SIEMENS

Technical Description
Order number: 6AR1943-4AA00-2CA0 **December 1999**

# SICOMP IMC01

**Compact Process Computer**

Order number: 6AR1025-1XXxx-0AA0

# IMC01-BSP, Version 1.0

**Basic software for the SICOMP IMC01 compact process computer**

**SICOMP Industrial Microcomputer**

(4)J31069-D2094-U001-A0-7618

## Product History of the Technical Description

| Revision [1)] | Record of Changes | Date |
|---|---|---|
| A0 | First edition | 12/99 |

*1) Corresponds to the 4th block of digits of the drawing number in the footer*

## Explanation of Notation

* An asterisk behind the signal name indicates a low-active signal (e.g., IOR*).

/ A slash between two signal names separates two level-dependent functions of one signal. Example: C/D* means Command for high level and Data for low level.

– Connections indicated with a dash in a connector assignment table are reserved (i.e., bus or I/O interface).

**Signal** Special signals not included in these specifications are indicated in bold print in the signal assignment tables and then explained (e.g., **AMSEOP**).

# Safety Notes for SICOMP Boards

## ESD protection measures

**Caution**

When handling boards and other components carrying this symbol, always adhere to ESD protection guidelines (**E**lectrostatic **S**ensitive **D**evices).

- Never touch the boards unless required work makes this absolutely necessary.
- When working with the boards, use a conductive and grounded work surface.
- Wear a grounding bracelet.
- Never touch the pins, connections or printed circuits of the boards.
- Never permit the boards or components to be touched by chargeable objects (e.g., synthetic materials).
- Keep the boards or components at least 10 cm away from CRT units and television sets.
- Leave the boards in their special packaging until they are needed. When registering boards, etc. do not remove the boards from their packaging or touch them.
- Boards may only be installed or removed when the voltage is off.

## Related SICOMP Literature

For more information on installing and handling SICOMP boards, see the SICOMP IMC system manual.

**Caution:**

Danger of injury (e.g., sharp edges of housing components) when performing service and maintenance work!

# Notes on the CE Seal

**EC guidelines EMC 89/336/EEC**

Products bearing the CE seal meet the requirements of EC guideline 89/336/EEC "electromagnetic compatibility" and the harmonized European standards (EN) listed below.

The EC declarations of conformity are retained for responsible authorities at the address below in accordance with article 10 of the above stated EC guideline.

Siemens Aktiengesellschaft
Bereich Automatisierungstechnik
A&D SE QTD QM
Postfach 2355
D-90713 Fuerth

**Area of use**

The products described are designed for use in industrial applications. They meet the following requirements.

| Area of Use | Requirements for | |
|---|---|---|
| | Interference Emission | Interference Immunity |
| Industrial | EN 50081-2: 1993 | EN 50082-2 : 1995 |

When an individual approval is granted, the products described can be used in Germany in residential areas (i.e., residential zones, trade and commerce zones and small business areas).

| Area of Use | Requirements for | |
|---|---|---|
| | Interference Emission | Interference Immunity |
| Residential | Individual approval | EN 50082-1: 1992 |

Individual approval is granted by the "Bundesamt für Post und Telekommunikation" and its branch offices.

**Installation guidelines**

The products fulfill the requirements under the following conditions.

1. The installation guidelines and the notes on safety in this manual have been adhered to.
2. In addition, the following rules have been complied with.

**Data/signal lines**

All signal lines which leave the device must be provided with shielding. The shields of the signal lines must be attached on both sides in shielded plug connectors or with the metallic HF clamps provided for this purpose.

**Other notes**

Since the device is designed for installation in cabinet panels, the interfaces on the back of the device are not provided with additional protection.

**Service work on switching cabinets**

To protect the boards from static electrical discharge, technicians must discharge themselves electro-statically before touching the back of the device or the components on the back.

**Updated technical data**

Contrary to the information contained in the "Technical Data" of the technical description, the following specifications on electromagnetic compatibility apply to boards/devices which carry the CE seal.

These specifications apply to devices which have been installed as described by the stated installation guidelines.

| **Specifications for Electromagnetic Compatibility** | **Test Values** |
|---|---|
| **Interference immunity against the discharge of static electricity** [1)] | |
| Tested in accordance with EN 61000-4-2 | Air discharge 8 kV<br>Contact discharge 4 kV |
| **Interference immunity against fast transient interference voltages (bursts)** | |
| Tested in accordance with EN 61000-4-4<br>Supply lines for 120/230 V AC<br>Signal lines (I/O and bus lines) | <br>2 kV<br>2 kV [2)] |
| **Interference immunity against electromagnetic fields** [1)] | |
| Tested in accordance with EN V 50140 (amplitude-modulated HF) | 80 to 1000 MHz<br>10 V/m<br>80% AM (1 kHz) |
| Tested in accordance with EN V 50140 (pulse-modulated HF) | 900 MHz<br>10 V/m<br>50% ED, 200 Hz repetition frequency |
| **Interference immunity against high frequency emission** | |
| Tested in accordance with EN V 50141 | 0.15 to 80 MHz<br>10 V<br>80% AM (1 kHz)<br>Source impedance 150 Ω |
| **Interference emission** [1)] | |
| Tested in accordance with EN 55011<br>Emission of electromagnetic fields<br>Interference emission over power lines | <br>Limit value class A, group 1<br>Limit value class A, group 1 |

*1) With closed metal housing on front of device*

*2) Signal lines which are not used for process control (e.g., cables to external printers: 1 kV)*

# Table of Contents


**1 Introduction 11**

1.1 Product Structure 12
1.1.1 Block Diagram of the Hardware Components (Basic Configuration) 12
1.1.2 Software Structure 13

1.2 System Prerequisites and Accessories 14

1.3 Required Commissioning Steps 15
1.3.1 STEP 5 Programming for Compact PLC 15
1.3.2 C or IMC0x-PLC Programming 15

1.4 Scope of Delivery 16

**2 Device Description 17**

2.1 Features 17
2.1.1 Physical Components 18
2.1.2 Power Supply 18
2.1.3 Processor and Memory Configuration 19
2.1.4 Interfaces 20

2.2 Interface Signals 21
2.2.1 Serial Interfaces 21
2.2.2 Interfaces of the I/O Card 23
2.2.3 Interfaces of the Expansion Card 25

2.3 Technical Data 27
2.3.1 Operational Values 27
2.3.2 Physical Values 28
2.3.3 Environmental Requirements 28

**3 Commissioning with STEP 5 Programming for Compact PLC 29**

3.1 Connecting the Power Supply and the PG to the IMC01 29

3.2 Installing the IMC01-BSP on the PG 30

3.3 Creating the Program with the PG 30

3.4 Setting the Type of Measurement for the Analog Inputs 30

3.5 Installation in a Front Panel 32

3.6 Connecting the I/O Devices 33
3.6.1 Installation Guidelines 33
3.6.2 Grounding Concept 34
3.6.3 Connection Cable from PG to IMC01 (Zero Modem Cable) 35

3.7 Parameterizing PROFIBUS 35

**4 Commissioning with C or IMC0x-PLC Programming 36**

4.1 Connecting the Power Supply and the Development Computer to the IMC01 36

4.2 Installing the IMC01-BSP on the Development Computer 37

4.3 Preparing the Program and Generating the System 37
4.3.1 Using the Sample Applications 39

| | | |
|---|---|---|
| 4.3.2 | Using the Coprocessor Emulation | 39 |
| 4.4 | Loading and Testing the RMOS System and the User Program | 40 |
| 4.4.1 | Behavior during Booting/Reset | 40 |
| 4.4.2 | Establishing the Connection | 41 |
| 4.4.3 | Deleting Software from the IMC01 | 41 |
| 4.4.4 | Loading an RMOS System or a User Program | 42 |
| 4.4.5 | Starting the System or User Program | 43 |
| 4.4.6 | Testing and Loading User Programs with the Debugger | 43 |
| 4.5 | Setting the Type of Measurement for the Analog Inputs | 44 |
| 4.6 | Installation in a Front Panel | 45 |
| 4.7 | Connecting the I/O Devices | 46 |
| 4.7.1 | Installation Guidelines | 46 |
| 4.7.2 | Grounding Concept | 47 |
| 4.7.3 | Connection Cable from PC to IMC01 (Zero Modem Cable) | 48 |
| 4.8 | Parameterizing PROFIBUS | 48 |
| **5** | **Compact PLC** | **49** |
| 5.1 | Reloading the Compact PLC to User FLASH Memory | 49 |
| 5.2 | Data Block Interfaces for the Peripherals | 50 |
| 5.2.1 | Character Output on the Display (FB_DIS) | 50 |
| 5.2.2 | Using the HW Peripherals (FB_IN, FB_OUT) | 52 |
| 5.2.3 | Configuration in DB_CONFIG | 59 |
| **6** | **BIOS Functions** | **61** |
| 6.1 | Call Back Functions | 62 |
| 6.2 | Functions for Analog Input | 63 |
| 6.2.1 | bai_get | 63 |
| 6.2.2 | bai_getmode | 65 |
| 6.2.3 | bai_init | 66 |
| 6.2.4 | bai_read | 67 |
| 6.2.5 | bai_setmode | 69 |
| 6.2.6 | bai_strtconv | 70 |
| 6.3 | Functions for Analog Output | 72 |
| 6.3.1 | bao_init | 72 |
| 6.3.2 | bao_set | 73 |
| 6.3.3 | bao_setmode | 74 |
| 6.4 | Real Time Clock Functions | 75 |
| 6.4.1 | bc_init_rtc | 75 |
| 6.4.2 | bc_stime | 76 |
| 6.4.3 | bc_time | 77 |
| 6.5 | Functions for Digital Input | 79 |
| 6.5.1 | bdi_get | 80 |
| 6.5.2 | bdi_getblock | 81 |
| 6.5.3 | bdi_getmbit | 82 |
| 6.5.4 | bdi_getmirror | 83 |
| 6.5.5 | bdi_init | 84 |
| 6.5.6 | bdi_mirror | 84 |
| 6.5.7 | bdi_resetmbit | 85 |

| | | |
|---|---|---|
| 6.5.8 | bdi_setmbit | 85 |
| 6.5.9 | bdi_setmirror | 86 |
| 6.6 | Functions for Digital Output | 87 |
| 6.6.1 | bdo_flush | 88 |
| 6.6.2 | bdo_getmbit | 89 |
| 6.6.3 | bdo_getmirror | 90 |
| 6.6.4 | bdo_init | 91 |
| 6.6.5 | bdo_mflush | 91 |
| 6.6.6 | bdo_reset | 92 |
| 6.6.7 | bdo_resetmbit | 92 |
| 6.6.8 | bdo_set | 93 |
| 6.6.9 | bdo_setmbit | 93 |
| 6.6.10 | bdo_setmirror | 94 |
| 6.6.11 | bdo_status | 95 |
| 6.7 | Display Functions | 96 |
| 6.7.1 | bg_cleardevice | 96 |
| 6.7.2 | bg_getmaxx | 97 |
| 6.7.3 | bg_getmaxy | 97 |
| 6.7.4 | bg_gotoxy | 98 |
| 6.7.5 | bg_init | 99 |
| 6.7.6 | bg_outchar | 99 |
| 6.7.7 | bg_outtext | 100 |
| 6.7.8 | bg_screensaver | 101 |
| 6.7.9 | bg_setcursor | 102 |
| 6.7.10 | bg_setcursortype | 103 |
| 6.7.11 | bg_wherex | 104 |
| 6.7.12 | bg_wherey | 104 |
| 6.8 | General Functions | 105 |
| 6.8.1 | bgen_battstat | 106 |
| 6.8.2 | bgen_crc | 107 |
| 6.8.3 | bgen_init | 107 |
| 6.8.4 | bgen_version | 108 |
| 6.9 | Keyboard Functions | 109 |
| 6.9.1 | bk_getchar | 109 |
| 6.9.2 | bk_keybinit | 110 |
| 6.9.3 | bk_LEDflush | 111 |
| 6.9.4 | bk_setkeytab | 111 |
| 6.10 | Functions for Temperature Measurement | 112 |
| 6.10.1 | bpt_calibrate | 112 |
| 6.10.2 | bpt_gettemp | 113 |
| 6.10.3 | bpt_init | 114 |
| 6.10.4 | bpt_sel_chnum | 114 |
| 6.11 | Functions for Watchdog Processing | 115 |
| 6.11.1 | bwd_init | 115 |
| 6.11.2 | bwd_trigger | 116 |
| 6.12 | Functions for DP Slave Programming | 117 |
| 6.12.1 | dp_GetErrorState | 117 |
| 6.12.2 | dp_GetShortDPVersion | 118 |
| 6.12.3 | dp_GetStationAddress | 118 |
| 6.12.4 | dp_Init | 119 |
| 6.12.5 | dp_ReadOutputBuffer | 120 |
| 6.12.6 | dp_WriteInputBuffer | 121 |

6.12.7 Structure of DP_ERROR_STATE 122

6.13 Functions of the Hardware Configuration 123
6.13.1 imc01ReadHwStatus 124

**7 Appendix 125**

7.1 Key Codes 125

7.2 Error Messages during the Loading Procedure 126

**8 Abbreviations and Terms 127**

# 1 Introduction

## Application areas of the device

The SICOMP-IMC01 is a compact computer recommended for the following applications when RMOS is used.

- Process monitoring and operator control on the floor
- Control via digital and analog inputs and outputs

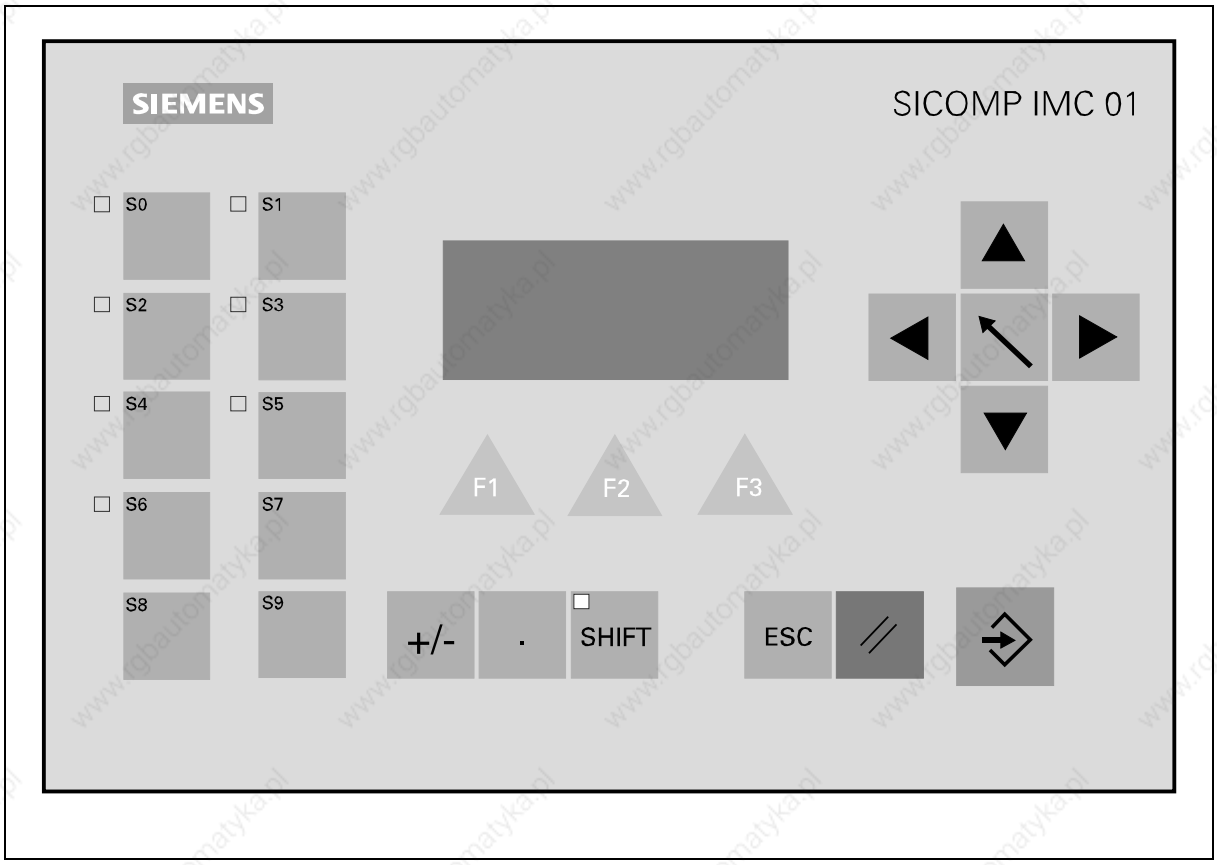



Figure 1.1 Front view of the IMC01 with display and operator control elements

# 1.1 Product Structure

## 1.1.1 Block Diagram of the Hardware Components (Basic Configuration)

The following block diagram shows a summary of the layout of the IMC01.

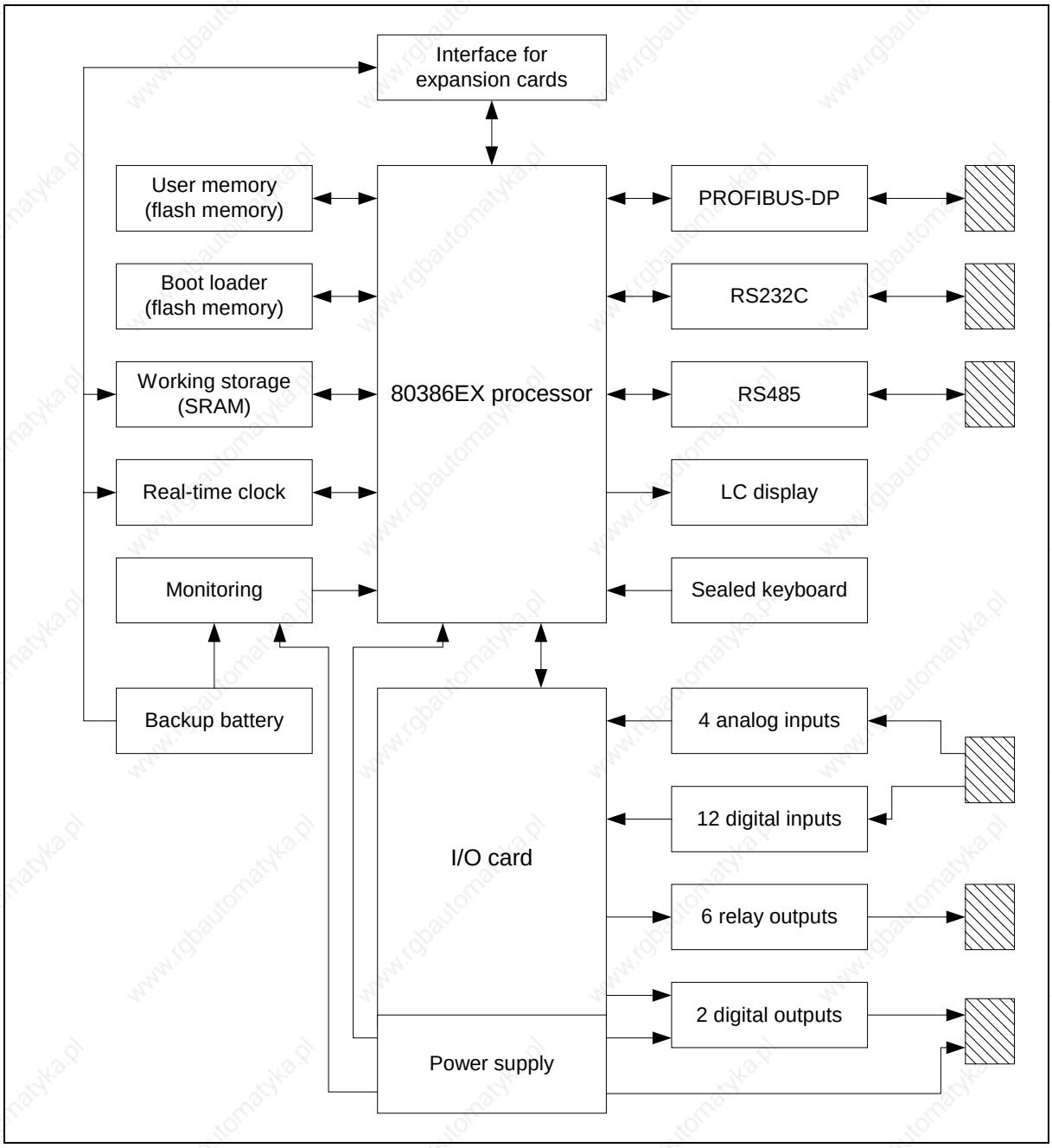



Figure 1.2 Block diagram of the hardware components (basic configuration)

### 1.1.2 Software Structure

The following figures show a summary of the software structure of the IMC01 and the developer's computer for Compact PLC and C or IMC0x-PLC programming.

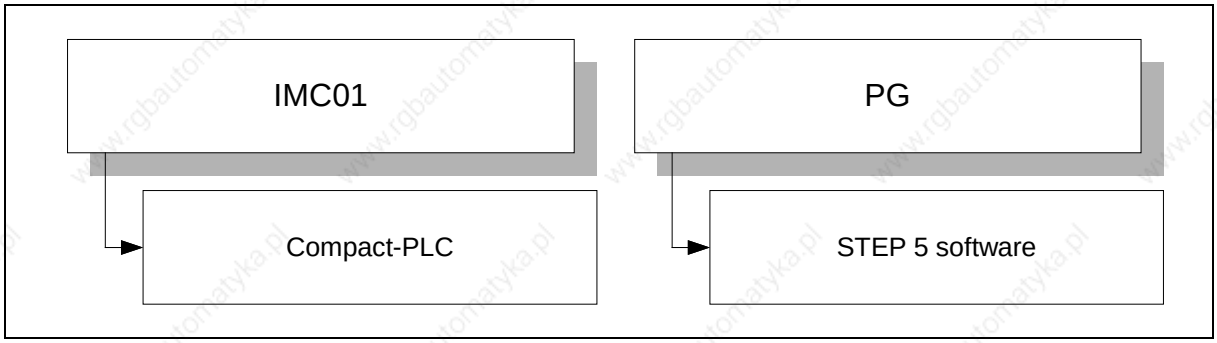



Figure 1.3 Structure of the software components for Compact PLC on delivery

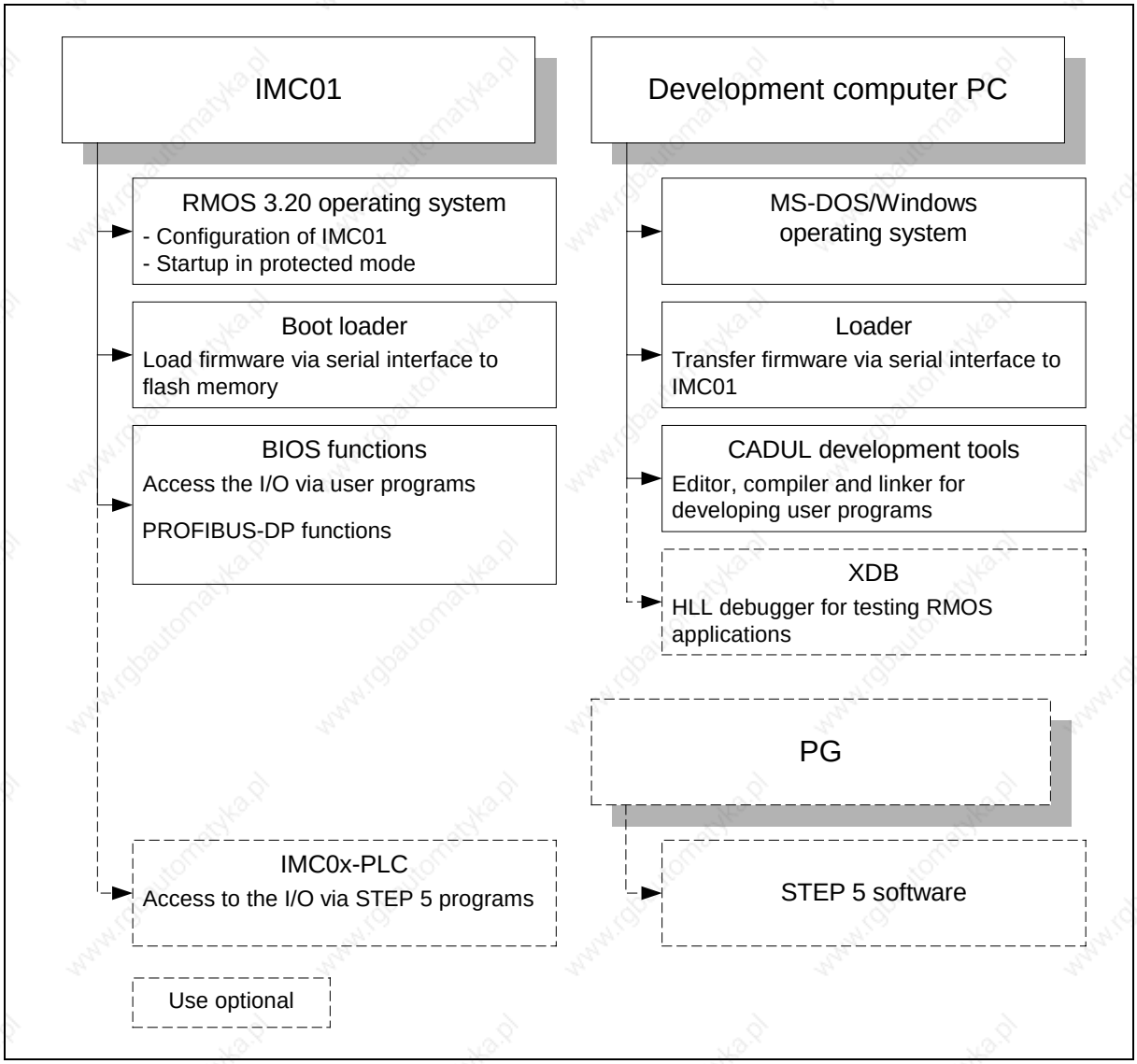



Figure 1.4 Structure of the software components for C or IMC0x-PLC programming

## 1.2 System Prerequisites and Accessories

| **Required Operational Accessories** | **Recommended by Siemens** |
|---|---|
| Power supply with 24 V DC,<br>at least 2 A continuous current | SITOP Power Standard [2)]<br>basic E24/2 (2 A) 6EP1331-1SL11<br>basic E24/5 (5 A) 6EP1333-1SL11<br>basic E24/10 (10 A) 6EP1334-1SL11 |
| **Required Developmental Accessories For Compact PLC** | |
| PG with STEP 5 software | STEP 5, version 6.65 or starting with version 7.12 |
| **Required Developmental Accessories for C/IMC0x-PLC programming** | |
| IBM-AT-compatible computer with 3½" floppy disk drive<br>Installed on this:<br>- RMOS3 V3.20 [1)]<br>- Compiler: CADUL CC386, starting with V700 O<br>- Linker: CADUL LINK386, starting with V207<br>- Assembler: CADUL AS386, starting with V300 I<br>- Debugger: CADUL XDB386, starting with V401 A (not mandatory) | |
| Zero modem cable (shielded) for transmission from development computer to the IMC01 | See chapter 3.6.3 or 4.7.3. |
| **Other Accessories** | **Order No.** |
| IMC0x-PLC (including English documentation)<br>STEP 5 software | 6AR1403-3AD00-2AA0 |

*1) Creating a system using this description requires a basic knowledge of RMOS. Training courses are offered by Siemens. Contact your local Siemens office for details.*

*2) See KT 10 catalog (combination technology).*

## 1.3 Required Commissioning Steps

Before commissioning the IMC01, perform the following steps.

### 1.3.1 STEP 5 Programming for Compact PLC

1. Connect the power supply and the PG to the IMC01.
2. Install the IMC01-BSP on the PG.
3. Create the program with the PG.
4. Set the type of measurement for the analog inputs.
5. Install in the front panel.
6. Connect the I/O devices.
7. Parameterize PROFIBUS.

These steps are described in detail in chapter 3.

**Note:**
After repairs, the device is returned in its original status on delivery (i.e., steps 3 to 7 must be repeated).

### 1.3.2 C or IMC0x-PLC Programming

1. Connect the power supply and the development computer to the IMC01.
2. Install the IMC01-BSP on the development computer.
3. Prepare program and generate system.
4. Load and test the RMOS system and the user program.
5. Set the type of measurement for the analog inputs.
6. Install in the front panel.
7. Connect the I/O devices.
8. Parameterize PROFIBUS.

These steps are described in detail in chapter 4.

**Note:**
After repairs, the device is returned in its original status on delivery (i.e., steps 4 to 8 must be repeated).

## 1.4 Scope of Delivery

### Device

- IMC01 basic model or with expansion card
- 2 or 4 plug connectors, inserted with I/O card or expansion card
- Assembly kit
  – 6 M4 nuts with washers and spring rings for mounting the device
  – 1 M5 nut with washer, spring ring and contact disk for connection of ground

### Documentation

- Technical Description
- Product floppy disk IMC01-BSP
- Software product slip
- Customer Support Service Fax (Software Registration Fax)
- IMC0x-PLC reference manual

# 2 Device Description

## 2.1 Features

The IMC01 consists of the following components.

- LC display, 16 x 4 characters
- Sealed keyboard with 24 keys, 8 of which are equipped with an LED
- Front plate with sealing frame for protection class IP65
- Metal housing with ventilation openings
- Computer unit with 80386EX processor
- FLASH memories with BIOS and user memory, can be programmed on-board
- SRAM work memory
- Serial interfaces for loading an RMOS system from the development computer and for use as desired
- Number of digital and analog inputs/outputs depends on the model.
- DP slave bus connection
- Extensive BIOS functions for input/output operations via keyboard, display, digital and analog inputs/outputs, and much more

These components will now be described in detail.

### 2.1.1 Physical Components

**Front plate**

The IMC01 is designed for installation in switching cabinets or consoles. Sealing on all sides provides a protection rating of IP65 on the front.

The accessories required for installation are included with the device.

**LC display**

The alphanumeric LC display has LED backlighting and can represent four lines with 16 characters each.

**Sealed keyboard**

Each key activation (i.e., pressing or releasing) triggers a call-back function.

Two labeling strips can be inserted in openings on the bottom of the device to label keys S0 to S9 (e.g., the tens block).

**Metal housing**

The housing of the IMC01 is made of refined stainless steel.
Ventilation openings on the top and bottom permit operation without a fan within a certain specified temperature range. See chapter 2.3.3.

All connection elements can be accessed on the bottom of the device.

### 2.1.2 Power Supply

The device is powered by a single 24 Volt direct current power supply. All required voltages (e.g., 5 V for the processor) are generated internally from this voltage.

### 2.1.3 Processor and Memory Configuration

**Processor** 80386EX from INTEL, clock frequency currently 24.576 MHz, integrated controller functions

**Watchdog** Can be enabled and disabled via software. Hardware reset after approx. 250 msec when the trigger command is not issued.

**Working memory** 1-Mbyte SRAM (2 x 512 Kbytes)

Saved data are protected against power failure by gold capacitor and extra backup battery.

**Program memory** 2-Mbyte FLASH memory divided into 128-Kbyte sectors

The first sector contains the boot loading program and is not available to the user. All other sectors can be cleared on-board and rewritten.

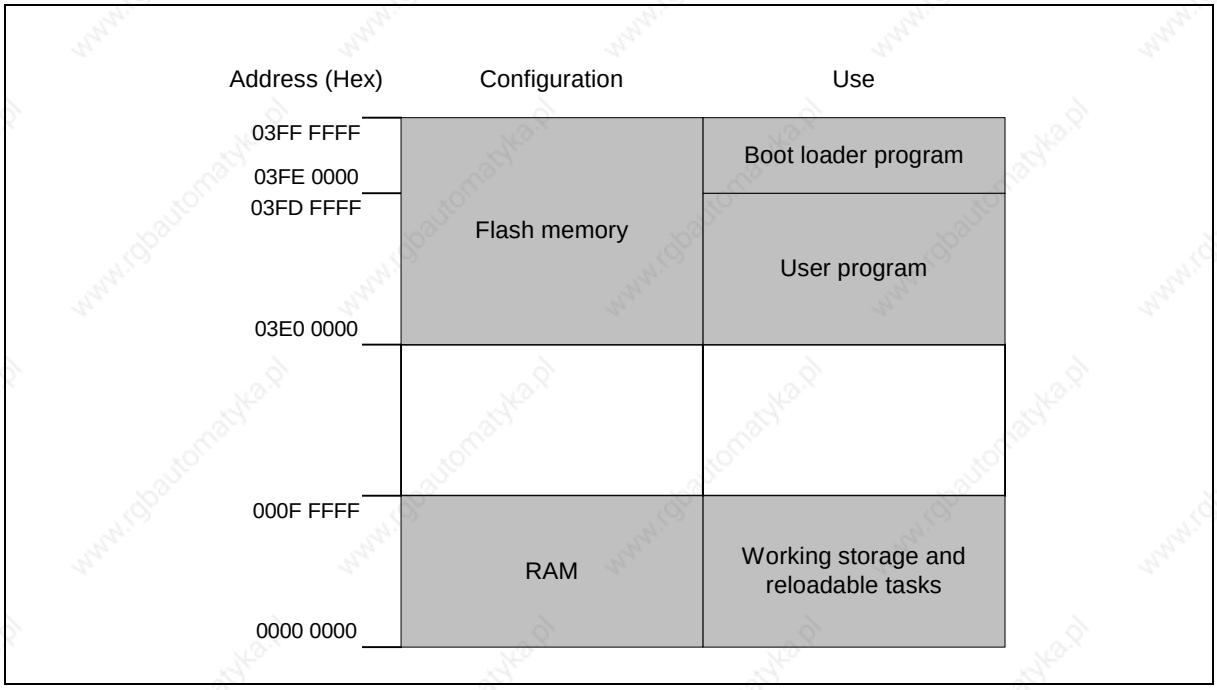



Figure 2.1 Organization of memory

### 2.1.4 Interfaces

The IMC01 is equipped with the following interfaces. IMC01-BSP contains BIOS functions for processing the interfaces. See chapter 6.

## Basic model

| | |
|---|---|
| **Digital inputs and outputs** | 12 digital inputs (24 V, non-floating) |
| | 2 digital outputs (24 V, P-switching, non-floating) |
| | 6 relay outputs, floating |
| **Analog inputs/outputs** | 2 analog inputs (can be set individually with plug-in jumpers for measuring voltage or temperature |
| | 2 analog inputs (can be set individually with plug-in jumpers for measuring voltage or temperature (PT100) |
| **Serial interfaces** | 1 RS 232 (COM1), maximum of 38,400 baud |
| | 1 RS 485 (COM2), semi-duplex, maximum of 38,400 baud |
| **PROFIBUS-DP interface** | PROFIBUS interface with SPC3 ASIC, DP slave function, maximum of 1.5 Mbaud, floating |

## Interfaces on the expansion card

The optional expansion card provides the following additional interfaces.

| | |
|---|---|
| **Digital inputs and outputs** | 12 digital inputs (24 V, non-floating) |
| | 2 digital outputs (24 V, P-switching, non-floating) |
| | 6 relay outputs, floating |
| **Analog inputs/outputs** | 2 analog inputs, can be set individually with plug-in jumpers for measuring voltage or temperature |
| | 2 analog inputs, can be set individually with plug-in jumpers for measuring voltage or temperature (PT100) |
| | 1 analog output (floating) |

# 2.2 Interface Signals

## 2.2.1 Serial Interfaces

### Serial interface COM1 (RS 232): 1X7

**9-pin sub D plug connector**

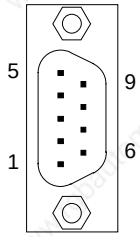



Table 2.1 Allocation of the COM1 serial interface (RS 232)

| Connection | Signal | Meaning |
|---|---|---|
| 1 | – | |
| 2 | RxD | Receiving data |
| 3 | TxD | Sending data |
| 4 | Reserved | Used internally. Do not connect. |
| 5 | GND | Signal ground |
| 6 | – | |
| 7 | RTS | Request to send |
| 8 | CTS | Ready to send |
| 9 | – | |

*– = Not used*

### COM2 (RS 485) serial interface: 1X8

**9-pin sub D socket**

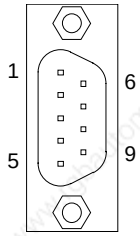



Table 2.2 Allocation of the COM2 (RS 485) serial interface

| Connection | Signal | Meaning |
|---|---|---|
| 1 | RA | Terminating resistors for signal line A |
| 2 | RS485_B | Signal line B of the RS 485 differential signal |
| 3 | RS485_B | Signal line B of the RS 485 differential signal |
| 4 | Reserved | Used internally. Do not connect. |
| 5 | RTS | Request to send (TTL signal) |
| 6 | GND | Signal ground |
| 7 | RB | Terminating resistors for signal line B |
| 8 | RS485_A | Signal line A of the RS 485 differential signal |
| 9 | RS485_A | Signal line A of the RS 485 differential signal |

*– = Not used*

**Note:**
The terminating resistors can be activated by connecting pin 1 with pin 8/9 or pin 7 with pin 2/3.

## PROFIBUS-DP interface: 1X9

**9-pin sub D socket**

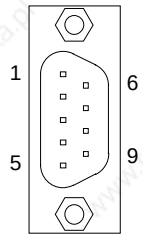



Table 2.3 Allocation of the PROFIBUS-DP interface

| Connection | Signal | Meaning |
|---|---|---|
| 1 | – | |
| 2 | – | |
| 3 | RS485_B | Signal line B of the RS 485 differential signal |
| 4 | RTS | Request to send (TTL signal) |
| 5 | GND1 | Signal ground [1)] |
| 6 | VCC1 | +5 V voltage supply [1)] |
| 7 | – | |
| 8 | RS485_A | Signal line A of the RS 485 differential signal |
| 9 | – | |

*– = Not used*

*1) Only for bus terminating resistors*

## 2.2.2 Interfaces of the I/O Card

### Digital and analog inputs: 1X1

**25-pin sub D socket**

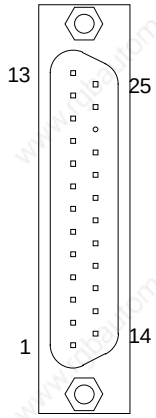



Table 2.4 Allocation of 1X1

| Connection | Signal | Meaning |
|---|---|---|
| 1 | I0.0 | Digital input 1 |
| 2 | I0.1 | Digital input 2 |
| 3 | I0.2 | Digital input 3 |
| 4 | I0.3 | Digital input 4 |
| 5 | I0.4 | Digital input 5 |
| 6 | I0.5 | Digital input 6 |
| 7 | M24 | Reference potential for digital inputs |
| 8 | AGND1 | Analog ground of the inputs |
| 9 | AGND1 | Analog ground of the inputs |
| 10 | AGND1 | Analog ground of the inputs |
| 11 | AGND1 | Analog ground of the inputs |
| 12 | AGND1 | Analog ground of the inputs |
| 13 | – | |
| 14 | I0.6 | Digital input 7 |
| 15 | I0.7 | Digital input 8 |
| 16 | I1.0 | Digital input 9 |
| 17 | I1.1 | Digital input 10 |
| 18 | I1.2 | Digital input 11 |
| 19 | I1.3 | Digital input 12 |
| 20 | P18 | Sensor voltage, 18 V |
| 21 | AI 1 | Analog input 1 |
| 22 | AI 2 | Analog input 2 |
| 23 | AI 3 | Analog input 3 |
| 24 | AI 4 | Analog input 4 |
| 25 | – | |

– = *Not used*

## Relay outputs: 1X2

**12-pin Combicon plug connector**

Table 2.5 Allocation of 1X2

| Connection | Signal | Meaning |
|---|---|---|
| 1 | O0.0 | Make-contact, digital output 1 |
| 2 | O0.0 | Make-contact, digital output 1 |
| 3 | O0.1 | Make-contact, digital output 2 |
| 4 | O0.1 | Make-contact, digital output 2 |
| 5 | O0.2 | Make-contact, digital output 3 |
| 6 | O0.2 | Make-contact, digital output 3 |
| 7 | O0.3 | Make-contact, digital output 4 |
| 8 | O0.3 | Make-contact, digital output 4 |
| 9 | O0.4 | Make-contact, digital output 5 |
| 10 | O0.4 | Make-contact, digital output 5 |
| 11 | O0.5 | Make-contact, digital output 6 |
| 12 | O0.5 | Make-contact, digital output 6 |

## Power supply and digital outputs: 1X3

**4-pin Combicon plug connector**

Table 2.6 Allocation of 1X3

| Connection | Signal | Meaning |
|---|---|---|
| 1 | P24 | Supply voltage |
| 2 | O0.6 | Digital output 7 |
| 3 | O0.7 | Digital output 8 |
| 4 | M24 | Reference potential |

## 2.2.3 Interfaces of the Expansion Card

### Digital inputs, analog inputs and outputs: 1X4

**25-pin sub D socket**

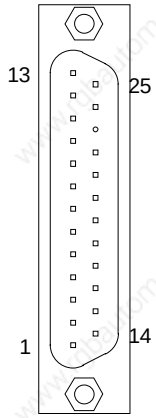



Table 2.7 Allocation of 1X4

| Connection | Signal | Meaning |
|---|---|---|
| 1 | I2.0 | Digital input 13 |
| 2 | I2.1 | Digital input 14 |
| 3 | I2.2 | Digital input 15 |
| 4 | I2.3 | Digital input 16 |
| 5 | I2.4 | Digital input 17 |
| 6 | I2.5 | Digital input 18 |
| 7 | M24 | Reference potential for digital inputs |
| 8 | AGND1 | Analog ground of the inputs |
| 9 | AGND1 | Analog ground of the inputs |
| 10 | AGND1 | Analog ground of the inputs |
| 11 | AGND1 | Analog ground of the inputs |
| 12 | AGND1 | Analog ground of the inputs |
| 13 | AO1 | Analog output 1 |
| 14 | I2.6 | Digital input 19 |
| 15 | I2.7 | Digital input 20 |
| 16 | I3.0 | Digital input 21 |
| 17 | I3.1 | Digital input 22 |
| 18 | I3.2 | Digital input 23 |
| 19 | I3.3 | Digital input 24 |
| 20 | P18 | Sensor voltage, 18 V |
| 21 | AI 1 | Analog input 5 |
| 22 | AI 2 | Analog input 6 |
| 23 | AI 3 | Analog input 7 |
| 24 | AI 4 | Analog input 8 |
| 25 | AGND2 | Analog ground of the outputs |

## Relay outputs: 1X5

**12-pin Combicon plug connector**

Table 2.8 Allocation of 1X5

| Connection | Signal | Meaning |
|---|---|---|
| 1 | O1.0 | Make-contact, digital output 9 |
| 2 | O1.0 | Make-contact, digital output 9 |
| 3 | O1.1 | Make-contact, digital output 10 |
| 4 | O1.1 | Make-contact, digital output 10 |
| 5 | O1.2 | Make-contact, digital output 11 |
| 6 | O1.2 | Make-contact, digital output 11 |
| 7 | O1.3 | Make-contact, digital output 12 |
| 8 | O1.3 | Make-contact, digital output 12 |
| 9 | O1.4 | Make-contact, digital output 13 |
| 10 | O1.4 | Make-contact, digital output 13 |
| 11 | O1.5 | Make-contact, digital output 14 |
| 12 | O1.5 | Make-contact, digital output 14 |

## Power supply and digital outputs: 1X6

**4-pin Combicon plug connector**

Table 2.9 Allocation of 1X6

| Connection | Signal | Meaning |
|---|---|---|
| 1 | P24 | Supply voltage |
| 2 | O1.6 | Digital output 15 |
| 3 | O1.7 | Digital output 16 |
| 4 | M24 | Reference potential |

# 2.3 Technical Data

## 2.3.1 Operational Values

| Power Supply | Typical | Permissible/Maximum |
|---|---|---|
| Supply voltage (P24) | 24 V DC | 20.4 to 28.8 V |
| Supply voltage fluctuation | 18.5 to 30.2 V | In acc. w. DIN 19240 |
| Current consumption (with outputs)<br>Without expansion card<br>With expansion card | <br>1.3 A<br>2.4 A | |
| Current consumption (without outputs)<br>Without expansion card<br>With expansion card | <br>0.3 A<br>0.4 A | |
| External fuses (must be provided by you) | | Rapid-action, max. of 2.5 A per feeder line |
| Voltage drop at 24 V / 0.9 A (without external buffering) | | 1 msec [1)] |
| Data retention with uninterrupted buffering | 3 years | |

*1) When this time is exceeded, an NMI is triggered followed by a RESET 0.5 msec later.*

| Digital Inputs and Outputs | | Typical | Permissible/Maximum |
|---|---|---|---|
| Input voltage | H signal<br>L signal | 24 V<br>0 V | 15 to 30 V<br>–3 to +5 V |
| Input current | H signal<br>L signal | 1.5 mA<br>0 mA | 15 mA<br>15 mA |
| Filter time constant | | 1 msec | |
| Output voltage | H<br>L | <br>Depends on load<br>(open output) | Min. of P24 – 3 V |
| Output current | H<br>L | | 0.5 A<br>0.4 mA |

| Relay Outputs | | Typical | Permissible/Maximum |
|---|---|---|---|
| Type of contact | | Make-contact | |
| Switching voltage | | | 250 V AC |
| Switching current | | | 8 A |
| Switching capacity | DC<br>AC | | 270 W<br>2000 W |
| Contact resistance (initial value) | | ≤ 100 mΩ at 24 V/1 A | |
| Test voltage, contact/winding | | | 4000 V AC |
| Switching frequency | | | 0.1 Hz |
| Life span at max. switching capacity | | $10^5$ switching cycles | 2 x $10^7$ switching cycles |
| Fuses | | Must be provided by you | |
| Protection against capacitive or inductive loads | | Must be provided by you | |

| Analog Inputs | Typical | Permissible/Maximum |
|---|---|---|
| Voltage measuring ranges | 0 to 5 V, 0 to 10 V, ±5 V, ±10 V | |
| Current measuring range | 0 to 20 mA | |
| Temperature measuring range | –200° C to +230° C (PT100) | |
| Impedance during voltage measurement | | Min. of 5 kΩ |
| Load during current measurement | 200 Ω | |
| Resolution | 12 bits | Min. of 10 bits |
| Conversion time | 200 µsec | |
| Channel switchover<br>During temperature measurement<br>Other | <br>3 msec<br>200 µsec | |

| Analog Output (on Expansion Card) | Typical | Permissible/Maximum |
|---|---|---|
| Output range | 0 to 20 mA | |
| Load | | 300 Ω |
| Resolution | 10 bits | |
| Conversion time | 2.5 msec | |
| Potential isolation | Only functional isolation due to EMC<br>No safety-relevant isolation | |

## 2.3.2 Physical Values

| Dimensions and Weight | Typical | Permissible/Maximum |
|---|---|---|
| Front plate without seal<br>(width x height x thickness) | 200 x 133 x 6 mm | |
| Mounting depth with front plate | 118 mm | 120 mm |
| Weight<br>Basic model<br>With expansion card | <br>1.35 kg<br>1.50 kg | <br>1.50 kg<br>1.65 kg |

## 2.3.3 Environmental Requirements

| | Operation | Transportation and Storage |
|---|---|---|
| Ambient temperature | –20° to 60° C [1)] | –40° to 70° C |
| Relative humidity at 25° C | 5 to 80% | 5 to 95% |
| Permissible operating altitude | 2500 m above sea level | No restrictions |
| Permissible temperature changes | 10° C/30 min. (without condensation) or 0.5° C/min. | |
| Vibration<br>In acc. w. IEC 68-2-6,<br>10 cycles on 3 axes | 10 Hz to 58 Hz: 0.075 mm amplitude<br>58 Hz to 500 Hz: 1 g | 5 Hz to 9 Hz: 3.5 mm amplitude<br>9 Hz to 500 Hz: 1 g |
| Shock | IEC 68-2-29 Test E/A<br>3 times per axis<br>2 directions per axis<br>Total of 18 shocks<br>15 g/11 msec | |

*1) These specifications are valid for free convection and vertical mounting position.*

# 3 Commissioning with STEP 5 Programming for Compact PLC

To commission the IMC01, proceed exactly as described below.

**Caution:**

Commissioning, installation and wiring may only be performed by qualified personnel who are familiar with and observe the general rules of technology and the applicable regulations and standards.

Before connecting the I/O devices and starting your user application, make sure that persons or systems will not be endangered by execution of the program.

1. Connect the power supply and the PG to the IMC01.
2. Install the IMC01-BSP on the PG.
3. Create the program with the PG.
4. Set the type of measurement for the analog inputs.
5. Install in the front panel.
6. Connect the I/O devices.
7. Parameterize PROFIBUS.

## 3.1 Connecting the Power Supply and the PG to the IMC01

The device is powered by the 4-pin pin strips on the I/O card (1X3, see chapter 2.2.2) and, if present, on the expansion card (1X6, see chapter 2.2.3).

**Caution:**

Suitable fuses (see chapter 2.3.1) must be installed in the feeder lines of the supply voltage, and, if necessary, an on/off switch.

Before turning on the supply voltage, make sure that all other connections are wired correctly.

Do not plug or unplug connection cables while the power is on. To do so may damage the device.

The PG is connected to the IMC01 with a zero modem cable on COM1. See chapter 3.6.3.

**Note:**
Remember that COM1 of the IMC01 is an RS 232 interface and not a 20 mA interface.

## 3.2 Installing the IMC01-BSP on the PG

If you are using only the Compact PLC, the IMC01-BSP can be installed without first installing RMOS V3.20. Cf. chapter 4.2.

### Installation

1. Turn on the PG, and start MS-DOS.
2. Place the product floppy disk in the floppy disk drive.
3. Set up a new directory (e.g., C:\IMC01).
4. Copy the PRODUCT.EXE file to the new directory.
5. Unpack the files with the command:

   ```
   product -d
   ```

## 3.3 Creating the Program with the PG

The STEP 5 software developed with the PG is stored on the buffered RAM where it is retained even after the device is turned off.

Before the user program can be stored in flash memory, it must be converted to binary format with the CVSTEPV.EXE program. Use the GEN_BIN.BAT batch file in directory SYSIMC1\COMPACT.PLC\STEP5 for this.

The user program can now be transferred to flash memory.

To do this, perform the following steps.

1. Switch the IMC01 off.
2. Start the LD_MC5.BAT batch file in directory SYSIMC1\COMPACT.PLC\LOADER.
3. Turn on the IMC01 again.

Use the LD_SYS.BAT batch file in directory SYSIMC1\COMPACT.PLC\LOADER to restore the status on delivery again.

## 3.4 Setting the Type of Measurement for the Analog Inputs

Plug-in jumpers are available on the I/O card and, if present, on the expansion card to set the type of measurement for the analog inputs. These jumpers are accessible from the top on the opened device. The figure below shows a view of the top of the opened device with the plug-in jumpers as preset at the factory.

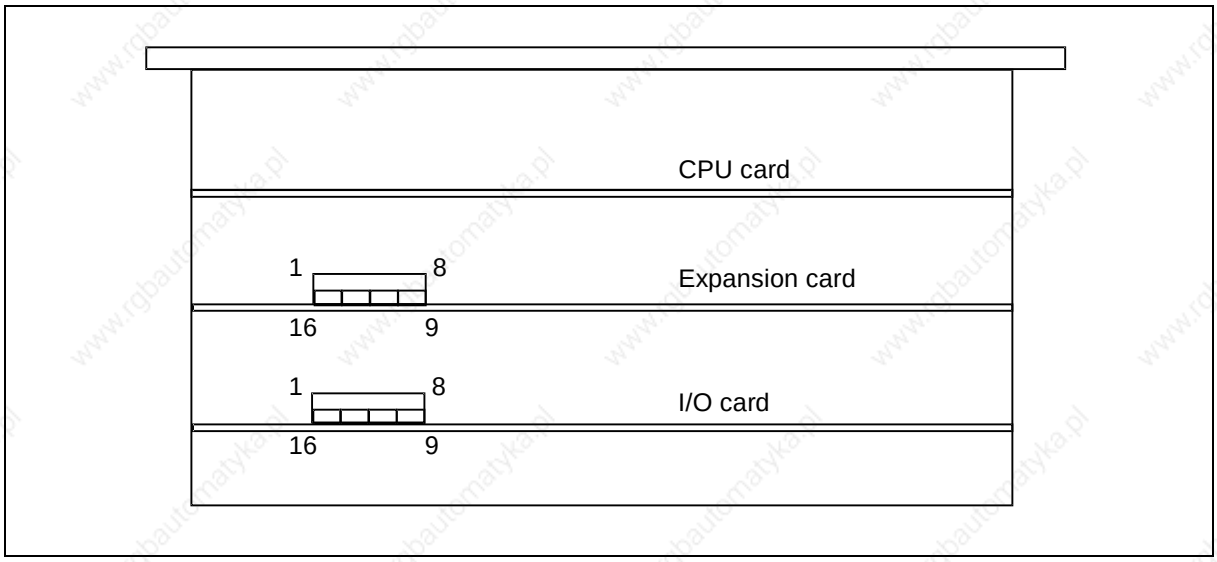



Figure 3.1 Location and presettings of the plug-in jumpers (view of the top)

Table 3.1 Settings on the I/O card

| Jumper Position | Default Setting | |
|---|---|---|
| Analog input 1 | Current measurement (pins 9-10) | Voltage measurement (pins 1-2) |
| Analog input 2 | Current measurement (pins 11-12) | Voltage measurement (pins 3-4) |
| Analog input 3 | Temperature measurement (pins 13-14) | Voltage measurement (pins 5-6) |
| Analog input 4 | Temperature measurement (pins 15-16) | Voltage measurement (pins 7-8) |

Table 3.2 Settings on the expansion card

| Jumper Position | Default Setting | |
|---|---|---|
| Analog input 5 | Current measurement (pins 9-10) | Voltage measurement (pins 1-2) |
| Analog input 6 | Current measurement (pins 11-12) | Voltage measurement (pins 3-4) |
| Analog input 7 | Temperature measurement (pins 13-14) | Voltage measurement (pins 5-6) |
| Analog input 8 | Temperature measurement (pins 15-16) | Voltage measurement (pins 7-8) |

## 3.5 Installation in a Front Panel

The device is installed from the front in the front panel of switching cabinets or consoles and secured on the back with 6 nuts. A mounting cutout and 6 bored holes for the mounting bolts must be provided for the front panel.

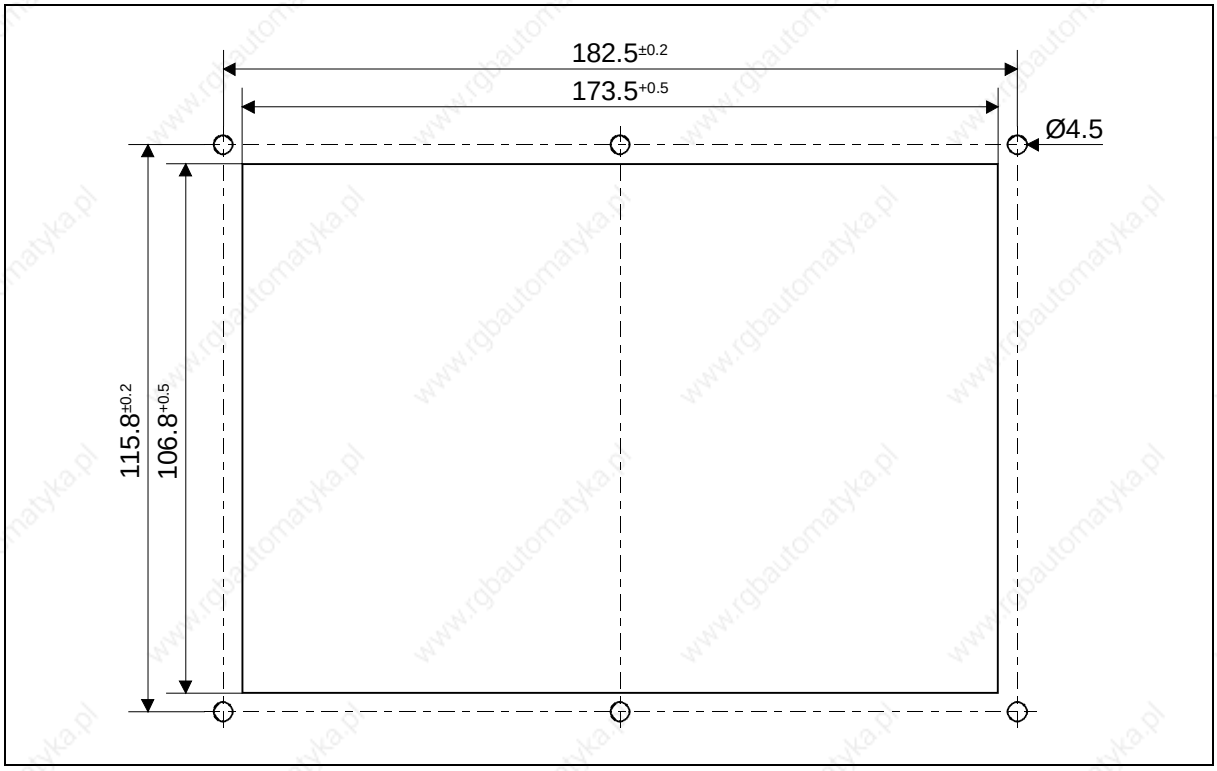



Figure 3.2 Mounting cutout and bored holes

The front panel may not exceed 8 mm in thickness. The mounting depth of the device is 112 mm. Remember to leave sufficient space under the device for the connection plugs. For the size of the front plate, see chapter 2.3.2.

**Note:**
To prevent the device from overheating during operation, adhere to the following points.

- Do not expose the front plate to direct sunlight.
- Make sure that the ventilation openings in the device housing are not obstructed after installation.
- The standard installation position is vertical with the plug connectors pointing down.

**Note:**
The following conditions must be met to achieve a protection rating of IP65.

- Levelness of the front plate: Maximum of 0.5 mm
- Correct positioning of the sealing around the front plate
- Typical moment of torsion of the nuts: 1.5 Nm

## 3.6 Connecting the I/O Devices

The following figure shows the bottom of the IMC01 with the connection elements.

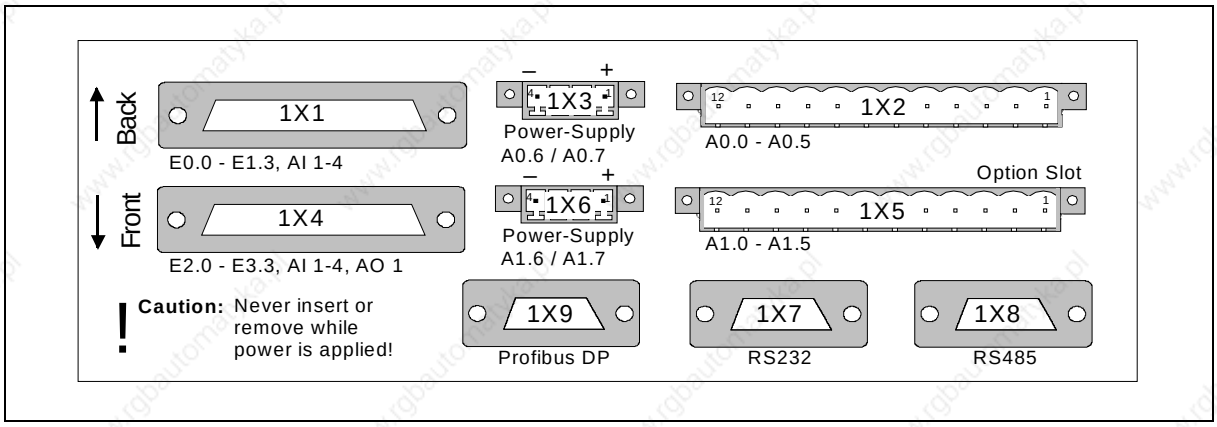



Figure 3.3 Bottom of the device with connection elements

### 3.6.1 Installation Guidelines

- The housing must be connected to the protective conductor with the threaded bolts on the back of the device.
- If you want to use the relay outputs with both SELV (Safety Extra Low Voltage) and higher voltages (e.g., alternating current power networks), one relay each must be left uncircuited between the SELV outputs and the power network outputs to ensure secure isolation of the SELV current circuit from the power network voltage.
- Do not plug or unplug connection cables while the power is on. To do so may damage the device.
- Do not plug or unplug the counter plugs while the power is on when voltages of ≥ 48 V DC are being used.
- When connecting the cables, ensure that any disconnected connections do not cause parasitic voltages (e.g., between 230 V outputs and the other connections of the controller).
- Unconnected power lines can destroy the device (e.g., feed-in on the digital outputs but no voltage supply on the device).
- With the electronic outputs, no countervoltage (i.e., external power source due to wire jumper or similar) may be applied to the plug connector.
- Protection against pole reversal of the supply voltages only exists in connection with an external fuse.
- A short circuit of the voltage supply pins of the PROFIBUS interface which lasts longer than one second may destroy the interface.
- There is a stud bolt on the back for connection of the shield. Ensure good conductivity (HF) when connecting the shield to cabinet ground.
- The galvanic connection of cabinet ground and M24 of the 24 V voltage supplies is made at the central grounding point outside the device.
- To the extent possible, all cables must be shielded.

- All ground potentials which enter and exit the device are galvanically connected with each other internally.
- All 24 V voltage lines entering the device are connected to the shield potential with capacitors (22 nF/500 V).

## 3.6.2 Grounding Concept

The figure below shows how the ground potentials of the board are connected to each other.

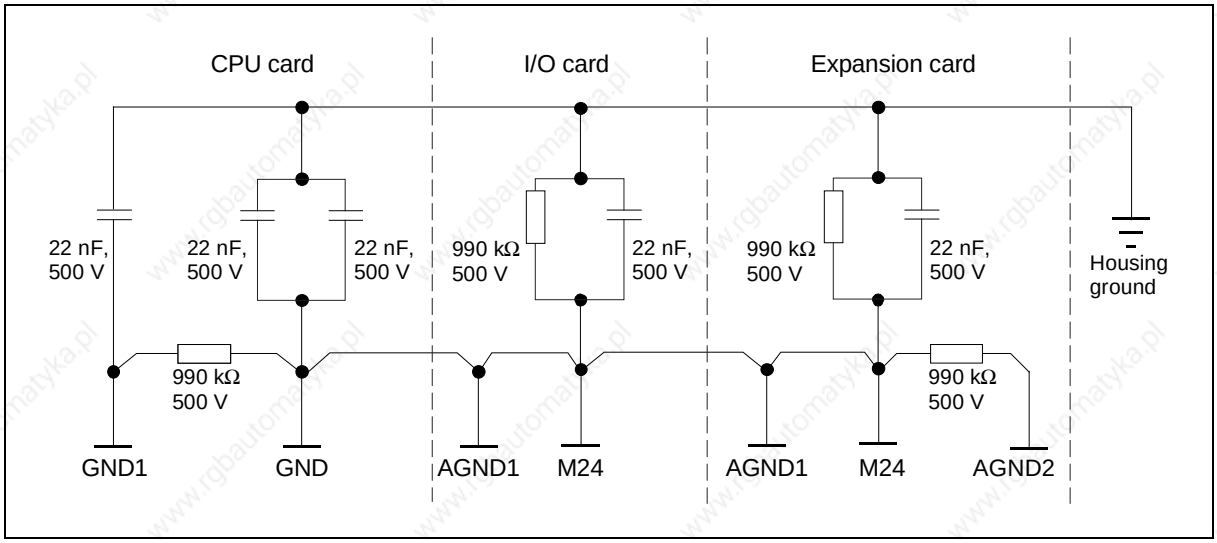


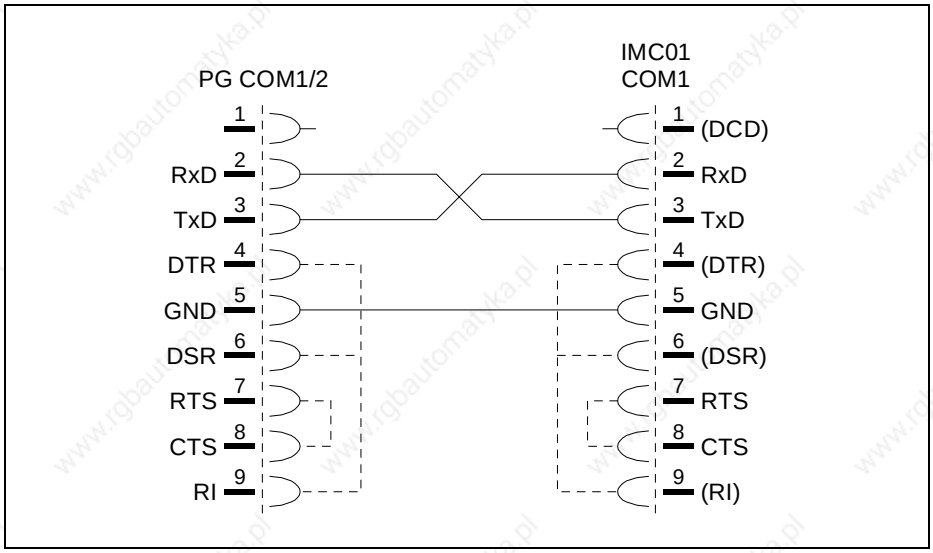

Figure 3.4 Grounding potentials of the board

| Designation | Reference Potential for |
|---|---|
| AGND1 | Analog Inputs |
| GND | Serial interfaces |
| M24 | 24 V voltage supply, digital inputs/outputs |
| GND1 | PROFIBUS |
| AGND2 | Analog output |

The signal ground potentials are galvanically connected with each other. They are high-ohmic and capacitively coupled with the housing ground.

### 3.6.3 Connection Cable from PG to IMC01 (Zero Modem Cable)

**Serial interfaces** Connection from PG to IMC01




The connections indicated by broken lines are not absolutely necessary but do not interfere with data communication either.

## 3.7 Parameterizing PROFIBUS

The IMC01 has already been parameterized. Only the bus address still has to be set. See chapter 5.2.3.

Parameter files are available in directory DPS/GSD to enter the IMC01 in the master configuration (COM PROFIBUS).

Transfer these files to the appropriate directories of COM PROFIBUS.

# 4 Commissioning with C or IMC0x-PLC Programming

To commission the IMC01, proceed exactly as described below.

**Caution:**

Commissioning, installation and wiring may only be performed by qualified personnel who are familiar with and observe the general rules of technology and the applicable regulations and standards.

Before connecting the I/O devices and starting your user application, make sure that persons or systems will not be endangered by execution of the program.

1. Connect the power supply and the development computer to the IMC01.
2. Install the IMC01-BSP on the development computer.
3. Prepare program and generate system.
4. Load and test the RMOS system and the user program.
5. Set the type of measurement for the analog inputs.
6. Install in the front panel.
7. Connect the I/O devices.
8. Parameterize PROFIBUS.

## 4.1 Connecting the Power Supply and the Development Computer to the IMC01

The device is powered by the 4-pin pin strips on the I/O card (1X3, see chapter 2.2.2) and, if present, on the expansion card (1X6, see chapter 2.2.3).

**Caution:**

Suitable fuses (see chapter 2.3.1) must be installed in the feeder lines of the supply voltage, and, if necessary, an on/off switch.

Before turning on the supply voltage, make sure that all other connections are wired correctly.

Do not plug or unplug connection cables while the power is on. To do so may damage the device.

The development computer is connected to the IMC01 with a zero modem cable. See chapter 4.7.3.

## 4.2 Installing the IMC01-BSP on the Development Computer

### Prerequisites

RMOS3 V3.20 must be installed on the development computer before you can begin installing the included software.

### Installation

1. Turn on the development computer, and start MS-DOS.
2. Change to the RMOS master directory. For example:

   ```
   cd \RM3DEV
   ```

3. Place the product floppy disk in the floppy disk drive.
4. Start the installation program:

   ```
   a:install
   ```

For the directories and files which are set up, see the **READIMC1.1P0** file on the product floppy disk. If you change the directory structure, you will also have to adjust the entries in the batch files.

## 4.3 Preparing the Program and Generating the System

The procedure for program preparation and system generation depends on whether you start your user task with the RMOS system, or you want to load it during running operation.

Subdirectories APLSTART, APLBIOS and APLLOAD contain sample applications to illustrate the preliminary steps used in preparing the program.

Before calling GENSYSC.BAT, the paths of the development tools must be set (e.g., via search path or configuration variables).

## How to proceed for system generation

| Step | Files Required | Files Generated |
|---|---|---|
| **Compile** the IMC01-specific RMOS source files if you made changes to them. | GENBIOS.BAT and *.C in SYSIMC1\BIOS | *.OBJ |
| **Edit** the program sources | *.C in SYSIMC1\APLSTART or SYSIMC1\APLBIOS | |
| **Compile** [1)] | *.C in SYSIMC1\APLSTART or SYSIMC1\APLBIOS, IMC01BAS.H in SYSIMC1\BIOS | *.OBJ |
| **Link and locate** [1)] | SYSIMC1\APLSTART, SYSIMC1\APLBIOS or SYSIMC1\APLLOAD<br>IMC01BAS.LIB in SYSIMC1\BIOS | RM3_PC1.LOC<br>RM3_PC1.MAP<br>RM3NUC.GAT |
| **Generate** the system file [1)] | QCOFTS | RM3_PC1.SYS |

*1) Compiling, linking and locating, and generating the system file are performed with GENSYSC.BAT.*

Transferring the system file is described in chapter 4.4.

## How to proceed for loadable tasks

| Step | Files Required | Files Generated |
|---|---|---|
| **Generate** a suitable system for loadable tasks with coprocessor emulation enabled (e.g., SYSIMC5\APLLOAD). | GENSYSC.BAT in SYSIMC1\APLLOAD | RM3_PC1.SYS |
| **Edit** the user task | *.C in EXPIMC1\APLSTART | |
| **Compile** the user task [1)] | *.C in EXPIMC1\APLSTART<br>IMC01BAS.H in SYSIMC1\BIOS | *.OBJ |
| **Link** the user task [1)] | BIOSAPI.OBJ in SYSIMC1\BIOS | APLSTART.386 |

*1) Compiling and linking are performed with APLSTART.BAT.*

If you are going to use the system file and the user task from this example without changes, the same effect is produced on the IMC01 as with the example from SYSIMC1\APLSTART.

Transferring the system file and loading user tasks are described in chapter 4.4.

**Note:**
RMOS 3.20 always requires a coprocessor or a coprocessor emulation for the execution of tasks which were loaded with LOADRM.EXE or the Cadul HLL Debugger XDB. Since the IMC01 does not have a coprocessor, it is essential to include the coprocessor emulation here. This has already been done in our APLLOAD example.

### 4.3.1 Using the Sample Applications

**APLSTART** Simple beginner's example with text outputs on the display

**APLLOAD** Generation of an RMOS system to which tasks can be loaded via the HLL Debugger or LOADRM.EXE. Since there are no outputs to the I/O devices in the first task in this example, the user task is not interrupted.

**APLBIOS** Extensive example containing executable sample programs as individual tasks in addition to the BIOS functions. You can use the TEST.H header file to control which tasks you want to start.

| Task | Meaning | File Name (.C or .OBJ) |
|---|---|---|
| UTSK_AO | Processing of the analog outputs | UTSKAO |
| UTSK_AI | Processing of the analog inputs | UTSKAI |
| UTSK_DO | Processing of the digital outputs | UTSKDO |
| UTSK_DI | Processing of the digital inputs | UTSKDI |
| UTSK_KEYB | Processing of the keyboard (call interface) | UTSKKEYB |
| UTSK_Dummy | Dummy task, no function | UTSKDUMM |

### Using the sample task APLBIOS on the IMC01

When keyboard processing is active, you can perform the following actions.

| Key | Effect |
|---|---|
| F1 | Indicates analog inputs 0 to 3 |
| F2 | Indicates analog inputs 8 to 11 |
| F3 | Indicates the digital inputs |
| RESET | Triggers a system reset after expiration of the watchdog |

### 4.3.2 Using the Coprocessor Emulation

To be able to use the numeric coprocessor emulation, the following entries must be made before system generation.

1. Remove the commentary characters from the following line in the RMCONF.C file.
   `#define FPU_EMULATOR`
2. In the CCAPL.INP file, add the `-VNDP` option to all compiler calls. This option has already been set when the IMC01 is delivered. The CCAPL.INP file contains the compiler options and is called from the CCAPL.BAT file. This makes it easy to compile a single file.

**Note:**
Use of the coprocessor emulation is described in the READIMC1.1P0 file.

## 4.4 Loading and Testing the RMOS System and the User Program

The steps required to load the RMOS system and your user program to the IMC01 and test them there are described below.

### Preparations

Copy the RM3_PC1.SYS system file to subdirectory UTILIMC1. One of the examples included in subdirectory SYSIMC1 can also be used.

### Loading and starting

1. Establish connection.
2. Clear entire user FLASH memory if necessary.
3. Transfer RMOS system and user program.
4. Start the system.
5. Load user tasks if necessary.

These steps are explained in chapters 4.4.2 to 4.4.6.

**Note:**
The first four steps are executed by the FLASH.BAT batch file and documented there. The COM1 interface of the development computer is used. If you want to use another interface, you will have to change the appropriate settings in this file.

### 4.4.1 Behavior during Booting/Reset

After the power is turned on or after a hardware reset (e.g., triggered by the watchdog), the processor begins executing the boot program.

First, it initializes its internal registers and then checks to determine whether a serial connection to the development computer exists and software must be loaded. See chapter 4.4.2. If not, the RMOS operating system is started and execution of the user program begins.

**Note:**
It is the responsibility of the user to ensure that executable program code is located at the start address of the user program. See chapter 2.1.3.

### 4.4.2 Establishing the Connection

Proceed as follows to establish the connection between the IMC01 and the development computer.

1. Plug a zero modem cable into COM1 of the IMC01 and a free serial interface on your development computer.
2. Enter the following command after the MS-DOS prompt.
   Be sure to use the correct upper and lower case letters.

   ```
   synchro COMx 19200
   ```

   (COMx is the COM1 or COM2 computer interface used.)
3. Turn on the power supply of the IMC01.

   The FLASH load program is started on the IMC01. The development computer indicates whether the connection was established successfully.

   The IMC01 start screen shows the following information.

   - "`Flashloader`"
     The screen remains until the next reset. The display of the IMC01 also shows the function currently being executed by the loader.
   - "`starting System`..."
     After approx. two seconds the RMOS application is started. The monitor is cleared. If the screen remains, no valid system is loaded.

### 4.4.3 Deleting Software from the IMC01

New software can only be loaded correctly in the user FLASH memory of the IMC01 when the required sectors have been deleted before the transfer.

Proceed as follows to delete the entire user FLASH memory or individual sectors from it.

1. Make sure that the connection between the IMC01 and the development computer is established. See chapters 4.4.1 and 4.4.2.
2. Enter the following command after the MS-DOS prompt.

   `send COMx 19200` ***`Configuration-file`***

   ***`Configuration-file`*** is the complete file name (including path) of an ASCII file (e.g., DELFW.CFG) with the following contents.

| Line No. | Contents (Hex) [1] | Meaning |
|---|---|---|
| 1 | 7 | Number of valid entries in the file |
| 2 | 0 | Reserved |
| 3 | 14 | "Delete FLASH memory" command |
| 4 | 0 | Delete entire user FLASH memory<br>or sector with the specified number (1 to $n_{max}$ [2]) |
| 5 | 0 | Reserved |
| 6 | 0 | Reserved |
| 7 | 9600 | Baud rate for commands which follow<br>(hexadecimal, 38,400 = 0x9600) |
| From 8 | Any | Examples: Commentary or remarks |

*1) All numerical values are hexadecimal numbers.*

*2) $n_{max}$ = (Size of the user FLASH memory (in kbytes) /128) – 1*

### 4.4.4 Loading an RMOS System or a User Program

Proceed as follows to load a new RMOS system or a user program to the IMC01.

1. Make sure that the connection between the IMC01 and the development computer is established and the clear command was executed. See chapters 4.4.1 and 4.4.2.
2. Enter the following command after the MS-DOS prompt.

   `send COMx 38400 Configuration-file, , Binary-file`

   *`Configuration-file`* is the complete file name (including path) of an ASCII file (e.g., LOADFW.CFG) with the following contents.

| Line No. | Contents (Hex) [1] | Meaning |
|---|---|---|
| 1 | 7 | Number of valid entries in the file |
| 2 | 0 | Reserved |
| 3 | 16 | "Load FLASH memory" command |
| 4 | 0 | Reserved |
| 5 | 0 | Transfer entire binary file or the number (hexadecimal) of bytes to be transferred from the binary file |
| 6 | 0 | Byte offset for the start address of the user FLASH memory (hexadecimal) |
| 7 | 0 | Baud rate for commands which follow unchanged<br>or new baud rate (hexadecimal) |
| From 8 | Any | Examples: Commentary or remarks |

*1) All numerical values are hexadecimal numbers.*

*`Binary file`* is the complete file name (including path) of the RMOS system file or user program file to be loaded (e.g., RM3_PC1.SYS).

**Note:**
This procedure cannot be used to reload a user task. See chapter 4.4.6.

### 4.4.5 Starting the System or User Program

To terminate the connection between the IMC01 and the development computer and to start the RMOS system with your user task, enter the following command after the MS-DOS prompt.

`send COMx 19200` *`Configuration-file`*

*`Configuration-file`* is the complete file name (including path) of an ASCII file (e.g., STARTFW.CFG) with the following contents.

| Line No. | Contents (Hex) [1] | Meaning |
|---|---|---|
| 1 | 7 | Number of valid entries in the file |
| 2 | 0 | Reserved |
| 3 | 17 | "Start application" command |
| 4 | 0 | Reserved |
| 5 | 0 | Reserved |
| 6 | 0 | Reserved |
| 7 | 0 | Reserved |
| From 8 | Any | Examples: Commentary or remarks |

*1) All numerical values are hexadecimal numbers.*

This concludes data transfer to the IMC01, deletes the contents of the SRAM, and restarts the device with a hardware reset triggered by the watchdog. Since the program for connection establishment on the development computer is not active, the IMC01 boots with the RMOS system and starts the user program.

**Note:**
The parameters of the COM1 interface are set to the following values.
Baud rate of 19200 (default setting after startup)
No parity bit
8 data bits
1 stop bit

### 4.4.6 Testing and Loading User Programs with the Debugger

To test a user program, you can use the low-level Debugger included with RMOS or the XDB from Organon. Proceed as described in the applicable manuals. You must use the COM1 serial interface on the IMC01.

**Note:**
Since the RMOS system file is loaded in the FLASH memory, you can only use hardware breaks here. The XDB.INI file under subdirectory SYSIMC1\APLSTART already contains an appropriate setting. Copy this file to the directory of XDB.

Reloadable tasks are loaded in work memory (SRAM), and software breaks can be used.

During the testing phase, you should work with reloadable tasks so that the entire system does not have to be transferred again when errors occur. The example from subdirectory APLLOAD can be used as the system file, and added step-by-step to your already tested user modules.

When generating a reloadable task, the included BIOSAPI.OBJ file must be linked. The APLSTART.BAT batch file in subdirectory EXPIMC1\APLSTART\CADUL contains an example of this.

**Note:**
As with every RMOS system, the RMRS232C device driver must be linked in before tasks can be reloaded with LOADRM.EXE. Use the following entry in the CONFIG.SYS file.

```
Device=<RMOS path>\UTIL\ RMRS232C.EXE SSDEV {COM1|COM2} 19200
```

## 4.5 Setting the Type of Measurement for the Analog Inputs

Plug-in jumpers are available on the I/O card and, if present, on the expansion card to set the type of measurement for the analog inputs. These jumpers are accessible from the top on the opened device. The figure below shows a view of the top of the opened device with the plug-in jumpers as preset at the factory.

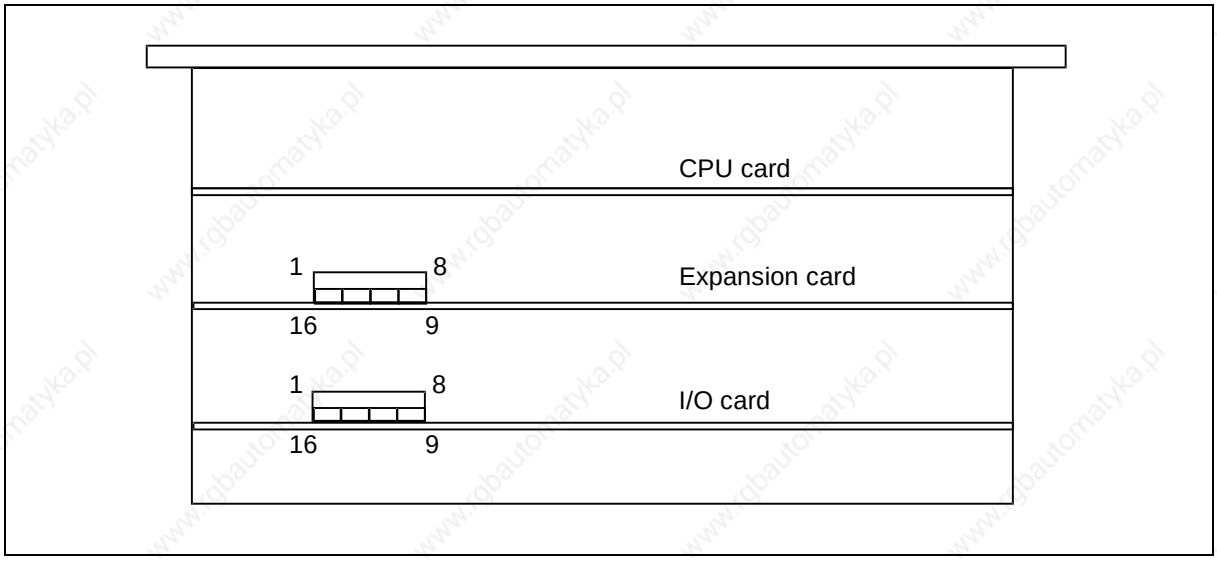



Figure 4.1 Location and presettings of the plug-in jumpers (view of the top)

Table 4.1 Settings on the I/O card

| Jumper Position | Default Setting | |
|---|---|---|
| Analog input 1 | Current measurement (pins 9-10) | Voltage measurement (pins 1-2) |
| Analog input 2 | Current measurement (pins 11-12) | Voltage measurement (pins 3-4) |
| Analog input 3 | Temperature measurement (pins 13-14) | Voltage measurement (pins 5-6) |
| Analog input 4 | Temperature measurement (pins 15-16) | Voltage measurement (pins 7-8) |

Table 4.2 Settings on the expansion card

| Jumper Position | Default Setting | |
|---|---|---|
| Analog input 5 | Current measurement (pins 9-10) | Voltage measurement (pins 1-2) |
| Analog input 6 | Current measurement (pins 11-12) | Voltage measurement (pins 3-4) |
| Analog input 7 | Temperature measurement (pins 13-14) | Voltage measurement (pins 5-6) |
| Analog input 8 | Temperature measurement (pins 15-16) | Voltage measurement (pins 7-8) |

## 4.6 Installation in a Front Panel

The device is installed from the front in the front panel of switching cabinets or consoles and secured on the back with 6 nuts. A mounting cutout and 6 bored holes for the mounting bolts must be provided for the front panel.

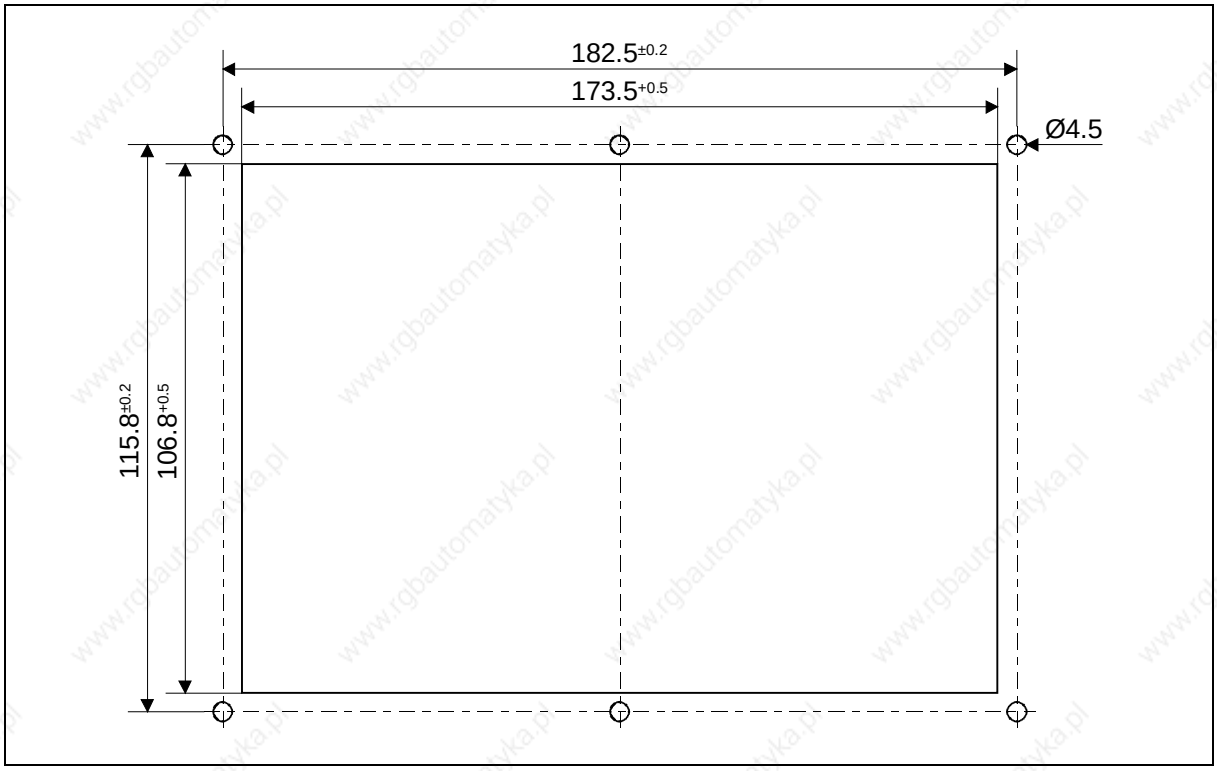



Figure 4.2 Mounting cutout and bored holes

The front panel may not exceed 8 mm in thickness. The mounting depth of the device is 112 mm. Remember to leave sufficient space under the device for the connection plugs. For the size of the front plate, see chapter 2.3.2.

**Note:**
To prevent the device from overheating during operation, adhere to the following points.

- Do not expose the front plate to direct sunlight.
- Make sure that the ventilation openings in the device housing are not obstructed after installation.
- The standard installation position is vertical with the plug connectors pointing down.

**Note:**
The following conditions must be met to achieve a protection rating of IP65.

- Levelness of the front plate: Maximum of 0.5 mm
- Correct positioning of the sealing around the front plate
- Typical moment of torsion of the nuts: 1.5 Nm

## 4.7 Connecting the I/O Devices

The following figure shows the bottom of the IMC01 with the connection elements.

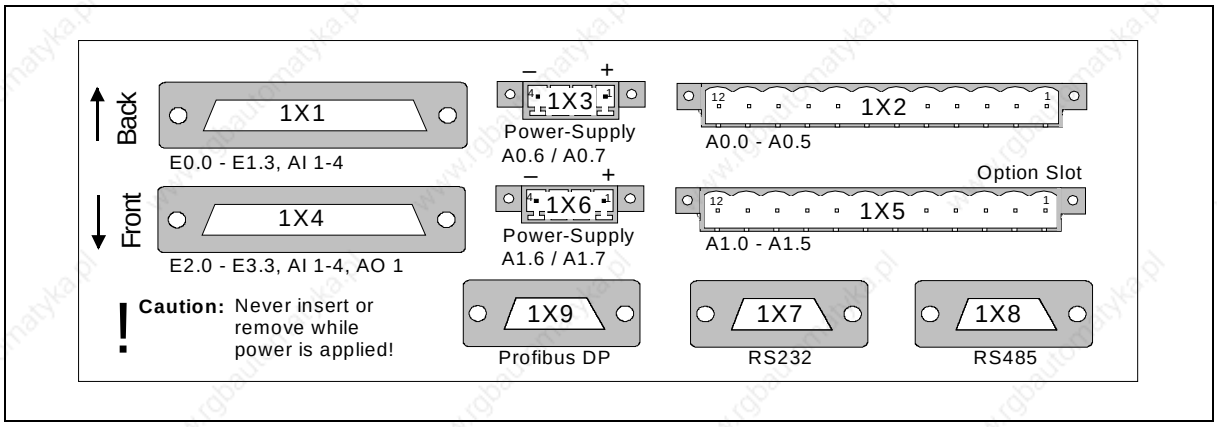



Figure 4.3 Bottom of the device with connection elements

### 4.7.1 Installation Guidelines

- The housing must be connected to the protective conductor with the threaded bolts on the back of the device.
- If you want to use the relay outputs with both SELV (Safety Extra Low Voltage) and higher voltages (e.g., alternating current power networks), one relay each must be left uncircuited between the SELV outputs and the power network outputs to ensure secure isolation of the SELV current circuit from the power network.
- Do not plug or unplug connection cables while the power is on. To do so may damage the device.
- Do not plug or unplug the counter plugs while the power is on when voltages of ≥ 48 V DC are being used.
- When connecting the cables, ensure that any disconnected connections do not cause parasitic voltages (e.g., between 230 V outputs and the other connections of the controller).
- Unconnected power lines can destroy the device (e.g., feed-in on the digital outputs but no voltage supply on the device).
- With the electronic outputs, no countervoltage (i.e., external power source due to wire jumper or similar) may be applied to the plug connector.
- Protection against pole reversal of the supply voltages only exists in connection with an external fuse.
- A short circuit of the voltage supply pins of the PROFIBUS interface which lasts longer than one second may destroy the interface.
- There is a stud bolt on the back for connection of the shield. Ensure good conductivity (HF) when connecting the shield to cabinet ground.
- The galvanic connection between cabinet ground and M24 of the 24 V voltage supplies is made at the central grounding point outside the device.
- To the extent possible, all cables must be shielded.

- All ground potentials which enter and exit the device are galvanically connected with each other internally.
- All 24 V voltage lines entering the device are connected to the shield potential with capacitors (22 nF/500 V).

## 4.7.2 Grounding Concept

The figure below shows how the ground potentials of the board are connected to each other.

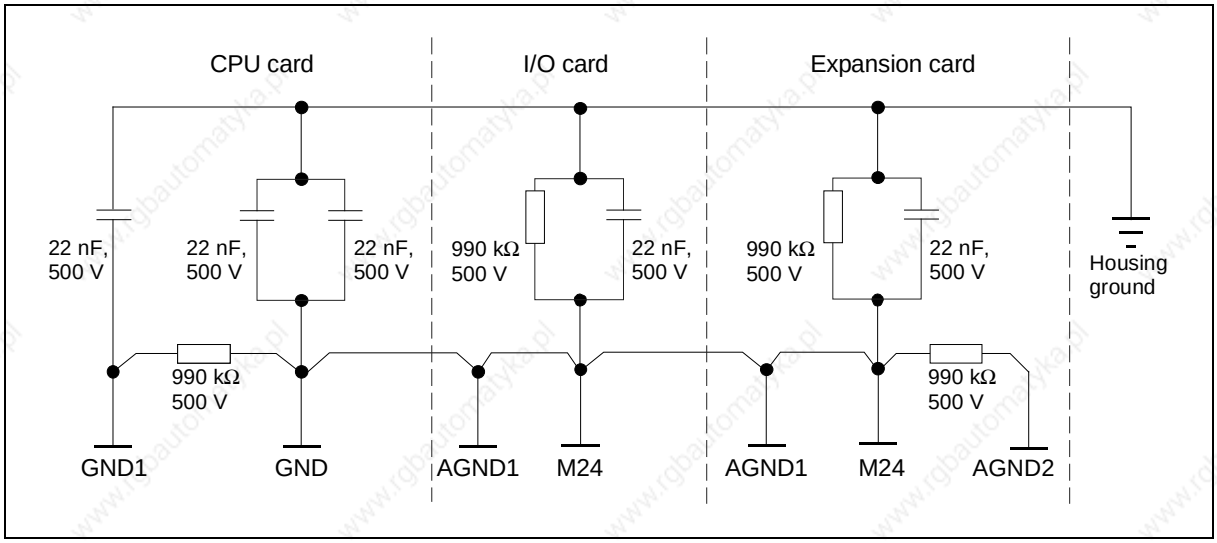


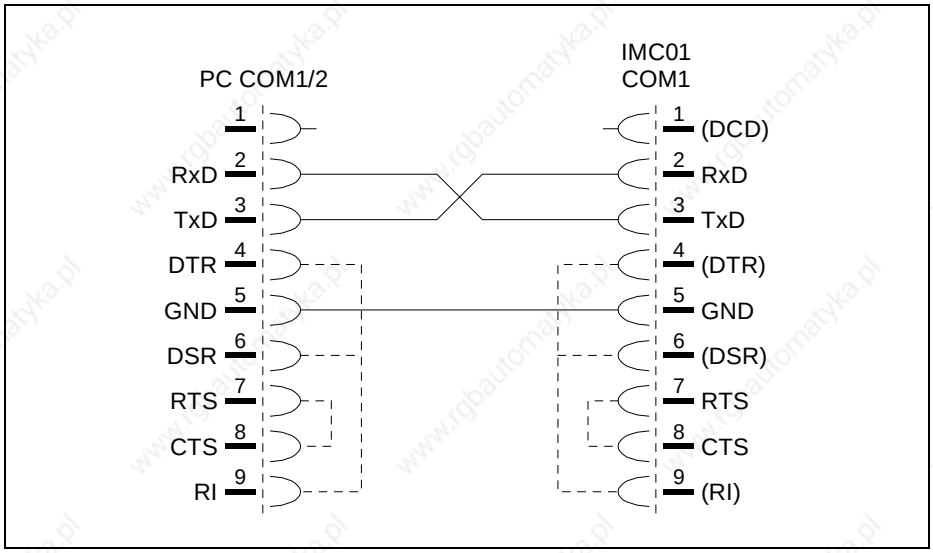

Figure 4.4 Grounding potentials of the board

| Designation | Reference Potential for |
|---|---|
| AGND1 | Analog inputs |
| GND | Serial interfaces |
| M24 | 24 V voltage supply, digital inputs/outputs |
| GND1 | PROFIBUS |
| AGND2 | Analog output |

The signal ground potentials are galvanically connected with each other. They are high-ohmic and capacitively coupled with the housing ground.

### 4.7.3 Connection Cable from PC to IMC01 (Zero Modem Cable)

**Serial interfaces** Connection from PC to IMC01




The connections indicated by broken lines are not absolutely necessary but do not interfere with data communication either.

## 4.8 Parameterizing PROFIBUS

The IMC01 has already been parameterized. Only the bus address still has to be set. See chapter 6.12.4.

Parameter files are available in directory DPS/GSD to enter the IMC01 in the master configuration (COM PROFIBUS).

Transfer these files to the appropriate directories of COM PROFIBUS.

# 5 Compact PLC

The Compact PLC has already been loaded in the device and can be used immediately after power-on. The Compact PLC is a preconfigured RMOS system with the following components.

- RMOS3.20
- IMC01 PLC
- DP slave coupling (32 bytes of digital inputs, 32 bytes of digital outputs)
- I/O interface in STEP 5 data blocks

A data block interface can be used to manipulate the following peripherals from the STEP 5 program.

- 24 digital inputs, 16 digital outputs
- 8 analog inputs (current, voltage and PT100 temperature)
- 1 analog output
- DP slave
- Keys and LEDs of the operator panel
- LCD display
- Hardware clock

HLL function blocks are used to couple the STEP 5 and RMOS worlds. If used, these blocks must be added to OB1 for the applicable interface functionality.

- HLL FB 208 FB_DIS Character output on the display
- HLL FB 210 FB_IN Read in the peripherals
- HLL FB 211 FB_OUT Output to the peripherals

## 5.1 Reloading the Compact PLC to User FLASH Memory

When needed, the Compact PLC can be reloaded to the IMC01 with the FLASH loader. The software for this is located in directory SYSIMC1\COMPACT.PLC. Here you will find, among others, a finished RM3_PC1.SYS system, a sample STEP 5 program and a batch file for downloading with the FLASH loader.

Directory SYSIMC1\COMPACT.PLC\LOADER contains the following batch files for the reload.

LD_SYS.BAT Reload the system, including the STEP 5 program image.bin from the SYSIMC1\COMPACT.PLC\STEP5 directory

LD_MC5.BAT Reload the STEP 5 program SYSIMC1\COMPACT.PLC\STEP5\image.bin.

The system is not loaded again.

# 5.2 Data Block Interfaces for the Peripherals

## 5.2.1 Character Output on the Display (FB_DIS)

The function block for character output on the display (FB_DIS) permits specific texts to be displayed with the STEP 5 application. The data block must contain at least 64 data words.

Function block FB_DIS uses data block DB_DIS whose values are indicated on the display.

### Structure of DB_DIS

| | 15 | 0 | |
|---|---|---|---|
| DW 0 to DW 7 | Character code | Character code | 1st line |
| DW 8 to DW 15 | Character code | Character code | 2nd line |
| DW 16 to DW 23 | Character code | Character code | 3rd line |
| DW 24 to DW 31 | Character code | Character code | 4th line |

### Call interface of FB_DIS (FB208)

The function block must be set up by the STEP 5 programmer. Remember, however, that no instructions at all are permitted in the function block.


**STL**

FB208

Name: FB_DIS

BEZ : DBNR E/A/D/B/T/Z: E BI/BY/W/D: W

: BE

**Example**

| **Call in OB1** | | **Description** |
|---|---|---|
| L | DB_DIS | Load number of data block DB_DIS |
| T | MB 8 | Transfer to flag byte 8 |
| SPA | FB_DIS | Call HLL function block FB_DIS |
| DBNR: | MW 8 | Transfer number of DB_DIS as formal operand |
| .... | | |

**Caution**

The formal operand must be transferred in a flag word. Otherwise the function block will not be processed.

The data block number is transferred to the function block in the left-hand half of the flag word.

A status is returned in the right-hand half of the flag word.

- FFh = The referenced data block cannot be used. It either does not exist or is too small. The function block was not processed.
- 00h = The referenced data block could be used.

### 5.2.2 Using the HW Peripherals (FB_IN, FB_OUT)

The IMC0x-PLC controls the digital inputs/outputs with the conventional standard mechanisms (external I/O). The other HW peripherals are handled by the HLL function blocks.

The function blocks for reading in (FB_IN) and reading out (FB_OUT) the peripherals both use the same DB_PER data block to store their data. The data block must be set up by the STEP 5 programmer. The data block has a length of 132 DWs.

## Structure of DB_PER

| | 15 | 0 | |
|---|---|---|---|
| DW 0 | Analog input AI1 | | Input area |
| DW 1 | Analog input AI2 | | |
| DW 2 | Analog input AI3 | | |
| DW 3 | Analog input AI4 | | |
| DW 4 | Analog input AI5 | | |
| DW 5 | Analog input AI6 | | |
| DW 6 | Analog input AI7 | | |
| DW 7 | Analog input AI8 | | |
| DW 8 | Configuration of the analog inputs | | |
| DW 9 | Current LED status | | |
| DW 10 | Reserved | | |
| DW 11 | Free | Hours | Current time |
| DW 12 | Minutes | Seconds | |
| DW 13 | Day of the week | Day | Current date |
| DW 14 | Month | Year | |
| DW 15 | Results of the background test | | |
| DW 16 | Analog output | | Output area |
| DW 17 | Current LED status | | |
| DW 18 | Reserved | | |
| DW 19 | Status | Hours | Set new time |
| DW 20 | Minutes | Seconds | |
| DW 21 | Day of the week | Day | Set new date |
| DW 22 | Month | Year | |
| DW 23 to DW 24 | Free | | |
| DW 25 | Second key | First key | Key codes |
| DW 26 to DW 28 | Free | | |
| DW 29 | Station address | | DP |
| DW 30 to DW 45 | DP "input" area | | |
| DW 46 to DW 99 | Reserved | | |
| DW 100 to DW 115 | DP "output" area | | |
| DW 116 to DW 131 | Reserved | | |

## DW 0 to DW 7

The analog inputs are 12 bits in length. The entered values are raw ADC data which require interpretation by the PLC program. The analog inputs have different interface options whose settings can be determined with DW 6.

When channels AI3, AI4, AI7 und AI8 are configured for temperature measurement, the values are specified in tenths of a degree.

When the jumper setting is "voltage," the voltage measuring range (unipolar/bipolar, 5 V/10 V) can be set for these channels in DB_CONFIG.

- AI1 Voltage or current
- AI2 Voltage or current
- AI3 Voltage or temperature
- AI4 Voltage or temperature
- AI5 Voltage or current *)
- AI6 Voltage or current *)
- AI7 Voltage or temperature *)
- AI8 Voltage or temperature *)

*) *FFFFh is always entered if there is no expansion PCB.*

## DW 8

The configuration of the analog inputs can be determined with this data word.

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | EXT | AI8 | AI7 | AI6 | AI5 | AI4 | AI3 | AI2 | AI1 |
| | | | | | | | nv=0<br>v = 1 | U=0<br>T = 1 | U=0<br>T = 1 | U=0<br>I = 1 | U=0<br>I = 1 | U=0<br>T = 1 | U=0<br>T = 1 | U=0<br>I = 1 | U=0<br>I = 1 |

*EXT: Info bit showing whether expansion module is installed*
*Not present (nv) = 0*
*Present (v) = 1*

## DW 9

The current status of the LEDs can be determined with this data word.

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | LED 8 | LED 7 | LED 6 | LED 5 | LED 4 | LED 3 | LED 2 | LED 1 |
| | | | | | | | | Off=0<br>On=1 | Off=0<br>On=1 | Off=0<br>On=1 | Off=0<br>On=1 | Off=0<br>On=1 | Off=0<br>On=1 | Off=0<br>On=1 | Off=0<br>On=1 |

## DW 11 to DW 12

The time is displayed as shown below.

- Hours: 0 to 23 hr
- Minutes: 0 to 59 min
- Seconds: 0 to 59 sec

## DW 13 to DW 14

The date is displayed as shown below.

- Day of the week: 1 to 7, whereby 1 = Monday, .. , 6 = Saturday, 7 = Sunday
- Day: 1 to 31
- Month: 1 to 12
- Year: 0 to 99

## DW 15

The data word contains the states of the test results of a test task running cyclically in the background.

The following are tested.

**Backup battery** Load test at intervals of approx. 6 hours

**RAM** Dynamic read/write test

**Configuration** Plausibility test of the configuration data (driver switchover, interface parameters, and so on) on the EEPROM

**Short circuit** Tests the electronic digital outputs for short circuit

IO: Both bits are set when there is a short circuit on one of the electronic digital outputs of the basic board.

EXT: Both bits are set when there is a short circuit on one of the electronic digital outputs of the expansion board.

**DP** Checks status of the DP bus'

**Trap** Checks for excessive interrupt load on the system since last power-on

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Trap | | | | | | | DP | EXT | EXT | IO | IO | | RAM | FLSH | BATT |
| OK=0<br>Hit=1 | | | | | | | OK=0<br>Def=1 | OK=0<br>ks=1 | OK=0<br>ks=1 | OK=0<br>ks=1 | OK=0<br>ks=1 | | OK=0<br>Def=1 | OK=0<br>Def=1 | OK=0<br>Def=1 |

OK = Okay, no error

Def = Not okay, defect, or no master detected for DP

ks = Short circuit

Hit = System load too high

## DW 16

The resolution of the analog output is 10 bits.The entered values are raw ADC data which require evaluation by the PLC program. Values greater than $2^{10}$ are reduced to the maximum value of $2^{10}$.

## DW 17

The indication status of the LEDs can be set with this data word.

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | LED 7 | LED 6 | LED 5 | LED 4 | LED 3 | LED 2 | LED 1 | LED 0 |
| | | | | | | | | Off=0<br>On=1 | Off=0<br>On=1 | Off=0<br>On=1 | Off=0<br>On=1 | Off=0<br>On=1 | Off=0<br>On=1 | Off=0<br>On=1 | Off=0<br>On=1 |

## DW 19 to DW 22

The time/date of the real-time clock cannot only be read. It can also be set. For permissible formats, see DW 9 to DW 12. The value to be set is entered in the appropriate data byte. The individual values (e.g., seconds, day of the week, and so on) can also be set. After the function block is called, all entered values are reset to FFh. DL 19 contains the results (status) which were set.

- 00h = All values were accepted in the real-time clock.
- Value range of one of the values violated
  01h Erroneous value for hours
  02h Erroneous value for minutes
  04h Erroneous value for seconds
  10h Erroneous value for day of the week
  20h Erroneous value for day
  40h Erroneous value for month
  80h Erroneous value for year

If there are several errors, the error values are or-linked with each other.

**Note:**
No check is made for relationship when the clock block is set. For example, 02.30.97 is a legal value. The value is not automatically corrected until the next change.

The contents of DW 19 to DW 22 should be preset to FFFFh since otherwise the time/date would be changed when FB_OUT is called for the first time.

## DW 25

The key codes of the keys currently being pressed are read out in this data word. The right-hand data byte (DR25) contains the key which was pressed first, and the left-hand data byte (DL25) contains the key which was pressed second. Only two simultaneously pressed keys can be detected. A second simultaneously activated key cannot be detected for keys S0 to S7.

The key codes correspond to the "special keycodes" described in IMC01BAS.H. See chapter 7.1.

## DW 29

This data word contains the currently valid station address. This address is taken from DB_CONFIG during startup. See chapter 5.2.3. The DP master can change the configured station address.

## DW 30 to DW 45

The IMC0x-PLC can write the DP "input" area (i.e., from the viewpoint of the DP master) as desired. This area is used to store the data which are to be indicated to the DP master as the process image. The area has a length of 64 data words.

## DW 100 to DW 115

The DP master can write the DP "output" area (i.e., the output area as seen from the viewpoint of the DP master) as desired. This area is used to store the data which the IMC0x-PLC is to use as control data. The area has length of 32 data words.

## Call interface of FB_IN (FB210)

The function block must be set up by the STEP 5 programmer. Remember, however, that no instructions are permitted at all for this function block.


**STL**

FB210

Name: FB_IN

BEZ : DBNR E/A/D/B/T/Z: E BI/BY/W/D: W

: BE

**Example**

| **Call in OB1** | | **Description** |
|---|---|---|
| L | DB_PER | Load number of the I/O data block |
| T | FW10 | Transfer to flag word 10 |
| SPA | FB_IN | Call HLL function block FB_IN |
| DBNR: | FW 10 | Transfer the number of DB_PER as formal operand |
| .... | | |

## Call interface of FB_OUT (FB211)

The function block must be set up by the STEP 5 programmer. Remember, however, that no instructions are permitted at all for this function block.


**STL**

FB211

Name: FB_OUT

BEZ : DBNR E/A/D/B/T/Z: E BI/BY/W/D: W

: BE

**Example**

| **Call in OB1** | | **Description** |
|---|---|---|
| L | DB_PER | Load number of the I/O data block |
| T | FW10 | Transfer to flag word 10 |
| SPA | FB_OUT | Call HLL function block FB_OUT |
| DBNR: | FW 10 | Transfer the number of DB_PER as formal operand |
| . . . . | | |

### Caution

The following applies to both function blocks.

The formal operand must be transferred in a flag word. Otherwise the function block will not be processed.

The data block number is transferred to the function block in the left-hand half of the flag word.

A status is returned in the right-hand half of the flag word.

- FFh = The referenced data block cannot be used. It either does not exist or is too small. The function block was not processed.
- FEh = The data block referenced in FB_OUT is not the same as the one in FB_IN. The function block was not processed.
- 00h = The referenced data block could be used.

### 5.2.3 Configuration in DB_CONFIG

The following settings can be made in the configuration data block.

- DP slave bus address
- Priority of the DP communication task
- Pause time of the DP communication task
- Configuration of the analog inputs (unipolar/bipolar, 5 V/10 V measuring range)

The DB number is permanently set to 2.

#### Structure of DB_CONFIG

| | 15 | 0 |
|---|---|---|
| DW 0 | (Reserved) | DP bus address |
| DW 1 | (Reserved) | Priority of DP task |
| DW 2 | (Reserved) | Pause time of DP task |
| DW 3 | DP status | |
| DW 4 | DP reason (extra info on DP status) | |
| DW 5 | Configuration of analog input AI1 | |
| DW 6 | Configuration of analog input AI2 | |
| DW 7 | Configuration of analog input AI3 | |
| DW 8 | Configuration of analog input AI4 | |
| DW 9 | Configuration of analog input AI5 | |
| DW 10 | Configuration of analog input AI6 | |
| DW 11 | Configuration of analog input AI7 | |
| DW 12 | Configuration of analog input AI8 | |

#### DL 0

Specification of the DP slave address. Values from 1 to 127 can be used. Enter 0 if no DP slave is to be configured.

#### DL 1, DL_2

The DP is processed in a system task. This task processes the DP slave cyclically in an endless loop. A pause is provided after each pass to give lower-priority tasks computing time.

The priority of the DP task and the pause time in msec can be set in DL1 and DL2. Standard values are set if you enter 0 (the default value). This permits you to optimize the entire system.

**Note:**
Although the entire system can be optimized, we strongly recommend leaving the standard settings (default value 0) as is. Incorrect settings will endanger the stability of the system.

## DW 3

Return messages of the DP slave. The current status of the DP instance is returned here. The values are identical to the return message of the `dp_Init` function. See chapter 6.12.7.

## DW 4

The reason for the status message in DW 3 is entered here. The values are identical to the return message of the `dp_Init` function. See chapter 6.12.7.

## DW 5 to DW 12

The analog inputs are configured here. The setting is not valid unless the jumper is inserted on "voltage measurement." With current measurement, the measuring range is permanently set to 0 to 20 mA. With temperature measurement, the measuring range is permanently set to 0.1 degrees.

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | | Bipolar | 5 V |
| | | | | | | | | | | | | | | Bip = 1<br>Uni = 0 | 5V = 1<br>10V = 0 |

## Examples

- AQ1 = 0 to 10 V, unipolar → DW5 = 0x0000
- AQ2 = 0 to 5 V, unipolar → DW6 = 0x0001
- AQ3 = -10 to 10 V bipolar → DW7 = 0x0002
- AQ4 = -5 to 5 V, bipolar → DW8 = 0x0003

# 6 BIOS Functions

The IMC1BIOS.LIB library file of the IMC01-BSP contains functions which allow you to access the hardware of the device.

**Note:**
Do not confuse the **basic input-output system** of the IMC01 with the boot BIOS of a PC.

The following description of these functions is organized by groups of hardware to be processed.


| Function Groups | Call [1)] | Page |
|---|---|---|
| Functions for Analog Input | bai_... | 63 |
| Functions for Analog Output | bao_... | 72 |
| Real Time Clock Functions | bc_... | 75 |
| Functions for Digital Input | bdi_... | 79 |
| Functions for Digital Output | bdo_... | 87 |
| Display Functions | bg_... | 96 |
| General Functions | bgen_... | 105 |
| Keyboard Functions | bk_... | 109 |
| Functions for Temperature Measurement | bpt_... | 112 |
| Functions for Watchdog Processing | bwd_... | 115 |
| Functions for DP Slave Programming | dp_... | 117 |
| Functions of the Hardware Configuration | imc01... | 123 |


*1) ... = function-specific name*

The examples in the descriptions are only provided for better comprehension of the particular function. Subdirectory SYSIMC1\APLBIOS offers an executable example.

**Note:**
Interface compatibility with the IMC05 requires fixed parameters for the IMC01 for some of the BIOS functions. For example, CHARACTER must always be entered for the `coord_class` parameter of the `bg_gotoxy` function.

## 6.1 Call Back Functions

When called, various initialization functions are provided with a pointer to a user function. This so-called call back function is then called by the applicable RMOS interrupt routine when a certain event occurs.

A call back function does not contain transfer parameters and must be programmed so that it can be executed within the time available to the interrupt routine. If necessary, extensive processing must be transferred to an additional processing function in S status, or to a task.

**Note:**
Identical type BIOS routines generally do not have "reentrat" capability. For more details, see the READIMC1.1P0 file.

A call back function is specified for the following initialization functions.

Table 6.1 Initialization functions with transfer of a call back function

| Call | Function | Event | RMOS Proc. Status |
|---|---|---|---|
| bk_keybinit | Initialize keyboard | Key control (i.e., press and release) | S |
| bai_strtconv | Start AD conversion of an analog input | After conclusion of AD conversion | S |

If you transfer the value 0 instead of a pointer, no user function is performed.

**Note:**
When a dynamic task is deleted from work memory, all initialization functions should be called with NULL pointers at the end to disable the appropriate interrupts.

## 6.2 Functions for Analog Input

Table 6.2 Functions for analog input

| Call | Function | Page |
|---|---|---|
| bai_get | Read last conversion result of an analog input | 63 |
| bai_getmode | Read configuration of an analog input | 65 |
| bai_init | Initialize analog inputs and set voltage ranges | 66 |
| bai_read | Start AD conversion and read result in a function call | 67 |
| bai_setmode | Set operating mode of the analog inputs | 69 |
| bai_strtconv | Start AD conversion of an analog input | 70 |

### 6.2.1 bai_get

**Function**

Read result of the last AD conversion started with `bai_strtconv` for the specified analog input

**Syntax**

```
#include <imc01bas.h>
unsigned int bai_get(unsigned int number);
```

**Parameters**

| Parameter Name | Meaning |
|---|---|
| `number` | Number of the analog input to be read<br>0 to 3: Analog inputs 1 to 4 on I/O card<br>4 to 7: Used internally<br>8 to 11: Analog inputs 5 to 8 on expansion card<br>12 to 15:Used internally |

**Return value**

| Function Value | Meaning |
|---|---|
| 0xFFFF | Invalid |
| Other | Result of the AD conversion |

**Description**

The function reads the result of the last AD conversion of the specified analog input. It should be called in the call back function specified when `bai_strtconv` was called.

The conversion result has a resolution of 12 bits.

Conversion result for the "voltage measurement" jumper setting

Setting 0 to +10 V
0x0000 0 Volt
0x0FFF +10 Volt

Setting 0 to +5 V
0x0000 0 Volt
0x0FFF +5 Volt

Setting –10 V to +10 V
0x0000 0 Volt
0x07FF +10 Volt
0x0800 –10 Volt

Setting –5 V to +5 V
0x0000 0 Volt
0x07FF +5 Volt
0x0800 –5 Volt

Conversion result for the "current measurement" jumper setting
0x0000 0 mA
0x0CCC +20 mA

**Note:**
When the "temperature measurement" jumper setting is used, inputs AI3, AI4, AI7 and AI8 must be read by the `bpt_`... BIOS functions. See chapter 6.10.

**Example**

```
#include <imc01bas.h>

unsigned int value;

void testfunction(void)
{
    value = bai_get(1);
}
```

**See also** bai_strtconv

## 6.2.2 bai_getmode

**Function** Read configuration (jumper setting) of an analog input

**Syntax**

```
#include <imc01bas.h>
unsigned char  bai_getmode(unsigned int number);
```

**Parameters**

| Parameter Name | Meaning |
|---|---|
| number | Number of the analog input<br>0 to 3: Analog inputs 1 to 4 on I/O card<br>8 to 11: Analog inputs 5 to 8 on expansion card |

**Return value**

| Function Value | Meaning |
|---|---|
| 0 | Jumper is set to voltage measurement. |
| 0xFF | Jumper is set to current measurement (AI1, AI2, AI5 and AI6) or temperature measurement (AI3, AI4, AI7 and AI8). |

**Description** The function reads the position of the plug-in jumper with which the specified analog input is set to voltage or current or temperature measurement.

**Example**

```
#include <imc01bas.h>

void testfunction(void)
{
    int muster = 0;

    for (i=0;i<16;i++)
        {
        if (bai_getmode (i))
            muster |= 1<<i;
        else
            muster &= ~(1<<i);
        }

    printf ("AI-Konfigmuster = %x\n", muster);
}
```

**See also** bai_read, bai_get, bai_strtconv, bpt_...

## 6.2.3 bai_init

**Function**

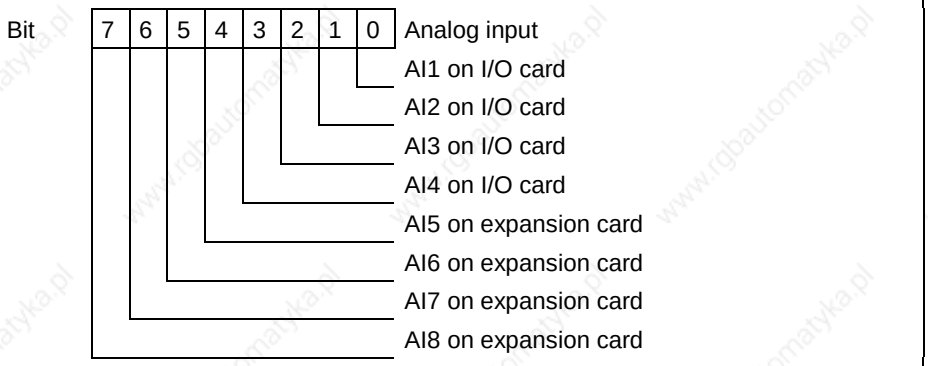
Initialize analog inputs and set voltage ranges

**Syntax**

```
#include <imc01bas.h>
void bai_init(unsigned char volt_range_bitmsk);
```

**Parameters**

| Parameter Name | Meaning |
|---|---|
| `volt_range_bitmsk` | Bit pattern for setting the voltage range of analog inputs<br>Bit = 0: $U_{max}$ = 10 V<br>Bit = 1: $U_{max}$ = 5 V |




**Return value**

None

**Description**

The routine initializes processing of the analog inputs and assigns 0xFFFF (invalid) to their conversion results. The voltage range is set for the inputs set to voltage measurement.

BIOS calls the routine during startup, and sets all voltage ranges to 10 V.

The `bai_setmode` function sets the operating mode (unipolar/bipolar).

Setting of the voltage range is irrelevant to inputs which are set to current or temperature measurement.

**Example**

```
#include <imc01bas.h>

void testfunction(void)
{
    unsigned char voltrange_bitmask = 0; /* range 10 V */
    bai_init(voltrange_bitmask);
}
```

**See also**

bai_setmode, bai_get, bai_strtconv,

### 6.2.4 bai_read

**Function**

Start AD conversion of an analog input, wait for conclusion of AD conversion, and read result

**Syntax**

```
#include <imc01bas.h>
unsigned int bai_get(unsigned int number);
```

**Parameters**

| Parameter Name | Meaning |
|---|---|
| number | Number of the analog input to be converted<br>0 to 3: Analog inputs 1 to 4 on I/O card<br>8 to 11: Analog inputs 5 to 8 on expansion card |

**Return value**

| Function Value | Meaning |
|---|---|
| 0xFFFF | Invalid |
| Other | Result of the AD conversion |

**Description**

The function starts the AD conversion of an analog input, waits until it has been concluded, and reads the conversion result. In comparison to the `bai_strtconv` and `bai_get` functions, this is an easier but slower method of reading an analog input.

Conversion result for the "voltage measurement" jumper setting

Setting 0 to +10 V
0x0000 0 Volt
0x0FFF +10 Volt

Setting 0 to +5 V
0x0000 0 Volt
0x0FFF +5 Volt

Setting –10 V to +10 V
0x0000 0 Volt
0x07FF +10 Volt
0x0800 –10 Volt

Setting –5 V to +5 V
0x0000 0 Volt
0x07FF +5 Volt
0x0800 –5 Volt

Conversion result for the "current measurement" jumper setting
0x0000 0 mA
0x0CCC +20 mA

**Note:**
Mixed use of `bai_read` and `bai_strtconv`/`bai_get` is permitted but must be decoupled in time by the user.

**Example**

```
#include <imc01bas.h>

unsigned int value 1;

unsigned int value 2;

void testfunction(void)
{
    value 1 = bai_read(0); /* convert ADC channel 1, read value*/
  value 2 = bai_read(1); /* convert ADC channel 2, read value*/

}
```

**See also** bai_get, bai_strtconv, bpt_read

## 6.2.5 bai_setmode

**Function**

Configuring analog inputs for functions `bai_strtconv` and `bai_get`

**Syntax**

```
#include <imc01bas.h>
int bai_setmode(unsigned int number,
                unsigned char mode);
```

**Parameters**

| Parameter Name | Meaning |
|---|---|
| `number` | Number of the input to be configured<br>0 to 3: Analog inputs 1 to 4 on I/O card<br>8 to 11: Analog inputs 5 to 8 on expansion card |
| `mode` | Operating mode of the analog input<br>AI_BIPOLAR Bipolar mode<br>AI_UNIPOLAR: Unipolar mode |

**Return value**

| Function Value | Meaning |
|---|---|
| BAI_NO_ERROR | Function executed successfully. |
| BAI_INV_PARAM | Parameter invalid |

**Description**

The function sets the operating mode of an analog input.

Bipolar mode $-U_{max}$ to $+U_{max}$

Unipolar mode 0 to $+U_{max}$

The function is called by BIOS during startup and sets all inputs to unipolar mode.

Function `bai_init` sets the voltage range $U_{max}$.

**Example**

```
#include <imc01bas.h>

void testfunction(void)
{
    bai_setmode (0, AI_BIPOLAR);
}
```

**See also**

bai_init

## 6.2.6 bai_strtconv

**Function**

Start AD conversion of an analog input

**Syntax**

```
#include <imc01bas.h>
int bai_strtconv(unsigned int number,
                 void (*call_back_fct)());
```

**Parameters**

| Parameter Name | Meaning |
|---|---|
| number | Number of the analog input to be converted<br>0 to 3: Analog inputs 1 to 4 on I/O card<br>4 to 7: Used internally<br>8 to 11: Analog inputs 5 to 8 on expansion card<br>12 to 15:Used internally |
| *call_back_fct | Pointer to function which is called after AD conversion has been concluded |

**Return value**

| Function Value | Meaning |
|---|---|
| 0 | Function executed successfully |
| –1 | Invalid parameter (number too large) |
| –2 | Call not permitted since an AD conversion is still in progress |
| –3 | Analog inputs in shutdown mode |

**Description**

The function starts the AD conversion of an analog input.

When the AD conversion has been completed, the call back function specified is called. The converted analog value should be read there. See bai_get function.

**Note:**
The call back function must be specified. A new conversion cannot be started until the call back function has been executed.

The bai_strtconv function is required when bai_get is used. When bai_read is used, bai_strtconv may not be called.

Since inputs AI3,4,7,and 8 must be read out by the bpt_... BIOS functions when the "temperature" jumper setting is used, bai_strtconv may not be called for temperature channels. When mixed operation (temperature inputs and voltage/current) is used, make sure that more than one conversion is never triggered simultaneously. For example, bpt_read may not be called for a temperature channel until a conversion (of a current/voltage channel) started with bai_strtconv has been concluded.

To trigger the conversion of several analog inputs at the same time, set bit 15 in `number`. This starts the AD conversion of all inputs from 0 to the input number specified.

Example: `number` = 0x8003: inputs 0 to 3

The call back function is not called until all conversions have been completed.

**Example**

```
#include <imc01bas.h>

extern _FIXED _FAR void test_fct(void);

void testfunction(void)
{
    bai_strtconv(1,test_fct);
}
```

**See also** bai_get, bai_read

## 6.3 Functions for Analog Output

Table 6.3 Functions for analog output

| Call | Function | Page |
|---|---|---|
| bao_init | Initialize analog output | 72 |
| bao_set | Output value to analog output | 73 |
| bao_setmode | Configure analog output (with IMC01 dummy for compatibility with IMC05) | 74 |

### 6.3.1 bao_init

**Function** Initialize analog output

**Syntax**

```
#include <imc01bas.h>
void bao_init();
```

**Return value** None

**Description** The routine initializes processing of the analog output and resets this to 0 Volt.

The routine is called by BIOS during startup. It does not need to be called by the user.

### 6.3.2 bao_set

**Function**

Output value to analog output

**Syntax**

```
#include <imc01bas.h>
unsigned char bao_set(unsigned int number,
                      unsigned int value);
```

**Parameters**

| Parameter Name | Meaning |
|---|---|
| `number` | Number of the analog output (always 0) |
| `value` | Value to be output |

**Return value**

| Function Value | Meaning |
|---|---|
| 0xFF | Output performed successfully |
| 0 | Output not performed (`value` or `number` too large or output deactivated) |

**Description**

The function writes a value to the analog output specified.

00 Approx. 0 mA
1000 (0x3e8) Approx. 20 mA

Only 0 is permitted as the output number since only one output is available with IMC01.

**Example**

```
#include <imc01bas.h>

void testfunction(void)
{
    bao_init();
    bao_set(0, 1000);    /* output 20 mA */
}
```

### 6.3.3 bao_setmode

**Function**

Dummy function for IMC01. The output must always be set permanently to 0 to 20 mA and cannot be configured.

**Syntax**

```
#include <imc01bas.h>
int bao_setmode(unsigned int number,
                unsigned char mode);
```

**Parameters**

| Parameter Name | Meaning |
|---|---|
| number | Dummy |
| mode | Dummy |

**Return value**

| Function Value | Meaning |
|---|---|
| BAO_NO_ERROR | Function executed successfully |

**Description**

This is a dummy function for IMC01. It is included for interface compatibility with the IMC05 and should not be used for the IMC01.
The function sets the operating mode of the analog outputs for the IMC05.

## 6.4 Real Time Clock Functions

Table 6.4 Real time clock functions

| Call | Function | Page |
|---|---|---|
| bc_init_rtc | Initialize real time clock | 75 |
| bc_stime | Set date and time | 76 |
| bc_time | Read date and time | 77 |

### 6.4.1 bc_init_rtc

**Function** Initialize real time clock

**Syntax**

```
#include <imc01bas.h>
void bc_init_rtc(void);
```

**Return value** None

**Description** The routine starts processing of the real time clock. This does not change the previously valid date and time.

The routine is called by BIOS during startup. It does not need to be called by the user.

**Example**

```
#include <imc01bas.h>

void testfunction(void)
{
    bc_init_rtc();
}
```

**See also** bc_stime, bc_time, RmSetHWClockTime, RmGetHWClockTime

## 6.4.2 bc_stime

**Function** Set date and time

**Syntax**

```
#include <imc01bas.h>
void bc_stime(RTC_TIME *time,
              RTC_DATE *date,
              unsigned char day);
```

**Parameters**

| Parameter Name | Meaning |
|---|---|
| *time | Pointer to structure with the time |
| *date | Pointer to structure with the date |
| day | Day of the week |

**Return value** None

**Description** This routine sets the time, date and day of the week for the real time clock. The structures for the values are described under the routine bc_time in chapter 6.4.3.

If one of the pointers contains the value NULL or a value is invalid, the real time clock is not set with this value.

**Example**

```
#include <imc01bas.h>

void testfunction(void)
{
    RTC_TIME time;
    RTC_DATE date;

/* Set time to Monday, 18:02:00 */

    time.H1 = 8;
    time.H10 = 1;
    time.M1 = 2;
    time.M10 = 0;
    time.S1 = 0;
    time.S10 = 0;
    bc_stime(&time, NULL, MONDAY);
}
```

**See also** bc_time

### 6.4.3 bc_time

**Function**

Read date and time

**Syntax**

```
#include <imc01bas.h>
void bc_time(RTC_TIME *time,
             RTC_DATE *date,
             unsigned char *day);
```

**Parameters**

| Parameter Name | Meaning |
|---|---|
| *time | Pointer to structure for the time |
| *date | Pointer to structure for the date |
| *day | Pointer to a byte for the day of the week |

**Return value**

None

**Description**

This routine reads the time, date and the day of the week from the real time clock.

The time is stored in the following structure.

| Structure Element | Type | Meaning | Values |
|---|---|---|---|
| .S1 | unsigned char | Seconds, ones position | 0 to 9 |
| .S10 | unsigned char | Seconds, tens position | 0 to 5 |
| .M1 | unsigned char | Minutes, ones position | 0 to 9 |
| .M10 | unsigned char | Minutes, tens position | 0 to 5 |
| .H1 | unsigned char | Hour, ones position | 0 to 9 |
| .H10 | unsigned char | Hour, tens position | 0 to 2 |

The date is stored in the following structure.

| Structure Element | Type | Meaning | Values |
|---|---|---|---|
| .D1 | unsigned char | Day, ones position | 0 to 9 |
| .D10 | unsigned char | Day, tens position | 0 to 3 |
| .M1 | unsigned char | Month, ones position | 0 to 9 |
| .M10 | unsigned char | Month, tens position | 0 to 1 |
| .Y1 | unsigned char | Year, ones position | 0 to 9 |
| .Y10 | unsigned char | Year, tens position | 0 to 9 |

The day of the week is stored in the following structure.

| Constant | Meaning | Value |
|---|---|---|
| SUNDAY | Sunday | 0 |
| MONDAY | Monday | 1 |
| TUESDAY | Tuesday | 2 |
| WEDNESDAY | Wednesday | 3 |
| THURSDAY | Thursday | 4 |
| FRIDAY | Friday | 5 |
| SATURDAY | Saturday | 6 |

**Example**

```
#include <imc01bas.h>

void testfunction(void)
{
    RTC_TIME time;
    RTC_DATE date;
    unsigned char dayoftheweek;

    bc_time(&time,&date, &dayoftheweek);
}
```

**See also** bc_stime

## 6.5 Functions for Digital Input

Table 6.5 Functions for digital input (DI = digital Input)

| Call | Function | Page |
|---|---|---|
| bdi_get | Read digital input | 80 |
| bdi_getblock | Read block of digital inputs | 81 |
| bdi_getmbit | Read bit from DI image | 82 |
| bdi_getmirror | Read DI image | 83 |
| bdi_init | Initialize digital inputs | 84 |
| bdi_mirror | Update DI image | 84 |
| bdi_resetmbit | Reset bit in DI image | 85 |
| bdi_setmbit | Set bit in DI image | 85 |
| bdi_setmirror | Overwrite DI image | 86 |

The following diagram shows the accesses which can be performed with these functions and the use of the image of the digital inputs (DI image).

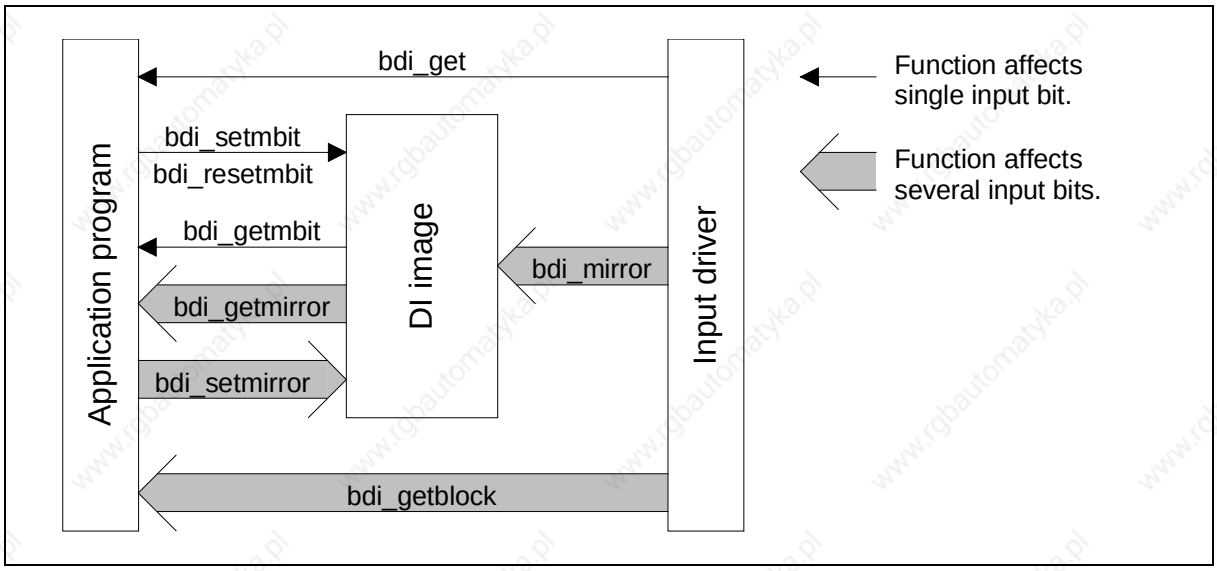



Figure 6.1 Use of the DI image

The DI image can be used for test purposes (e.g., to simulate I/O devices which have not been connected yet).

### 6.5.1 bdi_get

**Function** Read digital input

**Syntax**

```
#include <imc01bas.h>
unsigned char bdi_get(unsigned int di_num);
```

**Parameters**

| Parameter Name | Meaning |
|---|---|
| di_num | Number of the digital input to be read |

**Return value**

| Function Value | Meaning |
|---|---|
| 0 | Input is reset. |
| 0xFF | Input is set. |

**Description** The function reads the status of a digital input directly and bypasses the image of the digital inputs.

If di_num contains an invalid value, 0 is returned as the result.

**Example**

```
#include <imc01bas.h>

void testfunction(void)
{
    bg_gotoxy(0,1, CHARACTER);
    bg_outtext((bdi_get(3) == 0) ?
        ´DI = set´ : ´DI = reset´);
}
```

**See also** bdi_getmirror

## 6.5.2 bdi_getblock

**Function**

Read block of digital inputs

**Syntax**

```
#include <imc01bas.h>
void bdi_getblock(unsigned int from_di,
                  unsigned int to_di,
                  unsigned short *dptr);
```

**Parameters**

| Parameter Name | Meaning |
|---|---|
| from_di | Number of the first digital input to be read |
| to_di | Number of the last digital input to be read |
| *dptr | Pointer to the result |

**Note:**
In principle, any values can be used for from_di and to_di.
However, using values for from_di and to_di which are divisible by 8 speeds up processing considerably.

**Return value**

None

**Description**

The routine reads a block of digital inputs and stores the result by bit in the integer variables referenced.

Bit 0: DI0
Bit 23: DI23

The image of the digital inputs is not affected.

**Example**

```
#include <imc01bas.h>

void testfunction(void)
{
    int de_abbild;    /* image for max. of 24 dig. inputs */

    bdi_getblock(0, 23, (unsigned short *) &de_abbild);
            /* read inputs 0 to 23 */
}
```

**See also**

bdi_get, bdi_mirror

### 6.5.3 bdi_getmbit

**Function** Read bit from DI image

**Syntax**

```
#include <imc01bas.h>
unsigned char bdi_getmbit(unsigned int di_num);
```

**Parameters**

| Parameter Name | Meaning |
|---|---|
| di_num | Number of the bit to be read |

**Return value**

| Function Value | Meaning |
|---|---|
| 0 | Bit is reset. |
| 0xFF | Bit is set. |

**Description** The function reads a bit from the image of the digital inputs. If `di_num` contains an invalid value, 0 is returned as the result.

**Example**

```
#include <imc01bas.h>

void testfunction(void)
{
    bdi_mirror();
    bit_1 = bdi_getmbit(1);
    bdi_resetmbit(3);
    bit_3 = bdi_getmbit(3);
    bdi_setmbit(3);
    bit_3_ = bdi_getmbit(3);

    bg_gotoxy(0, 0, CHARACTER);
    bg_outtext(bit_1  ? " set" : "reset");
}
```

**See also** bdi_setmbit

### 6.5.4 bdi_getmirror

**Function** Read DI image

**Syntax**

```
#include <imc01bas.h>
unsigned int bdi_getmirror(unsigned char *pdi,
                           unsigned int bitcnt);
```

**Parameters**

| Parameter Name | Meaning |
|---|---|
| *pdi | Pointer to variable in which the result is stored |
| bitcnt | Number of bits to be processed |

**Return value** Number of bits which were processed

**Description** The function reads the first bitcnt bits from the image of the digital inputs and writes the result in the character string referenced.

Only the maximum number of bits actually contained in the image of the digital inputs is stored.

**Example**

```
#include <stdlib.h>
#include <imc01bas.h>

void testfunction(void)
{
    unsigned int pattern = {0x12};
    unsigned int result;
    unsigned int number;

    number = bdi_init();
    bdi_setmirror(&pattern, 16);
    number = bdi_getmirror(&result, 16);

    bg_gotoxy(0,1,CHARACTER);
    bg_outtext(itoa(number));
}
```

**See also** bdi_mirror, bdi_setmirror

### 6.5.5 bdi_init

**Function** Initialize digital inputs

**Syntax**
```
#include <imc01bas.h>
unsigned int bdi_init();
```

**Return value** Number of digital inputs available

**Description** The function resets all digital inputs and initializes their processing.

The function is called by BIOS during startup. It does not need to be called by the user.

**Example**
```
#include <stdlib.h>
#include <imc01bas.h>

void testfunction(void)
{
    unsigned int number;

    number = bdi_init();
    bg_gotoxy(0,1,CHARACTER);
    bg_outtext(itoa(number));
}
```

### 6.5.6 bdi_mirror

**Function** Update DI image

**Syntax**
```
#include <imc01bas.h>
void bdi_mirror();
```

**Return value** None

**Description** The routine reads all digital inputs and updates the internal BIOS DI image.

**See also** bdi_get, bdi_getmbit, bdi_setmbit, bdi_resetmbit

### 6.5.7 bdi_resetmbit

**Function** Reset bit in DI image

**Syntax**

```
#include <imc01bas.h>
void bdi_resetmbit(unsigned int di_num);
```

**Parameters**

| Parameter Name | Meaning |
|---|---|
| di_num | Number of the bit to be reset |

**Return value** None

**Description** The routine resets the specified bit in the image of the digital inputs. If `di_num` contains an invalid value, processing is not performed.

**See also** bdi_get, bdi_getmbit, bdi_resetmbit

### 6.5.8 bdi_setmbit

**Function** Set bit in DI image

**Syntax**

```
#include <imc01bas.h>
void bdi_setmbit(unsigned int di_num);
```

**Parameters**

| Parameter Name | Meaning |
|---|---|
| di_num | Number of the bit to be set |

**Return value** None

**Description** The routine sets the specified bit in the image of the digital inputs. If `di_num` contains an invalid value, processing is not performed.

**See also** bdi_get, bdi_getmbit, bdi_resetmbit

### 6.5.9 bdi_setmirror

**Function** Overwrite DI image

**Syntax**

```
#include <imc01bas.h>
void bdi_setmirror(unsigned char *pdi,
                   unsigned int bitcnt);
```

**Parameters**

| Parameter Name | Meaning |
|---|---|
| *pdi | Pointer to variable containing the bit pattern to be written |
| bitcnt | Number of bits to be processed |

**Return value** None

**Description** The routine overwrites the first bitcnt bits in the image of the digital inputs with the bit pattern specified.

**Example**

```
#include <imc01bas.h>

static unsigned char de_abbild[8];
        /* Digital inputs and outputs */

void testfunction(void)
{
    bdi_setmirror (&de_abbild[0], 1);
}
```

**See also** bdi_mirror, bdi_getmirror

## 6.6 Functions for Digital Output

Table 6.6 Functions for digital output (DO = digital output)

| Call | Function | Page |
|---|---|---|
| bdo_flush | Output block of digital outputs | 88 |
| bdo_getmbit | Read bit from DO image | 89 |
| bdo_getmirror | Read DO image | 90 |
| bdo_init | Initialize digital outputs | 91 |
| bdo_mflush | Output DO image | 91 |
| bdo_reset | Reset digital output | 92 |
| bdo_resetmbit | Reset bit in DO image | 92 |
| bdo_set | Set digital output | 93 |
| bdo_setmbit | Set bit in DO image | 93 |
| bdo_setmirror | Overwrite DO image | 94 |
| bdo_status | Read group error status of the output drivers | 95 |

The output drivers are always processed in groups of 8 outputs. To circumvent this restriction, BIOS maintains an image of the digital outputs which you can also access by bit.

The following diagram shows the accesses which can be performed with these functions and the use of the image of the digital outputs (DO image).

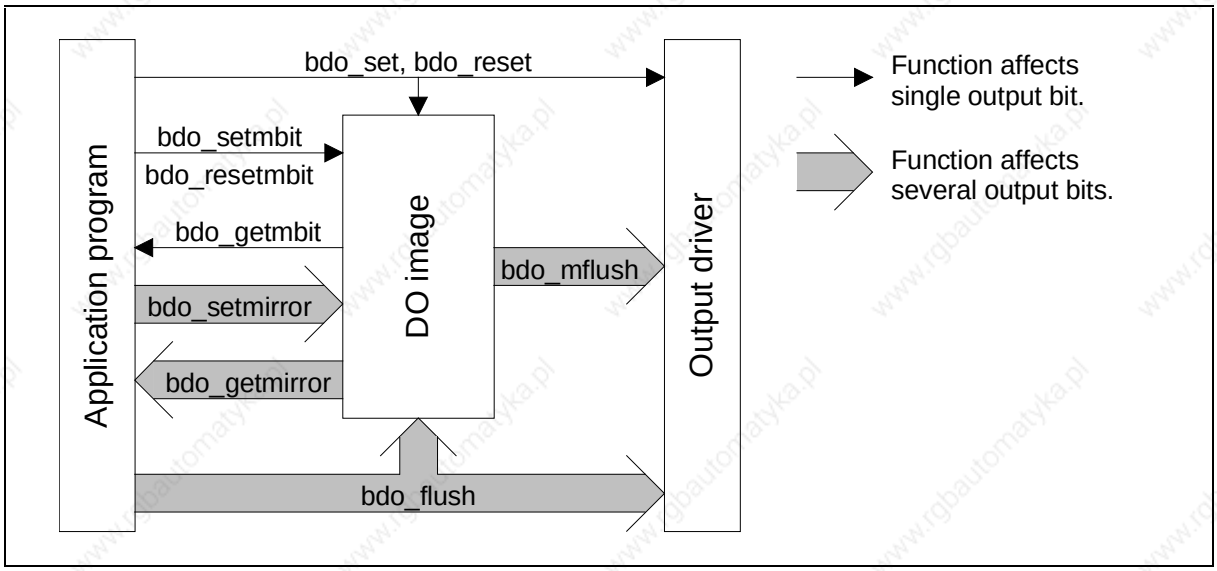



Figure 6.2 Use of the DO image

### 6.6.1 bdo_flush

**Function** Output block of digital outputs

**Syntax**

```
#include <imc01bas.h>
void bdo_flush(unsigned int from_do,
               unsigned int to_do,
               unsigned char *dptr);
```

**Parameters**

| Parameter Name | Meaning |
|---|---|
| from_do | Number of the first digital output |
| to_do | Number of the last digital output |
| *dptr | Pointer to the bit pattern to be written |

**Note:**
In principle, any values can be used for from_do and to_do.
However, using values for from_do and to_do which are divisible by 8 speeds up processing considerably.

**Return value** None

**Description** The routine outputs the referenced bit pattern to the specified block of digital outputs.

Bit 0: DO0
Bit 15: DO15

**Example**

```
#include <imc01bas.h>

void testfunction(void)
{
    int do_mask = 0x15;
        /* specify bit mask for outputs */
    bdo_flush(6, 10, &do_mask);
        /* Set DO 6, 8, 10; reset DO 7, 9 */
}
```

**See also** bdo_set, bdo_reset, bdo_mflush

## 6.6.2 bdo_getmbit

**Function**

Read bit from DO image

**Syntax**

```
#include <imc01bas.h>
unsigned char bdo_getmbit(unsigned int do_num);
```

**Parameters**

| Parameter Name | Meaning |
|---|---|
| do_num | Number of the bit to be read |

**Return value**

| Function Value | Meaning |
|---|---|
| 0 | Bit is reset. |
| 0xFF | Bit is set. |

**Description**

The function reads a bit from the image of the digital outputs. A result of 0 is returned when do_num contains an invalid value.

**Example**

```
#include <imc01bas.h>

void testfunction(void)
{
    bdo_setmbit(3);
    bit_3 = bdo_getmbit(3);

    bg_gotoxy(0,0,CHARACTER);
    bg_outtext(bit_3 ? "set" : "reset");
}
```

**See also**

bdo_setmbit

### 6.6.3 bdo_getmirror

**Function**

Read DO image

**Syntax**

```
#include <imc01bas.h>
unsigned int bdo_getmirror(unsigned char *pdo,
                           unsigned int bitcnt);
```

**Parameters**

| Parameter Name | Meaning |
|---|---|
| `*pdo` | Pointer to variable in which the result is stored |
| `bitcnt` | Number of bits to be processed |

**Return value**

Number of bits which were processed

**Description**

The function reads the first `bitcnt` bits from the image of the digital outputs and writes the result in the character string referenced.

Only the maximum number of bits actually contained in the image of the digital outputs is stored.

**Example**

```
#include <stdlib.h>
#include <imc01bas.h>

void testfunction(void)
{
    unsigned int pattern = {0x12};
    unsigned int result;
    unsigned int number;
    char temp_string [10];

    number = bdo_init();
    bdo_setmirror(&pattern, 16);
    number = bdo_getmirror(&result, 16);

    bg_gotoxy(0,1,CHARACTER);
    bg_outtext(itoa(number, temp_string, 10));
}
```

**See also**

bdo_mflush, bdo_setmirror

### 6.6.4 bdo_init

**Function** Initialize digital outputs

**Syntax**

```
#include <imc01bas.h>
unsigned int bdo_init(void (*call_back_fct)());
```

**Parameters**

| Parameter Name | Meaning |
|---|---|
| *call_back_fct | Pointer to call back function which is called when the group error status of the output drivers changes |

**Return value** Number of digital outputs available

**Description** The function initializes processing of the digital outputs by BIOS and installs the call back function for processing error interrupts.

All bits are reset in the image of the digital outputs (DO image).

The function is called by BIOS during startup (without pointer to a call back function). The function does not need to be called by the user unless the call back function is to be installed.

**Example**

```
#include <stdlib.h>
#include <imc01bas.h>

void testfunction(void)
{
    unsigned int number;

    number = bdo_init(NULL);
    bg_gotoxy(0,1,CHARACTER);
    bg_outtext(itoa(number));
}
```

**See also** bdo_status

### 6.6.5 bdo_mflush

**Function** Output DO image

**Syntax**

```
#include <imc01bas.h>
void bdo_mflush();
```

**Return value** None

**Description** The routine outputs the image of the digital outputs to the output drivers.

**See also** bdo_set

### 6.6.6 bdo_reset

**Function** Reset digital output

**Syntax**

```
#include <imc01bas.h>
void bdo_reset(unsigned int do_num);
```

**Parameters**

| Parameter Name | Meaning |
|---|---|
| do_num | Number of the digital output to be reset |

**Return value** None

**Description** The routine resets a bit in the DO image and outputs the corresponding group to the digital outputs. No processing is performed when do_num contains an invalid value.

**Example**

```
#include <imc01bas.h>

void testfunction(void)
{
    bdo_reset(2);
}
```

**See also** bdo_setmirror, bdo_setmbit

### 6.6.7 bdo_resetmbit

**Function** Reset bit in DO image

**Syntax**

```
#include <imc01bas.h>
void bdo_resetmbit(unsigned int do_num);
```

**Parameters**

| Parameter Name | Meaning |
|---|---|
| do_num | Number of the bit to be reset |

**Return value** None

**Description** The routine resets the specified bit in the image of the digital outputs. No processing is performed when do_num contains an invalid value.

**See also** bdo_setmbit, bdo_getmbit

### 6.6.8 bdo_set

**Function** Set digital output

**Syntax**

```
#include <imc01bas.h>
void bdo_set(unsigned int do_num);
```

**Parameters**

| Parameter Name | Meaning |
|---|---|
| do_num | Number of the digital output to be set |

**Return value** None

**Description** The routine sets a bit in the DO image and outputs the corresponding group to the digital outputs. No processing is performed when do_num contains an invalid value.

**Example**

```
#include <imc01bas.h>

void testfunction(void)
{
    bdo_set(2);
}
```

**See also** bdo_setmirror, bdo_setmbit

### 6.6.9 bdo_setmbit

**Function** Set bit in DO image

**Syntax**

```
#include <imc01bas.h>
void bdo_setmbit(unsigned int do_num);
```

**Parameters**

| Parameter Name | Meaning |
|---|---|
| do_num | Number of the bit to be set |

**Return value** None

**Description** The routine sets the specified bit in the image of the digital outputs. No processing is performed when do_num contains an invalid value.

**See also** bdo_set, bdo_getmbit

### 6.6.10 bdo_setmirror

**Function** Overwrite DO image

**Syntax**

```
#include <imc01bas.h>
void bdo_setmirror(unsigned char *pdo,
                   unsigned int bitcnt);
```

**Parameters**

| Parameter Name | Meaning |
|---|---|
| `*pdo` | Pointer to variable containing the bit pattern to be written |
| `bitcnt` | Number of bits to be processed |

**Return value** None

**Description** The routine overwrites the first `bitcnt` bits of the image of the digital outputs with the bit pattern transferred.

**Example**

```
#include <imc01bas.h>

static unsigned char do_mirror[2];

void testfunction(void)
{
    bdo_setmirror (&do_mirror[0], 1);
}
```

**See also** bdo_mflush, bdo_getmirror

## 6.6.11 bdo_status

**Function**

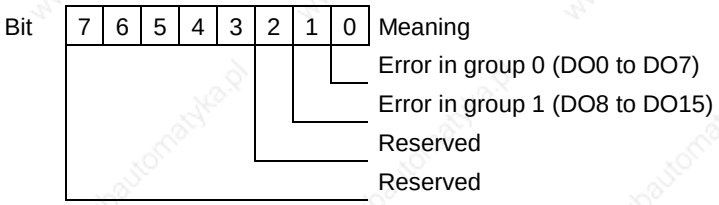
Read group error status of the output drivers

**Syntax**

```
#include <imc01bas.h>
unsigned char bdo_status();
```

**Return value**

When one of the following bits is set (= 1) in the return value, this indicates that an error has occurred.




**Description**

This function determines the group error status of the output drivers of all digital outputs.

The error bit of an output group is set under the following conditions.

- Overcurrent (short circuit) or thermal overload
- Undervoltage on one of the outputs

For example, the function can be used in the call back function of the `bdo_init` function.

**Example**

```
#include <imc01bas.h>

void testfunction(void)
{
    unsigned char result;

    result = bdo_status();
}
```

**See also**

bdo_init

# 6.7 Display Functions

Table 6.7 Display functions

| Call | Function | Page |
|---|---|---|
| bg_cleardevice | Clear display | 96 |
| bg_getmaxx | x dimension of display | 97 |
| bg_getmaxy | y dimension of display | 97 |
| bg_gotoxy | Position cursor (cursor becomes invisible) | 98 |
| bg_init | Initialize display | 99 |
| bg_outchar | Output character | 99 |
| bg_outtext | Output character string | 100 |
| bg_screensaver | Screen saver | 101 |
| bg_setcursor | Position cursor (cursor attributes are retained) | 102 |
| bg_setcursortype | Specify cursor attributes | 103 |
| bg_wherex | x position of text cursor | 104 |
| bg_wherey | y position of text cursor | 104 |

## 6.7.1 bg_cleardevice

**Function** Clear display

**Syntax**

```
#include <imc01bas.h>
void bg_cleardevice();
```

**Return value** None

**Description** After deletion, blanks are output in all text positions. The cursor position is shifted to 0, 0.

**Example**

```
#include <imc01bas.h>

void testfunction(void)
{
    bg_cleardevice();
}
```

**See also** bg_setcolor

### 6.7.2 bg_getmaxx

**Function** Determine maximum x dimension of the display (in characters)

**Syntax**
```
#include <imc01bas.h>
int bg_getmaxx();
```

**Return value** Integer value, depends on the hardware used:
currently always 16

**Example**
```
#include <imc01bas.h>

void testfunction(void)
{
    char temp_string [10];

    bg_cleardevice();
    bg_outtext(itoa(bg_getmaxx(), temp_string, 10));
}
```

**See also** bg_getmaxy

### 6.7.3 bg_getmaxy

**Function** Determine maximum y dimension of the display (in characters)

**Syntax**
```
#include <imc01bas.h>
int bg_getmaxy();
```

**Return value** Integer value, depends on the hardware used:
to date, always 4

**Example**
```
#include <imc01bas.h>

void testfunction(void)
{
    char temp_string [10];

    bg_cleardevice();
    bg_outtext(itoa(bg_getmaxy(), temp_string, 10));
}
```

**See also** bg_getmaxx

### 6.7.4 bg_gotoxy

**Function**

Position text cursor

**Syntax**

```
#include <imc01bas.h>
unsigned char bg_gotoxy(unsigned int x_pos,
                        unsigned int y_pos,
                        unsigned int coord_class);
```

**Parameters**

| Parameter Name | Meaning |
|---|---|
| x_pos | x position of the text cursor (0 = left) |
| y_pos | y position of the text cursor (0 = top) |
| coord_class | Type of position specified<br>CHARACTER Character position |

**Return value**

| Function Value | Meaning |
|---|---|
| 0xFF | Positioning performed successfully |
| 0 | No positioning performed |

**Description**

The function positions the cursor for the text output but does not indicate it at its new position. Positioning is not performed when an invalid position is specified.

When the IMC01 is used, only CHARACTER may be specified for coord_class.

**Example**

```
#include <ctype.h>
#include <imc01bas.h>

void testfunction(void)
{
    int length, i;
    char *string = "TOLOWER";
    length = strlen(string);
    for (i=0; i<length-1; i++)
        string[i] = tolower(string [i]);
    bg_gotoxy(0, 1, CHARACTER);
    bg_outtext(string);
}
```

**See also**

bg_setcursor, bg_setcursortype

### 6.7.5 bg_init

**Function** Initialize display

**Syntax**
```
#include <imc01bas.h>
void bg_init();
```

**Return value** None

**Description** This routine sets the display controller with the standard values.

- Text attribute = BG_NORM
- Cursor attribute: BG_NOCURSOR and BG_AUTOCURSOR
- Display cleared

The routine is called by BIOS during startup. It does not need to be called by the user.

**Example**
```
#include <imc01bas.h>

void testfunction(void)
{
    bg_init();
    bg_outtext("Initialization performed");
}
```

### 6.7.6 bg_outchar

**Function** Output character

**Syntax**
```
#include <imc01bas.h>
void bg_outchar(char _character);
```

**Parameters**

| Parameter Name | Meaning |
|---|---|
| _character | Character to be output |

**Return value** None

**Description** The routine outputs the character at the position of the text cursor.

The text cursor is shifted to the right of the output text.

**Example**
```
#include <imc01bas.h>

void testfunction(void)
{
    bg_cleardevice();
    bg_gotoxy(0,1, CHARACTER);
    bg_outchar(´A´);
    bg_outchar(´B´);
}
```

**See also** bg_outtext, bg_settextattr, bg_setcursortype,

### 6.7.7 bg_outtext

**Function** Output character string

**Syntax**

```
#include <imc01bas.h>
void bg_outtext(char *stringptr);
```

**Parameters**

| Parameter Name | Meaning |
|---|---|
| *stringptr | Pointer to character string to be output |

**Return value** None

**Description** The routine outputs the character string referenced by the pointer at the position of the text cursor.

The text cursor is shifted to the right of the output text.

**Example**

```
#include <imc01bas.h>

void testfunction(void)
{
    bg_cleardevice();
    bg_gotoxy(0,1, CHARACTER);
    bg_outtext("Output a");

    bg_gotoxy(0,2, CHARACTER);
    bg_outtext("character string");

}
```

**See also** bg_outchar, bg_setcursortype,

## 6.7.8 bg_screensaver

**Function** Screen saver

**Syntax**

```
#include <imc01bas.h>
void bg_screensaver(unsigned char state);
```

**Parameters**

| Parameter Name | Meaning |
|---|---|
| state | 0x01: Activate screen saver<br>0x00: Deactivate screen saver |

**Return value** None

**Description** The function turns the backlight of the display on or off. Correct use of this routine can extend the life of the backlight significantly.

**Note:**
Switching the backlight on and off too frequently damages the backlight just as much as leaving it on for unnecessarily long periods of time.

**Example**

```
#include <imc01bas.h>

void testfunction(void)
{
    bg_screensaver(0x01);
}
```

### 6.7.9 bg_setcursor

**Function** Indicate text cursor at new position

**Syntax**

```
#include <imc01bas.h>
unsigned char bg_setcursor(unsigned int x_pos,
                           unsigned int y_pos,
                           unsigned int coord_class);
```

**Parameters**

| Parameter Name | Meaning |
|---|---|
| x_pos | x position of the text cursor (0 = left) |
| y_pos | y position of the text cursor (0 = top) |
| coord_class | Type of position specified<br>CHARACTER Character position |

**Return value**

| Function Value | Meaning |
|---|---|
| 0xFF | Positioning performed successfully |
| 0 | No positioning performed |

**Description** The function positions the cursor for text output and indicates it at the new position with the cursor attributes set. Positioning is not performed when an invalid position is specified.

**Example**

```
#include <imc01bas.h>

void testfunction(void)
{
    unsigned char dummy;

    bg_gotoxy(0,2, CHARACTER);
    bg_outtext("Curspos. = 2,3");
    dummy = bg_setcursor(2,3,CHARACTER);
}
```

**See also** bg_gotoxy, bg_setcursortype

## 6.7.10 bg_setcursortype

**Function** Specify cursor attributes

**Syntax**

```
#include <imc01bas.h>
void bg_setcursortype(unsigned int cur_type,
                      unsigned int auto_cursor);
```

**Parameters**

| Parameter Name | Meaning |
|---|---|
| cur_type | Type of cursor indication<br>BG_NOCURSOR Cursor is invisible.<br>BG_BLINKCURSOR Flashing block cursor<br>BG_UNDERLINECURSOR Underline cursor |
| auto_cursor | BG_AUTOCURSOR_RIGHT<br>Cursor positioned to right next to output text |

**Return value** None

**Example**

```
#include <imc01bas.h>

void testfunction(void)
{
    bg_cleardevice();
    bg_setcursor(0,1, CHARACTER);
    bg_setcursortype(BG_BLINKCURSOR, BG_AUTOCURSOR_RIGHT);
}
```

**See also** bg_setcursor

## 6.7.11 bg_wherex

**Function** Determine current x position of the text cursor (in characters)

**Syntax**

```
#include <imc01bas.h>
int bg_wherex();
```

**Return value**

| Function Value | Meaning |
|---|---|
| 0 | Cursor on left edge of display |
| ... | |
| $x_{max}$ | Cursor on right edge of display |

**Example**

```
#include <imc01bas.h>

void testfunction(void)
{
    int x_pos;
    char temp_string [10];

    bg_cleardevice();
    bg_gotoxy(10, 3, CHARACTER);
    x_pos = bg_wherex();
    bg_outtext(itoa(x_pos, temp_string, 10));
}
```

**See also** bg_getmaxx, bg_wherey, bg_gotoxy

## 6.7.12 bg_wherey

**Function** Determine current y position of the text cursor (in characters)

**Syntax**

```
#include <imc01bas.h>
int bg_wherey();
```

**Return value**

| Function Value | Meaning |
|---|---|
| 0 | Cursor on top edge of display |
| ... | |
| $y_{max}$ | Cursor on bottom edge of display |

**Example**

```
#include <imc01bas.h>

void testfunction(void)
{
    int y_pos;
    char temp_string [10];

    bg_cleardevice();
    bg_gotoxy(0, 1, CHARACTER);
    y_pos = bg_wherey();
    bg_outtext(itoa(y_pos, temp_string, 10));
}
```

**See also** bg_getmaxy, bg_wherex, bg_gotoxy

## 6.8 General Functions

Table 6.8 General BIOS functions

| Call | Function | Page |
|---|---|---|
| bgen_battstat | Read battery status | 106 |
| bgen_crc | Generate CMOS checksum for a specified memory area | 107 |
| bgen_init | Initialize BIOS functions | 107 |
| bgen_version | Read BIOS version | 108 |

### 6.8.1 bgen_battstat

**Function**

Read battery status

**Syntax**

```
#include <imc01bas.h>
unsigned char bgen_battstat(unsigned int function);
```

**Parameters**

| Parameter Name | Meaning |
|---|---|
| function | Subfunction:<br>0 Battery not under load<br>1 Battery under load<br>2 Read battery status |

**Return value**

| Function Value | Meaning |
|---|---|
| 0xFF | Battery is okay. |
| 0 | Battery is defective. |

**Description**

The function switches test loading of the battery on or off and reads the battery status.

The battery must be under load prior to measurement. How long the battery must be under load depends on when the last measurement was made.

**Notes:**

- Batttery status can only be checked when the battery is under load.
- An erroneous measurement will result when the battery has not been under load for quite some time. Ensure that the battery is under load before testing it. How long the battery must be placed under load depends on when the battery was placed under load last. We recommend placing the battery under load once a day for 15 minutes and then checking the battery status at the end.

**Example**

```
#include <imc01bas.h>

void waittime(void)
{...
}

void testfunction(void)
{
    bgen_battstat(1)
    waittime ();
    if (!bgen_battstat(2))
        outtext("battery defective");
    bgen_battstat(0)
}
```

### 6.8.2 bgen_crc

**Function** Generate CMOS checksum for a specified memory area

**Syntax**

```
#include <imc01bas.h>
unsigned int bgen_crc(unisgned int init,
                      unsigned long linadr,
                      unsigned int lng);
```

**Parameters**

| Parameter Name | Meaning |
|---|---|
| init | Starting value for checksum calculation |
| linadr | Linear starting address for checksum calculation |
| lng | Area length in bytes |

**Return value** CRC checksum calculated

**Description** The function calculates the CRC checksum of the specified memory area and adds it to the starting value.

Start addresses (see Figure 2.1)

**Example**

```
#include <imc01bas.h>

void testfunction(void)
{
    unsigned int crc;
        /* Checksum of 1-Mbyte Flash memory */
    crc = bgen_crc(0, 0x3e00000, 0x100000);
}
```

### 6.8.3 bgen_init

**Function** Initialize general BIOS functions

**Syntax**

```
#include <imc01bas.h>
void bgen_init();
```

**Return value** None

**Description** The system already executes this initialization routine for general BIOS functions during startup. It does not need to be called by the user.

**Example**

```
#include <imc01bas.h>

void testfunction(void)
{
    bgen_init();
}
```

### 6.8.4 bgen_version

**Function** Read BIOS version

**Syntax**

```
#include <imc01bas.h>
unsigned char *bgen_version();
```

**Return value** Pointer to a string containing the version of BIOS (in accordance with C convention with 0x00 as end identifier)

Layout: "V *xx.yy*"

**Example**

```
#include <imc01bas.h>

void testfunction(void)
{
    bg_gotoxy(0,1, CHARACTER);
    bg_outtext(bgen_version());
}
```

## 6.9 Keyboard Functions

Table 6.9 Keyboard functions

| Call | Function | Page |
|---|---|---|
| bk_getchar | Read character from keyboard | 109 |
| bk_keybinit | Initialize keyboard | 110 |
| bk_LEDflush | Control keyboard LEDs | 111 |
| bk_setkeytab | Load new keyboard table | 111 |

### 6.9.1 bk_getchar

**Function** Read character from keyboard

**Syntax**

```
#include <imc01bas.h>
unsigned int bk_getchar();
```

**Return value** Character read in:

1st (low) byte: Character code

2nd (high) byte: State:
- 0xFF: Pressed
- 0: Released

**Description** This call can be used to read the last keyboard character recognized. The function should be used within the call back function (see initialization) if you do not want to lose key activations.

Constants for the key codes are defined in header file IMC01BAS.H.

**Example**

```
#include <imc01bas.h>

void testfunction(void);
{
    union
    {   int bios_char;
        struct
        { char code;
          unsigned char pressed;
        } character;
    } char_union;

    char_union.bios_char = bk_getchar();
    bg_gotoxy(0,0,CHARACTER);
    bg_outchar(char_union.character.code);
}
```

**See also** bk_setkeytab

## 6.9.2 bk_keybinit

**Function** Initialize keyboard

**Syntax**

```
#include <imc01bas.h>
void bk_keybinit(void (*call_back_fct)());
```

**Parameters**

| Parameter Name | Meaning |
|---|---|
| *call_back_fct | Pointer to function to be called when keyboard is activated. The call back function does not receive transfer parameters.<br>If you specify a NULL pointer, no function is called when the keyboard is activated. |

**Return value** None

**Description** The routine initializes keyboard processing and loads the standard keyboard table. It generates a task which cyclically reads the keyboard and calls the specified call back function when a change in status is detected.

BIOS keyboard processing only holds (internally) the last character pressed or released. This character can be read during the call back function with bk_getchar.

**Note:**
The task which processes the keyboard calls the call back function. To maintain interface compatibility with the IMC05, no parameter is provided to set the task priority. Although the system sets this to a fixed value of 10 for the IMC01, it can be changed later by the application with RMOS function RmSetTaskPriority. The task ID required by RmSetTaskPriority can be determined by the RmGetEntry function. File APLBIOS\FIRSTTSK.C contains an example. For additional information on using these RMOS functions, see the RMOS reference manual.

The call back function which was specified last is always used. This is particularly important to keep in mind when several tasks are to be executed simultaneously.

**Example**

```
#include <imc01bas.h>

unsigned int key;

void keyprocessing()
{
    key = bk_getchar();
}

void testfunction(void)
{
    bk_keybinit(keyprocessing);
    bg_outtext("keyboard initialized");
}
```

### 6.9.3 bk_LEDflush

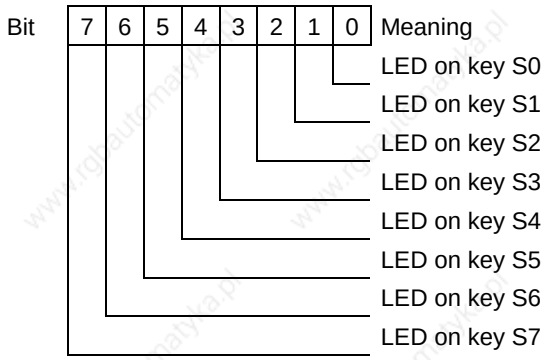
**Function** Control keyboard LEDs

**Syntax**

```
#include <imc01bas.h>
void bk_LEDflush(unsigned char value);
```

**Parameters**

| Parameter Name | Meaning |
|---|---|
| value | Bit pattern for controlling the 8 LEDs<br>Bit = 1: LED is on.<br>Bit = 0: LED is off. |




**Return value** None

**Example**

```
#include <imc01bas.h>

void testfunction(void);
{
    bk_LEDflush(0x01);        /* LED on S3 on. All others off. */
}
```

### 6.9.4 bk_setkeytab

**Function** Dummy function for IMC01

**Syntax**

```
#include <imc01bas.h>
void bk_setkeytab(unsigned char *keybtab);
```

**Parameters**

| Parameter Name | Meaning |
|---|---|
| *keybtab | Pointer to keyboard table |

**Return value** None

**Description** This is a dummy function for IMC01. It is included for interface compatibility with the IMC05.
This call is used with the IMC05 to provide BIOS with a new keyboard table for converting scan codes to character codes.
Since the scan codes, unlike the key codes, are dependent on the hardware, you should only use the key codes and execute any conversions yourself in your application.

## 6.10 Functions for Temperature Measurement

Table 6.10 Functions for temperature measurement

| Call | Function | Page |
|---|---|---|
| bpt_calibrate | Calibrate analog inputs for temperature measurement | 112 |
| bpt_gettemp | Determine temperature | 113 |
| bpt_init | Initialize analog inputs for temperature measurement | 114 |
| bpt_sel_chnum | Switch the temperature channel | 114 |

### 6.10.1 bpt_calibrate

**Function** Calibrate temperature inputs

**Syntax**

```
#include <imc01bas.h>
unsigned int bpt_calibrate(void)
```

**Return value**

| Function Value | Meaning |
|---|---|
| 0 | Calibration failed |
| 0xFF | Calibration successful |

**Description** The function determines the reference voltages of the temperature channels.

These reference voltages are used to generate a so-called temperature equation with which a measured voltage can be converted later on the PT100 to an appropriate temperature.

The function uses `bpt_select` to electrify the reference channels on the basic and expansion board in succession.

While the function is running, no other analog value conversion may be taking place.

The function is called by BIOS during startup. It does not need to be called by the user.

## 6.10.2 bpt_gettemp

**Function**

Determines temperature of the transferred temperature channel

**Syntax**

```
#include <imc01bas.h>
unsigned int bpt_gettemp(unsigned int channel)
```

**Parameters**

| Parameter Name | Meaning |
|---|---|
| Channel | Number of the temperature channel<br>2 and 3 for the basic board and<br>6 and 7 for the expansion board (if present) |

**Return value**

| Function Value | Meaning |
|---|---|
| 0xFFFF | Invalid |
| Other | Temperature value in 0.1 degrees |

**Description**

The function switches the PT100 measuring input to the selected channel and waits for a fixed reloading time. It starts the conversion and waits for it to be concluded. The function then returns the temperature value as a return value.

The reloading time is permanently set to 3 msec. Although the calling task is blocked during this time, the system provides computing time for lower-priority tasks.
As an alternative, the reloading time can also be managed by the application itself. If the bpt_sel_chnum function is called for the same channel before bpt_gettemp is called, bpt_gettemp starts the conversion immediately and no longer automatically waits for the reloading time to expire.

**Note:**
Remember that, when a switch is made between two channels with different potentials, a reload occurs which requires a sufficient waiting period.
3 msec are required for the maximum possible voltage difference.

### 6.10.3 bpt_init

**Function** Initialize analog inputs for temperature measurement

**Syntax**
```
#include <imc01bas.h>
void bpt_init(void);
```

**Return value** None

**Description** This function initializes the complete functionality of temperature measurement. Since this function is called already during system startup, it does not need to be called again by the application.

### 6.10.4 bpt_sel_chnum

**Function** Switching the temperature channel. The function is only required to speed up processing of bpt_gettemp by disabling a wait time in bpt_gettemp and permitting the application to implement this instead.

**Syntax**
```
#include <imc01bas.h>
unsigned int bpt_gettemp(unsigned int cannel)
```

**Parameters**

| Parameter Name | Meaning |
|---|---|
| Channel | Number of the temperature channel<br>2 and 3 for the basic board and<br>6 and 7 for the expansion board (if present) |

**Return value**

| Function Value | Meaning |
|---|---|
| _TRUE | OK. Function was executed. |
| _FALSE | Error. Function couldn't be executed. |

**Description** As an alternative, the reloading time can also be managed by the application itself. If the `bpt_sel_chnum` function is called for the same channel before `bpt_gettemp` is called, `bpt_gettemp` starts the conversion immediately and no longer automatically waits for the reload time to expire.

**Note:**
Remember that, when a switch is made between two channels with different potentials, a reload occurs which requires a sufficient waiting period.
3 msec are required for the maximum possible voltage difference.

## 6.11 Functions for Watchdog Processing

Table 6.11 Functions for watchdog processing

| Call | Function | Page |
|---|---|---|
| bwd_init | Activate/deactivate hardware watchdog | 115 |
| bwd_trigger | Trigger hardware watchdog | 116 |

### 6.11.1 bwd_init

**Function** Activate/deactivate hardware watchdog

**Syntax**

```
#include <imc01bas.h>
void bwd_init(unsigned char activate);
```

**Parameters**

| Parameter Name | Value/Meaning |
|---|---|
| activate | 0: Deactivate<br>1. Activate |

**Return value** None

**Description** This routine activates or deactivates the hardware watchdog of the IMC01.

If, after activation, the watchdog is not reset cyclically, a hardware reset is triggered.

**Example**

```
#include <imc01bas.h>

void testtask(void)
{
    bwd_init(1);    /* Watchdog activated */
    while(1)
    {
    bwd_trigger();    /* Trigger watchdog cyclically */
    RmPauseTask(100);    /* Wait 100 msec */
    }
}
```

**See also** bwd_trigger

### 6.11.2 bwd_trigger

**Function** Trigger hardware watchdog

**Syntax**

```
#include <imc01bas.h>
void bwd_trigger();
```

**Return value** None

**Description** This routine resets the hardware watchdog of the IMC01. A hardware reset is triggered if this routine is not called within approx. 250 msec after the watchdog was activated.

**See also** bwd_init

## 6.12 Functions for DP Slave Programming

Table 6.12 Functions for DP slave programming

| Call | Function | Page |
|---|---|---|
| dp_GetErrorState | Scan current status of the DP slave | 117 |
| dp_GetShortDPVersion | Scan software version | 118 |
| dp_GetStationAddress | Scan station address | 118 |
| dp_Init | Initialize and start complete DP subunit | 119 |
| dp_ReadOutputBuffer | Call output data of DP master | 120 |
| dp_WriteInputBuffer | Transfer input data to DP master | 121 |

### 6.12.1 dp_GetErrorState

**Function** Scan current status of the DP slave

**Syntax**

```
void _FIXED _FAR dp_GetErrorState(
        DP_ERROR_STATE _FIXED _FAR *psErrorState)
```

**Parameters**

| Parameter Name | Meaning |
|---|---|
| psErrorState | Pointer to a structure of type DP_ERROR_STATE (See chapter 6.12.7.) |

**Return value** None

**Description** This function supplies the current status of the DP slave (IMC01).

The status of a DP slave is primarily of interest if a data transmission has failed.

**Example**

```
typedef struct
{
    unsigned short wL2dpState;
        /* Current status of the L2DP unit */
....unsigned short wL2dpReason;
        /* Reason for the particular status */
} DP_ERROR_STATE;
struct DP_ERROR_STATE dp_status;
dp_GetErrorState(&dp_status);
```

**See also** dp_ReadOutputBuffer, dp_WriteInputBuffer

### 6.12.2 dp_GetShortDPVersion

**Function** Scan software version

**Syntax**
```
unsigned short _FIXED _FAR dp_GetShortDPVersion(void)
```

**Return value** Version identifier:

High byte: Version

Low byte: Release

**Description** This function supplies the current software version of the DP slave (IMC01).

### 6.12.3 dp_GetStationAddress

**Function** Scan station address

**Syntax**
```
unsigned char _FIXED _FAR dp_GetStationAddress(void)
```

**Return value** Current station address of the IMC01 in PROFIBUS-DP system

**Description** This function supplies the currently set station address of the IMC01. If the master does not assign a new address, the slave address set with `dp_Init` is used.

### 6.12.4 dp_Init

**Function** Initialize and start complete DP subunit

**Syntax**

```
unsigned short _FIXED _FAR dp_Init(
                    unsigned char bParamStationAddress,
                    unsigned char bParamTaskPriority,
                    unsigned int bParamPauseTime)
```

**Parameters**

| Parameter Name | Meaning |
|---|---|
| bParamStationAddress | Initial station address of the IMC01 on the PROFIBUS-DP system |
| bParamTaskPriority | Priority of the DP communication task in the system |
| bParamPauseTime | Wait time in the endless loop of the DP task<br>This provides computing time for lower-priority tasks. |

**Return value**

| Function Value | Meaning |
|---|---|
| DP_NO_ERR | Function executed successfully |
| DP_ERR_OUT_OF_MEM | Not enough work memory to set up the DP task |
| DP_ERR_INVALID_SIZE | Invalid value for stack size of the DP task |
| DP_ERR_INVALID_STRING | Invalid string for catalog name of the DP task |
| DP_ERR_INVALID_TASK_ENTRY | Invalid jump address for the DP task |
| DP_ERR_INVALID_PTR | Character string for the DP task name would trigger a protection violation. |
| DP_ERR_INVALID_ID | Invalid ID for the DP task |
| DP_ERR_INVALID_TYPE | Invalid task priority |
| DP_ERR_TASK_NOT_DORMANT | DP task was not started from the DORMANT state. |
| DP_ERR_TASK_KILLED | DP task was already killed before being started. |
| DP_UNKNOWN_ERROR | Unknown error |

**Description** This function initializes and starts the complete DP subunit. This generates and starts the communication task.

### 6.12.5 dp_ReadOutputBuffer

**Function** Call output data of DP master

**Syntax**

```
_BOOL _FIXED _FAR dp_ReadOutputBuffer(
        unsigned char _FIXED _FAR *pbReadBuffer)
```

**Parameters**

| Parameter Name | Meaning |
|---|---|
| pbReadBuffer | Pointer to a buffer with a minimum size of 32 bytes |

**Return value**

| Function Value | Meaning |
|---|---|
| _TRUE | Function executed successfully |
| _FALSE | No valid data stored in the buffer |

**Description**

This function calls the output data transferred by the DP master and stores the data in the specified buffer.

During one cycle, the master can transfer 32 bytes to the slave for use as desired.

The received data are only stored in the buffer when the slave is in "data exchange" status.

**Note:**
The size of the buffer is not checked. A wrong size may cause the system to crash.

**Example**

```
#define DP_PERIPH_OUT_LEN     32
        /* Output data from the master's viewpoint */

unsigned char dp_ReadOutBuf[DP_PERIPH_OUT_LEN];

        /**** Read in output data of the DP master */

if (dp_ReadOutputBuffer (&dp_ReadOutBuf[0]) != OK)
    printf ("Error l2dp_ReadOutputBuffer\n");
```

**See also** dp_GetErrorState

### 6.12.6 dp_WriteInputBuffer

**Function** Transfer input data to DP master

**Syntax**

```
_BOOL _FIXED _FAR dp_WriteInputBuffer(
          unsigned char _FIXED _FAR *pbWriteBuffer)
```

**Parameters**

| Parameter Name | Meaning |
|---|---|
| pbWriteBuffer | Pointer to a buffer with a minimum size of 32 bytes |

**Return value**

| Function Value | Meaning |
|---|---|
| _TRUE | Function executed successfully |
| _FALSE | No data transferred to the master |

**Description**

This function transfers the input data in the specified buffer to the DP master.

During one cycle, the master can receive 32 bytes from a slave.

The data in the buffer are only transferred to the master when the slave is in "data exchange" status.

**Note:**
The size of the buffer is not checked. A wrong size may cause the system to crash.

**Example**

```
#define DP_PERIPH_IN_LEN        32
        /* Input data from the master's viewpoint */
unsigned char dp_WriteInBuf[DP_PERIPH_IN_LEN];
        /***** Send input I/O to the DP master **/
if (dp_WriteInputBuffer (&dp_WriteInBuf[0]) != OK)
    printf ("Error l2dp_WriteInputBuffer\n");
```

**See also** dp_GetErrorState

### 6.12.7 Structure of DP_ERROR_STATE

**Syntax**

```
typedef struct
    {
       unsigned short wdpState;
       unsigned short wdpReason;
    }DP_ERROR_STATE;
```

**Structure elements**

| Name | Meaning |
|---|---|
| wdpState | Current status of the DP unit |
| wdpReason | Reason for the status transition |

## Constants for wdpState

| Constant | Meaning |
|---|---|
| DP_STATE_NO_INIT | Not initialized (original state) |
| DP_STATE_WAIT_PRM | Slave is waiting to be parameterized by the master. |
| DP_STATE_WAIT_CFG | Slave is waiting to be configured by the master. |
| DP_STATE_DATA_EXCHG | Data exchange |
| DP_STATE_ERROR | Unknown error state |

## Constants for wdpReason

| Constant | Meaning |
|---|---|
| DP_REASON_NO_INIT | Initialization has still not been performed. |
| DP_REASON_INIT_DONE | Initialization was performed successfully. |
| DP_REASON_INIT_FAULT | Error during initialization |
| DP_REASON_PRM_OK | Parameterization was successful. |
| DP_REASON_PRM_FALSE | Parameterization failed. |
| DP_REASON_CFG_OK | Configuration was successful. |
| DP_REASON_CFG_FALSE | Configuration failed. |
| DP_REASON_CARRIER_LOST | Transmission medium lost |
| DP_REASON_MASTER_LOST | Master was concluded. |

## 6.13 Functions of the Hardware Configuration

Table 6.13 Functions of the hardware configuration

| Call | Function | Page |
|---|---|---|
| imc01ReadHwStatus | Read hardware configuration | 124 |

### 6.13.1 imc01ReadHwStatus

**Function** Read hardware configuration

**Syntax**

```
#include <imc01bas.h>
IMC1_HW_CONFIG* imc01ReadHwStatus();
```

**Return value** Pointer to a structure of type IMC1_HW_CONFIG

**Description** The function returns a pointer to a global data field in BIOS which describes the hardware configuration of the device. The structure of the data field is defined in IMC01BAS.H as shown below.

```
typedef struct _IMC1_HW_CONFIG
 {
  IMC1_MAINBOARD    mboard;        /* main board */
 }  IMC1_HW_CONFIG;
```

**Example**

```
#include <imc01bas.h>

IMC1_HW_CONFIG* pHw;
int i;
i = 0;

while(1)
{
    i++;

    pHw = imc01ReadHwStatus();

    printf ("\n\nmainboard hardware version= %d",
        pHw->mboard.hw_version);
    printf ("\n0: IO-board not present, 1:present = %x",
        pHw->mboard.base_periph);
    printf ("\n0: EXT-board not present, 1:present= %x",
        pHw->mboard.ext_periph);
    printf ("\nRam memory start address= %xh",
        pHw->mboard.sram_linadr);
    printf ("\nRam memory size (only mainboard)= %xh",
        pHw->mboard.sram_size);
    printf ("\nFeprom memory start address= %xh",
        pHw->mboard.feprom_linadr);
    printf ("\nflash eprom memory size in byte= %xh",
        pHw->mboard.feprom_size);
    printf ("\nnumber of digital inputs= %d",
        pHw->mboard.numof_di);
    printf ("\nnumber of digital outputs= %d",
        pHw->mboard.numof_do);
    printf ("\nnumber of analog  inputs= %d",
        pHw->mboard.numof_ai);
    printf ("\nnumber of analog  outputs= %d",
        pHw->mboard.numof_ao);
    printf ("\nnumber of digital inputs with interrupt= %x",
        pHw->mboard.numof_int_di);
}
```

**Structure elements**

| Element name | Meaning |
|---|---|
| mboard | Configuration of the basic device. The setup is described in IMC01BAS.H. |

# 7 Appendix

## 7.1 Key Codes

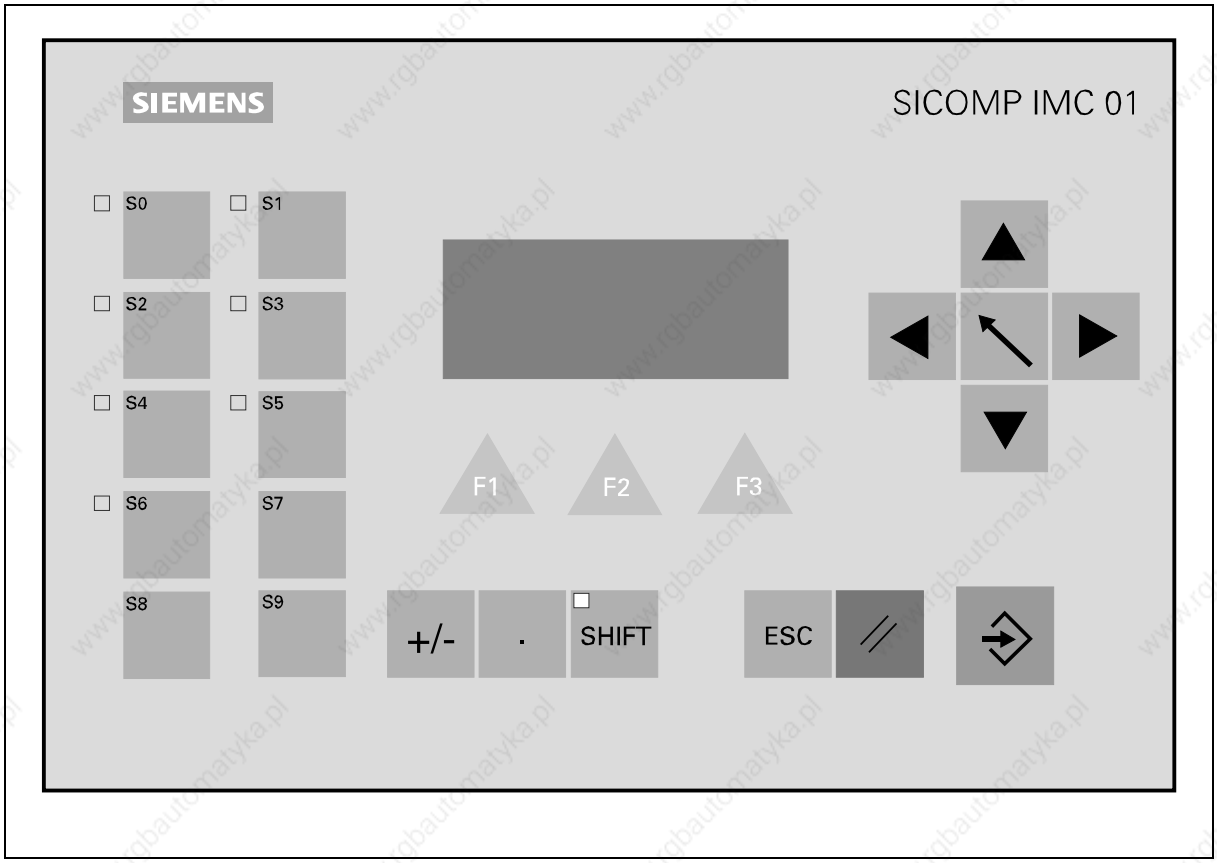



Figure 7.1 View of the front of the IMC01 with key designations

| Key Code | Key |
|---|---|
| 00h | No key |
| 01h | HOME |
| 02h | Arrow up |
| 03h | Arrow down |
| 04h | Arrow to left |
| 05h | Arrow to right |
| 07h | SHIFT |
| 0Ah | // (RESET) |
| 0Eh | ESC |
| 10h | Enter |
| 11h | +/– |
| 2Eh | . |
| | |

| Key Code | Key |
|---|---|
| 30h | S0 |
| 31h | S1 |
| 32h | S2 |
| 33h | S3 |
| 34h | S4 |
| 35h | S5 |
| 36h | S6 |
| 37h | S7 |
| 38h | S8 |
| 39h | S9 |
| 41h | F1 |
| 42h | F2 |
| 43h | F3 |

## 7.2 Error Messages during the Loading Procedure

The error messages listed below can be indicated on the development computer during the loading procedure. Commentary is not provided for self-explanatory messages.

```
Error 100: file command.txt not found !
```

```
Error 101: Illegal baud rate
```

Only standard baud rates can be used (e.g., 9600, 19200, and so on).

```
Error 105: Function called with wrong number of parameter !
```

Too many or too few call parameters

```
Error 106: Configuration file not found !
```

The configuration file (e.g., DELFW.CFG) specified for SEND.EXE does not exist.

```
Error 107: Binary file not found !
```

The binary file specified for SEND.EXE does not exist.

```
Error 108: RS232 communication error !
```

Block checksum error of the XMODEM protocol after block was repeated three times

```
Error 109: Illegal 'offset' in configuration file !
```

The specified file offset is larger than the total length of the binary file.

```
Error 110: Illegal 'netto data length' in configuration file !
```

The specified number of bytes to be transferred is greater than the total length of the binary file.

```
Error 112: Key pressed to abort data transfer !
```

Data transmission was terminated by pressing a key.

```
Error 113: Illegal COM channel !
```

Only the COM1 and COM2 interfaces can be used.

# 8 Abbreviations and Terms

| | |
|---|---|
| AC | Alternating Current |
| AD | Analog → Digital |
| AD conversion | Conversion of analog voltage to a digital value |
| ASCII | American Standard Code for Information Interchange |
| BIOS | Basic Input/Output System |
| BK | Pin terminal |
| BSP | Board Support Package (i.e., software package with board-related software) |
| COM1, COM2 | Serial interfaces of the development computer |
| CRC | Determination of a checksum (Cyclic Redundancy Check) |
| DA | Digital → Analog |
| DA conversion | Conversion of a digital value into an analog voltage |
| DC | Direct Current |
| DI | Digital Input |
| DIL | Dual in Line |
| DO | Digital Output |
| DP | Distributed Periphery |
| ED | Switch-on duration (percentage) |
| EMC | Electro-Magnetic Compatibility |
| EN | European standard |
| ESD | Electrostatic Sensitive Device |
| FLASH memory | Non-volatile memory which can be deleted in blocks |
| FPU | Floating Point Unit, used by the coprocessor |
| HF | High Frequency |
| HLL | High Level Language (e.g., C language, in contrast to machine languages such as Assembler) |
| LCD | Liquid Crystal Display |
| LED | Light Emitting Diode |
| LSB | Least Significant Bit |
| MSB | Most Significant Bit |
| NMI | Non Maskable Interrupt |
| PLC | Programmable Logic Controller |
| PROFIBUS | Process Field Bus |
| PS | Power Supply |
| RDC | Direction discriminator |
| RMOS | Real-time Multitasking Operating System |

| | |
|---|---|
| RTC | Real-Time Clock |
| SDI | Serial Data Input, signal of an absolute value encoder |
| Soft-PLC | Software-implemented programmable logic controller |
| SRAM | Static Random Access Memory |
| STEP 5 | Machine language for programming programmable logic controllers |
| Zero modem cable | For serial data transfer from the development computer to the IMC01 |