

SMART2

Micro PLC and Real-Time Computer for Industrial Applications

ID 19342, Rev. 0200
Jul 99

This page was intentionally left blank.

Preface

Revision History ..... *0 - 4*
For Your Safety ..... *0 - 5*
Special Handling and Unpacking Instructions *0 - 6*
HV Safety Instructions ..... *0 - 7*
Two Years Warranty ..... *0 - 8*
Table of Contents ..... *0 - 9*

© *PEP Modular Computers*

Revision History

Manual/Product Title:	SMART2	
Manual ID Number:	19342	
Rev. Index	**Brief Description of Changes**	**Date of Issue**
0100	Initial Issue	Apr. 98
0110	Ch. 7.2 (SM-COUNT2) added; minor changes	Aug. 98
0120	Ch 7.2: OS-9 calls added; uniform manual ID	May 99
0200	Ch3: “Fieldbus modules” renamed in “Network plug-in modules”, SM2-ETH added; Tables 1-1, 1-4, 1-5 and 1-6 amended.	July 99

This document contains proprietary information of *PEP Modular Computers*. It may not be copied or transmitted by any means, passed to others, or stored in any retrieval system or media, without the prior consent of *PEP Modular Computers* or its authorized agents.

The information in this document is, to the best of our knowledge, entirely correct. However, *PEP Modular Computers* cannot accept liability for any inaccuracies, or the consequences thereof, nor for any liability arising from the use or application of any circuit, product, or example shown in this document.

PEP Modular Computers reserve the right to change, modify, or improve this document or the product described herein, as seen fit by *PEP Modular Computers* without further notice.

For your safety

This *PEP* product is carefully designed for a long, fault-free life. However, its life expectancy can be drastically reduced by improper treatment during unpacking and installation. Therefore, in the interest of your own safety and of correct operation of your new PEP product, please take care of the following guidelines:

- Before installing your new *PEP* product, please always switch off your mains supply. This applies also to installing SM/SM2 modules in a SMART2 or SMART I/O system.
- In order to maintain *PEP's* product warranty, please do not alter or modify this product in any way. Changes or modifications to the device, which are not explicitly approved by *PEP Modular Computers* and described in this manual or received from *PEP* Technical Support as a special handling instruction, will void your warranty.
- This device should only be installed in or connected to systems that fulfill all necessary technical and specific environmental requirements. This applies also to the operational temperature range of the specific board version, which must not be exceeded. If batteries are present, their temperature restrictions must be taken into account.
- In performing all necessary installation and application operations, please follow the instructions supplied by this manual.
- Keep all the original packaging material for future storage or warranty shipments. If it is necessary to store or ship the board, please re-pack it the original way.

Special Handling and Unpacking Instructions

Electronic boards are sensitive to static electricity. Therefore, care must be taken during all handling operations and inspections with this product, in order to ensure product integrity at all times.

- Do not handle this product out of its protective enclosure while it is not being worked with, or unless it is otherwise protected.
- Whenever possible, unpack or pack this product only at EOS/ESD safe work stations.
- Where safe work stations are not guaranteed, it is important for the user to be electrically discharged before touching the product with his/her hands or tools. This is most easily done by touching a metal part of your system housing.
- Particularly, observe standard anti-static precautions when changing SM/SM2 modules, ROM devices, jumper settings etc. If the product contains batteries for RTC or memory back-up, ensure that the board is not placed on conductive surfaces, including anti-static plastics or sponges. They can cause short circuits and damage the batteries or tracks on the board.

HV Safety Instructions

This chapter of the safety instructions applies to **HIGH-VOLTAGE APPLIANCES (> 60 V)** only, namely the SM-ACI and SM-ACO modules.

Your new *PEP* product was developed and tested carefully to provide all features necessary to ensure the reknown electrical safety requirements. However, serious electrical shock hazards exist during all installation, repair and maintenance operations with this product. Therefore, always unplug the power cable to avoid exposure to hazardous voltage.

All operations on this device have to be carried out by sufficiently skilled personnel only.

Two Years Warranty

PEP Modular Computers grants the original purchaser of *PEP* products a ***TWO YEARS LIMITED HARDWARE WARRANTY*** as described in the following. However, no other warranties that may be granted or implied by anyone on behalf of *PEP* are valid unless the consumer has the expressed written consent of *PEP Modular Computers*.

PEP Modular Computers warrants their own products, excluding software, to be exempt of manufacturing and material defects for a period of 24 consecutive months from the date of purchase. This warranty is not transferable nor extendible to cover any other users or long-term storage of the product. It does not cover products which have been modified, altered or repaired by any other party than *PEP Modular Computers* or their authorized agents. Furthermore, any product which has been, or is suspected of being damaged as a result of negligence, unproper use, incorrect handling, servicing or maintenance, or which has been damaged as a result of excessive current/voltage or temperature, or which has had its serial number(s), any other markings or parts thereof altered, defaced or removed will also be excluded from this warranty.

If the customer's eligibility for warranty has not been voided, in case of any claim, the product should be returned at the earliest possible convenience to the original place of purchase, together with a copy of the original document of purchase, a full description of the application the product is used in and a description of the defect. Pack the product in such a way as to ensure safe transportation (see our safety instructions).

PEP provides for repair or replacement of any part, assembly or subassembly at their own discretion, or to refund the original cost of purchase, if appropriate. In the event of repair, refunding or replacement of any part, the property of the removed or replaced parts reverts to *PEP*

Modular Computers, and the remaining part of the original guarantee, or any new guarantee to cover the repaired or replaced items, will be transferred to cover the new or repaired items. Any extensions to the original guarantee are considered gestures of goodwill, and will be defined in the “Repair Report” issued by *PEP* with the repaired or replaced item.

PEP Modular Computers will not accept liability for any further claims resulting directly or indirectly from any warranty claim, other than the above specified repair, replacement or refunding. Particularly, all claims for damage to any system or process in which the product was employed, or any loss incurred as a result of the product not functioning at any given time, are excluded. The extent of *PEP Modular Computers* liability to the customer shall not exceed the original purchase price of the item for which the claim exists.

PEP Modular Computers issues no warranty or representation, either explicit or implicit, with respect to its products, reliability, fitness, quality, marketability or ability to fulfil any particular application or purpose. As a result, the products are sold “as is,” and the responsibility to ensure their suitability for any given task remains with the purchaser. In no event will *PEP* be liable for direct, indirect or consequential damages resulting from the use of its hardware or software products, or documentation, even if *PEP* were advised of the possibility of such claims prior to the purchase of the product or during any period since the date of its purchase.

Please remember that no *PEP Modular Computers* employee, dealer or agent is authorized to make any modification or addition to the above specified terms, either verbally or in any other form written or electronically transmitted, without the company’s consent.

Table of Contents

Chapter 1

1. General Information 1-3

1.1 Special Safety Instructions ..... 1 - 6

1.2 Wall/Cabinet Mounting ..... 1 - 7

1.3 Product Overview ..... 1 - 9

1.4 System Overview ..... 1 - 10

1.5 Module Overview ..... 1 - 11

1.6 Product Information ..... 1 - 17

1.7 Installation ..... 1 - 18

1.8 SMART2 Starter Kit ..... 1 - 20

Chapter 2

2. SMART2-BASE 2-3

2.1 Basic Concept of the SMART2-BASE 2-3

2.2 Main Features 2-7

2.3 SMART2-BASE Interfaces 2-12
2.4 Configuration 2-14
2.5 Pinouts 2-15
2.6 Compatibility with SMART I/O Systems 2-18
2.7 Software 2-20

Chapter 3

3. Basic Modules 3-4
3.1 SM2-CPU 3-4
3.2 Network Plug-In Modules 3-19
3.3 SM-DC24V 3-44

Chapter 4

4. Digital Modules 4-5
4.1 SM-DIN1 4-5
4.2 SM-DOUT1 4-19
4.3 SM-REL1 4-33

4.4 SM-ACI1 4-51
4.5 SM-ACO1 4-71

Chapter 5

5. Analog Modules 5-7
5.1 SM-DAD1 5-7
5.2 SM-PT100 5-29
5.3 SM-THERM 5-59
5.4 SM-ADC1 5-89
5.5 SM-DAC1 5-107

Chapter 6

6. Communication Modules 6-5
6.1 SM-SSI 6-5

Chapter 7

7. Counter Modules 7-5

7.1 SM-COUNT1 7-5

7.2 SM-COUNT2 7-42

Introduction

© PEP Modular Computers

This page was intentionally left blank.

Table of Contents

1.1 Special Safety Instructions ..... 1 - 6
1.2 Wall/Cabinet Mounting ..... 1 - 7
1.3 Product Overview ..... 1 - 9
1.4 System Overview ..... 1 - 10
1.5 Module Overview ..... 1 - 11
1.5.1 Base/Extension Unit and Basic Modules ..... 1 - 11
1.5.2 Input/Output Modules ..... 1 - 13
1.5.3 Software and Accessories ..... 1 - 16
1.6 Product Information ..... 1 - 17
1.6.1 Electrical Parameters ..... 1 - 17
1.7 Installation ..... 1 - 18
1.8 SMART2 Starter Kit ..... 1 - 20
1.8.1 What Is the “SMART2 Starter Kit”? ..... 1 - 20
1.8.2 Step 1: Checking the Hardware ..... 1 - 20
1.8.3 Step 2: SMART2-BASE ..... 1 - 22
1.8.4 Step 3: SM Module Installation ..... 1 - 23
1.8.5 Step 4: Connecting the SM2-CPU ..... 1 - 24
1.8.6 Step 5: Connecting the SM2-DC24V3 ..... 1 - 25
1.8.7 Step 6: Connecting the SM-DIN1 ..... 1 - 27
1.8.8 Step 7: Getting Started with Software ..... 1 - 28

 © PEP Modular Computers

This page was intentionally left blank.

1. General Information

CE Conformity

The SMART2 product line complies with the requirements of the EU Directive 89/336/EEC “Electromagnetic Interferences” and all harmonized European standards contained therein.

All *PEP* products are intended for use in industrial climates where extreme environments exist. Dirt and dust, temperature extremes, varying humidity levels, vibration, noise, shock and electromagnetic interference must all be considered. Only when certain precautions have been followed can *PEP* guarantee the performance of the product stated in the data sheet. In most cases, controllers are situated in close proximity to electromechanical devices like relays, transformers, motor controllers and high-frequency switches etc. In such situations, a variety of interference sources are present and affect the performance characteristics of the controller.

Attention!

The SMART2 product line comprises class “A” appliances in accord with the above mentioned standards on EMI emission. However, in households and light industry environments some components of a SMART2 system may cause radio interference.

In this case the user is responsible for taking all appropriate preventive measures.

By observing the following precautions, the possible interference can be minimized or eliminated.

- Protect the fixing elements of your carrier unit(s) by earthing.
- Use screened/shielded cable connections and place cables according to the relevant standards. Use a shielded mains supply cable.
- Where shielded cables are used, connect the cable shields to the SMART2 shield connection frame. However, to avoid interference generated by and transmitted via the cables, it is advisable to provide earthing of the cable shield on both cable ends.
- Connect earth protections as directly as possible to the closest protection points, avoiding earth protection loops.
- Use cable of the correct diameter and type.
- For unused module sockets, install a blank front-panel.
- Leave enough room between hot bodies and the controller for air to circulate freely.
- Place the controller as far away as possible from components emitting electrical interference signals.
- Where possible, separate analog modules from digital ones. Separate digital and analog cabling.

1.1 Special Safety Instructions



High Voltage!

Only replace/insert modules in the carrier unit with the power OFF.



ESD Sensitive Device!

Please handle the controller components with care. In particular, SM modules containing highly integrated CMOS components are very sensitive to static discharges.

1.2 Wall/Cabinet Mounting

Figure 1-1: SMART2-BASE Carrier Unit — DIN Rail Mounting

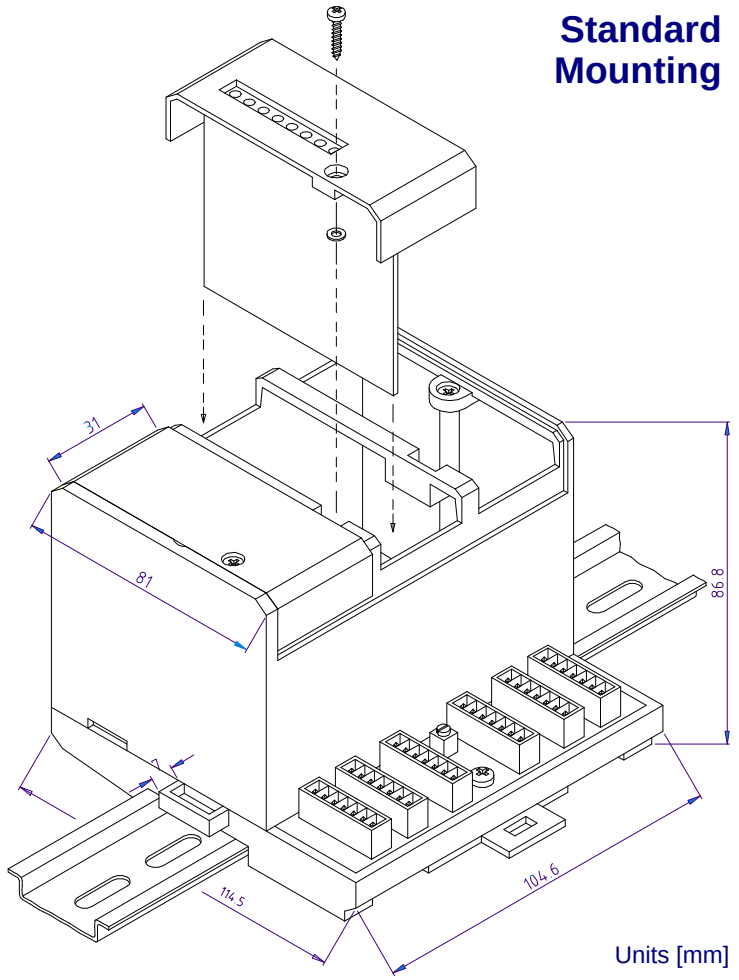
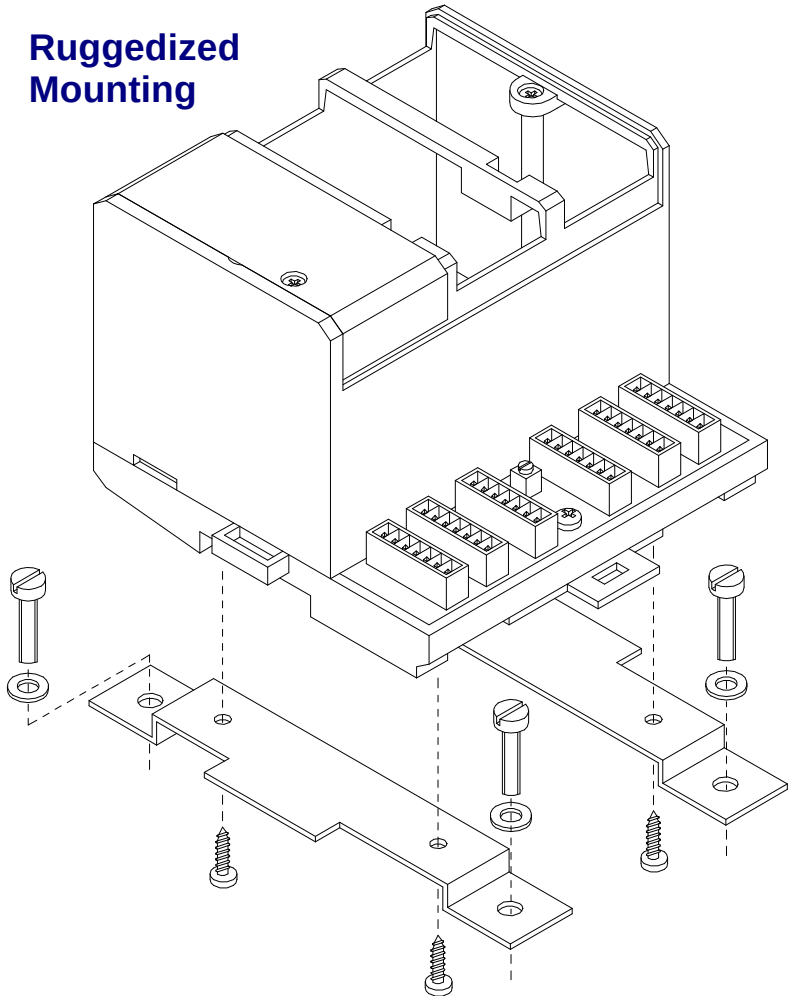


Figure 1-2: SMART2-BASE Carrier Unit — Bracket Mounting



The above drawings illustrate wall/cabinet mounting of a SMART2-BASE unit by fixing the controller to a DIN rail or, optionally, fixing it with brackets. Please note that all measurements are in millimeters.

1.3 Product Overview

SMART2 is based on a cost-effective open system for industrial automation and industrial computing. By programming the SMART2 using the standard ISaGRAF workbench for IEC 1131-3 PLC programming languages or the Ultra-C compiler (DOS, OS-9) for ANSI-C real-time programming, the SMART2 can be used as a micro PLC and a real-time computer system.

Equipped with a real-time fieldbus such as PROFIBUS or CAN, SMART2 allows a fully transparent real-time network architecture. This architecture provides open communication between PEP systems and third party I/O systems, as well as MMI. Profibus and/or CAN not only allow I/O communication, but also file transfer, remote log-in and remote debugging.

SMART2 is designed for maximum flexibility which is achieved by complete system modularity. It consists of a carrier unit, SMART2-BASE, and various controller, fieldbus, power supply and I/O modules to be installed on the carrier unit. The carrier unit can be used either as a system's main controller unit (base unit), or as an extension to the base unit. To make it a base unit, the user will need to provide the carrier with a controller module. This module is designed around a main controller chip, namely the Motorola MC68LC302 CPU running at 20 MHz. If used without the controller module, SMART2-BASE can be run as an extension to a base unit.

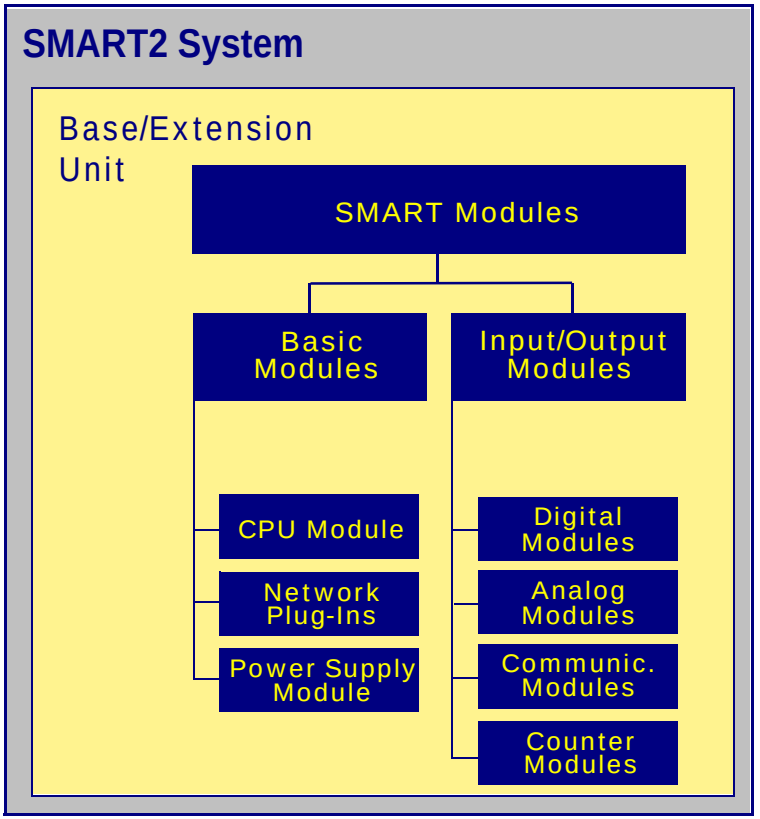
Local controllers on special fieldbus modules provide for communication, thus freeing the main controller for other tasks. Communication between the CPU and the communication processor is made via a common-memory interface. Non-volatile memory (goldcap-backed SRAM and FLASH memory) allows a secure and long-term backup of the application program and data.

Well suited for machinery manufacturers and all OEMs, the SMART2 is an ideal companion for VME9000 and IUC9000 systems. In conjunction with

these high-end VME or IUC computers and PLCs, SMART2 allows the decentralization of I/O functions through fieldbusses that have gained world-wide acceptance such as the PROFIBUS. SMART2 systems provide enhanced performance as compared to dumped remote I/O systems, since they support local intelligence — an IEC1131-3 programming environment and ANSI-C programming under real-time OS.

1.4 System Overview

Figure 1-3: Conceptual SMART2 System Overview



1.5 Module Overview

In the following, a brief module overview is given for SMART2 and/or SMART I/O systems.



PEP Advantage

Strictly speaking, the SM modules should be named “SM/SM2” modules, since both SMART I/O and SMART2 modules are supported indiscriminately by both systems.* However, for the sake of simplicity, these modules are called “SM” modules throughout this manual.

* Exception: the SMART2 basic modules are designed especially for use in SMART2 systems, while the SM-RS232 serial comunication module may be used only in SMART I/O systems



Note:

All SM modules are delivered without screw terminals.

1.5.1 Base/Extension Unit and Basic Modules

As will be described in detail, SMART2 systems use just one type of carrier unit, SMART2-BASE, both as a base and as extension units. — Basic Modules are those SM modules supporting system functions like, for instance, CPU, fieldbus and power supply functions.

Table 1-1: Base/Extension Unit and Basic Modules

Product	Description	Order ID
SMART2-BASE	SMART2 carrier unit for SM modules, intended to be used as a SMART2 base or a SMART2/SMART I/O extension unit. — Carrier board for installation/ interfacing of three SM modules: first slot includes controller/fieldbus module option, max. 5 base/ extension units combined in a system; housing and terminal block.	16070
SM2-CPU	SMART2 controller module, Motorola 68LC302, 20 MHz; DRAM 512 kB *(option: 1.5 MB);* FLASH 1 MB; SRAM 256 kB. Interfaces: 2 serial RS232s, 1 with full modem support; serial SPI master channel, max. 4 MHz; FPC connector to fieldbus module; system supply connector +5 V DC, max. 3A.	16087 *Option: 16788*
SM2-DP	SMART2 Profibus DP fieldbus plug-in module for high-speed networking and/or large data volumes, to be fitted onto an SM controller module or directly into a base/extension unit.	16085
SM2-CAN	SMART2 CAN fieldbus plug-in module for higher-distance networking, to be fitted onto an SM controller module or directly into a base/extension unit.	16492
SM2-ETH	SMART2 Ethernet plug-in module for higher-distance, large-quantity, non-time-critical network communication, to be fitted onto an SM controller module or directly into a base/extension unit.	16492
SM2-DC24V	SMART power supply module for SMART2 systems, 18...36 V DC to 5 V DC/2 A *(options: 1 A/0.6 A)* + 4 digit. outputs 24 V DC/0.5 A.	16969 *Options:* 16491 18070

1.5.2 Input/Output Modules

Table 1-2: Digital I/O Modules

Product	Description	Order ID
SM-DIN1	SMART module with 8 optoisolated 24 V DC digital inputs.	04229
SM-DOUT1	SMART module with 8 optoisolated 24 V DC/500 mA digital outputs.	04231
SM-DAD1	SMART module with 4-channel A/D (±10 V), 2-channel D/A (±10 V, 0..10 V).	09868
SM-PT100	SMART module with 4 optoisolated 2, 3 or 4-wire PT100 inputs.	12405
SM-THERM	SMART module with 4 optoisolated thermocouple inputs.	12426
SM-REL1	SMART module with 6 optoisolated normally open relay outputs.	12238
SM-ACI1	SMART module with 8-channel 80-264 V AC input.	14753
SM-ACO1	SMART module with 6-channel 80-264 V AC output	14752

Table 1-3: Analog I/O Modules

Product	Description	Order ID
SM-ADC1	SMART module with 6 12-bit, ±10 V optoisolated analog inputs.	13380
SM-ADC1	SMART module with 6 12-bit, 0..20 mA optoisolated analog inputs.	13868
SM-DAC1	SMART module with 2 12-bit, ±10V optoisolated analog outputs.	13379
SM-DAC1	SMART module with 2 12-bit, 0..20 mA optoisolated analog outputs with current sensing for broken sensor detection.	14018
SM-DAC1	SMART module with 6 12-bit, ±10 V optoisolated analog outputs.	14019
SM-DAC1	SMART module with 6 12-bit, 0..20 mA optoisolated analog outputs with current sensing for broken sensor detection.	14020

Table 1-4: Communication Module(s)

Product	Description	Order ID
SM-SSI	SMART module with 1 SSI channel providing a 24 V DC digital input and a 24 V DC, 500mA digital output.	12825
SM2-ETH	SMART2 Ethernet plug-in module for higher-distance, large-quantity, non-time-critical network communication, to be fitted onto an SM controller module or directly into a base/extension unit.	16492

Table 1-5: Counter Modules

Product	Description	Order ID
SM-COUNT1	SMART module with one “up/down” incremental counter, 3 differ. and 2 digit. inputs 24 V DC *(options: 5V DC [a] and 12 V DC [b])*, 1 digit. output 24 V DC/0.1 A. Resolution: 24-bit.	15528 *Options: 18337[a] 17516[b]*
SM-COUNT2	SMART module with two “up” incremental counters, 2 diff. counter, 1 diff. common clear and 2 gate inputs 24 V DC, 2 digit. outputs 24 V DC/0.1 A *(options: 5V DC [a] and 12 V DC [b])*. Resolution: 16-bit with 8-bit prescaler.	15529 *Options: 18339[a] 18338[b]*

1.5.3 Software and Accessories

Table 1-6: Software

Product	Description	Order ID
SM2-CPU-FIRM	ISaGRAF Firmware, V 3.3x, PLC CASE programming tool for SM2-CPU modules.	20433
SM2-CPU-FIRM	OS-9 Firmware, V 3.03 forSM2-CPU modules with SM2-DP and SM2-CAN support.	20431
SM2-CPU-FIRM	OS-9 Firmware, V 3.03 forSM2-CPU modules with SM2-ETH support.	20432
ISaGRAF-ROM-START	CD-ROM kit V 3.30 for SMART2 enabling the generation of customized firmware. Platform: Win95/NT, PC-based.	13829
OS9TKT-SMART2	Target CPU kit for SMART2 (OS-9 v3.x/Ix.x disks) for development under native OS-9.	17663
OS9DPXW-SMART2	OS-9/68000 FasTrak for Windows development pack for SMART2. Contains extended OS-9 v3.x/Ix.x, PEP utilities and I/O drivers with necessary makefiles for complete application and firmware EPROM generation.	17662

Table 1-7: Accessories

Product	Description	Order ID
SM2-SCR7	7-pin screw terminal blocks (120 V – SMART2 modules only!), odd/even numbering. Set of 10 pieces	16498
SM-DUMMY-FP	Dummy insert for unused SM module slots. Set of 20 pieces.	10894
SM2-CPU-FP	Dummy insert for SM2-CPU without fieldbus modules.	16490
SMART2-SHIELD	Element for grounding shielded signal cables; to be used with SMART2-BASE unit.	16811
CABLE-SM-RS232-9	RS232 cable, length = 3 m, with 9-pin DSUB (female) and RJ45 connectors for PC operation.	10890
CABLE-SM-RS232-MOD	RS232 cable, length = 3 m, with 25-pin DSUB (male) and RJ45 connectors for modem operation.	10891

1.6 Product Information

1.6.1 Electrical Parameters

It is possible to install up to fifteen SM modules in a complete SMART2 system, which would then consist of 1+4 SMART2-BASE carrier units. However, attention must be paid to the following table that shows the individual component power requirements.

Attention!

Please do not exceed the maximum power limit of 10 W as damage to the SM2-DC24V1 power supply module may result due to overheating or entering a rest status caused by the thermal cutoff protection switch.

Table 1-8: Power Limitations of SM Modules

Module	Minimum Power Consumption	Maximum Power Consumption
SM2-CPU	650 mW	1100 mW
SM2-DC24V	30 mW	80 mW
SM2-DP	800mW	1400mW
SM2-CAN	650mW	1000mW
SM-DIN1	5 mW	100 mW
SM-DOUT1	5 mW	270 mW
SM-DAD1	350 mW	450 mW

Table 1-8: Power Limitations of SM Modules

Module	Minimum Power Consumption	Maximum Power Consumption
SM-PT100	350 mW	450 mW
SM-THERM	350 mW	450 mW
SM-REL1	23 mW	160 mW
SM-ADC1	350 mW	450 mW
SM-DAC1 (±10 V)	5 mW	660 mW
SM-DAC1 (0..20 mA)	400 mW	600mW
SM-SSI	275 mW	400 mW
SM-ACI1	3 mW	70 mW
SM-ACO1	30 mW	280 mW
SM-COUNT1	350mW	370mW
SM-COUNT2	350mW	370mW

1.7 Installation

The SMART2-BASE carrier units are supplied without screw terminal blocks for the I/O slots, SM modules or blank panels. These must be ordered separately as required.

SMART2-BASE

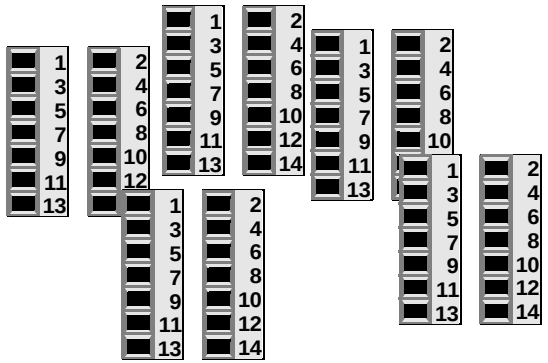
Up to five SMART2-BASE carrier units may be cascaded depending on the power consumption of the individual SM modules.

The configuration of SMART2-BASE units together with SMART-BASE and/or SMART-EXT carrier units is illustrated in detail in chapter 2.6, **"Compatibility with SMART I/O Systems"**, of this manual.

SMART2-BASE Accessories

Void places on your SMART2-BASE, which you may wish to occupy later, should be covered by blank panels that can be ordered in packs of 20.

Figure 1-4: SMART2 I/O Terminal Blocks



Seven-pole screw terminal blocks for your I/O lines are available in packs of five. The seven-pole terminal blocks you may have been using already with SMART-BASE and SMART-EXT are different in size and, therefore, cannot be used on the SMART2-BASE.

SM Modules

The SM modules are fitted into the relevant sockets on the SMART2-BASE carrier unit(s). The first carrier unit of a multiple-unit system hosts the CPU and the power supply module. A typical example of such a main carrier unit is given by the SMART2 Starter Kit. The corresponding installation procedure is illustrated in the following chapter.

1.8 SMART2 Starter Kit

If you have acquired and are about to start up a SMART2 Starter Kit, or if you want to familiarize yourself with the SMART2 modularity concept before starting up a larger system, you will benefit by going through the following steps detailed in the SMART2 Starter Kit description. All you need to do so is a screwdriver and approximately **2 hours** of time.

Otherwise you may skip this chapter and move on to the next relevant topic.

1.8.1 What is the “SMART2 Starter Kit”?

The SMART2 Starter Kit is a complete I/O control system limited to its most essential components. It provides all the necessary sub-assemblies for a small application based on a SMART2-BASE carrier unit, the yellow box you can find in your SMART2 Starter Kit, and includes a power supply module, a CPU module and an I/O module with four digital output and eight digital input channels.

To instal and run your ISaGRAF PLC software you need a 486-type or better PC provided with a CD-ROM drive. On your PC, Windows 3.1x, WIN95 or Windows NT should be running, and it should be provided with at least 8 MB program memory.

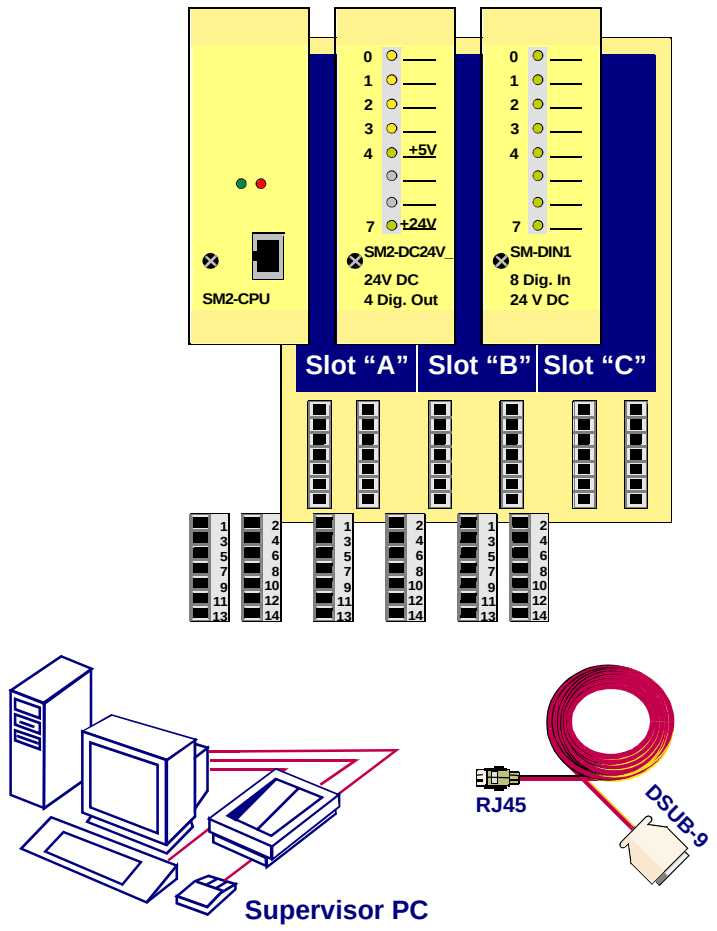
In the following, a quick overview of the system is given, and you will be guided through the first steps of installing and starting up a SMART2 system.

1.8.2 Step 1: Checking the Hardware

The SMART2 Starter Kit is based on a hardware configuration having the following key features:

- CPU unit with Motorola 68LC302 / 20MHz
- RS232 with full modem support
- 24V/3W power supply unit
- 4 channels with 24V/0.5A digital out
- 8 channels for 24V digital in

Figure 1-5: SMART2 Starter Kit Installation — Steps 1/2



1.8.3 Step 2: SMART2-BASE

SMART2-BASE is the carrier unit of your SMART2 Starter Kit. It has three slots into which the SM modules will be fitted during steps 3 to 5 of the installation example.

For reasons of simplicity the three slots present on the SMART2-BASE are numbered with the capital letters “A” to “C”. Usually in a multiple-unit SMART2 system, the slots of every single SMART2-BASE carrier unit are referred to by the first three letters of the alphabet, while numbers from “0” to “14” identify the slots with reference to the entire system.

Therefore, the SMART2 Starter Kit, being a one-unit system, the carrier unit slots “A” to “C” are identical with the system slots “0” to “2”.

Note:

If after completion of the installation practice example you want to reconfigure your system to mount a different SM module, please refer to the description of the specific module you intend to install.

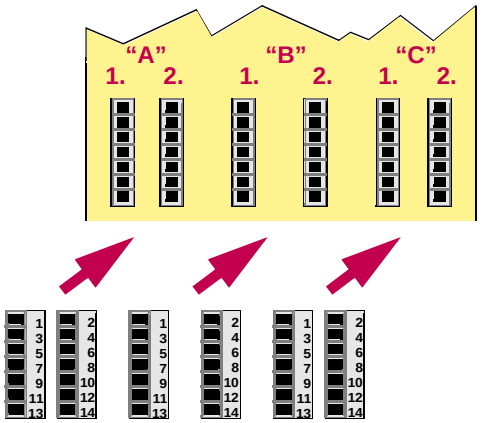


Figure 1-6: SMART2 Starter Kit Installation — Step 3

Before proceeding with the next installation step, please mount the screw terminal blocks “A1” to “C2” according to the illustration on the left.

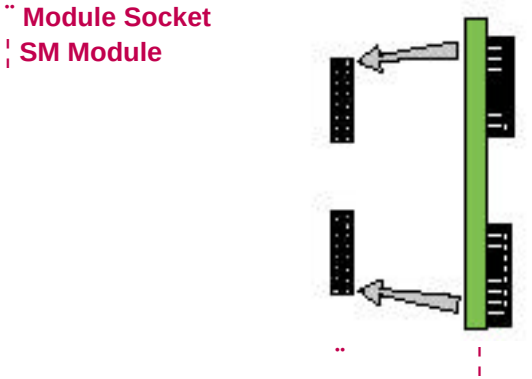
1.8.4 Step 3: SM Module Installation



High Voltage!

Before inserting or extracting a module from its slot, please make sure the system power supply has been disconnected. Failure to do so may damage your module/system.

Figure 1-7: Correct Module Installation



All slots of your Starter Kit modules already have a module in place. Please check now that the system configuration is correct. The slot assignment must be as follows:

- Slot “0” (“A”) = SM2-CPU
- Slot “1” (”B”) = SM2-DC24V3
- Slot “2” (”C”) = SM-DIN1



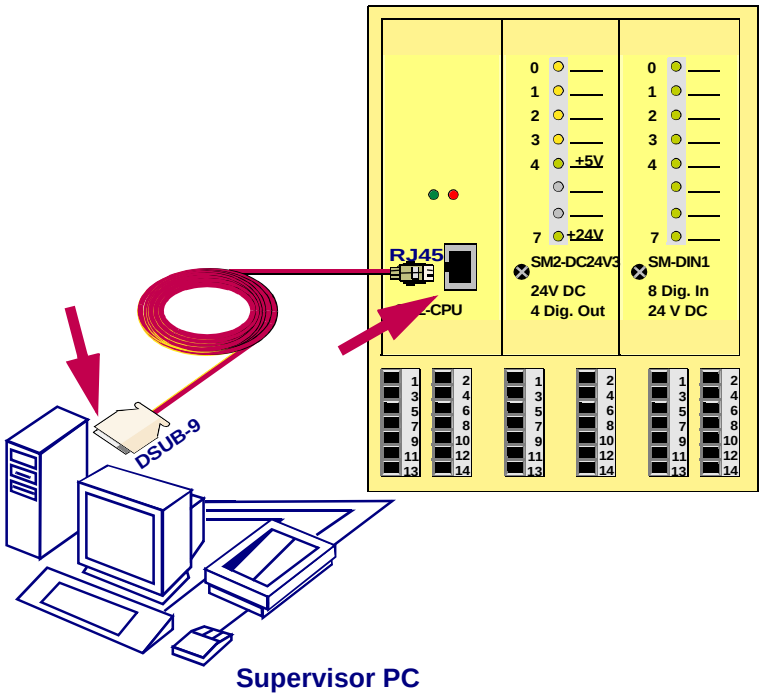
PEP Advantage

PEP Modular Computers supplies upgraded modules with your starter kit. In the case of the SM2-DC24V power supply module this implies a higher power rating. However, module functionality will remain unchanged.

1.8.5 Step 4: Connecting the SM2-CPU

The SMART2 CPU is the connection interface to your PC. On the one hand it controls all the other slots, on the other hand it provides the ISaGRAF task for your on-line serial 232 connection.

Figure 1-8: SMART2 Starter Kit Installation — Step 4



Front Connectors

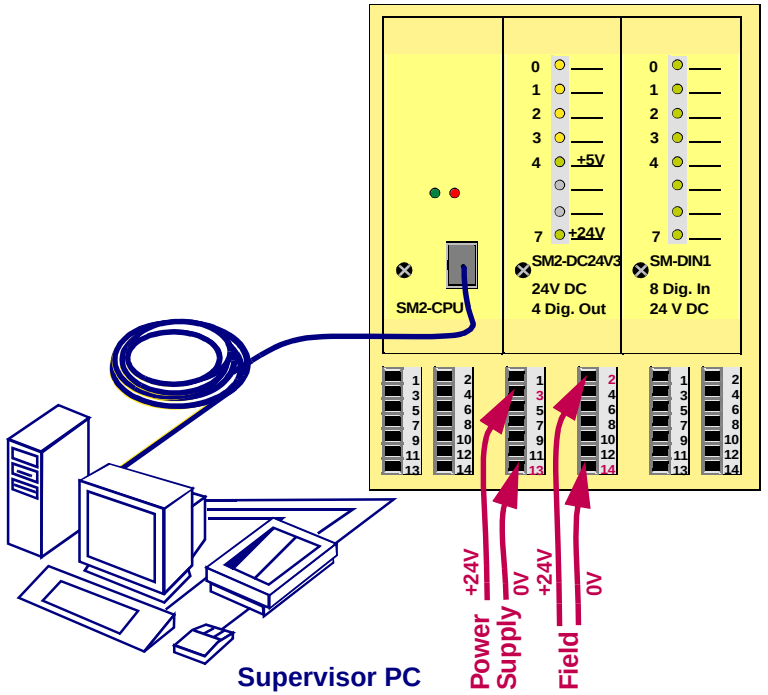
The front connector consisting of the screw terminal blocks A1 and A2 is not connected in the SMART2 Starter Kit configuration. For further details on its usage, please consult the corresponding chapter of the User's Manual.

Serial Connector

A serial connection is established by connecting the telephone connector of the RS232 serial connection cable in your SMART2 Starter Kit box to the RJ45 socket on the front panel of the SM2-CPU module. The DSUB-9 connector of the serial cable is to be connected to the next free serial interface of your supervisor PC, for instance COM2.

1.8.6 Step 5: Connecting the SM2-DC24V3

Figure 1-9: SMART2 Starter Kit Installation — Step 5



The SM2-DC24V3 module has two different functions within your SMART2 system, namely:

- 24 V/5 V converter with 3 W power supply for the whole system;
- 4 digital 24 V output channels.

Front Connector — 24 V Supply Circuit

The 24 V supply circuit is connected to screw terminal connector B.1 of your SMART2 system. For this purpose, please connect your +24 V power supply line to pin 3 of the front connector and the ground line of the 24 V power supply circuit to pin 13 of the same terminal block.



High Voltage!

Before continuing with the start-up operations, please make sure the system power supply is disconnected. Failure to do so may damage your module/system.

Front Connector — 24 V Field Circuit

The 24 V field signalling circuit is connected to screw terminal connector B.2 of your SMART2. For this purpose, please connect the +24 V field current to pin 2 of the front connector and the ground line of the 24 V field circuit to pin 14 of the same terminal block.

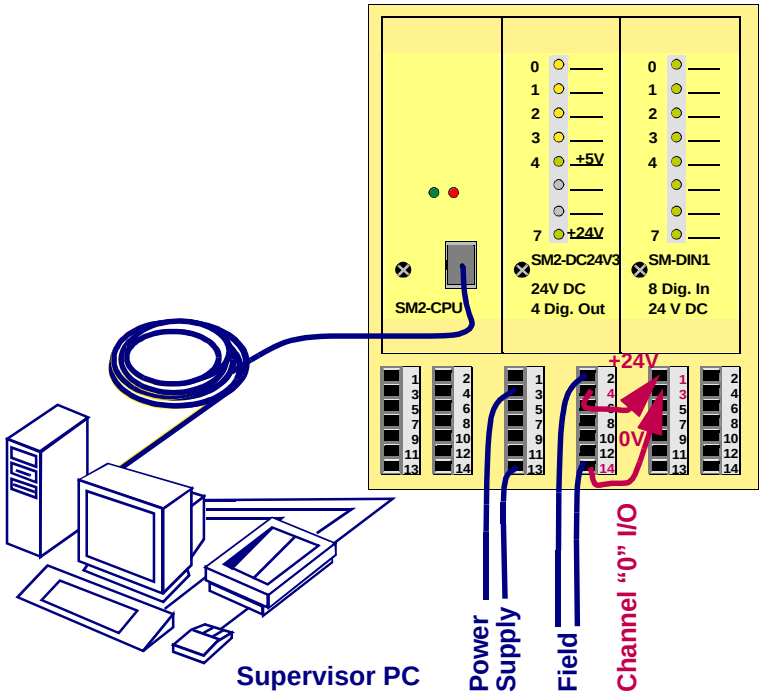
By doing so, pin 4 of screw terminal block B2, which is the channel “0” output terminal of the SM2-DC24V3 module, can now be driven by the +24 V field signals.

If you do not have a 24 V field signalling circuit to connect to your SMART2 system, you may instead bridge pins 3 and 2 for the +24 V signal and pins 13 and 14 for the relative grounding.

The remaining output channels are not used in this start-up example.

1.8.7 Step 6: Connecting the SM-DIN1

Figure 1-10: SMART2 Starter Kit Installation — Step 6



The SM-DIN1 module has 8 digital 24 V inputs. In order to drive its channel “0” input, please connect pin 4 of the screw terminal block B.2, which is the channel “0” output terminal of the SM2-DC24V3 module, to pin 1 of screw terminal block C.1, which is the channel “0” input terminal of the SM-DIN1 module.

The remaining input channels are not used in this start-up example.

At this point you may power up your supervisor PC and your SMART2 system.



High Voltage!

Before powering up, please check once again that your system is configured correctly. Make sure also that your serial interface is linked to your SMART2 system.

1.8.8 Step 7: Getting Started with Software

After having completed steps 1 to 6, you may move on to loading the software. To get started with the ISaGRAF PLC software supplied with your SMART2 Starter Kit you will need a 486-PC or better with a CD-ROM drive running under Windows 95 or Windows NT. Please proceed in the following order:

Installing ISaGRAF

- Install ISaGRAF from the “CJI” CD-ROM supplied with your SMART2 system or Starter Kit.
- *If running under Windows NT:*
  - log in as “Administrator”;
  - for target communication, write the string `NT=1` into section `[WS001]` of your ISA.INI file.
- Re-boot your computer.

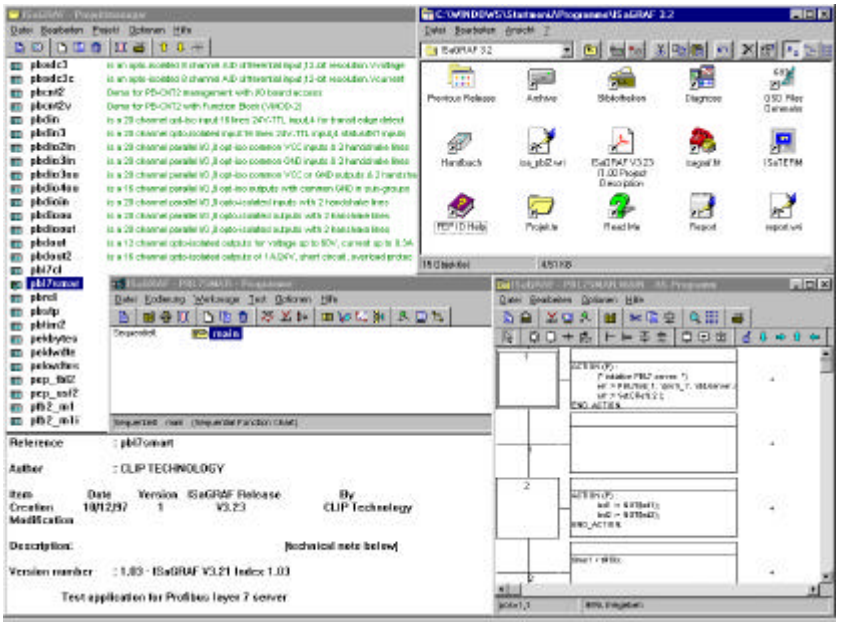
Starting and Working with ISaGRAF

- Start ISaGRAF by double-clicking the icon “Projects” in your ISaGRAF Windows Group. The ISaGRAF Group Manager will open.
- Use the pull-down menu to open “File – Project Group”. A dialog box will open.
- Select directory `[InstallationDir]\apl_pep`.
  For example, select project group `pep_iap`. All projects contained by

this group will appear.

- Double-click the project `PBDIN`. A list will appear showing all programs belonging to this project.
- Double-click one of the programs listed. The selected program will be opened in the Program Editor.
- Consult your ISaGRAF on-line help for further information and assistance.

Figure 1-11: ISaGRAF Man-Machine Interface



Legend:

Top right: WINDOWS 95 window with the installed ISaGRAF icons;

Left: Project Manager, the starting point for working on any project;

Center left: open project with a survey of all programs contained within it;

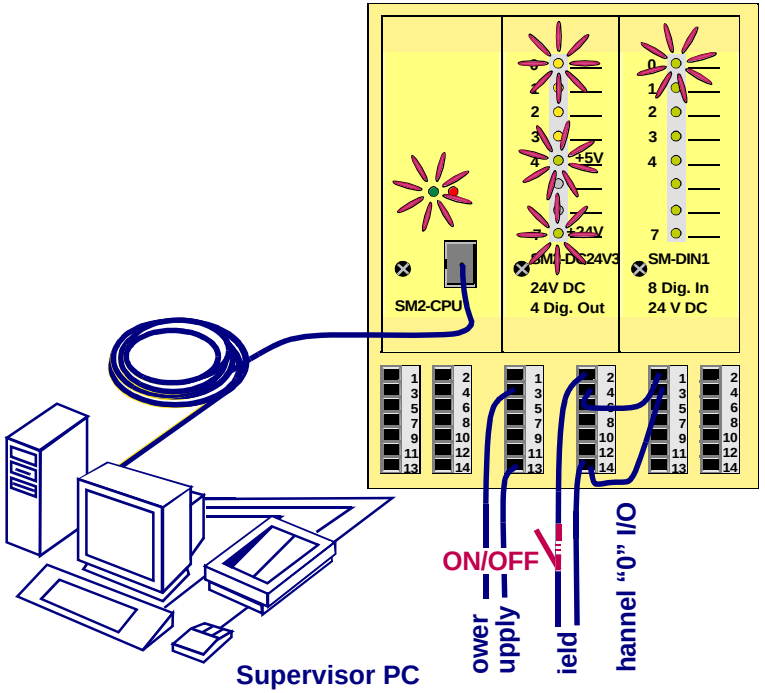
Bottom right: Program Editor with open program.

After having completed the first steps with ISaGRAF, you may try to switch to the on-off positions the output channel “0” of the SM2-DC24V3. If you have carried out all the installation steps correctly, the following LEDs should now be illuminated.

Table 1-9: SMART2 Starter Kit Installation — LEDs

SM2-DC24V3 Output Channel “0” Status	SM2-CPU	SM2-DC24V3	SM-DIN1
Unpowered	GREEN	“4”, “7”	—
Powered	GREEN	“0”, “4”, “7”	“0”

Figure 1-12: SMART2 Starter Kit Installation — Step 7



If the results of your SMART2 Starter Kit installation are different from those described in table 1-7, please check whether you have correctly carried out steps 2 to 7 of the installation.

If necessary, disconnect your system completely, extract the SM modules from their slots and repeat steps 1 to 7.



High Voltage!

Before disconnecting your applications and disassembling your SMART2 Starter Kit, please make sure the system power supply has been disconnected. Failure to do so may damage your modules/system.

In the event that the results are still different from those described, please consult the following chapters of this manual:

- **Introduction**
- **SMART2-BASE**
- **SM2-CPU**
- **SM2-DC24V_**
- **SM-DIN1**

In order to verify whether your installation environment complies with the general requirements of a SMART2 system.

This page was intentionally left blank.

SMART2-BASE

© PEP Modular Computers

This page was intentionally left blank.

Table of Contents

2.1 Basic Concept of the SMART2-BASE 2-5
2.1.1 Base/Extension Concept 2-5
2.1.2 Power Supply Concept 2-7
2.1.3 Basic Concept of the CPU/Fieldbus Modules ... 2-7
2.2 Main Features 2-9
2.2.1 Key Functions 2-9
2.2.2 System Parameters 2-9
2.2.3 Specification 2-10
2.2.4 Board Overview 2-11
2.2.5 SMART2-BASE Overall View 2-12
2.2.6 Functional Description 2-13
2.3 SMART2-BASE Interfaces 2-14
2.3.1 SMC Interface 2-14
2.3.2 SM Interface 2-15
2.3.3 SPI Extension Interface 2-15
2.4 Configuration 2-16
2.5 Pinouts 2-17
2.5.1 SMC Socket Pinout 2-17
2.5.2 SM Socket Pinout 2-18
2.5.3 Screw Terminals 2-19
2.5.4 SPI Extension IN and OUT Connectors 2-20

2.6 Compatibility with SMART I/O Systems 2-20
2.7 Software 2-22
2.7.1 ISaGRAF 2-22
2.7.2 'C' Programming 2-22

2. SMART2-BASE

The modularity of the SMART2-BASE makes it a multiple-use carrier unit capable of containing any of the following functions.

Within a SMART2 System:

- main carrier containing the CPU and power supply modules
- SMART2 extension carrier containing any I/O module

Within a SMART I/O System:

- SMART I/O extension carrier containing I/O modules (see chapter 5.6)

2.1 Basic Concept of the SMART2-BASE

2.1.1 Base/Extension Concept

Due to the enhanced modularity concept of SMART2, SMART2-BASE was developed as a basic unit to be used in two different ways, depending on whether a CPU module is present on the unit or not. Generally speaking, in a SMART2 system there must be one SMART2-BASE unit carrying a CPU module. This unit becomes the intelligent main unit of the system, into which up to two other SM modules may be fitted to provide I/O and related functions.

In addition, other SMART2-BASE units may be installed as extensions of the main unit, each capable of housing up to three SM modules per unit with supplementary tasks. The maximum number of extension units envisaged in a

system is four. Thus, a system may comprise a maximum of 1+4 SMART2-BASE carrier units..

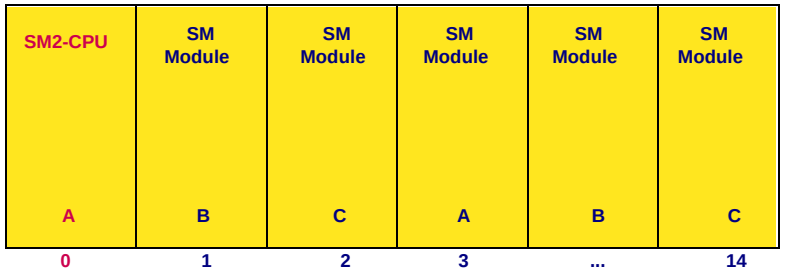
Note:

In a system, only one SMART2-BASE unit may be provided with a CPU module. All other possible SMART2-BASE units must be without such a module, in order to avoid controller conflicts with the first base unit.

PEP Advantage

The addition of other SMART2-BASE units for extension purposes is not compulsory. It depends only on the customer's specific configuration needs.

Figure 2-1: Typical SMART2-BASE Base/Extension System



Legend:

.. SMART2-BASE used as a main controller unit;
¦ SMART2-BASE used as an extension unit.

Terminology

The SMART2-BASE slots are numbered and lettered in the following way:

- *"0" to "14":* absolute numbering of all possible slots in a system, where:
  - "0" is the system's CPU carrier slot
  - "1" to "2" are I/O slots of the main system unit;
  - "3" to "5", "6" to "8", "9" to "11" and "12" to "14" are I/O slots of possible extension units.
- *"A" to "C":* relative numbering (lettering) of each single unit, independent of whether the unit is used as a main unit or an extension unit.

2.1.2 Power Supply Concept

In order to save space and to increase further the flexibility of the SMART2-BASE unit, the DC/DC power supply unit is no longer a fixed part of the SMART2 design, but has taken on the appearance of any of the *PEP Modular Computers'* SMART modules — SM2-DC24V_.



PEP Advantage

Due to the SMART2 philosophy of maximum flexibility and modularity, any external, third party DC/DC product meeting the 5V specifications and system power requirements may be used as a power supply unit. In this case, the supply input has to come across the left SPI extension input cable of the controller module front connector interface.

However, the most appropriate way to provide a complete power supply solution is by using a *PEP* SM-DC24V_ module.

All details concerning the SM2-DC24V_ like, for instance, positioning of the module in the SMART2-BASE unit, are described in the “SM2-DC24V_” chapter of this manual.

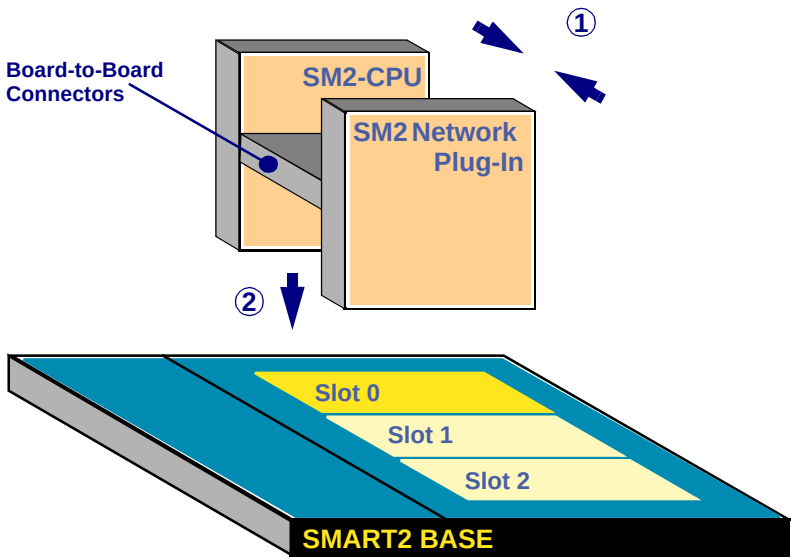
2.1.3 Basic Concept of the CPU/Fieldbus Modules

One objective of the SMART2 developers was extended modularity in the fieldbus selection. This goal has been attained and provides a solution which allows easy adaptation of any required fieldbus interface to the controller module of the system.

This controller module, the SM2-CPU, has the conventional format of any of the SMART modules. In combination with one of the possible fieldbus modules it represents a practical system intelligence and communication unit which can be arranged most economically in a single module slot.

However, the SM2-CPU is a completely autonomous CPU board which may also be used on your SMART2-BASE unit without the fieldbus “half-module”.

Figure 2-2: Typical SM2-CPU/Network Module “Sandwich”



2.2 Main Features

2.2.1 Key Functions

The SMART2-BASE key functions/features are as follows:

- Carrier unit for CPU, fieldbus, power supply and I/O modules;
- One design, two units: SMART2 main + extension carrier;
- Compatible with SMART I/O and SMART2 modules;
- 25% system space saved with 56 digital I/O’s compared to SMART I/O;
- Up to 112 I/O’s.

2.2.2 System Parameters

Table 2-1: SMART2-BASE System Parameters

SM2-BASE	System Parameters
Number of Carrier Units	Max. 5 SMART2 units in a system
Number of Slots	Max. 15, including 1 CPU slot
Number of I/O's:	max. 112 (with 14 x 8 I/O's per slot typically)
Power Supply: • Voltage • Max. Current • Typ. System requirement	 5 V DC ±2% Max. 3 A 1 A
Power Supply: • Base Unit • Extension Unit	 SM2-DC24V_; SM2-CPU (ext. DC/DC unit) Extension connector cable
Mounting	Modular DIN rail or wall mounting

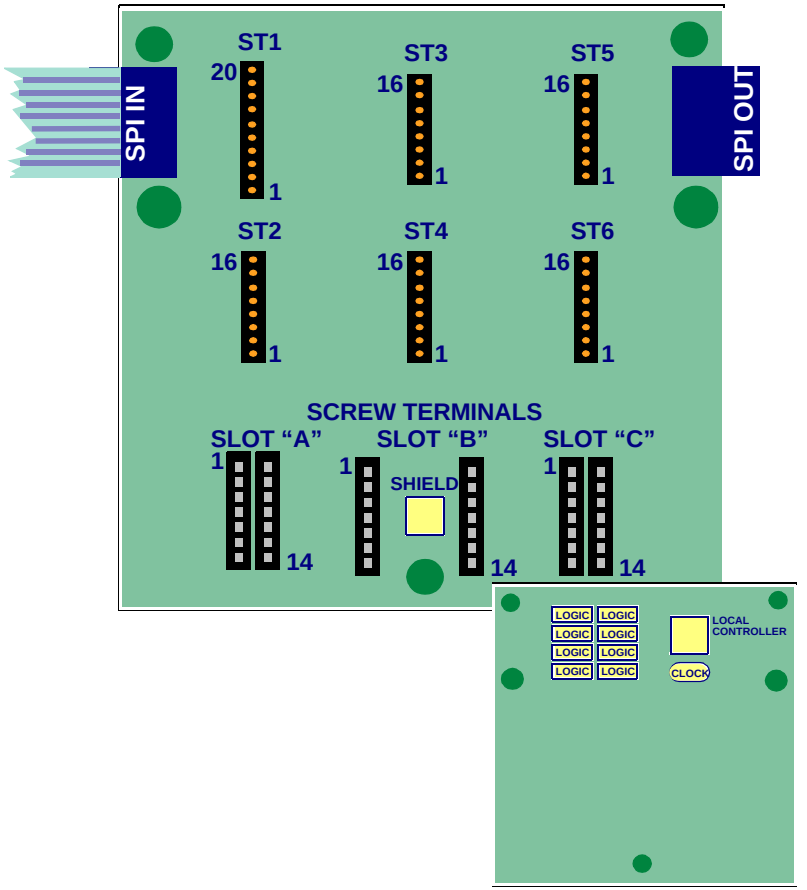
2.2.3 Specifications

Table 2-2: SM2-BASE Technical Specifications

SM2-BASE	Specifications
Number of Slots	3 SM slots; first slot includes controller/fieldbus option
Slot Configuration: • Screw Terminal Pins per Slot • I/O's per Slot	 Max. 14 pins per slot Max. 8 I/O's per slot
I/O Current/Voltage	Max. 3 A/max. 125 V AC
Controller	MC68HC705C8A
Controller Data: • Speed • Firmware	 4 MHz *PEP* SMART2 Firmware, Ed. 2.0 of 23 Jun 97
Interfaces	• 1 SMART I/O-compatible Extension Input • 1 SMART I/O-compatible Extension Output
Interface Speed	• Max. 2 MHz serial, SPI-based • Max. 100 µs parallel access
Power Consumption	Typically 5 mA at 5 V DC (carrier unit only)
Temperature: • Industrial • Extended	 0°C to +70°C -40°C to +85°C
Operating Humidity	0% to 95%, non-condensing

2.2.4 Board Overview

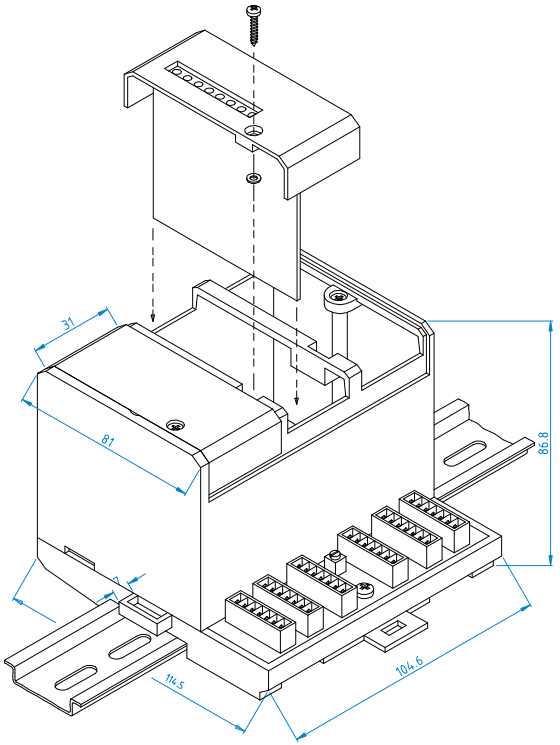
Figure 2-3: SMART2-BASE Schematic Board Layout — Front View and (Reduced) Reverse View



PEP Advantage

Connector ST1 of a SMART2-BASE unit (slot “A”) is provided with 20 pins, all other connectors of the board with 16. Pins 1, 2, 19 and 20 of ST1 are used by the CPU when the SMART2-BASE works as a main controller unit. If used as an extension unit, these pins remain unused.

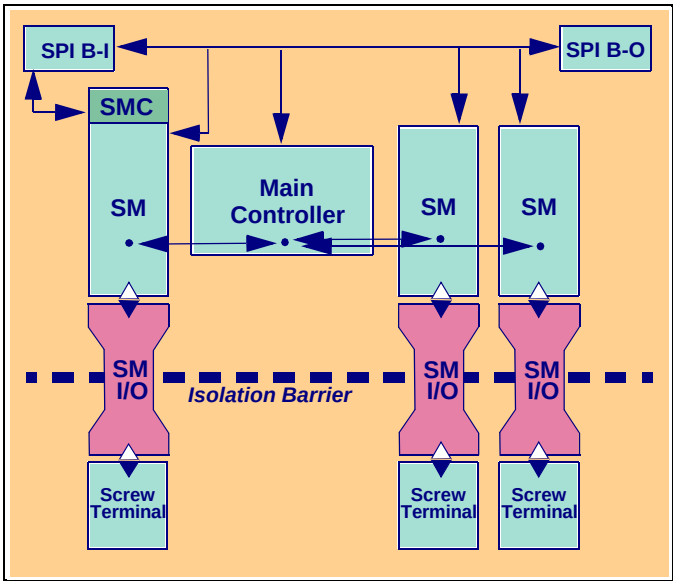
2.2.5 SMART2-BASE Overall View



2.2.6 Functional Description

Block Diagram

Figure 2-4: SMART2-BASE Block Diagram



Legend:

◀▷	SMART2-BASE/SM2 Module/SMART2-BASE I/O flow
→	SMART2-BASE internal flow
SMC	Via ST1 pins 1, 2, 19, 20 in case the SMART2-BASE is used as a controller main unit (see chapter 2.1.1)
SM	SMART I/O or SMART2 module/module port (see chapters 2.3.1 and 2.3.2)
SPI	SPI Bus. B-I = Bus IN, B-O = Bus OUT (see chapter 2.3.3.)

2.3 SMART2-BASE Interfaces

On the SMART2-BASE, no matter whether it is used as a SMART2 main controller or extension unit, we will find 3 different types of interfaces. These are the SMC, SM and SPI Extension interfaces.

The +5 V DC supply will also be described in the following chapters, although it is not strictly an input/output function.

2.3.1 SMC Interface

With the SMART2-BASE SMC port an intelligent SMART2 module may be installed to control the complete SMART2 system.

SM Interface

Except for the SMC extension described in chapter 2.3.1, all SMART2-BASE ports from “A” to “C” are defined as standard SM socket interfaces consisting of two separated standard 8-pin rows.



PEP Advantage

Strictly speaking, the SM interface should be named “SM/SM2” interface, since both SMART I/O and SMART2 modules are supported indiscriminately. However, for the sake of simplicity, this interface is called “SM” interface throughout this manual.

2.3.2 SPI Extension Interface

The SPI Bus IN socket is placed on the left side of the SMART2-BASE unit. A 10-wire flat cable connection is soldered to the connector, with I/O functions towards the logically preceding extension unit (“n-1”) or, if preceded by a main controller unit, to the main controller unit.

The SPI Bus OUT connector is a standard right-angle 2x5-pin connector placed on the right side of the SMART2-BASE unit, with I/O functions towards the subsequent extension unit (“n+1”).

If compared to the interfaces of the existing SMART-EXT extension unit, the SMART2-BASE SPI extension interfaces do not differ in any functional way. Furthermore, they are fully software compatible with the SMART-EXT extension unit. All signals needed and software protocols are available as before.

2.4 Configuration



PEP Advantage

The SMART2-BASE does not require any jumper/bridge configuration by the customer. The only hardware operations required are to plug in the power supply cable, to connect the SPI bus connectors where one or more SMART2 extension units are foreseen, and to fit in the desired SM CPU, fieldbus, I/O and or power supply modules.

At power-up or after a “Reset” condition, the system firmware starts with a default setting of all parallel ports. This default is naturally the input mode for each I/O point. Together with the pull-ups this means a logic high level for each I/O pin.

This state remains stable, until the user starts to configure the I/O registers by software.

2.5 Pinouts



High Voltage!

Dangerous voltages may be present at the terminals.

2.5.1 SMC Socket Pinout

Table 2-3: SMART2-BASE SMC Socket – Digital Side

Pin 1	*CM*	***Pin 2***	*_SCS*
Pin 3	D0	**Pin 4**	D1
Pin 5	D2	**Pin 6**	D3
Pin 7	D4	**Pin 8**	D5
Pin 9	D6	**Pin 10**	D7
Pin 11	System GND	**Pin 12**	Serial RxD
Pin 13	System VCC	**Pin 14**	Serial RxD / TxD
Pin 15	_CS Slot	**Pin 16**	Serial CLKw
Pin 17	Reset	**Pin 18**	SLOT “A” _Interrupt
Pin 19	*SCLK*	***Pin 20***	*STXD*

Italics: SMC extension of a regular SM socket.

Table 2-4: SMART2-BASE SMC Socket – Process Side

Pin 1	Screw Term. 13	**Pin 2**	Screw Term. 13
Pin 3	Screw Term. 1	**Pin 4**	Screw Term. 2
Pin 5	Screw Term. 3	**Pin 6**	Screw Term. 4
Pin 7	Screw Term. 5	**Pin 8**	Screw Term. 6
Pin 9	Screw Term. 7	**Pin 10**	Screw Term. 8
Pin 11	Screw Term. 9	**Pin 12**	Screw Term. 10
Pin 13	Screw Term. 11	**Pin 14**	Screw Term. 12
Pin 15	Screw Term. 14	**Pin 16**	Screw Term. 14

2.5.2 SM Socket Pinout

Table 2-5: SMART2-BASE SM Socket – Digital Side

Pin 1	P1	**Pin 2**	P2
Pin 3	P3	**Pin 4**	P4
Pin 5	P5	**Pin 6**	P6
Pin 7	P7	**Pin 8**	P8
Pin 9	System GND	**Pin 10**	Serial RxD
Pin 11	System VCC	**Pin 12**	Serial TxD
Pin 13	_CS Slot	**Pin 14**	Serial CLK
Pin 15	_Reset	**Pin 16**	Slot _Interrupt

Table 2-6: SMART2-BASE SM Socket – Process Side

Pin 1	Screw Term. 13	**Pin 2**	Screw Terminal 13
Pin 3	Screw Term. 1	**Pin 4**	Screw Terminal 2
Pin 5	Screw Term. 3	**Pin 6**	Screw Terminal 4
Pin 7	Screw Term. 5	**Pin 8**	Screw Terminal 6
Pin 9	Screw Term. 7	**Pin 10**	Screw Terminal 8
Pin 11	Screw Term. 9	**Pin 12**	Screw Terminal 10
Pin 13	Screw Term. 11	**Pin 14**	Screw Terminal 12
Pin 15	Screw Term. 14	**Pin 16**	Screw Terminal 14

2.5.3 Screw Terminals

Each of the three SMART2-BASE screw terminals consists of a double-row terminal board having a pin-to-pin distance of 3.81 mm and providing 14 free connections per I/O slot. The two 7-terminal rows of each connector are separated as shown in chapter 2.2.4, “Board overview”.

The terminal pinout depends on the type of SM modules, and the pinout of the relevant module(s) should be looked up in the appropriate section of this manual.

Note:

The seven-pole terminal blocks you may have been using already with SMART-BASE and SMART-EXT are different in size and, therefore, cannot be used on the SMART2-BASE.

The SMART2-BASE terminal blocks have a specified maximum operating voltage of 125V AC at 3A. Therefore for all SM modules specified up to 264V AC, such as SM-ACO1, SM-ACI1 or SM-REL1, a SMART-EXT carrier unit must be used.

2.5.4 SPI Extension IN and OUT Connectors

Table 2-7: SMART2-BASE SPI Extension Connectors

Pin 1:	System GND	**Pin 2:**	Serial RxD
Pin 3:	System GND	**Pin 4:**	Serial TxD
Pin 5:	Serial Ext. _Select	**Pin 6:**	Serial CLK
Pin 7:	System VCC	**Pin 8:**	_Reset
Pin 9:	System VCC	**Pin 10:**	Extension _Interrupt

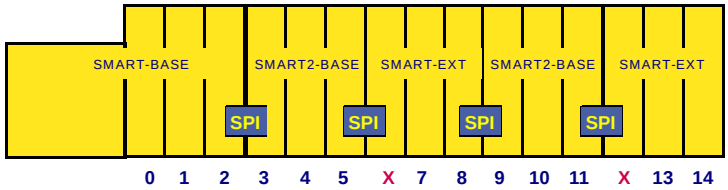
2.6 Compatibility with SMART I/O Systems

The SMART2-BASE and SMART-EXT units can be used together as extensions in a single SMART I/O system.

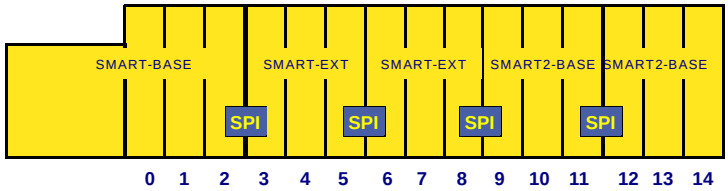
There is only one point that should be taken into consideration. The SMART-EXT slot identification numbers may start only with odd numbers. The reason for this is the fact that there are some combinations which should not be used. The following example shows how to correctly configure a mixed system:

Figure 2-5: SMART2-BASE in a SMART I/O System

WRONG CONFIGURATION.



RIGHT CONFIGURATION.



Legend:

X Slot identification problems;
0...14 No slot identification problems.

Case 1: Although the system is not configured very well, it will work without any problems with the SMART2 library. However, older systems may have problems finding the maximum number of slots, since the SMART I/O slot count maximum is 11.

Case 2: Correct arrangement. There are no missing slot numbers which may confuse the system in the process of slot count identification.

2.7 Software

2.7.1 ISaGRAF



Note:

For users of the ISaGRAF software it is not necessary to program the SMART2 interfaces, since they are completely controlled by ISaGRAF. Therefore, no such programming is envisaged for the SM2-CPU.



PEP Advantage

All necessary and/or useful information on ISaGRAF is available directly via software in the ISaGRAF on-line help program.

2.7.2 'C' Programming

Application developers using 'C' under OS9 don't need to access hardware and/or I/O devices directly, since libraries are provided giving easy access to all needed hardware functions.

The libraries are structured like the ones used for SMART I/O systems. A detailed description can be found in the corresponding file headers. The files are located in the following directories:

```
/dd/BSP/SMART/DEFS
/dd/BSP/SMART/<ioboard>/DEFS
```

Demo applications are located as follows:

```
/dd/APPLIC/SMART
```

Please note that the libraries are included inside the *PEP* BSP SMART I/O 2 only.

Basic Modules

© PEP Modular Computers

This page was intentionally left blank.

Table of Contents

3.1 SM2-CPU ..... 3 - 8
3.1.1 Installation ..... 3 - 8
3.1.2 Main Features ..... 3 - 9
3.1.3 Specifications ..... 3 - 10
3.1.4 Board Overview ..... 3 - 11
3.1.5 Front Panel ..... 3 - 13
3.1.6 Functional Description ..... 3 - 13
3.1.7 Block Diagram ..... 3 - 14
3.1.8 SM2-CPU Interfaces ..... 3 - 15
3.1.9 Configuration ..... 3 - 15
3.1.10 Pinouts ..... 3 - 16
3.1.11 Software ..... 3 - 17
3.1.11.1 ISaGRAF ..... 3 - 17
3.1.11.2 'C' Programming ..... 3 - 18
3.2 Network Plug-In Modules ..... 3 - 19
3.2.1 SM2-DP ..... 3 - 21
3.2.1.1 Specifications ..... 3 - 22
3.2.1.2 Board Overview ..... 3 - 23
3.2.1.3 Front Panel View ..... 3 - 24
3.2.1.4 Functional Description ..... 3 - 24
3.2.1.5 Block Diagram ..... 3 - 25
3.2.1.6 Configuration ..... 3 - 25
3.2.1.7 Pinouts ..... 3 - 26

3.2.1.8 Hardware Hints ..... 3 - 26
3.2.1.9 ISaGRAF ..... 3 - 28
3.2.1.10 'C' Programming ..... 3 - 28
3.2.2 SM2-CAN ..... 3 - 29
3.2.2.1 Specifications ..... 3 - 30
3.2.2.2 Board Overview ..... 3 - 31
3.2.2.3 Front Panel Layout ..... 3 - 32
3.2.2.4 Functional Description ..... 3 - 32
3.2.2.5 Block Diagram ..... 3 - 33
3.2.2.6 Configuration ..... 3 - 33
3.2.2.7 Pinouts ..... 3 - 34
3.2.2.8 Hardware Hints ..... 3 - 34
3.2.2.9 ISaGRAF ..... 3 - 36
3.2.2.10 'C' Programming ..... 3 - 36
3.2.3 SM2-ETH ..... 3 - 37
3.2.3.1 Specifications ..... 3 - 38
3.2.3.2 Board Overview ..... 3 - 39
3.2.3.3 Front Panel Layout ..... 3 - 40
3.2.3.4 Functional Description ..... 3 - 40
3.2.3.5 Block Diagram ..... 3 - 41
3.2.3.6 Configuration ..... 3 - 41
3.2.3.7 Pinouts ..... 3 - 42
3.2.3.8 ISaGRAF ..... 3 - 42
3.2.3.9 OS-9 Programming ..... 3 - 43

3.3 SM2-DC24V_ ..... 3 - 44
3.3.1 Installation ..... 3 - 45
3.3.2 Main Features ..... 3 - 46
3.3.3 Board Overview ..... 3 - 49
3.3.4 Front Panel Layout ..... 3 - 50
3.3.5 Functional Description ..... 3 - 50
3.3.6 Block Diagram ..... 3 - 51
3.3.7 Configuration ..... 3 - 52
3.3.8 Pinouts ..... 3 - 53
3.3.9 Connection ..... 3 - 54
3.3.10 Output Circuit ..... 3 - 55
3.3.11 ISaGRAF ..... 3 - 55
3.3.12 'C' Programming ..... 3 - 56

This page was intentionally left blank.

3. SMART2 Basic Modules

The SMART2 Basic Modules described in this chapter are the following:

Controller Modules

One controller module is capable of managing all SMART2 system controller functions. This module, SM2-CPU, is described in chapter 3.1 of this manual.

Network Plug-In Modules

Depending on the specific requirements of your SMART2 system, different plug-in modules may be used as alternatives for different performance requirements:

- SM2-DP (PROFIBUS)
- SM2-CAN (CAN)
- SM2-ETH (Ethernet)

These modules are described in chapter 3.2 of this manual.

Power Supply Modules

SMART2-BASE may be powered via an external AC/DC supply unit or, more suitably, via a dedicated SM2 module designed for this purpose and called SM2-DC24V_.

This module, which exists in three different variants with differing power outputs, is described in chapter 3.3 of this manual.

3.1 SM2-CPU

The SM2-CPU belongs to the series of *PEP Modular Computers'* controller modules, which are part of the SMART2 concept. The module supports all functions needed for the control of a SMART2 system.

Basic Concept of the Module

The SM2-CPU has the conventional format of any of the *PEP Modular Computers'* SMART modules. In combination with one of the possible fieldbus modules it represents a practical system intelligence and communication unit which can be arranged most economically in a single module slot.

However, SM2-CPU is a completely autonomous CPU board which may also be used independently on your SMART2 BASE unit, i.e. without the fieldbus "half-module".

Software

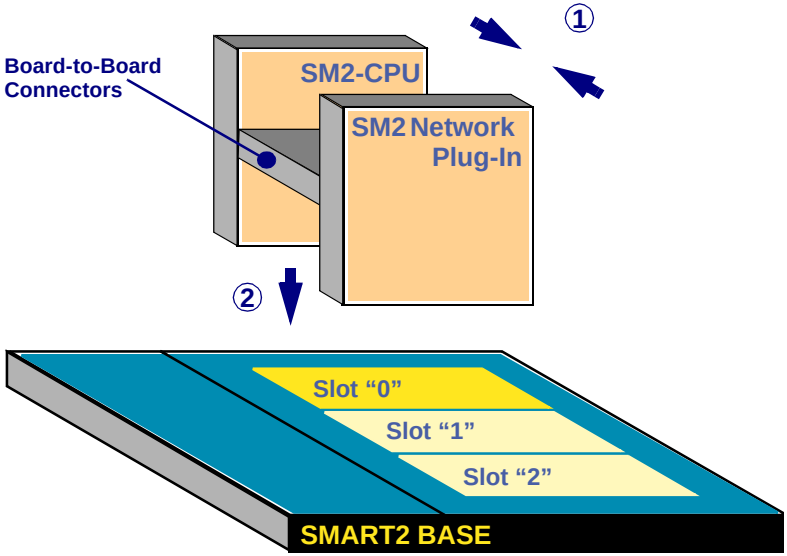
SM2-CPU supports both ISaGRAF and OS9 software. It does not require any jumper setting.

3.1.1 Installation

Due to the basic hardware concept of the SM2-CPU module, it would usually be used together with a SMART2 fieldbus module. Normally both modules are supplied as a single controller/network unit, ready to be mounted into slot "0" of your SMART2 BASE. However, disassembly and re-assembly of the two "carrier module" and the plug-in module is possible by proceeding as follows:

1. Assemble the SM2-CPU and the network module by fitting the corresponding male/female board-to-board connectors into each other;
2. Mount the SM2-CPU/network controller assembly into slot "0" of your SMART2-BASE.

Figure 3-1: Typical SM2-CPU/Network Module "Sandwich"



3.1.2 Main Features

The SM2-CPU module envisages the following key features/functions:

- SMART2 system control;
- Network module administration;
- Operating system monitoring.

3.1.3 Specifications

Table 3-1: SM2-CPU Technical Specifications

SM2-CPU	Specifications
Controller:	Motorola 68LC302, 100-pin QFP package
Controller Speed	20 MHz
Power Consumption	Typically 130mA, max. 200mA at 5V DC and 20MHz
DRAM	512 kByte, option: 1.5 MByte
FLASH	1MByte, sector size 128 kByte
SRAM	256 kByte, 80 kByte Users RAM Disk
RTC	V3021, seconds - minutes - hours - years
Buffer Capacitor	Goldcap type. RTC and SRAM > 2 days at 25°C, typ. 5 days, battery option
ID Byte	None.
LEDs:	• Power ON, bootstrap loader ENABLED (green) • HALT (red)
Interfaces	• Serial RS232: *SCC1:* screw terminal, no HW handshaking support -> RTS signal for possible RS485 adapter; *SCC2:* full modem support; • Serial SPI master channel, max. 4 MHz (system default 1 MHz); • FPC connector to network module; • SMC connector to SMART2-BASE, slot “0”; • System supply connector +5 V DC, max. 3A
Temperature Range: • Industrial • Extended	 0°C to +70°C -40°C to +85°C
Operating Humidity	0% to 95%, non-condensing.
Weight	90 g (without network plug-in module)

3.1.4 Board Overview

Figure 3-2: SM2-CPU Schematical Board Layout — Front View

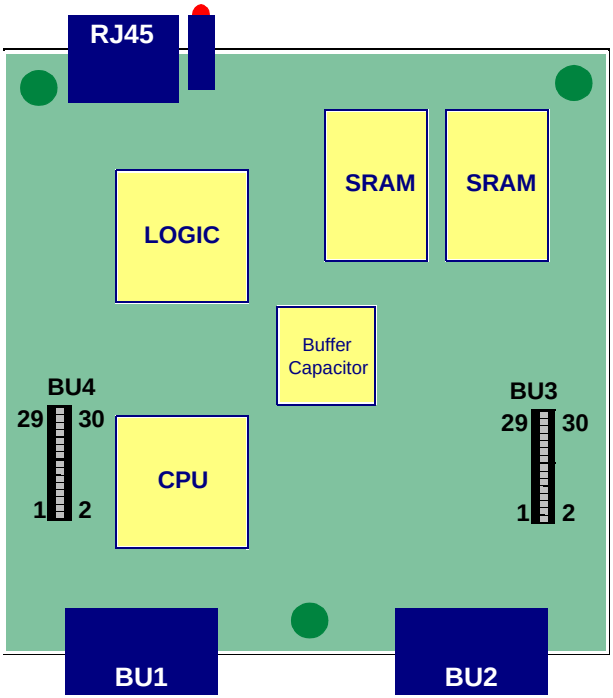
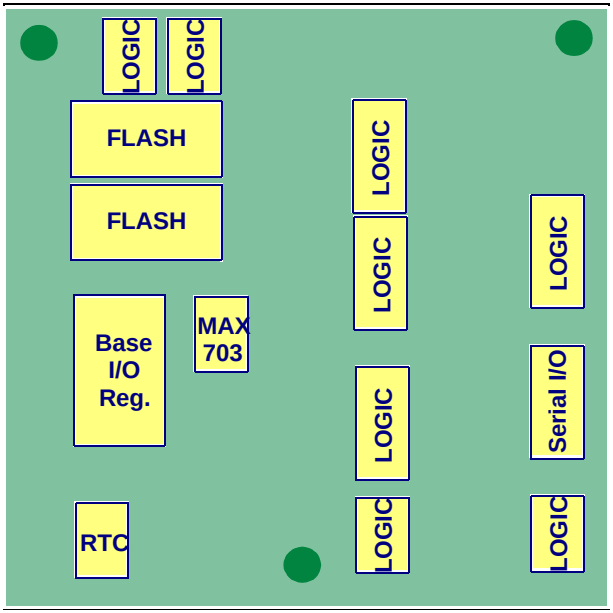
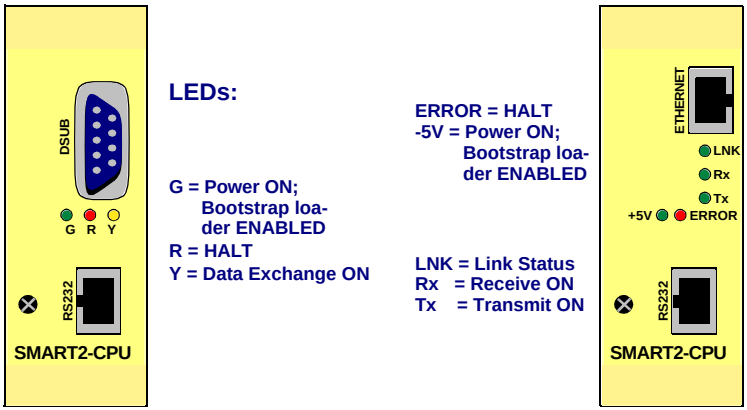


Figure 3-3: SM2-CPU Schematical Board Layout — Reverse View



3.1.5 Front Panel

Figure 3-4: Combined SM2-CPU/Network Module Front Panels



SM2-CPU plus SM2-DP or SM2-CAN

SM2-CPU plus SM2-ETH

3.1.6 Functional Description

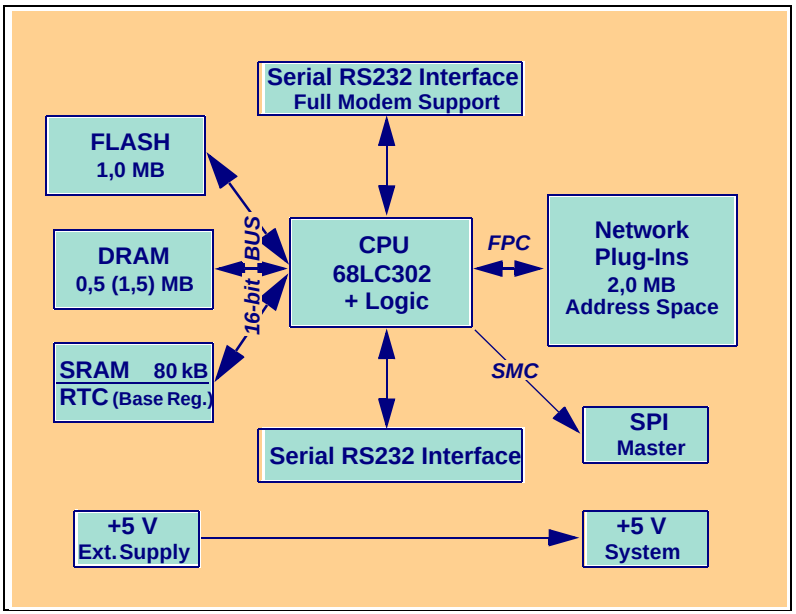
The SM2-CPU is designed to fulfill a series of control functions which can be summarized as follows:

- 1 MByte FLASH for application program storage;
- 0,5 (1,5) MByte DRAM for application tasks;
- 80 kByte SRAM for RAM disk support;
- Real Time Clock;
- SPI-master interface for SMART2 system control;
- RS232 interface, full MODEM support;
- RS232 interface (debug port);
- FPC network interface with 2,0 MByte of address space;
- Direct, feed-through +5V system supply.

The following block diagram shows the relationships between the CPU and the addressable board components.

3.1.7 Block Diagram

Figure 3-5: SM2-CPU Block Diagram



3.1.8 SM2-CPU Interfaces

On the SM2-CPU three different interfaces are found: an SMC interface, as well as an FPC and two RS232 connectors. These interfaces are described in the following chapters.

SMC Interface

This port enables the SM2-CPU to control the entire SMART2 system.

Serial Interfaces

The board is equipped with two RS232 serial interfaces for system monitoring tasks. The main port is a fully MODEM-supported interface with a RJ45 connector.

The second RS232 serial interface, which is available at the screw terminal port pins, supports only RxD and TxD lines and does not support hardware handshakes.

FPC Interface

This interface, originally designed as a “Fieldbus Processor Connector” interface, provides the access to a common memory interface structure for the data exchange between the controller module itself and a network module.

3.1.9 Configuration

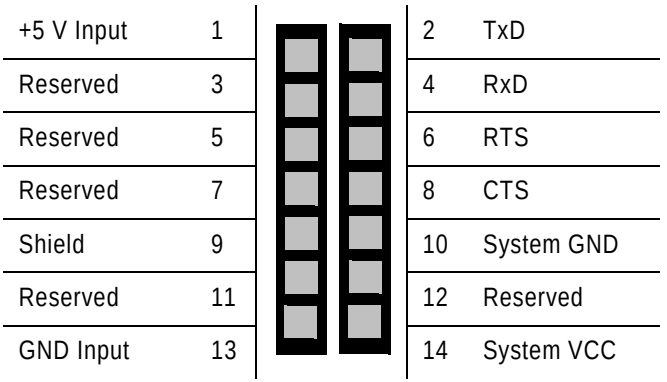
On the SM2-CPU no customer configuration setting is necessary. The default power-up setting of the main RS232 Baud rate is 9600 Baud.

3.1.10 Pinouts

Screw Terminal Blocks

The front connector, i.e. screw terminal block of a SMART2 module is made of a double row of seven pins that are separated as shown on the following page (for a complete overview of the a SMART2-BASE carrier unit see the figures 1-1 or 1-2).

Figure 3-6: Front Connector Pinouts



Warning!

The +5 V supply input has to be sourced with a TTL-Logic compatible 5V DC source. It is assumed that any external supply unit is fused or short-circuit protected. Please make sure that the system supply exists only once in the system.

RJ45 Connector BU5

Table 3-2:BU5 Pinouts

Pin 1	DSR	Pin 2	RTS
Pin 3	GND	Pin 4	TxD
Pin 5	RxD	Pin 6	DCD
Pin 7	CTS	Pin 8	DTR

3.1.11 Software

Software functions are relevant for the SM2-CPU concerning two different aspects, namely communication with

- SMART2-BASE;
- SMART2 network modules.

Since different fieldbus modules require slightly different software functions, these are described in the relevant manual chapters, while in the underlying chapter only the software functions concerning communication with SMART2-BASE are dealt with.

3.1.11.1 ISaGRAF

Note:

For users of the ISaGRAF software it is not necessary to program the SMART2 interfaces, since they are completely controlled by ISaGRAF. Therefore, no such programming is envisaged for the SM2-CPU.



PEP Advantage

All necessary and/or useful information on ISaGRAF is available directly via software in the ISaGRAF on-line help program.

3.1.11.2 'C' Programming

Application developers using C under OS9 don't need to access hardware and/ or I/O devices directly, since libraries will be provided giving easy access to all needed hardware functions.

The libraries are structured like the ones used for SMART I/O systems. A complete description can be found in the following directories:

```
/dd/BSP/SMART/DEFS
/dd/BSP/SMART/<ioboard>/DEFS
```

Demo applications are located as follows:

```
/dd/APPLIC/SMART
```

Please note that the libraries are included inside the *PEP* BSP SMART I/O 2 only.

3.2 Network Plug-In Modules

The SMART2 network plug-ins are slave interfaces designed for field I/O control and/or communication purposes. Depending on the requirements of your network system, the following plug-in modules are available and may be used alternatively:

Table 3-3: SMART2 Network Plug-In Modules

Module	Application Highlight
SM2-DP	Protocol: PROFIBUS DP • High-speed networking/large data volumes • Comfortable, high-level software language
SM2-CAN	Protocol: CANopen • Higher-distance networking • Stability with typical industrial field applications
SM2-ETH	Protocol: Ethernet • Higher-distance/high-speed networking/ large data volumes; • Comfortable, “API”-oriented operation for non-time-critical field applications and communication purposes.

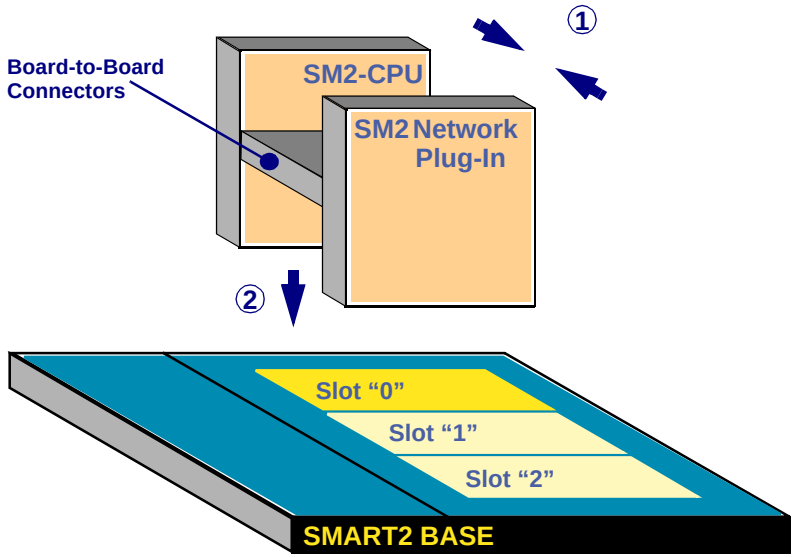
Installation

Any of the SMART2 network plug-ins may be chosen to represent a network processing and/or communication unit in combination with the SM2-CPU intelligent controller module, to be arranged most economically in a single module slot.

Normally both sub-units are supplied as a single controller/network unit, ready to be mounted into slot “0” of your SMART2-BASE. However, disas-

sembly and re-assembly of the two half-modules is possible proceeding as follows:

Figure 3-7: Typical SM2-CPU/Network Module “Sandwich”



3. Assemble the SM2-CPU and the SMART2 network controller module by fitting the corresponding male/female board-to-board connectors into each other;
4. Mount the SM2-CPU/network controller assembly into slot “0” of your SMART2 BASE.

3.2.1 SM2-DP

The SM2-DP network controller module is a PROFIBUS DP fieldbus slave interface capable of high Baud rates, i.e. up to 12 MBd, and designed for exchanging large data volumes with a comfortable, high-level software language. Therefore, it is particularly suitable for processor-to-processor data exchange applications.

The layer 2 functions are implemented by hardware. The fieldbus initialization software is stored in the on-board memory. Communication with a controller unit (SM2-CPU or SMART-BASE) is accomplished via a shared-memory interface. An LED indicates that data exchange is taking place on the PROFIBUS.

Basic Concept of the Module

In combination with the new SM2-CPU controller module the SM2-DP represents a network controller and fieldbus communication unit which can be arranged most economically in a single module slot.



PEP Advantage

SM2-DP is just one of the possible fieldbus/ network modules that may be combined with the SM2-CPU controller module, depending on the specific functional requirements of your data exchange system.

Software

SM2-DP runs with a PROFIBUS standard driver. It does not require any programming, nor jumper setting. However, the possibility of programming the FLASH memory via the FPC interface is envisaged for updating purposes.

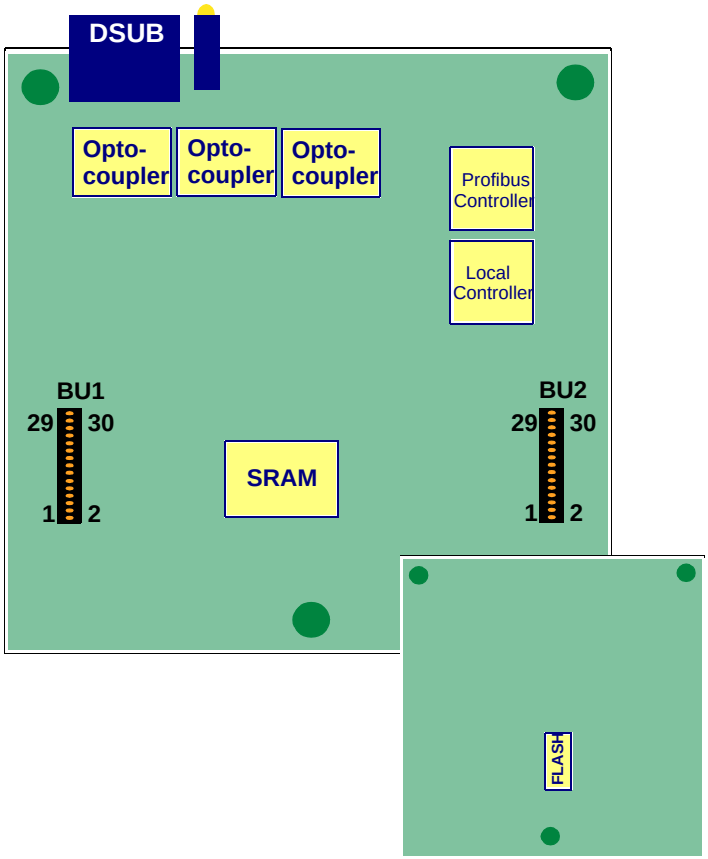
3.2.1.1 Specifications

Table 3-4: SM2-DP Technical Specifications

SM2-DP	Specifications
Controllers	• 80C31 controller at 24 MHz • 12 MBd SPC3 DP slave controller
Power Consumption	Typically 250 mA at 5 V DC and 24 MHz
Local Power-up/Reset	4.65 V sense level
Local Watchdog	1.6 s
FLASH	64 kByte
SRAM	32 kByte
LED	Data exchange ON (yellow)
ID Byte	None
Interfaces	• 9-pin DSUB serial PROFIBUS interface (12 MBd); • FPC connector to controller module;
PROFIBUS Isolation	1 kV DC
Temperature Range: Industrial Extended	 0°C to +70°C -40°C to +85°C
Operating Humidity	0% to 95%, non-condensing.
Weight	70 g

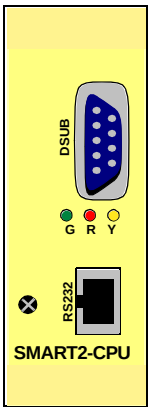
3.2.1.2 Board Overview

Figure 5-8: SM2-DP Schematical Board Layout — Front View and (Reduced) Reverse View



3.2.1.3 Front Panel View

Figure 5-9: Combined SM2-CPU/DP Front Panel Layout



LEDs:

G = Power ON; Bootstrap loader ENABLED
R = HALT
Y = Data Exchange

3.2.1.4 Functional Description

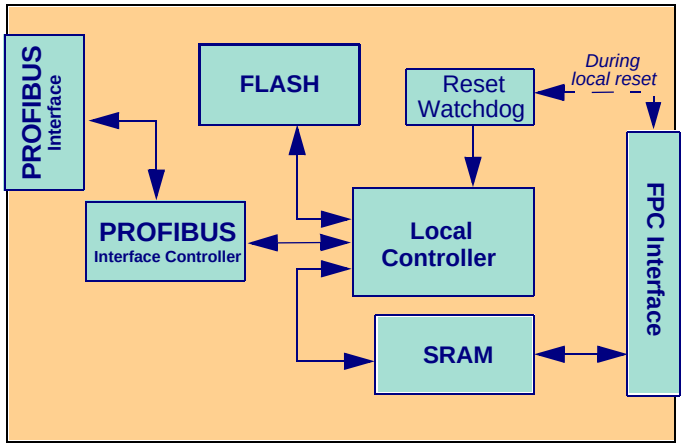
The SM2-DP plug-in module is provided with a 12 MBd PROFIBUS DP fieldbus interface controller and a 24 MHz local controller. An on-board FLASH memory contains the local firmware.

The local controller initializes the RS485 interface controller and handles the communication with the shared memory (FPC connection to the SM2-CPU module). The interface controller is opto-isolated from the system side. Its signals are connected to a 9-pin DSUB according to DIN/PROFIBUS specifications.

After power-up the local controller switches to reset mode. At this point, the CPU main controller starts the local firmware via the local controller. In the event of power failure and abnormal software hang-ups a local reset/watchdog restarts the controller.

3.2.1.5 Block Diagram

Figure 5-10: SM2-DP Block Diagram



3.2.1.6 Configuration

PEP Advantage

The SM2-DP does not require any customer configuration. The only hardware operation required is to plug the PROFIBUS cable into the 9-pin DSUB connector.

3.2.1.7 Pinouts

The PROFIBUS connector is a DIN-compliant 9-pin DSUB socket supplying a potential-free termination power.

Table 3-5: SM2-DP PROFIBUS Connector Pinouts

Pin 1	Shield	Pin 6	5 V
Pin 2	—	Pin 7	—
Pin 3	Positive data	Pin 8	Negative data
Pin 4	—	Pin 9	—
Pin 5	Ground		

3.2.1.8 Hardware Hints

Cable Shielding

The cable shielding has to be connected to the DSUB housing/pin 1. The housing and pin 1 are routed to the SM2-DP/CPU metal distance screws.



Note:

Please refer to the SMART2 “Shielding” chapter for all hints concerning practical shield routing.

Connector and Line Termination

The DSUB connector assures connection to termination power. No termination network is available on the SM2-DP itself.



Note:

The SM2-DP has no internal line termination which, therefore, must be installed at the ends of the network (ISO 11898).



Note:

The maximum allowed auxiliary termination power is 90 mA/5 V DC. Assuming a power supply voltage of +5V emanating from the PROFIBUS connector (pin 6), the following resistor value is recommended:

R_{DP} 124 ohm ± 2%, min. 0.25 W

3.2.1.9 ISaGRAF



Note:

For users of the ISaGRAF software it is not necessary to program the SMART2 interfaces, since they are completely controlled by ISaGRAF. Therefore, no such programming is envisaged for the SM2-DP.



PEP Advantage

All necessary and/or useful information on ISaGRAF is available directly via software in the ISaGRAF on-line help program.

3.2.1.10 'C' Programming

Application developers using C under OS9 don't need to access hardware and/or I/O devices directly, since libraries will be provided giving easy access to all needed hardware functions.

The libraries are structured like the ones used for SMART I/O systems. A detailed description can be found in the corresponding file headers. The files are located in the following directories:

```
/dd/BSP/SMART/DEFS
/dd/BSP/SMART/<ioboard>/DEFS
```

Demo applications are located as follows:

```
/dd/APPLIC/SMART
```

Please note that the libraries are included inside the *PEP* BSP SMART I/O 2 only.

3.2.2 SM2-CAN

The SM2-CAN plug-in module is a CAN slave interface designed for reliable field I/O control, capable of higher-distance networking, i.e. up to 1 km length, and is therefore particularly suitable for applications in the industrial and automotive field etc.

All layer 2 functions are implemented by hardware. Fieldbus initialization software is stored in the on-board memory, while communication with the controller unit (SM2-CPU or SMART-BASE) is accomplished via a shared-memory interface. A LED indicates that data exchange is taking place on the CAN bus.

Basic Concept of the Module

In combination with the new SM2-CPU controller module the SM2-CAN represents a network controller and fieldbus communication unit which can be arranged most economically in a single module slot.



PEP Advantage

SM2-CAN is just one of the possible fieldbus/ network modules that may be combined with the SM2-CPU controller module, depending on the specific functional requirements of your data exchange system.

Software

SM2-CAN runs with a CAN standard driver. It does not require any programming, nor jumper setting. However, the possibility of programming the FLASH memory via FPC interface is envisaged for updating purposes.

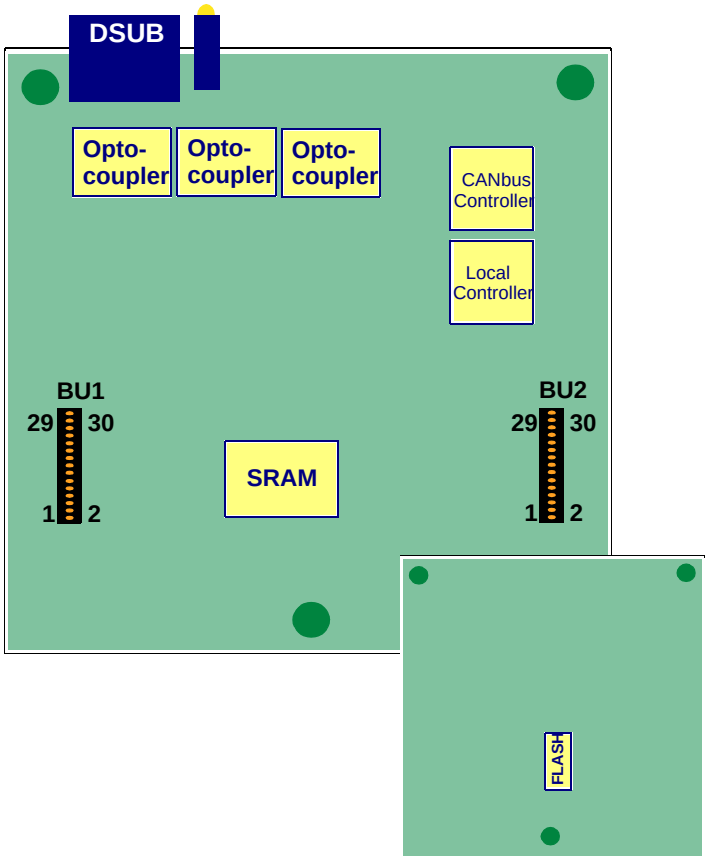
3.2.2.1 Specifications

Table 3-6: SM2-CAN Technical Specifications

SM2-CAN	Specifications
Controllers	• 80C31 controller at 24 MHz • 1 MBd SJA1000 CAN 2.0B slave controller
Power Consumption	Typically 290 mA at 5 V DC and 24 MHz
Local Power-up/Reset	4.65 V sense level
Local Watchdog	1.6 s
FLASH	64 kByte
SRAM	56 kByte
LED	Data exchange ON (yellow)
ID Byte	None
Interfaces	• 9-pin DSUB serial CAN interface (1 MBd); • FPC connector to controller module;
Network Performance	• max. 40 m at 1 MBd • max. 1000 m at 50 kBd
CAN Bus Isolation	1 kV DC
Temperature Range: • Industrial • Extended	 0°C to +70°C -40°C to +85°C
Operating Humidity	0% to 95%, non-condensing.
Weight	75 g

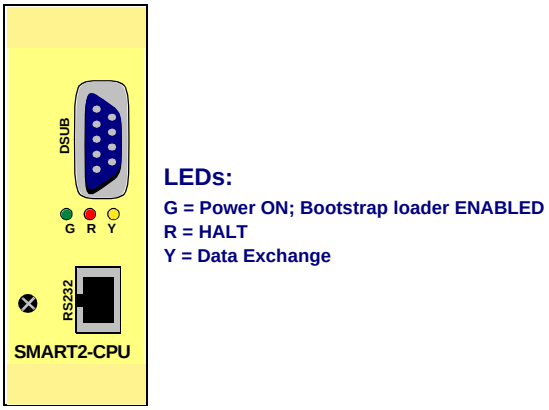
3.2.2.2 Board Overview

Figure 5-11: SM2-CAN Schematical Board Layout — Front View and (Reduced) Reverse View



3.2.2.3 Front Panel Layout

Figure 5-12: Combined SM2-CPU/CAN Front Panel Layout



3.2.2.4 Functional Description

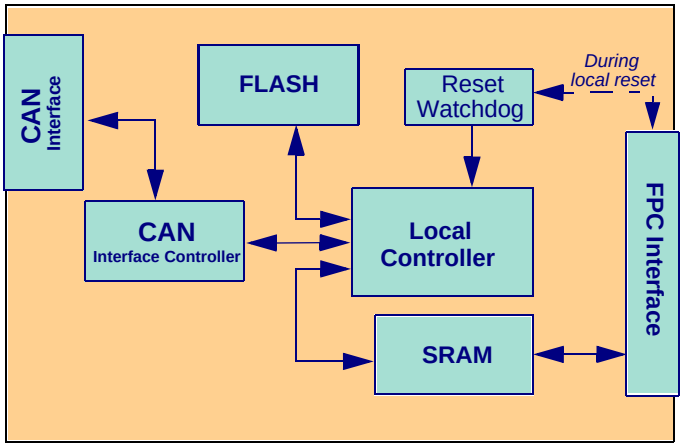
The SM2-CAN plug-in module is provided with a 1 MBd CAN fieldbus interface controller and a 24 MHz local controller. An on-board FLASH memory contains the local firmware.

The local controller initializes the RS485 interface controller and handles the communication with the shared memory (FPC connection to the SM2-CPU module). The interface controller is opto-isolated from the system side. Its signals are connected to a 9-pin DSUB according to CAN specifications.

After power-up the local controller switches to reset mode. At this point, the CPU main controller starts the local firmware via the local controller. In the event of power failure and abnormal software hang-ups a local reset/watchdog restarts the controller.

3.2.2.5 Block Diagram

Figure 5-13: SM2-CAN Block Diagram



3.2.2.6 Configuration

PEP Advantage

The SM2-CAN does not require any customer configuration. The only hardware operation required is to plug the CAN bus cable into the 9-pin DSUB connector.

3.2.2.7 Pinouts

The CAN connector is a 9-pin DSUB socket complying with CAN specifications and supplying a potential-free termination power.:

Table 3-7: SM2-CAN CANbus Connector Pinouts

Pin 1	—	Pin 6	Ground
Pin 2	Differential transceiver low	Pin 7	Differential transceiver high
Pin 3	Ground	Pin 8	—
Pin 4	—	Pin 9	*5 V**
Pin 5	Shield		

* Pin 9 is not connected to the on-board circuitry (optional).

3.2.2.8 Hardware Hints

Cable Shielding

The cable shielding has to be connected to the DSUB housing/pin 1. The housing and pin 1 are routed to the SM2-CAN/CPU metal distance screws.



Note:

Please refer to the SMART2 “Shielding” chapter for all hints concerning practical shield routing.

Connector and Line Termination

The DSUB connector assures connection to termination power. No termination network is available on the SM2-CAN itself.



Note:

The SM2-CAN has no internal line termination which, therefore, must be installed at the ends of the network (ISO 11898).



Note:

The maximum allowed auxiliary termination power is 90 mA/5 V DC. Assuming a power supply voltage of +5V emanating from the CAN connector (pin 6), the following resistor value is recommended:

R_{CAN} 150 ohm ± 2%, min. 0.25 W

3.2.2.9 ISaGRAF



Note:

For users of the ISaGRAF software it is not necessary to program the SMART2 interfaces, since they are completely controlled by ISaGRAF. Therefore, no such programming is envisaged for the SM2-CAN.



PEP Advantage

All necessary and/or useful information on ISaGRAF is available directly via software in the ISaGRAF on-line help program.

3.2.2.10 'C' Programming

Application developers using C under OS9 don't need to access hardware and/or I/O devices directly, since libraries will be provided giving easy access to all needed hardware functions.

The libraries are structured like the ones used for SMART I/O systems. A detailed description can be found in the corresponding file headers. The files are located in the following directories:

```
/dd/BSP/SMART/DEFS
/dd/BSP/SMART/<ioboard>/DEFS
```

Demo applications are located as follows:

```
/dd/APPLIC/SMART
```

Please note that the libraries are included inside the *PEP* BSP SMART I/O 2 only.

3.2.3 SM2-ETH

The SM2-ETH plug-in module is a full-duplex switched Ethernet controller board which provides the full functionality of a standard 10Mbit Ethernet 10BaseT interface.

It supports the IEEE 802.3 Ethernet Standards and the 802.3x Flow Control. Internally, it integrates an appropriate amount of buffer memory to gain reasonable Ethernet performance. Externally, it is connected via a 10BASE-T (twisted-pair) interface. The SM2-ETH provides LEDs for indicating the link status as well as the receive and transmit activities.

Basic Concept of the Module

In combination with the new SM2-CPU controller module the SM2-ETH represents a network controller and communication unit which can be arranged most economically in a single module slot.



PEP Advantage

SM2-ETH is just one of the possible fieldbus/ network modules that may be combined with the SM2-CPU controller module, depending on the specific functional requirements of your data exchange system.

Software

SM2-ETH is provided with an Internet Support Package (ISP) driver equivalent to layer 2, as well as the necessary descriptors and utilities to run the module. Additional configuration data is loaded by the driver from an onboard dedicated serial EEPROM during power up. The operation principle of the ISP driver is described in a separate driver manual.

The user operates the module via an “API” which is made available by the operating system, while the module does not require any jumper setting.

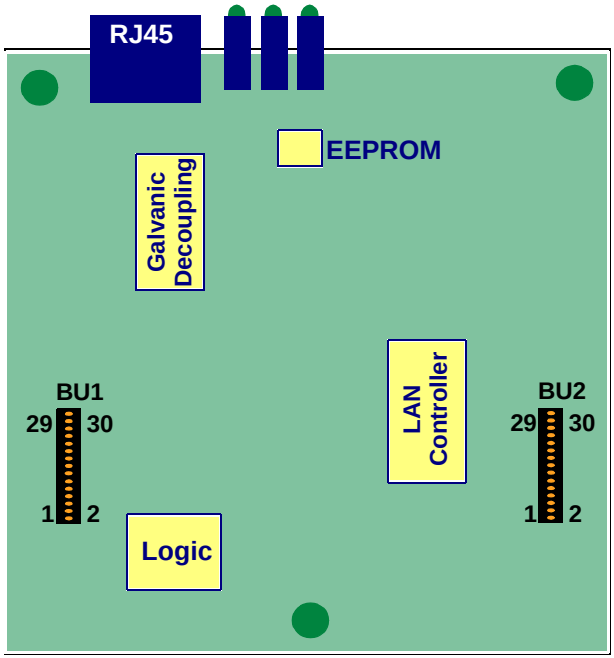
3.2.3.1 Specifications

Table 3-8: SM2-ETH Technical Specifications

SM2-ETH	Specifications
Controllers	SMSC LAN91C96 controller
Power Consumption	Typically 100 mA at 5 V DC
RAM	6 kByte
LEDs	• Link status(top, green) • Receive ON(center, green) • Transmit ON(bottom, green)
ID Byte	None
Interfaces	• RJ45 interface for 2-pair 5 UTP/STP cable; • FPC connector to controller module;
Network Performance	10 MBd (IEEE 802.3)
Temperature Range: Industrial Extended	 0°C to +70°C -40°C to +85°C
Operating Humidity	0% to 95%, non-condensing.
Weight	60 g

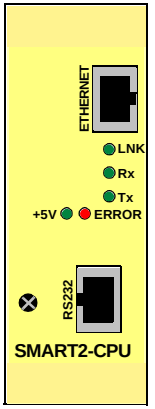
3.2.3.2 Board Overview

Figure 5-14: SM2-ETH Schematical Board Layout — Front View



3.2.3.3 Front Panel Layout

Figure 5-15: Combined SM2-CPU/ETH Front Panel Layout



LEDs:

LNK = Link Status
Rx = Receive ON
Tx = Transmit ON

ERROR = HALT
-5V = Power ON; Bootstrap loader ENABLED

3.2.3.4 Functional Description

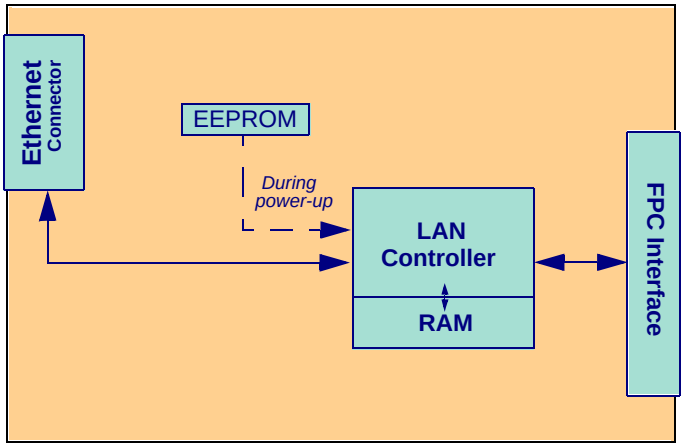
The SM2-ETH communication plug-in is based on the LAN91C96 full-duplex Ethernet controller which integrates 6kByte of on-chip RAM for package buffering. This memory is controlled by an on-chip memory management unit offering enhanced performance if compared to ring buffer and linked-list architecture. It integrates also 10Base-T transceiver functions on chip.

Controller Registers

The default base address and other setup data are retrieved by the LAN controller during reset from an on-board serial EEPROM. The Ethernet controller occupies 16 Byte of address space in all and is accessed via a 16-bit wide data path.

3.2.3.5 Block Diagram

Figure 5-16: SM2-ETH Block Diagram



3.2.3.6 Configuration

PEP Advantage

The SM2-ETH is easy to use thanks to its “API”-oriented operation. It does not require any hardware configuration. The only hardware operation required is to plug the Ethernet cable into the Ethernet connector.

3.2.3.7 Pinouts

The Ethernet connector is an un-shielded RJ45 socket complying with Ethernet specifications:

Table 3-9: SM2-ETH Ethernet Connector Pinouts

Pin 1	Transmit +	Pin 5	—
Pin 2	Transmit –	Pin 6	Receive –
Pin 3	Receive +	Pin 7	—
Pin 4	—	Pin 8	—

3.2.3.8 ISaGRAF



Note:

For users of the ISaGRAF software it is not necessary to program the SMART2 interfaces, since they are completely controlled by ISaGRAF. Therefore, no such programming is envisaged for the SM2-ETH.



PEP Advantage

All necessary and/or useful information on ISaGRAF is available directly via software in the ISaGRAF on-line help program.

3.2.3.9 OS-9 Programming

The SM2-ETH provides a standard Ethernet interface. Therefore, application developers can find all details on OS-9 application programming in the Microware ISP Reference Manual, v. 2.1.

3.3 SM2-DC24V_

Within a SMART2 system, the SM2-DC24V_ module supports two functions, that of DC/DC system power supply, and four 4 digital outputs.

Nominally, the DC/DC part is a 24V-to-5V converter. Its standard input range is 18-36 V DC. Depending on your SMART2 system's overall power consumption, different variants of the module with higher/lower current output are available, namely:

- SM2-DC24V1 2.0 A;
- SM2-DC24V2 1.0 A;
- SM2-DC24V3 0.6 A.



Note:

When using one of these DC/DC converters under E2 temperature conditions (up to +85°C), a certain temperature derating beginning at +70°C limits the total available current.

The presence of input voltage is indicated on the front panel by two LEDs, one for the 24 V supply line and the other for the internal 5 V system supply.

The specifications of the four digital outputs of SM2-DC24V_ are identical to those of the SM-DOUT1 module. Also identical is the front panel pin numbering. Each channel has a maximum output switching frequency of 1 kHz, with the output in the ON state when the system issues a logical '0'. The maximum continuous output supply current is limited to 500 mA (resistive load) and provision has been made to cater for inductive loads.

The user interface is realized by four LEDs, one per output channel, switching on when the output is ON.

Basic Concept of the Module

In order to save slace and to increase further the flexibility of the SMART2-BASE unit, the DC/DC power supply unit is no longer a fixed part of the

SMART2 design. Therefore, it has assumed the conventional format common to the *PEP Modular Computers'* SM modules..



PEP Advantage

Due to the SMART2 philosophy of maximum flexibility and modularity, any external third party DC/DC product meeting the logical 5 V specifications and system power requirements may be used as a power supply unit. In this case, the supply input has to come across the left SPI extension input cable of the controller module front connector interface.

However, the most appropriate way to provide for a complete power supply solution is to use the *PEP* SM2-DC24V_ module.

Software

SM2-DC24V_ does not require any jumper setting. Programming of the module is similar to that of the module SM-DOUT1 (see chapter 4.2 of this manual).

3.3.1 Installation

In order to keep transition resistance values on the power supply line via SPI bus low, all main consumers of a SMART2 system should be grouped in proximity to the SM2-DC24V_ module. Main consumers are, for instance, the CPU plus fieldbus module, or the SMART analog modules.

As a rule, one could say that two such boards consuming 0.5 W each and, thus, reaching approx. 10% of the system's maximum consumption, may be considered a "main consumer".

Usually, the combined CPU plus fieldbus module being a main consumer fit into slot “0” of a system and the power supply module would be placed in slot “1”. Therefore, it is recommended to install all other main consumers in the subsequent slots of the system.

3.3.2 Main Features

Key Functions

- SMART2 system supply;
- Digital output functions.

Specifications

Table 3-10: SM2-DC24V_ Technical Specifications

SM2-DC24V_		Specifications
Common	Power Consumption	• min. 25 mW • max. 80 mW (LED's on)
	SPI Access	Max. (=default) 1 MHz
	ID Byte	$21
	Weight	90 g
	Temperature: • Industrial • Extended	 0°C to +70°C -40°C to +85°C
	Operating Humidity	0% to 95%, non-condensing
	Weight	100 g

Table 3-10: SM2-DC24V_ Technical Specifications

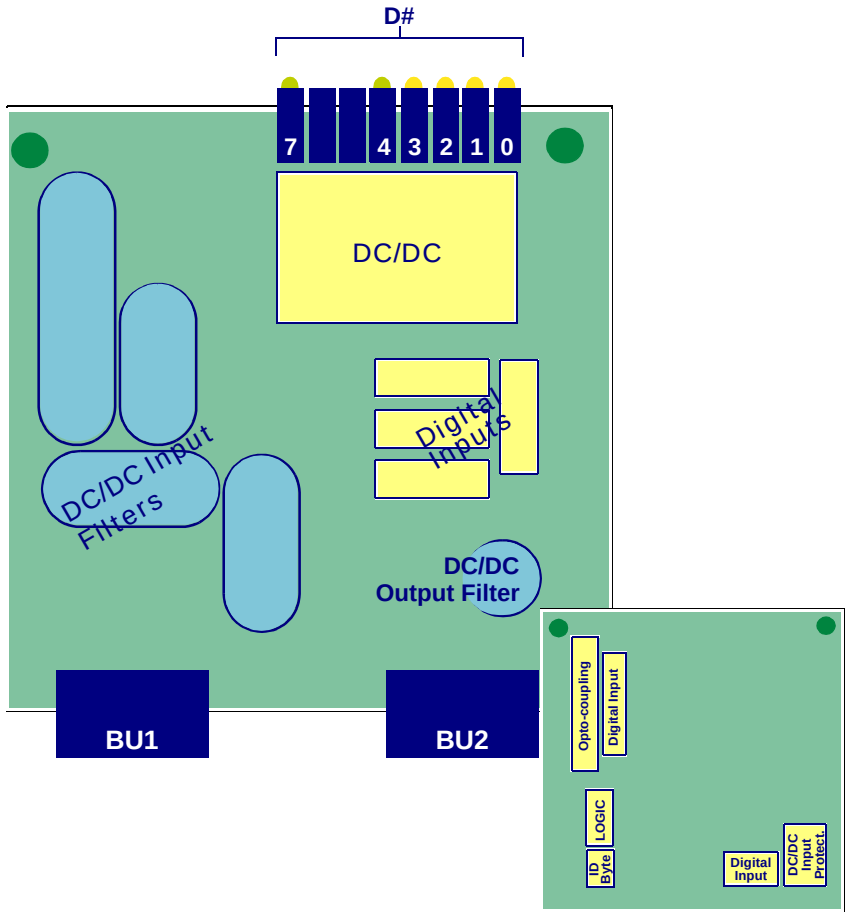
SM2-DC24V_		Specifications
DC/DC Converter	Input Voltage	18-36 V DC
	Reverse Voltage Protection	40 V DC
	Output Voltage	5V, adjustment ± 5%
	Output Current/ Power	• DC24V12.0 A/10 W) DC • DC24V21.0 A/ 5 W) DC • DC24V30.6 A/ 3 W) DC
	Power Derating	T > +70°C: 60% of total current
	Efficiency	73% at two thirds of max. load
	Isolation	1 kV
	Short Circuit Protection	Continuous
	LEDs (green): • Channel 4 • Channel 7	 5V DC supply signal 24V DC supply signal

Table 3-10: SM2-DC24V_ Technical Specifications

SM2-DC24V_		Specifications
Digital Output	Channels	4
	Transistor	Bipolar, TIP125
	Output Voltage	24V DC
	Output Current	max. 500 mA
	Type	High side switch
	Speed	< 1 kHz (square wave)
	Protection	Power ON/OFF
	Overvoltage	90V / 10ms
	LEDs (yellow):	Channels 0-3 active

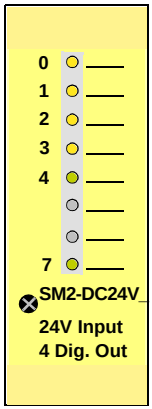
3.3.3 Board Overview

Figure 3-17: SM2-DC24V_ Schematical Board Layout — Front View and (Reduced) Reverse View



3.3.4 Front Panel Layout

Figure 3-18: SM2-DC24V_ Front Panel Layout



3.3.5 Functional Description

DC/DC Converter

The 18..36V-to5V DC converter is provided with additional filters to satisfy all EMI requirements.

Digital I/O Interface

The system interface is low active, which means that a logical ‘0’ activates the LED and opto-isolation stage causing the output to switch the 24 V VCC to power the load. The driving output device is a pair of Darlington transistors protected against inductive loads by an additional clamp diode. Furthermore, because each output stage requires three terminals to remain independent and only 14 such terminals are available, the common VCC and GND lines have been grouped. Hence, each pair of outputs is coupled to one VCC line and to each pair of VCC lines one groundis assigned.

Note:

Individual outputs should not be cascaded, as it cannot be guaranteed that power sharing will be proportional due to the transistor characteristics.

3.3.6 Block Diagram

Figure 3-19: SM2-DC24V_ Block Diagram — DC/DC Part

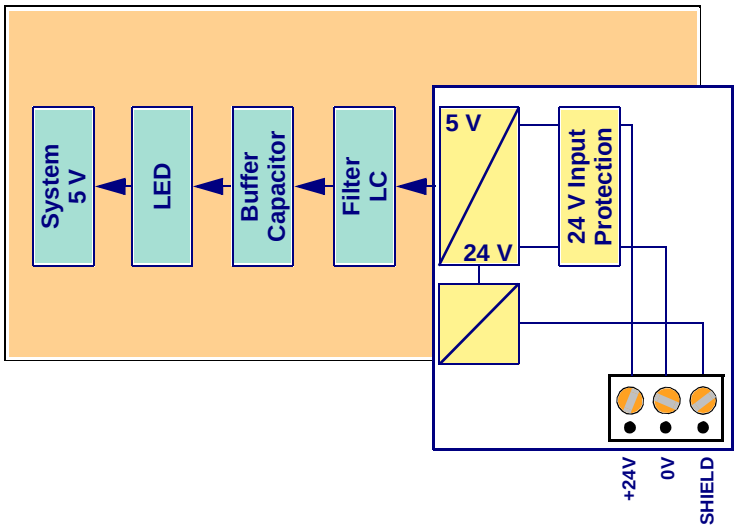
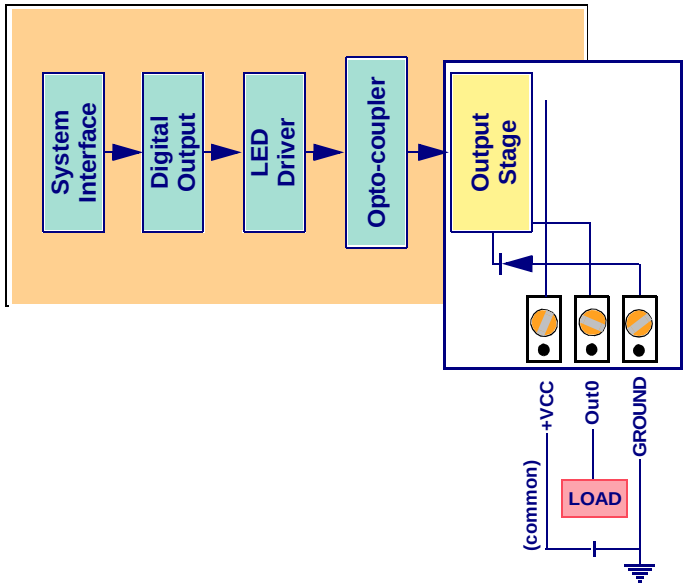


Figure 3-20: SM2-DC24V_ Block Diagram — Digital Output Part



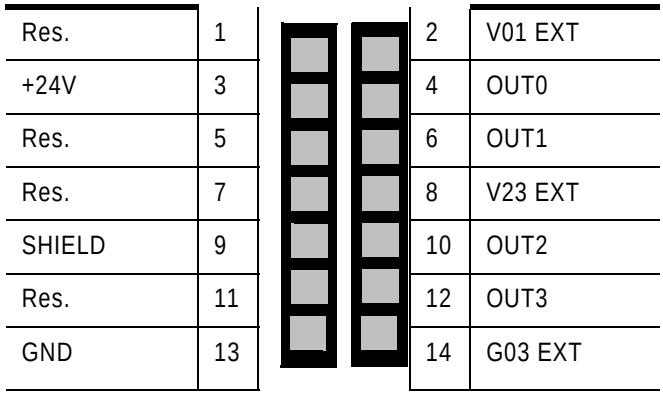
3.3.7 Configuration

On the SM2-DC24V_ no jumper configuration is foreseen.

3.3.8 Pinouts

The front connector, i.e. screw terminal block of a SMART2 module is made of a double row of seven pins that are separated as shown on the following page (for a complete overview of the a SMART2-BASE carrier unit see the figures 1-1 or 1-2).

Figure 3-21: Front Connector: Screw Terminals



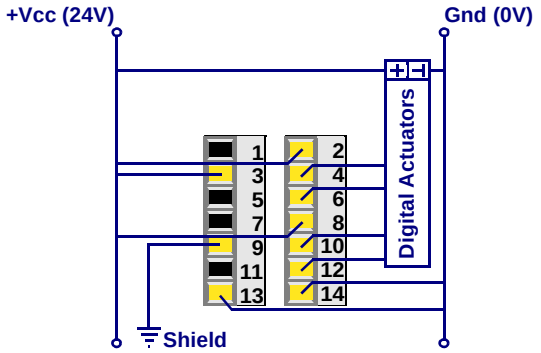
Res.	1		2	V01 EXT
+24V	3		4	OUT0
Res.	5		6	OUT1
Res.	7		8	V23 EXT
SHIELD	9		10	OUT2
Res.	11		12	OUT3
GND	13		14	G03 EXT

Table 3-11: SM2-DC24V_ Screw Terminal Front Connector

Pin	Signal	Description	Pin	Signal	Description
1	Res.	Reserved (+5V supply)	2	V01 EXT	Vcc for channels 0 & 1
3	+24V	External +24V supply	4	OUT0	Output 0
5	Res.	Reserved	6	OUT1	Outout 1
7	Res.	Reserved	8	V23 EXT	Vcc for channels 2 & 3
9	SHIELD	Shield connection	10	OUT2	Output 2
11	Res.	Reserved	12	OUT3	Outout 3
13	GND	External GND supply	14	G03 EXT	GND for channels 0..3

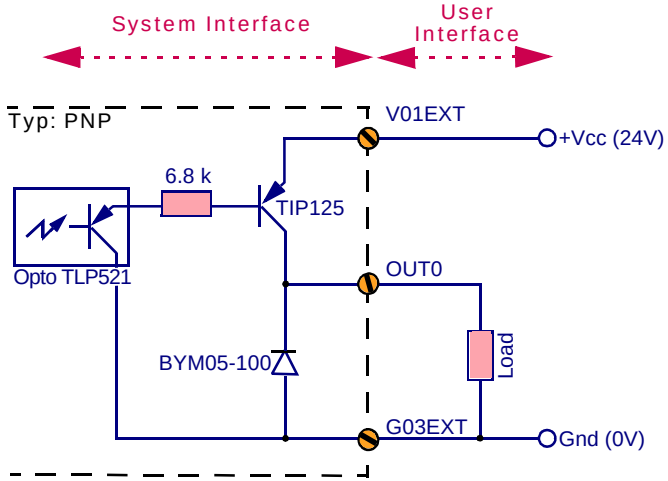
3.3.9 Connection

Figure 3-22: SM2-DC24V_ Connection Diagram



3.3.10 Output Circuit

Figure 3-23: SM2-DC24V_ Output Circuit



3.3.11 ISaGRAF

Note:

For users of the ISaGRAF software it is not necessary to program the SMART2 interfaces, since they are completely controlled by ISaGRAF. Therefore, no such programming is envisaged for the SM2-CPU.



PEP Advantage

All necessary and/or useful information on ISaGRAF is available directly via software in the ISaGRAF on-line help program.

3.3.12 'C' Programming

Application developers using C under OS9 don't need to access hardware and/or I/O devices directly, since libraries will be provided giving easy access to all needed hardware functions.

The libraries are structured like the ones used for SMART I/O systems. A detailed description can be found in the corresponding file headers. The files are located in the following directories:

```
/dd/BSP/SMART/DEFS
/dd/BSP/SMART/<ioboard>/DEFS
```

Demo applications are located as follows:

```
/dd/APPLIC/SMART
```

Please note that the libraries are included inside the *PEP* BSP SMART I/O 2 only.

Digital Modules

Table of Contents

Chapter 4

4.1 SM-DIN1 4-5
4.1.1 Introduction 4-5
4.1.2 Specifications 4-5
4.1.3 Front Panel Layout 4-6
4.1.4 Board Overview 4-7
4.1.5 Functional Description 4-8
4.1.6 Configuration 4-9
4.1.7 Pinout 4-10
4.1.8 'C' Programming 4-12
4.1.9 ISaGRAF Programming 4-16
4.2 SM-DOUT1 4-19
4.2.1 Introduction 4-19
4.2.2 Specifications 4-19
4.2.3 Front Panel Layout 4-20
4.2.4 Board Overview 4-20
4.2.5 Functional Description 4-21
4.2.6 Configuration 4-22

 © PEP Modular Computers

4.2.7 Pinouts 4-22
4.2.9 ISaGRAF Programming 4-29
4.3 SM-REL1 4-33
4.3.1 Introduction 4-33
4.3.2 Specifications 4-33
4.3.3 Front Panel Layout 4-34
4.3.4 Board Overview 4-34
4.3.5 Functional Description 4-35
4.3.6 Configuration 4-36
4.3.7 Pinouts 4-36
4.3.8 'C' Programming 4-38
4.3.9 ISaGRAF Programming 4-47
4.4 SM-ACI1 4-53
4.4.1 Introduction 4-53
4.4.2 Specifications 4-54
4.4.3 Front Panel Layout 4-55
4.4.4 Board Overview 4-56
4.4.5 Functional Description 4-58
4.4.6 Configuration 4-59
4.4.7 Screw Terminal Pinouts 4-61
4.4.8 'C' Programming 4-64
4.4.9 ISaGraf Programming 4-68

4.5 SM-ACO1 4-71
4.5.1 Introduction 7-71
4.5.2 Specifications 7-71
4.5.3 Front Panel Layout 7-72
4.5.4 Board Overview 7-73
4.5.5 Functional Description 7-74
4.5.6 Configuration 7-75
4.5.7 Pinouts 7-76
4.5.8 ISaGRAF 7-77
4.5.9 'C' Programming 7-78

4. Digital Modules

4.1 SM-DIN1

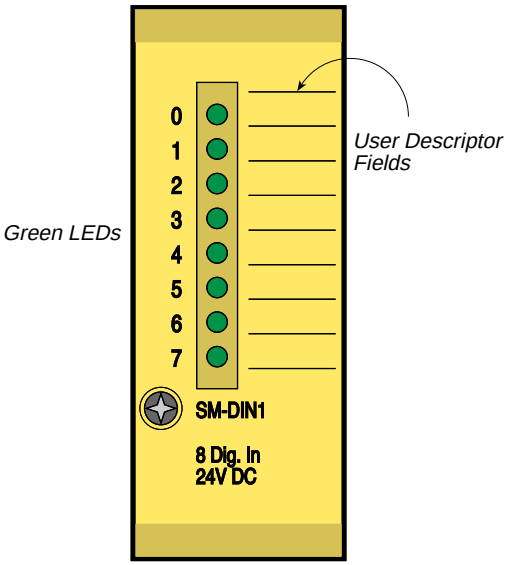
4.1.1 Introduction

The SM-DIN1 provides 8 optoisolated 24V DC digital inputs arranged in 6 independent groups with respect to the ground connections. The maximum input switching frequency is set to 200Hz with the system registering a logical '1' when the input exceeds 10V. Logical '0' is returned when the input falls below 8V. A low-pass filter restricts signals exceeding the filter limit and registers a logical '0' with the system. The user interface is realized by 8 green LEDs (one per input channel) which illuminate when the input exceeds 10V DC and a configurable filter on the last channel for INTx interrupts (where 'x' lies between 1 and 4 depending on which slot the module is located).

4.1.2 Specifications

Isolation	2.5 kV optoisolated from the system
Input	8 Digital Channels (24V) DC ± 10% 6 Independent Groups Common GND < 5mA Input Current Filter set to 200 Hz (5ms) Overvoltage Protection (rate) : 300V/90µs Overvoltage Protection (continuous) : 35V Switch ON Delay : 2ms (approx.) Switch OFF Delay : 2ms (approx.) High Level : > 10V Low Level : < 8V
Front Panel	Green LEDs (ON when Input > 10V)
Options	User configurable R-C filter on the last channel for INTx generation
Power Consumption	5mW (min.), 100mW (max.)
Temperature Range	Standard (0°C to +70°C), Extended (-40°C to +85°C)
Module Weight	40g
ID Byte	$01, read by the SPI Interface

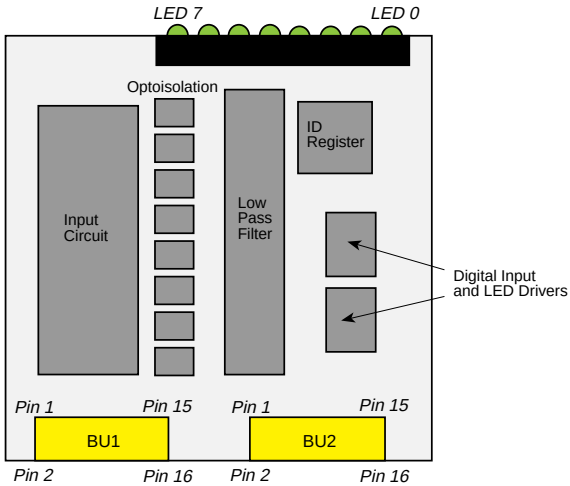
4.1.3 Front Panel Layout



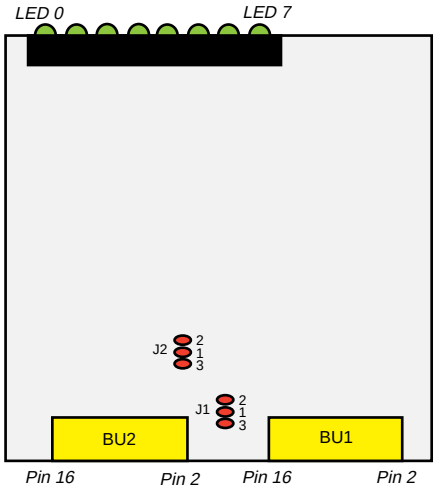
© PEP Modular Computers GmbH

4.1.4 Board Overview

Component Side

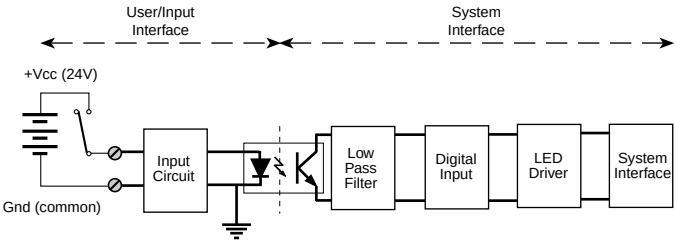


Solder Side



4.1.5 Functional Description

Figure 4.1.5.1: SM-DIN1 Schematic Diagram



The input circuit comprises a Zener diode requiring 3mA to drive it beyond the 'knee' that borders between 'OFF' and 'ON'. More simply, input voltages between 8V and 10V DC produce an unknown digital result.

After the optoisolation part of the circuit, an R-C first-order, low-pass filter prevents noise and signals greater than 200Hz from entering the system digital input. This filter is factory set on all channels but may be altered to a value according to customer specifications.

The LED driver activates the relevant LED when the input exceeds 10V DC.

As previously mentioned, the last channel is capable of issuing IRQx interrupts when enabled by software. This interrupt will only be acknowledged under OS-9 if the module is on the SMART-BASE or occupies the first slot of the first attached SMART-EXT unit. This channel is factory set at 200Hz but higher frequencies may be catered for. The table below shows the solder-jumper settings for this filter configuration.

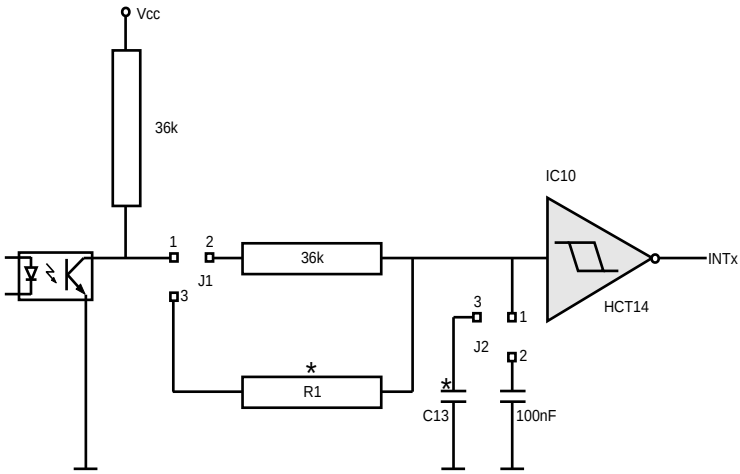
Jumper	Settings	Description
J1	*1-2*	*Standard input filter 200Hz*
	1-3	Customer specific input filter
J2	*1-2*	*Standard input filter 200Hz*
	1-3	Customer specific input filter

 © PEP Modular Computers GmbH

4.1.6 Configuration

Although two solder jumpers exist on the board (J1 and J2), they should remain at their factory settings unless a user-specific filter is required that cuts off at higher frequencies for example. If a user-specific filter is required then contact *PEP Modular Computers* for further advice and refer to the illustration shown in figure 4.1.6.1 of the filtering components.

Figure 4.1.6.1: SM-DIN1 Configurable Filter

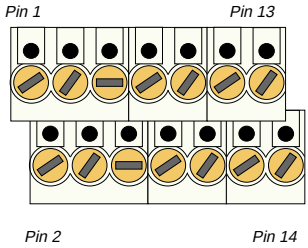


Surface Mounted Devices (SMD) are used in the production of the SM-DIN1 modules. The components to be changed (marked with an asterisk in figure 4.1.6.1), need not necessarily be SMD. Refer to the Board Overview (solder side) for the approximate position of these jumpers. It should be noted that when calculating component values for a specific filter, the capacitor/resistor relationship is almost linear. Therefore, it is suggested that only the capacitor should be changed. Hence, for a doubling of filter frequency input the value of the capacitor should be reduced by half.

4.1.7 Pinouts

Screw Terminal Pinouts

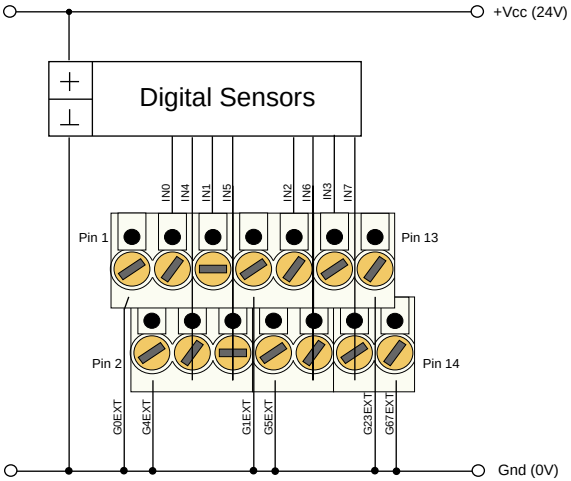
The following shows the pinout/signal relationship for the SM-DIN1 when connected to a particular screw terminal block.



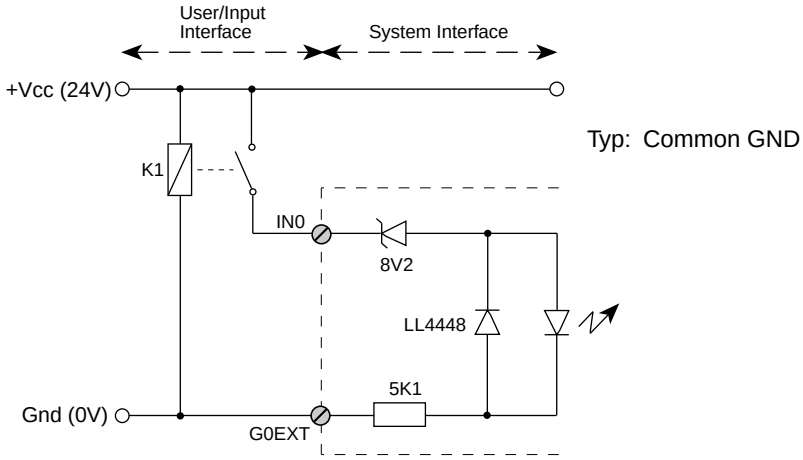
Pin Nr.	Signal	Description	Pin Nr.	Signal	Description
1	G0EXT	GND for channel 0	2	G4EXT	GND for channel 4
3	IN0	Input Channel 0	4	IN4	Input Channel 4
5	IN1	Input Channel 1	6	IN5	Input Channel 5
7	G1EXT	GND for channel 1	8	G5EXT	GND for channel 5
9	IN2	Input Channel 2	10	IN6	Input Channel 6
11	IN3	Input Channel 3	12	IN7	Input Channel 7
13	G23EXT	GND for channels 2&3	14	G67EXT	GND for channels 6&7

 © PEP Modular Computers GmbH

Connection



Input Circuit



4.1.8 'C' Programming

4.1.8.1 SM-DIN1 Library

The SM-DIN1 library of functions *smartio.l* provide a convenient way of accessing the SM-DIN1 module.

- All functions are written in ANSI C;
- Prototypes are to be found in the file *din1lib.h*.

Hardware Requirements

- SMART2 Base Unit or SMART I/O Base and Extension Units;
- SM-DIN1 Module.

Software Requirements

The compiler from one of the following:

- Ultra C Version 1.1.2 or higher;
- FasTrak 2.0.2 or higher.

The leftmost SM-Module has number 0 assigned to it a far as programming is concerned although physically this is slot 1!

Before a library function can be used, the function *SMDIN1Init* must first be called. This allocates the requested resources. Furthermore, this function needs to be called for each SM-DIN1 Module called within the task. Upon completion of the application, the function *SMDIN1DeInit* needs to be called for each SM-DIN1 Module that has been initialized.

In order to illustrate the use of the SM-DIN1 library, an application example called *demodin1.c* can be found in the SMART I/O application directory (normally found in */<device>/APPLIC/SMART/CMDS*).

 © PEP Modular Computers GmbH

4.1.8.2 SMDIN1Init

Syntax

```
error_code SMDIN1Init(u_int8 PortNr);
```

Description

This function initialises the SM-DIN1 Module on port *PortNr* by allocating space in memory for resources and resetting all inputs.

Input

`u_int8 PortNr`	SM-Port number to initialise (from 0 to 10)

Output

`error_code`	`SUCCESS`	
	`E_BMODE`	If the module is unknown

or standard OS-9 error code (refer to the OS-9 Technical Manual Error Codes Section).

Example

```
RetVal = SMDIN1Init(0);
```

4.1.8.3 SMDIN1DeInit

Syntax

```
error_code SMDIN1DeInit(u_int8 PortNr);
```

Description

This function de-initialises the SM-DIN1 Module on port *PortNr* by releasing all resources assigned to it thereby freeing memory for other uses.

Input

`u_int8 PortNr`	SM-Port number to de-initialise

Output

`error_code`	`SUCCESS` or standard OS-9 error code (refer to the OS-9 Technical Manual Error Codes Section).

Example

```
RetVal = SMDIN1DeInit(0);
```


© PEP Modular Computers GmbH

4.1.8.4 SMDIN1Get

Syntax

```
error_code SMDIN1Get(u_int8 PortNr, u_int8 *buffer);
```

Description

This function fetches the status of ALL 8 input lines of the port *PortNr* and writes it to the address pointed to by *buffer* with bit 0 of the buffer representing input 0 of the module.

Input

`u_int8 PortNr`	SM-Port number to fetch data
`u_int8 *buffer`	Pointer to Buffer where data is to be stored

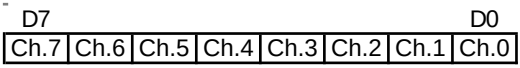
Output

`error_code`	`SUCCESS` or OS-9 error code (refer to the OS-9 Technical Manual Error Codes Section).

Example

```
RetVal = SMDIN1Get(0, buffer);
```

Buffer Explanation:



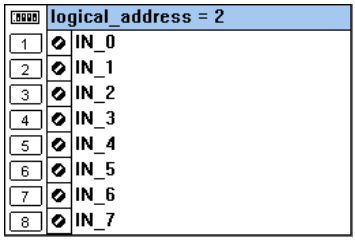
Bit Value : 0 = No input or not connected
1 = Input Active

4.1.9 ISaGRAF Programming

4.1.9.1 The ISaGRAF Board Parameters

Information on board parameters may be found in the PEP online help and ISaGRAF online help or user's manual.

Figure 4.1.9.1.1 a Typical Screen Section for the SM-DIN1 Module



Logical Address

The 8 inputs of this module may be clearly seen together with the logical address at which this particular board is residing. Remember, this logical address is the same as the physical slot position! In the example shown here, the board being addressed is in slot 2 i.e. the second of the SMART-BASE slots. Up to 11 slots are catered for in the SMART I/O concept; 3 on the SMART-BASE and 2 for each connected SMART-EXT (up to a total of 4).

4.1.9.2 The ISaGRAF Operate Calls

Operate calls are built into a program using ST or FBD languages when defining the project. A typical use could be at the initialization stage to check that the SMART-Modules are in fact located where they have been programmed to be. The syntax of the operate call is as follows:

Syntax

```
<return variable> := OPERATE(<source var>, COMMAND,
                             <source data>);
```

Here the `return variable` is assigned a value associated with the selected `COMMAND` parameter. Each SMART-Module possesses its own set of these `COMMANDS`.

Example

```
<error.code>:= OPERATE(<channel>, O_INIT_CODE, 0);
```

`O_INIT_CODE` is one of a number of distinct commands recognized by the PEP Modular Computers' implementation of board drivers and checks for example that the board is located where the program expects it to be.

`channel` provides channel specific information and in the example shown here, any of the 8 input channels may be used. The last parameter is not usually used by PEP implementations and is set to 0 (zero).

The `error.code` returns a value of zero if no error was detected, otherwise it returns a non-zero value depending on the error encountered. A list of these error codes may be found in the PEP online help.

 © PEP Modular Computers GmbH

A complete list of the operate COMMANDS may be obtained by selecting a project from the ISaGRAF ***projects*** group, opening an application and observing the ***Common defines*** in the ***Dictionary*** pull-down menu. Note that not all calls in the list may be used within the SMART2 environment however, the calls applicable to this module are :

O_INIT_CODE : The syntax and usage have already been explained.

 © PEP Modular Computers GmbH

4.2 SM-DOUT1

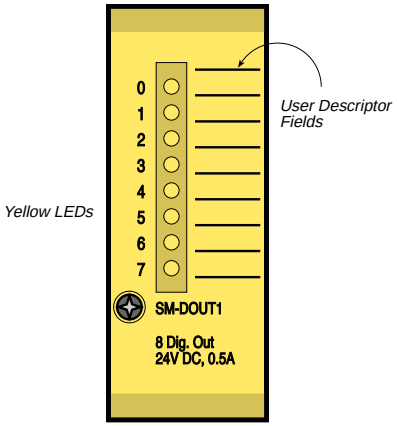
4.2.1 Introduction

The SM-DOUT1 provides eight optoisolated 24V DC (typ.) digital outputs arranged in 2 independent groups. The maximum output switching frequency is limited to 1 kHz (square wave) with the output in the ON state when the system issues a logical '0'. The maximum continuous output supply current is limited to 500mA (resistive loads) and provision has been made to cater for inductive loads. The user interface is realized by 8 yellow LEDs (1 per output channel) which illuminate when the output is ON.

4.2.2 Specifications

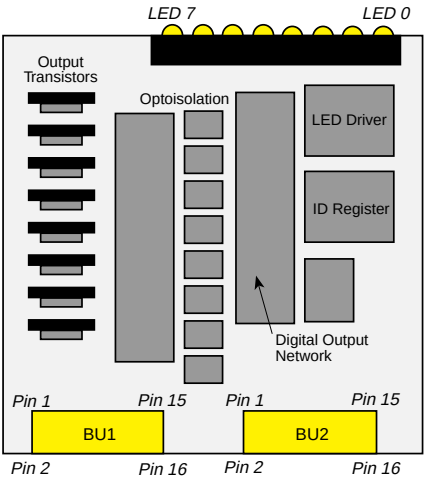
Isolation	2.5 kV optoisolated from the system
Output	8 Digital Channels 2 Independent Groups High-Side Switch (Common Vcc) 500mA Continuous Current (per channel) Inductive Load Protection (clamp diode) Power ON/OFF Protection Overvoltage Protection : 90V/10µs (35V) Output Voltage : 24V DC Switch ON Delay (resistive) : 3µs (at 24V) Switch OFF Delay (resistive) : 130µs (at 24V) Max. Frequency : 1 kHz square wave Output ON : Intern. Logical '0' Output OFF : Intern. Logical '1' Outputs after Reset : OFF
Front Panel	Yellow LEDs ON when the output is ON
Power Consumption	5mW (min.), 270mW (max.)
Temperature Range	Standard (0°C to +70°C), Extended (-40°C to +85°C)
Module Weight	70g
ID Byte	$10, read by the SPI Interface

4.2.3 Front Panel Layout



4.2.4 Board Overview

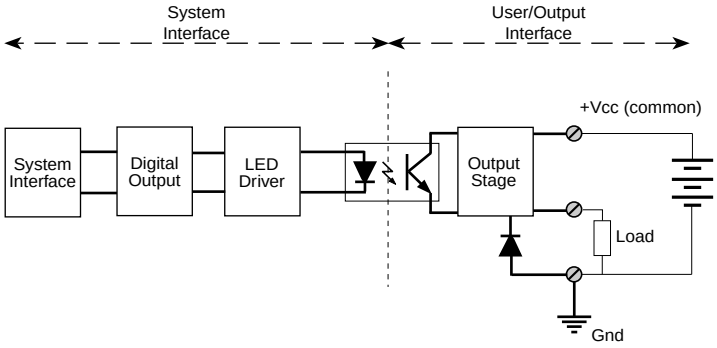
Component Side



© PEP Modular Computers GmbH

4.2.5 Functional Description

Figure 4.2.5.1: SM-DOUT1 Schematic Diagram



The system interface is low active which means that a logical '0' activates the LED and optoisolation stage causing the output to switch the 24V Vcc to power the load.

The driving output device is a Darlington transistor pair protected against inductive loads by an additional clamp diode. Furthermore, because each output stage requires three terminals to remain independent and only 14 such terminals are available, the common Vcc and GND lines have been grouped. Hence, each pair of outputs is coupled to one Vcc line and to each pair of Vcc lines is one ground.

Warning !

Individual outputs should not be cascaded as it cannot be guaranteed that power sharing will be proportional due to the transistor characteristics.

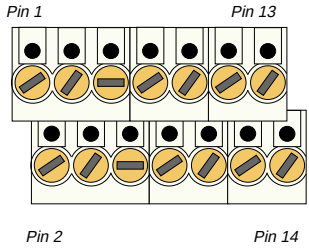
4.2.6 Configuration

There are no jumpers to configure on the SM-DOUT1.

4.2.7 Pinouts

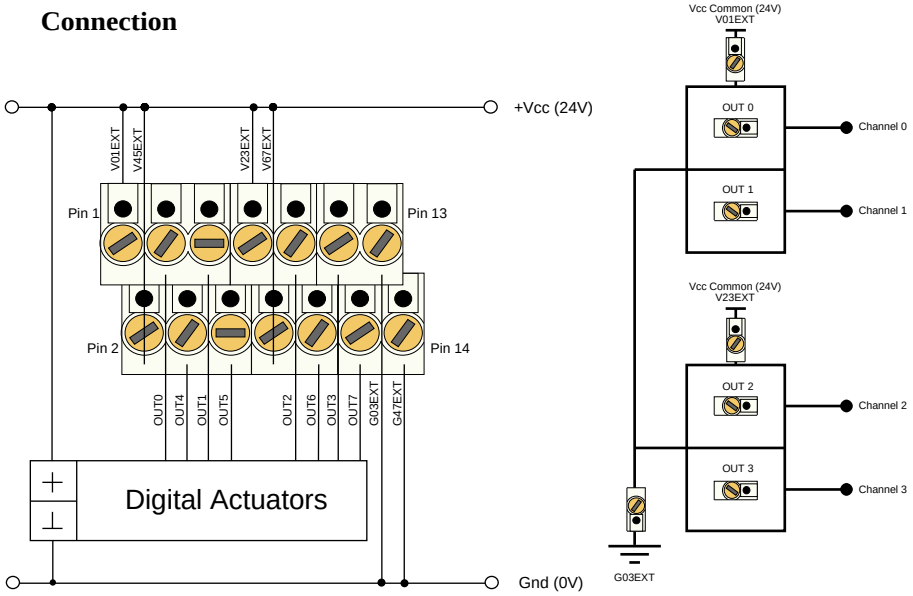
Screw Terminal Pinouts

The following shows the pinout/signal relationship for the SM-DOUT1 when connected to a particular screw terminal block.

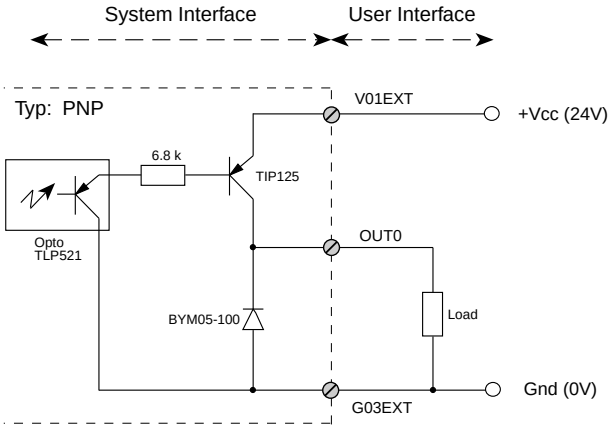


Pin Nr.	Signal	Description	Pin Nr.	Signal	Description
1	V01EXT	Vcc for channels 0 & 1	2	V45EXT	Vcc for channels 4 & 5
3	OUT0	Output 0	4	OUT4	Output 4
5	OUT1	Output 1	6	OUT5	Output5
7	V23EXT	Vcc for channels 2 & 3	8	V67EXT	Vcc for channels 6 & 7
9	OUT2	Output 2	10	OUT6	Output 6
11	OUT3	Output 3	12	OUT7	Output 7
13	G03EXT	GND for channels 0..3	14	G47EXT	GND for channels 4..7

Connection



Output Circuit



4.2.8 'C' Programming

4.2.8.1 SM-DOUT1 Library

The SM-DOUT1 library of functions *smartio.l* provide a convenient way of accessing the SM-DOUT1 module.

- All functions are written in ANSI C;
- Prototypes are to be found in the file *dout1lib.h*.

Hardware Requirements

- SMART2 Base Unit or SMART I/O Base and Extension Units;
- SM-DOUT1 Module.

Software Requirements

The compiler from one of the following:

- Ultra C Version 1.1.2 or higher;
- FasTrak 2.0.2 or higher.

The leftmost SM-Module has number 0 assigned to it a far as programming is concerned although physically this is slot 1!

Before a library function can be used, the function *SMDOUT1Init* must first be called. This allocates the requested resources. Furthermore, this function needs to be called for each SM-DOUT1 Module called within the task. Upon completion of the application, the function *SMDOUT1DeInit* needs to be called for each SM-DOUT1 Module that has been initialised.

In order to illustrate the use of the SM-DOUT1 library, an application example called *demdout1.c* can be found in the SMART I/O application directory (normally found in */<device>/APPLIC/SMART/CMDS*).

 © PEP Modular Computers GmbH

Note

If the same SM-DOUT1 Module is to be accessed from different tasks, the user must make sure that the actual output status is ensured by using for example, semaphores within OS-9.

4.2.8.2 SMDOUT1Init

Syntax

```
error_code SMDOUT1Init(u_int8 PortNr);
```

Description

This function initializes the SM-DOUT1 Module on port *PortNr* and sets all output lines to OFF.

Input

`u_int8 PortNr`	SM-Port number to initialize (from 0 to 10)

Output

`error_code`	`SUCCESS`	
	`E_BMODE`	If the module is unknown

or standard OS-9 error code (refer to the OS-9 Technical Manual Error Codes Section).

Example

```
RetVal = SMDOUT1Init(0);
```

4.2.8.3 SMDOUT1DeInit

Syntax

```
error_code SMDOUT1DeInit(u_int8 PortNr);
```

Description

This function de-initializes the SM-DOUT1 Module on port *PortNr* by releasing all resources assigned to it thereby freeing memory for other uses.

Input

`u_int8 PortNr`	SM-Port number to de-initialize

Output

`error_code`	`SUCCESS` or standard OS-9 error code (refer to the OS-9 Technical Manual Error Codes Section).

Example

```
RetVal = SMDOUT1DeInit(0);
```


© PEP Modular Computers GmbH

4.2.8.4 SMDOUT1Get

Syntax

```
error_code SMDOUT1Get(u_int8 PortNr, u_int8 *buffer);
```

Description

This function fetches the output status of the port *PortNr* and writes it to the memory location pointed to by *buffer*. This buffer shows the status of all 8 outputs with bit 0 representing output 0 of the module.

Input

`u_int8 PortNr`	SM-Port number to fetch data
`u_int8 *buffer`	Pointer to a buffer where data is to be stored.

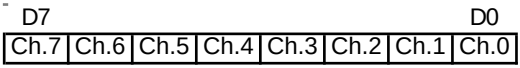
Output

`error_code`	`SUCCESS` or standard OS-9 error code (refer to the OS-9 Technical Manual Error Codes Section).

Example

```
RetVal = SMDOUT1Get(0, buffer);
```

Buffer Explanation:



Bit Value : 0 = Output is ON
1 = Output is OFF

4.2.8.5 SMDOUT1Set

Syntax

```
error_code SMDOUT1Set(u_int8 PortNr, u_int8 *buffer);
```

Description

This function sets the output of the port *PortNr* with the contents of the address pointed to by *buffer*. When writing to this buffer, observe that all 8 outputs are set at the same time.

Input

`u_int8 PortNr`	SM-Port number to fetch data
`u_int8 *buffer`	Pointer to buffer where the data is to be written

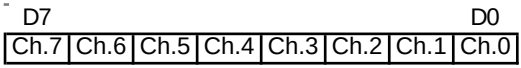
Output

`error_code`	`SUCCESS` or standard OS-9 error code (refer to the OS-9 Technical Manual Error Codes Section).

Example

```
RetVal = SMDOUT1Set(0, buffer);
```

Buffer Explanation:



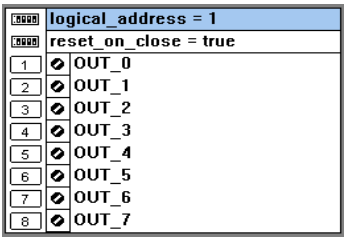
Bit Value : 0 = Output is ON
1 = Output is OFF

4.2.9 ISaGRAF Programming

4.2.9.1 The ISaGRAF Board Parameters

Information on board parameters may be found in the PEP online help and ISaGRAF online help or user's manual.

Figure 4.2.9.1.1 a Typical Screen Section for the SM-DOUT1 Module



Logical Address

The 8 outputs of this module may be clearly seen together with the logical address at which this particular board is residing. Remember, this logical address is the same as the physical slot position! In the example shown here, the board being addressed is in slot 1 i.e. the first of the SMART-BASE slots. Up to 11 slots are catered for in the SMART I/O concept; 3 on the SMART-BASE and 2 for each connected SMART-EXT (up to a total of 4).

Module Specific Information

`reset_on_close` : This variable allows ISaGRAF to return the status of all the output channels to zero (0V) when the application is closed.

4.2.9.2 The ISaGRAF Operate Calls

Operate calls are built into a program using ST or FBD languages when defining the project. A typical use could be at the initialization stage to check that the SMART-Modules are in fact located where they have been programmed to be. The syntax of the operate call is as follows:

Syntax

```
<return variable> := OPERATE(<source var>, COMMAND,
                             <source data>);
```

Here the `return variable` is assigned a value associated with the selected `COMMAND` parameter. Each SMART-Module possesses its own set of these `COMMANDS`.

Example

```
<error.code>:= OPERATE(<channel>, O_INIT_CODE, 0);
```

`O_INIT_CODE` is one of a number of distinct commands recognized by the PEP Modular Computers' implementation of board drivers and checks for example that the board is located where the program expects it to be.

`channel` provides channel specific information and in the example shown here, any of the 8 input channels may be used. The last parameter is not usually used by PEP implementations and is set to 0 (zero).

The `error.code` returns a value of zero if no error was detected, otherwise it returns a non-zero value depending on the error encountered. A list of these error codes may be found in the PEP online help.

 © PEP Modular Computers GmbH

A complete list of the operate COMMANDS may be obtained by selecting a project from the ISaGRAF ***projects*** group, opening an application and observing the ***Common defines*** in the ***Dictionary*** pull-down menu. Note that not all calls in the list may be used within a SMART2 system.
however, the calls applicable to this module are :

O_INIT_CODE : The syntax and usage have already been explained.

This page has been left blank intentionally.

© PEP Modular Computers GmbH

4.3 SM-REL1

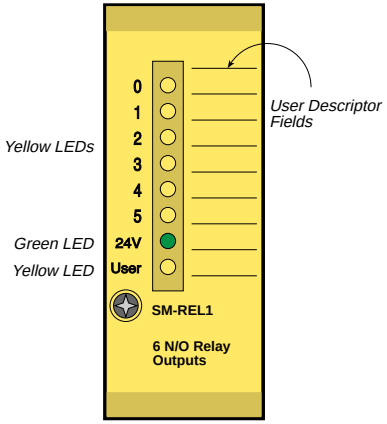
4.3.1 Introduction

The SM-REL1 provides power switching up to 250V AC or 220V DC with surge protection and the ability to switch inductive loads being built-in features of this versatile module. A freely programmable LED on the front panel provides the user interface together with 6 relay status LEDs and an external 24V supply signal LED.

4.3.2 Specifications

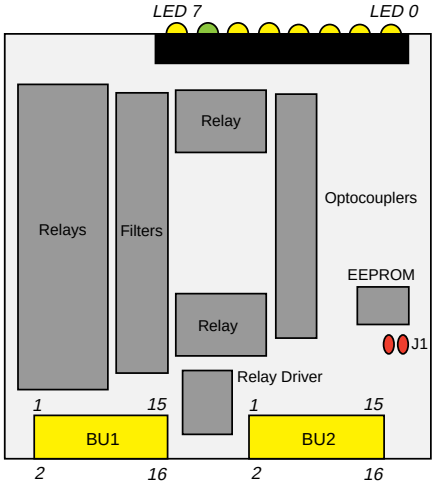
Isolation	System	: 2.5 kV Optoisolated
	Between Channels	: 1.5 kV
	Between Contacts	: 1.0 kV
	Surge Voltage	: 1.5 kV
Output	6 Normally Open Relay Contacts	
	Max. Switching Voltage	: 250V AC / 220V DC
	Max. Switching Current	: 2A
	Max. Switching Load (Resistive)	: 60W / 125 VA
	Switch ON Current Rate	: 8A/3ms
	Sustained Current	: 3A
	Switch OFF Current Rate	: 2A/2ms
	Max. Switching Frequency	: 100 Hz
	Contact Resistance	: 50mΩ
	Settling Time	: 4ms
	Power ON/OFF Protection	: Glitch-free Power ON
	Expected Life	: 2 Million Cycles
Input	Ext. VCC	: 24V DC ± 20% @ 100mA
Front Panel	6 Yellow LEDs	: Relay CLOSED
	1 Green LED	: Ext. VCC Ready
	1 Yellow LED	: User
Power Cons.	23mW (min.), 160mW (max.)	
Temperature Range	Standard (0°C to +70°C)	
Module Weight	61g	
ID Byte	$20, read by the SPI Interface	

4.3.3 Front Panel Layout



4.3.4 Board Overview

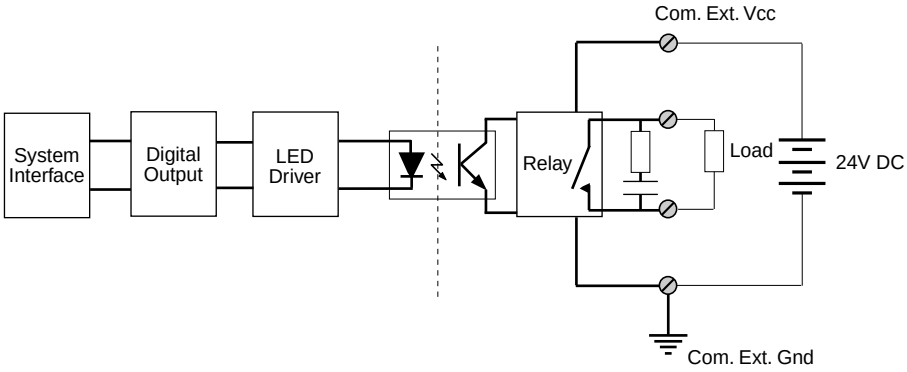
Component Side



 © PEP Modular Computers GmbH

4.3.5 Functional Description

Figure 4.3.5.1: SM-REL1 Schematic Diagram



The individual relays are low active which means that a digital '0' delivered by the system closes the relay and illuminates the LED on the front-panel corresponding to the required channel. Conversely, a logical '1' appearing on the external VCC line suggests that this supply is connected and active.

The operational life of the relays is recorded in terms of operating cycles and refers to mechanical failure. The contacts however could be damaged through excessive load switching causing sparks to break down the contact surface. This in turn could increase the resistance and lead to an MTBF that is shorter than that quoted in the specifications.

256 bytes of serial EEPROM contain the module ID number and production data although space has been allocated for future use.

4.3.6 Configuration

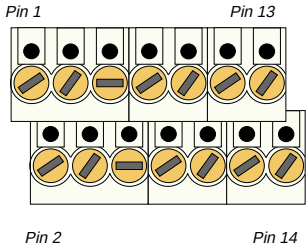
Jumper J1 - EEPROM Protection

Jumper	Settings	Description
J1	*set*	*EEPROM is not hardware write protected*
	open	EEPROM is hardware write protected

4.3.7 Pinouts

Screw Terminal Pinouts

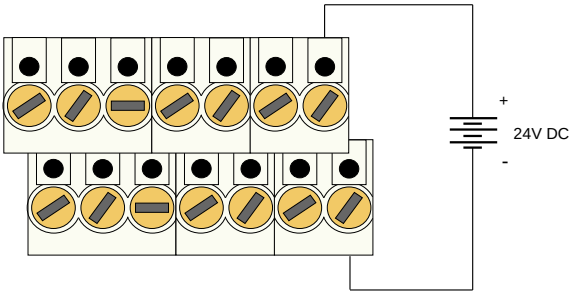
The following shows the pinout/signal relationship for the SM-REL1 when connected to a particular screw terminal block.



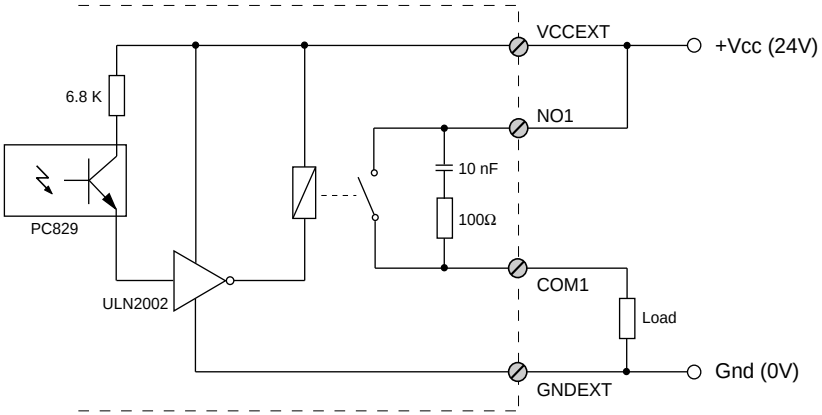
Pin Nr.	Signal	Description	Pin Nr.	Signal	Description
1	NO0	N/O Relay 0 Conn.	2	COM0	Relay 0 Common Conn.
3	NO1	N/O Relay 1 Conn.	4	COM1	Relay 1 Common Conn.
5	NO2	N/O Relay 2 Conn.	6	COM2	Relay 2 Common Conn.
7	NO3	N/O Relay 3 Conn.	8	COM3	Relay 3 Common Conn.
9	NO4	N/O Relay 4 Conn.	10	COM4	Relay 4 Common Conn.
11	NO5	N/O Relay 5 Conn.	12	COM5	Relay 5 Common Conn.
13	VCCEXT	External 24V DC VCC	14	GNDEXT	External GND

 © PEP Modular Computers GmbH

Connection



Input Circuit



4.3.8 'C' Programming

4.3.8.1 SM-REL1 Library

The SM-REL1 library of functions *smartio.l* provide a convenient way of accessing the SM-REL1 module.

- All functions are written in ANSI C;
- Prototypes are to be found in the file *rel1lib.h*.

Hardware Requirements

- SMART2 Base Unit or SMART I/O Base and Extension Units;
- SM-REL1 Module.

Software Requirements

The compiler from one of the following:

- Ultra C Version 1.1.2 or higher;
- FasTrak 2.0.2 or higher.

The leftmost SM-Module has number 0 assigned to it a far as programming is concerned although physically this is slot 1!

Before a library function can be used, the function *SMREL1Init* must first be called. This allocates the requested resources. Furthermore, this function needs to be called for each SM-REL1 Module called within the task. Upon completion of the application, the function *SMREL1DeInit* needs to be called for each SM-REL1 Module that has been initialized.

In order to illustrate the use of the SM-REL1 library, an application example called *demrel1.c* can be found in the SMART2 application directory (normally found in */<device>/APPLIC/SMART/CMDS*).

 © PEP Modular Computers GmbH

Note

If the same SM-REL1 Module is to be accessed from different tasks, the user must make sure that the actual output status is ensured by using for example, semaphores within OS-9.

4.3.8.2 SMREL1Init

Syntax

```
error_code SMREL1Init(u_int8 PortNr);
```

Description

This function initializes the SM-REL1 Module on port *PortNr* by setting the relays to OPEN, relay LED status to OFF and the external Vcc as input.

Input

`u_int8 PortNr`	SM-Port to initialize (from 0 to 10)

Output

`error_code`	`SUCCESS`	
	`E_BMODE`	if the module is unknown or standard OS-9 error code (refer to the OS-9 Technical Manual Error Codes Section).

Example

```
RetVal = SMREL1Init(0);
```

4.3.8.3 SMREL1DeInit

Syntax

```
error_code SMREL1DeInit(u_int8 PortNr);
```

Description

This function deinitializes the SM-REL1 Module on the port *PortNr* by freeing the resources tied up with this module and setting all relays to OPEN, all LEDs to OFF and setting the ext. Vcc line to input.

Input

`u_int8 PortNr`	SM-Port number to de-initialize (from 0 to 10)

Output

`error_code`	`SUCCESS`	
	`E_BMODE`	if the module is unknown

or standard OS-9 error code (refer to the OS-9 Technical Manual Error Codes Section).

Example

```
RetVal = SMREL1DeInit(0);
```

4.3.8.4 SMREL1Reset

Syntax

```
error_code SMREL1Reset(u_int8 PortNr);
```

Description

This function resets the SM-REL1 Module by setting all relays to OPEN and all LEDs to OFF. This function does not free the resources as with the SMRELl1DeInit function call but it does ensure that the relay status (and LEDs) is as expected.

Input

`u_int8 PortNr`	Number of the SM port to reset

Output

`error_code`	`SUCCESS`	
	`E_BMODE`	if the module is unknown
	or standard OS-9 error code (refer to the OS-9 Technical Manual Error Codes Section).	

Example

```
RetVal = SMREL1Reset(0);
```

4.3.8.5 SMREL1GetRly

Syntax

```
error_code SMREL1GetRly(u_int8 PortNr,
                        u_int8 *buffer);
```

Description

This function gets the input of the port *PortNr* and writes it to the *buffer*. Bits 0-5 of the buffer reflect the Relay setting:

0 = relay closed
1 = relay open

Bits 6 and 7 are cleared.

Input

`u_int8 PortNr`	SM-Port number to get data
`u_int8 *buffer`	Pointer to Buffer where data is written

Output

`error_code`	`SUCCESS`
	`E_BMODE` if the module is unknown
	or standard OS-9 error code (refer to the OS-9 Technical Manual Error Codes Section).

Example

```
RetVal = SMREL1GetRly(0, buffer);
```

 © PEP Modular Computers GmbH

4.3.8.6 SMREL1SetRly

Syntax

```
error_code SMREL1SetRly(u_int8 PortNr,
                        u_int8 *buffer);
```

Description

This function sets the output of the port *PortNr* with the *buffer* contents. Bits 0-5 of the buffer are used to set the Relays:

0 = close relay
1 = open relay

Bits 6 and 7 have no function.

Input

`u_int8 PortNr`	SM-Port number to set data
`u_int8 *buffer`	Pointer to buffer where data is stored

Output

`error_code`	`SUCCESS`	
	`E_BMODE`	if the module is unknown
	or standard OS-9 error code (refer to the OS-9 Technical Manual Error Codes Section).	

Example

```
RetVal = SMREL1SetRly(0, buffer);
```

4.3.8.7 SMREL1GetLed

Syntax

```
error_code SMREL1GetLed(u_int8 PortNr,
                        u_int8 *buffer);
```

Description

This function gets the status of the User LED on the port *PortNr* and writes it to the *buffer*. If the LED is OFF then it returns 0, otherwise it returns 1. This function of the LED is for the user to decide.

Input

`u_int8 PortNr`	SM-Port number to get data
`u_int8 *buffer`	Pointer to buffer where data is written

Output

`error_code`	`SUCCESS`	
	`E_BMODE`	if the module is unknown
	or standard OS-9 error code (refer to the OS-9 Technical Manual Error Codes Section).	

Example

```
RetVal = SMREL1GetLed(0, buffer);
```


© PEP Modular Computers GmbH

4.3.8.8 SMREL1SetLed

Syntax

```
error_code SMREL1SetLed(u_int8 PortNr,
                        u_int8 *buffer);
```

Description

This function sets the status of the User LED on the port *PortNr* with the contents of the *buffer*. If a bit is set to 0 then the LED is OFF otherwise it is ON.

Input

`u_int8 PortNr`	SM-Port number to write data
`u_int8 *buffer`	Pointer to buffer where data is stored

Output

`error_code`	`SUCCESS`
	`E_BMODE` if the module is unknown
	or standard OS-9 error code (refer to the OS-9 Technical Manual Error Codes Section).

Example

```
RetVal = SMREL1SetLed(0, buffer);
```

4.3.8.9 SMREL1GetExtVcc

Syntax

```
error_code SMREL1GetExtVcc(u_int8 PortNr,
                           u_int8 *buffer);
```

Description

This function gets the input of the port *PortNr* and writes it to *buffer*. If there is no external Vcc, then it returns 0, otherwise it returns 1.

Input

`u_int8 PortNr`	SM-Port number to get data
`u_int8 *buffer`	pointer to buffer where data is written

Output

`error_code`	`SUCCESS`	
	`E_BMODE`	if the module is unknown
	or standard OS-9 error code (refer to the OS-9 Technical Manual Error Codes Section).	

Example

```
RetVal = SMREL1GetExtVcc(0, buffer);
```

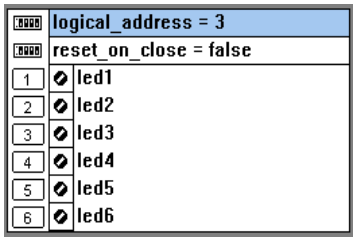

© PEP Modular Computers GmbH

4.3.9 ISaGRAF Programming

4.3.9.1 The ISaGRAF Board Parameters

Information on board parameters may be found in the PEP online help and ISaGRAF online help or user's manual.

Figure 4.3.9.1.1 a Typical Screen Section for the SM-REL1 Module



Logical Address

The 6 outputs of this module may be clearly seen together with the logical address at which this particular board is residing. Remember, this logical address is the same as the physical slot position! In the example shown here, the board being addressed is in slot 3 i.e. the third of the SMART-BASE slots. Up to 11 slots are catered for in the SMART I/O concept; 3 on the SMART-BASE and 2 for each connected SMART-EXT (up to a total of 4).

Module Specific Information

`reset_on_close` : This variable allows ISaGRAF to set all the output relays to open when the application closes.

4.3.9.2 The ISaGRAF Operate Calls

Operate calls are built into a program using ST or FBD languages when defining the project. A typical use could be at the initialization stage to check that the SMART-Modules are in fact located where they have been programmed to be. The syntax of the operate call is as follows:

Syntax:

```
<return variable> := OPERATE(<source var>, COMMAND,
                             <source data>);
```

Here the `return variable` is assigned a value associated with the selected `COMMAND` parameter. Each SMART-Module possesses its own set of these `COMMANDS`.

Example:

```
<error.code>:= OPERATE(<channel>, O_INIT_CODE, 0);
```

`O_INIT_CODE` is one of a number of distinct commands recognized by the PEP Modular Computers' implementation of board drivers and checks for example that the board is located where the program expects it to be.

`channel` provides channel specific information and in the example shown here, any of the 8 input channels may be used. The last parameter is not usually used by PEP implementations and is set to 0 (zero).

The `error.code` returns a value of zero if no error was detected, otherwise it returns a non-zero value depending on the error encountered. A list of these error codes may be found in the PEP online help.

 © PEP Modular Computers GmbH

A complete list of the operate COMMANDS may be obtained by selecting a project from the ISaGRAF *projects* group, opening an application and observing the *Common defines* in the *Dictionary* pull-down menu. Note that not all calls in the list may be used within the SMART I/O environment. However, the calls applicable to this module are:

O_INIT_CODE

Syntax and usage as explained above.

O_SET_LED

This operate call allows the user to control the yellow LED on the board's front panel. The syntax is as follows:

```
<analog var> := OPERATE(<iovar>, O_SET_LED,
<var>);
```

where

`<analog var>`	represents the analog return variable and can be zero, if the operate call is supported and the correct `<var>` is passed, or non-zero, if an error occurred.
`<iovar>`	represents any module variable, and
`<var>`	represents the LED to be controlled and must be one of `OA_LED1_ON` or `OA_LED1_OFF`.

O_EXT_POWER

This operate call can detect whether the external power has failed or not, and has the following syntax:

```
<analog var> := OPERATE(<iovar>,
```

```
O_EXT_POWER, 0);
```

where

`<analog var>`	represents the analog return variable and can be zero, if the operate call is supported and the correct `<var>` is passed, or non-zero, if an error occurred.
`<iovar>`	represents any module variable

4.4 SM-ACI1

4.4.1 Introduction

The SM-ACI1 provides eight optoisolated 80 V AC to 264 V AC inputs arranged in six independent groups with respect to common neutral connections. The input frequency range is 47-63 Hz, with the system registering a logical ’1’ when the input is active. A logical ’0’ is returned when the input is no longer active. However, a low-pass filter restricts signals exceeding the filter limit, thus registering a logical ’0’ with the system.

The user interface is realized by eight green LEDs (one per input channel), which switch on when the inputs are energized, and by a configurable filter on the last channel used for INTx interrupts, that can be mounted on the specific interrupt slots envisaged for this purpose.

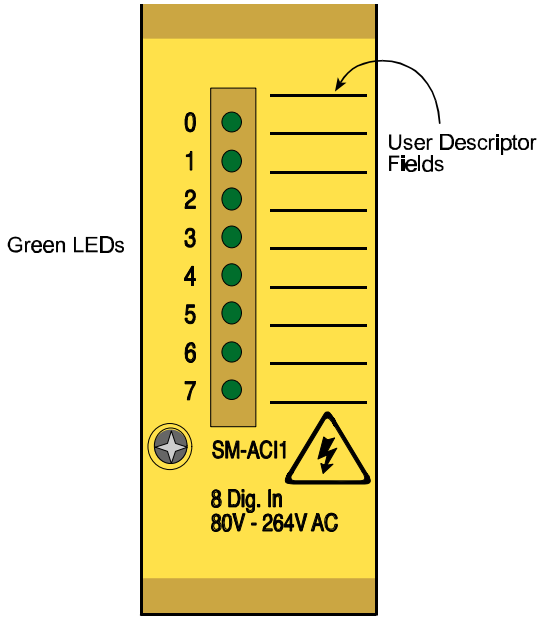
4.4.2 Specifications

Table 4-1: Specifications

Isolation	2.5 kV optoisolated towards the system
Input	8 channels (80 V AC - 264 V AC) Common neutral < 0.5 mA input current Input frequency range 47-63 Hz Overvoltage protection (continuous): 350 V AC Switch-ON delay: < 21 ms Switch-OFF delay: < 16 ms High level: > 20 V Low level: < 10 V
Front panel	Green LEDs (switched on, when INPUT is active)
Options	User-configurable RC filter on the last channel for INTx generation
Power consumption	3 mW min., 70 mW max.
Temperature range	Standard:0°C to +70°C Extended:-40°C to +85°C
Module weight	38 g
ID Byte	$02, read by the SPI interface

4.4.3 Front Panel Layout

Figure 4-1: Front Panel Layout



4.4.4 Board Overview

Figure 4-2: Component Side

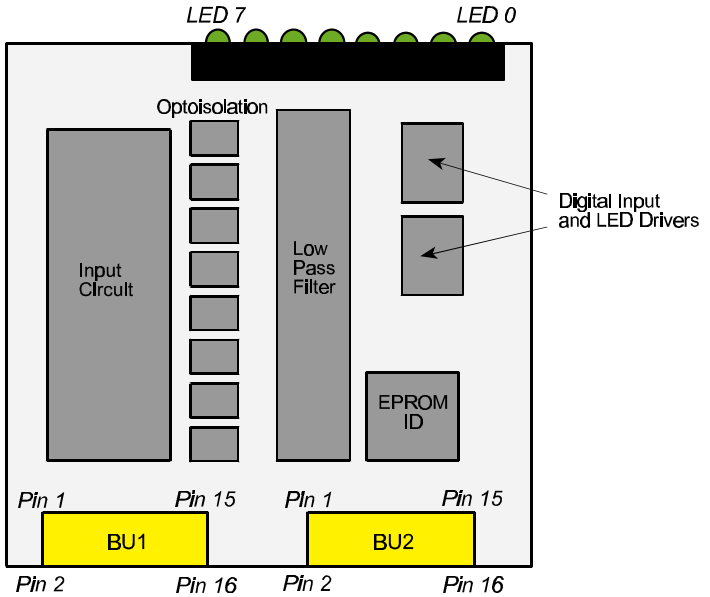
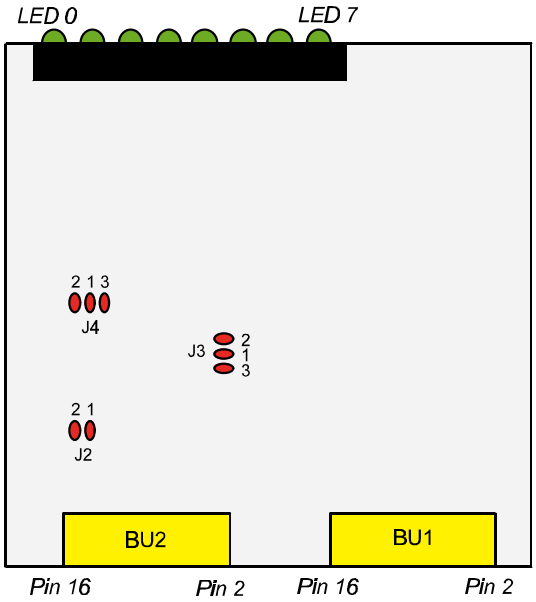
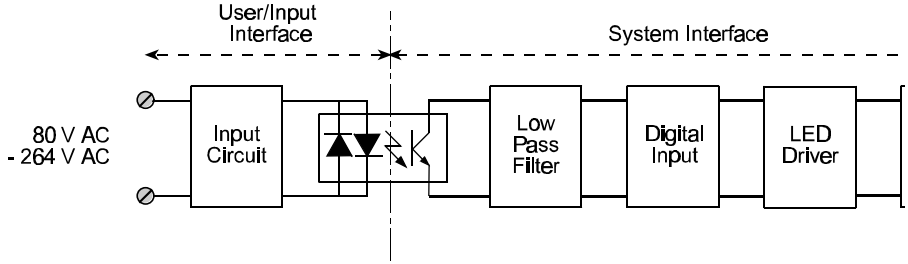


Figure 4-3: Solder Side



4.4.5 Functional Description

Figure 4-4: SM-ACI1 Schematic Diagram



© *PEP Modular Computers*

Downstream the optoisolation part of the circuit, an RC first-order, low-pass filter converts signals greater than 40 Hz into logically suitable values before they enter the system's digital input.

The LED driver activates the relevant LED when the input increases to more than 30 V AC.

As previously mentioned, the last channel is capable of issuing INTx interrupts, when enabled by software. This interrupt will only be acknowledged under OS-9, if the module is on the SMART-BASE, or occupies the first slot of the attached SMART-EXT unit. This channel is manufacturer-set at 40 Hz, but other frequencies may be catered for. The table below shows the settings of the solder jumpers for this filter configuration.

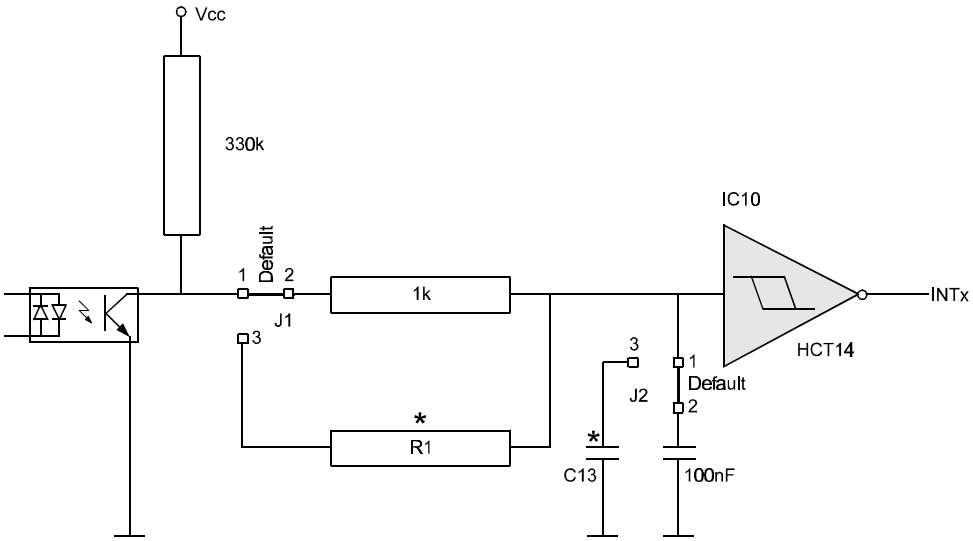
Table 4-2: Jumper Settings

Jumper Settings		Description
J3	1-2 1-3	Standard input filter 40 Hz Customer-specific input filter
J4	1-2 1-3	Standard input filter 40 Hz Customer-specific input filter

4.4.6 Configuration

Although two solder jumpers (J3 and J4) exist on the board, they should remain at their manufacturer settings unless a user-specific filter is required, that for instance cuts off at higher frequencies. If a user-specific filter is required, then contact please *PEP Modular Computers* for any further advice, and refer to the diagram of the filtering components shown in figure 4-5.

Figure 4-5: SM-ACI1 Configurable Filter



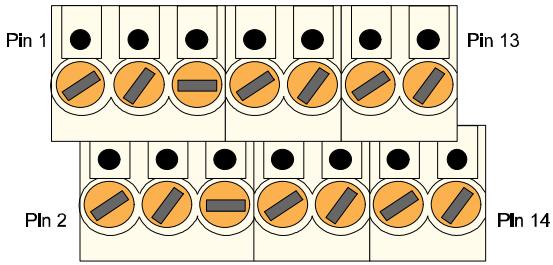
© *PEP Modular Computers*

Surface-Mounted Devices (SMDs) are used in the production of SM-ACI1 modules. The components to be changed, marked with an asterisk in figure 4-5, need not necessarily be SMDs. Please, refer to the board overview figure (solder side, figure 4-3) for the approximate position of the jumpers. Notice should be taken of the fact, that when calculating component values for a specific filter, the capacitor/resistor relationship is almost linear. For this purpose, it is suggested that only the capacitor be changed. Hence, for a doubling of the filter frequency input, the value of the capacitor should be reduced by half.

4.4.7 Screw Terminal Pinouts

In the following, the pinout/signal relationship for the SM-ACI1, when connected to a particular screw terminal block, is shown.

Figure 4-6: Screw Terminal Pinout



WARNING!

Dangerous voltages are present at the terminals.

Table 4-3: Pinout/Signal Relationship

Pin #	Signal	Description	Pin #	Signal	Description
1	N0	Neutral channel 0	2	N4	Neutral channel 4
3	P0	Phase channel 0	4	P4	Phase channel 4
5	P1	Phase channel 1	6	P5	Phase channel 5
7	N1	Neutral channel 1	8	N5	Neutral channel 5
9	P2	Phase channel 2	10	P6	Phase channel 6
11	P3	Phase channel 3	12	P7	Phase channel 7
13	N23	Neutral channels 2, 3	14	N67	Neutral channels 6, 7

Figure 4-7: Connection

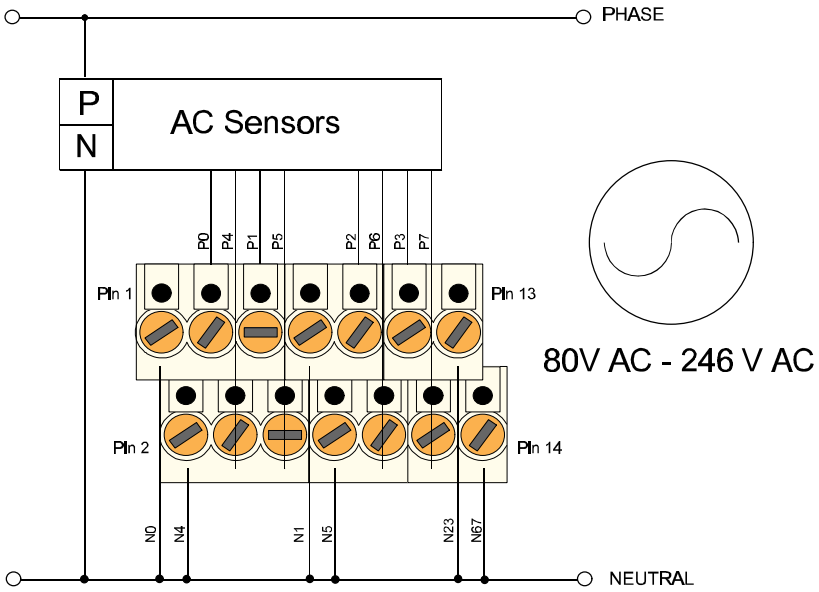
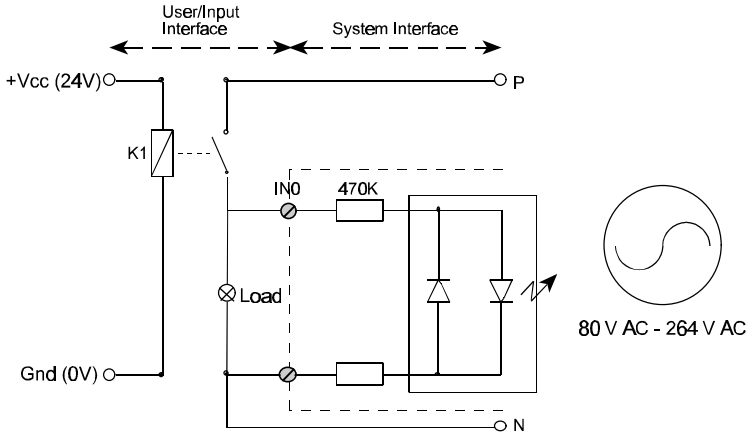


Figure 4-8: Example: Input Circuit (Only One Channel Shown)



4.4.8 ANSI 'C' Programming

4.4.8.1 SM-ACI1 Library

The SM-ACI1 library of functions *smartio.l* provide a convenient way of accessing the SM-ACI1 module.

- All functions are written in ANSI C;
- Prototypes are to be found in the file *aci1lib.h*.

Hardware Requirements

- SMART I/O Base Module or Base Module and Extension unit;
- SM-ACI1 Module.

Software Requirements

The compiler belonging to one of the following environments:

- Ultra C Version 1.1.2 or higher;
- FasTrak for Windows version 2.02 or higher.

Attention: As far as programming is concerned, to the SM-Module on the very left the figure '0' is assigned, although physically it is slot 1.

Before a library function can be used, the function *SMACI1Init* must be called first. By this the requested resources are allocated. Beyond, this function needs to be called for each SM-ACI1 Module called within the task. Upon completion of the application, the function *SMACI1DeInit* needs to be called for each SM-ACI1 Module that has been initialized.

In order to illustrate the use of the SM-ACI1 library, an application facsimile called *demoaci1.c* can be found in the SMART I/O application directory (normally to be found in */<device>/APPLIC/SMART*/SRC).

4.4.8.2 SMACI1Init

Syntax

```
error_code SMACI1Init(u_int8 PortNr);
```

Description:

This function initializes the SM-ACI1 Module on port *PortNr* by allocating space in memory for resources and resetting all inputs.

Input

```
u_int8 PortNr
```

SM-Port number to initialize.

Output

```
error_code SUCCESS E_BMODE
```

If the module is unknown, or standard OS-9 error code (refer to the OS-9 Technical Manual Error Codes Section).

Example

```
RetVal = SMACI1Init(0);
```

4.4.8.3 SMACI1DeInit

Syntax

```
error_code SMACI1DeInit(u_int8 PortNr);
```

Description

This function de-initializes the SM-ACI1 Module on port *PortNr* by releasing all resources assigned to it thereby freeing memory for other uses.

Input

```
u_int8 PortNr
```

SM-Port number to de-initialize.

Output

```
error_code SUCCESS
```

or standard OS-9 error code (refer to the OS-9 Technical Manual Error Codes Section).

Example

```
RetVal = SMACI1DeInit(0);
```

4.4.8.4 SMACI1Get

Syntax

```
error_code SMACI1Get(u_int8 PortNr, u_int8
*buffer);
```

Description

This function fetches the status of ALL 8 input lines of the port *PortNr* and writes it to the address pointed to by *buffer* with bit 0 of the buffer representing input 0 of the module.

Input

```
u_int8 PortNr
```

SM-Port number to fetch data.

```
u_int8 *buffer
```

Pointer to buffer where data is to be stored.

Output

```
error_code SUCCESS
```

or OS-9 error code (refer to the OS-9 Technical Manual Error Codes Section).

Example

```
RetVal = SMACI1Get(0, buffer);
```

Buffer Explanation:

D7							D0
Ch. 7	Ch. 6	Ch. 5	Ch. 4	Ch. 3	Ch. 2	Ch. 1	Ch. 0

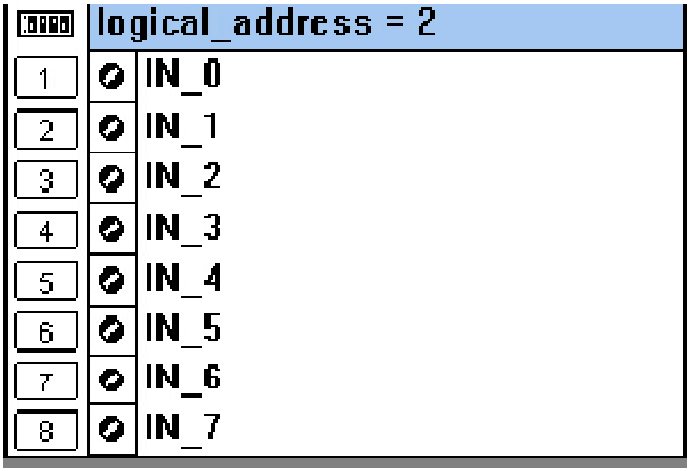
Bit Value: 0 = No input or not connected
1 = Input Active

4.4.9 ISaGRAF Programming

4.4.9.1 ISaGRAF Board Parameters

Information on board parameters may be found in the PEP on-line help and ISaGRAF on-line help or user's manual.

Figure 4-9: Typical Screen Section for the SM-ACI1Module



Logical Address

The 8 inputs of this module may be clearly seen together with the logical address at which this particular board is residing. Remember, this logical address is the same as the physical slot position! In the example shown here, the board being addressed is in slot 2 i.e. the second of the SMART-BASE slots. Up to 11 slots are catered for in the SMART I/O concept; 3 on the SMART-BASE and 2 for each connected SMART-EXT (up to a total of 4).

4.4.9.2 ISaGRAF Operate Calls

Operate calls are built into a program using ST or FBD languages when defining the project. A typical use could be at the initialization stage to check that the SMART-Modules are in fact located where they have been programmed to be. The syntax of the operate call is as follows:

Syntax

```
<return variable> := OPERATE(<source var>,
COMMAND,<source data>);
```

Here the *return variable* is assigned a value associated with the selected *COMMAND* parameter. Each SMART-Module possesses its own set of such COMMANDS.

Example

```
<error.code>:= OPERATE(<channel>,
O_INIT_CODE, 0);
```

O_INIT_CODE is one of a number of distinct commands recognized by the *PEP Modular Computers'* implementation of board drivers and checks, for example, that the board is located where the program expects it to be.

Channel provides channel specific information and in the example shown here, any of the 8 input channels may be used. The last parameter is not usually used by PEP implementations and is set to 0 (zero).

The error.code returns a value of zero if no error was detected, otherwise it returns a non-zero value depending on the error encountered. A list of these error codes may be found in the PEP on-line help.

A complete list of the operate *COMMANDS* may be obtained by selecting a project from the ISaGRAF *projects* group, opening an application and observing the *Common defines* in the *Dictionary* pull-down menu. Note that not all calls in the list may be used within the SMART I/O environment, however, the calls applicable to this module are:

`O_INIT_CODE`

Syntax and usage as explained above.

4.5 SM-ACO1

4.5.1 Introduction

The SM-ACO1 provides power switching up to 264 V AC (max.) with the capability to switch inductive loads as a built-in feature of this versatile module. Six TRIAC (three-terminal AC semiconductor) switches are connected to six TRIAC status LEDs on the front panel, and two user programmable LEDs complete the user interface.

4.5.2 Specifications

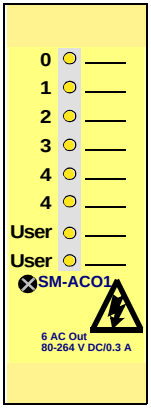
Table 5-1: SM-ACO1 Technical Specifications

SM-ACO1	Specifications
Isolation • System • Between Channels • Between Contacts	 2.5 kV optoisolated 1.5 kV 1.0 kV
Outputs	6 TRIAC Conductors
Output Voltage	• SMART2: 80...125 V AC (terminal limitation) • SMART I/O: 80..264 V AC
Current Voltage	Peak current: 3.0 A; Over-temperature 70°C: 0.3 A; Over-temperature 85°C: 0.2 A Load (Resistive): 720 W Frequency: 47..68 Hz Delay Time: ≤10.6 ms Zero-Cross Switching Protection: glitch-free “Power ON”
Power Consumption	Minimum: 30 mW Maximum: 110 mW

Table 5-1: SM-ACO1 Technical Specifications

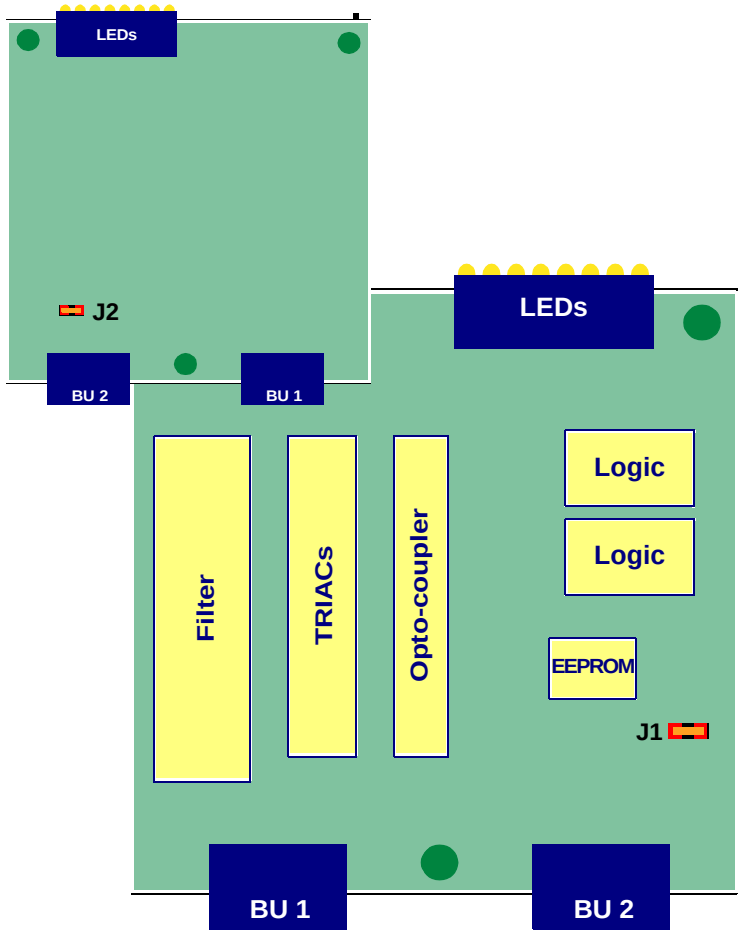
SM-ACO1	Specifications
User Interface	6 TRIAC status LEDs 2 user programmable LEDs
Temperature Range: • Industrial • Extended	 0°C to +70°C -40°C to +85°C
ID Byte	$30 (read by the SPI interface)
Operating Humidity	10% to 95%, non-condensing.

4.5.3 Front Panel Layout



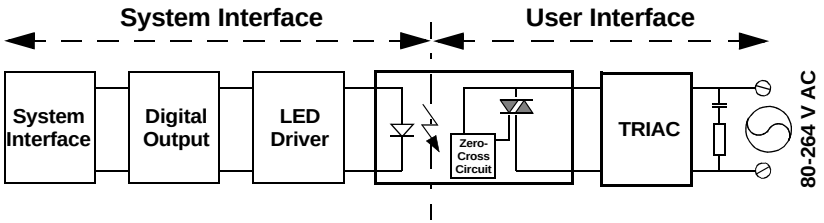
4.5.4 Board Overview

Figure 4-1:SM-ACO1 Schematic Board Layout — Front View and Reverse View (Small Figure)



4.5.5 Functional Description

Figure 4-2:SM-ACO1 Functional Diagram



The single TRIACs are low active, which means that a digital “0” delivered by the system activates them illuminating the corresponding LEDs on the front panel.

Zero-Cross Switching

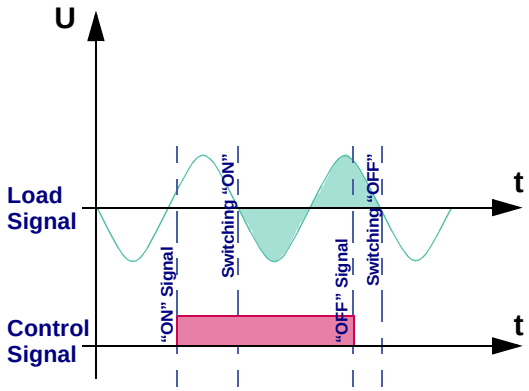
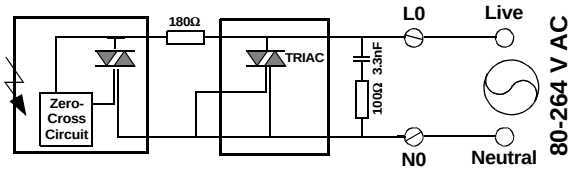


Figure 4-3: Zero-Cross Switching

The TRIAC output is switched ON/OFF when the load signal crosses the zero value. Thus high switching currents and the consequent losses and switching disturbances are avoided. The maximum switching delay time resulting therefrom amounts to a half-sinus, i.e. ≤10.6 ms (47 Hz).

256 Bytes of EEPROM contain the module ID number and production data although space has been allocated for future use.

Figure 4-4:SM-ACO1 Input Circuit



4.5.6 Configuration

Jumper J1

This jumper is factory-set. Its setting may not be changed by the customer.

Jumper J2

This jumper enables/disables the EEPROM hardware write protection. It may be set as follows:

Table 4-2: Jumper J2 – EEPROM Protection

Setting	Description
Closed	EEPROM hardware write protection disabled
Open	EEPROM hardware write protection enabled

4.5.7 Pinouts

Figure 4-5 shows the pinout of the SM-ACO1 module front connector.

Attention!

Due to a max. operating voltage of 125 V AC of the SMART2-BASE terminal blocks, a SMART-EXT carrier unit must be used with the SM-ACO1 module, should your system require the handling of signals in the 80..264 V AC range.

High Voltage!

When connecting/disconnecting loads to your SM-ACO1 module, please ensure that the system and field power supplies are not powered. Otherwise dangerous voltages are present at the terminals that might endanger your health/life or damage your module/system.

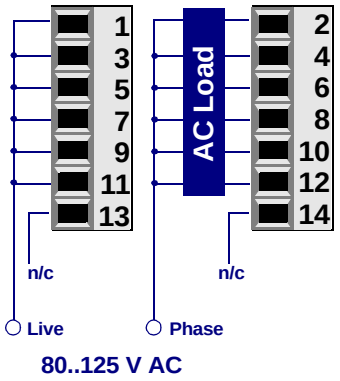


Figure 4-5:SM-ACO1 Terminal Block Connection for SMART2

Table 4-3: SM-ACO1 Front Connector Pinout

Pin	Signal	Description	Pin	Signal	Description
1	L0	Live Ch. "0"	2	N0	Neutral Ch. "0"
3	L1	Live Ch. "1"	4	N1	Neutral Ch. "1"
5	L2	Live Ch. "2"	6	N2	Neutral Ch. "2"
7	L3	Live Ch. "3"	8	N3	Neutral Ch. "3"
9	L4	Live Ch. "4"	10	N4	Neutral Ch. "4"
11	L5	Live Ch. "5"	12	N5	Neutral Ch. "5"
13	n/c	Not connected	14	n/c	Not connected

4.5.8 ISaGRAF

PEP Advantage

For users of the ISaGRAF software it is not necessary to program this interface, since it is completely handled by ISaGRAF.

PEP Advantage

All necessary and/or useful information on ISaGRAF is available directly via software in the ISaGRAF on-line help program.

4.5.9 'C' Programming

4.5.9.1 SM-ACO1 Library

The SM-ACO1 library of functions *smartio.l* provides a convenient way of accessing the SM-ACO1 module:

- All functions are written in ANSI C;
- Prototypes are to be found in the file *dout1lib.h*.

Hardware Requirements

- SMART2-BASE carrier unit or equivalent SMART I/O unit;
- SM-ACO1 module.

Software Requirements

One of the following compilers:

- UltraC, version 1.1.2 or higher;
- FasTrak, version 2.0.2 or higher.

For information about the numbering of the SM/SM2 modules installed in your SMART2 system, programmers should refer to page 2-6: "Terminology".

Important!

Generally it can be said that SMART2 and SMART I/O system slots are numbered from "0" to the maximum "14". Thus, the leftmost SM/SM2 module is referred to as "0".

Before a library function can be used, the function *SMACO1Init* must be called first. This allocated the required resources. Furthermore, this function needs to be called for each SM-ACO1 module called within a task. Upon

completion of the application, the function *SMACO1DeInit* needs to be called for each SM-ACO1 module that has been initialized.

> ***Attention!***
>
> If the same SM-ACO1 module must be accessed by different tasks, programmers should make sure that their actual output status is guaranteed by using, for instance, OS-9 semaphores.

4.5.9.2 SMACO1Init()

Syntax

```
error_code SMACO1Init(u_int8 PortNum);
```

Description

This function initializes the SM-ACO1 module on port number *PortNum* and sets all output lines to *OFF*.

Input

`u_int8 PortNum`	SMART2 port number to be initialized (0 through 14)

Output

`error_code` `SUCCESS`

`E_BMODE` in case the module is unknown

or standard OS-9 error code (please refer to the “Error Codes” section of the OS-9 Technical Manual).

Example

```
RetVal = SMACO1Init(0).
```

4.5.9.3 SMACO1DeInit

Syntax

```
error_code SMACO1DeInit(u_int8 PortNum);
```

Description

This function de-initializes the SM-ACO1 module on port number *PortNum* by releasing all resources assigned to it and thus freeing memory resources for other users.

Input

`u_int8 PortNum`	SMART2 port number to be de-initialized (0 through 14)

Output

`error_code SUCCESS`

or standard OS-9 error code (please refer to the “Error Codes” section of the OS-9 Technical Manual).

Example

```
RetVal = SMACO1DeInit(0).
```

4.5.9.4 SMACO1Get()

Syntax

```
error_code SMACO1Get(u_int8 PortNum, u_int8 *buffer);
```

Description

This function fetches the output status of port number *PortNum* and writes it to the memory location pointed to by *buffer*. This buffer shows the status of all eight outputs with bit “0” representing output “0” of the module.

> ***Important!***
>
> The buffer only has the function of memorizing the output status. It does not read back the process output values.

Input

`u_int8 PortNum`	SMART2 port number to fetch data;
`u_int8 *buffer`	Pointer to a buffer where data are stored.

Output

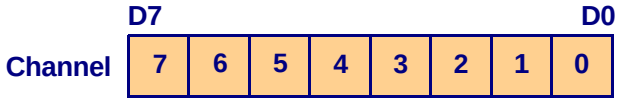
`error_code SUCCESS`

or standard OS-9 error code (please refer to the “Error Codes” section of the OS-9 Technical Manual).

Example

```
RetVal = SMACO1DeGet(0, buffer).
```

Buffer Explanation



Bit Value: **“0” = Output ON**
“1” = Output “OFF”

4.5.9.5 SMACO1Set()

Syntax

```
error_code SMACO1Set(u_int8 PortNum, u_int8 *buffer);
```

Description

This function sets the output of port number *PortNum* to the contents of the address pointed to by *buffer*. When writing to this buffer, programmers should observe that all eight outputs are set at the same time.

Input

`u_int8 PortNum`	SMART2 port number to write data;
`u_int8 *buffer`	Pointer to a buffer where data are written.

Output

`error_code SUCCESS`

or standard OS-9 error code (please refer to the “Error Codes” section of the OS-9 Technical Manual).

Example

```
RetVal = SMACO1Set(0, buffer).
```

Buffer Explanation



Bit Value: **“0” = Output ON**
“1” = Output “OFF”

4.5.9.6 SMACO1SetLed

Syntax

```
error_code SMACO1SetLed(u_int8 PortNum, u_int8 LedNum);
```

Description

This function switches *on* LED number *LedNum*, i.e. one of the two user programmable LEDs on the SM-ACO1 front panel.

Input

`u_int8 PortNum`	SMART2 port number to be accessed;
`u_int8 LedNum`	Number of the LED to switch *on* (1-2).

Output

`error_code` `SUCCESS`

`E_SEEK` in case the module was not initialized

`E_BMODE` in case the module is unknown

or standard OS-9 error code (please refer to the “Error Codes” section of the OS-9 Technical Manual).

Example

```
RetVal = SMACO1SetLed(0, 2).
```

4.5.9.7 SMACO1ClrLed()

Syntax

```
error_code SMACO1ClrLed(u_int8 PortNum, u_int8 LedNum);
```

Description

This function switches *off* LED number *LedNum*, i.e. one of the two user programmable LEDs on the SM-ACO1 front panel.

Input

`u_int8 PortNum`	SMART2 port number to be accessed;
`u_int8 LedNum`	Number of the LED to switch *off* (1-2).

Output

`error_code SUCCESS`

`E_SEEK` in case the module was not initialized

`E_BMODE` in case the module is unknown

or standard OS-9 error code (please refer to the “Error Codes” section of the OS-9 Technical Manual).

Example

```
RetVal = SMACO1ClrLed(0, 2).
```

Analog Modules

© *PEP Modular Computers*

Table of Contents

Chapter 5

5.1 SM-DAD1 5-5

5.1.1 Introduction 5-5
5.1.2 Specifications 5-5
5.1.3 Front Panel Layout 5-6
5.1.4 Board Overview 5-7
5.1.5 Functional Description 5-8
5.1.6 Configuration 5-10
5.1.7 Pinouts 5-11
5.1.8 'C' Programming 5-14
5.1.9 ISaGRAF Programming 5-23

5.2 SM-PT100 5-27

5.2.1 Introduction 5-27
5.2.2 Specifications 5-27
5.2.3 Front Panel Layout 5-28
5.2.4 Board Overview 5-29
5.2.5 Functional Description 5-31

5.2.6 Configuration 5-31
5.2.7 Pinouts 5-33
5.2.9 ISaGRAF Programming 5-54
5.3 SM-THERM 5-57
5.3.1 Introduction 5-57
5.3.2 Specifications 5-57
5.3.3 Front Panel Layout 5-58
5.3.4 Board Overview 5-58
5.3.5 Functional Description 5-59
5.3.6 Configuration 5-60
5.3.7 Pinouts 5-60
5.3.8 'C' Programming 5-62
5.3.9 ISaGRAF Programming 5-83
5.4 SM-ADC1 5-87
5.4.1 Introduction 5-87
5.4.2 Specifications 5-87
5.4.3 Front Panel Layout 5-88
5.4.4 Board Overview 5-88
5.4.5 Functional Description 5-89
5.4.6 Configuration 5-89
5.4.7 Screw Terminal Pinouts 5-90
5.4.8 'C' Programming 5-90
5.4.9 ISaGraf Programming 5-99

5.5 SM-DAC1 5-105
5.5.1 Introduction 5-105
5.5.2 Specifications 5-105
5.5.3 Front Panel Layout 5-106
5.5.4 Board Overview 5-106
5.5.5 Functional Description 5-107
5.5.6 Configuration 5-108
5.5.7 Pinouts 5-108
5.5.8 'C' Programming 5-111
5.5.9 ISaGRAF Programming 5-121

5. Analog Modules

5.1 SM-DAD1

5.1.1 Introduction

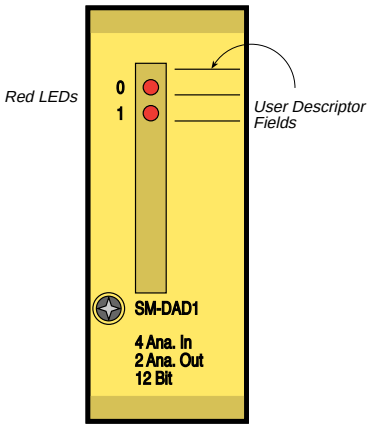
The SM-DAD1 provides a fast, 12-bit, bipolar (± 10V DC), 4-channel analog to digital converter and, apart from its ADC role, may also be used as a fast 12-bit, 2-channel unipolar/bipolar digital to analog converter. Two configurable red LEDs on the front panel form the user interface while an on-board EEPROM stores unique calibration data (obtained in steady-state conditions at 25°C STP) required by the signal converters.

5.1.2 Specifications

Input	4 Differential Voltage Inputs Input Range ± 10V 12-bit ADC with 1-bit non-linearity 10-bit repeating accuracy Overvoltage Protection (± 35V) Input Impedance approx. 450 kΩ Filter set to 10kHz On-chip Reference Voltage 13µs Conversion Time @ 1MHz SCLK
Output	2 Unipolar / Bipolar Outputs Short-Circuit Protected 12-bit DAC with 1-bit non-linearity Output Range ± 10V or 0..10V 5mA max. Output Current Power-up Reset without glitch (Unipolar) 4mV Zero Code Offset Error (Unipolar) Settling Time 40µs (-10V to +10V)
EEPROM	256-byte serial EEPROM for Calibration Data
Front Panel	Red LEDs (user configurable)
Power Cons.	350mW (min.), 450mW (max.)
Temperature Range	Standard (0°C to 70°C), Extended (-40°C to +85°C)
Temperature Drift	6 LSB (max.) across temp. range -40°C to +85°C (ADC) 4 LSB (max.) across temp. range -40°C to +85°C (DAC)
Module Weight	70g
ID Byte	$50, read by the SPI Interface

© PEP Modular Computers GmbH

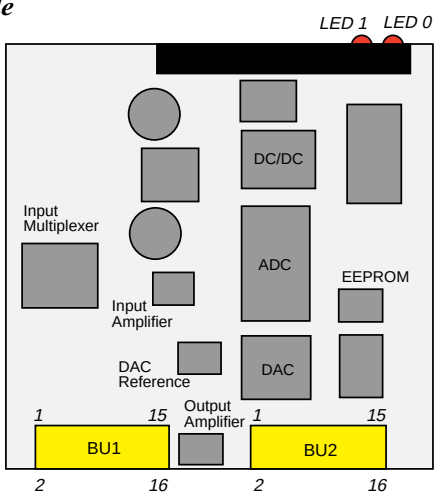
5.1.3 Front Panel Layout



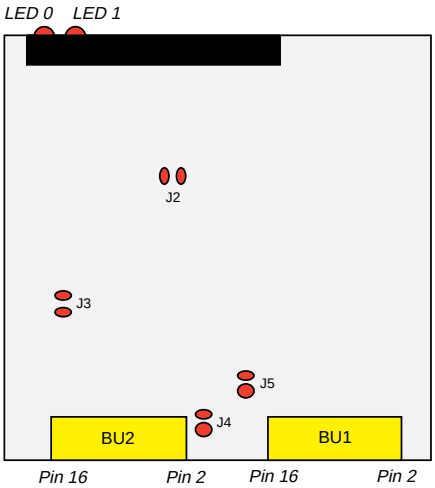
© PEP Modular Computers GmbH

5.1.4 Board Overview

Component Side



Solder Side



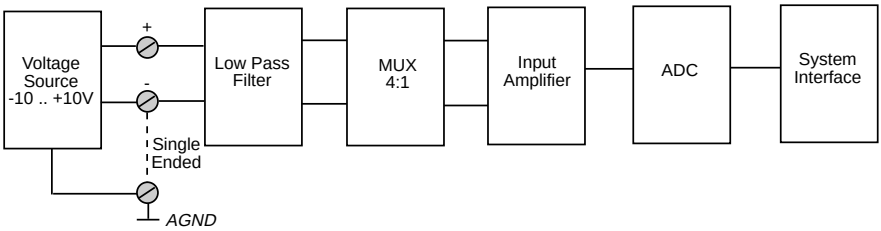
5.1.5 Functional Description

The SM-DAD1 has 4 differential voltage inputs and 2 unipolar/bipolar voltage outputs. An on-board EEPROM contains board specific calibration data, module ID byte and production data. An on-board switched-mode regulator provides ±15V for the analog I/O and an additional linear regulator provides the -5V required by the ADC.

5.1.5.1 Input Circuitry

The input circuitry of the SM-DAD1 consists of an R-C first-order, low-pass filter to prevent noise and signals greater than 10kHz from entering the system, a 1:4 channel multiplexer, followed by a differential amplifier, divider and a 12-bit fast analog to digital converter (ADC).

Figure 5.1.5.1: SM-DAD1 Input Schematic Diagram



The ADC, using successive approximation and input track and hold is referenced by its own on-chip voltage source supplying 4.096V. To cater for bipolar full-scale inputs, this reference source is split (-2.048V .. +2.048V).

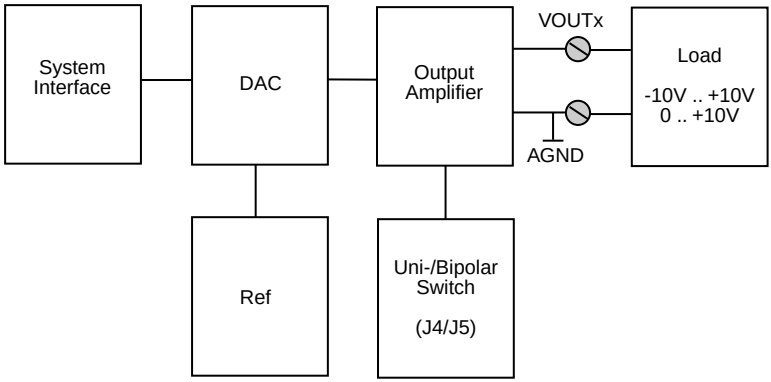
The incoming voltage signals are multiplexed and reduced via a 1:5 divider network so that the ADC is supplied with a scalable input voltage range.

© PEP Modular Computers GmbH

5.1.5.2 Output Circuitry

The digital to analog converter (DAC) section consists of a twin output 12-bit DAC with external (on-board) 10.0V reference and an operational amplifier for unipolar/bipolar outputs.

Figure 5.1.5.2: SM-DAD1 Output Schematic Diagram



The DAC is a complete dual 12-bit multiplying DAC without the need for external trimming. Bipolar/unipolar mode is selected via jumpers J4 and J5 which allow both output channels to be configured separately.

It should be noted that in bipolar mode, the power-up condition is not glitch free. Therefore it is suggested that the connected load be switched a short time later to provide enough time for the DAC to initialize.

© PEP Modular Computers GmbH

5.1.6 Configuration

The SM-DAD1 has 5 solder jumpers which may be configured as follows:

Jumpers J1 and J2

These jumpers are reserved for PEP use only.

Jumper J3 - EEPROM Protection

Jumper	Settings	Description
J3	*set*	*EEPROM is not hardware write protected*
	open	EEPROM is hardware write protected

Jumper J4 and J5 - DAC Output

Jumper	Settings	Description
J4	*open*	*VOUT0 unipolar 0V .. +10V*
	set	VOUT0 bipolar -10V .. +10V
J5	*open*	*VOUT1 unipolar 0V .. +10V*
	set	VOUT1 bipolar -10V .. +10V

Note

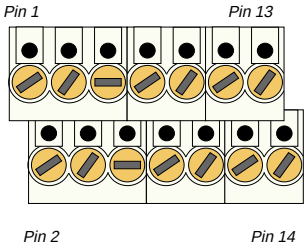
If altering the mode (unipolar/bipolar) of the DAC output, be sure that software is aware of the change.

 © PEP Modular Computers GmbH

5.1.7 Pinouts

Screw Terminal Pinouts

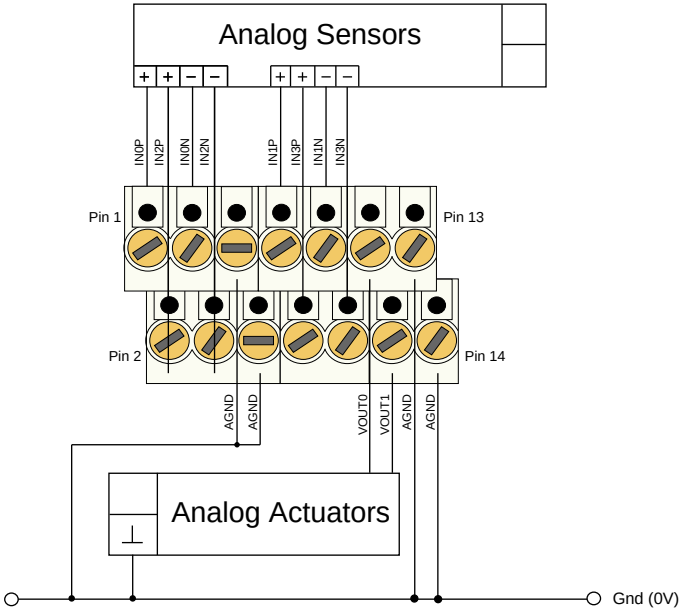
The following shows the pinout/signal relationship for the SM-DAD1 when connected to a particular screw terminal block.



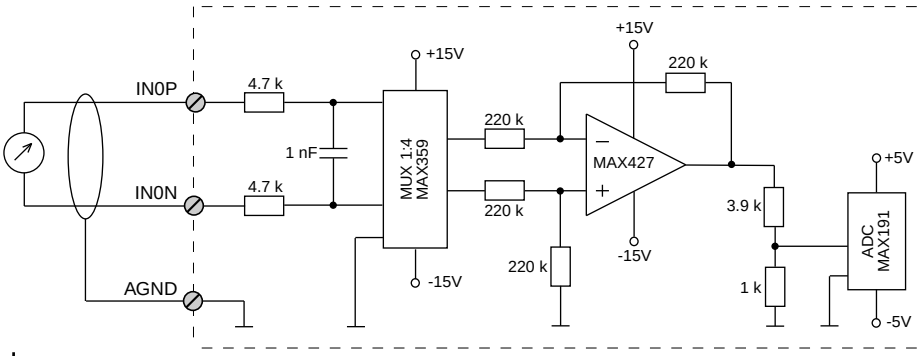
Pin Nr.	Signal	Description	Pin Nr.	Signal	Description
1	IN0P (+)	Analog Input Ch. 0+	2	IN2P (+)	Analog Input Ch. 2+
3	IN0N (-)	Analog Input Ch. 0-	4	IN2N (-)	Analog Input Ch. 2-
5	AGND	Analog GND	6	AGND	Analog GND
7	IN1P (+)	Analog Input Ch. 1+	8	IN3P (+)	Analog Input Ch. 3+
9	IN1N (-)	Analog Input Ch. 1-	10	IN3N (-)	Analog Input Ch. 3-
11	VOUT0	Analog Output Ch. 0	12	VOUT1	Analog Output Ch. 1
13	AGND	Analog GND	14	AGND	Analog GND

© PEP Modular Computers GmbH

Connection

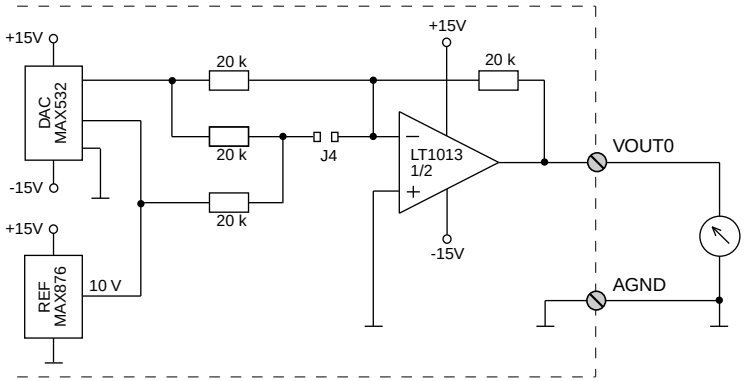


A/D Circuit



© PEP Modular Computers GmbH

D/A Circuit



Jumpers J4 and J5 should be set if the DAC should operate in bipolar mode. It should be noted that the D/A circuit shown above represents only one of the two output channels. J5 is naturally the jumper for mode switching on the second channel.

© PEP Modular Computers GmbH

5.1.8 'C' Programming

5.1.8.1 SM-DAD1 Library

The SM-DAD1 library of functions *smartio.l* provide a convenient way of accessing the SM-DAD1 module.

- All functions are written in ANSI C;
- Prototypes are to be found in the file *dad1lib.h*.

Hardware Requirements

- SMART 2 Base Unit or SMART I/O Base and Extension Units;
- SM-DAD1 Module.

Software Requirements

The compiler from one of the following:

- Ultra C Version 1.1.2 or higher;
- FasTrak 2.0.2 or higher.

The leftmost SM-Module has number 0 assigned to it a far as programming is concerned although physically this is slot 1!

Before a library function can be used, the function *SMDAD1Init* must first be called. This allocates the requested resources. Furthermore, this function needs to be called for each SM-DAD1 Module called within the task. Upon completion of the application, the function *SMDAD1DeInit* needs to be called for each SM-DAD1 Module that has been initialized.

In order to illustrate the use of the SM-DAD1 library, an application example called *demodad1.c* can be found in the SMART I/O application directory (normally found in */<device>/APPLIC/SMART/CMDS*).

 © PEP Modular Computers GmbH

Note

If the same SM-DAD1 Module is to be accessed from different tasks, the user must make sure that the actual output status is ensured by using for example, semaphores within OS-9.

5.1.8.2 SMDAD1Init

Syntax

```
error_code SMDAD1Init(u_int8 PortNr);
```

Description

This function initializes the resources needed to use the SM-DAD1 on port *PortNr*, including:

- testing of the SM-Module ID;
- reading the factory-set conversion table held in the EEPROM;
- setting the ADC to bipolar mode and the DAC to unipolar mode;
- fetching the resources.

Input

`u_int8 PortNr`	SM-Port number to initialize (from 0 to 10)

Output

`error_code`	`SUCCESS` or	
	`E_BMODE`	if wrong or no module
	`E_MEMFUL`	if no memory resources available

or standard OS-9 error code (refer to the OS-9 Technical Manual Error Codes Section).

Example

```
RetVal = SMDAD1Init(0);
```

5.1.8.3 SMDAD1DeInit

Syntax

```
error_code SMDAD1DeInit(u_int8 PortNr);
```

Description

This function de-initializes the SM-DAD1 Module on the port *PortNr* by freeing the resources tied up with it and setting all outputs and inputs to zero.

Input

`u_int8 PortNr`	SM-Port number to de-initialize (from 0 to 10)

Output

`error_code`	`SUCCESS`
	`E_SEEK` if module not initialized
	or standard OS-9 error code (refer to the OS-9 Technical Manual Error Codes Section).

Example

```
RetVal = SMDAD1DeInit(0);
```


© PEP Modular Computers GmbH

5.1.8.4 SMDAD1GetVRaw

Syntax

```
error_code SMDAD1GetVRaw(u_int8 PortNr, u_int8
                         Channel, int16 *pValue);
```

Description

This function fetches the conversion result of a selected channel without performing a gain or offset correction. The result is always a signed integer value. If bipolar mode is selected, the function returns the 2's complement value of the original read.

Input

`u_int8 PortNr`	SM-Port to be accessed
`u_int8 Channel`	Number of the ADC input channel to access (0-3)
`int16 *pValue`	Pointer to the location where the conversion result will be stored

Output

`error_code`

`SUCCESS`	
`E_SEEK`	if module was not initialized
`E-BMODE`	if the module is unknown

or standard OS-9 error code (refer to the OS-9 Technical Manual Error Codes Section).

Example

```
RetVal = SMDAD1GetVRaw(0, 2, 0x07FF);
```

Value (16-bit HEX)			
Unipolar		**Bipolar**	
Positive Full-Scale	$07FF	Positive Full-Scale	$07FF
+1LSB	$0001	+1LSB	$0001
0	$0000	-1LSB	$FFFF
		Negative Full-Scale	$F800

© PEP Modular Computers GmbH

5.1.8.5 SMDAD1GetV

Syntax

```
error_code SMDAD1GetV(u_int8 PortNr, u_int8 Channel,
                      int16 *pValue);
```

Description

This function fetches the conversion result of a selected channel. A gain and offset correction is done with the use of the selected conversion table. The result is always a signed integer value. If bipolar mode is selected, the function returns the 2's complement value of the original read.

Input

`u_int8 PortNr`	SM-Port to be accessed
`u_int8 Channel`	Number of the ADC input channel to access (0-3)
`int16 *pValue`	Pointer to the location where the conversion result will be stored

Output

`error_code`	`SUCCESS`	
	`E_SEEK`	if module was not initialized
	`E-BMODE`	if the module is unknown

or standard OS-9 error code (refer to the OS-9 Technical Manual Error Codes Section).

Example

```
RetVal = SMDAD1GetV(0, 2, 0x07FF);
```

Value (16-bit HEX)			
Unipolar		**Bipolar**	
Positive Full-Scale	$07FF	Positive Full-Scale	$07FF
+1LSB	$0001	+1LSB	$0001
0	$0000	-1LSB	$FFFF
		Negative Full-Scale	$F800

© PEP Modular Computers GmbH

5.1.8.6 SMDAD1PutVRaw

Syntax

```
error_code SMDAD1PutVRaw(u_int8 PortNr, u_int8
                         Channel, u_int16 Value);
```

Description

This function sets the selected DAC channel to the chosen value. No gain or offset correction is performed.

Input

`u_int8 PortNr`	SM-Port number to be accessed
`u_int8 Channel`	Number of the DAC output channel to access (0-1)
`u_int16 Value`	Value to be written

Output

`error_code`	`SUCCESS`	
	`E_SEEK`	if module was not initialized
	`E-BMODE`	if the module is unknown

or standard OS-9 error code (refer to the OS-9 Technical Manual Error Codes Section).

Example

```
RetVal = SMDAD1PutVRaw(0, 1, 0x0FFF);
```

Value (16-bit HEX)			
Unipolar		**Bipolar**	
Positive Full-Scale	$0FFF	Positive Full-Scale	$0FFF
+1LSB	$0001	+1LSB	$0801
0	$0000	0	$0800
		-1LSB	$07FF
		Negative Full-Scale	$0000

5.1.8.7 SMDAD1PutV

Syntax

```
error_code SMDAD1PutV(u_int8 PortNr, u_int8 Channel,
                      int16 Value);
```

Description

This function sets the selected DAC channel to the chosen value. A gain and offset correction is performed with the use of the automatically selected conversion table depending whether a user-defined table has been specified or not. Hence, if a user-defined table exists then this will be used, otherwise the default factory set table will be used.

Input

`u_int8 PortNr`	SM-Port number to be accessed
`u_int8 Channel`	Number of the DAC output channel to access (0-1)
`int16 Value`	Value to be written

Output

`error_code`

`SUCCESS`	
`E_SEEK`	if module was not initialized
`E-BMODE`	if the module is unknown

or standard OS-9 error code (refer to the OS-9 Technical Manual Error Codes Section).

Example

```
RetVal = SMDAD1PutV(0, 1, 0x7FF);
```

Value (16-bit HEX)			
Unipolar		**Bipolar**	
Positive Full-Scale	$0FFF	Positive Full-Scale	$07FF
+1LSB	$0001	+1LSB	$0001
0	$0000	0	$0000
		-1LSB	$FFFF
		Negative Full-Scale	$F800

© PEP Modular Computers GmbH

5.1.8.8 SMDAD1SetLed

Syntax

```
error_code SMDAD1SetLed(u_int8 PortNr, u_int8 LedNr);
```

Description

This function switches on one of the two LEDs fitted to the SM-DAD1.

Input

`u_int8 PortNr`	SM-Port number to be accessed
`u_int8 LedNr`	Number of the LED to switch on (1-2)

Output

`error_code`	`SUCCESS`	
	`E_SEEK`	if module was not initialized
	`E-BMODE`	if the module is unknown
	or standard OS-9 error code (refer to the OS-9 Technical Manual Error Codes Section).	

Example

```
RetVal = SMDAD1SetLed(0, 2);
```


© PEP Modular Computers GmbH

5.1.8.9 SMDAD1ClrLed

Syntax

```
error_code SMDAD1ClrLed(u_int8 PortNr, u_int8 LedNr);
```

Description

This function switches off one of the two LEDs fitted to the SM-DAD1.

Input

`u_int8 PortNr`	SM-Port number to be accessed
`u_int8 LedNr`	Number of the LED to switch off (1-2)

Output

`error_code`	`SUCCESS`	
	`E_SEEK`	if module was not initialized
	`E-BMODE`	if the module is unknown
	or standard OS-9 error code (refer to the OS-9 Technical Manual Error Codes Section).	

Example

```
RetVal = SMDAD1ClrLed(0, 2);
```

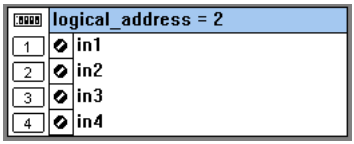

© PEP Modular Computers GmbH

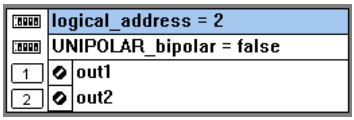
5.1.9 ISaGRAF Programming

5.1.9.1 The ISaGRAF Board Parameters

Information on board parameters may be found in the PEP online help and ISaGRAF online help or user's manual.

Figure 5.1.9.1.1 Typical Screen Sections for the SM-DAD1 Module





Logical Address

The 4 inputs and 2 outputs of this module may be clearly seen together with the logical address at which this particular board is residing. Remember, this logical address is the same as the physical slot position! In the example shown here, the board being addressed is in slot 2 i.e. the second of the SMART-BASE slots. Up to 11 slots are catered for in the SMART I/O concept; 3 on the SMART-BASE and 2 for each connected SMART-EXT (up to a total of 4).

Module Specific Information

`UNIPOLAR_bipolar :` Like all I/O board parameters, the default is set in upper-case lettering. Here, in this example both outputs of the module are set to function in bipolar mode. Hardware jumpers J4 and J5 must be set accordingly as it is impossible under ISaGRAF to configure the outputs individually.

5.1.9.2 The ISaGRAF Operate Calls

Operate calls are built into a program using ST or FBD languages when defining the project. A typical use could be at the initialization stage to check that the SMART-Modules are in fact located where they have been programmed to be. The syntax of the operate call is as follows:

Syntax

```
<return variable> := OPERATE(<source var>, COMMAND,
                             <source data>);
```

Here the `return variable` is assigned a value associated with the selected `COMMAND` parameter. Each SMART-Module possesses its own set of these `COMMANDS`.

Example

```
<error.code>:= OPERATE(<channel>, O_INIT_CODE, 0);
```

`O_INIT_CODE` is one of a number of distinct commands recognized by the PEP Modular Computers' implementation of board drivers and checks for example that the board is located where the program expects it to be.

`channel` provides channel specific information and in the example shown here, any of the 8 input channels may be used. The last parameter is not usually used by PEP implementations and is set to 0 (zero).

The `error.code` returns a value of zero if no error was detected, otherwise it returns a non-zero value depending on the error encountered. A list of these error codes may be found in the PEP online help.

 © PEP Modular Computers GmbH

A complete list of the operate COMMANDS may be obtained by selecting a project from the ISaGRAF ***projects*** group, opening an application and observing the ***Common defines*** in the ***Dictionary*** pull-down menu. Note that not all calls in the list may be used within the SMART 2 environment however, the calls applicable to this module are :

O_INIT_CODE : The syntax and usage have already been explained.

O_SET_LED : This operate call allows the user to control the yellow LED on the board's front panel. The syntax is as follows:

```
<analog var> := OPERATE(<iovar>, O_SET_LED, <var>);
```

where `<analog var>` represents the analog return variable and can be zero if the operate call is supported and the correct `<var>` is passed or non-zero if an error occurred.

`<iovar>` represents any module variable and

`<var>` represents the LED to be controlled and must be one of `OA_LED1_ON` or `OA_LED1_OFF`.

O_ENABLE_CONV : This operate calls tells ISaGRAF to enable the conversion of a particular channel of a connected SM-DAD1 module. The syntax is as follows:

```
<analog var> := OPERATE(<iovar>, O_ENABLE_CONV, 0);
```

where `<analog var>` represents the analog return variable and can be zero if the operate call is supported and the correct `<iovar>` is passed or non-zero if an error occurred.

`<iovar>` is the variable name for the channel where the operate call is to take effect.

O_DISABLE_CONV : This operate call tells ISaGRAF to disable the conversion of a particular channel of a connected SM-DAD1 module. The syntax is as follows:

```
<analog var> := OPERATE(<iovar>, O_DISABLE_CONV, 0);
```

where `<analog var>` represents the analog return variable and can be zero if the operate call is supported and the correct `<iovar>` is passed or non-zero if an error occurred.

`<iovar>` is the variable name for the channel where the operate call is to take effect.

© PEP Modular Computers GmbH

5.2 SM-PT100

5.2.1 Introduction

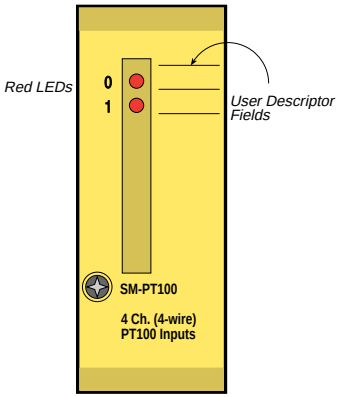
The SM-PT100 is used for 2, 3 or 4-wire temperature measurement using PT100 sensors driven by an on-board constant current source. Temperature measurements between -200°C and + 850°C are possible with corrective calibration for the two possible PT100 4-wire classes being provided in driver software for OS-9 operation. A fast 16-bit delta-sigma A/D converter ensures an absolute accuracy of 0.2°C.

5.2.2 Specifications

Isolation	1.0 kV Optoisolated from the system
Input	4 PT100 groups 4-wire connection supported Open Sensor Detection Self Calibration Overvoltage Protection : ± 25V (power connected) : ± 35V (power disconnected) Temperature Range : -200°C to 850°C Temperature Drift : 1 LSB/°C (not compensated) Conversion Time : typ. 125 ms, max. 500 ms (depends on filter settings) Input Impedance : 10 MΩ (min) Resolution : 16-bit Absolute Accuracy : ± 0.2°C (sensor errors excluded) Linearization : by software
EEPROM	256-bit serial EEPROM for Calibration Data
Front Panel	2 Red LEDs
Power Cons.	350mW (min.), 450mW (max.)
Temperature Range	Standard (0°C to +70°C), Extended (-40°C to +85°C)
Module Weight	40g
ID Byte	$60, read by the SPI Interface

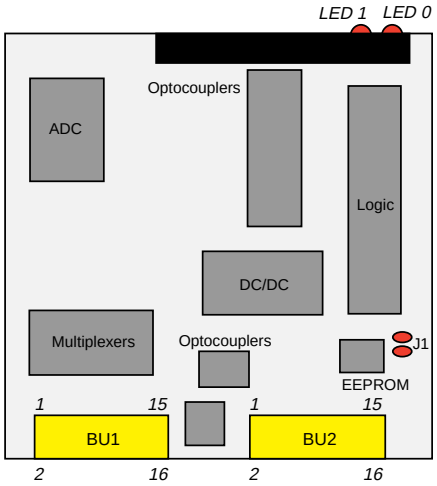
 © PEP Modular Computers GmbH

5.2.3 Front Panel Layout



5.2.4 Board Overview

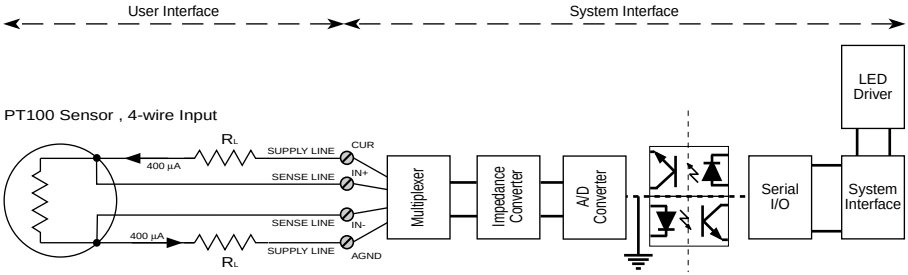
Component Side



© PEP Modular Computers GmbH

5.2.5 Functional Description

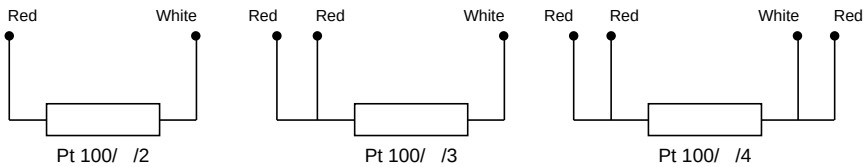
Figure 5.2.5.1: SM-PT100 Schematic Diagram



The SM-PT100 has provision for 4 sets of 4-wire PT100 sensors driven by an on-board generated 400µA constant current source. The differential analog input consists of a channel multiplexer, operational amplifier voltage feedback impedance converter and a high-resolution 16-bit delta-sigma converter. 1000V DC system isolation is provided by an optoisolated DC/DC converter while system flexibility is provided by a 256 bytes of serial EEPROM for ID number, production and board-specific calibration data. Two front-panel LEDs show the conversion/calibration activity.

The PT100 is a platinum resistance, which at 0ºC has a resistance of 100Ω and conforms to the international DIN IEC 751 standard for *Industrial Platinum Resistance Thermometer Sensors*. Two classes A and B exist under this standard with class B experiencing a slightly worse initial error.

Figure 5.2.5.2 Supported Sensor Connections



The two- and three-wire measurement methods shown in figure 5.2.5.2 are not compensated and will experience errors due to line-resistances. The best and recommended method is the 4-wire system shown in figure 5.2.5.1. which eliminates all line resistances automatically. The sense lines, being connected to a high impedance differential amplifier (10MΩ), introduce almost zero error resulting from line-resistances.

Table 5.2.5.2 shows the deviation from the required temperature for both class A and class B PT100 sensors.

Table 5.2.5.2 Tolerances between Class A and Class B Sensors

Temperature ℃	Tolerance			
	Class A		Class B	
	℃	Ω	℃	Ω
-200	± 0.55	± 0.24	± 1.3	± 0.56
-100	± 0.35	± 0.14	± 0.8	± 0.32
0	± 0.15	± 0.06	± 0.3	± 0.12
100	± 0.35	± 0.13	± 0.8	± 0.30
200	± 0.55	± 0.20	± 1.3	± 0.48
300	± 0.75	± 0.27	± 1.8	± 0.64
400	± 0.95	± 0.33	± 2.3	± 0.79
500	± 1.15	± 0.38	± 2.8	± 0.93
600	± 1.35	± 0.43	± 3.3	± 1.06
650	± 1.45	± 0.46	± 3.6	± 1.13
700	–	–	± 3.8	± 1.17
800	–	–	± 4.3	± 1.28
850	–	–	± 4.6	± 1.34

Driver software running under OS-9 supplies linearization data for use with DIN standard PT100 sensors. The on-board EEPROM contains board specific calibration data for self calibration purposes.

© PEP Modular Computers GmbH

5.2.6 Configuration

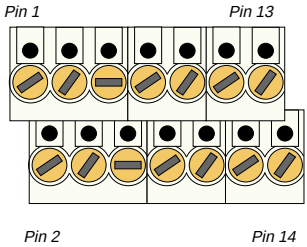
Jumper J1 - EEPROM Protection

Jumper	Settings	Description
J1	*set*	*EEPROM is not hardware write protected*
	open	EEPROM is hardware write protected

5.2.7 Pinouts

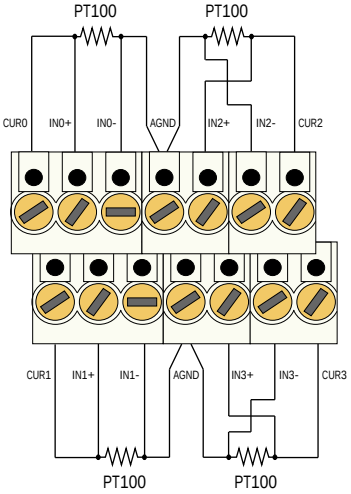
Screw Terminal Pinouts

The following shows the pinout/signal relationship for the SM-PT100 when connected to a particular screw terminal block.

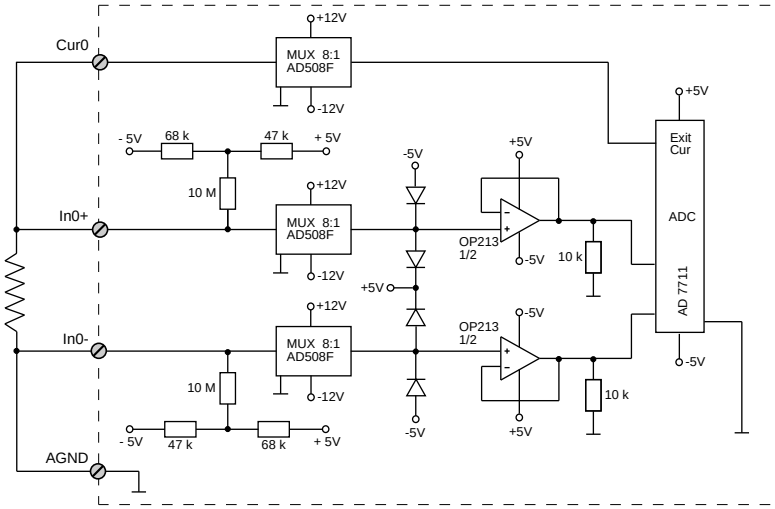


Pin Nr.	Signal	Description	Pin Nr.	Signal	Description
1	Cur 0	400µA source for sensor	2	Cur 1	400µA source for sensor
3	In 0+	Sense line 0 positive	4	In 1+	Sense line 1 positive
5	In 0 -	Sense line 0 negative	6	In 1 -	Sense line 1 negative
7	AGND	GND between 2 sensors	8	AGND	GND between 2 sensors
9	In 2+	Sense line 2 positive	10	In 3+	Sense line 3 positive
11	In 2 -	Sense line 2 negative	12	In 3 -	Sense line 3 negative
13	Cur 2	400µA source for sensor	14	Cur 3	400µA source for sensor

Connection



Input Circuit



© PEP Modular Computers GmbH

5.2.8 'C' Programming

5.2.8.1 SM-PT100 Library

The SM-PT100 library of functions *smartio.l* provide a convenient way of accessing the SM-PT100 module.

- All functions are written in ANSI C;
- Prototypes are to be found in the file *adclib.h*.

Hardware Requirements

- SMART2 Base Unit or SMART I/O Base and Extension Units;
- SM-PT100 Module.

Software Requirements

The compiler from one of the following:

- Ultra C Version 1.1.2 or higher;
- FasTrak version 2.0.2 or higher.

The leftmost SM-Module has number 0 assigned to it a far as programming is concerned although physically this is slot 1!

Before a library function can be used, the function *SMADCInit* must first be called. This allocates the requested resources. Furthermore, this function needs to be called for each SM-PT100 module called within the task. Upon completion of the application, the function *SMADCDeinit* needs to be called for each SM-PT100 module that has been initialized.

In order to illustrate the use of the SM-PT100 library, an application example called *demoadc.c* can be found in the SMART I/O application directory (normally found in */<device>/APPLIC/SMART/CMDS*).

To compile the source, execute the following commands at the shell prompt:

```
> chd /<dev>/APPLIC/SMART
> make all
```

To execute the example:

```
load /<dev>/BSP/SMART/SMADC/adc_task   * load HW task
load /<dev>/APPLIC/SMART/CMDS/demoadc  * load example
demoadc [slot]
```

Slot = 0,1 .. 10 specifies the port of the SM-PT100 module. (Default = 0)

Note that the procedure names are generic and are applicable to both the SM-PT100 and SM-THERM modules.

Initialization

To start the HW Task, the OS-9 module *adc_task* must be loaded, then the function *SMADCInit* must be called. To read data from a channel, the type of sensor attached to it must be known. The PEP header file *BSP/SMART/SMADC/DEFS/smadc.h* contains a list of sensors that are supported. To set the sensor type, the function *SMADCSetSensorType* needs to be called.

Read RAW Data

To force the HW task to read RAW data from a channel, the function *SMADCEnableRead* must be called. As soon as the HW task has received data from this channel, a flag is set indicating a valid RAW value. Now the user's application is able to read this value by using the function *SMADCReadRaw*.

 © PEP Modular Computers GmbH

Read Temperature Values

The HW task is prepared with a function that converts RAW values of a channel to degrees Celsius. To enable this facility, the function *SMADCEnableConversion* must be called after calling *SMADCEnableRead.* If the HW task has calculated these converted values, the function *SMADCReadConverted* can be used to get the value.

Calibration

Before the first value is fetched from the ADC, a calibration cycle is performed by the HW task. The user also has the facility to force the HW task to execute a calibration cycle by calling either *SMADCCalibrate* for one calibration or *SMADCSetCyclicCalib*. Then a calibration is executed if a user defined interval has expired.

Deinitialization:

If the task calling the SMADC library function finishes, the function *SMADCDeInit* should be called by this task to free memory and resources otherwise tied up with the application.

Other Functions

For more information on other functions, refer to the description of the functions later in this document or the example *demoadc.c*.

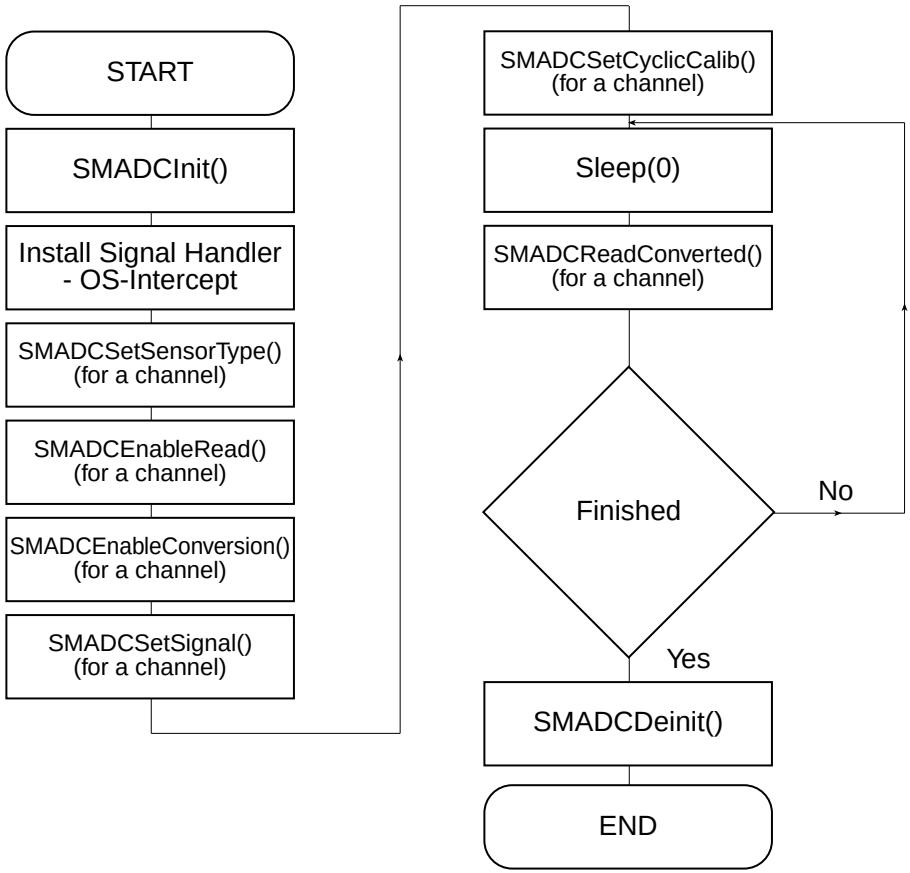
LEDs

Both LEDs on the SM-PT100 module are used to show the status of the HW task. LED1 is the upper one; LED2 is the lower one.

LED1	LED2	Status
off	off	ADC is not in use
on	off	Calibration in progress
off	on	AD conversion in progress
on	on	Standardization to reference value in progress

Figure 5.2.8.1. illustrates a typical programming structure for the correct operation of an SM-PT100 SMART-Module.

Figure 5.2.8.1 SM-PT100 Programming Flow Diagram



© PEP Modular Computers GmbH

5.2.8.2 SMADCInit

Syntax

```
error_code SMADCInit(u_int8 PortNr);
```

Description

This function initializes the SM-PT100 Module on port *PortNr*, and:
- identifies boards supported by this library
- provides an internal list of known sensors (contact PEP Modular Computers for advice on sensors not included in this list)
- creates or links the data module for communication with HW task (the module name is 'ADC_Data')
- starts the task accessing the Hardware (HW task)
- prepares parameters depending on the used SM-PT100 module.

Input

`u_int8 PortNr`	Port number of module to initialize (from 0 to 10)

Output

`error_code`	`SUCCESS`	
	`E_MNF`	If wrong type of module or no module is fitted on the selected port
	`E_PRCABT`	If starting of the HW task failed

or standard OS-9 error code (refer to the OS-9 Technical Manual Error Codes Section).

Example

```
RetVal = SMADCInit(0);
```

5.2.8.3 SMADCCalibrate

Syntax

```
error_code SMADCCalibrate(u_int8 PortNr);
```

Description

This function signals the HW task that a calibration cycle has to be started for the SM-PT100 at slot *PortNr*. It is performed automatically when the module is initialized and may be called at any time (if the sensor for example is moved to a different location with different temperature ranges). All 4 channels of the module will be calibrated simultaneously.

Input

`u_int8 PortNr`	Port number of module to calibrate (from 0 to 10)

Output

`error_code`	`SUCCESS`	
	`E_MNF`	If wrong type of module or no module is fitted on the selected port
	or standard OS-9 error code (refer to the OS-9 Technical Manual Error Codes Section).	

Example

```
RetVal = SMADCCalibrate(0);
```

 © PEP Modular Computers GmbH

5.2.8.4 SMADCSetCyclicCalib

Syntax

```
error_code SMADCSetCyclicCalib(u_int8 PortNr, u_int
                               secs);
```

Description

This function sets the time between two calibration cycles in seconds. If secs=0, then cyclic calibration is disabled.

Input

u_int8 PortNr	Port number of module for calibration (from 0 to 10)
u_int16 secs	Time in seconds between two calibration cycles

Output

error_code	SUCCESS	
	E_MNF	If wrong type of module or no module is fitted on the selected port
	or standard OS-9 error code (refer to the OS-9 Technical Manual Error Codes Section).	

Example

```
RetVal = SMADCSetCyclicCalib(0 , 120);
```


© PEP Modular Computers GmbH

5.2.8.5 SMADCSetSensorType

Syntax

```
error_code SMADCSetSensorType(u_int8 PortNr, u_int8
                    channel, u_int16 sensorCode);
```

Description

This function checks if an SM-PT100 board is fitted on the specified port, if the channel is valid, if the sensorCode is known and sets the gain, precision and mode to their default values for the type of sensor connected.

Input

`u_int8 PortNr`	Port number om module to set (from 0 to 10)
`u_int8 channel`	Number of specified channel (0 to 3)
`u_int16 sensorCode`	Code of the sensor (a list of PEP defined sensor types can be found in the file *linear.h*)

Output

`error_code`	`SUCCESS`	
	`E_MNF`	If wrong type of module or no module is fitted on the selected port
	`E_PARAM`	If wrong channel number is specified
	`E_BTYP`	If sensor type is not known

or standard OS-9 error code (refer to the OS-9 Technical Manual Error Codes Section).

Example

```
RetVal = SMADCSetSensorType(0, 1, PT100);
```

 © PEP Modular Computers GmbH

5.2.8.6 SMADCGetSensorType

Syntax

```
error_code SMADCGetSensorType(u_int8 PortNr, u_int8
                              channel, u_int16 *pBuffer);
```

Description

This function checks if an SM-PT100 board is fitted on the specified port and if the channel is valid. It gets the code of the sensor type of the specified channel and stores it in a user defined buffer. The user has to take care that there is enough place to store the data. A list of PEP supported sensor types can be found in the file *linear.h*.

Input

`u_int8 PortNr`	Port number from which to fetch sensor type (from 0 to 10)
`u_int8 channel`	Number of specified channel (0 to 3)
`u_int16*pBuffer`	Pointer to a user defined buffer

Output

`error_code`	`SUCCESS`	
	`E_MNF`	If wrong type of module or no module is fitted on the selected port
	`E_PARAM`	If wrong channel number is specified

or standard OS-9 error code (refer to the OS-9 Technical Manual Error Codes Section).

Example

```
RetVal = SMADCGetSensorType(0, 1, &Buffer);
```

5.2.8.7 SMADCSetPrecision

Syntax

```
error_code SMADCSetPrecision(u_int8 PortNr, u_int8
                          channel, u_int8 precision);
```

Description

This function checks if an SM-PT100 board is fitted on the specified port and if the channel is valid. It sets the precision of the AD Converter. The valid precision values are defined in the file smadc.h. Note that large precision/ filter values cause longer AD conversion times.

Input

`u_int8 PortNr`	Port number to set precision (from 0 to 10)
`u_int8 channel`	Number of specified channel (0 to 3)
`u_int8 precision`	Resolution of ADC; filter value

Output

`error_code`

`SUCCESS`	
`E_MNF`	If wrong type of module or no module is fitted on the selected port
`E_PARAM`	If wrong channel number is specified
`E_BMODE`	Illegal precision value (must be within 19 to 2000).

or standard OS-9 error code (refer to the OS-9 Technical Manual Error Codes Section).

Example

```
RetVal = SMADCSetPrecision(0, 1, 0x100);
```

 © PEP Modular Computers GmbH

5.2.8.8 SMADCGetPrecision

Syntax

```
error_code SMADCGetPrecision(u_int8 PortNr, u_int8
                          channel, u_int16 *pBuffer);
```

Description

This function checks if an SM-PT100 board is fitted on the specified port and if the channel is valid. It gets the precision/filter setting of the AD Converter and stores the value in a user defined buffer. The user has to take care that there is enough space to store the data.

Input

`u_int8 PortNr`	Port number from which to get data (from 0 to 10)
`u_int8 channel`	Number of specified channel (0 to 3)
`u_int16*pBuffer`	Pointer to a user defined buffer

Output

`error_code`

`SUCCESS`	
`E_MNF`	If wrong type of module or no module is fitted on the selected port
`E_PARAM`	If wrong channel number is specified

or standard OS-9 error code (refer to the OS-9 Technical Manual Error Codes Section).

Example

```
RetVal = SMADCGetPrecision(0, 1, &Buffer);
```

5.2.8.9 SMADCSetMode

Syntax

```
error_code SMADCSetMode(u_int8 PortNr, u_int8 channel,
                        u_int8 mode);
```

Description

This function checks if an SM-PT100 board is fitted on the specified port and if the channel is valid and sets the mode of the AD Converter. The valid values (BIPOLAR, UNIPOLAR) are defined in the file *smadc.h*.

Input

`u_int8 PortNr`	Port number to set mode (0 to 10)
`u_int8 channel`	Number of specified channel (0 to 3)
`u_int8 mode`	Mode of ADC: BIPOLAR or UNIPOLAR

Output

`error_code`	`SUCCESS`	
	`E_MNF`	If wrong type of module or no module is fitted on the selected port
	`E_PARAM`	If wrong channel number is specified
	`E_BMODE`	Illegal mode
	or standard OS-9 error code (refer to the OS-9 Technical Manual Error Codes Section).	

Example

```
RetVal = SMADCSetMode(0, 1, BIPOLAR);
```

 © PEP Modular Computers GmbH

5.2.8.10 SMADCGetMode

Syntax

```
error_code SMADCGetMode(u_int8 PortNr, u_int8 channel,
                        u_int8 *pBuffer);
```

Description

This function checks if an SM-PT100 board is fitted on the specified port and if the channel is valid. It gets the mode setting of the channel and stores the value in a user defined buffer. The user has to take care that the buffer is large enough to store the data.

Input

`u_int8 PortNr`	Port number to fetch the mode (0 to 10)
`u_int8 channel`	Number of specified channel (0 to 3)
`u_int8 *pBuffer`	Pointer to a user defined buffer

Output

`error_code`

`SUCCESS`	
`E_MNF`	If wrong type of module or no module is fitted on the selected port
`E_PARAM`	If wrong channel number is specified

or standard OS-9 error code (refer to the OS-9 Technical Manual Error Codes Section).

Example

```
RetVal = SMADCGetMode(0, 1, &Buffer);
```


© PEP Modular Computers GmbH

5.2.8.11 SMADCSetSignal

Syntax

```
error_code SMADCSetSignal(u_int8 PortNr, u_int8
channel, process_id procID, signal_code sigCode);
```

Description

This function defines the code of the signal that is sent by the HW task when the conversion of the channel is ready. The signal is sent to the process with the ID *procID*. If *sigCode* is set to 0, no signal will be sent (disable sending of signals).

Input

`u_int8 PortNr`	Port number of SM-PT100 to set the signal (from 0 to 10)
`u_int8 channel`	Number of specified channel (0 to 3)
`process_id procID`	ID of the process to send the signal to
`signal_code sigCode`	Signal code to be sent

Output

`error_code`	`SUCCESS`	
	`E_MNF`	If wrong type of module or no module is fitted on the selected port
	`E_PARAM`	If wrong channel number is specified
	`E_SIGNAL`	Illegal signal code used (it must be ⊕ 0x100)

or standard OS-9 error code (refer to the OS-9 Technical Manual Error Codes Section).

Example

```
RetVal = SMADCSetSignal(0, 1, 0, 0x100);
```

 © PEP Modular Computers GmbH

5.2.8.12 SMADCSetGain

Syntax

```
error_code SMADCSetGain(u_int8 PortNr, u_int8 channel,
                        u_int8 gain);
```

Description

This function checks if an SM-PT100 board is fitted on the specified port and if the channel is valid and sets the value of the gain amplifier. The valid values for gain are defined in the file *smadc.h*. The default values for the attached sensor are substituted here when the sensor type is selected!

Input

`u_int8 PortNr`	Port number to set gain (from 0 to 10)
`u_int8 channel`	Number of specified channel (0 to 3)
`u_int8 gain`	Value of gain amplifier (0 to 7 for gain factors from 1 to 128)

Output

`error_code`

`SUCCESS`	
`E_MNF`	If wrong type of module or no module is fitted on the selected port
`E_PARAM`	If wrong channel number is specified
`E_BMODE`	Illegal gain value

or standard OS-9 error code (refer to the OS-9 Technical Manual Error Codes Section).

Example

```
RetVal = SMADCSetGain(0, 1, 2);     {gain of 4}
```

5.2.8.13 SMADCGetGain

Syntax

```
error_code SMADCGetGain(u_int8 PortNr, u_int8 channel,
                        u_int8 *pBuffer);
```

Description

This function checks if an SM-PT100 board is fitted on the specified port and if the channel is valid. It gets the gain amplifier setting of the ADC and stores the value in a user defined buffer where the user has to take care that the buffer is large enough to store the data.

Input

`u_int8 PortNr`	Port number to fetch the gain setting (from 0 to 10)
`u_int8 channel`	Number of specified channel (0 to 3)
`u_int8 *pBuffer`	Pointer to an user defined buffer

Output

`error_code`

`SUCCESS`	
`E_MNF`	If wrong type of module or no module is fitted on the selected port
`E_PARAM`	If wrong channel number is specified

or standard OS-9 error code (refer to the OS-9 Technical Manual Error Codes Section).

Example

```
RetVal = SMADCGetGain(0, 1, &Buffer);
```

 © PEP Modular Computers GmbH

5.2.8.14 SMADCEnableRead

Syntax

```
error_code SMADCEnableRead(u_int8 PortNr, u_int8
                           channel, BOOLEAN enable);
```

Description

This function checks if an SM-PT100 board is fitted on the specified port and if the channel is valid. It forces the HW task to read raw values from the *enabled* channels of the device one after the other. The HW task writes the results to the data module. If the flag *enable* is TRUE (<>0), then cyclic ADConversion is executed, else (enable=FALSE) and cyclic conversion is disabled. The function invalidates RAW and Converted data within the data module.

Input

`u_int8 PortNr`	Port number to enable the reading function (from 0 to 10)
`u_int8 channel`	Number of specified channel (0 to 3)
`BOOLEAN enable`	Flag for enabling or disabling reading of the specified channel

Output

`error_code`

`SUCCESS`	
`E_MNF`	If wrong type of module or no module is fitted on the selected port
`E_PARAM`	If wrong channel number is specified

or standard OS-9 error code (refer to the OS-9 Technical Manual Error Codes Section).

Example

```
RetVal = SMADCEnableRead(0, 1, FALSE);
```

5.2.8.15 SMADCEnableConversion

Syntax

```
error_code SMADCEnableConversion(u_int8 PortNr, u_int8
                          channel, BOOLEAN enable);
```

Description

This function checks if an SM-PT100 board is fitted on the specified port and if the channel is valid and forces the HW task to convert the raw values. If the flag *enable* is TRUE (<>0), then conversion of raw values to degrees Celsius is executed, else if (*enable*=FALSE) no conversion takes place. The function invalidates converted data, if enable is set to FALSE.

Input

`u_int8 PortNr`	Port number to enable data conversion (from 0 to 10)
`u_int8 channel`	Number of specified channel (0 to 3)
`BOOLEAN enable`	Flag for enabling or disabling conversion of raw values

Output

`error_code`

`SUCCESS`	
`E_MNF`	If wrong type of module or no module is fitted on the selected port
`E_PARAM`	If wrong channel number is specified
`E_BMODE`	If conversion can't be enabled because a bad or no sensor type is specified

or standard OS-9 error code (refer to the OS-9 Technical Manual Error Codes Section).

Example

```
RetVal = SMADCEnableConversion(0, 1, FALSE);
```

 © PEP Modular Computers GmbH

5.2.8.16 SMADCReadRaw

Syntax

```
error_code SMADCReadRaw(u_int8 PortNr, u_int8 channel,
                        u_int16 *pBuffer);
```

Description

This function checks if an SM-PT100 board is fitted on the specified port and if the channel is valid. It reads the raw value of the last conversion and stores it in a user defined buffer. The user has to take care that the buffer is large enough to store the data.

Input

`u_int8 PortNr`	Port number from which raw data is to be read (from 0 to 10)
`u_int8 channel`	Number of the channel to be read (0 to 3)
`u_int16 *pBuffer`	Pointer to a user defined buffer

Output

`error_code`

`SUCCESS`	
`E_MNF`	If wrong type of module or no module is fitted on the selected port
`E_PARAM`	If wrong channel number is specified
`E_NOTRDY`	If no data is available at the moment - wait or poll the input

or standard OS-9 error code (refer to the OS-9 Technical Manual Error Codes Section).

Example

```
RetVal = SMADCReadRaw(0, 1, &Buffer);
```

5.2.8.17 SMADCReadConverted

Syntax

```
error_code SMADCReadConverted(u_int8 PortNr, u_int8
                              channel, double *pBuffer);
```

Description

This function reads the converted value of the last AD Conversion and stores the value in degrees Celcius in a user defined buffer. The user has to take care that the buffer is large enough to store the data.

Input

`u_int8 PortNr`	Port number from which the input is read (from 0 to 10)
`u_int8 channel`	Number of the channel to be read (0 to 3)
`double *pBuffer`	Pointer to a user defined buffer

Output

`error_code`	`SUCCESS`	
	`E_MNF`	If wrong type of module or no module is fitted on the selected port
	`E_PARAM`	If wrong channel number is specified
	`E_NOTRDY`	If no data is available at the moment or the sensor is defect or values are returned that cannot be converted - wait or poll the input or check sensor.
	or standard OS-9 error code (refer to the OS-9 Technical Manual Error Codes Section).	

Example

```
RetVal = SMADCReadConverted(0, 1, &Buffer);
```

 © PEP Modular Computers GmbH

5.2.8.18 SMADCDeinit

Syntax

```
error_code SMADCDeinit(u_int8 PortNr);
```

Description

This function frees the resources attached to this module if it was initialised.

Input

`u_int8 PortNr`	Number of the port to deinitialise

Output

`error_code`	`SUCCESS` or standard OS-9 error code (refer to the OS-9 Technical Manual Error Codes Section).

Example

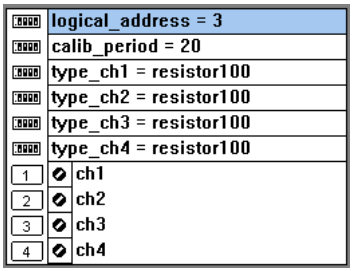
```
RetVal = SMADCDeinit(0);
```

5.2.9 ISaGRAF Programming

5.2.9.1 The ISaGRAF Board Parameters

Information on board parameters may be found in the PEP online help and ISaGRAF online help or user's manual.

Figure 5.2.9.1.1 a Typical Screen Section for the SM-PT100 Module



Logical Address

The 4 inputs of this module may be clearly seen together with the logical address at which this particular board is residing. Remember, this logical address is the same as the physical slot position! In the example shown here, the board being addressed is in slot 3 i.e. the third of the SMART-BASE slots. Up to 11 slots are catered for in the SMART I/O concept; 3 on the SMART-BASE and 2 for each connected SMART-EXT (up to a total of 4).

Module Specific Information :

`calib_period` : This is the time in seconds for cyclic calibration if required. An entry of zero disables the function.

`type_ch(x)` : This parameter is either `resistor100` or `PT100` depending whether the returned value should be raw ohmic or calibrated in degrees Celsius respectively.

5.2.9.2 The ISaGRAF Operate Calls

Operate calls are built into a program using ST or FBD languages when defining the project. A typical use could be at the initialization stage to check that the SMART-Modules are in fact located where they have been programmed to be. The syntax of the operate call is as follows:

Syntax

```
<return variable> := OPERATE(<source var>, COMMAND,
                             <source data>);
```

Here the `return variable` is assigned a value associated with the selected `COMMAND` parameter. Each SMART-Module possesses its own set of these `COMMANDS`.

Example

```
<error.code>:= OPERATE(<channel>, O_INIT_CODE, 0);
```

`O_INIT_CODE` is one of a number of distinct commands recognized by the PEP Modular Computers' implementation of board drivers and checks for example that the board is located where the program expects it to be.

`channel` provides channel specific information and in the example shown here, any of the 8 input channels may be used. The last parameter is not usually used by PEP implementations and is set to 0 (zero).

The `error.code` returns a value of zero if no error was detected, otherwise it returns a non-zero value depending on the error encountered. A list of these error codes may be found in the PEP online help.

 © PEP Modular Computers GmbH

A complete list of the operate COMMANDS may be obtained by selecting a project from the ISaGRAF ***projects*** group, opening an application and observing the ***Common defines*** in the ***Dictionary*** pull-down menu. Note that not all calls in the list may be used within the SMART2 environment however, the calls applicable to this module are :

O_INIT_CODE : The syntax and usage have already been explained.

O_ENABLE_CONV : This operate calls tells ISaGRAF to enable the conversion of a particular channel of a connected SM-PT100 module. The syntax is as follows:

```
<analog var> := OPERATE(<iovar>, O_ENABLE_CONV, 0);
```

where `<analog var>` represents the analog return variable and can be zero if the operate call is supported and the correct `<iovar>` is passed or non-zero if an error occurred.

`<iovar>` is the variable name for the channel where the operate call is to take effect.

O_DISABLE_CONV : This operate call tells ISaGRAF to disable the conversion of a particular channel of a connected SM-PT100 module. The syntax is as follows:

```
<analog var> := OPERATE(<iovar>, O_DISABLE_CONV, 0);
```

where `<analog var>` represents the analog return variable and can be zero if the operate call is supported and the correct `<iovar>` is passed or non-zero if an error occurred.

`<iovar>` is the variable name for the channel where the operate call is to take effect.

 © PEP Modular Computers GmbH

5.3 SM-THERM

5.3.1 Introduction

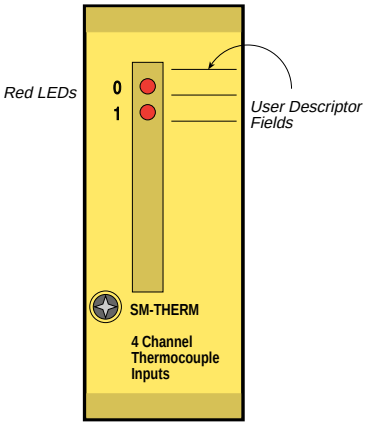
The SM-THERM supports up to four groups of thermoelement sensors together with a cold junction (ext. PT100) for accurate temperature measurement. The sensor types shown in the table below may be connected with corrective calibration for all sensor classes being provided by driver software running under OS-9. A fast 16-bit delta-sigma A/D converter ensures an absolute accuracy of 0.2°C.

5.3.2 Specifications

Isolation	1.0 kV Optoisolated from the system
Input	4 Thermocouple groups 1 PT100 for Cold Junction (2-wire) Open Sensor Detection Self Calibration Overvoltage Protection : ± 25V (power connected) : ± 35V (power disconnected) Temperature Range : Refer to table below Temperature Drift : 1 LSB/°C (not compensated) Conversion Time : typ. 125 ms, max. 500 ms (depends on filter settings) Input Impedance : 10 MΩ (min) Resolution : 16-bit Absolute Accuracy : ± 0.2°C (sensor errors excluded) Linearization : by software
EEPROM	256-bit serial EEPROM for Calibration Data
Front Panel	2 Red LEDs
Power Cons.	350mW (min.), 450mW (max.)
Temperature Range	Standard (0°C to +70°C), Extended (-40°C to +85°C)
Module Weight	40g
ID Byte	$61, read by the SPI Interface

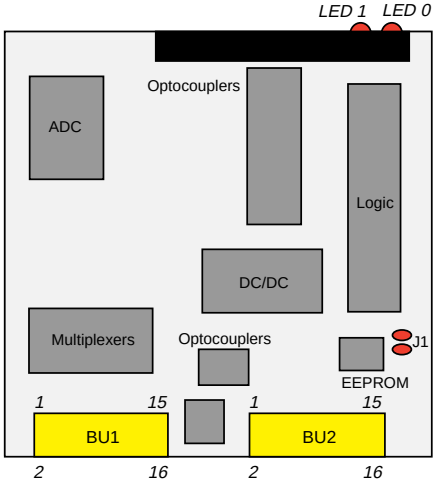
Sensor Category	Temperature (Low Range)	Temperature (High Range)	Sensor Category	Temperature (Low Range)	Temperature (High Range)
R	-50°C	1760°C	T	-270°C	400°C
S	-50°C	1760°C	E	-270°C	1000°C
B	0°C	1820°C	K	-270°C	1370°C
J	-210°C	1200°C			

5.3.3 Front Panel Layout



5.3.4 Board Overview

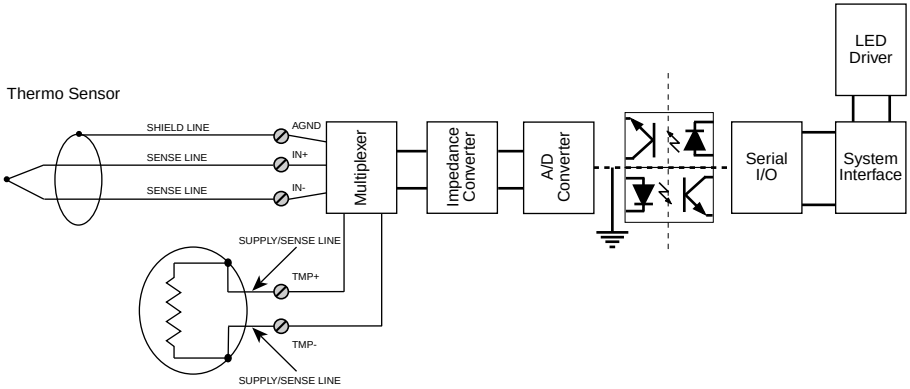
Component Side



© PEP Modular Computers GmbH

5.3.5 Functional Description

Figure 5.3.5.1: SM-THERM Schematic Diagram



External PT100 Sensor, 2-wire Cold Junction Input

The SM-THERM allows four sets of three-wire thermocouples to be connected together with one additional PT100 element used as a cold junction point reference. The differential analog input consists of a channel multiplexer, operational amplifier voltage feedback impedance converter and a high-resolution 16-bit delta-sigma converter. 1000V system isolation is provided by an optoisolated DC/DC converter while system flexibility is provided by 256-bytes of serial EEPROM for ID number, production and board specific calibration data. Two front-panel LEDs show the conversion/ calibration activity.

In principle any thermocouple conforming to the DIN43710 or IEC584 specifications can be connected. Sensor breakdown recognition is performed by the software which also caters for and corrects absolute temperature readings with a connected cold junction point (PT100).

5.3.6 Configuration

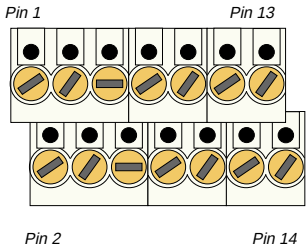
Jumper J1 - EEPROM Protection

Jumper	Settings	Description
J1	*set*	*EEPROM is not hardware write protected*
	open	EEPROM is hardware write protected

5.3.7 Pinouts

Screw Terminal Pinouts

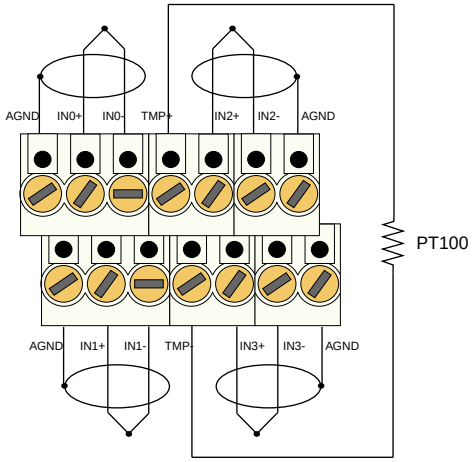
The following shows the pinout/signal relationship for the SM-PT100 when connected to a particular screw terminal block.



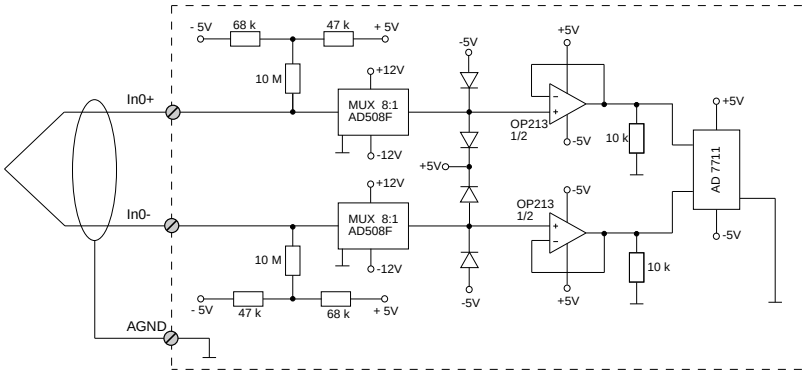
Pin Nr.	Signal	Description	Pin Nr.	Signal	Description
1	AGND	Analog Shield	2	AGND	Analog Shield
3	In 0+	Positive Sense Line (0)	4	In 1+	Positive Sense Line (1)
5	In 0 -	Negative Sense Line (0)	6	In 1 -	Negative Sense Line (1)
7	TMP + †	PT100 Cold Jn. supply	8	TMP - †	PT100 Cold Jn. supply
9	In 2+	Positive Sense Line (2)	10	In 3+	Positive Sense Line (3)
11	In 2 -	Negative Sense Line (2)	12	In 3 -	Negative Sense Line (3)
13	AGND	Analog Shield	14	AGND	Analog Shield

† Cold Junction point PT100 connection

Connection



Input Circuit



5.3.8 'C' Programming

5.3.8.1 SM-THERM Library

The SM-THERM library of functions *smartio.l* provide a convenient way of accessing the SM-THERM module.

- All functions are written in ANSI C;
- Prototypes are to be found in the file *adclib.h*.

Hardware Requirements

- SMART2 Base Unit or SMART I/O Base and Extension Units;
- SM-THERM Module.

Software Requirements

The compiler from one of the following:

- Ultra C Version 1.1.2 or higher;
- FasTrak version 2.0.2 or higher.

The leftmost SM-Module has number 0 assigned to it a far as programming is concerned although physically this is slot 1!

Before a library function can be used, the function *SMADCInit* must first be called. This allocates the requested resources. Furthermore, this function needs to be called for each SM-THERM module called within the task. Upon completion of the application, the function *SMADCDeInit* needs to be called for each SM-THERM module that has been initialized.

In order to illustrate the use of the SM-THERM library, an application example called *demoadc.c* can be found in the SMART2 application directory (normally found in */<device>/APPLIC/SMART/CMDS*).

 © PEP Modular Computers GmbH

To compile the source, execute the following commands at the shell prompt:

```
> chd /<dev>/APPLIC/SMART
> make all
```

To execute the example:

```
load /<dev>/BSP/SMART/SMADC/adc_task   * load HW task
load /<dev>/APPLIC/SMART/CMDS/demoadc * load example
demoadc [slot]
```

Slot = 0,1 .. 10 specifies the port of the SM-THERM module. (Default = 0)

Note that the procedure names are generic and are applicable to both the SM-PT100 and SM-THERM modules.

Initialization

To start the HW Task, the OS-9 module *adc_task* must be loaded, then the function *SMADCInit* must be called. To read data from a channel, the type of sensor attached to it must be known. The PEP header file *BSP/SMART/SMADC/DEFS/smadc.h* contains a list of sensors that are supported. To set the sensor type, the function *SMADCSetSensorType* needs to be called.

Read RAW Data

To force the HW task to read RAW data from a channel, the function *SMADCEnableRead* must be called. As soon as the HW task has received data from this channel, a flag is set indicating a valid RAW value. Now the user's application is able to read this value by using the function *SMADCReadRaw*.

Read Temperature Values

The HW task is prepared with a function that converts RAW values of a channel to degrees Celsius. To enable this facility, the function *SMADCEnableConversion* must be called after calling *SMADCEnableRead.* If the HW task has calculated these converted values, the function *SMADCReadConverted* can be used to get the value.

Calibration

Before the first value is fetched from the ADC, a calibration cycle is performed by the HW task. The user also has the facility to force the HW task to execute a calibration cycle by calling either *SMADCCalibrate* for one calibration or *SMADCSetCyclicCalib*. Then a calibration is executed if a user defined interval has expired.

Deinitialization

If the task calling the SMADC library function finishes, the function *SMADCDeInit* should be called by this task to free memory and resources otherwise tied up with the application.

Other Functions

For more information on other functions, refer to the description of the functions later in this document or the example *demoadc.c*.

LEDs

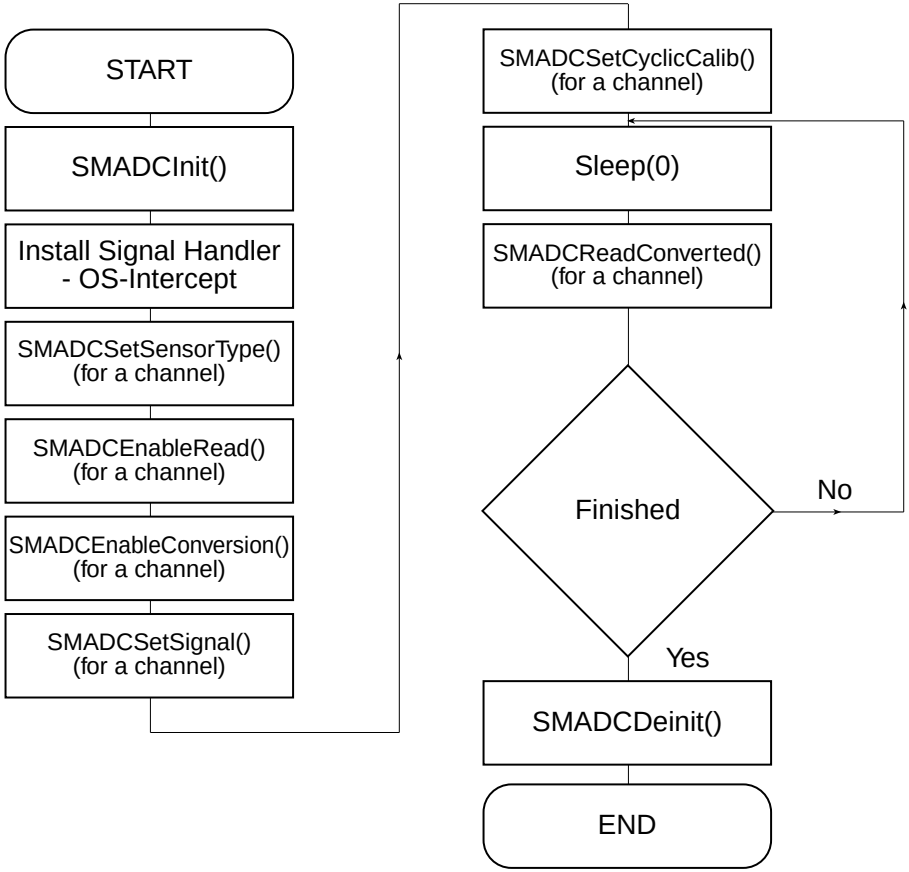
Both LEDs on the SM-THERM module are used to show the status of the HW task. LED1 is the upper one; LED2 is the lower one.

LED1	LED2	Status
off	off	ADC is not in use
on	off	Calibration in progress
off	on	AD conversion in progress
on	on	Standardisation to reference value in progress

 © PEP Modular Computers GmbH

Figure 5.3.8.1. illustrates a typical programming structure for the correct operation of an SM-THERM SMART-Module.

Figure 5.3.8.1 SM-THERM Programming Flow Diagram



5.3.8.2 SMADCInit

Syntax

```
error_code SMADCInit(u_int8 PortNr);
```

Description

This function initializes the SM-THERM Module on port *PortNr*, and:
- identifies boards supported by this library
- provides an internal list of known sensors (contact PEP Modular Computers for advice on sensors not included in this list)
- creates or links the data module for communication with HW task (the module name is 'ADC_Data')
- starts the task accessing the Hardware (HW task)
- prepares parameters depending on the used SM-THERM module.

Input

`u_int8 PortNr`	Port number of module to initialize (from 0 to 10)

Output

`error_code`	`SUCCESS`	
	`E_MNF`	If wrong type of module or no module is fitted on the selected port
	`E_PRCABT`	If starting of the HW task failed

or standard OS-9 error code (refer to the OS-9 Technical Manual Error Codes Section).

Example

```
RetVal = SMADCInit(0);
```


©1996 PEP Modular Computers GmbH

5.3.8.3 SMADCCalibrate

Syntax

```
error_code SMADCCalibrate(u_int8 PortNr);
```

Description

This function signals the HW task that a calibration cycle has to be started for the SM-THERM at slot *PortNr*. It is performed automatically when the module is initialised and may be called at any time (if the sensor for example is moved to a different location with different temperature ranges). All 4 channels of the module will be calibrated simultaneously.

Input

`u_int8 PortNr`	Port number of module to calibrate (from 0 to 10)

Output

`error_code`	`SUCCESS`	
	`E_MNF`	If wrong type of module or no module is fitted on the selected port

or standard OS-9 error code (refer to the OS-9 Technical Manual Error Codes Section).

Example

```
RetVal = SMADCCalibrate(0);
```

5.3.8.4 SMADCSetCyclicCalib

Syntax

```
error_code SMADCSetCyclicCalib(u_int8 PortNr,
                               u_int secs);
```

Description

This function sets the time between two calibration cycles in seconds. If secs=0, then cyclic calibration is disabled.

Input

`u_int8 PortNr`	Port number of module for calibration (from 0 to 10)
`u_int16 secs`	Time in seconds between two calibration cycles

Output

`error_code`	`SUCCESS`	
	`E_MNF`	If wrong type of module or no module is fitted on the selected port

or standard OS-9 error code (refer to the OS-9 Technical Manual Error Codes Section).

Example

```
RetVal = SMADCSetCyclicCalib(0 , 120);
```


© PEP Modular Computers GmbH

5.3.8.5 SMADCSetSensorType

Syntax

```
error_code SMADCSetSensorType(u_int8 PortNr, u_int8
                    channel, u_int16 sensorCode);
```

Description

This function checks if an SM-THERM board is fitted on the specified port, if the channel is valid, if the *sensorCode* is known and sets the gain, precision and mode to their default values for the type of sensor connected.

Input

`u_int8 PortNr`	Port number om module to set (from 0 to 10)
`u_int8 channel`	Number of specified channel (0 to 3)
`u_int16 sensorCode`	Code of the sensor (a list of PEP defined sensor types can be found in the file *linear.h*)

Output

`error_code`	`SUCCESS`	
	`E_MNF`	If wrong type of module or no module is fitted on the selected port
	`E_PARAM`	If wrong channel number is specified
	`E_BTYP`	If sensor type is not known

or standard OS-9 error code (refer to the OS-9 Technical Manual Error Codes Section).

Example

```
RetVal = SMADCSetSensorType(0, 1, B_TYPE);
```

5.3.8.6 SMADCGetSensorType

Syntax

```
error_code SMADCGetSensorType(u_int8 PortNr, u_int8
                              channel, u_int16 *pBuffer);
```

Description

This function checks if an SM-THERM board is fitted on the specified port and if the channel is valid. It gets the code of the sensor type of the specified channel and stores it in a user defined buffer. The user has to take care that there is enough place to store the data. A list of PEP supported sensor types can be found in the file *linear.h*.

Input

`u_int8 PortNr`	Port number from which to fetch sensor type (from 0 to 10)
`u_int8 channel`	Number of specified channel (0 to 3)
`u_int16*pBuffer`	Pointer to a user defined buffer

Output

`error_code`	`SUCCESS`	
	`E_MNF`	If wrong type of module or no module is fitted on the selected port
	`E_PARAM`	If wrong channel number is specified

or standard OS-9 error code (refer to the OS-9 Technical Manual Error Codes Section).

Example

```
RetVal = SMADCGetSensorType(0, 1, &Buffer);
```

 © PEP Modular Computers GmbH

5.3.8.7 SMADCSetPrecision

Syntax

```
error_code SMADCSetPrecision(u_int8 PortNr, u_int8
                          channel, u_int8 precision);
```

Description

This function checks if an SM-THERM board is fitted on the specified port and if the channel is valid. It sets the precision of the AD Converter. The valid precision values are defined in the file *smadc.h*. Note that large precision/filter values cause longer AD conversion times.

Input

`u_int8 PortNr`	Port number to set precision (from 0 to 10)
`u_int8 channel`	Number of specified channel (0 to 3)
`u_int8 precision`	Resolution of ADC; filter value

Output

`error_code`

`SUCCESS`	
`E_MNF`	If wrong type of module or no module is fitted on the selected port
`E_PARAM`	If wrong channel number is specified
`E_BMODE`	Illegal precision value (must be within 19 to 2000).

or standard OS-9 error code (refer to the OS-9 Technical Manual Error Codes Section).

Example

```
RetVal = SMADCSetPrecision(0, 1, 0x100);
```

5.3.8.8 SMADCGetPrecision

Syntax

```
error_code SMADCGetPrecision(u_int8 PortNr, u_int8
                          channel, u_int16 *pBuffer);
```

Description

This function checks if an SM-THERM board is fitted on the specified port and if the channel is valid. It gets the precision/filter setting of the AD Converter and stores the value in a user defined buffer. The user has to take care that there is enough space to store the data.

Input

`u_int8 PortNr`	Port number from which to get data (from 0 to 10)
`u_int8 channel`	Number of specified channel (0 to 3)
`u_int16*pBuffer`	Pointer to a user defined buffer

Output

`error_code`

`SUCCESS`	
`E_MNF`	If wrong type of module or no module is fitted on the selected port
`E_PARAM`	If wrong channel number is specified

or standard OS-9 error code (refer to the OS-9 Technical Manual Error Codes Section).

Example

```
RetVal = SMADCGetPrecision(0, 1, &Buffer);
```

 © PEP Modular Computers GmbH

5.3.8.9 SMADCSetMode

Syntax

```
error_code SMADCSetMode(u_int8 PortNr, u_int8 channel,
                        u_int8 mode);
```

Description

This function checks if an SM-THERM board is fitted on the specified port and if the channel is valid and sets the mode of the AD Converter. The valid values (BIPOLAR, UNIPOLAR) are defined in the file *smadc.h*.

Input

`u_int8 PortNr`	Port number to set mode (0 to 10)
`u_int8 channel`	Number of specified channel (0 to 3)
`u_int8 mode`	Mode of ADC: BIPOLAR or UNIPOLAR

Output

`error_code`	`SUCCESS`	
	`E_MNF`	If wrong type of module or no module is fitted on the selected port
	`E_PARAM`	If wrong channel number is specified
	`E_BMODE`	Illegal mode

or standard OS-9 error code (refer to the OS-9 Technical Manual Error Codes Section).

Example

```
RetVal = SMADCSetMode(0, 1, BIPOLAR);
```

5.3.8.10 SMADCGetMode

Syntax

```
error_code SMADCGetMode(u_int8 PortNr, u_int8 channel,
                        u_int8 *pBuffer);
```

Description

This function checks if an SM-THERM board is fitted on the specified port and if the channel is valid. It gets the mode setting of the channel and stores the value in a user defined buffer. The user has to take care that the buffer is large enough to store the data.

Input

`u_int8 PortNr`	Port number to fetch the mode (0 to 10)
`u_int8 channel`	Number of specified channel (0 to 3)
`u_int8 *pBuffer`	Pointer to a user defined buffer

Output

`error_code`	`SUCCESS`	
	`E_MNF`	If wrong type of module or no module is fitted on the selected port
	`E_PARAM`	If wrong channel number is specified

or standard OS-9 error code (refer to the OS-9 Technical Manual Error Codes Section).

Example

```
RetVal = SMADCGetMode(0, 1, &Buffer);
```

 © PEP Modular Computers GmbH

5.3.8.11 SMADCSetSignal

Syntax

```
error_code SMADCSetSignal(u_int8 PortNr, u_int8
channel, process_id procID, signal_code sigCode);
```

Description

This function defines the code of the signal that is sent by the HW task when the conversion of the channel is ready. The signal is sent to the process with the ID *procID*. If *sigCode* is set to 0, no signal will be sent (disable sending of signals).

Input

`u_int8 PortNr`	Port number of SM-THERM to set the signal (from 0 to 10)
`u_int8 channel`	Number of specified channel (0 to 3)
`process_id procID`	ID of the process to send the signal to
`signal_code sigCode`	Signal code to be sent

Output

`error_code`

`SUCCESS`	
`E_MNF`	If wrong type of module or no module is fitted on the selected port
`E_PARAM`	If wrong channel number is specified
`E_SIGNAL`	Illegal signal code used (it must be ⊕ 0x100)

or standard OS-9 error code (refer to the OS-9 Technical Manual Error Codes Section).

Example

```
RetVal = SMADCSetSignal(0, 1, 0, 0x100);
```

5.3.8.12 SMADCSetGain

Syntax

```
error_code SMADCSetGain(u_int8 PortNr, u_int8 channel,
                        u_int8 gain);
```

Description

This function checks if an SM-THERM board is fitted on the specified port and if the channel is valid and sets the value of the gain amplifier. The valid values for gain are defined in the file *smadc.h*. The default values for the attached sensor are substituted here when the sensor type is selected!

Input

`u_int8 PortNr`	Port number to set gain (from 0 to 10)
`u_int8 channel`	Number of specified channel (0 to 3)
`u_int8 gain`	Value of gain amplifier (0 to 7 for gain factors from 1 to 128)

Output

`error_code`

`SUCCESS`	
`E_MNF`	If wrong type of module or no module is fitted on the selected port
`E_PARAM`	If wrong channel number is specified
`E_BMODE`	Illegal gain value

or standard OS-9 error code (refer to the OS-9 Technical Manual Error Codes Section).

Example

```
RetVal = SMADCSetGain(0, 1, 2);     {gain of 4}
```

 © PEP Modular Computers GmbH

5.3.8.13 SMADCGetGain

Syntax

```
error_code SMADCGetGain(u_int8 PortNr, u_int8 channel,
                        u_int8 *pBuffer);
```

Description

This function checks if an SM-THERM board is fitted on the specified port and if the channel is valid. It gets the gain amplifier setting of the ADC and stores the value in a user defined buffer where the user has to take care that the buffer is large enough to store the data.

Input

`u_int8 PortNr`	Port number to fetch the gain setting (from 0 to 10)
`u_int8 channel`	Number of specified channel (0 to 3)
`u_int8 *pBuffer`	Pointer to an user defined buffer

Output

`error_code`

`SUCCESS`	
`E_MNF`	If wrong type of module or no module is fitted on the selected port
`E_PARAM`	If wrong channel number is specified

or standard OS-9 error code (refer to the OS-9 Technical Manual Error Codes Section).

Example

```
RetVal = SMADCGetGain(0, 1, &Buffer);
```

5.3.8.14 SMADCEnableRead

Syntax

```
error_code SMADCEnableRead(u_int8 PortNr, u_int8
                           channel, BOOLEAN enable);
```

Description

This function checks if an SM-THERM board is fitted on the specified port and if the channel is valid. It forces the HW task to read raw values from the *enabled* channels of the device one after the other. The HW task writes the results to the data module. If the flag *enable* is TRUE (<>0), then cyclic ADConversion is executed, else (enable=FALSE) and cyclic conversion is disabled. The function invalidates RAW and Converted data within the data module.

Input

`u_int8 PortNr`	Port number to enable the reading function (from 0 to 10)
`u_int8 channel`	Number of specified channel (0 to 3)
`BOOLEAN enable`	Flag for enabling or disabling reading of the specified channel

Output

`error_code`

`SUCCESS`	
`E_MNF`	If wrong type of module or no module is fitted on the selected port
`E_PARAM`	If wrong channel number is specified

or standard OS-9 error code (refer to the OS-9 Technical Manual Error Codes Section).

Example

```
RetVal = SMADCEnableRead(0, 1, FALSE);
```

 © PEP Modular Computers GmbH

5.3.8.15 SMADCEnableConversion

Syntax

```
error_code SMADCEnableConversion(u_int8 PortNr, u_int8
                          channel, BOOLEAN enable);
```

Description

This function checks if an SM-THERM board is fitted on the specified port and if the channel is valid and forces the HW task to convert the raw values. If the flag *enable* is TRUE (<>0), then conversion of raw values to degrees Celsius is executed, else if (*enable*=FALSE) no conversion takes place. The function invalidates converted data, if enable is set to FALSE.

Input

`u_int8 PortNr`	Port number to enable data conversion (from 0 to 10)
`u_int8 channel`	Number of specified channel (0 to 3)
`BOOLEAN enable`	Flag for enabling or disabling conversion of raw values

Output

`error_code`

`SUCCESS`	
`E_MNF`	If wrong type of module or no module is fitted on the selected port
`E_PARAM`	If wrong channel number is specified
`E_BMODE`	If conversion can't be enabled because a bad or no sensor type is specified

or standard OS-9 error code (refer to the OS-9 Technical Manual Error Codes Section).

Example

```
RetVal = SMADCEnableConversion(0, 1, FALSE);
```

5.3.8.16 SMADCReadRaw

Syntax

```
error_code SMADCReadRaw(u_int8 PortNr, u_int8 channel,
                        u_int16 *pBuffer);
```

Description

This function checks if an SM-THERM board is fitted on the specified port and if the channel is valid. It reads the raw value of the last conversion and stores it in a user defined buffer. The user has to take care that the buffer is large enough to store the data.

Input

`u_int8 PortNr`	Port number from which raw data is to be read (from 0 to 10)
`u_int8 channel`	Number of the channel to be read (0 to 3)
`u_int16 *pBuffer`	Pointer to a user defined buffer

Output

`error_code`	`SUCCESS`	
	`E_MNF`	If wrong type of module or no module is fitted on the selected port
	`E_PARAM`	If wrong channel number is specified
	`E_NOTRDY`	If no data is available at the moment - wait or poll the input

or standard OS-9 error code (refer to the OS-9 Technical Manual Error Codes Section).

Example

```
RetVal = SMADCReadRaw(0, 1, &Buffer);
```

 © PEP Modular Computers GmbH

5.3.8.17 SMADCReadConverted

Syntax

```
error_code SMADCReadConverted(u_int8 PortNr, u_int8
                              channel, double *pBuffer);
```

Description

This function reads the converted value of the last AD Conversion and stores the value in degrees Celcius in a user defined buffer. The user has to take care that the buffer is large enough to store the data.

Input

`u_int8 PortNr`	Port number from which the input is read (from 0 to 10)
`u_int8 channel`	Number of the channel to be read (0 to 3)
`double *pBuffer`	Pointer to a user defined buffer

Output

`error_code`	`SUCCESS`	
	`E_MNF`	If wrong type of module or no module is fitted on the selected port
	`E_PARAM`	If wrong channel number is specified
	`E_NOTRDY`	If no data is available at the moment or the sensor is defect. - wait or poll the input or check sensor
	or standard OS-9 error code (refer to the OS-9 Technical Manual Error Codes Section).	

Example

```
RetVal = SMADCReadConverted(0, 1, &Buffer);
```

5.3.8.18 SMADCDeinit

Syntax

```
error_code SMADCDeinit(u_int8 PortNr);
```

Description

This function frees the resources attached to this module if it was initialized.

Input

`u_int8 PortNr`	Number of the port to de-initialize

Output

`error_code`	`SUCCESS` or standard OS-9 error code (refer to the OS-9 Technical Manual Error Codes Section).

Example

```
RetVal = SMADCDeinit(0);
```

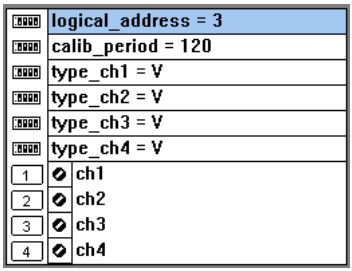

© PEP Modular Computers GmbH

5.3.9 ISaGRAF Programming

5.3.9.1 The ISaGRAF Board Parameters

Information on board parameters may be found in the PEP online help and ISaGRAF online help or user's manual.

Figure 5.3.9.1.1 a Typical Screen Section for the SM-THERM Module



Logical Address

The 4 inputs of this module may be clearly seen together with the logical address at which this particular board is residing. Remember, this logical address is the same as the physical slot position! In the example shown here, the board being addressed is in slot 3 i.e. the third of the SMART-BASE slots. Up to 11 slots are catered for in the SMART I/O concept; 3 on the SMART-BASE and 2 for each connected SMART-EXT (up to a total of 4).

Module Specific Information

`calib_period:` This is the time in seconds for cyclic calibration if required. An entry of zero disables the function.

`type_ch(x):` This parameter is set depending on the sensor type or may be 'V' if the measured values are required in microvolts with a module in bipolar mode.

5.3.9.2 The ISaGRAF Operate Calls

Operate calls are built into a program using ST or FBD languages when defining the project. A typical use could be at the initialization stage to check that the SMART-Modules are in fact located where they have been programmed to be. The syntax of the operate call is as follows:

Syntax:

```
<return variable> := OPERATE(<source var>, COMMAND,
                             <source data>);
```

Here the `return variable` is assigned a value associated with the selected `COMMAND` parameter. Each SMART-Module possesses its own set of these `COMMANDS`.

Example:

```
<error.code>:= OPERATE(<channel>, O_INIT_CODE, 0);
```

`O_INIT_CODE` is one of a number of distinct commands recognized by the PEP Modular Computers' implementation of board drivers and checks for example that the board is located where the program expects it to be.

`channel` provides channel specific information and in the example shown here, any of the 8 input channels may be used. The last parameter is not usually used by PEP implementations and is set to 0 (zero).

The `error.code` returns a value of zero if no error was detected, otherwise it returns a non-zero value depending on the error encountered. A list of these error codes may be found in the PEP online help.

 © PEP Modular Computers GmbH

A complete list of the operate `COMMANDS` may be obtained by selecting a project from the ISaGRAF ***projects*** group, opening an application and observing the ***Common defines*** in the ***Dictionary*** pull-down menu. Note that not all calls in the list may be used within the SMART2 environment however, the calls applicable to this module are :

`O_INIT_CODE` : The syntax and usage have already been explained.

`O_ENABLE_CONV` : This operate calls tells ISaGRAF to enable the conversion of a particular channel of a connected SM-THERM module. The syntax is as follows:

```
<analog var> := OPERATE(<iovar>, O_ENABLE_CONV, 0);
```

where `<analog var>` represents the analog return variable and can be zero if the operate call is supported and the correct `<iovar>` is passed or non-zero if an error occurred.

`<iovar>` is the variable name for the channel where the operate call is to take effect.

`O_DISABLE_CONV` : This operate call tells ISaGRAF to disable the conversion of a particular channel of a connected SM-THERM module. The syntax is as follows:

```
<analog var> := OPERATE(<iovar>, O_DISABLE_CONV, 0);
```

where `<analog var>` represents the analog return variable and can be zero if the operate call is supported and the correct `<iovar>` is passed or non-zero if an error occurred.

`<iovar>` is the variable name for the channel where the operate call is to take effect.

 © PEP Modular Computers GmbH

This page has been left blank intentionally.

© PEP Modular Computers GmbH

5.4 SM-ADC1

5.4.1 Introduction

The SM-ADC1 provides a fast, 12-bit, bipolar (± 10V DC), 6-channel analog to digital converter. Two configurable red LEDs on the front panel form the user interface while an on-board EEPROM stores unique calibration data required by the signal converter.

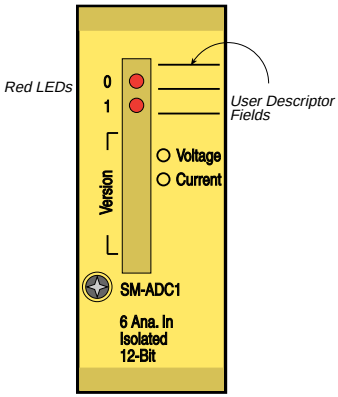
A version of the SM-ADC1 for current input (0 .. 20mA) is also available.

5.4.2 Specifications

Isolation	1.0 kV Optoisolated from the system
Input	6 Differential Inputs Input Range ± 10V (0 .. 20mA) 12-bit ADC with 1-bit non-linearity 10-bit Repeating Accuracy Overvoltage Protection ± 35V Input Impedance approx. 440kΩ (124Ω) 10kHz Filter On-Chip Ref. Voltage 13µs Conversion Time @ 1MHz SCLK †
EEPROM	256 Byte serial EEPROM for Calibration Data
Front Panel	2 Red LEDs (User Configurable)
Power Cons.	350mW (min.), 450mW (max.)
Temperature Range	Standard (0°C to +70°C), Ext./Storage (-40°C to +85°C)
Module Weight	70g
ID Byte	$62, read by the SPI Interface

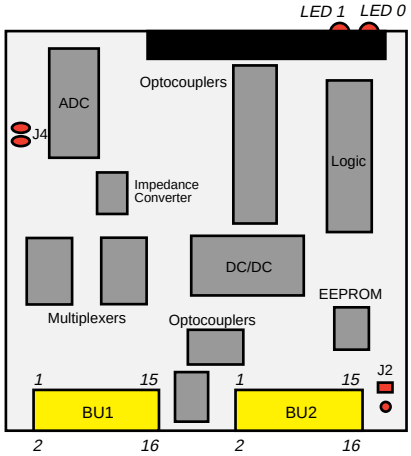
† The conversion time applies per channel

5.4.3 Front Panel Layout



5.4.4 Board Overview

Component Side



© PEP Modular Computers GmbH

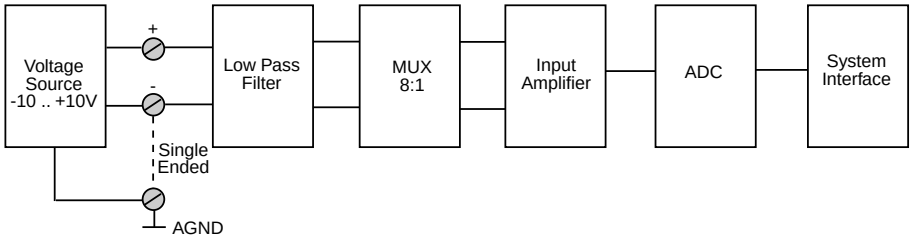
5.4.5 Functional Description

The SM-ADC1 has 6 differential voltage or current inputs depending on the ordered type. An on-board EEPROM contains board specific calibration data, module ID byte, sub ID byte and production data. A DC/DC converter generates the 5V to ±15V power requirements for the ADC converter and provides additional system isolation.

5.4.5.1 Input Circuitry

The input circuitry of the SM-ADC1 consists of an R-C first-order, low-pass filter to prevent noise and signals greater than 10kHz from entering the system, a 1:6 channel multiplexer, followed by a differential amplifier, divider and a 12-bit fast analog to digital converter (ADC).

Figure 5.4.5.1: SM-ADC1 Input Shematic Diagram



The ADC, using successive approximation and input track and hold is referenced by its own on-chip voltage source supplying 4.096V. To cater for bipolar full-scale inputs, this reference source is split (-2.048V .. +2.048V).

The incoming voltage signals are multiplexed and reduced via a 1:5 divider network so that the ADC is supplied with a scalable input voltage range.

5.4.6 Configuration

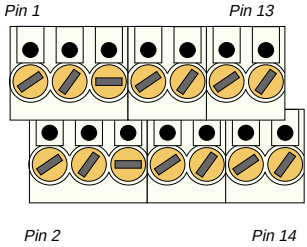
Although the SM-ADC1 has one solder jumper (J2) for EEPROM configuration, the user should not interfere with it.

Jumper J4 likewise should not be interfered with as it is factory set at the time of ordering and controls the unipolar/bipolar mode of ADC operation .

5.4.7 Pinouts

Screw Terminal Pinouts

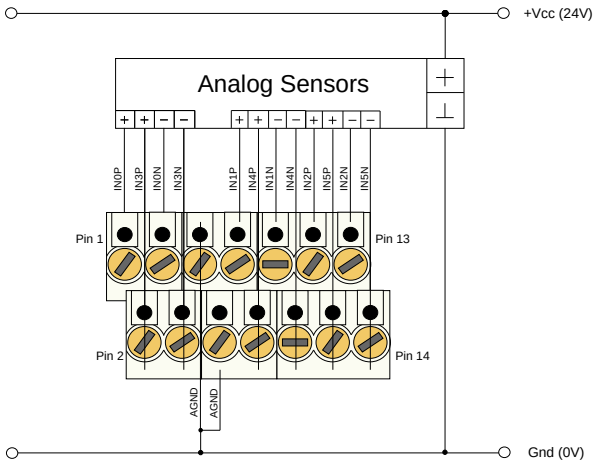
The following shows the pinout/signal relationship for the SM-ADC1 when connected to a particular screw terminal block.



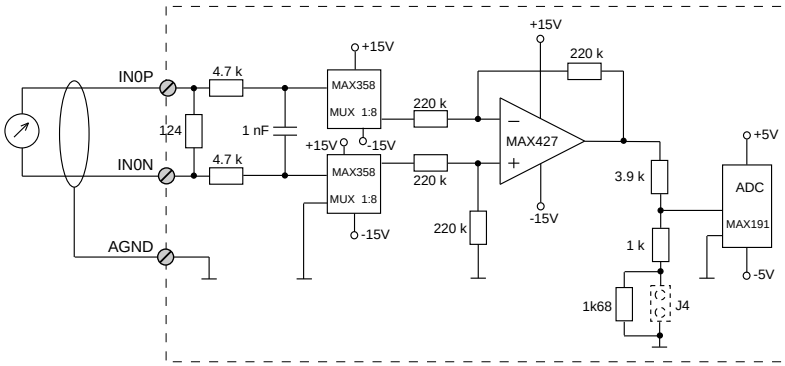
Pin Nr.	Signal	Description	Pin Nr.	Signal	Description
1	IN0+	Analog Input Ch. 0+	2	IN3+	Analog Input Ch. 3+
3	IN0-	Analog Input Ch. 0-	4	IN3-	Analog Input Ch. 3-
5	AGND	Analog GND	6	AGND	Analog GND
7	IN1+	Analog Input Ch. 1+	8	IN4+	Analog Input Ch. 4+
9	IN1-	Analog Input Ch. 1-	10	IN4-	Analog Input Ch. 4-
11	IN2+	Analog Input Ch. 2+	12	IN5+	Analog Input Ch. 5+
13	IN2-	Analog Input Ch. 2-	14	IN5-	Analog Input Ch. 5-

© Modular Computers GmbH

Connection:



Input Circuit:



5.4.8 'C' Programming

5.4.8.1 SM-ADC1 Library

The SM-ADC1 library of functions *smartio.l* provide a convenient way of accessing the SM-ADC1 module.

- All functions are written in ANSI C;
- Prototypes are to be found in the file *adc1lib.h*.

Hardware Requirements

- SMART2 Base Unit or SMART I/O Base and Extension Units;
- SM-ADC1 Module.

Software Requirements

The compiler from one of the following:

- Ultra C Version 1.1.2 or higher;
- FasTrak 2.0.2 or higher.

The leftmost SM-Module has number 0 assigned to it a far as programming is concerned although physically this is slot 1!

Before a library function can be used, the function *SMADC1Init* must first be called. This allocates the requested resources. Furthermore, this function needs to be called for each SM-ADC1 Module called within the task. Upon completion of the application, the function *SMADC1DeInit* needs to be called for each SM-ADC1 Module that has been initialised.

In order to illustrate the use of the SM-ADC1 library, an application example called *demoadc1.c* can be found in the SMART I/O application directory (normally found in */<device>/APPLIC/SMART/CMDS*).

5.4.8.2 SMADC1Init

Syntax

```
error_code SMADC1Init(u_int8 PortNr);
```

Description

This function initialises the SM-ADC1 Module on port *PortNr*, and:
- tests the module's ID
- test the module's sub-ID for the determination of bipolar or unipolar operation (voltage/current input)
- reads the conversion table held in EEPROM
- allocates space for the fetched resources

Input

`u_int8 PortNr`	Port number of module to initialise (from 0 to 10)

Output

`error_code`	`SUCCESS` or	
	`E_BMODE`	if wrong or no module
	`E_MEMFUL`	if no memory resources available

or standard OS-9 error code (refer to the OS-9 Technical Manual Error Codes Section).

Example

```
RetVal = SMADC1Init(0);
```

5.4.8.3 SMADC1GetVRaw

Syntax

```
error_code SMADC1GetVRaw(u_int8 PortNr, u_int8
                         Channel, int16 *pValue);
```

Description

This function fetches the conversion result of a selected channel without performing a gain or offset correction. The result is always a signed integer value.

Input

`u_int8 PortNr`	SM-Port to be accessed
`u_int8 Channel`	Number of the ADC input channel to access (0-3)
`int16 *pValue`	Pointer to the location where the conversion result will be stored

Output

`error_code`	`SUCCESS`	
	`E_SEEK`	if module was not initialised
	`E-BMODE`	if the module is unknown

or standard OS-9 error code (refer to the OS-9 Technical Manual Error Codes Section).

Example

```
RetVal = SMADC1GetVRaw(0, 2, 0x07FF);
```

Value (16-bit HEX)			
Unipolar		**Bipolar**	
Positive Full-Scale	$0FFF	Positive Full-Scale	$07FF
+1LSB	$0001	+1LSB	$0001
0	$0000	-1LSB	$FFFF
		Negative Full-Scale	$F800

© PEP Modular Computers GmbH

5.4.8.4 SMADC1GetV

Syntax

```
error_code SMADC1GetV(u_int8 PortNr, u_int8 Channel,
                      int16 *pValue);
```

Description

This function fetches the conversion result of a selected channel. A gain and offset correction is performed and the result stored as a signed integer value.

Input

`u_int8 PortNr`	SM-Port to be accessed
`u_int8 Channel`	Number of the ADC input channel to access (0-3)
`int16 *pValue`	Pointer to the location where the conversion result will be stored

Output

`error_code`	`SUCCESS`	
	`E_SEEK`	if module was not initialised
	`E-BMODE`	if the module is unknown

or standard OS-9 error code (refer to the OS-9 Technical Manual Error Codes Section).

Example

```
RetVal = SMADC1GetV(0, 2, 0x07FF);
```

Value (16-bit HEX)			
Unipolar		**Bipolar**	
Positive Full-Scale	$0FFF	Positive Full-Scale	$07FF
+1LSB	$0001	+1LSB	$0001
0	$0000	0	$0000
		-1LSB	$FFFF
		Negative Full-Scale	$F800

© PEP Modular Computers GmbH

5.4.8.5 SMADC1SetLed

Syntax

```
error_code SMADC1SetLed(u_int8 PortNr, u_int8 LedNr);
```

Description

This function switches on one of the two LEDs fitted to the SM-ADC1.

Input

`u_int8 PortNr`	SM-Port number to be accessed
`u_int8 LedNr`	Number of the LED to switch on (1-2)

Output

`error_code`	`SUCCESS`	
	`E_SEEK`	if module was not initialised
	`E-BMODE`	if the module is unknown
	or standard OS-9 error code (refer to the OS-9 Technical Manual Error Codes Section).	

Example

```
RetVal = SMADC1SetLed(0, 2);
```


© PEP Modular Computers GmbH

5.4.8.6 SMADC1ClrLed

Syntax

```
error_code SMADC1ClrLed(u_int8 PortNr, u_int8 LedNr);
```

Description

This function switches off one of the two LEDs fitted to the SM-ADC1.

Input

`u_int8 PortNr`	SM-Port number to be accessed
`u_int8 LedNr`	Number of the LED to switch off (1-2)

Output

`error_code`	`SUCCESS`	
	`E_SEEK`	if module was not initialised
	`E-BMODE`	if the module is unknown

or standard OS-9 error code (refer to the OS-9 Technical Manual Error Codes Section).

Example

```
RetVal = SMADC1ClrLed(0, 2);
```

5.4.8.7 SMADC1DeInit

Syntax

```
error_code SMADC1DeInit(u_int8 PortNr);
```

Description

This function deinitialises the SM-ADC1 Module on the port *PortNr* by freeing the resources tied up with it and setting all outputs and inputs to zero.

Input

`u_int8 PortNr`	SM-Port number to de-initialise (from 0 to 10)

Output

`error_code`	`SUCCESS`
	`E_SEEK` if module not initialised
	or standard OS-9 error code (refer to the OS-9 Technical Manual Error Codes Section).

Example

```
RetVal = SMADC1DeInit(0);
```

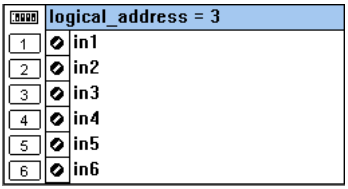

© PEP Modular Computers GmbH

5.4.9 ISaGRAF Programming

5.4.9.1 The ISaGRAF Board Parameters

Information on board parameters may be found in the PEP online help and ISaGRAF online help or user's manual.

Figure 5.4.9.1.1 Typical Screen Section for the SM-ADC1 Module



Logical Address

The 6 inputs of this module may be clearly seen together with the logical address at which this particular board is residing. Remember, this logical address is the same as the physical slot position! In the example shown here, the board being addressed is in slot 3 i.e. the third of the SMART-BASE slots. Up to 11 slots are catered for in the SMART I/O concept; 3 on the SMART-BASE and 2 for each connected SMART-EXT (up to a total of 4).

 © PEP Modular Computers GmbH

5.4.9.2 The ISaGRAF Operate Calls

Operate calls are built into a program using ST or FBD languages when defining the project. A typical use could be at the initialization stage to check that the SMART-Modules are in fact located where they have been programmed to be. The syntax of the operate call is as follows:

Syntax

```
<return variable> := OPERATE(<source var>, COMMAND,
                             <source data>);
```

Here the `return variable` is assigned a value associated with the selected `COMMAND` parameter. Each SMART-Module possesses its own set of these `COMMANDS`.

Example

```
<error.code>:= OPERATE(<channel>, O_INIT_CODE, 0);
```

`O_INIT_CODE` is one of a number of distinct commands recognized by the PEP Modular Computers' implementation of board drivers and checks for example that the board is located where the program expects it to be.

`channel` provides channel specific information and in the example shown here, any of the 8 input channels may be used. The last parameter is not usually used by PEP implementations and is set to 0 (zero).

The `error.code` returns a value of zero if no error was detected, otherwise it returns a non-zero value depending on the error encountered. A list of these error codes may be found in the PEP online help.

A complete list of the operate COMMANDS may be obtained by selecting a project from the ISaGRAF ***projects*** group, opening an application and observing the ***Common defines*** in the ***Dictionary*** pull-down menu. Note that not all calls in the list may be used within the SMART2 environment however, the calls applicable to this module are :

O_INIT_CODE : The syntax and usage have already been explained.

O_ENABLE_CONV : This operate call tells ISaGRAF to enable the conversion of a particular channel of a connected SM-ADC1 module. The syntax is as follows:

```
<analog var> := OPERATE(<iovar>, O_ENABLE_CONV, 0);
```

where `<analog var>` represents the analog return variable and can be zero if the operate call is supported and the correct `<iovar>` is passed or non-zero if an error occurred.

`<iovar>` is the variable name for the channel where the operate call is to take effect.

O_DISABLE_CONV : This operate call tells ISaGRAF to disable the conversion of a particular channel of a connected SM-ADC1 module. The syntax is as follows:

```
<analog var> := OPERATE(<iovar>, O_DISABLE_CONV, 0);
```

where `<analog var>` represents the analog return variable and can be zero if the operate call is supported and the correct `<iovar>` is passed or non-zero if an error occurred.

`<iovar>` is the variable name for the channel where the operate call is to take effect.

 © PEP Modular Computers GmbH

O_ENABLE_LINE_CHECK:

This operate call allows the user to enable open-loop checking on the current version of the SM-ADC1. The syntax is as follows:

```
<analog var> := OPERATE(<iovar>, O_ENABLE_LINE_CHECK,
                        <var>);
```

where	`<analog var>`	represents the analog return variable and can be zero if the operate call is supported and the correct `<iovar>` is passed or non-zero if an error occurred.
	`<iovar>`	is the variable name for the channel where the operate call is to take effect.
	<var>	is not used.

O_DISABLE_LINE_CHECK:

This operate call allows the user to disable open-loop checking on the current version of the SM-ADC1. The syntax is as follows:

```
<analog var> := OPERATE(<iovar>, O_DISABLE_LINE_CHECK,
                        <var>);
```

where	`<analog var>`	represents the analog return variable and can be zero if the operate call is supported and the correct `<iovar>` is passed or non-zero if an error occurred.
	`<iovar>`	is the variable name for the channel where the operate call is to take effect.
	<var>	is not used.

 © PEP Modular Computers GmbH

O_SET_LINE_LIMIT:

This operate call allows the user to set the open loop detect level on the current version of the SM-ADC1. The syntax is as follows:

```
<analog var> := OPERATE(<iovar>, O_SET_LINE_LIMIT,
                        <var>);
```

where `<analog var>` represents the analog return variable and can be zero if the operate call is supported and the correct `<iovar>` is passed or non-zero if an error occurred.

`<iovar>` is the variable name for the channel where the operate call is to take effect.

<var> is the current value in mA and calculated using the following formula:

$$var = \frac{0xFFF0}{0x14} \text{ mA}$$

O_GET_LINE_STATE:

This operate call allows the user to check the line state on the current version of the SM-ADC1. The syntax is as follows:

```
<analog var> := OPERATE(<iovar>, O_GET_LINE_STATE,
                        <var>);
```

where `<analog var>` represents the analog return variable and is zero if an open loop is detected. Otherwise it returns 1.

`<iovar>` is the variable name for the channel where the operate call is to take effect.

<var> is not used.

This page has been left blank intentionally.

© PEP Modular Computers GmbH

5.5 SM-DAC1

5.5.1 Introduction

The SM-DAC1 provides a fast, 12-bit, bipolar (± 10V DC), 2/6-channel digital to analog converter. Two configurable red LEDs on the front panel form the user interface while an on-board EEPROM stores unique calibration data required by the signal converter.

A version of the SM-DAC1 for current output (0 .. 20mA) is also available.

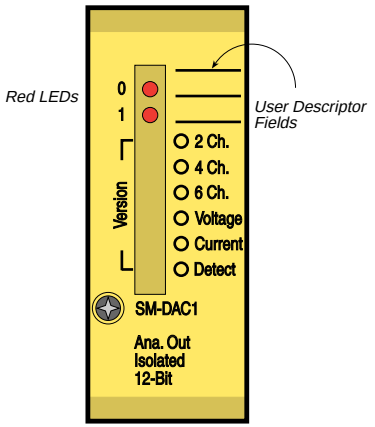
5.5.2 Specifications

Isolation	1.0 kV Optoisolated from the system
Output	2/6 Outputs Output Range ± 10V DC (0 .. 20mA) 12-bit DAC with 1-bit non-linearity 11-bit Repeating Accuracy Glitch-Free Power-On-Reset 2mA (max.) output current (voltage version) 75µs Conversion Time @ 1MHz SCLK †
Ext. Vcc	24V DC (-10% .. +20%)
EEPROM	256 Byte serial EEPROM for Calibration Data
Front Panel	2 Red LEDs (User Configurable)
Power Cons.	660mW (max. voltage version) 400mW (max. current version)
Temperature Range	Standard (0°C to +70°C), Ext./Storage (-40°C to +85°C)
Module Weight	70g
ID Byte	$40, read by the SPI Interface

† The conversion time applies per channel for a full scale voltage swing of (-10V .. +10V)

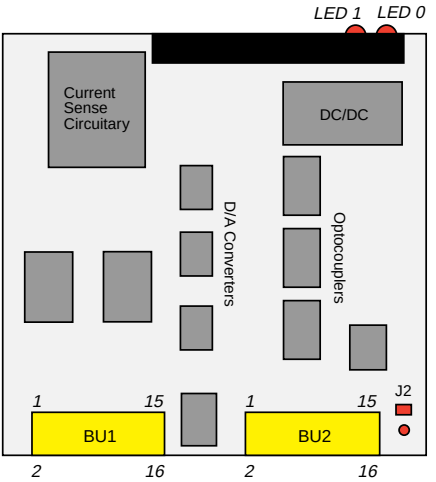
©1996 PEP Modular Computers GmbH

5.5.3 Front Panel Layout



5.5.4 Board Overview

Component Side



© PEP Modular Computers GmbH

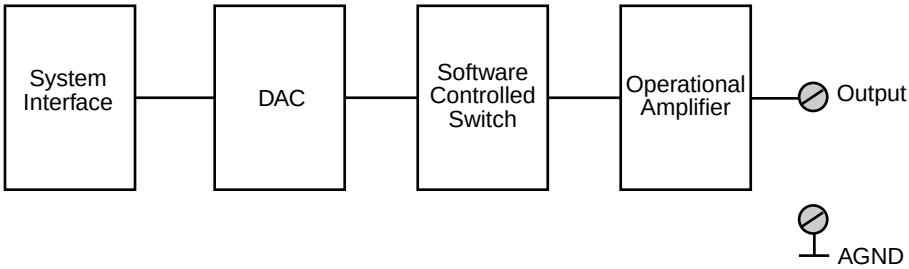
5.5.5 Functional Description

The SM-DAC1 has 2, 4 or 6 differential voltage or current outputs depending on the ordered type. An on-board EEPROM contains board specific calibration data, module ID byte, sub ID byte and production data. A DC/DC converter generates the 5V to ±15V power requirements for the D/A converter and provides additional system isolation.

5.5.5.1 Output Circuitry

The output stage of the SM-DAC1 comprises a fast D/A converter followed by a software controlled switch that ensures the on-board op-amp remains inactive and produces 0V at it's output during power-up. The current version has an additional MOSFET after this op-amp.

Figure 5.5.5.1: SM-DAC1 Output Shematic Diagram



The current version of this module utilizes current sensing that detects open loads or broken lines. The individual comparators of this sensing circuit alarm the software when currents below 4mA are detected.

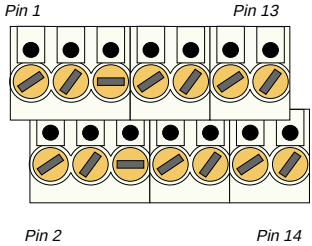
5.5.6 Configuration

Although the SM-DAC1 has one solder jumper for EEPROM configuration, the user should not interfere with it.

5.5.7 Pinouts

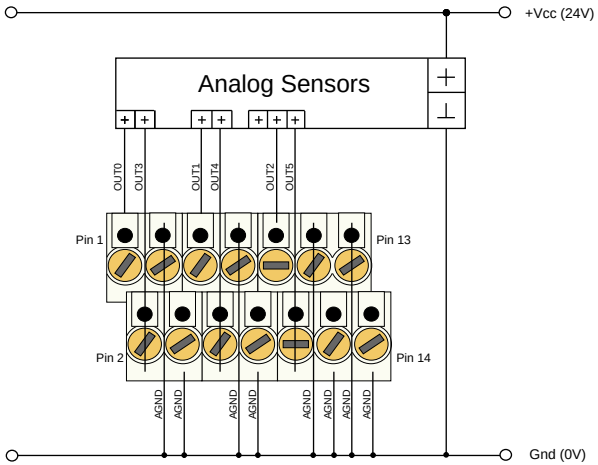
Screw Terminal Pinouts

The following shows the pinout/signal relationship for the SM-DAC1 when connected to a particular screw terminal block.

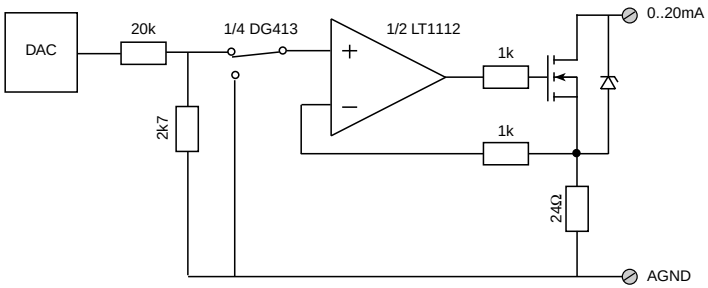


Pin Nr.	Signal	Description	Pin Nr.	Signal	Description
1	OUT0	Analog Output Ch. 0	2	OUT3	Analog Output Ch. 3
3	AGND	Analog GND	4	AGND	Analog GND
5	OUT1	Analog Output Ch. 1	6	OUT4	Analog Output Ch. 4
7	AGND	Analog GND	8	AGND	Analog GND
9	OUT2	Analog Output Ch. 2	10	OUT5	Analog Output Ch. 5
11	AGND	Analog GND	12	AGND	Analog GND
13	AGND	Analog GND	14	AGND	Analog GND

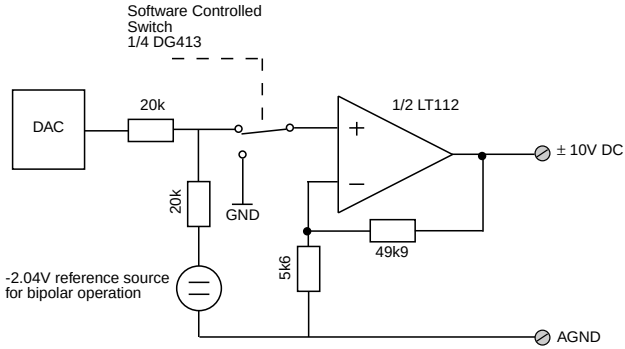
Connection:



Current Output Circuit:



Voltage Output Circuit:



© PEP Modular Computers GmbH

5.5.8 'C' Programming

5.5.8.1 SM-DAC1 Library

The SM-DAC1 library of functions *smartio.l* provide a convenient way of accessing the SM-DAC1 module.

- All functions are written in ANSI C;
- Prototypes are to be found in the file *dac1lib.h.*

Hardware Requirements

- SMART2 Base Unit or SMART I/O Base and Extension Units;
- SM-DAC1 Module.

Software Requirements

The compiler from one of the following:

- Ultra C Version 1.1.2 or higher;
- FasTrak 2.0.2 or higher.

The leftmost SM-Module has number 0 assigned to it a far as programming is concerned although physically this is slot 1!

Before a library function can be used, the function *SMDAC1Init* must first be called. This allocates the requested resources and should be noted that it is not designed for multi-tasking. Furthermore, this function needs to be called for each SM-DAC1 Module called within the task. Upon completion of the application, the function *SMDAC1DeInit* needs to be called for each SM-DAC1 Module that has been initialised.

In order to illustrate the use of the SM-DAC1 library, an application example called *demodac1.c* can be found in the SMART I/O application directory (normally found in */<device>/APPLIC/SMART/CMDS*).

 ©1996 PEP Modular Computers GmbH

5.5.8.2 SMDAC1Init

Syntax

```
error_code SMDAC1Init(u_int8 PortNr);
```

Description

This function initialises the SM-DAC1 Module on port *PortNr*, and:
- tests the module's ID
- reads the conversion table held in EEPROM
- allocates space for the fetched resources

Input

`u_int8 PortNr`	Port number of module to initialise (from 0 to 10)

Output

`error_code`	`SUCCESS` or	
	`E_BMODE`	if wrong or no module
	`E_MEMFUL`	if no memory resources available

or standard OS-9 error code (refer to the OS-9 Technical Manual Error Codes Section).

Example

```
RetVal = SMDAC1Init(0);
```

 © PEP Modular Computers GmbH

5.5.8.3 SMDAC1OpenLoop

Syntax

```
error_code SMDAC1OpenLoop(u_int8 PortNr, u_int8
                          pStatus);
```

Description

This function should only be used for the current version of the SM-DAC1 module and checks the wires for open loop in a 4..20mA environment. The result is stored in a register with the following definition.

Register value:	111110	-> channel 0 connected and o.k
	101111	-> channel 4 connected and o.k

Input

`u_int8 PortNr`	SM-Port to be accessed
`u_int8 pStatus`	Pointer to the location where the status is stored: 0 -> line connected and o.k 1 -> line below 4mA

Output

`error_code`	`SUCCESS`
	`E_SEEK` if module was not initialised
	`E-BMODE` if the module is unknown
	or standard OS-9 error code (refer to the OS-9 Technical Manual Error Codes Section).

Example

```
RetVal = SDAC1OpenLoop(0, &Status);
```

Note

If this function is called and requested for use with a voltage version of the SM-DAC1 module then an error will be returned. Likewise, if a channel is requested that is out of range then an error is returned.

©1996 PEP Modular Computers GmbH

5.5.8.4 SMADAC1Operate

Syntax

```
error_code SMDAC1Operate(u_int8 PortNr);
```

Description

This function powers up the outputs as they are disconnected from the screw terminals by default. This is the software control of the software switch shown in the schematics earlier in this section of the manual.

Input

`u_int8 PortNr`	SM-Port to be accessed

Output

`error_code`	`SUCCESS`	
	`E_SEEK`	if module was not initialised
	`E-BMODE`	if the module is unknown
	or standard OS-9 error code (refer to the OS-9 Technical Manual Error Codes Section).	

Example

```
RetVal = SMAC1Operate(0);
```


© PEP Modular Computers GmbH

5.5.8.5 SMDAC1StandBy

Syntax

```
error_code SMDAC1StandBy(u_int8 PortNr);
```

Description

This function places the module connected on Port *PortNr* in standby mode where the outputs are disconnected from the screw terminal block.

Input

`u_int8 PortNr`	SM-Port number to be accessed

Output

`error_code`	`SUCCESS`	
	`E_SEEK`	if module was not initialised
	`E-BMODE`	if the module is unknown
	or standard OS-9 error code (refer to the OS-9 Technical Manual Error Codes Section).	

Example

```
RetVal = SMDAC1StandBy(0);
```


© PEP Modular Computers GmbH

5.5.8.6 SMDAC1PutVRaw

Syntax

```
error_code SMDAC1PutVRaw(u_int8 PortNr, u_int8
                         Channel, u_int16 Value);
```

Description

This function sets the selected DAC channel to the chosen value. No gain or offset correction is performed and the given value will always be masked to a 12-bit value.

Input

`u_int8 PortNr`	SM-Port number to be accessed
`u_int8 Channel`	Number of the DAC output channel to access (0-5)
`u_int16 Value`	Value to be written

Output

`error_code`	`SUCCESS`	
	`E_SEEK`	if module was not initialized
	`E-BMODE`	if the module is unknown

or standard OS-9 error code (refer to the OS-9 Technical Manual Error Codes Section).

Example

```
RetVal = SMDAC1PutVRaw(0, 1, 0x0FFF);
```

Value (16-bit HEX)			
Unipolar		**Bipolar**	
Positive Full-Scale	$0FFF	Positive Full-Scale	$0FFF
+1LSB	$0001	+1LSB	$0801
0	$0000	0	$0800
		-1LSB	$07FF
		Negative Full-Scale	$0000

© PEP Modular Computers GmbH

5.5.8.7 SMDAC1PutV

Syntax

```
error_code SMDAC1PutV(u_int8 PortNr, u_int8 Channel,
                      int16 Value);
```

Description

This function sets the selected DAC channel to the chosen value. A gain and offset correction is performed with the use of the automatically selected conversion table depending whether a user-defined table has been specified or not. Hence, if a user-defined table exists then this will be used, otherwise the default factory set table will be used.

Input

`u_int8 PortNr`	SM-Port number to be accessed
`u_int8 Channel`	Number of the DAC output channel to access (0-5)
`int16 Value`	Value to be written

Output

`error_code`

`SUCCESS`	
`E_SEEK`	if module was not initialized
`E-BMODE`	if the module is unknown

or standard OS-9 error code (refer to the OS-9 Technical Manual Error Codes Section).

Example

```
RetVal = SMDAC1PutV(0, 1, 0x0FFF);
```

Value (16-bit HEX)			
Unipolar		**Bipolar**	
Positive Full-Scale	$0FFF	Positive Full-Scale	$07FF
+1LSB	$0001	+1LSB	$0001
0	$0000	0	$0000
		-1LSB	$FFFF
		Negative Full-Scale	$F800

5.5.8.8 SMDAC1SetLed

Syntax

```
error_code SMDAC1SetLed(u_int8 PortNr, u_int8 LedNr);
```

Description

This function switches on one of the two LEDs fitted to the SM-DAC1.

Input

`u_int8 PortNr`	SM-Port number to be accessed
`u_int8 LedNr`	Number of the LED to switch on (1-2)

Output

`error_code`	`SUCCESS`	
	`E_SEEK`	if module was not initialized
	`E-BMODE`	if the module is unknown
	or standard OS-9 error code (refer to the OS-9 Technical Manual Error Codes Section).	

Example

```
RetVal = SMDAC1SetLed(0, 2);
```


© PEP Modular Computers GmbH

5.5.8.9 SMDAC1ClrLed

Syntax

```
error_code SMDAC1ClrLed(u_int8 PortNr, u_int8 LedNr);
```

Description

This function switches off one of the two LEDs fitted to the SM-DAC1.

Input

`u_int8 PortNr`	SM-Port number to be accessed
`u_int8 LedNr`	Number of the LED to switch off (1-2)

Output

`error_code`	`SUCCESS`	
	`E_SEEK`	if module was not initialized
	`E-BMODE`	if the module is unknown

or standard OS-9 error code (refer to the OS-9 Technical Manual Error Codes Section).

Example

```
RetVal = SMDAC1ClrLed(0, 2);
```


© PEP Modular Computers GmbH

5.5.8.10 SMDAC1DeInit

Syntax

```
error_code SMDAC1DeInit(u_int8 PortNr);
```

Description

This function deinitialises the SM-DAC1 Module on the port *PortNr* by freeing the resources tied up with it and shuts down the outputs.

Input

`u_int8 PortNr`	SM-Port number to de-initialise (from 0 to 10)

Output

`error_code`	`SUCCESS`
	`E_SEEK` if module not initialised
	or standard OS-9 error code (refer to the OS-9 Technical Manual Error Codes Section).

Example

```
RetVal = SMDAC1DeInit(0);
```

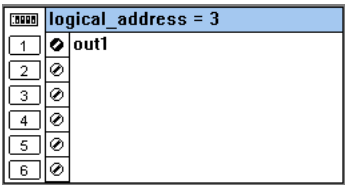

© PEP Modular Computers GmbH

5.5.9 ISaGRAF Programming

5.5.9.1 The ISaGRAF Board Parameters

Information on board parameters may be found in the PEP online help and ISaGRAF online help or user's manual.

Figure 5.5.9.1.1 Typical Screen Section for the SM-DAC1 Module



Logical Address

One of the outputs of this module may be clearly seen together with the logical address at which this particular board is residing. Remember, this logical address is the same as the physical slot position!

5.5.9.2 The ISaGRAF Operate Calls

Operate calls are built into a program using ST or FBD languages when defining the project. A typical use could be at the initialization stage to check that the SMART-Modules are in fact located where they have been programmed to be. The syntax of the operate call is as follows:

Syntax

```
<return variable> := OPERATE(<source var>, COMMAND,
                             <source data>);
```

Here the `return variable` is assigned a value associated with the selected `COMMAND` parameter. Each SMART-Module possesses its own set of these `COMMANDS`.

Example

```
<error.code>:= OPERATE(<channel>, O_INIT_CODE, 0);
```

`O_INIT_CODE` is one of a number of distinct commands recognized by the PEP Modular Computers' implementation of board drivers and checks for example that the board is located where the program expects it to be.

`channel` provides channel specific information and in the example shown here, any of the 8 input channels may be used. The last parameter is not usually used by PEP implementations and is set to 0 (zero).

The `error.code` returns a value of zero if no error was detected, otherwise it returns a non-zero value depending on the error encountered. A list of these error codes may be found in the PEP online help.

 © PEP Modular Computers GmbH

A complete list of the operate COMMANDS may be obtained by selecting a project from the ISaGRAF ***projects*** group, opening an application and observing the ***Common defines*** in the ***Dictionary*** pull-down menu. Note that not all calls in the list may be used within the SMART2 environment however, the calls applicable to this module are :

O_INIT_CODE : The syntax and usage have already been explained.

O_SET_LED : This operate call allows the user to control the two red LEDs on the board's front panel. The syntax is as follows:

```
<analog var> := OPERATE(<iovar>, O_SET_LED, <var>);
```

where `<analog var>` represents the analog return variable and can be zero if the operate call is supported and the correct `<var>` is passed or non-zero if an error occurred.

`<iovar>` represents any module variable and

`<var>` represents the LED to be controlled and must be one of:

`OA_LED1_ON, OA_LED2_ON, OA_LED1_OFF` or `OA_LED2_OFF.`

O_ENABLE_CONV : This operate calls tells ISaGRAF to enable the conversion of a particular channel of a connected SM-DAC1 module. The syntax is as follows:

```
<analog var> := OPERATE(<iovar>, O_ENABLE_CONV, 0);
```

where `<analog var>` represents the analog return variable and can be zero if the operate call is supported and the correct `<iovar>` is passed or non-zero if an error occurred.

`<iovar>` is the variable name for the channel where the operate call is to take effect.

O_DISABLE_CONV: This operate call tells ISaGRAF to disable the conversion of a particular channel of a connected SM-DAC1 module. The syntax is as follows:

```
<analog var> := OPERATE(<iovar>, O_DISABLE_CONV, 0);
```

where `<analog var>` represents the analog return variable and can be zero if the operate call is supported and the correct `<iovar>` is passed or non-zero if an error occurred.

`<iovar>` is the variable name for the channel where the operate call is to take effect.

O_GET_LINE_STATE:

This operate call allows the user to check the line state on the current version of the SM-DAC1. The syntax is as follows:

```
<analog var> := OPERATE(<iovar>, O_GET_LINE_STATE,
                        <var>);
```

where `<analog var>` represents the analog return variable and is zero if an open loop is detected. Otherwise it returns 1.

`<iovar>` is the variable name for the channel where the operate call is to take effect.

<var> is not used.

 © PEP Modular Computers GmbH

Communication Modules

This page was intentionally left blank.

Table of Contents

Chapter 6

6.1 SM-SSI 6-5

6.1.1 Introduction 6-5
6.1.2 Specifications 6-5
6.1.3 Front Panel Layout 6-6
6.1.4 Board Overview 6-6
6.1.5 Functional Description 6-7
6.1.6 SSI Operation 6-8
6.1.7 Register Description 6-10
6.1.8 MATCH Function 6-14
6.1.9 Tested Sensors 6-14
6.1.10 Configuration 6-15
6.1.11 Pinout 6-15
6.1.12 'C' Programming 6-17
6.1.13 ISaGRAF Programming 6-24

This page was intentionally left blank.

6.1 SM-SSI

6.1.1 Introduction

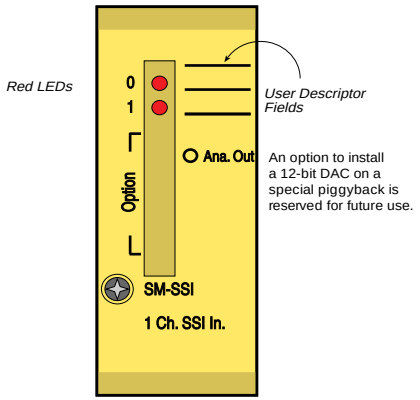
The SM-SSI is an optoisolated single-channel, 24-bit RS422 interface for absolute encoders. Configurable data input together with flexibility afforded by software configurable encoder types transforms this SMART Module into an intelligent interface suitable for the most demanding applications. Data flow is indicated by the presence of two red LEDs on the front panel.

6.1.2 Specifications

Isolation	2.5 kV optoisolated from the system
Input	1 optoisolated SSI channel, 24-bit binary or Gray code (software configurable), RS422 type 2 optoisolated 24V DC digital inputs for common GND and End-of-Range switches with readback and IRQ support
Output†	2 optoisolated 24V DC/300mA digital outputs for Encoder Reset and Encoder Direction 1 optoisolated 24V DC/300mA digital output for Match
SSI Interface	None, Even or Odd (software configurable), Data, Parity Software configurable (data bits, parity) encoder type Software detection of line break SSI defined, RS422 type Clock Output 62.5, 125, 250 or 500kHz (programmable) Tx. Pulse Freq. Data flow controlled by software and indicated by LED status Up to 24-bit Compare Register with pass-through compare Interrupt, software and LED control of Match Result
Front Panel	2 Red LEDs indicating MATCH and data conversation
Ext. Vcc	24V DC (±5%)
Power Cons.	325 mW (min.), 375mW (max.)
Temperature Range	Standard (0˚C to +70˚C), Extended/Storage (-40˚C to +85˚C)
Operating Humidity	0% to 95% (non-condensing)

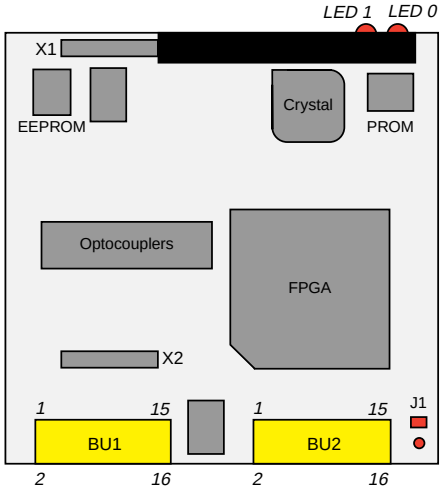
† *All outputs are referenced to the common Ext. Vcc*

6.1.3 Front Panel Layout



6.1.4 Board Overview

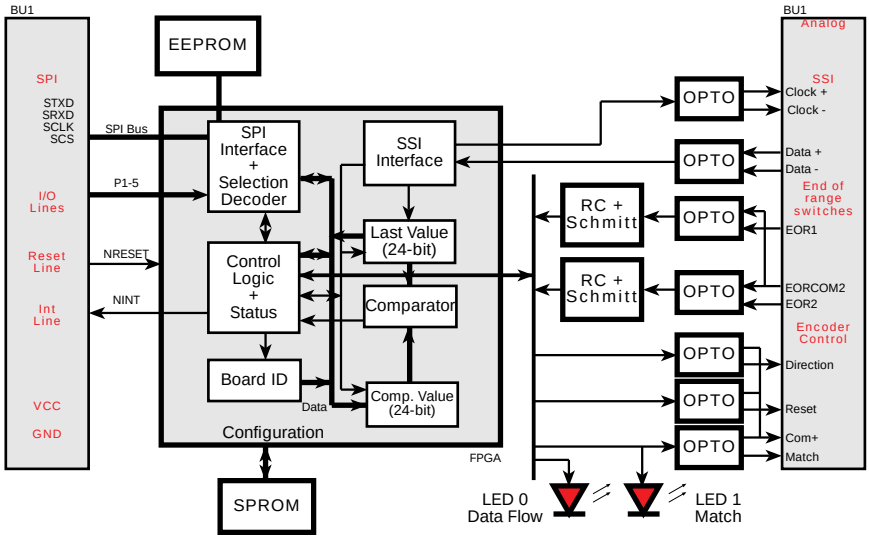
Component Side



© PEP Modular Computers GmbH

6.1.5 Functional Description

Figure 6.1.5.1: SM-SSI Schematic Diagram



The Synchronous Serial Interface working together with a photographically etched encoder disk allows absolute codes (positions) of rotating axis to be continuously monitored. The greatest advantage of the SSI communication process compared to that of standard serial/parallel procedures is that the controller actually regulates the timing and speed of the data transmission which further optimizes data security.

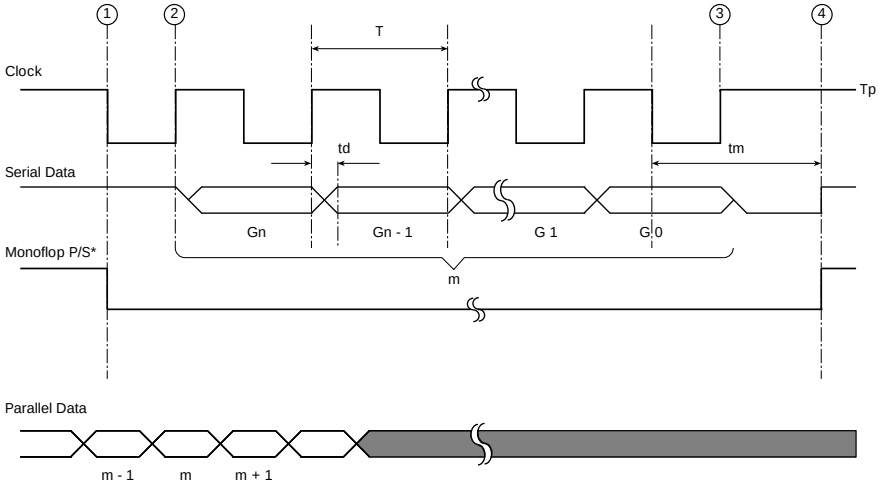
Simplicity in design with low conventional component count coupled with the inherent safety offered through accurate axis position reading even after power fail, enable this compact module to be utilized for equipment operating even in extreme environmental conditions.

6.1.6 SSI Operation

With reference to figures 6.2.6.1 and 6.2.6.2, the operation of the SSI interface will be discussed.

- The position of the encoder disk is continuously clocked.
- Position data is ready for conversion by the parallel -> serial converter.
- The controller (SM-SSI) demands the axis angle from the encoder which starts the SSI clock - the number of clock pulses required is dependent on the data width (24-bits).

Figure 6.1.6.1: SM-SSI Timing Diagram



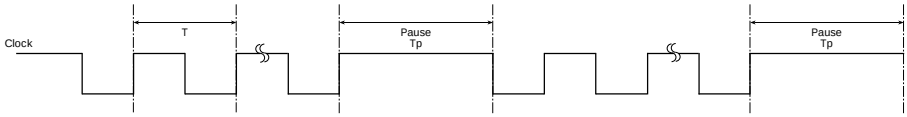
m = Stored parallel data
td = Delay time for the first clock
Gn = MSB of Gray* code
Tp = Pause time in SSI clock
T = SSI clock period
G0 = LSB of Gray* code
tm = Monoflop time

* Binary code may also be converted.

 © PEP Modular Computers GmbH

- The first HIGH to LOW clock edge (1) in figure 6.2.6.1, triggers the monoflop and allows the current parallel data word present in the parallel -> serial converter to be processed. The monoflop prevents other data transfers to this parallel to serial converter.
- The crossing of the first LOW to HIGH clock edge after the trigger (2) in figure 6.2.6.1 allows the MSB of the Gray code or binary signal to be transferred.
- With every successive LOW to HIGH transition of the SSI clock, the next bit in the data stream is transferred; this action continues until all the data bits have been transferred. The SSI clock continually retriggers the monoflop forcing it's output to remain low and thereby preventing unwanted data from being processed.
- When the controller (SM-SSI) has received the LSB of the data stream it stops the SSI clock as shown in point (3) of figure 6.2.6.1.
- The monoflop ceases to be triggered and after a time 'tm' returns to a high state as shown in (4) of figure 6.2.6.1. This allows a new data value to be transferred to the parallel -> serial converter.

Figure 6.1.6.2: SM-SSI Timing Diagram for serial transfer



© PEP Modular Computers GmbH

6.1.7 Register Description

The SM-SSI SMART-Module connected to a SMART I/O slot uses the SPI transparent mode for on-board data reading and writing and 5 parallel lines for data selection. On board are a set of registers and peripherals shown in the table below:

No.	Description	Direction
1	Internal 16-bit Status Register	Read Only
1	Internal 24-bit Control Register	Write Only
1	Internal 8-bit Identification Register	Read Only
1	Internal 24-bit Data Register	Read Only
1	Internal 24-bit Compare Register	Write Only
1	SPI compatible serial EEPROM	Read/Write
1	Optional SPI compatible serial Digital to Analog Converter	Read/Write

Due to the fact that SPI transmissions are full duplex, some read only registers are accessed at the same time as write only registers in order to optimize data transmission. Therefore, when the Control Register is written to, simultaneously the Status Register is read and similarly when writing to the Compare Register, the Data Register is read.

256 bytes of serial EEPROM contain the module ID number and production data although and further space has been allocated for future use.

6.1.7.1 Control Register

The 24-bit control register is divided into three sub-registers which are cleared after a power-on reset and control:

- **FDL**: Format Data Length
- **P** : Parity Enable/Disable
- **ENI** : Interrupt Masks
- **SOT**: Start Transmission
- **CF** : Data Clock Frequency
- **AR** : Automatic Reading Mode
- **RST**: Encoder Reset Control
- **PT** : Parity Type (Odd/Even)
- **DT** : Data Type (Gray/Binary)
- **CM** : Clear Match output

6.1.7.2 CTRL1 Register

CTRL1-7	CTRL1-6	CTRL1-5	CTRL1-4	CTRL1-3	CTRL1-2	CTRL1-1
RST	AR	SOT	DIR	EN13-MH	EN12-EOR	ENI1-TC

ENIO-P	enable / disable (1 / 0) interrupt on parity error
ENI1-TC	enable / disable (1 / 0) interrupt on transmission complete
ENI2-EOR	enable / disable (1 / 0) interrupt on end-of-range switch 1 / 2
ENI3-MH	enable / disable (1 / 0) interrupt on comparison match
DIR	direction control (‘1’ activates the optocouplers)
SOT	start transmission control (‘1’ = set; single-shot-set by lib.)
AR	select automatic repeat mode (when set -> ‘1’)
RST	set / clear (1 / 0) Reset output

6.1.7.3 CTRL2 Register

CTRL2-7	CTRL2-6	CTRL2-5	CTRL2-4	CTRL2-3	CTRL2-2	CTRL2-1
CF1	CF0	P	FDL4	FDL3	FDL2	FDL1

FDL0-4	number of data bits, excluding parity, minus 1
P	enable / disable (1 / 0) use of parity (encoder dependent)
CF1-0	data transmission frequency:
	0-0 62.5 kHz
	1-0 125 kHz
	0-1 250 kHz
	1-1 500 kHz

6.1.7.4 CTRL3 Register

CTRL3-7	CTRL3-6	CTRL3-5	CTRL3-4	CTRL3-3	CTRL3-2	CTRL3-1
unused	unused	unused	unused	unused	PT	DT

CM	A logical ‘1’ clears the MATCH output and LED
DT	data type (0:Gray, 1:Binary) according to encoder type If ‘0’ is set, the result in the data register is binary.
PT	parity type (0:Odd, 1:Even) according to encoder type.

6.1.7.5 Compare Register

This 24-bit register indicates the comparison value and is cleared after a power-on reset.

6.1.7.6 Identification Register

The identification register is read only and fixed at a value of $81.

6.1.7.7 Status Register

The 16-bit status register is divided into two sub-registers which are partly cleared after a power-on reset and show:

- **IP** : Interrupt Pending flags
- **PE** : Parity Error
- **TC** : Transmission Complete
- **OL** : Open Line
- **DF** : Data Flow
- **ER1** : End-of-Range 1
- **ER2** : End-of-Range 2
- **MH** : Match

IP, PE, TC, OL, DF and MH flags are cleared after a reset. ER1 and ER2 represent the current hardware condition

6.1.7.8 STAT1 Register

ST1-7	ST1-6	ST1-5	ST1-4	ST1-3	ST1-2	ST1-1
OL	unused	DF	IP4-MH	IP3-ER2	IP2-ER1	IP1-TC

IP0-PE	interrupt pending on parity error
IP1-TC	interrupt pending on transmission complete
IP2-ER1	interrupt pending on end-of-range switch 1
IP3-ER2	interrupt pending on end-of-range switch 2
IP4-MH	interrupt pending on comparison match
DF	SSI data flow (True/False)
OL	open line (True/False)

Note

Interrupt pending flags (IP0 - IP4) are set by the interrupt source if the corresponding line is not masked in the CTRL1 register. All flags are cleared when the STAT1 register is read.

6.1.7.9 STAT2 Register

ST2-7	ST2-6	ST2-5	ST2-4	ST2-3	ST2-2	ST2-1
unused	unused	unused	MH	ER2	ER1	TC

PE parity error on last transmission (True/False)
TC transmission complete (True/False)
ER1 current status of end-of-range switch 1
ER2 current status of end-of-range switch 2
MH comparison match (True/False)

Note

The MH flag is set internally as soon as the encoder reaches the preset register value with the flag being cleared immediately the status is read. This feature is intended for use with the board in automatic mode to allow slow polling of the status registers while ensuring that a match is not lost.

6.1.7.10 Data Register

The data register, cleared after a power-on reset is a 24-bit register containing the result of the last encoder reading. When using encoders with less than 24-bit data, the result is always aligned from the lowest significant bit and the used upper bits are always cleared.

6.1.8 MATCH Function

The match output, which is useful for detecting when an event occurs, is set using an A > B comparator where:

A = the SSI data
B = the compare register

The MATCH output is set on an edge transition of the comparator output as follows:

When counting up: MATCH is set when A passes from (A<=B) to (A>B)
When counting down: MATCH is set when A passes from (A>B) to (A<=B)

6.1.9 Tested Sensors

The sensors from the following manufacturers have been tested in conjunction with the SM-SSI in an SSI environment and conform to the high standards set by PEP Modular Computers.

Hohner SSI-AVM 10-10 Multi-turn axial 24-bit Gray code
Hengstler RA58S Single-turn 12-bit Gray code
Balluff BTL 3-S11-0250-F-S50 Trans sonar distance gauge 24-bit Gray code.

 © PEP Modular Computers GmbH

6.1.10 Configuration

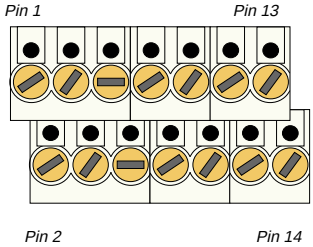
Jumper J1 - EEPROM Protection

Jumper	Settings	Description
J1	*set*	*EEPROM is not hardware write protected*
	open	EEPROM is hardware write protected

6.1.11 Pinouts

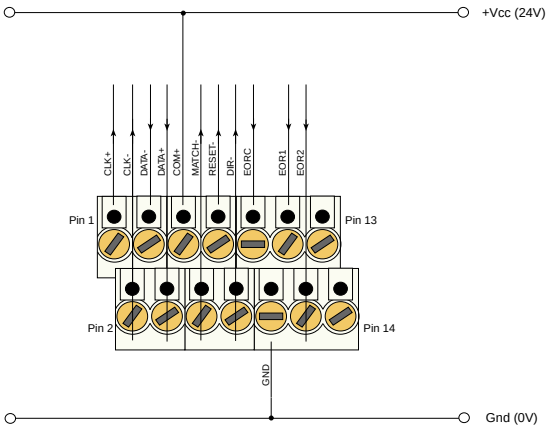
Screw Terminal Pinouts

The following shows the pinout/signal relationship for the SM-SSI when connected to a particular screw terminal block.

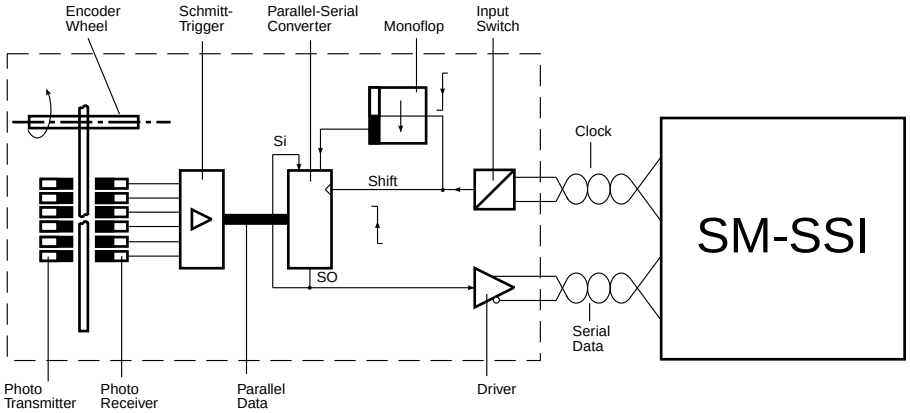


Pin Nr.	Signal	Description	Pin Nr.	Signal	Description
1	CLK+	SSI Clock +	2	CLOCK-	SSI Clock -
3	DATA-	SSI Data -	4	DATA+	SSI Data +
5	COM+	Common +	6	MATCH-	Match -
7	RESET-	Encoder Reset -	8	DIR-	Direction -
9	EORC	EOR Common	10	GND	Ground
11	EOR1	End-of-Range 1	12	EOR2	End-of-Range 2
13	Reserved		14	Reserved	

Connection



Input Circuit



6.1.12 'C' Programming

6.1.12.1 SM-SSI Library

The SM-SSI library of functions *smartio.l* provide a convenient way of accessing the SM-SSI module.

- All functions are written in ANSI C;
- Prototypes are to be found in the file *ssilib.h*.

Hardware Requirements

- SMART I/O Base Module or Base Module and Extension unit;
- SM-SSI1 Module.

Software Requirements

The compiler from one of the following:

- Ultra C Version 1.1.2 or higher;
- FasTrak 2.02 or higher.

The leftmost SM-Module has number 0 assigned to it a far as programming is concerned although physically this is slot 1!

Before a library function can be used, the function *SMSSIInit* must first be called. This allocates the requested resources. Furthermore, this function needs to be called for each SM-SSI Module called within the task. Upon completion of the application, the function *SMSSIDeInit* needs to be called for each SM-SSI Module that has been initialized.

In order to illustrate the use of the SM-SSI library, an application example called *demssi.c* can be found in the SMART I/O application directory (normally found in */<device>/APPLIC/SMART/CMDS*).

6.1.12.2 SMSSIInit

Syntax

```
error_code SMSSIInit(u_int8 PortNr);
```

Description

This function initializes the SM-SSI Module on port *PortNr,* resetting and clearing the status of registers and allocating resource memory.

Input

`u_int8 PortNr`	SM-Port to initialize (from 0 to 10)

Output

`error_code`	`SUCCESS`	
	`E_BMODE`	if the module is unknown

or standard OS-9 error code (refer to the OS-9 Technical Manual Error Codes Section).

Example

```
RetVal = SMSSIInit(0);
```

6.1.12.3 SMSSIDeInit

Syntax

```
error_code SMSSIDeInit(u_int8 PortNr);
```

Description

This function deinitializes the SM-SSI Module on the port *PortNr* by freeing the resources tied up with this module.

Input

`u_int8 PortNr`	SM-Port number to de-initialize (from 0 to 10)

Output

`error_code`	`SUCCESS`	
	`E_BMODE`	if the module is unknown or standard OS-9 error code (refer to the OS-9 Technical Manual Error Codes Section).

Example

```
RetVal = SMSSIDeInit(0);
```


© PEP Modular Computers GmbH

6.1.12.4 SMSSISetSetPoint

Syntax

```
error_code SMSSISetSetPoint(u_int8 PortNr,
                            u_int32 Setpoint);
```

Description

This function copies the setpoint to an internal register which will only be sent to the SM-SSI when the function **SMSSISetCtrlReg()** is called. This function is used to set the preset value for the comparator if testing for a match.

Input

`u_int8 PortNr`	Number of the SM port to set
`u_int32 Setpoint`	Value of the comparator register (0x0 -> 0xFFFFFF) 24-bit

Output

`error_code`	`SUCCESS`	
	`E_BMODE`	if the module is unknown
	or standard OS-9 error code (refer to the OS-9 Technical Manual Error Codes Section).	

Example

```
RetVal = SMSSISetSetPoint(0, 0xFF00FF);
```

6.1.12.5 SMSSISetCtrlReg

Syntax

```
error_code SMSSISetCtrlReg(u_int8 PortNr, long
CtrlReg1, long CtrlReg2, long CtrlReg3, u_int32
*pData);
```

Description

This function sets the internal register images of all 3 parts of the 24-bit control register with values <> -1 and sends them to the SM-SSI. Simultaneously, the contents of the data register holding the last value is read back to the address pointed to by *pData*.

Input

`u_int8 PortNr`	SM-Port number to send register (0 to 10)
`long CntrlReg1`	Control register 1 value (0x0 -> 0xFF)
`long CntrlReg2`	Control register 2 value (0x0 -> 0xFF)
`long CntrlReg3`	Control register 3 value (0x0 -> 0xFF)
`u_int32 *pData`	Pointer to the value returned from the data register (0x0 -> 0xFFFFFF)

Output

`error_code`	`SUCCESS`
	`E_BMODE` if the module is unknown
	or standard OS-9 error code (refer to the OS-9 Technical Manual Error Codes Section).

Example

```
RetVal = SMSSISetCtrlReg(0, 0x80, 0x17, 0x1, &Data);
```

6.1.12.6 SMSSIGetStatus

Syntax

```
error_code SMSSIGetStatus(u_int8 PortNr,
                u_int8 *pStatus1, u_int8 *pStatus2);
```

Description

This function reads the status of the module selected on port *PortNr*.

Input

`u_int8 PortNr`	SM-Port number to get status
`u_int8 *pStatus1`	Pointer to the value read in the status register 1 (0x0 -> 0xFF)
`u_int8 *pStatus2`	Pointer to the value read in the status register 2 (0x0 -> 0xFF)

Output

`error_code`	`SUCCESS`
	`E_BMODE` if the module is unknown
	or standard OS-9 error code (refer to the OS-9 Technical Manual Error Codes Section).

Example

```
RetVal = SMSSIGetStaus(0, &Status1, &Status2);
```



6.1.12.7 SMSSIGetData

Syntax

```
error_code SMSSIGetData(u_int8 PortNr,
                        u_int32 *pData);
```

Description

This function returns the absolute position data from the SM-SSI. If automatic mode is selected then a wait will not be performed otherwise SOT will be set and the read performed only when TC has been set.

Input

`u_int8 PortNr`	SM-Port number to get data
`u_int32 *pData`	Pointer to the value returned from the data register (0x0 -> FFFFFF)

Output

`error_code`	`SUCCESS`
	`E_BMODE` if the module is unknown
	or standard OS-9 error code (refer to the OS-9 Technical Manual Error Codes Section).

Example

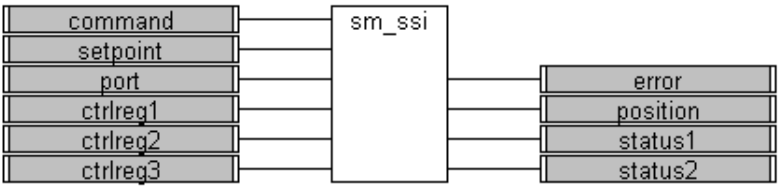
```
RetVal = SMSSIGetData(0, &pData);
```

6.1.13 ISaGRAF Programming

6.2.13.1 The ISaGRAF Board Parameters

Because this module has not been defined as a physical board in the ISaGRAF project, it has been created as a function block as shown in figure 6.2.13.1.1.

Figure 6.2.13.1.1 Function block of the SM-SSI Module



Command

One of:

`FB_SSI_INIT`	This is the command that must be executed to initialize the module before operating the SSI interface - normally at the beginning of an ISaGRAF program cycle. Also, prior to using this call, if the **ctrlx** variables have been defined with a value, then these will be sent to the SM-SSI Module.
`FB_SSI_SETPOINT`	This gives the user the possibility to enter a value for the comparator register when testing for a match between the actual read-back value and the desired value. The compare value should be defined prior to using the call.

FB_SSI_READ	In order that the current state of the encoder may be read, this command is issued - nor mally in a loop built into the ISaGRAF cycle. The read-back value appears in the position variable.

As can be seen from figure 6.2.11.1.1, the inputs to the module are displayed on the left-hand side of the SSI block and the outputs are indicated on the right-hand side.

setpoint	This is the register which contains the value of the setpoint which is sent to the SSI module with the **Command** statement.
port	The port register is used for setting the port number of the attached SM-SSI Module.
ctrx	The control register elements; for a description of the register please refer to the relevant section appearing earlier.
error	If an error occurs then it's value appears in this variable.
position	The position of the encoder following a read is recorded in this variable.
statusx	The status register elements; for a description of the register please refer to the relevant section appearing earlier.

 © PEP Modular Computers GmbH

This page has been left blank intentionally.

© PEP Modular Computers GmbH

Counter Modules

© PEP Modular Computers

This page was intentionally left blank.

Table of Contents

7.1 SM-COUNT1 ..... 7 - 5
7.1.1 Introduction ..... 7 - 5
7.1.2 Specifications ..... 7 - 5
7.1.3 Front Panel Layout ..... 7 - 7
7.1.4 Board Overview ..... 7 - 8
7.1.5 Functional Description ..... 7 - 9
7.1.6 Pinout ..... 7 - 17
7.1.7 ISaGRAF ..... 7 - 18
7.1.8 'C' Programming ..... 7 - 18
7.2 SM-COUNT2 ..... 7 - 42
7.2.1 Introduction ..... 7 - 42
7.2.2 Specifications ..... 7 - 42
7.2.3 Front Panel Layout ..... 7 - 44
7.2.4 Board Overview ..... 7 - 45
7.2.5 Functional Description ..... 7 - 46
7.2.6 Pinout ..... 7 - 51
7.2.7 ISaGRAF ..... 7 - 52
7.2.8 'C' Programming ..... 7 - 52

 © PEP Modular Computers

This page was intentionally left blank.

7. Counter Modules

Depending on the specific requirements of your SMART2 or SMART I/O system, different counter modules may be used according to performance requirements:

- SM-COUNT1 one “up/down” counter with 24-bit resolution;
- SM-COUNT2 two “up” counters with 16-bit resolution

7.1 SM-COUNT1

7.1.1 Introduction

The SM-COUNT1 is an opto-isolated counter module for general purpose usage in a SMART2 or SMART I/O system. The single-counter board operates with 24 bit resolution and envisages four “up/down” counting functions: three counter inputs and one counter output. In addition, there are two end-of-range inputs with interrupt capability and one freely programmable output.

7.1.2 Specifications

Table 7-1: SM-COUNT1 Technical Specifications

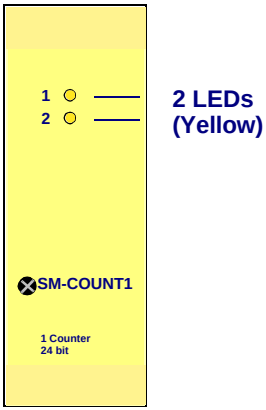
SM-COUNT1	Specifications
Inputs	• 3 differential inputs: “A”, “B”, “STROBE/CLEAR/ UP_DOWN” • 2 end-of-range inputs (relative to COM GND)

Table 7-1: SM-COUNT1 Technical Specifications

SM-COUNT1	Specifications
Input Specifications: • Voltage Ranges • Current • Frequency • Switching	 5 V DC ± 5%, 12 V DC ± 25%, 24 V DC ± 25% 10 mA < 300 kHz, embedded filter 1.55 V/6.7 V/15.1 V respectively
Outputs	• 1 digital output from the counter: TOUT0 • 1 output for general purposes: TOUT1
Output Characteristics: • Voltage • Current • Frequency • Switching	 24 V DC ±25% 100 mA, inductive-load protection < 100 kHz (duty cycle = 1:1) High-side switch
Isolation (I/O to system)	2.5 kV AC
Power Consumption	350 mW (min.), 370 mW (max.)
Bus Interface	SPI
ID Byte	$70
Temperature Range: Industrial Extended	 0°C to +70°C -40°C to +85°C
Operating Humidity	10% to 95%, non-condensing.
Weight	48 g

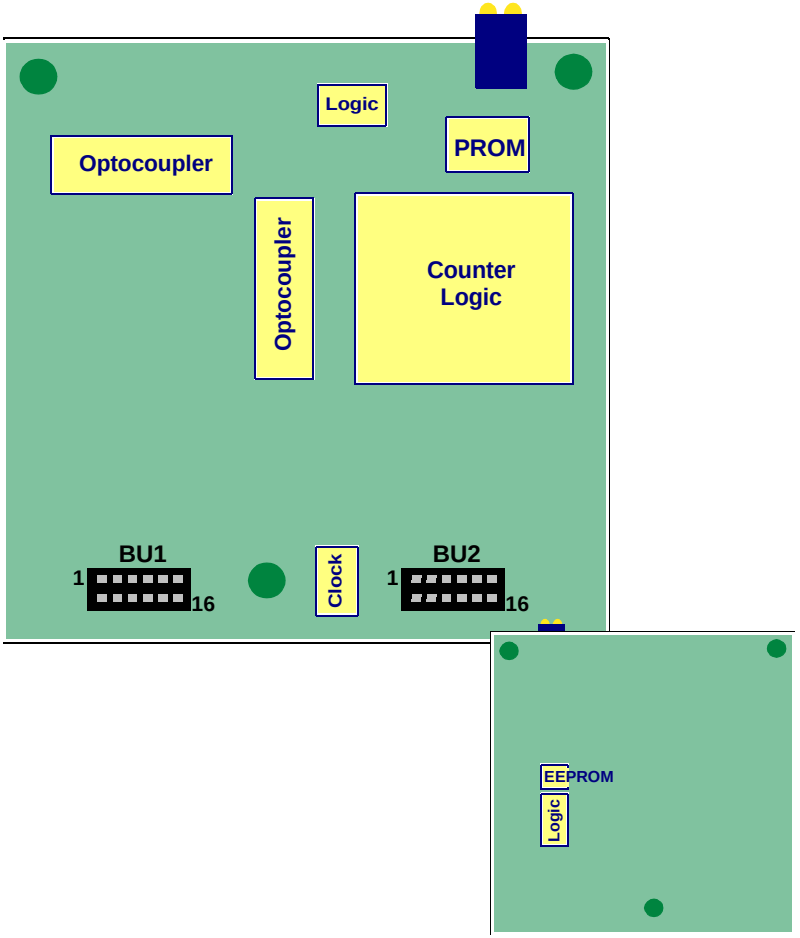
7.1.3 Front Panel Layout

Figure 7-1: SM-COUNT1 Front Panel Layout



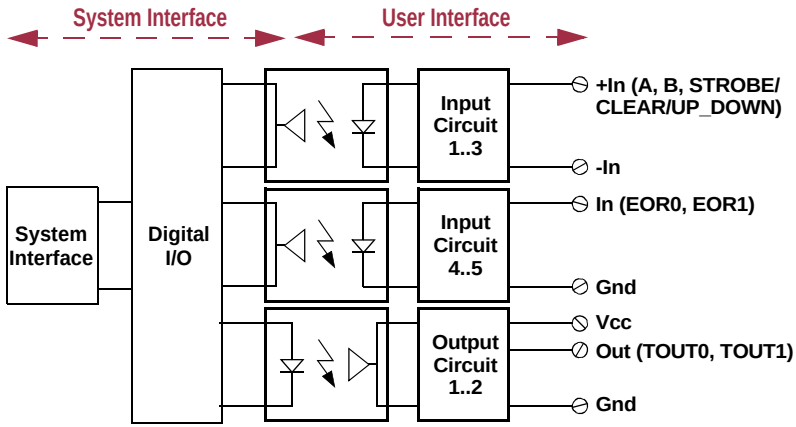
7.1.4 Board Overview

Figure 7-2: SM-COUNT1 Schematic Board Layout — Front View and Reverse View (Small Figure)



7.1.5 Functional Description

Figure 7-3: SM-COUNT1 Functional Description

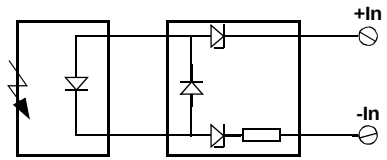


7.1.5.1 Input/Output Functions

Differential Inputs

Three differential inputs allow the control of all counter, gate and mode input signals.

Figure 7-4: SM-COUNT1 Differential Input Circuits

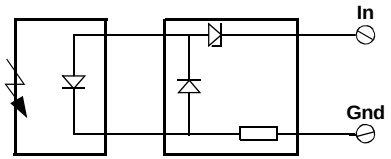


End-of-Range Inputs

Two end-of-range inputs (EOR0 and EOR1) with interrupt capabilities are available to the user. The status of these inputs is given by the status register.

Each EOR input can be configured to generate an interrupt on the rising edge. To do so, please set the corresponding bit in the control register.

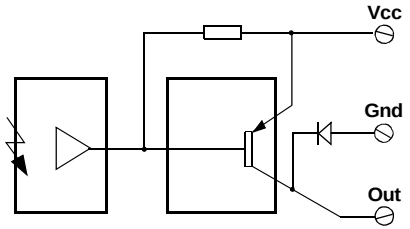
Figure 7-5: SM-COUNT1 End-of-Range Input Circuits



Mode-Dependent Output

This output defines specific counter events, which depend on the currently set counter mode. For example a 1:10 pulse threshold may be set to break down a 100 kHz frequency into 10 kHz counter steps.

Figure 7-6: SM-COUNT1 Output Circuits



General Purpose Output

One general purpose output is available to the user. Its status is controlled by the control register.

7.1.5.2 Counter Modes

Depending on the transducer type and the specific counter/gate correlation, there are two different counter modes with a total of four different resolutions:

1. Normal Counter Mode
2. Quadrature Counter Mode
   - Resolution X1
   - Resolution X2
   - Resolution X4

Normal Counter Mode

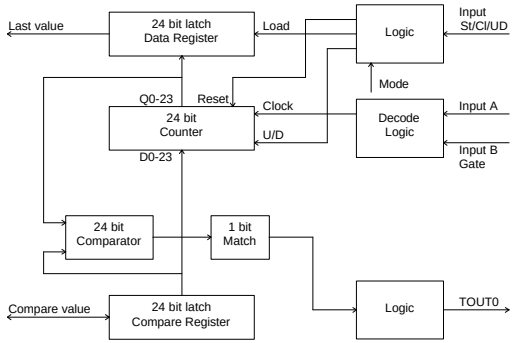
In this mode, the typical counter interfaces have three input lines (“A”, “B”, and “Strobe/Clear/Up&Down”) and one output line (TOUT0). The input line “A” is used to receive counter pulses, input line “B” acts as a gate that disables counting when inactive and the third input, "Strobe/Clear/Up_Down", may be used as either a strobe, clear signal or as an UP/DOWN direction control input. The output line provides a match between the current counter value and a value pre-set in the compare register.

Normal counter operation allows the counter value to be read transparently and offers a continuous comparison between the current value and a previously set reference (Compare Register). In addition, the output (TOUT0) is activated when the two values match. The counter may be set to count up or down and counting may be disabled using input “B” (gate function). The "Strobe/Clear/Up&Down" input monitors the counting direction or allows the data register to be loaded with the current counter value or may be used to clear the counter.

Interrupts can be enabled on a match (counter equal to the Compare Register), on a gate and on a strobe input.

Typically, input “A” acts as the count input and counting will always occur on the rising edge of the incoming pulse. The Control Register (refer to the section on programming) defines the operational mode. When programming, be sure that the other two inputs (“B” and "Strobe/Clear/Up&Down") are properly defined.

Figure 7-7: SM-COUNT1 Normal-Mode Counter Logic



With input "B" acting as a gate, the counter can be made to stop when the gate is inactive. An interrupt can be generated on the rising edge of the gate (inactive → active), when counting is re-enabled.

Input "Strobe/Clear/Up_Down" acts as a clear or strobe where the clear is level sensitive (asynchronous clear) and the strobe is re-triggerable and rising edge sensitive. Interrupts can be generated on the rising edge of the strobe signal. If the "Strobe/Clear/Up_Down" input is configured as a clear or strobe input, the counter will always count UP. Otherwise, it can be defined as an UP/DOWN counter with the direction depending on the level of this input.

The following logic table shows how the counter behaves. The delay between a rising edge of input "A" and the counting lasts between one and two clock cycles, i.e. between 250 ns and 500 ns.

Table 7-2: Normal-Mode Signal/Gate Correlation

"A"	"B"	Counting Pulse
Signal = 	Gate active	"UP" or "DOWN"
Signal = 	Gate inactive	No Pulse

Quadrature Counter Mode

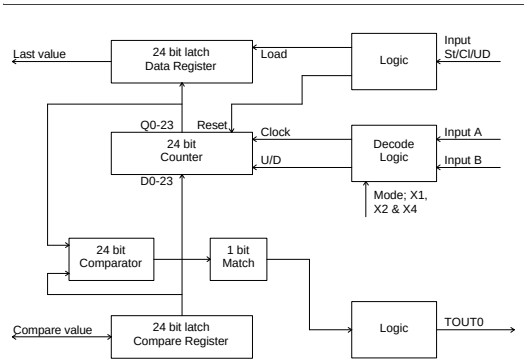
In this mode, the counter typically interfaces with a relative encoder. Inputs “A” and “B” are used to receive signals from the encoder with three counter resolutions (X1, X2 and X4) being available for selection. A third input, “Strobe/Clear/Up&Down”, may be used as either a strobe or a clear signal. The output line provides a match between the current counter value and the compare register.

Quadrature counter operation allows the encoder value to be read transparently and offers a continuous comparison between the encoder position and a previously set reference. In addition, output TOUT0 is activated when the two values match. The “Strobe/Clear/Up&Down” input may be used as either a strobe or a clear signal.

Interrupts can be used on the match output and the strobe signal.

Typically, the “A” and “B” inputs act as the count inputs from an encoder and the mode of counter operation is defined in the Control Register (see programming section). The delay between an edge of the inputs “A” or “B” and a counting pulse lasts between two and three clock cycles, i.e. between 500 ns and 750 ns.

Figure 7-8: SM-COUNT1 Quadrature-Mode Register Map



UP/DOWN counting depends on the phase of the two inputs “A” and “B” with the status being recorded in the Status Register.

To change the counting direction, simply interchange inputs “A” and “B”.

Input "Strobe/Clear/Up&Down" can act as a clear or strobe with the clear being level sensitive and the strobe is re-triggerable and rising edge sensitive and capable of generating an interrupt.

In addition to these features, a comparator is available on the module which activates a MATCH output and possibly an interrupt if the comparator and counter values are the same. The status of the match and comparison results are available in the Status Register (see programming section). Once a match has occurred, it may be cleared by accessing the Control Register.

The following tables show a correlation between the quadrature counter mode, the active edge (counting) and the direction of counting.

Table 7-3: Quadrature-Mode Signal/Gate Correlation (Resolution "X1")

"A"	"B"	Counting Pulse
Signal = 	Gate = "0"	"UP"
Signal = 	Gate = "1"	No Pulse.
Gate = "1"	Signal = 	
Gate = "0"	Signal = 	
Gate = "0"	Signal = 	"DOWN"
Gate = "1"	Signal = 	No Pulse.
Signal = 	Gate = "1"	
Signal = 	Gate = "0"	

Table 7-4: Quadrature-Mode Signal/Gate Correlation (Resolution “X2”)

“A”	“B”	Counting Pulse
Signal = 	Gate = “0”	“UP”
Gate = “0”	Signal = 	
Signal = 	Gate = “1”	No Pulse.
Gate = “1”	Signal = 	
Gate = “0”	Signal = 	“DOWN”
Signal = 	Gate = “0“	
Gate = “1”	Signal = 	No Pulse.
Signal = 	Gate = “1“	

Table 7-5: Quadrature-Mode Signal/Gate Correlation (Resolution "X4")

"A"	"B"	Counting Pulse
Signal = 	Gate = "0"	"UP"
Gate = "1"	Signal = 	
Gate = "0"	Signal = 	
Signal = 	Gate = "1"	
Gate = "0"	Signal = 	"DOWN"
Signal = 	Gate = "1"	
Signal = 	Gate = "0"	
Gate = "1"	Signal = 	

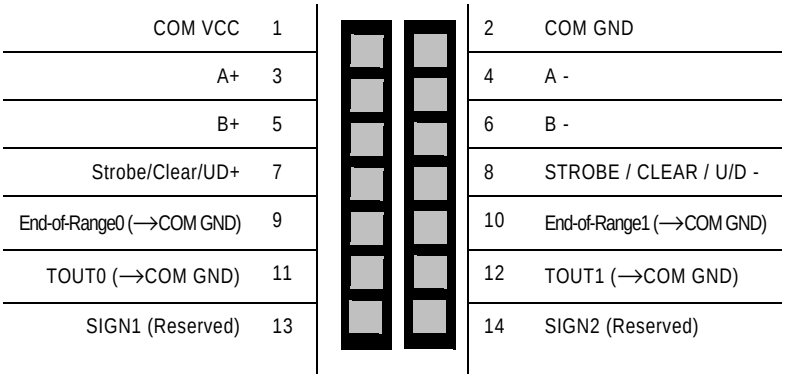
7.1.5.3 Board Operation

After power-up, all outputs are disabled and all registers are cleared. The counter is initialized to function in Normal Mode and is disabled.

7.1.6 Pinout

The following table shows the pinout/signal relationship for the SM-COUNT1 with reference to the module's screw terminal blocks.

Figure 7-9: Connector Pinout



7.1.7 ISaGRAF

Note:

For users of the ISaGRAF software it is not necessary to program this interface, since it is completely controlled by ISaGRAF. Therefore, no such programming is envisaged for the SM-COUNT1.

PEP Advantage

All necessary and/or useful information on ISaGRAF is available directly via software in the ISaGRAF on-line help program.

7.1.8 'C' Programming

7.1.8.1 SM-COUNT1 Library

The SM-COUNT1 library of functions `smartio.l` provides a convenient way of accessing the SM-COUNT1 module.

- All functions are written in ANSI C
- Prototypes are to be found in the file `smcount1.h`

Hardware Requirements

- SMART-BASE or SMART2-BASE main carrier unit with up to four SMART-EXT and/or SMART2-BASE carrier units used as extension units to the main carrier unit;
- SM-COUNT1 module

Software Requirements

One of the following compilers:

- Ultra C Version 1.3.1 or higher
- Fastrak 2.0.3 or higher



Note:

The leftmost (first) SM-Module of the SMART I/O or the SMART2 system has the number “0” assigned to it, as far as programming is concerned.

Please note that on a SMART2-BASE this slot is reserved for the CPU module.

Before a library function can be used, the function `SMCNT1Init()` must be called first. This allocates the requested resources. It must be noted that this library is not designed for multi-tasking. Furthermore, this function needs to be called for each SM-COUNT1 module called within the task. Upon completion of the application, the function `SMCNT1DeInit()` needs to be called for every single SM-COUNT1 Module that has been initialized.

In order to illustrate the use of the SM-COUNT1 library, an application example called `democnt1.c` can be found in the SMART application directory (normally found in `/<device>/APPLIC/SMART/SRC)`.



Note:

If the same SM-COUNT1 module is to be accessed from different tasks, please ensure the actual output status by using for example OS-9 semaphores.

7.1.8.2 SMCNT1Init()

Syntax

```
error_code SMCNT1Init(u_int8 PortNr)
```

Description

This function initializes the SM-COUNT1 module on port `PortNr`, and

- tests the module's ID;
- initializes the SMART hardware for the access of the attached SM module (if not already done);
- allocates internal resources.

Input

`u_int8 PortNr`	Module port number to be initialized

Output

`error_code`	`SUCCESS`	or
	`E_BMODE`	if wrong or no module is found in the slot
	`E_KWNMOD`	module is already initialized
	or standard OS-9 error code (please refer to the OS-9 Technical Manual's Error Codes section)	

Example

```
RetVal = SMCNT1Init(1);
```

7.1.8.3 SMCNT1DeInit()

Syntax

```
error_code SMCNT1DeInit(u_int8 PortNr)
```

Description

This function de-initializes the SM-COUNT1 module on the port *PortNr* by

- setting it to a defined state;
- freeing the resources tied up with it.

Input

`u_int8 PortNr`	Module port number to be de-initialized

Output

`error_code`	`SUCCESS`	or
	`E_SEEK`	module not initialized
	or standard OS-9 error code (please refer to the OS-9 Technical Manual's Error Codes section)	

Example

```
RetVal = SMCNT1DeInit(1);
```

7.1.8.4 SMCNT1SetMode()

Syntax

```
error_code SMCNT1SetMode(u_int8 PortNr, u_int8 Mode)
```

Description

This function sets the operation mode of the SM-COUNT1 on the specified port `PortNr`, and:

- sets the GATE polarity;
- defines the use of the input lines.

The following predefined settings are available:

`SMCNT1_NM`	Normal mode
`SMCNT1_X1`	X1 Quadrature mode
`SMCNT1_X2`	X2 Quadrature mode
`SMCNT1_X4`	X4 Quadrature mode
`SMCNT1_G_AL`	Gate Polarity (B input) active low (normal mode only)
`SMCNT1_G_AH`	Gate Polarity (B input) active high (normal mode only)
`SMCNT1_IN_CLR`	St/Cl/UD input line is used as clear input
`SMCNT1_IN_STR`	St/Cl/UD input line is used as strobe input
`SMCNT1_IN_UD`	St/Cl/UD input line is used as UD indicator input (normal mode only)

These definitions can be found in the SM-COUNT1 specific header file `smcount1.h`.



Note:

This function writes 0 to the lowest 2 bits of the control register 0. This implies:
- match output is untouched;
- counter is NOT cleared.

For more details about modes and input lines refer to the hardware section of the SM-COUNT1 chapter.

Input

`u_int8 PortNr`	Module port number
`u_int8 Mode`	The above listed predefined settings are available

Output

`error_code`	`SUCCESS`	or
	`E_SEEK`	module not initialized
	`E_BPADDR`	wrong register page
	or standard OS-9 error code (please refer to the OS-9 Technical Manual's Error Codes section)	

Example

```
RetVal = SMCNT1SetMode(1, SMCNT1_G_AL+SMCNT1_IN_STR
+SMCNT1_NM);
```

7.1.8.5 SMCNT1GetMode()

Syntax

```
error_code SMCNT1GetMode(u_int8 PortNr,
u_int8 *pMode)
```

Description

This function reads and interpretes the current mode of the SM-COUNT1 and stores the result in the variable that `pMode` points to.

The following are predefined macros to analyze the mode setting available:

`SMCNT1_NM`	Normal mode
`SMCNT1_X1`	X1 Quadrature mode
`SMCNT1_X2`	X2 Quadrature mode
`SMCNT1_X4`	X4 Quadrature mode
`SMCNT1_G_AL`	Gate Polarity (B input) active low (normal mode only)
`SMCNT1_G_AH`	Gate Polarity (B input) active high (normal mode only)
`SMCNT1_IN_CLR`	St/Cl/UD input line is used as clear input
`SMCNT1_IN_STR`	St/Cl/UD input line is used as strobe input
`SMCNT1_IN_UD`	St/Cl/UD input line is used as UD indicator input (normal mode only)

These definitions are made in the SM-COUNT1 specific header file `smcount1.h`.

Input

`u_int8 PortNr`	Module port number
`u_int8 *pMode`	Pointer to the location where the function stores the mode setting.

Output

`error_code`	`SUCCESS`	or
	`E_SEEK`	module not initialized
	`E_BPADDR`	wrong register page
	or standard OS-9 error code (please refer to the OS-9 Technical Manual's Error Codes section)	

Example

```
u_int8 Mode;
...
RetVal = SMCNT1GetMode(1, &Mode);
```

7.1.8.6 SMCNT1SetComp()

Syntax

```
error_code SMCNT1SetComp(u_int8 PortNr,
u_int32 CompVal)
```

Description

This function sets the 24 bit compare register of the SM-COUNT1 with the 24 last significant bits of `CompVal`.

Input

`u_int8 PortNr`	Module port number
`u_int32 CompVal`	New value of the SM-COUNT1 compare register.

Output

`error_code`	`SUCCESS`	or
	`E_SEEK`	module not initialized
	`E_BPADDR`	wrong register page
	or standard OS-9 error code (please refer to the OS-9 Technical Manual's Error Codes section)	

Example

```
RetVal = SMCNT1SetComp(1, 4242);
```

7.1.8.7 SMCNT1GetComp()

Syntax

```
error_code SMCNT1GetComp(u_int8 PortNr,
u_int32 *pCompVal)
```

Description

This function reads the 24 bit value of the SM-COUNT1 compare register and stores it at the location `pCompVal` points to.

Input

`u_int8 PortNr`	Module port number
`u_int32 *pCompVal`	Pointer to the location where the function stores the compare register contents.

Output

`error_code`	`SUCCESS`	or
	`E_SEEK`	module not initialized
	`E_BPADDR`	wrong register page
	or standard OS-9 error code (please refer to the OS-9 Technical Manual's Error Codes section)	

Example

```
u_int32 CompVal;
...
RetVal = SMCNT1GetComp(1, &CompVal);
```

7.1.8.8 SMCNT1GetCounter()

Syntax

```
error_code SMCNT1GetCounter(u_int8 PortNr,
u_int32 *pCounterVal)
```

Description

This function reads the 24 bit value of the SM-COUNT1 counter register and stores it at the location `pCounterVal` points to.



Note:

In the case of a strobe the counter is latched and will be read with the next counter access. Otherwise the actual counter value is read.

Input

`u_int8 PortNr`	Module port number
`u_int32 *pCounterVal`	Pointer to the location where the function stores the counter register contents.

Output

`error_code`	`SUCCESS`	or
	`E_SEEK`	module not initialized
	`E_BPADDR`	wrong register page
	or standard OS-9 error code (please refer to the OS-9 Technical Manual's Error Codes section)	

Example

```
u_int32 CounterVal;
...
RetVal = SMCNT1GetCounter(1, &CounterVal);
```

7.1.8.9 SMCNT1Clr()

Syntax

```
error_code SMCNT1Clr(u_int8 PortNr, u_int16 ClrCode)
```

Description

With this function the match output and the counter register can be cleared.

The following predefined macros are available:

`SMCNT1_CLR_MATCH_OUT`

`SMCNT1_CLR_CNT`

These definitions are made in the SM-COUNT1 specific header file `smcount1.h`.

Input

`u_int8 PortNr`	Module port number
`u_int16 ClrCode`	Bits to indicate events to clear.

Output

`error_code`	`SUCCESS`	or
	`E_BMODE`	illegal setting selected
	`E_SEEK`	module not initialized
	`E_BPADDR`	wrong register page
	or standard OS-9 error code (please refer to the OS-9 Technical Manual's Error Codes section)	

Example

```
RetVal = SMCNT1Clr(1, SMCNT1_CLR_MATCH_OUT
+SMCNT1_CLR_CNT);
```

7.1.8.10 SMCNT1SetTOUT1()

Syntax

```
error_code SMCNT1SetTOUT1(u_int8 PortNr)
```

Description

This function switches the output TOUT1 on.

For more information about the output TOUT1 refer to the hardware section of the SM-COUNT1 manual.

Input

`u_int8 PortNr`	Module port number

Output

`error_code`	`SUCCESS`	or
	`E_SEEK`	module not initialized
	`E_BPADDR`	wrong register page
	or standard OS-9 error code (please refer to the OS-9 Technical Manual's Error Codes section)	

Example

```
RetVal = SMCNT1SetTOUT1(1);
```

7.1.8.11 SMCNT1ClrTOUT1()

Syntax

```
error_code SMCNT1ClrTOUT1(u_int8 PortNr)
```

Description

This function switches the output TOUT1 off.

For more information about the output TOUT1 refer to the hardware section of the SM-COUNT1 manual.

Input

`u_int8 PortNr`	Module port number

Output

`error_code`	`SUCCESS`	or
	`E_SEEK`	module not initialized
	`E_BPADDR`	wrong register page
	or standard OS-9 error code (please refer to the OS-9 Technical Manual's Error Codes section)	

Example

```
RetVal = SMCNT1ClrTOUT1(1);
```

7.1.8.12 SMCNT1Start()

Syntax

```
error_code SMCNT1Start(u_int8 PortNr)
```

Description

This function starts counting on the SM-COUNT1.

Input

`u_int8 PortNr`	Module port number

Output

`error_code`	`SUCCESS`	or
	`E_SEEK`	module not initialized
	`E_BPADDR`	wrong register page
	or standard OS-9 error code (please refer to the OS-9 Technical Manual's Error Codes section)	

Example

```
RetVal = SMCNT1Start(1);
```

7.1.8.13 SMCNT1Stop()

Syntax

```
error_code SMCNT1Stop(u_int8 PortNr)
```

Description

This function stops counting on the SM-COUNT1.

Input

`u_int8 PortNr`	Module port number	

Output

`error_code`	`SUCCESS`	or
	`E_SEEK`	module not initialized
	`E_BPADDR`	wrong register page
	or standard OS-9 error code (please refer to the OS-9 Technical Manual's Error Codes section)	

Example

```
RetVal = SMCNT1Stop(1);
```

7.1.8.14 SMCNT1EnDisInt()

Syntax

```
error_code SMCNT1EnDisInt(u_int8 PortNr,
u_int16 IntCtrl)
```

Description

This function enables and/or disables interrupt sources of the SM-COUNT1.

There are predefined settings for each interrupt source to enable:

SMCNT1_INT_MATCH	Enables match interrupt
SMCNT1_INT_GATE	Enables gate interrupt
SMCNT1_INT_STROBE	Enables strobe interrupt
SMCNT1_INT_EOR0	Enables end-of-range-0 interrupt
SMCNT1_INT_EOR1	Enables end-of-range-1 interrupt

All interrupts that are not enabled are automatically disabled.

These definitions are made in the SM-COUNT1 specific header file `smcount1.h`.



Note:

Enabling interrupts does NOT mean that they are routed to the SMART CPU. This function enables only the possibility to monitor some hardware events by reading the interrupt status register.

Input

`u_int8 PortNr`	Module port number
`u_int16 IntCtrl`	Control word to enable/disable interrupt sources of the SM-COUNT1

Output

`error_code`	`SUCCESS`	or
	`E_SEEK`	module not initialized
	`E_BPADDR`	wrong register page
	or standard OS-9 error code (please refer to the OS-9 Technical Manual's Error Codes section)	

Examples

These examples relate to:

- enabling match and gate interrupts and
- disabling of strobe and the end-of-range interrupts

```
RetVal = SMCNT1EnDisInt (1,
SMCNT1_INT_MATCH+SMCNT1_INT_GATE);
```

7.1.8.15 SMCNT1IntStatus()

Syntax

```
error_code SMCNT1IntStatus(u_int8 PortNr,
u_int16 *pStatus)
```

Description

This function reads the status of the interrupt sources of the SM-COUNT1 and stores the results in a variable that `pStatus` points to.

There are predefined settings for each interrupt source:

`SMCNT1_INT_MATCH`	Causes match interrupt
`SMCNT1_INT_GATE`	Causes gate interrupt
`SMCNT1_INT_STROBE`	Causes strobe interrupt
`SMCNT1_INT_EOR0`	Causes end-of-range-0 interrupt
`SMCNT1_INT_EOR1`	Causes end-of-range-1 interrupt

These definitions are made in the SM-COUNT1 specific header file `smcount1.h`.

For more details about the interrupt sources of the SM-COUNT1 refer to the hardware section of the manual.



Note:

The interrupt bits of the SM-COUNT1 will only be set if they are enabled. To enable an interrupt, please use the following function: `SMCNT1EnDisInt()`.

Input

`u_int8 PortNr`	Module port number
`u_int16 *pStatus`	Pointer to the location where the function stores the interrupt status

Output

`error_code`	`SUCCESS`	or
	`E_SEEK`	module not initialized
	`E_BPADDR`	wrong register page
	or standard OS-9 error code (please refer to the OS-9 Technical Manual's Error Codes section)	

Example

```
u_int16 Status;
...
RetVal = SMCNT1IntStatus(1, &Status);
```

7.1.8.16 SMCNT1ClrInt()

Syntax

```
error_code SMCNT1ClrInt(u_int8 PortNr, u_int16 ClrInt)
```

Description

This function clears the interrupt bits of the SM-COUNT1. The bits are defined by the variable `ClrInt`, by setting the corresponding bit.

The following predefined macros are available to clear interrupts:

`SMCNT1_INT_MATCH`	clears the match interrupt
`SMCNT1_INT_GATE`	clears the gate interrupt
`SMCNT1_INT_STROBE`	clears the strobe interrupt
`SMCNT1_INT_EOR0`	clears the end of range 0 interrupt
`SMCNT1_INT_EOR1`	clears the end of range 1 interrupt

These definitions are made in the SM-COUNT1 specific header file `smcount1.h`.

Input

`u_int8 PortNr`	Module port number
`u_int16 ClrInt`	Interrupt bits to be cleared

Output

error_code	SUCCESS	or
	E_BMODE	illegal setting selected
	E_SEEK	module not initialized
	E_BPADDR	wrong register page
	or standard OS-9 error code (please refer to the OS-9 Technical Manual's Error Codes section)	

Example

```
RetVal = SMCNT1ClrInt(1, SMCNT1_INT_MATCH
+SMCNT1_INT_STROBE);
```

7.1.8.17 SMCNT1Status()

Syntax

```
error_code SMCNT1Status(u_int8 PortNr,
u_int16 *pStatus)
```

Description

This function reads the status of the SM-COUNT1 and stores the result in the variable that `pStatus` points to.

There are predefined settings for each status bit:

`SMCNT1_STAT_MATCH`	status of match output
`SMCNT1_STAT_COMP`	counter equals comparator
`SMCNT1_STAT_UD`	direction of counting
`SMCNT1_STAT_EOR0`	status of end of range 0 input
`SMCNT1_STAT_EOR1`	status of end of range 1 input

These definitions are made in the SM-COUNT1 specific header file `smcount1.h`.

Input

`u_int8 PortNr`	Module port number
`u_int16 *pStatus`	Pointer to the location where the function stores the board status

Output

error_code	SUCCESS	or
	E_BMODE	illegal setting selected
	E_SEEK	module not initialized
	E_BPADDR	wrong register page
	or standard OS-9 error code (please refer to the OS-9 Technical Manual's Error Codes section)	

Example

```
u_int16 Status;
...
RetVal = SMCNT1Status(1, &Status);
```

7.2 SM-COUNT2

7.2.1 Introduction

The SM-COUNT2 is an opto-isolated counter module for general purpose usage in a SMART system. It includes two “up” counters with 16-bit resolution and provides galvanic isolation from the process interface. In addition to the two inputs for each counter, there is an optional “common clear” input with interrupt capability.

7.2.2 Specifications

Table 7-6: SM-COUNT2 Technical Specifications

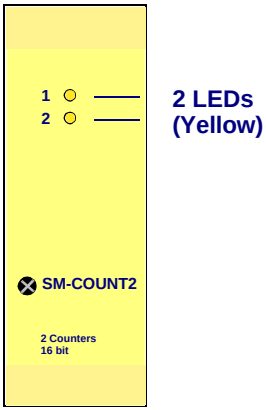
SM-COUNT2	Specifications
Inputs	• 3 differential inputs: “TIN0”, “TIN1”, “COM CL” • 2 gate inputs (relative to COM GND): “GATE0”, “GATE1”
Input Specifications: • Voltage Ranges • Current • Frequency • Switching	 5 V DC ± 5%, 12 V DC ± 25%, 24 V DC ± 25% 10 mA < 300 kHz, embedded filter 1.55 V/6.7 V/15.1 V respectively
Outputs	• 1 digital output per counter • 1 output for general purposes (32-bit mode only)
Output Specifications: • Voltage • Current • Frequency • Switching	 24 V DC ±25% 100 mA, inductive-load protection < 100 kHz (duty cycle = 1:1) High-side switch
Isolation (I/O to system)	2.5 kV AC

Table 7-6: SM-COUNT2 Technical Specifications

SM-COUNT2	Specifications
Power Consumption	350 mW (min.), 370 mW (max.)
Bus Interface	SPI
ID Byte	$71
Temperature Range: Industrial Extended	 0°C to +70°C -40°C to +85°C
Operating Humidity	10% to 95%, non-condensing.
Weight	48 g

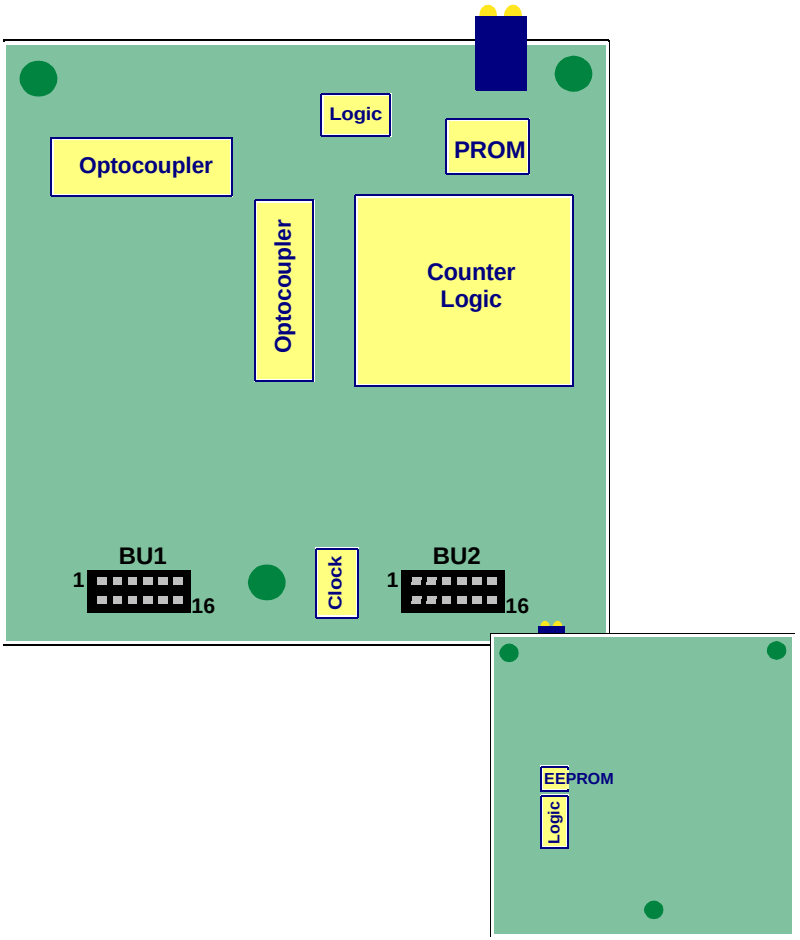
7.2.3 Front Panel Layout

Figure 7-10: SM-COUNT2 Front Panel Layout



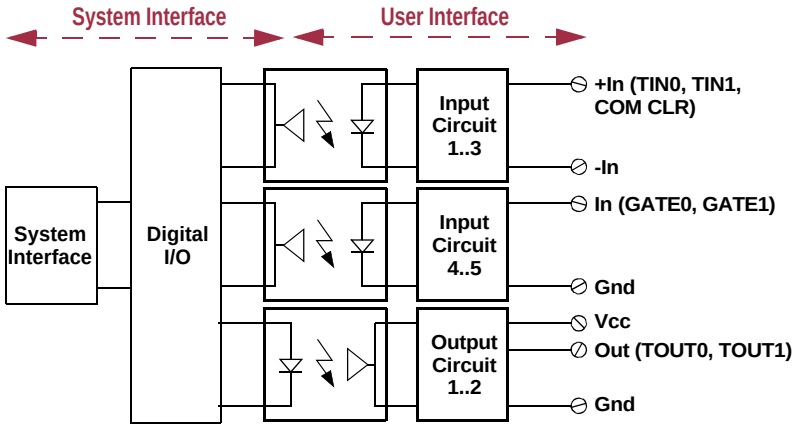
7.2.4 Board Overview

Figure 7-11: SM-COUNT2 Schematic Board Layout — Front View and (Reduced) Reverse View



7.2.5 Functional Description

Figure 7-12: SM-COUNT2 Functional Description



7.2.5.1 Input/Output Functions

Differential Inputs/Connection Modes

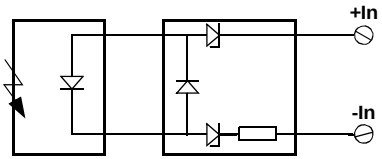
The SM-COUNT2 counter module supports two 16-bit counters. There is the possibility of cascading both counters internally, thus giving place to the equivalent of a 32-bit counter. Therefore, the following counter connection modes can be selected via software:

- Parallel connection mode
- Cascaded connection mode

In the parallel connection mode the differential inputs are TIN0, TIN1 and COM CL. The input line TIN_ is used to receive counter pulses, the common input COM CL may be used as a clear signal.

In the cascaded connection mode, the counting channel (channel “0”) acts as a 32-bit counter with three input lines, TIN0, GATE0 and Common Clear (COM CL) and one output line, TOUT0. The status of the input lines TIN1 and GATE1 can be detected by reading the register of the status channel (channel “1”).

Figure 7-13: SM-COUNT2 Differential Input Circuits



Each channel allows the counter value to be read transparently and offers a continuous comparison between the current value and a previously set reference (compare register). In addition the output (TOUT_) is activated when the two values match.

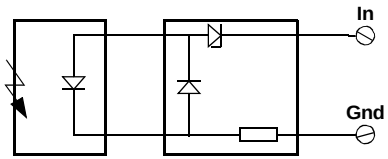
The COM CL input can be enabled for each counter via software. If enabled and active, this level sensitive input will clear the counter value.

Interrupts can be enabled on a match (counter value equal to the compare register) and on the rising edge of the common clear input.

Gate Inputs

Two gate inputs, GATE0 and GATE1 are available for the user. The status of these inputs is given by the relating status register.

Figure 7-14: SM-COUNT2 Gate Input Circuits

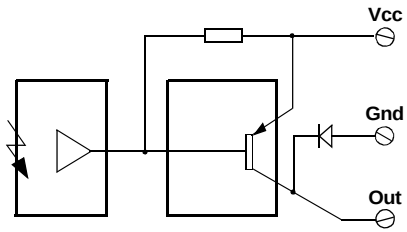


The GATE_ input may be enabled or disabled via software in the control register. If enabled, counting refers to the rising edges of the TIN_ input only, if the GATE_ input is active.

Mode-Dependent Output

This output defines specific counter events, which depend on the currently set counter mode. For example, a 1:10 pulse threshold may be set to break down a 100 kHz frequency into 10 kHz counter steps.

Figure 7-15: SM-COUNT2 Output Circuits



General Purpose Output

In the cascaded connection mode the output line TOUT1 acts as a general purpose output. Its status is controlled by the control register.

7.2.5.2 Counter Modes

Independent of the mode, whether parallel or cascaded, that the two counters are connected in, there are two different counter modes relating to the enabling/disabling of the gate input:

1. Counting with gate input
2. Counting without gate input

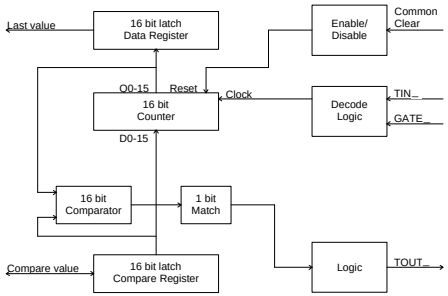


Figure 7-16:SM-COUNT2 Counter Logic

Counting without Gate Input

Every rising edge of the input TIN_ is counted by the counter, regardless of the current state of input GATE_. The output TOUT_ is set whenever the counter value (data register) equals the compare value (compare register).

Table 7-7: Counting without Gate Input

TIN0/TIN1	GATE0/GATE1	Counting Pulse
Signal = 	Gate = “1”	“UP”
Signal = 	Gate = “0”	“UP”

The above logic table shows the counter behaviour. The counting occurs between 250ns to 500ns later than the rising edge of the input TIN_.

Counting with Gate Input

In this counting mode, the input signals are counted only if the gate is active. Otherwise the pulses are not counted.

Table 7-8: Counting with Gate Input

TIN0/TIN1	GATE0/GATE1	Counting Pulse
Signal = 	Gate = “0”	No Pulse.
Signal = 	Gate = “1”	“UP”

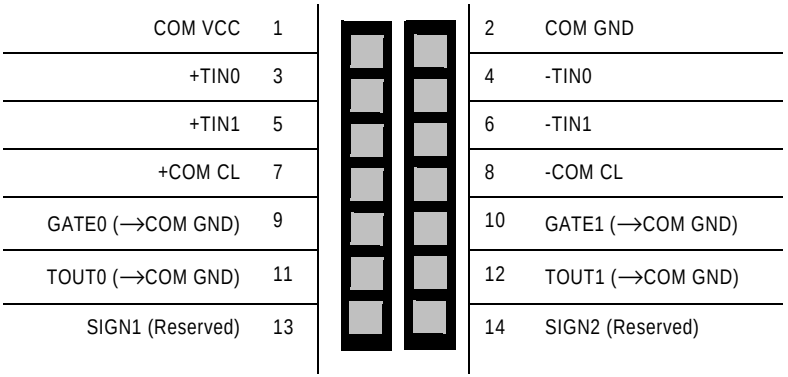
7.2.5.3 Board Operation

After power-up, all outputs are passive and all registers are cleared. The counters are initialized to function in parallel and with the Gate Input Mode being disabled.

7.2.6 Pinout

The following table shows the pinout/signal relationship for the SM-COUNT2 with reference to the module's screw terminal blocks.

Figure 7-17: Connector Pinout



7.2.7 ISaGRAF



Note:

For users of the ISaGRAF software it is not necessary to program this interface, since it is handled completely by ISaGRAF. Therefore, no such programming is envisaged for the SM-COUNT2.



PEP Advantage

All necessary and/or useful information on ISaGRAF is available directly via software in the ISaGRAF on-line help program.

7.2.8 'C' Programming

7.2.8.1 SM-COUNT2 Library

The SM-COUNT2 library of functions provides a convenient way of accessing the SM-COUNT2 module.

- All functions are written in ANSI C
- Prototypes are to be found in the file `smcount2.h`

Hardware Requirements

- SMART-BASE or SMART2-BASE main carrier unit with up to four SMART-EXT and/or SMART2-BASE carrier units used as extension units to the main carrier unit;
- SM-COUNT2 module

Software Requirements

One of the following compilers:

- Ultra C Version 1.3.1 or higher
- Fastrak 2.0.3 or higher



Note:

The leftmost (first) SM-Module of the SMART I/O or the SMART2 system has the number “0” assigned to it, as far as programming is concerned.

Please note that on a SMART2-BASE this slot is reserved for the CPU module.

Before a library function can be used, the function `SMCNT2Init()` must be called first. This allocates the requested resources. It must be noted that this library is not designed for multi-tasking. Furthermore, this function needs to be called for each SM-COUNT2 module called within the task. Upon completion of the application, the function `SMCNT2DeInit()` needs to be called for every single SM-COUNT2 Module that has been initialized.

In order to illustrate the use of the SM-COUNT2 library, an application example called `democnt2.c` can be found in the SMART application directory (normally found in `/<device>/APPLIC/SMART/SRC`).



Note:

If the same SM-COUNT2 module is to be accessed from different tasks, please ensure the actual output status by using for example OS-9 semaphores.

7.2.8.2 SMCNT2Init()

Syntax

```
error_code SMCNT2Init(u_int8 PortNr)
```

Description

This function initializes the SM-COUNT2 module on port `PortNr`, and

- tests the module's ID;
- initializes the SMART hardware for the access of the attached SM module (if not already done);
- allocates internal resources.

Input

`u_int8 PortNr`	Module port number to be initialized

Output

`error_code`	`SUCCESS`	or
	`E_BMODE`	if wrong or no module is found in the slot
	`E_KWNMOD`	module is already initialized
	or standard OS-9 error code (please refer to the OS-9 Technical Manual's Error Codes section)	

Example

```
RetVal = SMCNT2Init(1);
```

7.2.8.3 SMCNT2DeInit()

Syntax

```
error_code SMCNT2DeInit(u_int8 PortNr)
```

Description

This function de-initializes the SM-COUNT2 module on the port `PortNr` by

- setting it to a defined state;
- freeing the resources tied up with it.

Input

`u_int8 PortNr`	Module port number to be de-initialized

Output

`error_code`	`SUCCESS`	or
	`E_SEEK`	module not initialized
	or standard OS-9 error code (please refer to the OS-9 Technical Manual's Error Codes section)	

Example

```
RetVal = SMCNT2DeInit(1);
```

7.2.8.4 SMCNT2SetMode()

Syntax

```
error_code SMCNT2SetMode(u_int8 PortNr, u_int8 Mode,
u_int8 Channel)
```

Description

This function sets the operation mode of the SM-COUNT2 on the specified port `PortNr`.

The following predefined settings are available:

`SMCNT2_C_CLR`	Common clear input enabled
`SMCNT2_EN_GATE`	Gate function enabled
`SMCNT2_MODE_32`	32-bit mode (channel “0” only) enabled

These definitions can be found in the SM-COUNT2 specific header file `smcount2.h`.



Note:

This function concerns the above mentioned bits only, i.e.
- start/stop bit not concerned;
- match output not concerned;
- counter not cleared;
- interrupt enable bits not concerned.

For more details about modes and input lines refer to the hardware section of the SM-COUNT2 chapter.

Input

`u_int8 PortNr`	Module port number
`u_int8 Mode`	The above listed predefined settings are available
`u_int8 Channel`	Channel number (“0” or “1”)

Output

`error_code`	`SUCCESS`	or
	`E_SEEK`	module not initialized
	`E_BPADDR`	wrong register page
	`E_ILLPRM`	wrong channel number
	`E_BMODE`	attempt to set mode for channel “1” despite 32-bit counter is enabled
	or standard OS-9 error code (please refer to the OS-9 Technical Manual’s Error Codes section)	

Example

```
RetVal = SMCNT2SetMode(1, SMCNT2_C_CLR, 0);
```

7.2.8.5 SMCNT2GetMode()

Syntax

```
error_code SMCNT2GetMode(u_int8 PortNr,
u_int8 *pMode, u_int8 Channel)
```

Description

This function reads and interpretes the current mode of the SM-COUNT2 and stores the result in the variable that pMode points to.

The following are predefined macros to analyze the mode setting available:

SMCNT2_START	Counter running
SMCNT2_C_CLR	Common clear input enabled
SMCNT2_EN_GATE	Gate function enabled
SMCNT2_EN_MATCH_INT	Interrupt on comparison match enabled
SMCNT2_EN_CCLR_INT	Interrupt on common clear enabled
SMCNT2_MODE_32	32-bit mode (channel “0” only) enabled

These definitions are made in the SM-COUNT2 specific header file smcount2.h.

Input

u_int8 PortNr	Module port number
u_int8 *pMode	Pointer to the location where the function stores the mode setting.
u_int8 Channel	Channel number (“0” or “1”)

Output

error_code	SUCCESS	or
	E_SEEK	module not initialized
	E_BPADDR	wrong register page
	E_ILLPRM	wrong channel number
	E_BMODE	attempt to retrieve mode for channel “1” despite 32-bit counter is enabled
	or standard OS-9 error code (please refer to the OS-9 Technical Manual’s Error Codes section)	

Example

```
u_int8 Mode;
...
RetVal = SMCNT2GetMode(1, &Mode, 0);
```

7.2.8.6 SMCNT2SetComp()

Syntax

```
error_code SMCNT2SetComp(u_int8 PortNr,
u_int32 CompVal, u_int8 Channel)
```

Description

This function sets the 16/32-bit compare register of the SM-COUNT2.



Note:

In 16-bit mode the last 16 significant bits of `CompVal` are valid, in 32-bit mode the channel number is ignored.

Input

`u_int8 PortNr`	Module port number
`u_int32 CompVal`	New value of the SM-COUNT2 compare register.
`u_int8 Channel`	Channel number (“0” or “1”)

Output

`error_code`	`SUCCESS`	or
	`E_SEEK`	module not initialized
	`E_BPADDR`	wrong register page
	or standard OS-9 error code (please refer to the OS-9 Technical Manual’s Error Codes section)	

Example

```
RetVal = SMCNT2SetComp(1, 4242, 0);
```

7.2.8.7 SMCNT2GetComp()

Syntax

```
error_code SMCNT2GetComp(u_int8 PortNr,
u_int32 *pCompVal, u_int8 Channel)
```

Description

This function reads the 16/32-bit value of the SM-COUNT2 compare register depending on the mode selected, and stores it at the location `pCompVal` points to.



Note:

In 16-bit mode the last 16 significant bits of `CompVal` are valid, in 32-bit mode the channel number is ignored.

Input

`u_int8 PortNr`	Module port number
`u_int32 *pCompVal`	Pointer to the location where the function stores the compare register contents.
`u_int8 Channel`	Channel number (“0” or “1”)

Output

`error_code`	`SUCCESS`	or
	`E_SEEK`	module not initialized
	`E_BPADDR`	wrong register page
	or standard OS-9 error code (please refer to the OS-9 Technical Manual’s Error Codes section)	

Example

```
u_int32 CompVal;
...
RetVal = SMCNT2GetComp(1, &CompVal, 0);
```

7.2.8.8 SMCNT2GetCounter()

Syntax

```
error_code SMCNT2GetCounter(u_int8 PortNr,
u_int32 *pCounterVal, u_int8 Channel)
```

Description

This function reads the 16/32-bit value of the SM-COUNT2 counter register depending on the mode selected, and stores it at the location `pCounterVal` points to.



Note:

In 16-bit mode the last 16 significant bits of `CounterVal` are valid, in 32-bit mode the channel number is ignored.

Input

`u_int8 PortNr`	Module port number
`u_int32 *pCounterVal`	Pointer to the location where the function stores the counter register contents.
`u_int8 Channel`	Channel number (“0” or “1”)

Output

`error_code`	`SUCCESS`	or
	`E_SEEK`	module not initialized
	`E_BPADDR`	wrong register page
	or standard OS-9 error code (please refer to the OS-9 Technical Manual’s Error Codes section)	

Example

```
u_int32 CounterVal;
...
RetVal = SMCNT2GetCounter(1, &CounterVal, 0);
```

7.2.8.9 SMCNT2Clr()

Syntax

```
error_code SMCNT2Clr(u_int8 PortNr, u_int16 ClrCode,
u_int8 Channel)
```

Description

With this function the match output and the counter register can be cleared.

The following predefined macros are available:

`SMCNT2_CLRM`	Clears match output
`SMCNT2_RST`	Clears counter

These definitions are made in the SM-COUNT2 specific header file `smcount2.h`.



Note:

In 32-bit mode the channel number is ignored.

Input

`u_int8 PortNr`	Module port number
`u_int16 ClrCode`	Bits to indicate events to clear.
`u_int8 Channel`	Channel number (“0” or “1”)

Output

`error_code`	`SUCCESS`	or
	`E_BMODE`	illegal setting selected
	`E_SEEK`	module not initialized
	`E_BPADDR`	wrong register page
	or standard OS-9 error code (please refer to the OS-9 Technical Manual’s Error Codes section)	

Example

```
RetVal = SMCNT2Clr(1, SMCNT2_CLRM
+SMCNT2_RST, 0);
```

7.2.8.10 SMCNT2SetTOUT1()

Syntax

```
error_code SMCNT2SetTOUT1(u_int8 PortNr)
```

Description

This function switches the output TOUT1 on.

For more information about the output TOUT1 refer to the hardware section of the SM-COUNT2 manual.



Note:

TOUT1 can be set in 32-bit counter mode only.

Input

`u_int8 PortNr`	Module port number

Output

`error_code`	`SUCCESS`	or
	`E_SEEK`	module not initialized
	`E_BPADDR`	wrong register page
	or standard OS-9 error code (please refer to the OS-9 Technical Manual's Error Codes section)	

Example

```
RetVal = SMCNT2SetTOUT1(1);
```

7.2.8.11 SMCNT2ClrTOUT1()

Syntax

```
error_code SMCNT2ClrTOUT1(u_int8 PortNr)
```

Description

This function switches the output TOUT1 off.

For more information about the output TOUT1 refer to the hardware section of the SM-COUNT2 manual.



Note:

TOUT1 can be cleared in 32-bit counter mode only.

Input

`u_int8 PortNr`	Module port number

Output

`error_code`	`SUCCESS`	or
	`E_SEEK`	module not initialized
	`E_BPADDR`	wrong register page
	or standard OS-9 error code (please refer to the OS-9 Technical Manual's Error Codes section)	

Example

```
RetVal = SMCNT2ClrTOUT1(1);
```

7.2.8.12 SMCNT2Start()

Syntax

```
error_code SMCNT2Start(u_int8 PortNr, u_int8 Channel)
```

Description

This function starts counting on the SM-COUNT2.



Note:

In 32-bit mode the channel number is ignored.

Input

`u_int8 PortNr`	Module port number
`u_int8 Channel`	Channel number (“0” or “1”)

Output

`error_code`	`SUCCESS`	or
	`E_SEEK`	module not initialized
	`E_BPADDR`	wrong register page
	or standard OS-9 error code (please refer to the OS-9 Technical Manual’s Error Codes section)	

Example

```
RetVal = SMCNT2Start(1, 0);
```

7.2.8.13 SMCNT2Stop()

Syntax

```
error_code SMCNT2Stop(u_int8 PortNr, u_int8 Channel)
```

Description

This function stops counting on the SM-COUNT2.



Note:

In 32-bit mode the channel number is ignored.

Input

`u_int8 PortNr`	Module port number
`u_int8 Channel`	Channel number (“0” or “1”)

Output

`error_code`	`SUCCESS`	or
	`E_SEEK`	module not initialized
	`E_BPADDR`	wrong register page
	or standard OS-9 error code (please refer to the OS-9 Technical Manual’s Error Codes section)	

Example

```
RetVal = SMCNT2Stop(1, 0);
```

7.2.8.14 SMCNT2EnDisInt()

Syntax

```
error_code SMCNT2EnDisInt(u_int8 PortNr,
u_int16 IntCtrl, u_int8 Channel)
```

Description

This function enables and/or disables the interrupt sources of the SM-COUNT2.

There are predefined settings for each interrupt source to enable:

SMCNT2_EN_MATCH_INT	Enables match interrupt
SMCNT2_EN_CCLR_INT	Enables common clear interrupt

All interrupts that are not enabled are automatically disabled.

These definitions are made in the SM-COUNT2 specific header file `smcount2.h`.

Note:

Enabling interrupts does NOT mean that they are routed to the SMART CPU. This function enables only the possibility to monitor some hardware events by reading the interrupt status register.

Note:

In 32-bit mode the channel number is ignored.

Input

`u_int8 PortNr`	Module port number
`u_int16 IntCtrl`	Control word to enable/disable interrupt source of the SM-COUNT2
`u_int8 Channel`	Channel number (“0” or “1”)

Output

`error_code`	`SUCCESS`	or
	`E_BMODE`	illegal setting selected
	`E_SEEK`	module not initialized
	`E_BPADDR`	wrong register page
	`E_ILLPRM`	wrong channel number
	or standard OS-9 error code (please refer to the OS-9 Technical Manual’s Error Codes section)	

Example

```
RetVal = SMCNT2EnDisInt (1,
SMCNT2_EN_MATCH_INT+SMCNT2_EN_CCLR_INT, 0);
```

7.2.8.15 SMCNT2IntStatus()

Syntax

```
error_code SMCNT2IntStatus(u_int8 PortNr,
u_int16 *pStatus, u_int8 Channel)
```

Description

This function reads the status of the interrupt sources of the SM-COUNT2 and stores the results in a variable that `pStatus` points to.

There are predefined settings for each interrupt source:

`SMCNT2_INT_MATCH`	Match interrupt occurred
`SMCNT2_INT_CCLR`	Common clear interrupt occurred

These definitions are made in the SM-COUNT2 specific header file `smcount2.h`.

For more details about the interrupt sources of the SM-COUNT2 refer to the hardware section of the manual.

Note:

The interrupt bits of the SM-COUNT2 will only be set if they are enabled. To enable an interrupt, please use the function `SMCNT2EnDisInt()`.

Note:

In 32-bit mode the channel number is ignored.

Input

`u_int8 PortNr`	Module port number
`u_int16 *pStatus`	Pointer to the location where the function stores the interrupt status
`u_int8 Channel`	Channel number (“0” or “1”)

Output

`error_code`	`SUCCESS`	or
	`E_SEEK`	module not initialized
	`E_BPADDR`	wrong register page
	`E_ILLPRM`	wrong channel number
	or standard OS-9 error code (please refer to the OS-9 Technical Manual’s Error Codes section)	

Example

```
u_int16 Status;
...
RetVal = SMCNT2IntStatus(1, &Status, 0);
```

7.2.8.16 SMCNT2ClrInt()

Syntax

```
error_code SMCNT2ClrInt(u_int8 PortNr, u_int16 ClrInt,
u_int8 Channel)
```

Description

This function clears the interrupt bits of the SM-COUNT2. The bits are defined by the variable `ClrInt`, by setting the corresponding bit.

The following predefined macros are available to clear interrupts:

`SMCNT2_INT_MATCH`	Clears match interrupt
`SMCNT2_INT_CCLR`	Clears common clear interrupt

These definitions are made in the SM-COUNT2 specific header file `smcount2.h`.



Note:

In 32-bit mode the channel number is ignored.

Input

`u_int8 PortNr`	Module port number
`u_int16 ClrInt`	Interrupt bits to be cleared
`u_int8 Channel`	Channel number (“0” or “1”)

Output

error_code	SUCCESS	or
	E_BMODE	illegal setting selected
	E_SEEK	module not initialized
	E_BPADDR	wrong register page
	E_ILLPRM	wrong channel number
	or standard OS-9 error code (please refer to the OS-9 Technical Manual's Error Codes section)	

Example

```
RetVal = SMCNT2ClrInt(1, SMCNT2_INT_MATCH
+SMCNT2_INT_CCLR, 0);
```

7.2.8.17 SMCNT2Status()

Syntax

```
error_code SMCNT2Status(u_int8 PortNr,
u_int16 *pStatus, u_int8 Channel)
```

Description

This function reads the status of the SM-COUNT2 and stores the result in the variable that `pStatus` points to.

There are predefined settings for each status bit:

`SMCNT2_STAT_MATCH`	Match output status
`SMCNT2_STAT_EQUAL`	Counter value equals comparator
`SMCNT2_STAT_TIN`	Status of input `TIN`
`SMCNT2_STAT_GATE`	Status of input `GATE`

These definitions are made in the SM-COUNT2 specific header file `smcount2.h`.



Note:

In 32-bit mode the channel number is ignored.

Input

`u_int8 PortNr`	Module port number
`u_int16 *pStatus`	Pointer to the location where the function stores the board status
`u_int8 Channel`	Channel number (“0” or “1”)

Output

error_code	SUCCESS	or
	E_SEEK	module not initialized
	E_BPADDR	wrong register page
	E_ILLPRM	wrong channel number
	or standard OS-9 error code (please refer to the OS-9 Technical Manual's Error Codes section)	

Example

```
u_int16 Status;
...
RetVal = SMCNT2Status(1, &Status, 0);
```