

**HiCO332**

**PC/104  
Process I/O Card**

**HiCO332-DOC  
User  
Manual**

**Copyright**

**emtrion**

© Copyright 2007 emtrion GmbH

All rights reserved. Without written permission this documentation may neither be photocopied nor stored on electronic media. The information contained in this documentation is subject to change without prior notification. We do not assume any liability for erroneous information or its consequences. The trademarks of other companies that are used identify the products of these companies exclusively.

| <b>Manual Revision no.</b> | <b>Changes</b>    | <b>Date</b> |
|----------------------------|-------------------|-------------|
| 1                          | First edition     | 06-30-99/Rr |
| 2                          | Change to emtrion | 02-27-07/ma |

This document is published by:

emtrion GmbH  
Greschbachstr. 12  
D-76229 Karlsruhe

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

Februar 2007 - 002

## Table of Contents

|           |  |           |
|-----------|--|-----------|
| <b>1.</b> | <b>Overview</b>                              | <b>6</b>  |
| 1.1.      | Introduction .....                           | 6         |
| 1.2.      | Block Diagram .....                          | 7         |
| 1.3.      | Basic Characteristics of the HiCO332 .....   | 8         |
| <b>2.</b> | <b>Starting Up the Card</b>                  | <b>9</b>  |
| <b>3.</b> | <b>System Description</b>                    | <b>9</b>  |
| 3.1.      | Processor Core and Memory .....              | 9         |
| 3.1.1.    | Pulse Generation .....                       | 10        |
| 3.1.2.    | Chip Select Signals .....                    | 10        |
| 3.1.3.    | Interrupts .....                             | 11        |
| 3.1.4.    | Port Connections of the MC68332 .....        | 12        |
| 3.2.      | Memory .....                                 | 14        |
| 3.2.1.    | Flash .....                                  | 14        |
| 3.2.2.    | SRAM .....                                   | 14        |
| 3.2.3.    | Dual Ported Memory .....                     | 15        |
| 3.3.      | Binary Ports .....                           | 15        |
| 3.3.1.    | 24V Inputs/Outputs .....                     | 15        |
| 3.3.2.    | TTL Inputs/Outputs .....                     | 16        |
| 3.4.      | Asynchronous Serial Ports .....              | 16        |
| 3.4.1.    | RS-232 Ports .....                           | 16        |
| 3.4.2.    | RS-485 Port .....                            | 17        |
| 3.4.3.    | Ports of the DUART .....                     | 17        |
| 3.5.      | Analog Ports .....                           | 18        |
| 3.5.1.    | Analog Inputs .....                          | 19        |
| 3.5.2.    | Analog Outputs .....                         | 20        |
| 3.5.3.    | Behavior On Reset, Enabling of Outputs ..... | 21        |
| 3.6.      | TPU Interface .....                          | 21        |
| <b>4.</b> | <b>Supply Voltage</b>                        | <b>25</b> |
| 4.1.      | Supply for the Analog Section .....          | 25        |
| 4.2.      | Supply for the Digital Section .....         | 25        |
| 4.3.      | Battery .....                                | 25        |
| <b>5.</b> | <b>Interface to the PC/104 Bus</b>           | <b>25</b> |
| 5.1.      | Software Reset by PC .....                   | 26        |

---

|           |  |           |
|-----------|--|-----------|
| 5.2.      | Interrupts.....                              | 26        |
| <b>6.</b> | <b>Firmware</b>                              | <b>28</b> |
| 6.1.      | Package Contents .....                       | 28        |
| 6.2.      | Installation .....                           | 28        |
| 6.3.      | Hardware Requirements .....                  | 29        |
| 6.4.      | Bootstrap Loader.....                        | 29        |
| 6.4.1.    | Initializing and Starting the Firmware ..... | 29        |
| 6.4.2.    | Application Interface.....                   | 30        |
| 6.5.      | Application Interface.....                   | 32        |
| 6.5.1.    | Configuration and Firmware Start .....       | 32        |
| 6.5.2.    | Encoder .....                                | 32        |
| 6.5.3.    | Analog Interface .....                       | 36        |
| 6.5.4.    | Binary Inputs/Outputs.....                   | 39        |
| 6.5.5.    | Serial Ports .....                           | 41        |
| 6.5.6.    | DPM Communication .....                      | 47        |
| 6.6.      | System Configuration .....                   | 52        |
| 6.6.1.    | Exceptions and Interrupts .....              | 52        |
| 6.6.2.    | Debouncing of the Binary Inputs .....        | 53        |
| 6.6.3.    | Software Handshake .....                     | 54        |
| <b>7.</b> | <b>Connector Pinouts</b>                     | <b>55</b> |
| 7.1.      | Locations of Connectors .....                | 55        |
| 7.2.      | PC/104, X2/X3 .....                          | 56        |
| 7.3.      | Voltage Supply, X8.....                      | 57        |
| 7.4.      | Serial Ports .....                           | 57        |
| 7.4.1.    | RS-232 Ports, X4 .....                       | 57        |
| 7.4.2.    | RS-485 Port, X5 .....                        | 58        |
| 7.5.      | Binary Input/Output .....                    | 58        |
| 7.5.1.    | TTL Input/Output, X14.....                   | 58        |
| 7.5.2.    | 24V Input/Output, X1 .....                   | 59        |
| 7.6.      | Encoder Interface, TPU, X6 .....             | 60        |
| 7.7.      | Analog Ports, X13 .....                      | 61        |
| 7.8.      | BDM Interface, X19 .....                     | 61        |
| <b>8.</b> | <b>Technical Data</b>                        | <b>63</b> |
| 8.1.      | Mechanical Data.....                         | 63        |
| 8.2.      | Electrical Data.....                         | 63        |
| 8.2.1.    | Supply Voltage .....                         | 63        |
| 8.3.      | Ambient Conditions .....                     | 63        |

---

|            |  |           |
|------------|--|-----------|
| <b>9.</b>  | <b>Component Placement Plan</b>                  | <b>65</b> |
| 9.1.       | Component (Top) Side .....                       | 65        |
| 9.2.       | Solder (Bottom) Side .....                       | 66        |
| 9.3.       | Positions of Resistors for Shared Interrupt..... | 67        |
| <b>10.</b> | <b>Supplemental Information on Components</b>    | <b>69</b> |

# 1. Overview

## 1.1. Introduction

HiCO332 is an intelligent and universal card for all types of motion control applications. In particular, the many different ports and powerful processor allow the HiCO332 to be implemented wherever high performance is required in a very small space. Figure 1 shows a simplified description of the card architecture.

## 1.2. Block Diagram

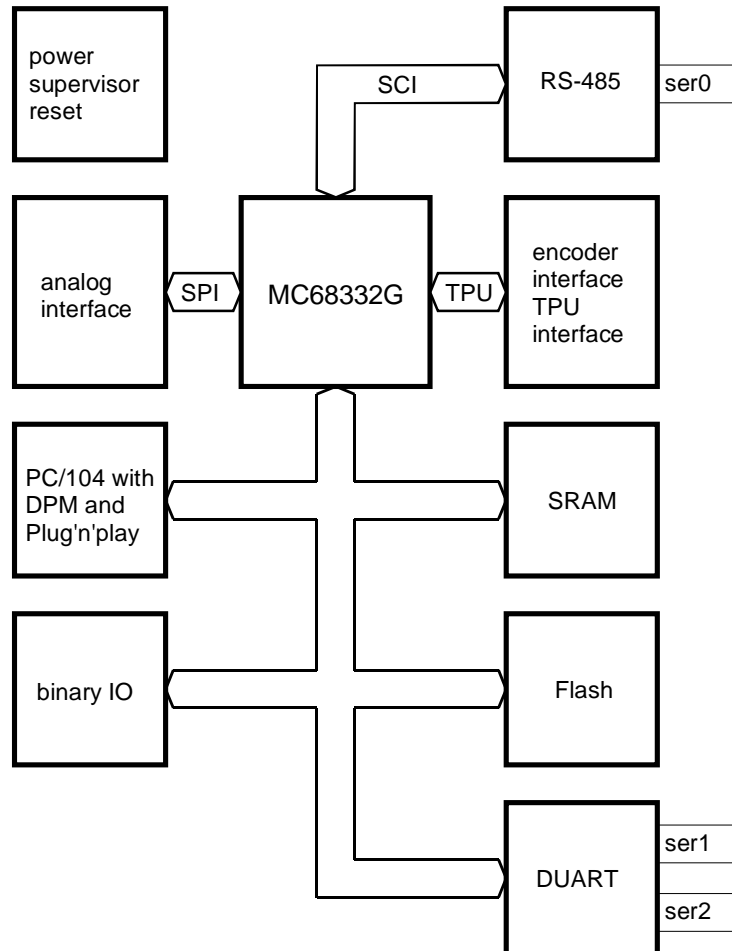


Figure 1: Architecture of the HiCO332 Card

### 1.3. Basic Characteristics of the HiCO332

| Feature               | Function   |
|-----------------------|--|
| Processor             | MC68332G   |
| Clock frequency       | 20 MHz (PLL)   |
| SRAM battery buffered | 128KByte (16 Bit)  |
| Flash                 | 512KByte (16 Bit)  |
| Dual-ported memory    | 2KByte (8 Bit)   |
| Serial interface      | 2 x RS232, 1 x RS485   |
| Analog interface      | 8 x analog input, 2 x analog output                                    |
| Binary interface      | 8 x TTL input, 8 x TTL output<br>8 x 24V input, 8 x 24V output         |
| Encoder interface     | 2 x encoder interfaces with A/#A, B/#B<br>and Z/#Z, RS485 level or TTL |
| TPU interface         | 10 TPU channels, TTL   |
| PC interface          | PC/104   |
| Interrupts on ISA-Bus | via PnP controller   |
| DPM base address      | via PnP controller   |
| Shared Interrupt      | IRQ5   |
| Power supply          | 5 VDC, +- 15VDC, 350 mA  |
| Backup-Battery        | External battery, 3V   |
| Operation temperature | 0°C to +70°C   |
| Storage temperature   | -40°C to +85°C   |
| Dimensions            | 96 x 90 x 20 mm <sup>3</sup>   |

## 2. Starting Up the Card

Before you install the HiCO332, please read and observe the following important instructions. These instructions apply to all cards with electrostatically sensitive components.

- Before touching the card, please electrostatically discharge yourself by touching a grounded object.
- All tools used to install the card must be free of electrostatic charges.
- Unplug the electrical power cord for your device before installing or removing the card.
- Take hold of the card only along the PC-board edges and avoid touching any components or conductive traces.

HiCO332 is a card in PC/104 format. It may be operated separately or in a PC/104 system. The following supplemental points should be observed when installing the card:

- Before installing the card check to make sure that all plug-in jumpers are configured properly.
- HiCO332 is connected to other components by the 104-pin plug-in connectors X2, X3. Four mounting holes in the corners of the PC-board are used to mechanically fasten the connectors.
- Connect all necessary peripheral components and connect the supplemental X8 voltage supply.
- If HiCO332 is operated as an autonomous device, the voltage supply may be fed-in exclusively over the X8 voltage supply connection.
- Before switching on the power, please check the correct position of all the connectors and the power supply (5V, +-15V and 24V)

## 3. System Description

### 3.1. Processor Core and Memory

The processor used is the Motorola MC68332G-20 microcontroller. The processor's components include three subprocessors which act independently of one another: CPU32, TPU and QSPI.

The CPU32 is the actual processor for executing the program. The QSPI is used for synchronous serial or asynchronous serial communication. The TPU (Timing Processing Unit) can handle autonomous tasks such as generating signal strings or acquiring or analyzing signal strings.

With the G types of processor the special TPU micro programs are provided, designed especially for motion control tasks. These are simple counter or timer programs, FQD (Fast Quadrature Decoder), NITC (New Input Timing Capture) and functions as complex as PWM (Pulse Width Modulation) or drive functions for stepper motors.

Please refer to Motorola documentation [12] and [13] for detailed information.

### 3.1.1. Pulse Generation

The pulse frequency is generated by a 32768Hz external crystal clock. The internal, programmable PLL uses this to generate the internal system clock of up to 20.78MHz.

The DUART is supplied with its own crystal oscillator at 3.6872MHz.

### 3.1.2. Chip Select Signals

The following processor chip select signals are provided:

- **#CSBOOT**: Chip select for the flash chip.  
16 bit data bus, size 256K x 16 bits, 70ns, 0 Waitstates, internal generation of #DSACK.
- **#CS0, #CS1**: Chip select for the SRAM.  
16 bit data bus, size 64K x 16 bits, 0 Waitstates, internal generation of #DSACK. #CS0 selects the SRAM, #CS1 selects the lower byte; the upper byte is selected by ADR0.
- **#CS2**: Chip select for the DPM.  
8 bit data bus, size 2K x 8 bits, 0 Waitstates, external generation of #DSACK via #BUSY of the DPM. Read-modify-write cycles are locked during semaphore operations like the TAS command. Internal autovectoring for interrupts must be set up via #CS5.
- **#CS3**: Chip select for the DUART.  
8 bit data bus, 16 IO addresses, external generation of #DSACK
- **#CS4**: Chip select for binary interface.

8 bit data bus, four IO addresses.

Common to all chip select signals is the address strobe characteristic.

### 3.1.3. Interrupts

The CPU32 has 7 interrupt priorities, whereby 1 represents the lowest and 7 the highest priority. On the HiCO332 the interrupts 3 through 6 can be allocated to the external sources DUART (Signal: #IRQ\_UAR) and DPM (Signal: #IRQ\_DPM) by jumpers:

| Source | Ext/int | Priority | Function                                  |
|--------|---------|----------|---|
| DPM    | Ext.    | 3 – 6    | Interrupt by the DPM for PC communication |
| DUART  | Ext.    | 3 – 6    | RS-232 ports ser1 and ser2                |
| SCI    | Int.    | 1 – 7    | RS-485 port ser0                          |
| Timer  | Int     | 1 - 7    | PIT                                       |
| TPU    | Int     | 1 - 7    | Example: Encoder                          |

The external interrupts may not be fed in over a common input. The following jumper settings are necessary for the allocations given below:

| Jumper | Closed | Function          |
|--------|--------|-------------------|
| X12    | 1-2    | #IRQ_DPM at #IRQ3 |
|        | 2-3    | #IRQ_UAR at #IRQ3 |
| X11    | 1-2    | #IRQ_DPM at #IRQ4 |
|        | 2-3    | #IRQ_UAR at #IRQ4 |
| X10    | 1-2    | #IRQ_DPM at #IRQ5 |
|        | 2-3    | #IRQ_UAR at #IRQ5 |
| X9     | 1-2    | #IRQ_DPM at #IRQ6 |
|        | 2-3    | #IRQ_UAR at #IRQ6 |

### 3.1.4. Port Connections of the MC68332

#### 3.1.4.1. Port E

| Port   | Name     | Function                     |
|--------|----------|------------------------------|
| PORTE0 |          | Reserved internally          |
| PORTE1 |          | -                            |
| PORTE2 |          | Reserved internally          |
| PORTE3 |          | Reserved internally          |
| PORTE4 | #EN_ENC1 | Enable: Encoder channel 1    |
| PORTE5 |          | Reserved internally          |
| PORTE6 | #EN_ENC2 | Enable: Encoder channel 2    |
| PORTE7 | #PO_FAIL | Monitoring of voltage supply |

### 3.1.4.2. Port F

| Port   | Name     | Function                 |
|--------|----------|--------------------------|
| PORTF0 | #B_24_ON | Enable: 24V interface    |
| PORTF1 | #A_O_ON  | Enable: Analog interface |
| PORTF2 | #DIAG_24 | Diagnosis: 24V interface |
| PORTF3 |          | Reserved internally      |
| PORTF4 |          | Reserved internally      |
| PORTF5 |          | Reserved internally      |
| PORTF6 |          | Reserved internally      |
| PORTF7 | #B_O_ON  | Enable: TTL interface    |

### 3.1.4.3. Port QS

| Port         | Name     | Function                  |
|--------------|----------|---------------------------|
| PQS0 to PQS2 |          | SPI interface             |
| PQS3         | #PCS_AD1 | CS for A/D converter 1    |
| PQS4         | #PCS_DA1 | CS for D/A converter 1    |
| PQS5         | #PCS_DA2 | CS for D/A converter 2    |
| PQS6         | #CONVERT | Not used, must be logic 1 |

### 3.1.4.4. TPU

| Port        | Name | Function              |
|-------------|------|-----------------------|
| TPU.0       | Z_1  | Zero pulse: Encoder 1 |
| TPU.1       | A_1  | Channel A, Encoder 1  |
| TPU.2       | B_1  | Channel B, Encoder 1  |
| TPU.3       | Z_2  | Zero pulse: Encoder 2 |
| TPU.4       | A_2  | Channel A Encoder 2   |
| TPU.5       | B_2  | Channel B Encoder 2   |
| TPU.6 to 15 |      | Available for use     |

## 3.2. Memory

A total of three memory areas are available to the user: SRAM, Flash and DPM.

### 3.2.1. Flash

Used for program memory is a 256kx16bit flash memory (Am29F400B). This flash is only supplied with 5V and can be erased and programmed in the system. It is sectorized, i.e. it is possible to erase and write sector-by-sector.

For additional information, in particular regarding the structure of sectors and the programming algorithm please refer to the data sheet [3].

### 3.2.2. SRAM

A 64kx16bit SRAM (KM6161000BLT) is used for data memory. The memory contents can be buffered by a battery (3V) that is not located on the HiCO332.

### 3.2.3. Dual Ported Memory

The 2kx8bit Dual Ported Memory (CY7C132) is only available in the PC/104 variant of the HiCO332.

This chip provides control signals independently for both sides (MC68332 and PC via the PC/104 bus). Conflicts due to accessing the same cells are prevented by means of Busy signals. Furthermore, read-modify-write accesses are supported, allowing the MC68332 to handle semaphores. Interrupts can be triggered in both directions. Please refer to the data sheet [4] for further information.

---

**ATTENTION:**

The Busy signal is used to generate the DSACK signal for the MC68332. When the MC68332's bus monitor is active, consideration must be given to the duration that the PC can block the DPM. If problems should arise, it may be necessary to ensure - by programming - that between each DPM access the PC makes an access outside of the DPM address space but within the ISA address space. This is the only way to cancel a Busy signal that might be active.

---

## 3.3. Binary Ports

### 3.3.1. 24V Inputs/Outputs

Each of the eight inputs and outputs are electrically isolated by an optocoupler and are supplied externally. On the HiCO332 card are L6376 high-side switches from SGS-Thomson, which can drive a maximum current of 100mA.

The inputs are each terminated with a pull-down resistor of 4.7KOhm. This means that an open input (24V side) yields a logic 0.

The outputs are designed as readback and reset capable registers. After a HW reset (Power-on or dedicated reset signal from the PC) the outputs are deactivated and switched to logic 0 by 100KOhm pull-down resistors. The high-side switches are opened.

The output registers are enabled with a positive edge of the #B\_24\_ON signal. The idle value, or value after a HW reset of this signal is logic 1.

The signal is implemented by a port pin of the HiCO332.

| Port   | Name     | Function                | HW Reset |
|--------|----------|-------------------------|----------|
| PORTF0 | #B_24_ON | 0→1: Enable 24V outputs | 1        |

If an error occurs on the output side of the high-side switches, e.g. short circuit, overtemperature, overvoltage or undervoltage, a diagnostic signal is activated at PORTF2. The firmware is responsible for handling this error.

| Port   | Name     | Function  |
|--------|----------|---|
| PORTF2 | #DIAG_24 | 0: Error in the high-side-switches<br>1: No error |

### 3.3.2. TTL Inputs/Outputs

The 8 inputs are terminated with a 100kOhm pull-down resistor. An open input always yields a logic zero when it is read.

The 8 outputs are designed as readback and reset capable registers. After a HW reset (Power-on or dedicated reset signal from the PC) the outputs are deactivated and switched to logic 0 by the 100KOhm pull-down resistors.

The output registers are enabled on a positive edge of the signal #B\_O\_ON. The idle value or value after a HW reset of this signal is logic 1.

The signal is implemented by a port pins of the '332.

| Port   | Name    | Function                | HW Reset |
|--------|---------|-------------------------|----------|
| PORTF7 | #B_O_ON | 0→1: Enable TTL outputs | 1        |

## 3.4. Asynchronous Serial Ports

### 3.4.1. RS-232 Ports

The two RS-232 ports ser1 and ser2 are implemented by the intelligent SCC68692 DUART from Philips. The handshake signals RTS and CTS are implemented by the

DUART's port pins. The DUART is capable of servicing the handshake signals autonomously. The firmware only needs to initialize this functionality.

Please refer to the data sheet [5] for further instructions on programming.

The level conversion is performed by the ADM211 chip. The ports are not electrically isolated.

### 3.4.2. RS-485 Port

This port ser0 is implemented by the SCI of the MC68332 as a four-wire interface. The signal #485\_OFF, output port of the DUART, can be used to toggle between transmit and receive. It is not possible to transmit and receive simultaneously.

The user can obtain a two-wire interface from the four-wire interface by connecting the transmitter with the receiver.

| Port | Name     | Function   |
|------|----------|--|
| OP7  | #485_off | 0: Transmitter deactivated, receiver activated<br>1: Transmitter activated, receiver deactivated |

The level conversion is performed by standard RS-485 transceiver chips. The interface is not electrically isolated.

As an option, the receiving channel may be terminated with a resistor network (470 Ohm to GND, 150 Ohm between signal lines, 470 Ohm to Vcc). This involves setting the appropriate jumpers on the PC board.

### 3.4.3. Ports of the DUART

The port connections of the DUART are used primarily for the handshake signals of the two RS-232 ports, and they do not need to be serviced by the firmware.

The port connections OP 5 to 7 are used for selection of the analog multiplexer and for transfer direction control of the RS-485 port.

Two registers of the DUART are provided for setting or resetting the outputs (see [5]). A special condition should be noted with regard to the SCC68692. Setting of an output port means physically that the port connection goes low. Similarly, clearing results in an output of high.

Input ports:

| Port     | Name  | Function      |
|----------|-------|---------------|
| IP0      | #CTS1 | #CTS for ser1 |
| IP1      | #CTS2 | #CTS for ser2 |
| IP2 to 5 |       | Available     |

Output ports:

| Port    | Name     | Function                         |
|---------|----------|----------------------------------|
| OP0     | #RTS1    | #RTS for ser1                    |
| OP1     | #RTS2    | #RTS for ser2                    |
| OP2...4 |          | Not used                         |
| OP5     | ANA_SEL0 | Channel select for analog inputs |
| OP6     | ANA_SEL1 | Channel select for analog inputs |
| OP7     | #485_OFF | TX enable: RS-485 port ser0      |

### 3.5. Analog Ports

There are a total of eight analog inputs and two analog outputs on the HiCO332 card. Analog/digital conversion is performed by the AD7890-10 serial A/D converter from Analog Devices [6], and digital/analog conversion is performed by the MAX531 chip from Maxim [7].

To achieve continuous data flow and relieve loading of the CPU32, the QSPI of the 68332 is used for communication with the A/D and D/A converters. The connection is not made as a daisy chain, rather it is quasi parallel at the synchronous serial port. Chip select signals are offered for selection by the QSPI.

After suitable programming, the QSPI is capable of cyclically sending messages to all nodes of the synchronous serial bus and receiving their reply messages. The

application only needs to write to the transfer register area of the QSPI or read-out the receive register area. The QSPI ensures consistency of the data.

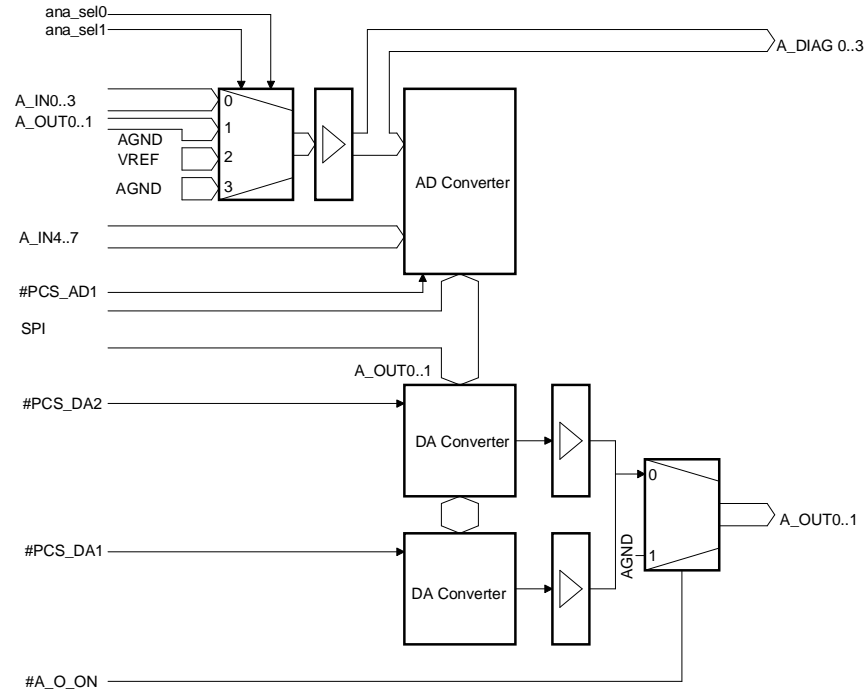


Figure 2: Block Circuit Diagram of the Analog Interface

### 3.5.1. Analog Inputs

The eight ground-referenced inputs are designed for an input voltage range of  $\pm 10V$ . The conversion is made with 12-bit resolution. Coding is in two's complement.

The inputs `A_IN0` to 3 are fed into the input of a 4x(4:1) analog multiplexer through an overvoltage protector and a 2<sup>nd</sup> order active filter. The input amplifier acts as an impedance converter. The filter is sized by the RC sections, and the limit frequency is approx. 1KHz.

As a function of the signals ANA\_SEL0 and ANA\_SEL1 the multiplexer connects either the analog input signal, ground, internally conditioned reference voltage of 2.5V or the D/A converter outputs to inputs ANA0 to 3 of the A/D converter (see Figure 2 and the following table).

| ANA_SEL1 | ANA_SEL0 | Function  |
|----------|----------|---|
| 0        | 0        | Switch A_IN0 to 3 to the analog inputs of the A/D converter   |
| 0        | 1        | Switch 2.5V reference voltage to the analog inputs of the A/D converter   |
| 1        | 0        | Connect the analog inputs ANA0 and ANA1 of the A/D converter to the analog outputs of the D/A converter, and connect the remaining two to ground. |
| 1        | 1        | Connect ground to the A/D converter's analog inputs.<br>This is also the position after a HW Reset.   |

The analog multiplexer can thereby also be utilized for amplification- and zero point adjustment.

The signals ANA\_SEL0 and ANA\_SEL1 are obtained from the DUART's port signals.

The analog inputs ANA0 to 3 of the A/D converter are fed out for diagnostic purposes without additional circuitry. The adapter for test equipment must have high resistance; otherwise feedback effects may occur.

The inputs A\_IN4 to 7 are fed directly to the A/D converter's inputs through an overvoltage protector, but without additional circuitry.

### 3.5.2. Analog Outputs

The output supplies a voltage range of  $\pm 10V$  with a resolution of 12 bits. Coding is binary. Due to special hardware properties the value to be output must be shifted by one bit position to the left when it is written to the transfer register.

The D/A converter converts the binary value to a ground-referenced voltage. A subsequent amplification stage provides for sufficient internal resistance.

A maximum current of 5mA is available.

Located after the particular D/A converter output is an analog multiplexer which switches the output to 0V on a HW reset or in response to a special signal.

After the HW reset the output remains at 0V and can only be enabled again by a special signal.

### 3.5.3. Behavior On Reset, Enabling of Outputs

After a HW reset the outputs of the D/A converter are disabled, and instead 0V is output by means of an analog multiplexer.

Simultaneously, the analog inputs are switched to ground by an analog multiplexer. At this point use is made of the situation where the processor's port outputs are switched to inputs after a reset and are connected to Vcc via a pull-up. The DUART outputs are set to high by this reset.

The analog multiplexer of the D/A converter can be switched to its operating position again by the signal #A\_O\_ON.

| Port   | Name    | Function  | HW Reset |
|--------|---------|---|----------|
| PORTF1 | #A_O_ON | 0: Enable D/A outputs<br>1: Disable D/A output, output 0V | 1        |

## 3.6. TPU Interface

The TPU offers user-friendly methods for analyzing the time relationships of signals, counting events and even generating complex pulse sequences. Various microprograms are included in the TPU for this purpose, and they are very easy to initialize and use.

The TPU of the MC68332G processor contains specific programs for motion control applications. Among others, the FQD (Fast Quadrature Decoding) is worthy of mention.

One option that permits quick and easy connection of an angular or incremental encoder is a direct connection to the TPU. External adaptation circuits must provide

for appropriate level adaptation. Another option is to utilize the RS-485 drivers that are already on-board (without electrical isolation), and which serve as level converters for a total of two encoders (see Figure 3).

Both encoder interfaces take the tracks A and B as well as the zero pulse Z into consideration.

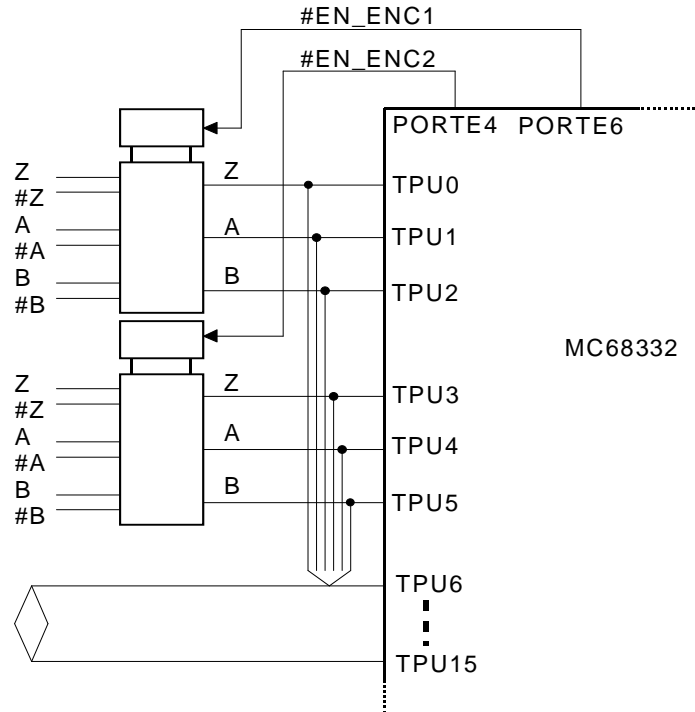


Figure 3: TPU Connection

The choice between RS-485 connection or direct connection is made using the control signals #EN\_ENC1 and #EN\_ENC2 as described in the following table.

| Port | Name | Function | HW Reset |
|------|------|----------|----------|
|------|------|----------|----------|

|        |          |  |   |
|--------|----------|--|---|
| PORTE4 | #en_enc1 | 0: Activate RS-485 driver for encoder 1<br>1: Deactivate RS-485 driver for encoder 1 | 1 |
| PORTE6 | #en_enc2 | 0: Activate RS-485 driver for encoder 2<br>1: Deactivate RS-485 driver for encoder 2 | 1 |

If channels 0 - 5 of the TPU are programmed as inputs of the FQD, this yields the following assignments:

| Port       | Name    | Function                      |
|------------|---------|-------------------------------|
| TPU0       | Z_1_TPU | Zero pulse: Encoder channel 1 |
| TPU1       | A_1_TPU | Track A: Encoder channel 1    |
| TPU2       | B_1_TPU | Track B: Encoder channel 1    |
| TPU3       | Z_2_TPU | Zero pulse: Encoder channel 2 |
| TPU4       | A_2_TPU | Track A: Encoder channel 2    |
| TPU5       | B_2_TPU | Track B: Encoder channel 2    |
| TPU6 to 15 | -       | Available for use             |

The remaining TPU channels are available directly and without additional circuitry.



## 4. Supply Voltage

### 4.1. Supply for the Analog Section

A voltage supply of  $\pm 15V$  is needed for the analog section of the HiCO332. This must be fed in via connector X8. It is not possible to supply the analog section of the card via the PC/104 bus. This voltage supply is monitored by comparators. If the value of one of the two voltages drops below the allowable value, the signal #PO\_FAIL is activated.

The firmware must evaluate this signal accordingly.

| Port   | Name     | Function   |
|--------|----------|--|
| PORTE7 | #PO_FAIL | 1: Analog supply OK, i.e. $>11V$ or $< -11V$<br>0: Analog supply failure. $<11V$ or $> -11V$ |

### 4.2. Supply for the Digital Section

The voltage is supplied at 5V with approx. 350mA. The supply voltage can either be fed-in indirectly via the PC/104 connector or directly via the X8 voltage supply connector.

If the voltage supply falls below the threshold value of 4.5V, a reset is triggered on the HiCO332. This reset does not affect other nodes on the PC/104 bus.

### 4.3. Battery

A 3V battery for buffering the SRAM contents can also be connected via the X8 connector.

## 5. Interface to the PC/104 Bus

The card is connected to the ISA bus by an NM95MS15 plug and play controller from National Semiconductor/Fairchild [8],[9]. This chip manages the IO address space's base address and size, the allocation of interrupt numbers, and the memory area's base address and size.

The controller is fully programmed. The allowable memory area lies between the segments D800 and EF00, and the allowable IO area between the addresses 0x200 and 0x3ff. The allowable interrupt numbers are: 5, 7, 9, 10, 11, 12 and 15. The configured data can either be taken directly from the controller or from the PC's ESCD database. See also the literature [10], [11] regarding this.

If there are no other cards on the PC/104 bus, the DPM lies in the D800 segment with offset 0, the IO address lies at 0x200 and the interrupt number is set (provided it is not in use by other components of the PC) to IRQ5. However, this may differ from PC manufacturer to PC manufacturer.

## 5.1. Software Reset by PC

The reset is performed by a write access to the IO base address. The reset has been executed successfully when there is an acknowledge from the firmware via the DPM.

## 5.2. Interrupts

The Interrupt Sharing Mechanism proposed by the PC/104 Consortium has been implemented, so that multiple MSRs can share one interrupt of the ISA bus. Interrupt numbers 5, 10, 11 and 15 can be made share-capable by suitably populating two resistors.

In the delivered state the number 5 is set up as the shared interrupt; and the other interrupts are configured as standard interrupts.

| Interrupt | Populated Resistor | Function                     |
|-----------|--------------------|------------------------------|
| 5         | R9, R6 open        | Not shared                   |
|           | R6, R9 open        | Shared (delivered state)     |
| 10        | R11, R4 open       | Not shared (delivered state) |
|           | R4, R11 open       | Shared                       |
| 11        | R10, R5 open       | Not shared (delivered state) |
|           | R5, R10 open       | Shared                       |
| 15        | R8, R7 open        | Not shared (delivered state) |
|           | R7, R8 open        | Shared                       |

## **6. Firmware**

### **6.1. Package Contents**

The delivery of HiCODRV includes the documentation and a CD containing the required files.

Please take note of the \_readme.txt file which can also be found on the CD. This file contains information that was unavailable when the printed documentation was prepared.

### **6.2. Installation**

Copy all files from the directories:

- INCLUDE
- LIB

to your working directory.

## 6.3. Hardware Requirements

HiCODRV runs on the HiCO332 board. The software supports the following hardware features:

- 8 24V inputs
- 8 24V outputs
- 8 TTL outputs
- 8 TTL inputs
- 8 analog inputs
- 2 analog outputs
- 2 quadrature encoder
- 3 serial ports
- DPM communication with a Host PC via PC/104 bus
- Loading of the firmware via DPM interface
- Deletion and writing of 2 Flash sectors with 16KB each

In order to properly start the application that is linked with the `hicodrv.lib` library, a bootstrap loader v1.04 or later is needed.

## 6.4. Bootstrap Loader

### 6.4.1. Initializing and Starting the Firmware

The bootstrap loader first initializes the hardware. This includes the deletion of the DPM on the PC/104 Bus. After that 0xa5 appears at the DPM address 0x7e1. The bootstrap loader then checks its own flash area by way of a checksum. If this checksum does not correspond to the one entered, 0x88 will be entered in the DPM address 0x7e3, the check sum read (16-bit value) in DPM address 0x00, the calculated checksum entered in DPM address 0x02, and the CPU halted.

After checking its own checksum the bootstrap loader verifies the firmware's checksum. If successful, the firmware will be started at 0x88020. If the checksum is incorrect, bits 7 and 5 are set in DPM address 0x7e3, and the program jumps to the function loading a firmware. See section "Application Interface" on p. 32 for a more detailed description of this function.

## 6.4.2. Application Interface

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

Returns: 8-bit version- and 8-bit revision number of the loader

**BOOT\_VERS** yields both version- and revision number of the bootstrap loader. The higher byte of the return value contains the 8-bit-wide version number, the lower byte the 8-bit-wide revision number.

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

| src | Source from where the firmware is loaded |
|-----|--|
| 0   | ser 0                                    |
| 1   | ser 1                                    |
| 2   | ser 2                                    |
| 16  | DPM                                      |

Returns:

|    |   |
|----|---|
| -1 | Source unknown                            |
| -2 | Initialization error during communication |

The function can be called from within the application and serves to load a new firmware. After deleting the flash, the function does not return to the calling location any more, but executes a cold start. This means that the bootstrap loader is started just like after a hardware reset, performing the hardware initialization and checksum verification. The "src" parameter indicates the source from where the firmware is loaded. Currently the bootstrap loader only supports 0x10 for the "src" parameter for loading the firmware via DPM.

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

|        |   |
|--------|---|
| *start | Start address of the sector to be deleted |
|--------|---|

Returns:

|   |                            |
|---|----------------------------|
| 0 | Block deleted              |
| 1 | Invalid start address      |
| 2 | Flash error while deleting |

Valid start addresses are the 8-KB values 0x84000 and 0x86000. All interrupts are disabled during the deletion.

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

|              |   |
|--------------|---|
| *destination | Start address of the block in the flash |
| *source      | Start address of the source data        |
| *wordcount   | Number of the words to be counted       |

Returns:

|   |                                       |
|---|---------------------------------------|
| 0 | Data written                          |
| 1 | Odd source- or target address         |
| 2 | A target address is outside the block |
| 3 | Flash error during writing            |

The 2 start addresses must be on even addresses. In order to correctly write data the flash area to be written must be deleted in advance. If this is not done, FL\_BL\_WR either returns the value 0 or 3, depending on the previous contents of the locations to be written. Only 16-bit-wide values may be written. All interrupts are disabled while the flash is being programmed.

## 6.5. Application Interface

### 6.5.1. Configuration and Firmware Start

After executing the bootstrap area, the 'start' label in the 'start.s' module is accessed. You are able to enter application-specific initializations in assembler in this module. Start.s jumps to sysinit.c. Here, the configuration data for the timer and interrupt priorities may be set. In addition, application-specific initializations may be entered in C. At the end of all the initializations, a jump to the application (i.e. main) is performed.

The timer\_init function serves to initialize the interrupt timer.

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

|          |   |
|----------|---|
| prio     | Interrupt level for the timer interrupt         |
| millisec | Number of milliseconds for the clock {1..15938} |

Returns:            Time set in milliseconds

Since the timer cannot be exactly set in steps of milliseconds, the function returns the time actually set.

By default, the call of the timer\_init function is contained within sysinit.c, with the default values being PIT\_IRQ (6) and 10 ms.

For the configuration of the driver various variables are available which can also be set in sysinit.c.

### 6.5.2. Encoder

The encoders are directly connected to the 68332's TPU via RS485 drivers. The TPU provides functions to capture the quadrature signals.

#### **short TPUInit (short ch)**

|    |                       |
|----|-----------------------|
| ch | Channel number {1, 2} |
|----|-----------------------|

Returns

|     |                         |
|-----|-------------------------|
| 0:  | Successful              |
| -1: | Invalid channel numbers |

***TPUinit*** initializes the TPU with the functions FQD and NITC for the corresponding channel.

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

|      |   |                              |
|------|---|------------------------------|
| ch   | Channel number {1, 2}                       |                              |
| mode | Connection of the encoder inputs to the TPU |                              |
|      | TPU_RS485                                   | Connection via RS-485 driver |
|      | TPU_TTL                                     | Connection via TTL signals   |

Returns:

|    |                         |
|----|-------------------------|
| 0  | Successful              |
| -1 | Invalid channel number  |
| -2 | Invalid connection mode |

***TPUMode*** configures the encoder interface for the selected channel. This can be done with the RS485 interface or a direct connection via TTL signal.

#### **short TPUReset (short ch)**

|    |                       |
|----|-----------------------|
| ch | Channel number {0, 1} |
|----|-----------------------|

Returns:

|    |                        |
|----|------------------------|
| 0  | Successful             |
| -1 | Illegal channel number |

**TPUReset** resets all channel counters to 0.

**short TPURed (short ch, short \*pos, short \*edge\_time, short \*time\_base, short \*z\_pulse)**

|           |                                   |
|-----------|-----------------------------------|
| ch        | Channel number {0, 1}             |
| pos       | Current value of position counter |
| edge_time | Time of last edge change          |
| time_base | Time base of edge analysis        |
| Z_pulse   | Number of Z-pulses                |

Returns:

|    |                         |
|----|-------------------------|
| 0  | Successful              |
| -1 | Illegal channel numbers |

**TPURed** returns the values of the requested channel.

**short TPURedPos (short ch, short \*pos)**

|     |                                       |
|-----|---------------------------------------|
| ch  | Channel number {0, 1}                 |
| pos | Current value of the position counter |

Returns:

|    |                        |
|----|------------------------|
| 0  | Successful             |
| -1 | Illegal channel number |

*TPUReadPos* only returns the value for the position counter.

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

|    |                       |
|----|-----------------------|
| ch | Channel number {0, 1} |
|----|-----------------------|

Returns:

|    |                        |
|----|------------------------|
| 0  | Successful             |
| -1 | Illegal channel number |

*TPUIntInstall* installs an application interrupt handler. This interrupt handler is called with each NITC interrupt of the TPU and is used to analyze the encoder's Z track.

### **short TPUIntEnable (short ch)**

|    |                       |
|----|-----------------------|
| ch | Channel number {0, 1} |
|----|-----------------------|

Returns:

|    |                                |
|----|--------------------------------|
| 0  | Successful                     |
| -1 | Illegal channel number         |
| -2 | No interrupt handler installed |

***TPUIntEnable*** enables the interrupt for the requested channel. The interrupt handler must have been previously installed through *TPUIntInstall*.

#### **short TPUIntDisable (short ch)**

|    |                       |
|----|-----------------------|
| ch | Channel number {0, 1} |
|----|-----------------------|

Returns:

|    |                        |
|----|------------------------|
| 0  | Successful             |
| -1 | Illegal channel number |

***TPUIntDisable*** disables the interrupt for the requested channel.

### **6.5.3. Analog Interface**

The QSPI of the 68332 is used to read in the values of the A/D converters in the looping mode. Consequently values which are maximally 200 us old are valid at any time. The values supplied by the converters have a width of 12 bits.

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

|         |                           |
|---------|---------------------------|
| inChan  | Number of input channels  |
| outChan | Number of output channels |

Returns:

|    |                            |
|----|----------------------------|
| 0  | Successful                 |
| -1 | Invalid number of channels |

***analogConfig*** configures the analog I/O channels.

If no analog output is implemented on the HW, 0 must be specified for the number of output channels.

#### **short analogIn (short ch)**

|    |                          |
|----|--------------------------|
| ch | Channel number {0,...,7} |
|----|--------------------------|

Return:

Value of the channel in the two's complement

***analogIn*** returns the value of an analog input channel as a 16-bit-wide two's complement value. If a non-initialized channel is selected, an undefined value will be returned.

#### **short analogInSrc (short source)**

|        |   |                                  |
|--------|---|----------------------------------|
| source | 0 | Ground                           |
|        | 1 | Analog outputs                   |
|        | 2 | Reference voltage (2.5V)         |
|        | 3 | Analog amplifier (analog inputs) |

Returns:

|    |                          |
|----|--------------------------|
| 0  | Successful               |
| -1 | Illegal value for source |

***analogInSrc*** selects the input source for the channels 0 to 3.

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

|    |                       |
|----|-----------------------|
| ch | Channel number {0, 1} |
|----|-----------------------|

|       |              |
|-------|--------------|
| Value | Output value |
|-------|--------------|

Returns

|    |                      |
|----|----------------------|
| 0  | Successful           |
| -1 | Illegal value for ch |

***analogOut*** sets the 16-bit-wide two's complement value on the selected analog channel.

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

Returns:

|    |                                 |
|----|---------------------------------|
| 0  | Analog power supply correct     |
| -1 | Analog power supply not correct |

***analogPower*** checks if the analog power supply (+-15V) is available. An error is reported when the voltage is below 11V.

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

Return:

|   |    |
|---|----|
| 0 | ok |
|---|----|

***analogOutEn*** enables the output multiplexer for the analog output voltage.

### 6.5.4. Binary Inputs/Outputs

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

|        |                  |
|--------|------------------|
| outVal | new output value |
|--------|------------------|

***binOutTTLUpdate*** outputs a new value to the TTL outputs.

**byte binInTTLUpdate (void)**

Return:

8-bit input value

***binInTTLUpdate*** returns the debounced value of the TTL inputs.

**short binOut24Update (byte outVal)**

|        |                  |
|--------|------------------|
| outVal | new output value |
|--------|------------------|

Returns:

|    |   |
|----|---|
| 0  | Successful                              |
| -1 | Error message on the high-side switches |

***binOut24Update*** returns a new value to the 24-VDC outputs.  
An error value is returned when the high-side switches report an error.

**byte binIn24Update (void)**

Return:

8-bit input value

***binIn24Update*** returns the debounced value of the 24-VDC inputs.

**short binOut24En (void)**

Returns:

|    |   |
|----|---|
| 0  | Successful                              |
| -1 | Error message on the high-side switches |

***binOut24En*** enables the 24-VDC outputs.

**void binOutTTLEn (void)**

***binOutTTLEn*** enables the TTL outputs.

### 6.5.5. Serial Ports

HiCO332 provides 3 serial ports called ser 0, ser 1 and ser 2. Ser 0 is implemented with the SCI of the 68332. Ser 1 and ser 2 are implemented via an additional DUART and connected via RS232 driver devices.

Ser 0 is designed for use in half duplex mode on a two-wire line. The driver of the transmitter is only enabled during transmit phases. Receives are not possible. The driver device is enabled via an entry of a character in the transmit buffer (using the functions putStr or putCh). The driver device is disabled after a character has been transmitted, if there is no further character in the transmit buffer. With high baud rates chances are that the driver device is disabled after each character until the next character/bit is being entered in the transmit buffer.

Since the RS485 interface is in the half duplex mode, no handshake can be set for ser 0.

Usually, all ports are interrupt-driven. During initialization the handshake mode to be used may be set.

The buffers for transmit and receive characters are allocated by the application layer and used during initialization. After initialization these buffers are no longer available for the application layer.

**int serInit (sCOMINIT \*cominit)**

|         |           |
|---------|-----------|
| cominit | Structure |
|---------|-----------|

Returns:

|     |  |
|-----|--|
| 0   | Initialization successful                            |
| -1  | invalid value for com                                |
| -2  | invalid value for bpchar                             |
| -3  | invalid value for stop                               |
| -4  | invalid value for parity                             |
| -5  | invalid value for baud                               |
| -6  | invalid value for hndShk                             |
| -7  | Handshake for com 0 disallowed                       |
| -10 | invalid value for baud for this com                  |
| -11 | invalid combination of bpchar, stop, parity for com0 |

**serInit** initializes a serial port.

The FIFOs for transmit and receive are established, the port's hardware is initialized, the interrupt specified and the receive interrupt enabled.

A variable of the sCOMINIT structure contains the necessary information to perform initialization.

```
typedef struct COMINIT
{
    byte    com;                // Port number {0..2}
    byte    bpchar;            // Bits per character
    byte    stop;              // Stop bits
    byte    parity;            // Parity
    lword   baud;              // Baud rate
    byte    hndShk;            // Handshake: none, HW, SW
    byte    *recBufStart;      // Start receive buffer
    lword   recBufSize;        // Length (bytes) receive buffer
    byte    *traBufStart;      // Start transmit buffer
    lword   traBufSize;        // Length (bytes) transmit buffer
} sCOMINIT;
```

The following Table shows the constants defined for the structure's elements and the valid values for the ports.

| Element | Constant                                 | ser 0  | ser 1 and ser 2                 |
|---------|--|--|---------------------------------|
| bpchar  | BPC_5<br>BPC_6<br>BPC_7<br>BPC_8         | No<br>No<br>with 2 stop<br>bits<br>with 1 stop bit | Yes<br>Yes<br>Yes<br>Yes        |
| stop    | S_BITS1<br>S_BITS2                       | Yes<br>Yes   | Yes<br>Yes                      |
| parity  | P_NONE<br>P_EVEN<br>P_ODD                | Yes<br>Yes<br>Yes                                  | Yes<br>Yes<br>Yes               |
| baud    | B4_8<br>B9_6<br>B19_2<br>B38_4<br>B115_2 | Yes<br>Yes<br>Yes<br>Yes<br>Yes                    | Yes<br>Yes<br>Yes<br>Yes<br>Yes |
| hndShk  | HS_NONE<br>HS_HW<br>HS_SW                | Yes<br>No<br>No                                    | Yes<br>Yes<br>Yes               |

After initialization incoming characters are read in the receive buffer. They can be entered in the transmit buffer with **putCh** and **putStr**.

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

|        |  |
|--------|--|
| com    | Port number  |
| string | Pointer to the characters to be received                         |
| maxlen | Number of the characters to be read                              |
| err    | Pointer to int for error report (with empty receive buffer only) |

|  |  |
|--|--|
|  | Significance of bits:<br>[31] Break received (com 1 and 2) or noise error (com 0)<br>[30] Framing error<br>[29] Parity error<br>[28] Overrun error<br>[27] FIFO overflow |
|--|--|

Return:

Number of copied characters

**getStr** copies maxlen characters from the receive buffer to string. If there are less than maxlen characters in the receive buffer, these will be copied.

Error checking is as follows: If a hardware error is detected in the interrupt handler (framing, parity, etc.), incoming characters are discarded from this time. The same applies with an overflow of the receive buffer. With the first read from an empty buffer, the error is reported to the application layer, the error flag cleared and the hardware reset. After this, receives are again possible.

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

|     |  |
|-----|--|
| com | Port number  |
| err | Pointer to int for the transfer of bit-coded errors<br>(with empty receive buffer only)<br><br>Significance of the bits:<br>[31] Break received (com 1 and 2) or noise error (com 0)<br>[30] Framing error<br>[29] Parity error<br>[28] Overrun error<br>[27] FIFO overflow<br>[00] XOFF sent or RTS disabled with enabled handshake |

Returns:

Character read as 8-bit value

-1 as 32-bit value: no character present

For error checking see *getStr*.

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

|     |   |
|-----|---|
| com | Port number   |
| err | Pointer to error flag<br>Significance of the bits:<br>[31] Break received (com 1 and 2) or noise error (com 0)<br>[30] Framing error<br>[29] Parity error<br>[28] Overrun error<br>[27] FIFO overflow |

Return:

Number of valid characters in the receive buffer

*recState* returns the state of the receiver.

The errors are reported even if there are characters present in the buffer.

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

|        |  |
|--------|--|
| com    | Port number  |
| string | Pointer to the characters to be sent   |
| len    | Number of the characters to be sent  |
| err    | Returns the state of the receiving device with handshaking<br>0: Receiving device ready-to-receive<br>1: Receiving device not ready-to-receive |

Returns:

Number of characters received

0 ---> FIFO full

**putStr** copies maximally len characters to the transmit buffer.

If the transmit buffer gets filled during copying, communication is aborted. The return value contains the number of copied characters. To ensure that all characters can be entered, **traState** must be called in advance. With handshake being enabled err carries the information whether the receiving device is ready-to-receive.

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

|     |  |
|-----|--|
| com | Port number  |
| ch  | Character to be sent   |
| err | Returns the state of the receiving device with handshake<br>0: Receiving device ready-to-receive<br>1: Receiving device not ready-to-receive |

Returns:           0: Characters received  
                  -1: FIFO full

putCh enters a character in the transmit buffer. If the buffer is full it returns the value -1, otherwise zero. With handshake being enabled err carries the information whether the receiving device is ready-to-receive.

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

|     |   |
|-----|---|
| com | Port number   |
| err | Pointer to error flag<br>0: Receiving device is ready-to-receive or no handshaking<br>1: Receiving device is not ready-to-receive (XOFF received or CTS disabled) |

Returns:           Number of free characters in the transmit buffer

traState returns the state of the transmitter.

**int serState (int com)**

|     |             |
|-----|-------------|
| com | Port number |
|-----|-------------|

Returns:            [0]: transmit pending  
                      [1]: receive pending

serState returns the state of data flow on a serial port, which includes the port's hardware. Only when the last character of a transmit procedure has been completely transmitted, serState returns a zero in bit 0. If the receive buffer is empty and no character is pending a zero is returned in bit 1.

### 6.5.6. DPM Communication

For DPM communication the driver layer provides 2 data buffers for reads and writes. These buffers have a size of 896 bytes each. More useful data cannot be processed with a single read or write.

Communication via DPM is mostly user specific. The driver module dmp.c and the corresponding sources of the host PC demo application show one way to communicate and may be taken here as an example.

DPM communication may be as follows:

- polling mode without interrupts
- interrupt-driven mode

Both communications devices should be operated in the same mode.

#### 6.5.6.1. Initialization

To initialize DPM communication the functions ***dpmlnit***, ***dpmEnable*** and ***dpmGetState*** are supplied. The operating mode (interrupt-driven or polling) of the DPM is set with ***dpmlnit***, via ***dpmEnable*** the receiving device is signalled that communication has been initialized and communication is established. ***dpmGetState*** serves to determine the state of the receiving device.

**void dpmlnit ( void )**

**dpmInit** initializes the communication buffers, the driver-internal flags for DPM communication and installs the interrupt handler in interrupt-driven mode. Please see the source code `dpm.c` on the CD. These buffers and initializations are user specific and may be changed. In the module `dpm.c` a simple initialization is done to correspond with the demo application on the host PC.

#### **int dpmEnable (void)**

Return:

State of receiving device (see `dpmGetState`)

**dpmEnable** informs the driver layer of the receiving device that the DPM has been initialized and enabled.

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

Returns:

|         |   |
|---------|---|
| RET_OK  | Receiving device initialized, communication may start |
| NOT_RDY | Receiving device not ready                            |

### 6.5.6.2. Polling Mode

Two functions are provided to write data to or read data from the DPM:

**dpmWrite** writes `len` bytes to the data area for the 486.

The `accMode` byte in the allocated status area (see sources `dpm.c`) serves to control access. `accMode` controls whether an error is returned to the application or whether the application is waiting when the DPM is allocated by the receiving device. `wrMode` determines whether the data contained in a filled buffer will be overwritten or an error is returned to the application.

**dpmRead** reads `len` bytes from the data area host-PC->332 to address `dst`. `accMode` determines, whether an error is returned to the application or whether the application is waiting when the buffer is allocated.

#### **int dpmWrite (byte \*src, sDPM\_BUFSTAT \*dst, int len, int accMode, int wrMode)**

|         |  |   |
|---------|--|---|
| src     | Pointer to the address of the data to be sent            |   |
| dst     | Pointer to the buffer, where the data have to be written |   |
| len     | Number of data bytes                                     |   |
| accMode | Access control   |   |
|         | 0  | If DPM is allocated by receiving device, return error to application.               |
|         | 1  | If DPM is allocated by receiving device, wait.                                      |
| wrMode  | Access control   |   |
|         | 0  | If old data are not retrieved by the receiving device, return error to application. |
|         | 1  | Overwrite old data even if they are not retrieved by the receiving device.          |

Returns:

|    |  |
|----|--|
| 0  | Data were written  |
| -1 | Data area allocated by the receiving device and accMode is 0       |
| -2 | Old data were not received by the receiving device and wrMode is 0 |
| -3 | len too large; data do not fit in the data area                    |

**dpmWrite** checks whether the data fit in the DPM's data buffer 332->host-PC. It then allocates the buffer. If this is not possible and accMode is 0, an error is returned. If accMode is 1 it waits until the receiving device enables the buffer. It then checks whether the data were read by the receiving device. If this is not the case and wrMode is 0, dpmWrite returns with an error to the application. If wrMode is 1 or if the data were read by the receiving device, the data are copied to the buffer, the number of the copied bytes specified for the receiving device, the buffer for the read routine of the receiving device set to valid and the buffer deallocated.

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

|         |  |   |
|---------|--|---|
| src     | Pointer to the DPM buffer which data is to be read |   |
| Dst     | Pointer to target address                          |   |
| Len     | Maximum number of bytes to be copied               |   |
| AccMode | Access control                                     |   |
|         | 0  | If the DPM is allocated by the DPM, return error to application |
|         | 1  | If the DPM is allocated by the DPM, wait until it is enabled    |
| RdMode  | Access control                                     |   |
|         | 0  | If no new data are available, return error to application       |
|         | 1  | read data independently of updates through the receiving device |
| format  | 0  | No format conversion  |
|         | 1  | word swapping   |
|         | 2  | lword swapping  |

Returns:

Number of bytes read if no error occurred

-1: Data area is allocated by the receiving device and accMode = 0

**dpmRead** allocates the data area.

If this fails and accMode is 0, an error is returned to the application. If accMode is 1 it is waited until the receiving device enables the data area. **dpmRead** then verifies whether the data area contains new data. If this is not the case and rdMode is 0, an error is returned to the application. After this, the data are read from the buffer to address dst, swapping in accordance with format (not yet implemented). The number of bytes read results from the smaller number of len and the value entered by the receiving device for the length of the valid data.

### **6.5.6.3. Interrupt Mode**

not yet implemented.

## 6.6. System Configuration

To configure the system two source files are supplied: `start.s` (assembler) and `sysinit.c` (C). `start.s` allows you to implement your own initializations at assembler level. The label `start` is the entry and is being jumped to from within the bootstrap area, provided all checks were successful. At the end of `start.s` the application jumps to `sysinit.c`, where hardware-based initializations may be made in C. In addition, application-specific initializations may be entered. At the end of `sysinit.c` the user specific application (i.e. `main`) is started.

### 6.6.1. Exceptions and Interrupts

The interrupt table is created by `HiCoDrv` at 0 and initialized. All unused interrupt vectors will be initialized to a branch address, where into address `STAT332` in the DPM (offset `0x7e3`) the value `0xA0` and into address `ERR332` of the DPM (offset `0x7e6`) the vector number of the exception is written.

The only vector to be initialized to another address by calling `evt_init` is the vector for `TRAP #0`. It is internally used and not available to the application. All interrupt vectors used by the hardware are set only when the respective hardware function is initialized.

The CPU32 has 7 interrupt priorities, with 1 having lowest and 7 the highest priority. With `HiCO332` the interrupt inputs 3, 4, 5, and 6 are available. They can be allocated to the external interrupt source by use of jumpers. The internal interrupt sources can be allocated to the priorities 1 to 7. The application interface passes the desired interrupt priorities with the initialization function. The following table gives an overview of all interrupt sources and priorities that can be allocated.

| Source | ext/<br>int | Prio | De-<br>fault | Vector<br>No. | Offset<br>(hex) | Use                        |
|--------|-------------|------|--------------|---------------|-----------------|----------------------------|
| DPM    | ext         | 3..6 | 3            | AVEC          | 150             | PC com-<br>muni-<br>cation |
| DUART  | ext         | 3..6 | 4            | AVEC          | 14C             | ser1 and<br>ser2           |
| SCI    | int         | 1..7 | 5            | 82            | 148             | ser0<br>(RS485)            |

| Source | ext/<br>int | Prio | De-<br>fault | Vector<br>No. | Offset<br>(hex) | Use           |
|--------|-------------|------|--------------|---------------|-----------------|---------------|
| Timer  | int         | 1..7 | 6            | 80            | 140             | PIT<br>(VRTX) |
| TPU    | int         | 1..7 | 2            | 64 to 79      | 100..1<br>3C    | Encoder       |

The interrupt vectors belonging to these sources will be entered by the driver layer when the HiCO332 is initialized. The vector numbers are the default values. When the interrupts are initialized, the values set in `sysinit.c` are used for prioritization. The default settings are as follows:

```

prio_tpu      =      TPU_IRQ;          //IRQ 2 for the encoders
prio_dpm      =      DPM_IRQ;          //IRQ 3 for DPM communication
prio_duart    =      DUART_IRQ;        //IRQ 4 for the two DUART
//interfaces
prio_sci      =      SCI_IRQ;          //IRQ 5 for the SCI interface
prio_pit      =      PIT_IRQ;          //IRQ 6 for the Periodic
Interrupt
//Timer

```

If one of the values used is outside the valid range, the above default value will be used. The use of identical interrupt priorities for different sources is not supported by the hard- or firmware and causes erratic system behaviour of the board.

The interrupt priorities for

- DPM
- DUART

must be configured via jumpers **and** the firmware. In doing so note that the jumper setting must be in accordance with the settings made in `sysinit.c`.

### 6.6.2. Debouncing of the Binary Inputs

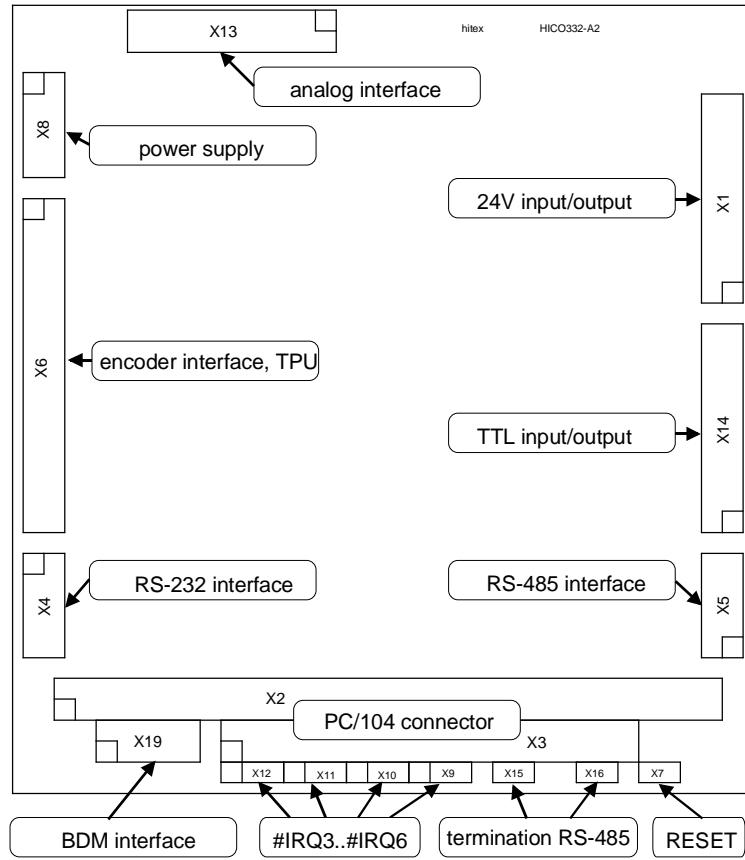
The debouncing of the binary inputs occurs in the timer's interrupts handler. One input pin must be stable for at least `binIn_tau` numbers of systicks to be accepted as unbounded. The default value for the `binIn_tau` variable in `sysinit.c` is 5.

### 6.6.3. Software Handshake

When software handshake is selected with the initialization of the serial ports, the receive interrupt handler transmits an XOFF to the receiving device as soon as the level of the receive buffer has reached xofflimit. Xofflimit is an int type and provides the information in %. If the level of the receive buffer is below xonlimit, an XON is transmitted to the receiving device. The two values can be entered in sysinit.c, if required by the application. Default is 80 % for xofflimit and 20 % for xonlimit.

## 7. Connector Pinouts

### 7.1. Locations of Connectors



## 7.2. PC/104, X2/X3

Type 64-pin + 40-pin socket terminal strip with long pins, 2.54 mm

|    | Row A   | Row B  | Row C | Row 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. Voltage Supply, X8

Type 10-pin post connector, 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. Serial Ports

Type 10-pin post connector, 2.54 mm

#### 7.4.1. RS-232 Ports, 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 Port, X5

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

## 7.5. Binary Input/Output

Type 20-pin post connector, 2.54 mm

### 7.5.1. TTL Input/Output, 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 Input/Output, X1**

| 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   | + 24V  | 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  | + 24V  | 20  | GND    |

## 7.6. Encoder Interface, TPU, X6

Type 32-pin post connector, 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. Analog Ports, X13

Type 20-pin post connector, 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

Type 10-pin post connector, 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. Technical Data

### 8.1. Mechanical Data

|                   |                                       |
|-------------------|---------------------------------------|
| <b>Weight</b>     | 80 g                                  |
| <b>PC-Board</b>   | Glass epoxy FR-4, UL listed, 6 layers |
| <b>Dimensions</b> | 96 mm x 90 mm x 22 mm                 |

### 8.2. Electrical Data

#### 8.2.1. Supply Voltage

|                            |                    |
|----------------------------|--------------------|
| <b>Supply Voltage</b>      | 5 V, +/-5%, +- 15V |
| <b>Current Consumption</b> | max 0.35 A         |

### 8.3. Ambient Conditions

|                      |  |
|----------------------|--|
| <b>Temperature</b>   | 0 to +70°C in operation<br>-40 to +85°C in storage |
| <b>Rel. Humidity</b> | 0 to 95 %, not condensating                        |











## 10. Supplemental Information on Components

- [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] Am29F400B, 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