



# **SMART I/O User's Manual**

## **8 Channel AC Input Unit**

### **Micro PLCs and Real-Time Computers**

**Manual ID 09901, Rev. Index 0500 of 08 Jan. 98**

This page was intentionally left blank.



---

## *Preface*

---

<i>Revision History .....</i>	<i>0-4</i>
<i>For Your Safety .....</i>	<i>0-5</i>
<i>Special Handling and Unpacking In- structions .....</i>	<i>0-6</i>
<i>HV Safety Instructions .....</i>	<i>0-6</i>
<i>Two Years Warranty.....</i>	<i>0-7</i>
<i>Table of Contents.....</i>	<i>0-9</i>



**Revision History**

Manual/Product Title:		SMART I/O User's Manual		
Manual ID Number:		09901		
Rev. Index	Brief Description of Changes	PCB Index		Date of Issue
0100	Initial Issue	02	02	Dec. 94
0200	General Update, SM-DAD1 & Software Library Added	02	02	Feb. 95
0300	New MS Modules Added, D5 Format	02	02	Mar. 96
0400	New MS Modules Added, Corrections Including Update of SM-DAD1 'C' Programming Section	02	02	Oct 96
0500	Standard Preface, AC11 Module Added	02	02	Dec. 97

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*.





---

## **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 into a system, please, always switch off your power mains. This applies also to installing piggybacks.
- ☞ 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 only the instructions supplied by the present manual.
- ☞ Keep all the original packaging material for future storage or warranty shipments. If it is necessary to store or ship the board, warranty shipments. If it is necessary to store or ship the board, please, re-pack it in 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 piggybacks, 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.

Your new *PEP* product was developed and tested carefully to provide all features necessary to ensure the renown 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, improper 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, he may return the product 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 on 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 sub-assembly 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 ownership 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 exist.

*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 of the purchaser. In no event will *PEP* be liable for direct, indirect or consequential damages resulting from the use of our 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 *Product Overview* ..... 1-8

1.2 *Ordering Information* ..... 1-9

1.3 *Product Information* ..... 1-10

1.4 *Installation* ..... 1-13

1.5 *ISaGRAF Installation* ..... 1-18

---

**Chapter 2**

2. *SMART-BASE* ..... 2-3

2.1 *Specifications* ..... 2-4

2.2 *Board Overview* ..... 2-5

2.3 *Functional Description* ..... 2-6

2.4 *Configuration* ..... 2-8



---

2.5	<i>Pinouts</i> .....	2-9
2.6	<i>'C' Programming</i> .....	2-17
2.7	<i>ISaGRAF Programming</i> .....	2-32
2.8	<i>Flash Utility</i> .....	2-37

---

## Chapter 3

---

3.	<i>SMART-EXT</i> .....	3-3
3.1	<i>Specifications</i> .....	3-3
3.2	<i>Board Overview</i> .....	3-4
3.3	<i>Functional Description</i> .....	3-5
3.4	<i>Pinouts</i> .....	3-6

---

## Chapter 4

---

4.	<i>Digital Modules</i> .....	4-5
4.1	<i>SM-DIN1</i> .....	4-5
4.2	<i>SM-DOUT1</i> .....	4-19



---

4.3	<i>SM-REL1</i> .....	4-33
4.4	<i>SM-ACI1</i> .....	4-51

---

## Chapter

# 5

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

---

## Chapter

# 6

6.	<i>Communications Modules</i> .....	6-5
6.1	<i>SM-RS232</i> .....	6-5
6.2	<i>SM-SSI</i> .....	6-17

This page was intentionally left blank.



# Table of Contents

- General Information..... 1-3
  - Weights & Measures ..... 1-4
  - 1.1 Product Overview ..... 1-8
  - 1.2 Ordering Information ..... 1-9
  - 1.3 Product Information ..... 1-10
  - 1.4 Installation ..... 1-13
    - 1.4.1 Overview ..... 1-13
    - 1.4.2 SMART I/O Module Installation ..... 1-15
    - 1.4.3 RJ45 Telephone Connector Installation ..... 1-16
    - 1.4.4 Screw Terminal Block Installation ..... 1-16
    - 1.4.5 Battery Installation ..... 1-17
  - 1.5 ISaGRAF-Installation ..... 1-18
    - 1.5.1 Before Installing ..... 1-18
    - 1.5.2 Installation of the ISaGRAF for Windows Workbench ..... 1-19
    - 1.5.3 Installation of PEP Library Functions ..... 1-21
    - 1.5.4 Demo Application ..... 1-26



This page has been left blank intentionally.





## General Information

All PEP products are intended for use in industrial climates where extreme environments exist. Dirt, temperature extremes, varying humidity levels, vibration, noise, shock and electromagnetic signals must all be considered. Only when certain precautions have been followed can PEP guarantee the performance of the product stated in the data sheet.

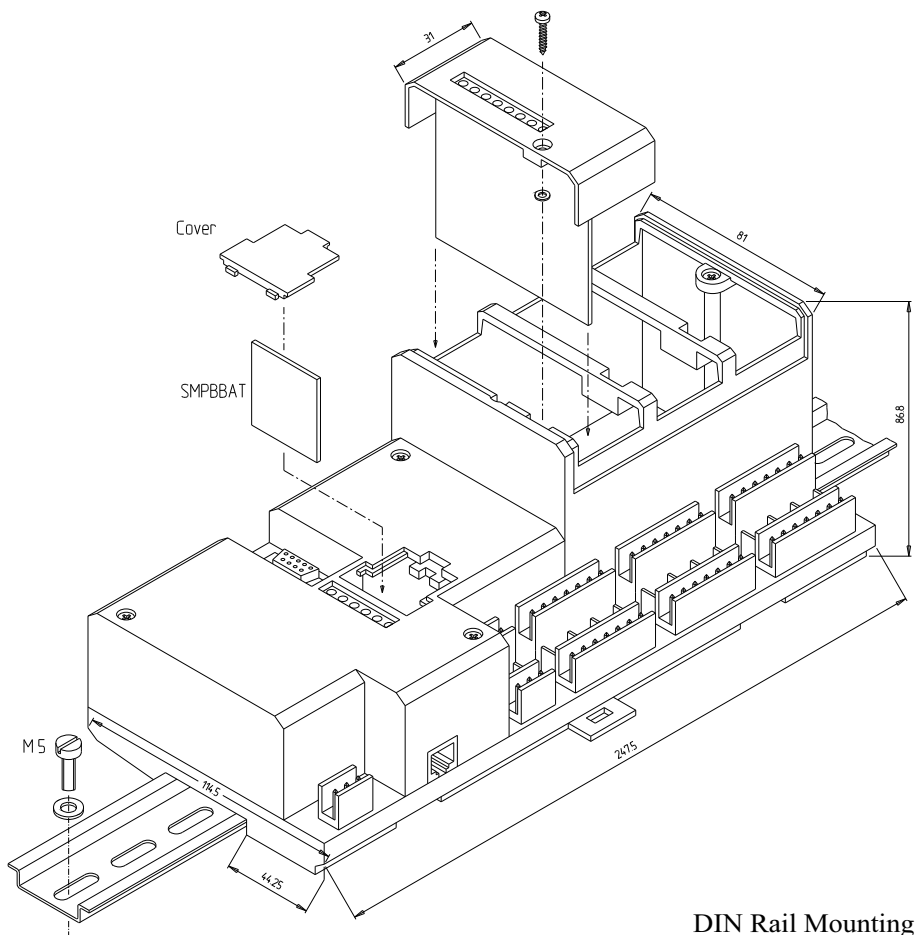
In most cases, controllers are situated in close vicinity to electromechanical devices like relays, transformers, motor controllers and high-frequency switches etc. In such situations, a variety of disturbance sources are present and effect the performance characteristics of the controller. However, by observing the following precautions, many of the bad effects can be minimized.

- Earth protect the controller fixing
- Use screened/shielded cable connections
- Place cables according to relevant standards
- Observe the use of the correct cable diameter and type
- For unused module sockets, install a blank front-panel
- Leave enough room between a 'hot body' and the controller for air to circulate freely
- Place the controller as far away as possible from 'noisy' components
- Separate digital and analog cabling
- Only replace/insert modules in the carrier unit with the power OFF
- Handle the controller components with care; Modules containing highly integrated CMOS components are very sensitive to static discharges
- Try to separate analog modules from their digital cousins.

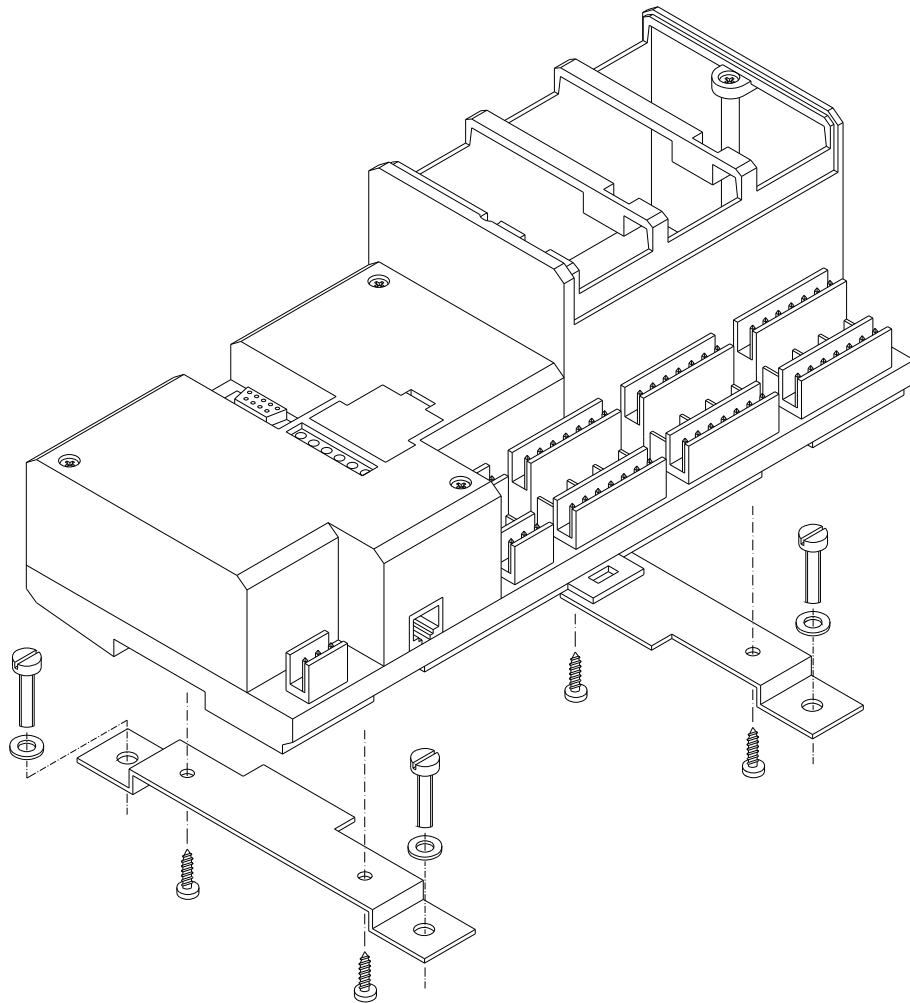


## Weights & Measures

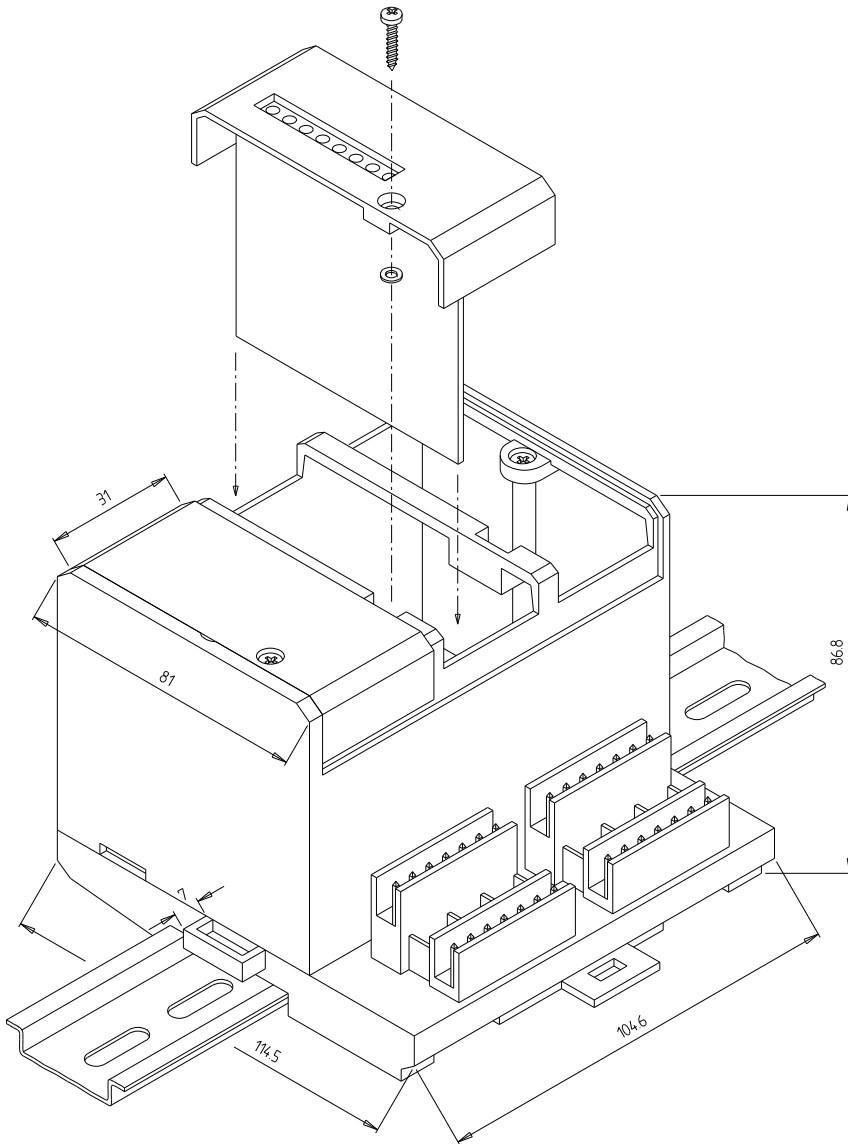
The following line drawings serve to illustrate the method of fixing the controller to a DIN rail or brackets for wall/cabinet mounting. Note that all measurements are in millimetres.

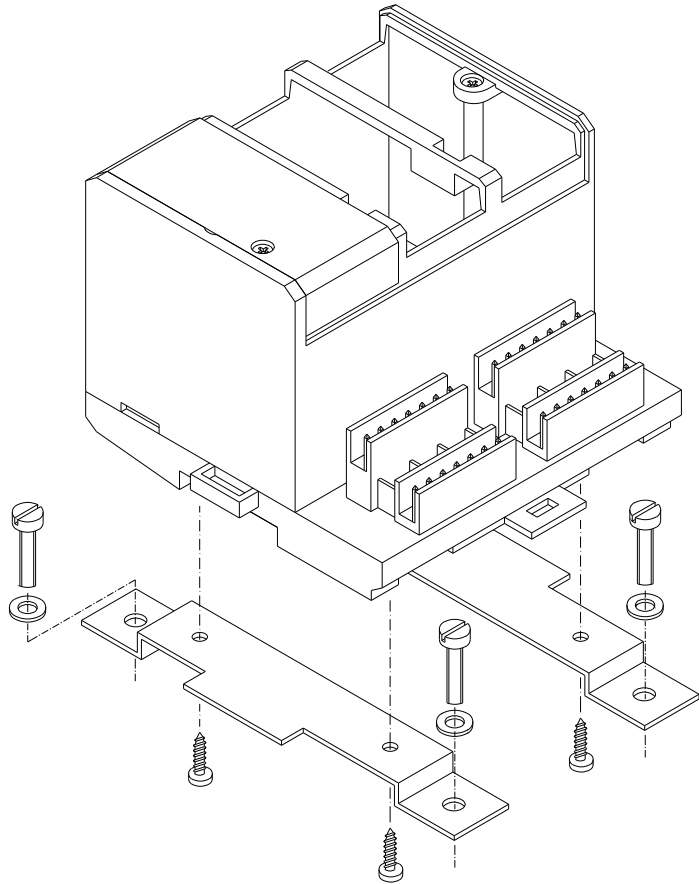






Optional Bracket Mounting





Unit	Weight	Unit	Weight
SMART-BASE	650g	SM-THERM	40g
SMART-EXT	250g	SM-ADC1	70g
SM-DIN1	40g	SM-DAC1	70g
SM-DOUT1	70g	SM-SSI	70g
SM-REL1	61g	SM-CNT1	N/A
SM-DAD1	70g	SM-RS232	40g
SM-PT100	40g		



## 1.1 Product Overview

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

Equipped with the standard real-time fieldbus PROFIBUS, it allows the use of the SMART I/O in 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 not only allows I/O communication, but also file transfer, remote login and remote debugging facilities.

SMART I/O is designed around the MC68302 CPU from Motorola which has two on-chip microprocessors. One is the industry standard 68HC000 running at 20MHz, and the second is a communication orientated RISC processor. Fieldbus protocols use the power of this RISC CPU, freeing the 68HC000 for other tasks. Communication between the 68HC000 and the communication processor is made using on-chip dual-ported RAM. Nonvolatile memory (battery 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 SMART I/O is an ideal companion for VME9000 and IUC9000 systems. Connected together with these high-end VME or IUC computers and PLCs, SMART I/O allows the decentralization of I/O functions through the world wide accepted PROFIBUS fieldbus.

SMART I/O systems provide more power than dumped remote I/O systems since they support local intelligence: an IEC1131-3 programming environment as well as ANSI-C programming under real-time OS. All this power is provided with excellent price/performance characteristics.



## 1.2 Ordering Information

Product	Description	Order Nr.
SMART-BASE	Micro PLC & real-time computer, 1 MByte EPROM, 512 kByte DRAM, 64kByte SRAM, OS-9 v3.0, ISaGRAF v3.0x, PROFIBUS v3.12, Layer 2 & 7, RTC, full modem RS232 (8-pin RJ45 connector), 190 mAh battery, housing and terminal block for the 24V DC power supply	13843
SMART-BASE	As product 13843 but with additional 1 MByte Flash memory (TSOP) on solder side	13844
SMART-EXT	Expansion module for the SMART-BASE supporting 2 SMART-Modules Delivered without terminal blocks, SMART-Modules or blank panels	4228
SM-DIN1	SMART-Module with 8 optoisolated 24V DC digital inputs	4229
SM-DOUT1	SMART-Module with 8 optoisolated 24V DC/500mA digital outputs	4231
SM-DAD1	SMART-Module with 4 channel A/D ( $\pm 10V$ ), 2 channel D/A ( $\pm 10V$ , 0..10V)	9868
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-RS232	SMART-Module with RS232 (Rx & Tx) interface	12461
SM-SSI	SMART-Module with 1 SSI channel providing a 24V DC digital input and a 24V DC, 500mA digital output	12825
SM-ADC1	SMART-Module with 6 12-bit, $\pm 10V$ optoisolated analog inputs	13380
SM-ADC1	SMART-Module with 6 12-bit, 0..20mA optoisolated analog inputs	13868
SM-DAC1	SMART-Module with 2 12-bit, $\pm 10V$ optoisolated analog outputs	13379
SM-DAC1	SMART-Module with 2 12-bit, 0..20mA optoisolated analog outputs with current sensing for broken sensor detection	14018
SM-DAC1	SMART-Module with 6 12-bit, $\pm 10V$ optoisolated analog outputs	14019
SM-DAC1	SMART-Module with 6 12-bit, 0..20mA optoisolated analog outputs with current sensing for broken sensor detection	14020

All SM-Modules are delivered without screw terminals.



Product	Description	Order Nr.
ISaGRAF- ROM-START	ROM kit v3.x for SMART I/O enables the generation of custom firmware EPROMs. Platform can be a PC or OS-9 development system	13829
OS9TRG-RG- SMART	Target CPU kit for SMART I/O (OS-9 v3.x/lx.x disks)	11299
OS9DEVFT- WIN-SMART	OS-9/68000 FasTrak for Windows development pack for SMART I/O. Contains extended OS-9 v3.x/lx.x, PEP utilities and I/O drivers with necessary makefiles for complete application and EPROM generation.	13887
OS9DEVFT- UNIX-SMART	OS-9/68000 FasTrak for UNIX development pack for SMART I/O. Contains extended OS-9 v3.x/lx.x, PEP utilities and I/O drivers with necessary makefiles for complete application and EPROM generation.	13926
OS9-PFB- SMART	OS-9 PROFIBUS starter kit II for 1 node. Includes license, disk and manual	1662
OS9-PFB- SMART(FT)	OS-9 PROFIBUS starter kit II for 1 node operating under FasTrak for Windows. Includes license, disk and manual	12666
Cable	3 meters with 9-pin D-Sub (female) & RJ45 connectors for PC operation	10890
Cable	3 meters with 25-pin D-Sub (male) & RJ45 connectors for modem operation	10891
Screw Term. SCR-2*7.	For the SMART-BASE timer I/O. Pack of 5, 2x3 array	10892
Dummy FP	For the SMART-Modules. Pack of 5, 2x7 array	10893
Battery	For unused SMART-Module slots. Available in packs of 20	10894
Battery	3V, 190mAh lithium battery (button) BR2032 for use in the standard temperature range (0°C to +70°C)	11281
Battery	3V, 850mAh lithium battery (cylinder) CR14250 for use in the extended temperature range (-40°C to +85°C)	11282

### 1.3 Product Information

The SMART-BASE is a control module possessing an RS485 PROFIBUS and serial RS232 interface as standard and provision for up to three SMART-Modules. The modules currently available are discussed later. Normally, the base unit is mounted on a DIN rail (see Weights & Measures section) which in turn may be fixed in a cabinet. SMART-EXT units may be attached in a similar manner.



The attachment of both units is achieved by sliding them over the DIN rail with the clip assembly pulled out and then releasing it when correctly positioned.

The 24V supply source should possess the following characteristics:

Voltage		Current	
min	max	Continuous	Peak
18V	36V	400mA	1.5A (2ms)

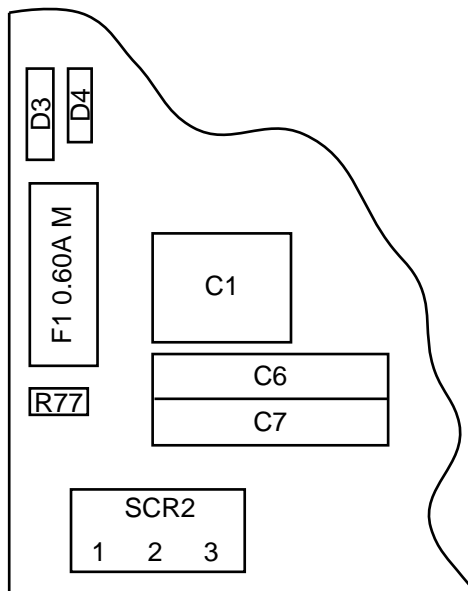
Because of power limitations, it should be noted that although up to 11 SMART-Modules are possible in a complete configuration, attention should be paid to the following tables that show the individual component power requirements. The maximum of 6750mW should not be exceeded as damage to the on-board DC-DC converter may result due to overheating or entering a reset status due to the thermal cutoff protection switching mechanism.

SM-BASE	Power Consumption	
	min	max
CPU Core	1500mW	1700mW
Profibus	250mW	750mW

Module	Power Consumption		Module	Power Consumption	
	min	max		min	max
SM-DIN1	5mW	100mW	SM-ADC1	350mW	450mW
SM-DOUT1	5mW	270mW	SM-DAC1	5mW	400mW
SM-DAD1	350mW	450mW	SM-DAC1	350mW	660mW
SM-PT100	400mW	500mW	SM-SSI	unknown	unknown
SM-THERM	400mW	500mW			
SM-RS232	23mW	75mW			
SM-REL1	23mW	160mW			



An on-board fuse protects the 24V DC input circuitry from damage through higher voltages than those expected or AC voltages being inadvertently applied to the system. This fuse, should it be assumed defect (the Power In LED on the housing will not be illuminated), may be accessed by removing the cover of the SMART-BASE and accessing the holder on the left-hand side as shown in the illustration below.



In the event of a blown fuse, replace it with the same size and type as the one installed.

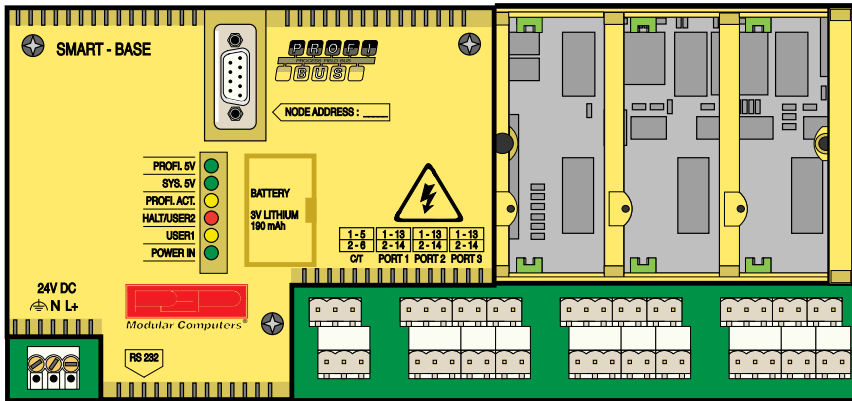




## 1.4 Installation

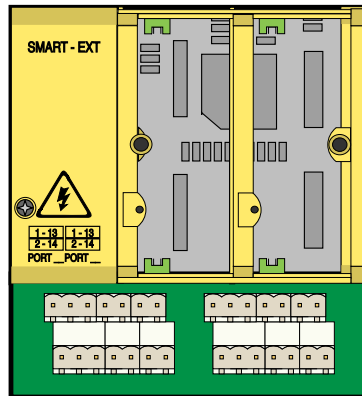
### 1.4.1 Overview

The SMART-BASE and SMART-EXT units are supplied without screw terminal blocks for the I/O slots, SMART Module piggybacks or blank panels. These must be ordered separately to meet the requirements of individual specifications.



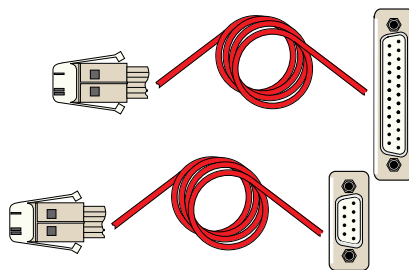


Up to 4 SMART-EXT units can be cascaded depending on the power consumption of the individual SMART Modules.



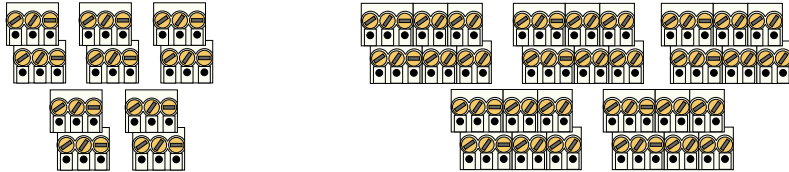
SMART I/O Modules or blank panels must be ordered separately to the SMART-BASE or SMART-EXT units. Blank panels come in packs of 20.

Two RS232 cables are available. One terminates with a female 9-pin D-Sub connector for PC use and the other terminates in a male 25-pin D-Sub connector for Modem operation.





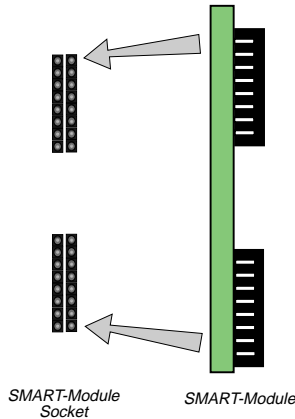
Screw terminal connectors are available in packs of 5.



### 1.4.2 SMART I/O Module Installation

The SMART I/O Modules are fitted into the relevant sockets (ST1 - ST6; 3 slots) on the SMART-BASE or in sockets on the SMART-EXT unit. It is important that the Modules are inserted the correct way. The Figure below illustrates this procedure.

Figure 1.4.2.1 : SMART I/O Module Installation



**WARNING!**

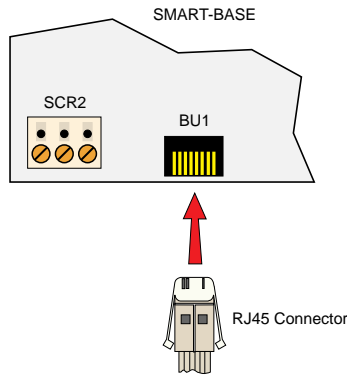
Once fitted on the board, the Module sockets and components should be on the right hand side of the Module.



### 1.4.3 RJ45 Telephone Connector Installation

The RJ45 connector is fitted into the RS232 Telephone connector (BU1) on the SMART-BASE. This is illustrated in the Figure below.

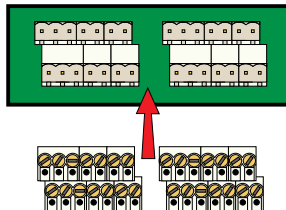
**Figure 1.4.3.1 : RJ45 Telephone Connector Installation**



### 1.4.4 Screw Terminal Block Installation

The Screw Terminal Blocks are easily fitted to the SMART-BASE or SMART-EXT by pushing them onto the relevant Screw Terminal, as shown in the figure below.

**Figure 1.4.4.1 : Screw Terminal Block Installation**

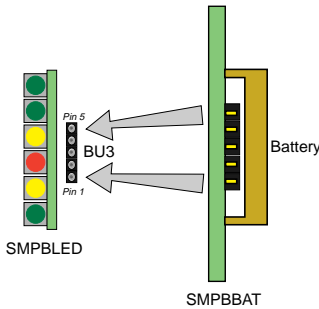




### 1.4.5 Battery Installation

The battery piggyback SMPBBAT is fitted into the socket BU3 on the SMART-BASE. It is important that the piggyback is inserted in the correct way. The figure below illustrates this procedure.

**Figure 1.4.5.1: Battery Piggyback Installation**



<b>Battery Type</b>	3V Lithium BR2032	3V Lithium CR14250
<b>Battery Capacity</b>	190mAh	850mAh
<b>Typical Data Retention Time *</b>	40 Days	590 Days

\* this is the time without the main power being applied

#### **WARNING!**

Once fitted on the board, the battery should be on the right hand side of the SMPBBAT piggyback. If the battery needs to be replaced, it must only be done with a replacement SMPBBAT piggyback, the order number of which is shown in the Ordering Information section of this manual.

The temperature on the battery must not exceed +70°C, due to the risk of battery damage! For SMART I/O modules with extended temperature ranges of up to 85°C, a special lithium battery must be fitted to the SMPBBAT.



## 1.5 ISaGRAF-Installation

### 1.5.1 Before Installing

ISaGRAF is a Windows™ 3.xx based software development tool requiring a minimum of 10 MB of hard disk space and 4 MB of available memory.

Before installing ISaGRAF, make a backup copy of each DOS disk in the package and write-protect them to prevent accidental overwriting of files.

Note : The backup disks must have the same volume labels as the original ISaGRAF disks. Use the Windows *Copy Disk...* command on the *Disk* menu in the File Manager to create backup disks with the original volume labels and disk contents.

If it is intended to install ISaGRAF in a directory other than the default (C:\ISAWIN) then remember to provide the full path of the new directory when prompted during installation.

Altogether, 10 DOS disks and 2 OS-9 disks are supplied for ISaGRAF installation; four for the Workbench, two composite and four for the ISaGRAF Target and are labelled:

- “Workbench Disk 1/4”
- “Workbench Disk 2/4”
- “Workbench Disk 3/4”
- “Workbench Disk 4/4”
- “Lib/Appli/Help Disk 1/2”
- “Lib/Appli/Help Disk 2/2”
- “Samples for OS-9 1/1”
- “Target Disk 1/2” for DOS
- “Target Disk 2/2” for DOS
- “Profibus FMS for ISaGRAF, Documentation, Disk 1/1”
- “Target Disk 1/2” for OS-9
- “Target Disk 2/2” for OS-9



### 1.5.2 Installation of the ISaGRAF for Windows Workbench

The following steps should be followed to ensure successful installation of the ISaGRAF software. Initially the disk labelled **Workbench Disk 1/4** will be required.

- Start Windows
- Insert diskette **Workbench Disk 1/4** into the floppy drive (usually A:)
- Select **File** from the Windows Program Manager and select **Run ...**
- Type **A:INSTALL** in the command field and select **OK**

The ISaGRAF installation will start automatically. Figure 1.5.2.1 illustrates a typical opening screen.

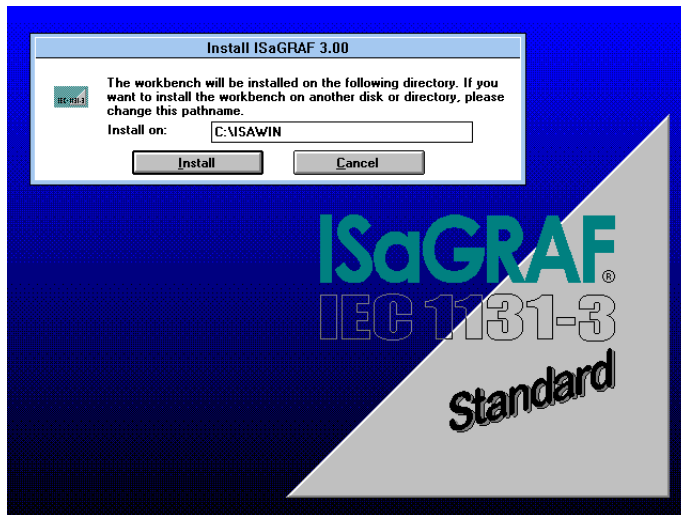


Figure 1.5.2.1 Typical Opening Screen



Having checked the installation directory (default is **C:\ISAWIN**) and selected **Install**, the program progresses by asking whether the complete system should be installed or just certain sections. The selection possibilities are shown in figure 1.5.2.2.

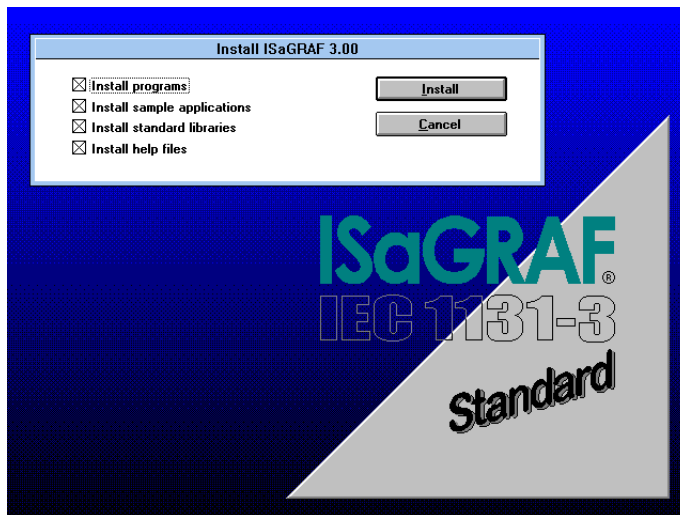


Figure 1.5.2.2 Installation Selection

The default is for a complete installation, i.e. all files. Once confirmed, the installation copies the required files to the installation directory and unpacks their contents. This procedure will take a few minutes. Upon completion, the windows desktop will show a new program group containing the files shown in figure 1.5.2.3.



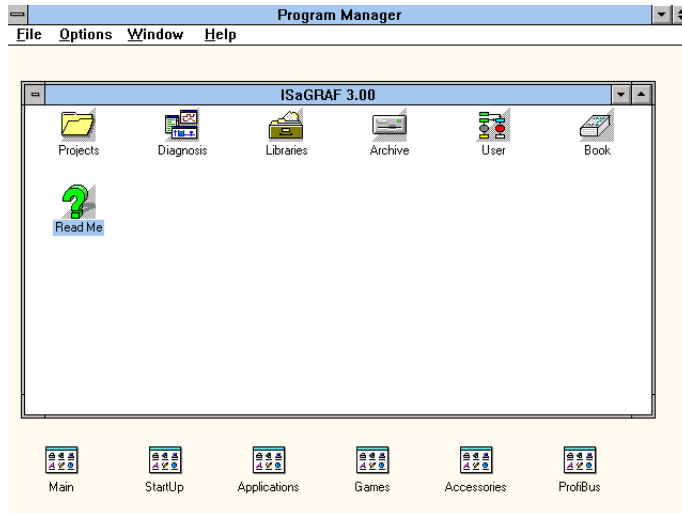


Figure 1.5.2.3 ISaGRAF Program Group

### 1.5.3 Installation of PEP Library Functions

The library functions are adapted to suit the SMART I/O and other PEP products and should be installed using the two diskettes labelled **LIB/APPLI/HELP**.



These libraries for projects, I/O boards, 'C' functions and common data are extracted by following the described procedure:

- Start Windows if not already started
- Insert diskette **Lib/Appli/Help Disk 1/2** into the floppy drive
- Select **File** from the Windows Program Manager and select **Run ...**
- Type **A:\INSTALL** in the command field and select **OK**

Figure 1.5.3.1 illustrates the Installation Start-up screen.

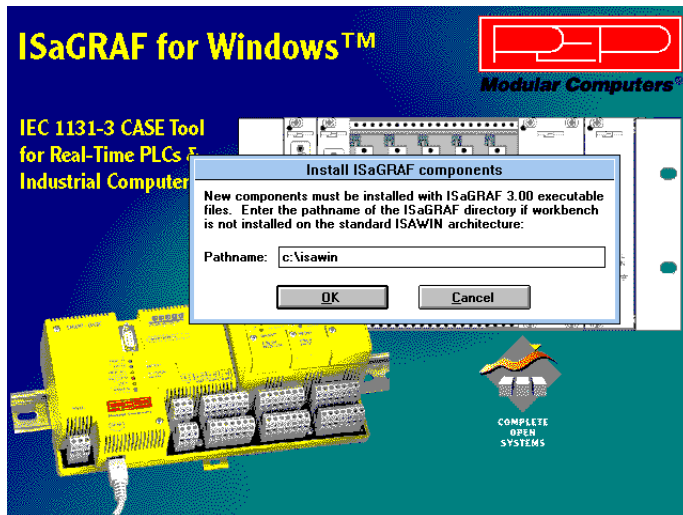


Figure 1.5.3.1 The Installation Start-up Screen



It should be noted that the standard ISA-Terminal is configured for COM2. If another port is required then the switch to the new one is made by firstly starting the *ISA-Terminal* program and then selecting the *Settings* from the *Communication* pull-down menu. Here the possibility exists to select the desired communications port. When leaving the ISA-Terminal environment, remember to save the configuration if changes have been made.

The installation of the ISaGRAF development tool is now complete and access is provided to a full IEC1131-3 programming platform.

The following sections deal with the application of this tool with the SMART I/O and other PEP PLCs. Note that 'C' programming is not an option for the SMART I/O Starter Kit and if required must be ordered separately through one of the PEP offices or agents.



In order to verify that the hardware and software have been correctly setup, the following procedure should be followed.

- Connect the D-Sub connector end of the terminal cable to the chosen COM port of the computer. The other end, with the telephone type connector should be pushed into place in the RS232-port of the SMART I/O base (see figure 1.4.3.1)
- With the power supply turned OFF, connect the power plug to SCR-2 on the SMART base.
- Start the *ISA-Terminal* program.
- Switch on the power supply to the SMART I/O; three green LEDs should illuminate on the control panel (not the SMART Modules!). The terminal should display the messages shown in figure 1.5.3.2.

```

Terminal - ISAWDS.TRM
Datei Bearbeiten Einstellungen Telefon Übertragung Hilfe
OS-9/68K System Bootstrap
Found OS-9 kernel module at $00C01A18
$
Software configuration:
ISaGRAF v3.02 I1.0
Profibus U3.12 I2.0
Created 16. Nov. 95

setenv HOME /dd
mkdir /dd/SVS
mkdir: can't make "/dd/SVS". Error #000:218
mkdir /dd/CHDS
mkdir: can't make "/dd/CHDS". Error #000:218
mkdir /dd/APL
mkdir: can't make "/dd/APL". Error #000:218
echo "super user:0,0,130,/dd/CHDS,/dd.shell" >/dd/SYS/password
chx /dd/cmds
/dd/startup
shell: can't execute "/dd/startup" - Error #000:216
chd /dd/APL
ex startprg -t:s -p:isatst -t:/term <<>>/nil&; sleep -s 2; ex isakersm <<>>/nil
&
*16
    
```

Logout	Application	3xEnter	Ebene: 1
Root	System	del isa11	15:05:21

Figure 1.5.3.2 Power Up Messages



The error messages that are shown in figure 1.5.3.2 are normal as the system is trying to create files or directories in the RAM disk that are already present.

If no further messages appear, then the installation is complete. Should the terminal result in anything different than shown then check through the installation steps again before contacting PEP for help.

The three green LEDs mentioned earlier show the state of

- PROFIBUS 5V
- System 5V and
- Power ON



### 1.5.4 Demo Application

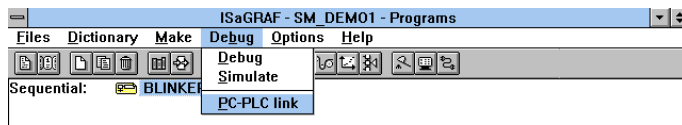
Several demonstration applications are delivered with the ISaGRAF set of disks and are installed automatically. The applications suitable for use with the SMART I/O are prefixed **SM-** and serve to show how the SMART I/O can be used through practical examples. Two such programs, the **SM-DEMO1** and **2** are discussed here :

**SM\_DEMO1** should be used if the SMART I/O has the SM\_DOUT1 module in the first of the SMART Module slots and has SM\_DIN1 in slot 2 (the last) on the SMART-BASE.

**SM\_DEMO2** should be used if the modules are reversed.

To install the demonstrations correctly, observe the following procedure and make sure the correct diskette is in place :

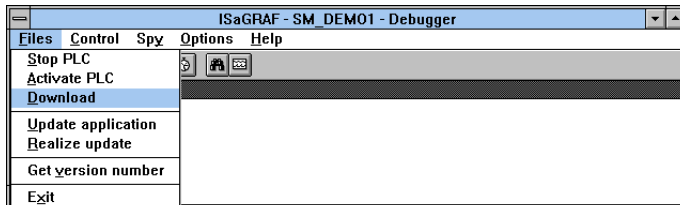
- Run the *Projects* Program and select SM\_DEMO1 or SM\_DEMO2 by double clicking on the icon;
- Go to the *Debug/PC-PLC Link* Menu to ensure that the correct COM port is selected for communication to the SMART I/O;



- When confirmed, select *Debug/Debug*;
- From the Debugger Window, select *File* from the pull-down menu if an application is already executing (indicated by *Run* in the status line);



- Finally, from the **Files/Download** pull-down menu, select Motorola Target Code.



The chosen SM\_DEMO will be downloaded to the SMART I/O (target system) and the application will begin automatically.

This is a simple SFC-program which will activate channel 7 of the SM-DOUT1 when the first SFC-step (init) is encountered thereby illuminating the diode. During the next cycle, step 2 is encountered and activates channel 2 and at the same time deactivates channel 3. The program will wait for 1 second in this state before reversing the action. The effect is that the second and third LED illuminate alternately with a pause of 1 second between.



This page has been left blank intentionally.







# Table of Contents

- 2. SMART-BASE .....2-3
  - 2.1 Specifications ..... 2-4
  - 2.2 Board Overview ..... 2-5
  - 2.3 Functional Description ..... 2-6
  - 2.4 Configuration ..... 2-8
    - 2.4.1 Jumper J1: Boot Selection (Pin Connector) ..... 2-8
    - 2.4.2 Jumper J6: LED Function (Pin Connector) ..... 2-9
  - 2.5 Pinouts ..... 2-9
    - 2.5.1 SMART Module Piggyback Connectors ..... 2-10
    - 2.5.2 Screw Terminal Pinouts ..... 2-12
    - 2.5.3 Timer I/O Screw Terminal (SCR1) ..... 2-12
    - 2.5.4 Supply Screw Terminals (SCR2) ..... 2-14
    - 2.5.5 RS232 Telephone Connector (BU1) ..... 2-15
    - 2.5.6 RS485 D-Sub Connector for Half-Duplex Operation (Profibus) ..... 2-15
    - 2.5.7 SPI Connector (ST7) ..... 2-16
  - 2.6 'C' Programming ..... 2-17
    - 2.6.1 SMART-BASE Library ..... 2-17
    - 2.6.2 SMTselIn ..... 2-18
    - 2.6.3 SMTsettout ..... 2-20
    - 2.6.4 SMTpre ..... 2-21
    - 2.6.5 SMTstasto ..... 2-23
    - 2.6.6 SMTrd ..... 2-24
    - 2.6.7 SMTtin ..... 2-25
    - 2.6.8 SMTstat ..... 2-26
    - 2.6.9 SMTtout ..... 2-27
    - 2.6.10 SMLed ..... 2-28
    - 2.6.11 SMwdon ..... 2-29
    - 2.6.12 SMwdtrig ..... 2-30
    - 2.6.13 SMwdoff ..... 2-31





2.7 ISaGRAF Programming .....	2-32
2.7.1 The ISaGRAF Board Parameters .....	2-32
2.7.2 The ISaGRAF Operate Calls .....	2-33
2.8 Flash Utility .....	2-37



---

## 2. SMART-BASE



The SMART-BASE is the main unit to which up to three SMART-Modules may be connected to fulfil a given I/O task with I/O enhancement being provided through the connection of a SMART-EXT unit which itself may accommodate up to 2 SMART-Modules. A counter/timer is also a standard feature providing direct access to IRQ4 of the I/O controller. The driving force behind the SMART-BASE is the MC68302 microprocessor from Motorola operating at 20MHz.

The SMART-BASE, complete with built-in RS232 and RS485 (PROFIBUS) interfaces is connected to the outside world by RJ45 and 9-pin D-Sub connectors respectively. Connected SMART-Modules use industrial standard, plug-in screw terminals.

Program code is stored in EPROM or FLASH memory thereby doing away with disk accesses and ensuring data security in harsh industrial environments. A SMART-BASE unit with additional 1MByte FLASH memory may be ordered as an option in which to extend the standard code and supply user-specific functionality.



## 2.1 Specifications

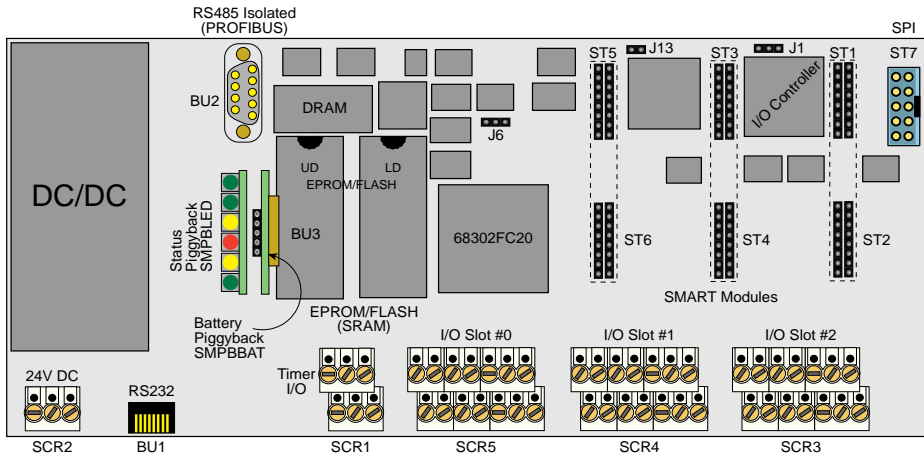
<b>DC/DC</b>	
Nominal Input Voltage	24V DC
Input Voltage Range	18V - 36V DC
Input Current	140mA typ. @ 24V (static) 400mA typ. @ 24V (full load)
Main Output Voltage	5V DC / 1.2A ± 2.5%
Auxiliary Output Voltage	5V DC / 150mA (PROFIBUS/RS485) ± 5%
Switching Frequency	100kHz
Main Output Ripple	± 10mV typ.
Max. Efficiency	68% typ. (between 74% and 100% load)
Galvanic Isolation	500V DC max. to/from supply source
<b>Controlling Unit</b>	
CPU	MC68302 @ 20MHz
EPROM or FLASH*	1 MByte or 256 kByte on Jedec 32-pin sockets (16-bit access)
FLASH*	1 MByte or 256 kByte additionally soldered on rear side of the board (16-bit access)
DRAM	512 kByte, byte/word access (16-bit)
SRAM	64 kByte, byte/word access (16-bit)
EEPROM	93C46, 32-byte system data
Galvanic Isolation	2.5kV DC to/from process I/O
<b>Common</b>	
Temperature Range	
- Storage	-40°C to + 85°C (without battery)
- Operating	0°C to +70°C
- Extended	-40°C to +85°C
Operating Humidity	0 to 95% non-condensing
Weight	640g typ. without SMART-Modules, blank panels or screw terminals

\* Option

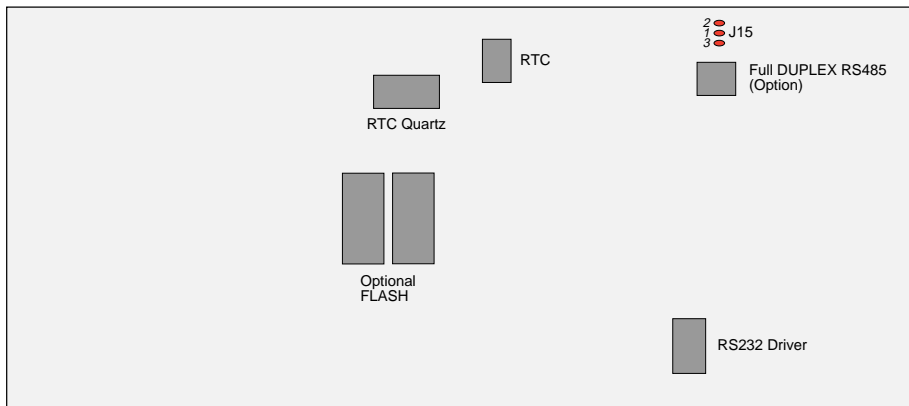


## 2.2 Board Overview

### Front view



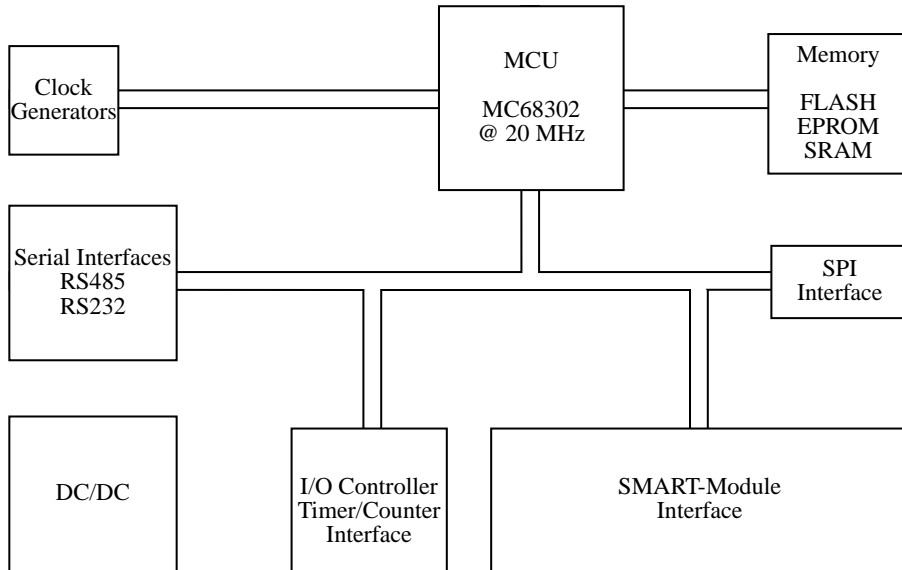
### Rear View





### 2.3 Functional Description

**Figure 2.3.0.1: SMART I/O Block Diagram**



The MCU, an MC68302 microprocessor operating at 20 MHz is responsible for handling all communication between the various interfaces and on-board memory.

An RS485 highspeed PROFIBUS 2-wire interface is optically isolated from the system and may be configured for full duplex operation (a 4-wire interface available on request for OEM volume). Likewise, a fully configured RS232 modem interface is available for program downloading for example on SMART I/O systems supporting FLASH memory for this task or to the RAM disk etc.



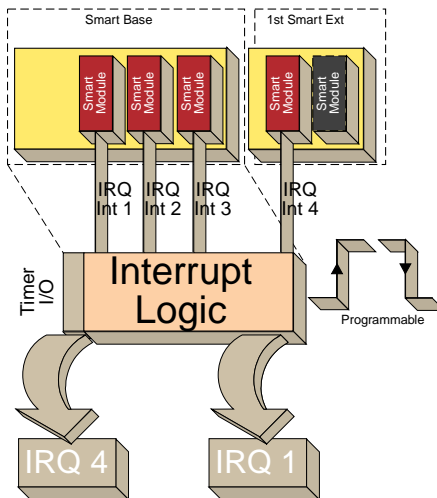
The isolated DC/DC converter is based on a switched mode regulating system operating at 100 kHz and supplies power to both the system and the isolated RS485 (PROFIBUS) interface.

The SPI interface, a 3-wire communication protocol providing SCLK, RxD and TxD is embedded within a larger 10-wire interface for handling communication between the base unit and SMART-Modules attached either directly or on extension units.

Up to three SMART-Modules are supported on the SMART-BASE unit with connection to the outside world being provided by industrial standard screw terminals. The same calibre of terminal is used for the counter/timer, which is directly coupled to the system I/O controller. A more detailed description of the counter/timer appears in the pinout section of this chapter.

Figure 2.3.0.2 shows the interrupt handling capability of the I/O controller. It should be noted that the controller can only be programmed with the use of a PEP defined software protocol that ensures compatibility in case of hardware changes.

**Figure 2.3.0.2 IRQ Capabilities**

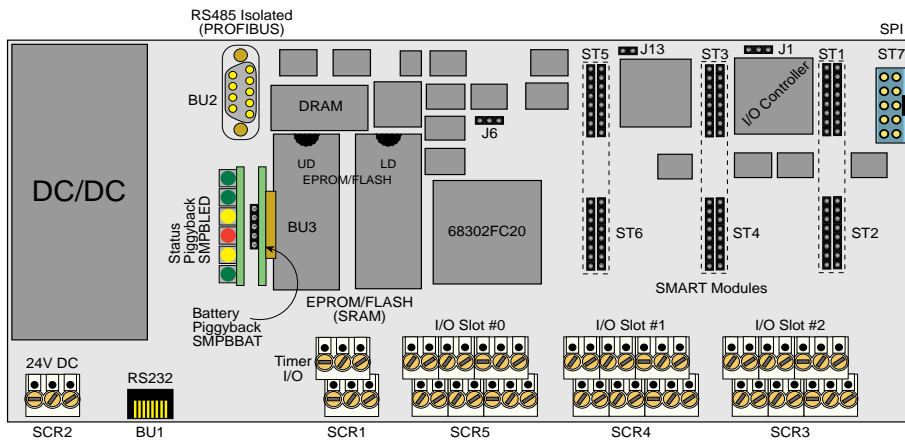




## 2.4 Configuration

The SMART BASE has 2 configurable jumpers which are explained in the following sections. The jumper settings marked in *italics* in the tables are default.

**Figure 2.4.0.1: SMART BASE Jumper Layout (Front View)**



### 2.4.1 Jumper J1: Boot Selection (Pin Connector)

The jumper J1 selects whether the SMART I/O boots directly from OS-9 or from ISaGRAF.

Jumper J1	Description
1-3	<i>Application Boot (ISaGRAF)</i>
1-2	OS-9 Shell Boot





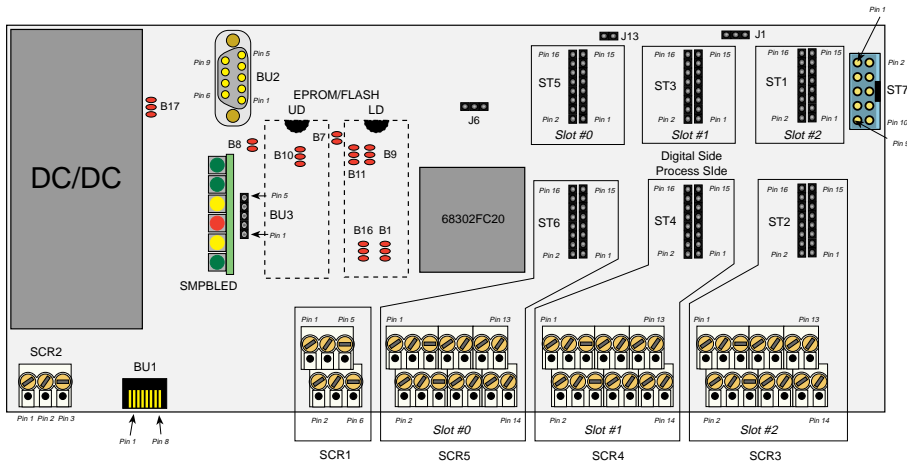
### 2.4.2 Jumper J6: LED Function (Pin Connector)

This jumper selects the function of the red LED; halt or user defined. The user defined function that is supported in software will only take effect if this jumper is set accordingly.

Jumper J6	Description
1-3	Processor HALT function monitor
1-2	User defined function

## 2.5 Pinouts

Figure 2.5.0.1: SMART-BASE Connector Overview



# WARNING !

Dangerous voltages may be present at the terminals.



**Note**

Slot# numbers are counted from #0 up to #10 while the ISaGRAF logic counts from #1 to #11!

**2.5.1 SMART Module Piggyback Connectors**

There are three sets of SMART Module piggyback connectors available on the SMART-BASE, each divided into two sets of 2x8 standard pin rows.

Pinouts digital side (ST1, ST3 and ST5)

*SMART-Module location #0 (ST5) pinouts*

Pin Nr.	Signal	Description	Pin Nr.	Signal	Description
1	PA8	68302 Port A8	2	PA9	68302 Port A9
3	PA10	68302 Port A10	4	PA11	68302 Port A11
5	PA12	68302 Port A12	6	PA13	68302 Port A13
7	PA14	68302 Port A14	8	PA15	68302 Port A15
9	System GND	GND	10	Serial Rx/D	RxD
11	System VCC	5V VCC	12	Serial Tx/D	TxD
13	CS-SM1	Port Select (Module 0)	14	Serial CLK	CLK
15	Reset	Reset (Power ON/OFF)	16	SM1 Interrupt PI/T	INT3 to the I/O Controller

*SMART-Module location #1 (ST3) pinouts*

Pin Nr.	Signal	Description	Pin Nr.	Signal	Description
1	PITB0	I/O Controller Port B0	2	PITB1	I/O Controller Port B1
3	PITB2	I/O Controller Port B2	4	PITB3	I/O Controller Port B3
5	PITB4	I/O Controller Port B4	6	PITB5	I/O Controller Port B5
7	PITB6	I/O Controller Port B6	8	PITB7	I/O Controller Port B7
9	System GND	GND	10	Serial Rx/D	RxD
11	System VCC	5V VCC	12	Serial Tx/D	TxD
13	CS-SM2	Port Select (Module 1)	14	Serial CLK	CLK
15	Reset	Reset (Power ON/OFF)	16	SM2 Interrupt PI/T	INT2 to the I/O Controller



*SMART-Module location #2 (ST1) pinouts*


Pin Nr.	Signal	Description	Pin Nr.	Signal	Description
1	PITA0	I/O Controller Port A0	2	PITA1	I/O Controller Port A1
3	PITA2	I/O Controller Port A2	4	PITA3	I/O Controller Port A3
5	PITA4	I/O Controller Port A4	6	PITA5	I/O Controller Port A5
7	PITA6	I/O Controller Port A6	8	PITA7	I/O Controller Port A7
9	System GND	GND	10	Serial RxD	RxD
11	System VCC	5V VCC	12	Serial TxD	TxD
13	CS-SM3	Port Select (Module 2)	14	Serial CLK	CLK
15	Reset	Reset (Power ON/OFF)	16	SM3 Interrupt PI/T	INT1 to the I/O Controller




**Pinouts process side (ST6, ST4 and ST2) for Modules #0 to #2**

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

The PC board connections to the screw terminals are capable of absorbing a continuous current of up to 3A each. However, pins 13 and 14 can support up to 6 Amps.



## WARNING !

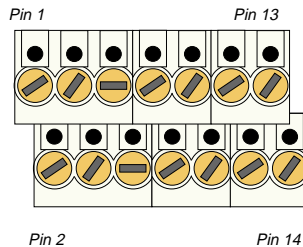


Dangerous voltages may be present at the terminals.



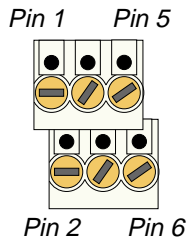
### 2.5.2 Screw Terminal Pinouts

The following shows the pinout for a screw terminal block suited for use with SMART-Modules. The pinouts of these blocks depends on the SMART Modules that are fitted.



### 2.5.3 Timer I/O Screw Terminal (SCR1)

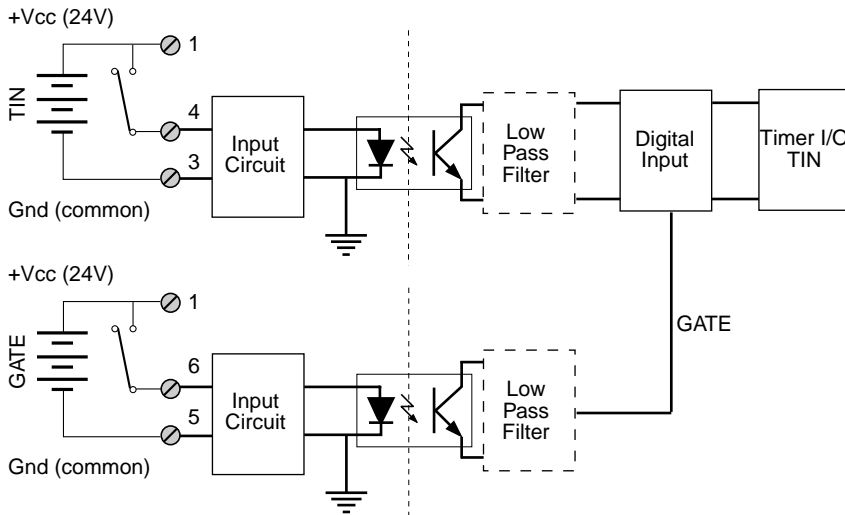
Pin Nr.	Signal	Description	Pin Nr.	Signal	Description
1	External VCC	5V VCC	2	TOUT	Timer OUT
3	External GND	Ground for TIN, TOUT	4	TIN	Timer IN
5	External GND	Ground for TGATE	6	GATE	GATE Connection





To understand the functionality of the counter/timer, it is necessary to understand the purpose of TIN, TOUT and TGATE. Figure 2.5.3.1 shows the block diagram of TIN.

**Figure: 2.5.3.1 Timer I/O (TIN) Schematic**



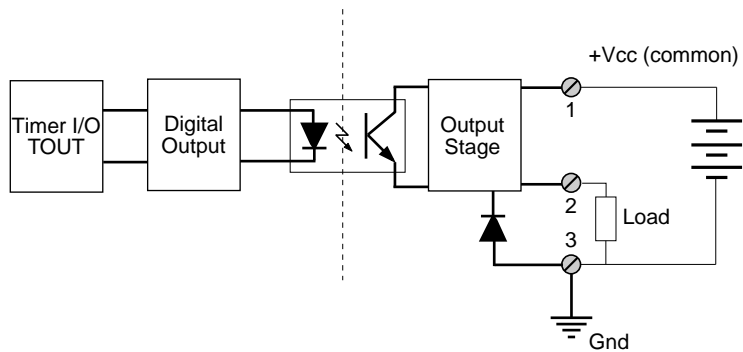
The I/O Controller timer is used for the generation of the TOUT and TIN functions with the three timer I/O lines being fully isolated from the system. The internal clock of the timer/counter is 6MHz and can be prescaled to enable lower frequencies as necessary.

With the GATE permanently active (relay closed), every pulse detected on the TIN line will be acknowledged up to a frequency of 20kHz. Otherwise, TIN will only be recognized for the duration that the GATE is active.



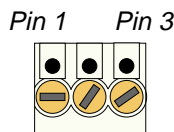
Figure 2.5.3.2 shows the TOUT block diagram. Here, the output is only active when an interrupt on level 4 has been acknowledged by the I/O controller or a previously set timer has decremented to 0. The driving stage of the output consists of a Darlington connected transistor pair protected from inductive loads by a clamp diode. This TOUT line can generate square-wave pulses from 0.2ms to 178ms and can deliver 500mA continuously at 24V DC. The maximum frequency of TOUT is 5kHz.

**Figure 2.5.3.2: Timer I/O (TOUT) Schematic**



**2.5.4 Supply Screw Terminals (SCR2)**

Pin Nr.	Signal
1	SHIELD
2	GND
3	+24V

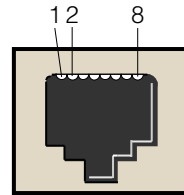




### 2.5.5 RS232 Telephone Connector (BU1)

In order to meet the needs of widespread standards, the RS232 connector is selected as a telephone connector, an 8-pin RJ12 telephone jack with full MODEM support.

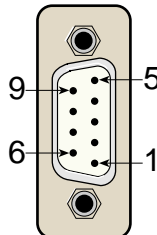
Pin Nr.	Signal	Description
1	DSR	Data Set Ready
2	RTS	Ready to Send
3	System GND	System GND
4	TxD	Transmit Line
5	RxD	Receive Line
6	DCD	Data Carrier Detect
7	CTS	Clear to Send
8	DTR	Data Terminal Ready



### 2.5.6 RS485 D-Sub Connector (BU2) for Half-Duplex Operation (Profibus)

Pin Nr.	Signal	Description	Pin Nr.	Signal	Description
1	SHIELD	Shield Isolation	6	Aux. +5V	Auxiliary +5V
2	N/C	Not Connected	7	N/C	Not Connected
3	T/RxD +	Transmit / Receive Line	8	T/RxD -	Transmit / Receive Line
4	CNTR +*	Control Line	9	CNTR -*	Control Line
5	Aux. GND	Auxiliary Ground			

\* optional, for full-duplex operation





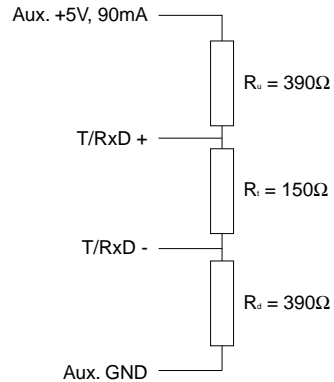
The full-duplex description may be found in the *SMART-I/O Advanced User's Guide*.

**Note**

There is no internal line termination as laid down in DIN 19245 Part 1 and must be performed externally.

The line termination is achieved as shown in the figure. Assuming a power supply voltage of +5V emanating from the PROFIBUS connector (pin 6), the following resistor values are recommended.

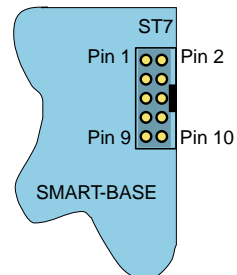
- R<sub>t</sub> 150Ω ± 2%, min 0.25W
- R<sub>U</sub> 390Ω ± 2%, min 0.25W
- R<sub>d</sub> 390Ω ± 2%, min 0.25W



**2.5.7 SPI Connector (ST7)**

The SPI connector is a 2x5 standard pinrow connector, located on the right-hand side of the SMART-BASE to enable easy connection of the SMART-EXT module using a flat ribbon cable connection.

Pin Nr.	Signal	Pin Nr.	Signal
1	System VCC	2	Serial RxD
3	System VCC	4	Serial TxD
5	Serial Ext. Select	6	Serial CLK
7	System GND	8	Reset
9	System GND	10	Serial Ext. Interrupt (Controller I/O IRQ4)







## 2.6 'C' Programming

### 2.6.1 SMART-BASE Library

The SMART-BASE library of functions *smartio.l* provide a convenient way of accessing various features of the SMART-BASE.

- All functions are written in ANSI C;
- Prototypes are to be found in the file *BSP/SMART/DEFS/SMAC.h*.

#### Hardware Requirements

- SMART I/O Base 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 examples provided here are primarily concerned with the timer/counter. Other aspects of SMART BASE programming may be found in the Advanced User's guide.



## 2.6.2 SMTselIn

### Syntax

```
error_code SMTselIn(u_int8 mode);
```

### Description

This function selects one of four possible counter/timer input (TIN) configurations utilizing the 6MHz internal counter/timer clock.

### Input

`u_int8 mode` This represents the four input configurations. They are:

`MODE00` The Simple I/O/TIN input pin carries the Simple I/O and the CLK and prescaler are used. The prescaler is decremented on the falling transition of the clock pin; the 24-bit counter is decremented, rolls over or is loaded from the counter preload registers when the prescaler rolls over from \$00 to \$1F.

`MODE01` The Simple I/O/TIN serves as a timer input and the CLK and prescaler are used. The prescaler and counter are decremented as in `MODE00`.

`MODE10` The Simple I/O/TIN pin serves as a timer input and the prescaler is used. The prescaler is decremented following the rising transition of the TIN pin after being synchronized with the internal clock. The 24-bit counter is decremented, rolls over or is loaded from the counter preload registers when the prescaler rolls over from \$00 to \$1F.



**MODE11** The Simple I/O/TIN pin serves as a timer input and the prescaler is not used. The 24-bit counter is decremented, rolls over or is loaded from the counter preload registers following the rising edge of the TIN pin after being synchronized with the internal clock.

### Output

<code>error_code</code>	<code>SUCCESS</code>	
	<code>E_BMODE</code>	Unsupported mode or standard OS-9 error code (refer to the OS-9 Technical Manual Error Codes Section).

### Example

```
RetVal = SMTselIn(MODE00);
```

The counter/timer contains a 24-bit synchronous down counter that is loaded from three 8-bit counter preload registers. The 24-bit counter may be clocked by the output of a 5-bit (divide by 32) prescaler or by an external timer input (TIN). If the prescaler is used, it may be clocked by the system clock (6 MHz CLK pin) or by the TIN external input. The counter signals the occurrence of an event primarily through zero detection. (A zero is when the counter of the 24-bit timer is equal to zero). This sets the zero detect status (ZDS) bit in the timer status register. It may be checked by the processor or may be used to generate a timer interrupt. The ZDS bit can be reset by writing a one to the timer status register in that bit position independent of timer operation.

The general operation of the timer is flexible and easily programmable. The timer is fully configured and controlled by programming the 8-bit timer control register. It controls 1) the choice between Simple I/O operation and the timer operation of the three timer pins, 2) whether the counter is loaded from the counter preload registers or rolls over when zero detect is reached, 3) the clock input, 4) whether the prescaler is used and 5) whether the timer is enabled.



### 2.6.3 SMTsettout

#### Syntax

```
error_code SMTsettout(u_int8 mode);
```

#### Description

This function sets the timer output (TOUT) control.

#### Input

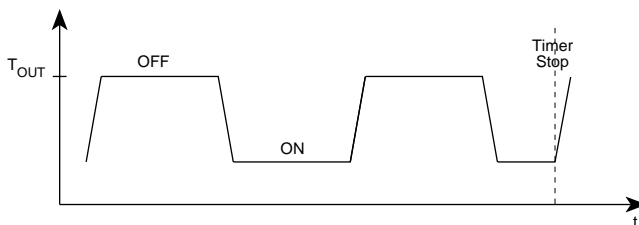
<code>u_int8 mode</code>	Two modes of TOUT control are available. They are: <code>MODE_PORTC</code> <code>tout</code> has the Simple I/O function. <code>MODE_SQUARE</code> <code>tout</code> toggles on counter zero. When the timer is stopped, <code>tout</code> is high (see diagram).
--------------------------	--

#### Output

<code>error_code</code>	<code>SUCCESS</code> <code>E_BMODE</code> Unsupported mode or standard OS-9 error code (refer to the OS-9 Technical Manual Error Codes Section).
-------------------------	---

#### Example

```
RetVal = SMTsettout(MODE_SQUARE);
```



The high state of TOUT illustrated in the above diagram shows that the output is deactivated i.e. OFF. Only when TOUT is in the low state can a load be driven.



## 2.6.4 SMTpre

### Syntax

```
error_code SMTpre(u_int32 *value);
```

### Description

This function reads / sets the timer preload register.

### Input

<code>u_int32 *value</code>	Pointer to a variable that holds the value to set. The previous value is returned to the variable. If value is 0, only the read is performed.
-----------------------------	---

### Output

<code>error_code</code>	SUCCESS	
	E_BMODE	Requested value is out of range or standard OS-9 error code (refer to the OS-9 Technical Manual Error Codes Section).

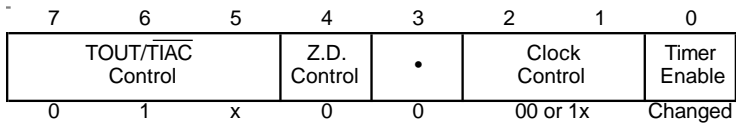
### Example

```
RetVal = SMTpre(0);
```



**Example for a Square Wave Generator**

The Timer Control Register



In this configuration, the timer produces a square wave at the TOUT pin which is connected to the user's circuitry. The TIN pin may be used as a clock input. The processor loads the counter preload registers and the timer control register and then enables the timer. When the 24-bit counter passes from \$000001 to \$000000, the ZDS status bit is set and the TOUT is toggled. At the next clock to the 24-bit counter it is again loaded with the contents of the CPRs and thereafter decrements.





### 2.6.5 SMTstasto

#### Syntax

```
error_code SMTstasto(u_int8 mode);
```

#### Description

This function starts / stops the timer; the zero-detect control bit is set at the start; the counter rolls over on reaching zero or is loaded with the value set in the preload register and continues counting down.

#### Input

u_int8 mode	Two modes are available. They are:
	TMR_START            Start timer
	TMR_STOP            Stop timer

#### Output

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

#### Example

```
RetVal = SMTstasto(TMR_START);
```



## 2.6.6 SMTrd

### Syntax

```
error_code SMTrd(u_int32 *value);
```

### Description

This function reads the actual timer count value.

## NOTE !

A stable read value can only be achieved if the timer is not running. Therefore, a read request to the running timer terminates with `E_DEVBSY`.

### Input

<code>u_int32 *value</code>	Pointer to a variable in which to place the read value.
-----------------------------	---

### Output

<code>error_code</code>	<code>SUCCESS</code>	
	<code>E_DEVBSY</code>	Timer is currently running or standard OS-9 error code (refer to the OS-9 Technical Manual Error Codes Section).

### Example

```
RetVal = SMTrd(buffer);
```





## 2.6.7 SMTtin

### Syntax

```
error_code SMTtin(u_int8 *value);
```

### Description

This function reads the current level present on TIN/PC2.

### Input

u_int8 *value	Pointer to a variable in which to place the read value. 0 represents low, 1 represents high
---------------	---

### Output

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

### Example

```
RetVal = SMTtin(buffer);
```



### 2.6.8 SMTstat

#### Syntax

```
error_code SMTstat(u_int8 *value);
```

#### Description

This function reads the timer status register and clears it if set.

#### Input

`u_int8 *value`                      Pointer to a variable in which to place the read value. 0 represents not set, 1 represents set

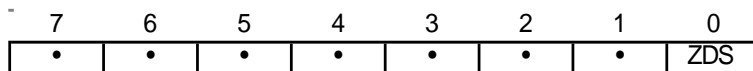
#### Output

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

#### Example

```
RetVal = SMTstat(buffer);
```

#### Description of the Timer Status Register (TSR)



The timer status register contains one bit from which the zero detect status can be determined. The ZDS status bit (bit 0) is an edge-sensitive flip-flop that is set to one when the 24-bit counter decrements from \$000001 to \$000000. The ZDS status bit is cleared to zero following the direct reset operation or when the timer is halted. This register is always readable without consequence. A write access performs a direct reset operation if bit 0 in the written data is one!



### 2.6.9 SMTout

#### Syntax

```
error_code SMTout(u_int8 *value);
```

#### Description

This function reads the actual status of the TOUT pin.

#### Input

<code>u_int8 *value</code>	Pointer to a variable in which to place the read value. 0 represents a low level while a 1 represents a high level.
----------------------------	---

#### Output

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

#### Example

```
RetVal = SMTout(buffer);
```



### 2.6.10 SMLed

#### Syntax

```
error_code SMLed(u_int8 led, u_int8 value);
```

#### Description

This function switches on / off user LED's.

#### Input

<code>u_int8 led</code>	Two options are available. They are: USERL1     Yellow LED on piggyback USERL2     Red LED on piggyback (only if the LED is not jumpered as 68302 HALT.
-------------------------	--

<code>u_int8 value</code>	Indicates the status of the LED. 0            Switch LED off Not 0        Switch LED on
---------------------------	---

#### Output

<code>error_code</code>	SUCCESS E_BMODE        LED does not exist or standard OS-9 error code (refer to the OS-9 Technical Manual Error Codes Section).
-------------------------	--

#### Example

```
RetVal = SMLed(USERL1, 0);
```



### 2.6.11 SMwdon

#### Syntax

```
error_code SMwdon(u_int32 *time);
```

#### Description

This function activates the watchdog timer of the 68302. If timeout is reached, the system is reset; in normal operating mode, this must be avoided by periodically triggering the watchdog using the function *SMwdtrig*.

#### Input

`u_int32 *time` Time in ms. The range is from 1ms to approx. 13 seconds. The function returns the real set time in `*time`.

#### Output

`error_code` SUCCESS  
E\_BMODE Time is out of range or subsequent error  
or standard OS-9 error code (refer to the OS-9 Technical Manual Error Codes Section).

#### Example

```
RetVal = SMwdon(time);
```



## 2.6.12 SMwdtrig

### Syntax

```
error_code SMwdtrig(void);
```

### Description

This function re-triggers the watchdog of the 68302 preventing a timeout and subsequent system reset.

### Output

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

### Example

```
RetVar = SMwdtrig();
```





### 2.6.13 SMwdoff

#### Syntax

```
error_code SMwdoff(void);
```

#### Description

This function deactivates the watchdog timer.

#### Output

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

#### Example

```
RetVar = SMwdoff();
```

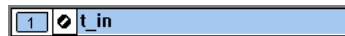


## 2.7 ISaGRAF Programming

### 2.7.1 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 2.7.1.1 Typical Screen Section for the SMART-BASE**



Because the communication to the timer/counter is performed using Operate Calls, there is no need for manual selection of such things as a logical address etc. `t_in` represents a variable for the counter timer and is used for the operate call to decide which port to access.





### 2.7.2 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. A number of these `COMMANDS` exist for the SMART-BASE.

#### 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 a SMART-Module is located where the program expects it to be.

`channel` provides channel specific information and in the example shown here, any of the 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.



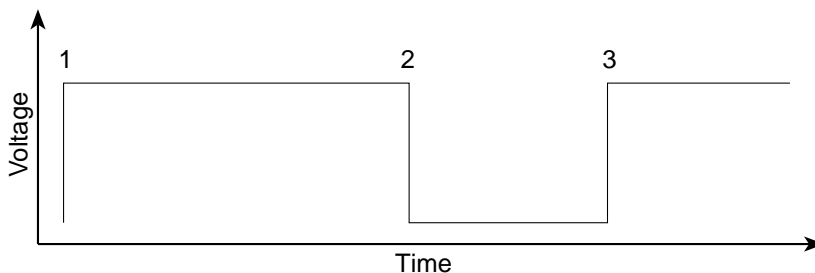
A complete list of the operate **COMMANDS** may be obtained by selecting a project from the ISaGRAF *projects* group, opening an application and observing the *Common defines* in the *Dictionary* pull-down menu. Note that not all calls in the list may be used within the SMART I/O environment however,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

**O\_INIT\_CODE** : The syntax and usage have already been explained.

**O\_POWERFAIL\_SET** : The purpose of this operate call is to detect when (if) the power to the PLC (SMART I/O) has failed. The function is normally built into the initialization stage of an application and has the following syntax :

```
RetVar := OPERATE(s_time, O_POWERFAIL_SET, 0);
```

When the application is initialized, the start time is recorded in battery backed ram (1) at the given address (*s\_time*). If the power to the PLC fails (2) and recovers at a later stage (3), the software makes a comparison with the actual clock time (via the RTC) and the time stored in this memory location. If a discrepancy exists then the RetVar will record this fact. Refer to the PEP online help for more information pertaining to the implementation of this operate call.





**O\_START\_COUNTER** : This call starts the counter; it's syntax is as follows :

```
<RetVar> := OPERATE(<iovar>, O_START_COUNTER, <>null>);
```

where the <iovar> is typically t\_in.

**O\_READ\_COUNTER** : With this call the contents of the counter register may be read. When this call is issued, the counter is stopped, it's register read and then restarted. If a high-frequency input exists then pulses may be lost (not counted). The same effect may also be true due to timeslicing during timer stop and start operations. Therefore it is recommended that this call be only used for low frequency inputs (<1kHz). There is no detection for counter overflow and the call should not be used for count down operations or square wave generation. It's syntax is as follows :

```
<RetVar> := OPERATE(<iovar>, O_READ_COUNTER, <>null>);
```

where the <iovar> is typically t\_in.

**O\_STOP\_COUNTER** : This call stops the counter counting up or down and square-wave generation and has the following syntax :

```
<RetVar> := OPERATE(<iovar>, O_STOP_COUNTER, <>null>);
```

where the <iovar> is typically t\_in.

The counter/timer is configured automatically as MODE10 (refer to the 'C' Programming section) and from within ISaGRAF there is no ability to alter this configuration.



**O\_PRELOAD** : With this call the counter preload register can be set. The syntax is as follows :

```
<RetVar> := OPERATE(<iovar>, O_PRELOAD, <prevar>);
```

where the <iovar> is typically t\_in and <prevar> is the value for the preload register which lies between 1 and 0xFFFFF.

**O\_START\_CNTDWN** : This function starts the counter counting-down. When the counter reaches zero, it rolls over to the maximum value of 0xFFFFF on the following clock pulse and starts afresh. The syntax is as follows :

```
<RetVar> := OPERATE(<iovar>, O_START_CNTDWN, <prevar>);
```

where the <iovar> is a variable attached to the Timer I/O (typically t\_in) and <prevar> may be one of OA\_PITCLOCK for selection of the 6 MHz/32 clock input or OA\_TIN for selection of t\_in inputs.





## 2.8 Flash Utility

The flash utility allows ISaGRAF applications, a new ISaGRAF kernel or other OS-9 modules to be loaded into FLASH memory. The SMART I/O may be equipped with 1 MByte of memory depending on the version ordered. If an application is stored in FLASH then it will be loaded into the system at start-up.

The main features of this utility are:

- Erase memory
- Fill memory
- Append memory
- Read memory

The following syntax should be observed:

```
flash -b[board] <parameter(s)>
```

### Examples

To erase the FLASH:

```
Isa: flash -b=SMART -p=ff
```

To write a file to FLASH (base address is \$D00000):

```
flash -b=SMART -d=d00000 -i=path/filename
```

To append a file to existing FLASH contents:

```
flash -b=SMART -d=d00000 -u -i=path/filename
```



### Example to download ISA11 module and store in FLASH

Start the isa terminal in MS Windows:

Press the <ENTER> key three times to display the following screen prompt:

```
Isa:
```

Start kermit on the target system

```
Isa: kermit ri <ENTER>
```

Select the menu item *transfers-send binary file* from the windows terminal and select the file *flash* on the PC to transfer

Load *flash* into memory

```
Isa: load -ld/dd/APL/flash
```

Start kermit on the target system again

```
Isa: kermit ri <ENTER>
```

Select the menu item *transfers-send binary file* from the windows terminal and select the file *isa11* on the PC to transfer

Append the file to the existing flash contents

```
flash -b=SMART -d=d00000 -u -i=/dd/APL/isa11
```



# Table Of Contents

- 3. SMART-EXT .....3-3
  - 3.1 Specifications ..... 3-3
  - 3.2 Board Overview ..... 3-4
  - 3.3 Functional Description ..... 3-5
  - 3.4 Pinouts ..... 3-6
    - 3.4.1 SMART Module Piggyback Connectors ..... 3-7
    - 3.4.2 Parallel I/O Screw Terminals (SCRA and SCRB) ..... 3-8
    - 3.4.3 SPI Connectors (ST5 and BU1) ..... 3-9





This page has been left blank intentionally





## 3. SMART-EXT



The SMART-EXT is a carrier unit enabling the connection of a further 2 SMART-Modules thereby enhancing the I/O capacity of the SMART I/O system. Up to 4 of these extensions may be cascaded via a 10-wire flat-band cable with integrated 3-wire SPI (Serial Peripheral Interface). To achieve a common interface between modules, a SMART I/O 'C' library is provided by PEP.

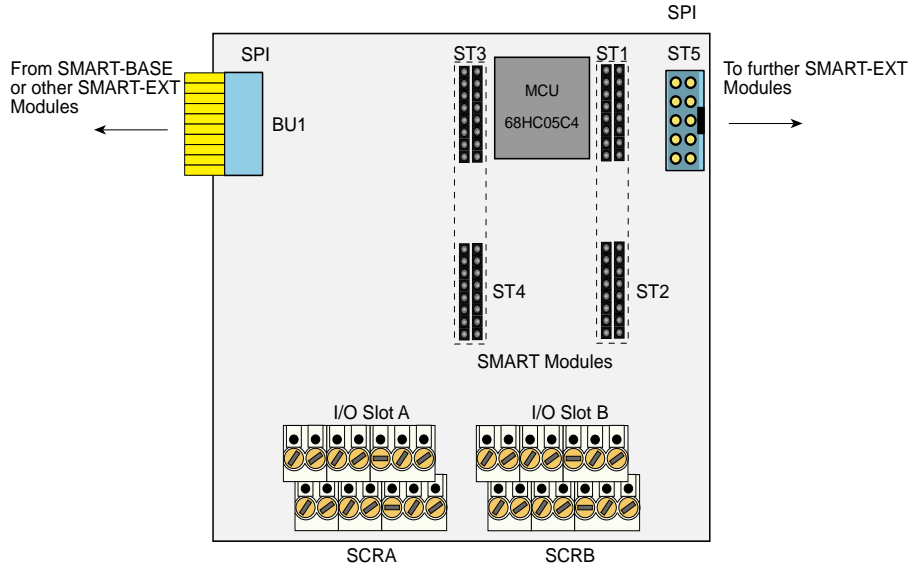
### 3.1 Specifications

Controller	MC68HC705C8A
- Frequency	4 MHz
- Firmware	PEP Firmware for OTP device
Interface Speed	Set to 1 MHz
Power Consumption	Typ. 25 mW
Temperature Range	
- Storage	-20°C to + 125°C
- Standard	0°C to +70°C
- Extended	-40°C to +85°C
Operating Humidity	0 to 95% non-condensing
Weight	260g (without modules, blank panels or screw terminals)

The SPI follows a Motorola defined communication protocol which is beyond the scope of this manual to describe. More information may be found by referring to the relevant data-sheets.



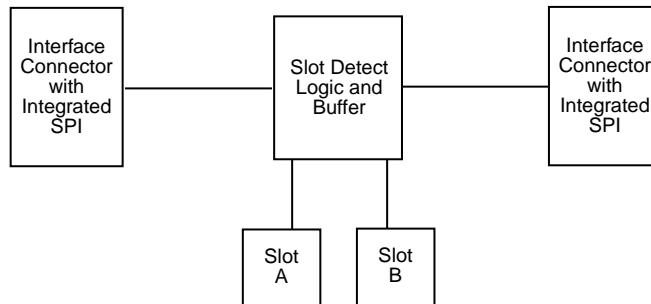
### 3.2 Board Overview





### 3.3 Functional Description

**Figure 3.3.1: SMART-EXT Block Diagram**



The SMART-EXT is a carrier board for up to 2 SMART-Modules with data transfer between SMART-BASE and EXT units being performed by the interface connector incorporating the Motorola synchronous Serial Peripheral Interface (SPI).

The maximum bit transfer rate is set to 1 MHz which ensures a typical response time from an addressed extension slot of approximately 50 $\mu$ s.

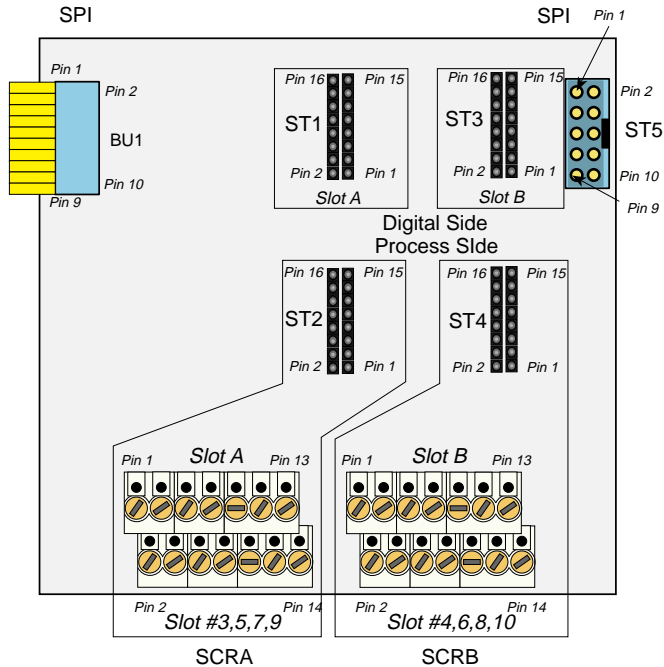
The MCU (Micro Controller Unit - 68HC05C4) handles the SPI transfers and the slot-dependent commands and actions through the SMART-EXT firmware burned into the MCU mask or in an OTP (One-Time Programmable) PROM area.


In order to communicate with a particular SMART-EXT slot, a specific PEP defined protocol has to be observed. All necessary functions are provided in the OS-9 'C' library SMAC.L which should be included when writing a specific C' application program.




### 3.4 Pinouts

Figure 3.4.0.1: SMART-EXT Pinout Overview





## WARNING !



Dangerous voltages may be present at the terminals.



### 3.4.1 SMART Module Piggyback Connectors

There are two sets (one for each module) of SMART-Module piggyback connectors present on the SMART-EXT, each divided into two sets of 2x8 standard pin rows. The communication to these connectors is achieved in part via the 10-wire flat-band interface cable and directly by the MCU.

#### Pinouts digital side (ST1 and ST3)

Refer to figure 3.4.0.1 for the correct location of these pin-row connectors.

*Slot A from 68HC05C4 MCU port A (ST1)*

Pin Nr.	Signal	Description	Pin Nr.	Signal	Description
1	PA1	Port A Pin 1	2	PA2	Port A Pin 2
3	PA3	Port A Pin 3	4	PA4	Port A Pin 4
5	PA5	Port A Pin 5	6	PA6	Port A Pin 6
7	PA7	Port A Pin 7	8	PA8	Port A Pin 8
9	System GND	GND	10	Serial RxD	RxD to SM-BASE
11	System VCC	System +5V VCC	12	Serial TxD	TxD from SM-BASE
13	CS-SLOTA	Chip Select A	14	Serial CLK	CLK from SM-BASE
15	Reset	Power ON/OFF Reset	16	Slot A Interrupt line	INT4 to the I/O Controller

*Slot B from 68HC05C4 MCU port B (ST3)*

Pin Nr.	Signal	Description	Pin Nr.	Signal	Description
1	PB1	Port B Pin 1	2	PB2	Port B Pin 2
3	PB3	Port B Pin 3	4	PB4	Port B Pin 4
5	PB5	Port B Pin 5	6	PB6	Port B Pin 6
7	PB7	Port B Pin 7	8	PB8	Port B Pin 8
9	System GND	GND	10	Serial RxD	RxD from SM-BASE
11	System VCC	System +5V VCC	12	Serial TxD	TxD from SM-BASE
13	CS-SLOTB	Chip Select B	14	Serial CLK	CLK from SM-BASE
15	Reset	Power ON/OFF Reset	16	Slot B Interrupt line	Interrupt P/I/T (H4)



### Pinouts process side (ST2 and ST4)

Refer to figure 3.4.0.1 for the correct location of these pin-row connectors.

*Slots A and B (ST2 and ST4)*

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

The PC board connections to the screw terminals are capable of absorbing a continuous current of up to 3A each. However, pins 13 and 14 can support up to 6 Amps.



### 3.4.2 Parallel I/O Screw Terminals (SCRA and SCRB)

The screw terminal blocks are individually composed of 7 free connections which, when stacked provide 14 free connections per I/O slot. The pinout functionality depends on the type of SMART-Modules fitted and the relevant module should be referred to in the appropriate section of this manual.



### 3.4.3 SPI Connectors (ST5 and BU1)

On both sides of the board are standard 2x5 pinrow connectors (BU1, ST5), that provide the interface connection between SMART-EXT units and the SMART-BASE.

A 10-wire flat cable is soldered directly on the left-hand side of the board (BU1), that interfaces the SMART-BASE module or other (earlier cascaded) SMART-EXT modules.

#### Pinouts for this 10-pin connector

Pin Nr.	Signal	Pin Nr.	Signal
1	System VCC (+5V)	2	Serial RxD from SMART-BASE
3	System VCC (+5V)	4	Serial TxD from SMART-BASE
5	Serial Ext. Select	6	Serial CLK from SMART-BASE
7	System GND	8	Reset (Power ON/OFF)
9	System GND	10 (BU1)	Serial Ext. Interrupt (INT4 1st Module only)
		10 (ST5)	N/C

#### Note

Pin 10 of BU1 is only useful on the first SMART-EXT connected to the SMART-BASE. For this reason, SMART-Modules utilizing this feature can only be connected to the first slot of the first extension unit.



This page has been left blank intentionally.







---

*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 Pinouts ..... 4-10

    4.1.8 'C' Programming ..... 4-12

    4.1.9 ISaGRAF Programming ..... 4-16

4.2 SM-DOOUT1 ..... 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

---



4.2.7 Pinouts .....	4-22
4.2.9 ISaGRAF Programming .....	4-29
<b>4.3 SM-REL1 .....</b>	<b>4-33</b>
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
<b>4.4 SM-ACI1 .....</b>	<b>4-53</b>
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

This page was intentionally left blank.

## 4. Digital Modules

### 4.1 SM-DIN1

#### 4.1.1 Introduction

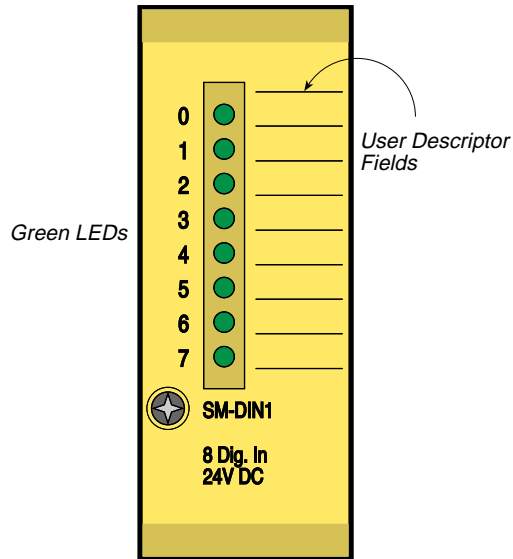
The SM-DIN1 provides 8 optoisolated 24V DC digital inputs arranged in 6 independent groups with respect to the ground connections. The maximum input switching frequency is set to 200Hz with the system registering a logical '1' when the input exceeds 10V. Logical '0' is returned when the input falls below 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 $\pm$ 10% 6 Independent Groups Common GND < 5mA Input Current Filter set to 200 Hz (5ms) Overvoltage Protection (rate) Overvoltage Protection (continuous) : 300V/90 $\mu$ s Switch ON Delay : 35V Switch OFF Delay : 2ms (approx.) High Level : 2ms (approx.) Low Level : > 10V : < 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



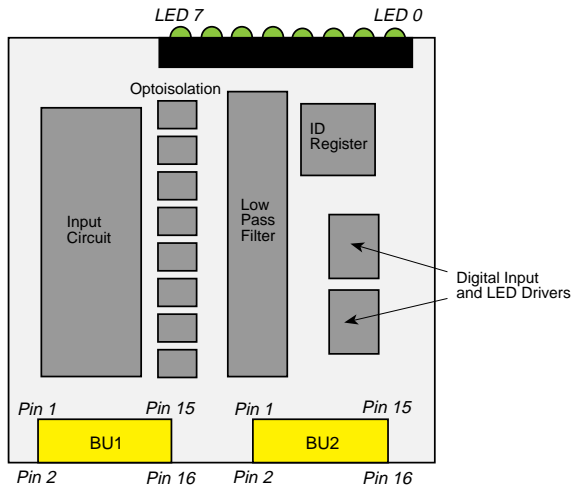
### 4.1.3 Front Panel Layout



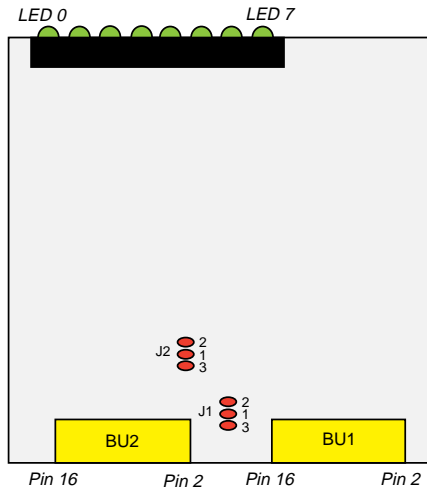


### 4.1.4 Board Overview

#### Component Side



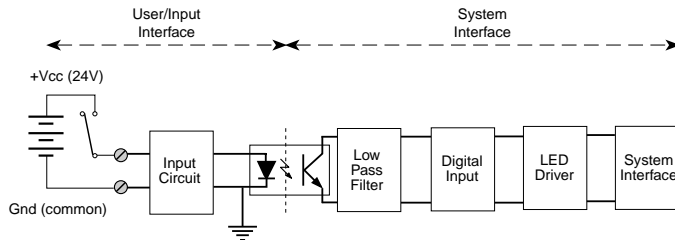
#### Solder Side





### 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

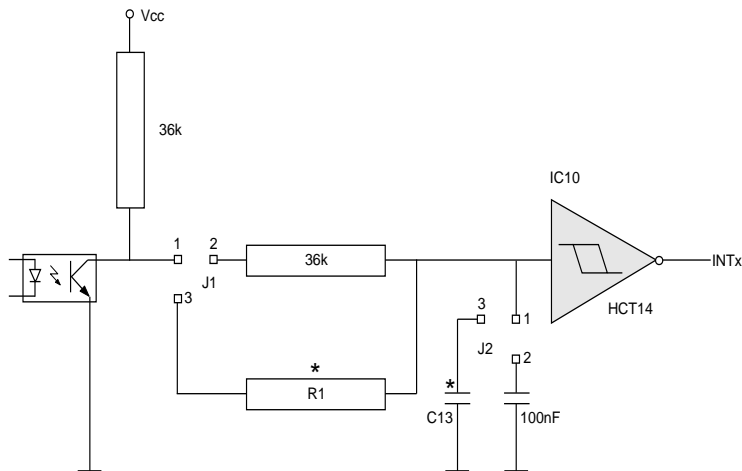




### 4.1.6 Configuration

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**



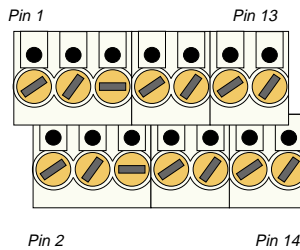
Surface Mounted Devices (SMD) are used in the production of the SM-DIN1 modules. The components to be changed (marked with an asterisk in figure 4.1.6.1), need not necessarily be SMD. Refer to the Board Overview (solder side) for the approximate position of these jumpers. It should be noted that when 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.

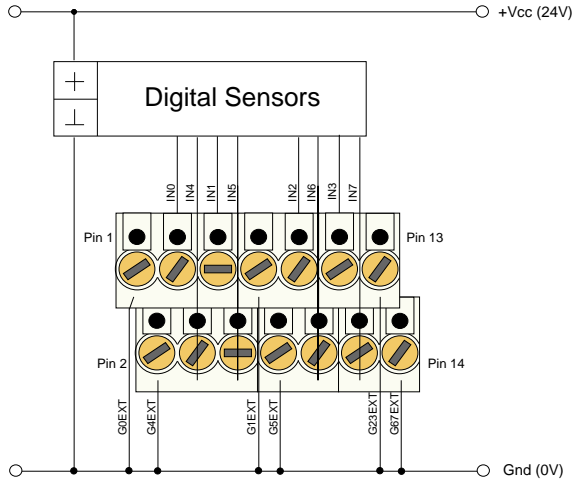


Pin Nr.	Signal	Description	Pin Nr.	Signal	Description
1	G0EXT	GND for channel 0	2	G4EXT	GND for channel 4
3	IN0	Input Channel 0	4	IN4	Input Channel 4
5	IN1	Input Channel 1	6	IN5	Input Channel 5
7	G1EXT	GND for channel 1	8	G5EXT	GND for channel 5
9	IN2	Input Channel 2	10	IN6	Input Channel 6
11	IN3	Input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

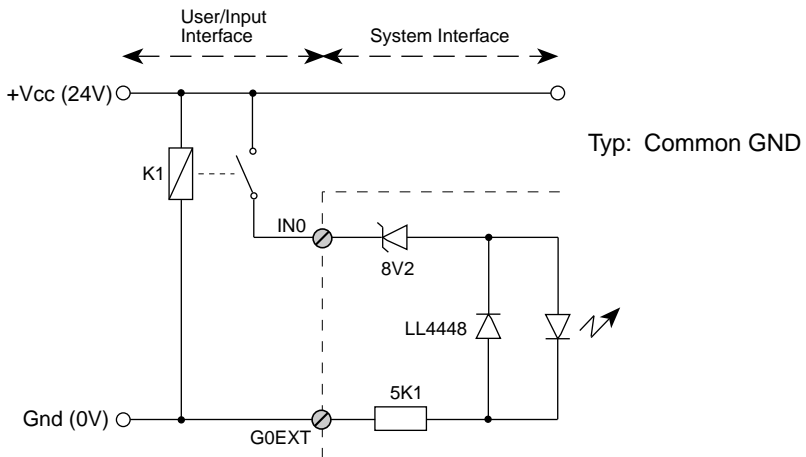




**Connection**



**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

- SMART I/O Base Module or Base Module and Extension unit;
- 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*).



### 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

<code>u_int8 PortNr</code>	SM-Port number to initialise (from 0 to 10)
----------------------------	--

#### Output

<code>error_code</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);
```





#### 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
u_int8 *buffer
```

SM-Port number to fetch data  
Pointer to Buffer where data is to be stored

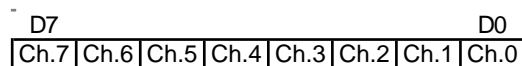
##### Output

```
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 = 2	
1	<input checked="" type="checkbox"/> IN_0
2	<input checked="" type="checkbox"/> IN_1
3	<input checked="" type="checkbox"/> IN_2
4	<input checked="" type="checkbox"/> IN_3
5	<input checked="" type="checkbox"/> IN_4
6	<input checked="" type="checkbox"/> IN_5
7	<input checked="" type="checkbox"/> IN_6
8	<input checked="" type="checkbox"/> IN_7

### 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.



---

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** : The syntax and usage have already been explained.





## 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.

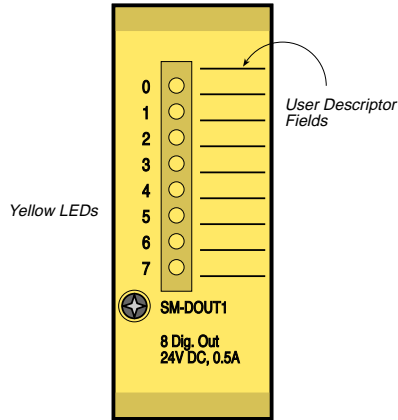
### 4.2.2 Specifications

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



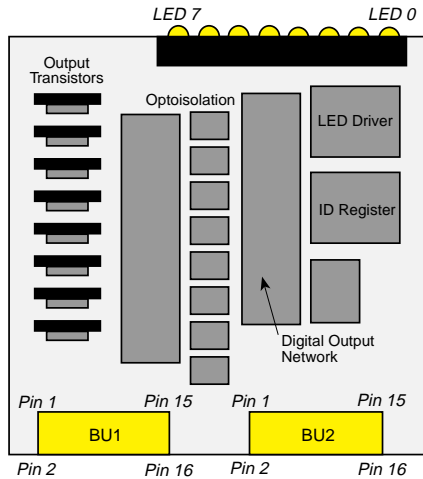


### 4.2.3 Front Panel Layout



### 4.2.4 Board Overview

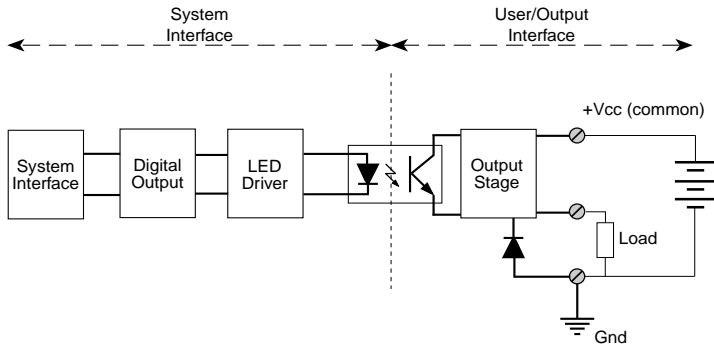
#### Component Side





### 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.



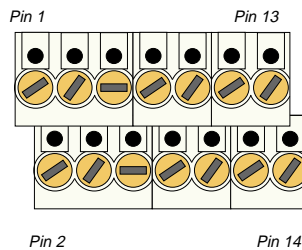
### 4.2.6 Configuration

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.

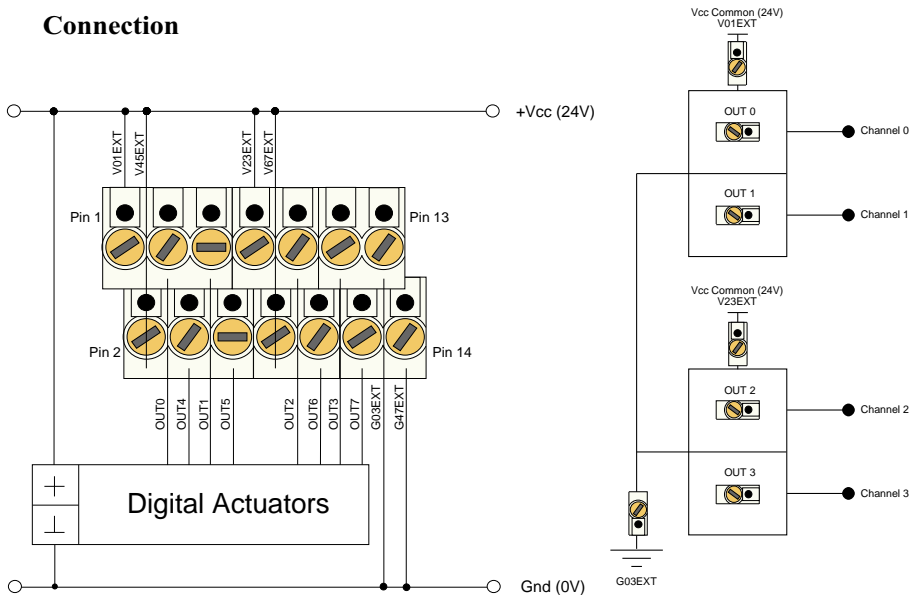


Pin Nr.	Signal	Description	Pin Nr.	Signal	Description
1	V01EXT	Vcc for channels 0 & 1	2	V45EXT	Vcc for channels 4 & 5
3	OUT0	Output 0	4	OUT4	Output 4
5	OUT1	Output 1	6	OUT5	Output5
7	V23EXT	Vcc for channels 2 & 3	8	V67EXT	Vcc for channels 6 & 7
9	OUT2	Output 2	10	OUT6	Output 6
11	OUT3	Output 3	12	OUT7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

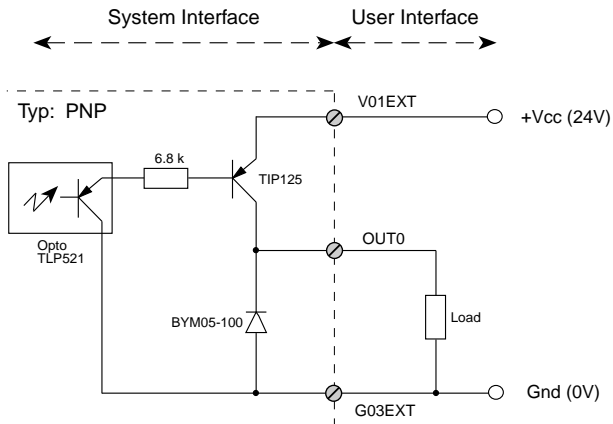




**Connection**



**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

- SMART I/O Base Module or Base Module and Extension unit;
- 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*).



**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**

<code>u_int8 PortNr</code>	SM-Port number to initialize (from 0 to 10)
----------------------------	--

**Output**

<code>error_code</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

<code>u_int8 PortNr</code>	SM-Port number to de-initialize
----------------------------	---------------------------------

#### Output

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

#### Example

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



### 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.

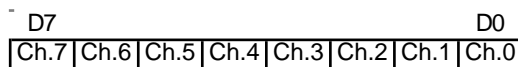
#### Output

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

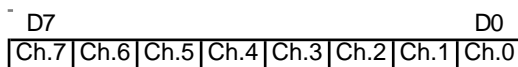
**Output**

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 = 1
reset_on_close = true
1 <input checked="" type="checkbox"/> OUT_0
2 <input checked="" type="checkbox"/> OUT_1
3 <input checked="" type="checkbox"/> OUT_2
4 <input checked="" type="checkbox"/> OUT_3
5 <input checked="" type="checkbox"/> OUT_4
6 <input checked="" type="checkbox"/> OUT_5
7 <input checked="" type="checkbox"/> OUT_6
8 <input checked="" type="checkbox"/> OUT_7

### 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.



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** : The syntax and usage have already been explained.



This page has been left blank intentionally.







## 4.3 SM-REL1

### 4.3.1 Introduction

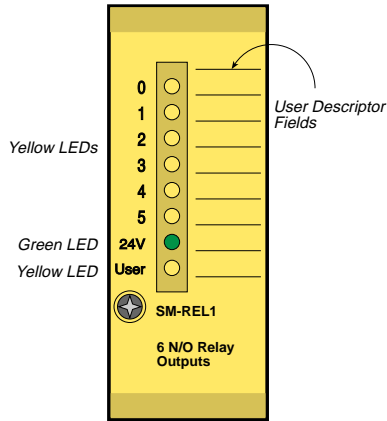
The SM-REL1 provides power switching up to 250V AC or 220V DC with surge protection and the ability to switch inductive loads being built-in features of this versatile module. A freely programmable LED on the front panel provides the user interface together with 6 relay status LEDs and an external 24V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

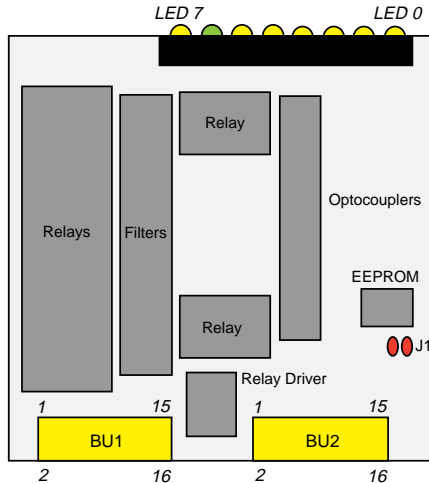


### 4.3.3 Front Panel Layout



### 4.3.4 Board Overview

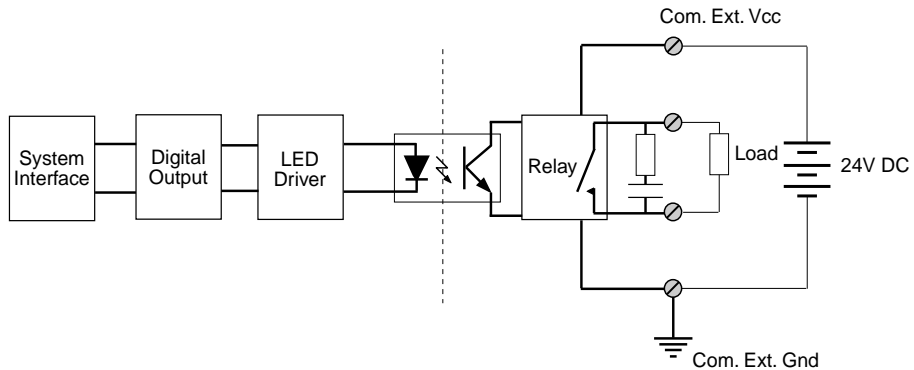
#### *Component Side*





### 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

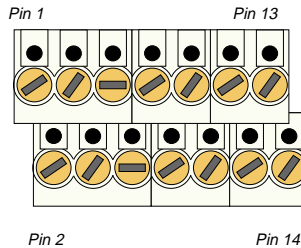
#### Jumper J1 - EEPROM Protection

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

### 4.3.7 Pinouts

#### Screw Terminal Pinouts

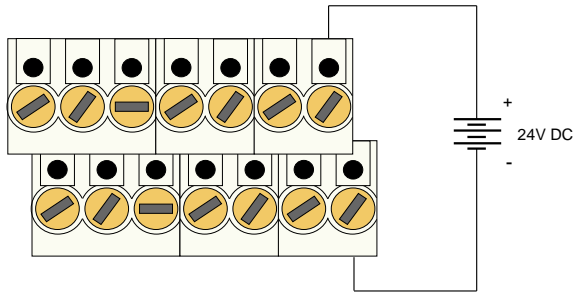
The following shows the pinout/signal relationship for the SM-REL1 when connected to a particular screw terminal block.



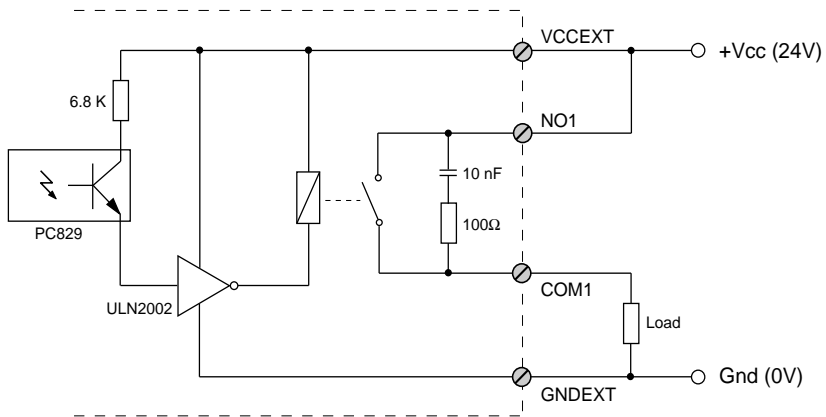
Pin Nr.	Signal	Description	Pin Nr.	Signal	Description
1	NO0	N/O Relay 0 Conn.	2	COM0	Relay 0 Common Conn.
3	NO1	N/O Relay 1 Conn.	4	COM1	Relay 1 Common Conn.
5	NO2	N/O Relay 2 Conn.	6	COM2	Relay 2 Common Conn.
7	NO3	N/O Relay 3 Conn.	8	COM3	Relay 3 Common Conn.
9	NO4	N/O Relay 4 Conn.	10	COM4	Relay 4 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



**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

- SMART I/O Base Module or Base Module and Extension unit;
- 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 as 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 *demrell.c* can be found in the SMART I/O application directory (normally found in */<device>/APPLIC/SMART/CMDS*).

**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**

<code>u_int8 PortNr</code>	SM-Port to initialize (from 0 to 10)
----------------------------	---

**Output**

<code>error_code</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

<code>u_int8 PortNr</code>	SM-Port number to de-initialize (from 0 to 10)
----------------------------	---

#### Output

<code>error_code</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 SMREL1DeInit function call but it does ensure that the relay status (and LEDs) is as expected.

##### Input

<code>u_int8 PortNr</code>	Number of the SM port to reset
----------------------------	--------------------------------

##### Output

<code>error_code</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);
```



#### 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

<code>u_int8 PortNr</code>	SM-Port number to set data
<code>u_int8 *buffer</code>	Pointer to buffer where data is stored

##### Output

<code>error_code</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);
```



#### 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);
```



## 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 = 3
reset_on_close = false
1 led1
2 led2
3 led3
4 led4
5 led5
6 led6

### 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.



---



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

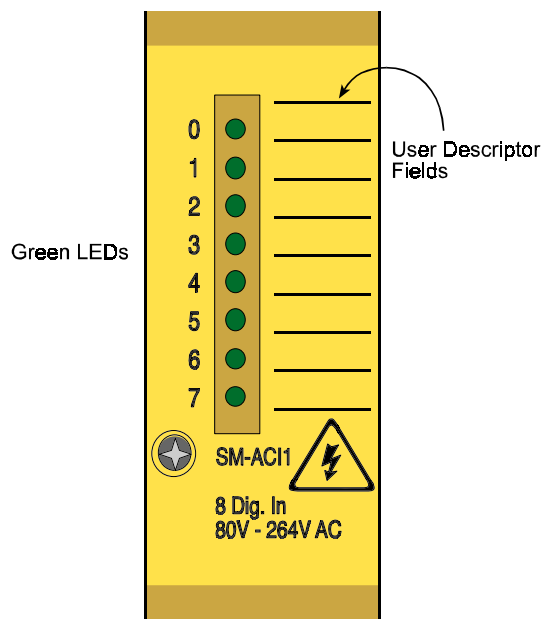
**Table 4-1: Specifications**

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 < 21 ms Switch-ON delay: < 16 ms Switch-OFF delay: > 20 V High level: < 10 V Low level:
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

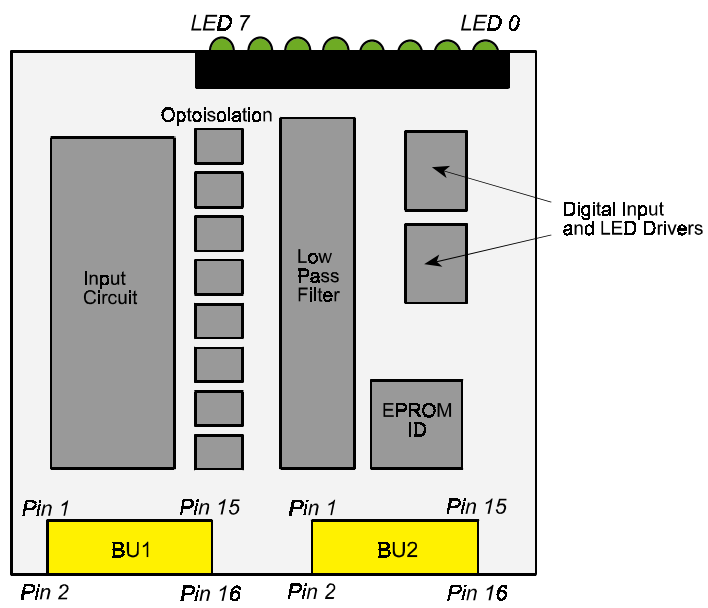
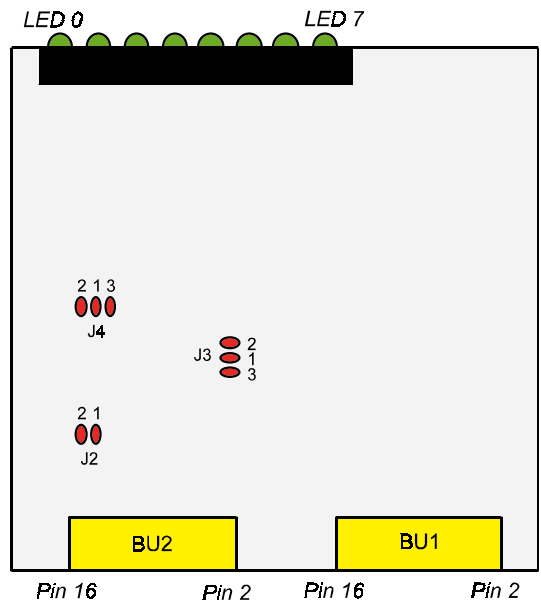


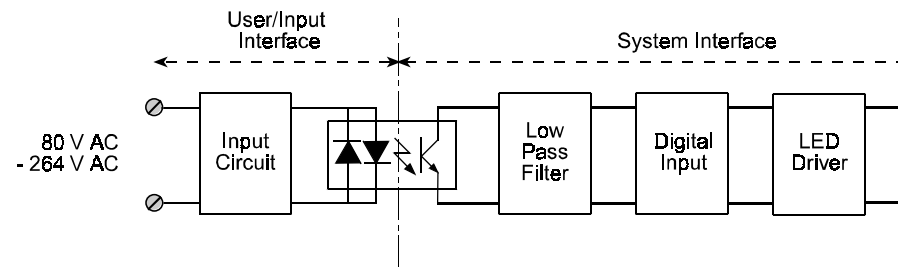


Figure 4-3: Solder Side



4.4.5 Functional Description

Figure 4-4: SM-AC11 Schematic Diagram





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.

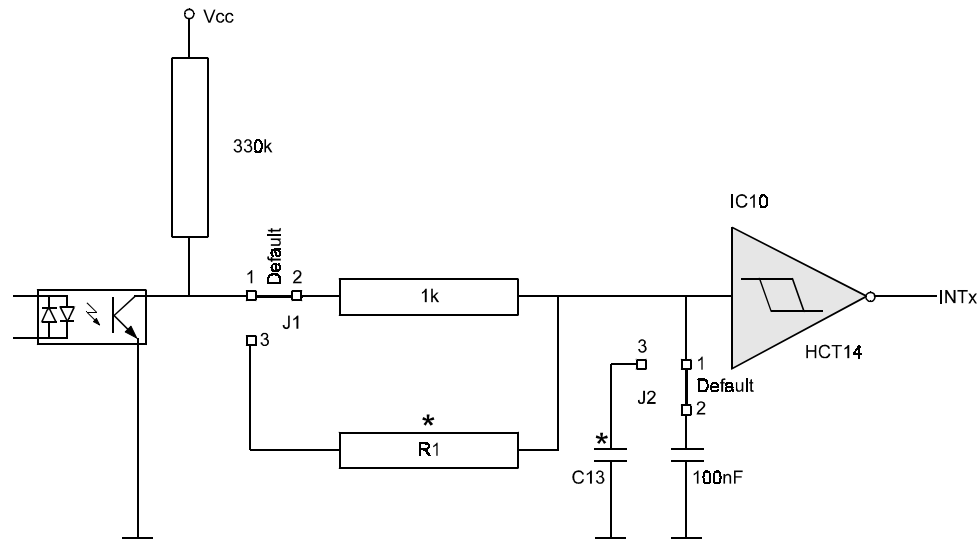
**Table 4-2: Jumper Settings**

Jumper Settings		Description
J3	1-2	Standard input filter 40 Hz
	1-3	Customer-specific input filter
J4	1-2	Standard input filter 40 Hz
	1-3	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-AC11 Configurable Filter



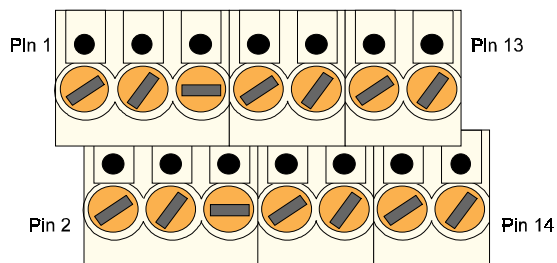


Surface-Mounted Devices (SMDs) are used in the production of SM-AC11 modules. The components to be changed, marked with an asterisk in figure 4-5, need not necessarily be SMDs. Please, refer to the board overview figure (solder side, figure 4-3) for the approximate position of the jumpers. Notice should 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-AC11, when connected to a particular screw terminal block, is shown.

**Figure 4-6: Screw Terminal Pinout**



WARNING!

*Dangerous voltages are present at the terminals.*

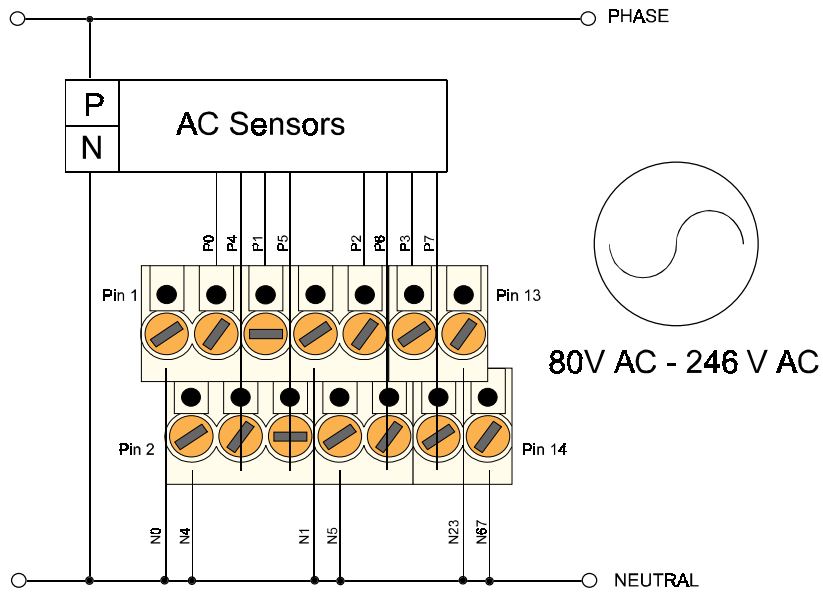


**Table 4-3: Pinout/Signal Relationship**

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

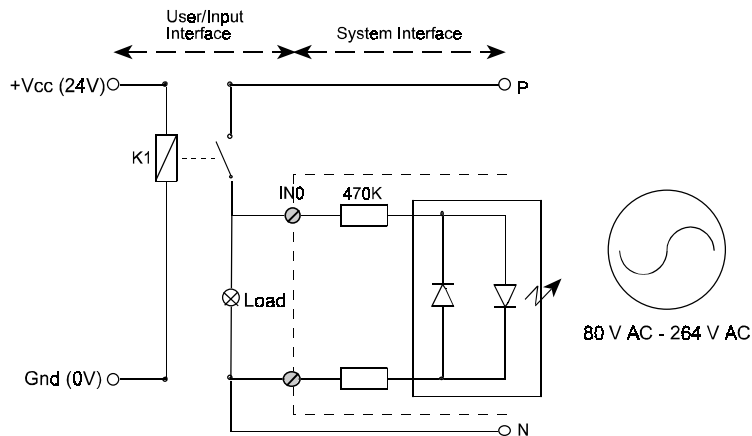


Figure 4-7: Connection





**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 *acilib.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 = 2	
1	<input checked="" type="radio"/>	IN_0	
2	<input checked="" type="radio"/>	IN_1	
3	<input checked="" type="radio"/>	IN_2	
4	<input checked="" type="radio"/>	IN_3	
5	<input checked="" type="radio"/>	IN_4	
6	<input checked="" type="radio"/>	IN_5	
7	<input checked="" type="radio"/>	IN_6	
8	<input checked="" type="radio"/>	IN_7	





### 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.



# *Analog Modules*

*for*

# *SMART I/O*

5



# Table of Contents

<b>5. Analog Modules</b> .....	<b>5-7</b>
5.1 SM-DAD1 .....	5-7
5.1.1 Introduction .....	5-7
5.1.2 Specifications .....	5-7
5.1.3 Front Panel Layout .....	5-8
5.1.4 Board Overview .....	5-9
5.1.5 Functional Description .....	5-10
5.1.5.1 Input Circuitry .....	5-10
5.1.5.2 Output Circuitry .....	5-11
5.1.6 Configuration .....	5-12
5.1.7 Pinouts .....	5-13
5.1.8 'C' Programming .....	5-16
5.1.8.1 SM-DAD1 Library .....	5-16
5.1.8.2 SMDAD1Init .....	5-17
5.1.8.3 SMDAD1DeInit .....	5-18
5.1.8.4 SMDAD1GetVRaw .....	5-19
5.1.8.5 SMDAD1GetV .....	5-20
5.1.8.6 SMDAD1PutVRaw .....	5-21
5.1.8.7 SMDAD1PutV .....	5-22
5.1.8.8 SMDAD1SetLed .....	5-23
5.1.8.9 SMDAD1ClrLed .....	5-24
5.1.9 ISaGRAF Programming .....	5-25
5.1.9.1 The ISaGRAF Board Parameters .....	5-25
5.1.9.2 The ISaGRAF Operate Calls .....	5-26
5.2 SM-PT100 .....	5-29
5.2.1 Introduction .....	5-29
5.2.2 Specifications .....	5-29
5.2.3 Front Panel Layout .....	5-30
5.2.4 Board Overview .....	5-30
5.2.5 Functional Description .....	5-31
5.2.6 Configuration .....	5-33



5.2.7 Pinouts .....	5-33
5.2.8 'C' Programming .....	5-35
5.2.8.1 SM-PT100 Library .....	5-35
5.2.8.2 SMADCInit .....	5-39
5.2.8.3 SMADCCalibrate .....	5-40
5.2.8.4 SMADCSetCyclicCalib .....	5-41
5.2.8.5 SMADCSetSensorType .....	5-42
5.2.8.6 SMADCGetSensorType .....	5-43
5.2.8.7 SMADCSetPrecision .....	5-44
5.2.8.8 SMADCGetPrecision .....	5-45
5.2.8.9 SMADCSetMode .....	5-46
5.2.8.10 SMADCGetMode .....	5-47
5.2.8.11 SMADCSetSignal .....	5-48
5.2.8.12 SMADCSetGain .....	5-49
5.2.8.13 SMADCGetGain .....	5-50
5.2.8.14 SMADCEnableRead .....	5-51
5.2.8.15 SMADCEnableConversion .....	5-52
5.2.8.16 SMADCReadRaw .....	5-53
5.2.8.17 SMADCReadConverted .....	5-54
5.2.8.18 SMADCDeinit .....	5-55
5.2.9 ISaGRAF Programming .....	5-56
5.2.9.1 The ISaGRAF Board Parameters .....	5-56
5.2.9.2 The ISaGRAF Operate Calls .....	5-57
5.3 SM-THERM .....	5-59
5.3.1 Introduction .....	5-59
5.3.2 Specifications .....	5-59
5.3.3 Front Panel Layout .....	5-60
5.3.4 Board Overview .....	5-60
5.3.5 Functional Description .....	5-61
5.3.6 Configuration .....	5-62
5.3.7 Pinouts .....	5-62
5.3.8 'C' Programming .....	5-64
5.3.8.1 SM-THERM Library .....	5-64
5.3.8.2 SMADCInit .....	5-68
5.3.8.3 SMADCCalibrate .....	5-69
5.3.8.4 SMADCSetCyclicCalib .....	5-70



5.3.8.5 SMADCSetSensorType .....	5-71
5.3.8.6 SMADCGetSensorType .....	5-72
5.3.8.7 SMADCSetPrecision .....	5-73
5.3.8.8 SMADCGetPrecision .....	5-74
5.3.8.9 SMADCSetMode .....	5-75
5.3.8.10 SMADCGetMode .....	5-76
5.3.8.11 SMADCSetSignal .....	5-77
5.3.8.12 SMADCSetGain .....	5-78
5.3.8.13 SMADCGetGain .....	5-79
5.3.8.14 SMADCEnableRead .....	5-80
5.3.8.15 SMADCEnableConversion .....	5-81
5.3.8.16 SMADCReadRaw .....	5-82
5.3.8.17 SMADCReadConverted .....	5-83
5.3.8.18 SMADCDeinit .....	5-84
5.3.9 ISaGRAF Programming .....	5-85
5.3.9.1 The ISaGRAF Board Parameters .....	5-85
5.3.9.2 The ISaGRAF Operate Calls .....	5-86
5.4 SM-ADC1 .....	5-89
5.4.1 Introduction .....	5-89
5.4.2 Specifications .....	5-89
5.4.3 Front Panel Layout .....	5-90
5.4.4 Board Overview .....	5-90
5.4.5 Functional Description .....	5-91
5.4.5.1 Input Circuitry .....	5-91
5.4.6 Configuration .....	5-92
5.4.7 Pinouts .....	5-92
5.4.8 'C' Programming .....	5-94
5.4.8.1 SM-ADC1 Library .....	5-94
5.4.8.2 SMADC1Init .....	5-95
5.4.8.3 SMADC1GetVRaw .....	5-96
5.4.8.4 SMADC1GetV .....	5-97
5.4.8.5 SMADC1SetLed .....	5-98
5.4.8.6 SMADC1ClrLed .....	5-99
5.4.8.7 SMADC1Deinit .....	5-100
5.4.9 ISaGRAF Programming .....	5-101
5.4.9.1 The ISaGRAF Board Parameters .....	5-101





- 5.4.9.2 The ISaGRAF Operate Calls ..... 5-102
- 5.5 SM-DAC1 ..... 5-107
  - 5.5.1 Introduction ..... 5-107
  - 5.5.2 Specifications ..... 5-107
  - 5.5.3 Front Panel Layout ..... 5-108
  - 5.5.4 Board Overview ..... 5-108
  - 5.5.5 Functional Description ..... 5-109
    - 5.5.5.1 Output Circuitry ..... 5-109
  - 5.5.6 Configuration ..... 5-110
  - 5.5.7 Pinouts ..... 5-110
  - 5.5.8 'C' Programming ..... 5-113
    - 5.5.8.1 SM-DAC1 Library ..... 5-113
    - 5.5.8.2 SMDAC1Init ..... 5-114
    - 5.5.8.3 SMDAC1OpenLoop ..... 5-115
    - 5.5.8.4 SMADAC1Operate ..... 5-116
    - 5.5.8.5 SMDAC1StandBy ..... 5-117
    - 5.5.8.6 SMDAC1PutVRaw ..... 5-118
    - 5.5.8.7 SMDAC1PutV ..... 5-119
    - 5.5.8.8 SMDAC1SetLed ..... 5-120
    - 5.5.8.9 SMDAC1ClrLed ..... 5-121
    - 5.5.8.10 SMDAC1DeInit ..... 5-122
  - 5.5.9 ISaGRAF Programming ..... 5-123
    - 5.5.9.1 The ISaGRAF Board Parameters ..... 5-123
    - 5.5.9.2 The ISaGRAF Operate Calls ..... 5-124





This page has been left blank intentionally.





# 5. Analog Modules

## 5.1 SM-DAD1

### 5.1.1 Introduction

The SM-DAD1 provides a fast, 12-bit, bipolar ( $\pm 10V$  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.

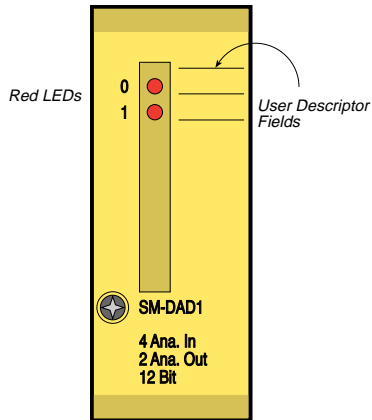
### 5.1.2 Specifications

Input	4 Differential Voltage Inputs Input Range $\pm 10V$ 12-bit ADC with 1-bit non-linearity 10-bit repeating accuracy Overvoltage Protection ( $\pm 35V$ ) Input Impedance approx. 450 k $\Omega$ Filter set to 10kHz On-chip Reference Voltage 13 $\mu s$ Conversion Time @ 1MHz SCLK
Output	2 Unipolar / Bipolar Outputs Short-Circuit Protected 12-bit DAC with 1-bit non-linearity Output Range $\pm 10V$ or 0..10V 5mA max. Output Current Power-up Reset without glitch (Unipolar) 4mV Zero Code Offset Error (Unipolar) Settling Time 40 $\mu s$ (-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





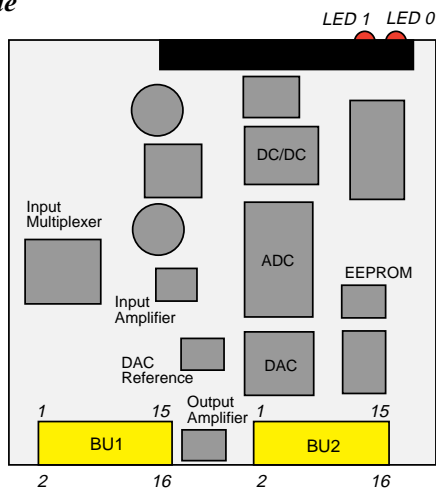
### 5.1.3 Front Panel Layout



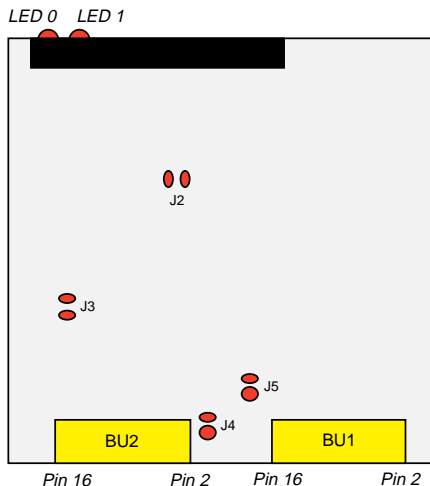


### 5.1.4 Board Overview

#### Component Side



#### Solder Side





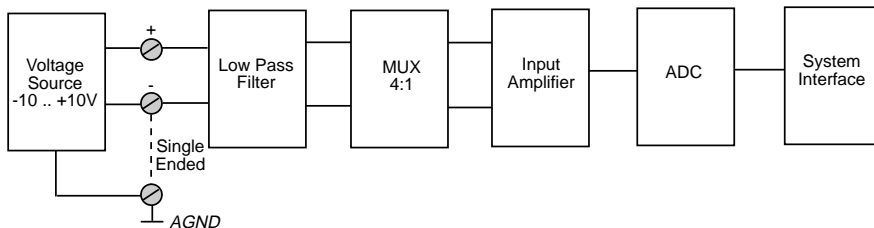
### 5.1.5 Functional Description

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  $\pm 15\text{V}$  for the analog I/O and an additional linear regulator provides the  $-5\text{V}$  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 ( $-2.048\text{V} .. +2.048\text{V}$ ).

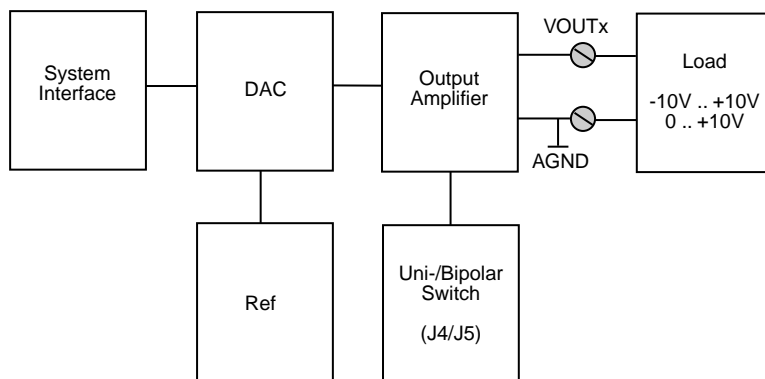
The incoming voltage signals are multiplexed and reduced via a 1:5 divider network so that the ADC is supplied with a scalable input voltage range.



### 5.1.5.2 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.



### 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.

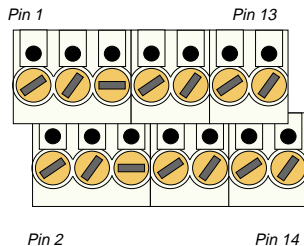




### 5.1.7 Pinouts

#### Screw Terminal Pinouts

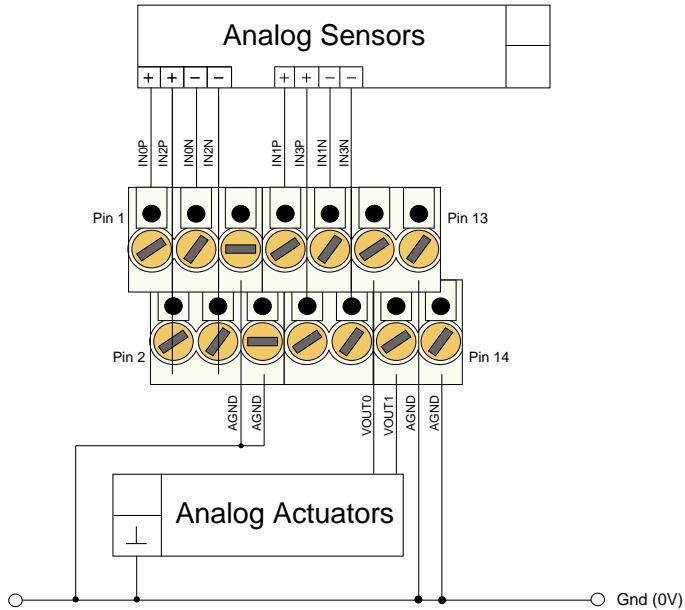
The following shows the pinout/signal relationship for the SM-DAD1 when connected to a particular screw terminal block.



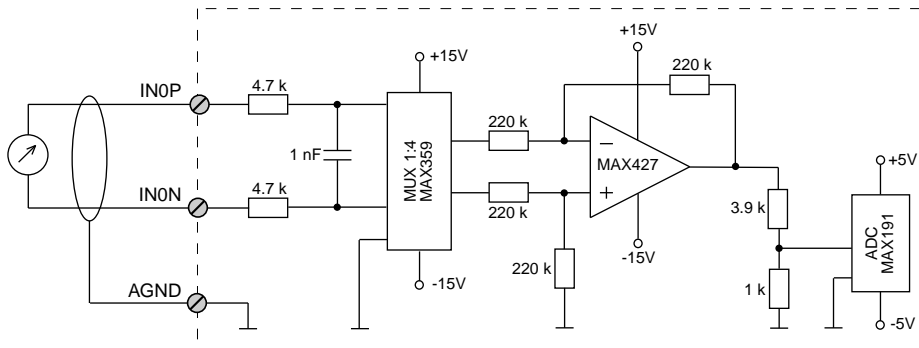
Pin Nr.	Signal	Description	Pin Nr.	Signal	Description
1	IN0P (+)	Analog Input Ch. 0+	2	IN2P (+)	Analog Input Ch. 2+
3	IN0N (-)	Analog Input Ch. 0-	4	IN2N (-)	Analog Input Ch. 2-
5	AGND	Analog GND	6	AGND	Analog GND
7	IN1P (+)	Analog Input Ch. 1+	8	IN3P (+)	Analog Input Ch. 3+
9	IN1N (-)	Analog Input Ch. 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



**Connection**

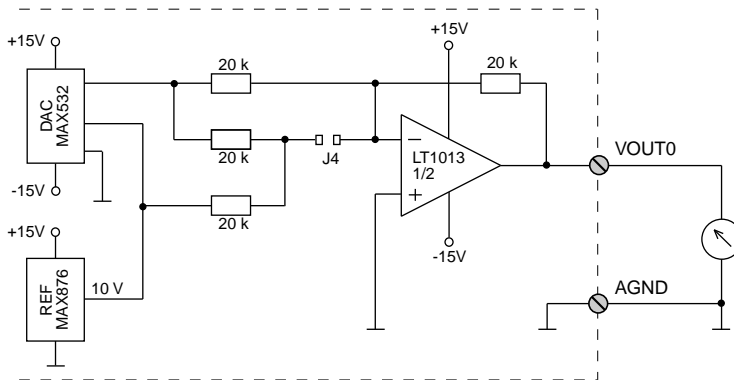


**A/D Circuit**





### 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.



## 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 I/O Base Module or Base Module and Extension unit;
- 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*).



**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**

<code>u_int8 PortNr</code>	SM-Port number to initialize (from 0 to 10)
----------------------------	--

**Output**

<code>error_code</code>	SUCCESS or
	<code>E_BMODE</code> if wrong or no module
	<code>E_MEMFUL</code> 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

<code>u_int8 PortNr</code>	SM-Port number to de-initialize (from 0 to 10)
----------------------------	---

#### Output

<code>error_code</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);
```



### 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





### 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





### 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



### 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);
```



### 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);
```



### 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 = 2	
1	in1
2	in2
3	in3
4	in4

logical_address = 2	
UNIPOLAR_bipolar = false	
1	out1
2	out2

#### 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.



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** : 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.







## 5.2 SM-PT100

### 5.2.1 Introduction

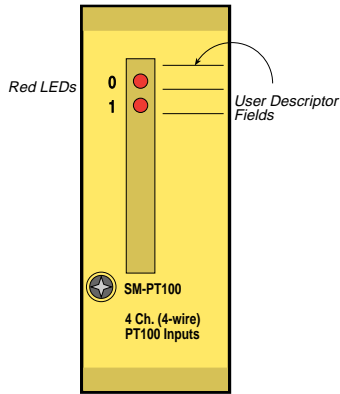
The SM-PT100 is used for 2, 3 or 4-wire temperature measurement using PT100 sensors driven by an on-board constant current source. Temperature measurements between -200°C and + 850°C are possible with corrective calibration for the two possible PT100 4-wire classes being provided in driver software for 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	

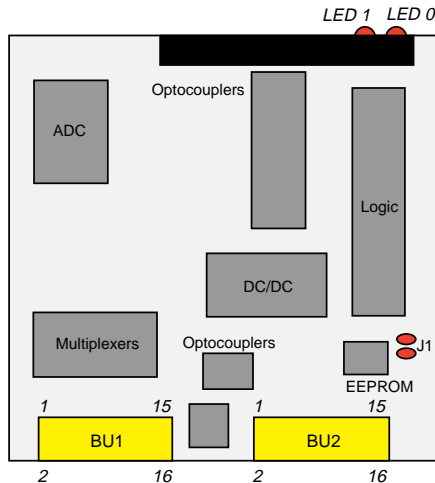


### 5.2.3 Front Panel Layout



### 5.2.4 Board Overview

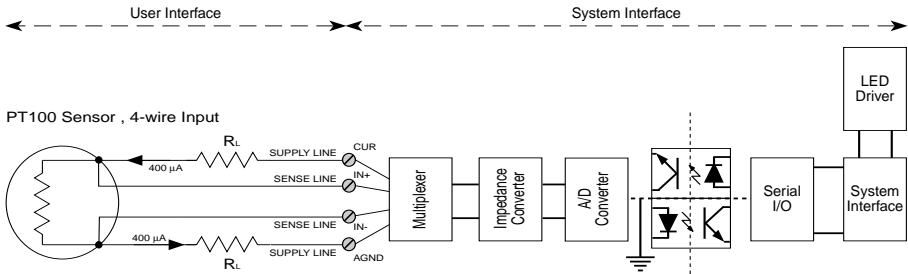
#### *Component Side*





### 5.2.5 Functional Description

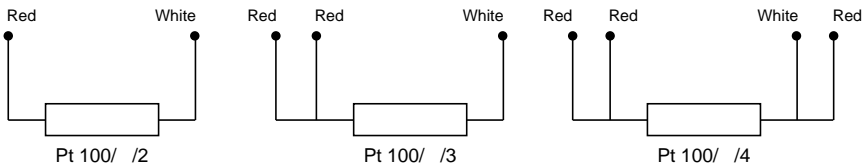
**Figure 5.2.5.1: SM-PT100 Schematic Diagram**



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 0°C 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 (10M $\Omega$ ), 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 °C	Tolerance			
	Class A		Class B	
	°C	$\Omega$	°C	$\Omega$
-200	$\pm 0.55$	$\pm 0.24$	$\pm 1.3$	$\pm 0.56$
-100	$\pm 0.35$	$\pm 0.14$	$\pm 0.8$	$\pm 0.32$
0	$\pm 0.15$	$\pm 0.06$	$\pm 0.3$	$\pm 0.12$
100	$\pm 0.35$	$\pm 0.13$	$\pm 0.8$	$\pm 0.30$
200	$\pm 0.55$	$\pm 0.20$	$\pm 1.3$	$\pm 0.48$
300	$\pm 0.75$	$\pm 0.27$	$\pm 1.8$	$\pm 0.64$
400	$\pm 0.95$	$\pm 0.33$	$\pm 2.3$	$\pm 0.79$
500	$\pm 1.15$	$\pm 0.38$	$\pm 2.8$	$\pm 0.93$
600	$\pm 1.35$	$\pm 0.43$	$\pm 3.3$	$\pm 1.06$
650	$\pm 1.45$	$\pm 0.46$	$\pm 3.6$	$\pm 1.13$
700	–	–	$\pm 3.8$	$\pm 1.17$
800	–	–	$\pm 4.3$	$\pm 1.28$
850	–	–	$\pm 4.6$	$\pm 1.34$

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.



### 5.2.6 Configuration

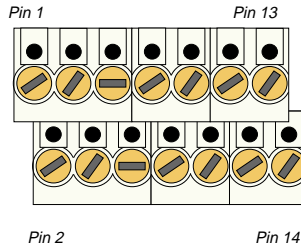
#### Jumper J1 - EEPROM Protection

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.

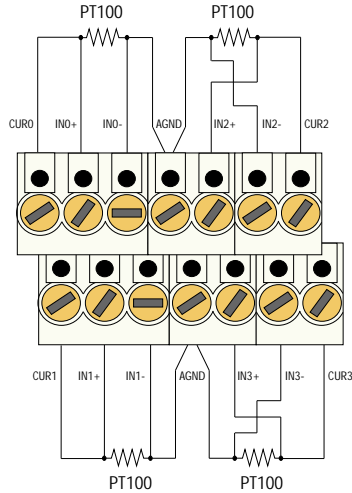


Pin Nr.	Signal	Description	Pin Nr.	Signal	Description
1	Cur 0	400µA source for sensor	2	Cur 1	400µA source for sensor
3	In 0+	Sense line 0 positive	4	In 1+	Sense line 1 positive
5	In 0 -	Sense line 0 negative	6	In 1 -	Sense line 1 negative
7	AGND	GND between 2 sensors	8	AGND	GND between 2 sensors
9	In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

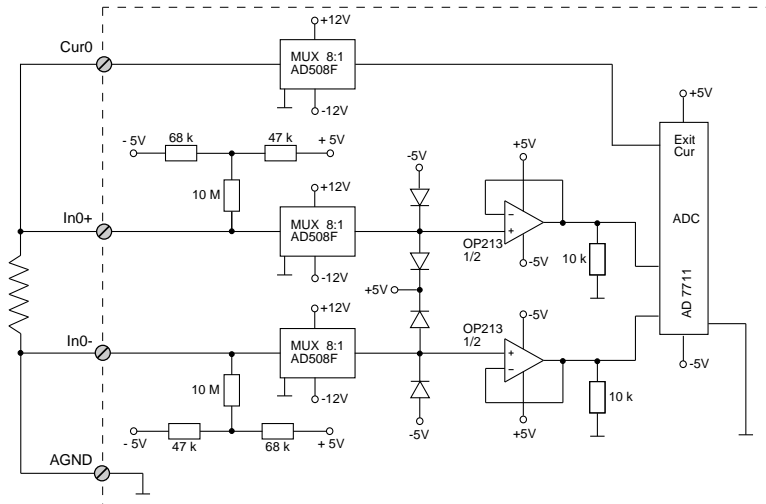




**Connection**



**Input Circuit**





## 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

- SMART I/O Base Module or Base Module and Extension unit;
- 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 as 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/CMD5*).



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/CMDSD/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*.





## 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

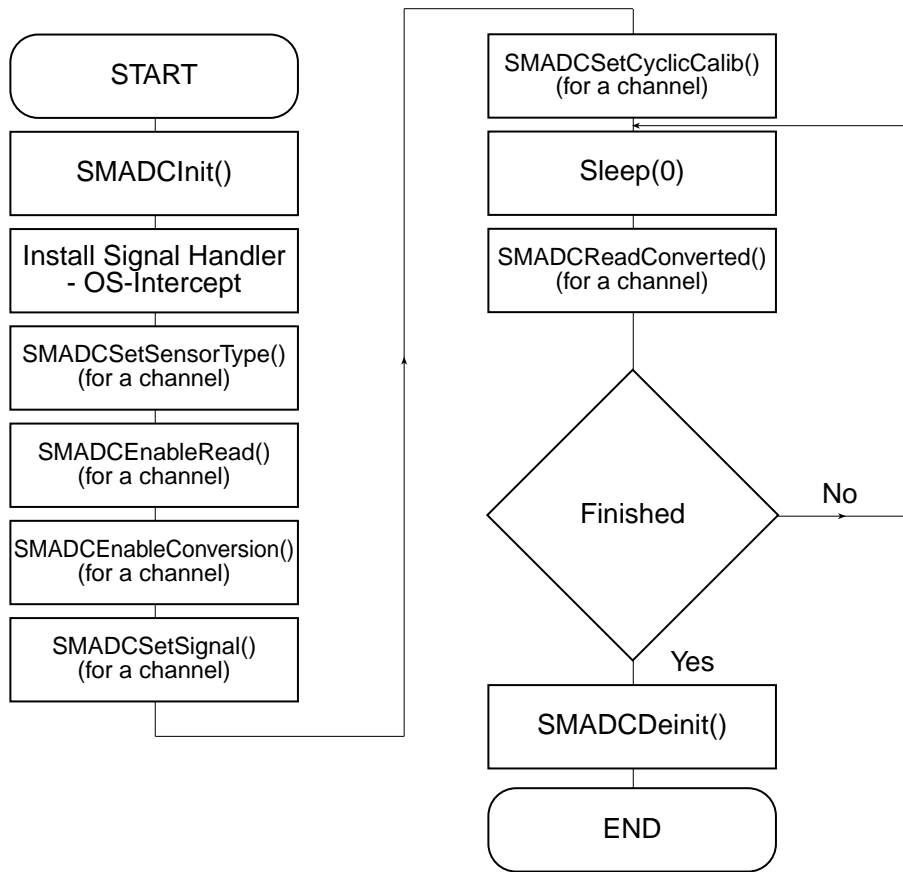
Both LEDs on the SM-PT100 module are used to show the status of the HW task. LED1 is the upper one; LED2 is the lower one.

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**





### 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

<code>u_int8 PortNr</code>	Port number of module to initialize (from 0 to 10)
----------------------------	--

#### Output

<code>error_code</code>	<code>SUCCESS</code>	
	<code>E_MNF</code>	If wrong type of module or no module is fitted on the selected port
	<code>E_PRCABT</code>	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

<code>u_int8 PortNr</code>	Port number of module to calibrate (from 0 to 10)
----------------------------	--

#### Output

<code>error_code</code>	SUCCESS
	<code>E_MNF</code> 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);
```



### 5.2.8.4 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);
```



### 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 on 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 <i>linear.h</i> )

#### 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);
```



### 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);
```





### 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

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

#### Output

<code>error_code</code>	SUCCESS
	<code>E_MNF</code> If wrong type of module or no module is fitted on the selected port
	<code>E_PARAM</code> 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);
```



### 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);
```



### 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 $\oplus 0x100$ )
	or standard OS-9 error code (refer to the OS-9 Technical Manual Error Codes Section).

#### Example

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



### 5.2.8.12 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

<code>u_int8 PortNr</code>	Port number to set gain (from 0 to 10)
<code>u_int8 channel</code>	Number of specified channel (0 to 3)
<code>u_int8 gain</code>	Value of gain amplifier (0 to 7 for gain factors from 1 to 128)

#### Output

<code>error_code</code>	SUCCESS
	<code>E_MNF</code> If wrong type of module or no module is fitted on the selected port
	<code>E_PARAM</code> If wrong channel number is specified
	<code>E_BMODE</code> 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);
```



#### 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

<code>u_int8 PortNr</code>	Port number to enable the reading function (from 0 to 10)
<code>u_int8 channel</code>	Number of specified channel (0 to 3)
<code>BOOLEAN enable</code>	Flag for enabling or disabling reading of the specified channel

##### Output

<code>error_code</code>	SUCCESS
	<code>E_MNF</code> If wrong type of module or no module is fitted on the selected port
	<code>E_PARAM</code> 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);
```





### 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

<code>u_int8 PortNr</code>	Port number from which raw data is to be read (from 0 to 10)
<code>u_int8 channel</code>	Number of the channel to be read (0 to 3)
<code>u_int16 *pBuffer</code>	Pointer to a user defined buffer

#### Output

<code>error_code</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);
```



### 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**

<input type="checkbox"/>	logical_address = 3
<input type="checkbox"/>	calib_period = 20
<input type="checkbox"/>	type_ch1 = resistor100
<input type="checkbox"/>	type_ch2 = resistor100
<input type="checkbox"/>	type_ch3 = resistor100
<input type="checkbox"/>	type_ch4 = resistor100
<input type="checkbox"/> 1	<input checked="" type="checkbox"/> ch1
<input type="checkbox"/> 2	<input checked="" type="checkbox"/> ch2
<input type="checkbox"/> 3	<input checked="" type="checkbox"/> ch3
<input type="checkbox"/> 4	<input checked="" type="checkbox"/> ch4

### 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.



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** : 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.



## 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

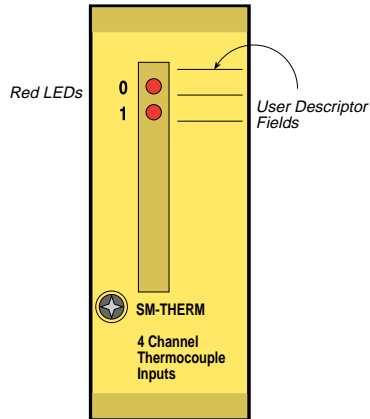
Isolation	1.0 kV Optoisolated from the system	
Input	4 Thermocouple groups	
	1 PT100 for Cold Junction (2-wire)	
	Open Sensor Detection	
	Self Calibration	
	Overvoltage Protection	: ± 25V (power connected) : ± 35V (power disconnected)
	Temperature Range	: Refer to table below
	Temperature Drift	: 1 LSB/°C (not compensated)
	Conversion Time	: typ. 125 ms, max. 500 ms (depends on filter settings)
	Input Impedance	: 10 M.Ω (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			



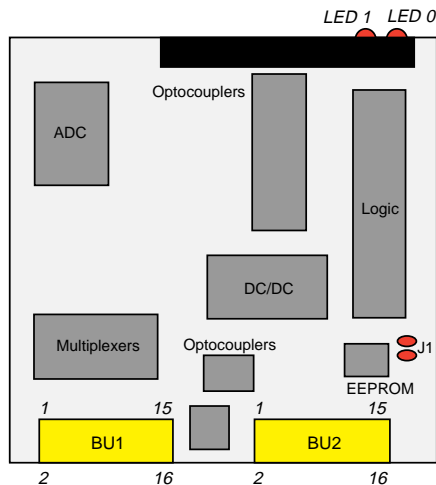


### 5.3.3 Front Panel Layout



### 5.3.4 Board Overview

#### *Component Side*

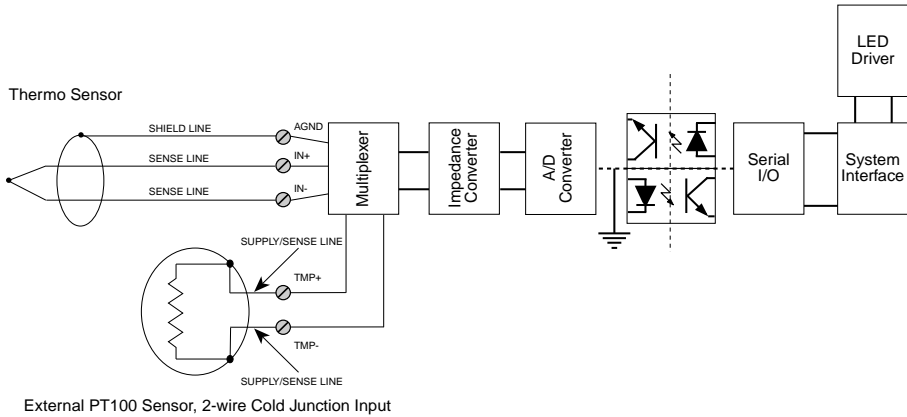






### 5.3.5 Functional Description

Figure 5.3.5.1: SM-THERM Schematic Diagram



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

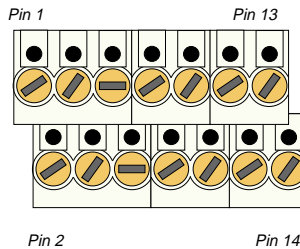
#### Jumper J1 - EEPROM Protection

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.

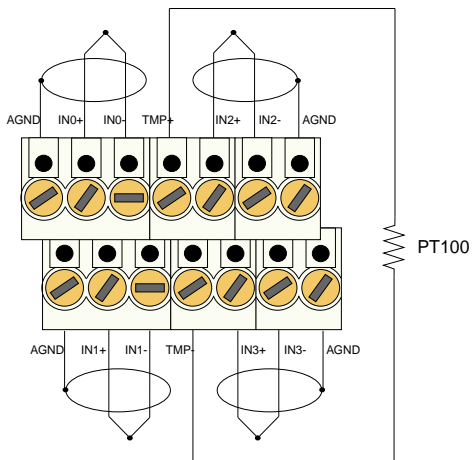


Pin Nr.	Signal	Description	Pin Nr.	Signal	Description
1	AGND	Analog Shield	2	AGND	Analog Shield
3	In 0+	Positive Sense Line (0)	4	In 1+	Positive Sense Line (1)
5	In 0 -	Negative Sense Line (0)	6	In 1 -	Negative Sense Line (1)
7	TMP + †	PT100 Cold Jn. supply	8	TMP - †	PT100 Cold Jn. supply
9	In 2+	Positive 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

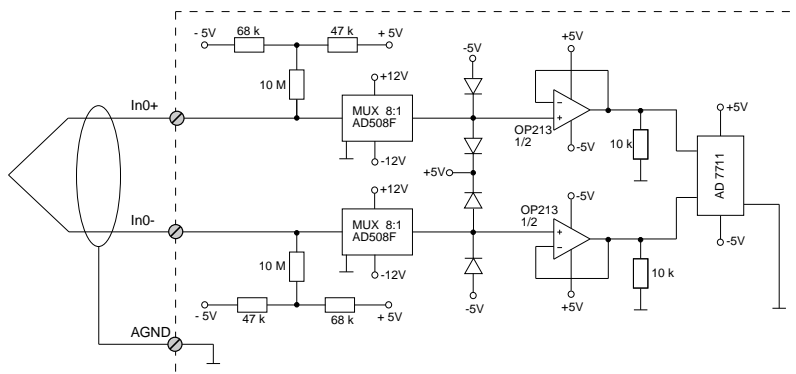
† Cold Junction point PT100 connection



### 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

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

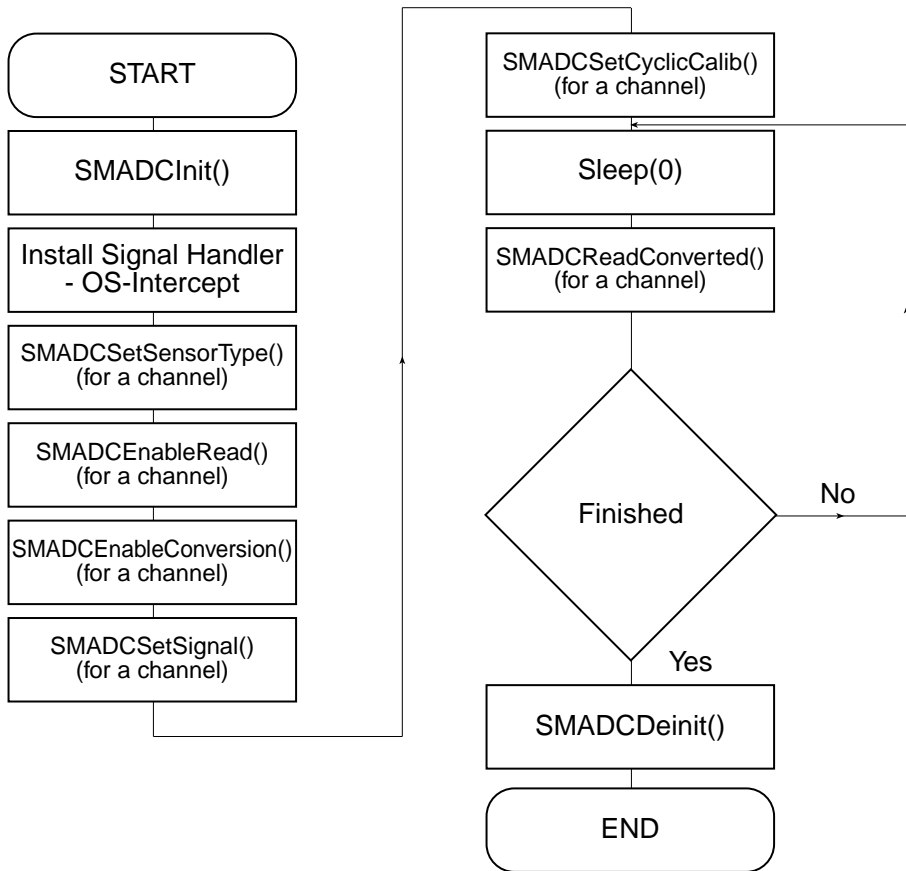
Both LEDs on the SM-THERM module are used to show the status of the HW task. LED1 is the upper one; LED2 is the lower one.

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



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

<code>u_int8 PortNr</code>	Port number of module to initialize (from 0 to 10)
----------------------------	--

#### Output

<code>error_code</code>	SUCCESS
	<code>E_MNF</code> If wrong type of module or no module is fitted on the selected port
	<code>E_PRCABT</code> 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);
```





### 5.3.8.3 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

<code>u_int8 PortNr</code>	Port number of module to calibrate (from 0 to 10)
----------------------------	---

#### Output

<code>error_code</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);
```



### 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

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

#### Output

<code>error_code</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);
```



### 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

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

#### Output

<code>error_code</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);
```



### 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

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

#### Output

<code>error_code</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);
```





### 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

<code>u_int8 PortNr</code>	Port number of SM-THERM to set the signal (from 0 to 10)
<code>u_int8 channel</code>	Number of specified channel (0 to 3)
<code>process_id procID</code>	ID of the process to send the signal to
<code>signal_code sigCode</code>	Signal code to be sent

#### Output

<code>error_code</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 $\oplus 0x100$ )
	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}
```



### 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 (<math>\neq 0</math>), 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);
```



### 5.3.8.15 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);
```



### 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);
```





### 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 =	3
calib_period =	120
type_ch1 =	V
type_ch2 =	V
type_ch3 =	V
type_ch4 =	V
1	ch1
2	ch2
3	ch3
4	ch4

#### 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.



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** : 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.



This page has been left blank intentionally.





## 5.4 SM-ADC1

### 5.4.1 Introduction

The SM-ADC1 provides a fast, 12-bit, bipolar ( $\pm 10V$  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.

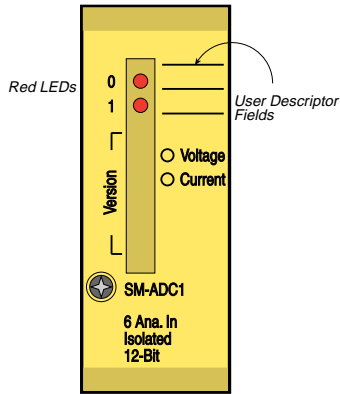
### 5.4.2 Specifications

Isolation	1.0 kV Optoisolated from the system
Input	6 Differential Inputs Input Range $\pm 10V$ (0 .. 20mA) 12-bit ADC with 1-bit non-linearity 10-bit Repeating Accuracy Overvoltage Protection $\pm 35V$ Input Impedance approx. 440k $\Omega$ (124 $\Omega$ ) 10kHz Filter On-Chip Ref. Voltage 13 $\mu$ s 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

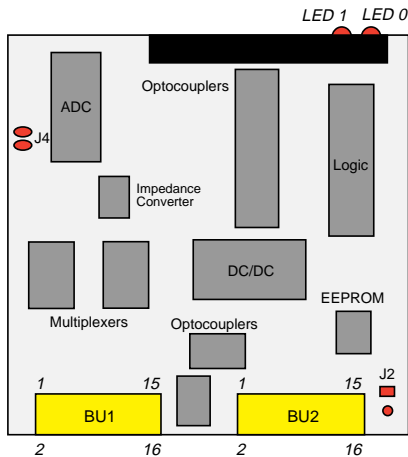


### 5.4.3 Front Panel Layout



### 5.4.4 Board Overview

#### *Component Side*





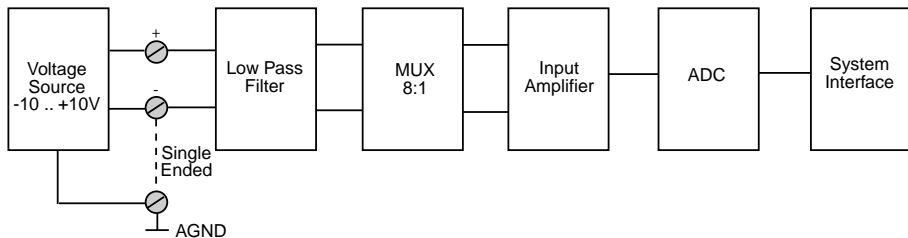
### 5.4.5 Functional Description

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  $\pm 15V$  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 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.



### 5.4.6 Configuration

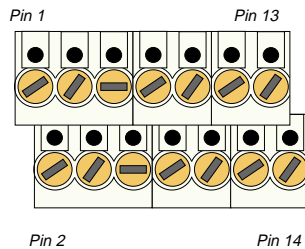
Although the SM-ADC1 has one solder jumper (J2) for EEPROM configuration, the user should not interfere with it.

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.

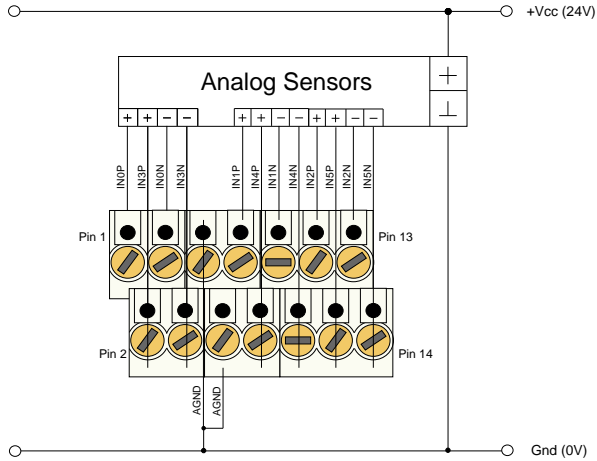


Pin Nr.	Signal	Description	Pin Nr.	Signal	Description
1	IN0+	Analog Input Ch. 0+	2	IN3+	Analog Input Ch. 3+
3	IN0-	Analog Input Ch. 0-	4	IN3-	Analog Input Ch. 3-
5	AGND	Analog GND	6	AGND	Analog GND
7	IN1+	Analog Input Ch. 1+	8	IN4+	Analog Input Ch. 4+
9	IN1-	Analog Input Ch. 1-	10	IN4-	Analog Input Ch. 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-

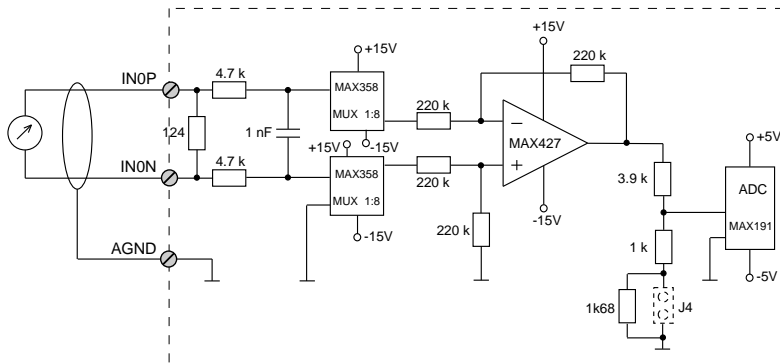




**Connection:**



**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

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

<code>u_int8 PortNr</code>	Port number of module to initialise (from 0 to 10)
----------------------------	--

#### Output

<code>error_code</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



#### 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



### 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);
```



### 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

<code>u_int8 PortNr</code>	SM-Port number to de-initialise (from 0 to 10)
----------------------------	---

#### Output

<code>error_code</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);
```





### 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 = 3	
1	in1
2	in2
3	in3
4	in4
5	in5
6	in6

#### 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).





### 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 SMART I/O 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.

**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.

**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:

$$\text{var} = \frac{0\text{xFFF0}}{0\text{x14}} \text{ mA}$$

**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.





## 5.5 SM-DAC1

### 5.5.1 Introduction

The SM-DAC1 provides a fast, 12-bit, bipolar ( $\pm 10V$  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.

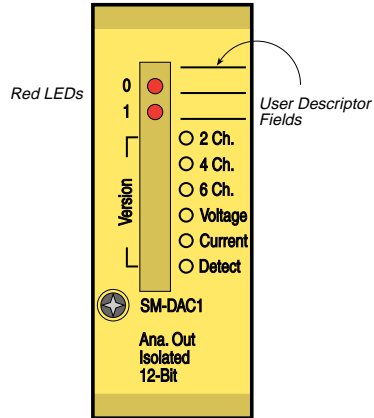
### 5.5.2 Specifications

Isolation	1.0 kV Optoisolated from the system
Output	2/6 Outputs Output Range $\pm 10V$ 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) 75 $\mu$ s 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)

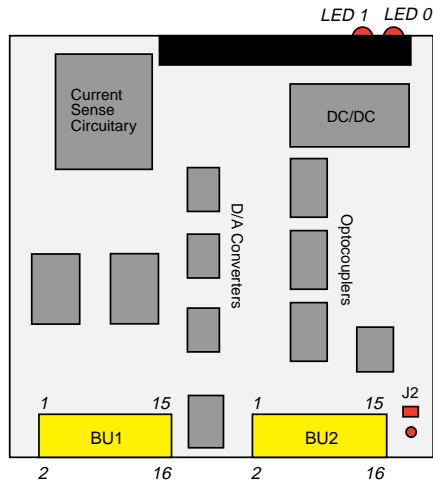


### 5.5.3 Front Panel Layout



### 5.5.4 Board Overview

#### *Component Side*







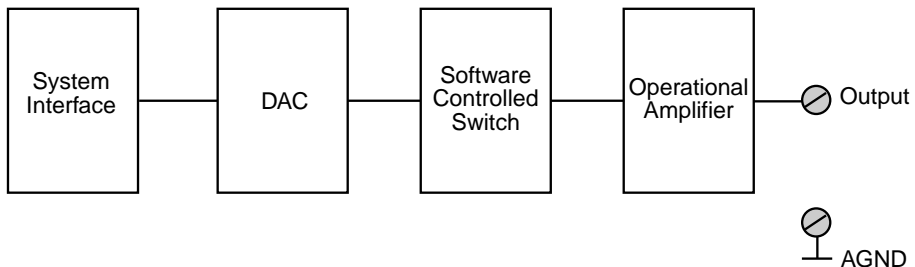
### 5.5.5 Functional Description

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  $\pm 15V$  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 Schematic 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.



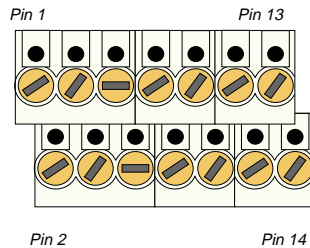
### 5.5.6 Configuration

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

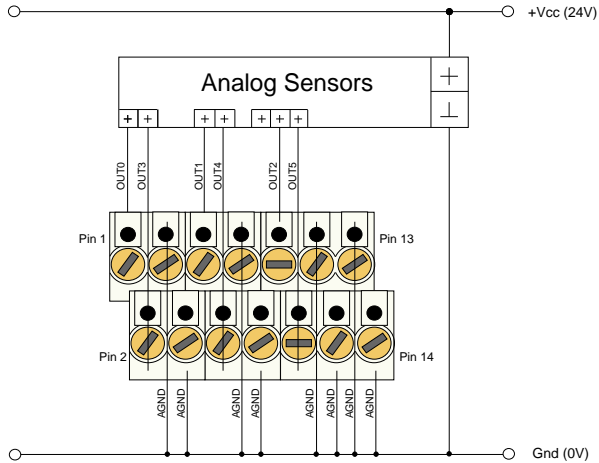
The following shows the pinout/signal relationship for the SM-DAC1 when connected to a particular screw terminal block.



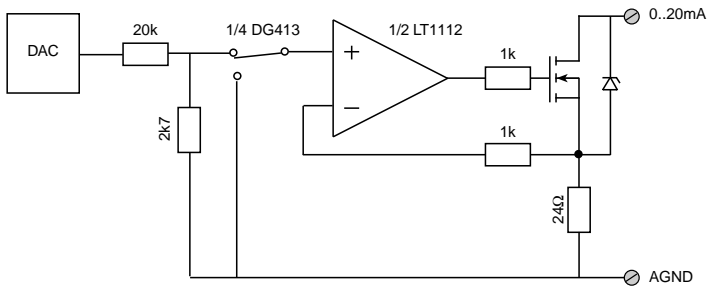
Pin Nr.	Signal	Description	Pin Nr.	Signal	Description
1	OUT0	Analog Output Ch. 0	2	OUT3	Analog Output Ch. 3
3	AGND	Analog GND	4	AGND	Analog GND
5	OUT1	Analog Output Ch. 1	6	OUT4	Analog Output Ch. 4
7	AGND	Analog GND	8	AGND	Analog GND
9	OUT2	Analog Output Ch. 2	10	OUT5	Analog Output Ch. 5
11	AGND	Analog GND	12	AGND	Analog GND
13	AGND	Analog GND	14	AGND	Analog GND



**Connection:**

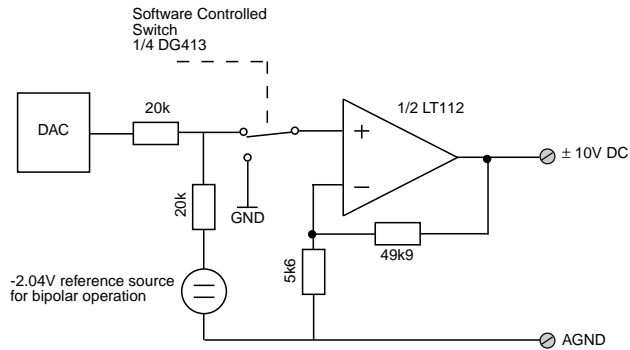


**Current Output Circuit:**





**Voltage Output Circuit:**





## 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

- SMART I/O Base Module or Base Module and Extension unit;
- 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 as 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*).



### 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

<code>u_int8 PortNr</code>	Port number of module to initialise (from 0 to 10)
----------------------------	---

#### Output

<code>error_code</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);
```



### 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.



#### 5.5.8.4 SMADAC1Operate

##### Syntax

```
error_code SMADAC1Operate(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 = SMADAC1Operate(0);
```







### 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

<code>error_code</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);
```



### 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



### 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);
```





### 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);
```



### 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

<code>u_int8 PortNr</code>	SM-Port number to de-initialise (from 0 to 10)
----------------------------	---

#### Output

<code>error_code</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);
```



### 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 = 3	
1	out1
2	⊘
3	⊘
4	⊘
5	⊘
6	⊘

#### 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! 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).





### 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.





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** : 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.





*Communications Modules*

*for*

*SMART I/O*



# Table of Contents

<b>6. Communications Modules .....</b>	<b>6-5</b>
6.1 SM-RS232 .....	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 Configuration .....	6-8
6.1.7 Pinouts .....	6-8
6.1.8 ISaGRAF Programming .....	6-10
6.1.8.1 The ISaGRAF Board Parameters .....	6-10
6.1.8.2 The ISaGRAF Operate Calls .....	6-12
6.1.9 OS-9 Programming .....	6-14
6.2 SM-SSI .....	6-17
6.2.1 Introduction .....	6-17
6.2.2 Specifications .....	6-17
6.2.3 Front Panel Layout .....	6-18
6.2.4 Board Overview .....	6-18
6.2.5 Functional Description .....	6-19
6.2.6 SSI Operation .....	6-20
6.2.7 Register Description .....	6-22
6.2.7.1 Control Register .....	6-22
6.2.7.2 CTRL1 Register .....	6-23
6.2.7.3 CTRL2 Register .....	6-23
6.2.7.4 CTRL3 Register .....	6-23
6.2.7.5 Compare Register .....	6-24
6.2.7.6 Identification Register .....	6-24
6.2.7.7 Status Register .....	6-24
6.2.7.8 STAT1 Register .....	6-24
6.2.7.9 STAT2 Register .....	6-25
6.2.7.10 Data Register .....	6-25



6.2.8 MATCH Function .....	6-26
6.2.9 Tested Sensors .....	6-26
6.2.10 Configuration .....	6-27
6.2.11 Pinouts .....	6-27
6.2.12 'C' Programming .....	6-29
6.2.12.1 SM-SSI Library .....	6-29
6.2.12.2 SMSSIInit .....	6-30
6.2.12.3 SMSSIDeInit .....	6-31
6.2.12.4 SMSSISetSetPoint .....	6-32
6.2.12.5 SMSSISetCtrlReg .....	6-33
6.2.12.6 SMSSIGetStatus .....	6-34
6.2.12.7 SMSSIGetData .....	6-35
6.2.13 ISaGRAF Programming .....	6-36
6.2.13.1 The ISaGRAF Board Parameters .....	6-36



This page has been left blank intentionally.



## 6. Communications Modules



### 6.1 SM-RS232

#### 6.1.1 Introduction

The SM-RS232 provides serial communication observing a true RS232 interface definition. Interface parameters such as baud rate, stop bits, data bits and parity are defined by software. The hardware itself supports baud-rates up to 120 kbaud but the delivered software only allows rates up to 19200 baud. Communication tasks are handled directly by the SCC3 port of the 68302 chip on the SMART-BASE hence this module can only be used in the first slot of the SMART-BASE as this is the only slot where the CPU port lines can be used serially. Two LEDs showing data transmission provide the user interface.

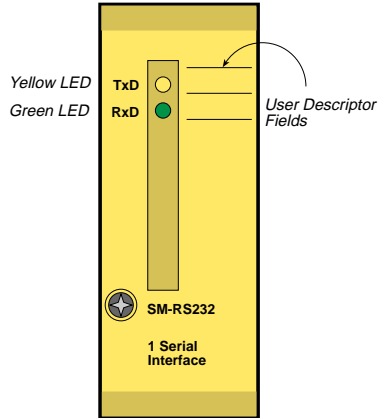
#### 6.1.2 Specifications

Input	Transceiver RxD and CTS † Up to 19200 Baud
Output	Transceiver TxD and DTR † Up to 19200 Baud
EEPROM	256 Byte Serial EEPROM
Front Panel	Green RxD LED Yellow TxD LED
Power Consumption	23mW (min.), 75mW (max.)
Module Weight	30g
ID Byte	\$80 Read by the SPI Interface

† CTS/DTR lines are NOT supported by PEP standard software. XON/XOFF software protocol is used instead.

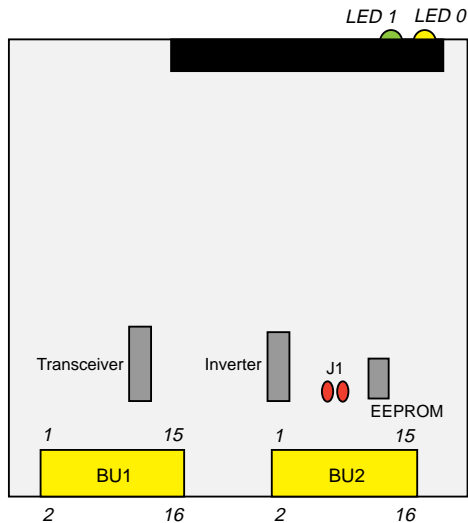


### 6.1.3 Front Panel Layout



### 6.1.4 Board Overview

#### Component Side

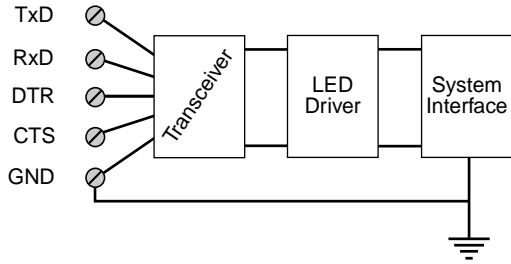






### 6.1.5 Functional Description

**Figure 4.3.5.1: SM-RS232 Schematic Diagram**



The serial interface is realized using the Tx<sub>D3</sub> and Rx<sub>D3</sub> lines of the 68302 SCC3 port on the SMART-BASE and for this reason this module can only be used in the first slot on the SMART-BASE. Additionally, a software programmed DTR function can be created with the CTS line being connected to the interrupt line that is routed to the I/O controller of the SMART-BASE. Although it is beyond the scope of this manual to discuss this connection, PEP Modular Computers can provide advice on its implementation if required.

The two LEDs on the front-panel indicate the status of the Rx<sub>D</sub> and Tx<sub>D</sub> lines (ON means that the line is low).

An on-board serial EEPROM currently contains the ID-Byte for module recognition and should not be altered. In the future, other module specific data may be stored to provide more flexibility.





### 6.1.6 Configuration

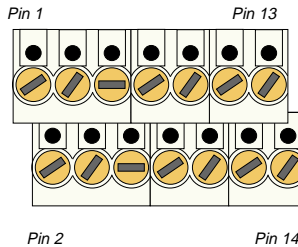
#### Jumper J1 - EEPROM Protection

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

### 6.1.7 Pinouts

#### Screw Terminal Pinouts

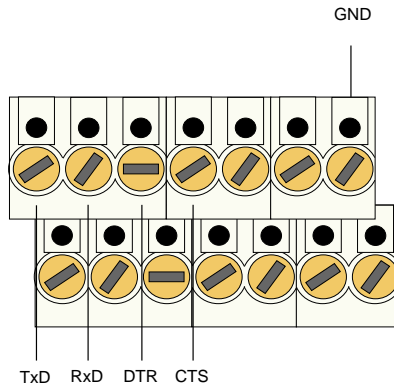
The following shows the pinout/signal relationship for the SM-RS232 when connected to a particular screw terminal block.



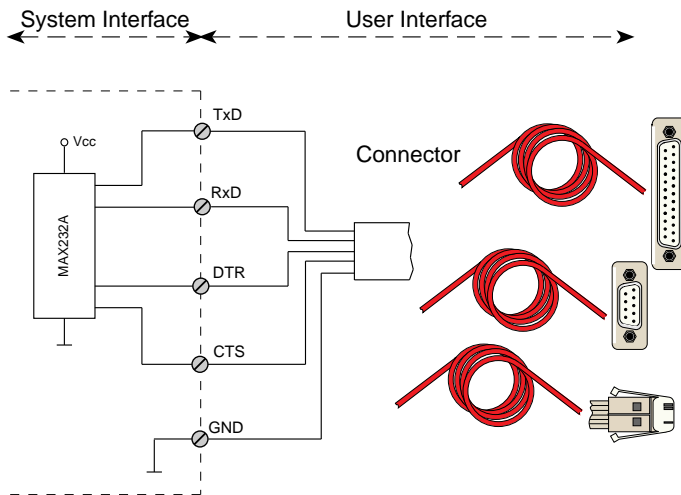
Pin Nr.	Signal	Description	Pin Nr.	Signal	Description
1	TxD	Transmit Signal	2	N/C	No Connection
3	RxD	Receive Signal	4	N/C	No Connection
5	DTR	Data Terminal Ready	6	N/C	No Connection
7	CTS	Clear to Send	8	N/C	No Connection
9	N/C	No Connection	10	N/C	No Connection
11	N/C	No Connection	12	N/C	No Connection
13	GND	Ground	14	GND	Ground



### Connection



### Interface





## 6.1.8 ISaGRAF Programming

### 6.1.8.1 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 6.1.8.1.1 Typical Screen Sections for the SM-RS232 Module**

dev_file_name = scc3
dev_mod_name = scc3
hw_handshake = false
echo = true
char_length = 8
stop_bits = 0
parity = 0
baud_rate = 9600
xon_char = 11
xoff_char = 13
eor_char = D
<input checked="" type="checkbox"/> keyboard

dev_file_name = scc3
dev_mod_name = scc3
hw_handshake = false
echo = true
char_length = 8
stop_bits = 0
parity = 0
baud_rate = 9600
xon_char = 11
xoff_char = 13
eor_char = D
<input checked="" type="checkbox"/> display

## Logical Address

This particular module is accessed over an OS-9 device descriptor and driver which are already present in EPROM on the SMART-BASE. Because the SM-RS232 module can only occupy the first slot on the SMART-BASE, the descriptor is able to access the module directly without the need of a manually entered logical address.

## Module Specific Information

dev\_file\_name : This must be set to scc3  
 dev\_mod\_name : This must be set to scc3  
 hw\_handshake : A hardware handshaking flag that is not supported for SMART I/O applications  
 echo : A boolean flag allowing incoming characters to be echoed to the screen.



<code>char_length :</code>	This represents the number of data-bits within a transmitted character and may be 5, 6, 7 or 8
<code>stop_bits :</code>	The number of stop-bits supported by the communication protocol may be : 0 for 1 stop-bit 1 for 1.5 stop-bits or 2 for 2 stop-bits
<code>parity :</code>	The parity bit of a data string may be : 0 for no parity 1 for odd parity or 2 for even parity
<code>baud_rate :</code>	The board rates supported by the SM-RS232 are standard values up to 19200 baud
<code>xon_char :</code>	XON character; typically set to 11 hex
<code>xoff_char :</code>	XOFF character: typically set to 13 hex
<code>eor_char :</code>	Character to be used to detect the end of record; is usually set to 0D hex.

The input and output devices are represented by keyboard and display respectively.



### 6.1.8.2 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 SMART I/O environment however, the calls applicable to this module are :

**O\_INIT\_CODE** : The syntax and usage have already been explained.

**O\_SERIAL\_READ** : This operate call prevents the system from being blocked due to polling of incoming characters. Hence, during every ISaGRAF cycle, the program checks if the buffer for incoming data contains data. If so, the data will be collected. The syntax is as follows:

```
<chars_in> := OPERATE(<ser_in>,O_SERIAL_READ,<chars>);
```

where   <chars\_in>    represents the actual number of characters read from the buffer.  
           <ser\_in>       is the input device name.  
           <chars>       the number of characters expected to be read.

### Example

```
ret := OPERATE(keyboard, O_SERIAL_READ, 10);
```

If the default value 0D hex is used for the eor\_char then the scanning of the input will terminate when either the <RETURN> key is pressed or 10 characters have been read.



### 6.1.9 OS-9 Programming

Due to the fact that the device descriptor for the serial I/O is burnt into EPROM, it cannot be over-written with new values. Therefore a copy has to be generated in RAM where the parameters may be manipulated. This procedure is only useful for those users who wish to alter the default parameter settings. The procedure is therefore as follows :

From the \$ prompt that is displayed under OS-9, copy the descriptor to the ramdisk, thus

```
$ save scc3          <return>
```

change the revision number from 0 to 1 so that OS-9 selects the newest:-

```
$ fixmod -ua=8001 scc3  <return>
```

Module:scc3 - Fixing header parity - Fixing module CRC

now load the new revision into memory, i.e. a copy from the ramdisk:-

```
$ load -ld scc3       <return>
```

```
load:loaded /scc3
```

delete the copy held on the ramdisk to free the space:-

```
$ del scc3           <return>
```

check the port settings:-

```
$ xmode/scc3        <return>
```

```
/scc3
```

```
noupc bsp bsl echo if null=0 pause pag=24 bsp=08  
del=18 eor=0D eof=1B reprint=04 dup=01 psc=17 abort=03  
quit=05 bse=08 bell=07 type=00 baud=9600 xon=11  
xoff=13 tabc=09 tabs=4
```





modify the settings, for example:-

```
$ xmode baud=19200 cs=7 par=none /scc3 <return>
```

check that the system has taken the parameters:-

```
$ xmode/scc3 <return>  
/scc3  
noupc bsp bsl echo if null=0 pause pag=24 bsp=08  
del=18 eor=0D eof=1B reprint=04 dup=01 psc=17 abort=03  
quit=05 bse=08 bell=07 type=00 baud=19200 xon=11  
xoff=13 tabc=09 tabs=4
```

refer to the OS-9 help page for parameter descriptions by using the following:-

```
$ xmode -? <return>
```

and finally initialise the device:-

```
$ iniz scc3 <return>
```



This page has been left blank intentionally.





## 6.2 SM-SSI

### 6.2.1 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.

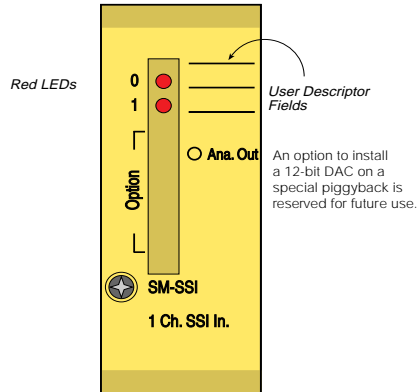
### 6.2.2 Specifications

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 ( $\pm 5\%$ )
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

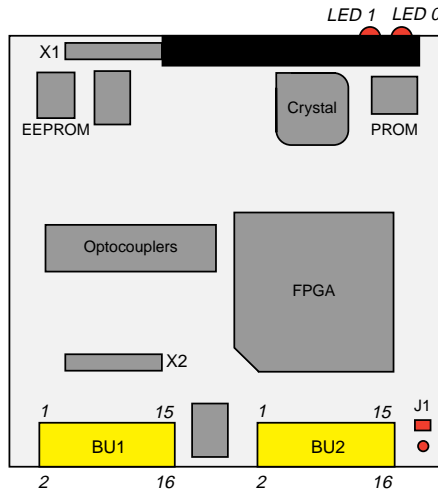


### 6.2.3 Front Panel Layout



### 6.2.4 Board Overview

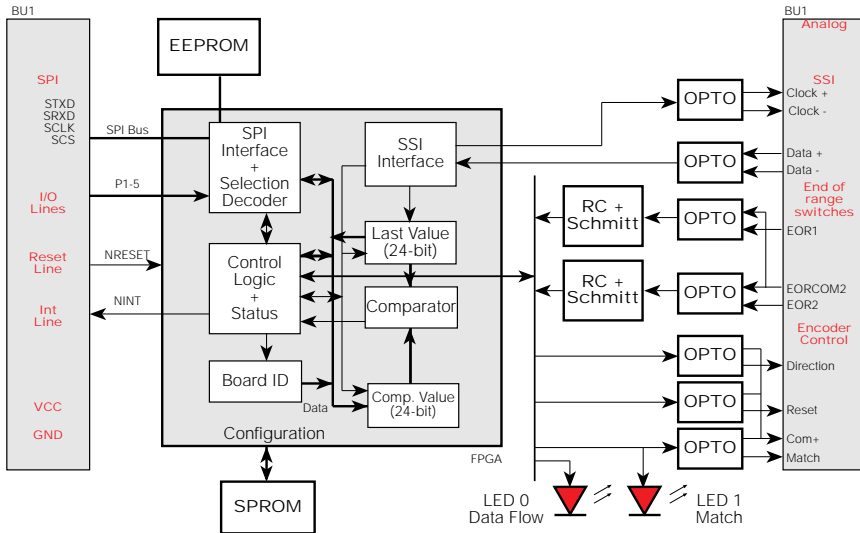
#### Component Side





### 6.2.5 Functional Description

**Figure 6.2.5.1: 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.



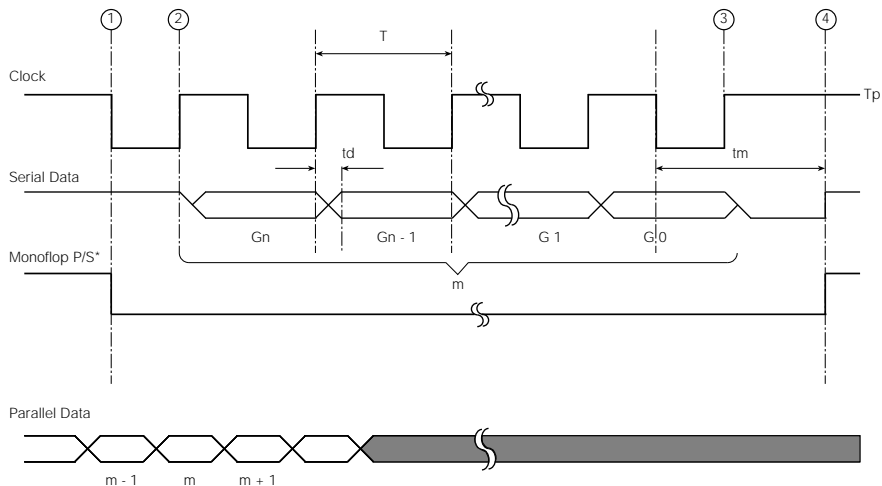


### 6.2.6 SSI Operation

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 6.2.6.1: SM-SSI Timing Diagram**



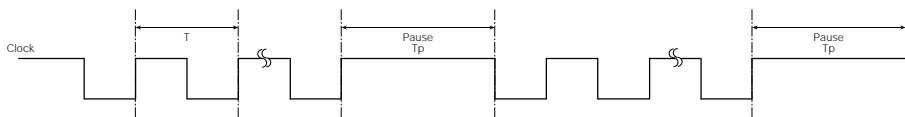
- |  |                           |
|--|---------------------------|
| $m$ = Stored parallel data             | $T$ = SSI clock period    |
| $t_d$ = Delay time for the first clock | $G_0$ = LSB of Gray* code |
| $G_n$ = MSB of Gray* code              | $t_m$ = Monoflop time     |
| $T_p$ = Pause time in SSI clock        |                           |

\* Binary code may also be converted.



- 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 6.2.6.2: SM-SSI Timing Diagram for serial transfer**





### 6.2.7 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.

#### 6.2.7.1 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





### 6.2.7.2 CTRL1 Register

CTRL1-7	CTRL1-6	CTRL1-5	CTRL1-4	CTRL1-3	CTRL1-2	CTRL1-1
RST	AR	SOT	DIR	EN13-MH	EN12-EOR	EN11-TC

ENIO-P	enable / disable (1 / 0) interrupt on parity error
EN11-TC	enable / disable (1 / 0) interrupt on transmission complete
EN12-EOR	enable / disable (1 / 0) interrupt on end-of-range switch 1 / 2
EN13-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

### 6.2.7.3 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

### 6.2.7.4 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.



### 6.2.7.5 Compare Register

This 24-bit register indicates the comparison value and is cleared after a power-on reset.

### 6.2.7.6 Identification Register

The identification register is read only and fixed at a value of \$81.

### 6.2.7.7 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

### 6.2.7.8 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.

### 6.2.7.9 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.

### 6.2.7.10 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.



### 6.2.8 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  $(A \leq B)$  to  $(A > B)$

When counting down: MATCH is set when A passes from  $(A > B)$  to  $(A \leq B)$

### 6.2.9 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.



### 6.2.10 Configuration

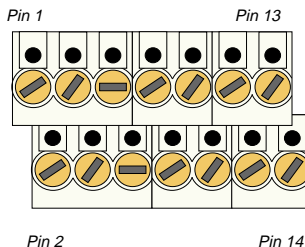
#### Jumper J1 - EEPROM Protection

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

### 6.2.11 Pinouts

#### Screw Terminal Pinouts

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

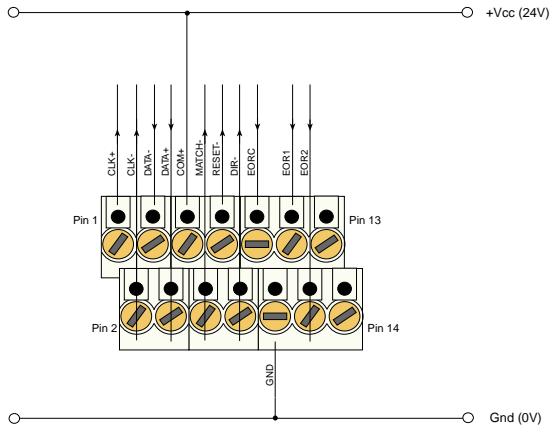


Pin Nr.	Signal	Description	Pin Nr.	Signal	Description
1	CLK+	SSI Clock +	2	CLOCK-	SSI Clock -
3	DATA-	SSI Data -	4	DATA+	SSI Data +
5	COM+	Common +	6	MATCH-	Match -
7	RESET-	Encoder Reset -	8	DIR-	Direction -
9	EORC	EOR Common	10	GND	Ground
11	EOR1	End-of-Range 1	12	EOR2	End-of-Range 2
13		Reserved	14		Reserved

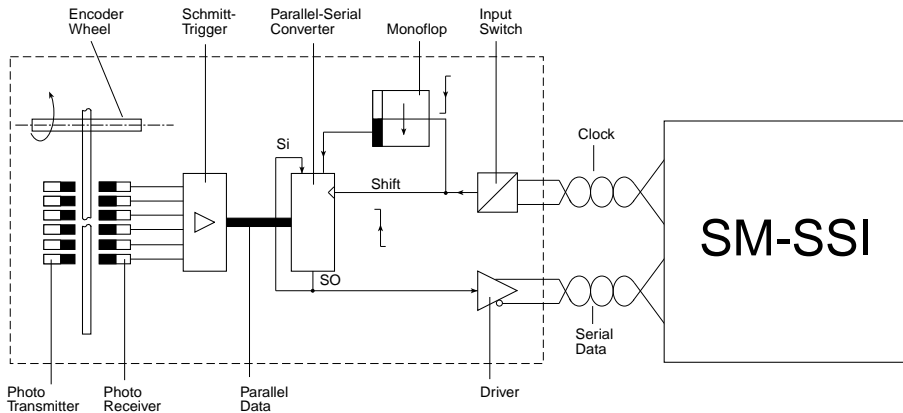




**Connection**



**Input Circuit**





## 6.2.12 'C' Programming

### 6.2.12.1 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 as far as programming is concerned although physically this is slot 1!

Before a library function can be used, the function *SMSSInit* 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 *SMSSDeInit* 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*).



### 6.2.12.2 SMSSIIinit

#### Syntax

```
error_code SMSSIIinit(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

<code>u_int8 PortNr</code>	SM-Port to initialize (from 0 to 10)
----------------------------	---

#### Output

<code>error_code</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 = SMSSIIinit(0);
```







### 6.2.12.3 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

<code>u_int8 PortNr</code>	SM-Port number to de-initialize (from 0 to 10)
----------------------------	---

#### Output

<code>error_code</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);
```



#### 6.2.12.4 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 -> 0xFFFFFFF) 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);
```



### 6.2.12.5 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  $\langle \rangle$  -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 -> 0xFFFFFFF)

#### 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);
```



### 6.2.12.6 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);
```



### 6.2.12.7 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 -> FFFFFFF)

#### 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);
```

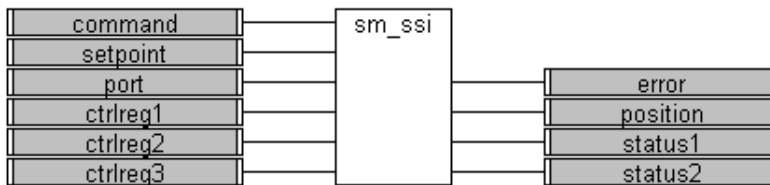


## 6.2.13 ISaGRAF Programming

### 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 - normally 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.



This page has been left blank intentionally.

