   T
In-Rate:         R
Out-Rate:        r
Totzeitverluste: v
```

```
Totzeit:        Z

    alle mit kürzerem Abstand als Z gehen verloren
    v = R * Int[0,Z](R*exp(-R*Z))
      = R * (1 - exp(-R*Z))
    r = R - v = R * exp(-R*Z)
    R = Umkehrung nur numerisch lösbar und doppeldeutig!

numerische Lösung mit Newton Näherung für R*T < .99:
    R = r * (1 + r*Z); /* first approximation */
    for(i=0; i<10; i++) {
        f = R * exp(-R*Z) - r;
        d = exp(-R*Z) * (1 - R*Z);
        R = R - f / d;            /* Newton */
    }
```

Für die unscharfe Totzeit, wie sie die Totzeitkarte liefert, ist eine Mittelung über die auftretenden Totzeiten nötig. Es werden gleichverteilte Totzeiten angenommen:

```
    v' = int[Z1,Z2](v)dZ / int[Z1,Z2]dZ
           int[Z1,Z2](v)dZ = R*(Z2-Z1) + exp(-Z2*R) - exp(-Z1*R)
           int[Z1,Z2]dZ = T        (Z1 = Z-T/2; Z2 = Z+T/2)
    v' = R*[1- 1/(R*T) * exp(-R*Z) * (exp(R*T/2) - exp(-R*T/2))]
    r' = R - v' = R * exp(-R*Z) / (R*T) * (exp(R*T/2) - exp(-R*T/2))
       = r / (R*T) * (exp(R*T/2) - exp(-R*T/2))
    Fr'= |(r'-r)/r| =
       = 1/(R*T) * (exp(R*T/2) - exp(-R*T/2)) - 1
RT << 1:   Fr' ~ (R*T)**2 /24
```

Die Korrekturen sind vernachlässigbar klein;
z.B. 10MHz Clock; Rate R=1MHz; Totzeit Z=1us: Fr' = .04%

```
    R' = Umkehrung nur numerisch lösbar und doppeldeutig!
numerische Lösung mit Newton Näherung für R*T < .99:
    R = r * (1 + r*Z);        /* first approximation */
    for(i=0; i<10; i++) {
        f = 1/T * exp(-R*Z) *(exp(R*T/2.) - exp(-R*T/2.)) - r;
        d = exp(-R*Z) * (-Z/T*(exp(R*T/2.) - exp(-R*T/2.))
                        + 1/2.*(exp(R*T/2.) + exp(-R*T/2.)));
        R = R - f / d;        /* Newton */
    }
```

4.8.7 Anwendungen

4.8.7.1 Dominierende Totzeit

Gelegentlich sind Totzeiverluste nicht berechenbar da z.B. die Totzeit nicht bekannt oder veränderlich ist. In einem solchen Fall kann mit der "Programmierbaren Totzeit" eine Totzeit erzeugt werden, die die vorausgehenden, unbekannten Totzeiten dominiert.

Hier soll untersucht werden, inwiefern eine dominierende Totzeit geeignet ist, unbekannte Totzeiten so zu überdecken, dass Totzeitkorrekturrechnungen möglich werden.

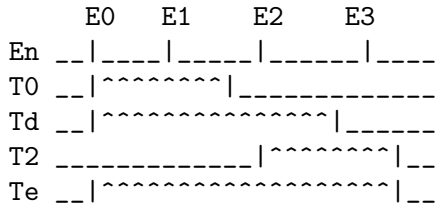
Wie die folgenden Überlegungen zeigen, kann eine solche dominierende Totzeit nur näherungsweise funktionieren. Die Rechnungen zeigen zeigen in welcher Größenordnung Abweichungen zu erwarten sind und wie man sie eventuell minimieren kann.

Wie Simulationen gezeigt haben, sind die Abweichungen für eine nicht-paralysierende dominierende Totzeit deutlich kleiner als für eine paralysierende.

Nicht-paralysierende dominierende Totzeit

Die primäre Totzeit wird ebenfalls als nicht-paralysierend und konstant vorausgesetzt.

Für nicht-paralysierende Totzeiten ergibt sich folgendes Verhalten:

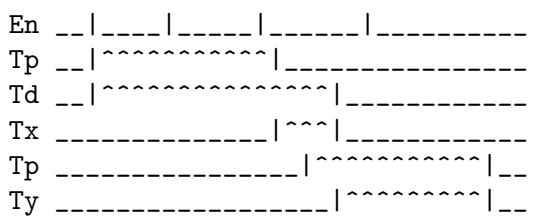


- Ein erstes Ereignis E0 hat eine primäre Totzeit T0 und löst die dominierende Totzeit Td aus.
- Das folgende Ereignis E1 geht bereits durch die primäre Totzeit T0 verloren und liefert keinen Beitrag zur Totzeit.
- Das Ereignis E2 wird durch die dominierende Totzeit Td verworfen verlängert aber die effektive Totzeit Te.
- Das Ereignis E3 hingegen fällt in die Totzeit von E2 und geht damit ebenfalls verloren.

Die Totzeitverluste können daher größer sein als die dominierende Totzeit erwarten lässt. Für T0=0 oder Td=T0 ist die Totzeitverlustrechnung Td exakt.

Rechnung

```
R:  Eingangsrate vor allen Totzeiten
r:  Ausgangsrate nach den Totzeiten
r': Ausgangsrate bei T0 = 0
Tp: primäre Totzeit
Td: dominierende Totzeit (nicht-paralysierend Vd=R*Td)
Vp: zusätzliche Totzeitverluste durch Tp bei Tp < Td
Vd: Totzeitverluste durch Td bei Tp = 0
```



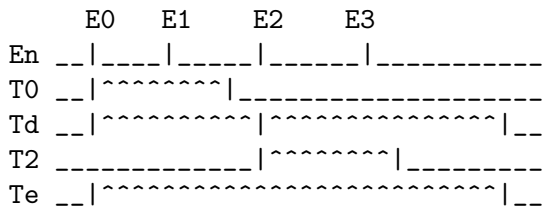

```
Tx = Td - Tp
```

```
Ty = Tp - Tx/2
Vy = R*Ty * R*Tx = R^2 * (Td-Tp) * (3Tp-Td)/2
  für Td = Tp   --> Vy = 0
  für Td = 2Tp  --> Vy = 1/2 * (R*Tp)^2
```

Paralysierende dominierende Totzeit
Für eine nicht-paralysierende primäre Totzeit und eine paralysierende dominierende Totzeit ergibt sich folgendes Verhalten:

- Ein erstes Ereignis E0 hat eine primäre Totzeit T0 und löst die dominierende Totzeit Td aus.
- Das folgende Ereignis E1 geht bereits durch die primäre Totzeit T0 verloren und hat deshalb keine weitere dominierende Totzeit zur Folge.
- Das Ereignis E2 wird durch die dominierende Totzeit Td verworfen löst aber eine neue dominierende Totzeit Td aus.
- Das Ereignis E3 hingegen fällt in die Totzeit von E2 und geht damit ebenfalls verloren ohne Td neu zu starten.

Durch die primären Verluste von E1 und E3 wird die effektive Totzeit Td zu kurz. Für T0->0 verschwindet dieser Verlust.



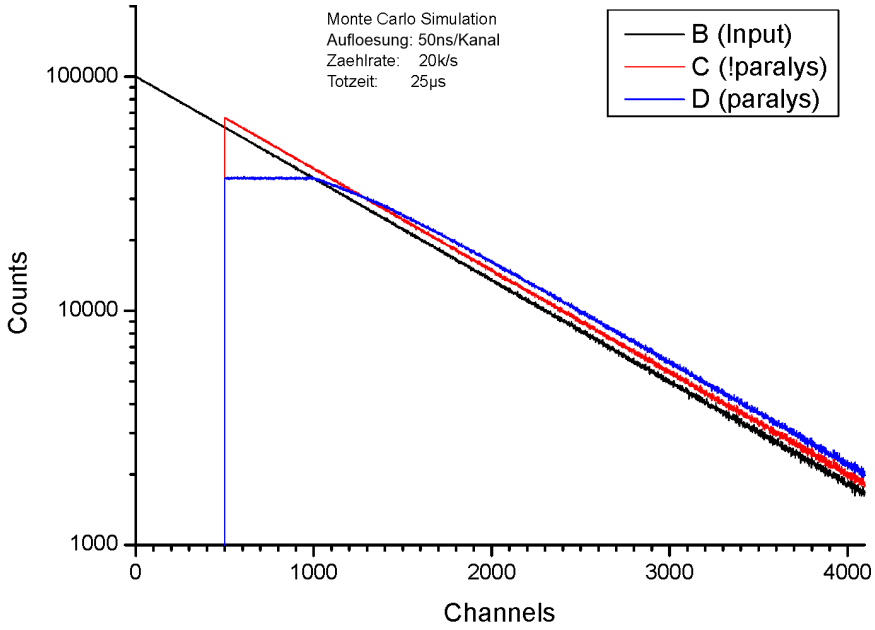
Rechnung schwierig, noch ungelöst....

Simulationsrechnungen
Die folgenden Grafiken sind Ergebnisse von Monte-Carlo-Simulationen, da für echte Messungen keine ausreichend gute Poisson-verteilte Ereignisraten zur Verfügung standen. Der Versuch das Ganze mittels Faltungsintegralen zu berechnen ist ebenfalls an der Komplexität der Probleme gescheitert.

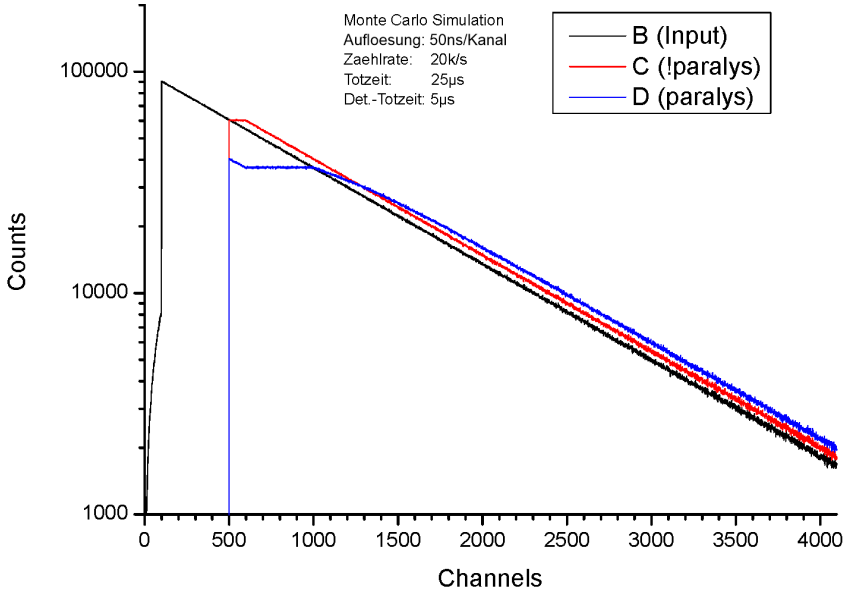
Sie zeigen die Abstandsverteilungen von Poisson-verteilten Ereignissen nach Durchlaufen von ein bzw. zwei Totzeiten. Wie man sieht haben Totzeiten einen markanten Einfluss auf die Abstandsverteilung und sind nicht nur ein Abschneiden von kurzen Abständen.

Die Abstandsverteilung von Poisson-verteilten Ereignissen ist eine abfallende Exponentialfunktion, in logarithmischer Darstellung also eine Gerade. Eine solche Verteilung lässt sich aus Zufallszahlen leicht herstellen, sodass eine Monte-Carlo-Rechnung ein bequemes

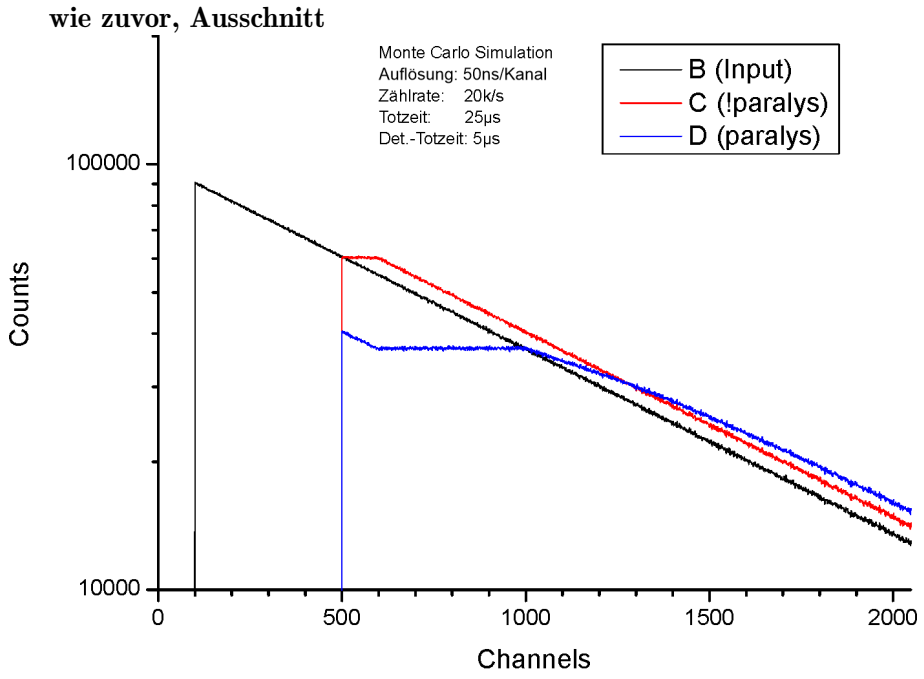
Werkzeug zur Untersuchung der Auswirkungen einer Totzeit bei den verschiedenen Anwendungen ist.



Abstandsverteilung nach einer Totzeit bei Poisson-verteilter Eingangsrate



Abstandsverteilung nach kurzer Totzeit gefolgt von langer Totzeit bei Poisson-verteilter Eingangsrate



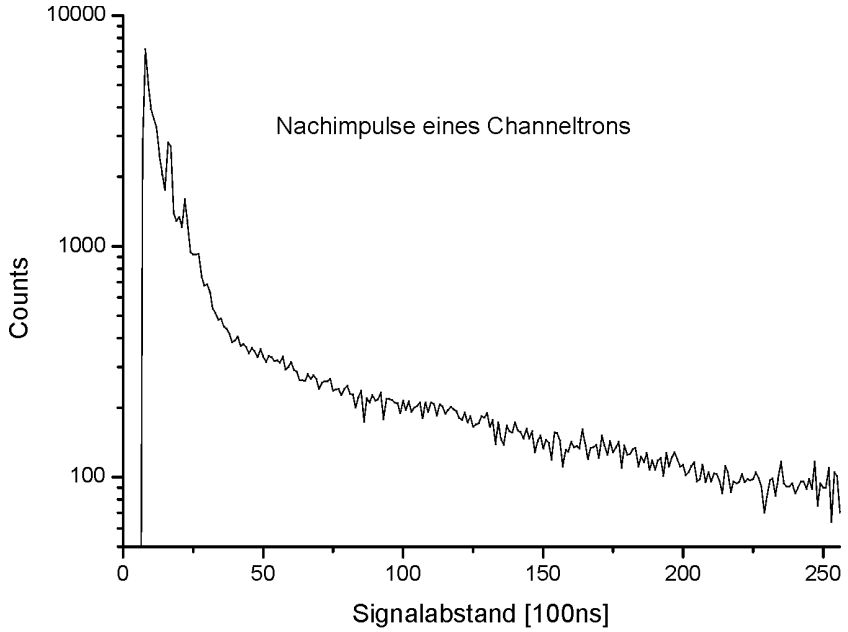
4.8.7.2 Beseitigen von Nachimpulsen

Siehe Zählratenstatisik Doku:
`http://www.strz.uni-giessen.de/ExpHelp/statistik/statistik.html`
`http://www.strz.uni-giessen.de/ExpHelp/statistik/statistik.pdf`

Wie man sich leicht überlegen kann, ist die nicht-paralysierende Totzeit nicht geeignet zum Abschneiden von z.B. Nachimpulsen eines Detektors, da nach wie vor beliebig kurze Ereignisabstände auftreten. Die paralysierende Totzeit hingegen entfernt alle Ereignisse, die zu ihrem Vorgänger einen kürzeren Abstand haben als die Totzeit.

4.8.7.3 Beseitigen von Nachimpulsen

Insbesondere Channeltrons haben die unschöne Eigenschaft nach einem echten Ereignis noch Nachimpulse zu liefern, die eine Wirkungsquerschnittmessung deutlich verfälschen können.



Totzeit (500ns) und Nachimpulse (bis ca. 4 us) eines Channeltrons

In dieser logarithmischen Darstellung der Ereignisabstände ist eine markannte Abweichung von der zu erwartenden Geraden durch die Poisson-verteilten Ereignisse zu erkennen. Mit einer geeigneten Totzeit kann der Bereich der Nachimpulse ausgeblendet werden.
Wie man sich leicht überlegen kann, ist die nicht-paralysierende Totzeit nicht dazu geeignet, da nach wie vor beliebig kurze Ereignisabstände auftreten können.
Die paralysierende Totzeit hingegen entfernt alle Ereignisse, die zu ihrem Vorgänger einen kürzeren Abstand haben als die Totzeit. Allerdings mit Nebenwirkungen, s.o.

Siehe auch:
`http://www.strz.uni-giessen.de/ExpHelp/datarout/datarout.html`
`http://www.strz.uni-giessen.de/ExpHelp/datarout/datarout.pdf`

4.9 CAN Controller

4.9.1 Aufgabe, Funktion

Der CAN-Bus (Controller Area Network) wurde von Bosch entwickelt zur Datenerfassung und Steuerung in Automobilen. Auf Grund der außerordentlichen Fehlertoleranz des CAN-Protokolls (Hamming Distanz = 6) sowie seiner Fähigkeit zur Selbstarbitrierung und seiner Multi-Master-Fähigkeit eignet er sich auch hervorragend zum Einsatz bei Experimenten und wird inzwischen gelegentlich auch dazu verwendet.
Details findet man z.B. unter:

```
http://www.can.bosch.com
http://www.mjschofield.com
```

Die CAN-Controller-Karte ist ein Interface zwischen dem Routing-Bus und dem CAN-Bus unter Verwendung des CC770 CAN-Controller-Bausteins von Bosch. Die Steuerung erfolgt über einen programierbaren ispLSI1032E Logikbaustein. Für die Initialisierungsdaten enthält die Karte einen EPROM.

Die CAN-Controller-Karte ist **nicht galvanisch getrennt** vom CAN-Bus, was zur Folge hat, dass alle am CAN-Bus angeschlossenen Einheiten eine solche galvanische Trennung haben sollten, falls sie nicht am gleichen 220V Netz angeschlossen sind und sich nicht räumlich dicht beieinander befinden.

Der CAN-Bus muss an beiden Enden mit 120 Ohm abgeschlossen sein. Auf der CAN-Controller-Karte kann mittels Jumper ein solcher Abschlusswiderstand aktiviert werden, was aber nur sinnvoll ist, wenn die Karte am Ende des Busses angeschlossen ist!

Nachdem die Karte mit Spannung versorgt ist, initialisieren sich zunächst die ispLSI1032E und CC770 Bausteine, danach werden die EPROM-Daten zum CC770 CAN-Controller übertragen. Das initialisieren des CC770 mit den EPROM-Daten kann auch jederzeit mit einen Init-Befehl durch die Software erfolgen. Danach kann die Software die vorausgegangene Initialisierung beliebig ändern, um eine passende Konfiguration zu erhalten. Die Initialisierung kann auch vollstdig durch Software erfolgen, ein EPROM muss dann nicht vorhanden sein.

Die CAN-Controller-Karte belegt zwei Registeradressen des Address-Decoders (Siehe Abschnitt 3.2 [Address-Decoder], Seite 5.).

4.9.2 Bedienelemente

LED "Select":	leuchtet, wenn eines der zwei Register adressiert wird
9-pol. DSUB-Stecker:	Anschlussstecker für den CAN-Bus
on Board Jumper:	Option für den CAN-Bus-Abschlusswiderstand

4.9.3 Programmierung

Überblick:

```
Register 0 out:    Control/Mode/Address output
    Bit 15 == 0        Control output
        Bit 14             Reset command
        Bit 13             Init command
    Bit 15 == 1        Mode/Address output
```

```
        Bits 11..08        Mode register
        Bits 07..00        Address register

Register 0 in:     Status input
        Bit 14             Reset command activ
        Bit 13             Init command activ
        Bit 07             Controller busy
        Bit 06             Interrupt flag
        Bit 03             Error flag

Register 1 in/out: Data Register
        Bits 07..00        8 bit data transfer
        Bits 15..00        16 bit data transfer (project)
```

Mode-Register:

```
                 0              1
x---     read data      write data    direction of data transfer
-x--    8 bit data    (16 bit data)   currently only 8 bit transfers
--x-    fixed addr      incr. addr    address auto increment
---x      not used        not used
```

Address-Register:

Der CC770 CAN-Controller von Bosch hat 256 adressierbare Register, deren Funktion in der CC770 Anleitung zu finden ist:
`http://www.can.bosch.com/docu/InternetSpecification_CC770.pdf` oder
`http://www.strz.uni-giessen.de/ExpHelp/cntlrout/InternetSpecification_CC770.pdf`.
Für einen Datentransfer mit einem dieser Register des CC770 ist zuerst das Adressregister der CAN-Controller-Karte mit der gewünschten CC770-Registeradresse zu laden. Im Modus Address-Inkrement wird das Adressregister nach jedem Datentransfer um eins erhöht, um fortlaufende Daten ohne neue Adressierung lesen zu können.

Command-Register:

- Reset command
  Löscht die Error-Flag
- Init command
  Veranlasst eine Initialisierung des CC770 CAN-Controllers durch Übertragen der EPROM-Daten zum CC770.

Status-Register:

- Reset command activ
- Init command activ
- Controller busy:
- Interrupt flag:
- Error flag:

4.10 8-Kanal 12-Bit ADC

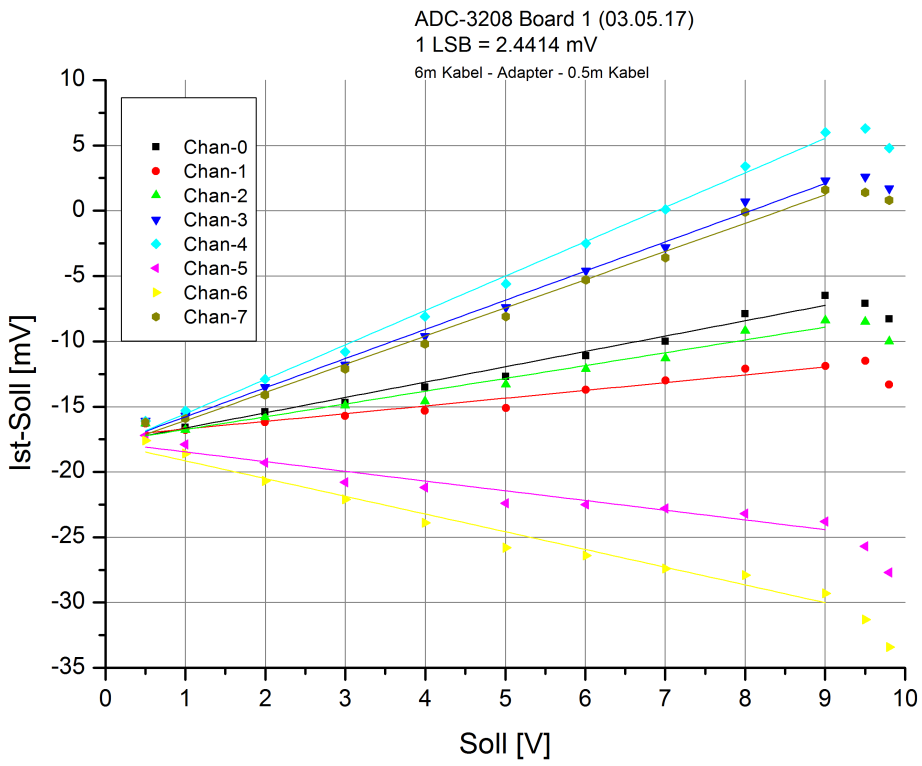
4.10.1 Aufgabe, Funktion

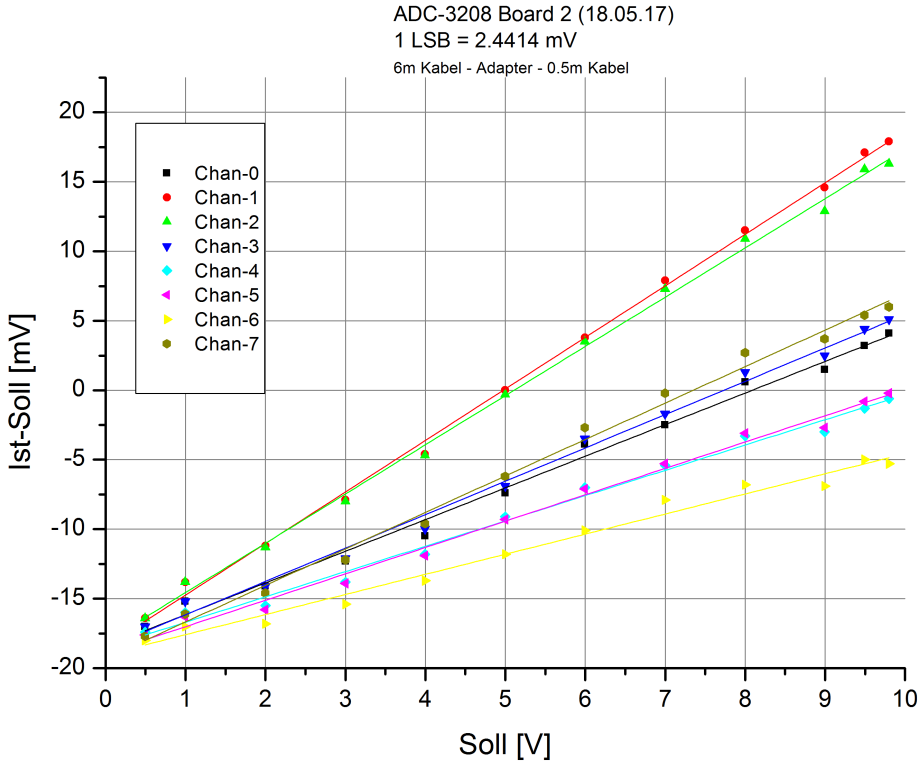
Die 8-Kanal 12-Bit ADC-Karte wurde entwickelt um die Plattenströme der neuen Elektronenkanone in Labor 017 zu überwachen.

- Die analogen Signale werden über einen 25-poligen SubD-Stecker angeschlossen. Die Steckerbelegung ist auf der Seite des Stromsensor-Einschubs dummerweise anders:

Kanal	ADC-Board		Sensor-Einschub	
	+	-	+	-
0	1	2	1	14
1	3	4	2	15
2	5	6	3	16
3	7	8	4	17
4	9	10	5	18
5	11	12	6	19
6	14	15	7	20
7	16	17	8	21

- Die Schirmung erfolgt auf der Senderseite.
- Die Signale sind auf der Senderseite potentialfrei verdrahtet.
- Die Minus-Signale sind auf dem ADC-Board mit der Analogmasse verbunden.
- Die ankommenden analogen Signale werden durch acht 'Instrumentation Amplifier' (AD620B) mit Gleichtaktunterdrückung empfangen. Eine Twisted-Pair Kabelverbindung ist deshalb empfehlenswert.
- Die Eingänge der AD620B werden mittels Diode zu den Versorgungsspannungen vor Überspannungen geschützt. Dabei dürfen 6 mA auf Dauer nicht überschritten werden. Um dies bei fehlenden Versorgungsspannungen zu vermeiden, sollten die Zuleitungen mit 10 kOhm in Serie beschalten werden.
- Unbenutzte Eingänge müssen abgeschlossen werden (0 - 1 MOhm), da sich sonst mit der Zeit eine negative Spannung aufbaut, die drastische Fehlmessungen des nachfolgenden ADCs zur Folge hat.
- Als Eingangsspannung sind 0V - +10V, entsprechend 0 - 4095 digital, zugelassen (1 LSB = 2.44 mV).
- Es sind keine Trimmer vorgesehen zur Korrektur von Nullpunkt- und Verstärkungsfehlern.
  Der Offset lag bei beiden Boards bei ca. -18 mV und ist nicht durch die Spezifikationen der verwendeten Halbleiter zu erklären. Möglicherweise ist er bedingt durch die Leiterbahnführung auf den Boards.
  Die Genauigkeit der Verstärkung ist gegeben durch die verwendete Widerstandsserie (1%). Durch Korrekturwiderstände könnte sie verbessert werden.
- **!!! Warnung !!!**
  Der MCP3208B ADC Baustein ist mehrfach kaputt gegangen. Vermutlich immer wenn die Karte bei eingeschaltetem Überrahmen gesteckt oder entfernt wurde!


ADC-3208 Board 1 (03.05.17)
1 LSB = 2.4414 mV
6m Kabel - Adapter - 0.5m Kabel
Chan-0
Chan-1
Chan-2
Chan-3
Chan-4
Chan-5
Chan-6
Chan-7
Ist-Soll [mV]
Soll [V]


ADC-3208 Board 2 (18.05.17)
1 LSB = 2.4414 mV
6m Kabel - Adapter - 0.5m Kabel
Chan-0
Chan-1
Chan-2
Chan-3
Chan-4
Chan-5
Chan-6
Chan-7
Ist-Soll [mV]
Soll [V]

Ergebnisse eines linearen Fits von 0.5V bis 9V Sollwert für die ADC-Kanäle 0 bis 7.

```
Gleichung       y = A + B*x [mV]
           ADC-Board1              ADC-Board2
 A0     -20.9   +- 0.3         -21.7   +- 0.4
 B0       2.31  +- 0.05          3.15  +- 0.07
 A1     -21.5   +- 0.4         -21.5   +- 0.4
 B1       0.79  +- 0.08          3.66  +- 0.08
 A2     -21.6   +- 0.4         -21.7   +- 0.3
 B2       2.13  +- 0.07          4.56  +- 0.05
 A3     -21.5   +- 0.4         -21.4   +- 0.3
 B3       3.30  +- 0.07          3.37  +- 0.05
 A4     -21.5   +- 0.4         -22.3   +- 0.2
 B4       3.68  +- 0.07          2.83  +- 0.04
 A5     -22.2   +- 0.4         -22.1   +- 0.3
 B5       0.46  +- 0.08          2.78  +- 0.05
 A6     -21.7   +- 0.4         -22.5   +- 0.4
 B6      -0.26  +- 0.07          2.30  +- 0.07
 A7     -22.2   +- 0.2         -23.5   +- 0.4
 B7       3.21  +- 0.04          3.63  +- 0.07

A0    -17.81631 +- 0.36094
B0      1.17450 +- 0.06755
A1    -17.31443 +- 0.26169
B1      0.59412 +- 0.04897
A2    -17.73253 +- 0.35248
B2      0.98035 +- 0.06596
A3    -18.01110 +- 0.33027
B3      2.23223 +- 0.06181
A4    -18.16802 +- 0.30115
B4      2.63357 +- 0.05636
A5    -17.72569 +- 0.41256
B5     -0.74348 +- 0.07721
A6    -17.79580 +- 0.4471
B6     -1.35637 +- 0.08367
A7    -18.22818 +- 0.37118
B7      2.15909 +- 0.06946

A0    -17.81631 +- 0.36094       -18.42244 +- 0.39507
B0      1.17450 +- 0.06755         2.27836 +- 0.06305
A1    -17.31443 +- 0.26169       -18.43301 +- 0.29590
B1      0.59412 +- 0.04897         3.70690 +- 0.04722
A2    -17.73253 +- 0.35248       -18.09811 +- 0.33390
B2      0.98035 +- 0.06596         3.54296 +- 0.05328
A3    -18.01110 +- 0.33027       -18.55780 +- 0.36544
B3      2.23223 +- 0.06181         2.40070 +- 0.05832
A4    -18.16802 +- 0.30115       -18.51179 +- 0.33983
```

```
B4      2.63357 +- 0.05636        1.82093 +- 0.05423
A5    -17.72569 +- 0.41256      -18.90742 +- 0.33075
B5     -0.74348 +- 0.07721        1.89731 +- 0.05278
A6    -17.79580 +- 0.44710      -19.03359 +- 0.36677
B6     -1.35637 +- 0.08367        1.44670 +- 0.05853
A7    -18.22818 +- 0.37118      -19.29849 +- 0.39532
B7      2.15909 +- 0.06946        2.62594 +- 0.06309
```

4.10.2 Bedienelemente

```
LED "Busy " :  Anzeige einer Schreiboperation
TP          :  Testpunkt zur Ausgabe von internen Signalen des ispLSI1032E
```

4.10.3 Programmierung

Die MCP3208-Karte belegt eine Control-Routing-Adresse für Ein/Ausgabe (Siehe Abschnitt 3.2 [Address-Decoder], Seite 5.). Mit der Ausgabe der ADC-Kanalnummer startet der ADC die Konvertierung des Analogsignals des angegebenen Kanals. Die Konvertierung benötigt 19 * 600ns = 11400ns. Während dieser Zeit wird das Acknowledge-Signal (AddAcc) für die Adressierung unterdrückt und erneute ADC-Starts werden ignoriert. Eingaben während dieser Konvertierungsphase liefern ungültige Werte und haben als Kennung das höchstwertige Bit gesetzt (0x8xxx). Dies könnte neben der Abfrage des AddAcc-Signals ebenfalls zur Synchronisierung mit dem ADC genutzt werden, es hat sich jedoch gezeigt, dass eine solche Eingabeoperation auf der Analogseite zu einer erheblichen Störung führt (ca. 30mV Schwankungen im Ergebnis). Nach der Konvertierungsphase kann das 12-bit Ergebnis beliebig oft mit einem Eingabebefehl abgerufen bzw. eine neue Konvertierung gestartet werden.

Zur Synchronisation mit dem ADC stehen zwei Statusabfragen zur Verfügung. Die Abfrage des Routing Status ist die bessere Variante da eine Dateneingabe mit Prüfung des Statusbits auf der ADC-Karte zu einer digitalen Störung führt!

```
Register Ausgabe:   Ausgabe der Kanalnummer
    Bits 15..04     0
    Bits 03..00     Kanalnummer 0-7

Register Eingabe:   Status/ADC-Daten Eingabe
    Bit 15 == 1     ADC busy, Daten ungültig
    Bit 15 == 0     ADC ready, Daten gültig
    Bits 11..00     ADC Daten

Routing Status Eingabe:
    (Routing Status & 0x0008) != 0 -> ADC busy, Daten ungültig
    Diese Statusabfrage ist nur bei Verwendung einer neuen
    Address-Decoder-Karte (EW 3/04) möglich da die alten
    ebenfalls dieses Bit bedienen.
    (Siehe Abschnitt 3.2 [Address-Decoder], Seite 5.)
```

4.10.4 Analoge Bauteile

Die acht Eingänge sind mit dem AD620B Instrumentation Amplifier bestückt:

	Typ	Max	
Nonlinearity	10	95	ppm
Gain Error	0.10	0.15	%
Input Offset	15	50	uV
Output Offset	200	500	uV

Die Konvertierung erfolgt mit einem 12-bit, 8-Kanal ADC MCP3208B:

	Typ	Max	
Integral Nonlinearity	+-0.75	+-1	LSB
Diffenrential Nonlinearity	+-0.5	+-1	LSB
Offset Error	+-1.25	+-3	LSB
Gain Error	+-1.25	+-5	LSB

(1 LSB = 2.4414 mV)

4.11 Dual 16/18-Bit standalone DAC-Boards

Die dual 16/18-Bit DAC-Karten wurde entwickelt zur Steuerung der Spannungen der verschiedenen Platten der neuen Elektronenkanone in Labor 017. Sie sind keine Control-Routing-Boards und gehören deshalb eigentlich nicht in diese Abteilung der Hardware-Beschreibungen.

Es existieren drei Varianten dieser DACs mit unterschiedlichen Eigenschaften, die sich jedoch alle bezüglich der Programmierung im Rahmen ihrer Möglichkeiten gleich verhalten. Bei den 16 Bit-DACs kommen die zwei niederwertigsten Bits nicht zur Anwendung:

- Dual 18 Bit DAC LTC2758 auf DC1684A-A Board
  Firmware: dac2758-18-pso32
  Bereiche [V] bei 0 - 0x3ffff digital: 0..+5, 0..+10, -2.5..+2.5, -5..+5, -10..+10, -2.5..+7.5
- Dual 16 Bit DAC LTC2752 auf DC1684A-B Board
  Firmware: dac2752-16-pso32
  Bereiche [V] bei 0 - 0x3ffff digital: 0..+5, 0..+10, -2.5..+2.5, -5..+5, -10..+10, -2.5..+7.5
- 2 * 16 Bit DAC AD5541CR
  Firmware: dac5541-16-pso32
  Bereiche [V] bei 0 - 0x3ffff digital: 0..+10

Alle drei DAC-Karten besitzen ein eigenes Netzteil und benötigen nur 230V AC zu ihrem Betrieb. Die Spannungszuführung und Signalabführung erfolgen über eine Steckerleiste in den DAC-Boxen zu deren Rückseite. Die LTC-Karten haben zusätzlich einen dreipoligen Stecker zur Spannungszuführung und 2 * 2 Steckerstifte für die Signale auf dem Board als Einbauversion.

Die DAC-Karten werden über Lichtleiter angesteuert und sind deshalb für einen Betrieb auf höherem Potential geeignet. Als passender Lichtleitersender wurde das PSO14-Control-Routing-Board mit Firmware PSO32 (Siehe Abschnitt 4.1 [Serielle Ausgabe (PSO14)], Seite 8.) entwickelt. Die Datenübertragung benötigt ca. 19us.

Die Ausgänge der DACs sind potentialfrei, deshalb ist Folgendes zu beachten:

- **Der Masse-Pol muss mit einem definierten Potential (max. 500V gegen Schutzerde) beschaltet werden.**
- **Die beiden Masse-Pole einer DAC-Karte sind verbunden und müssen deshalb auf dem selben Potential liegen.**
- **Bei höhergelegtem Potential der Ausgänge ist aus Sicherheitsgründen eine Kabelverbindung mit Schutzerde-Schirmung zu verwenden.**
  Werden die DACs in den zugehörigen Boxen betrieben, so steht Schutzerde auf Pin 2 der Diodenstecker zur Verfügung zum Anschluss der Schirmung.

```
Pin   Signal
 1    Out
 2    Schutzerde
 3    Masse
```

Die DAC-Karten sollten vorzugsweise ganz in der Nähe der zu steuernden Hochspannungsnetzgeräte platziert werden damit nur eine kurze Leitungslänge für die analogen Steuersignale benötigt wird.

4.11.1 Probleme und Lösungen

Durchlaufverzögerung des LWL-Empfängers SFH551V
Der SFH551V ist relativ zum Bit-Takt (400ns) ziemlich lahm. Insbesondere störend für das Timing sind die unterschiedlichen Durchlaufverzögerungen für Licht-an und Licht-aus, die zu einer Verkürzung des elektrischen Licht-aus Signals um bis zu max. 150ns führen, je nach Intensität des LWL-Signals und Temperatur. Hinzu kommen noch 50ns Unschärfe durch das Einsychronisieren auf den internen Clock-Takt (20MHz). Das Timing kann bis auf 200ns verkürzte Bits sicher erkennen, gemessen am Ausgang des SFH551V. Im Zweifel hilft ein längeres LWL-Kabel.
Auf einer PSO14-Karte musste ein LWL-Sender ausgetauscht werden da er zu hell war und den SFH551V völlig übersteuerte.

4.11.2 DAC2752/2758-16/18-PSO32

4.11.2.1 Funktion (DAC2752/2758)

Zum Einsatz kommen sogn. Demo-Boards DC1684A mit den DACs LTC2752 (16 Bits) bzw. LTC2758 (18 Bits), die auf eine Europakarte mit Netzteil, LWL-Empfänger, Offset-/Gain-Abgleich und einem FPGA zur Steuerung montiert sind.

4.11.2.2 Bedienelemente (DAC2752/2758)

Weitere Infos siehe: DEMO MANUAL DC1684A und LTC2752/LTC2758 Data Sheets

Manuelle Einstellung des Spannungsbereichs auf dem DC1684A-Board

```
Jumper MSPAN
         0   programmierte Einstellung, beide DACs getrennt
         1   manuelle Einstellung mittels Sn, beide DACs gemeinsam

Jumpers  S2   S1   S0
          0    0    0     0V to 5V
          0    0    1     0V to 10V
          0    1    0    -5V to 5V
          0    1    1   -10V to 10V
          1    0    0  -2.5V to 2.5V
          1    0    1  -2.5V to 7.5V
```

Offset und Gain Abgleich
Mittels vier Potis können, falls nötig, Offset und Gain abgeglichen werden wenn die zugehörigen Jumper auf EXT gesetzt sind:

```
              Jumper    Poti
   Offset-A   VOSADJA    ?
   Gain-A     GEADJA     ?
   Offset-B   VOSADJB    ?
   Gain-B     GEADJB     ?
```

Weitere Jumpers mit fester Einstellung

```
Jumper   Position
VREFA       5V      Referenzspannungsquelle DAC-A
VREFB       5V      Referenzspannungsquelle DAC-B
VCC         REF     5V Spannungsquelle
!CLR         1      Asynchrones Clear
!LDAC        1      Asynchrones Load
!RFLAG       1      Reset Flag Output
```

Anschlüsse und Testpunkte

```
LWL-Buchse       :  Lichtleiterempfänger
DACA Stecker OUTA:  Kanal A analoges Ausgangssignal
DACA Stecker GNDA:  Kanal A analoge Masse
DACB Stecker OUTB:  Kanal B analoges Ausgangssignal
DACB Stecker GNDB:  Kanal B analoge Masse
TP 0             :  Serielle LWL Daten
TP 1             :  Serielle DAC Daten
TP 2             :  DAC Chip Select
TP 3             :  DAC Clock

230 V Netzspannung an Steckerleiste VG32AC:
24 a,c   Nullleiter
28 a,c   230V Leiter
32 a,c   Schutzleiter

230 V Netzspannung an Stecker:
???   Nullleiter
???   230V Leiter
???   Schutzleiter
```

4.11.2.3 Programmierung (DAC2752/2758)

Die DAC2752/2758-Karten erhalten ihre Daten über Lichtleiter. Ein geeigneter LWL-Sender ist die PSO14-Karte mit Firmware PSO32 (Siehe Abschnitt 4.1 [Serielle Ausgabe (PSO14)], Seite 8.).

Das Format der DAC-Daten ist das gleiche für die DAC2752- und DAC2758-Karten. Für die DAC2752-Karten werden die beiden niederwertigsten Bits jedoch nicht verwertet. Die Read Back Funktion der DACs wird nicht unterstützt.
Bit-Folge, MSB zuerst:

```
 4 Bit   DAC-Kommando
         0 0 1 0    Write Span DACn
         0 0 1 1    Write Code DACn
         0 1 0 0    Update DACn
         0 1 0 1    Update all DACs
         0 1 1 0    Write Span DACn, Update DACn
         0 1 1 1    Write Code DACn, Update DACn
```

```
 4 Bit   DAC-Adresse
         0 0 0 x    DACA
         0 0 1 x    DACB
         1 1 1 x    all DACs
Danach folgt der DAC-Code bzw. der Span-Code, MSB zuerst:
18 Bit   DAC-Code   / 12 Bit   don't care
 6 Bit   don't care / 04 Bit   Span-Code
                    /  8 Bit   don't care
```

4.11.2.4 Probleme und Lösungen (DAC2752/2758)

Einschwingverhalten

Das Einschwingen erfolgt exponentiell mit folgenden Halbwertszeiten:

Bereich [V]	0 - 5	0 - 10
DAC2752	200ns	500ns
DAC2758	150ns	250ns

Verhalten beim An-/Abschalten der DAC-Karten

Beim Anschalten der DAC-Karten pendelt die Ausgangsspannung, vermutlich abhängig von dem Verhalten der Spannungsversorgung, zwischen erheblichen positiven und negativen Werten.
Verhalten bei dem HAMEG HM8040-2 Triple Power Supply:

DAC2752		DAC2758	
1ms	-0.5V	1ms	-2V
50ms	0->4V	0.1ms	1V
0.5ms	6V	50ms	-0.5->0V
		1ms	3V

Störungen durch das Schaltnetzteil

Die Messungen wurden als Differenzmessung mit zwei Tastköpfen durchgeführt. Dabei ist das Ergebnis abhängig von der Tastkopferdung. Bei den Messungen am Ausgang der DAC-Boxen waren die Störungen z.B. deutlich geringer wenn beide Tastkopferdungen mit der Schutzerde der Box verbunden waren.

Für die verbauten Morsun Schaltnetzteile wird Ripple & Noise mit max. 100mVss im Datenblatt angegeben. Tatsächlich wurde auf den Morsun-Ausgängen eine gedämpfte Hf-Störung (~2MHz, ~150mVss) im Takt der Schaltfrequenz (T~7/14us) angeregt, die auch genau so auf den DAC-Ausgängen zu beobachten war.

Mit folgenden Änderungen konnte die Hf-Störung für die DAC2752/2758 erheblich reduziert werden:

1. Nachrüsten der vergessenen 100nF SMD-Kondensatoren an den Morsun-Ausgängen.
2. 10nF, Cy zwischen Schutzerde und 5V-GND.
3. 3.9nF, Cy zwischen Schutzerde und +-15V-GND.
   (3.9 + 6.8)nF führten zu keiner weiteren Verbesserung.
   5V-GND und 15V-GND sind auf der DAC-Karte verbunden.
   Vor (1) führten 10nF zu einem Hin- und Herflippen im Sekundentakt.
4. Standard-Einbaustecker mit Netzfilter in Netzzuleitung und einem zusätzlichen 20nF Entstörkondensator zwischen den Zuleitungen, ohne den das Ergebnis aber gleich war.

Mit diesen Maßnahmen lässt sich die Hf-Störung auf einen durch den Schaltvorgang hervorgerufenen low/high-Zacken (je 100ns) von 10mVss reduzieren. Dazwischen ca. 5mVss Restwelligkeit.

Bei Betrieb der Karte in einer der DAC-Boxen erhält man am Diodenstecker das gleiche Ergebnis. Allerdings erst nach Umlöten der falsch gepolten Netzfilter (Line - Load vertauscht). Alle Karten in der Box haben ein gemeinsames Netzfilter.

4.11.2.5 Anwendung bei der Elektronenkanone (DAC2752/2758)

Anschluss FUG

Getestet wurde mit dem alten, 1.5kV FUG (Plus-Pol geerdet).
!!!Achtung: mit geerdetem Minus-Pol liegen die DAC-Anschlüsse auf Plus-HV-Potential!!!
Der Anschluss erfolgte von einer DAC-Box über 1.5m geschirmtes Kabel. Die Schirmung war mit der Schutzerde der DAC-Box verbunden und hatte keine Verbindung mit dem FUG.

Gemessen am FUG-Eingang treten gedämpfte Schwingungen auf:

```
100mVss, 10MHz
```

Mit 150nF oder galvanischer Verbindung zwischen Masse und Schutzerde am zur Karte gehörenden Nachbar-Ausgang der DAC-Box reduziert sich die Störung:

```
50mVss, 10MHz
```

Diese HF-Störung wirkt sich nur unwesentlich auf den FUG-Ausgang aus:

```
off   0   14   [V] FUG HV
10   10   10   [mVss] Rauschen
 0   40   40   [mVss] HF Spitzen, weniger wenn DAC off
 0   20   20   [mVss] 50Hz Brumm
```

Anschluss Kathodenregelung

Über die beiden DACs einer im Regeleinschub eingebauten DAC-Karte wird der Sollwert (grob/fein) vorgegeben. Den Istwert erzeugt ein 1/400 HV-Teiler aus der FUG-Hochspannung. Ein Integralregler liefert den Stellwert für das Kepco. Das Kepco gibt die Vorspannung für das FUG.

Durch die Regelung wird der 50Hz Brumm des alten FUG auf <10mVss reduziert. Im Takt der 50Hz Halbwellen ist jedoch für 5ms eine andauernde HF-Störung zu sehen.

Folgendes wurde untersucht um die Ursache der Störung zu finden:

1. Alles an: 50mVss, 5ms, 50Hz synchron; 150mVss Spitzen am Ende; anschließend 5ms Pause; <10mVss Brumm
2. Regelung aus: gleiches Ergebnis bei 20mVss Brumm.
3. FUG aus: gleiches Ergebnis ohne Brumm.
4. Regelung und FUG aus: gleiches Ergebnis ohne Brumm.
5. Kepco aus: 2Vss Brumm (halbe Halbwelle!), aber nur wenn der FUG-DAC angeschlossen und angeschaltet ist und nicht wenn die DAC-Masseleitung mit Schutzerde verbunden ist.
6. 150nF an Kepco-HV-Ausgang: verringert die Störung nicht.

7. 150nF an FUG-HV-Eingang: verringert die Störung deutlich.
   Die Anstiegszeit (~50us) wird etwas kürzer (!), das Überschwingen verstärkt sich.
8. 150nF zwischen Masse und Schutzerde am FUG-DAC-Nachbarausgang: ähnliches Ergebnis wie zuvor jedoch größere Hf-Spitzen.
9. 150nF am FUG-DAC-Nachbarausgang der DAC-Box zwischen Masse und Signal bringt keine Reduzierung der Störung.
10. 330nF, 250V Cy am Ausgang der Regelung zwischen Masse und Schutzerde beseitigt die HF-Störungen durch die Regelung (DAC-Box abgeschaltet).
    Galvanische Verbindung führt zu 50Hz-Brumm durch Brummschleife.
    Hat keinen Einfluss auf das Einschwingverhalten der Regelung.
11. 250nF, 5kV am Eingang der Regelung zwischen Masse und HV-Signal bügelt alle HF-Störungen aus.

Folgendes wurde zur Reduzierung der Störungen unternommen, gemessen am HV-Eingang der Regelung:

1. 330nF, 250V Cy am Ausgang der Regelung zwischen Masse und Schutzerde beseitigt die HF-Störungen durch die Regelung. Hat keinen Einfluss auf das Einschwingverhalten der Regelung.

   ```
   FUG: 0V, Kepco: 25V - 0V
   Rauschen  Brumm  HF  Spitzen  [mVss]
      10       10    -     20   FUG-DAC off
   ```

2. 150nF, 250V Cy am Nachbarausgang der DAC-Box zwischen Masse und Schutzerde beseitigt die HF-Störungen durch den FUG-DAC in der Box, hat jedoch einen Einfluss auf das Einschwingverhalten der Regelung. Der Überschwinger bei einem Spannungssprung (10V) erhöht sich von 0.5% auf 0.75% und die 50%-Anstiegszeit vermindert sich von 40us auf 35us.

   ```
   FUG: 14V, Kepco: 25V - 14V
   Rauschen  Brumm  HF  Spitzen  [mVss]
      10       10   40     50   FUG-DAC on,
      10       10  <10     20   FUG-DAC on, 150nF am Nachbarausgang
   ```

3. 250nF, 5kV an der Kanone zwischen Kathode und WW-Zone ist recht unhandlich, könnte aber auch interessant werden...

50Hz-Brumm an KEPCO Ausgang:

Die DAC-Ausgänge haben je nach dem wo sie positioniert sind einen geringen Brummanteil (<5mVss). Im Verbund mit der Kathodenspannungsregelung (Spannungsteiler, Regelung, KEPCO) tritt jedoch zum Teil erheblicher Brumm auf.

```
Alle Geräte abgeschaltet und Netzstecker von:
   Regelung     aus   ein   ein   ein   ein   ein   ein
   KEPCO        aus   aus   ein   aus   ein   ein   gedreht
   FUG          aus   aus   aus   ein   ein gedreht ein
   Brumm [mVss]   2     2    10     5    15    30    30
Alle Geräte eingeschaltet, 1/200 Teiler:
   ohne Masseverbindung zwischen den Geräten:       60mVss
   mit Masseverbindung zwischen Regelung und KEPCO: 60mVss
```

```
   mit Masseverbindung zwischen Regelung und FUG
     und spezieller Leitungsführung:                10mVss
Alle Geräte eingeschaltet, 1/1000 Teiler:
   ohne Masseverbindung zwischen den Geräten:           600mVss
   mit Masseverbindung zwischen Regelung und KEPCO:    1000mVss
   mit Masseverbindung zwischen Regelung, Teiler, KEPCO: 80mVss
     Lässt sich durch zusätzliche Verbindung vom
     Deckel der Regelung zum Teiler noch reduzieren:     30mVss
   mit Masseverbindung zwischen Regelung und FUG:       200mVss
   mit Masseverbindung zwischen Regelung, Teiler, FUG:  200mVss
```

4.11.3 DAC5541-16-pso32

4.11.3.1 Funktion (DAC5541)

- Die Ausgangsspannung ist 0V - +10V, entsprechend 0 - 0x3ffff digital. Die beiden niederwertigsten Bits werden nicht verwertet.
- Der neue Spannungswert wird am Ende der Übertragung aktiviert.
- Die Ausgänge können mit maximal 15 mA belastet werden.
- Es sind keine Trimmer vorgesehen zur Korrektur von Offset-Fehlern da die Bausteine ausreichend präzise sein sollten. Im Test haben sich dann jedoch Offset-Fehler bis 30mV gezeigt, die als Ursache eine ungeschickte Leiterbahnführung des analogen Grounds im Layout haben. Deshalb war es nötig mit fliegender Verdrahtung den analogen Ground an zwei strategisch wichtigen Stellen zu verbessern. Mit dieser Aktion halten sich die Offset-Fehler innerhalb +-2mV. Bessere Werte sind noch zu erreichen mit Trimmern an den OpAmps AD820. Für das DAC-Board 5 wurde dies durchgeführt mit festen Widerstandswerten da die Offset-Fehler nach der Verdrahtungskorrektur noch zu groß waren.
- Die Verstärkung wurde mit Korrekturwiderständen justiert da die verwendete Widerstandsserie nicht ausreichend präzise ist (1%).

```
Testdaten der DAC-Boards, Abweichungen vom Sollwert
Sollwerte 0    1    2    4    6    8    9   10   V
-------------------------------------------------
DAC1/0   -1   -2   -2   -2   -2   -2   -2   -2  mV
DAC1/1   -1   -2   -2   -2   -2   -2   -2   -2  mV
DAC2/0    0    0    0    0    0    0    0    0  mV
DAC2/1   +2   +2   +2   +2   +2   +2   +2   +2  mV
DAC3/0   +2   +2   +2   +2   +2   +2   +2   +2  mV
DAC3/1    0    0    0    0    0    0    0    0  mV
DAC4/0    1    0    1    0    0    0    0   -1  mV
DAC4/1    1    1    1    0    0    0   -1   -1  mV
DAC5/0    0   -1   -1   -1   -1   -1   -1   -1  mV
DAC5/1    0    0    0    0    0    0    0    0  mV
```

- Laut Datenblatt sollten die DACs in ca. 2 us ihren Endwert erreichen. Im Test zeigte sich jedoch, dass sie zunächst ca. 10% überschwingen (auch zu negativen Werten!) um sich dann mit einer Halbwertszeit von ca. 50us exponentiell dem Sollwert zu nähern

(Ursache unklar). Dieses Überschwingen konnte kompensiert werden durch 1.5nF * 30KOhm = 45us in der Rückkopplung des AD820. Die Halbwertszeit reduzierte sich damit auf ca 25us. Nach 250us ist die Abweichung vom Sollwert also nur noch 0.0001.

- Die Konvertierung erfolgt mit zwei 16-bit seriellen DACs AD5541CR mit folgenden Eigenschaften:

```
                                Typ     Max
Relative Accuracy             +-0.5    +-1     LSB
Diffenrential Nonlinearity    +-0.5    +-1     LSB
Zero Code Error               +-0.3    +-0.7   LSB
Gain Error                    +-0.5    +-2     LSB
Voltage Settling Time           1              us
( 1 LSB = 0.15259 mV)
```

- Die nachfolgende Verstärkung erfolgt mit AD820A Op-Amps mit folgenden Eigenschaften:

```
                    Typ    Max
Offset              0.5     3   mV
Offset Drift        2           uV/K
Output Current             15   mA
```

4.11.3.2 Bedienelemente (DAC5541)

Die Funktion einiger Bedienelemente hat sich im Laufe der Weiterentwicklung geändert. So war ursprünglich aus historischen Gründen für jeden Kanal eine eigene LWL-Verbindung vorgesehen, nun wird aber nur noch eine einzige benötigt, weshalb beide jetzt die gleiche Funktion haben. Ebenso haben die Testpunkte eine neue Funktion erhalten.

```
LWL-Buchse 0     :  Lichtleiterempfänger Kanal 0 und 1
LWL-Buchse 1     :  Lichtleiterempfänger Kanal 0 und 1
DAC0 Stecker OUT :  Kanal 0 analoges Ausgangssignal
DAC0 Stecker GNDA:  Kanal 0 analoge Masse
DAC1 Stecker OUT :  Kanal 1 analoges Ausgangssignal
DAC1 Stecker GNDA:  Kanal 1 analoge Masse
TP 0 word        :  Serielle LWL Daten
TP 1 word        :  Serielle DAC Daten
TP 0 bit         :  DAC Chip Select
TP 1 bit         :  DAC Clock

230 V Netzspannung an Steckerleiste VG32AC:
24 a,c   Nullleiter
28 a,c   230V Leiter
32 a,c   Schutzleiter
```

4.11.3.3 Programmierung (DAC5541)

Die DAC5541-Karte erhält ihre Daten über Lichtleiter. Ein geeigneter LWL-Sender ist die PSO14-Karte mit Firmware PSO32 (Siehe Abschnitt 4.1 [Serielle Ausgabe (PSO14)], Seite 8.).

Das Format der DAC-Daten ist das gleiche wie für die DAC2752- und DAC2758-Karten, es kann jedoch nur eine Untermenge der Kommandos ausgeführt werden:

```
 4 Bit   DAC-Kommando
         nur für die Bit-Muster (0 x 1 1) erhält der DAC5541 neue Daten
 4 Bit   DAC-Adresse
         x 0 0 x    DAC0
         x 0 1 x    DAC1
         x 1 1 x    DAC0 und DAC1
18 Bit   DAC-Daten, MSB zuerst
 6 Bit   nicht genutzt
```

4.11.3.4 Probleme und Lösungen (DAC5541)

Einschwingverhalten

Laut Datenblatt sollten die DACs in ca. 2 us ihren Endwert erreichen. Im Test zeigte sich jedoch, dass sie zunächst ca. 10% überschwingen (auch zu negativen Werten!) um sich dann mit einer Halbwertszeit von ca. 50us exponentiell dem Sollwert zu nähern (Ursache unklar). Dieses Überschwingen konnte kompensiert werden durch 1.5nF * 30KOhm = 45us in der Rückkopplung des AD820. Die Halbwertszeit reduzierte sich damit auf ca. 25us. Nach 250us ist die Abweichung vom Sollwert also nur noch 0.0001.

Verhalten beim An-/Abschalten der DAC-Karte

Beim Anschalten der DAC-Karte sollte der AD5541CR laut Datenblatt einen Reset durchführen. Möglicherweise klappt dies in seltenen Fällen nicht immer.
Beim Abschalten der DAC-Karte geht der Ausgang kurzzeitig zu negativen Werten (ca. -5V) um nach ca. 400ms 0V zu erreichen. Mit einer Diode am Ausgang können diese negativen Werten auf -0.6V/200ms begrenzt werden. Eine gute Idee ist auch, dafür zu sorgen, dass die DACs immer nur gemeinsam mit den Hochspannungsgeräten abgeschaltet werden.

Störungen durch das Schaltnetzteil

Die DAC5541-Karten werden ebenfalls durch das Schaltnetzteil stark gestört. Gegenmaßnahmen wie bei den DAC2752/2758-Karten wurden bisher noch nicht untersucht.

5 Spezielle Interface-Karten

An dieser Stelle sind Interface-Karten verzeichnet, die nur als Einzelexemplare für ein spezielles Labor/Experiment entwickelt wurden. Details findet man möglicherweise in den Unterlagen der zugehörigen Experimente. Diese Aufstellung enthält auch Oldies, in der Hoffnung, sie eines Tages wieder recyclen zu können.

5.1 Schrittmotor-Interface (Labor 016)

5.1.1 Aufgabe, Funktion

Interface zu einer Eigenbau-Schrittmotorsteuerung, mit der ein Schlitz durch die Strahlen gefahren wird zur Strahlprofilanalyse (IIF-Messprogramm).

Mit einem Befehl können maximal 65565 Schritte in vorwärts oder rückwärts Richtung ausgeführt werden. Die Schrittgeschwindigkeit wird von der Schrittmotorsteuerung vorgegeben.

Der Schrittmotor wird angehalten bei folgenden Ereignissen:

- die vorgegebene Anzahl der Schritte ist abgearbeitet
- die untere Grenze der Schlitzbewegung ist erreicht
- die obere Grenze der Schlitzbewegung ist erreicht
- durch ein Reset-Kommando

Das Schrittmotor-Interface belegt vier Registeradressen des Address-Decoders (Siehe Abschnitt 3.2 [Address-Decoder], Seite 5.).

5.1.2 Bedienelemente

Das Schrittmotor-Interface enthält nur einen 7-poligen Phonostecker zum Anschluss der Schrittmotor-Steuerung:

```
Pin  Signal
 1    !Schrittmotor online (in)
 2    !Alarm Obergrenze (in)
 3    !Alarm Untergrenze (in)
 4     Schrittmotor-Takt (in)
 5    !Start Motor (out)
 6    !Hochlauf (out)
 7     Masse
```

5.1.3 Programmierung

```
Register 0:           Reset-Kommando (0 Bits)

Register 1 out:       Anzahl der Schrittmotorschritte (16 Bits)
                      im 1-Komplement:
                        0 -> 0xff
                        1 -> 0xfe

Register 2 out:       Start-Kommando, Richtung (1 Bit)
```

```
        Bit 0         0 = Schlitz nach unten
        Bit 0         1 = Schlitz nach oben

Register 3 in:        Status-Eingabe
        Bit 15        1 = Schrittmotor ist online
        Bit 14        1 = Schlitz am oberen Limit
        Bit 13        1 = Schlitz am unteren Limit
        Bit 12        1 = busy, Schrittmotor läuft
```

5.2 Probenwechsler-Steuerung (Schacht)

!!! unvollständig, mehr im Ordner "Probenwechsler" !!!

5.2.1 Aufgabe, Funktion

Die Probenwechsler-Steuerung besteht aus zwei Control-Routing-Karten. Sie wurde entwickelt zur Steuerung des Probenwechslers von H. Schacht (s.h. Diplomarbeit H. Schacht: Ein automatisierter Gamma-Messplatz für Photonenaktivierungsanalyse (1992)).

Der Probenwechsler transportiert mit Hilfe von 4 Schrittmotoren die Proben aus einem Magazin vor den Detektor und wieder zurück. Ein weiterer Motor bedient ein Eingangstor zur Bleiburg, in der gemessen wird. Über 9 Sensoren wird der Transport und der Status des Eingangstores überwacht.

Die Probenwechsler-Steuerung belegt sechs Registeradressen des Address-Decoders (Siehe Abschnitt 3.2 [Address-Decoder], Seite 5.).

Es existiert nur eine handverdrahtet Version der Karten.

5.2.2 Bedienelemente

```
LED "Busy"     :  Anzeige Probenwechsler-Steuerung aktiv
LED "Phase 1"  :  Anzeige Phase 1 aktiv
LED "Phase 2"  :  Anzeige Phase 2 aktiv
LED "Phase 3"  :  Anzeige Phase 3 aktiv
```

Die Probenwechsler-Steuerung enthält einen 40-poligen Flachbandstecker zum Anschluss des Probenwechslers:

```
 Pin  Signal           Pin  Signal
 1-2  Masse            21  !Motor ready (in)
 3    Sensor 1 (in)    23   Motor Error (in)
 4    Sensor 2 (in)    25  !autonomer Stop (in)
 5    Sensor 3 (in)    27   Motor Adress-Bit 0 (out)
 6    Sensor 4 (in)    29   Motor Adress-Bit 1 (out)
 7    Sensor 5 (in)    31   Motor Adress-Bit 2 (out)
 8    Sensor 6 (in)    33  !Motor aktivieren (out)
 9    Sensor 7 (in)    35   Motor-Drehrichtung (out)
 10   Sensor 8 (in)    37   Signale ungültig (out)
 11   Sensor 9 (in)    39   Schritttakt (out)
 12,13..20 Masse       22,24..40 Masse
```

Mit jeder Modus-Ausgabe wird für ca. 200us "Signale ungültig" gesetzt, damit der Probenwechsler den Augenblick der Signalpegeländerung überbrücken kann.

5.2.3 Programmierung

```
Register 0 out:       Ausgabe Schrittzahl Phase 1 und 3
        Bits  7-0     Schritte Phase 1, Beschleunigung
        Bits 15-8     Schritte Phase 3, Verzögerung

Register 1 out:       Ausgabe Schrittzahl Phase 2
        Bits 15-0     Schritte Phase 2, Konstantlauf

Register 2 out:       Ausgabe Beschleunigungswert
        Bits 5-0      Beschleunigungswert N

Register 3 out:       Modus-Ausgabe
        Bit 2-0       Motoradresse
        Bit 4         1 = Motor aktivieren
        Bit 5         Motor-Drehrichtung

Register 4 out:       Start-Kommando, (0 Bits)

Register 5 in:        Status-Eingabe
        Bit 15        0 = Motor ready
        Bit 14        1 = Motor Error
        Bit 13        0 = autonomer Stop
        Bit 12        1 = Steuerung busy
        Bit 11        1 = Störung (gelöscht durch Status-Eingabe)
        Bits 8-0      Positions-Sensoren
```

Zusammenhänge in den Phasen 1 und 3 (Beschleunigung/Verzögerung):

```
Taktperiode        : T = 1 us
Weg                : S [steps]
Zeit               : t [sec]
Geschwindigkeit    : V [steps/sec]
Beschleunigung     : B [steps/sec^2]
Beschleunigungswert: N (1..63)

    B = N / T^2 * 15.92 * 10^-12 steps/sec^2
      = N * 15.92 steps/sec^2

Wenn V und t gegeben sind errechnen sich S und N folgendermaßen:
    S = V * t / 2
    N = V / t * 0.0628 steps/sec^2

Phasen 1 und 3 interne Limits (T = 1us):
    S = 1..255 steps  (programmiert)
    N = 1..63         (programmiert)
    Vmin = 10.17 steps/sec
    Vmax = 640.9 steps/sec
```

```
        Bmin = 15.92 steps/sec^2
        Bmax = 1003. steps/sec^2
        V * t = 2..510 steps

    Phase 2 interne Limits:
        S = 1..56535 steps  (programmiert)
```

5.2.4 Probenwechsler Details

Motore und Sensoren
Zur Rückmeldung eines Endanschlages und zur Normierung beim Start hat der Probenwechsler 9 induktive Näherungschalter installiert. Das Karussell besitzt nur einen Sensor zur Normierung. Alle Sensoren melden High-Pegel, wenn sie ansprechen.

Motor	**Sensor (Normierung)**	**Sensor**
1 Karussell	1 unter Teller (1-Loch)	-
2 Hub	2 unterer Sensor	3 unter Teller (20-Loch)
3 Wechsel	4 innerer Sensor (1-Loch)	5 Bleiburgsensor
4 Transport	6 torseitiger Sensor	7 detektorseitiger Sensor
5 Tor	8 oberer Sensor	9 unterer Sensor

Schrittzahlen
Motor 5 ist ein Analogmotor, dessen Laufzeit über die Anzahl der Schritte gesteuert wird. Um einen Sensor wieder mit Sicherheit zu öffnen, ist eine gewisse Anzahl von Schritten zurückzufahren. Die Probenplätze auf dem Karussell sind 300 Schritte von einander entfernt. Was war bloß mit dem Totpunkt???

Motor	**Anfang - Ende**	**Sensor öffnen**	**nächste Pos.**
1 Karussell	6000 Steps/Umdr.	20 Steps	300 Steps/Probenplatz
2 Hub	1000 Steps/Umdr.	40 Steps	500 Steps/Totpunkt
3 Wechsel	3940 Steps	20 Steps	
4 Transport	~12480 Steps	20 Steps	
5 Tor	~3200 Steps	20 Steps	

Empfohlene Geschwindigkeiten und Beschleunigungen
Mehr zu S und N Siehe Abschnitt 5.2.3 [Programmierung], Seite 51.

Motor	**Geschwindigkeit**	**Beschleunigungsdauer**	**S**	**N**
1 Karussell	200 Steps/sec	1 sec	100	13
2 Hub	500 Steps/sec	1 sec	250	31
3 Wechsel	500 Steps/sec	1 sec	250	31
4 Transport	500 Steps/sec	1 sec	250	31
5 Tor	100 Steps/sec	0.1 sec	5	63

6 Zählratenstatistik

Die Zählratenstatistik ist inzwischen zu einem eigenen Paper geworden.

HTML-Version:
`http://www.strz.uni-giessen.de/ExpHelp/statistik/statistik.html`

PDF-Version:
`http://www.strz.uni-giessen.de/ExpHelp/statistik/statistik.pdf`

7 Technische Details

7.1 Routing-Bus

7.1.1 Routing-Bus Signale

Der Routing-Bus (Siehe [Abb Control-Routing-Bus], Seite 56.) besteht aus zwei Teilen, dem 'Allgemeinen Bus', der über die ganze Breite des Überrahmens geht und dem 'Privat-Bus', der erst durch das Stecken benachbarter Karten entsteht und auf diese begrenzt ist.

Allgemeiner Bus (A-Bus)
Der allgemeine Bus belegt die Anschlüsse 1a, 1c usw. bis 21c. Die Anschlüsse mit dem Suffix a und c führen im Gegensatz zum Privat-Bus zu verschiedenen Bus-Leitungen und dürfen nicht gebrückt werden.

- **Stromversorgung (Standard):**

```
+ 5V / 10A
+15V / 1A
-15V / 1A
```

- **DO - D15** (Datenleitungen):
  Sechszehn Datenleitungen stehen zur Verfügung. Low auf einer Leitung bedeutet, dass das Daten-Bit gesetzt ist. Offene Leitung heißt, das Daten-Bit ist nicht gesetzt.

- **10 MHz-Clock/-Clock**:
  Quarzstabilisierter Mastertakt der Routing-Steuerung.
  Tastverhältnis: 1:2
  Stabilität: 10ppm

- **1 MHz-Clock**:
  Vom Mastertakt abgeleiteter 1MHz Takt.
  Tastverhältnis: 1:2

- **Reset** (Reset Routing):
  Reset-Signal, das sowohl beim Anschalten der Netzspannung (Power-up-Reset, ca. 200ms) erscheint als auch durch Programm erzeugt werden kann (Siehe ⟨undefined⟩ [xxxx], Seite ⟨undefined⟩.).

- **PAddr0..2** (Steckplatzadresse):
  Für jeweils zwei benachbarte Steckplätze ist eine gemeinsame Steckplatzadresse 0-7 verdrahtet (Siehe [Abb Routing-Ueberrahmen], Seite 4.). Die Address-Decoder müssen so gesteckt werden, dass sie verschiedene Adressen einnehmen, da sie nur dann eindeutig zu adressieren sind.

- **MAddr0..2** (Moduladresse):
  Ein Steckplatz gilt als angewählt, wenn die Bits 0-2 von Steckplatz- und Moduladresse übereinstimmen. Nur dann darf eine Interface-Karte auf die Steuersignale der Routing-Steuerung reagieren und Datentransfers durchgeführen.

- **RAddr0..2** (Registeradresse):
  Für jede Steckplatzadresse können noch 8 Register direkt adressiert werden.

- **Address_Valid** (Freigabetakt für den Adressvergleich):
  Mit diesem Signal zeigt die Routing-Steuerung an, dass die am Bus liegenden Adressen MADR und RADR gültig sind.

- **Address_Error** (Rückmeldung bei erkannter Adresse):
  Wenn das Experiment-Interface seine Adresse entschlüsselt hat, so kann es diese Leitung auf "low" ziehen, um anzuzeigen, dass das adressierte Register existiert. Nicht alle Interfaces nutzen diese Option. Eine alte Version des Address-Decoders hingegen bedient dieses Signal bereits wenn die Moduladresse stimmt (Siehe Abschnitt 3.2 [Address-Decoder], Seite 5.).

- **Enable_Read** (Anforderung von Eingabedaten):
  Mit diesem Signal öffnet das Interface die Datengatter des adressierten Registers und gibt die Daten auf den Routing-Bus.

- **Data_Accepted** (Quittung der Dateneingabe):
  Mit diesem Signal (200ns) meldet die Routing-Steuerung die erfolgte Übernahme der Eingabedaten. Das Interface kann dieses Signal ignorieren oder z.B. sein Datenregister löschen.

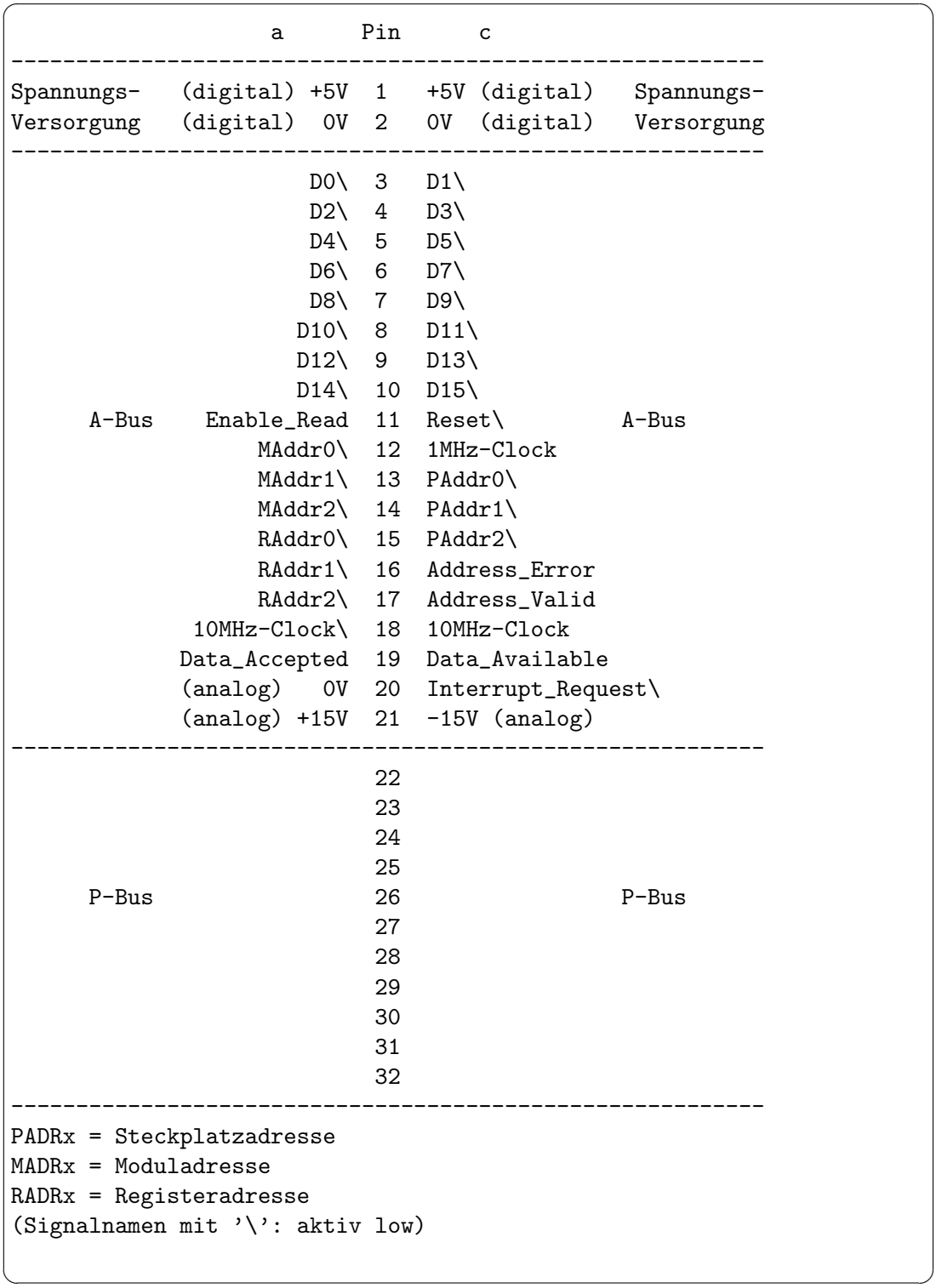
- **Data_Available** (Ausgabedaten stehen bereit):
  Mit diesem Signal (200ns) zeigt die Routing-Steuerung an, dass Ausgabedaten auf dem Routing-Bus zur Übernahme bereit stehen.

- **Interrupt_Request** (Interrupt-Anmeldung):
  Ein Low-Pegel auf dieser Leitung meldet bei der Routing-Steuerung einen Interrupt-Wunsch an. Diese Leitung kann auch über die Statuseingabe abgefragt werden. Dies ist eine Sammelleitung für ITs und muss deshalb mit 'Open-Collector' oder 'Tri-State' Bausteinen angesteuert werden (Wired-Or-Schaltung). Falls unterschiedliche ITs auftreten, können diese nur unterschieden werden, wenn sie in Registern gespeichert werden, die durch die Software abgeprüft werden können (Siehe Abschnitt 4.7 [Interrupt Eingabe], Seite 21.).

Privat-Bus (P-Bus)

Der Privat-Bus baut sich erst durch Stecken der Karten auf. Eine fehlende Karte oder das Fehlen entsprechender Brücken auf den Karten unterbrechen den Privat-Bus. Er reicht von den Leitungen 22 bis 32 einschließlich. Die Leitungen mit dem Suffix a führen zur linken Nachbarkarte und die mit c zur rechten. Soll ein Bus aufgebaut werden, so sind die Anschlüsse a und c einer Leitung zu brücken. Die individuelle Nutzung des Privat-Busses ist den einzelnen Kartenbeschreibungen zu entnehmen.

Abb.: Control-Routing-Bus



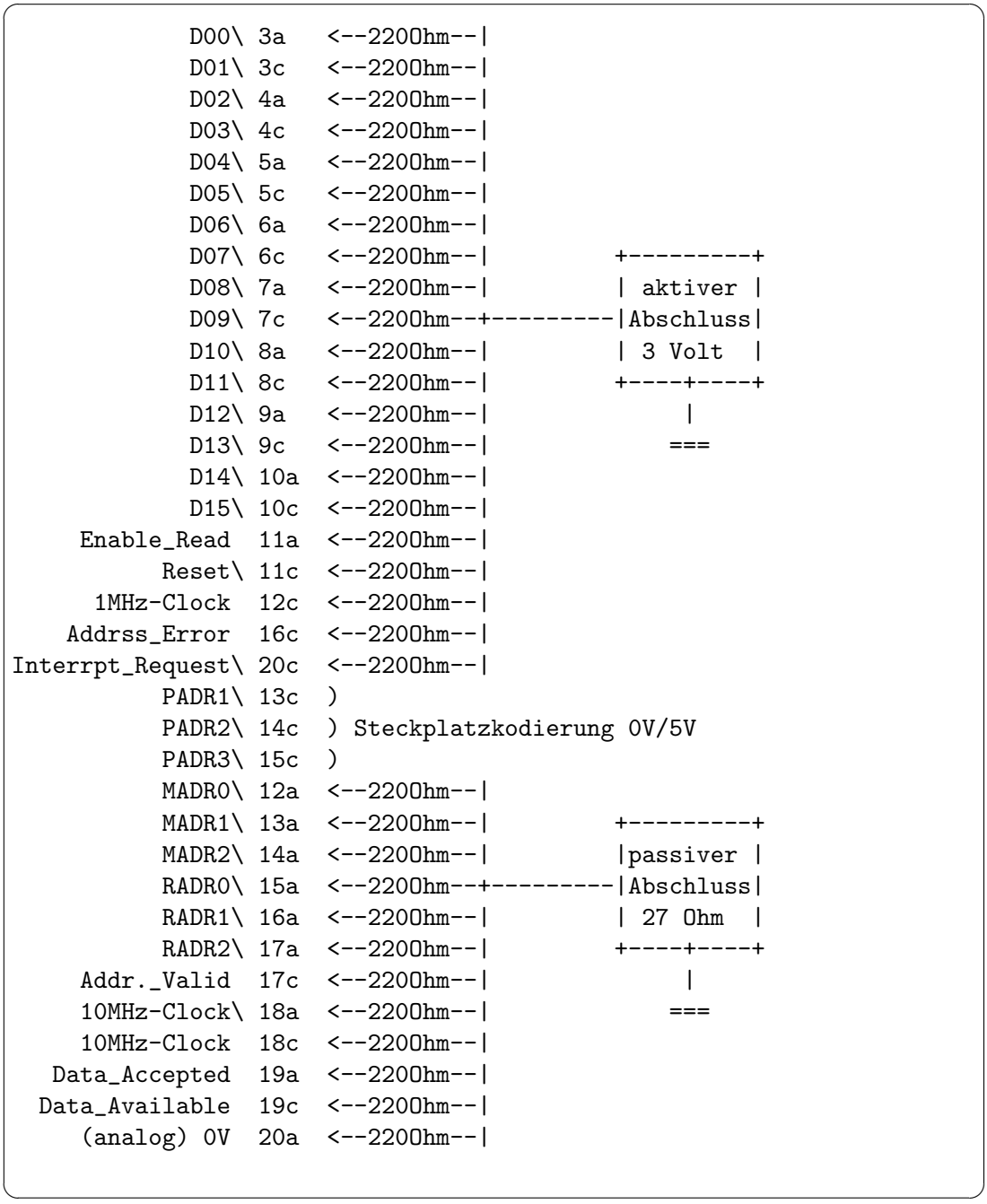
a		Pin	c	
Spannungs-	(digital) +5V	1	+5V (digital)	Spannungs-
Versorgung	(digital) 0V	2	0V (digital)	Versorgung
	D0\	3	D1\	
	D2\	4	D3\	
	D4\	5	D5\	
	D6\	6	D7\	
	D8\	7	D9\	
	D10\	8	D11\	
	D12\	9	D13\	
	D14\	10	D15\	
A-Bus	Enable_Read	11	Reset\	A-Bus
	MAddr0\	12	1MHz-Clock	
	MAddr1\	13	PAddr0\	
	MAddr2\	14	PAddr1\	
	RAddr0\	15	PAddr2\	
	RAddr1\	16	Address_Error	
	RAddr2\	17	Address_Valid	
	10MHz-Clock\	18	10MHz-Clock	
	Data_Accepted	19	Data_Available	
	(analog) 0V	20	Interrupt_Request\	
	(analog) +15V	21	-15V (analog)	
		22		
		23		
		24		
		25		
P-Bus		26		P-Bus
		27		
		28		
		29		
		30		
		31		
		32		

PADRx = Steckplatzadresse
MADRx = Moduladresse
RADRx = Registeradresse
(Signalnamen mit '\': aktiv low)

7.1.2 Routing-Bus Abschluss

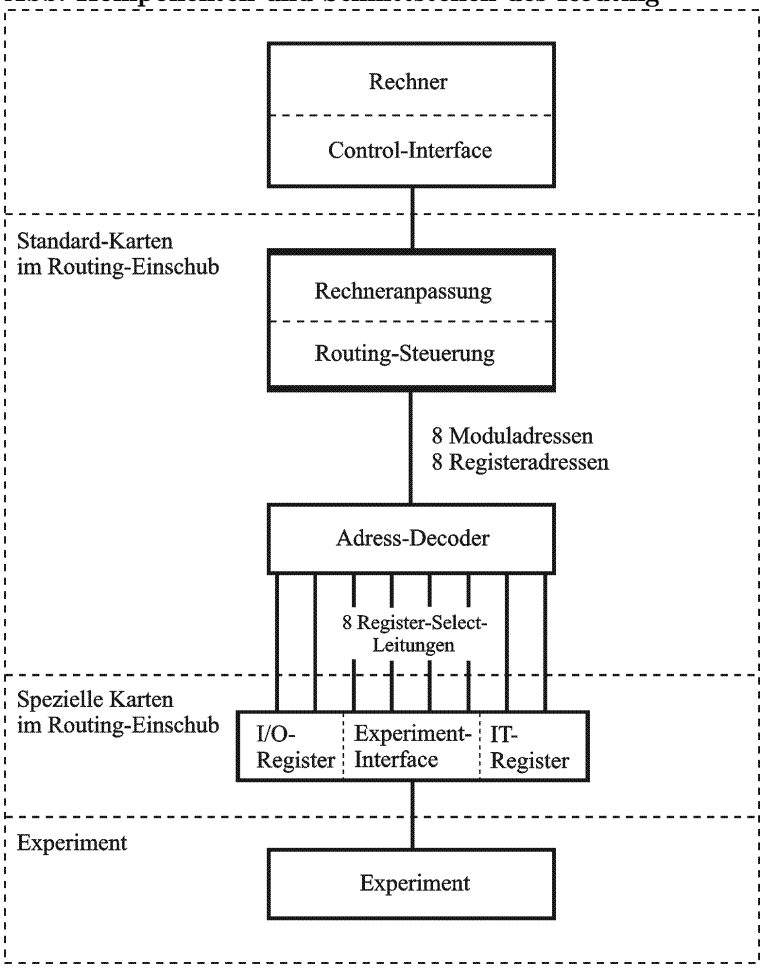
Der Bus-Abschluss ist normalerweise ganz links auf der Rückseite der ersten VG-Steckerleiste installiert. Es bestehen jedoch auch Sonderlösungen durch von vorne gesteckte Karten, insbesondere wenn der Überrahmen in einen Data-Routing-Bus und einen Control-Routing-Bus aufgeteilt ist.
Der Bus-Abschluss ist je nach Aufgabe der Bus-Leitungen als aktiver Abschluss (ca. 3V) oder durch Pulldown-Widerstände realisiert (Siehe [Abb Routing-Bus-Abschluss], Seite 58.). Der 27 Ohm Widerstand, mit denen die Pulldowns zusammengefasst sind, ist experimentell ermittelt und verbessert die Signale ganz erheblich. Insbesondere vermindert er ein Übersprechen, das im Data-Routing zum Ausfall einzelner 10MHz-Takte beim Schalten der *SADR(0:3)*-Leitungen geführt hatte. Warum das Ganze gerade in dieser Beschaltung ordentlich läuft, weiß keiner so recht. Die von den Interface-Karten erzeugten Signale können mittels Treiber-Bausteinen mit mindestens 15mA (besser 24mA) Low-Signalstrom (z.B. 'LS245, 'LS373, 'LS374) auf den Bus gegeben werden.
Bei der Entwicklung von Interface-Karten bitte stets darauf achten, dass sowohl **Sender- als auch Empfänger-Bausteine möglichst nahe am Bus platziert werden!** Denn ein solcher Bus ist eine sehr heikle Hf-Übertragungsstrecke (ca. 20MHz), die man durch falsch aufgebaute Steckkarten empfindlich stören kann.

Abb.: Routing-Bus-Abschluss



7.2 Komponenten und Schnittstellen

Abb. Komponenten und Schnittstellen des Routing



7.2.1 Address-Decoder

Zur Erleichterung des Anschlusses von Experiment-Interfaces wurde eine standardisierte Karte zur Entschlüsselung der binär verschlüsselten Modul- und Register-Adressen entwickelt. Die Address-Decoder-Karte belegt eine Steckplatz abhängige Moduladresse (3 Bits) und entschlüsselt die zugehörigen Registeradressen (3 Bits). Über den P-Bus (VG-Stecker Pins 22c - 29c; Siehe Abschnitt 7.1 [Routing-Bus], Seite 54.) gibt sie 8 Register-Select-Signale an rechts nachfolgende Experiment-Interfaces weiter. Ihr Einsatz kann entfallen, wenn die Experiment-Interfaces die Entschlüsselung selber vornehmen.

Sobald das Signal "Address valid" anliegt, vergleicht die Address-Decoder-Karte ihre Steckplatzadresse PADR<0-2> mit der aktuellen Moduladresse MADR<0-2>. Bei Gleichheit aktiviert sie einen 3 zu 8 Demultiplexer, der aus der codierten Registeradresse RADR<0-2> 8 einzelne Register-Select-Signale erzeugt (Siehe Abschnitt 2.1 [Aufbau des Routing], Seite 2.), die sie über den P-Bus (Siehe Abschnitt 7.1 [Routing-Bus], Seite 54.) den benachbarten Experiment-Interfaces zur Verfügung stellt.

Die über den P-Bus angeschlossenen Experiment-Interfaces nutzen die Register-Select-Signale zusammen mit den Routing-Bus-Signalen "Enable Read", "Data Accepted" und "Data Available" zur Steuerung der Ein- und Ausgabe von Daten.

Es existieren zwei Versionen der Address-Decoder-Karte, die sich in den Bedienungselementen und Anzeigen und zum Teil auch in der Funktion unterscheiden:

Bedienungselemente:
alte Version:

```
Schalter: Run/Stop
    Run :  Die Address-Decodierung ist in Betrieb
    Stop:  Die Address-Decodierung ist außer Betrieb und alle
           Register-Select-Signale sind abgeschaltet (high).
```

neue Version: keine Bedienungselemente

Anzeigen:
alte Version:

```
LED : Select
      LED leuchtet: Address-Decoder-Karte ist durch Moduladresse selektiert
LEDs: 4 2 1
      Anzeige der Steckplatzadresse
      LED leuchtet: Bit der angegebenen Wertigkeit ist gesetzt
```

neue Version:

```
LEDs: 0 1 2 3 4 5 6 7
      Register select
      LED n leuchtet: Register n ist selektiert
LEDs: 4 2 1
      Anzeige der Steckplatzadresse
      LED leuchtet: Bit der angegebenen Wertigkeit ist gesetzt
```

Funktionelle Unterschiede:
alte Version:
Ursprünglich war geplant, dass das "Address_Error"-Signal (Routing-Bus VG-Stecker Pin 16c) von jedem Experiment-Interface gelöscht wird, dessen Register adressiert werden. Tatsächlich wurde jedoch auf der alten Address-Decoder-Version "Address_Error" bereits gelöscht, wenn diese über die Moduladresse angesprochen wurde, und die Experiment-Interfaces haben möglicherweise das "Address_Error"-Signal nicht bedient. Damit kann jedoch die Software nicht feststellen, ob eine benötigte Experiment-Interface-Karte fehlt oder defekt ist. Durch Auftrennen der Verbindung nach Pin 16c auf dem Address-Decoder ist diese Panne zu beheben. Dann müssen gegebenenfalls aber die Experiment-Interfaces nachgerüstet werden (s.u.).

neue Version:
Die neue Address-Decoder Version überlässt das Löschen des "Address_Error"-Signals den Experiment-Interfaces, die dann zum Teil aber nachgerüstet werden müssen (teilweise bereits geschehen), falls die Software "Address_Error" abprüft.

Nachrüstung:
Als Nachrüstung genügt eine Diode vom Eingang der verwendeten "Register Select"-Signale zum Ausgang des "Address_Error"-Signals, z.B.:

```
22c ---|<|--- 16c
```

7.2.2 Address-Decoder <-> Routing-Steuerung

7.2.3 Routing-Steuerung

7.2.4 USB Rechner-Interface

Mit dem UpGrade auf *VxWorks 7* werden die MVME-Rechner nicht mehr unterstützt, es besteht jedoch die Möglichkeit für eine Rechneranpassung über USB, so dass PCs als Laborrechner zum Einsatz kommen können.

7.2.4.1 USB Eigenschaften

USB transportiert die Daten nicht Byte für Byte sondern in Paketen, die sich mit konkurrierenden Paketen die Bandbreite des USB teilen müssen. Dies führt dazu, dass die Steuerung eines Experimentes 'holprig' werden kann. Man kennt diesen Effekt von der USB-Mouse am Bildschirm.

Als USB-Controler auf der Routing-Seite ist der FT2232H-Chip im Einsatz. Er verwendeten den Bulk-Mode (Massendaten) für den Datentransport. Für USB2 ergeben sich damit folgende Werte:

```
Die Daten werden in Pakete zu 512 Bytes gepackt.
Maximal 15 Pakete werden in Frames zu 125us transportiert.
==> 15 * 0.5kB * 8000/s = 60MB/s
```

Normalerweise schaffen dies aber die beteiligten Controler schon nicht.
Der FT2232H kann laut Datenblatt im verwendeten 'Parallel FIFO Mode' 8Mb/s übertragen.
Die Dateneingabe des Data-Routing erreicht 4.4Mb/s wenn die Daten (4 Bytes) vom FPGA auf dem Routing Controler Board erzeugt werden.

Um den Datendurchsatz zu optimieren hat USB den Latency-Timer, der dafür sorgt, dass die Pakete gut gefüllt werden, indem er die Übertragung verzögert. Für die Übertragung von einzelnen Bytes kann dies aber ziemlich hinderlich sein für die Antwortzeiten bei der Experimentsteuerung. Für Maus und Tastatur hat USB deshalb einen eigenen Transfer-Mode.
Der Latency-Timer des FT2232H ist programmierbar (min 10ms, default 160ms).

Eine weitere Einschränkung ergibt sich dadurch, dass USB keine Interrupts übertragen kann und diese deshalb beständig abgefragt werden müssen.

Alle diese Hindernisse müssen bei der Programmierung berücksichtigt werden, um ausreichende Antwortzeiten zu erhalten (s.h. Programm USBtst). Eine entscheidende Hilfe sind dabei die I/O-Buffer und der programmierbare Logik-Baustein (FPGA: ispLSI 1024E) auf dem Routing-Controler-Board.

```
Ring-Buffer im Rechner (2/Port): 128kB
    Die Ring-Buffer können überlaufen.
    Füllstand geprüft mit: ioctl(), usb2SerialRingBfrSize
I/O-Buffer des FT2232H (2/Port):   4kB
    Die I/O-Buffer laufen nicht über, FT2232H regelt das?
Taktfrequenz des FPGA:   10MHz
```

Die Ring-Buffer können überlaufen. Die Software kann mit:

Wenn möglich, sollte man USB-Endgeräte mit hoher I/O-Rate nicht auf das selbe Bus-System legen. PCs bieten meist mehrere davon an, was aber den Anschlüssen nicht anzusehen ist. Mit dem USBtst-Programm lässt sich die USB-Konfiguration ausgeben.

7.2.4.2 Data-Transfer (USB)

Das Control Routing wird zur Steuerung eines Experimentes eingesetzt, indem Register geladen und gelesen werden und Interrupts behandelt werden. Im Gegensatz zu dem Flachbandkabel-Anschluss der VME-Version überträgt USB jedoch keine einzelnen Zeichen sondern Datenpakete und kennt keinen Interrupt. Auch werden die Daten nicht direkt übertragen sobald sie anstehen sondern der USB Host Controler auf der Rechnerseite pollt das Routing an, um Empfangs- bzw. Sendebereitschaft abzufragen. Weitere Aktivitäten auf dem USB-Bus können ebenfalls zu einer Verzögerung der Datenübertragung führen.

Um trotzdem eine glatte Ablaufsteuerung des Experimentes zu erreichen, wurde folgender Weg eingeschlagen:
Die Routing-Seite ist mit einem programmierbaren Logikbaustein (FPGA) versehen, der als Ablaufsteuerung mittels USB-Port des FT2232H (Data-Port) übertragene Kommandos verarbeiten kann:

```
Lade übertragene Daten in adressiertes Register.
Lese Daten aus adressiertem Register und übertrage sie.
Warte auf Interrupt und übertrage eine Meldung.
Warte auf Ereignis (z.B. ADC fertig) und übertrage eine Meldung.
```

Da rechnerseitig für Ein- und Ausgabe je ein 128kB Ring Buffer eingerichtet ist und der USB-Chip (FT2232H) je einen 4kB Puffer bereitstellt, kann vom Rechner ein umfangreicher Messzyklus als Paket (buffered I/O) für eine reaktionsschnelle Ablaufsteuerung abgeschickt werden.

Über einen zweiten USB-Port des FT2232H (Control-Port) können folgende Kommandos an die Ablaufsteuerung gegeben werden:

```
Statusabfrage.
Start/Stop des Routing.
Abbruch eines anstehenden Wait-Kommandos.
Löschen eines gespeicherten Interrupts.
```

Mittlere Antwortzeiten:
Für das Control-Routing sind sowohl Ausgaben als auch Eingaben über USB erforderlich. So muss z.B. jede Registerabfrage zuvor mit einer entsprechenden Ausgabe (je 4 Bytes) angefordert werden, also ein Hin und Her von wenigen Bytes.

Für die unterschiedlichen Verfahren bei der Programmierung haben sich für das Beschreiben und/oder Lesen eines Registers mit jeweils 4 Bytes folgende, über 100000 Wiederholungen gemittelte Antwortzeiten ergeben (s.h. Programm USBtst):

```
Write/Read synchron in einer Task
  write/read register:  68.4 us  (direct I/O)
      Ausgabe: Reg.-Write-Kommando (write(), direct I/O)
      Ausgabe: Reg.-Read-Kommando (write(), direct I/O)
      Eingabe: Datum (read(), direct I/O)

  write/read register:  44.8 us  (buffered I/O)
      Ausgabe: Reg.-Write-Kommando (fwrite(), buffered I/O)
      Ausgabe: Reg.-Read-Kommando (fwrite(), buffered I/O)
      Eingabe: Datum (read(), direct I/O)
```

```
  write only register:    26.2 us  (direct I/O)
      Ausgabe: Reg.-Write-Kommando (write(), direct I/O)
     (Rückmeldung am Ende der Verarbeitung durch den FPGA)

  read only register:     43.4 us  (direct I/O)
      Ausgabe: Reg.-Read-Kommando (write(), direct I/O)
      Eingabe: Datum (read(), direct I/O)

  write only register:    1.3 us  (buffered I/O)
      Ausgabe: Reg.-Write-Kommando (fwrite(), buffered I/O)
     (Rückmeldung am Ende der Verarbeitung durch den FPGA)

Write/Read asynchron in getrennten Tasks
  write/read register:  75.4 us  (direct I/O)
      Ausgabe: Reg.-Write-Kommando (write(), direct I/O)
      Ausgabe: Reg.-Read-Kommando (write(), direct I/O)
      Eingabe: Datum (read(), direct I/O)

  write/read register: 10.9 us  (buffered I/O)
      Ausgabe: Reg.-Write-Kommando (fwrite(), buffered I/O)\n"
      Ausgabe: Reg.-Read-Kommando (fwrite(), buffered I/O)\n"
      Eingabe: Datum (read(), direct I/O)\n"

  read only register: 10.6 us  (buffered I/O)
      Ausgabe: Reg.-Read-Kommando (fwrite(), buffered I/O)\n"
      Eingabe: Datum (read(), direct I/O)\n"
```

Wie zu erwarten, lässt sich die Antwortzeit durch Verwenden von Buffered I/O (fwrite) bei der Ausgabe deutlich verringern. Denn damit werden die Daten zunächst im Ring-Buffer angesammelt bevor sie als Paket mit fflush() an das USB übergeben werden. Mit Direct I/O hingegen werden die Kommandos an das Routing einzeln mit jedem write() abgeliefert und vom USB mehr oder weniger dicht zu Paketen verpackt.

7.2.4.3 Event-Abhandlung (USB)

Der Routing-Controler kennt zwei unterschiedliche Events:

- Interrupt Request
  Interrupt-Anforderungen werden von den Routing Boards über eine gemeinsame Leitung als Impuls oder dauerhaftes Signal an die Routing Kontrolle übertragen und dort im IT-Register gespeichert.
  Mit den Kommandos 'Wait for Event' und 'Clear IT Trap Register' kann das IT-Register gelöscht werden, wenn die IT-Leitung nicht gesetzt ist.
  Da das USB-Protokoll keinen Interrupt-Transport enthält, kann nur eine Statusabfrage (39us) den Rechner von einer anstehenden Interrupt-Anforderung informieren.
- Register Ready
  'Register Ready' wird auf einer gemeinsame Leitung angezeigt wenn ein I/O-Register adressiert ist. Es wird z.B. genutzt als Fertigmeldung eines ADC oder zum Prüfen,

ob ein Register existiert. Das 'Register Ready' Signal ist über eine Statusabfrage zugänglich.

Ferner wird der Status beider Event-Leitungen bei jeder Daten-Eingabe im Status-Byte mitgeliefert. Jedoch möglicherweise nicht aktuell sondern zum Zeitpunkt als die Routing-Steuerung die Eingabe abgesandt hatte.

Mittlere Antwortzeiten:

```
Read Status:     39.2 us
    Ausgabe: Read-Status-Kommando (1 Byte, write(), direct I/O)
    Eingabe: Status (2 Bytes, read(), direct I/O)
```

7.2.4.4 Rechneranpassung (USB)

FTDI FT2232H Mini Modul

Die USB-Anbindung wird mit einem *FTDI FT2232H MINI MODULE* realisiert. Es zeigt sich über USB beim PC mit 2 seriellen I/O-Ports. VxWorks 7 hat einen brauchbaren Treiber für den FT2232H-Chip.

Aus der Auswahl von Industriestandards, die der *FT2232H* IC bietet, ist insbesondere der folgende wegen der großen FIFOs für die Rechneranpassung geeignet:

```
Asynchronous FIFO Interface Mode
Dual Port FIFO TX Buffer (4 Kbytes per interface)
Dual Port FIFO RX Buffer (4 Kbytes per interface)
```

Der FT2232H Baustein des Mini Moduls muss mit Hilfe des *FTDI-FT_PROG* Windows-Programmes über USB konfiguriert werden.
Mit *'DEVICES -> Scan and Parce'* wird der Ist-Zustand des EEPROM ausgelesen und kann editiert werden.
Und mit *'DEVICES -> Program'* wird der EEPROM programmiert.
Mit *'File -> Save As Template'* kann die Konfiguration eines bereits programmierten FT2232H auf einen neuen übertragen werden.

Die Data-Routing Konfiguration unterscheidet sich vom Data-Routing nur in der Serial Number:

```
USB Device Descriptor
  Custom VID/PID:   FTDI Default
  Vendor ID:        0403
  Product ID:       6010
  USB Version:      USB 2.0

USB Config Descriptor -> bmAttributes
  Bus Powered:       yes
  Self Powered:      no
  Max Bus Power:     150 mA
  USB Remote Wakeup: no
  Pull Down IO...:   no

USB String Descriptors
  Manufacturer:      FTDI
  Product Descr.:    FT2232H Mini Modul
```

```
  Ser. Num. Enabled: yes
  Auto Gen. Ser. No: no
  Serial Number:     IAMP-CtrlRout
  Ser. Num. Prefix:

Hardware Spezific
  DPRDRV              : 0
  Suspend on DBus7 Low: no
  Port A/B -> Hardware: 245 FIFO
  Port A/B -> Driver:   Virtual COM Port
  I/O Pins -> Group.. -> Slow Slew: no
  I/O Pins -> Group.. -> Drive:     8mA
  I/O Pins -> Group.. -> Schmitt..: no
```

Durch eine Fehlbedienung kann der EEPROM unbrauchbar werden. Dann hilft nur ein 'Disaster Recovery'. Von FTDI gibt es eine Anleitung dazu. Benötigt wird ein intaktes Mini Modul und das defekte muss vermutlich aus der Schaltung raus...

7.2.4.5 Programmierung (USB)

Das Mini Modul bietet zwei Ports. Über Port-1 erfolgt die Kontrolle des Datentransfers und die Statusabfrage. Port-0 dient dem Datentransfer.

```
******** Control Routing Control Port Programmierung ********
Ein Kommando an den Control Port besteht aus einem Byte, in dem nur das
höchstwertige und die drei niederwertigsten Bits ausgewertet werden:
   "***** Control commands"
   [x,x,x,x,x,0,0,0] "Send status"
      Es werden zwei Bytes zurück gesendet:
      1. Header Byte: [0,1,0,0,0,0,1,1]
      2. Status Byte: [Stp,ItL,ItR,Rdy,wIt,wRy,x,x]
         Stp == 1: 'Stop Routing' ist gesetzt
         ItL == 1: 'Interrupt-Leitung' ist gesetzt
         ItR == 1: 'Interrupt-Register' ist gesetzt
         Rdy == 1: 'Register-Ready-Leitung' ist gesetzt
         wIt == 1: 'Wait for ItR' ist aktiv
         wRy == 1: 'Wait for Rdy' ist aktiv

   [Stp,x,x,x,x,0,0,1] "Clear control bits"
      Stp == 1: 'Stop Routing' wird gelöscht.

   [Stp,x,x,x,x,0,1,0]; "Set control bits"
      Stp == 1: 'Stop Routing' wird gesetzt.

   [x,x,x,x,x,0,1,1]; "Generate an event"
      Mit diesem Kommando kann ein endloses Warten auf ein Ereignis (wIt, wRy)
      im Data-Port beendet werden.

   [x,x,x,x,x,1,0,0]; "Clear interrupt Trap Register"
```

```
      Ein im Routing Controler gespeicherter Interrupt (ItR) wird gelöscht
      falls keine neue Interupt-Anforderung (ItL) ansteht.

******** Control Routing Data Port Programmierung ********
* Ein Kommando an den Data Port besteht aus vier Bytes:
   1. Header Byte: [0,1,1,0,0,0,1,1]

   2. Command/Address Byte: [c,c,a,a,a,a,a,a]
      Address [a,a,a,a,a,a]: Routing Register Adresse
      Commands [c,c]:
        [0,0];  "Read Data"
            Die beiden Daten Bytes enthalten keine Information.
            Das adressierte Register wird ausgelesen und das Datenwort
            in der Rückmeldung zurück übertragen.
        [0,1];  "Write Data"
            Die beiden Daten Bytes werden zu dem adressierten Register
            übertragen. Es erfolgt keine Rückmeldung.
        [1,0];  "Wait for Event"
            Mit diesem Kommando wird auf ein Routing Ereignis gewartet.
            Der Ereignistyp wird im High Byte angegeben:
                [0,x,x,x,x,x,x,x]; "Wait for Routing Interrupt"
                [1,x,x,x,x,x,x,x]; "Wait for Routing Board Ready"
            Wenn das Ereignis eintrifft erfolgt eine Rückmeldung. Da die
            Data Bytes unverändert rückübertragen werden, können die
            x-Bits zur Synchronisierung der Software bei mehreren Event
            Kommandos in einem Paket verwendet werden.
            Solange ein Warten aktiv ist, werden keine weiteren Kommandos
            bearbeitet. Das Warten kann über ein Software generiertes
            Ereignis durch ein Control Command abgebrochen werden (s.o.).
        [1,1];  "Echo data"
            Die beiden Daten Bytes werden mit dem Status des adressierten
            Registers zurück übertragen.

   3. Data High Byte:
   4. Data Low Byte:
        Bei jeder Ausgabe eines Kommandos werden die beiden Data Bytes in
        das Datenregister übernommen und je nach Typ des Kommandos
        verändert oder unverändert bei der Rückmeldung wieder zurück
        übertragen.

* Eine Rückmeldung des Data Ports besteht aus vier Bytes:
   1. Header Byte: [0,1,1,0,0,0,1,1]

   2. Status Byte: [ITR,!RDY,x,x,x,x,x,x]
         ITR == 1: Ein Routing Interrupt Request steht an.
        !RDY == 1: Das adressierte Register meldet 'Not Ready' oder fehlt.
```

3. Data High Byte:
4. Data Low Byte:
   Als Data Bytes wird der vom Kommando abhägige Inhalt des Datenregisters übertragen.

7.2.5 VME Rechner-Interface

7.2.5.1 Programmierung (VME)

Programmierung des Rechner-Interfaces

```
*** Address decoding:
    A2 A1    Read     Write
     0  0    Status   IP-Control    Transfer ohne Wait States
     0  1    ----     IT-Vektor     Transfer ohne Wait States
     1  0    Data     Data          Transfer mit Wait States
     1  1    ----     Rout-Contl    Transfer mit Wait States

*** Bit   IP-Board Control word:
    15    cReset      Reset
    14    cItEnab     Enable Interrupts
    13    cDatTest    IP-board data test loop on
    12    cItRep      Repeat Interrupt
    11-00   0         not used

*** Bit   Status word:
    15    cReset      Reset on
    14    cItEnab     Interrupts enabled
    13    cDatTest    IP-board data test loop on
    12      0         not used

    11      0         not used
    10      0         not used
    09      0         not used
    08    sOnline     Routing online

    07    sError      global error flag
    06    sItErr      IT enabled but IT vector not set
    05    sBreak      Routing online break down detector
    04    sTimeout    Data transfer timeout

    03    sAdInv      Modul address invalid
    02    sItReq      Interrupt reqüst from Routing
    01    sItvOK      Interrupt vector set
    00      0         not used

*** Bit Routing-Board Control word:
    15      0          not used
    14    (cNoItTrap)  Projekt: no trapping of Interrupt pulses
    13     cDatTest    Rout.-board data test loop on
    12-07   0        not used
    06-04  MAddr02-00  Rout.-Bus modul address
    02-00  RAddr02-00  Rout.-Bus register address
```

7.2.5.2 Data Transfer Protokoll (VME)

(huber@ionix:~/servix/vme/IspLSI/cntlrout/vmeCntlRout)

```
Geschwindigkeitstest mit Programm "vmetst" (1000000 Transfers):
Data Write/Read
    IP board write/read      : 5.3us/Data für MVME162
    Routing board write/read : 7.4us/Data für MVME162, 40m
    Routing board write/read : 6.9us/Data für MVME162,  5m
    Test board write/read    : 7.4us/Data für MVME162, 40m
    Test board write/read    : 7.0us/Data für MVME162,  5m
    Test board read only     : 6.0us/Data für MVME162, 40m
    Test board read only     : 5.7us/Data für MVME162,  5m
Data Write/Read -> Interrupts
    IT handler only          : 11.3us/IT für MVME162,  5m
    IT + Signal handler      : 140.us/IT für MVME162,  5m

IP-Board:
    Data Transfer Master
    Sender, Receiver
    8 Mhz Clock

Routing Board:
    Data Transfer Slave
    Receiver, Sender
    10 Mhz Clock
```

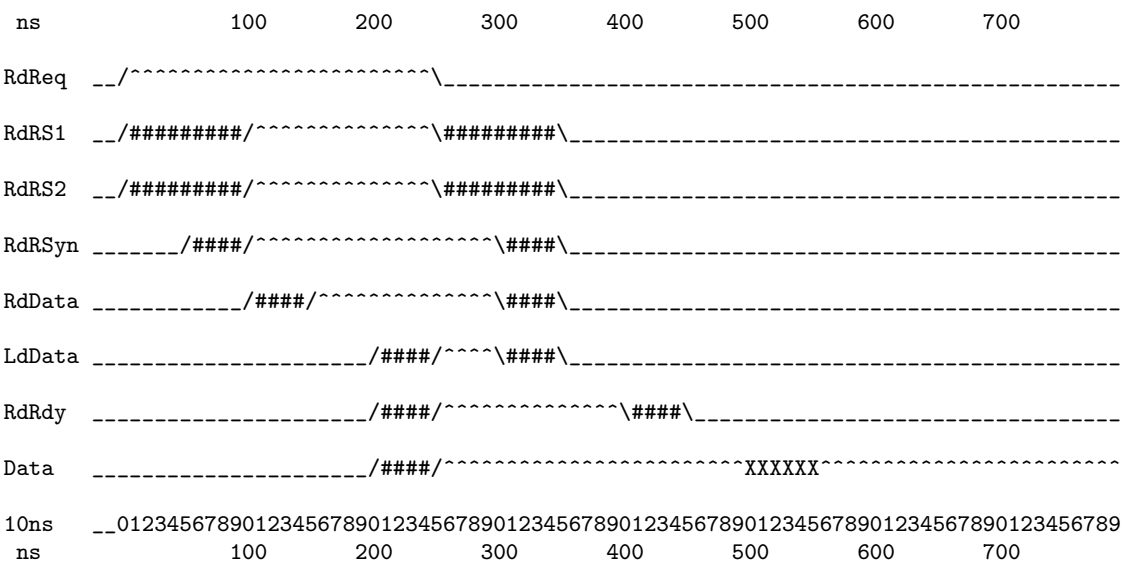
Data Transfer Protokoll (neue Variante 11.05.98)
********* Data from Routing:**
Der Receiver (Master) zeigt mit RdReq an, dass er zur Datenaufnahme bereit ist. Der Sender (Slave) antwortet mit der Übertragung von Daten. Es werden 2 * 8 Bits übertragen mit RdRdy einmal zu Beginn. RdRdy und erstes Daten-Byte gehen gleichzeitig auf die Leitung, dabei kann RdRdy gestört werden. Durch eine solche Störung kann sich das Erkennen von RdRdy jedoch nur maximal für die Dauer der Störung (ca. 20ns?) verzögern. Nach RdReq wartet der Receiver max. 8 * 125ns = 1000ns auf die RdRdy-Antwort des Senders. Das begrenzt die max. Kabellänge auf ca. 50m.

Timing: Data from Routing (Routing-Board)

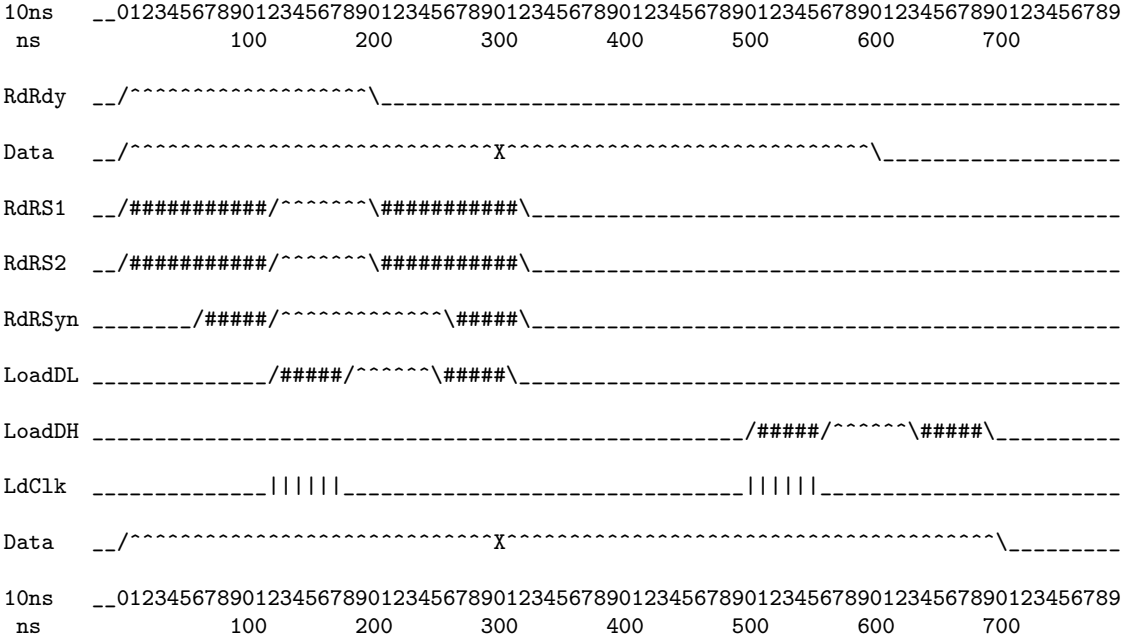
```
        2. Data-Byte verlängert wegen Reserven zum Ende hin!
20ns   __0246802468024680246802468024680246802468024680246802468024680246802468024680
 ns          100  200  300  400  500  600  700  800  900  000  100  200  300  400  500

RdRdy  __/^^^^^^^^^\_____________________________________________________________

Data   __/^^^^^^^^^^^^^^^X^^^^^^^^^^^^^^^^^^\_____________________________________

20ns   __0246802468024680246802468024680246802468024680246802468024680246802468024680
 ns          100  200  300  400  500  600  700  800  900  000  100  200  300  400  500
```

Timing: Data from Routing (Routing-Board)

```
10ns   __0123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789
```



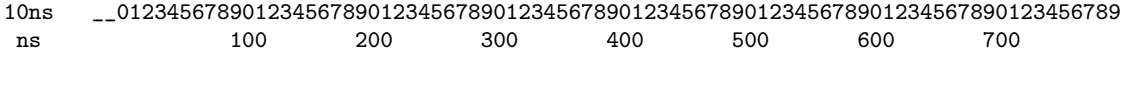
Timing: Data from Routing (IP-Board)

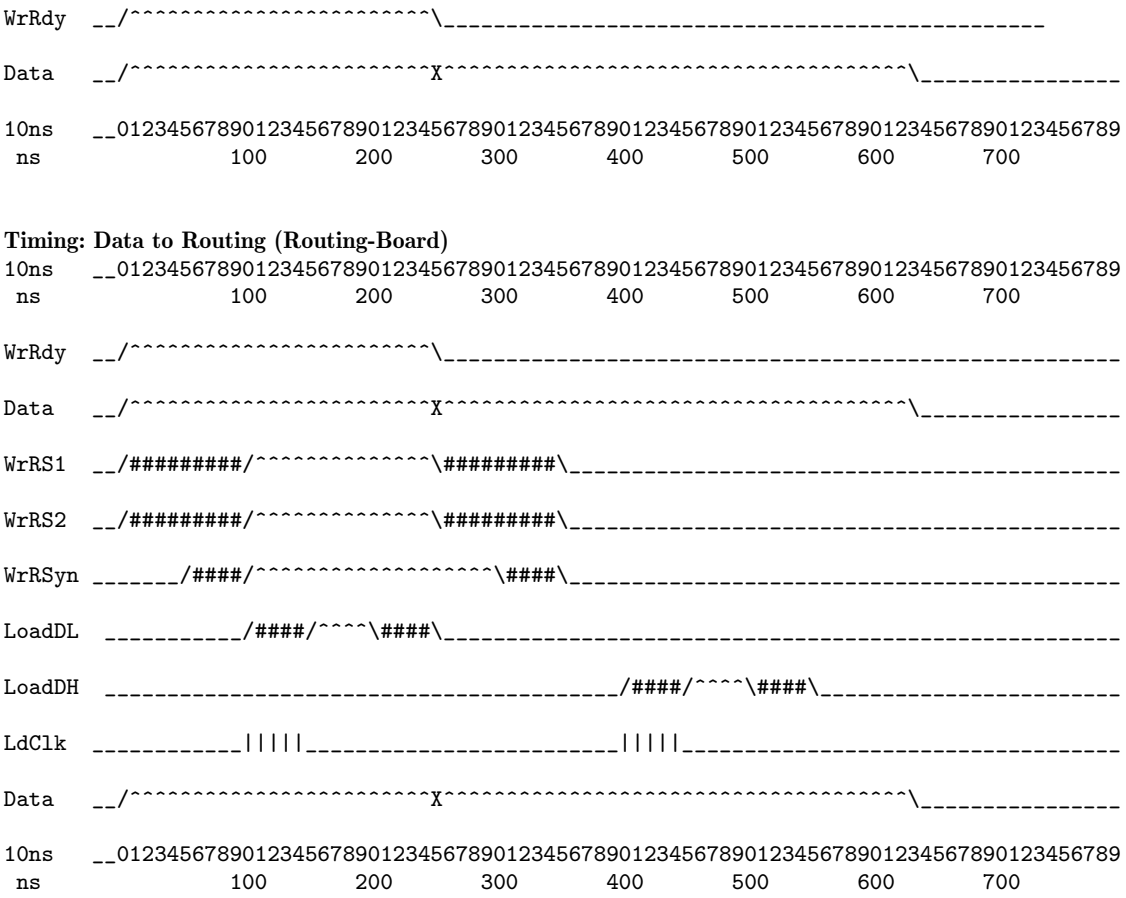


********* Data to Routing:**

Der Sender (Master) zeigt mit WrRdy an, dass er Daten sendet. Der Receiver (Slave) synchronisiert sich zum Empfang der Daten ein. Es werden 2 * 8 Bits übertragen mit WrRdy einmal zu Beginn. WrRdy und erstes Daten-Byte gehen gleichzeitig auf die Leitung, dabei kann WrRdy gestört werden. Durch eine solche Störung kann sich das Erkennen von WrRdy jedoch nur maximal für die Dauer der Störung (ca. 20ns?) verzögern.

Timing: Data to Routing (IP-Board)





Timing: Data to Routing (Routing-Board)

7.3 Schaltungsunterlagen

7.3.1 Board Interface-Steuerung

7.3.2 Board Routing-Steuerung

7.3.3 Board Rechneranpassung

7.3.4 Board Rechner-Interface

8 Oldies

8.1 Alte Routing-Steuerung

Diese Variante der Routing-Steuerung war an TR86-, PDP11- und VME-Systemen im Einsatz. Sie ist inzwischen durch eine Karte mit einem hochintegrierten, programmierbaren Baustein und verbesserter Funktionalität ersetzt worden.

8.1.1 Funktion

8.1.2 Bedienungselemente, Anzeigen

8.1.3 Schnittstellen

8.1.4 Routing-Bus

8.2 Alte Rechneranpassungen

8.2.1 PDP11 - DRV11-J - Anpassung

8.2.1.1 Adressausgabe und Statuseingabe

Zur Adressausgabe wird zunächst Port C der DRV11-J auf Ausgabe programmiert (falls er nicht bereits auf Ausgabe steht) und anschließend die Modul- und Registeradresse und ggf. das Reset-Bit ausgegeben. Die Ausgabe wird in der Routing-Steuerung gespeichert und bleibt gültig bis zur nächsten Ausgabe.

Zur Statuseingabe wird zunächst Port C auf Eingabe programmiert (falls er nicht bereits auf Eingabe steht) und anschließend das Statuswort eingegeben.

Aufbau von Adress- und Statuswort

Bit	Adressausgabe	Statuseingabe
0	RADR0\	Online
1	RADR1\	Address Error
2	RADR2\	Interrupt
3	xxx	0
4	MADR0\	0
5	MADR1\	0
6	MADR2\	0
7	xxx	0
8	Reset-Bit	xxx
9-15	xxx	xxx

RADR<0-2>\	3 Registeradress-Bits (invertiert)
MADR<0-2>\	3 Moduladress-Bits (invertiert)
Reset-Bit	Programmierung eines Reset des DRV11-Routing. Die Experiment-Interfaces können dieses Reset benutzen, müssen aber nicht.
Online	Das DRV11-Routing ist angeschaltet.
Address Error	Die angesprochene Moduladresse existiert nicht, oder die Adress-Decodierung bedient diese Leitung nicht.
Interrupt Request\	Zustand der Interrupt Request Leitung (invertiert). 0: ein angemeldeter Interrupt ist noch nicht bearbeitet.
xxx	unbenutztes bzw. undefiniertes Bit

8.2.1.2 Interrupt-Eingabe

Zur Anmeldung eines Interrupt-Wunsches (Exp.-IT) wird von dem betroffenen Experiment-Interface die Interrupt Request (IR-) Leitung auf low gezogen. Mit dem high/low-Übergang wird über die DRV11-J Karte (USER RPLY C, IMR-Bit 6) bei der PDP11 eine Programmunterbrechung angemeldet. Falls nicht gerade eine Programmunterbrechung mit gleicher oder höherer Priorität aktiv ist, wird die zugehörige Interrupt-Service-Routine (ISR), falls vorhanden, aktiviert, um den Interrupt abzuhandeln.

Wenn die IR-Leitung auf low gehalten wird, kann die IT-Anmeldung über eine Routing-Statuseingabe abgefragt werden. Dieses Polling-Verfahren ist zwar reaktionslangsamer, aber es erspart den Aufwand für eine Interrupt-Service-Routine.

Da nur eine einzige IR-Leitung zur Verfügung steht, müssen beim Auftreten unterschiedlicher Exp.-ITs diese unterscheidbar gemacht werden. Dies geschieht z.B. durch Speichern der einzelnen ITs in I/O-Registern (IT-Register) (Abb. 3.2.6.6). Ist ein Exp.-IT gespeichert, so zieht er die IR-Leitung solange auf low, bis er bearbeitet ist.

Der erste eintreffende Exp.-IT löst durch den high/low-Übergang auf der IR-Leitung eine Programmunterbrechung aus, d.h. die ISR wird aktiviert. Diese führt dann folgende Aufgaben aus:

- Durch Statuseingabe prüft sie, ob es einen Exp.-IT zu bearbeiten gibt. Möglicherweise ist nämlich der zu dieser Programmunterbrechung gehörige Exp.-IT bereits bei dem vorherigen ISR-Durchlauf bearbeitet worden. Falls kein Exp.-IT ansteht, ist ihre Aufgabe bereits erledigt.

- Falls ein Exp.-IT ansteht, beginnt die ISR alle IT-Register auszulesen bis sie einen gesetzten Exp.-IT findet. Diesen bearbeitet sie und löscht das IT-Register (bei einer Schaltung nach Abb. 3.2.6.6 ist dies bereits durch das Lesen erfolgt!).

- Anschließend beginnt sie wieder mit der Prüfung, ob noch ein IT-Register gesetzt ist usw. bis alle ITs gelöscht sind.

Das Ganze funktioniert nur reibungslos, wenn die
"PS<7:5>"-Priorität der ISR größer oder gleich ist der "Device
Interrupt Priority" des DRV11-J Interfaces. Darauf ist
unbedingt zu achten, damit es keine Überholvorgänge bei den
Interrupts gibt.

Varianten zu diesem Vorschlag sind ausdrücklich erlaubt!

Dies ist fast alles lediglich (durch entsprechende
Dokumentation untermauerte) Theorie und muss im Detail noch
einmal richtig durchgeprüft werden!

8.2.1.3 Programm-Beispiele

Beispiel 1: Read/Write Data; Read/Clear Interrupt
Über Port C erfolgt die Adressausgabe und die Statuseingabe,
über Port D die Datenein-/ausgabe. Der in einem Register
gespeicherte Interrupt wird über die Statuseingabe abgefragt
und über einen READ auf dieses Register gelöscht. Es wird
keine Interrupt-Routine eingesetzt.

```
; PDP11 MACRO Program

;Device addresses of DRV11-J
DRVCSA=:   164160
DRVDBA=:   164162
DRVCSB=:   164164
DRVDBB=:   164166
DRVCSC=:   164170
DRVDBC=:   164172
DRVCSD=:   164174
DRVDBD=:   164176
DRVVEC=:   274     ;Vector address of DRV11-J

DTADDR=:   000022  ;Address ('12'H) of Data: Modul = 1, Register = 2
ITADDR=:   000064  ;Address ('34'H) of Interrupt:
                   ;Modul = 3, Register = 4
DATA=:     123456  ;Data to be output

; System macros:
       .MCALL  EXIT$S
       DRERR$      ;Directive error symbols



START::
```

```
; I/O of data
       CLRB    DRVCSA          ;Reset DRV11 group 1
       CLRB    DRVCSC          ;Reset DRV11 group 2
       BISB    #1,DRVCSC+1     ;Port C for output
       MOV     #DTADDR,R0      ;Load modul/register address
       COMB    R0              ;Invert address bits
       MOV     R0,DRVDBC       ;Output of address
       BICB    #1,DRVCSC+1     ;Port C for input
       BIT     #1,DRVDBC       ;Test Routing status
       BEQ     99$             ;Routing offline
       BIT     #2,DRVDBC       ;Test Routing status
       BNE     99$             ;Modul not found
       BISB    #1,DRVCSD+1     ;Port D for output
       MOV     #DATA,DRVDBD    ;Write data
       BICB    #1,DRVCSD+1     ;Port D for input
       MOV     DRVDBD,R0       ;Read data

; Polling for Interrupt
       BICB    #1,DRVCSC+1     ;Port C for input
       BIT     #4,DRVDBC       ;Test Routing status
       BNE     99$             ;No interrupt found
       BISB    #1,DRVCSC+1     ;Port C for output
       MOV     #ITADDR,R0      ;Load modul/register address
       COMB    R0              ;Invert address bits
       MOV     R0,DRVDBC       ;Output of address
       BICB    #1,DRVCSC+1     ;Port C for input
       BIT     #2,DRVDBC       ;Test Routing status
       BNE     99$             ;Modul not found
       BICB    #1,DRVCSD+1     ;Port D for input
       MOV     DRVDBD,R0       ;Read and clear interrupt

99$:   EXIT$S
       .END START
```

Beispiel 2: Abhandlung eines Interrupts durch Interrupt-Routine
Die Interrupt-Enable/Disable-Routine und die
Interrupt-Service-Routine werden über das Page-Register 5
in den Adressraum des Betriebssystems gemapped. Diese
Programmteile müssen in "position independent code"
geschrieben werden!

```
; PDP11 MACRO Program

;Device addresses of DRV11-J
DRVCSA=:   164160
```

```
DRVDBA=:   164162
DRVCSB=:   164164
DRVDBB=:   164166
DRVCSC=:   164170
DRVDBC=:   164172
DRVCSD=:   164174
DRVDBD=:   164176
DRVVEC=:   274        ;Vector address of DRV11-J

ITADDR=:   000064     ;Address ('34'H) of Interrupt:
                      ;Modul = 3, Register = 4
EFN=:10.              ;Eventflag, set after interrupt


; System macros:
       .MCALL  CINT$S,ASTX$S,EXIT$S
       .MCALL  CLEF$S,SETF$S,WTSE$S
       DRERR$                 ;Directive error symbols



START::
       ;Reset DRV11J
       CLRB    DRVCSA         ;Reset Group 1
       CLRB    DRVCSC         ;Reset Group 2

       ;Test Routing
       BICB    #1,DRVCSC+1    ;Input on port C
       BIT     #1,DRVDBC      ;Test Routing status
       BEQ     99$            ;Routing offline

       ;Load DRV11-J mode bits: fixed prio, common vector,
       ;interrupt, polarity low
       MOVB    #202,DRVCSC

       ;Clear IMR bit 6, enable interrupt USER RPLY C
       MOVB    #056,DRVCSC

       ;Enable Auto-clear
       MOVB    #300,DRVCSC    ;Preselect ACR for writing
       MOVB    #100,DRVCSD    ;Set auto-clear bit 6

       ;Load vector address memory
       MOVB    #340,DRVCSC    ;Preselect VAM 0 for writing
       MOV     #DRVVEC,R0
       ASH     #-2,R0
       MOVB    R0,DRVCSD      ;Load vector address memory 0
```

```
        ;Set master mask bit, enable interrupts
        MOVB    #241,DRVCSA
        MOVB    #241,DRVCSC

        ;Connect to DRV11-J interrupt
        CINT$S  #DRVVEC,#P5BASE,#DRVISR,#DRVEDI,#PR4,#DRVAST
        BCS     99$             ;Error

        ;Wait for interrupt
        CLEF$S  #EFN            ;Clear EFN
        WTSE$S  #EFN            ;Wait for eventflag EFN set
        ;Reset Routing and DRV11J
        BISB    #1,DRVCSC+1     ;Output on port C
        MOV     #400,DRVDBC     ;Reset Routing
        CLRB    DRVCSA          ;Reset DRV11J group 1
        CLRB    DRVCSC          ;Reset DRV11J group 2

        ;Disconnect DRV11J interrupt
        CINT$S  #DRVVEC,#0,#0
        BCS     99$             ;Error

99$:    EXIT$S


         ;Executive APR5 mapped routines, AST routine
        .PSECT  AP5SCT  ;Programm section mapped to EXEC by APR5
P5BASE::                ;Base for APR5 mapping by executive

TSKTCB: .WORD   0       ;TCB address of task

; Enable/disable DRV11-J interrupts, Exec APR5 mapped

DRVEDI::        ;Needs position independent code!
        BCS     10$              ;Disconnect
        MOV     @#$TKTCB,TSKTCB ;Get TCB for later
        BISB    #2,@#DRVCSA+1   ;Enable interrupt
        RETURN

10$:    BICB    #2,@#DRVCSA+1   ;Disable interrupt
        RETURN



; DRV11-J interrupt service routine, FORK routine, Exec APR5 mapped

DRVISR::        ;Needs position independent code!
```

```
        MOV     R0,-(SP)          ;;;Save R0
        BISB    #1,@#DRVCSC+1    ;;;Port C for output
        MOV     #ITADDR,R0        ;;;Load modul/register address
        COMB    R0    ;;;Invert address bits
        MOV     R0,@#DRVDBC      ;;;Output of address
        BICB    #1,@#DRVCSD+1    ;;;Port D for input
        MOV     @#DRVDBD,R0      ;;;Read and clear interrupt
        MOV     (SP)+,R0          ;;;Restore R0
        CALL    @#$FORK2         ;;;Create system process
        CLR     (R3)              ;Declare FORK block free
        MOV     #EFN,R0           ;Eventflag EFN
        MOV     TSKTCB,R5         ;Task control block address
        CALL    @#$SETF          ;Set eventflag EFN, unstop task
        RETURN



; AST routine, Task level
; Doing nothing but symbol DRVAST has to be defined in CINT$
DRVAST:: MOV    R0,(SP)          ;Save R0, overwrite vector address
       ASTX$S                    ;Exit AST

       .END START



; TKB indirect command file
;
TEST/PR:0,TEST/-SP/-SH= TEST
SY11:RSX11S.STB/SS
LB11:11SLIB/LB
/
;Global common for IO page
RESCOM=EX11:KCOM.TSK/RW:7
//
```

8.2.1.4 Block- und Timing-Diagramme

Abb. 3.2.6.2 Symbole in den Blockdiagrammen (Abb. 3.2.6.3 - 3.2.6.6)

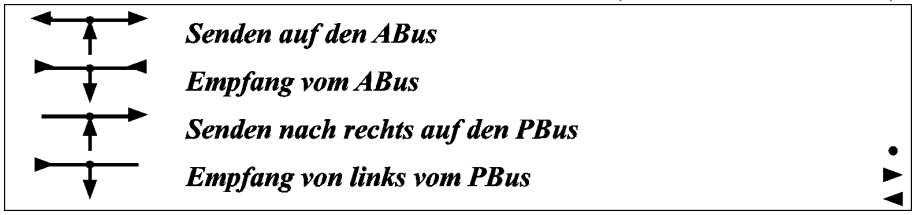


Abb. 3.2.6.3 Routing-Steuerung, Statuseingabe und Adressausgabe

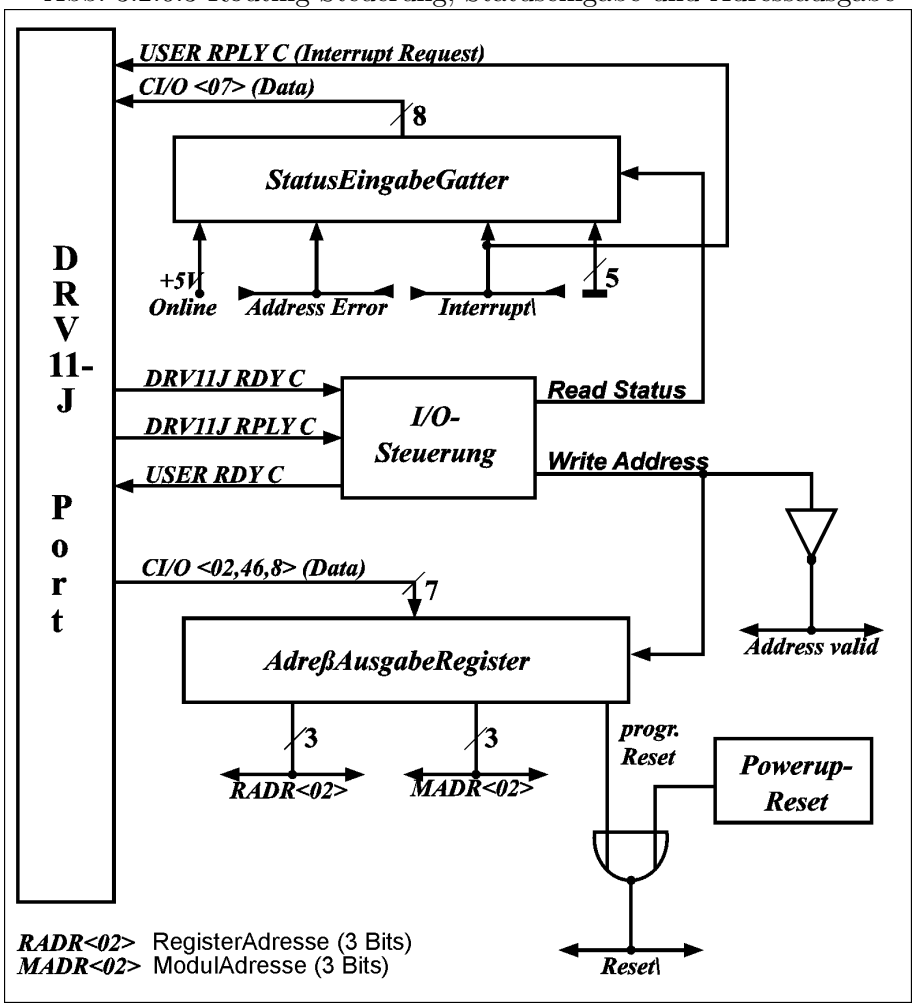


Abb. 3.2.6.4 Routing-Steuerung, Daten-Ein/Ausgabe

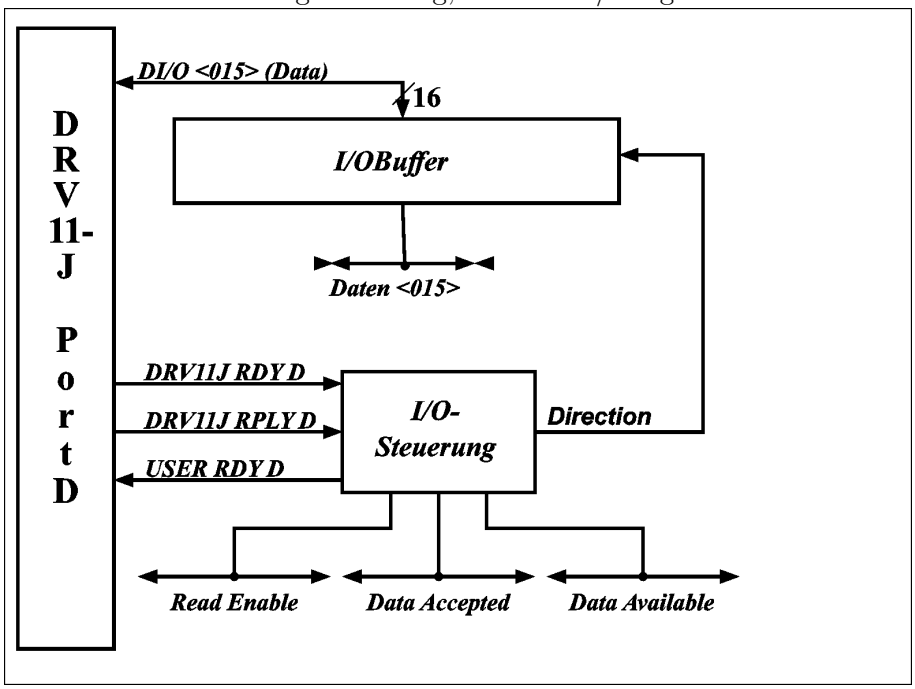


Abb. 3.2.6.5 Adress-Decodierung

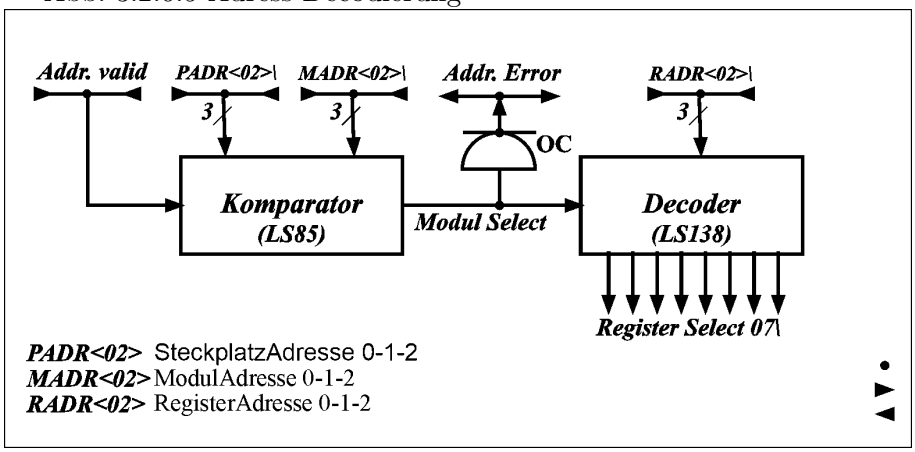


Abb. 3.2.6.6 Daten Ein-/Ausgabe und Interrupt Eingabe

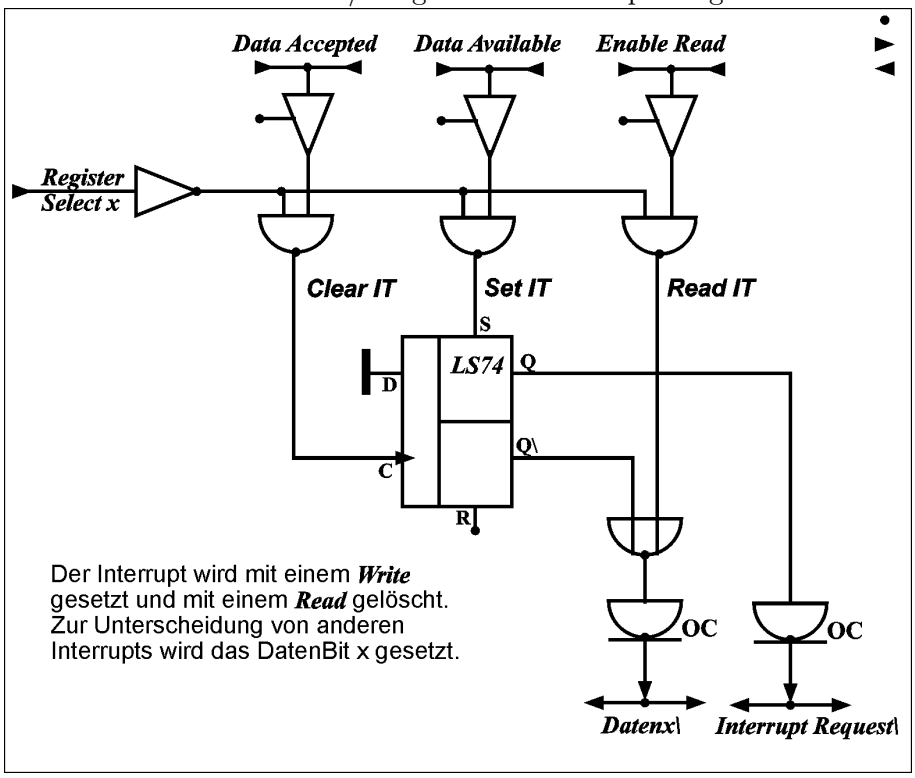
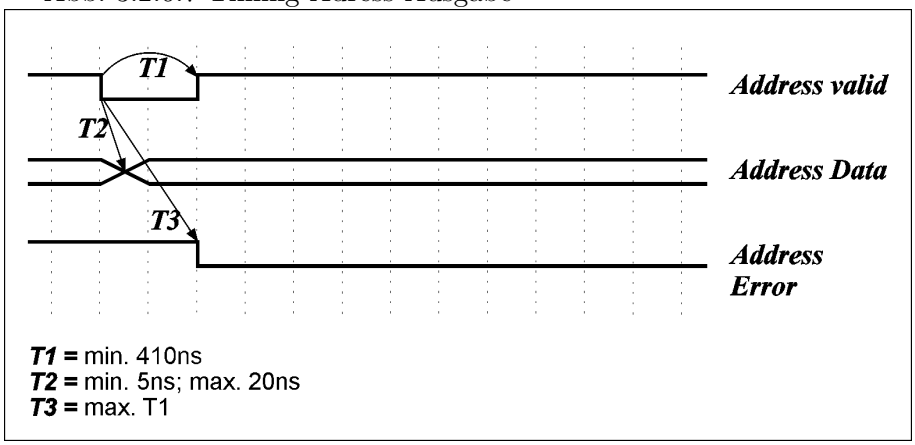
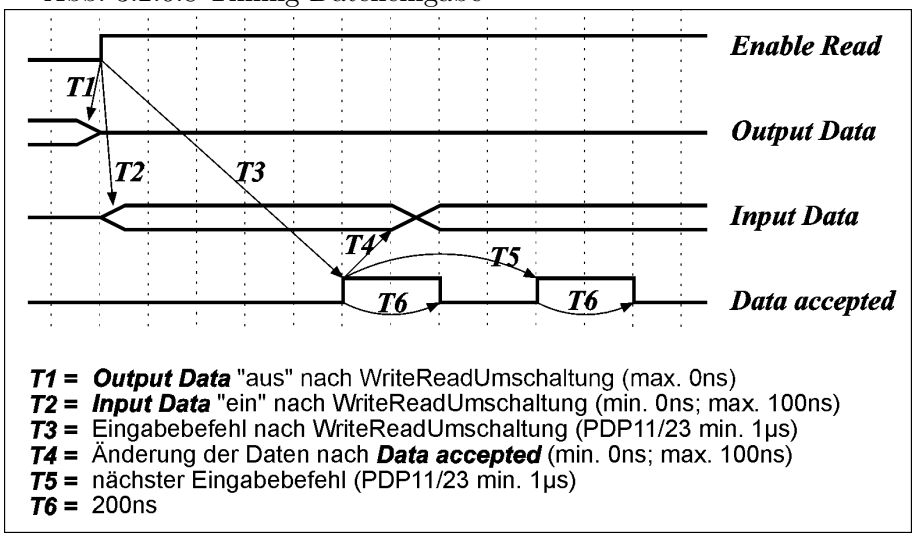


Abb. 3.2.6.7 Timing Adress-Ausgabe



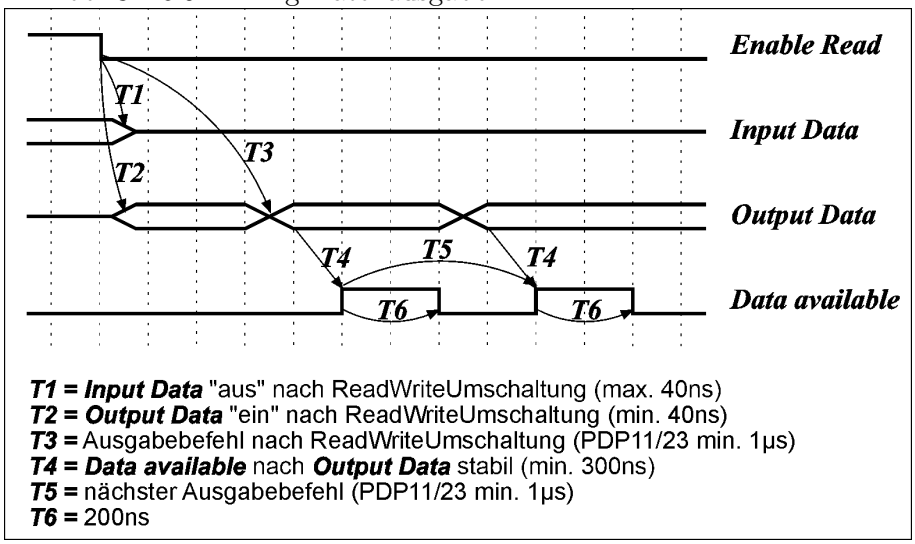
Bei der Ausgabe einer Modul-/Register-Adresse über Port C wird kurz vor der Änderung der Adressdaten das Signal *Address_Valid* auf low geschaltet. Es bleibt auf low bis zur Beendigung der Ausgabe (dies ist erheblich länger als notwendig, aber das macht nix!). Mit diesem Signal kann die Adressentschlüsselung des Interfaces abgeschaltet werden bis die Adressdaten stabil sind, um falsche Register-Selects zu vermeiden. Als Quittung für erkannte Adresse kann das Interface das Signal *Address_Error* auf low ziehen. Über eine Status-Eingabe über Port C kann die Software dieses Signal abfragen und damit feststellen, ob die angesprochene Adresse überhaupt vorhanden ist.

Abb. 3.2.6.8 Timing Dateneingabe



Nachdem der Port D des DRV11-J auf Eingabe programmiert wurde, nimmt die Routing-Steuerung die Ausgabedaten vom Bus und setzt das Signal *Enable_Read* auf high. Mit diesem Signal schaltet das Interface die Eingabe-Datengatter auf und gibt dadurch die Eingabedaten auf den Bus. Die tatsächliche Eingabe der Daten erfolgt erst mit einem nachfolgenden Eingabebefehl über Port D durch die Software, der irgendwann später erfolgen kann. Jede Eingabe wird mit dem Signal *Data_Accepted* quittiert. Danach können die Daten verändert und mit einer erneuten Eingabe eingelesen werden.

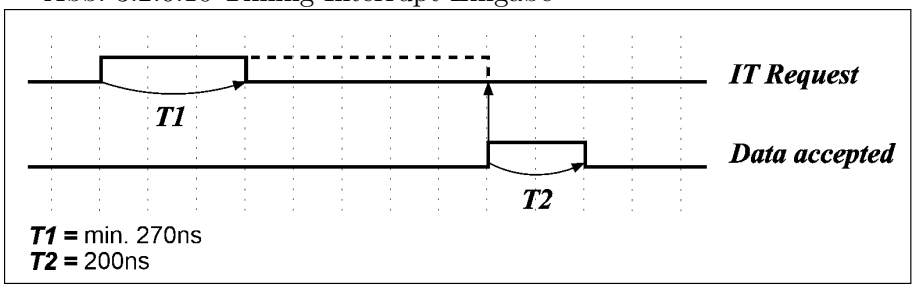
Abb. 3.2.6.9 Timing Datenausgabe



Nachdem der Port D des DRV11-J auf Ausgabe programmiert wurde, setzt die Routing-Steuerung das Signal *Enable_Read* auf low und schaltet die Ausgabedaten auf den Bus. Mit *Enable_Read* muss das Interface die Eingabedaten in angemessener Frist vom Bus nehmen. Die tatsächliche Ausgabe der Daten erfolgt erst mit einem nachfolgenden Ausgabebefehl über Port D durch die Software, der irgendwann später erfolgen kann. Jede Ausgabe wird

mit dem Signal *Data_Available* angezeigt, für dessen Dauer die Ausgabedaten auf den I/O-Leitungen bereit stehen.

Abb. 3.2.6.10 Timing Interrupt-Eingabe



Mit der Vorderflanke des Signals *Interrupt_Request* wird ein Interrupt gespeichert (sofern das DRV11-J entsprechend programmiert wurde) und in der PDP11 die Interrupt-Service-Routine aktiviert ist (falls sie existiert). Solange der *Interrupt_Request* ansteht, kann er über die Statuseingabe mittels Port C abgefragt werden. Damit ergibt sich folgende Möglichkeit der Interrupt-Behandlung: Jedes Interface, das einen Interrupt behandeln lassen will, speichert diesen in einem eigenen Interrupt-Register, und gibt dessen Inhalt auf die *Interrupt_Request* Leitung ("wired or"). Dadurch wird die Interrupt-Service-Routine aktiviert, die nun beginnt alle möglichen Interrupt-Register der Reihe nach auszulesen, um die Quelle des Interruptes zu ermitteln und zu löschen. Zwischendurch kann sie über eine Statuseingabe ermitteln, ob noch weitere Interrupts anstehen. Am Ende muss sie dies auf jeden Fall tun, da ein neuer Interrupt-Request auf einen bereits bestehenden zu keiner weiteren Aktivierung der Interrupt-Routine führt. Auf die nicht ganz triviale Programmierung einer Interrupt-Service-Routine kann auch verzichtet werden, wenn periodisch mittels Statuseingabe auf anstehende Interrupt-Requests abgefragt wird (Polling).