

Operating Manual

English



ML70B

Hottinger Baldwin Messtechnik GmbH
Im Tiefen See 45
D-64293 Darmstadt
Tel. +49 6151 803-0
Fax +49 6151 803-9100
info@hbm.com
www.hbm.com

Mat.: 7-2002.0573
DVS: A00863_05_E00_00 HBM: public
07.2018

© Hottinger Baldwin Messtechnik GmbH.

Subject to modifications.
All product descriptions are for general information only.
They are not to be understood as a guarantee of quality or
durability.

1	Safety instructions	5
2	Markings used	7
2.1	The markings used in this document	7
3	Introduction	8
4	Front panel	10
5	Connection boards	11
5.1	Connection Board AP71	13
5.2	Connection Board AP72	14
5.3	Connection Board AP75	15
5.4	Connection Board AP78	16
6	ML70B working methods	17
6.1	Time response	17
6.2	Data transmission	18
6.3	Program architecture	21
6.4	Saving programs	22
6.5	PLC_PRG main program	23
7	Introduction to programming	25
7.1	Installing the programming system	26
7.2	Program example	27
7.3	Translating the project	33
7.4	Starting the target system and loading the program	34
8	Communication with other amplifier channels	35
8.1	Loading measured values	35
8.2	Sending commands	36
9	Configuring hardware components	37
9.1	Analog outputs	37
9.2	Front panel LEDs	37

9.3	Outputting computed values	37
10	Connection board control	39
10.1	AP71/CAN	39
10.1.1	Basic CAN driver function	39
10.2	Serial communication (AP72)	42
10.2.1	Sending data	42
10.2.2	Receiving data	43
10.3	Digital inputs and outputs (AP75)	45
10.4	AP78 analog outputs	46
11	Generating dialogs	47
11.1	Dialog structure	47
11.2	Parameter properties	49
11.2.1	Properties of the NODE parameter type	49
11.2.2	Properties of the DINTPAR, INTPAR and REALPAR parameter types . 50	
11.2.3	Properties of the KEY parameter type	51
11.2.4	Properties of the TEXT parameter type	51
11.2.5	Properties of the MENUE parameter type	52
12	Saving setup parameters	57
13	Multi-program mode	61
14	Special cases	62
14.1	Equidistant measurement output	62
14.2	Changing the number of subchannels	62
15	Debug functions (PLC browser)	63
16	System variables	65
17	Error messages	66

1 Safety instructions

Appropriate use

The ML70B programmable module is to be used exclusively for measurement tasks and directly related control tasks. Use for any purpose other than the above shall be deemed to be not in accordance with the regulations.

To ensure safe operation, the device may only be operated in accordance with the information given in the Operating Manual. It is also essential to comply with the legal and safety requirements for the application concerned during use. The same applies to the use of accessories.

General dangers of failing to follow the safety instructions

The ML70B module complies with the state of the art and is fail-safe. The device may give rise to further dangers if it is inappropriately installed and operated by untrained personnel.

Any person instructed to carry out installation, commissioning, maintenance or repair of the device must have read and understood the Operating Manual and in particular the technical safety instructions.

Remaining dangers

The scope of performance and supply of the ML70B only covers part of the measurement technology range. In addition, equipment planners, installers and operators should plan, implement and respond to the safety engineering considerations of measurement technique in such a way as to minimize remaining dangers. Prevailing regulations must be complied with at all times. There must be reference to the remaining dangers connected with measurement technique.

Working safely

Error messages must only be acknowledged when the cause of the error has been removed and no further danger exists.

The device complies with the safety requirements of DIN EN 61010-Part 1 (VDE 0411-Part 1); Protection Class I.

To ensure adequate immunity from interference, use only *Greenline* shielded ducting (see HBM offprint "Greenline shielding design, EMC-compliant measuring cable; G36.35.0)

Conversions and modifications

No modifications that affect the design or the technical safety of the ML70B module may be carried out without our express agreement. Any modification shall exclude all liability on our part for any resulting damage.

In particular any repair and soldering work on motherboards is prohibited. When exchanging complete modules, use only original parts from HBM.

Qualified personnel

This instrument must only to be installed and used by qualified personnel, strictly in accordance with the technical data and the safety requirements and regulations listed below. It is also essential to comply with the legal and safety requirements for the application concerned during use. The same applies to the use of accessories.





Qualified personnel means persons entrusted with the installation, assembly, commissioning and operation of the product who possess the appropriate qualifications for their function.

Maintenance and repair work on an open device with the power on must only be carried out by trained personnel who are aware of the danger involved.

2 Markings used

2.1 The markings used in this document

Important instructions for your safety are specifically identified. It is essential to follow these instructions in order to prevent accidents and damage to property.

Symbol	Significance
 CAUTION	This marking warns of a <i>potentially</i> dangerous situation in which failure to comply with safety requirements <i>can</i> result in slight or moderate physical injury.
Notice	This marking draws your attention to a situation in which failure to comply with safety requirements <i>can</i> lead to damage to property.
 Important	This marking draws your attention to <i>important</i> information about the product or about handling the product.
 Tip	This marking indicates application tips or other information that is useful to you.
 Information	This marking draws your attention to information about the product or about handling the product.
<i>Emphasis</i> <i>See....</i>	Italics are used to emphasize and highlight text and references to other chapters and external documents.
Device -> New	Bold is used to emphasize and highlight menu options and dialog or window titles. The arrows between menu choices indicate the order in which the menu/sub menus have to be opened/executed.
<i>Input</i>	Bold-italic is used to indicate necessary inputs and entry fields in programs.

3 Introduction

The ML70B programmable module is a 4-division wide module occupying one slot in the MGCplus system device. The module can be programmed as required by the “CoDeSys” programming system in accordance with the internationally standardized PLC programming standard IEC61131-3.

What can the ML70B do?

- Process measured values from any other MGCplus modules at a sampling rate of 2400 Hz, including data that is loaded into the system device via the CAN bus or Profibus.
- Send commands to other amplifiers.
- Output computed values as measured values in MGCplus that can be displayed and further processed by the AB22A display and control panel and by the MGCplus Assistant.
- Control up to two connection boards of the AP71, AP72, AP75 and AP78 types (for combination options, see *chapter 5 “Connection boards”, page 11*).

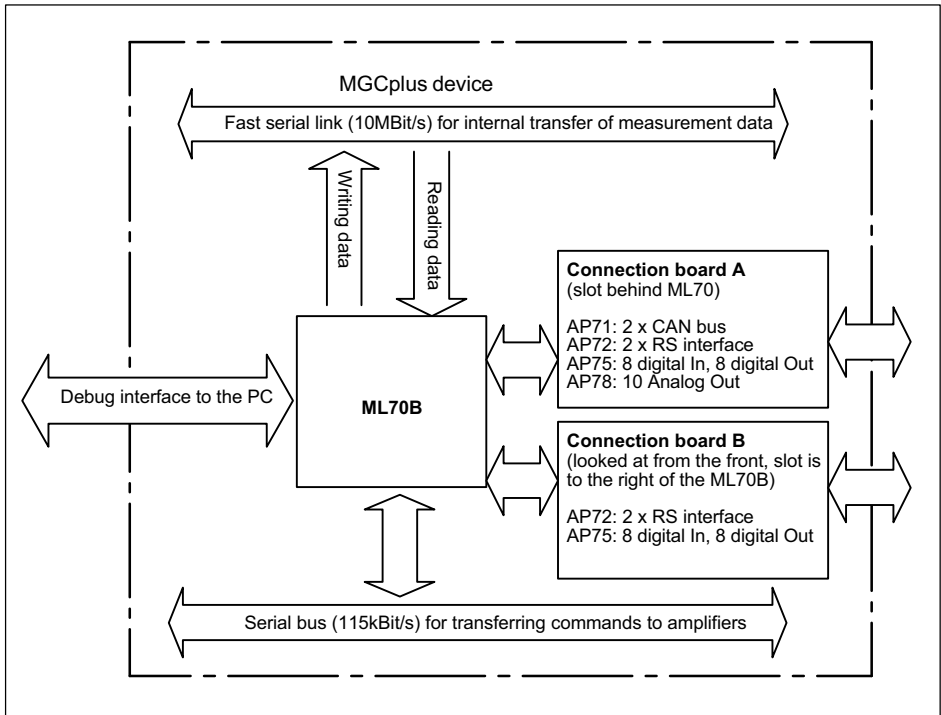
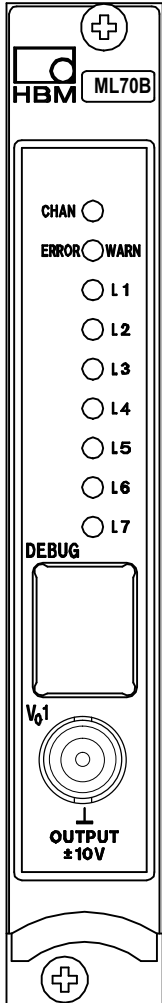


Fig. 3.1 ML70B in the MGCplus device

4 Front panel



Caption	Color	Meaning
CHAN.	Yellow	Channel selected
ERROR/WARN	Red	Error
L1	Red, yellow	As per programming
L2	Red, yellow	As per programming
L3	Red, yellow	As per programming
L4	Red, yellow	As per programming
L5	Red, yellow	As per programming
L6	Red, yellow	As per programming
L7	Red, yellow	As per programming

LEDs L1...L7 are programmable (see *chapter 9.2 "Front panel LEDs", Page 37*).

5 Connection boards

The ML70B can control up to two of the following connection boards:

- AP71 (2 x CAN bus)
- AP72 (2 x RS232/RS485/RS422; individually software-switchable)
- AP75 (8 x Digital-In, 8 x Digital-Out)
- AP78 (8 x Analog Out)

Each of the connection boards listed can be controlled individually. The following combinations are possible:

- AP71, AP72
- AP71, AP75
- AP72, AP72
- AP72, AP75
- AP75, AP72
- AP75, AP75
- AP78, AP72
- AP78, AP75

Connection boards can be plugged in directly behind the amplifier module or to the right of it, as seen from above (see Fig. 5.1).



Information

Only the connection board fitted behind the ML70B module has analog output voltages V_{O1} and V_{O2} !

The outputs from the AP78 connection board are tagged with AO (analog output) in the setup menus.

If the AP75 connection board is fitted directly behind the module, the I/Os are tagged with A. If it is fitted on the right, the I/O numbers are prefixed with B..

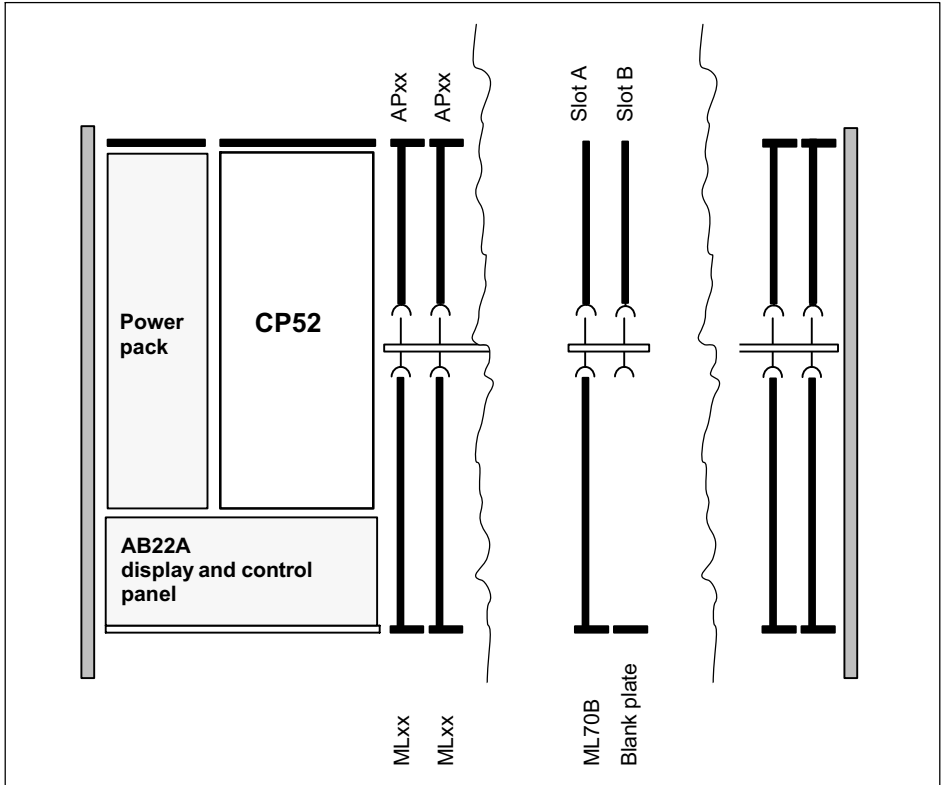


Fig. 5.1 Connection board combinations in the system device (top view)

5.1 Connection Board AP71

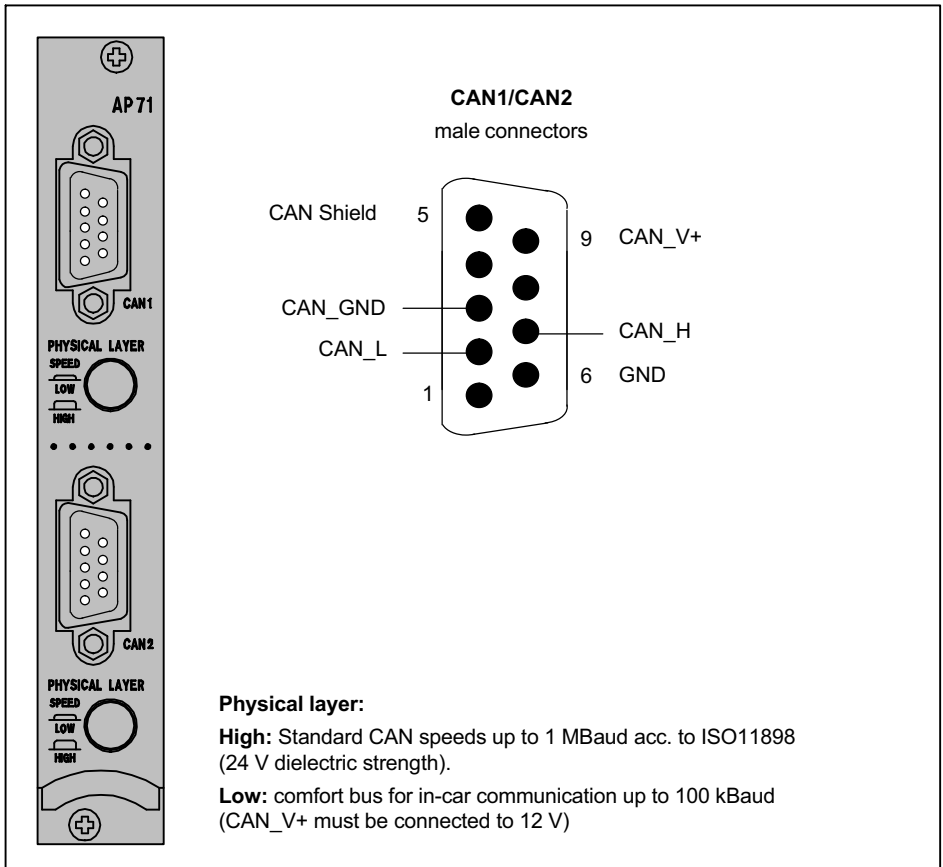
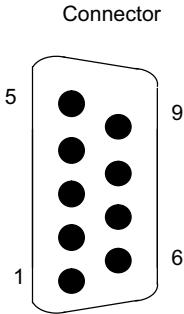


Fig. 5.2 Pin assignment AP71

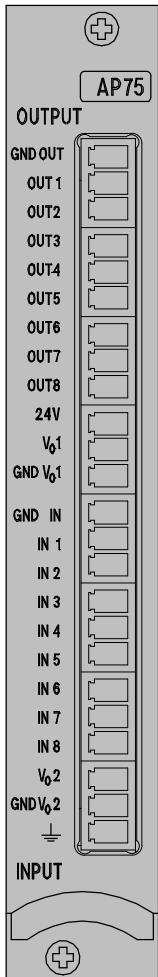
5.2 Connection Board AP72



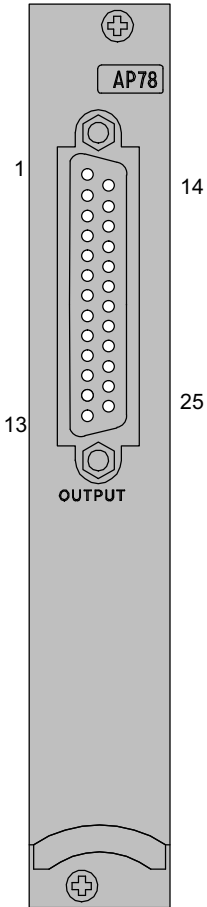
Pin	Function RS232	Function RS422	Function RS485
1	CD (not used)	-	-
2	RXD – Receive Data	RXA – Receive Data A (non inverse)	-
3	TXD – Transmit Data	TXB – Transmit Data B (inverse)	RXB – Receive Data B (inverse)
4	DTR –12V via series resistor	-	-
5	GND	GND	GND
6	DSR (not used)	-	-
7	RTS – Request To Send	TXA –Transmit Data A (non inverse)	RXA – Receive Data A (non inverse)
8	CTS – Clear To Send	RXB – Receive Data B (inverse)	-
9	RI (not used)	-	-

5.3 Connection Board AP75

An AP75 connection board has eight digital inputs and eight digital outputs. Inputs and outputs are electrically isolated and have their own grounding system (GND OUT: ground for outputs; GND IN: ground for inputs).



5.4 Connection Board AP78



Pin	Function
1	Connection board analogue output AO3
2	Connection board analogue output AO4
3	Connection board analogue output AO5
4	Connection board analogue output AO6
5	Connection board analogue output AO7
6	Connection board analogue output AO8
7	Connection board analogue output AO9
8	Connection board analogue output AO10
9	-
10	-
11	Amplifier analogue output VO1
12	Amplifier analogue output VO2
13	-
14	GND to AO3
15	GND to AO4
16	GND to AO5
17	GND to AO6
18	GND to AO7
19	GND to AO8
20	GND to AO9
21	GND to AO10
22	-
23	-
24	GND to VO1
25	GND to VO2

6 ML70B working methods

6.1 Time response

The ML70B has the following tasks:

- Loading the measured values
- Executing the loaded program
- Operating the connection boards
- Communication with MGCplus components (CP42 or CP52, amplifier channels, ext. computer, etc.)

These tasks are processed cyclically (see Fig. 6.1). Communication runs continuously in the background.

Loading of the measured values is interrupt-driven at a frequency of 2400 Hz. Measurement data also accumulates at this rate internally in the MGCplus. Once the measured values are loaded, the connection boards are operated and the user program is executed.

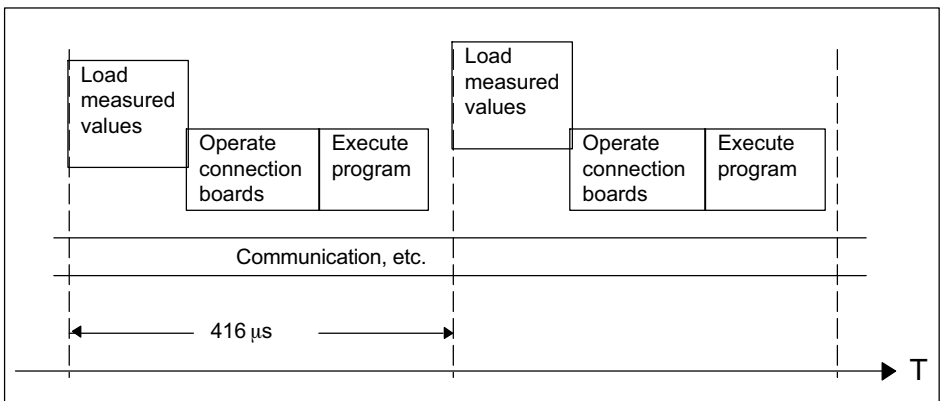


Fig. 6.1 ML70B cycle time

The maximum call frequency of 2400 Hz is only achieved if the IEC program is exited in less than 400 μs (1/2400 Hz). If a step takes longer, the call frequency is reduced accordingly (next-lowest frequency 1200 Hz).

This is why each program should be formulated in such a way that it can be exited as quickly as possible. This property basically distinguishes PLC programs from "normal" programs.

6.2 Data transmission

The amplifier modules (ML ...) and the communications processor (CP ...) in the MGCplus are connected by a fast synchronous data interface (link) via which measurement data can be exchanged in the system. Data is exchanged at a frequency of 2400 Hz.

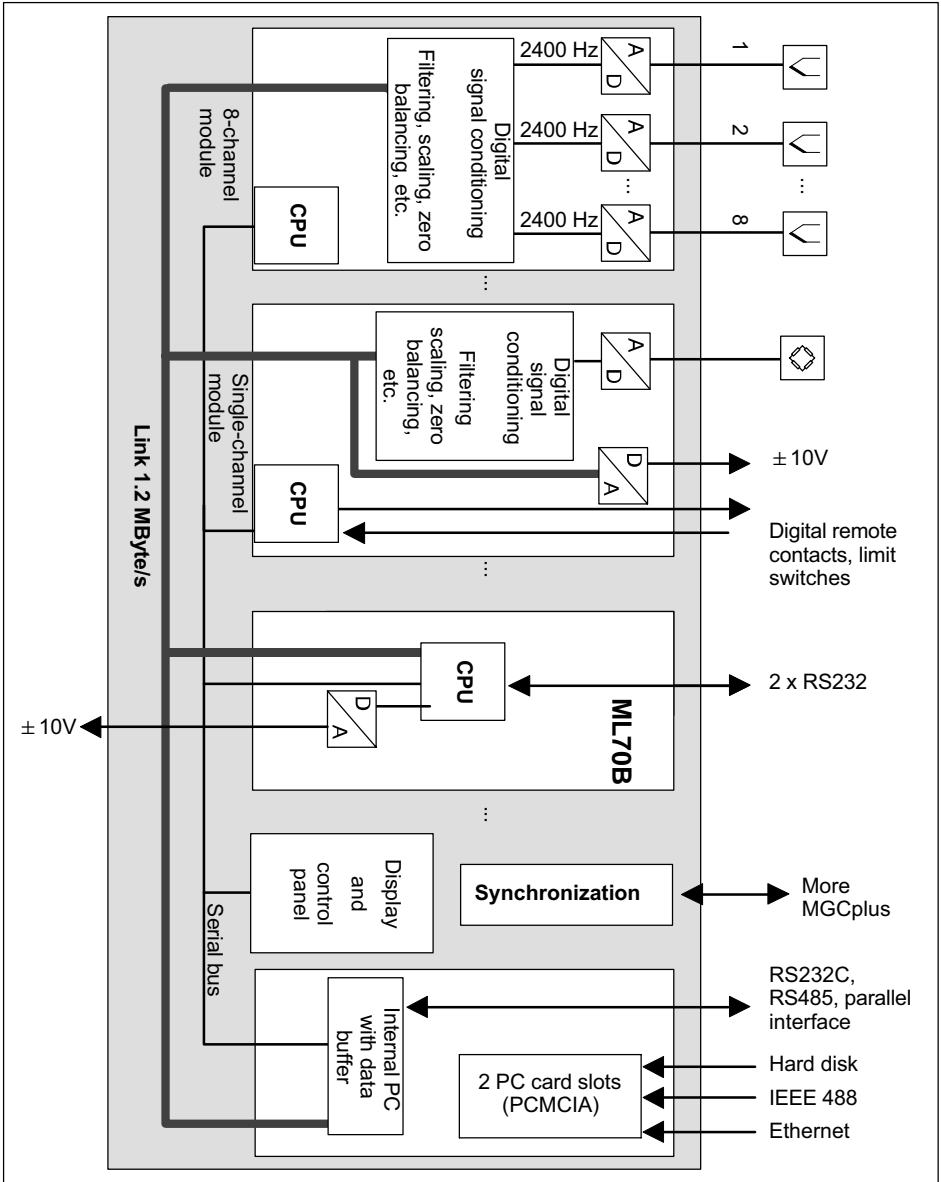


Fig. 6.2 Communication within MGCplus

Each module can output a total of 8 measured values, each at a transfer rate of 2400 Hz. For a single channel amplifier, this means that several signals (gross, net, peak value, etc.), can be output simultaneously.

Multi-channel amplifiers can only ever output one signal per subchannel for each of their eight subchannels, as only eight can be transmitted at any one time. For example, with a multi-channel amplifier, it is not possible to simultaneously request a gross signal and a net signal from one subchannel.

If measured values are requested by several components in the MGCplus, the control sequence must be defined. The communications processor (CP52) manages data transport, during which the requisite measured values must be requested. Each component that needs measured values must register this with the communications processor.

Conflicts will be caused if, for example, two signals are requested simultaneously from an 8-channel amplifier. In this case, the second request will be acknowledged with an error message.

When registering link resources, the following always applies: as long as resources are available these will be allocated and they will be allocated in the sequence in which they are requested.

6.3 Program architecture

For the reasons explained in the previous section, an ML70B program always comprises the following steps:

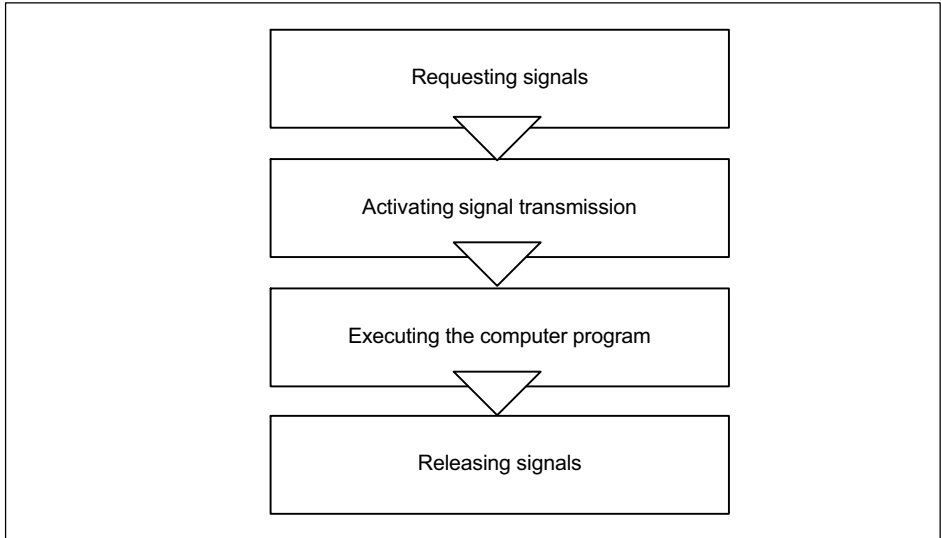


Fig. 6.3 Program architecture

This gives the same program architecture for all projects during which measured values are to be processed by MGCplus. We recommend that all measurement programs are structured as described in *Section 6.5 “PLC_PRG main program”*, page 23.

6.4 Saving programs

In the ML70B, a program is executed in internal volatile memory (RAM).

When the MGCplus is switched on, the system checks whether there is a program in non-volatile memory (FLASH). If there is, the stored program is copied from the FLASH into volatile memory and automatically executed.

The programming system can load another program into the ML70B and test it, without changing the existing program in the permanent memory. Provided the contents of the RAM are not copied to the FLASH, the ML70B will execute the old program from the FLASH after switching the MGCplus off and then back on again (see also Chapter 12 “Saving setup parameters”, page 57).

You can delete the contents of the FLASH if required:

1. Switch off the MGCplus device.
2. Withdraw the ML70B module from the housing. There is a 10-pin multipoint connector on the underside of the module.
3. Connect a bridge to pin 9 and pin10.
4. Switch the MGCplus device back on, which deletes the contents of the FLASH.
5. Switch the MGCplus device back off and remove the bridge.

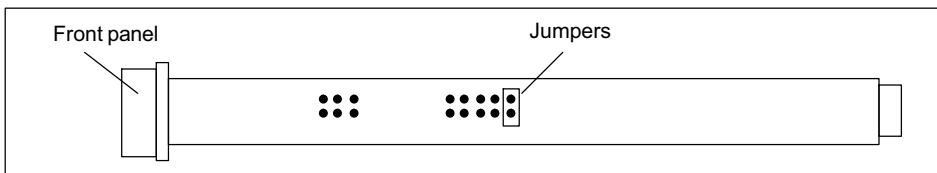


Fig. 6.4 ML70B amplifier module; view from below

Should a new program be saved permanently in the ML70B, it will be copied from the RAM to the FLASH ROM (see CoDeSys “Boot project generation”).

6.5 PLC_PRG main program

The ML70B calls the PLC_PRG main program synchronously with the measured values. Each program should be structured in the following form in SFC (sequential function chart) programming language:

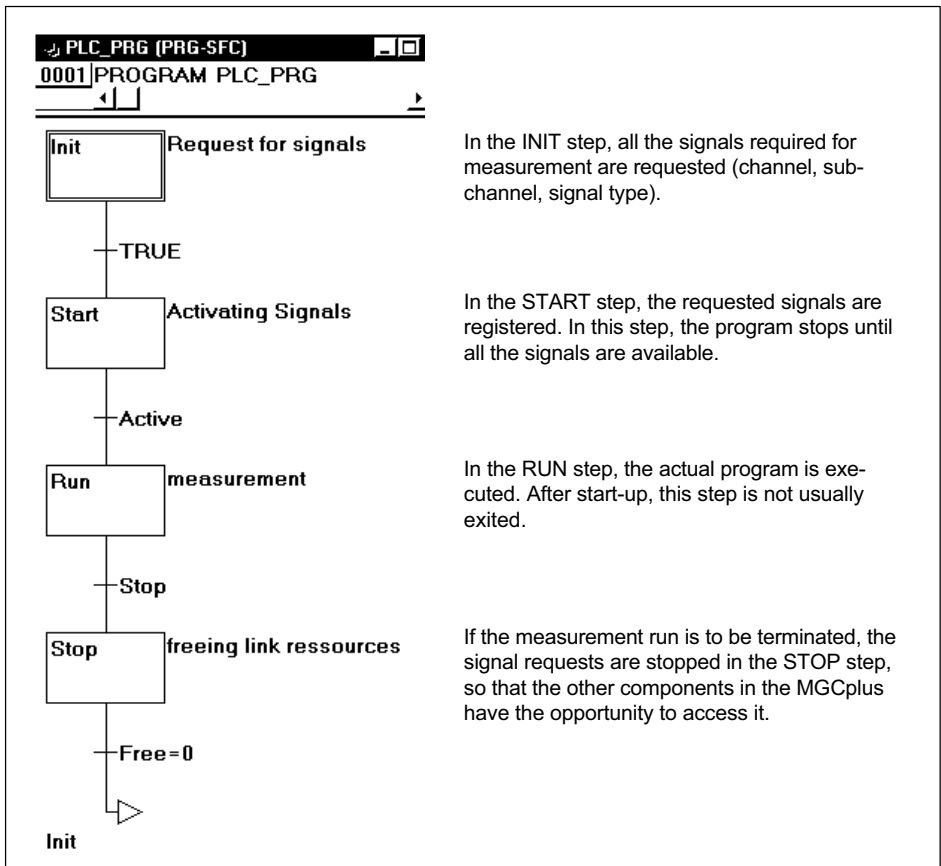


Fig. 6.5 Program run in ML70B

This diagram only plots the request and release of measurement data. The individual steps contain the actual function of the program.

The contents of the individual steps can be formulated in any IEC61131-3 language and are opened by double-clicking the individual step.



Important

The diagram shown should not be interpreted as a "normal" flowchart! Each time the program is called only one step is executed and the program is then exited again. If the transition condition between the two steps is met, the next time the program is called, the next step will be processed. If the condition is not met, the next time the program is called, the same step will be run again.

7 Introduction to programming

In this chapter a simple program example is used to explain the CoDeSys programming system and operation of the ML70B. For a detailed description of the programming system, refer to the online documentation of the CoDeSys installation.

System requirements

- MGCplus device with:
 - AB22A
 - CP42 or CP52 (optional)
 - Channel 1: ML55, ML01 or similar (single-channel amplifier)
 - Channel 2: ML55, ML01 or similar (single-channel amplifier)
 - Channel 3: ML70B with/without connection board
- PC with Windows operating system and serial interface

7.1 Installing the programming system

1. Put the MGCplus System-CD into the CD-ROM drive of your computer or download the data from <https://www.hbm.com/downloads>. Start the **Setup.exe** file, which is in the directory CoDeSys. The delivered CoDeSys version is V2.3.9.43.
2. Choose the required setup language and click **OK**.
3. Close the "Welcome..." window by clicking **Continue**.
4. In the setup type selection window, choose **Development system** and click **Continue**.
5. In the component selection window, all the checkboxes should be activated (default setting). Maintain the target folder setting and click **Continue**.
6. Choose the required language for the programming interface and click **Continue**.
7. Choose the required folder and click **Continue**.

After the summary is displayed, the development system is installed.

On the installation CD provided, the ML70B target system (Target) is installed automatically. But you can also install the target system later:

1. Use **Start** → **Programs** → **CoDeSys V2.3** to start the **InstallTarget** program.
2. Use the **Open...** button to open the file MGCplus_System_CD\MGCplus\CoDeSys\Targets\hbm\Hottinger.tnf.
3. Select the entry "Hottinger Baldwin Messtechnik GmbH" in the left-hand window and press the **Install button**. Answer the "Installation directory does not exist...?" question with Yes.

The entry "Hottinger Baldwin Messtechnik GmbH" should now appear in the right-hand window **Installed target systems**. This concludes the installation.

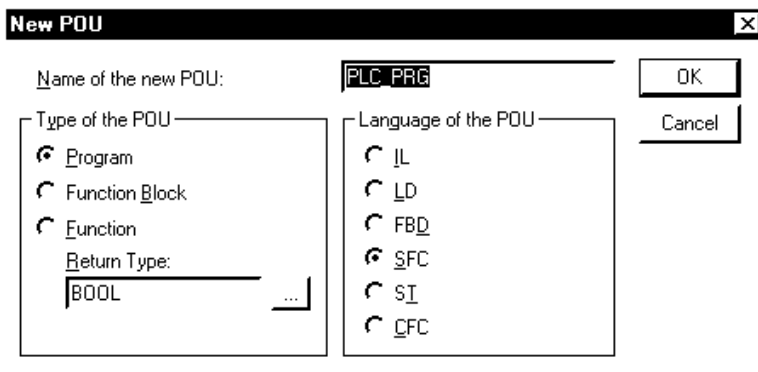
7.2 Program example

To determine the output of a motor, multiply the measured torque by the measured speed. The result is displayed in the display of the MGCplus device or in the Assistant.

Formula for calculating output:

$$\text{Output [W]} = \text{Torque [N} \cdot \text{m]} \cdot \text{Speed} \left[\frac{1}{\text{min}} \right] \cdot \frac{2 \cdot \pi}{60}$$

1. Start the CoDeSys programming system by using **Start** → **Programs** → **CoDeSys V2.3**.
2. Use **File** → **New** to generate a new project.
3. Select the target system *HBM_ML70B* and confirm with **OK** (the *New POU* window will then open automatically).
4. Select the *PLC_PRG* process object unit from *Program* type in *SFC* (sequential function chart) language and confirm with **OK**.



5. To enable you to access MGCplus components from the program, you must load the MGCPLUS.LIB library. Open **Window** → **Library management** and use **Insert** → **Additional library** → **MGCplus.lib** to add the library.

The library contains all the functions to enable you to access the hardware components of the MGCplus. Each available function is presented as a POU with a brief description of the significance of the parameters.

Special computing rules etc., are packed in the separate libraries.

Library Manager
 MGCplus.lib*11.1.02 14:42:22
 STANDARD.LIB*23.10.00 14:20:56
 IECSFC.LIB*23.10.00 14:20:30

POUs

- Access to Connection Boards
 - AP71 (Can Bus)
 - AP72 (Serial IO)
 - AP75 (Digital IO)
 - AP78 (Analog Out)
- Access to mgc channels
 - GetMgcInfos
 - Internal communication
 - Prepare Measurement
 - Read measuring values
- Access to ML70
 - Parameterizing ML70
 - GetDisplayLanguage (FUN)
 - GetML70ChanAdr (FUN)
 - InstallFasttask (FUN)
 - OutputSignal (FUN)
 - OutputSignalRaw (FUN)
 - OutputText (FUN)
 - SetLed (FUN)
 - SetScaling (FUN)**
 - SetSubChanDescriptor (FUN)
 - SetUccString (FUN)

FUNCTION SetScaling: INT
 (* setup scaling for ML70 output signal *)
 VAR_INPUT
 SubChan: INT; (* sub channel of the signal (1..8) *)
 Endscale: REAL; (* phys. endscale *)
 ZeroOffset: REAL; (* zero offset *)
 Decimals: INT; (* number of decimals *)
 Unit: STRING; (* string for phys. unit 4 character *)
 END_VAR

SETSCALING

- SubChan SetScaling
- Endscale
- ZeroOffset
- Decimals
- Unit

Once you have set up the new project, you will see the following window in the work area:

PLC_PRG (PRG-SFC)

```

0001 PROGRAM PLC_PRG
0002 VAR
0003 END_VAR
0004
    
```

Init

Trans0

Init

Creating the program architecture of the main program

As described in *Section 6.5*, the main program comprises four steps.

1. Click the hyphen of transition **Trans0** to make a dotted frame appear around Trans0 and in the context menu (click the right-hand mouse button), choose the command **Step transition (afterwards)**. Repeat this action twice up to step 4 (Step 4).
2. If you click directly on the name of a transition or a step, this will be marked in blue and can be modified. In this way, rename the three new steps as **START**, **RUN** and **STOP**.

Entering comments

3. Select a step by clicking just outside the frame, to make a dotted frame appear around the step. Click the right-hand mouse button on the step and select **Step attribute** from the context menu. You can then enter your comments in the window that opens.
4. To display the comments to the right of the steps, in the **Tools -> Options...** menu, select the **Comments** option.

Programming the “Init” step

The requisite signals in the system are called using the *RequestSignal()* function from library management. The channel number, the subchannel number and the signal number must be transmitted as call parameters. The coding for the signal types is explained in the “*Library Manager*” window. The return value provided by the function is an identifier (handle), by which the required measured value can later be queried.

In our example, the torque signal from channel 1, subchannel 1 is measured as a gross signal and the speed as a net signal from channel 2, subchannel 1.

1. Double-click on the “Init” step. You will be asked which language you require. Choose **ST** (structured text) and a new window will appear for the action for this step. In the first two program lines, enter the following:

```
HandleTorque := RequestSignal(1,1,0);
```

```
HandleRpm := RequestSignal(2,1,1);
```

You must declare the variables "HandleTorque" and "HandleRpm" as integers (INT).

You must then define the scaling for the result of the calculation:

For example, the maximum value of the torque is 500 N·m, the maximum speed 10000 min⁻¹.

The maximum output to be measured is:
 $10000 \cdot 100 \cdot 2 \cdot 3,14/60 \text{ W} = 104666 \text{ W}$

- In the third program line, enter the following:

SetScaling(1,120000.0,0.0, 1,'W');

The meaning of the parameters is as follows:

1	Scaling for subchannel 1 of the ML70B
120000.0	Maximum value that can occur during calculation
0.0	No zero offset
1	The result of the calculation is displayed with one decimal place
'W'	The measured value unit is 'W' (watt)

This concludes the programming of the "Init" step and you can close the **Action Init** window. A small triangle now appears in the top left corner of the INITstep to indicate that this step is programmed.

Programming the "START" step

The requisite signals are requested in the system by calling the *ActivateSignals()* function. The START step is called until the measurement signals are released. This meets the condition for the transition to the RUN step.

For the START step, enter the following program line:

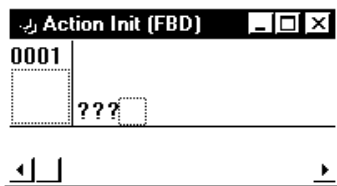
Ready := ActivateSignals(SIGNAL_REQUEST);

Declare the Ready variable as INT.

Programming the "START" step

The START step is programmed in FBD (function block diagram) language, the actual computing takes place here. In our program example, this step is then not exited.

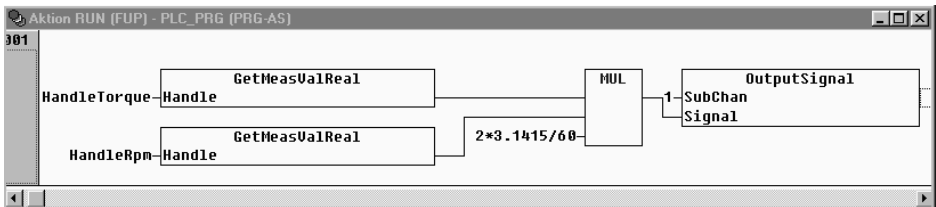
1. Double-click on the RUN step and choose FBD language. The following window will appear:



2. Position the mouse pointer in the square mark after the three question marks and by using the context menu (right-hand mouse button) execute the **POU** command. The default is for the **AND** process object unit to be inserted, the name is selected.
3. Press key F2 (input Help). This opens a dialog in which you can select from the available POUs. For our example, first select in the left-hand window the category **Standard functions**. Then, in the right-hand window, from the **Targets\Hottinger\...\MGCplus.lib** library in the **Access to ML70B** folder, select the **OutputSignal** function.
4. Now click the three question marks next to the input variable **SubChan** and enter the value **1**.
5. Select the connecting line next to the **Signal** input variable so that it is marked by a dotted frame and in the context menu execute the **POU** command.
6. Replace the selected text **AND** by entering **MUL** (for multiplication).
7. Again select the connecting line next to the upper input variable of 'MUL' and use the context menu to execute the **POU** command.
8. Press F2 and add the function **Standard functions** → **MGCplus.lib** → **Read measuring values** → **GetMeasValReal**.

9. Repeat this action for the second input variable of 'MUL'. Enter the variables HandleTorque and HandleRpm as the input variables of GetMeasValReal(). Now insert the correction factor for multiplication.
10. To do this, click the **MUL** process object unit and use the context menu to execute the **Input** command. You will see another input with ??? as the input variable. Select the three question marks and replace them with the expression $2 * 3.1415/60$

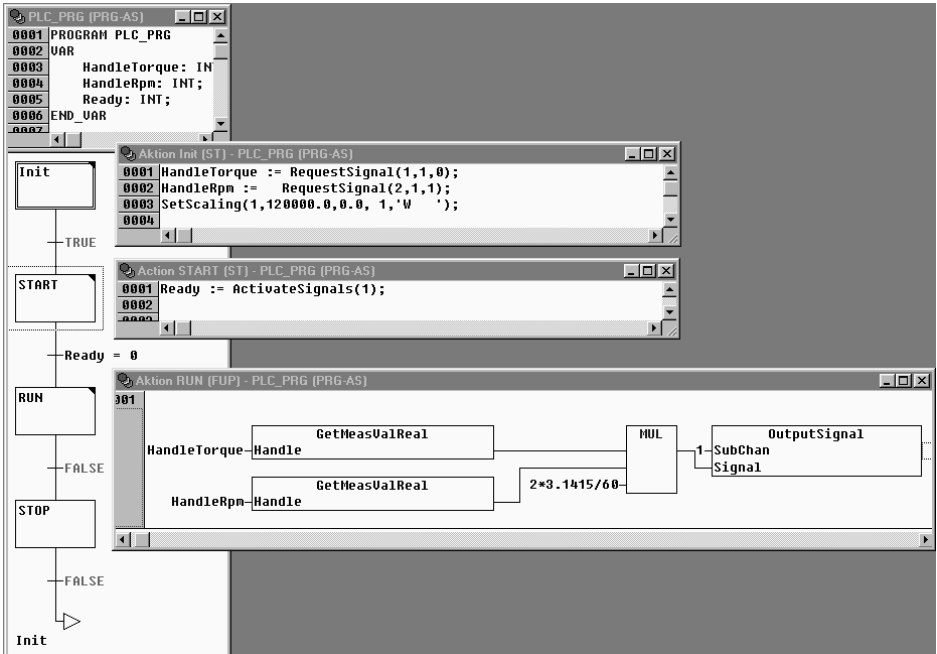
This concludes the programming of the RUN step. The resultant network should now look like this:



Defining transitions

Transitions are the conditions for moving from one step to the next. Overwrite the first transition after Init with the value TRUE. This will mean that the Init step is called once only. The second transition after START is given the condition Ready = 0 (the variable set in the Start step). When Ready = TRUE (i.e. when signal registration is concluded) the next step starts. The third and fourth transitions are given the value FALSE, i.e. the RUN step is never exited.

Programming is now complete. The resultant program should now look like this:



7.3 Translating the project

Translate the project using the menu command **Project** → **Translate all** or press function key **<F11>**. After the translation run, "0 errors" should be reported in the message window at the bottom right. If this is not the case, please check all entries for correctness. Also take note of the error messages.

Pressing function key **<F4>** takes you directly to the errors in the program.

7.4 Starting the target system and loading the program

The target system for this example is an MGCplus with the following configuration:

- AB22A display and control panel
- CP42 or CP52 communications processor (optional)
- Channel 1: ML55, ML01 or similar (single-channel amplifier)
- Channel 2: ML55, ML01 or similar (single-channel amplifier)
- Channel 3: ML70B

The PC and the ML70B communicate via a serial connection. Use the cable supplied to connect the ML70B via the DEBUG socket on the front panel to a serial PC interface. To set up the interface, proceed as follows:

- Execute the menu command **Online** → **Communications parameter**. In the dialog that then appears, choose the **New** button to configure the connection setup to your target system.
- In the new dialog, assign a suitable name to the connection and select **Serial RS232**. By clicking the COM1 interface repeatedly, you can change to COM2,3,... Quit and exit the dialog with **OK**.
- Use the menu command **Online** → **Login** in the programming system **CoDeSys** to establish the connection to the target system and start your program on the target system with **Online** → **Start**.

The result of multiplication can now be read on the AB22A when the ML70B is selected by the channel selection key CHANNEL + / -. You can now view the variables in the individual windows on the PC screen.

If the program is to be saved permanently in the ML70B, you must call the **Online** → **Generate boot project** menu. This transfers the program to non-volatile FLASH memory. The next time MGCplus is switched on, this program starts up automatically.

8 Communication with other amplifier channels

8.1 Loading measured values

Before measured values from other MGCplus amplifier channels can be loaded, the requisite signals must first be requested and then activated (see *Section 6.5 ff.*)

The measured values are loaded using the functions `GetMeasValReal()` and `GetMeasValRaw()`.

Example:

Measured value: REAL

Handle: INT

`MeasVal := GetMeasValReal(Handle)`

The variable “Handle” must have previously been assigned in the START step (see *Section 6.5 and Section 7.2*)

Access to the measured values is then only via this handle. The function `GetMeasValReal()` then always returns the current value for the particular handle as a physically scaled variable.

`GetMeasValRaw()` returns an unscaled raw value as a 32-bit integer in the format DINT. This data format is preferred when for reasons of time, calculations in REAL format have to be dispensed with. The raw value is converted to the physical measured value according to the following formula:

$$\text{Measured value} = \frac{\text{Raw value}}{7680000} \cdot \text{Full scale} - \text{Zero offset}$$

The full scale and the zero offset are defined in the amplifier scaling (*also see Section 9.3 “Outputting computed values, page 37”*) and can be queried by the amplifier using the function `GetChannelInfo()`.

8.2 Sending commands

The ML70B communicates with other amplifier modules using the functions `SendMgcCommand()`, `WaitMgcAnswer()` and `GetMgcAnswer()`.

In the first step, the command string is sent to the required channel or subchannel by calling the function `SendMgcCommand()`.

Then the function `WaitMgcAnswer()` is queried until the result 0 (OK) or <0 (errors) is obtained. You can then use the function `GetMgcAnswer()` to get the response string.

The screenshot displays a PLC programming environment with three main components:

- Variable Declaration:** A window titled "PLC_PRG (PRG-AS) (92/-1/9)" shows the following code:


```

0001 PROGRAM PLC_PRG
0002 VAR
0003   NewCmd: BOOL;      (* TRUE: New comm.
0004   Chan: INT;         (* channel to whic
0005   SubChan: INT;     (* sub channel to
0006   Command: STRING; (* command string
0007   Answer: STRING;  (* answer string
0008   StartSend: BOOL; (* command sent *
0009 END_VAR
0010 VAR_INPUT
0011   ErrorCode: INT;  (* returned error c
0012 END_VAR
            
```
- Action SendCmd:** A window titled "Action SendCmd (ST) (-1/-1/12)" shows the following code:


```

0001 IF NewCmd THEN
0002   ErrorCode := SendMgcCommand(Chan, SubChan, Command);
0003   IF ErrorCode = 0 THEN
0004     StartSend := TRUE;
0005     NewCmd := FALSE;
0006   END_IF
0007 END_IF
            
```
- Action WaitCmd:** A window titled "Aktion WaitCmd (ST) (-1/-1/14)" shows the following code:


```

0001 ErrorCode := WaitMgcAnswer(0);
0002 IF ErrorCode <= 0 THEN
0003   StartSend := FALSE;
0004   Answer := GetMgcAnswer(0);
0005 END_IF
            
```
- Ladder Logic:** A diagram shows a normally open contact labeled "TRUE" connected to a coil labeled "SendCmd". Below this, a coil labeled "StartSend" is connected to a coil labeled "WaitCmd". A feedback line from the "WaitCmd" coil goes back to the "StartSend" coil, labeled "StartSend = FALSE".

9 Configuring hardware components

9.1 Analog outputs

The analog outputs of the ML70B are actuated using the functions `SetAnalogOutputReal()` and `SetAnalogOutputInt()`.

Parameter 1 corresponds to the number of the analog output (1 = V_{O1} on the BNC connector or AP75/AP78; 2 = V_{O2} on the AP75/AP78).

With the `SetAnalogOutputReal()` function, for the second parameter, you can enter the required output voltage directly in volts (-10...+10.) as a REAL variable.

If computing in REAL takes up too much time, you can use the function `SetAnalogOutputInt()`. For the second parameter, a value between -30000 ... 30000 is given as an INT value. -30000 corresponds to -10 V, 30000 corresponds to 10 V.

9.2 Front panel LEDs

The LEDs on the front panel of the ML70B can be actuated using the function `SetLED()`.

Parameter 1 transfers the number of the LED (1...7), corresponding to the numbering on the front panel. Parameter 2 contains the required status of the LED: 0=Off, 1=Red, 2=Yellow

9.3 Outputting computed values

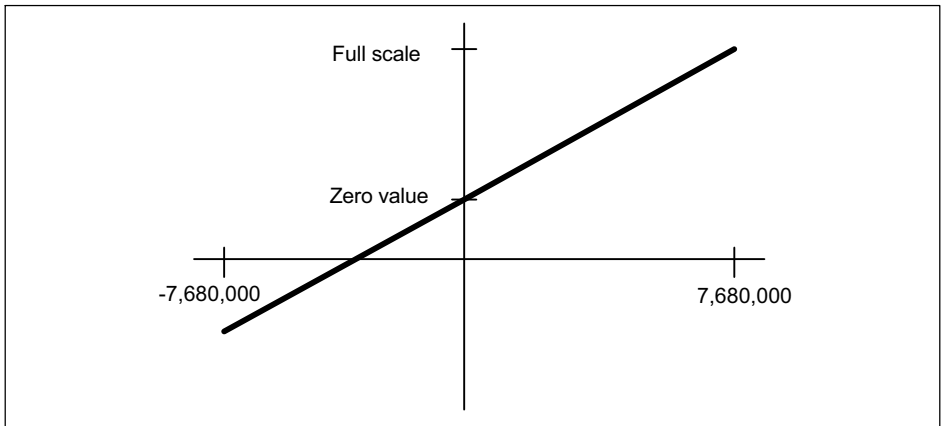
The function `OutputSignal()` outputs the computed values on the internal data bus. The first parameter transfers the number of the required subchannel on which the ML70B is to output the signal. The second parameter is the value to be output as a REAL value.



Important

In order for values to be properly presented by catman® or the AB22A, scaling must be specified before the first value is output.

In MGCplus, all measured values are transferred as 24-bit-integers. The number range runs from -7 680 000 ... +7 680 000. Each REAL number is converted to a 24-bit integer for transmission, in accordance with the following system.



At the receiver end, the integer is converted back to REAL values.

For this conversion, the full scale and the zero value must be specified for the ML70B using the `SetScaling()` function:

$$\text{Measured value} = \frac{\text{Link value}}{7.680.000} \cdot \text{Full scale} - \text{Zero value}$$

10 Connection board control

10.1 AP71/CAN

To explain CAN interface operation, we have included the program examples MGC_CanDemo, MGC_CanOpenDemo and MGC_SDOTerm. CAN_Hardware is operated by using the AP71 functions in the MGCplus.lib library. To make it easier to send and receive CAN messages, we have included an additional library MGCcan.lib, which contains all the functions you need.

10.1.1 Basic CAN driver function

Operation of the two CAN interfaces of the AP71 connection board is interrupt-driven by the firmware. The data structure shown below is the only way in which the IEC61131-3 program accesses the CAN interfaces. CAN messages are transmitted and received via a ring buffer. By comparing the read and write indexes, the IEC program can establish whether new messages have been received.

```

0001 TYPE CAN_Interface :
0002 STRUCT
0003     pRx_Buffer      : POINTER TO CAN_Message;
0004     pTx_Buffer      : POINTER TO CAN_Message;
0005     pCallbacks      : POINTER TO CAN_Callback;
0006
0007     CANMAX_RX_BUFFER: INT; (* Größe des Empfangsbuffers n := (n + 1) MOD CANMAX_RX_BUFFER *)
0008     CANMAX_TX_BUFFER: INT; (* Größe des Sendebuffers n := (n + 1) MOD CANMAX_TX_BUFFER *)
0009     CANMAX_CALLBACKS: INT; (* Größe des Callbackbuffers n := (n + 1) MOD CANMAX_CALLBACKS *)
0010
0011     wBaudrate       : WORD; (* Beim Init-Aufruf einzustellende Baudrate *)
0012
0013     nRx_Index_DRU   : INT := -1; (* Index des DRU's, für die zuletzt eingetragenen HSG in den Empfangsbuffer *)
0014     nTx_Index_DRU   : INT := -1; (* Index des DRU's, für die als nächstes zu versendende HSG aus dem Sendebuffer *)
0015     nWrite_DRU      : INT;       (* Index des DRU's, für die zuletzt eingetragenen HSG in den Empfangsbuffer
0016                                     (für DRU's, die Eintragungen in Empfangsbuffer überschreiben können) *)
0017     nRx_Index_IEC   : INT := -1; (* Index des IEC-Programms, für die als nächstes zu lesene HSG aus dem Empfangsbuffer *)
0018     nTx_Index_IEC   : INT := -1; (* Index des IEC-Programms, für die zuletzt eingetragenen HSG in den Sendebuffer *)
0019     nWrite_IEC      : INT;       (* Index des IEC-Programms, für die zur Zeit beschriebene HSG in den Sendebuffer *)
0020     bOverWrite      : BOOL;      (* hier kann der DRU anzeigen, daß ein Überlauf im Empfangsbuffer aufgetreten ist *)
0021     dwErrorCode     : DWORD;     (* hier kann der DRU Errors eintragen, die dann in IEC-Programm ausgewertet werden können *)
0022     pArray          : ARRAY[0..32] OF WORD; (* not used *)
0023 END_STRUCT
0024 END_TYPE
0025
0026
    
```

The library MGCcan.lib contains all the functions for user-friendly read out of CAN messages from CAN_Interface.

Example 1:

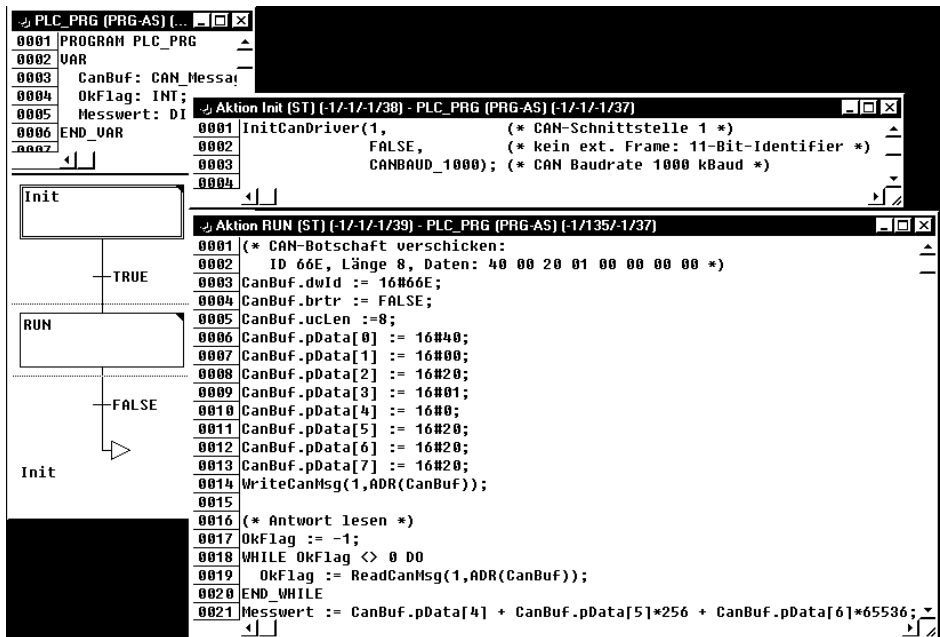
Communication with a CAN node at low level

The measured values are to be loaded continuously by a CAN node.

In the **Init** step, you must first initialize the required CAN interface. In this example, this is interface No. 1 on the AP71. The function `InitCanDriver()` is used to initialize both the data structure `CAN_Interface` and the hardware for the CAN interface.

In the **Run** step, the data structure `CanBuf` is filled with the required values and then the CAN message is sent out using the function `WriteCanMsg()`.

The response is loaded by using the function `ReadCanMsg()`. The function returns the value 0 when a new CAN message arrives.



```

0001 PROGRAM PLC_PRG
0002 VAR
0003   CanBuf: CAN_Messa;
0004   OkFlag: INT;
0005   Messwert: DI;
0006 END_VAR
0007
0008
0009
0010
0011
0012
0013
0014
0015
0016
0017
0018
0019
0020
0021
    
```

Aktion Init (ST) (-1/1-1/38) - PLC_PRG (PRG-AS) (-1/1-1/37)

```

0001 InitCanDriver(1, (* CAN-Schnittstelle 1 *)
0002 FALSE, (* kein ext. Frame: 11-Bit-Identifizier *)
0003 CANBAUD_1000); (* CAN Baudrate 1000 kBaud *)
0004
    
```

Aktion RUN (ST) (-1/1-1/39) - PLC_PRG (PRG-AS) (-1/135-1/37)

```

0001 (* CAN-Botschaft verschicken:
0002 ID 66E, Länge 8, Daten: 40 00 20 01 00 00 00 00 *)
0003 CanBuf.dwId := 16#66E;
0004 CanBuf.brtr := FALSE;
0005 CanBuf.ucLen :=8;
0006 CanBuf.pData[0] := 16#40;
0007 CanBuf.pData[1] := 16#00;
0008 CanBuf.pData[2] := 16#20;
0009 CanBuf.pData[3] := 16#01;
0010 CanBuf.pData[4] := 16#0;
0011 CanBuf.pData[5] := 16#20;
0012 CanBuf.pData[6] := 16#20;
0013 CanBuf.pData[7] := 16#20;
0014 WriteCanMsg(1,ADR(CanBuf));
0015
0016 (* Antwort lesen *)
0017 OkFlag := -1;
0018 WHILE OkFlag <> 0 DO
0019   OkFlag := ReadCanMsg(1,ADR(CanBuf));
0020 END_WHILE
0021 Messwert := CanBuf.pData[4] + CanBuf.pData[5]*256 + CanBuf.pData[6]*65536;
    
```

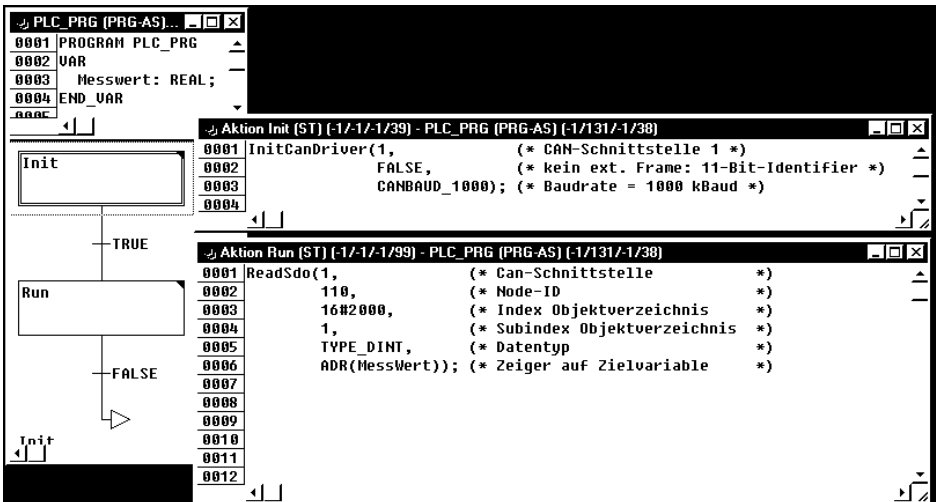

Example 2:

Using the CANopen protocol to control a CAN node

A measured value is to be continuously loaded using a CANopen SDO (service data object).

The functions `ReadSdo()` and `WriteSdo()` are available in the `MGCcan` library for reading and writing an object in the object catalog of a CANopen slave.

In our example, the CANopen slave has node-ID 110 and the measured value is in the object catalog under index `16#2000` and subindex 1 in DINT format.



The screenshot displays a PLC program with the following components:

- Program Declaration:**

```

0001 PROGRAM PLC_PRG
0002 VAR
0003   Messwert: REAL;
0004 END_VAR

```
- Ladder Logic:**
 - An **Init** step is connected to a **Run** step via a normally open contact labeled **TRUE**.
 - The **Run** step is connected to a timer T1 via a normally open contact labeled **FALSE**.
- Action Init (ST) [-1/1/-1/39] - PLC_PRG (PRG-AS) [-1/131/-1/38]:**

```

0001 InitCanDriver(1,          (* CAN-Schnittstelle 1 *)
0002                FALSE,     (* kein ext. Frame: 11-Bit-Identifizier *)
0003                CANBAUD_1000); (* Baudrate = 1000 kBaud *)
0004

```
- Action Run (ST) [-1/1/-1/99] - PLC_PRG (PRG-AS) [-1/131/-1/38]:**

```

0001 ReadSdo(1,          (* Can-Schnittstelle *)
0002           110,       (* Node-ID *)
0003           16#2000,   (* Index Objektverzeichnis *)
0004           1,         (* Subindex Objektverzeichnis *)
0005           TYPE_DINT, (* Datentyp *)
0006           ADR(Messwert)); (* Zeiger auf Zielvariable *)
0007
0008
0009
0010
0011
0012

```

In the **Init** step, the CAN interface is initialized (once). In the **Run** step, the function `ReadSdo()` is used to load the measured value of the CANopen slave.

10.2 Serial communication (AP72)

With the ML70B, you can control up to four serial interfaces. Interfaces No. 1 and No. 2 are at slot AP-A, with No. 3 and No. 4 at AP-B. You can choose between three types:

- RS232
- RS422 (full duplex 4-wire connection with push-pull signals)
- RS485 (half duplex 2-wire connection with push-pull signals)

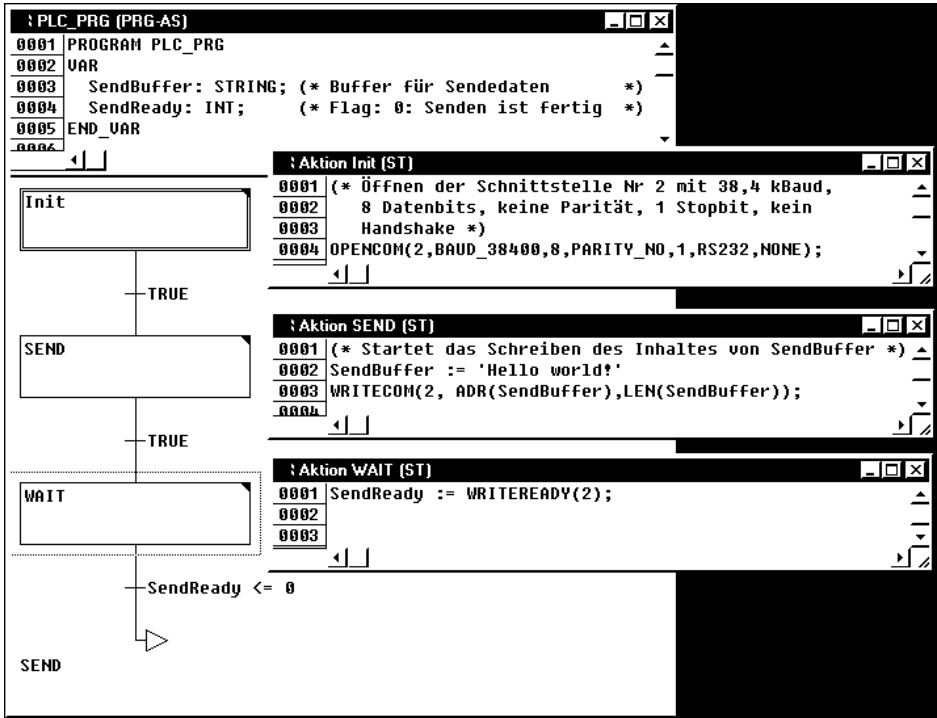
In order to send and receive data, corresponding buffers must be made available in the IEC program. The actual send and receive operation is (interrupt-driven) and runs in the background. The data types of the buffers depend on the particular application. STRING or ARRAY OF BYTE are usually advisable.

10.2.1 Sending data

To send data, you need the functions `OpenCom()`, `WriteCom()` and `WriteReady()`.

There are three steps to sending data:

1. Opening the interface (see diagram; “Init” status)
2. Preparing the data and starting the send operation (see below, “SEND” status)
3. Waiting until the send operation is completed (see below, “WAIT” status)



```

\ PLC_PRG (PRG-AS)
0001 PROGRAM PLC_PRG
0002 UAR
0003   SendBuffer: STRING; (* Buffer für Sendedaten *)
0004   SendReady: INT;     (* Flag: 0: Senden ist fertig *)
0005 END_UAR
0006

\ Aktion Init (ST)
0001 (* Öffnen der Schnittstelle Nr 2 mit 38,4 kBAud,
0002   8 Datenbits, keine Parität, 1 Stopbit, kein
0003   Handshake *)
0004 OPENCOM(2,BAUD_38400,8,PARITY_NO,1,RS232,NONE);

\ Aktion SEND (ST)
0001 (* Startet das Schreiben des Inhaltes von SendBuffer *)
0002 SendBuffer := 'Hello world!';
0003 WRITECOM(2, ADR(SendBuffer),LEN(SendBuffer));

\ Aktion WAIT (ST)
0001 SendReady := WRITEREADY(2);
0002
0003
  
```

The program example continuously sends the string "Hello world!" at interface 2. The constants needed to set up the interface (the parameters of the OPENCOM() function) can be found in library management in the "Data types" register.

10.2.2 Receiving data

To receive data, you need the functions OpenCom(), ReadCom() and ReadReady().

There are three steps to receiving data:

1. Opening the interface (see diagram; "Init" status)
2. Starting receiving ("RECEIVE" status)
3. Waiting until the expected data has been received (see below, WAIT status)

Example 1

The screenshot displays a PLC program window titled "\ PLC_PRG (PRG-AS)". The program code is as follows:

```

0001 PROGRAM PLC_PRG
0002 VAR
0003   RecBuffer: ARRAY[0..50] OF BYTE; (* Buffer für Empfangsdaten *)
0004   RecReady: INT; (* Flag: 0: Empfang ist fertig *)
0005 END_VAR
0006

```

The ladder logic diagram consists of three rungs:

- Rung 1:** A normally open contact labeled "Init" is connected to a coil labeled "RECEIVE".
- Rung 2:** A normally open contact labeled "TRUE" is connected to a coil labeled "RECEIVE".
- Rung 3:** A normally open contact labeled "RecReady <= 0" is connected to a coil labeled "RECEIVE".

Three action blocks are shown on the right side of the window:

- \ Aktion Init (ST):**

```

0001 (* Öffnen der Schnittstelle Nr 2 mit 9,6 kBaud,
0002   8 Datenbits, gerader Parität, 2 Stopbits, kein
0003   Handshake *)
0004 OPENCOM(2,BAUD_9600,8,PARITY_EVEN,2,RS485,NONE);

```
- \ Action RECEIVE (ST):**

```

0001 (* Startet den Empfang von 20 Zeichen
0002   mit einer Timeoutzeit von 10 sec *)
0003 READCOM(2, ADR(RecBuffer),20,0,10000);

```
- \ Aktion WAIT (ST):**

```

0001 RecReady := READREADY(2);
0002
0003

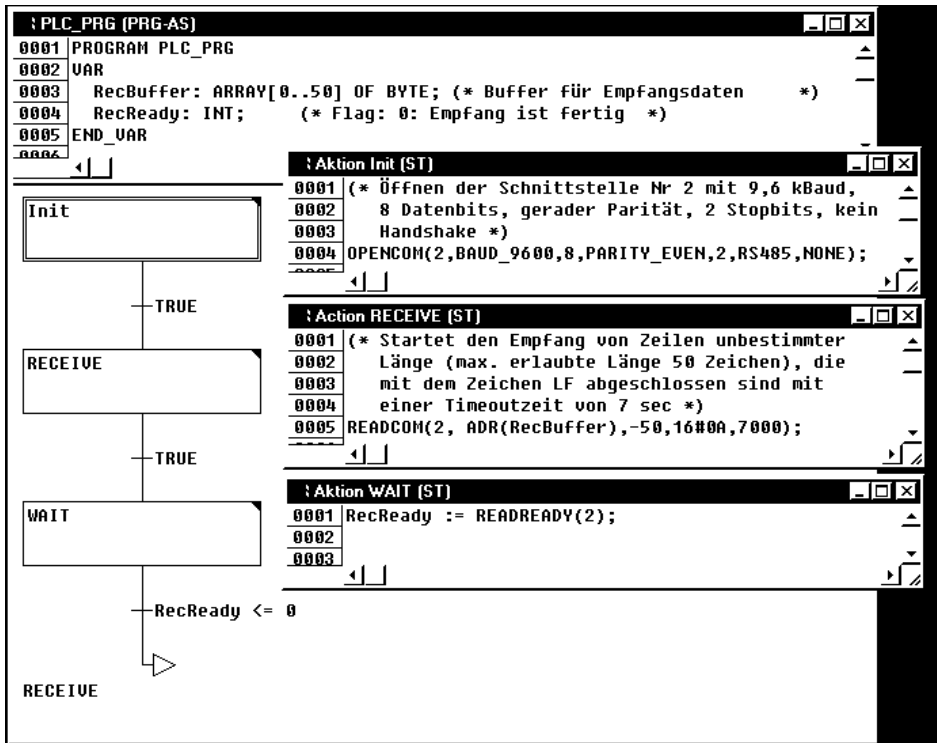
```

The program receives endless blocks of 20 characters each and writes these to the "RecBuffer".

Example 2

If the length of the received data block is unknown or variable, choose a receive mode that breaks off when an end label is received.

A negative number is given as the data length. The value of the figure is the maximum buffer length available. The required end label must also be given.



The screenshot displays a PLC program editor for 'PLC_PRG (PRG-AS)'. On the left, a ladder logic diagram shows three rungs: 'Init' (TRUE), 'RECEIVE' (TRUE), and 'WAIT' (RecReady <= 0). The 'RECEIVE' rung is currently active. On the right, three action blocks are visible:

- : Aktion Init [ST]**

```

0001 (* Öffnen der Schnittstelle Nr 2 mit 9,6 kBaud,
0002 8 Datenbits, gerader Parität, 2 Stopbits, kein
0003 Handshake *)
0004 OPENCOM(2,BAUD_9600,8,PARITY_EVEN,2,RS485,NONE);
        
```
- : Action RECEIVE [ST]**

```

0001 (* Startet den Empfang von Zeilen unbestimmter
0002 Länge (max. erlaubte Länge 50 Zeichen), die
0003 mit dem Zeichen LF abgeschlossen sind mit
0004 einer Timeoutzeit von 7 sec *)
0005 READCOM(2, ADR(RecBuffer),-50,16#0A,7000);
        
```
- : Aktion WAIT [ST]**

```

0001 RecReady := READREADY(2);
0002
0003
        
```

At the top of the editor, the following code is visible:

```

0001 PROGRAM PLC_PRG
0002 VAR
0003   RecBuffer: ARRAY[0..50] OF BYTE; (* Buffer für Empfangsdaten *)
0004   RecReady: INT; (* Flag: 0: Empfang ist fertig *)
0005 END_VAR
0006
    
```

10.3 Digital inputs and outputs (AP75)

You can use the function `SetOutputAP75()` (MGCplus.lib) to set the digital outputs of the AP75 (caution: you need an external supply).

The first parameter specifies the number of the output. Outputs 1...8 are found on the AP75 at slot A (slot directly behind the ML70B). Outputs 9..16 are located on the AP75 at slot B.

The second parameter specifies the level: TRUE = 24 V, FALSE = 0 V.

The inputs are read by using the function `GetInputAp75()`. Numbering is the same as for the digital outputs. The return value TRUE corresponds to a level > 10 V at the input, the return value FALSE corresponds to 0 V.

Example

Output3 of the AP75 at slot A should be set to HIGH level:

```
SetOutputAp75(3,TRUE);
```

10.4 AP78 analog outputs

The functions `SetAnalogOutputReal()` and `SetAnalogOutputInt()` are used to control the analog outputs on the AP78.

The number of the analog output (3 ... 10; 1 and 2 are reserved for the analog outputs of the ML70B) is given to these functions as parameter 1.

The function `SetAnalogOutputReal()` has as its second parameter the required output voltage directly in volts (-10....+10.) as a REAL variable.

If computing in REAL takes up too much time, you can use the function `SetAnalogOutputInt()`. For the second parameter, a value between -30000 ... 30000 is given as an INT value. -30000 corresponds to -10 V, 30000 corresponds to 10 V.

11 Generating dialogs

11.1 Dialog structure

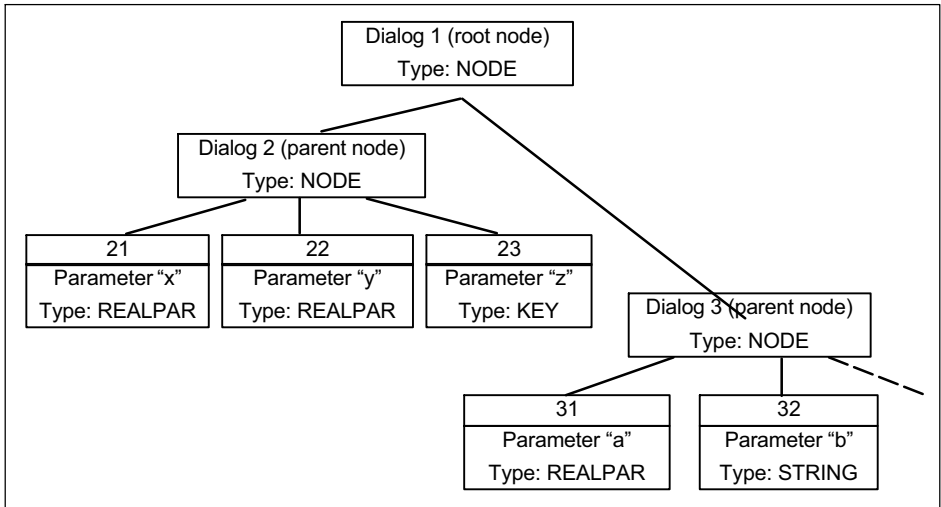
The computing module ML70B gives you the opportunity to create dialog boxes that can be called in the MGCplus Assistant or on the ABxx display and control panel. On the ABxx, you can then allocate the required dialogs to function keys F3 and F4 in setup mode. It is also possible to define actions that can be triggered by the function keys in measuring mode.

The dialogs are formed from various parameter types. Parameters of the "node" type make up a new dialog box, all the other parameter types make up its contents (menu entries, buttons, selection fields, etc.).

The dialog parameters form a tree structure. In top position is a root node, which makes up the output display. You can keep branching one level lower to further "subdialogs" via additional "parent nodes". The parameters directly subordinate to a node form a self-contained dialog.

Each node and each parameter are identified by a unique number. The root node always has the number 1. You can generate a maximum of 19 dialogs below the root node, assigning them the numbers 2 ... 999 (apart from 20, this number is already taken).

Parameter numbers 20 000 to 20 499 are reserved for the ABxx function key assignments. The root node has the number 10 000. You can assign the required functions to keys F1 ... F4 in setup mode via the setup window "Display" → "F-keys".



To generate a dialog, you must first declare two variables for each parameter in the IEC program:

- The variable itself
- A function block with the particular properties and a function for the entry in the parameter list

Parameter type	Data type for the parameter	Data type for the parameter properties	Remarks
NODE	INT	CreateNode	Node
REALPAR	REAL	CreateRealPar	Parameter of the REAL type
INTPAR	INT	CreateIntPar	Parameter of the INT type
DINTPAR	DINT	CreateDintPar	Parameter of the DINT type
KEY	INT	CreateKeyPar	Button (corresponds to a Windows button)
MENUE	INT	CreateMenuePar	Selection menu (corresponds to a Windows ComboBox)
TEXT	STRING	CreateTextPar	Parameter of the STRING type

Tab. 11.1 Parameter types

11.2 Parameter properties

Each parameter is described by several properties and is entered in the parameter list via a function call. Each property is pre-assigned a start value. Only those properties that differ from the preassignment have to be set in the program.

11.2.1 Properties of the NODE parameter type

NODEs combine parameters into groups.

pValue: POINTER TO INT	Pointer to the variable (which must also be defined in the IEC program) containing the status of the node.
ParId: INT	Number of the node (1...19) Nodes 2..9 are presented in the SET mode of the ABxx under F3, nodes 11..19 under F4.
Name: STRING	Name of the node: This text is presented in the ABxx dialog when the relevant key (F3 or F4) is pressed in SET mode.
Root: INT	The entry here shows the root node to which the node belongs (nothing has to be entered here for node 1)

To generate nodes, use the function `CreateNode()`;



Information

The node with the number 1 (root node) is already set in the ML70B.

11.2.2 Properties of the DINTPAR, INTPAR and REALPAR parameter types

These parameter types are used to present numeric parameters of the DINT, INT and REAL types.

<Type> represents below the respective types DINT, INT and REAL

pValue: POINTER TO <Type>	Pointer to the variable (which also has to be defined in the IEC program) to be modified by the parameter assignment dialog
ParId: INT	Number of the parameter (2...999)
Name: STRING	Name of the parameter: This text is presented in the ABxx dialog, together with the value of the parameter
Root: INT	The entry here indicates the root node to which the parameter belongs.
MinVal: <Type>	Minimum value, which the parameter setting must not fall below
MaxVal: <Type>	Maximum value, which the parameter setting must not exceed
EditWidth: INT	Editing width
Decimals: INT	Number of decimal places to be presented for the parameter
ScalFact: REAL	Scale factor for presentation on the ABxx
Offset: REAL	Offset for presentation on the ABxx
Unit: STRING	Physical unit as a string with 4 characters, shown in the parameter assignment dialog together with the parameter.
Flags: PARFLAGS	Special properties (not previously supported)

The parameters ScalFact and Offset can be used to present the value of the parameter in a different scaling in the indicator. If the parameter is to be presented without scaling (normal case), Scalfact = 1.0 and Offset = 0.0

Display value = Parameter * ScalFact - Offset

Generate the parameters by using the function blocks `CreateRealPar()`, `CreateIntPar()` and `CreateDintPar()`.

11.2.3 Properties of the KEY parameter type

Use the KEY parameter type to generate the buttons in the dialogs. Buttons can be used to trigger actions or open further subdialogs.

To create a button, use the function block `CreateKey()`

pValue: POINTER TO INT	Pointer to the variable (which also has to be defined in the IEC program) to present the status of the button in the dialog
ParId: INT	Number of the parameter (2...999)
Name: STRING	Name of the button: This text is used to label the button in the ABxx and Assistant dialogs.
Root: INT	Number of the root node to which the parameter belongs.
Flags: PARFLAGS	Special properties (not previously supported)

11.2.4 Properties of the TEXT parameter type

Use this parameter type to create the texts in the dialogs.

You generate a text by using the function block `CreateTextPar()`.

pValue: POINTER TO STRING	Pointer to the text (which also has to be defined in the IEC program) to be modified by the dialog
ParId: INT	Number of the parameter (2...999)
Name: STRING	Name of the parameter: This text is presented in the ABxx dialog, together with the value of the parameter
Root: INT	Number of the root node to which the parameter belongs
Flags: PARFLAGS	Special properties (not previously supported)

11.2.5 Properties of the MENUE parameter type

Use this parameter type to create selection fields in dialog boxes. A selection field is presented by an INT variable. You can select up to 20 different values, with each individual value being represented by a text.

Use the function block `CreateMenuPar()` to generate a selection field.

pValue: POINTER TO INT	Pointer to the variable (which also has to be defined in the IEC program) to be modified by the menu
ParId: INT	Number of the parameter (2...999)
Name: STRING	Name of the parameter: This text is presented in the ABxx dialog, together with the value of the parameter
Root: INT	Number of the root node to which the parameter belongs.
Flags: PARFLAGS	Special properties (not previously supported)
Items: ARRAY[1..20] OF INT	Numerical values from which a selection is made
ItemTexts: ARRAY[1..20] OF STRING	Texts for the numerical values

Example 1

The measured values of two channels are to be multiplied. The result is to be output at the analog output and checked to make sure that the value does not fall below the minimum nor exceed the maximum.

The following must be set:

P_{min} : lower limit

P_{max} : upper limit

Delete: Delete key

Rate: Output rate of the analog output

AnalogP1: Point 1 of the analog output characteristics

AnalogP2: Point 2 of the analog output characteristics

Within the display, the parameters are to be divided into the following groups:

<div style="border: 1px solid black; padding: 2px; display: inline-block;">Limits</div> <div style="border: 1px solid black; padding: 2px; display: inline-block;">Output</div>			
System	Display	Parameters	Options

Output dialog:
Selection field for "Limits" or "Output"

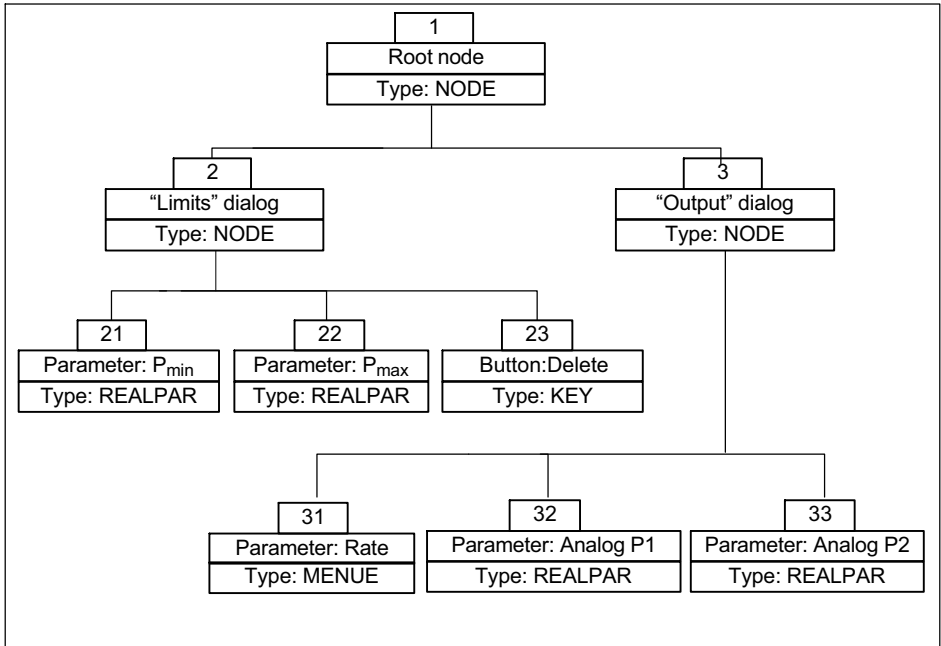
Limits		Channel 15.1	
Minimum output	2000	W	
Maximum output	10000 W		
<div style="border: 1px solid black; padding: 2px; display: inline-block;">Clear LEDs</div>			
System	Display	Parameters	Options

In the "Limits" window, the parameters P_{min} and P_{max} and the Delete key are presented

Output		Channel 15.1	
Output rate	2400	Hz	
Point 1 (0.0V)	0.000 Nm		
Point 2 (10.0V)	10.000 Nm		
System	Display	Parameters	Options

In the "Output" window, the output rate and points 1 and 2 are presented

The dialogs form the following tree structure:



In order to define a dialog of this type, you must encode the properties of the parameters in the IEC-61131-3 program. In the MGCplus.LIB library, you will find in the "Parametering ML70B" subgroup prototypes of functional process object units for describing the various parameter types.

You will need the following source code to encode the depicted parameter tree in the IEC program. This code must only be run through once. We recommend integrating this code with the INIT step of the measurement program (see Section 6.5).

An individual dialog (parameter group) comprises the node (here numbers 2 and 3) and the individual parameters.

Each parameter is linked to the node by the property "Root".

IEC61131-3 - Source Text

```

(* ----- "Limits" menu ----- *)

AttrNode2.pValue := ADR(Node2);    (* "Limits" menu node *)
AttrNode2.Name := 'Limits';      (* Display text *)
AttrNode2.ParId := 2;            (* Parameter number *)
AttrNode2.Root := 1;            (* Root node *)
AttrNode2();                    (* Generating the entry in the param. list *)

AttrPmin.pValue := ADR(Pmin);     (* Minimum torque *)
AttrPmin.ParId := 21;             (* Parameter number *)
AttrPmin.Name := 'Minimum output';(* Display text *)
AttrPmin.Root := 2;              (* Root node *)
AttrPmin.Decimals := 3;          (* Number of decimal places for indicators *)
AttrPmin.Unit := 'Nm';           (* Physical unit *)
AttrPmin();                      (* Generating the entry in the param. list *)

AttrPmax.pValue := ADR(Pmax);     (* Maximum force *)
AttrPmax.ParId := 22;            (* Parameter number *)
AttrPmax.Name := 'Maximum output';(* Display text *)
AttrPmax.Root := 2;             (* Root node *)
AttrPmax.Decimals := 3;         (* Number of decimal places for indicators *)
AttrPmax.Unit := 'Nm';          (* Physical unit *)
AttrPmax();                     (* Generating the entry in the param. list *)

AttrKey1.pValue := ADR(ClearLeds); (* "Clear LEDs key *)
AttrKey1.ParId := 23;            (* Parameter number *)
AttrKey1.Name := 'Clear LEDs';  (* Display text *)
AttrKey1.Root := 2;            (* Root node *)
AttrKey1();                    (* Generating the entry in the param. list *)

(* ----- "Output" menu ----- *)

AttrNode3.pValue := ADR(Node3);   (* "Output" menu *)
AttrNode3.Name := 'Output';      (* Display text *)
AttrNode3.ParId := 3;            (* Parameter number *)
AttrNode3.Root := 1;            (* Root node *)
AttrNode3();                    (* Generating the entry in the param. list *)

AttrMenuRate.pValue := ADR(Rate); (* "Output rate" selection menu *)
AttrMenuRate.ParId := 31;        (* Parameter number *)
AttrMenuRate.Name := 'Output rate';(* Display text *)
AttrMenuRate.Root := 3;         (* Root node *)
AttrMenuRate.Items[1] := 1;     (* Numerical value 1st menu entry *)
AttrMenuRate.ItemTexts[1] := '2400 Hz';(* Text 1st menu entry *)
AttrMenuRate.Items[2] := 2;     (* Numerical value 2nd menu entry *)
AttrMenuRate.ItemTexts[2] := '1200 Hz';(* Text 2nd menu entry *)
AttrMenuRate.Items[3] := 4;     (* Numerical value 3rd menu entry *)
AttrMenuRate.ItemTexts[3] := '600 Hz';(* Text 3rd menu entry *)
AttrMenuRate.Items[4] := 8;     (* Numerical value 4th menu entry *)
AttrMenuRate.ItemTexts[4] := '300 Hz';(* Text 4th menu entry *)
AttrMenuRate();                (* Generating the entry in the param. list *)

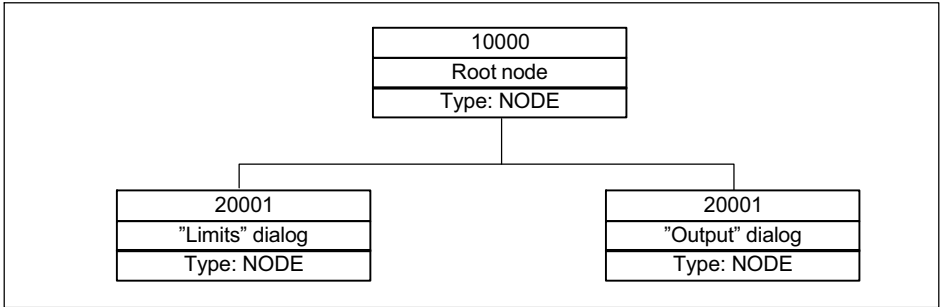
AttrP1.pValue := ADR(AnalogP1);   (* Characteristic point 1 analog output *)
AttrP1.ParId := 32;              (* Parameter number *)
AttrP1.Name := 'Point 1 (0.0 V)'; (* Display text *)
AttrP1.Root := 3;               (* Root node *)
AttrP1.Decimals := 3;           (* Number of decimal places for indicators *)
AttrP1.Unit := 'Nm';            (* Physical unit *)
AttrP1();                      (* Generating the entry in the param. list *)

AttrP2.pValue := ADR(AnalogP2);   (* Characteristic point 2 analog output *)
AttrP2.ParId := 33;              (* Parameter number *)
AttrP2.Name := 'Point 2 (10.0 V)'; (* Display text *)
AttrP2.Root := 3;               (* Root node *)
AttrP2.Decimals := 3;           (* Number of decimal places for indicators *)
AttrP2.Unit := 'Nm';            (* Physical unit *)
AttrP2();                      (* Generating the entry in the param. list *)
    
```

Example 2

With a user program, "Start" and "Stop" should be assigned to the function keys.

This example gives the following tree structure:



IEC Source Text

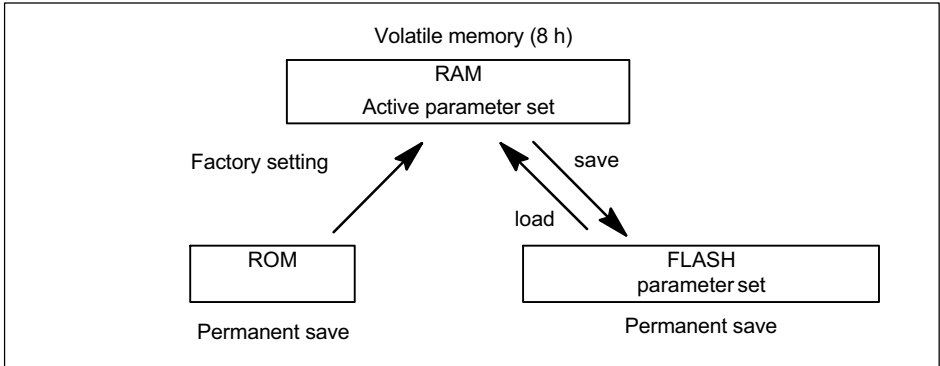
```

(* --- Function keys ----- *)
AttrNodeF.pValue := ADR(NodeF); (* Node F-keys *)
AttrNodeF.Name := 'F-keys'; (* Display text *)
AttrNodeF.ParId := 10000; (* Parameter number *)
AttrNodeF(); (* Generating the entry in the
parameter list *)
AttrKeyF1.pValue := ADR(FStart); (* "Start" key *)
AttrKeyF1.ParId := 20000; (* Parameter number *)
AttrKeyF1.Name := 'Start'; (* Display text *)
AttrKeyF1.Root := 10000; (* Root node *)
AttrKeyF1(); (* Generating the entry in the
parameter list *)
AttrKeyF2.pValue := ADR(FStop); (* "Stop" key *)
AttrKeyF2.ParId := 20001; (* Parameter number *)
AttrKeyF2.Name := 'Stop'; (* Display text *)
AttrKeyF2.Root := 10000; (* Root node *)
AttrKeyF2(); (* Generating the entry in the
parameter list *)
    
```

The functions are triggered from the ABxx, by incrementing the values of the variables Fstart or Fstop each time the key is pressed. The IEC program must therefore query the the values of the variables cyclicly in order to be able to execute the desired function.

12 Saving setup parameters

You can save the dialog parameters described in *Chapter 11* “Generating dialogs”, page 47. All the MGCplus modules save in accordance with the following principle:

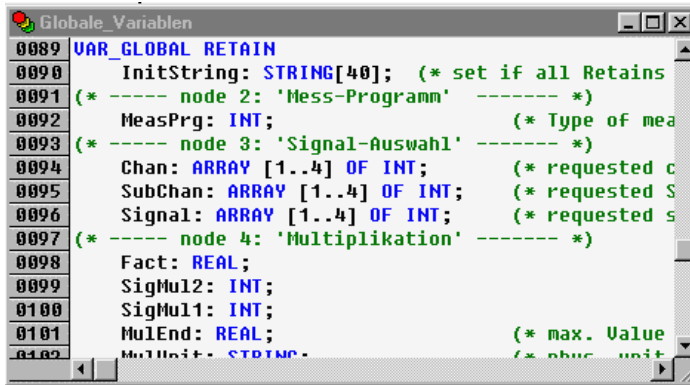


The settings of a module are always read from the parameter set in working memory (RAM), where they can be retained for up to 8 hours in the event of a power failure.

When the current parameter set is saved, a copy is stored in non-volatile memory (FLASH). When loading, the parameter set from the FLASH is copied to the working memory. When loading the parameters preset at the factory, these are loaded from the ROM into working memory.

The example which follows clarifies the effects of this principle of operation on your user program. The example used is the standard application in the `..\CodeSys..\Targets\HBM\ML70B_DemoPrj` directory.

All the variables that are to be saved permanently must be stored in the so-called "RETAIN area" as global variables. This area is *not* initialized when the MGCplus is switched off and on. Data saved here is saved for about 8 hours in the event of a power failure.



```

0089 UAR_GLOBAL RETAIN
0090   InitString: STRING[40]; (* set if all Retains
0091 (* ----- node 2: 'Mess-Programm' ----- *)
0092   MeasPrg: INT;           (* Type of mea
0093 (* ----- node 3: 'Signal-Auswahl' ----- *)
0094   Chan: ARRAY [1..4] OF INT; (* requested c
0095   SubChan: ARRAY [1..4] OF INT; (* requested S
0096   Signal: ARRAY [1..4] OF INT; (* requested s
0097 (* ----- node 4: 'Multiplikation' ----- *)
0098   Fact: REAL;
0099   SigMul2: INT;
0100   SigMul1: INT;
0101   MulEnd: REAL;           (* max. Value
0102   MulUnit: STRING;       (* phys. unit

```

If the display and control panel or an external interface (e.g. Assistant) sends the **Save** command to the ML70B, the ML70B operating system copies the RETAIN area to non-volatile memory. If the **Load** command is sent to the ML70B, the operating system copies the parameter set from the FLASH to the RETAIN area.

You must run the **Load factory settings** function in your program, as the operating system cannot be aware of the factory settings of the user program.

The IEC user program uses the system variables SYSV_TDDREQUEST and SYSV_TDDCMD to query whether a TDD Command (Save/Load) has been received.

```

Aktion RUN (ST) [-1/1/1/18] - PLC_PRG (PRG-AS) [-1/1/...
0005 IF SYSV_TDDREQUEST <> 0 THEN
0006   HandleTddCommand(); (* TDD bearbeiten *)
0007   SYSV_TDDREQUEST := 0;
0008 END_IF

HandleTddCommand (PRG-ST) [-1/1/1/1874]
0001 PROGRAM HandleTddCommand
0002 VAR
0003 END_VAR

0001 CASE SYSV_TDDCMD OF
0002   0: InitAllDlGVars(0); (* Werkseinstellung *)
0003   1: InitAllDlGVars(1); (* Laden *)
0004 END_CASE
0005 InitAktDlG(1);
0006

```

The operating system increments the variable `SYSV_TDDREQUEST` when a TDD command has been received.

The system variable is queried here cyclically. When a TDD command is received, the `HandleTddCommand` program is called.

The variable `SYSV_TDDCMD` shows the type of command received:

- 0: Load factory setup
- 1: Load
- 2: Save

The program also operates the "Load" command here, because if some of the parameters are changed, it becomes necessary to make additional internal program settings.

When saving the RETAIN area, the operating system calculates a check sum. When the MGCplus is switched on, the operating system first checks whether the RETAIN data check sum is correct. If the check sum does not conform, the parameter set from the FLASH is automatically copied to the RAM.



Information

But this design of program does not guarantee that the parameter set goes with the program currently being executed. The parameter set could, for example, be from a previously loaded program.

Which is why your program should start by running the following check:

```

Aktion Init (ST) [-1/1/1/20] - PLC_PRG (PRG-AS) [-1/164/1/9]
0001 (* create list of parameters *)
0002
0003 IF InitString <> 'StdAppData1.0' THEN (* init string ok *)
0004   RecallFlag := 0; (* no: perform factory setup *)
0005   InitString := 'StdAppData1.0'; (* Write init string *)
0006 ELSE
0007   RecallFlag := 1;
0008 END_IF
0009

```

If the string `InitString` does not contain a value suitable for the application, the factory settings are automatically executed so that all the parameter set variables are preset with meaningful values.

13 Multi-program mode

Several independent programs can be executed seemingly "simultaneously" (multitasking). The "Task configuration" entry is located in the "Resources" list. You can assign different programs to various tasks here. Up to 32 different tasks are possible. You can assign each individual task a call frequency in ms increments and a priority.

The precise description of task configuration operation can be found in CoDeSys online Help under **Contents -> Resources -> Task configuration**.



Information

The call frequency of the IEC program in the ML70B is 2400 Hz (see Section 6.1 "Time response", page 17).

The call frequency of the tasks has to be given in milliseconds. There is only equidistant measurement data transfer if the following formula for *Divisor* produces an integer value:

$$Divisor = \frac{2400 \text{ Hz}}{1000} \cdot Timeintervall_Task \text{ call [ms]}$$

The highest output frequency for precisely equidistant measurement output during multitasking is thus 200 Hz.

14 Special cases

14.1 Equidistant measurement output

In some applications, it may be necessary to output measured values at synchronized intervals (equidistant). The following steps are necessary to discover the maximum program turnaround time and then to define this time as the output rate.

1. Log in.
2. Delete the system variable `SYSV_MAXEXEETIME` in the "Global variables" window.
3. Start the program.
4. Read out the global variable `SYSV_MAXEXEETIME`.
5. Add the line `SYSV_REQEXEETIME := <value>` at the start of the IEC program. Insert the discovered value for `<value>`.

This defines the maximum turnaround time as the output rate.

The time is specified in increments of $1/2400 \text{ Hz} = 416.6 \mu\text{s}$.

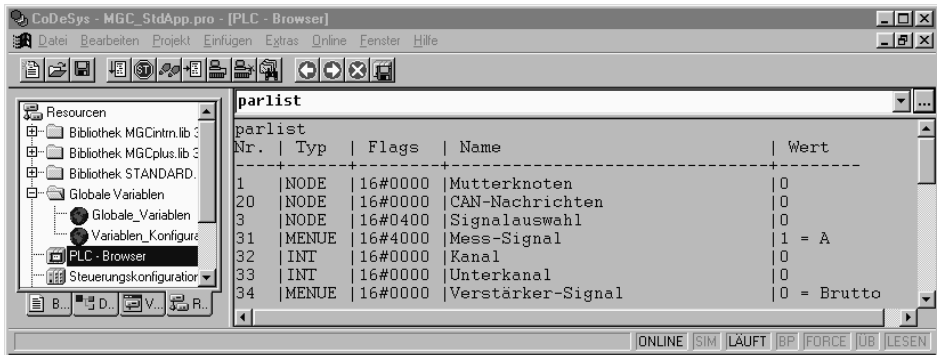
14.2 Changing the number of subchannels

You can choose between 1 and 128 subchannels as the number of subchannels for the ML70B.

Send the command **PAR9990, <number of subchannels> via the external interface**. Then switch the MGCplus off and back on again. The number of subchannels is now permanently set to the required value.

15 Debug functions (PLC browser)

The CoDeSys programming system has some debug functions to make troubleshooting easier in the programs. There is a PLC browser entry in the "Resources" list. The browser comprises a command line and a display window. The results of the commands entered in the command line are shown in the display window.



There is a detailed guide to browser operation in the CoDeSys online Help under **Contents -> Resources -> PLC browser**. More help for entering commands can be found via the button labelled "..." to the right of the PLC browser input line.

The following commands are available in the PLC browser:

Command	Description
comp	Compare memory
compd	Compare memory with memory dump
mem	Hex dump of a memory area
memc	Hex dump relative to the start address of the code on the control
memd	Hex dump relative to the database address on the control
memset	Set memory area: <start address>, <fill byte>, <length>
metrics	Display PLC metrics
reflect	Reflect current command line, for test purposes

Command	Description
dpt	Read data pointer table.
ppt	Read out POU table.
pid	Read project ID.
pinf	Read project info
?	Output list of available commands
cycletimes	Display of cycle times of the program on the ML70B
parlist	Display of ML70B parameter list
signals	Display of list of all signals requested by ML70B

16 System variables

Variable name	Type	Meaning
SYSV_ActualExecTime	WORD	Execution time of the last step measured in time increments of 1/2400 Hz = 416 μ s
SYSV_MaxExecTime	WORD	Max. execution time of all the last steps measured in time increments of 1/2400 Hz = 416 μ s
SYSV_ReqExecTime	WORD	Execution time of the last step measured in time increments of 1/2400 Hz = 416 μ s
SYSV_ParChange-FromML70B	WORD	Each time a parameter is changed from outside (ext. interface CP or AB), incremented by ML70B
SYSV_ParChangeFromIEC	WORD	
SYSV_TddRequest	WORD	Incremented when a TDD command is received from outside (ext. interface CP or AB)
SYSV_TddCmd	WORD	Last received TDD command
SYSV_TddPar	WORD	Last parameter set received with TDD

17 Error messages

Many of the functions from the MGCPLUS.LIB library return error codes that can give a more accurate insight into the cause of the error. Negative values are errors, positive values are status information.

Error code	Meaning
0	No error
-100	Error when calling RemovePar(): this parameter does not exist
-101	The root node specified does not exist
-102	Storage overflow when defining parameter
-103	Too many items in node
-104	Item list already empty
-105	Cannot delete item (1 and 20)
-106	Root node 20 not permitted!
-107	Node numbers 1 and 20 are reserved
-108	Incorrect parameter number
200	ActivateSignals() still running
-210	OpenComPort(): incorrect PrtNr: this serial interface does not exist
-211	OpenComPort(): incorrect parity info
-212	OpenComPort(): incorrect stop bit number
-213	OpenComPort(): baud rate does not exist
-214	OpenComPort(): incorrect data bit number
-215	Com port not open
-216	ReadCom() BUFF parameter: invalid pointer
-217	Timeout exceeded during serial receive
-218	Line length exceeded during serial receive
-219	Receive error at serial interface (parity, framing or overrun error)
-220	Hardware handshake only possible for RS232
-221	Software handshake not possible for RS485
-230	Incorrect channel number

Error code	Meaning
-231	Incorrect subchannel number
-233	SetScaling(): Endsacle 0 not permitted
-234	SetScaling(): 0...5 decimal place permitted
240	Wait for response
-241	Command protocol already running
-242	Timeout at internal MGC interface
-250	Too many signals requested with RequestSignal()
-251	ActivateSignals(): signal not available
-260	SetAnalogOutput(): analog output does not exist

HBM Test and Measurement

Tel. +49 6151 803-0

Fax +49 6151 803-9100

info@hbm.com

measure and predict with confidence



A00863_05_E00_00 7-2002.0573 HBM: public

www.hbm.com