STNASINT-E

SYSTEM INTRODUCTION

Model No.: STNASINT-E

Version 1.2

PM/FL/ZR-7/99

SEMINAR START

1 INTRODUCTION

General Information

You have chosen to participate in a system introduction with B&R in order to learn about B&R hardware, programming and application possibilities.

In the next few days - for the duration of this seminar - you will be working together with your seminar leader.

Some of you will want to learn about controllers and their uses while others will have a greater interest in more advanced B&R controllers and programming.

We will do our best to provide you with the maximum of information in your area of interest.

The technical expertise of your seminar leader is not the only factor responsible for your personal success during this seminar.

Success depends on co-operation and interaction between the course members and the seminar leader, as well as the attitude and application of individual course members towards teamwork and the course in general.

Introduction - Seminar Leader

Please allow the seminar leader to introduce her/himself. Make notes if necessary.

_____

_____

_____

_____

Introduction - Course Members

You should get to know your colleagues as you will be working together as a group and also in smaller groups during the seminar (name, company, product, application area, current level of knowledge).

_____

_____

_____

_____

_____

_____

_____

2 SCHEDULE

The time available during the seminar is a very important factor.

Start: _____

Lunch Break: _____

End: _____

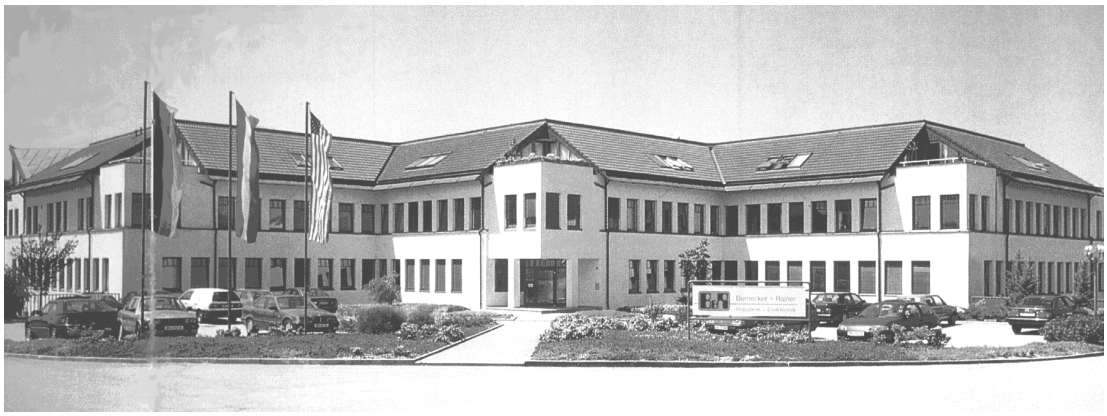
We will taking short breaks at various intervals through out the seminar for tea and coffee and to give the smokers the chance to light up!

3 SEMINAR OVERVIEW

- Introducing B&R
- System Overview
- Hardware
- Programming System
- B&R2000
- Ladder Diagram
- Instruction List
- Structured Text
- Automation Basic
- Service Info
- Project Management

INTRODUCING B&R

1 NOTES



_____

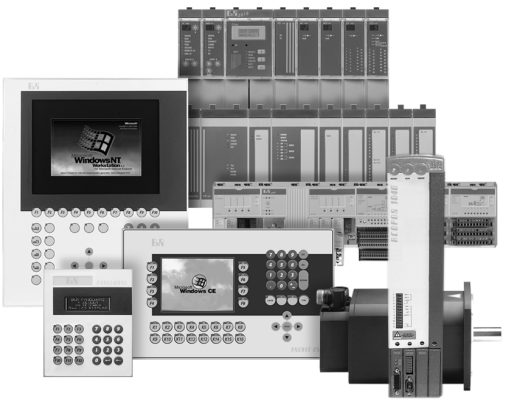
_____

_____

_____

_____

_____



SYSTEM OVERVIEW

1 B&R GENERAL INFORMATION

B&R is Your Worldwide **Automation Partner**.

In addition to the **control systems** MULTI and B&R 2000, the following automation products are also available.

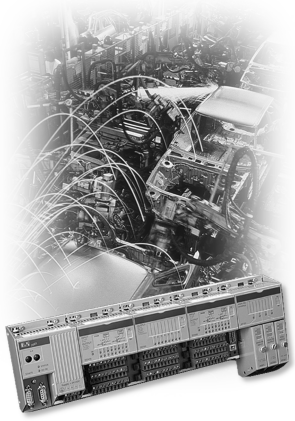
PANELWARE for **simple machine visualization**.

IPC (**Industrial PCs**) for full Graphic Visualization and Data Collection.
Machine visualization software up to **SoftPLC**

Automation Net for Management, Control, and Field Bus Levels.

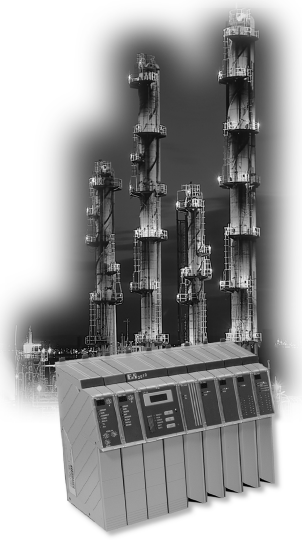
B&R Positioning System from individual axis to CNC applications.

2 B&R PCC SYSTEM



B&R2003

B&R2005



B&R2010

The B&R 2000 PCC family is an Automation System that provides new levels of performance, operational safety and function. The B&R2003, B&R2005 and B&R2010 systems cover the entire range of user requirements from simple Logic Control up to complex remote automations systems.

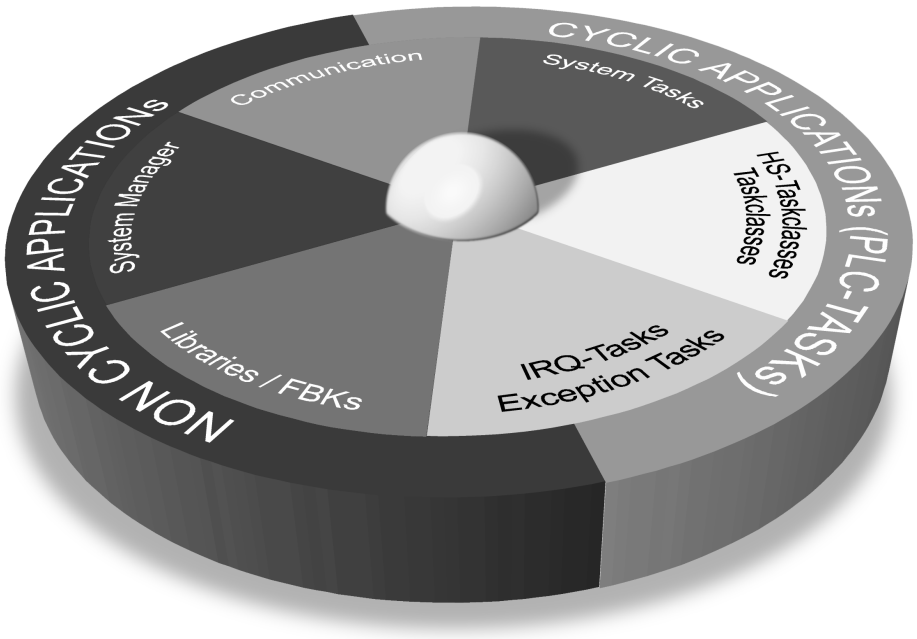
The B&R **SlotPLC** integrates real time in an IPC system. The Dual Processor Architecture guaranties total industrial PC performance for visualization, and improves the operating safety of the PCC system.

The B&R **SoftPLC** enables flexible distribution of system resources between Windows NT and the B&R real time core.

Connection of inputs and outputs is created by expansions, field bus and remote I/O.

B&R2000 PCC Systems

2.1 Operating System



Deterministic multitasking operating system allows timing to be set to an optimum level. Due to the modular construction of the operating system, these settings can be set precisely to match the application itself. AS and integrated error log book offer the user powerful diagnosis tools.

2.2 Programming Device

Windows and DOS offer the user two platforms.

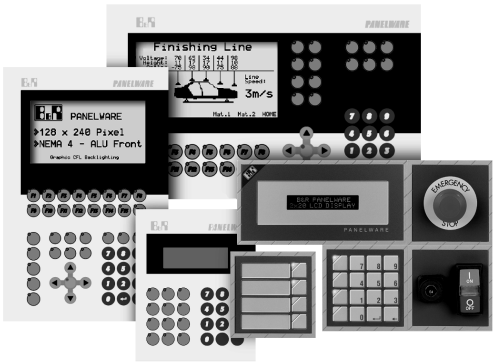
Both support various programming languages according to IEC 1131-3 and the symbolic variable declaration.

For control of the system, variable and system monitors offer varying debugging possibilities.

The included standard software is described thoroughly in ONLINE-help.

2.3 Visualization

PANELWARE is a modular operator panel with keypads, displays and controllers for small visualization systems. With its various components, it is possible to achieve up to 7 million panel variations. Customized panels are also available.

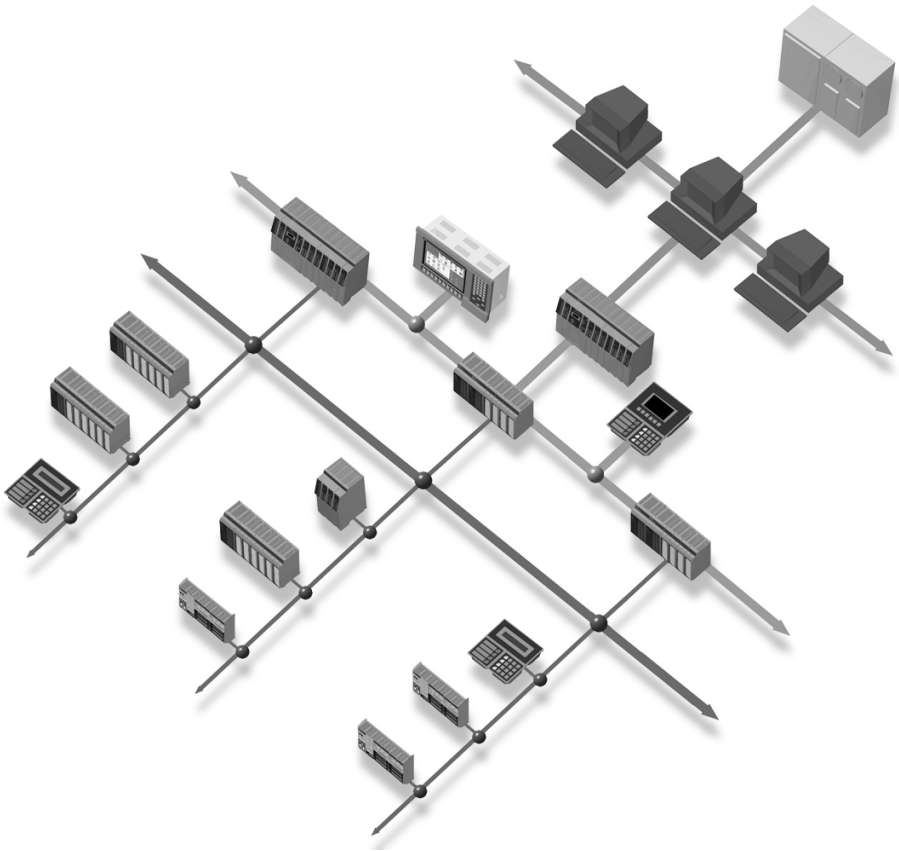


IPC 2000: An AT Compatible Industrial PC. Versions with FLASH disk or Hard disk. Versions with Flat panels, Monitor and PANELWARE keypad modules.

IPC 5000: Modular industrial PC. Pentium Processor. Operate with remote capable Flat displays and PANELWARE keypad modules. ISA and PCI Bus. The IPC 5600 is intended for use in 19" racks.

Industrial - PCs

2.4 B&R Automation Net



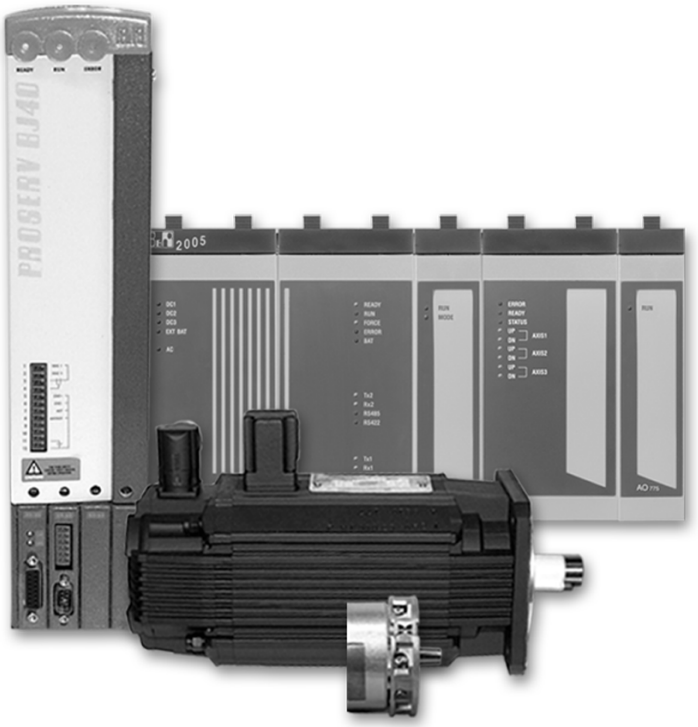
With three levels - management, control and field bus it is possible to network the entire application.

The management level (Ethernet) serves a operational data and statistical handing center.

The control level (Profibus and NET2000) is responsible for process visualization and control tasks.

The field bus level (CAN bus, B&R remote I/O), controls the connection of sensors and actuators.

2.5 B&R Positioning System



The positioning functions are fully integrated into the system.

The servo or stepper motor control provides the user with a full range of possibilities from simple positioning to CNC positioning.

CAM Profiles can be used to set parameters for the most complex relationship between individual axis.

A modular servo amplifier concept allows perfect integration into the standard B&R 2000 product family.

High performance motors with high power density have been developed for extremely fast movements.

B&R Positioning System

3 INTERNATIONAL LICENCES / STANDARDS

B&R products and services meet all required standards. These include the international standards of organizations such as ISO, IEC and CENELEC, as well as national standards of organizations such as UL, CSA, FCC, VDE, ÖVE etc.

We give special consideration to the reliability of our products in industry. The requirements of the product standard IEC 61131-2 for electromagnetic immunity, for example, are exceeded.

3.1 Licenses

Area	Description
USA and Canada	
c UL us	All important B&R products are tested and listed by Underwriters Laboratories and are checked quarterly by a UL inspector. The quality symbol is valid in the USA and Canada and makes it considerably easier to license your machines and systems in these areas.
Europe	
CE	All harmonized EN standards for the valid guidelines are met.
Russian Federation	
CH01	B&R has a GOST certification for all products.
Offshore	
Germanischer Lloyd CERTIFIED SAFETY	The compact controller and individual PANELWARE modules are certified by Germanischen Lloyd.
BUREAU VERITAS	The entire B&R SYSTEM 2005 is certified by Bureau Veritas.

3.2 Standards

Requirements determined by EU guidelines and past experience.

Hardware is put through the following **tests**:

Name	Limiting Value	Description
Noise Immunity		
IEC 61000-4-2 EN 61000-4-2	15 kV Discharge 8 kV Contact Discharge	ESD (electrostatic discharge)
IEC 61000-4-3 EN 61000-4-3	26 – 1000 MHz: 10V/m, 80% Amplitude Modulation 1kHz	HF irradiation
IEC 61000-4-4 EN 61000-4-4	Power Supply 4 kV Digital I/O s (24V) 2 kV Remaining Services 1 kV	BURST The defined B&R limiting values are double that of the values defined in the standard.
IEC 61000-4-5 EN 61000-4-5	1 kV sym. / 2 kV asym.	SURGE
IEC 61000-4-6 EN 61000-4-6	0.15 - 80 MHz: 10 V, 80% AM	Power initiated HF- coupling conducted disturbance, HF inductance
Emitted Disturbance		
EN 50081-2, EN 55022	150 - 500 kHz.: 66-56 dB(µV/m) 0,5 - 5 MHz.: 56 dB(µV/m) 5 - 20 MHz.: 60 dB(µV/m) 30 - 230 MHz.: 30 dB(µV/m) 230 - 1000 MHz.: 37 dB(µV/m)	Emission is measured at a distance of 30m.
Operating Temperature, Humidity, Vibration Resistance		
Operating Temperature	0 to 60 °C	
Humidity	5 to 95% (non-condensing)	
Vibration Resistance	10 - 150 Hz: 1g	

4 PRODUCTION, ENVIRONMENT, SAFETY

The Quality of our products and services has been a priority for many years and is of great importance to B&R. We achieve the **highest level** of quality for our customers.

Since 1993 the company has qualified for the international QM standard **ISO-9001** certificate.

Quality control in the production department covers the following areas. After installation all circuit boards are tested via a **completely automated SMD production in-circuit** test. After final assembly extensive **function tests** are carried out on the control level. These tests are followed by a "**Burn in**".

Environmental awareness is an issue important in every area of the company. It is our obligation to care for the environment and to ensure that only environmentally friendly means of manufacturing are used.

The **safety** of our workers is in the workplace is guaranteed and constantly improved through continued investment and assessment.

5 TERMS AND DEFINITIONS

Analog/Digital
Numerical representation of a physical value. The physical value is converted to a digital value by an A/D converter, according to the resolution.

Binary Data, figures and status information that is made up of 0's or 1's (binary system = base 2 counting system).

BOOL The binary display is also known as boolean display.
0 .. FALSE, 1 .. TRUE;

Bit Smallest unit of information (binary 0 or 1)

Byte Unit of information consisting of 8 Bits.
Values represented either as:

2#0000_0000	-	2#1111_1111	binary
0	-	255	decimal
16#00	-	16#FF	hexadecimal

ASCII **A**merican **S**tandard **c**ode for **I**nformation **I**nterchange
Standardized character set.

Address Identifies location in the memory, or I/O data point on the I/O bus.

Variable Defined memory space, I/O Addresses and internal memory.

Bus Method of communicating addresses, data and information. Links the CPU with the memory, interfaces, co-processors and I/O modules.

CPU Central Processing unit; processes application programs. CPUs are available with a range of performance capabilities and additional functions

APM The application program is developed for the appropriate project and loaded into the application program memory.

ROM Read Only Memory

EPROM Read only Memory which can be erased by UV light (Erasable Programmable ROM).

EEPROM Electrical Erasable Read Only Memory (Electric Erasable Programmable ROM).

Flash-PROM
Electrical Erasable Read Only Memory. However, due to it's construction only the whole chip can be erased.

RAM Random Access Memory. Used to store data while the program is being processed. Must be buffered by batteries.

Programming System
For programming, documenting and testing the required input, visualization and storage devices. A PC is normally the programming system, using the appropriate software (PG software).

Online All PCCs have an interface to connect to the programming device at their disposal. Using this interface programs can be loaded to the controller and program information can be read back.

Ladder diagram LAD
Programming language with graphical symbols as found in circuit diagrams. Can also be used in combination with functions plan and logic plan..

Instruction List
IL. Notation for formulation controller programs. This enables program statements to be arranged as a list.

High Level Language
Collective term for programming languages that allow program orientated formulation that can be run on any type of computer. E.g. C, Pascal, ST.

C Programming language that was developed in connection with the UNIX operating system by D.Ritchie in the AT&T Laboratories.

ANSI Abbreviation for: **A**merican **N**ational **S**tandards **I**nstitute.

ANSI C Standardized C

Function Blocks

Transferal of efficient and complex functions into a "Black Box" which have fixed input and output parameters. A FBK can be called in LAD, IL or ST

Expansion System expansion external to the main base plate. Does not have its own CPU

Remote Placing I/Os at a large distance (up to several kilometers). The transfer rate depends on the distance. Normally does not have its own CPU.

Network Communication between two or more intelligent systems over a pre-defined connection path. You can differentiate between networks because of their physical structure, (two/four conductor cable, coaxial, fiber optic cable), their topology (ring, star, tree structure) and their protocol (language).

Well known networks are: Profibus, Ethernet, Arcnet, CAN, NET2000

Modular construction.

All B&R systems are supplied in a compact construction and can be made to individual customer specification. The advantages of this compact design is the wide ranging expansion possibilities and easy accessibility for servicing or changing the modules. This can also be applied to software which has been assembled from individual components.

Interface Enables communication transfer between the system and the outside world e.g. Printer, screen, PC, networks

Transfer rate

Serial interface download, also baud rate.
Unit : 1 BAUD (bd.) = 1 Bit/Second.

Control Panel

Small visualization device with keypad. Normally only shows numbers, letters and simple graphics.

IPC Industrial PC for medium sized and large visualizations.

WIN95 / WIN98 / WIN NT (4.0 SR3)
Graphical user interface of the Microsoft company.

UNIX Operating system developed in AT&T Laboratories. As it is written in C it is relatively easy to implement on different HW platforms.

LINUX Operating system, made famous by its inventor Linus Torvalds (UNIX for PCs).

OS/2 Operating system of IBM.

Source Code
Source Code is made up of program commands created by the programmer using a text editor and saved in file. This file contains the source code. The code is compiled and transferred to controller or PC and executed.

Source Short version of source code.

File / Document
Basic way of storing information on PCs. Documents such as programs are files. Different data types are assigned different display symbols.

Folder
A folder can contain files and other folders. Organize your work into folders to give an easier overview of what you have as you would do in your office or at home. Your directories are labelled as folders.

Directory / Path
Directory names or path names are used in text orientated operating systems instead of graphical symbols in order to find them easily.

Directory Tree
To give an overview display of directories, programs such as Windows Explorer have a directory tree display function.

Hard drive / Hard drive ID
Letters are used to identify the individual memory mediums (diskettes, hard disks, networks) for example A,B .. disk drives, C-Z .. hard disks, networks).

6 DISPLAY TYPES FOR NUMBERS

When programming it is important to assign values to various variables, therefore various standard types have been defined.

Saving numbers:

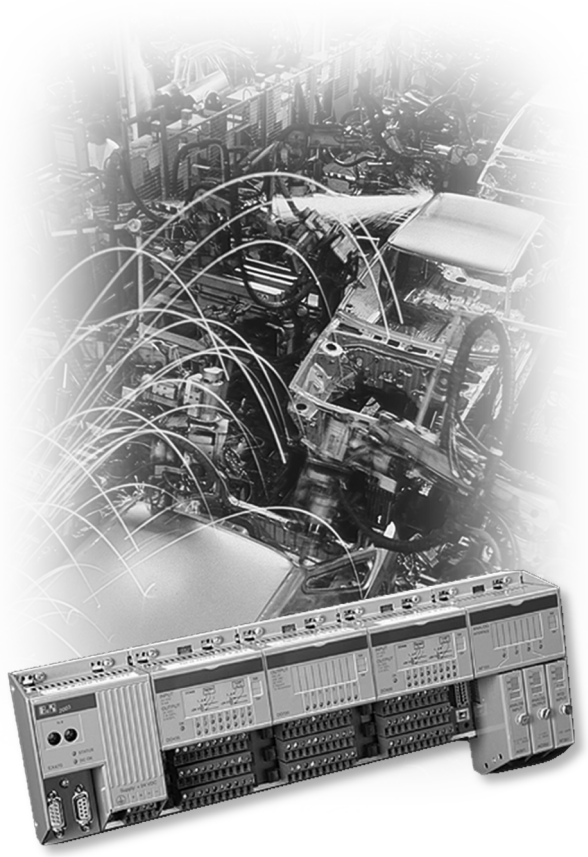
Name	Bit Width	Value Range	Use
BOOL	1	0 .. 1	digital I/Os
DINT	32	-2 147 483 648 .. 2 147 483 647	
INT	16	-32768 .. 32767	analog I/Os
SINT	8	-128 .. 127	
UDINT	32	0 .. 4 294 967 295	
UINT	16	0 .. 65535	
USINT	8	0 .. 255	
REAL	32	-3.4 E38 .. 3.4 E38	

Saving text and data formats:

Name	Width	Value Range	Use
STRING	1 – xx Byte	2 characters – 32767 characters	"Text"
TIME	4 Byte	0 .. 4 294 967 295 msec	Time difference
DATE_AND_TIME	4 Byte	Seconds since 1970	Date calculation

HARDWARE

1 SYSTEM B&R2003



SYSTEM CONFIGURATION:

- B&R2003 Main Controller
- B&R2003 Expansion
    - CAN I/O
    - Remote I/O
- B&R2003 Network

1.1 B&R2003 Main Controller

B&R2003 Module Rack

The system is provided to handle an I/O bus and the CP interface for system-compatible modules.

CP Interface (optional)	CPU PS	I/O bus secure data transfer max. 8 Modules
0		1 2 ...

The module rack has a maximum of 10 module slots.

The module rack is available with room for **2, 3, 4, 5, 6, 8** and **10** module slots.

The module slot on the immediate right of the CPU has module address 1, the second module slot takes address 2 etc.

The **CP interface** for the system module can be installed optionally on the left of the CPU. The CP interface is equipped with spaces for 4 screw-in modules and is assigned module address 0.

A **Digital module** or an **Analog interface module** can be used in any module slot. Every analog interface module has spaces for 4 screw-in modules.

Analog interface modules can only be operated in module slots 1-4 on the B&R 2003 system.

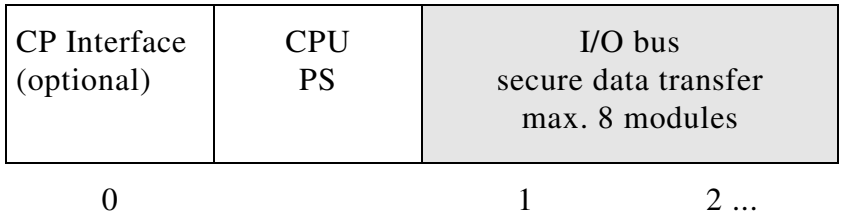
If there are free places on the module rack, a module slot protection cover (AC010) should be installed in the first free module place.

Installation and detailed module information can be found in the **B&R2003 User's Manual.**

B&R2003 Power Supply and CPU

Power supply	DC Module:	18-30V CPU 24V Supply
	AC Module:	85-264V; 47-63 Hz CPU 100V-240V Supply
Protection	Primary side:	**Fuse** protection See catalog or HW manual for details
	Secondary side:	Internal current limit control (**short circuit** and **overload** protection).

The CPU is always positioned on the left of the module rack!



B&R2003 CPU

Performance

This system is a highly user friendly**, real time, multitasking operating system** which is ideally supported by high performance processors and structured programming according to IEC1131-3. A real time clock is also available.

Communication

Interface service is carried out by the CPU.

- In coming data is stored in the buffer.
- Sent data is written to the buffer by the user.

This procedure ensures excellent interface operation.

There are two series interfaces at your disposal:

PG interface	(IF1):	RS232, modem capable (32 Byte FIFO)
User interface	(IF2):	CAN

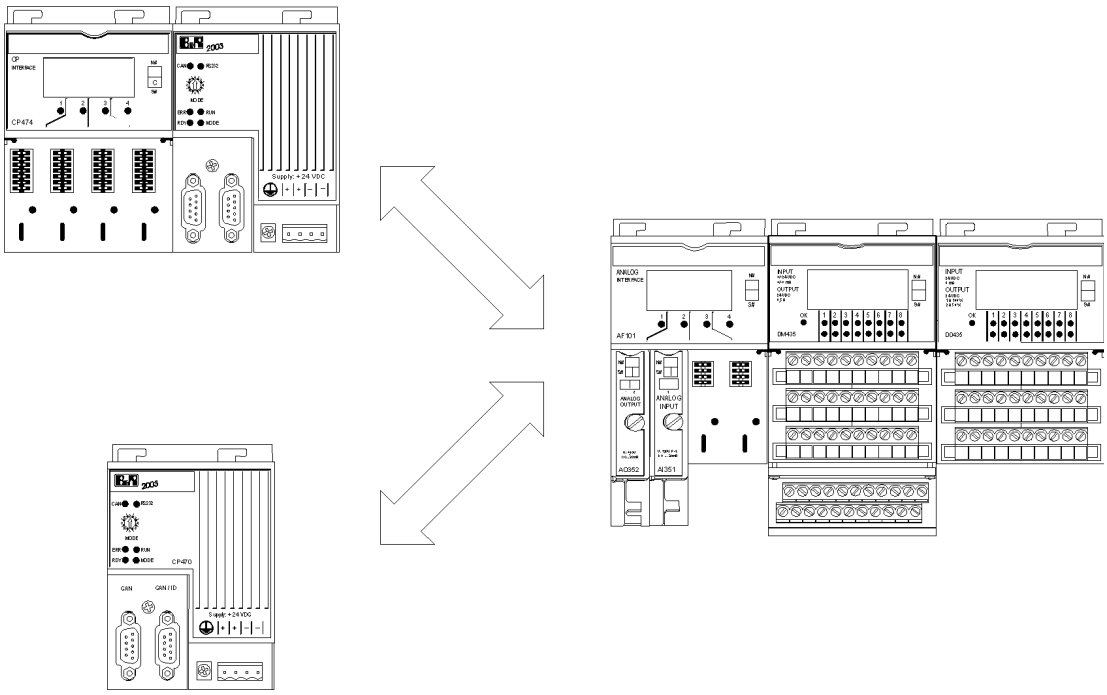
The interfaces are software compatible and can be operated as either online interfaces or data interfaces.

Special Features

Further interfaces can be installed in the first three places on the CP interface CPx74 (screw-in modules: RS232, RS485, RS422; CAN PROFIBUS DP Slave).

B&R2003 CPU Possibilities

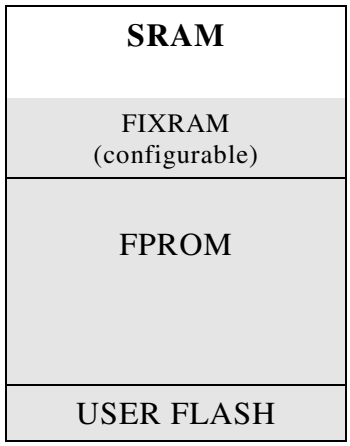
- CPU with interface (4 slots for system modules)
- CPU without interface



B&R2003 Application Memory (APM)

The application memory is integrated in the CPU !

SRAM and **FPROM** are available as the storage mediums for the **APM**.



- **Operating System** (update using **AS**)
- **Program** (tasks)
- **Documentation** (rebuild)
- **Data** (tables)
- **Variable Information** (name, type)

Power failure protection in the APM:

A project is downloaded to the APM-SRAM. The RAM memory is protected by a battery in the CPU.

To ensure **power failure protection**, the project must be burnt on the FPROM.

To ensure **Coldstart** protection, it is necessary to save the data module in FPROM or FIXRAM. (FIXRAM is an SRAM area, that is managed like FPROM).

During the project update in FPROM, the old task is made unrecognizable and the new one is then burned.

If the FPROM is full, the APM can be erased from the CPU by an AS utility.

B&R2003 Module

Digital input module

The status of the digital input is shown by the status LEDs. The most important differentiating characteristics to note are:

- Number of inputs
- **Nominal input voltage**
- **Response time**

Digital output modules

The status of the digital input is shown by the status LEDs. The most important differentiating characteristics to note are:

- Number of outputs
- Types: **Relay or transistor module**
- **Switching voltage and switching current**

Analog input module (supplied as screw-in modules)

The active A/D converter is shown on the analog interface. Each module slot is assigned a status LED (= Run-LED). The most important differentiating characteristics to note are:

- Type: **current or voltage modules** (unipolar or bipolar)
- Resolution: **12 Bit**

Temperature module

The active A/D converter is shown on the analog interface. Each module slot is assigned a status LED (= Run-LED).
The most important differentiating characteristics to note are:

- Number of channels
- Type: **PT100**, **FeCuNi**, **NiCrNi**, **PtRhPt, PT1000, KTY**
- Resolution: **$^1/_{10}$ °C, $^1/_{100}$ °C**

Analog output module

The active A/D converter is shown on the analog interface. Each module slot is assigned a status LED (= Run-LED).
The most important differentiating characteristics to note are:

- Type: **current or voltage modules** (unipolar or bipolar)
- Resolution: **12 Bit**

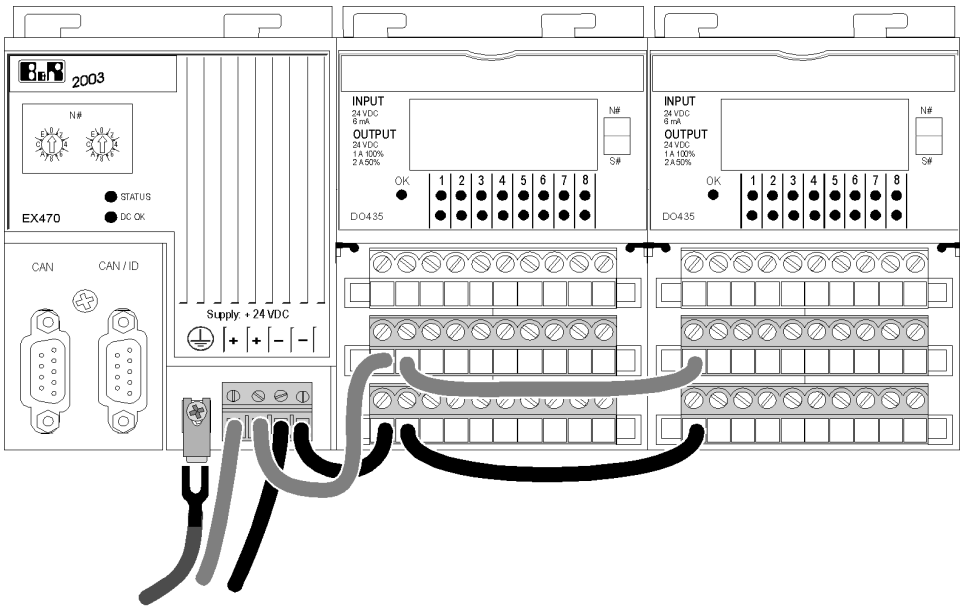
Other modules

- **Digital mixed module, Interface module, Counter module**

B&R2003 Terminal Block

Digital module

Terminal blocks are used for all digital modules.



Contact **10 electrical contacts** can be connected to the terminal block.

Efficiency The **separation between channel, supply and GND terminal blocks** permits an extremely efficient and clearly arranged connection scheme.

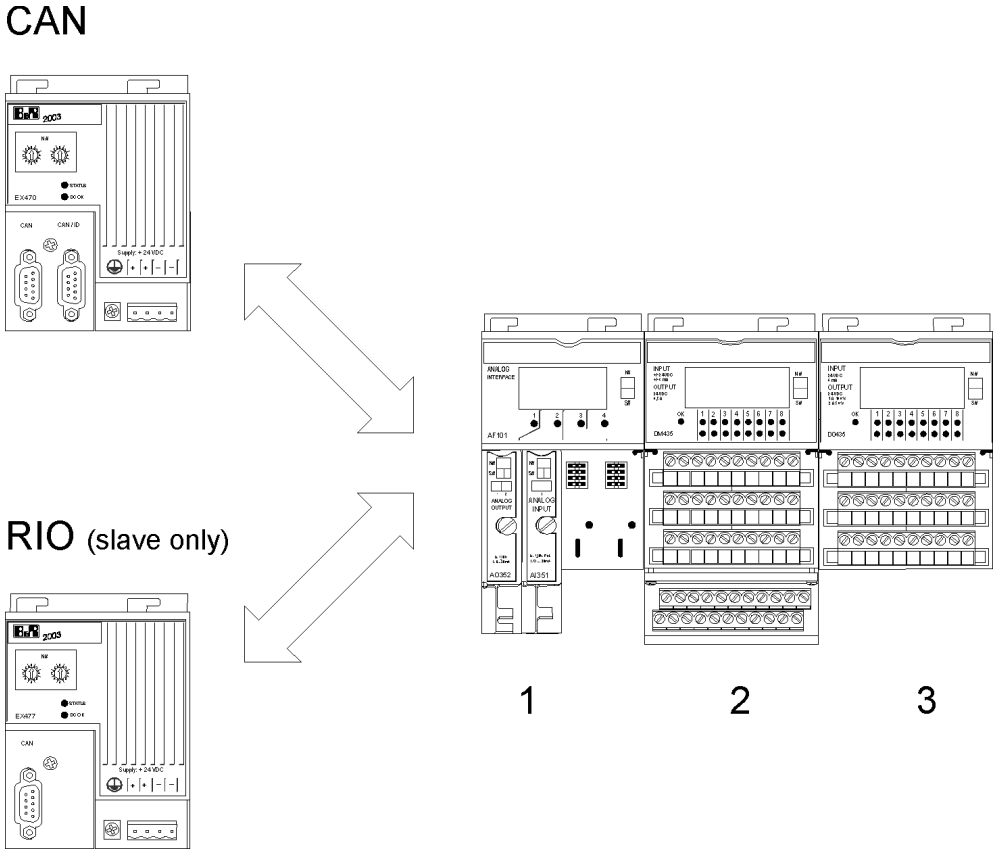
Ejection The terminal block enables **easy ejection** of the modules**.**

Analog modules and system modules

The connection is made according to the instructions on the analog screw-in module.

1.2 B&R2003 Expansion

The B&R2003 System can be expanded in two ways. With CAN and RIO.



B&R2003 CAN I/O

The system layout can be expanded over a large distance and the inputs positioned where required by installing CAN I/O. The 'remote controller' concept is achieved by this method.

A maximum of **64 CAN** stations can be connected.

The maximum bus length is **1000m.**

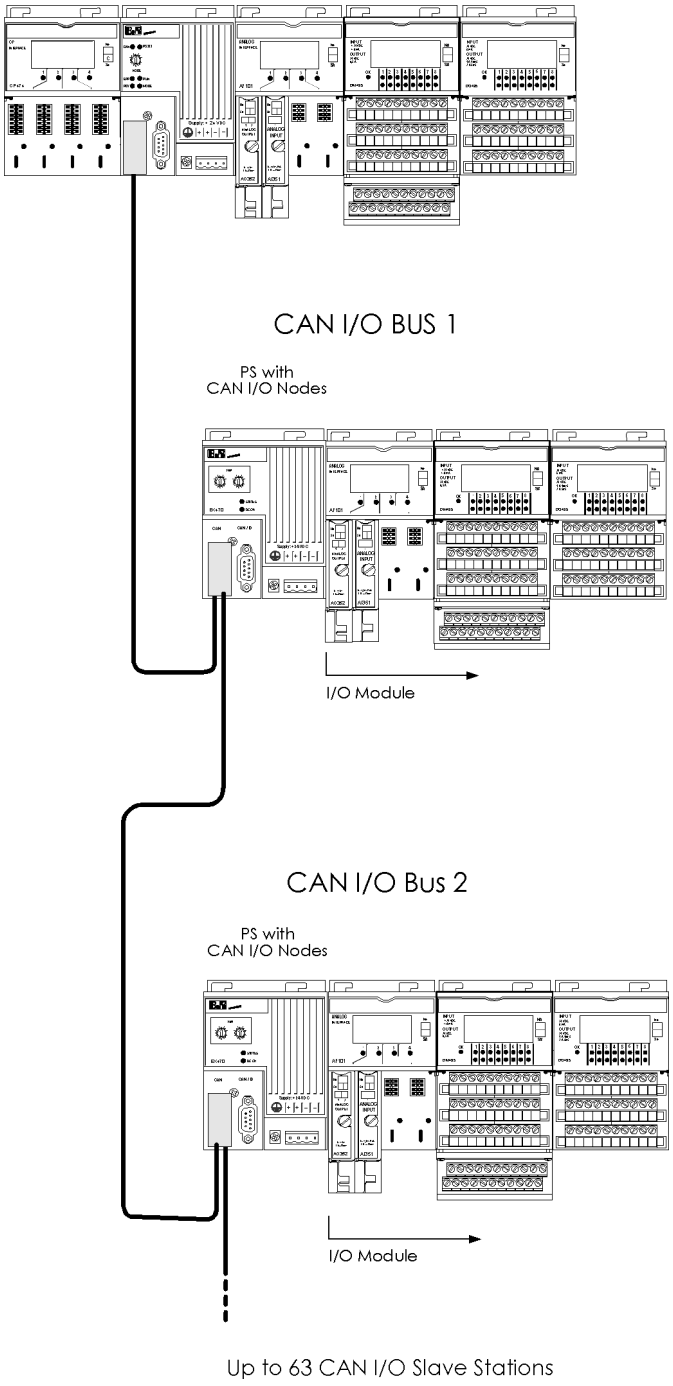
The transmission speed is between **500kBaud** and **10kBaud** depending on the expansion distance**.**

The **Master** can be used with any processor that has a CAN interface:
B&R2003, B&R2005, B&R2010, PANELWARE, PROVIT

Every CAN I/O station consists of a **CAN bus slave module (EX470/EX770)**. This module is used instead of a CPU and thus can be installed at the left end of the module rack.

Connection to the individual station is achieved by using **shielded three wire cable.**

B&R2003 CAN I/O



B&R2003 Remote I/O

The system layout can be expanded over a large distance and the inputs positioned where required by installing remote I/O. The 'remote controller' concept is achieved by this method. Cabling requirements are minimized by the **2 wire bus**.

The number of remote stations depends on the connection:

- 31 Remote I/O slave station
- 121 with repeater

Maximum **segment length** is **1200m**. Maximum distance between the first and last station is **4800m** using **3 repeaters**.

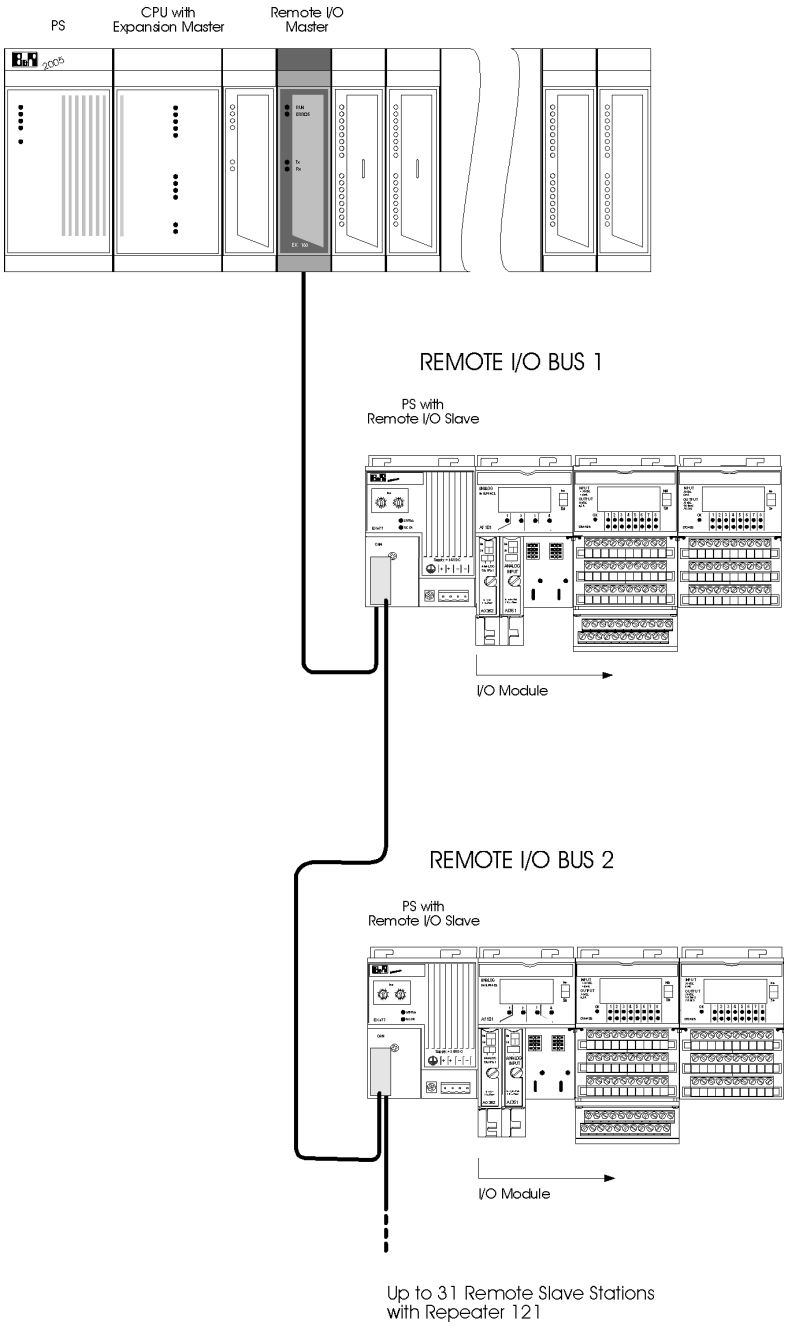
The transmission speed is between **2MBaud** and **100KBaud** depending on the expansion of the remote system.

The **remote I/O master is a B&R2005 or B&R2010 CPU.**

Every remote I/O slave station consists of a **remote I/O slave: EX477/EX777**. This module is used instead of the CPU.

The connection of the individual stations is achieved using **shielded twisted pair cable.**

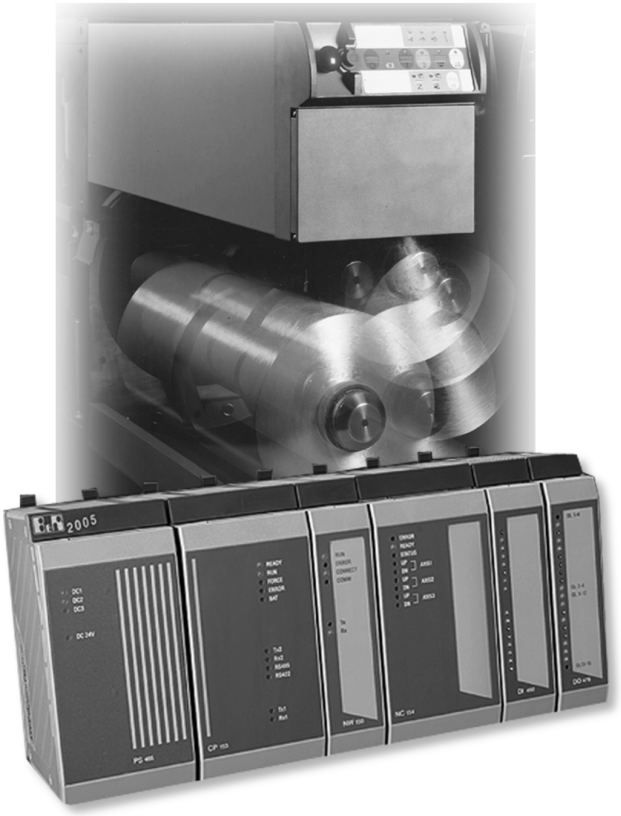
B&R2003 Remote I/O



1.3 B&R2003 Network

The B&R2003 system can be connected with its network partners via a serial interface. A CAN, or Profibus DP network can also be used.

2 SYSTEM B&R2005



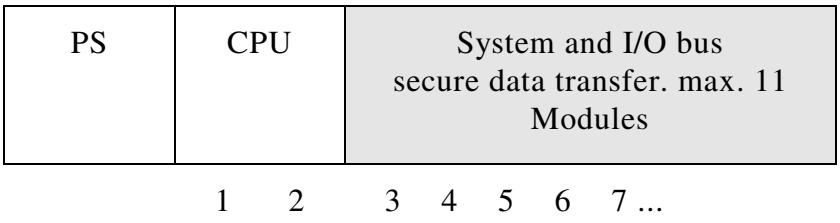
SYSTEM CONFIGURATION:

- B&R2005 Main Controller
- B&R2005 Expansion
    - Expansion
    - Remote I/O
    - CAN I/O
- B&R2005 Network

2.1 B&R2005 Main Controller

B&R2005 Base plate module

The system is used together with the I/O bus.



The **base plate module** is available with or without **battery provision** and has a maximum of **15 insert slot**s. It can be supplied with **6, 9, 12** or **15** module slots.

A maximum of 4 expansions can be used to increase the number of data points.

The modules can be placed in any slot on the main base plate except for the power supply which has a fixed position. Each slot on the base plate has a predetermined module address. The first slot next to the power supply is given the module address 1, the second takes module address 2 etc. CPUs are normally assigned slot 1.

The dummy module (BM150) should be inserted in any free slots on the base plate.

For installation and detailed module information, please refer to the:
B&R SYSTEM 2000 HARDWARE USERS MANUAL

B&R2005 Power Supply

Power Supply The output is 50 Watt

DC power supply:	24 V	18-30V	
AC power supply:	120V	92-133V;	47-63 Hz
	220 V	187-265V;	47-63 Hz

Different types of power supply are available:

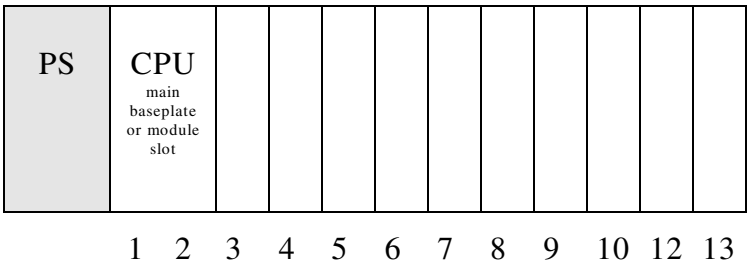
- Expansion slave
- Remote slave
- Expansion slots

Protection

Primary side: **Fuse** protection.
see catalog or HW manual for details

Secondary side: Internal current limit control (**short circuit** and **overload** protection). **Relay contact** is opened when error occurs. Secondary voltage is switched off.

The power supply has a fixed position on the base plate.



B&R2005 CPU

Performance

The high performance of the CPU is achieved by using a **processor with integrated RISC** (Reduced Instruction Set Computer). The system is highly user friendly incorporating a **real time and multitasking operating system**, which is ideally supported by high performance processors and structured programming according to IEC1131-3. A real time clock is also available.

Communication

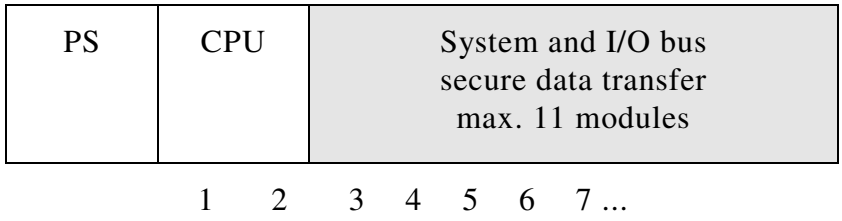
Interface service is carried out by the RISC.
Two series interfaces are available:

User interface:	RS485/RS422/TTY
PG interface:	RS232, modem capable

The interfaces are software compatible and can be operated as either online or data interfaces.

Slot

The CPU can be operated from any position on the main base plate. However, it is standard procedure to install the CPU next to the power supply. It is not possible to install a CPU in an expansion unit.



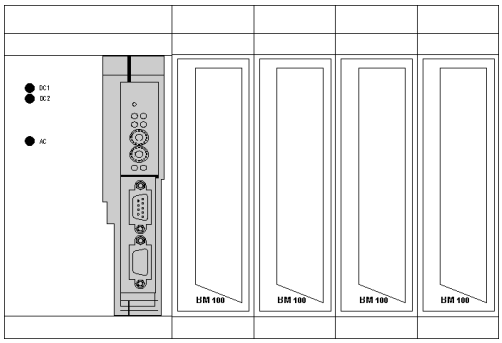
Special types of Module

The following modules can carry out the same functions as the CPU. When installed on the rack next to the CPU, they act as a parallel processor (ref: Application memory onboard, buffering through the backplane).

XP	...	In the power supply expansion slot Interface RS232, CAN
IF	...	If this module is installed in slot number 1, it takes over the functionality of the CPU. Otherwise the module operates as a parallel processor. Interface: RS232, CAN
IP	...	As IF. Interface: RS232, CAN. In addition this processor offers faster I/O's.

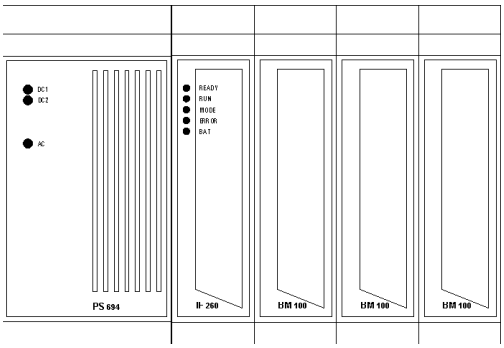
Possible combinations of PS, CPU and parallel processors

XP152 as the CPU



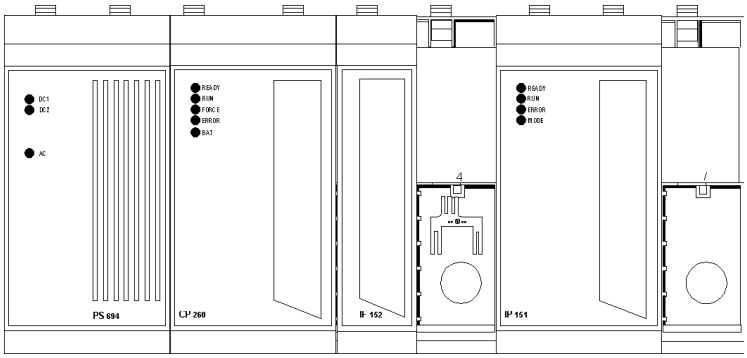
XP I/O I/O ...

Standard Configuration



PS IF260 I/O I/O

XP..., IF... or IP... as multi-processors

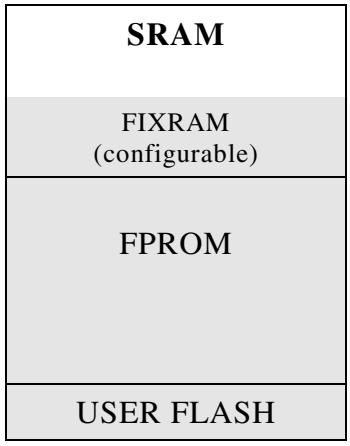


PS CP260 IFxxx Ipxxx

A maximum of two parallel processors are possible! CP260 a maximum of 4!

B&R2005 Application Memory

SRAM and **FPROM** are available as APM.



- **Operating system** (update through AS)
- **Program** (tasks)
- **Documentation** (rebuild)
- **Data** (table)
- **Variable Information** (name, type)

Power failure protection on the APM:

The project is transferred to the APM RAM.

Buffer options (see catalog)	Accu Gold leaf capacitor (min. 10 min, for battery change)
	Battery
	External buffer supply (optional use, AC240)

To ensure **power failure protection**, or to send prepared APM, the project must be burned on the APM PROM.

To ensure **COLDSTART** protection, it is necessary to save the data module in the FIXRAM (FIXRAM is a SRAM area that is managed like FPROM).

During the project update in FPROM, the old project is made unrecognizable and the new one is then burned. If the FPROM is full, the APM can be erased from the CPU by a PG utility.

B&R2005 Module

Digital input module

The status of the digital input is shown by the status LEDs.
The most important differentiating characteristics to note are:

- Number of inputs
- **Nominal input voltage**
- **Response time, counter inputs**

Digital output modules

The status of the digital input is shown by the status LEDs.
The most important differentiating characteristics to note are:

- Number of outputs
- Types: **Relay, transistor or triac module**
- **Switching voltage and switching current**
- Maximum **potential difference**

Analog input module

The active A/D converter is shown by the status LED (Run LED).
The most important differentiating characteristics to note are:

- Type: **current** or **voltage modules** (unipolar or bipolar)
- Resolution: **12 Bit**

Temperature module

The active A/D converter is shown by the status LED (Run-LED).
The most important differentiating characteristics to note are:

- Number of channels
- Type: **PT100 (**3-, 4 conductors**), FeCuNi** (type-L, -J), **NiCrNi** (type-K)
- Resolution: **$^1/_{10}$ °C, $^1/_{100}$ °C, $^1/_{10}$ F**

Analog output module

The active A/D converter is shown by the status LED (Run LED).
The most important differentiating characteristics to note are:

- Type: **current** or **voltage modules** (unipolar or bipolar)
- Resolution: **12 Bit**

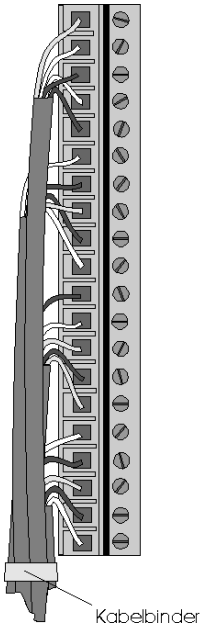
Other Modules

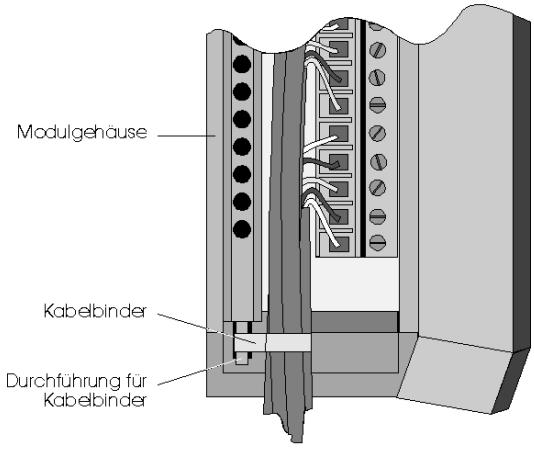
- **Analog mixed module/digital mixed module**
- **Network module, Interface module, Positioning and CNC module, Dummy module**

B&R2005 Terminal Block

Contact UP to **20 electrical contacts** can be connected to the terminal block. Screws and cage clamps are provided for this.

Ejection The terminal block enables **easy ejection** of the modules.





2.2 B&R2005 Expansion

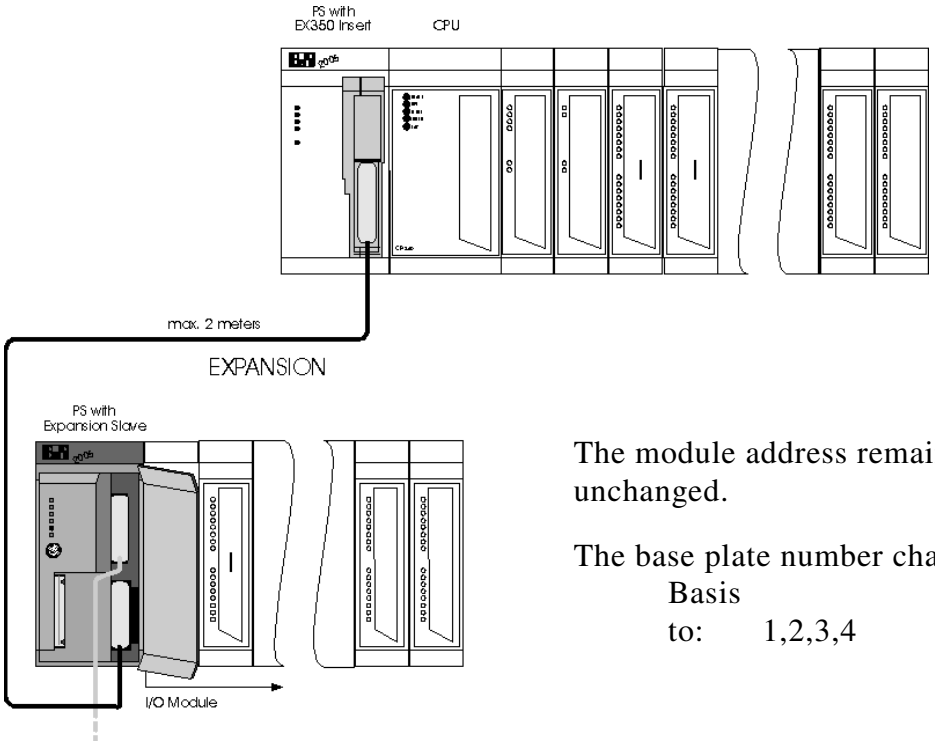
B&R2005 Expansion

Expansion is:

- Separating the I/O bus.
  The PCC can be ideally adjusted to suit the available space on the control cabinet.
- Expansion of the I/O bus.
  An additional 52 I/O modules can be operated from a maximum of 4 expansion stations.

The expansion master serves the **CPU via an integrated expansions master** or the **EX350 module**.

The **power supply** is served by an **expansions slave.** The **connection cable** between master and slave is available in **1m** or **2m** lengths.



The module address remains unchanged.

The base plate number changes from:
Basis
to: 1,2,3,4

B&R2005 Remote I/O

The system layout can be expanded over a large distance and the inputs positioned where required by installing a remote I/O. The remote controller concept is achieved by this method. Cabling requirements are reduced by the 2 wire bus.

The number of remote stations depends on the connection:

- 31 remote I/O slave stations
- 121 with repeater

Maximum **segment length** is **1200m**.
Maximum distance between the first and last station is **4800m** using **3 repeaters**.

The transmission speed is between **2MBaud** and **100KBaud** depending on the expansion of the remote system.

The **remote I/O master is a system module**, that in combination with the I/O module is installed on the main base plate.

Every remote I/O slave station comprises of a **power supply and remote I/O slave**.

The connection of the individual stations is achieved using shielded twisted pair cable.

B&R2005 with CAN I/O

The B&R2005 interface module with CAN interface can be connected with a B&R2003 CAN I/O.

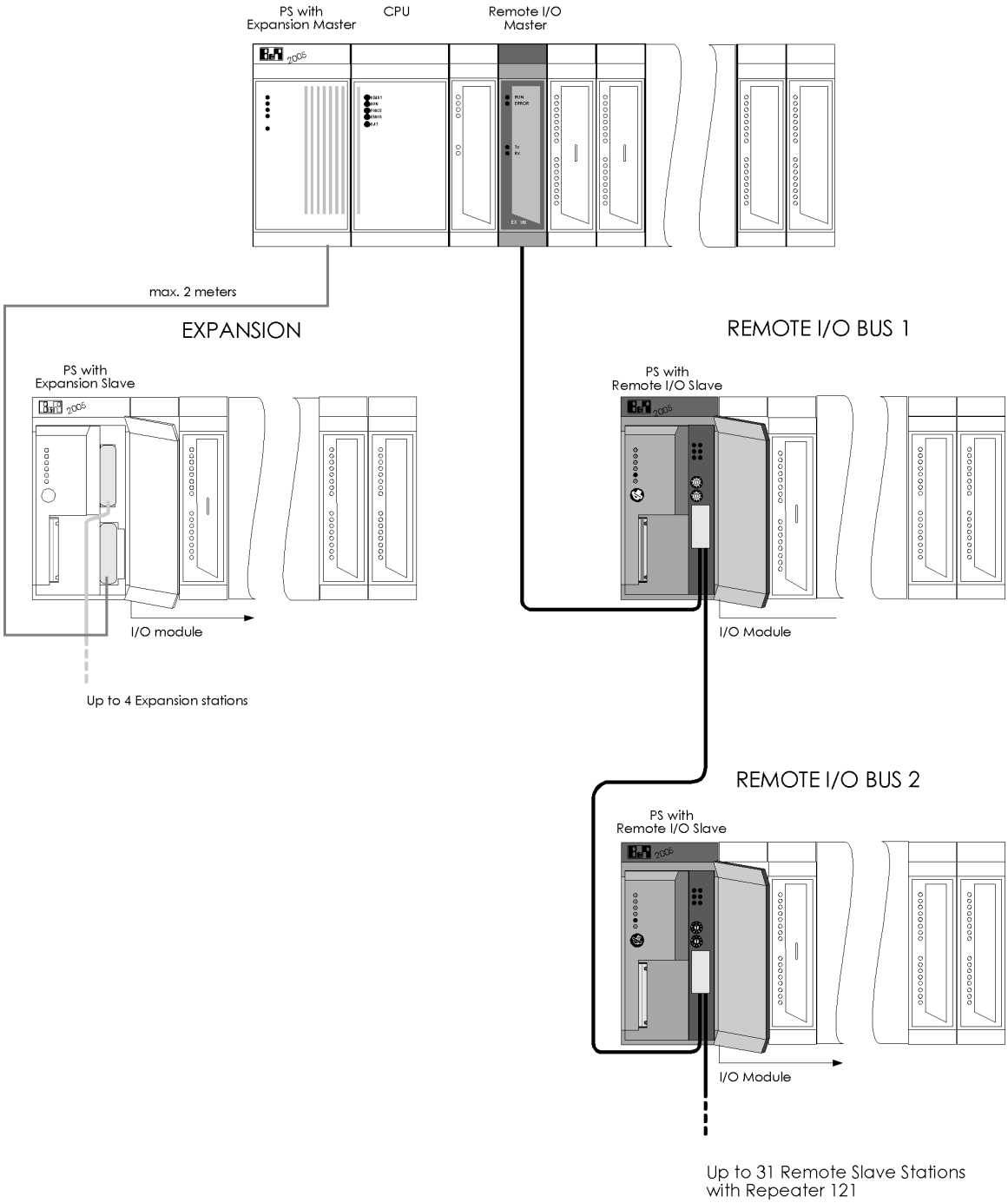
For detailed information see B&R2003 Hardware manual.

2.3 B&R2005 Networks

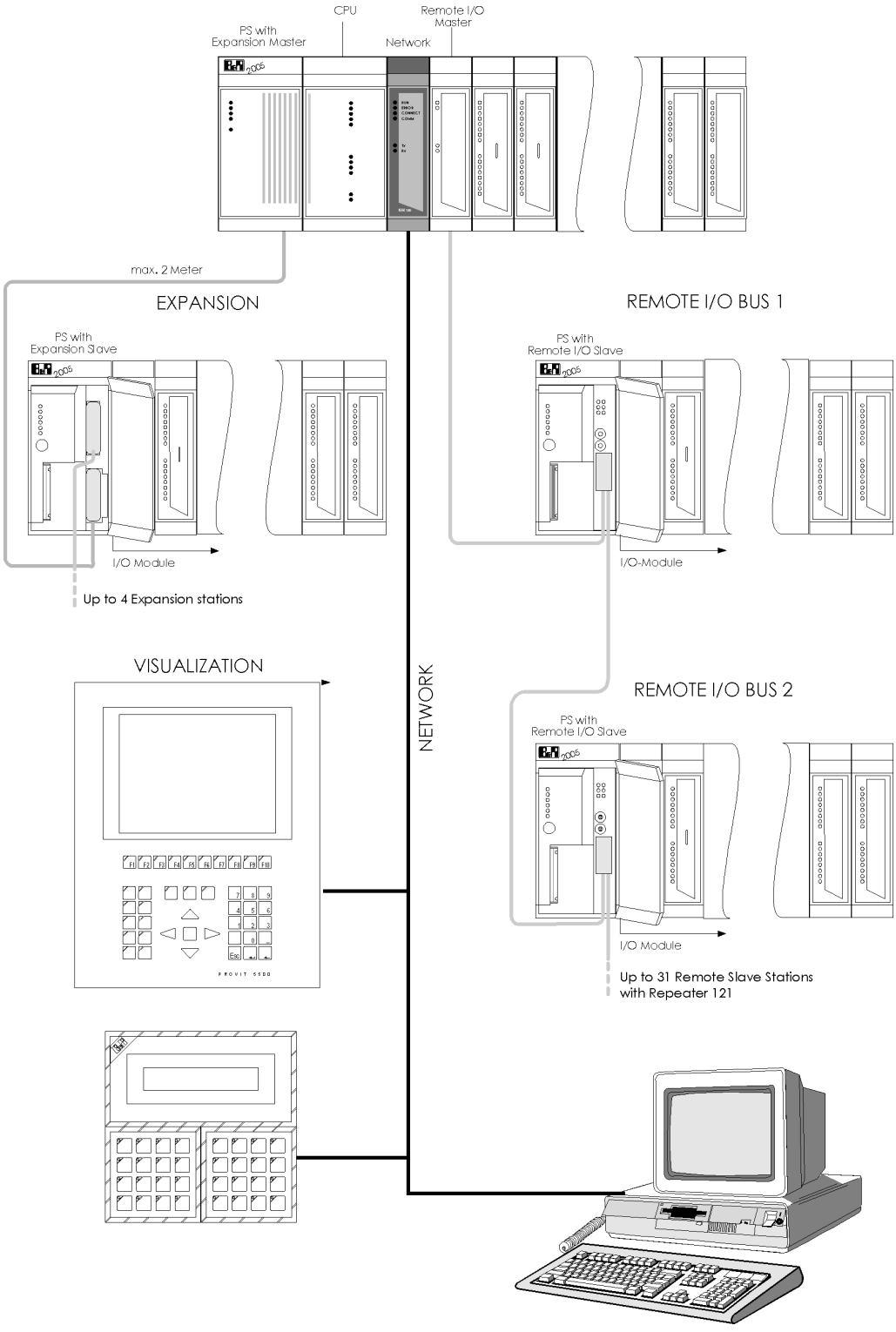
The B&R2005 can be connected to many different standard networks by installing a network module.

Programming is also possible via networks (PROFIBUS, CAN).

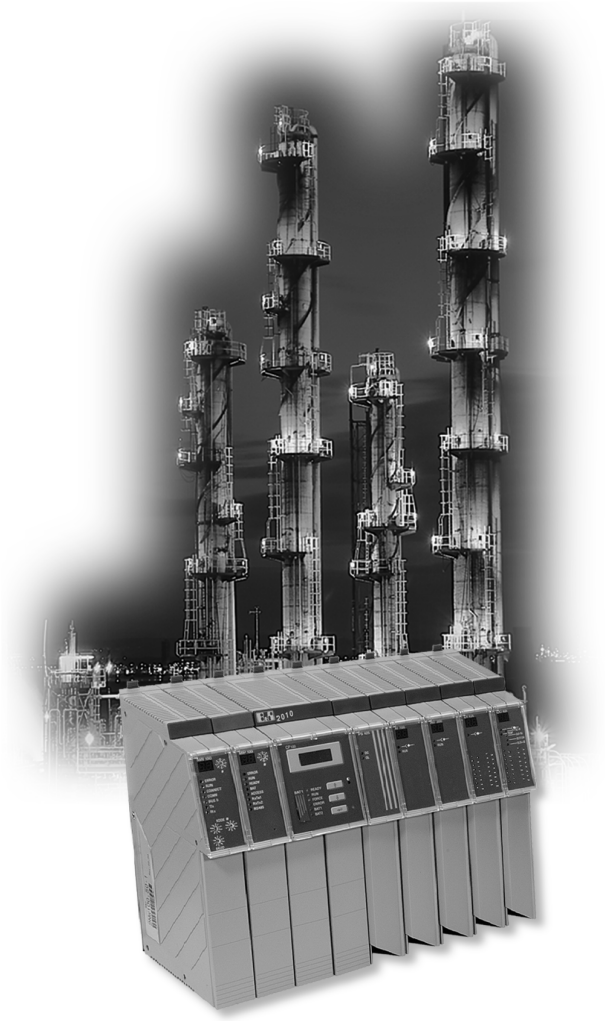
B&R2005 Expansion and Remote I/O



B&R2005 Possibilities



3 SYSTEM B&R2010



SYSTEM CONFIGURATION:

- B&R2010 Main Controller
- B&R2010 Expansion
    - Expansion
    - Remote I/O
    - CAN I/O
- B&R2010 Network

3.1 B&R2010 Main Controller

B&R2010 Base Plate Module

The system controls a separate system and I/O bus.

System bus max. 8 Module	CPU	**System and I/O bus** secure data control max. 20 Module/Rack

...4 3 2 1 1 2 3 4 5 6 7 ...

An **I/O bus segment** can comprise of up to a maximum of **20 modules.**

An **I/O bus** can be expanded with a maximum of **99 modules**.

The module slot on the immediate right of the CPU is assigned module address 1, the second slot is given module address 2, etc. Every I/O module address is visible to the user on a two character status display.

System and I/O buses are of modular construction.

The length is dependent on the number of modules installed on the main base plate.

The system and I/O buses are both terminated on the **main base plate** with a **bus termination connector**.

System Bus: 2 slot
4 slot with bus termination connector

CPU: with system bus connector

I/O-Bus: 1 slot with or without bus termination connector
4 slot without bus termination connector
1 slot for expansions slave and remote I/O slave

The dummy module (BM100) should be inserted in any free slots on the base plate.

For installation and detailed module information please refer to the:
B&R SYSTEM 2000 HARDWARE USERS MANUAL

B&R2010 Power Supply

Power Supply The output is 100 Watts

DC power supply: 18-30V

AC power supply: 90-270V; 47-63 Hz;

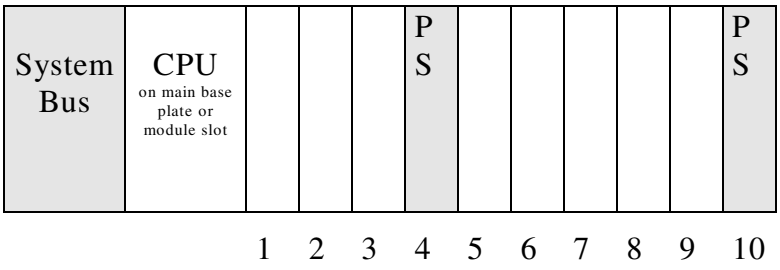
Protection

Primary side: **Fuse** protection.
See the catalog or HW manual for more details.

Secondary side: Internal current limit control (**short circuit** and **overload protection**). OL Led is lit when error occurs. **Relay contact** is opened when error occurs. Secondary voltage is switched off.

Slot

Power supply module is only inserted on the I/O bus The power supply should only be inserted according to the power demand requirements. The I/O bus power supply slots are freely available and should be distributed across the complete bus.



Special features

Decentralized supply concept. Each module provides its own voltage and can be inserted or removed even when the **power supply** is on (exceptions: CPU, APM, system module).

Supply redundancy is reached when you use twice as many power supplies than are required.

Using the toggle switch on the power supply (PS740) enables the secondary supply (generated 24V DC bus supply) to be switched:

- forwards to the terminal block or
- backwards to the I/O bus.

B&R2010 CPU

Performance

The high performance of the CPU is achieved by using a **processor with integrated RISC**, and by assigning tasks to more processors. The system is highly user friendly incorporating a **real time and multitasking operating system** which are ideally suited to the high performance processors and structured programming according to IEC1131-3. A real time clock is available.

Communication

Interface service is carried out by the RISC.

- Received data is stored in the buffer.
- Sent data is written to the buffer by the user.

This procedure offers excellent interface service.

Three series interfaces are at your disposal: (see catalog)

- PG interface (IF1): RS232, modem capable
- User interface (IF3): RS485, RS422 or CAN (depending on CPU type)
- User interface (IF2): RS232

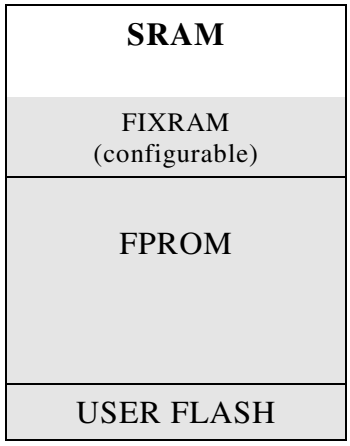
The interfaces are software compatible and can be operated as either online interfaces or data interfaces.

The CPU is mounted on its own CPU base plate module.

System bus max. 8 modules	CPU	**System and I/O bus** secure data transfer max. 99 Modules

B&R2010 Application memory

SRAM and **FPROM** are available as APM.



- **Operating system** (update via AS)
- **Program** (tasks)
- **Documentation** (rebuild)
- **Data** (Tables)
- **Variable Information** (name, type)

Power failure protection in the APM:

The project is downloaded to the APM SRAM.
RAM memory protection is achieved by:

CPU: Battery (see catalog)
Gold leaf capacitor (min. 10 min, for battery change)

APM: Battery (see catalog)

To ensure **power failure protection**, or to use the prepared APM, the project must be burnt on the APM PROM.

To ensure **COLDSTART protection**, it is necessary to save the data module in the FIXRAM (FIXRAM is an SRAM area, that is managed like FPROM).

During the update in FPROM the old project is made unrecognizable and the new one is then burned.

If the FPROM is full, the APM can be erased from the CPU using an AS utility.

B&R2010 Module

Digital Input

The status of the digital input is shown by the status LEDs.
The most important differentiating characteristics to note are:

- Number of inputs
- **Nominal input voltage**
- **Response time**
- Specific types e.g. **interrupt** operation

Digital output modules

The status of the digital input is shown by the status LEDs.
The most important differentiating characteristics to note are:

- Number of outputs
- Types: **Relay or transistor module**
- **Switching voltage and switching current**
- **Maximum potential difference**

Analog input module

The active A/D converter is shown by the status LED (Run LED).
The most important differentiating characteristics to note are:

- Type: **current** or **voltage modules** (unipolar or bipolar)
- Resolution: **12 Bit**

Temperature module

The active A/D converter is shown by the status LED (Run LED). The most important differentiating characteristics to note are:

- Number of channels
- Type: **PT100 (**3, 4 conductors**), FeCuNi** (type L)
- Resolution: **$^1/_{10}$ °C, $^1/_{100}$ °C**

Analog output module

The active A/D converter is shown by the status LED (Run LED).
The most important differentiating characteristics to note are:

- Number of channels
- Type: **current** or **voltage modules** (unipolar or bipolar)
- Resolution: **12 Bit**

Other modules

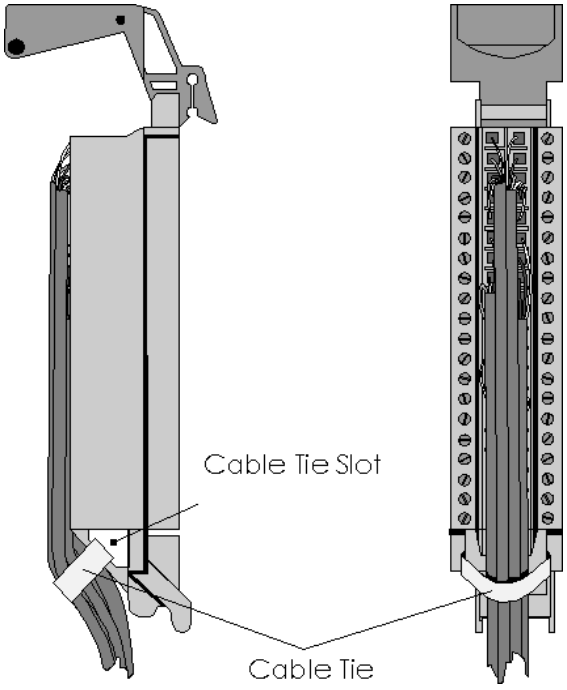
- **Mixed module, digital and analog**
- **Multi processor, network module, interface processor**
- **Multi-function module, drum sequencer, positioning module**
- **Dummy module**

B&R2010 Terminal Block

Contact Up to **40 electrical contacts** can be connected to the terminal block. The cable bunch can be fixed directly on the terminal block using a **cable tie**.

Ejection The terminal block enables **easy ejection** of the module and which functions as a locking lever at the same time.

Protection The terminal block has six **code switches** that prevent the terminal being exchanged. An insecurely installed terminal block is indicated by the **status LED** at the top of the module. This status can also be **verified using the program**.



3.2 B&R2010 Expansion

B&R2010 Expansion

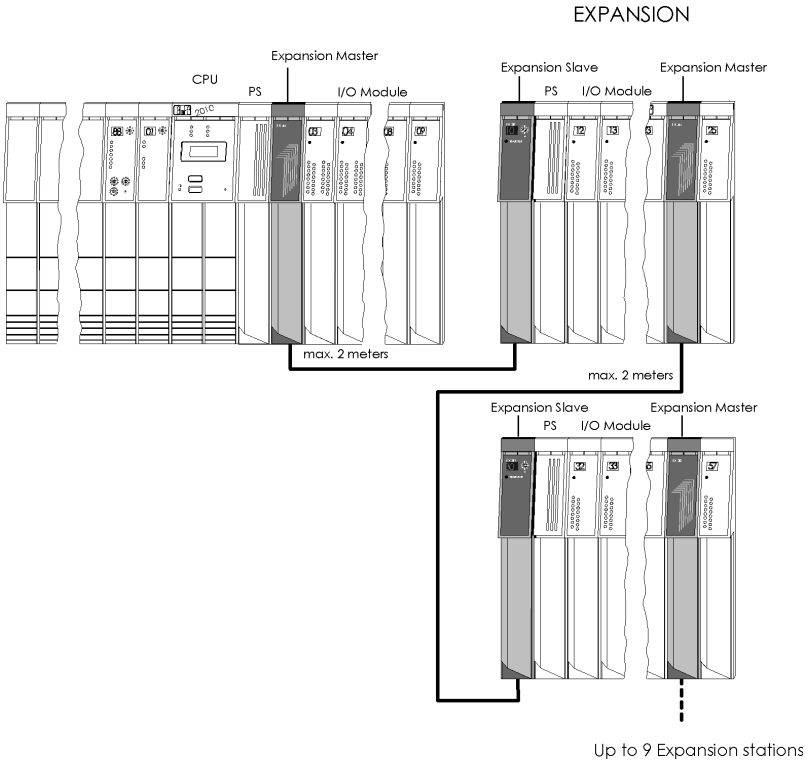
Expansion is:

- Separating the I/O bus.
  The PCC can be ideally adjusted to suit the space available in the control cabinet.
- Expansion of the I/O bus.
  An I/O bus segment (expansion station,..) can handle a maximum of 20 modules. An I/O bus can operate a maximum of 99 modules which corresponds to a bus length of 4m.

The expansion master used is the I/O bus module **expansion master**.

The expansion slave used is the I/O bus module **expansion slave**. The **expansion slave is the first module to be installed on it's own base plate**.

The **connection cable** between master and slave is available in **1m** and **2m** lengths.



B&R2010 Remote I/O

The system layout can be expanded over a large distance and the inputs positioned where required by installing remote I/O. The 'remote controller' concept is achieved by this method. Cabling requirements are reduced by the 2 wire bus.

The number of remote stations depends on the connection:

- 31 remote I/O slave stations
- 121 with repeater

Maximum **segment length** is **1200m**.

Maximum distance between the first and last station is **4800m using 3 repeaters**.

The transmission speed is between **2MBaud** and **100KBaud** depending on the expansion of the remote system.

The **remote I/O master is a system module**, that is installed in the system bus slot of the main base plate.

Every remote I/O slave station comprises of a **remote I/O slave** that is installed in the first expansion slot on the main base plate.

The connection of the individual stations is achieved using shielded twisted pair cable.

B&R2010 with CAN I/O

The B&R2010 interface module with CAN interface can be connected with a B&R2003 CAN I/O.

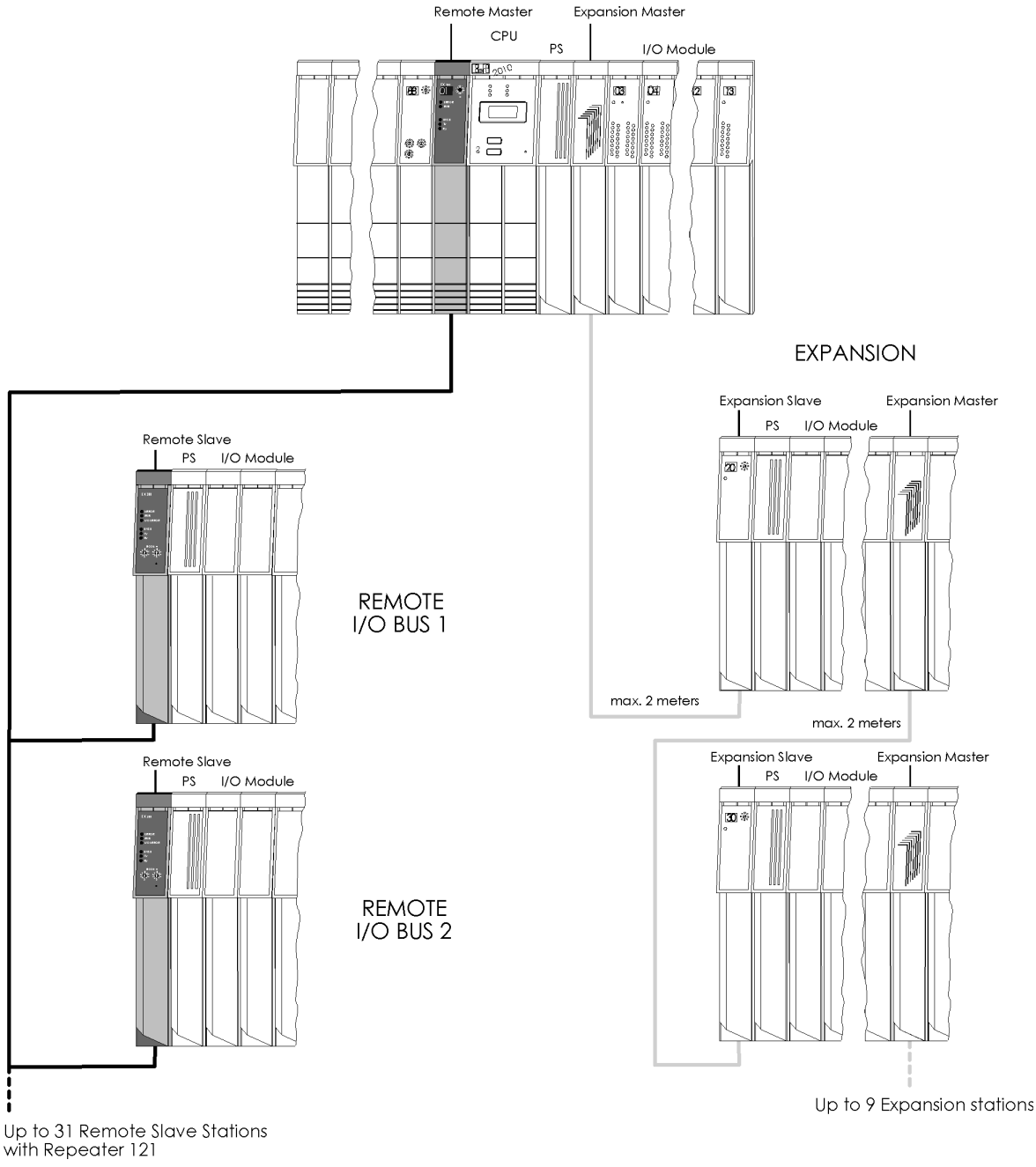
For detailed information see B&R2003 Hardware Manual.

3.3 B&R2010 Networks

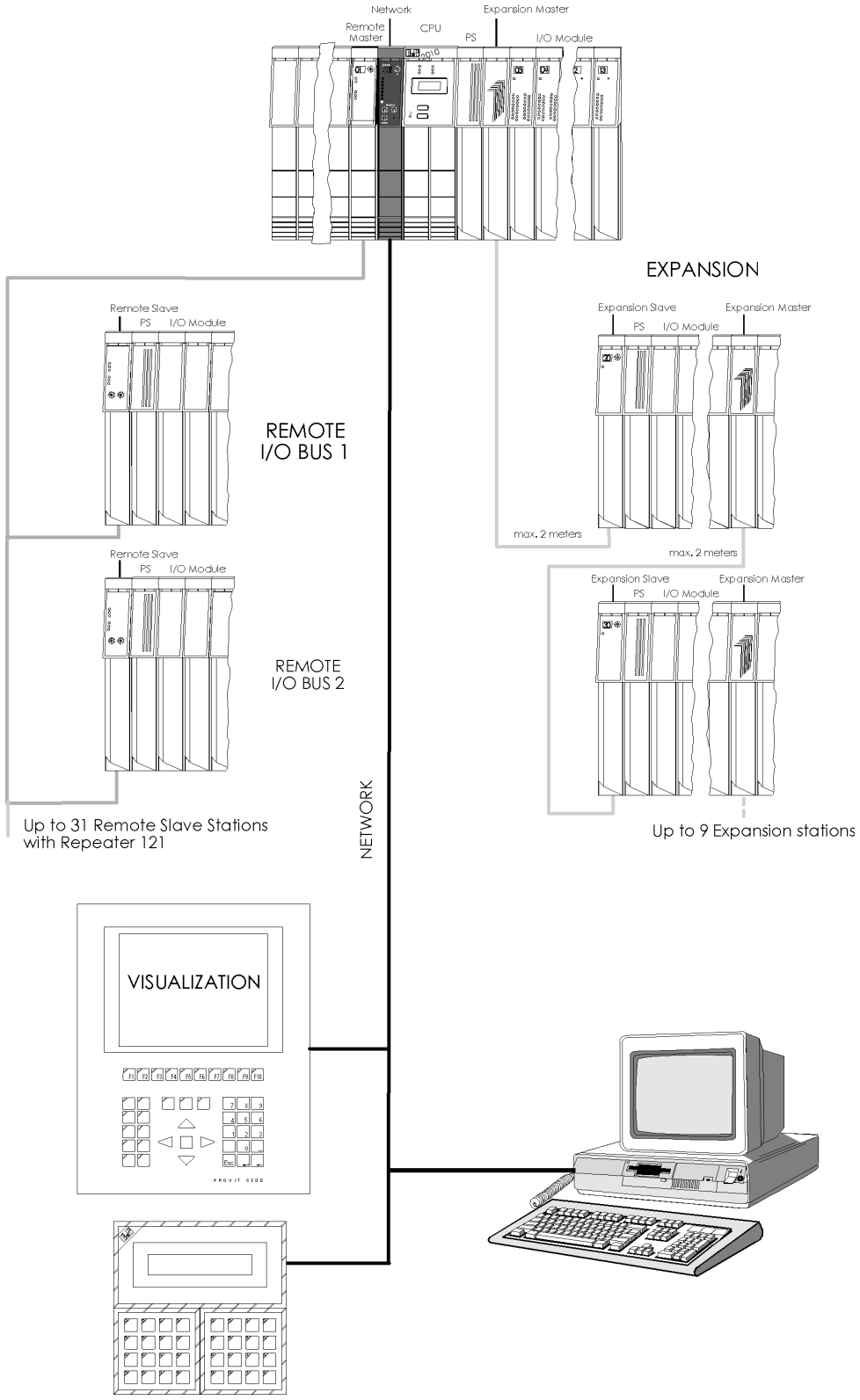
The B&R2010 can be connected to many different standard networks by installing a network module.

Programming is also possible via networks (PROFIBUS, CAN).

B&R2010 Expansion and Remote I/O



B&R2010 Possible Expansions



3.4 Combining B&R2003 - B&R2005 - B&R2010

The PCC systems B&R2003, B&R2005 and B&R2010 can be directly connected.

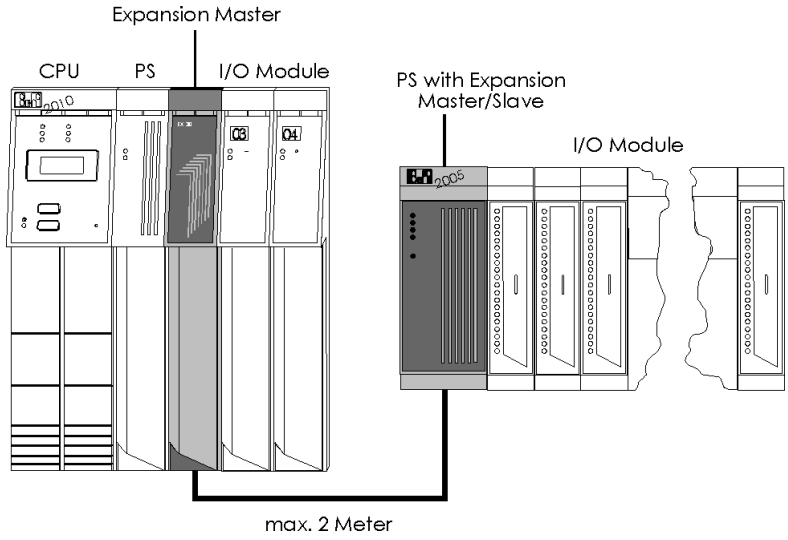
Expansion possibilities	Master	Slave	max. number of Slave stations	Expansion
Expansion	B&R2005 B&R2010 IPC with SOFT PLC	B&R2005 B&R2010	4 (B&R2005) 9 (B&R2010)	2m
Remote I/O	B&R2005 B&R2010 IPC SLOT PLC SOFT PLC	B&R2003 B&R2005 B&R2010	31 / 121	1200 m / 4800 m
CAN I/O	B&R2003 B&R2005 B&R2010 Panelware IPC with SLOT PLC SOFT PLC	B&R2003	63	1000m

Advantage of combining B&R2000-PCCs

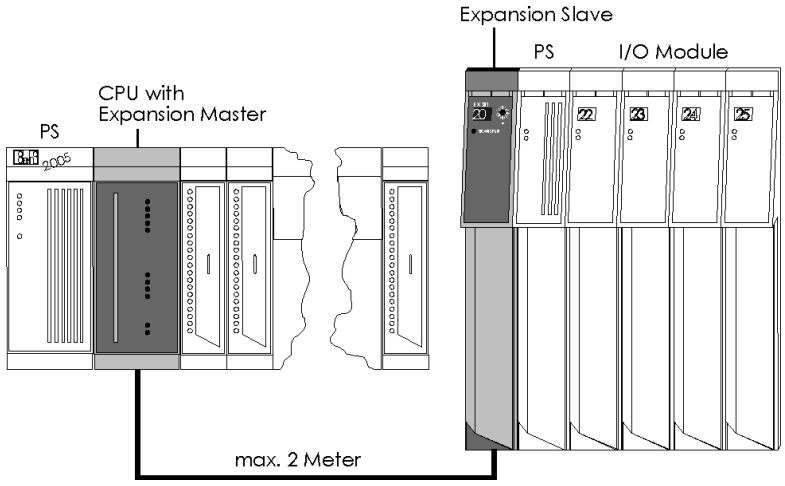
Combining B&R2000 PCCs means that only standard hardware needs to be used on the main unit, bus expansion, remote station and CAN station, which considerably reduces **HW costs.** This means that the **optimum price/performance** relationship can be achieved for every application.

3.5 Expansion

Coupling a B&R2005 and a B&R2010 to a local I/O bus by using expansion modules.

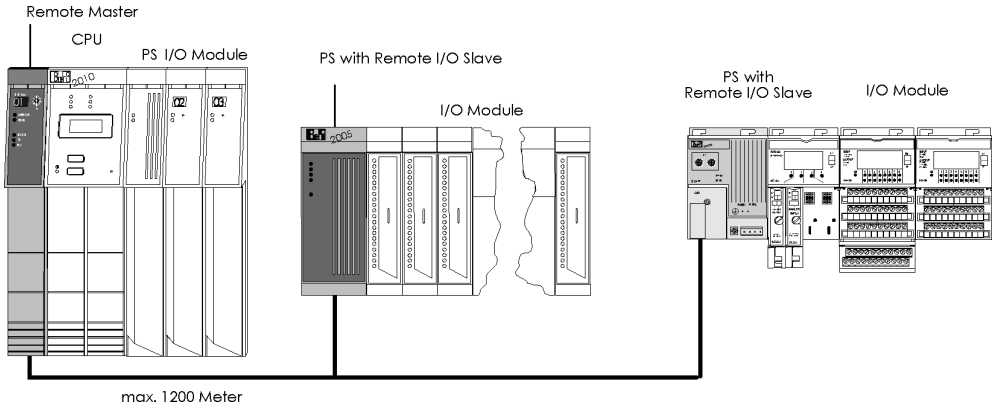


Coupling a B&R2010 and a B&R2005 to a local I/O bus by using expansion modules.

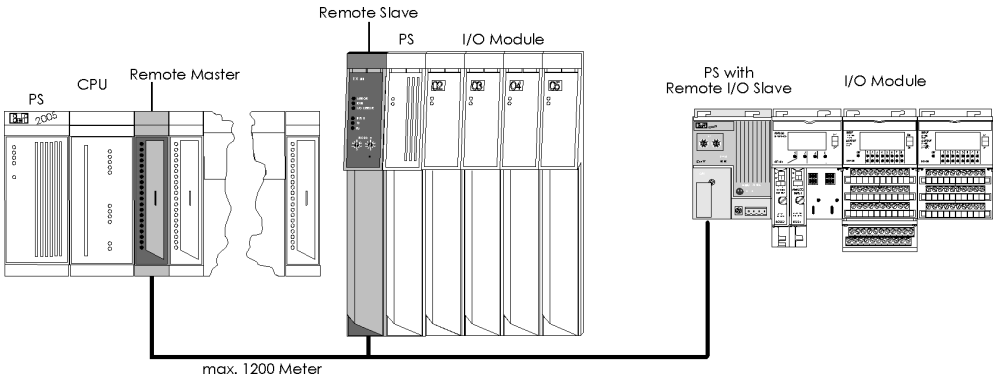


Remote I/O

Coupling a B&R2005 and B&R2003 to a B&R2010 CPU

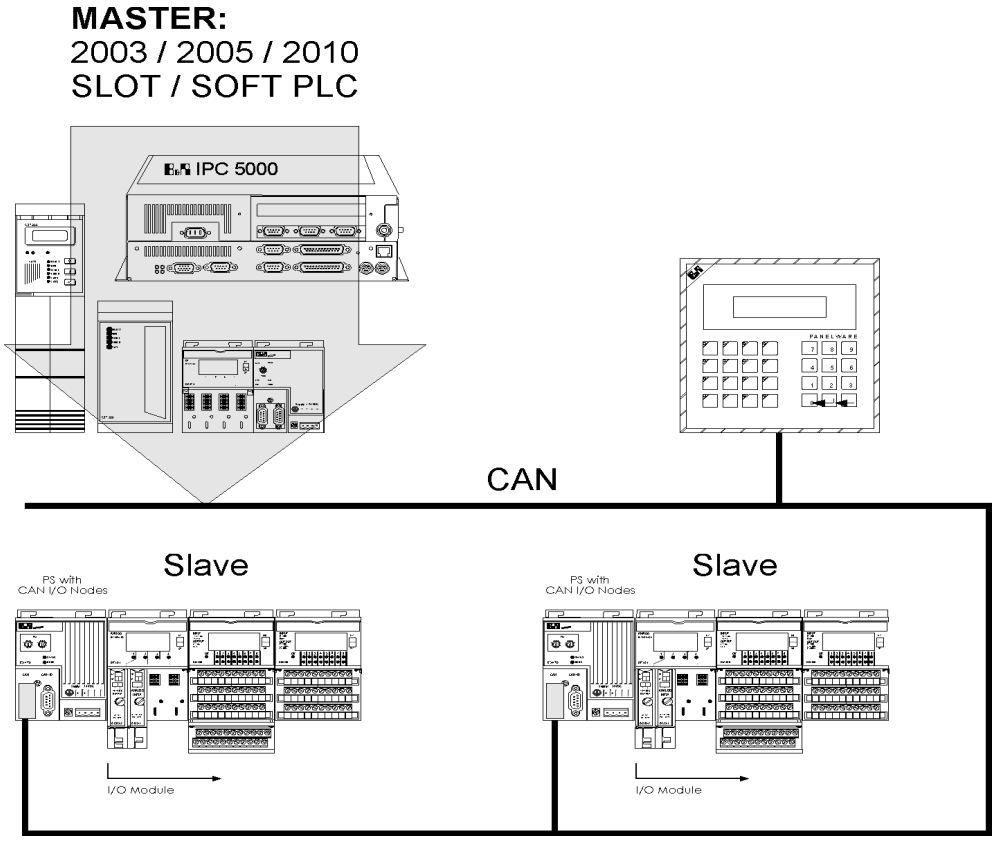


Coupling a B&R2010 and B&R2003 to a B&R2005 CPU

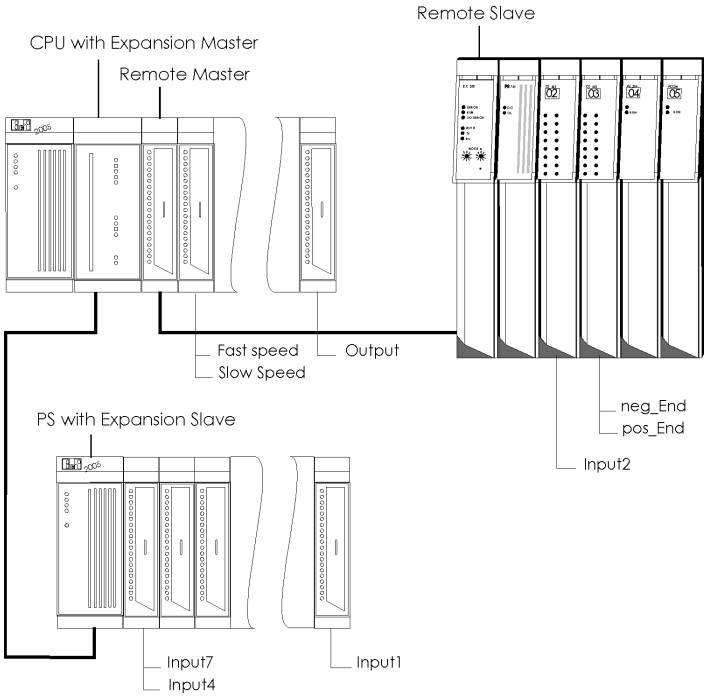


CAN I/O

CAN master stations (B&R2003, B&R2005, B&R2010, SLOT PLC or SOFT PLC) read/write on a B&R2003 CAN I/O station



3.6 I/O Access



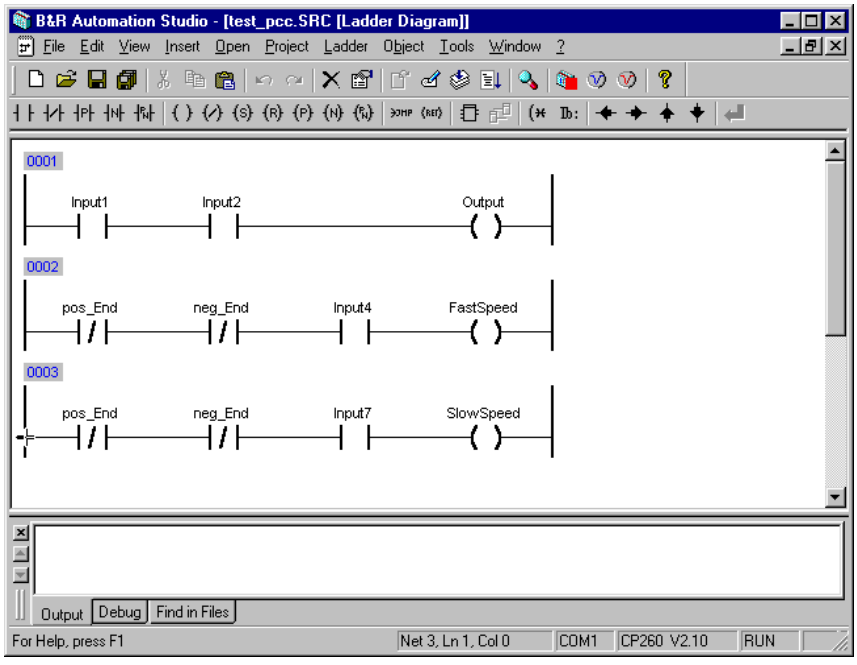
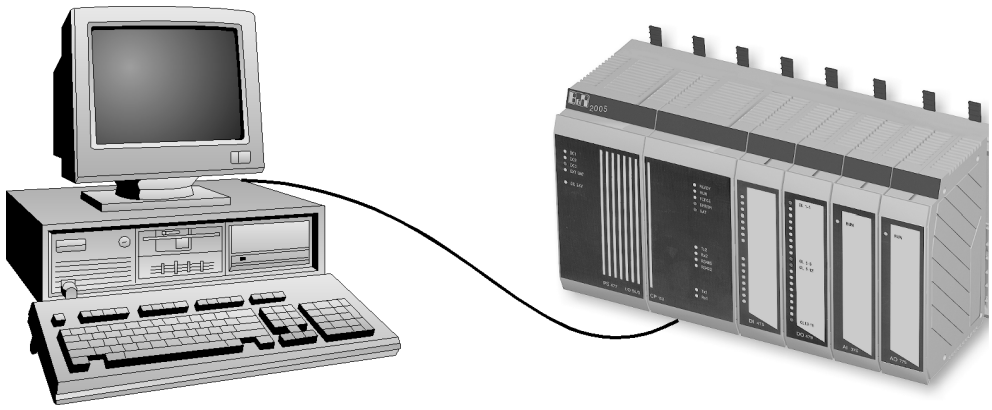


Fig. 4.1: Program in Automation Studio

PROGRAMMING SYSTEM

1 B&R AUTOMATION STUDIO

2 GENERAL INFORMATION



Easy to operate programming software

- Windows Look and Feel
- Mouse and keyboard operation
- Operator prompting
- Context sensitive help system
- Project and system management integrated in the programming system
- Software structuring via hardware

Programming Hardware

- Possible in Win95, Win98 and WinNT 4.0 (starting from SR3)
- No hardware protection (dongle)

Versatile Online-connection

- Serial interface (RS232)
- Modem (for service cost savings)
- Network

Programming according to STANDARD (IEC1131-3)

- Multi language capabilities (LAD, ST, IL, SFC)
- B&R Bonus Pack (ANSI C, B&R Automation Basic)
- Initialization sub-program (INIT-SP) possible for every task
- Function block (Standard libraries)
- Symbolic variable declaration

Simple testing facilities

- Project Monitor
- LAD Monitor
- PV Monitor (Watch)
  - Force - operation
  - Archive Mode
- Tracer (traces variables on the PCC)
- Line Coverage (program flow display)
- Source Level Debugger
- Profiler (DOS)

Versatile Documentation

- Project printing
- ASCII Import and Export

Overview Automation Studio

A PC with a Pentium Processor (or compatible) is required. Minimum requirements are: 16 MB main memory, ~80 MB hard disk (full installation), SVGA resolution (800 x 600). A free serial RS232 (COMx) interface is required for online connection (COMx).

Operation

Programming can be carried out with either a mouse or keyboard. The user can choose between menu bars, Pull-down menus, Pop-up menus and dialog boxes. A user friendly context sensitive help system is available.

3 SETTING UP THE COMPUTER

3.1 Installation

Automation Studio (AS) is provided exclusively on CD format.

If the Setup program does not start automatically, refer to the CD ROM folder.

Start Setup:

Start the set-up program. The rest of the set-up process is menu driven.

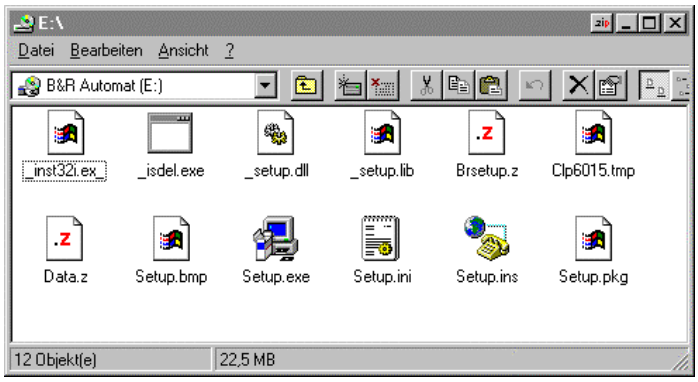


Fig. 5.1: Folder Setup CD

Providing User Information:

It is imperative that the correct information is given otherwise the installation process cannot continue.

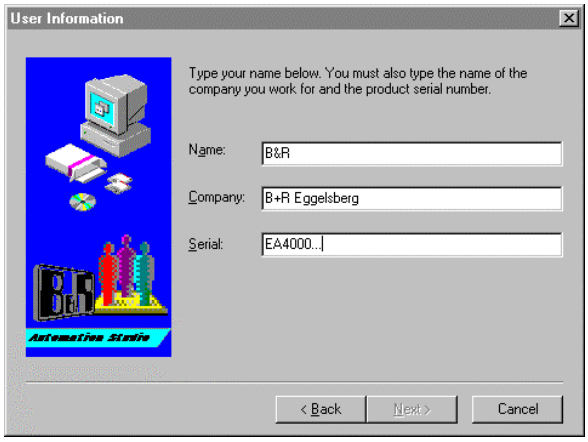


Fig. 5.2: User information

Final settings:

Occasionally extra dialog boxes may appear that have not been described in this section.

When all information has been correctly entered, the installation process is started by clicking on “Next”. This may take a few minutes.

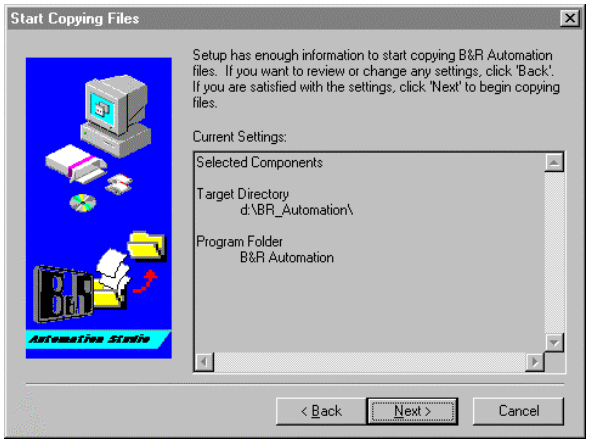


Fig. 5.3: Final dialog box

4 STARTING AUTOMATION STUDIO

During installation a program short cut is added to the start-up menu.

AS (**A**utomation **S**tudio) can now be activated via the start menu.

Fig. 5.4: Calling up the start menu

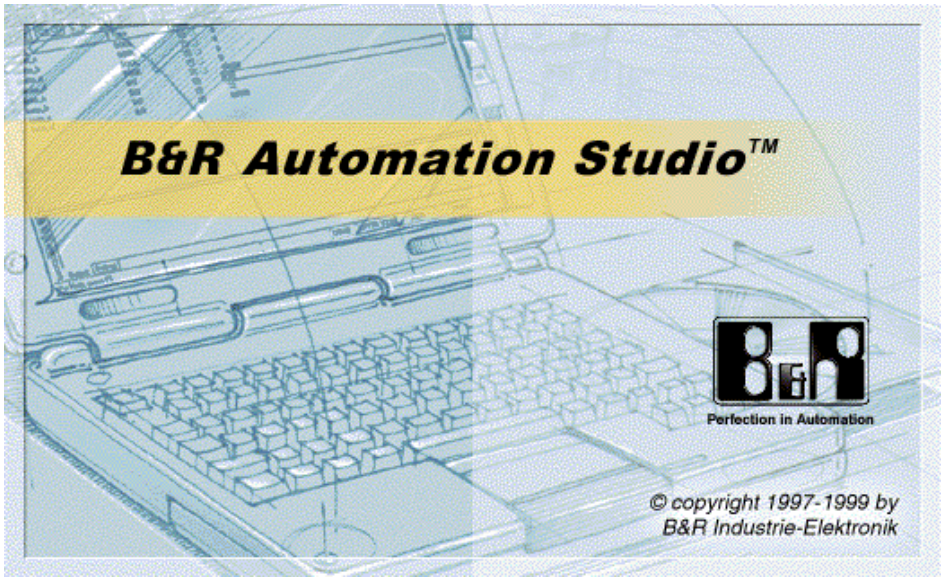


Fig. 5.5: Start up screen

5 OPERATION

The functions and elements of the windows correspond to the Microsoft window user guidelines.

Additional information is provided through context sensitive **online help** which can be found by pressing **<F1>**.

5.1 Screen Layout

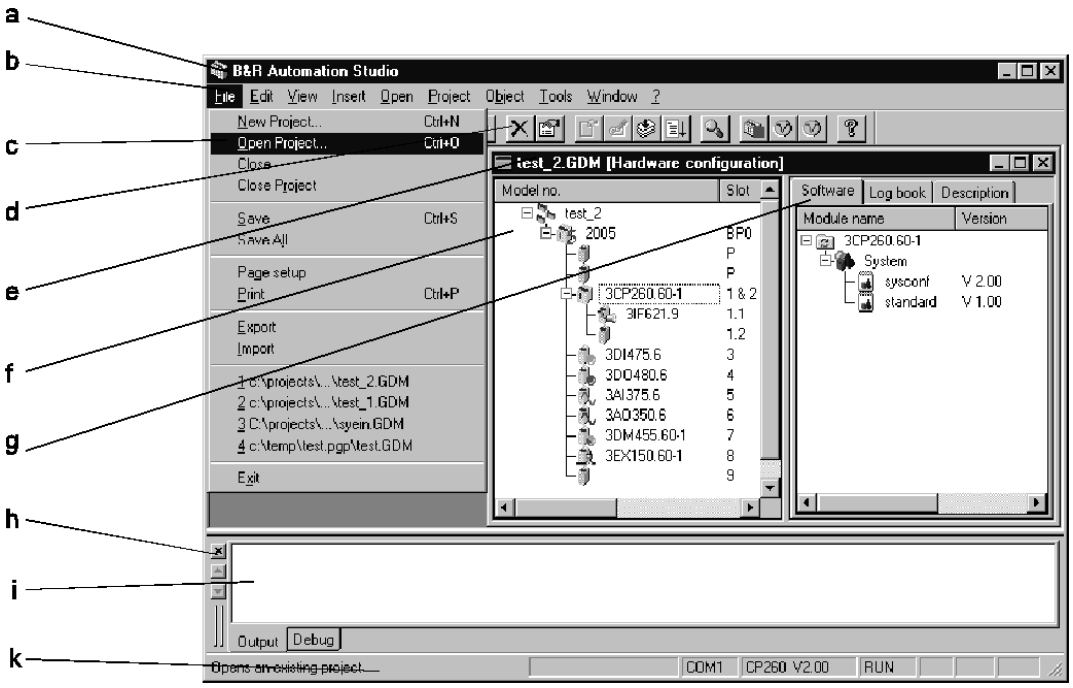


Fig. 5.6: Screen construction

a	.. Title bar	Program Name, Symbol size, Maximize, Close.
b	.. Menu bar	Main menu bar giving access to other menus.
c	.. Pull Down Menu	Different functions of the main menu.
d	.. Function buttons	Enabling quick selection of various commands.
e	.. Application window	Window containing hardware and software configurations.
f	.. Hardware config.	Window for the Hardware configuration.
g	.. Software config.	Window containing the program, system functions...
h	.. Closer	For closing windows.
i	.. Message window	Displays messages concerning Compiler and Debugger.
k	.. Status Line	Displays CPU type, Controller system, Connection status

It is possible open up to **20 windows** simultaneously. This enables the user to edit and monitor programming, read the context sensitive help menus and move text between windows etc, at the same time.

Opening a Window

Double-clicking on a icon will open the chosen Editor.

Cursor Movement

Both cursor keys and the mouse can be used to move the cursor position:

<PgUp>	Page up	**<Home>**	Cursor to start of the line
<PgDown>	Page down	**<End>**	Cursor to end of the line
<Ctrl>+<Home>	Beginning of file		
<Ctrl>+<End>	End of file		

Editor functions

<Ins>	Insert character, Insert Object (depending on Editor)	
<BkSp>	Delete previous character	
<Del>	Delete character, Delete Object	
<Enter>	Start of line: End of line:	Shift current line down Shift next line down
<Shift>+<Cursor>	Mark a block	
<Ctrl>+<C>	Copy a block	
<Ctrl>+<X>	Cut a block	
<Ctrl>+<V>	Insert block	

Switching Windows

It is possible to switch between open windows.

<Ctrl>+<F6> or **<Ctrl>+<Tab>**	To next window
Mouse click	In the desired window
Menu "Window"	Choose from menu listed Windows.
<F6>	from hardware tree to right page (SW tree) from cyclic program section to INIT SP

The chosen window is activated and positioned in the foreground.

Window adjustment

Mouse: Select one of the **edges** of the window with the **left mouse button**, hold button down and adjust it to the **required size**.

Select the title bar of the window with the **left mouse button**, hold button down and **move window to position required**.

Keyboard	**<Alt>+<Space>**	Opens the **pop-up menu** of the main window.
	<Alt>+<->	Opens the pop-up menu of the sub window

Closing Windows

Mouse: Clicking on **Close** (top right corner of the window) will close the application. If there is data remaining unsaved the user will be asked whether it should be saved or not. The window will then be closed.

Keyboard	**<Alt>+<F4>**	Closes the program
	<Ctrl>+<F4>	Closes the current sub window

6 PROJECT ORGANIZATION

A project contains all the information about a system/machine; this means all program modules and data banks including project management information.

Any number of projects can be organized on the computer. Saving or making backups is very easy. Programs or program parts can switched between projects with ease.

Project organization menu

For creating or selecting projects.

File: New Project

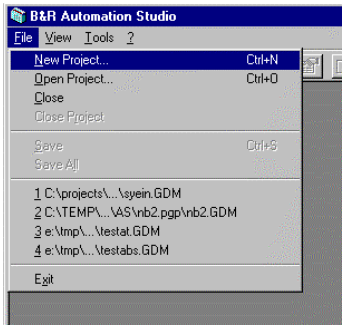


Fig. 5.7: File: New Project

File: Close Project:

Closes an open project. “New Project” opens a new project.

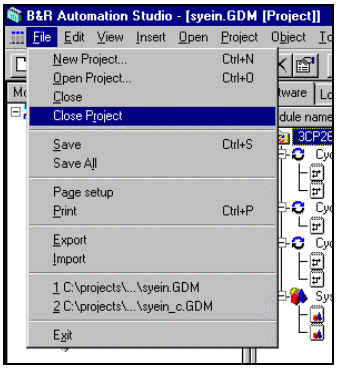


Fig. 5.8: File: Close Project

7 PROGRAM CONSTRUCTION

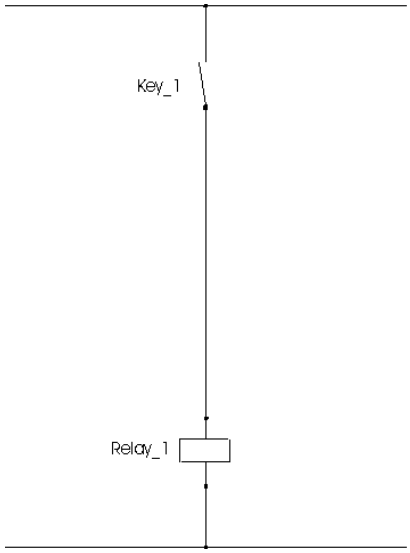
Constructing a program which runs successfully is made easy by the organization of the programming software.

On the following pages we will show you how to create a ladder diagram.

The following points regarding programming in B&R Automation Studio will be covered:

- Creating a new project
- Linking and assigning of the used variables
- Programming using Ladder Diagrams
- Downloading the program to the controller
- Testing the program with the LAD monitor
- Testing the program with the process variable monitor

Example: Converting a circuit diagram into a LAD.



Try to solve the problem above with the help of the information on the following pages of the training manual!

Project name: **proj_001**
LAD name: **l_logic1**

In all examples, variables will be shown as follows.

Name	Type	Scope	Attribute	Value	Remark
Key_1	BOOL	global	IP5.0.3.1	----------	digital IN channel 1
Relay_1	BOOL	global	QP5.0.4.1	* remanent	digital OUT channel 1

INFO:

digital IN = digital INPUT
digital OUT = digital OUTPUT

7.1 Explanation of the programming method:

INFO: During this section the following notations for the control explanation will be used.

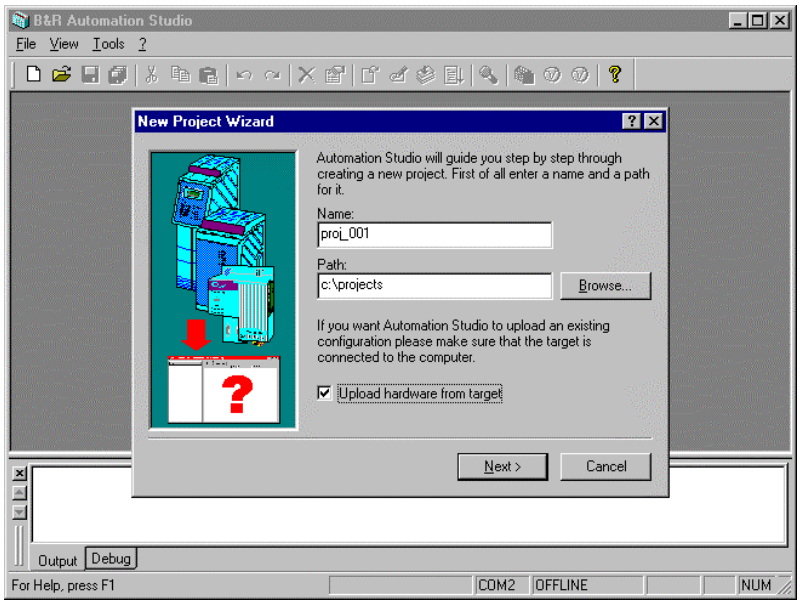


Fig.: 5.9: Explanation of the programming method

Toolbar Symbol and the corresponding menu:

File: New Project

Function Key: **<F1>** + Description

Other keys: <Enter>,
<Cursor>, ..

Confirm in dialog box: **[Next], [OK],** ...

Text for input: "l_logic1"

Example: Call up menu **File: New Project**

Menu call up with the keyboard
<Alt> + F for File
then
n or **N** for New Project

7.2 Creating a new project:

Automation Studio is a **hardware orientated Programming system**. The objects ease organization of the project (system, machine). All objects are then allocated to the project through the structured organization support.

Tools: Options:

Before we begin the programming, the connection should be checked. The online interface COM1 or COM2 must be correctly fitted to ensure hardware configuration from the controller can be down loaded.

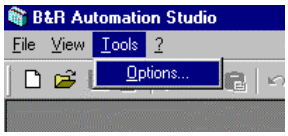


Fig. 5.10: Tools: Options

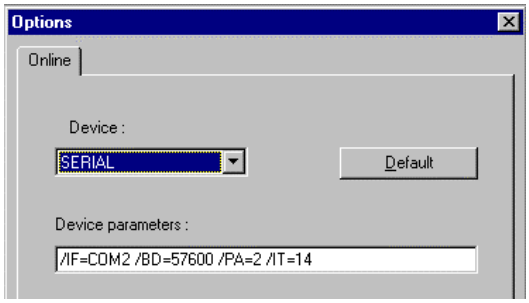


Fig. 5.11: Parameter string

In this Dialog box only the parameter **/IF=COM2** is to be set to the corresponding COM.

File: New Project:

Create the new training project. The “New Project Wizard” will help you along the way.

Project name: **proj_001**
Path: **c:\projects**

Upload from Target

When creating a new project the project data must be entered, as shown in the following diagram Activate the option "**Upload hardware from target**"!!

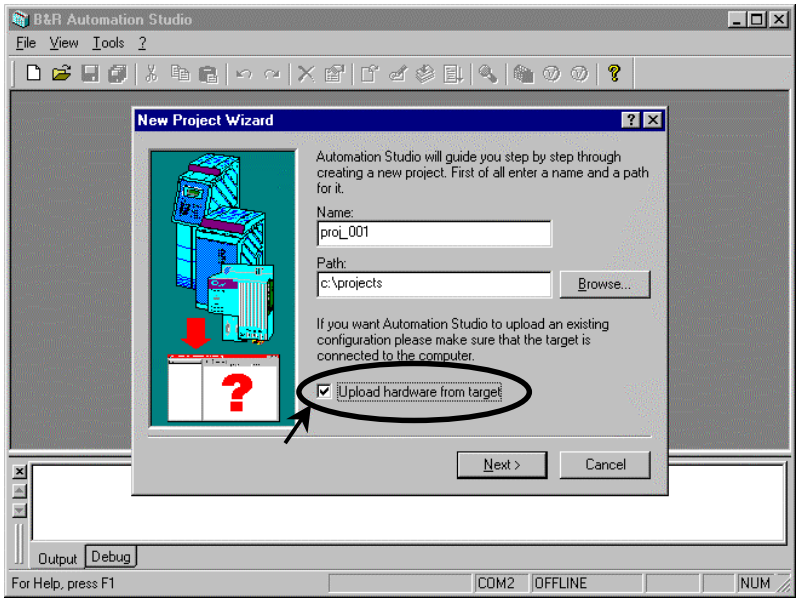


Fig. 5.12: New Project Wizard

[Next]: Settings shown are confirmed with [Next] until the last dialog is reached.

[Finish]: This button closes the creation of the new pro

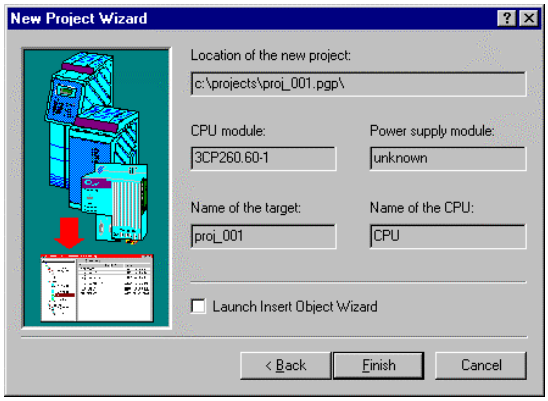


Fig. 5.13: New Project Wizard (Finish)

Automation Desktop

Figure 5.14 shows the Automation Desktop (controlling system of AS).

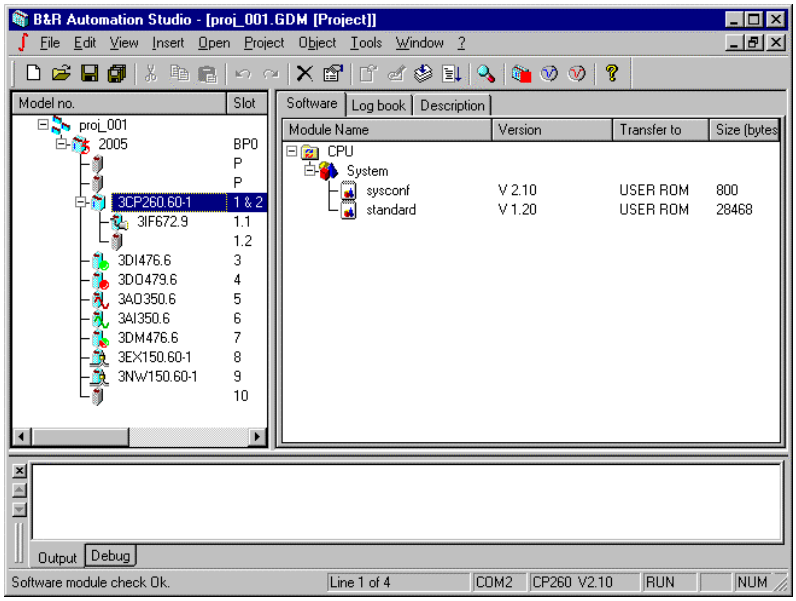


Fig. 5.14: Automation Desktop

Hardware Configuration:

On the left (Hardware Configuration) individual modules can be selected and inserted. The right side adjusts itself automatically.

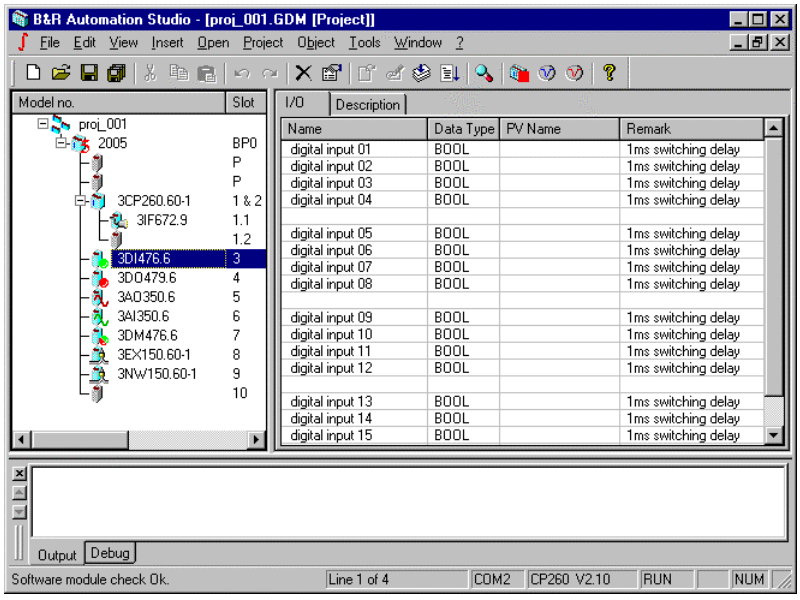


Fig. 5.15: Hardware Configuration

Description:

The button “description” gives hardware information concerning individual modules.

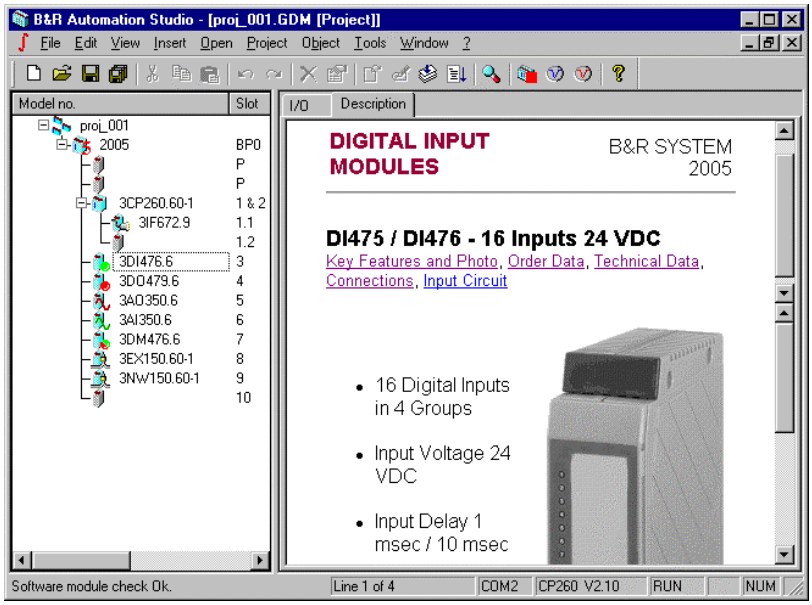


Fig. 5.16: Hardware Description

Hardware Manual Inclusive:

This window contains all information about the individual modules. Order numbers and descriptions are included which are otherwise only available from the hardware manual.

Variable Declaration Digital Input:

Enter the variable names in the column “PV Name” on the right side. The variable names can be up to 32 characters long. Longer text can entered in the column “Remark”. Long texts are dealt with exclusively in the Documentation and are not downloaded in to the control.

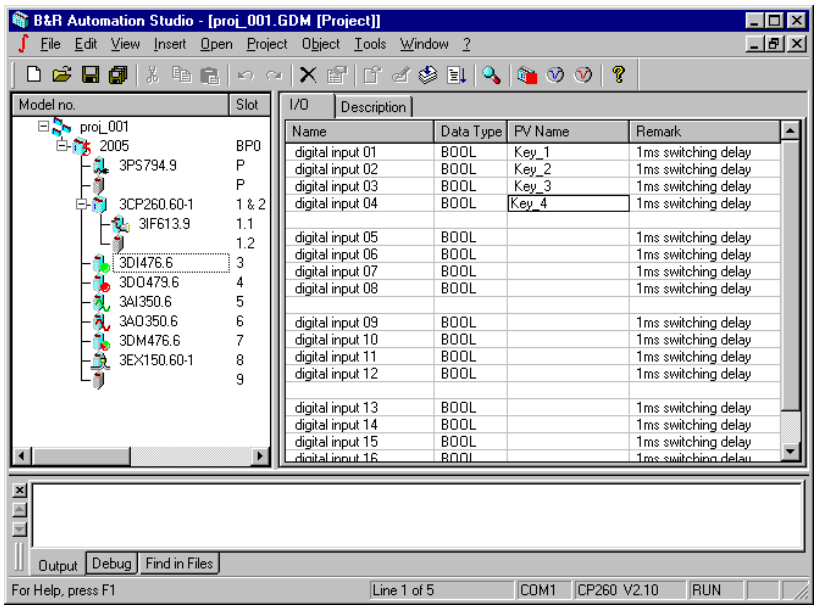


Fig. 5.17: Variable declaration Digital Input

Variable Declaration Digital Output:

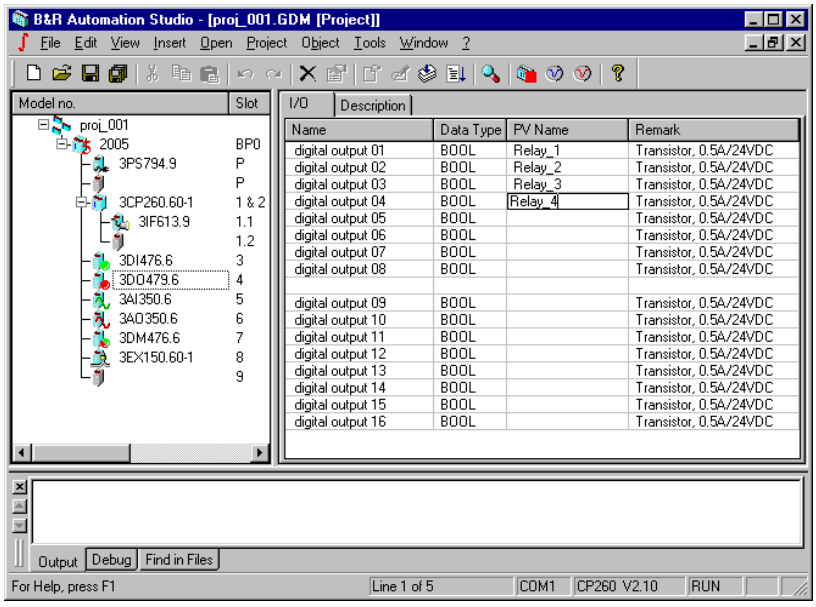


Fig. 5.18: Variable declaration Digital Output

CPU Icon:

Change to the left side and select the CPU icon.

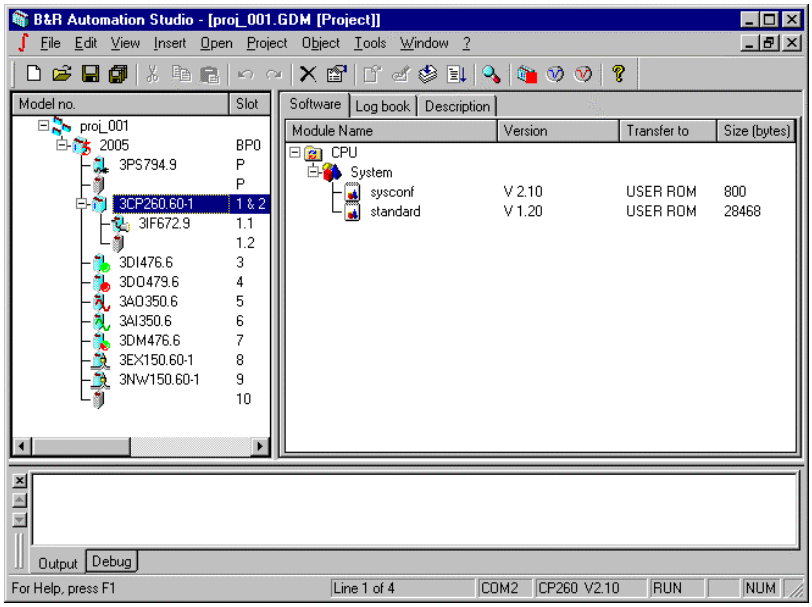


Fig. 5.19: CPU Icon

Insert: New Object:

Click on the CPU Icon in the **software configuration** with the **right mouse button** and select "Insert Object".

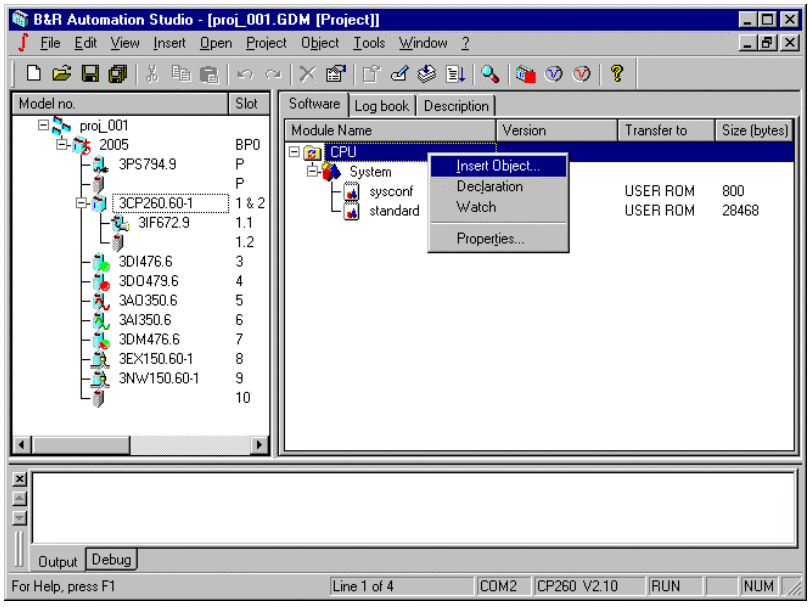


Fig. 5.20: Insert Object

Cyclic Object:

Select Cyclic Object and confirm by selecting "Continue".

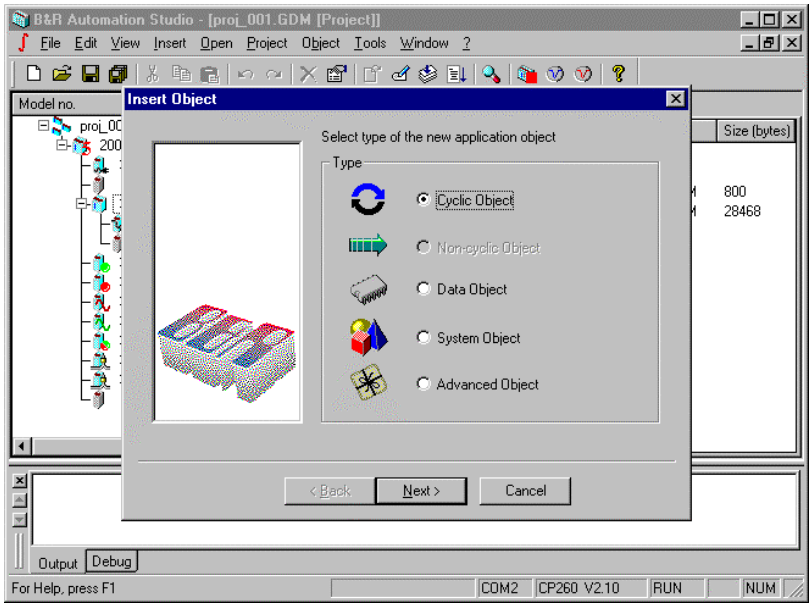


Fig. 5.21: Cyclic Object

L_logic1:

The name of the task is "l_logic1". The task type should remain in default settings.

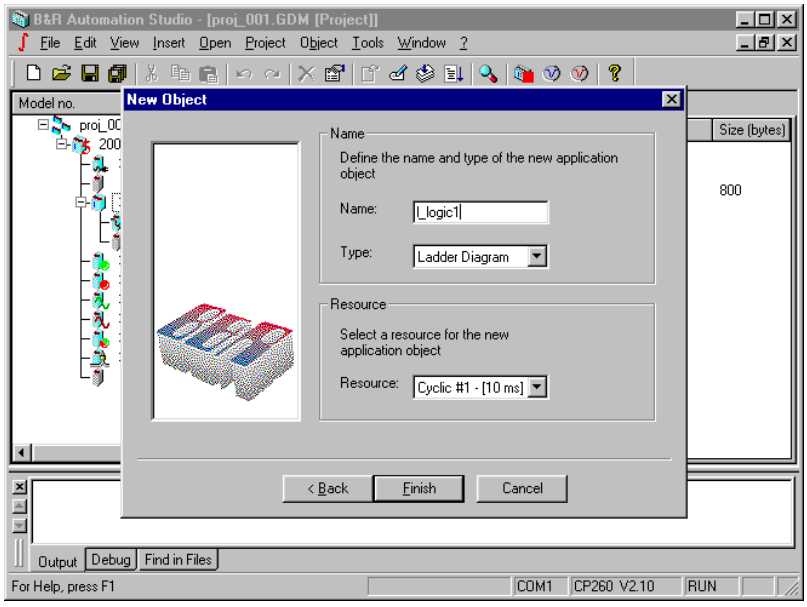


Fig. 5.22: "l_logic1" Task

"c": The "c" key or button: will insert "Key_1". **<Alt>+<Cursor>** will add connections (..).

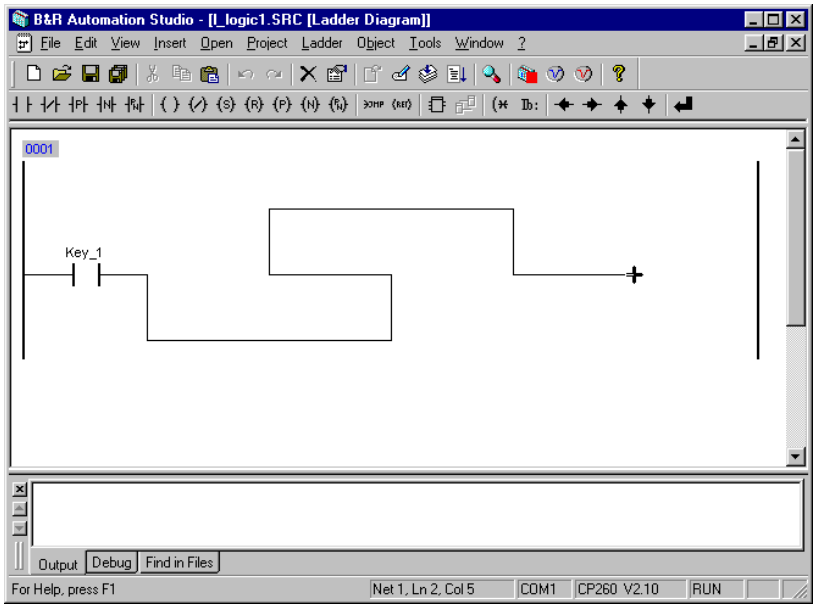


Fig. 5.23: LAD Editor

"C": Capital "C" or the button creates an output contact. **<Space>** opens a dialog box that shows **all the appropriate values**.

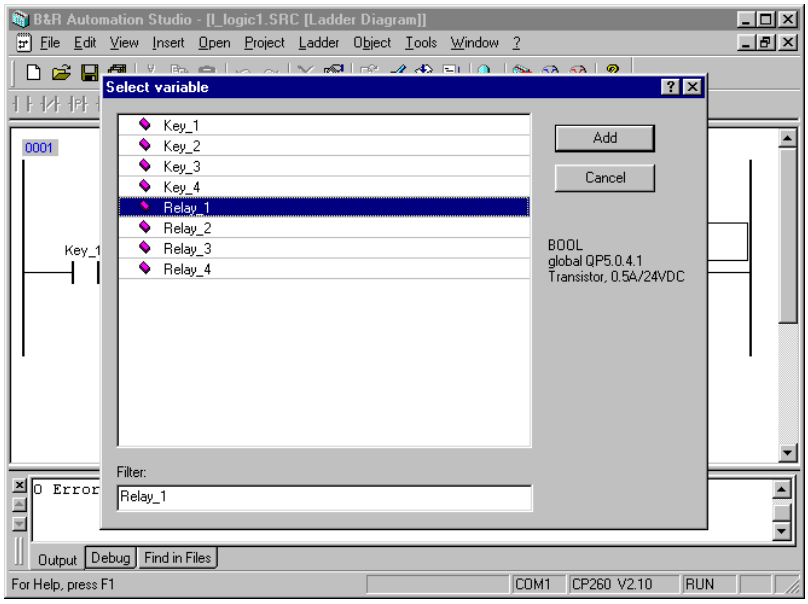


Fig. 5.24: Ladder Diagram Variable selection

<Enter>: Select "Relay_1" and also locks LAD with <Enter>.

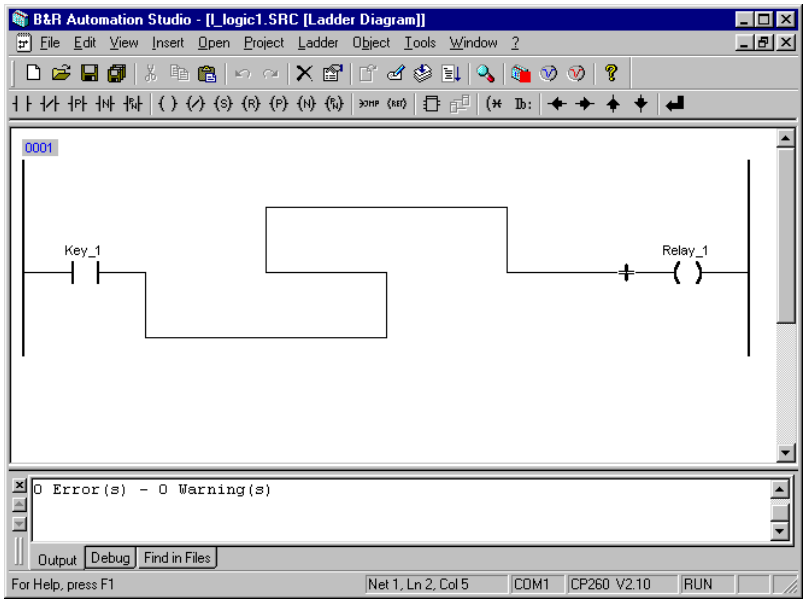


Fig. 5.25: LAD with Output

This step checks the LAD and optimizes it.

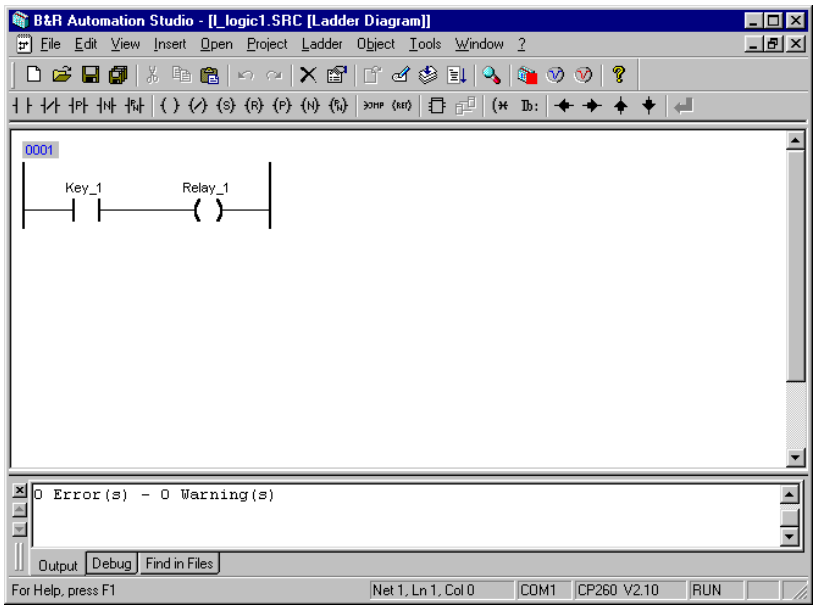


Fig. 5.26: Ladder Diagram Network overview

<Right Mouse Button>: Various display options can be selected with the right mouse button.

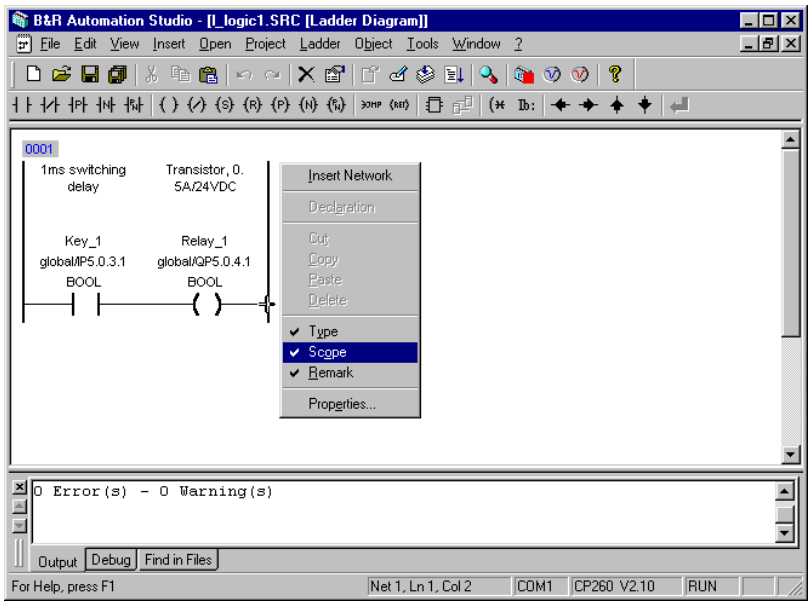


Fig. 5.27: Ladder Diagram display possibilities

Project: Transfer To Target:

Mit Project: Project: Transfer compiles all tasks and downloads them to the controller.

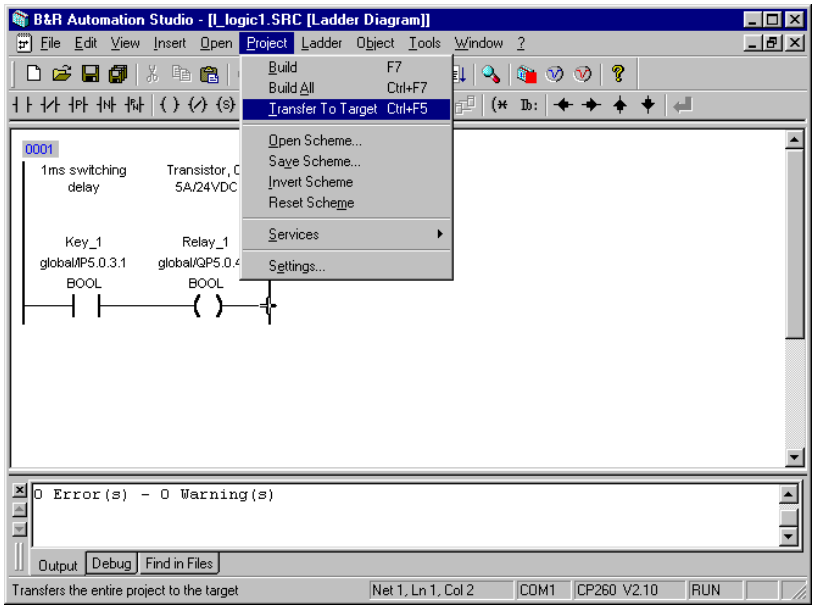
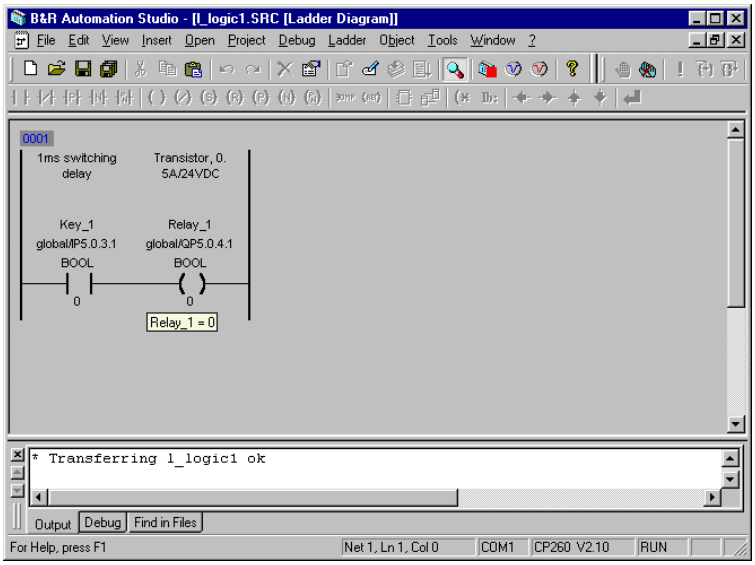
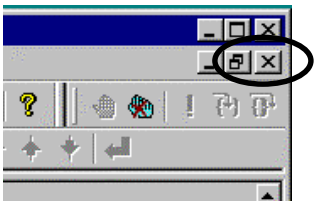


Fig. 5.28: Ladder diagram Download

View: Monitor:

Online-mode can be activated via the View menu : Monitor or by pressing the button. In Online-mode editing is **not** possible. Different monitors are active depending on the window opened.





Close LAD monitor

Fig. 5.29: LAD Monitor

7.3 System Monitor:

The PG remains in Online-mode even if the LAD-monitor is closed. The System-Monitor is activated.

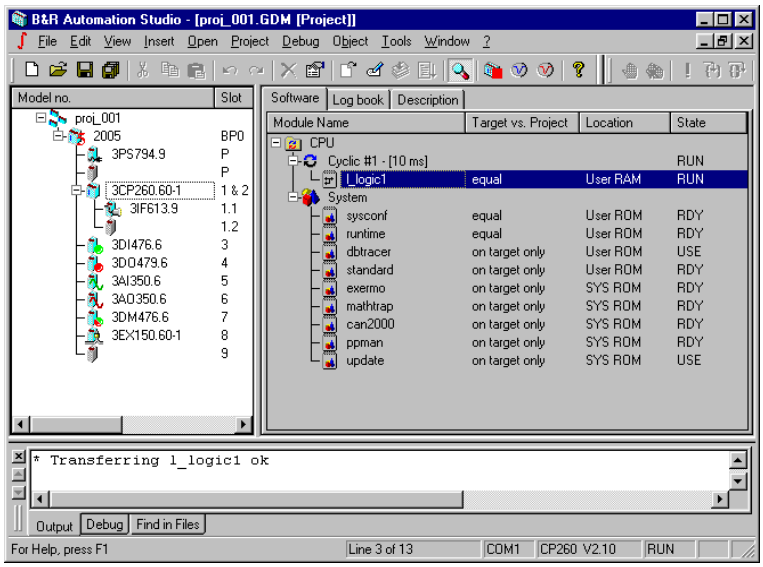
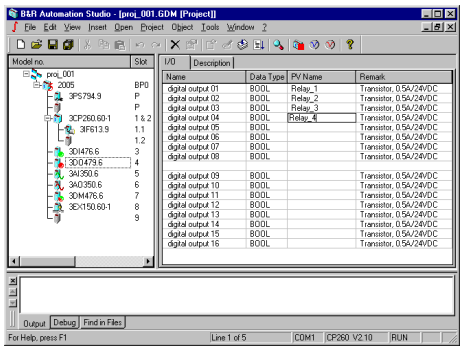
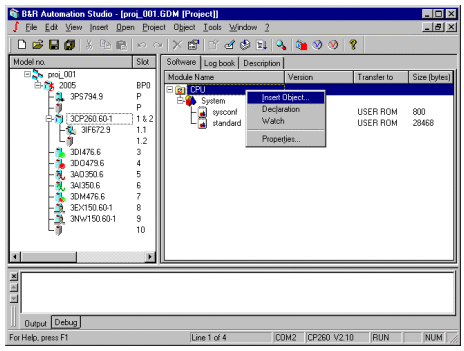


Fig. 5.30: System Monitor

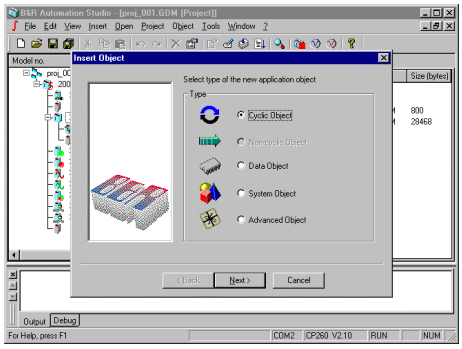
7.4 Summary Creating a Project



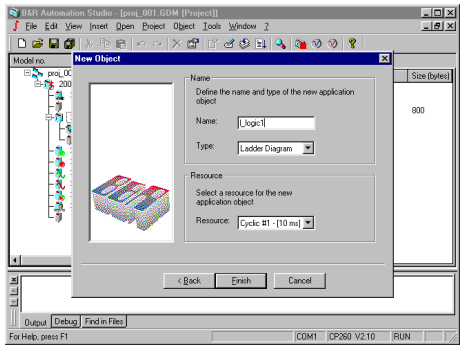
I/O - Assignment ->



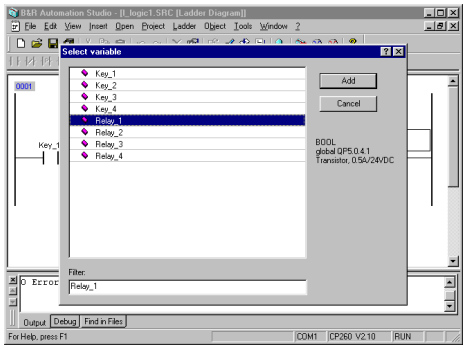
Insert Object



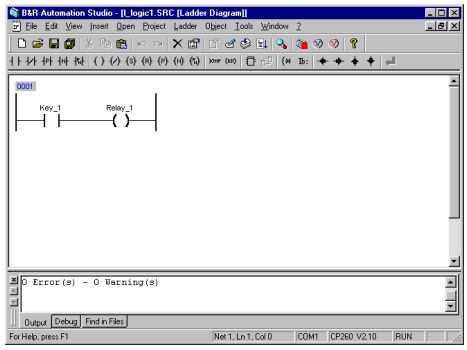
Cyclic Object ->



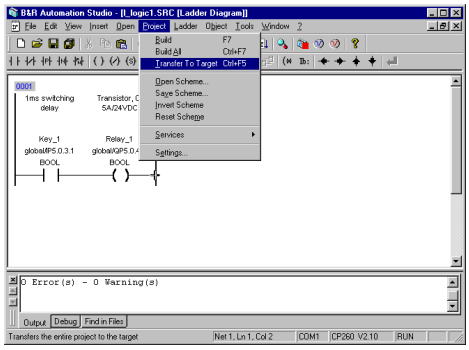
Select LAD



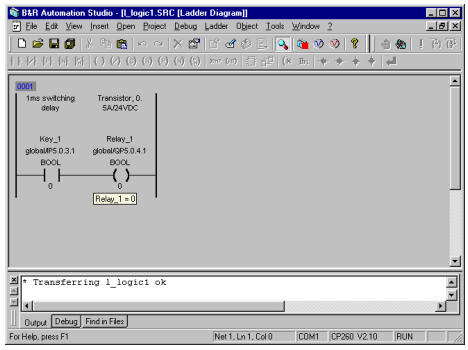
Select Variables ->



Network Overview

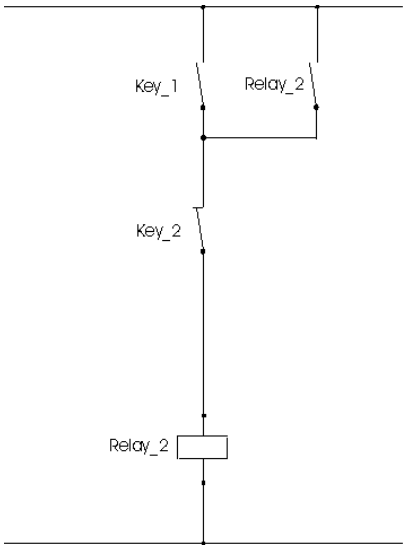


Transfer Project ->



LAD Monitor

Example: Convert the following circuit diagram of a latch into a Ladder diagram.



Name	Type	Scope	Attribute	Value	Remark
Key_1	BOOL	global	IP5.0.3.1	---	digital IN channel 1
Key_2	BOOL	global	IP5.0.3.2	---	digital IN channel 2
Relay_2	BOOL	global	QP5.0.4.2	* remanent	digital OUT channel 2

Project name: proj_001
LAD plan name: l_logic2

The program can be written using the same guidelines as outlined in the previous example. Test this task with a LAD monitor.

7.5 Defining Networks

IEC standards stipulate that links should be displayed as networks. Check the networks and their division into 2 networks using the  **<Enter>** key, or alternatively using the menu **Ladder: Verify Network**

Example: Two separate Networks

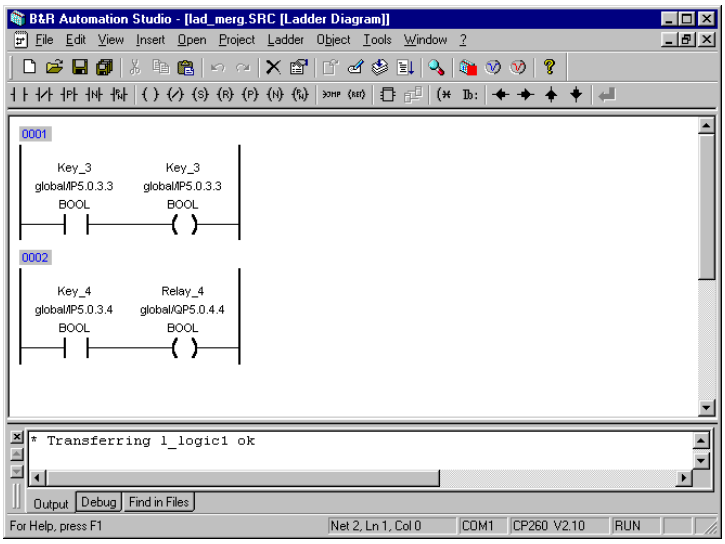


Fig. 5.31 Two Networks

If a line in a network is completed with **<shift><Enter>**, the network is expanded. This means that no new network is created.

Example: A two line Network

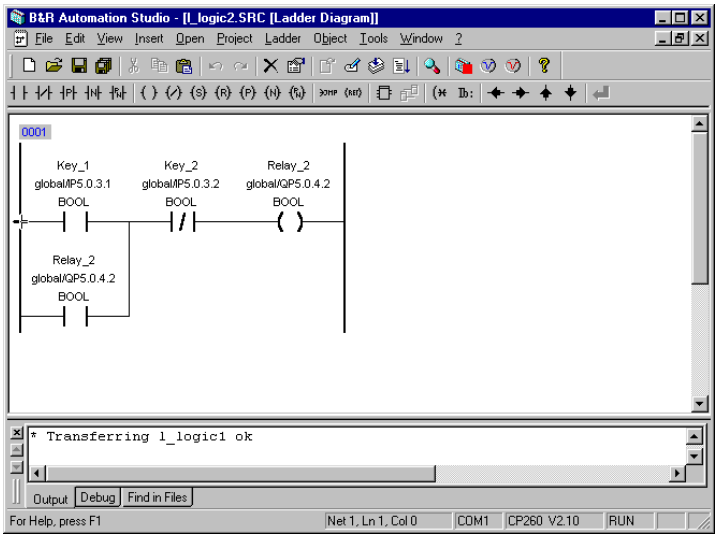


Fig. 5.32: Two Line Network

7.5.1 Linking Two Networks

Ladder: Merge Networks

Two networks can be connected using the menu option Ladder: Merge Networks. The key combination is **<Strg><Enter>.** The top network is always connected to the bottom network.

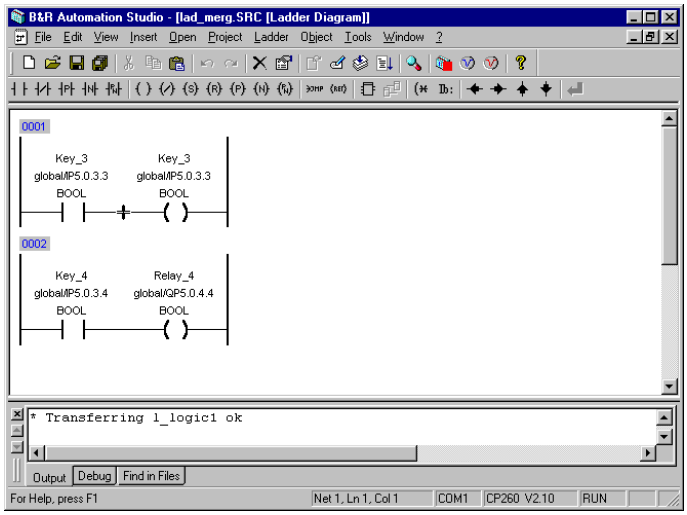


Fig. 5.33: Connection, Net 0001 with Net 0002

<Strg><Enter>: Merge Networks

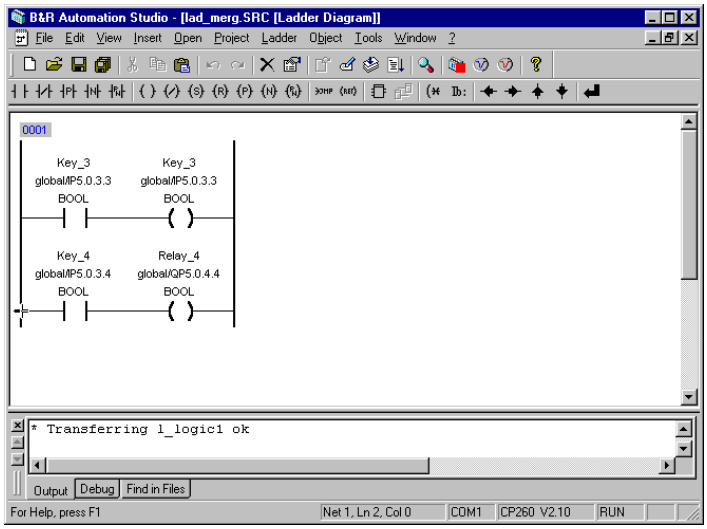


Fig. 5.34 : Connected Networks

If the networks are not connected and acknowledged with Verify, they are separated from the Editor.

7.6 Summary Program Toolbar, LADToolbar

Program Toolbar:

New Project	Open Project	Save File	Save All
Cut	Copy	Paste	Undo
Redo	Delete	Property	Open File
Declaration	Build	Transfer	Monitor
Stop PCC (Service)	Coldstart (blue)	Warmstart (red)	Help

LAD Toolbar:

Normally Open **<c>**	Normally Closed **<i>**	Pos Trans Pos. Edge **<p>**	Neg Trans Neg. Edge **<n>**
Trans Both Edges **<b>**	Coil Output **<shift><C>**	Negated Coil Negated Output **<shift><I>**	Set Coil Set **<shift><S>**
Reset Coil Reset **<shift><R>**	Pos. Trans. Coil Pos. Edge output. **<shift><P>**	Neg. Trans Coil Neg. Edge output. **<shift><N>**	Trans. Coil Both edge output. **<shift><B>**
Jump **<j>**	Return **<e>**	Function Block **<f>**	Analog Value **<space>**
Description **<d>**	Label **<l>**	Line Left **<Alt><CuLeft>**	Line Right **<Alt><CuRight>**
Line Up **<Alt><CuUp>**	Line Down **<Alt><CuDown>**		Verify **<Enter>**

B&R 2000

1 AUTOMATION STUDIO CONSTRUCTION

General Information.

Automated tasks are divided into separate objects for added clarity. These objects are project dependant.

- Different tasks on a processor that must be worked on simultaneously

The **Project Editor** is a high performance tool enabling straightforward analysis and lay-out management of the project.

Tasks concerning the **Hardware configuration** (left side) act to clarify the division of the Hardware.

The right side automatically relates to the left.

Specific properties of the modules are always available on the right.

A specific property of the CPU is the software.

1.1 Creating a Project

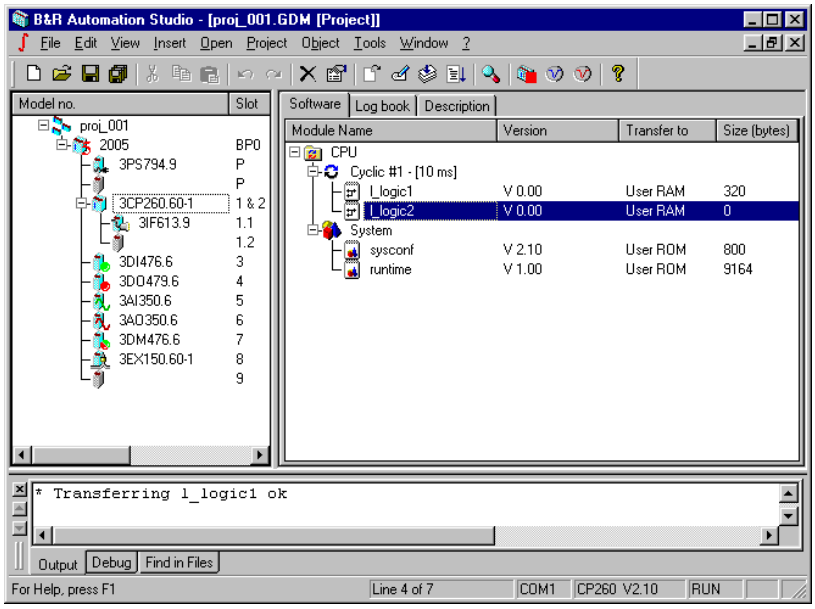


Fig. 6.1: Creating a Project

1.2 Software Configuration:

Regarding the software – all tasks (that are running on a processor), data modules, and system modules are treated a objects.

E.g. Cyclic Object:

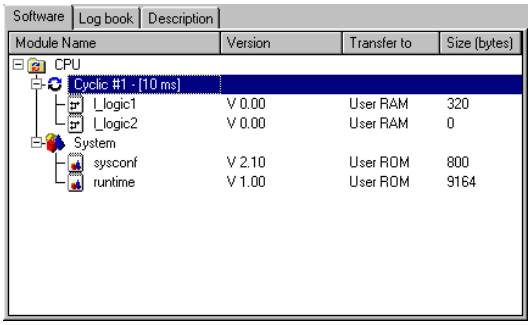


Fig. 6.2: Software Configuration

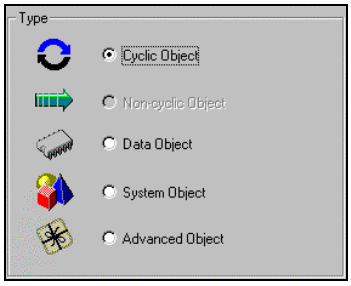


Fig. 6.3: Dialog SW configuration

1.2.1 Possible types of object are:

Cyclic Object

Written by the user in various programming languages.

Ladder Diagram	...	LAD
Instruction List	...	IL
Structured Text	...	ST
ANSI-C	...	High level language

Non Cyclic Object

System tasks: For special system solutions (default: inaccessible).

Data Object

Fulfills the function of a table with the option of writing the data object to the running time.

System Object

Object available for use from Libraries (net2000, CAN, profibus, plus more).

Advanced Object

e.g. NC-Axles, CAM disk ... (for future developments).

Exception Tasks

Are grouped as **Cyclic tasks**. The Exception task is represented in the **Parameter Resource**.

2 MULTITASKING ON THE PCC

2.1 Multitasking

Task A Task is an **independent part** of the program representing many processes. (Digital and analog connections, rules, positioning, measurements.....) e.g. "l_logic1"

Multitasking

During multi-tasking **the processor deals with one or more tasks.** These tasks are worked on simultaneously.

This division of the task is of great advantage and helps us:

- Structure the application
- Set the task in the best suitable programming language
- Program individual functions and test them
- Maintain the individual tasks with ease

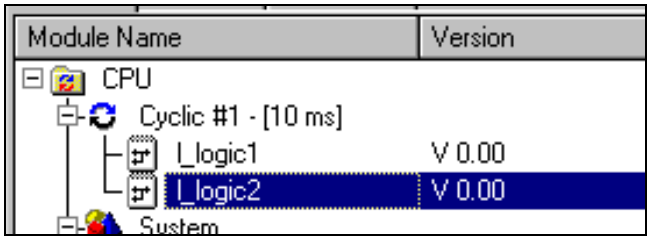
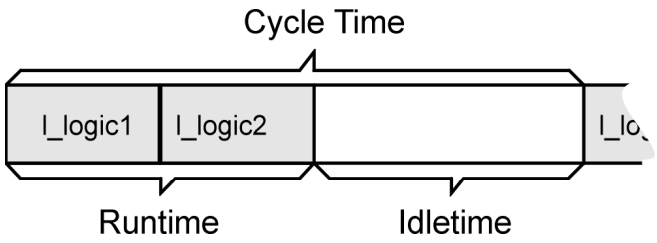


Fig. 6.4: Section of the Software Tree

For PCC use, a multitasking system must posses the following qualities:

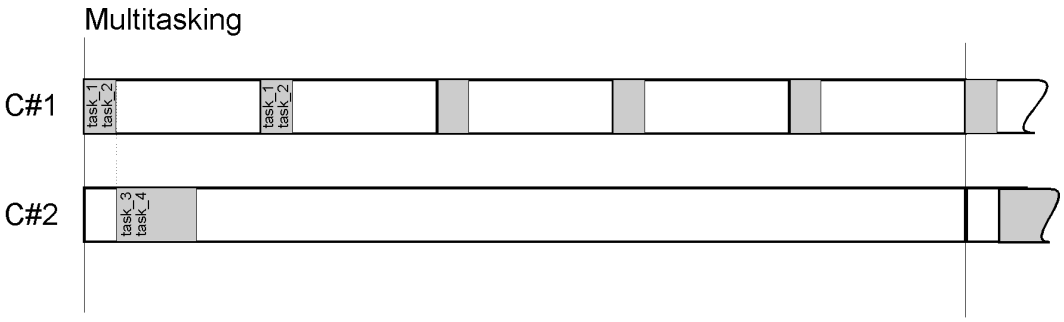
- Parallel processing of multiple Tasks
- Deterministic Multitasking (with accurate time predictions).
- Different and adjustable cycle times with monitors
- Identical (consistent) I/O-image for every Task class



Cycle time is predetermined and set by the user.

The Cycle time does not tell us anything about the **running time** of the task.

Rest time refers to the CPU-time within the system that is not being used. This is known as an "IDLE" state.



The task scheduler activates the required tasks at the beginning of a task class and also sets up the I/O window.

At each cycle starts the following checks are made:

- Whether the Task is finished within the set time
- Whether the new input window is fully available
- Whether the output window has been sent

2.2 Task Classes

A Task class comprises of all PCC tasks with the same cycle times.

The user can choose between:

- **Timer Task** (High Speed Task)
  Designed for very fast, time critical applications. Timer-tasks are activated by the **Hardware timer,** can be called up in very short intervals (from 1msec) and monitored.
- **Normal Task**
  A cyclically running program. The user provided cycle time is monitored by the system manager.

System B&R2003 / System B&R 2005

Resource	Cycle Time	Tolerance
Timer#1	3 msec	/
Cyclic#1	10 msec	20 msec
Cyclic #2	50 msec	50 msec
Cyclic #3	100 msec	100 msec
Cyclic #4	10 msec	30.000msec

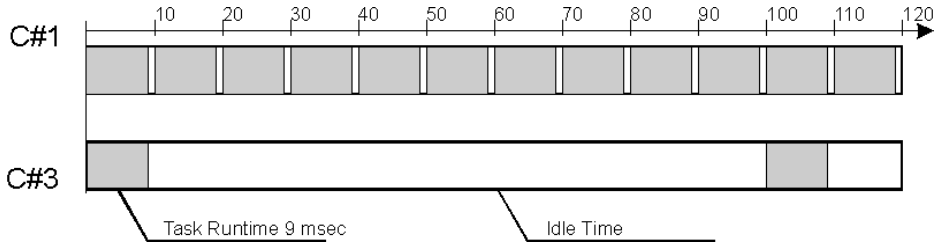
System B&R2010

Resource	Cycle Time	Tolerance
Timer#1	3 msec	/
Timer #2	5 msec	/
Timer #3	7 msec	/
Timer #4	9 msec	/
Cyclic#1	10 msec	20 msec
Cyclic #2	50 msec	50 msec
Cyclic #3	100 msec	100 msec
Cyclic #4	10 msec	30.000 msec

The values in the table are default values. These values can be changed in the "properties" dialog of the CPU..

All tasks in Cyclic#4 are completed on the principle "as fast as possible". The task start is always a multiple of 10msec. **The tolerance is added to the Task class cycle time.** If this time is breached the system manager generates a **maximum cycle time** injury**.**

Example 1: Comparison of Cyclic#1 and Cyclic#3 with a task of 9msec running time. The task will run through Cyclic#1 once and Cyclic#3 once.

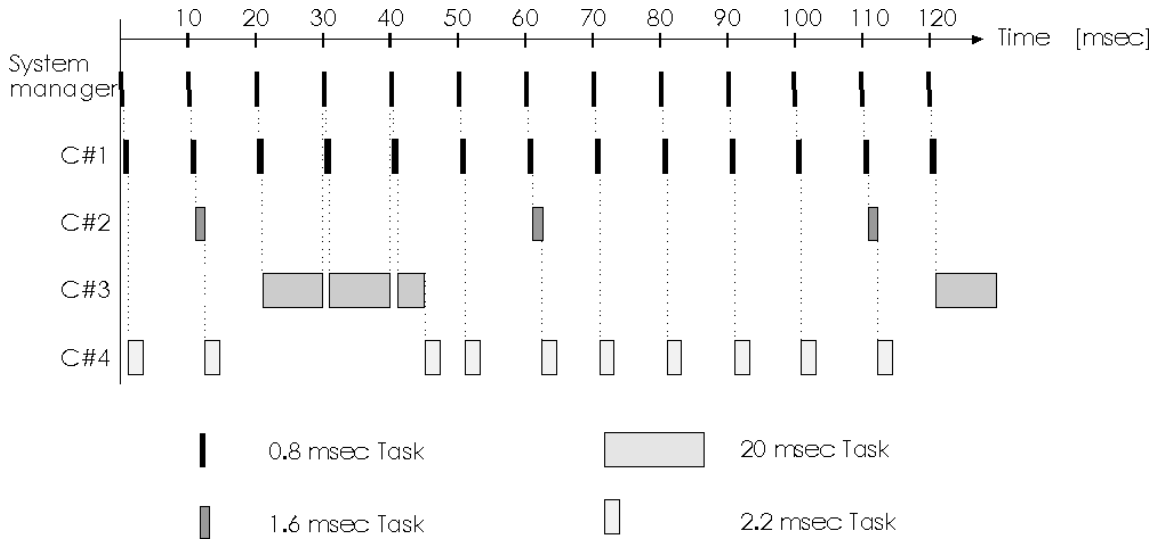


From the diagram the following points can easily be made:

Resource	Running time total in 100 msec	Idle time per run.	System utilization in %
Cyclic#1	10	1 msec	90 %
Cyclic#3	1	91 msec	9 %

Example 2: Tasks run in different task classes in a system:

Runtime	Resource	Task Class Cycle Time
0.8 msec	Cyclic#1	10 msec
1.6 msec	Cyclic #2	50 msec
20.0 msec	Cyclic #3	100 msec
2.2 msec	Cyclic #4	As fast as possible, when the system is free

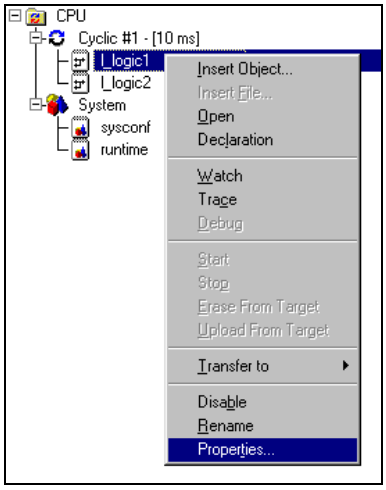


The above diagram shows more important points:

- Cyclic#1 has the highest priority of the normal Task Classes. Cyclic#2 then follows etc.
- The high priority classes interrupt the lower ones:
- The task classes with different cycle times are started with a 10msec delay.

2.3 Task Parameters

Edit: Properties:



The right mouse button opens the pop Up menu.

Fig. 6.5: Task Parameters

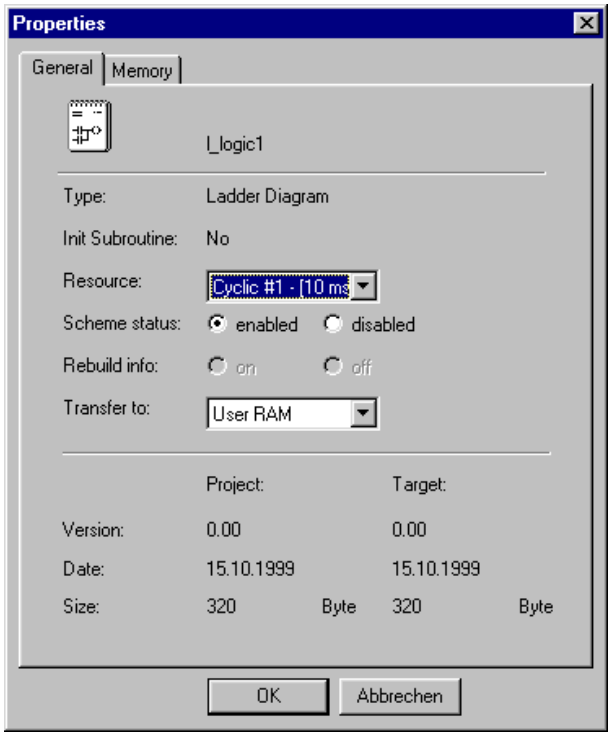


Fig. 6.6: Dialog Task Parameter

Memory: Current memory use is shown on the "Memory" page.

Type: Task Type (LAD, ANSI-C).

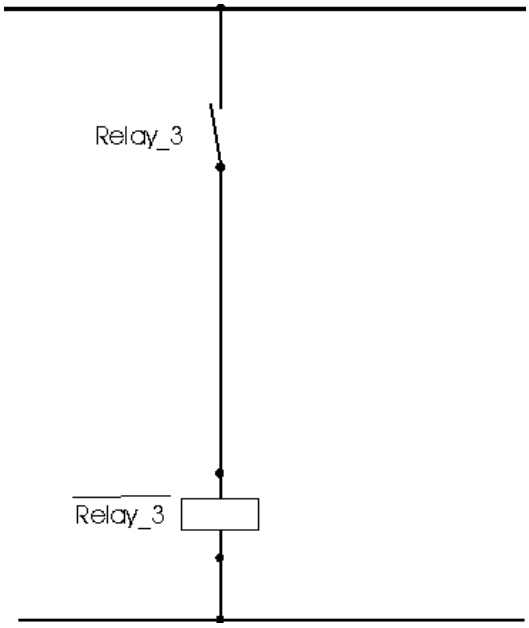
Init Subroutine:
Select Yes/No

Resource: List box settings for each task can be edited here.

Transfer Info: Target of the task Download. RAM, USER ROM, FIXRAM, MEMCARD.

Example: Now that the task classes have been thoroughly discussed, we can now look at the effect of changing task classes in the following exercise.

Create the following circuit diagram.



Name	Type	Scope	Attribute	Value	Remark
Relay_3	BOOL	global	QP5.0.4.3	---	**Digit. OUT**, channel 3

Project name: proj_001
Task name: l_cyclic

Resource:

1. Create "l_cyclic" using **Cyclic#3**.
2. Create "l_cyclic" using **Cyclic#2**.

Test for time differences between the two classes.

Starting Sequence, Running Sequence:

The setting of the starting or running sequence occurs in the software configuration. Control of the procedure runs from top to bottom.

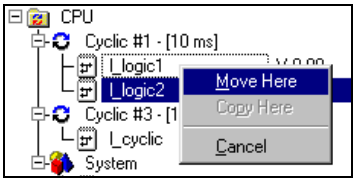


Fig. 6.7: Move Task Class

The tasks can be moved using the left or right mouse button, in a "Drag and Drop" maneuver.

When using the left mouse button no inquiry box will appear.

When using the right mouse button a dialogue box will be opened (see Fig. 6.7: Move Task Class)

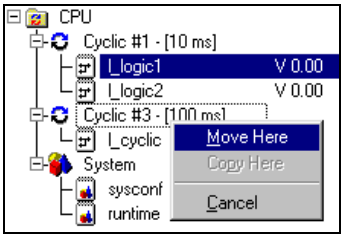


Fig. 6.8: Change Task Class

The tasks can be moved between the various Task classes.

In the diagrams shown here the tasks are moved with the right mouse button.

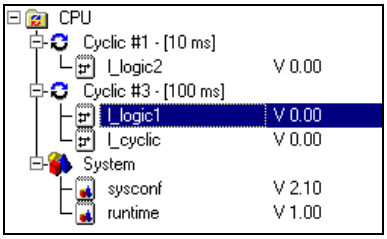
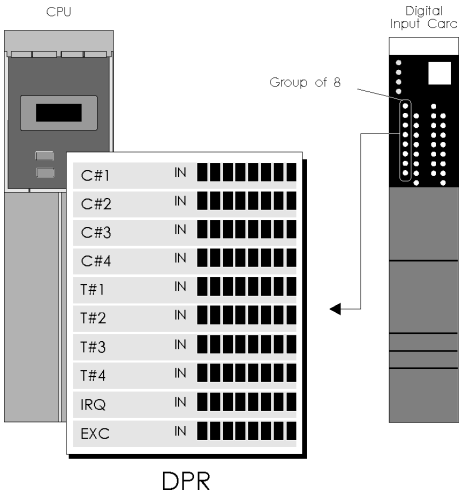


Fig. 6.9: Move Closed

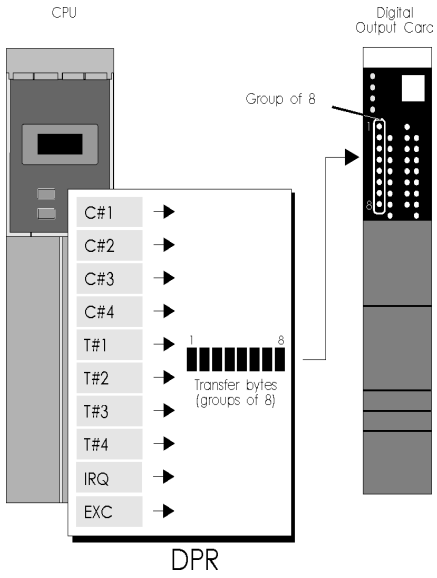
3 PCC I/O HANDLING

3.1 Displays.

The PCC I/O operation is shown as follows:



The input indicator is found at the beginning of the task class. Every task class has its own input indicator. The smallest unit of data that is transported by the I/0 bus is a BYTE.

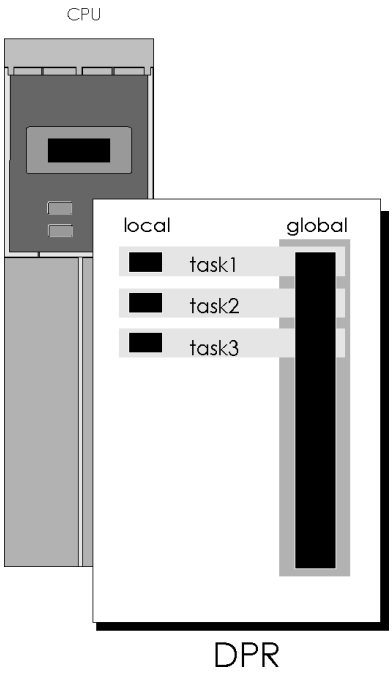


The task class specific output indicator is updated at the end of each Task Class. Every Task Class enters information on to the same output indicator.

Using direct I/O functions, individual Inputs or Outputs of a task can be read or written.

The internal variable image is read and written directly from the CPU.
Internal variables can be defined as follows:

- global ... Variable existing for the entire PCC
- local ... Variable existing only in the task



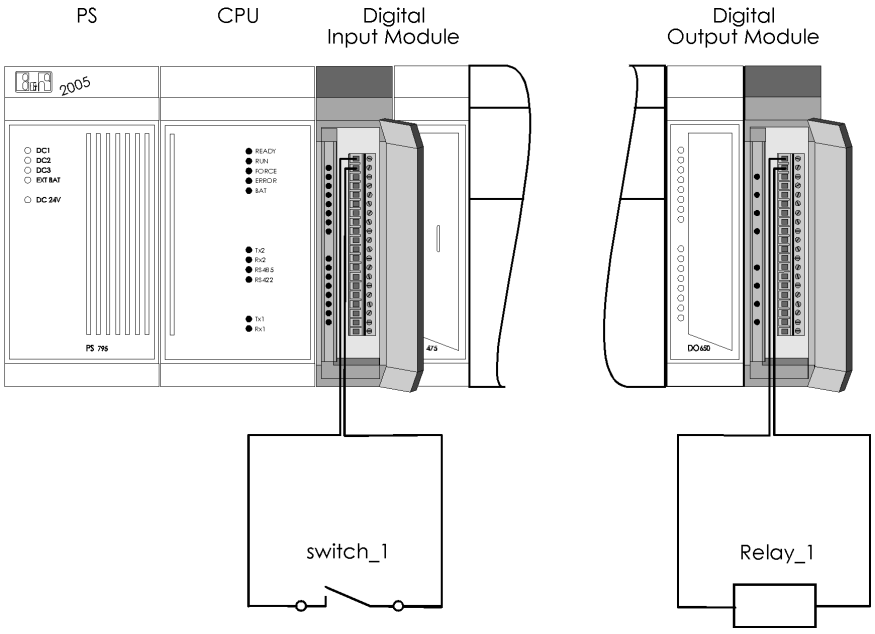
IMPORTANT: These are only used for internal variables.

DPR Controller

The **DPR controller** coordinates access to the multi-processors on the entire DPR. **Bit/Byte access** is also supported by the DPR controller e.g. the CPU does not differentiate whether a Bit, a negated Bit, or byte value is to be processed.

4 VARIABLE DECLARATION

Every contact in the program is referred to with a symbolic name. In the variable declaration the **relationship between symbolic names and the hardware** are defined (inputs, outputs, internal variables).



During variable declaration the following points are defined.

- Name ... Name of the PV
- Type ... Data type of the Variable
- Scope ... Validity class
- Attribute ... I/Os or internal declaration
- Value ... Initialization value
- Remark ... Additional descriptions (long name)

Name	Type	Scope	Attribute	Value	Remark
Key_1	BOOL	global	IP5.0.3.1	----------	1ms switching delay
Key_2	BOOL	global	IP5.0.3.2	----------	1ms switching delay
Key_3	BOOL	global	IP5.0.3.3	----------	1ms switching delay
Key_4	BOOL	global	IP5.0.3.4	----------	1ms switching delay
analog_in1	INT	global	IP5.0.5.1	----------	±10 V, resolution 12 Bit
analog_in2	INT	global	IP5.0.5.2	----------	±10 V, resolution 12 Bit
analog_in3	INT	global	IP5.0.5.3	----------	±10 V, resolution 12 Bit
analog_in4	INT	global	IP5.0.5.4	----------	±10 V, resolution 12 Bit
Relay_1	BOOL	global	QP5.0.4.1	* remanent	Transistor, 0.5A/24VDC
Relay_2	BOOL	global	QP5.0.4.2	* remanent	Transistor, 0.5A/24VDC
Relay_3	BOOL	global	QP5.0.4.3	* remanent	Transistor, 0.5A/24VDC
Relay_4	BOOL	global	QP5.0.4.4	* remanent	Transistor, 0.5A/24VDC
analog_out1	INT	global	QP5.0.6.1	* remanent	±10V, resolution 12Bit
analog_out2	INT	global	QP5.0.6.2	* remanent	±10V, resolution 12Bit

Fig. 6.10: Variable declaration overview

Name

- The first **32 characters** of the name are significant, further characters are entered only in remark
- The **first character** of a name **must** be a **letter** or "_"
- Only "_" is recognized as a non-letter character
- **Capitals** and **lower case text** are differentiated (case sensitive)
- The name **must not** include any **key words** (e.g. Command, operator...)

Type

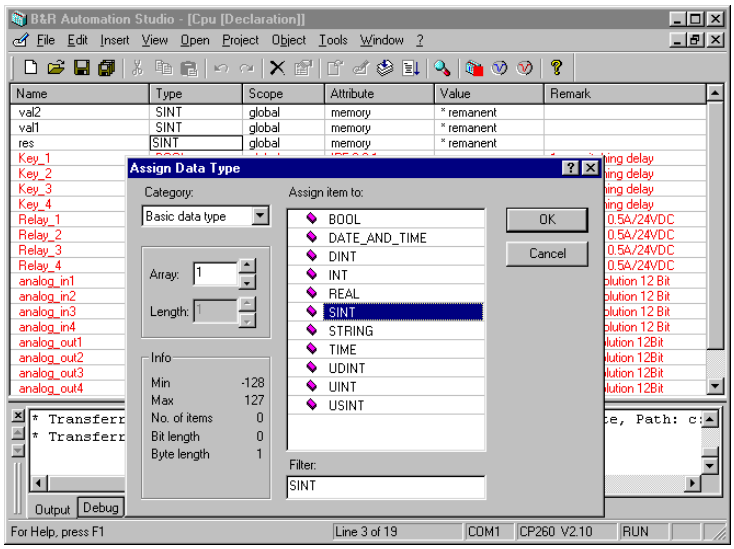


Fig. 6.11: Variable declaration Type selection

The possible types are checked and added to the declaration immediately. If, when using Functions blocks different types are possible, these can be selected using <Space>..

Possible types and relating value ranges

Name	Bit-Size	Value Range	Use
BOOL	1	0 .. 1	digital I/Os
DINT	32	-2 147 483 648 .. 2 147 483 647	
INT	16	-32768 .. 32767	analog I/Os
SINT	8	-128 .. 127	
UDINT	32	0 .. 4 294 967 295	
UINT	16	0 .. 65535	
USINT	8	0 .. 255	
REAL	32	-3.4 E38 .. 3.4 E38	

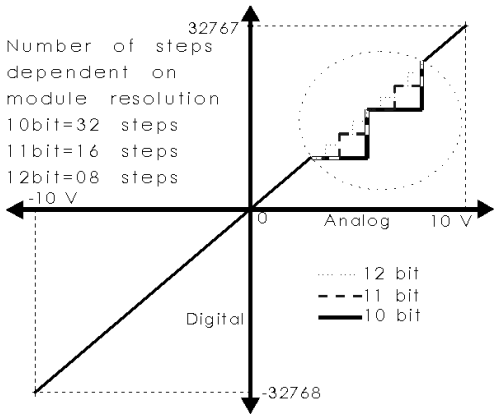
To save text and data formats:

Name	Width	Value Range	Use
STRING	1 – xx Byte	2 characters – 32767 characters	"Text"
TIME	4 Byte	0 .. 4 294 967 295 msec	Time difference
DATE_AND_TIME	4 Byte	Seconds since 1970	Date Calculation

Analog Inputs and Analog Outputs

The control generation B&R2000 always uses analog values between the range +32767 to –32768, independent of the hardware resolution of the card. This adaptation is conveyed to the card.

Data type is always INT (16 Bit)



Scope

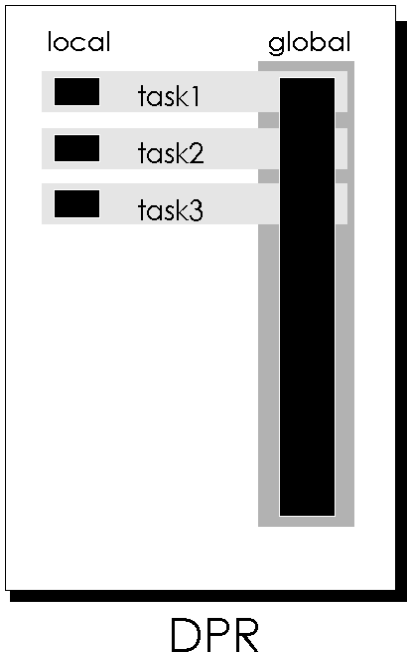
Name	Type	Scope	Attribute	Value	Remark
AValue	UINT	global	memory	* remanent	
Count_Up	CTU	global	memory	---------	
OutVal	UINT	local	memory	* remanent	
Value	UINT		memory	* remanent	

Fig. 6.12: Variable declaration. Validation

local ... Only recognized in the task: Variable for internal use

global ... Variables involved in data exchange between tasks. I/O symbols are assigned to attribute in I/Os.

task1	local_var = 19	global_var = 22
task2	local_var = 1	
task3	local_var = 26	



Attribute

Name	Type	Scope	Attribute	Value	Remark
AValue	UINT	global	memory	* remanent	
Count_Up	CTU	global		---------	
OutVal	UINT	global	memory	* remanent	
Rel_1	BOOL	global	constant	* remanent	Transistor, 2A/24VDC
Value	UINT	global	memory	* remanent	

Fig. 6.13: Variable declaration Attribute memory

Name	Type	Scope	Attribute	Value	Remark
AValue	UINT	global	constant	00123	Konstante für FUB
Count_Up	CTU	global		---------	
OutVal	UINT	global	memory	* remanent	
Rel_1	BOOL	global	constant	* remanent	Transistor, 2A/24VDC
Value	UINT	global	memory	* remanent	

Fig. 6.14: Variable declaration Attribute constant

I/O symbol ... abbreviated term of the I/O- point. Attributed to the hardware configuration.

Memory ... Internal operation memory.

Constant ... Constant value. Cannot be change in the program.

The individual variables can be identified by their color code.

I/Os .. red
Internal .. black
Constants .. green

Value

Name	Type	Scope	Attribute	Value	Remark
AValue	UINT	global	constant	00123	
AW_aus	INT	global	IP5.0.6.1	* remanent	±10V, resolution 12Bit
AW_ein	INT	global	IP5.0.5.1	----------	0..10 V, resolution 12 Bit
Count_Up	CTU	global	memory	----------	
OutVal	UINT	global	memory	* remanent	
Rel_1	BOOL	global	QP5.0.4.1	* remanent	Transistor, 2A/24VDC
Rel_2	BOOL	global	QP5.0.4.2	* remanent	Transistor, 2A/24VDC
Taster_1	BOOL	global	IP5.0.3.1	----------	10ms switching delay

Fig. 6.15: Variable declaration. Initialization

Initialization value for **Internals** or **outputs**.

The value assignment follows the variable declaration. The required value is entered in the position of *remnant.

At every initialization (after power cuts, warm starts, program download, cold start etc) the initial values are automatically entered. The program variables can of course be overwritten when the task runs.

Information: Variables that can be initialized though declaration, can also be overwritten by an **INIT SP**. The Init Sp is executed **after** the variable declaration.

Remark Additional text as commentary. Serves only as documentation and can be up to 34 characters long.

4.1.1 Classification of the program variables to hardware

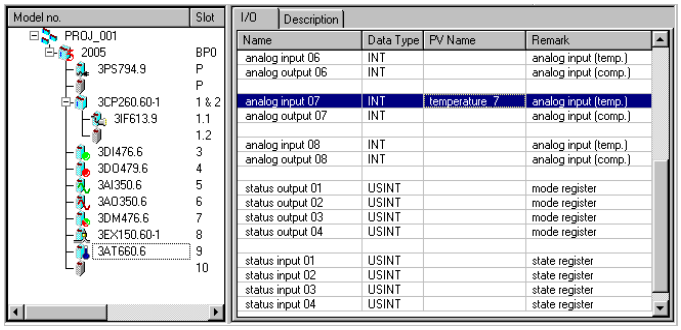


Fig. 6.16: I/O Classification

I/O classification AT660. (Fig: 6.15)

With the AT660 Module the user can select **"mode variables**" in additional to the normal analog I/O variables (Fig: 6.16).

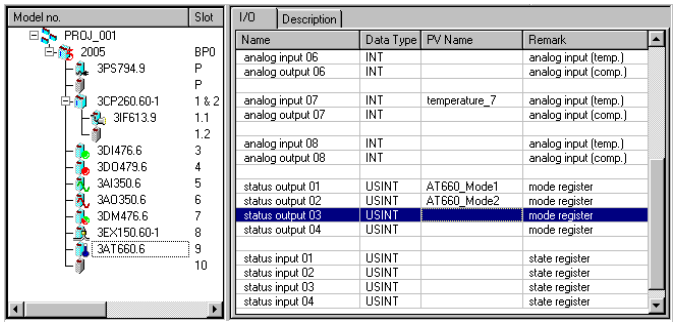


Fig. 6.17: Mode Classification

The **status information** of the module can be defined here. The declaration runs like those from normal variables.

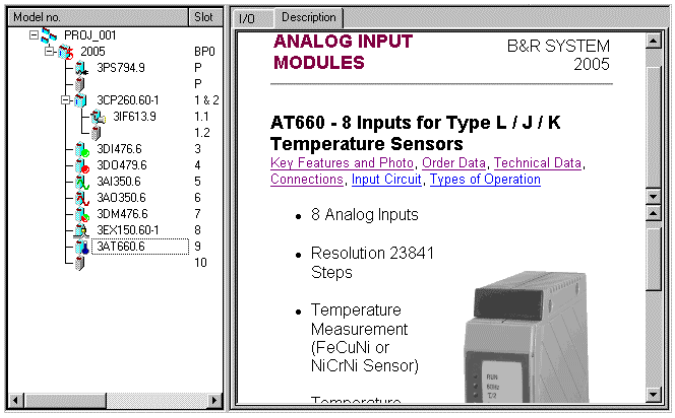


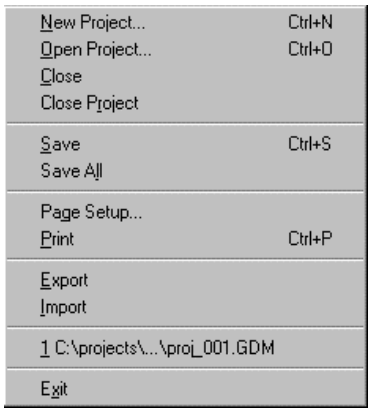
Fig. 6.18: Order information

Order information contains the documentation of each module.

The user has the most important current information.

5 MENU STRUCTURE

File:



Create New Project
Open existing project
Close current editor
Close current project.

Save current file
Save all changed files

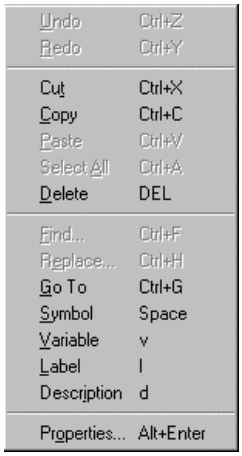
Page settings for print out.
Print

Export the project
Import

File list with recently opened files.

End program

Edit:



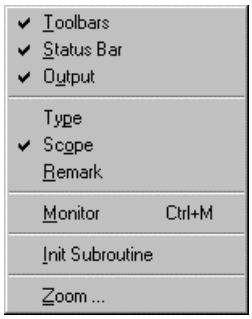
Reverse last alteration
Repeat last alteration

Cut out the marked text
Copy the marked text
Insert on to the page
Select all
Delete the marked text

Search for determined text
Search and replace
Go to Line.....
Edit Symbol
Edit Variable
Edit Label
Edit Description

Change Settings. Edit Variables

View:



Activate Toolbar
Activate Status list
Activate message window

Activate type in LAD
Activate scope in LAD
Activate remark in Ladder Diagram

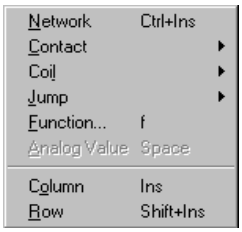
Activate monitor mode

Look at initialization sub routine

Select zoom mode

Init Subroutine: Init subroutine: The Init subroutine is directly run only after the program start (after power failure, download, warmstart, coldstart). All init subroutines are still called up before the cyclic tasks.

Insert: (Insertion in Ladder diagrams)



Insert new Network
Insert new contact
Insert new coil
Inset new Jump
Insert Functions Block
Insert analog value

Insert column
Insert row

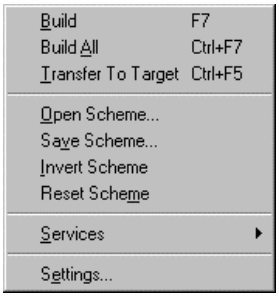
Open:

Open data type manager
Open library manager

Open active object
Open variable declaration

Open Watch (monitor)
Open tracer

Project:



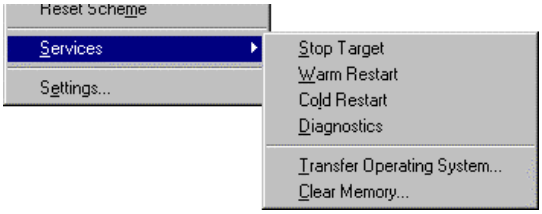
Compile project (only changed files)
Compile all files
Download project to target

Open scheme
Save scheme
Invest scheme
Reset Scheme

Open Service sub-menu

General settings for the project

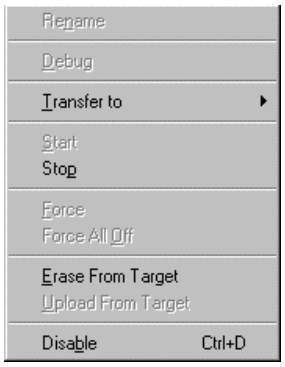
Project: Service:



Stop target
Reset with warm start
Reset with cold start
Switch to diagnosis mode

Transfer operating system
Clear memory

Object:



Rename the object

Activate Debug Mode (C-Tasks)

Transfer to..(RAM, USER ROM, FIXRAM ..)

Start Task
Stop Task

Variable force
All variables in current Watch: Switch off Force

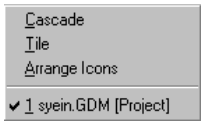
Delete object from the controller
Upload object from the controller

Deactivation of the object

Tools:

Program settings

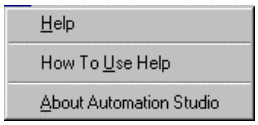
Window:



Cascade window
Tile window
Arrange Icons

Window list

Help:



Calls up online
Use of the Automation Studio

About Automation Studio

LADDER DIAGRAM

1 GENERAL INFORMATION

1.1 Programming languages

B&R provides the ideal programming language for every application and programmer preference. The spectrum comprises:

- Ladder diagram (LAD)
- Instruction List (IL)
- Structured Text (ST)
- Sequential Function Chart (SFC)
- B&R Automation Basic (AB)
- ANSI C

LAD **The functionality of contact, logic, and function plan are all combined in LAD. Due to its similarity with circuit diagrams ladder diagrams are the easiest and most visual form of digital and analog programming.**

IL Instruction List is a machine based language that can be used to create logic connects in a similar way to LAD.

ST This high level language is an easy to understand, high performance programming language for automation systems. Standard construction guarantees fast and efficient programming.

SFC SFC is a programming language developed to dissect tasks into clearly organized sections. SFC is suited to processes reliant on a sequence of steps e.g. car washes

AB This is a B&R high level language. It is easy to understand, high performance programming language designed for the latest generation of automation systems. Standard construction guarantees fast and efficient programming. Previously known as PL2000

ANSI C High level language. High performance programming designed for the latest generation of automation systems. Standard construction guarantees fast and efficient programming.

2 LADDER DIAGRAM

Ladder diagram corresponds to **IEC1131-3** standards.

Standardized variable declaration and program processing applies to all programming languages. These include:

- Standard data types
- Individual variable declaration
- Implicit data conversion (exception is LAD)
- Additional functions for hardware service

Programming in Ladder diagram is extremely popular because of its

- **Simple** programming technique
- **Clarity** when searching for mistakes

Size

The LAD offers **50 columns and 50 lines per net**. The total number of nets is only restricted by the memory size of the computer.

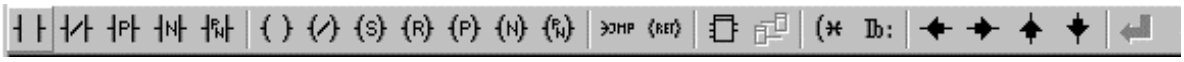
Plausibility test

An online Plausibility test is available whilst programming. The following points should be noted while programming:

- Digital inputs are entered in to columns 1-49.
- Analog inputs are locked directly on to the FBKs
- Line and contact types checked for type similarity
- Unknown variables are declared immediately
- When saving or leaving the network another check is run for mistakes. This check will find any outstanding failing outputs, open lines, unlabeled contacts or function blocks without in/output parameters and alert the user to their presence

2.1 Ladder diagram programming

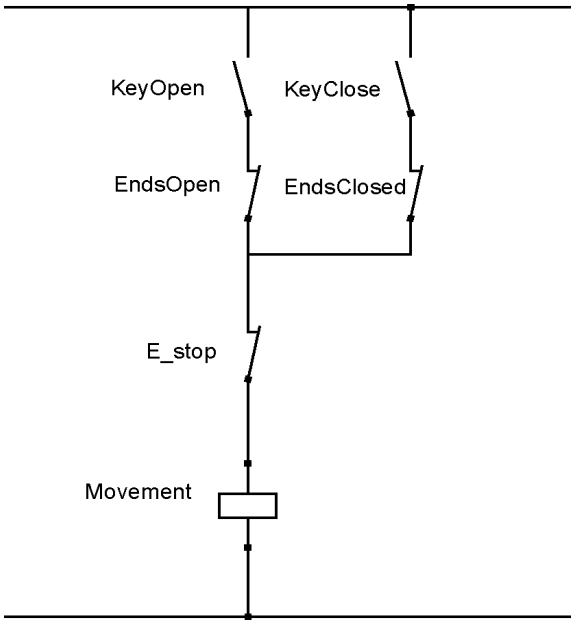
The following contacts are possible:



Normally Open **<c>**	Normally Closed **<i>**	Pos Trans **<p>**	Neg Trans **<n>**
Trans **<b>**	Coil **<shift><C>**	Negated Coil **<shift><I>**	Set Coil **<shift><S>**
Reset Coil **<shift><R>**	Pos. Trans Coil **<shift><P>**	Neg. Trans Coil **<shift><N>**	Trans. Coil **<shift><B>**
Jump **<j>**	Return **<e>**	Function Block **<f>**	Analog Value **<space>**
Description **<d>**	Label **<l>**		
Line Left **<Alt><CuLeft>**	Line Right **<Alt><CuRight>**	Line Up **<Alt><CuUp>**	Line Down **<Alt><CuDown>**
Verify **<Enter>**			

The Editor automatically shows the menu list when opening a Ladder Diagram. In addition the list can be activated or deactivated via the menu "**View: Toolbars**".

Example: Create the following Ladder diagram.



Name	Type	Scope	Attribute	Value	Remark
Movement	BOOL	global	QP5.0.4.1	* remanent	Dig. OUT, channel 1
EndsClosed	BOOL	global	IP5.0.3.3	----------	Dig. IN, channel 3
EndsOpen	BOOL	global	IP5.0.3.4	----------	Dig. IN, channel 4
E_stop	BOOL	global	IP5.0.3.5	----------	Dig. IN, channel 5
KeyClose	BOOL	global	IP5.0.3.1	----------	Dig. IN, channel 1
KeyOpen	BOOL	global	IP5.0.3.2	----------	Dig. IN, channel 2

Create a project called "proj_lad"

Project name: proj_lad
LAD name: l_logic3

Resource: C#3

2.2 Program Test

After creating the task it is downloaded in to the CPU and should be tested as soon as possible.

The **LAD – Monitor** is the ideal tool for this.

View: Monitor: Monitor: "View Monitor" or **<Control><M>** switches Automation Studio to monitor mode.

The LAD monitor is now displayed instead of the LAD editor.

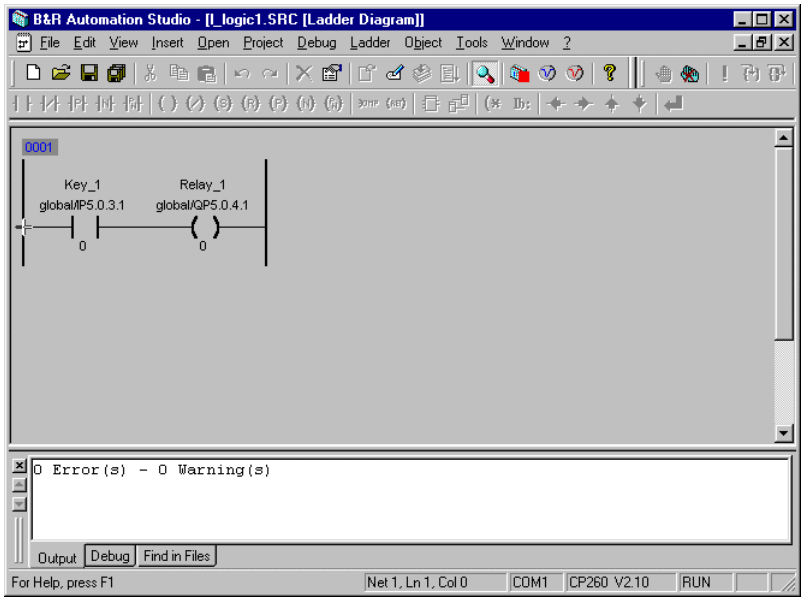


Fig. 7.1: Ladder Diagram Monitor

2.2.1 Functions of the LAD monitor.

The **right mouse** or Win95, Win98 Context button **<shift><F10>**, opens a pop up context menu. The monitor mode should be selected.

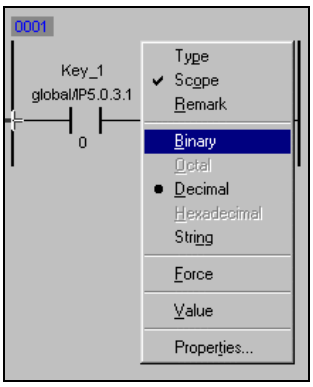


Fig.7.2: LAD Context menu

Display options

"Type"	Displays the active variable
"Scope"	Displays active scope
"Remark"	Displays long text box
"Binary"	Displays binary values True or False
"Decimal"	Displays decimal forms
"Hexadecimal"	Display in HEX format
"String"	Display in ASCII characters
"Force"	Activates or deactivates force mode
"Value"	Changes a value

The above options can be changed via the **View** menu.

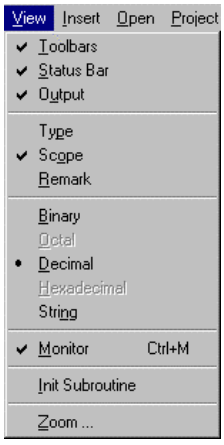


Fig. 7.3: View Menu

The functions of the menu are the same as above apart from Init subroutine and zoom.

Init Subroutine: Displays the initialization program

Zoom: Changes the zoom factor.

Conditional Setting and Re-setting

It is often necessary to create a small impulse. Such impulses are created through buttons, switches, increment providers, etc.

The preset-function is used for conditional setting. **The output is set by a recognized 1-signal and remains set** until deleted by a new command.

 Set Coil	Conditional Set	**<shift><S>**

The reset-function is used for conditional delete**. The output is deleted when a 1-signal is recognized**.

 Reset Coil	Conditional Reset	**<shift><R>**

Edges

Edges must be monitored in the form of input signal monitoring, for reasons of security, impulse counting etc. If the user wants to check a falling or increasing signal, the contact responsible is selected**. After the contact an impulse remains.**

 Pos Trans	Pos. trans reacts to rising edge	**<p>**
 Neg Trans	Neg. trans reacts to falling edge	**<n>**
 Trans	Trans reacts to both edges	**<b>**

If the request requires **memory space, independent of a task class cycle, the edge outputs** should be used.

 Pos. Trans. Coil	Pos. trans coil reacts to rising edge	**<shift><P>**
 Neg. Trans. Coil	Neg. trans coil reacts to falling edge	**<shift><N>**
 Trans. Coil	Trans. coil reacts to both edges	**<shift><B>**

Example: Program a connection with the output "RUN"

- The positive edge of input "START" sets output run.
- The negative edge of the input "STOP" stops output run.

Name	Type	Scope	Attribute	Value	Remark
RUN	BOOL	global	QP5.0.4.2	* remenant	Dig. OUT, channel 2
START	BOOL	global	IP5.0.3.6	----------	Dig. IN, channel 6
STOP	BOOL	global	IP5.0.3.7	----------	Dig. IN, channel 7

Solve this task in your project PROJ_LAD

Project name: proj_lad
LAD name: l_logic4

Resource: C#3

To close the functions of the Ladder Diagram the following commands are important: Jump, Label, Return and Description.

Jump	Conditional jump to labeled point	**<j>**
Label	Label target for jump.	**<l>**
Return	Move back to the next highest level.	**<e>**
Description	Description (commentary)	**<d>**

Jump: Conditional jump to labeled point.
If the connection for the jump is on **Logic 1**, then the jump will be carried out to this internal.

Label: Jump internal, Target for the Jump:

Return: Conditional exit of a function.

A function can be a task but can also be a functions block. As a result it is possible to use a functions block within a Ladder Diagram.

Description: Comment:

Used for documentation of the Ladder Diagram. Should be available in every Ladder Diagram!

Example: Create the following LAD and test the functions.

First create the LAD in Fig. 7.4, then continue to develop the **LAD** to resemble Fig 7.5 and test it.

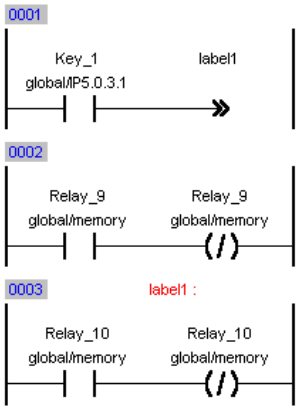


Fig. 7.4: jump_lad

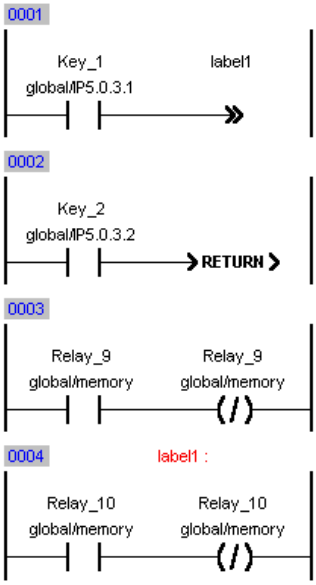


Fig. 7.5: expanded jump_lad

Name	Type	Scope	Attribute	Value	Remark
Relay_9	BOOL	global	QP5.0.4.9	* remanent	Dig. OUT, channel 9
Relay_10	BOOL	global	QP5.0.4.10	* remanent	Dig. OUT, channel 10
Key_1	BOOL	global	IP5.0.3.1	----------	Dig. IN, channel 1
Key_2	BOOL	global	IP5.0.3.2	----------	Dig. IN, channel 2

Project name: proj_lad
LAD name: jump_lad

Resource: C#3

2.3 Working with Function blocks.

Function block are in principle "black boxes" which present the user with complex functions represented in a graphical form to aid simplification.

The standard function blocks are always available.

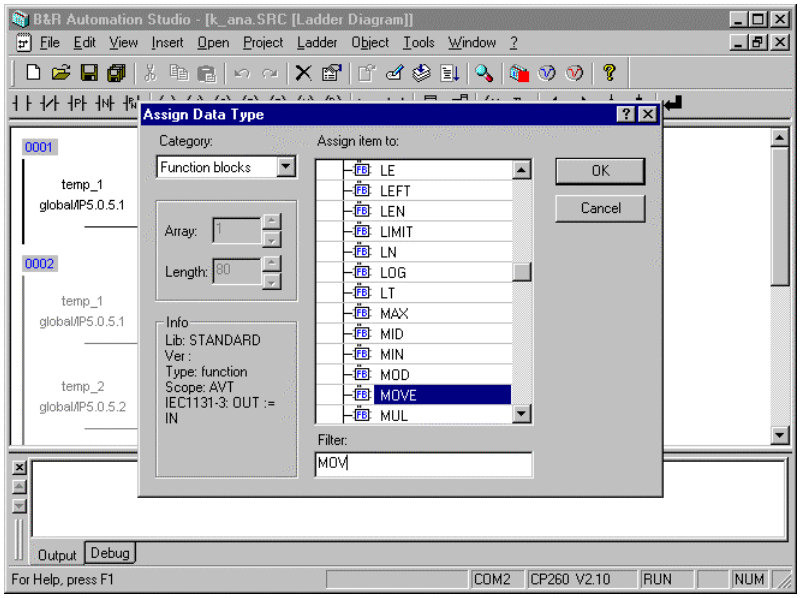


Fig. 7.6: Function block selection

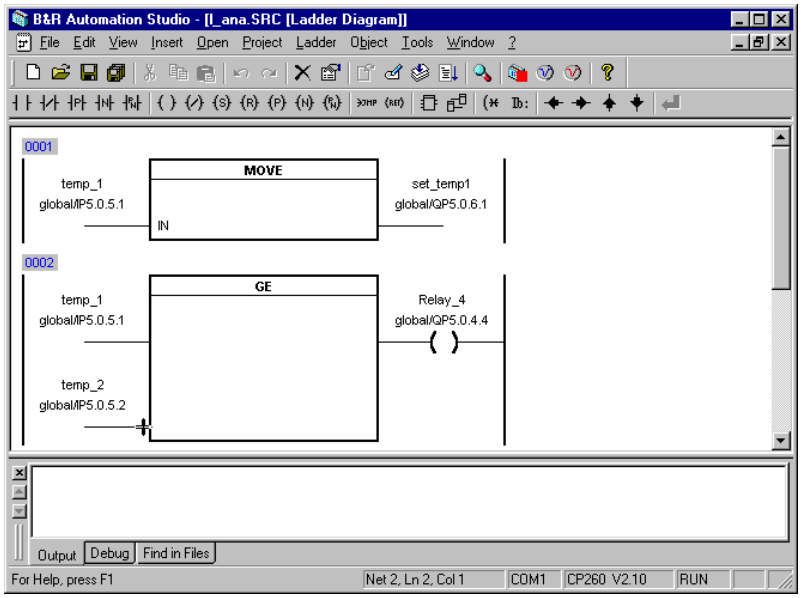


Fig. 7.7: LAD working with analog values

Example: The following example highlights the simplicity of working with functions blocks and analog values.

Entry and output of analog values in Ladder Diagram.

Analog input "TEMP1" -> Analog output "AV_OUT1"

Analog input "TEMP2" -> Analog output "AV_OUT2"

If TEMP1 >= TEMP2 an “output” should be set.

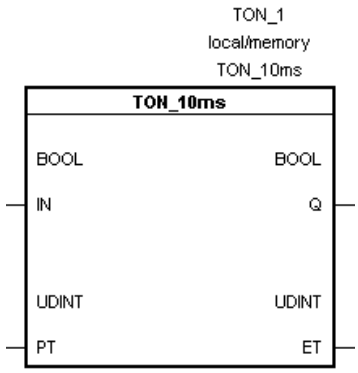
Function Block "GE".

Name	Type	Scope	Attribute	Value	Remark
AV_OUT1	INT	Global	QP5.0.6.1	* remanent	Ana. OUT, channel 1
AV_OUT2	INT	Global	QP5.0.6.2	* remanent	Ana. OUT, channel 2
TEMP1	INT	Global	IP5.0.5.1	----------	Ana. IN, channel 1
TEMP2	INT	global	IP5.0.5.2	----------	Ana. IN, channel 2
output	BOOL	Global	QP5.0.4.14	* remanent	Dig. OUT, Kanal 14

Project name: proj_lad
LAD name: l_ana

Resource: C#2

Example: Create a turn on delay of 2 seconds with the help of the function block "TON_10ms".



TON_10ms:

Timer on Delay

The setting of IN causes the following to occur: the value of ET is raised every 10msecs to 1 until the value of PT is reached. After time is run through (PT = ET) the output Q is set.

IN To delayed input

Q Output

PT Pre-set time

ET Elapse time

Name	Type	Scope	Attribute	Value	Remark
ELAPSE	TIME	global	Memory	* remanent	Elapsed time
PRESET	TIME	global	Memory	* remanent	Pre-set time
relay_5	BOOL	global	QP5.0.4.5	* remanent	Delayed EXIT
Key_1	BOOL	global	IP5.0.3.1	----------	To delayed input

Project name: proj_lad
LAD name: l_ton
Resource: C#2

2.4 Watch (PV- Monitor)

Open: Watch:

The PV monitor can be opened via the menu :**Open Watch**, or by clicking on an object with the **right mouse** button.

Fig. 7.9: Watch

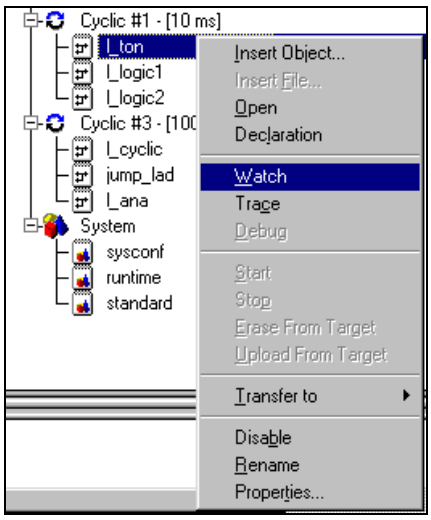


Fig. 7.8: PopUp Watch

Insert: Variable

Variables can be called up in the dialog box.

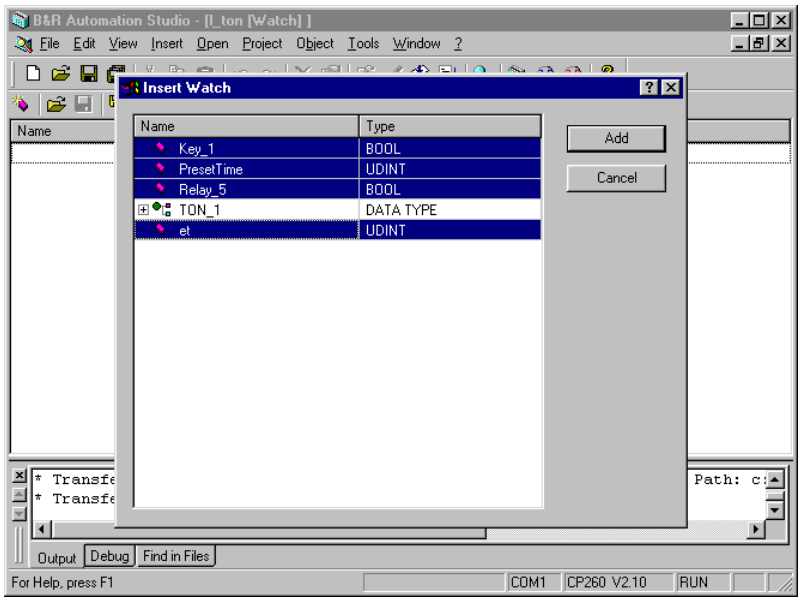


Fig. 7.10: Watch: Variable Selection

Object: Force:

I/Os can be forced, and consequently values can be saved in the memory.

Input (forcing possible) Output (forcing possible)

When changing the variables a security box will appear, after which the variable will be forced. As shown below:

Object: Force all off: Switch off all forcing:

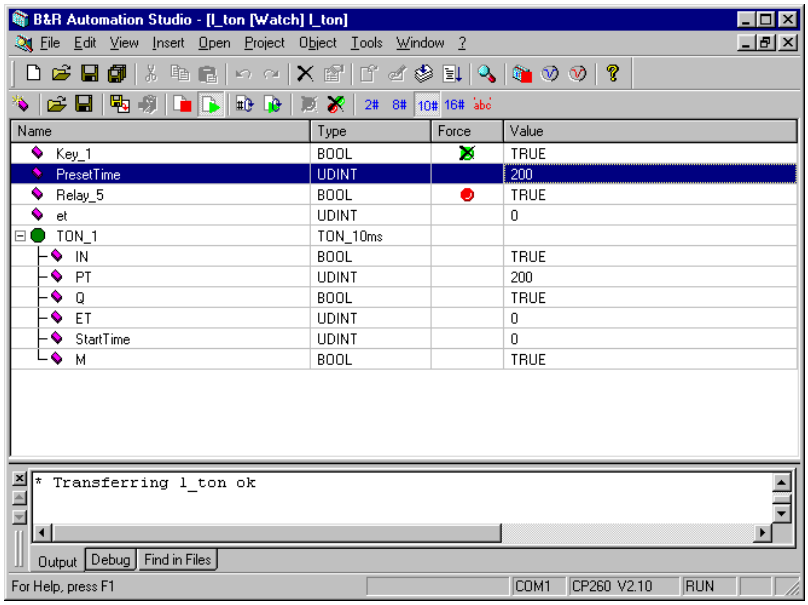


Fig. 7.11: Watch Window

Example: With the previous example of a TON function Block:

- Test Watch
- Test Force

View: Archive: Activates/deactivates archive mode.

The values can be downloaded to the CPU with the "**Object: Write Values**" button or "**Object: Write Value**" menu.

Supplementary altered values are displayed in a different color. It is therefore simple to see the difference between saved work and current work.

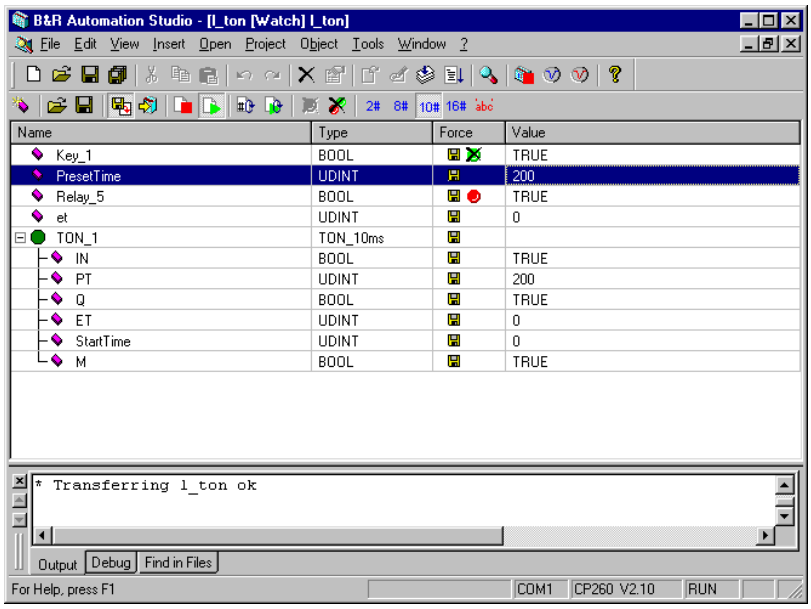


Fig. 7.12: Watch window archive Mode

Object: Save Data: Save dialog

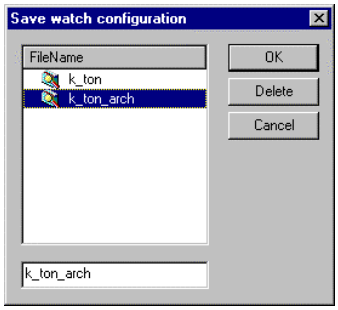


Fig. 7.13: Save Dialog

This dialog box enables the user to save the current value to the hard disk.

Object: Load Data: Load dialog

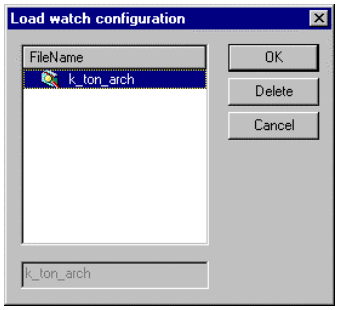


Fig. 7.14: Load Dialog

A watch window is loaded.

If the configuration is saved in the Archive mode the machine will revert (after a security check) to its previous settings. The data will be downloaded to the PCC only upon request.

Example: Secure a current variable image for configuration of a machine belonging to a customer.

- Secure values (archive)
- Change the values on the controller (watch)
- Download secured values from the hard disk to the controller (archive)

Watch Toolbar:

Insert Variable	Open Data Archive Mode	Save Data Archive Mode
Archive Mode activate / deactivate	Write Values Download values to PCC	
Stop Task or PCC stop	Start When total cycles=0, it will be switched in the cycle operation	Set Cycle Count Set number of cycles
Run Cycle Do number of cycles	Force On/Off Force I/O variables ON/OFF	Force All Off
Display as Binary	Display as Octal	Display as Decimal
Display as HEX	Display value as text	

Pop Up Menu: Click with right mouse button in Watch window:

Insert variable

Binary display
Octal display
Decimal display
Hexadecimal display
Display as text

Start
Stop
Cycle
Force

INSTRUCTION LIST

1 GENERAL INFORMATION

1.1 Programming languages

B&R provides the ideal programming language for every application and programmer preference. The spectrum comprises:

- Ladder diagram (LAD)
- Instruction List (IL)
- Structured Text (ST)
- Sequential Function Chart (SFC)
- B&R Automation Basic (AB)
- ANSI C

LAD The functionality of contact, logic, and function plan are all combined in LAD. Due to its similarity with circuit diagrams ladder diagrams are the easiest and most visual form of digital and analog programming.

IL **Instruction List is a machine based language that can be used to create logic connects in a similar way to LAD**.

ST This high level language is an easy to understand, high performance programming language for automation systems. Standard construction guarantees fast and efficient programming.

SFC SFC is a programming language developed to dissect tasks into clearly organized sections. SFC is suited to processes reliant on a sequence of steps e.g. car washes.

AB This is a B&R high level language. It is easy to understand, high performance programming language designed for the latest generation of automation systems. Standard construction guarantees fast and efficient programming. Previously known as PL2000.

ANSI C High level language. High performance programming designed for the latest generation of automation systems. Standard construction guarantees fast and efficient programming.

2 INSTRUCTION LIST

2.1 IL Construction

Label	Operator	Operand	Comment
JUMP:	*LD*	*temp_set*	*(* load set temperature value*)*

Label An identifier (name of the label) must be closed with a colon and be preceded by an IL instruction.

Labels are optional and can stand alone in lines.
A label can be a maximum of 32 characters long.

Operator All instructions (operators) are run by an Accumulator (accu). The length of the accu implicitly matches the operands used and fall in the range BOOL to REAL.

Operands

Process variables or value assignments can be used as operands.

Process variable as Operand: Entered as a symbolic name.
Value assignment as Operand: Value entries are numerical.

100	...	decimal
2#1000_1001	...	boolean
16#2FA89E2C	...	hexadecimal
1.25	...	real

Basic operations have only one operand.

Comment

Text is identified as comments in the following way:
(* Start End ***)**

Empty spaces are allowed in the source.

A comment can not cover several lines.

Instructions

An operator can either be a **Code word** (IL command) or a **Function name** (calling a function block).

Instructions			
LD	Accu = Operand (Op)	**LDN**	Accu = not Op
ST	Op = Accu	**STN**	Op = not Accu
S	Preset Op	**R**	Reset Op

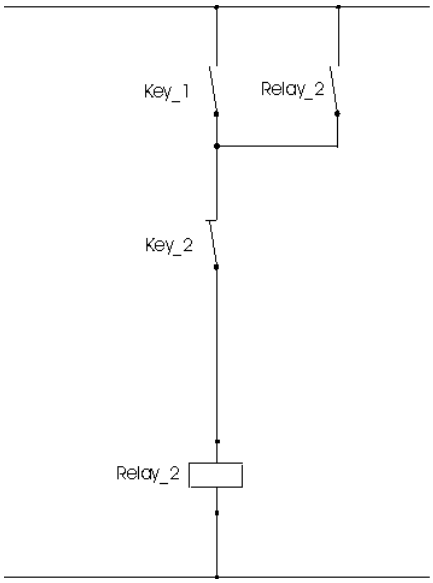
Logic Connections			
AND	Accu = Accu and Op	**ANDN**	Accu = Accu and not Op
OR	Accu = Accu or Op	**ORN**	Accu = Accu or not Op
XOR	Accu = Accu xor Op	**XORN**	Accu = Accu xor not Op

Arithmetic Connections			
ADD	Accu = Accu + Op	**SUB**	Accu = Accu - Op
MUL	Accu = Accu * Op	**DIV**	Accu = Accu / Op

Compare Commands			
GT	Accu > Op	**LT**	Accu < Op
GE	Accu >= Op	**LE**	Accu <= Op
NE	Accu <> Op	**EQ**	Accu = Op

Jump Command			
JMPC	Jump when Accu <> 0	**JMPCN**	Jump when Accu = 0
JMP	Unconditional Jump	**CAL**	Call an FBK

Example: Logic Connection

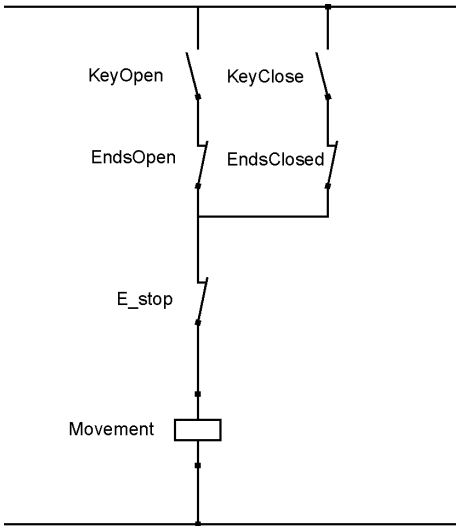


The solution clearly shows that logic programming in IL is very simple.

```
LD      Key_1       (* Accu = Key_1              *)
OR      Relay_2     (* Accu = Accu OR Relay_2   *)
ANDN    Key_2       (* Accu = Accu ANDN Key_2   *)
ST      Relay_2     (* Relay_2 = Accu           *)
```

Working with Brackets

Enclosing instructions with explicit brackets enables programming to be carried out without buffer memory.

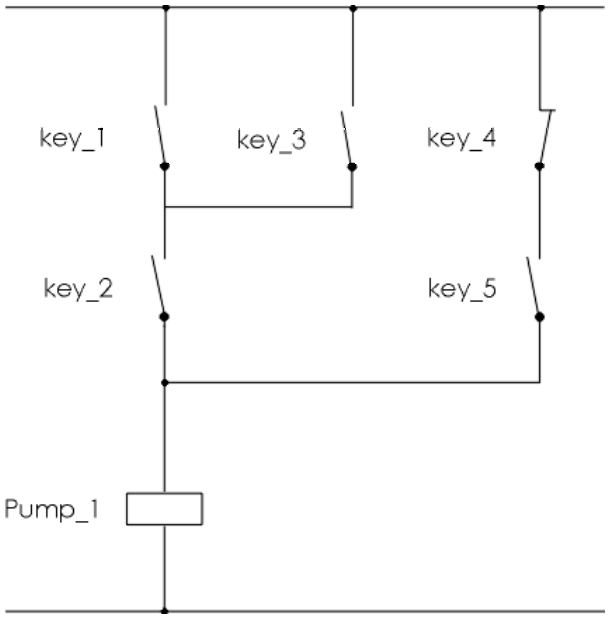



```
LD     KeyOpen       (* Accu = KeyOpen                    *)
ANDN   EndsOpen      (* Accu = Accu AND NOT EndsOpen     *)
OR(    KeyClose      (* Accu# = KeyClose                  *)
ANDN   EndsClosed    (* Accu# = Accu# AND NOT EndsClosed *)
)                    (* End brackets                      *)
                     (* Accu = Accu OR Accu#              *)
ANDN   E_stop        (* Accu = Accu AND NOT E_stop       *)
ST     Movement      (* Move = Accu                       *)
```

If the compiler recognizes a bracket a second accu (accu#) is opened that is linked to the main accu after the bracket is closed.

A maximum of 7 bracket layers can be created.

Example: Create the following circuit diagram in IL.



Project name: proj_il

Program name: i_logic1
Resource: Cyclic#4

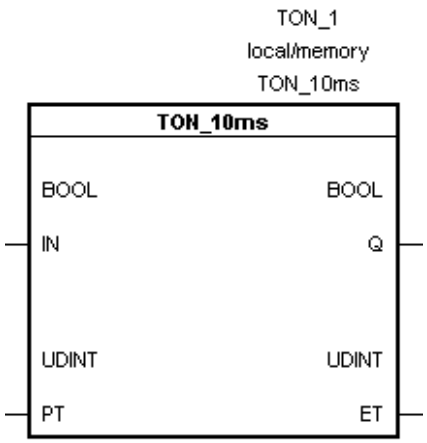
Something extra for you to consider:

- What should you look out for with variable declaration ?

2.1.1 Working with Function Blocks

CAL calls FBKs.

Example: Create a switching delay of 2.56 seconds.

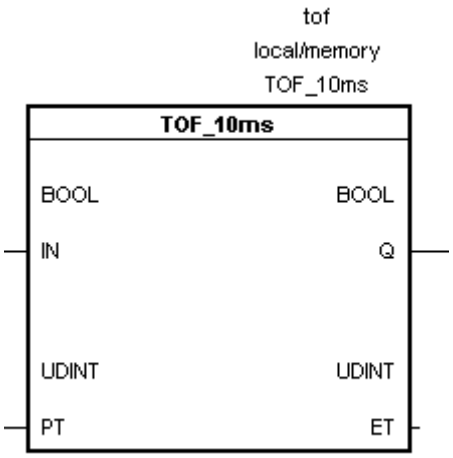



```
CAL    TON_1(IN := Input, PT := 0256)
LD     TON_1.Q
ST     Output
```

All input parameters are given at the end of the FBK name. Output variables can be called using the alias names TON_1.Q.

Always note the text type and sequence of the parameters to be carried out, and all the input parameters !

Example: Create a switching off delay of x seconds with the help of FBK **tof**, in which an analog input is divided by 100 and the TOF is given as a PT time.



Parameter:

IN ... Input to be delayed
PT ... Preset time in 10msecs
Q ... Output
ET ... Elapsed time

Function: By resetting "IN" the "PT" value is decremented by 1 every 10msec and saved as "ET". "Q" is deleted when the time has elapsed .

Information: The data type conversion from INT to UDINT can be carried out using the function INT_TO_UDINT.

Name	Type	Scope	Attribute	Value	Remark
Input	BOOL	global	IP5.0.3.1	---	digital IN channel 1
Output	BOOL	global	QP5.0.4.3	---	digital OUT channel 3
AvIN	INT	global	IP5.0.5.1	---	AV for time
Time	UDINT	local	memory	* remanent	Preset time for TOF

Project name: proj_il

Program name: i_tofdel
Resource: Cyclic#3

2.2 IL Monitor

The variables are displayed directly in Monitor mode in the text editor. Double click on the variable to change it.

The Watch window is also available for IL tasks.

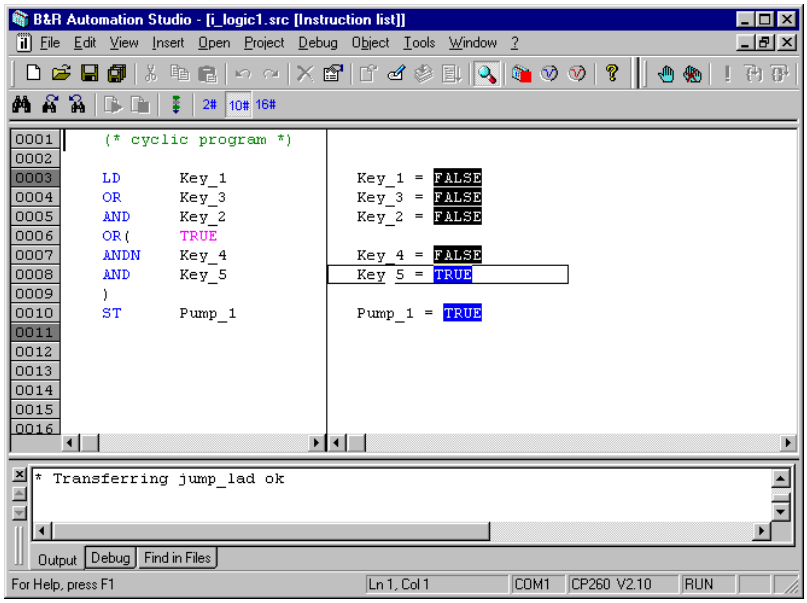


Fig. 8.1: IL in Monitor Mode

Example: Test your IL tasks using the monitor functions:

- IL Monitor
- Watch Window

STRUCTURED TEXT

1 GENERAL INFORMATION

1.1 Programming languages

B&R provides the ideal programming language for every application and programmer. The spectrum comprises:

- Ladder diagram (LAD)
- Instruction List (IL)
- Structured Text (ST)
- Sequential Function Chart (SFC)
- B&R Automation Basic (AB)
- ANSI C

LAD The functionality of contact, logic, and function plan are all combined in LAD. Due to its similarity with circuit diagrams, ladder diagrams are the easiest and most visual form of digital and analog programming.

IL Instruction List is a machine based language that can be used to create logic connects in a similar way to LAD.

ST **This high level language is an easy to understand, high performance programming language for automation systems. Standard construction guarantees fast and efficient programming**.

SFC SFC is a programming language developed to dissect tasks into clearly organized sections. SFC is suited to processes reliant on a sequence of steps e.g. car washes.

AB This is a B&R high level language. It is easy to understand, high performance programming language designed for the latest generation of automation systems. Standard construction guarantees fast and efficient programming. Previously known as PL2000.

ANSI C High level language. High performance programming designed for the latest generation of automation systems. Standard construction guarantees fast and efficient programming.

2 OVERVIEW

Lets take a closer look at **S**tructured **T**ext (abbr: **ST**).

ST is a text high level language.

ST language contruction corresponds to IEC1131-3.

Programming languages make the control tasks much easier to program, programs created in ST can be displayed with great clarity.

Unrecognized variables are defined immediately after use.

ST programming command groups:

- Logic Connections
- Arithmetic Operations
- Logic Compare Expressions
- Conditions
- Loops
- Select

3 PROGRAMMING

3.1 Operator Priorities

Using more than one operator in a line automatically triggers the question of priorities.

Operation	Symbol	Precedence
Brackets	()	highest precedence
Function call Example:	ID (argument list) LN(A), MAX(X)	
Potentiation	EXPT(IN1,IN2)	
Negation Complement	NOT	
Multiplication Division Modulo	* / MOD	
Addition Subtraction	+ -	
Compare	<, >, <=, >=	
Equal to Unequal to	= <>	
Boolean AND	AND	
Boolean Exclusive Or	XOR	
Boolean OR	OR	lowest priority

As shown above, bracketing operators can influence the order of precedence.

Example: Event = 6 + 7 * 5 – 3 (result = 38)
Event = (6 + 7) * (5 - 3) (result = 26)

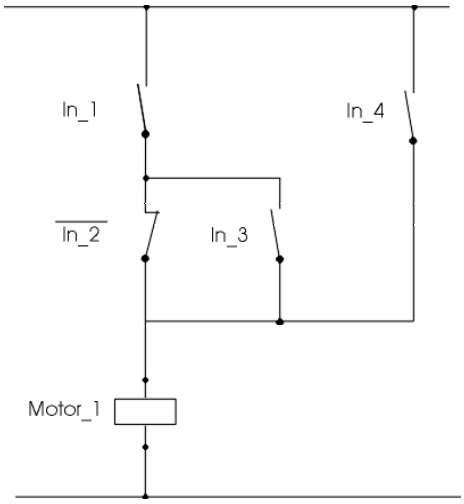
The order of precedence states that multiplication and division have higher priority than addition and subtraction. Brackets (highest priority) can be used to influence the order of precedence.

3.2 Logic Connections

Symbol	Logic Connection	Example
NOT	Negation	`a := NOT b;`
AND	logic AND	`a := b AND c;`
OR	logic OR	`a := b OR c;`
XOR	exclusive OR	`a := b XOR c;`

Logic operators are bit orientated during programming!

Example:

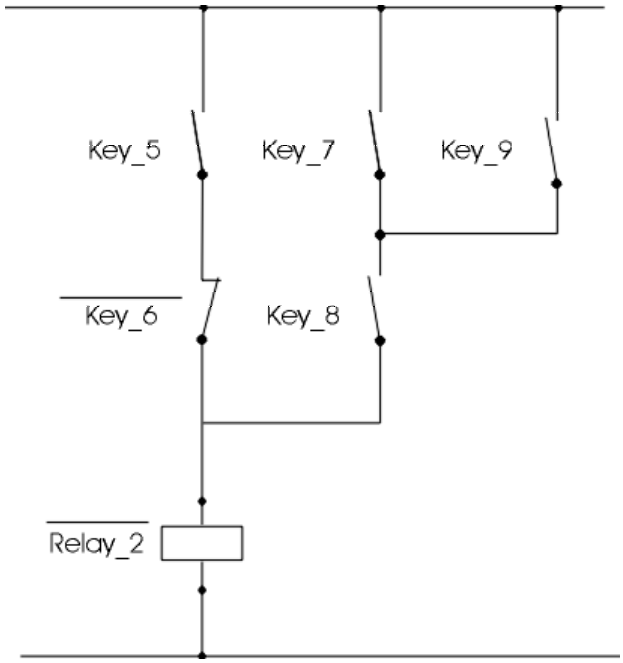


Name	Type	Scope	Attribute	Value	Remark
in_1	BOOL	global	20xx-I/O	----------	Dig. IN, channel 1
in_2	BOOL	global	20xx-I/O	----------	Dig. IN, channel 2
in_3	BOOL	global	20xx-I/O	----------	Dig. IN, channel 3
in_4	BOOL	global	20xx-I/O	----------	Dig. IN, channel 4
Motor_1	BOOL	global	20xx-I/O	* remanent	Dig. OUT, channel 1

```
Motor_1 := (in_1 AND (NOT in_2 OR in_3)) OR in_4;
```

The ST editor allows any number of parallel branches.

Example: Create the following circuit diagram in ST.



Name	Type	Scope	Attribute	Value	Remark
Relay_2	.....	.....	.....	.....	Dig. OUT, channel 2
key_5	.....	.....	.....	.....	Dig. IN, channel 5
key_6	.....	.....	.....	.....	Dig. IN, channel 6
key_7	.....	.....	.....	.....	Dig. IN, channel 7
key_8	.....	.....	.....	.....	Dig. IN, channel 8
key_9	.....	.....	.....	.....	Dig. IN, channel 9

Project name: proj_st
Task name: s_logic

Resource: C#3

3.3 Arithmetic Operations

A deciding factor in using high level programming languages is the simplicity in programming arithmetic operations.

ST offers all fundamental arithmetic functions such as:

Symbol	Arithmetic Operations	Example
:=	Assign	`a := b;`
+	Addition	`a := b + c;`
-	Subtraction	`a := b - c;`
*****	Multiplication	`a := b * c;`
/	Division	`a := b / c;`
mod	whole number division	`a := b mod c;`

The data type is an important factor when doing calculations.

Example: Event := 8 / 3;
Rest := 8 mod 3;

Data type:	**Real**	**Integer**
Event:	2,6667	2
Rest:	2,0000	2

Information: If you want to work with constant REAL values, a minimum of one semi-colon must be used (example: 3 .. INT; **3.0 .. REAL**)

Data Type Conversion:

If different data types are used during an operation, the compiler automatically carries out an **implicit data type conversion**:

Data Type	BOOL	SINT	INT	DINT	USINT	UINT	UDINT	REAL
BOOL	BOOL	x	x	x	x	x	x	x
SINT	x	SINT	INT	DINT	USINT	UINT	UDINT	REAL
INT	x	INT	INT	DINT	INT	UINT	UDINT	REAL
DINT	x	DINT	DINT	DINT	DINT	DINT	UDINT	REAL
USINT	x	USINT	INT	DINT	USINT	UINT	UDINT	REAL
UINT	x	UINT	UINT	DINT	UINT	UINT	UDINT	REAL
UDINT	x	UDINT	UDINT	UDINT	UDINT	UDINT	UDINT	REAL
REAL	x	REAL	REAL	REAL	REAL	REAL	REAL	REAL

× .... explicit conversion required according to the operation.

Example 1: Adding two SINT values (no conversion required).

```
Event := value1 + value2;
```

Example 2: Adding an INT value and a SINT value

```
Event := value1 + value2;
```

- The smaller value is implicitly matched to INT!
- The result must also be an INT value!

Procedure:

- Always converted to bits.
- If the variables are the same size they are converted to unsigned.

Information: Relationship between type and bit width:

SINT .. 8 BIT	"**S**" means short
INT .. 16 BIT	No special text type
DINT .. 32 BIT	"**D**" means double

Example 3: Adding 2 temperatures (must be INT !)

```
Average := (Temp1 + Temp2) / 2;
```

Problem: The sum of both temperatures can easily exceed the positive INT range (+32,767), which would lead to a false (negative) average value.

Solution: If the PVs cannot be added to the effect you want without causing overflow they must be **explicitly converted**. This is done with the help of the STANDARD library.

The functions are contructed as follows.

x .. output data type
y .. target data type

x_TO_y e.g. **BOOL_TO_DINT**(bool_var)

The correct solution is found by following the steps below:

- Convert variable Temp1 to DINT.
  (Compiler executes the lines from left to right).
- The second value is adjusted implicitly.

```
(INT_TO_DINT(Temp1) + Temp2) / 2
```

DINT is converted back to INT so that the analog value can be processed. So the program can be easily understood in years to come, you should convert every "Temp" value explicitly.

```
Average := DINT_TO_INT( (INT_TO_DINT(Temp1) +
                         INT_TO_DINT(Temp2) ) / 2);
```

Example: The room temperature for a large office is recorded at three points.
Your task is to calculate the arithmetic average of these 3 temperatures.

Name	Type	Scope	Attribute	Value	Remark
Average	.....	.....	.....	.....	Analog OUT, chan1
Temp1	.....	.....	.....	.....	Analog IN, channel 1
Temp2	.....	.....	.....	.....	Analog IN, channel 2
Temp3	.....	.....	.....	.....	Analog IN, channel 3

Project name: proj_st
Task name: s_arithm

Resource: C#4

3.4 Trace:

The trace function displays the value directly on the controller.

The upload function takes the values from the controller and displays them in various windows.

This function makes it possible to record values from different task classes in a specifically defined time sequence.

The smallest unit of time is the task class cycle time.

The value can be recorded at the start or end of a task.

Up to 8 values can be recorded simultaneously.

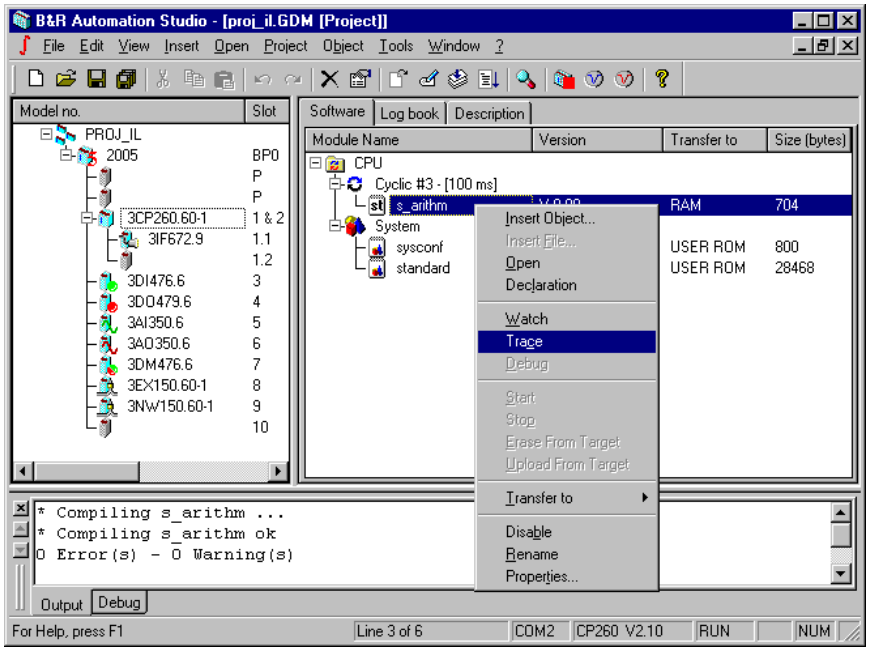


Fig. 9.1: Task Trace PopUp

 Insert: Trace: Inserts a trace configuration

 Insert: Variable: Inserts variables in the trace configuration

Use the following dialog to insert up to 8 variables.

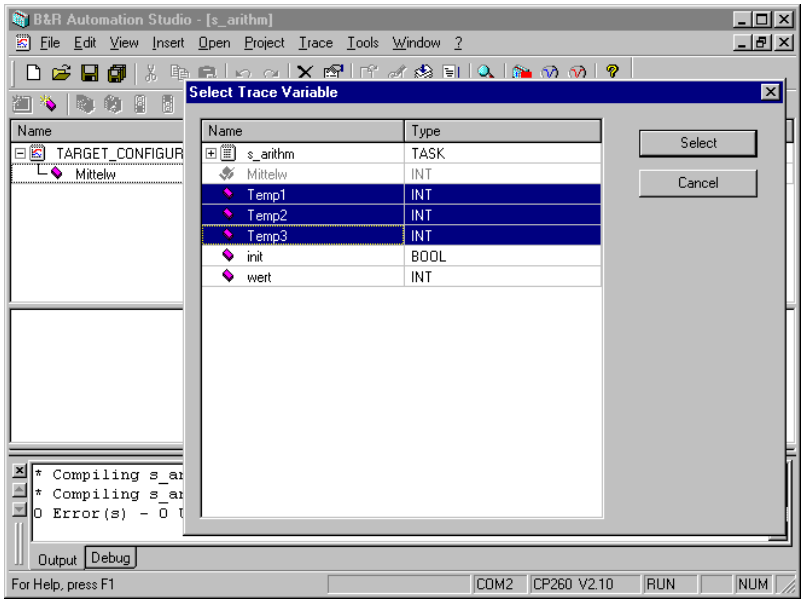


Fig. 9.2: Trace Select Variables

INFO BURST: The monitor mode must be active in order for the trace to be successfully installed.

Trace: Install:

Trace is installed on the controller. The trace automatically enters the variables in the memory module. The start point or the recording point can be configured by you.

Use Start and Stop to start and stop the tracer.

Trace: Show Target Data:

When the tracer is in stopped, the "Show Target Data" can be loaded from the controller.

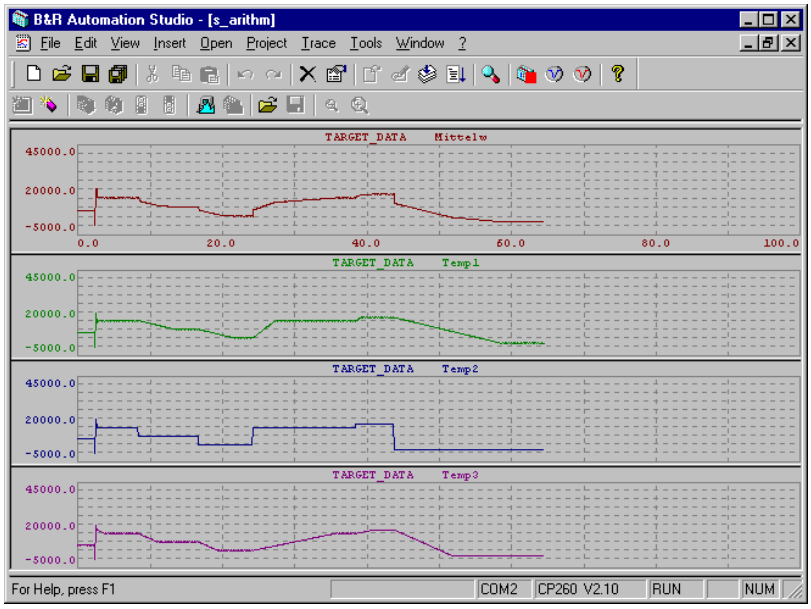


Fig. 9.3: Tracer Display Window

Click on the right mouse button and access various settings under "Display" or "Modify". For example, activate "Curve Cursor" and "Reference Cursor".

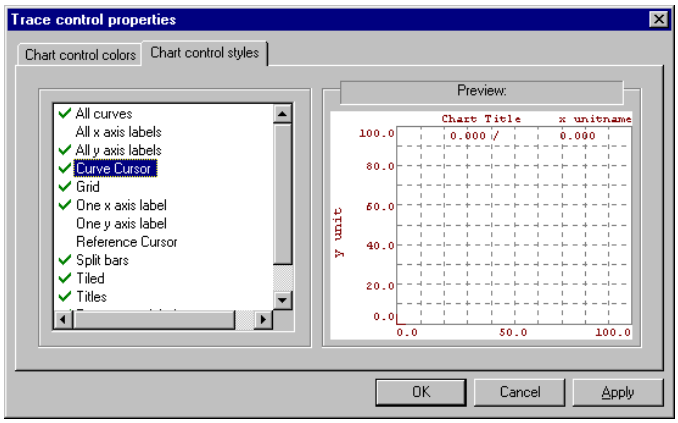


Fig. 9.4: Tracer Settings

The cursor serves as a measurement reference point on the curve.

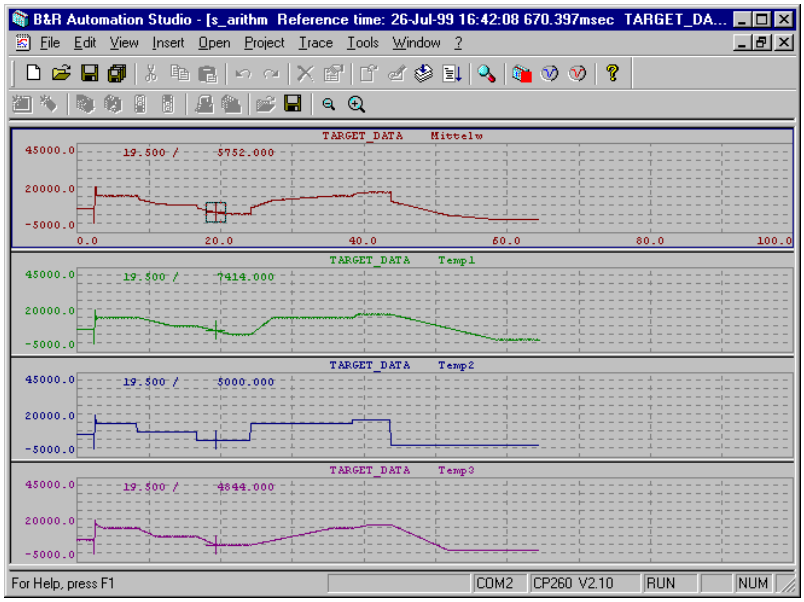


Fig. 9.5: Tracer Measurement Mode

Tracer: Zoom:

The zoom factor can be selected. This allows the opportunity to analyse target data in greater detail.

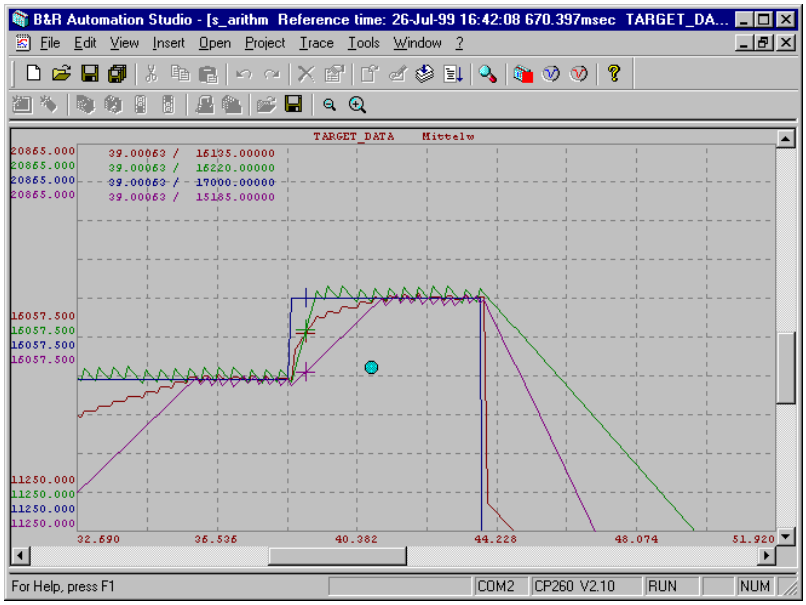


Fig. 9.6: Tracer Zoom in on the Curve

Example: Test the Tracer

Information: In order to set the right value range, turn the analog value up to the maximum and minimum values.

3.5 Logic Compare Expressions

High level programming languages such as ST enable the simple construction of branches for compare operations.

Symbol	Logic Compare Expressions	Example
=	Equal to	`IF a = b THEN ...`
<>	Not equal to	`IF a <> b THEN ...`
>	Greater than	`IF a > b THEN ...`
>=	Greater than or equal to	`IF a >= b THEN ...`
<	Less than	`IF a < b THEN ...`
<=	Less than or equal to	`IF a <= b THEN ...`

3.6 Conditions

Conditions	Example	Description
	`a := b * c;`	calculation
IF		Start Instruction
THEN	`IF a > 0 THEN`	
	`result := 1;`	condition fulfilled
ELSE	`ELSE`	
	`result:= 0;`	condition not fulfilled
END_IF	`END_IF`	End of instruction

- Compare is **true** -> do **THEN branch**
- Compare is **false** -> do **ELSE branch**

The **ELSE branch** of an IF instruction is **not conclusive**, meaning that when the compare result is fasle -> **END_IF**

Program	Description
IF condition **THEN**	IF statement
statement block	condition is true
statement block	
ENDIF	End of the IF block

INIT Value

We should spend some quality time thinking about INIT values.
There are two ways of initializing a process variable with a specific value:

- Program value assignment
  Initialization data is programmed in the INIT SP.
- Declared value assignment
  The initialization data is assigned directly to in the variable declaration of the value chosen as remanent or 0.

INIT SP The INIT SP is programmed in the same way as the task section. Only one task is executed that must be executed **once before all others**, such as:

- Assigning values to process variables.
- System or machine tasks, that must be executed once and before all other tasks.

The INIT SP is called like this:

Position the **Cursor in the Task Source code.**
View : Init Subroutine

Or move the "Splitter Bar" from the top of the window to the bottom.

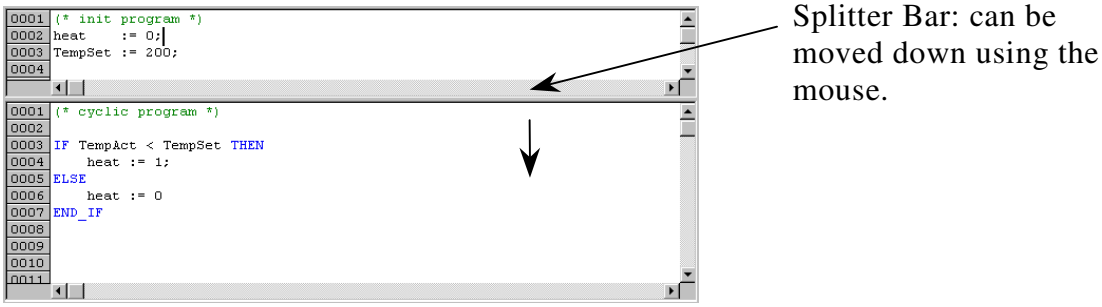
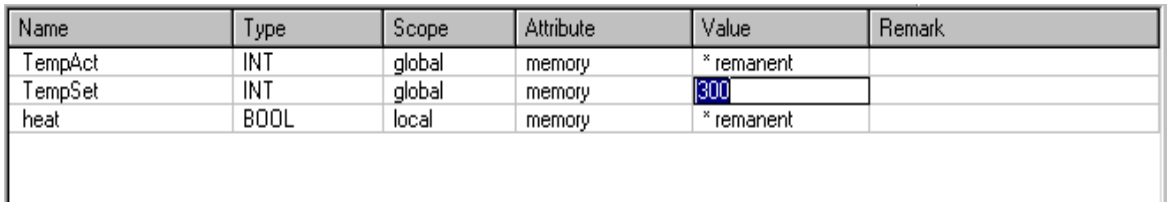


Fig. 9.7: Init SP in ST Task

IMPORTANT

- The INIT SPs of **all** tasks are executed in the defined order before the cyclic tasks.
- Remanent value assignments to function block variables can be carried out entirely within the INIT SP.
- All inputs and outputs are available in the Init SP.
- The Init SP does not monitor cycle time.

3.6.1 INIT VALUES in the Variable Declaration



Name	Type	Scope	Attribute	Value	Remark
TempAct	INT	global	memory	* remanent	
TempSet	INT	global	memory	300	
heat	BOOL	local	memory	* remanent	

Fig. 9.8: Initialization with Variable Declaration

As shown above, an initialization value can be entered in the VARIABLE or OUTPUTS in the "Init value" column.

The Initialization value is entered instead of:

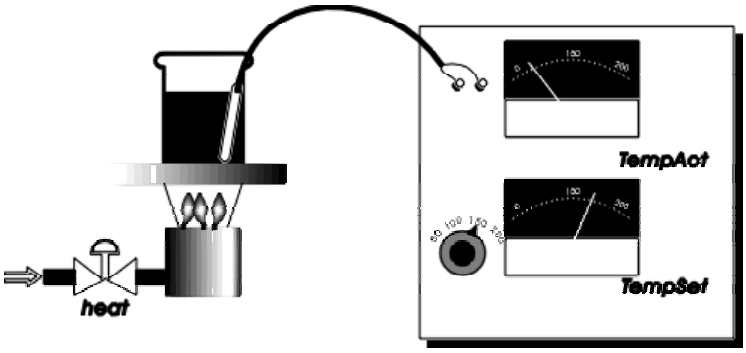
* remanent

Information The Init value is assigned to the PV before the INIT SP is executed !

Therefore the value assigned in the INIT SP is always valid!

Example The fluid in a beaker should be heated to a specific temperature using a gas burner.

If the "TempAct" is smaller than the preset temperature "TempSet", the burner "heat" must be activated.



Name	Type	Scope	Attribute	Value	Remark
Heat	.....	.....	.....	.....	Dig OUT, channel 3
TempAct	.....	.....	.....	.....	Analog IN, channel 4
TempSet	.....	.....	.....	.....	INTERNAL

Project name: proj_st
Task name: s_heat1

Resource: C#3

3.6.2 Defining Constants in Variable Declaration

Your machine or system has defined maximum and minimum values. These values are valid across the whole system and do not change after start up. These values are fixed in the program code.

Example: `IF TempSet > 300 ...`

The maximum value is 500 for a new system. You now have to change all program positions in all tasks. This takes a lot of time and effort.

This is why constants are used by most high level languages. Constants display values as text.

Example: `IF TempSet > TEMP_MAX ...`

Now only the value for the constant "TEMP_MAX" has to be changed in order to change all program sections. The entire project must then be compiled again and transferred.

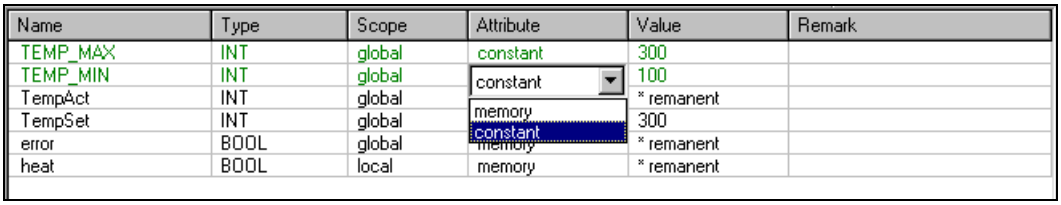


Fig. 9.9: Constant Definition in the Variable Declaration

The attribute can be changed from "memory" to "**constant**" and therefore a variable to a constant.

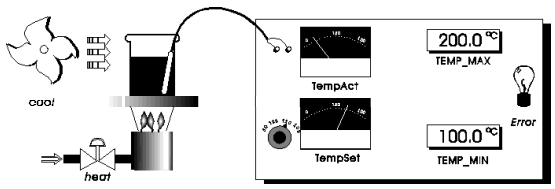
Information Constant values **cannot be changed** while the system is running.

If more than 2 different reactions are required by a condition, additional ELSIF branches can be added to the IF statement.

IF condition **THEN**
statement block
 	
ELSIF condition **THEN**
statement block
 	
ELSIF condition **THEN**
statement block
 	
ELSE
statement block
END_IF
	IF statement
	condition(1) is true
	
	condition(2) is true
	
	condition(3) is true
	
	all conditions are false
	End of the IF block

Information When ELSIF branching, plan the logic on paper first to avoid mistakes when programming!!!

Example The fluid in a beaker should be heated to a specific temperature using a gas burner.



The following bouondaries must be keep to:

Temp_SET > Temp_MAX -> Error

Temp_SET < Temp_MIN -> Error

In case of error: delete heat and cool!

Temp_ACT > Temp_SET -> Cooling

Temp_ACT < Temp_SET -> Heat

Name	Type	Scope	Attribute	Value	Remark
Error	.....	.....	.....	.....	Dig OUT, channel 4
heat	.....	.....	.....	.....	Dig OUT, channel 5
cool	.....	.....	.....	.....	Dig OUT, channel 6
TempAct	.....	.....	.....	.....	Anal IN, channel 1
TEMP_MAX	.....	.....	.....	.....	CONSTANT
TEMP_MIN	.....	.....	.....	.....	CONSTANT
TempSet	.....	.....	.....	.....	INTERNAL

Project name: proj_st
Task name: s_heat2
Resource: C#3

Additional Considerations:
Which variables shouldbe pre-initialized or defined using constants ?

3.7 Case Statement

CASE statements enable fast access to different actions depending on the value of the variable.

We will give a description of the case statement using the following example:

Case statement	Example	Description
CASE OF	`CASE Position OF`	Start Case Statement
1:	`1: Display := OVERVIEW;`	Only for first position
2,5:	`2,5: Display := NOT_SUPPORTED;`	Valid for pos. 2 **or** 5
6..10:	`6..10: Display := SETVALUE;`	Valid for pos. 6 **to** 10
11..20:	`11..20: Display := ACTVALUE;`	Valid for pos. 11 **to** 20
ELSE	`ELSE Display := ERROR;`	All other positions
END_CASE	`END_CASE`	End of the CASE statement

The **Expression** (between **CASE a**nd **OF**) must be **type UINT** and can have a value between **0 and 65535**!

Whole number values can be used as steps (UINT).

There are two possibilities to use several numbers in the same statement:

- Fields with **progressive number** (e.g. `6..10:`)
  There can only be two dots between numbers!
- Separate the numbers with a **comma** (e.g. `2,5:`)

A **colon** must be put after the step number.

The **ELSE** STATEMENT block processes all numbers that are not listed.

Only **one step per cycle** is processed.

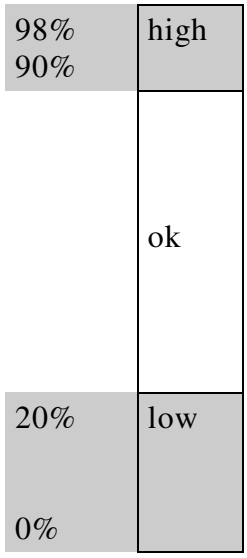
Example CASE statements are frequently used for step sequences. This example shows you how to use a CASE statement to carry out a simple comparison.

The contents of a beaker are tested against low, ok and high levels.

Use an output for each of the low, ok and high levels.

The level of liquid in the beaker is read by an analog value and is internally converted to a %.

An E-STOP should be triggered at 99 %



Name	Type	Scope	Attribute	Value	Remark

Project name: proj_st
Task name: s_case

Resource: C#4

3.8 Loops

Lopps enable one or more statements to be processed repeatedly depending on the condition. Three types of loops are defined in IEC1131-3.

Key Word	Program	Description

FOR
TO
BY
DO
 	
END_FOR
	FOR i := 0 **TO** 4 **BY** 2 **DO**
	
	    res := value + i;
	
	END_FOR
		Loop Start
		
		Loop End

Key Word	Program	Description

WHILE
DO
 	
END_WHILE
	i := 0;
	WHILE i <= 4 **DO**
	
	    res := value + i;
	    i := i + 1;
	END_WHILE
		Initialize Loop
		Start
		
		Loop End

Key Word	Program	Description

REPEAT
 	
UNTIL
END_REPEAT
	i := 0;
	REPEAT
	    res := value + i;
	    i := i + 1;
	UNTIL i > 4
	END_REPEAT
		Initialize Loop
		Start
		
		Loop End

NOTE **Take care to avoid creating endless loops as they will lead to a cycle time violation.**

Array

A normal PV can be seen as a box in which information can be entered and read from.

If you require several PVs, each PV must be separately defined and accessed.

An **array** is a **group of PVs with the same data type**. An array can be seen as group of safety deposit boxes (each box is the same, however the "contents" are always different). This allows the group to be accessed using a single name.

This short example shows the advantages of arrays which simplify the organization and display of large amounts of data.

An array is defined under length in the variable declaration.

Example: 5 weights are hanging on a crane.

We want to be able to access the weights using the PV names "Weight".

PV name	*Type[**Length**]*
Weight	INT[**5**]

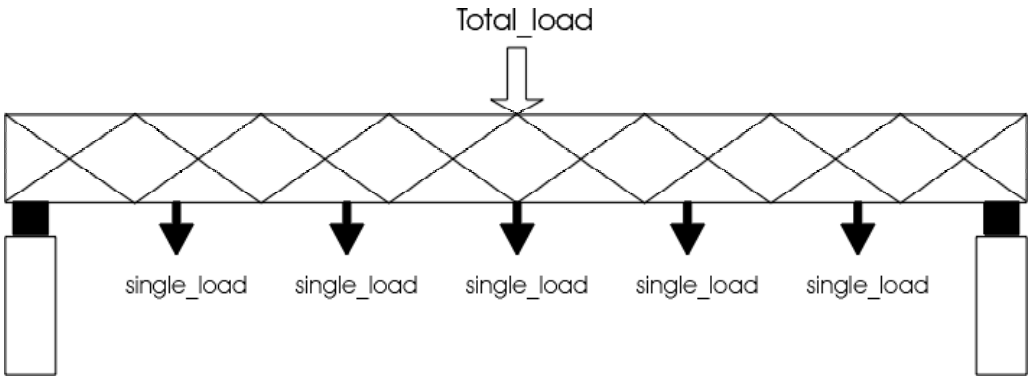
The groups of PVs can now be accessed with a common name "Weight". In order to access individual elements you still have to specify the index of the element required.

Variable name	Indizes				
	[0]	[1]	[2]	[3]	[4]
Weight	1000	4000	0500	0300	0020
	Weight[0]	Weight [1]	Weight [2]	Weight [3]	Weight [4]

Example Carry out the following task to illustrate the loop:

5 loads are suspended from a crane. To determine the total load you must add the individual loads together.

You can simulate the internal variables using PV monitor.



Project name: proj_st
Task name: s_last

Resource C#1

Additional Tasks:

- Create this example using **all** three **loop types**.
- Trigger a cycle time error in a new task with the help of a loop. An array is not required.
  (Task name: **s_runt; C#1**)
- Let the task run in task class Cyclic#3.

Important when using Arrays:

The length of a variable is assigned in the variable declaration:

Single variables: Length = 1
Array variables: Length > 1

The variable "Var" is defined as follows:

Name *Data type[Length]*

Var USINT[7]

Access to the "Var" array can look like this:

20	123	12	10	11	23	25	??
Var[0]	Var[1]	Var[2]	Var[3]	Var[4]	Var[5]	Var[6]	Var[7]

It is often overlooked when programming that an array begins with **Index 0**. The loop is developed up to the variable length defined (ex. Var, INT[7]), the undefined DPR areas (or others used) are read or overwritten.

3.9 Working with Function Blocks

Function blocks are called in the same way as commands. They are accessed directly by name. The enter and return variables are given in brackets (beginning with the first input).

Example: Create a switch on delay of 2 seconds.

```
(* Function call*)
Preset := 200;

TON_1(IN := Input, PT := Preset)

Output := TON_1.Q
```

or:

```
TON_1 (IN := Input, PT := 200)
```

The return parameter is returned to the hardware using standard procedures e.g.
Output := TON_1.Q;

Example Test TON

Then create a positive edge at an input with R_TRIG. Every time an edge is triggered, the counter increases by one.

Project name: proj_st
Program name: st_fbk

Resource: ?

4 SUMMARY

Operator Priorities:

Operation	Symbol	Priority
Brackets	()	highest
Function evaluation Example:	ID (argument list) LN(A), MAX(X)	
Potentiation	EXPT(IN1,IN2)	
Negation Complement	NOT	
Multiplication Division Modulo	* / MOD	
Addition Subtraction	+ -	
Compare	<, >, <=, >=	
Equal to Not equal to	= <>	
Boolean AND	AND	
Boolean Exclusive Or	XOR	
Boolean OR	OR	lowest

Logic Connections

Symbol	Logic Connections	Example
NOT	Negation	`a := NOT b;`
AND	logic AND	`a := b AND c;`
OR	logic OR	`a := b OR c;`
XOR	Exclusive OR	`a := b XOR c;`

Arithmetic Operations

Symbol	Arithmetic Operations	Example
:=	Assignment	`a := b;`
+	Addition	`a := b + c;`
-	Subtraction	`a := b - c;`
*****	Multiplication	`a := b * c;`
/	Division	`a := b / c;`
mod	whole number division rest	`a := b mod c;`

Implicit Data Type Conversion

Data type	BOOL	SINT	INT	DINT	USINT	UINT	UDINT	REAL
BOOL	BOOL	x	x	x	x	x	x	x
SINT	x	SINT	INT	DINT	USINT	UINT	UDINT	REAL
INT	x	INT	INT	DINT	INT	UINT	UDINT	REAL
DINT	x	DINT	DINT	DINT	DINT	DINT	UDINT	REAL
USINT	x	USINT	INT	DINT	USINT	USINT	UDINT	REAL
UINT	x	UINT	UINT	DINT	UINT	UINT	UINT	REAL
UDINT	x	UDINT	UDINT	UDINT	UDINT	UDINT	UDINT	REAL
REAL	x	REAL	REAL	REAL	REAL	REAL	REAL	REAL

× .... explicit conversion required depending on the data type

Explicit Data Type Conversion

Explicit Data Type Conversion carried out by STANDARD library functions.

The functions are constructed as follows.

x .. Output data type
y .. Target data type

x_TO_y e.g. **BOOL_TO_DINT**(bool_var)

Logic Compare Expressions

Symbol	Logic Compare Expressions	Example
=	Equal to	`IF a = b THEN ...`
<>	Not equal to	`IF a <> b THEN ...`
>	Greater than	`IF a > b THEN ...`
>=	Greater than or equal to	`IF a >= b THEN ...`
<	Less than	`IF a < b THEN ...`
<=	Less than or equal to	`IF a <= b THEN ...`

Conditions

Conditions	Example	Description
	`a := b * c;`	Calculation
IF		Start
THEN	`IF a > 0 THEN`	
	`result := 1;`	Condition fulfilled
ELSIF	`ELSIF a = 0 THEN`	Check against 2nd cond
	`Value := 100;`	
ELSE	`ELSE`	
	`result:= 0;`	no condition required
END_IF	`END_IF`	End of the entscheidun

Case Statement

Case	Example	Description
CASE OF	`CASE Position OF`	Start
1:	`1: Display := OVERVIEW;`	only for first position
2,5:	`2,5: Display := NoT_SUPPORTED;`	only valid for position 2 **or** 5
6..10:	`6..10: Display := SETVALUE;`	Valid for pos. 6 **to** 10
11..20:	`11..20: Display := ACTVALUE;`	Valid for pos. 11 **to** 20
ELSE	`ELSE Display := ERROR;`	All other positions
END_CASE	`END_CASE`	End of the CASE statement

Loops

Key Word	Program	Description

FOR
TO
BY
DO
 	
END_FOR
	FOR i := 0 **TO** 4 **BY** 2 **DO**
	
	    res := value + i;
	
	END_FOR
		Start
		
		End of Loop

Key Word	Program	Description

WHILE
DO
 	
END_WHILE
	i := 0;
	WHILE i <= 4 **DO**
	
	    res := value + i;
	    i := i + 1;
	END_WHILE
		Initialize Start
		
		End of Loop

Key Word	Program	Description

REPEAT
 	
UNTIL
END_REPEAT
	i := 0;
	REPEAT
	    res := value + i;
	    i := i + 1;
	UNTIL i > 4
	END_REPEAT
		Initialize Start
		
		End of Loop

AUTOMATION BASIC

1 GENERAL INFORMATION

1.1 Programming languages

B&R provides the ideal programming language for every application and programmer preference. The spectrum comprises:

- Ladder diagram (LAD)
- Instruction List (IL)
- Structured Text (ST)
- Sequential Function Chart (SFC)
- B&R Automation Basic (AB)
- ANSI C

LAD The functionality of contact, logic, and function plan are all combined in LAD. Due to its similarity with circuit diagrams ladder diagrams are the easiest and most visual form of digital and analog programming.

IL Instruction List is a machine based language that can be used to create logic connects in a similar way to LAD.

ST This high level language is an easy to understand, high performance programming language for automation systems. Standard construction guarantees fast and efficient programming.

SFC SFC is a programming language developed to dissect tasks into clearly organized sections. SFC is suited to processes reliant on a sequence of steps e.g. car washes.

AB **This is a B&R high level language. It is easy to understand, high performance programming language designed for the latest generation of automation systems. Standard construction guarantees fast and efficient programming. Previously known as PL2000.**

ANSI C High level language. High performance programming designed for the latest generation of automation systems. Standard construction guarantees fast and efficient programming.

SERVICE INFO

1 GENERAL INFORMATION

This chapter covers service techniques and programming options in further detail.

Buffer Concept

To protect the machine or system from data loss, the corresponding elements can be supplied in several ways.

Service Tools

Service technicians are often faced with the problem of users reporting machine or system failure but rarely giving any indication of the circumstances or possible causes of the failure.

Installing a New Operating System

Before a controller or APM can be used, an operating system must be installed.

Diagnosis without Programming Device

We want to examine the possible causes of errors without using AS. This information can be determined on site which can considerably speed up customer service by telephone.

Diagnosis with Programming Device

PG is required in order to access variables that are not displayed in the visualization. AS can evaluate all values available on the controller.

Project Management with Scheme

PopUp menus are provided to enable you to deactivate parts of the hardware as well as software. Variables that have a deactivated I/O assignment are transferred to the controller as "global" variables. This enables us to test the tasks that these variables use. Deactivated tasks are not transferred to the controller.

Debugging Possibilities

The SYSTEM monitor, LAD monitor and Watch monitor gives us a wide range of monitoring options. The line coverage function and the tracer can be followed precisely by the program on the controller, without influencing the controller. The debugger provides a tool to check the logical process of a program.

2 BUFFER CONCEPT

The following components are provided on PCC systems:

- RTC (Real Time Clock)
- DPR (Dual Ported RAM)
- USERRAM
- System RAM

Rechargeable batteries, lithium batteries as well as gold foil capacitors are also supplied depending on the system. External buffering is also possible.

2.1 Special features: B&R2010 / B&R2005 / B&R2003 / LS251

B&R2010 A **gold foil capacitor** serves as the short term buffer (min 10 minutes) in the case of power supply failure. Short term buffering guarantees problem free battery changes.
Long term buffering is carried out with a **lithium battery**. There is one battery in the CPU and one in the APM.

The MP100 multiprocessor does not have its own battery. This means that the APM battery must buffer the entire SRAM.

Data cannot be buffered on the IF100/IF101. The IF100 always boots after Power On with a coldstart. Data to be saved is stored on the CPU, or if this is not possible the data on the data module is saved in the FPROM.

B&R2005 In the event of power failure, the CPU tries first to do a short term buffer (min 10min) with the **gold foil capacitor** (if available). Long term buffering is carried out by a **rechargeable battery** on the CPU finally with a **lithium battery**.

The CP260 and IF260 are supplied for several months by an **internal accu**. After this time the **backplane** should be buffered by **battery**. The accu is only meant for short term buffering.

The CPU/PP module XP152 doesn't have its own buffer. The module is supplied by a **backplane with battery** or by the power supply with an **external buffer**. The battery module AC240 can be used for this. The **AC240** is equipped with 2 9V block batteries.

B&R2003, LS251
The CPU is buffered with a **lithium battery**!

2.2 Buffering The PCC for Standard Operation

The following buffer times have been calculated for standard operation:
Standard Operation: The PCC operates 8 hours a day, 5 days a week.

PCC System	Module	Buffering	Buffer Life (25 °C)	Buffer Life (40 °C)	Buffer Life (60 °C)
B&R2010	CP200+ME913	2*950 mAh Lithium Batt.	6 years	3 years	2 years
	CP100+ME910	2*950 mAh Lithium Batt.	9.5 years [1]	4.5 years	4 years
	CP100+ME913		7.5 years [1]	3.5 years	3 years
	MP100+ME910	1*950 mAh Lithium Batt.	5.5 years	2.5 years	2 years
	MP100+ME913		4 years	2 years	1.5 years
B&R 2005	CP260 IF260	NiCd rechargeable battery in Backplane	6 months 8 years [1]	2 months 4 years	n.V.
	CP15x+ME960	50 mAh rechargeable battery + 950 mAh Batt.	8 years [1]	5.5 years	5 years
	CP15x+ME963		7 years	5.5 years	5 years
	PS+Expansion + AC240 + XP152, IF152, IP151	2*9V extra Longlife Blockbatt.	4.5 years	2 years	1.5 years
B&R 2003	CP47x	950mAh Lit. Batt.	9 years [1]	4.5 years	1.5 years
B&R LS251	LS251	950mAh Lit. Batt.	8 years [1]	4 years	n.V.

The power requirements of the elements depends greatly on the temperature and the size of the RAM memory (see catalog).

IMPORTANT:

The life time of the batteries is 7 years at 45°C according to the data sheet. The life time of rechargeable batteries is 3 years at 45°C. After 3 years the battery has only got 70 % of its capacity left which it will lose in following years.

[1] The life time of the battery is 7 years according to the data sheet provided

2.3 Check the Buffer

The times given on the previous page are mainly helpful in determining when servicing is required. You must ensure you carry out a battery life check from time to time.

Check by Status LEDs

The status LEDs show you the buffer status.
The PCC carries out its own capacity test every minute.

Check by PCC Program (function)

The following function provides precise information about the status of the rechargeable battery or battery in the CPU or APM.

battery = SYS_battery()

Online Info: Buffering:

<right mouse button> on the Cpu in the hardware tree

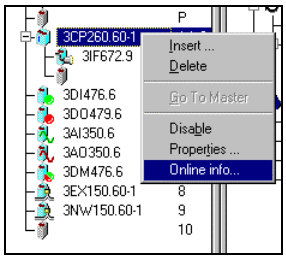


Fig. 11.1: Online Info

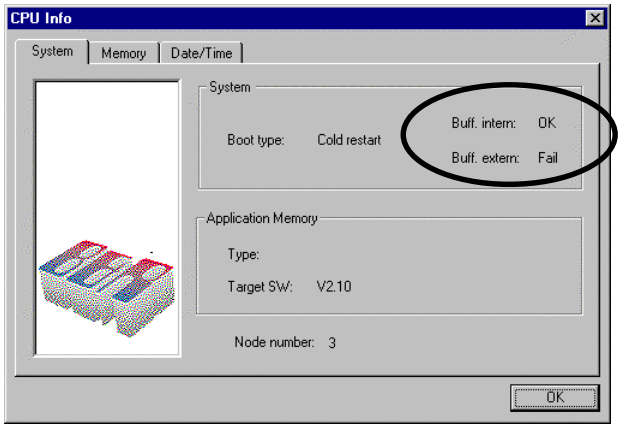


Fig. 11.2: Buffering Information

3 SERVICE TOOLS

Service technicians are often faced with the problem that a machine or system part failure is registered by the user without any information about the circumstances surrounding the failure or possible causes for the failure.

What questions can you ask to ascertain how the problem came about ?

- Was the system running? If yes, how long?
- Have you made any changes? Hardware or Software?
- Power cut ? (lightning hit, over voltage)
- Does the module receive the sensor signal?
  - Input Leds lit?
  - If yes, is the signal recognized by the CPU (Watch monitor)?
- Did the failure occur during operation or during system start up.
- What is shown on the display or status Leds?
- Has the configuration been changed (visualization, parameters)?
- Are the output Leds lit?
- Have the outputs changed?

3.1 Operating System

3.1.1 Reinstall PCC Operating System

If the service technician or programmer has to install an new APM or **new** XP152 / CP260 / IF260, the first set is to set up the PCCSW (operating system):

- Disconnect power to the PCC
- Check that the **Mode Switch** is set to **Boot** (if provided, 00 or sliding switch)
- Reconnect power
- Select menu: **Project: Services: Transfer Operating System**
- Select PCCSW version Vx.xx
- Set CAN settings to default
- The PCCSW is now ready
- Disconnect power to the CPU
- Set CPU Mode switch to **<> Boot** (> 00 or sliding switch)
- Turn on the power

Information: Since a new **LS251** is delivered with the operating system, you only have to check the **Mode switch**!

3.1.2 PCC Operating System Update

If the operating system V2.0 or higher is already available on the controller, a new operating system can easily be installed by selecting the menu option:

Project: Services: Transfer Operating System

3.1.3 PCC Operating System Update Mexxx (modular APMs):

- Disconnect power supply to PCC
- Push the APM cover off using your thumb.
  - Set switch to ERASE and replace APM in the CPU
- Write Protect switch -> Write Enable
- Power on
- Wait until the red LED on the APM is off
- Select menu point: **Project: Services: Transfer Operating System**
- Select PCCSW version Vx.xx
- Set CAN setting to default
- The PCCSW is now reset
- Disconnect power to the CPU
- Reset the APM switch from ERASE to OK and close the cover
- Switch on power -> The procedure is now complete

3.2 Diagnosis without Programming Device

The system provides the technician with a multitude of visual information in the form of status LEDs that can be accessed without the use of a programming device.

Standard LEDs available on all CPUs:

CPU	Ready LED	Lit	CPU running no problem. Also lit in Service Mode.
		Off	No operating system loaded. No APM installed
	Run LED	Lit	If the cyclic runtime system is active. Task classes run according to set cycle times. Also when no tasks are available.
		Off	Program Error System reset / stopped
	Error LED	Lit	CPU error, CPU is reset.
		Off	Run LED must be one.
	Mode LED (except CP15x)	Lit	When write access to Flash is made. Burn program or operating system.

LEDs dependent on CPU type (see HW manual).

Interfaces	Rx	blinking	Data being received
		Off	Nothing is being received
	Tx	blinking	Data is being sent
		Off	Nothing is being sent
	RS232	blinking	For CPUs where only one LED is available, only send and receive data is displayed.
	CAN	blinking	For CPUs where only one LED is available, only send and receive data is displayed.
	Force LED	Lit	Min 1 I/O has been forced (force relay on).
		Off	Not forced.

LEDs on I/O Cards

Digital module status LEDs	Input / Output	on	Input/output on (note circuit type: Sink/Source).
		off	Input/output off
Terminal Block	B&R2010	on	Terminal block not in module
		on	(check in program: ISP1.x.0,LSB = 1..terminal block missing).
		off	Terminal block installed in module (check in program: ISP1.x.0,LSB = 0..terminal block installed).

Analog module	RUN LED	on or blinking	AD/DA convertor working, module being accessed.
		off	Module not used (reserve) AD/DA convertor defective.
	Terminal Block B&R2010	on	Terminal block not in module
		off	Terminal block installed in module.

In order to automatically test the inputs, the test terminal must be installed where the outputs are connected. The outputs are switched on by the test task, read by the inputs and the result evaluated.

Visualization:

Visualization allows the PV's and the error module to be read if you program it to do that.

3.3 Diagnosis with Programming Device

You must access variables that are not displayed using PG. AS can evaluate all values that are available on the controller.

Variables and I/Os

In addition to the status LEDs on the module, the internal values of the individual process variables can be displayed and edited with the help of the Watch monitor.

Note: If the status LED of a digital input is lit and nothing is read from this channel (check PV in the Watch monitor), the channel is definitely defective. If the status LED is not lit and a value is read from the digital input, there is either a program error or the LED is faulty.

Analog Inputs/Outputs can only be checked by the Watch monitor.

Force: Used to set I/Os (digital and analog) to specific values. The force should be carried out in the Watch monitor (forced I/O indicated by a  in the [Force] column).

The force function is a very useful tool for testing I/Os. The output is set to a specific value. The result can either be read immediately on the output status LED, or measured using a specific measuring device.

If several I/Os are forced in a Watch monitor, they can be easily deleted using a menu option:

 **Object: Force All Off**

or right mouse button in **Force** menu

 **Project: Service: Warmstart:**

"Project: Service: Warmstart" also deletes all force tasks including those not shown in the Watch monitor.

3.3.1 System Logbook

All errors triggered by an EXCEPTION (system exception), are entered in the error logbook. If an exception is triggered, all outputs are switched off. The program on the PCC is stopped. In order to determine the cause of the error, select:

- **CPU select** in the hardware tree
- "**Log book**" tab in the software tree

A window is then opened which shows the last error in clear text. The top error is the most recent.

Some user access is recorded in the logbook while the program continues to run (e.g. clock change). These entries are recorded as WARNINGS.

Time	Error	Info	Module	Description
10.12.98 15:45:23.00	2075	16#00000000	Syss	Warning: Time/date changed

Time the CPU was changed.

Error descriptions can be obtained from AS Online help. Or user entries can be used in the error logbook:

If the operator exceeds defined limits, a user entry can be created and the CPU reset.

Time	Error	Info	Module	Extended information
??.??.?? ??:????.??	2222	16#12345678	????	Warning: ????????

In this case the service technician needs a reference list of user entries. The reference list also serves as an action list of tasks to be carried out.

3.4 Debugging Options

The LAD monitor, SYSTEM monitor and Watch monitor gives us mightily powerful monitoring possibilities. The line coverage function and tracer can trace the program on the controller with great precision without influencing the process. The debugger is also a powerful tool to examine the logical processing of a program.

System Monitor: Overview of the modules installed on the controller. System or application module.

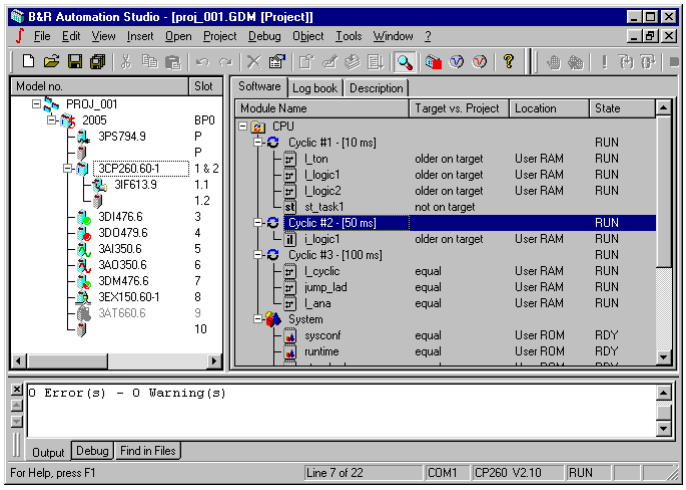


Fig. 11.3: System Monitor

LAD Monitor: Shows the status of the individual logic connections.

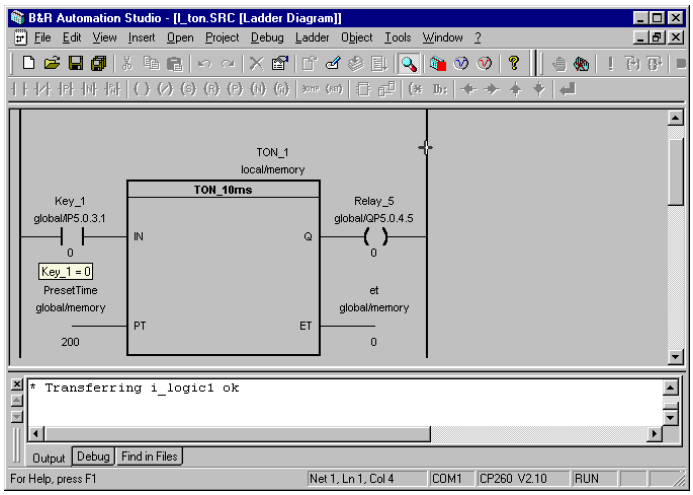


Fig. 11.4: LAD Monitor

Watch Monitor:

Tabular list of the different variables on the controller. Global and local variables for all tasks can be listed.

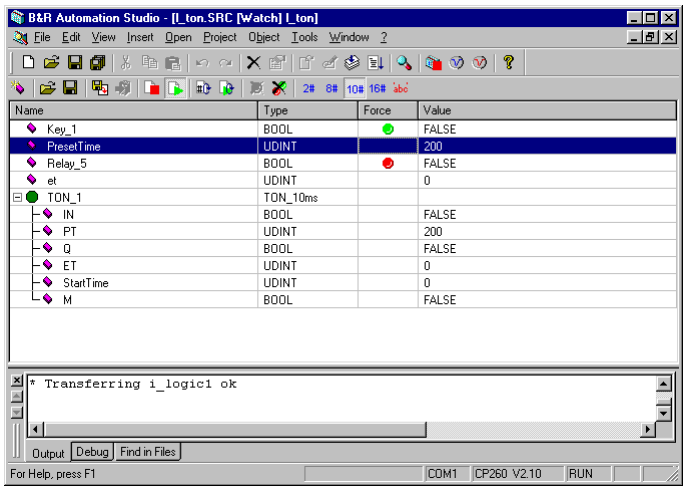


Fig. 11.5: Watch Monitor (PV Monitor)

Archive Mode:

The variable values on the controller are saved to hard disk and can be recalled later for further tests. The values can be changed, collated and transferred to the controller.

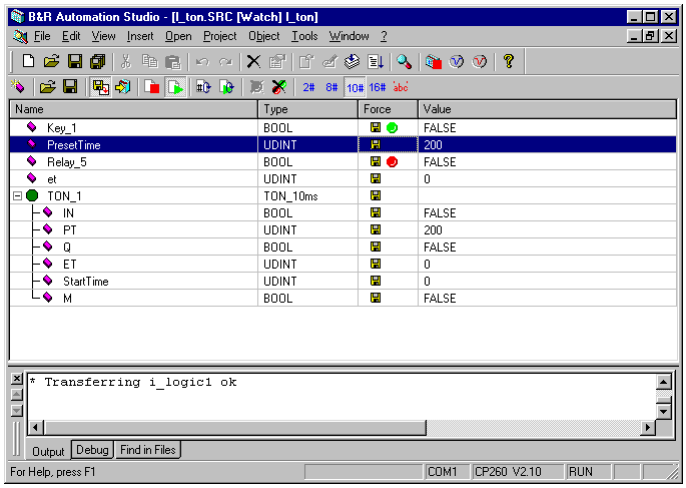


Fig. 11.6: Watch Archive Mode

Tracer: Provides real time images of the data on the controller. The values can be recorded at the start or end of the task.

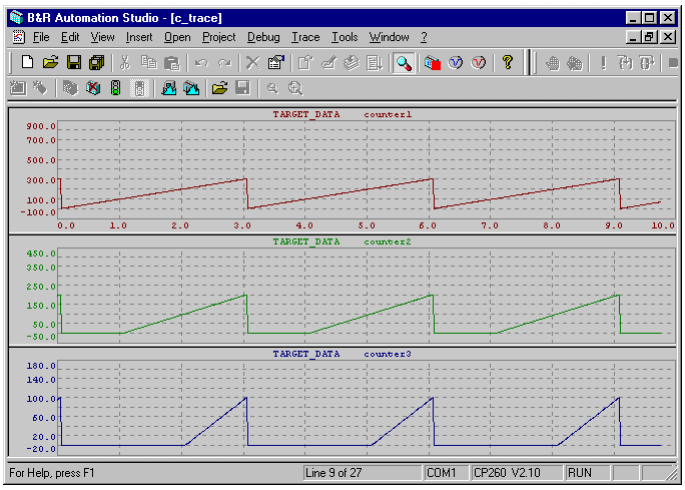


Fig. 11.7: Tracer

Line Coverage:

The line coverage function can be tested using the following C task.

```
/* ============================================================================
   #include is required to get the Standard Type Declaration.
============================================================================*/
#include <bur\plc.h>

_GLOBAL plcbit   heat;        /* The variables are defined as "GLOBAL"      */
_GLOBAL short    TempSet,     /* variables for our example.                 */
                 TempAct;

/* _LOCAL defines a local variable that should be stored in the DPR.        */
/* _GLOBAL defines a global variable that should be stored in the DPR. */


/* ============================================================================
    Initialization Subprogram: Started with _INIT.
============================================================================*/
_INIT void init(void)
{
}

/* ============================================================================
    Main program: Started with _CYCLIC.
============================================================================*/
_CYCLIC void cyclic(void)
{
if (TempAct < TempSet)
    {
        heat = 1;
    }
else
    {
        heat = 0;
    }

} /* end of cyclic() */
```

Line Coverage

Overview of program components currently being processed. The line coverage function shows a snapshot image.

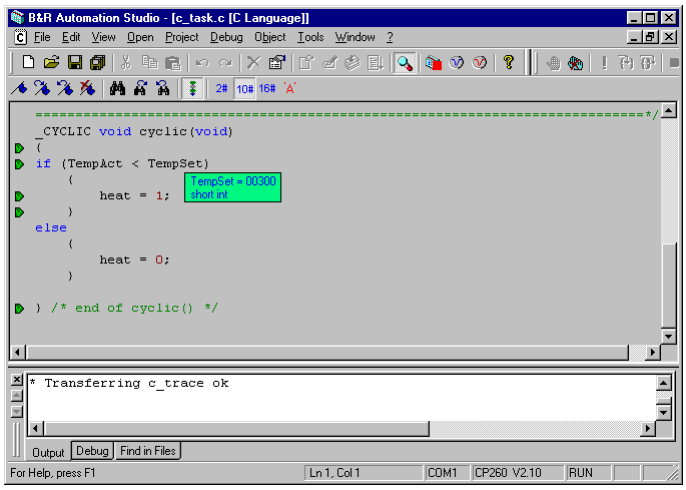


Fig. 11.8: Line Coverage

Debugger:

The debugger allows you to test the logical process of a program step by step.

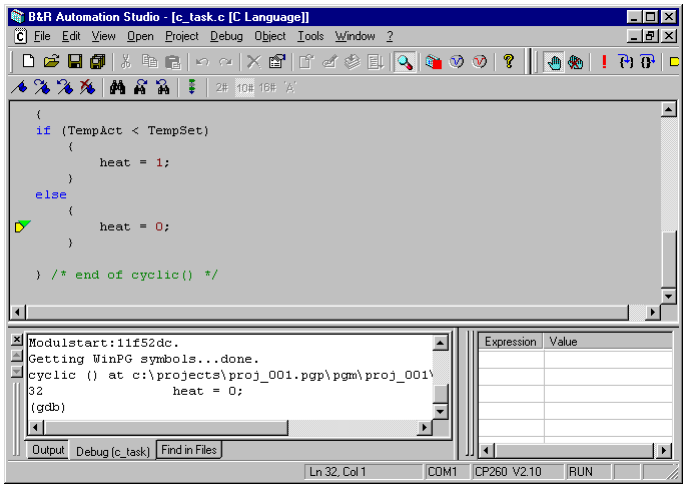


Fig. 11.9: Debugger

3.5 Project Management using Schemes

Parts of the hardware can be disabled using PopUp menus. Variables that have a deactivated I/O assignment are transferred to the controller as "global" variables. This enables us to test the tasks that use these variables. Deactivated tasks are **not** transferred to the controller.

Deactivate Hardware:

Click the right mouse button on the HW configuration to open the following PopUp menu.

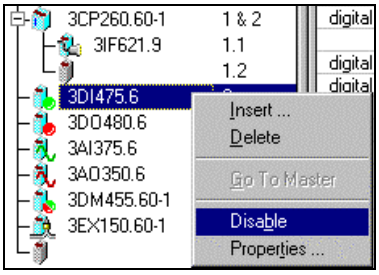


Fig. 11.10: Disable PopUp

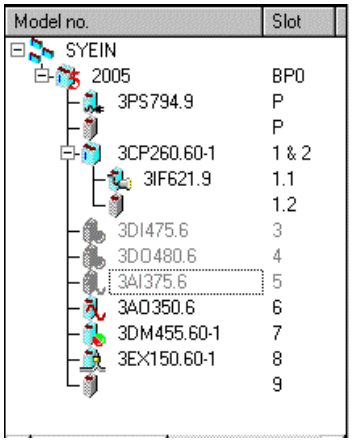


Fig. 11.11: Disabled Hardware

IMPORTANT:

If tasks that use the disabled hardware are transferred to the controller, the task variables are stored as "global" variables.

Software on the CPU can be tested without Hardware.

Deactivate Software

Object: Disable: Task is deactivated.

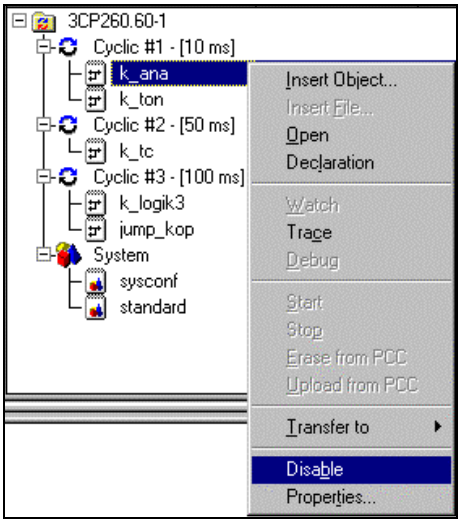


Fig. 11.12: SW Disable PopUp

Click on the task using the right mouse button. This PopUp allows you to disable a task. The task name is then shown in opaque text.

Save Scheme:

Project: Save Scheme:

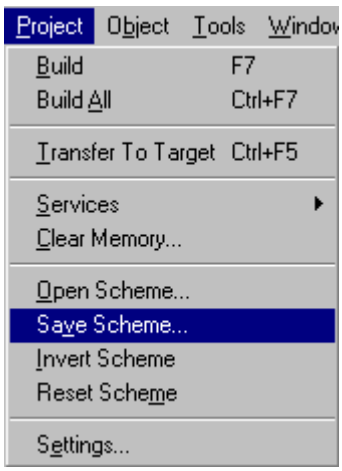


Fig. 11.13: Save Scheme

The hardware and software, deactivated by the "disable" command can be stored in a configuration.

"Open Scheme" opens an existing configuration.

"Invert Scheme" inverts the software configuration. This means that active tasks are deactivated and via versa.

"Reset Scheme" recalls all start parameters. All tasks are active.

Example Test project management using schemes

- Disable a few tasks and transfer the project to the controller
- Add some HW components. Now try to transfer the project. What happens?
- Disable HW that is not available and transfer it to the project.

3.6 Boot Mode

(see also System Software Reference Manual)

3.6.1 Normal Boot Procedure

- Delete Outputs:
  Deletes all outputs

- Sysconf:
  In the event of a "Warmstart" or "Coldstart", the configuration stored in USRROM is executed.

- Init Value Handling:
  All variable and output PVs are initialized with their "Init Value" (*remanent, set value).

- Enable Exception:
  Calls the INIT routines of all exception tasks and enables exception task classes. INIT routines can trigger exceptions whereby the exception task class must be enabled first.

- INIT routine handling:
  INIT routines of all other task classes are called as follows:
  Timer#1, Timer #2, Timer #3, Timer #4, Cyclic#1, Cyclic #2, Cyclic #3, Cyclic #4 and IRQTC.
  The INIT routines are processed within a task class in accordance to the sequence determined by the software tree.
  Attention: Cycle time monitoring is deactivated during INIT routine!
  Activate ready relay and set RUN LED.

- Cyclic Handling:
  Cyclic tasks are started as follows:
  Timer#1, Timer#2, Timer#3, Timer#4, Cyclic#1, Cyclic#2, Cyclic#3, Cyclic#4 and non-cyclic tasks (usertasks, communication).
  Tasks are processed within a task class in accordance to the sequence determined by the software tree.

3.6.2 Warmstart

- The system is initialized and started with the current data (initialization value or remanent value).
- All battery buffered data remains secure. Variable and outputs are initialized with the INIT value.
- The software remains unchanged and executed from the SRAM or USER ROM.
- **New System Module** (HW and/or SW) **not** recognized by warmstart.
- A warmstart is required to acknowledge a CPU error.

Trigger Warmstart:

-  **Project: Service: Warmstart**
- B&R2010: Select from "I" with mode button and then select button
- Turn power off then on again
- Call the function: "SYSreset(..)" with warmstart mode
- Change the key switch position from SERVICE to PROGRAM

Display recognition for B&R2010: "IN"

3.6.3 Coldstart

- The entire PCCSW is newly initialized.
  The SRAM is deleted. The system and project software is reconstructed by the USRROM and FIXRAM (optional MEMCARD).
- All data is deleted.
  Variables, CPU I/O, as well as the static area of global FBKs are initialized with the value 0.
- All installed system modules (SW system modules) are referenced and initialized.

Trigger Coldstart:

-  **Project: Services: Coldstart**
- B&R2005: Press the TOTALINIT button (depends on CPU)
- B&R2010: Select "T" with the mode button and then select button
- Change APM – A coldstart must be triggered after inserting the APM for safety reasons, this means that only non-volatile projects can be sent with a modular APM.
- Double RESET:
  Press the RESET button during the booting
- Call the function: "SYSreset(..)" with coldstart mode
- Change from diagnosis in the run mode

B&R2010 Display: "TI"

3.6.4 Reset/Watchdog

- The PCCSW remains unchanged and is reconstructed from the SRAM and USER ROM.
- All battery buffered data is accepted.
  Variables and outputs are initialized with the INIT value.
- Set system to the service mode which inactivates the project
- Entry in error logbook:
  WARNING Boot by WATCHDOG or manual reset

Trigger Reset/Watchdog:

-  **Project: Services: Stop Target**
- B&R2005, B&R2010 and LS251: Press the reset button
- Change the key switch position from PROGRAM to SERVICE
- Watchdog not accessed correctly by the system SW (checked every 200 msec)

B&R2010 Display: "RS – Service"

3.6.5 Error / Service

- If a fatal error occurs on the CPU it is recorded in the error logbook and displayed on the B&R2010 CPU status display.
- The CPU boots in error mode and goes into service mode.
- This enables error analysis to be carried out by Automation Studio.
- The error behavior is the same as in Reset/Watchdog mode.

Error Analysis:

- Error in boot or cyclic system
- User entry in error logbook with the function: "ERR_fatal(..)"

B&R2010 Display: "SERVICE"

3.6.6 Diagnosis

- Only firmware modules are initialized (corresponds to empty APM!).
- Modules in USRROM, FIXRAM, MEMCARD are not initialized.
- This means that errors caused by module burned to the USRROM, FIXRAM, MEMCARD, can be dealt with.
- The system is set in diagnosis mode.
- Booting is now only possible with a coldstart!
- USRROM, FIXRAM, MEMCARD can only be deleted in diagnosis mode!

Trigger Diagnosis Mode:

- **Project: Services: Diagnostics**
- Press coldstart (Total Init) button for approx. 5 sec until only the READY and ERROR LEDs are lit and the CPU boots in diagnosis mode
- B&R2005+B&R2003: Move Hex number switch to position "F" and PowerOff - PowerOn
- B&R2010: Simultaneously press <Cursor><top> + <Enter> keys. Then Power On.

B&R2010 Display: "DI – SERVICE"

PROJECT MANAGEMENT

1 GENERAL INFORMATION

The function of the B&R 2000 system is repeated and supplemented.

Project Organization

The individual Functions (tasks) that are to carry out one or more controls are summarized in the project.

Automation Desktop

The Automation desktop is what you use to program the B&R 2000 system. The functions range from "standard" task programming to expert modules.

Hardware Configuration

Hardware configuration determines the construction of the individual controls. The connections between variables and the physical I/Os is created here.

Software configuration

During software configuration the tasks are built. The workings of these tasks can be accurately defined using the properties dialog box.

2 PROJECT ORGANISATION

Individual functions (tasks) that carry out more than one control should be summarized in the project.

The project will be saved on to the hard disk as in the diagram below.

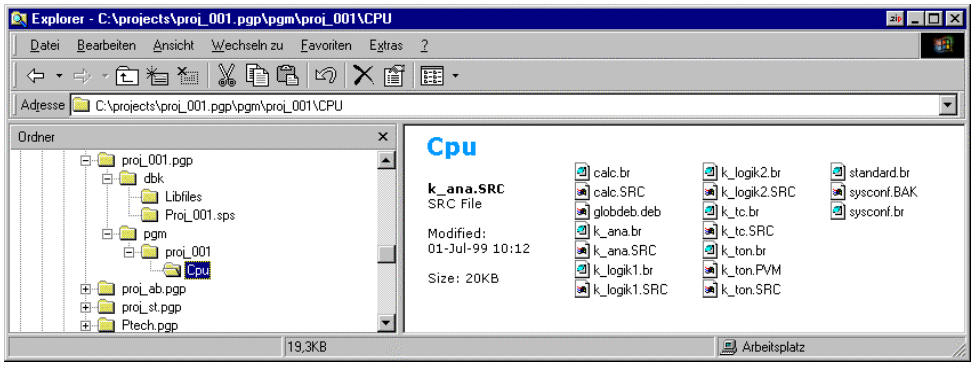


Fig. 12.1: AS directory structure

projects: General project directory. Customer names and details can be stored here since customers normally have several machine or system projects.

proj_001.pgp:
Directory in which all project data is saved (project name.PGP).

dbk: General database of AS. Information about the libraries. General settings.

Libfiles: Standard libraries relevant to the project are saved here.

proj_001.sps:
Database for the controller, hardware construction...

pgm: Path for the program module.

proj_001: Name for each base rack.

CPU: All tasks and relating settings are saved in this directory.

Data types/Extensions:

ID	Description	Type	Meaning
*.src	Source files	*.c	C Source Code
*.br	Executable files	*.h	Header file for C
*.pvm	Watch settings	*.bak	Backup file

3 AUTOMATION STUDIO

Automation desktop is the tool with which the B&R system 2000 can be programmed. The functions range from "normal" task programming to up to the expert modules.

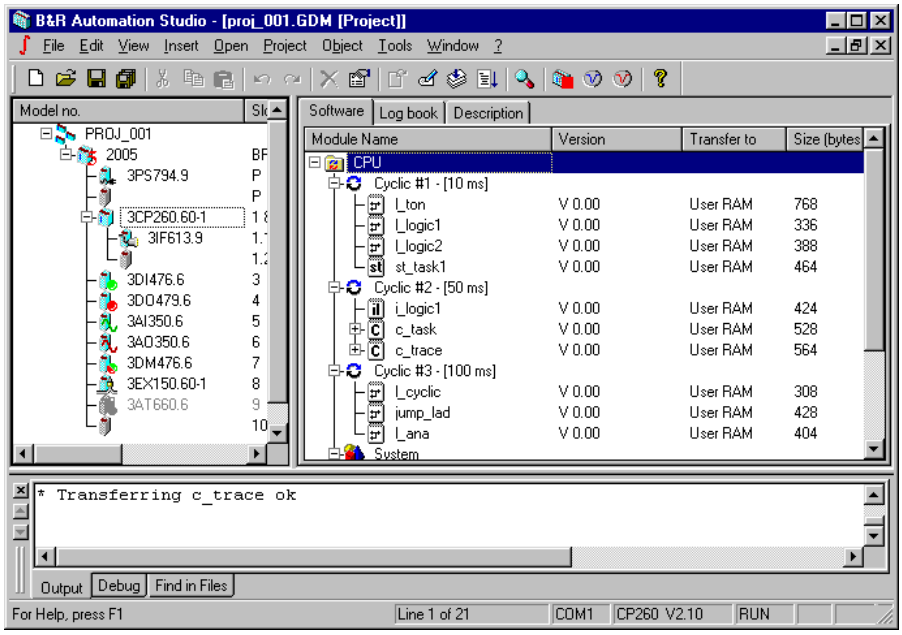


Fig. 12.2: Automation Desktop

Menus, button lists and pop up menus are at the users disposal.

The project hardware is setup for the in the left window (hardware configuration). While creating the project the user can automatically load for the controller.

The right window represents the software configuration in which the settings for all resources and tasks can be carried out.

The message box at the bottom of the screen alters the user to possible errors or compiler messages.

4 HARDWARE CONFIGURATION

The hardware configuration serves to construct the individual controls. In addition the connections between the global variables and the physical I/Os are made.

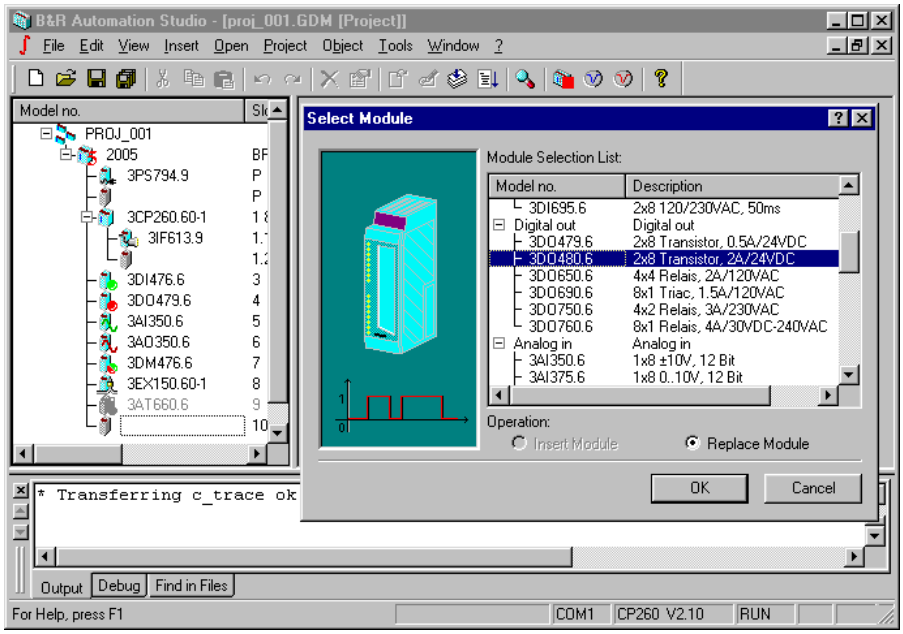


Fig. 12.3: Inserting a Module

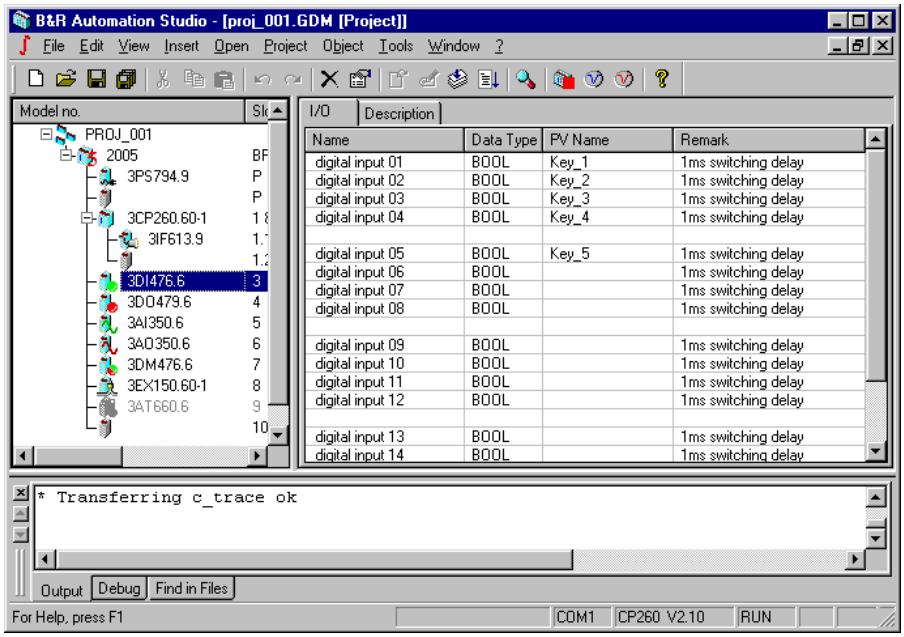


Fig. 12.4: Variable classification

5 SOFTWARE CONFIGURATION

The software configuration enables the user to accurately set the workings of the various resources.

5.1 System Settings

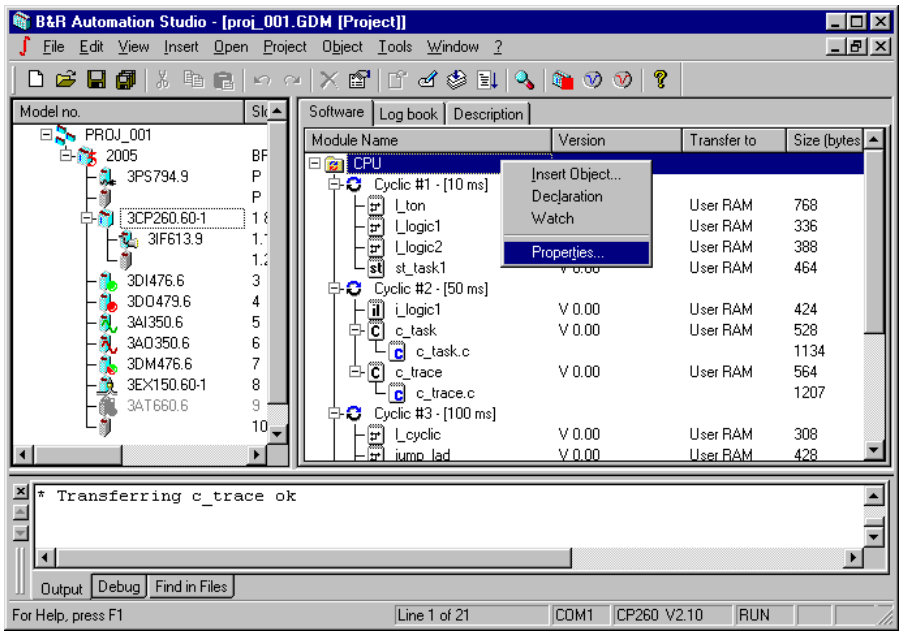


Fig. 12.5: System Properties

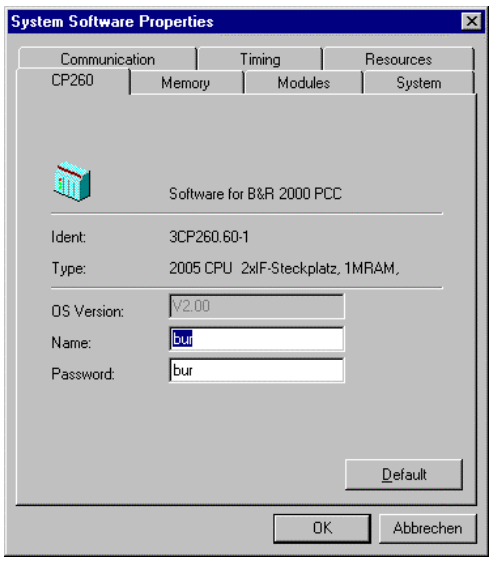


Fig. 12.6: System Configuration

Points can be altered here that will affect the whole controller. Programming techniques will be explained and illustrated during the training.

5.2 Task Class Settings

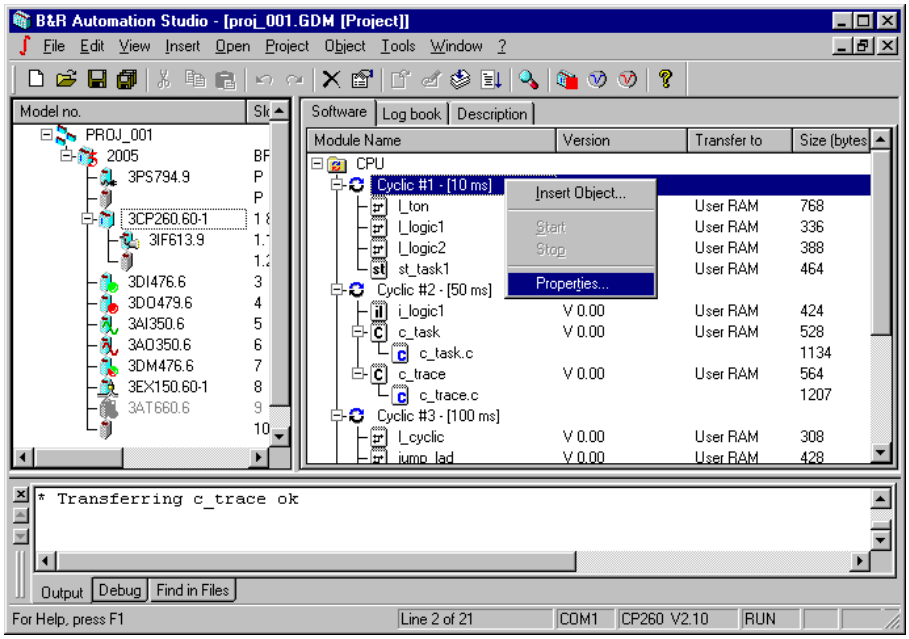


Fig. 12.7: Task Class Properties

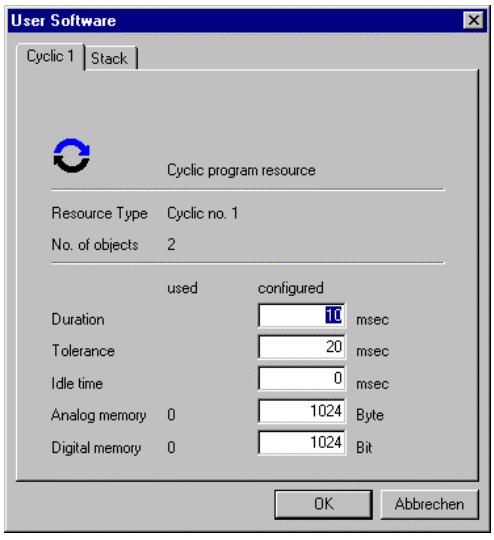


Fig. 12.8: Task class configuration

This dialog box enables the user to change the parameters of the task class. This is very important when it comes to optimizing a cycle time at the end of a project.

5.3 Task Properties

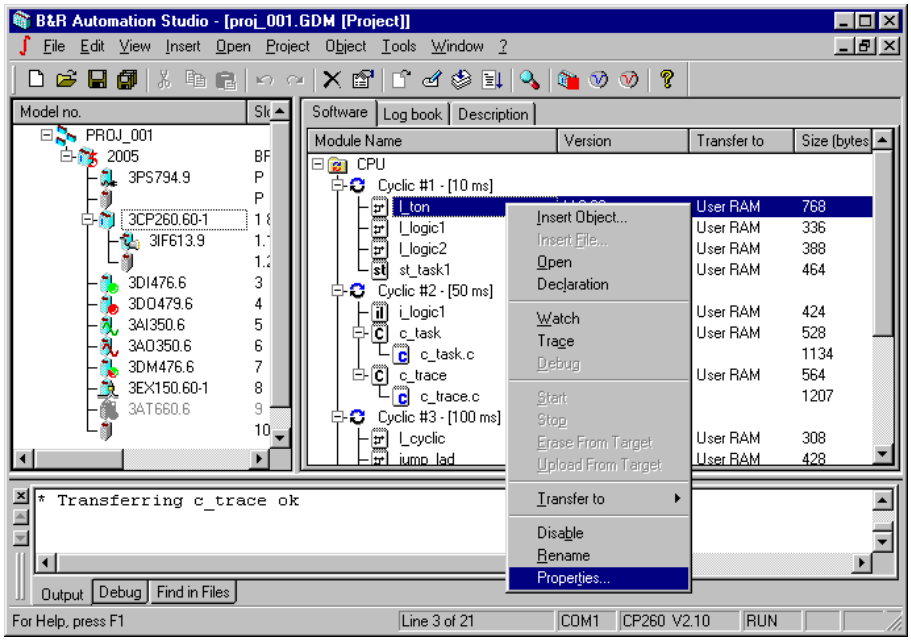


Fig. 12.9: Task Properties

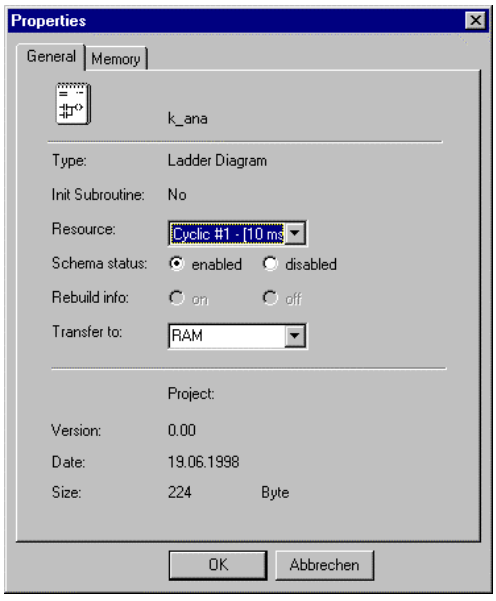


Fig. 12.10: Task Configuration

Dialog box for the Task parameter. For the configuration of the task class membership.

The "**Transfer to**" is especially important. When a task is tested it should not be downloaded in to RAM but into **USER ROM**. Only then is the program secured against power cuts.

5.4 Project saving

5.4.1 Saving mediums

SRAM, FIXRAM, FPROM or **MEMCARD** are all available as saving mediums on the controller.

SRAM The SRAM section of the APM is used as memory during **program setting**.

Transferring of the object occurs via :

Project: Transfer To Target

The individual objects are saved in their operational names on to the read only memory.

After a **coldstart** the entire **SRAM** is deleted!

Buffering of the SRAM is as follows: (see buffer concept)

- Rechargeable battery
- Battery

FIXRAM FIXRAM is part of the SRAM's, which retains its data after a cold start. A section (16, 32 k Byte, ..) of the SRAM can be defined as FIXRAM via the dialog box "System Property" (Fig. 8.5 and 8.6).

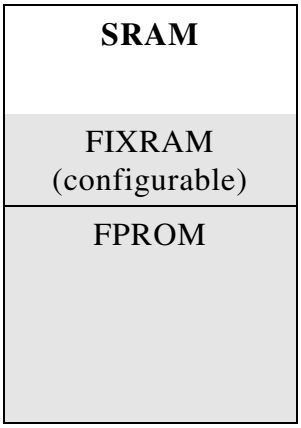
Data modules are saved in FIXRAX which have been changed by tasks (data acquisition, new constructions or machine parameters....)

To save objects in FIXRAM it is necessary to set parameters (when dealing with task parameters) the following point:

Transfer to: **FIXRAM**

After which the project must be downloaded to the controller.

Project: Transfer To Target



In FIXRAM objects are saved in the following way:

< Delete >

Object: Erase from PCC

Project: Clear Memory: Erase FIXRAM

FPROM The FPROM section of the USER ROM is used as a memory for **fully tested projects**.

To save objects in PROM it is necessary to set parameters for "Task Property" (Fig. 8.9 and 8.10):

Transfer to: **USERROM**

When **deleting from the PROM section or repeatedly burning,** the old object is rendered unrecognizable.

APM - PROM

~~Task1~~
Task2
Task1

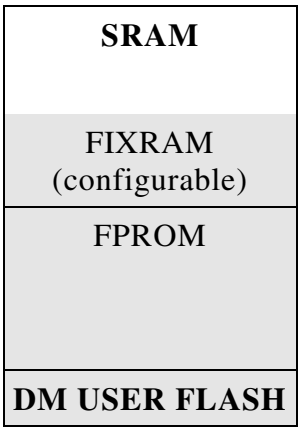
FPROM can be deleted directly from AS:

Project: Clear Memory: Erase FLASH

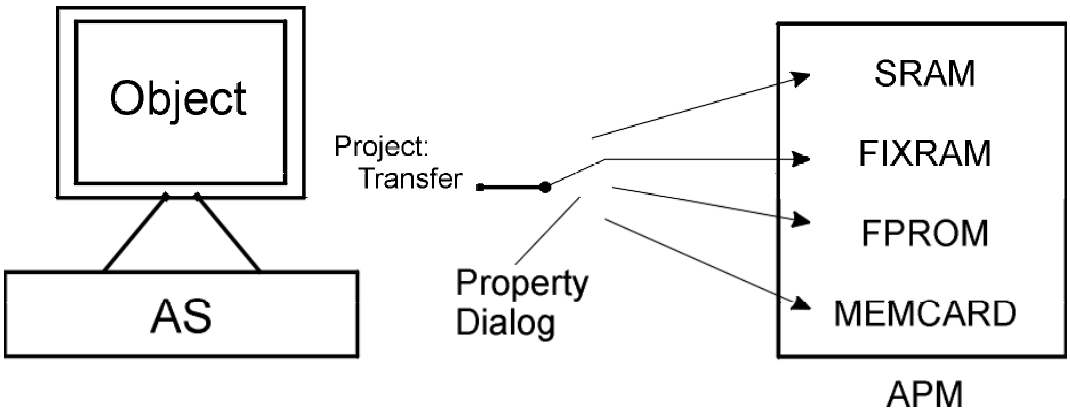
MEMCARD The **MEMCARD** for the CP260 operates exactly like the FPROM.

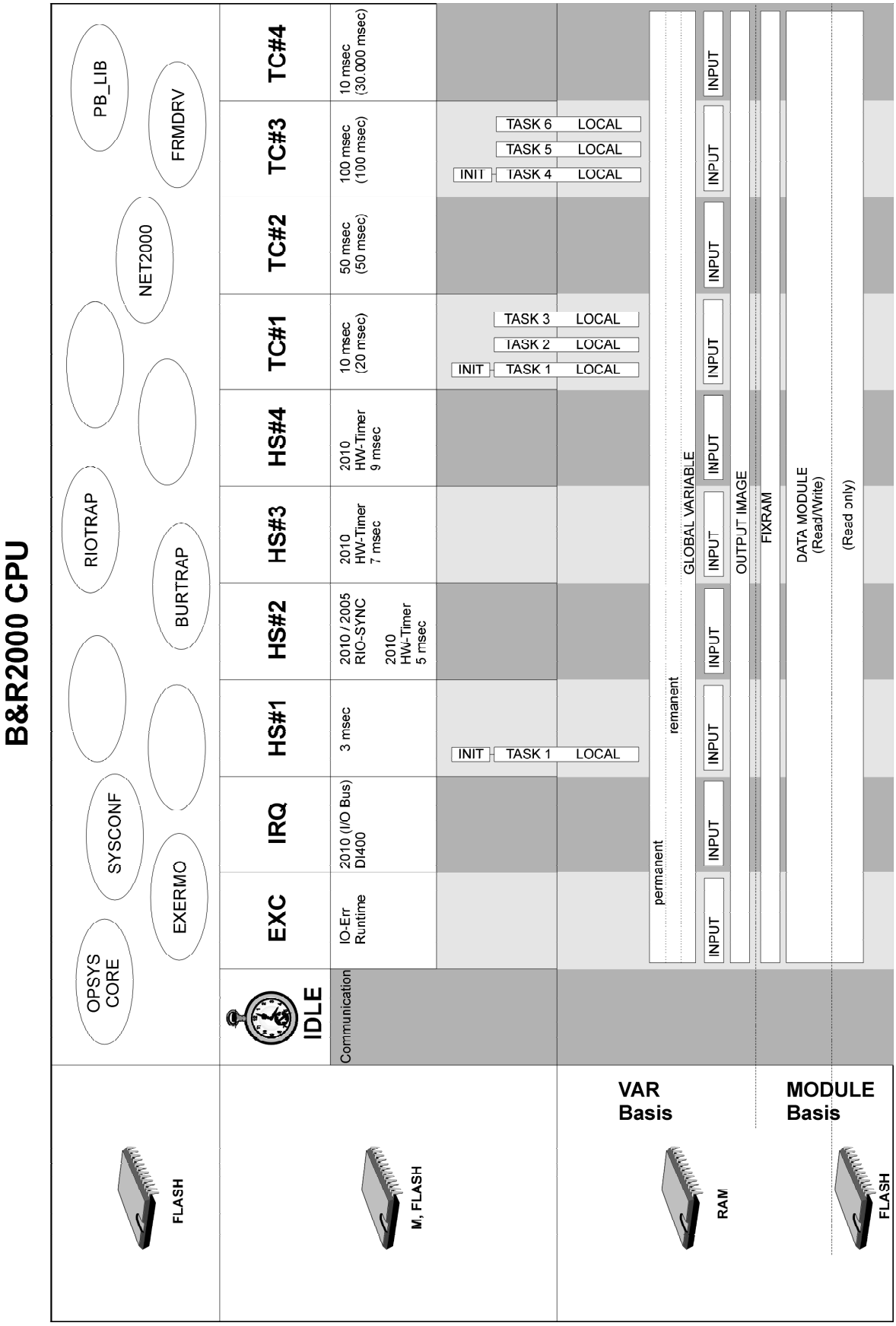
Data Module **USER FLASH**

- The **64 kByte** area DM USER FLASH can be **deleted and written with function blocks**, without affecting the rest of the flash area.
- This makes it possible to create and manage **data modules** in nicht flüchtigen memory using programs.
- The DM USER area is **available** starting from **PCCSW V2.0**.
- The FIXRAM is therefore not as relevant as before.
- **DM USER FLASH is not available with the B&R2010 and CP15x!!**



5.4.2 Memory Mediums Summary:




B&R2000 CPU
OPSYS CORE
EXERMO
SYSCONF
RIOTRAP
BURTRAP
NET2000
FRMDRV
PB_LIB
EXC
IRQ
HS#1
HS#2
HS#3
HS#4
TC#1
TC#2
TC#3
TC#4
IDLE
Communication
IO-Err Runtime
2010 (I/O Bus) DI400
3 msec
2010 / 2005 RIO-SYNC
2010 HW-Timer 5 msec
2010 HW-Timer 7 msec
2010 HW-Timer 9 msec
10 msec (20 msec)
50 msec (50 msec)
100 msec (100 msec)
10 msec (30.000 msec)
INIT TASK 1 LOCAL
INIT TASK 1 LOCAL
TASK 2 LOCAL
TASK 3 LOCAL
INIT TASK 4 LOCAL
TASK 5 LOCAL
TASK 6 LOCAL
permanent
remanent
GLOBAL VARIABLE
INPUT
OUTPUT IMAGE
FIXRAM
DATA MODULE (Read/Write)
(Read only)
FLASH
M, FLASH
VAR Basis
RAM
MODULE Basis
FLASH

SEMINAR REVIEW

1 SEMINAR REVIEW

- Introducing B&R
- System Overview
- Hardware
- Programming System
- B&R2000
- Ladder Diagram
- Instruction List
- Structured Text
- Automation Basic
- Service Info
- Project Management

2 SEMINAR OVERVIEW

B&R2000

POSITIONING

VISUALIZATION

MULTI

3 SALES AND SUPPORT LOCATIONS

For up to date addresses and product information look up our web pages:

HTTP://WWW.BR-AUTOMATION.COM