

**HiCO332**

**PC/104**

**Prozess-EA-Baugruppe**

**HiCO332-DOC**

**Handbuch**

**Copyright**

**emtrion**

© Copyright 2007 emtrion GmbH

Alle Rechte vorbehalten. Ohne schriftliche Genehmigung darf diese Dokumentation weder fotokopiert noch auf elektronischen Medien gespeichert werden. Die in dieser Dokumentation enthaltenen Informationen können ohne vorherige Ankündigung geändert werden. Für fehlerhafte Informationen und deren Folgen übernehmen wir keine Haftung. Die verwendeten Warenzeichen anderer Firmen bezeichnen ausschließlich die Produkte dieser Firmen.

<b>Handbuch Revisionsnr.</b>	<b>Änderungen</b>	<b>Datum</b>
<b>1</b>	Erste Ausgabe	29.01.99/ Rr
<b>2</b>	Änderung in emtrion	27.02.07/ma

Das Handbuch wird herausgegeben von:

emtrion GmbH  
Greschbachstr. 12  
D-76229 Karlsruhe

Tel: (0721) 62725-0  
Fax: (0721) 62725-19

Februar 2007 - 002

# Inhaltsverzeichnis

<b>1.</b>	<b>Übersicht</b>	<b>6</b>
1.1.	Einleitung.....	6
1.2.	Blockdiagramm .....	7
1.3.	Hauptmerkmale des HiCO332 .....	8
<b>2.</b>	<b>Inbetriebnahme der Baugruppe</b>	<b>9</b>
<b>3.</b>	<b>Systembeschreibung</b>	<b>10</b>
3.1.	Rechnerkern und Speicher.....	10
3.1.1.	Takterzeugung .....	10
3.1.2.	Chipselect-Signale.....	10
3.1.3.	Unterbrechungen.....	11
3.1.4.	Portanschlüsse des MC68332 .....	12
3.2.	Speicher.....	14
3.2.1.	Flash.....	14
3.2.2.	SRAM .....	14
3.2.3.	Dual Ported Memory.....	15
3.3.	Binäre Schnittstellen.....	15
3.3.1.	24V-Ein-/Ausgänge .....	15
3.3.2.	TTL-Ein-/Ausgänge .....	16
3.4.	Asynchron serielle Schnittstellen.....	17
3.4.1.	RS-232-Schnittstellen.....	17
3.4.2.	RS-485-Schnittstelle.....	17
3.4.3.	Ports des DUARTs .....	18
3.5.	Analogschnittstellen .....	19
3.5.1.	Analogeingänge .....	20
3.5.2.	Analogausgänge .....	21
3.5.3.	Verhalten bei Reset, Freischalten der Ausgänge .....	22
3.6.	TPU-Interface.....	22
<b>4.</b>	<b>Versorgungsspannung</b>	<b>25</b>
4.1.	Versorgung des Analogteiles .....	25
4.2.	Versorgung des Digitalteiles .....	25
4.3.	Batterie .....	25
<b>5.</b>	<b>Interface zum PC/104-Bus</b>	<b>26</b>
5.1.	Reset per SW durch PC .....	26

5.2.	Unterbrechungen.....	26
<b>6.</b>	<b>Firmware</b>	<b>28</b>
6.1.	Lieferumfang.....	28
6.2.	Installation.....	28
6.3.	Hardwarevoraussetzungen.....	29
6.4.	Bootstrap Loader.....	29
6.4.1.	Initialisierung und Firmwarestart.....	29
6.4.2.	Applikationsschnittstelle.....	30
6.5.	Applikationsschnittstelle.....	32
6.5.1.	Konfiguration und Firmwarestart.....	32
6.5.2.	Encoder.....	32
6.5.3.	A/D Wandler.....	35
6.5.4.	Binäre Ein- und Ausgänge.....	37
6.5.5.	Serielle Schnittstellen.....	38
6.5.6.	DPM Kommunikation.....	44
6.5.7.	Datenstrukturen im DPM.....	49
6.6.	Systemkonfiguration.....	49
6.6.1.	Exceptions und Interrupts.....	50
6.6.2.	Entprellung der binären Eingänge.....	51
6.6.3.	Software-Handshake.....	51
<b>7.</b>	<b>Steckerbelegungen</b>	<b>53</b>
7.1.	Lage der Stecker.....	53
7.2.	PC/104, X2/X3.....	54
7.3.	Spannungsversorgung, X8.....	55
7.4.	Serielle Schnittstellen.....	55
7.4.1.	RS-232-Schnittstellen, X4.....	55
7.4.2.	RS-485-Schnittstelle, X5.....	56
7.5.	Binäre Ein-/Ausgabe.....	56
7.5.1.	TTL-Ein-/Ausgabe, X14.....	56
7.5.2.	24V-Ein-/Ausgabe, X1.....	58
7.6.	Encoderinterface, TPU, X6.....	59
7.7.	Analoginterface, X13.....	60
7.8.	BDM-Interface, X19.....	60
<b>8.</b>	<b>Technische Daten</b>	<b>61</b>
8.1.	Mechanische Daten.....	61
8.2.	Elektrische Daten.....	61
8.2.1.	Versorgungsspannung.....	61

---

8.3.	Umgebungsbedingungen .....	61
<b>9.</b>	<b>Bestückungsplan</b>	<b>62</b>
9.1.	Bestückungsseite .....	62
9.2.	Lötseite .....	63
9.3.	Position der Widerstände für shared Interrupt.....	64
<b>10.</b>	<b>Zusätzliche Informationen zu Bauteilen</b>	<b>67</b>

# 1. Übersicht

## 1.1. Einleitung

HiCO332 ist eine intelligente und universelle Baugruppe für jegliche Art von Motion-Control-Applikationen. Besonders durch die vielen unterschiedlichen Schnittstellen und den leistungsfähigen Prozessor kann HiCO332 überall dort eingesetzt werden, wo auf engstem Raum hohe Leistungen gefordert sind. In Bild 1 ist grob die Architektur der Baugruppe dargestellt.

## 1.2. Blockdiagramm

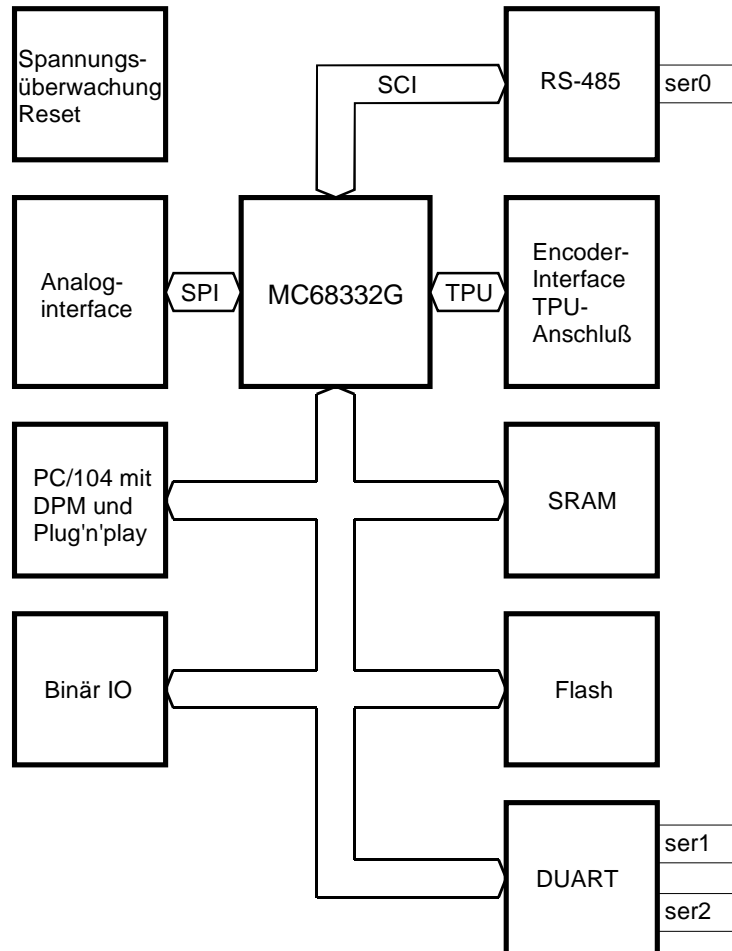


Bild 1: Architektur der Baugruppe HiCO332

### 1.3. Hauptmerkmale des HiCO332

Merkmal	Funktion
Prozessor	MC68332G
Taktfrequenz	20 MHz (PLL)
SRAM gepuffert	128KByte (16 Bit)
Flash	128KByte (16 Bit)
Koppelspeicher	2KByte (8 Bit)
serielle Schnittstellen	2 x RS232, 1 x RS485
Analoginterface	8 x Analogeingang, 2 x Analogausgang
Binäres Interface	8 x TTL-Eingang, 8 x TTL-Ausgang 8 x 24V-Eingang, 8 x 24V-Ausgang
Encoderinterface	2 x Encoderinterface mit A/#A, B/#B und Z/#Z, RS485-Pegel oder TTL
TPU-Interface	10 TPU-Kanäle, TTL
PC-Interface	PC/104
Interrupts auf ISA-Bus	Einstellbar über PnP-Controller
Basisadresse	Einstellbar über PnP-Controller
Shared Interrupt	Ab Werk IRQ5
Spannungsversorgung	5 VDC, +- 15VDC, 350 mA
Backup-Battery	Externe Batterie, 3V
Betriebstemperatur	-10°C bis +70°C
Lagertemperatur	-40°C bis +85°C
Abmessungen	96 x 90 x 20 mm <sup>3</sup>

## 2. Inbetriebnahme der Baugruppe

Bevor Sie den HiCO332 einbauen, lesen und beachten Sie folgende wichtige Hinweise. Diese Hinweise gelten für alle Baugruppen mit elektrostatisch gefährdeten Bauteilen.

- Vor Berührung der Baugruppe, entladen Sie sich bitte statisch. Sie erreichen das, indem Sie einen geerdeten Gegenstand berühren.
- Alle Werkzeuge, die Sie bei der Installation der Baugruppe verwenden, müssen frei von statischen Aufladungen sein.
- Ziehen Sie den Netzstecker Ihres Gerätes, bevor Sie die Baugruppe ein- oder ausbauen.
- Fassen Sie die Baugruppe nur am Rand der Platine an und berühren Sie möglichst keine Bauteile oder Leiterbahnen.

HiCO332 ist eine Baugruppe im PC/104-Format. Sie kann einzeln oder in einem PC/104-System betrieben werden. Beim Einbau der Baugruppe sind zusätzlich folgende Punkte zu beachten:

- Überprüfen Sie vor dem Einbau der Baugruppe, daß alle Steckbrücken richtig eingestellt sind.
- HiCO332 wird über den 104poligen Steckverbinder X2, X3 mit anderen Komponenten verbunden. Vier Befestigungsbohrungen in den Ecken der Leiterplatte dienen der mechanischen Befestigung.
- Schließen Sie alle notwendigen Peripherie-Komponenten und die zusätzliche Spannungsversorgung X8 an.
- Wird HiCO332 allein betrieben, so kann die Spannungsversorgung ausschließlich über den Spannungsversorgungs-Anschluß X8 zugeführt werden.

## 3. Systembeschreibung

### 3.1. Rechnerkern und Speicher

Als Prozessor wird der Motorola Microcontroller MC68332G-20 verwendet. Dieser Prozessor besteht u.a. aus den drei unabhängig voneinander agierenden Teilprozessoren: CPU32, TPU und QSPI.

Die CPU32 ist der eigentliche Prozessor zur Abarbeitung des Programmes. Die QSPI dient der synchronseriellen bzw. asynchronseriellen Kommunikation. Die TPU (timing processing unit) kann eigenständig Aufgaben wie das Erzeugen von Signalfolgen bzw. das Erfassen oder Analysieren von Signalfolgen übernehmen.

Mit dem G-Typen des Prozessors werden spezielle TPU-Mikroprogramme, die auf motion control Aufgaben ausgerichtet sind, zur Verfügung gestellt. Das sind einfache Zähl- oder Timerprogramme, FQD (fast quadratur decoder), NITC (new input timing capture) und so komplexe Funktionen wie PWM (pulse width modulation) oder Ansteuerfunktionen für Schrittmotoren.

Detaillierte Angaben entnehmen Sie bitte der Dokumentation von Motorola [12] und [13].

#### 3.1.1. Takterzeugung

Die Taktfrequenz wird durch einen externen Uhrenquarz mit 32768Hz erzeugt. Die interne, programmierbare PLL generiert daraus den internen Systemtakt von bis zu 20,78MHz.

Der DUART wird mit einem eigenen Quarz von 3,6872MHz versorgt.

#### 3.1.2. Chipselect-Signale

Die Chipselectsignale des Prozessors sind folgendermaßen verwendet:

- **#CSBOOT**: Chipselect für den Flash-Baustein.  
16Bit Datenbus, Größe 64K x 16Bit, 70ns, 0 Waitstates, interne Generierung von #DSACK.
- **#CS0, #CS1**: Chipselect für das SRAM.  
16Bit Datenbus, Größe 64K x 16Bit, 0 Waitstates, interne Generierung von #DSACK. #CS0 selektiert das SRAM, #CS1 selektiert das Lowerbyte, Upperbyte wird durch ADR0 selektiert.

- **#CS2:** Chipselect für das DPM.  
8Bit Datenbus, Größe 2K x 8Bit, 0 Waitstates, externe Generierung von #DSACK über #BUSY des DPM. Verriegelung von read-modify-write-Zyklen bei Semaphoreoperationen (TAS-Befehl). Über #CS5 muß das interne Autovektoring für Unterbrechungen eingerichtet werden.
- **#CS3:** Chipselect für den DUART.  
8 Bit Datenbus, 16 IO-Adressen, externe Generierung von #DSACK
- **#CS4:** Chipselect für binäres Interface.  
8 Bit Datenbus, vier IO-Adressen.

Allen Chipselectsignalen gemeinsam ist die Adreßstrobe-Charakteristik.

### 3.1.3. Unterbrechungen

Die CPU32 verfügt über 7 Unterbrechungsprioritäten, wobei 1 die niedrigste und 7 die höchste Priorität darstellt. Auf der Baugruppe HiCO332 können die Unterbrechungen 3...6 per Jumper den externen Unterbrechungsquellen DUART (Signal: #IRQ\_UAR) und DPM (Signal: #IRQ\_DPM) zugeordnet werden:

Quelle	Ext/int	Priorität	Funktion
DPM	Ext.	3...6	Unterbrechung durch das DPM bei PC-Kommunikation
DUART	Ext.	3...6	RS-232-Schnittstellen ser1 und ser2
SCI	Int.	1...7	RS-485-Schnittstelle ser0
Timer	Int	1...7	PIT
TPU	Int	1...7	Beispielsweise Encoder

Die externen Unterbrechungen dürfen nicht auf einen gemeinsamen Eingang geführt werden. Entsprechend der Zuordnung sind folgende Jumperstellungen notwendig:

Jumper	Geschlossen	Funktion
X12	1-2	#IRQ_DPM an #IRQ3
	2-3	#IRQ_UAR an #IRQ3
X11	1-2	#IRQ_DPM an #IRQ4
	2-3	#IRQ_UAR an #IRQ4
X10	1-2	#IRQ_DPM an #IRQ5
	2-3	#IRQ_UAR an #IRQ5
X9	1-2	#IRQ_DPM an #IRQ6
	2-3	#IRQ_UAR an #IRQ6

### 3.1.4. Portanschlüsse des MC68332

#### 3.1.4.1. Port-E

Port	Name	Funktion
PORTE0		Intern belegt
PORTE1		-
PORTE2		Intern belegt
PORTE3		Intern belegt
PORTE4	#EN_ENC1	Freigabe Encoderkanal 1
PORTE5		Intern belegt
PORTE6	#EN_ENC2	Freigabe Encoderkanal 2
PORTE7	#PO_FAIL	Überwachung der Spannungsversorgung

### 3.1.4.2. Port-F

Port	Name	Funktion
PORTF0	#B_24_ON	Freigabe 24V-Interface
PORTF1	#A_O_ON	Freigabe Analog-Interface
PORTF2	#DIAG_24	Diagnose 24V-Interface
PORTF3		Intern belegt
PORTF4		Intern belegt
PORTF5		Intern belegt
PORTF6		Intern belegt
PORTF7	#B_O_ON	Freigabe TTL-Interface

### 3.1.4.3. Port-QS

Port	Name	Funktion
PQS0..2		SPI-Interface
PQS3	#PCS_AD1	CS für A/D-Wandler 1
PQS4	#PCS_DA1	CS für D/A-Wandler 1
PQS5	#PCS_DA2	CS für D/A-Wandler 2
PQS6	#CONVERT	Ungenutzt, muß log. 1 sein

### 3.1.4.4. TPU

Port	Name	Funktion
TPU.0	Z_1	Nullimpuls Encoder 1
TPU.1	A_1	Kanal A Encoder 1
TPU.2	B_1	Kanal B Encoder 1
TPU.3	Z_2	Nullimpuls Encoder 2
TPU.4	A_2	Kanal A Encoder 2
TPU.5	B_2	Kanal B Encoder 2
TPU.6 bis 15		Frei verfügbar

## 3.2. Speicher

Dem Anwender stehen insgesamt drei Speicherbereiche zur Verfügung: SRAM, Flash und DPM.

### 3.2.1. Flash

Als Programmspeicher wird ein 64k x 16bit großes Flash (Am29F100B) verwendet. Dieses Flash wird mit 5V versorgt und kann im System gelöscht und programmiert werden. Es ist sektoriert, d. h. sektorweises Löschen und Beschreiben ist möglich.

Weitere Angaben, besonders zum Aufbau der Sektoren und zum Programmieralgorithmus sind dem Datenblatt [3] zu entnehmen.

### 3.2.2. SRAM

Als Datenspeicher wird ein 64k x 16bit großes SRAM (KM6161000BLT) verwendet. Der Speicherinhalt kann durch eine sich nicht auf der HiCO332 befindliche Batterie (3V) gepuffert werden.

### 3.2.3. Dual Ported Memory

Das 2k x 8bit große Dual Ported Memory (CY7C132) ist nur in der PC/104-Variante des HiCO332 verfügbar.

Dieser Baustein stellt für beide Seiten (MC68332 und PC über PC/104-Bus) unabhängig Kontrollsignale zur Verfügung. Zugriffskonflikte auf gleiche Zellen werden durch Busy-Signale verhindert. Darüberhinaus werden read-modify-write-Zugriffe für Semaphorbehandlungen seitens des MC68332 unterstützt. In beide Richtungen können Unterbrechungen ausgelöst werden. Weiterführende Informationen sind dem Datenblatt [4] zu entnehmen.

---

**Hinweis:**

Das Busy-Signal wird zur Generierung des DSACK-Signals für den MC68332 verwendet. Bei aktivem Busmonitor des MC68332G muß die Dauer berücksichtigt werden, die der PC das DPM blockieren kann. Bei Problemen muß programmiertechnisch dafür gesorgt werden, daß der PC zwischen jedem DPM-Zugriff einen Zugriff außerhalb des DPM-Adreßraumes aber innerhalb des ISA-Adreßraums macht. Nur so wird das evtl. anliegende Busy-Signal wieder zurückgenommen.

---

## 3.3. Binäre Schnittstellen

### 3.3.1. 24V-Ein-/Ausgänge

Die jeweils acht Ein-/Ausgänge sind galvanisch getrennt über Optokoppler geführt und werden extern versorgt. Auf der Baugruppe HiCO332 befinden sich High-Side-Switches L6376 von SGS-Thomson, die einen maximalen Strom von 100mA liefern können.

Die Eingänge sind mit einem Pull-Down-Widerstand von 4.7KOhm abgeschlossen. Das bedeutet, daß ein unbeschalteter Eingang (24V-Seitig) eine logische 0 liefert.

Die Ausgänge sind als rückles- und rücksetzbare Register ausgeführt. Nach einem HW-Reset sind (Power-on oder dediziertes Resetsignal durch den PC) die Ausgänge deaktiviert und über Pull-Down-Widerstände von 100KOhm auf logisch 0 geschaltet. Die High-Side-Switches sind geöffnet.

Die Freigabe der Ausgaberegister erfolgt mit einer positiven Flanke des Signals #B\_24\_ON. Der Ruhewert bzw. der Wert nach einem HW-Reset dieses Signals ist logisch 1.

Das Signal wird durch einen Portpin des HiCO332 realisiert.

Port	Name	Funktion	HW-Reset
PORTF0	#B_24_ON	0→1: 24V-Ausgänge freischalten	1

Tritt auf der Ausgangsseite der High-Side-Switches ein Fehler wie z.B. Kurzschluß, Übertemperatur, Über- oder Unterspannung auf, wird ein Diagnosesignal an PORTF2 aktiviert. Die Firmware ist für das Behandeln dieses Fehlers zuständig.

Port	Name	Funktion
PORTF2	#DIAG_24	0: Fehler bei den High-side-switches 1: Kein Fehler

### 3.3.2. TTL-Ein-/Ausgänge

Die 8 Eingänge sind mit einem Pull-Down-Widerstand von 100kOhm abgeschlossen. Ein unbeschalteter Eingang liefert beim Lesen immer eine logische Null.

Die 8 Ausgänge sind als rückles- und rücksetzbare Register ausgeführt. Nach einem HW-Reset sind (Power-on oder dediziertes Resetsignal durch PC) die Ausgänge deaktiviert und über Pull-Down-Widerstände von 100KOhm auf logisch 0 geschaltet.

Die Freigabe der Ausgaberegister erfolgt mit einer positiven Flanke des Signals #B\_O\_ON. Der Ruhewert, bzw. der Wert nach einem HW-Reset dieses Signals ist logisch 1.

Das Signal wird durch einen Portpins des HiCO332 realisiert.

Port	Name	Funktion	HW-Reset
PORTF7	#B_O_ON	0→1: TTL-Ausgänge freischalten	1

## 3.4. Asynchron serielle Schnittstellen

### 3.4.1. RS-232-Schnittstellen

Die beiden RS-232-Schnittstellen ser1 und ser2 werden durch den intelligenten DUART SCC68692 von Philips realisiert. Die Handshake-Signale RTS und CTS werden durch Portpins des DUARTs realisiert. Der DUART ist in der Lage die Handshake-Signale selbständig zu bedienen. Die Firmware muß diese Funktionalität nur initialisieren.

Weitere Hinweise zur Programmierung sind dem Datenblatt [5] zu entnehmen.

Die Pegelwandlung erfolgt durch den Baustein ADM211. Die Schnittstellen sind nicht galvanisch getrennt.

### 3.4.2. RS-485-Schnittstelle

Die Schnittstelle ser0 ist über die SCI des MC68332 als Vierdraht-Schnittstelle realisiert. Mit dem Signal #485\_OFF, output-Port des DUARTs, kann zwischen Senden und Empfangen umgeschaltet werden. Gleichzeitiges Senden und Empfangen ist nicht möglich.

Verbindet man den Sender mit dem Empfänger, wird aus der Vierdraht-Schnittstelle eine Zweidraht-Schnittstelle.

Port	Name	Funktion
OP7	#485_off	0: Transmitter deaktiviert, Receiver aktiviert 1: Transmitter aktiviert, Receiver deaktiviert

Die Pegelwandlung erfolgt durch Standard RS-485-Transceiver-Bausteine. Die Schnittstelle ist nicht galvanisch getrennt.

Optional ist der Abschluß des Empfangskanals durch ein Widerstandsnetzwerk (470Ohm nach GND, 150Ohm zwischen den Signalleitungen, 470Ohm nach Vcc) möglich. Hierzu müssen auf der Platine die Jumper X15 und X16 gesteckt werden.

### 3.4.3. Ports des DUARTs

Die Portanschlüsse des DUARTs werden im wesentlichen für die handshake-Signale der beiden RS-232-Schnittstellen verwendet und brauchen nicht durch die Firmware bedient zu werden.

Die Portanschlüsse OP5...7 dienen zur Auswahl des Analogmultiplexers und zur Transferrichtungssteuerung der RS-485-Schnittstelle.

Zum Setzen bzw. Zurücksetzen der Ausgänge stehen zwei Register des DUARTs zur Verfügung (siehe [5]). Eine Besonderheit ist beim SCC68692 zu beachten. Das Setzen eines Ausgabeports bedeutet physikalisch, daß der Portanschluß "low" wird. Entsprechend führt das Löschen zur Ausgabe von "high".

Eingabeports:

Port	Name	Funktion
IP0	#CTS1	#CTS für ser1
IP1	#CTS2	#CTS für ser2
IP2...5		Frei

Ausgabeports:

Port	Name	Funktion
OP0	#RTS1	#RTS für ser1
OP1	#RTS2	#RTS für ser2
OP2...4		Ungenutzt
OP5	ANA_SEL0	Kanalauswahl für Analogeingänge
OP6	ANA_SEL1	Kanalauswahl für Analogeingänge
OP7	#485_OFF	Sendefreigabe RS-485-Interface ser0

### 3.5. Analogschnittstellen

Auf der Baugruppe HICO332 befinden sich insgesamt acht analoge Eingänge und zwei analoge Ausgänge. Die Analog/Digital-Wandlung erfolgt durch den seriellen AD-Wandler AD7890-10 von Analog Devices [6] und die Digital/Analog-Wandlung erfolgt durch den Baustein MAX531 von Maxim [7].

Um einen kontinuierlichen Datenfluß zu erreichen und zur Entlastung der CPU32, wird die QSPI des 68332 für die Kommunikation mit dem A/D- und den D/A-Wandler verwendet. Die Anschaltung erfolgt nicht als daisy chain, sondern quasi parallel am synchronseriellen Interface. Zur Selektion stehen seitens der QSPI Chipselectsignale zur Verfügung.

Nach geeigneter Programmierung ist die QSPI in der Lage, zyklisch an alle Teilnehmer des synchronseriellen Busses Telegramme zu schicken und deren Antworttelegramme zu empfangen. Die Applikation muß somit lediglich den Transferregisterbereich der QSPI beschreiben oder den Receiverregisterbereich auslesen. Die QSPI sorgt für die Konsistenz der Daten.

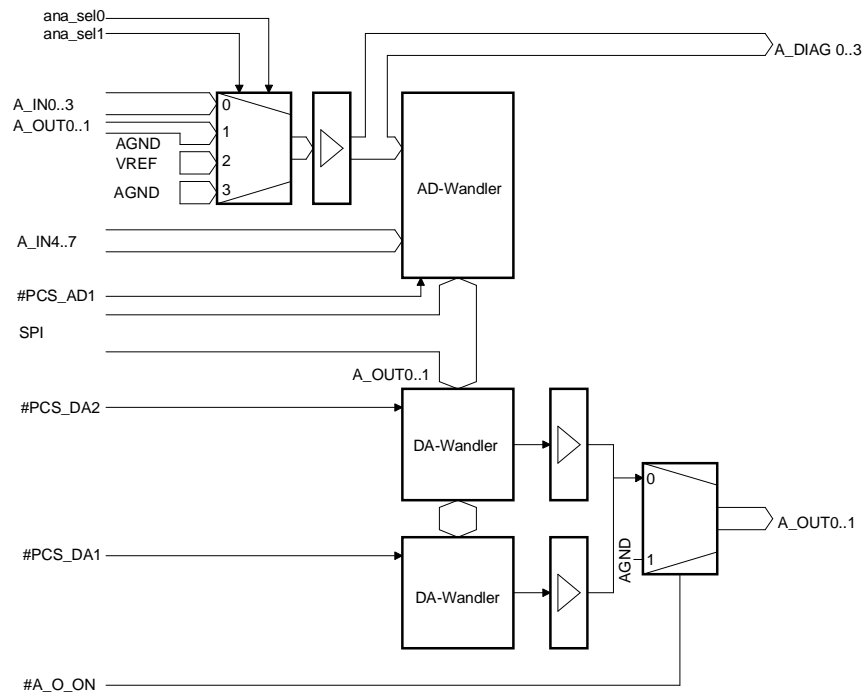


Bild 2: Blockschaltbild des Analoginterfaces

### 3.5.1. Analogeingänge

Die acht massebezogenen Eingänge sind für einen Eingangsspannungsbereich von  $\pm 10V$  ausgelegt. Die Wandlung erfolgt mit 12 Bit Auflösung und die Codierung im Zweierkomplement.

Die Eingänge `A_IN0...3` sind über einen Überspannungsschutz und einen aktiven Filter 2.Ordnung auf den Eingang eines 4x(4:1) Analogmultiplexers geführt. Der Eingangsverstärker agiert als Impedanzwandler. Die Dimensionierung des Filters erfolgt über die RC-Glieder; die Grenzfrequenz liegt bei ca. 1KHz.

Abhängig von den Signalen `ANA_SEL0` und `ANA_SEL1` schaltet der Multiplexer das analoge Eingangssignal, die Masse, die intern gepufferte Referenzspannung von 2.5V oder die Ausgänge der D/A-Wandler auf die Eingänge `ANA0...3` des A/D-Wandlers (siehe Bild 2 und nachfolgende Tabelle).

ANA_SEL1	ANA_SEL0	Funktion
0	0	A_IN0...3 auf die Analogeingänge des AD-Wandlers schalten
0	1	Referenzspannung von 2.5V auf die Analogeingänge des AD-Wandlers schalten
1	0	Analogeingänge ANA0 und ANA1 des AD-Wandlers mit den Analogausgängen der DA-Wandler verbinden, die restlichen zwei mit Masse
1	1	Masse auf die Analogeingänge des AD-Wandlers schalten. Dies ist auch die Stellung nach einem HW-Reset.

Der Analogmultiplexer kann somit auch zum Verstärkungs- und Nullpunktgleich herangezogen werden.

Die Signale ANA\_SEL0 und ANA\_SEL1 werden aus den Portsignalen des DUARTs gewonnen.

Die Analogeingänge ANA0...3 des AD-Wandlers sind zur Diagnose ohne weitere Beschaltung herausgeführt. Die Adaption eines Meßgerätes muß hochohmig erfolgen, da sonst Rückwirkungen stattfinden können.

Die Eingänge A\_IN4...7 sind über einen Überspannungsschutz und ohne weitere Beschaltung direkt an die Eingänge des AD-Wandlers geführt.

### 3.5.2. Analogausgänge

Der Ausgang liefert einen Spannungsbereich von  $\pm 10V$  bei einer Auflösung von 12 Bit. Die Codierung erfolgt binär. Auf Grund spezieller Eigenschaften der Hardware muß der auszugebende Wert um eine Bitposition nach links verschoben in das Transferregister geschrieben werden.

Der D/A-Wandler setzt den Binärwert in eine massebezogene Spannung um. Eine nachfolgende Verstärkerstufe sorgt für einen ausreichenden Innenwiderstand.

Als maximaler Strom werden 5mA zur Verfügung gestellt.

Hinter dem jeweiligen D/A-Wandlerausgang befindet sich ein Analogmultiplexer, der bei einem HW-Reset oder durch ein spezielles Signal den Ausgang auf 0V schaltet.

Nach dem HW-Reset bleibt der Ausgang auf 0V und kann nur durch ein spezielles Signal wieder freigeschalten werden.

### 3.5.3. Verhalten bei Reset, Freischalten der Ausgänge

Nach einem HW-Reset werden die Ausgänge des D/A-Wandlers gesperrt und stattdessen werden mittels eines Analogmultiplexers 0V ausgegeben.

Gleichzeitig werden die Analogeingänge durch einen Analogmultiplexer auf Masse geschaltet. An dieser Stelle wird ausgenutzt, daß die Portausgänge des Prozessors nach einem Reset als Eingänge geschaltet sind und durch einen Pull-Up mit Vcc verbunden sind. Die Ausgänge des DUARTs werden durch diesen Reset auf "High" gesetzt.

Der Analogmultiplexer des D/A-Wandlers kann durch das Signal #A\_O\_ON wieder in die Betriebsstellung geschaltet werden.

Port	Name	Funktion	HW-Reset
PORTF1	#A_O_ON	0: D/A-Ausgänge freischalten 1: D/A-Ausgang sperren, Ausgabe 0V	1

## 3.6. TPU-Interface

Die TPU bietet sehr komfortable Möglichkeiten zeitliche Verhältnisse von Signalen zu analysieren, Ereignisse zu zählen oder auch komplexe Impulsfolgen zu generieren. Dazu sind in der TPU verschiedene Microprogramme niedergelegt, die einfach initialisiert und bedient werden können.

Die TPU des Prozessors MC68332G beinhaltet spezielle Programme für motion control Anwendungen u.a. die FQD (fast quadratur decoding).

Um schnell und einfach einen Winkel- oder Inkrementgeber anschließen zu können, gibt es einerseits einen direkten Anschluß an die TPU. Externe Anpaßschaltungen müssen für die entsprechende Pegelanpassung sorgen. Andererseits sind on board bereits RS485-Treiber, ohne galvanische Trennung, als Pegelwandler für insgesamt zwei Geber vorgesehen (siehe Bild 3).

Beide Geberschnittstellen berücksichtigen die Spuren A und B und zusätzlich den Nullimpuls Z.

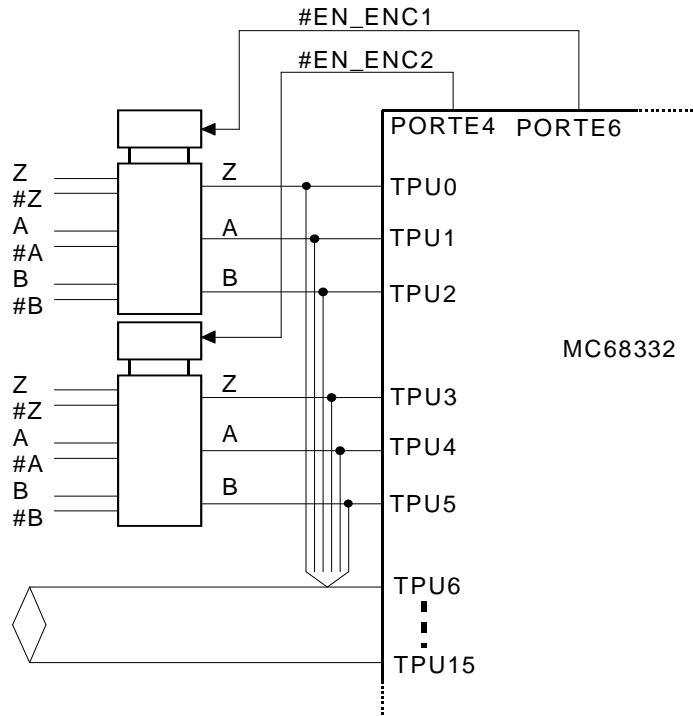


Bild 3: TPU-Anschaltung

Die Auswahl zwischen RS485-Anschaltung oder direkter Anschaltung erfolgt über die Steuersignale #EN\_ENC1 und #EN\_ENC2 wie in nachfolgender Tabelle beschrieben.

Port	Name	Funktion	HW-Reset
PORTE4	#en_enc1	0: RS-485-Treiber für Geber 1 aktivieren 1: RS-485-Treiber für Geber 1 deaktivieren	1
PORTE6	#en_enc2	0: RS-485-Treiber für Geber 2 aktivieren 1: RS-485-Treiber für Geber 2 deaktivieren	1

Werden die Kanäle 0 bis 5 der TPU als Eingänge der FQD programmiert, ergibt sich folgende Belegung:

Port	Name	Funktion
TPU0	Z_1_TPU	Nullimpuls Geberkanal 1
TPU1	A_1_TPU	Spur A Geberkanal 1
TPU2	B_1_TPU	Spur B Geberkanal 1
TPU3	Z_2_TPU	Nullimpuls Geberkanal 2
TPU4	A_2_TPU	Spur A Geberkanal 2
TPU5	B_2_TPU	Spur B Geberkanal 2
TPU6...15	-	Zur freien Verfügung

Die restlichen TPU-Kanäle werden direkt und ohne weitere Beschaltung zur Verfügung gestellt.

## 4. Versorgungsspannung

### 4.1. Versorgung des Analogteiles

Für den analogen Teil des HiCO332 werden +-15V als Spannungsversorgung benötigt. Die Einspeisung muß über den Stecker X8 erfolgen, eine Versorgung über den PC/104-Bus ist nicht möglich. Die Spannungsversorgung wird mit Komparatoren überwacht. Fällt der Betrag einer der beiden Spannungen unzulässig ab, wird das Signal #PO\_FAIL aktiviert.

Die Firmware muß dieses Signal entsprechend auswerten.

Port	Name	Funktion
PORTE7	#PO_FAIL	1: Analogversorgung ok, d. h. >11V bzw. < -11V 0: Analogversorgung fehlerhaft. <11V bzw. > -11V

### 4.2. Versorgung des Digitalteiles

Als Spannungsversorgung sind 5V mit ca. 350mA vorgesehen. Die Versorgungsspannung kann entweder indirekt über den PC/104-Stecker eingespeist werden oder direkt über den Spannungsversorgungsstecker X8.

Unterschreitet die Spannungsversorgung den Schwellwert von ca. 4,5V, wird ein Reset auf HiCO332 ausgelöst. Dieser Reset wirkt sich nicht auf andere Teilnehmer am PC/104-Bus aus.

### 4.3. Batterie

Ebenfalls über den Stecker X8 kann eine Batterie mit 3V zur Pufferung des SRAM-Inhalts angeschlossen werden.

## 5. Interface zum PC/104-Bus

Die Anschaltung der Baugruppe an den ISA-Bus erfolgt durch einen Plug-and-Play Controller NM95MS15 von National Semiconductor/Fairchild [8],[9]. Dieser Baustein verwaltet die Basisadresse und die Größe des IO-Adreßraums, die Vergabe der Unterbrechungsnummern sowie die Basisadresse und die Größe des Memory-Bereichs.

Der Controller ist fertig programmiert. Der zulässige Memory-Bereich liegt zwischen den Segmenten D800 und EF00 und der zulässige IO-Bereich zwischen den Adressen 0x200 und 0x3ff. Die zugelassenen Unterbrechungsnummern sind: 5, 7, 9, 10, 11, 12 und 15. Die eingestellten Daten können entweder direkt aus dem Controller oder aus der ESCD-Datenbank des PCs entnommen werden. Siehe hierzu auch die Literatur [10], [11].

Befindet sich keine weitere Karte am PC/104-Bus, liegt das DPM im Segment D800 mit Offset 0, die IO-Adresse liegt auf 0x200 und als Unterbrechungsnummer wird, sofern nicht verfügt, IRQ5 eingestellt sein. Dies kann sich aber von PC-Hersteller zu PC-Hersteller unterscheiden.

### 5.1. Reset per SW durch PC

Der Reset erfolgt durch einen Schreibzugriff auf die IO-Basisadresse. Bei erfolgreicher Durchführung des Resets erfolgt eine Rückmeldung der Firmware über das DPM.

### 5.2. Unterbrechungen

Implementiert ist der vom PC/104-Consortium vorgeschlagene Interrupt-Sharing-Mechanismus, so daß sich mehrere HiCO332 einen Interrupt des ISA-Busses teilen können. Sharefähig können die Unterbrechungsnummern 5, 10, 11 und 15 durch entsprechende Bestückung zweier Widerstände gemacht werden.

Im Auslieferungszustand ist die Nummer 5 als shared interrupt ausgelegt, die anderen Interrupts sind als Standard-Interrupt ausgelegt.

Interrupt	Bestückter Widerstand	Funktion
5	R9, R6 offen	Non shared
	R6, R9 offen	Shared (Auslieferung)
10	R11, R4 offen	Non shared (Auslieferung)
	R4, R11 offen	Shared
11	R10, R5 offen	Non shared (Auslieferung)
	R5, R10 offen	Shared
15	R8, R7 offen	Non shared (Auslieferung)
	R7, R8 offen	Shared

## 6. Firmware

### 6.1. Lieferumfang

HiCODRV umfaßt die Diskette mit den benötigten Dateien und die Dokumentation. Erstellen Sie von der Diskette eine Sicherheitskopie durch Kopieren aller Dateien.

Auf der Diskette befindet sich befindet sich die Datei `_readme.txt`, die Hinweise enthalten kann, die nach Drucklegung der Dokumentation bekannt wurden. Lesen Sie diese Datei aufmerksam.

### 6.2. Installation

Kopieren Sie alle Dateien aus den Verzeichnissen

- INCLUDE
- LIB

in Ihr Arbeitsverzeichnis.

## 6.3. Hardwarevoraussetzungen

HiCODRV ist lauffähig auf dem Board HiCO332. Die Software unterstützt folgende Hardwareeigenschaften:

- 8 24 V Eingänge
- 8 24 V Ausgänge
- 8 TTL Ausgänge
- 8 TTL Eingänge
- 8 analoge Eingänge
- 2 analoge Ausgänge
- 2 Quadraturencoder
- 3 serielle Schnittstellen
- DPM Kommunikation mit einem Host-PC über den PC/104 Bus
- Laden der Firmware über die DPM Schnittstelle
- Löschen und Schreiben von zwei Flash Sektoren mit je 16 kByte

Zum korrekten Start der Applikation, die mit der Library hicodrv.lib zusammengebunden ist, muß ein Bootstrap Loader ab Version 1.04 vorhanden sein.

## 6.4. Bootstrap Loader

### 6.4.1. Initialisierung und Firmwarestart

Der Bootstrap Loader initialisiert zuerst die Hardware. Hierzu gehört auch das Löschen des DPM am PC/104 Bus. Ist dies erfolgt, erscheint in der DPM-Adresse 0x7e1 die Kennung 0xa5. Danach prüft er seinen eigenen Flash-Bereich mit Hilfe einer Checksumme. Stimmt die berechnete Checksumme nicht mit der eingetragenen überein, wird in die DPM Adresse 0x7e3 0x88 eingetragen, die gelesene Checksumme (16 Bit Wert) in die DPM-Adresse 0x00, die berechnete in die DPM Adresse 0x02 eingetragen und die CPU angehalten.

Nach der Prüfung der eigenen Checksumme wird die Firmware Checksumme geprüft und bei Erfolg die Firmware an der Adresse 0x88020 gestartet. Stimmt die Firmware Checksumme nicht, wird in DPM Adresse 0x7e3 Bit 7 und Bit 5 gesetzt

und in die Funktion zum Laden einer Firmware verzweigt. Die Beschreibung dieser Funktion findet sich im folgenden Abschnitt.

## 6.4.2. Applikationsschnittstelle

### word **BOOT\_VERS** (void)

Returns: 8 Bit Versions- und 8 Bit Revisionsnummer des Loaders

**BOOT\_VERS** liefert Versions- und Revisionsstand des Bootstrap-Loaders. Im higher Byte des Rückgabewertes liegt die 8 Bit breite Versionsnummer, im lower Byte die 8 Bit breite Revisionsnummer.

### int **FW\_LOAD** (int src)

src	Quelle, aus der die Firmware geladen wird
0	ser 0
1	ser 1
2	ser 2
16	DPM

Returns -1 ungültige Quelle angegeben  
-2 Initialisierungsfehler bei Kommunikation

Die Funktion kann aus der Applikation aufgerufen werden und dient zum Laden einer neuen Firmware. Nach dem Löschen des Flash kehrt die Funktion nicht mehr an die aufrufende Stelle zurück, sondern führt nach Abschluß einen Kaltstart durch. Das bedeutet, daß der Bootstrap Loader wie nach einem Hardware-Reset gestartet wird und die Hardwareinitialisierungen und Checksummenprüfungen durchführt. Der Parameter src gibt die Quelle an, von der die Firmware geladen wird. Zur Zeit unterstützt der Bootstrap Loader für src nur den Wert 0x10 für das Laden über DPM.

**int FL\_DELETE (word \*start)**

*start	Startadresse des zu löschenden Sektors
--------	--

Returns           0: Block gelöscht  
                       1: ungültige Startadresse  
                       2: Flash Fehler beim Löschen

Als gültige Startadressen sind die Werte 0x84000 und 0x86000 zulässig. Diese beiden je 8 kB großen Sektoren stehen der Applikation zur Verfügung. Während des Löschens sind alle Interrupts gesperrt.

**int FL\_BL\_WR (word \*destination, word \*source, lword wordcount)**

*destination	Startadresse des Blocks im Flash
*source	Startadresse der Quelldaten
*wordcount	Anzahl der zu brennenden Words

Returns           0: Daten geschrieben  
                       1: ungerade Quell oder Zieladresse  
                       2: eine Zieladresse liegt außerhalb des Blocks  
                       3: Flash Fehler beim Schreiben

Die beiden Startadressen müssen auf geraden Adressen liegen. Der zu schreibende Flash-Bereich muß vorher gelöscht werden, damit Daten korrekt geschrieben werden können. Wird dies unterlassen, liefert FL\_BL\_WR entweder den Wert 0 oder den Wert 3 zurück, je nach vorigem Inhalt der zu schreibenden Zellen. Es können nur 16 Bit breite Werte geschrieben werden. Während der Programmierung des Flash sind alle Interrupts gesperrt.

## 6.5. Applikationsschnittstelle{TC \ 1 "6. Applikationsschnittstelle"}

### 6.5.1. Konfiguration und Firmwarestart{TC \ 2 "6.1 Konfiguration und Firmwarestart"}

Nach dem Durchlaufen des Bootstrap-Bereichs wird das Label start im Modul start.s angesprochen. In diesem Modul können applikationsspezifische Initialisierungen in Assembler eingefügt werden. Start.s verzweigt zu sysinit.c. Dort können die Konfigurationsdaten für den Timer und die Interruptprioritäten eingestellt werden. Außerdem können in sysinit.c applikationsspezifische Initialisierungen in C eingefügt werden.

Zur Initialisierung des Interrupt-Timers dient die Funktion timer\_init.

**word timer\_init (byte prio, word millisec)**

prio	Interruptlevel für den Timerinterrupt
millisec	Anzahl Millisekunden für den Takt {1..15938}

Returns:           eingestellte Zeit in Millisekunden

Da der Timer nicht exakt im Millisekunden Raster eingestellt werden kann, gibt die Funktion die tatsächlich eingestellte Zeit zurück.

In der Auslieferungsversion ist der Aufruf von timer\_init in sysinit.c enthalten. Es werden die Default Werte PIT\_IRQ (6) und 10 ms verwendet.

Zur Konfiguration der Treiberschicht stehen einige Variablen zur Verfügung, die ebenfalls in sysinit.c gesetzt werden. Die Konfiguration ist ausführlich in Kap. 8 beschrieben.

### 6.5.2. Encoder{TC \ 2 "6.2 Encoder"}

Die Encoder sind über RS485 Empfangstreiber oder direkt mit der TPU des 68332 verbunden. In der TPU stehen Funktionen zum Erfassen der Quadratursignale zur Verfügung.

**short TPUnit (short ch)**

ch	Kanalnummer {1, 2}
----	--------------------

Returns           0: Erfolg  
                       -1: unzulässige Kanalnummer

**TPUnit** initialisiert die TPU mit den Funktionen FQD und NITC für den entsprechenden Kanal.

**short TPUMode (short ch, short mode)**

ch	Kanalnummer {1, 2}	
mode	Anschaltung der Encodereingänge an die TPU	
	RS485	Anschaltung über RS-485-Treiber
	TTL	Anschaltung über TTL-Signale

Returns           0: Erfolg  
                       -1: unzulässige Kanalnummer  
                       -2: unzulässiger Anschaltungsmode

**TPUMode** stellt für den ausgewählten Kanal das Encoderinterface ein. Zur Verfügung stehen eine RS-485-Schnittstelle oder ein direkter Zugang über TTL-Signale.

**short TPUReset (short ch)**

ch	Kanalnummer {0, 1}
----	--------------------

Returns           0: Erfolg  
                       -1: Unzulässige Kanalnummer

**TPUReset** setzt alle Zähler des Kanals auf 0 zurück. Dieses Rücksetzen erfolgt durch die TPU.

**short TPURead (short ch, long \*pos, long \*edge\_time, long \*time\_base)**

ch	Kanalnummer {0, 1}
pos	Zeiger auf aktuellen Wert des Positionszählers
edge_time	Zeiger auf Zeitpunkt des letzten Flankenwechsels
time_base	Zeiger auf Zeitbasis der Flankenauswertung

Returns           0: Erfolg  
                       -1: Unzulässige Kanalnummer

**TPURead** liefert die Werte des angeforderten Kanals.

**short TPUReadPos (short ch, long \*pos)**

ch	Kanalnummer {0, 1}
pos	Zeiger auf aktuellen Wert des Positionszählers

Returns           0: Erfolg  
                       -1: Unzulässige Kanalnummer

**TPUReadPos** gibt lediglich den Wert für den Positionszähler zurück.

**short TPUIntInstall (short ch, void \*handler())**

ch	Kanalnummer {0, 1}
----	--------------------

Returns           0: Erfolg  
                       -1: Unzulässige Kanalnummer

**TPUIntInstall** installiert einen Applikationsinterrupthandler. Dieser Interrupthandler wird bei jedem NITC-Interrupt der TPU aufgerufen. Mit ihm kann die Spur Z des Encoder ausgewertet werden.

**short TPUIntEnable (short ch)**

ch	Kanalnummer {0, 1}
----	--------------------

Returns           0: Erfolg  
                      -1: Unzulässige Kanalnummer  
                      -2: kein Interrupthandler installiert

**TPUIntEnable** gibt den Interrupt für den angeforderten Kanal frei. Vorher muß ein Interrupthandler durch TPUIntInstall installiert worden sein.

**short TPUIntDisable (short ch)**

ch	Kanalnummer {0, 1}
----	--------------------

Returns           0: Erfolg  
                      -1: Unzulässige Kanalnummer

**TPUIntDisable** sperrt den Interrupt für den angeforderten Kanal.

**6.5.3. Analoginterface{TC \ 2 "6.3 A/D Wandler"}**

Die Werte der A/D Wandler werden durch die QSPI des 68332 im looping mode eingelesen. Somit stehen zu jeder Zeit gültige Werte zur Verfügung. Diese Werte sind max. 200 µs alt. Die von den Wandlern gelieferten Werte sind 12 Bit breit.

**short analogConfig (short inChan, short outChan)**

inChan	Anzahl der Eingangskanäle
outChan	Anzahl der Ausgangskanäle

Returns:           0: Erfolg  
                      -1: Kanalanzahl unzulässig

**analogConfig** konfiguriert die analogen E/A Kanäle. Die Anzahl der Ausgabekanäle muß mit 0 angegeben werden, wenn keine Analogausgabe auf der HW implementiert ist.

#### short analogIn (short ch)

ch	Kanalnummer {0,...,7}
----	-----------------------

Returns: Wert des Kanals im Zweierkomplement

**analogIn** gibt den Wert eines analogen Eingangskanals als 16 Bit breitem vorzeichenbehafteten Wert zurück. Wird ein nicht initialisierter Kanal angewählt erfolgt Rückgabe eines undefinierten Wertes.

#### short analogInSrc (short source)

source	0	Masse
	1	Analogausgänge
	2	Referenzspannung (2,5V)
	3	Analogverstärker (Analogeingänge)

Returns: 0: Erfolg  
-1: unzulässiger Wert für source

**analogInSrc** wählt die Eingangsquelle für die Kanäle 0 bis 3.

#### short analogOut (short ch, short value)

ch	Kanalnummer {0, 1}
Value	Ausgabewert

Returns: 0: Erfolg  
-1: unzulässiger Wert für ch

***analogOut*** gibt den 16 Bit breitem vorzeichenbehafteten Wert auf dem gewählten Analogkanal aus.

#### **short analogPower ( )**

Returns:           0: Analoge Spannungsversorgung ok  
                   -1: Fehler in der analogen Spannungsversorgung

***analogPower*** prüft, ob die analoge Spannungsversorgung von +-15V vorhanden ist. Ein Fehler wird gemeldet, wenn der Betrag der Spannung kleiner 11V ist.

#### **short analogOutEn ( )**

Returns:           0: ok

***analogOutEn*** schaltet den Ausgangsmultiplexer für die analogen Ausgangsspannung frei.

### **6.5.4. Binäre Ein- und Ausgänge{TC \ 2 "6.4 Binäre Ein- und Ausgänge"}**

#### **void binOutTTLUpdate (byte outVal)**

outVal	neuer Ausgabewert
--------	-------------------

***binOutTTLUpdate*** gibt einen neuen Wert an die TTL Ausgänge aus.

#### **byte binInTTLUpdate (void)**

Returns:           8 Bit Eingangswert

***binInTTLUpdate*** liefert den entprellten Wert der TTL Eingänge zurück. Zur Entprellung siehe Kap. 8.

#### **short binOut24Update (byte outVal)**

outVal	neuer Ausgabewert
--------	-------------------

Returns:           0: Erfolg  
                   -1: Fehlermeldung an den High-Side-Switches

**binOut24Update** gibt einen neuen Wert an die 24VDC Ausgänge aus. Es wird ein Fehlerwert zurückgeliefert, wenn die High-Side-Switches einen Fehler melden.

#### **byte binIn24Update (void)**

Returns:           8 Bit Eingangswert

**binIn24Update** liefert den entprellten Wert der 24 VDC Eingänge zurück. Zur Entprellung siehe Kap. 8.

#### **short binOut24En (void)**

Returns:           0: Erfolg  
                   -1: Fehlermeldung an den High-Side-Switches

**binOut24En** gibt die 24 VDC Ausgänge frei.

#### **void binOutTTLEn (void)**

**binOutTTLEn** gibt die TTL Ausgänge frei.

### **6.5.5. Serielle Schnittstellen{TC \ 26 ".5 Serielle Schnittstellen"}**

Auf HiCO332 sind 3 serielle Schnittstellen vorhanden, die mit ser 0, ser 1 und ser 2 bezeichnet werden. Ser 0 ist mit der SCI des 68332 realisiert und über RS485 Treiberbausteine nach außen geführt. Ser 1 und ser 2 sind über einen zusätzlichen DUART realisiert und über RS232 Treiberbausteine nach außen geführt.

Ser 0 ist für den Betrieb im half duplex mode an einer Zweidrahtleitung ausgelegt. Der Treiber des Senders ist nur während der Sendephasen aktiviert. Ein Empfangen (Mithören) ist dabei nicht möglich. Das Einschalten des Treiberbausteins erfolgt bei Eintrag eines Zeichens in den Sendepuffer (durch die Funktionen putStr oder

putCh). Das Abschalten erfolgt nach der Übertragung eines Zeichens, wenn kein weiteres Zeichen mehr im Sendepuffer enthalten ist. Bei hohen Baudraten kann dies bedeuten, daß nach jedem Zeichen der Treiberbaustein in den inaktiven Zustand geschaltet wird, bis das nächste Zeichen in den Sendepuffer eingetragen wird.

Aufgrund des Betriebs als RS485 Schnittstelle im half duplex mode ist für ser 0 kein Handshake einstellbar.

Alle seriellen Schnittstellen arbeiten grundsätzlich interruptgetrieben. Hierbei kann in der Initialisierungsphase das zu verwendende Handshake-Verfahren eingestellt werden.

Die Puffer zum Senden und Empfangen von Zeichen werden von der Applikationsschicht allokiert und in der Initialisierung übergeben. Nach der Initialisierung stehen diese Puffer für die Applikationsschicht nicht mehr zur Verfügung.

#### **int serlnit (sCOMINIT \*cominit)**

cominit	Zeiger auf Beschreibungsstruktur
---------	----------------------------------

Returns:

- 0: Initialisierung erfolgreich durchgeführt
- 1: ungültige Angabe für com
- 2: ungültiger Wert für bpchar
- 3: ungültiger Wert für stop
- 4: ungültiger Wert für parity
- 5: ungültiger Wert für baud
- 6: ungültiger Wert für hndShk
- 7: Handshake für com 0 nicht zulässig
- 10: ungültiger Wert für baud für diese com
- 11: ungültige Kombination aus bpchar, stop, parity für com0

**serlnit** initialisiert eine serielle Schnittstelle. Die FiFos zum Senden und Empfangen werden eingerichtet, die Hardware der Schnittstelle initialisiert, der Interrupt eingerichtet und der Empfangsinterrupt freigegeben.

Eine Variable der Struktur sCOMINIT enthält die notwendigen Angaben zum Durchführen der Initialisierung.

```

typedef struct COMINIT
{
    byte    com;                // Schnittstellenummer {0..2}
    byte    bpchar;            // Bits per character
    byte    stop;              // Stop Bits
    byte    parity;            // Parity
    lword   baud;              // Baudrate
    byte    hndShk;            // Handshake: none, HW, SW
    byte    *recBufStart;      // Start Empfangspuffer
    lword   recBufSize;        // Länge (Bytes) Empfangspuffer
    byte    *traBufStart;      // Start Sendepuffer
    lword   traBufSize;        // Länge (Bytes) Sendepuffer
} sCOMINIT;

```

Die folgende Tabelle zeigt die Konstanten, die für die Strukturelemente definiert sind und die zulässigen Wert für die Schnittstellen.

Element	Konstanten	ser 0	ser 1 und ser 2
bpchar	BPC_5 BPC_6 BPC_7 BPC_8	Nein Nein mit 2 Stopbits mit 1 Stopbit	ja ja ja ja
stop	S_BITS1 S_BITS2	Ja Ja	ja ja
parity	P_NONE P_EVEN P_ODD	Ja Ja Ja	ja ja ja
baud	B4_8 B9_6 B19_2 B38_4 B115_2	Ja Ja Ja Ja Ja	ja ja ja ja ja
hndShk	HS_NONE HS_HW HS_SW	Ja Nein Nein	ja ja ja

Nach der Initialisierung werden eingehende Zeichen in den Empfangspuffer gelesen. Mit **putCh** und **putStr** können Zeichen in den Sendepuffer eingetragen werden.

#### **int getStr (int com, byte \*string, int maxlen, int \*err)**

com	Nummer der Schnittstelle
string	Zeiger auf die zu empfangenen Zeichen
maxlen	Anzahl der zu lesenden Zeichen
err	<p>Zeiger auf int zur Übergabe von Fehlern (nur bei leerem Empfangspuffer)</p> <p>Bedeutung der Bits:</p> <p>[31] Break received (com 1 und 2) oder noise error (com 0)</p> <p>[30] Framing error</p> <p>[29] Parity error</p> <p>[28] Overrun error</p> <p>[27] FiFo overflow</p>

Returns:            Anzahl der kopierten Zeichen

**getStr** kopiert maximal maxlen Zeichen aus dem Empfangspuffer nach string. Sind weniger als maxlen Zeichen im Empfangspuffer vorhanden, werden diese kopiert. In err werden evt. Fehler übergeben.

Die Fehlerbehandlung ist folgendermassen implementiert: Wird im Interrupthandler ein Hardwarefehler (framing, parity usw.) erkannt, werden ab diesem Zeitpunkt eingehende Zeichen verworfen. Das gleiche gilt bei einem Überlauf des Empfangspuffers. Beim ersten Leseversuch aus einem leeren Puffer wird der Fehler an die Applikationsschicht gegeben, der Fehlermerker gelöscht und die Hardware zurückgesetzt. Danach ist das Empfangen wieder möglich.

#### **int getCh (int com, int \*err)**

com	Nummer der Schnittstelle
-----	--------------------------

err	<p>Zeiger auf int zur Übergabe von bitkodierten Fehlern (nur bei leerem Empfangspuffer)</p> <p>Bedeutung der Bits:</p> <p>[31] Break received (com 1 und 2) oder Noise error (com 0)</p> <p>[30] Framing error</p> <p>[29] Parity error</p> <p>[28] Overrun error</p> <p>[27] FiFo overflow</p> <p>[00] XOFF gesendet oder RTS deaktiviert bei aktivem Handshake</p>
-----	--

Returns:           gelesenes Zeichen als 8 Bit Wert  
                       -1 als 32 Bit Wert: kein Zeichen vorhanden

Zur Fehlerbehandlung siehe unter **getStr**.

#### **int recState (int com, int \*err)**

com	Nummer der Schnittstelle
err	<p>Zeiger auf Fehlermerker</p> <p>Bedeutung der Bits:</p> <p>[31] Break received (com 1 und 2) oder noise error (com 0)</p> <p>[30] Framing error</p> <p>[29] Parity error</p> <p>[28] Overrun error</p> <p>[27] FiFo overflow</p>

Returns:           Anzahl der gültigen Zeichen im Empfangspuffer

**recState** liefert den Zustand des Empfängers zurück. Die Fehler werden auch gemeldet, wenn noch Zeichen im Puffer stehen.

#### **int putStr (int com, byte \*string, int len, int \*err)**

com	Nummer der Schnittstelle
string	Zeiger auf die zu sendenden Zeichen
len	Anzahl der zu sendenden Zeichen
err	Liefert Status der Gegenstelle bei Handshake zurück 0: Gegenstelle empfangsbereit 1: Gegenstelle nicht empfangsbereit

Returns:           Anzahl der angenommenen Zeichen  
                  0 ---> FiFo full

**putStr** kopiert maximal len Zeichen in den Sendepuffer. Wird der Sendepuffer während des Kopierens voll, wird abgebrochen. Der Rückgabewert enthält die Anzahl der kopierten Zeichen. Soll sichergestellt werden, daß alle Zeichen eingetragen werden können, muß vorher **traState** aufgerufen werden. err enthält bei aktivem Handshake die Information ob die Gegenstelle empfangsbereit ist.

#### **int putCh (int com, byte ch, int \*err)**

com	Nummer der Schnittstelle
ch	zu sendendes Zeichen
err	Liefert Status der Gegenstelle bei Handshake zurück 0: Gegenstelle empfangsbereit 1: Gegenstelle nicht empfangsbereit

Returns:           0: Zeichen angenommen  
                  -1: FiFo voll

putCh trägt ein Zeichen in den Sendepuffer ein. Ist der Puffer voll, liefert es den Wert -1 zurück ansonsten den Wert 0. err enthält bei aktivem Handshake die Information ob die Gegenstelle empfangsbereit ist.

#### **int traState (int com, int \*err)**

com	Nummer der Schnittstelle
err	Zeiger auf Fehlermerker 0: Gegenstelle ist empfangsbereit oder kein Handshake eingestellt 1: Gegenstelle ist nicht empfangsbereit (XOFF empfangen oder CTS inaktiv)

Returns: Anzahl der freien Zeichen im Sendepuffer

traState liefert den Zustand des Senders zurück.

#### int serState (int com)

com	Nummer der Schnittstelle
-----	--------------------------

Returns: [0]: Senden in Arbeit  
[1]: Empfangen in Arbeit

serState liefert den Zustand des Datenverkehrs auf einer seriellen Schnittstelle. Hierbei wird die Hardware der Schnittstelle mit einbezogen. Erst wenn das letzte Zeichen eines Sendevorgangs komplett ausgesendet wurde, liefert serState in Bit 0 eine 0 zurück. In Bit 1 wird eine 0 zurückgeliefert, wenn der Empfangspuffer leer ist und zur Zeit kein Zeichen zum Abholen aus der Hardware bereit steht.

### 6.5.6. DPM Kommunikation{TC \ 2 "6.6 DPM Kommunikation"}

Zur DPM Kommunikation stellt die Treiberschicht 2 Datenpuffer zur Verfügung, einen zum Schreiben von Daten und einen zum Lesen. Diese Puffer haben jeweils eine Größe von 992 Byte. Mehr Nutzdaten können in einem Schreib- bzw. Lesevorgang nicht verarbeitet werden.

Die DPM Kommunikation kann in zwei unterschiedlichen Arten erfolgen

- polling Betrieb ohne Interrupts
- interruptgesteuerter Betrieb

Beide Kommunikationspartner sollten auf die gleiche Betriebsart eingestellt sein.

### 6.5.6.1. Initialisierung

Zum Initialisieren der DPM Kommunikation dienen die Funktionen ***dpmInit***, ***dpmEnable*** und ***dpmGetState***. Mit ***dpmInit*** wird die Betriebsart des DPM (interruptgesteuert oder polling) eingestellt, mit ***dpmEnable*** wird der Gegenseite signalisiert, daß die Initialisierung durchgeführt ist und Kommunikation erfolgen kann. Mit ***dpmGetState*** kann der Zustand der Gegenseite ermittelt werden.

**int dpmInit (int mode, (void \*) (rechHandler (byte \*src, int len)), int \*err)**

mode	POLL: kein interruptgesteuerter Betrieb INT: interruptgesteuerter Betrieb
rechHandler	Applikationshandler, der bei Empfang gerufen wird, wenn mode INT angewählt
src	Zeiger auf Beginn der empfangenen Daten
len	Anzahl Bytes, die empfangen werden
err	Zeiger auf Fehlermerker
	NODPMINTHNDL: keine Handleradresse übergeben bei mode INT

Returns: Status der Gegenstelle (siehe dpmGetState)

***dpmInit*** initialisiert die treiberinternen Merker zur DPM Kommunikation und installiert bei interruptgesteuertem Betrieb den Interrupthandler.

**int dpmEnable (void)**

Returns: Status der Gegenstelle (siehe dpmGetState)

***dpmEnable*** setzt für die Treiberschicht der Gegenseite eine Information, daß das DPM initialisiert ist und freigegeben wurde.

**int dpmGetState (int \*err)**

Returns:           RET\_OK: Gegenstelle initialisiert, Kommunikation kann beginnen  
                   NOT\_RDY: Gegenstelle ist nicht bereit

### 6.5.6.2. Polling Betrieb

Zum Schreiben und Lesen von Daten in oder aus dem DPM stehen 2 Funktionen zur Verfügung.

**dpmWrite** schreibt len Bytes in den Datenbereich für den 486. Zur Zugriffssteuerung dient ein Byte im zugeordneten Statusbereich. accMode steuert, ob bei allokiertem Puffer gewartet oder mit einem Fehler zur Applikation zurückgekehrt wird. wrMode steuert, ob bei belegtem Puffer die enthaltenen Daten überschrieben werden oder mit einem Fehler zur Applikation zurückgekehrt wird.

**dpmRead** liest len Bytes aus dem Datenbereich 486->332 an die Adresse dst. accMode legt fest, ob bei allokiertem Puffer gewartet oder mit einem Fehler in die Applikation zurückgekehrt wird.

**int dpmWrite (byte \*src, int len, int accMode, int wrMode)**

src	Zeiger auf die Adresse der zu sendenden Daten	
len	Anzahl Datenbytes	
accMode	Zugriffssteuerung	
	0	wenn DPM von Gegenseite allokiert, gib Fehler an die Applikation zurück
	1	wenn DPM von Gegenseite allokiert, warte bis es frei wird
wrMode	Zugriffssteuerung	
	0	wenn alte Daten von Gegenseite nicht abgeholt werden, gib Fehler an die Applikation zurück
	1	überschreibe alte Daten, auch wenn sie von der Gegenseite nicht abgeholt wurden

Returns: 0:           Daten wurden geschrieben  
           -1:           Datenbereich ist von Gegenseite allokiert und accMode ist 0

- 2: alte Daten wurden v. Gegenseite nicht abgeholt und wrMode ist 0
- 3: len zu groß; Daten passen nicht in den Datenbereich

**dpmWrite** prüft, ob die Daten in den Datenpuffer 332->486 des DPM passen. Dann allokiert es den Puffer. Ist dies nicht möglich und accMode ist 0, kehrt es mit Fehler zurück. Ist accMode 1 wartet es bis die Gegenseite den Puffer freigibt. Dann wird geprüft, ob die Daten von der Gegenseite ausgelesen wurden. Ist dies nicht der Fall und wrMode ist 0, so kehrt **dpmWrite** mit einem Fehler zur Applikation zurück. Ist wrMode 1 oder die Daten wurden von der Gegenseite ausgelesen, werden die Daten in den Puffer kopiert, die Anzahl der kopierten Bytes für die Gegenseite gesetzt, der Puffer für die Leseroutine der Gegenseite gültig gesetzt und der Puffer deallokiert.

**int dpmRead (byte \*dst, int len, int accMode, int rdMode, int format)**

Dst	Zeiger auf die Zieladresse	
Len	maximale Anzahl der zu kopierenden Bytes	
AccMode	Zugriffssteuerung	
	0	wenn DPM von Gegenseite allokiert, gib Fehler an die Applikation zurück
	1	wenn DPM von Gegenseite allokiert, warte bis es frei wird
RdMode	Zugriffssteuerung	
	0	wenn keine neuen Daten vorhanden, gib Fehler an die Applikation zurück
	1	lies Daten unabhängig von Aktualisierung durch Gegenseite
format	0	keine Formatwandlung
	1	word swapping
	2	lword swapping

Returns: Anzahl der gelesenen Bytes, wenn kein Fehler aufgetreten

- 1: Datenbereich ist von Gegenseite allokiert und accMode = 0

**dpmRead** allokiert den Datenbereich. Schlägt dies fehl und accMode ist 0, wird mit einem Fehler in die Applikation zurückgekehrt. Ist accMode 1 wird gewartet, bis die Gegenseite den Datenbereich freigibt. Dann prüft **dpmRead**, ob neue Daten im Datenbereich vorhanden sind. Falls nicht und rdMode ist 0, wird mit einem Fehler in die Applikation zurückgekehrt. Dann werden die Daten aus dem Puffer an die Adresse dst gelesen, wobei entsprechend format gewappt wird. Die Anzahl der gelesenen Bytes ergibt sich aus dem kleineren Wert von len und dem von der Gegenseite eingetragenen Wert für die Länge der gültigen Daten.

### 6.5.6.3. Interrupt Betrieb

**int dpmWrite (byte \*src, int len, int accMode, int wrMode)**

src	Zeiger auf die Adresse der zu sendenden Daten	
len	Anzahl Datenbytes	
accMode	Zugriffssteuerung	
	0	wenn DPM von Gegenseite allokiert, gib Fehler an die Applikation zurück
	1	wenn DPM von Gegenseite allokiert, warte bis es frei wird
wrMode	Zugriffssteuerung	
	0	wenn alte Daten von Gegenseite nicht abgeholt werden, gib Fehler an die Applikation zurück
	1	überschreibe alte Daten, auch wenn sie von der Gegenseite nicht abgeholt wurden

Returns 0: Daten wurden geschrieben  
 -1: Datenbereich ist von Gegenseite allokiert und accMode ist 0  
 -2: alte Daten wurden v. Gegenseite nicht abgeholt und wrMode ist 0  
 -3: len zu groß, Daten passen nicht in den Datenbereich

**dpmWrite** arbeitet im interruptgesteuerten Betrieb identisch zum Polling Betrieb. Nach dem Kopieren der Daten ins DPM wird lediglich zusätzlich ein Interrupt für die Gegenseite ausgelöst.

```
void recHandler (byte *src, int len)
```

src	Zeiger auf das 1. empfangene Zeichen
len	Anzahl der empfangenen Bytes

Diese Funktion hat die Applikationsschicht bei interruptgesteuerter DPM Kommunikation zur Verfügung zu stellen. Die Ansprungsadresse der Funktion wird beim Aufruf von `dpmlnit` übergeben. Der Hardwarehandler der Treiberschicht ermittelt die Anzahl der empfangenen Bytes und ruft die Funktion **recHandler** auf. Diese kann das Kopieren der empfangenen Daten entweder selbst durchführen oder die Funktion `dpmRead` aufrufen, die beim Polling Betrieb beschrieben ist.

### 6.5.7. Datenstrukturen im DPM

Das DPM wird in 5 Bereiche eingeteilt:

- Datenpuffer 332 -> 486 (000-3DF)
- Datenpuffer 486 -> 332 (3E0-7BF)
- Status zu Bereich 1 (7C0-7CF)
- Status zu Bereich 2 (7D0-7DF)
- allgemeine Statusdaten und Interrupt auslösende Zellen (7E0-7FF)

## 6.6. Systemkonfiguration{TC \ 1 "8. Systemkonfiguration"}

Zur Systemkonfiguration stehen die beiden Quelldateien `start.s` in Assembler und `sysinit.c` in C zur Verfügung. In `start.s` können auf Assemblerebene eigene Initialisierungen eingefügt werden. Das Label `start` ist der Einsprung und wird vom Bootstrap-Bereich aus angesprungen, wenn alle Checks erfolgreich waren. Am Ende von `start.s` wird nach `sysinit.c` gesprungen, wo in C hardwarenahe Initialisierungen vorgenommen werden. Auch hier können applikationsspezifische Initialisierungen eingefügt werden. Am Ende von `sysinit.c` wird `VRTXMC` initialisiert und gestartet, wodurch die Applikationstasks entsprechend den Einstellungen in 'user.def' anlaufen.

### 6.6.1. Exceptions und Interrupts{TC \ 2 "8.1 Exceptions und Interrupts"}

Die Interruptvektortabelle wird von HiCoDrv bei 0 angelegt und initialisiert. Alle nicht verwendeten Interruptvektoren werden auf eine Ansprungsadresse initialisiert, an der in die Adresse STAT332 im DPM (Offset 0x7e3) der Wert 0xA0 und in die Adresse ERR332 des DPM (Offset 0x7e6) die Vektornummer der Exception geschrieben werden.

Der einzige Vektor, der durch den Aufruf von `evt_init` auf eine andere Adresse initialisiert wird, ist der Vektor für den TRAP #0. Er wird intern verwendet und steht der Applikation nicht zur Verfügung. Alle Interruptvektoren, die hardwareseitig verwendet werden, werden erst bei der Initialisierung der entsprechenden Hardwarefunktion gesetzt.

Die CPU32 hat 7 Interruptprioritäten, wobei 1 die niedrigste, 7 die höchste Priorität darstellt. Aus dem 68332 herausgeführt sind bei HiCo332 die Interrupteingänge 3, 4, 5, und 6. Sie können den externen Interruptquellen per Jumper zugeordnet werden. Die internen Interruptquellen können einer der Prioritäten 1 bis 7 zugeordnet werden. Die Applikationsschicht übergibt die gewünschten Interruptprioritäten bei der Initialisierungsfunktion. Die folgende Tabelle gibt einen Überblick über alle Interruptquellen und die zuzuordnenen Prioritäten.

{PRIV ATE }Quelle	ext/-int	Prio	Default	VektorNr	Offset (hex)	Verwendung
DPM	ext	3..6	3	AVEC	150	PC-Kommunikation
DU-ART	ext	3..6	4	AVEC	14C	ser1 und ser2
SCI	int	1..7	5	82	148	ser0 (RS485)
Timer	int	1..7	6	80	140	PIT (VRTX)
TPU	int	1..7	2	64 bis 79	100..13C	Encoder

Die zu diesen Quellen gehörenden Interruptvektoren werden beim Initialisieren der Einheiten durch die Treiberschicht eingetragen. Die Vektornummern sind fest vorgegeben. Bei der Initialisierung der Interrupts werden die in `sysinit.c` vorbesetzten Werte für die Prioritäten verwendet. Die Default-Einstellungen sind:

```
prio_tpu      =      TPU_IRQ;      //IRQ 2 für die Encoder
prio_dpm      =      DPM_IRQ;      //IRQ 3 für die DPM-Kommunikation
prio_duart    =      DUART_IRQ;    //IRQ 4 für die beiden DUART
//Schnittstellen
prio_sci      =      SCI_IRQ;      //IRQ 5 für die SCI Schnittstelle
prio_pit      =      PIT_IRQ;      //IRQ 6 für den Peridic Interrupt
//Timer
```

Liegt einer der Werte außerhalb des zulässigen Bereichs, wird der oben angegebene Defaultwert verwendet. Die Verwendung gleicher Interruptprioritäten für verschiedene Quellen wird von der Hard- und Firmware nicht unterstützt und führt zu fehlerhaftem Verhalten der Baugruppe.

Die Interruptprioritäten für

- DPM
- DUART

müssen über Jumper **und** über die Firmware konfiguriert werden, da die Firmware die Jumper nicht einlesen kann. Es muß darauf geachtet werden, daß die Jumperstellung mit der Auswahl in sysinit.c übereinstimmt.

### 6.6.2. Entprellung der binären Eingänge{TC \ 2 "8.2 Entprellung der binären Eingänge"}

Die Entprellung der binären Eingänge findet im Interrupthandler des Timers statt. Ein Eingang muß die in der Variablen binIn\_tau angegebene Anzahl von systicks stabil sein, damit der neue Wert übernommen wird. Die Variable binIn\_tau wird in sysinit.c auf den Default Wert 5 eingestellt.

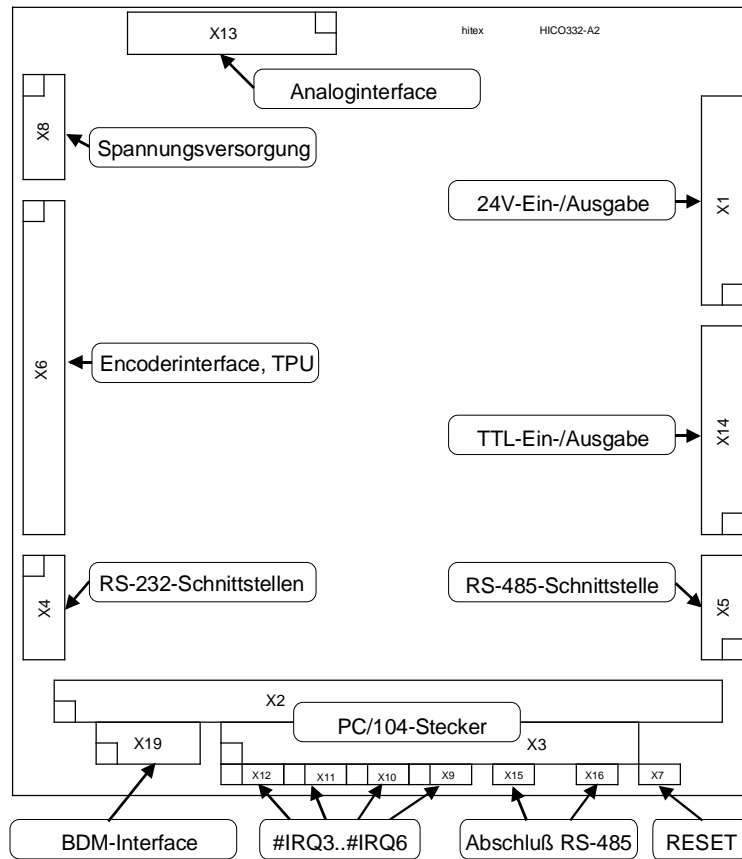
### 6.6.3. Software-Handshake{TC \ 2 "8.3 Software-Handshake"}

Wird bei der Initialisierung der seriellen Schnittstellen Software-Handshake eingestellt, sendet der Empfangs-Interrupthandler ein XOFF an den sendenden Partner, sobald der Füllstand des Empfangspuffers den Wert xofflimit erreicht hat. Xofflimit ist vom Typ int und enthält die Angabe in %. Wird durch Auslesen des Empfangspuffers ein Füllstand von xonlimit unterschritten, wird an die Gegenstelle ein XON gesendet. Die beiden Werte können in sysinit.c eingestellt werden, wenn

die Applikation dies erfordert. Die Default Einstellung ist 80 % für xofflimit und 20 % für xonlimit.

## 7. Steckerbelegungen

### 7.1. Lage der Stecker



## 7.2. PC/104, X2/X3

Typ 64pol + 40pol Buchsenleiste mit langen Pins, 2.54 mm

	Reihe A	Reihe B	Reihe C	Reihe D	
1	-	GND			
2	D7	RESET			
3	D6	+5 V			
4	D5	IRQ9			
5	D4	-			
6	D3	DRQ2			
7	D2	-12 V			
8	D1	-			
9	D0	+12 V	GND	GND	0
10	IOCHRDY	GND	SBHE	MEMCS16	1
11	AEN	SMEMW	A23	IOCS16	2
12	A19	SMEMR	A22	IRQ10	3
13	A18	IOW	A21	IRQ11	4
14	A17	IOR	A20	IRQ12	5
15	A16	-	A19	IRQ15	6
16	A15	-	A18	IRQ14	7
17	A14	DACK1	A17	-	8
18	A13	DRQ1	MEMR	-	9
19	A12	REF	MEMW	DACK5	10
20	A11	SYSCLK	D8	DRQ5	11
21	A10	IRQ7	D9	-	12
22	A9	IRQ6	D10	-	13
23	A8	IRQ5	D11	-	14
24	A7	-	D12	-	15
25	A6	IRQ3	D13	+5 V	16
26	A5	DACK2	D14	-	17
27	A4	TC	D15	GND	18
28	A3	BALE	-	GND	19
29	A2	+5 V			
30	A1	OSC			
31	A0	GND			
32	GND	GND			

### 7.3. Spannungsversorgung, X8

Typ 10pol Pfosten, 2.54 mm

Pin	Signal	Pin	Signal
1	GND	2	+5 V
3	GND	4	+5 V
5	GND	6	Batt
7	+12 V	8	-12 V
9	+12 V	10	-12 V

### 7.4. Serielle Schnittstellen

Typ 10pol Pfosten, 2.54 mm

#### 7.4.1. RS-232-Schnittstellen, X4

Pin	Signal	Pin	Signal
1	RxD2	2	CTS2
3	RxD1	4	RTS1
5	TxD1	6	CTS1
7	TxD2	8	RTS2
9	GND	10	-

### 7.4.2. RS-485-Schnittstelle, X5

Pin	Signal	Pin	Signal
1		2	TxD0
3	#TxD0	4	
5		6	RxD0
7	#RxD0	8	GND
9		10	

## 7.5. Binäre Ein-/Ausgabe

Typ 20pol Pfosten, 2.54 mm

### 7.5.1. TTL-Ein-/Ausgabe, X14

Pin	Signal	Pin	Signal
1	D_OU0	2	D_OU1
3	D_OU2	4	D_OU3
5	D_OU4	6	D_OU5
7	D_OU6	8	D_OU7
9	+ 5V	10	GND
11	D_IN0	12	D_IN1
13	D_IN2	14	D_IN3
15	D_IN4	16	D_IN5
17	D_IN6	18	D_IN7
19	+ 5V	20	GND



**7.5.2. 24V-Ein-/Ausgabe, X1**

Pin	Signal	Pin	Signal
1	+ 24V	2	0V
3	D_OU0	4	D_OU1
5	D_OU2	6	D_OU3
7	D_OU4	8	D_OU5
9	D_OU6	10	D_OU7
11	D_IN0	12	D_IN1
13	D_IN2	14	D_IN3
15	D_IN4	16	D_IN5
17	D_IN6	18	D_IN7
19	+ 24V	20	0V

## 7.6. Encoderinterface, TPU, X6

Typ 32pol Pfosten, 2.54 mm

Pin	Signal	Pin	Signal
1	A_1	2	#A_1
3	B_1	4	#B_1
5	Z_1	6	#Z_1
7	A_2	8	#A_2
9	B_2	10	#B_2
11	Z_2	12	#Z_2
13	GND	14	GND
15	Z1_TPU	16	A1_TPU
17	B1_TPU	18	Z2_TPU
19	A2_TPU	20	B2_TPU
21	TPU6	22	TPU7
23	TPU8	24	TPU9
25	TPU10	26	TPU11
27	TPU12	28	TPU13
29	TPU14	30	TPU15
31	GND	32	GND

## 7.7. Analoginterface, X13

Typ 20pol Pfofen, 2.54 mm

Pin	Signal	Pin	Signal
1	A_IN0	2	A_IN1
3	A_IN2	4	A_IN3
5	AGND	6	AGND
7	A_DIA0	8	A_DIA1
9	A_DIA2	10	A_DIA3
11	A_IN4	12	A_IN5
13	A_IN6	14	A_IN7
15	AGND	16	AGND
17	A_OU0	18	A_OU1
19	AGND	20	AGND

## 7.8. BDM-Interface, X19

Typ 10pol Pfofen, 2.54 mm

Pin	Signal	Pin	Signal
1	#DS	2	#BERR
3	GND	4	#BPKT
5	GND	6	FREEZE
7	#RESET	8	#IFETCH
9	+ 5V	10	#IPIPE

## 8. Technische Daten

### 8.1. Mechanische Daten

<b>Gewicht:</b>	100 g
<b>Leiterplatte:</b>	Glasepoxi FR-4, UL gelistet, 6 Lagen
<b>Abmessungen:</b>	96 mm x 90 mm x 22 mm

### 8.2. Elektrische Daten

#### 8.2.1. Versorgungsspannung

<b>Versorgungsspannung:</b>	5 V, +/-5%, +- 15V
<b>Stromaufnahme:</b>	max 0.35 A

### 8.3. Umgebungsbedingungen

<b>Temperatur</b>	-10 ... +70°C im Betrieb -40 ... +85°C bei Lagerung
<b>rel. Luftfeuchtigkeit</b>	0 ... 95 %, nicht kondensierend











## 10. Zusätzliche Informationen zu Bauteilen

- [1] PC/104 Specification  
Version 2.3, June 1996  
<http://www.controlled.com/pc104/techp1.html>
- [2] MAX531 +5V, Low-Power, Voltage-Output, Serial 12-Bit DAC  
Rev. 6; 2/97  
MAXIM
- [3] Am29F100B, CMOS 5.0V only, sector erase Flash Memory  
AMD
- [4] CY7C132, 2Kx8 Dual-Port Static RAM  
Cypress Semiconductor Corporation
- [5] SCC68692, Dual asynchronous receiver/transmitter (DUART)  
Philips Semiconductor
- [6] AD7890-10, Serial Data Acquisition System  
Analog Devices
- [7] MAX531, Serial 12-Bit DAC  
Maxim
- [8] NM95MS15, Plug and Play Front-End Device for ISA-Bus Systems  
1998 Fairchild Semiconductor Corporation
- [9] NM95MS15 User's Guide  
1998 Fairchild Semiconductor Corporation
- [10] Plug and Play BIOS Specification  
Version 10.A, May 5, 1994  
Compac Computer Corporation, Phoenix Technologies Ltd., Intel Corporation

- [11] Extended System Configuration Data Specification  
Version 1.0A, May 31, 1994  
Compac Computer Corporation, Phoenix Technologies Ltd., Intel  
Corporation
- [12] M68300 Family, MC68332, Users Manual;  
MC68332UM/AD;  
Motorola
- [13] M68300 Family, TPU, Time Processing Unit, Reference Manual;  
TPURM/AD;  
Motorola