



HEAD OFFICE

INMOTION TECHNOLOGIES AB

Solkraftsvägen 13
SE-135 70 Stockholm
SWEDEN
Telephone: +46 (0)8 682 64 00
Telefax: +46 (0)8 682 65 80

Info@inmotech.com

<http://www.inmotech.com>

APPLICATIONS CENTERS

GREAT BRITAIN

Inmotion Technologies
/Danaher Motion
Bridge Mills
Holmfirth
Huddersfield
HD9 3TW
Telephone: +44 (0)1484 68 83 25
Telefax: +44 (0)1484 68 83 26

GERMANY

ACC Motion GmbH
Zähringerstrasse 23
DE-77654 Offenburg
Telephone: +49 (0)781 919 08-0
Telefax: +49 (0)781 919 08-29

ITALY

Danaher Motion Srl
Via Brughetti Z.I.
IT-20030 Bovisio Masciago (MI)
Telephone: +39 0362 594 260
Telefax: +39 0362 594 263

SWEDEN

Inmotion Technologies AB
Solkraftsvägen 13
SE-135 70 Stockholm
Telephone: +46 (0)8 682 64 00
Telefax: +46 (0)8 682 65 80

Inmotion Technologies AB
Box 195
SE-234 23 Lomma
Telephone: +46 (0)40 41 48 50
Telefax: +46 (0)40 41 48 55

SWITZERLAND

ACC Motion SA
Wehntalerstrasse 6
CH-8154 Oberglatt
Telephone: +41 (0)1 851 5010
Telefax: +41 (0)1 851 5020

ACC Motion SA
La Pierreire
CH-1029 Villars-Ste-Croix
Telephone: +41 (0)21 863 6464
Telefax: +41 (0)21 863 6479

U.S.A.

Inmotion Technologies
211 Overlock Drive
Sewickley, PA 15143-2305
Telephone: +1 (412) 749 0710
Telefax: +1 (412) 749 0705

DMC²

Digital Motion Control System
Part B

User's Manual 5.1

Art.No. 9032 0027 01 (B)

11.07.2001

Inmotion Technologies AB
Stockholm, Sweden

© Inmotion Technologies AB, 2001.
All rights reserved.

Table of Contents

Table of Contents.....	3
Software System.....	11
General.....	11
New features DMC ² 5.0.....	11
Remarks.....	11
Definitions.....	11
System architecture.....	12
DMC ² software development.....	12
Timing and execution flow.....	14
Event Triggered Programming.....	15
DMC ² PL2 SW Flow.....	15
Function Block Diagram.....	16
Load a new firmware release.....	21
Boot mode command.....	22
Startup message.....	22
PL2 Native position language.....	25
Introduction.....	25
General.....	25
Argument types.....	26
Abbreviated argument types.....	26
Expression Operators.....	26
Arithmetic Operators; Performs an Arithmetic Operation (32 bit signed operations).....	27
Equality and Relational Operators; Perform a Test Operation.....	27
Relational Circular Operators; Perform a Test Operation.....	27
Binary Operators, Perform a Binary Operation.....	28
Mnemonic Operators.....	28
Scaling mnemonic operators.....	29
Compiler Extensions.....	29
Compiler Symbols.....	30
Predefined compiler symbols.....	31
Special Compiler Functions.....	31
Compiler directives.....	32
Spline function compilation directives.....	36
Conditional compilation directives.....	37
Multiline Macro.....	37
Macro Definition.....	38
Macro Arg Specifiers.....	38
Macro Call (Expansion).....	39
PL2 Mnemonics.....	41
General.....	41
Standard set Mnemonics.....	41
Trace Related Mnemonics.....	45
LAN1/ LAN1 Related Mnemonics.....	45
Anybus-S Related Mnemonics.....	46
Indexed Addressing Mnemonics.....	47
Text mode.....	49
Text output.....	49
Text input.....	51
Extended register groups.....	53

Introduction.....	53
Group members. (group xx).....	54
Stack handling.....	55
General.....	55
Function.....	55
Related Items.....	55
Example Usage.....	55
Group members (group 1).....	56
RD1, Resolver/Digital converter #1.....	57
General.....	57
Function.....	57
Related items.....	58
Example usage.....	58
Group members (group 2).....	58
RD2, Resolver/Digital converter #2.....	63
RD2, General.....	63
Function.....	63
Related items.....	64
Example usage.....	64
Group members (group 3).....	64
Pg, Profile generator.....	68
General.....	68
Function.....	68
Related items.....	68
Example usage.....	69
Improvements to 4.0.....	69
Modify destination position while positioning.....	69
Move to a target position 'behind' our current position.....	70
Very short motion profiles and/or very high deceleration rates.....	70
Group members (group 4).....	70
Motor, Motor interface.....	73
General.....	73
Function.....	73
Related items.....	73
Example usage.....	73
Group members (group 5).....	74
Induction motor specific members.....	76
Reg, PID regulator.....	79
General.....	80
Function.....	80
Related items.....	80
Example usage.....	80
Group members (group 6).....	81
Gear, Electronic gearbox.....	86
General.....	86
Function.....	86
Related items.....	87
Example usage.....	87
Position lock CAM.....	87
Time locked CAM.....	88
Master/Slave.....	88
Incremental CAM.....	88
Group members (group 7).....	89
Tmr, System timers.....	93
General.....	93
Function.....	93

Related items	93
Example usage	93
Group members (group 8)	94
Syslo, System I/O	96
General	96
Function	96
Related items	97
Example usage	97
Group members (group 9)	97
Int, Interrupt control	103
General	103
Function	103
Related items	103
Example usage	104
Group members (group 10)	104
In, Digital inputs	109
General	109
Function	109
Related items	109
Example usage	109
Group member (group 11)	110
Out, Digital outputs	112
General	112
Function	112
Related items	112
Example usage	112
Group members (group 12)	112
Vector, Interrupt vectors	114
General	114
Function	114
Related items	114
Example usage	114
Group members (group 13)	115
CAPTURE, Capture exact time of external events.....	118
General	118
Compatibility DMC1 to DMC ²	118
Function	119
Hardware change.....	119
Related items	119
Example usage	119
Group members (group 14)	120
Ana, Analog I/O	124
General	124
Function	124
Related items	125
Example usage	125
Group members (group 15)	125
EEprom	127
General	127
Function	127
Related items	128
Example usage	128
Group members (group 16)	128
Comm, Serial communication.....	130
General	130
Function	130

Related items	130
Example usage	130
Group members (group 17)	131
RD1Corr, Position corrector	133
General	133
Function	133
Related items	133
Example usage	133
Group members (group 18)	134
OptAD, analog to digital converter.	135
General	135
Conversion resolution	135
Calibration	135
Example usage	135
Amplifier gain setting.....	136
Example usage	136
Group members (group 22)	136
LAN1, Local area network 1	139
LAN1, Interrupt handling	139
LAN1, Double Buffering	140
LAN1, Specific Instructions.....	140
LAN1, Remote Frames in CAN.....	141
LAN1, Power Up	142
LAN1, High level communication protocols.	142
LAN1 communication scenarios.	142
Group members (group 28)	143
MsgObjLAN1, Helper for LAN1	149
Group members (group 29)	149
LAN2, Local area network 2	152
General	152
LAN2 commands	152
MsgObjLAN2, Helper for LAN2	153
MsgObjLAN2 commands	153
MultDiv,	154
General	154
Function	154
Related items	154
Example usage	154
Group members (group 49)	154
FlashMem.....	156
Group members (group 50)	156
ABIn	157
General	157
Function	157
Related Items.....	157
Example Usage.....	157
Group members (Group 52).....	157
ABOut	158
General	158
Function	158
Related Items.....	158
Example Usage.....	158
Group members (Group 53).....	158
DStore,.....	159
General	159
Function	159

Related items	159
Example usage	159
Group members (group 54)	160
ParArea	161
General	161
Function	161
Related items	161
Example usage	161
Group members (group 55)	161
XENDAT	163
General	163
Function	163
Manual mode	163
Automatic Serial Mode	163
Analog mode only	163
Combined serial and analog mode	163
Related items	164
Example usage	164
Group members (group 56)	165
Counter	172
General	172
Function	172
Related Items	172
Example Usage	172
Group members (Group 57)	172
Identifier	175
General	175
Function	175
Related Items	175
Example Usage	175
Group members (Group 58)	175
RDPDATA	176
General	176
Function	176
Related Items	176
Example Usage	176
Group members (Group 59)	177
SAnyBus	178
General	178
Function	178
Related items	178
Example usage	179
Group members (group 60)	183
AnyBus related PL instructions	186
Manipulate the AnyBus input buffer	187
Manipulate the AnyBus output buffer	187
Transfers the AnyBus input buffer	187
ABInMail	188
General	188
Function	188
Related Items	188
Example Usage	188
Group members (Group 61)	188
ABOutMail	189
General	189
Function	189

Related Items.....	189
Example Usage.....	189
Group members (Group 62).....	189
ABFBus.....	190
General	190
Function	190
Related Items.....	190
Example Usage.....	190
Group members (Group 63).....	190
EN1-EN4, Encoder1-4.....	191
General	191
Backward compatibility note	191
Function	191
Related Items.....	192
Example Usage.....	192
Group members (Group 64-67)	192
IENC.....	193
General	193
Function	193
Related Items.....	193
Example Usage.....	193
Group members (Group 69).....	193
ModEn3-ModEn4.....	195
General	195
Function	195
Related Items.....	195
Example Usage.....	195
Group members (Group 72-73)	195
Communication protocol	199
Introduction.....	199
Protocol Format.....	199
Description	199
Computer Mode.....	199
PL2 On line commands.....	203
PL2 On line commands	203
Command Line Editor (CLE)	204
ECT.....	207
Introduction.....	207
Definitions.....	207
Running ECT	207
The ECT desktop.....	207
The ECT main menu.....	208
File	208
Project.....	208
Options.....	208
Window	208
Help.....	208
Using the text editor	209
Selecting text	209
Edit.....	209
Accessing the Edit Application.....	210
When Editing Existing Files	210

Creating/Editing Source Code	211
Exiting Edit	212
Compile.....	212
Introduction	212
Accessing Compile	212
Setting up the Compiler	212
Compiling source code	213
Compiler output.....	213
Test.....	214
Introduction	214
Test setup	214
Test function	215
Accessing the test system	216
Test menu	216
File	216
Controller	216
Exiting the test enviroment.....	217

Software System

GENERAL

This section is intended to introduce the user to the fundamentals of the DMC² programming and software system. It contains the following parts:

- Definitions Important terms are explained.
- System architecture Overview of the DMC² software system.
- Timing and execution flow Timing and flow aspects important to the programmer.
- Event triggered programming The recommended programming technique for the DMC².
- Function block diagram Functionality blocks are described.
- Load New Firmware Download new Firmware to DMC².

NEW FEATURES DMC² 5.0

- Increased number of program lines , 8191.
- Increased number of user registers , 4096.
- Fieldbus support using Anybus-S modules from HMS.
- Endat interface for absolute encoders.
- Incremental encoder interface.
- Counter function for external event counting.
- Inverted conditional operators such as IfNot,IFAbsNot,added.
- Indexed subroutine call added.
- More connection possibilities for debug use.
- Software definition of rotational direction.
- Multiple commutation sources.

REMARKS

- The PL execution speed is much higher in a DMC2 (10 – 15 lines / servocycle), than in a DMC1, thus it is essential that PL2 program use explicit lines to wait for hardware, this may not have been a problem in the DMC1 because execution speed was between 4 and 1 PL lines/servo-cycle.
- DMC1 and new DMC² can only be synchronized with respect to I/O. The resolver can not be interchanged between DMC1 and new DMC², because the resolver system operates differently.

DEFINITIONS

PL2	The proprietary programming language for creating DMC ² application programs. Consists of PL2 statements. The language is register-based and line oriented. Most functions are accessed by manipulation of register values. Resides in FLASHROM
Firmware	The system software of the DMC ² , performing hardware manipulation, interpretation of the PL2 code and execution of predefined functions.
Function block	Firmware functions that perform a certain predefined user function, e.g. creating a movement profile, based on preset

System architecture

	register values and PL2 statements.
Compiler	PC software tool running on the PC to compile (translate) a PL2 statement text file (.pl2) to a binary format file (.hee) suitable for transmission to the DMC ² . Communication between the PC and the drive is assumed to be in the binary download format.
Terminal mode compiler	Line oriented compiler running in the DMC ² firmware allowing the user to modify and insert statements in the application program in the DMC ² . NOTE that the interpreter performs functions similar to the compiler but on a line-by-line basis. It is executed in the drive itself and is accessed online.
Interpreter	The interpreter executes as part of the DMC ² firmware and interprets PL2 commands. If an application program is running, statements are sequentially interpreted from the DMC ² application program memory. If it is not running the interpreter still reacts to online statements as described above under terminal mode compiler.

SYSTEM ARCHITECTURE

DMC² SOFTWARE DEVELOPMENT

The DMC² software system can be divided into two main parts, firmware (FW) and PL2 code.

- The FW (firmware) functions as a computer operating system. It manages all direct interactions with the hardware and provides the application programmer with uniform, high level programming tools. The FW is stored in part of the FLASH in the DMC².
- PL2 code is the instructions written by the application programmer to achieve application specific behavior of the DMC². It is created as a text file in a PC environment, compiled and downloaded to the DMC and stored in the FLASHROM for finalized application programs. While in text format the PL2 file may contain extensive comments and explanations. The downloadable files, however, are in a pseudo-machine language, stripped of all comments and labels.

The application programmer uses the following tools to create an application:

- ECT, Edit-Compile-Test, software package for PC-compatible computers. Allows the user to create application program text files, compile them and download the binary files to the DMC² as illustrated in [Figure 1](#).
- ECT includes facilities for working on-line with the DMC². This combined with the DMC's built-in line compiler may sometimes be a useful complement to the normal development method.

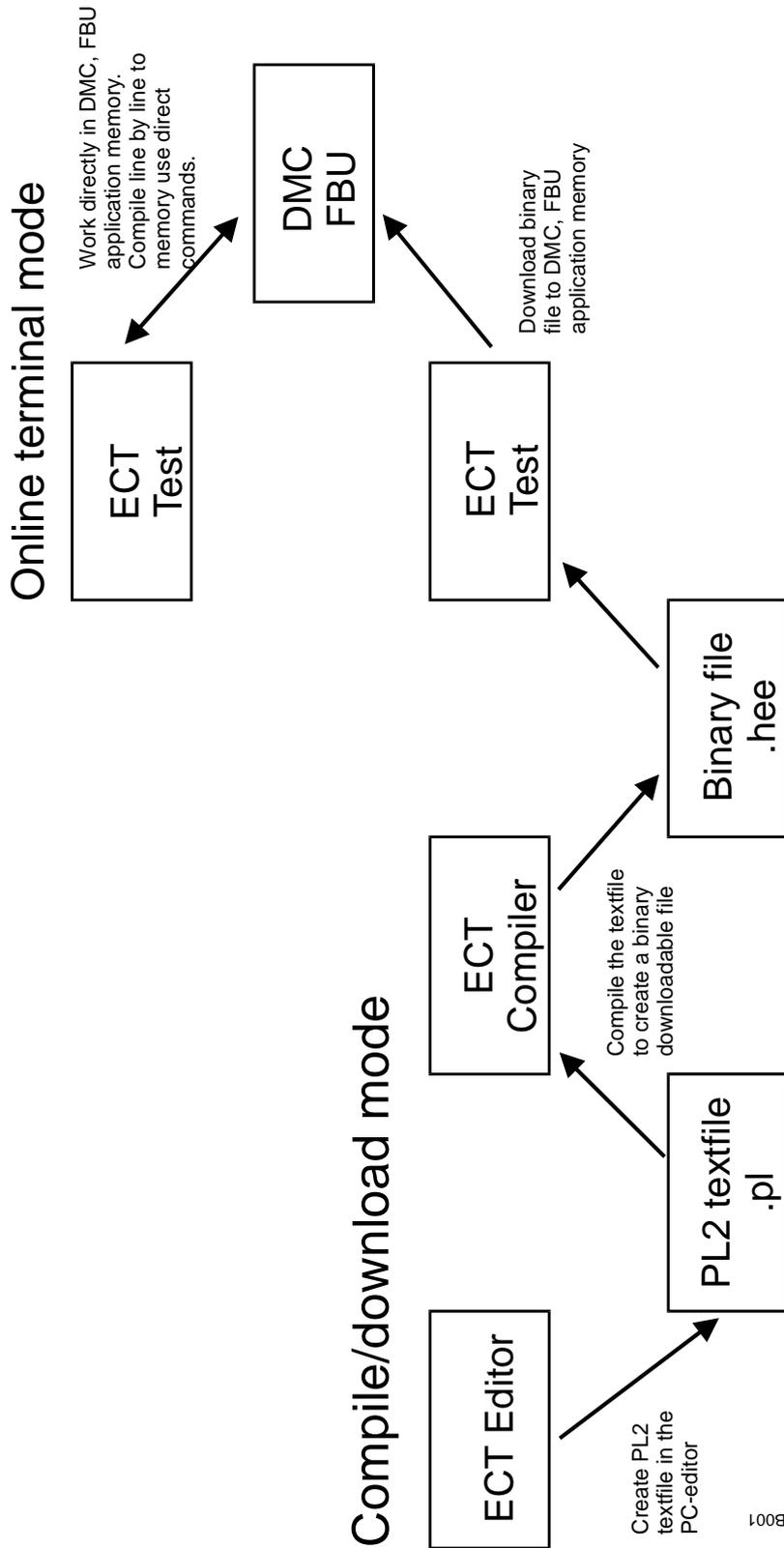


Figure 1. DMC² software development cycle.

TIMING AND EXECUTION FLOW

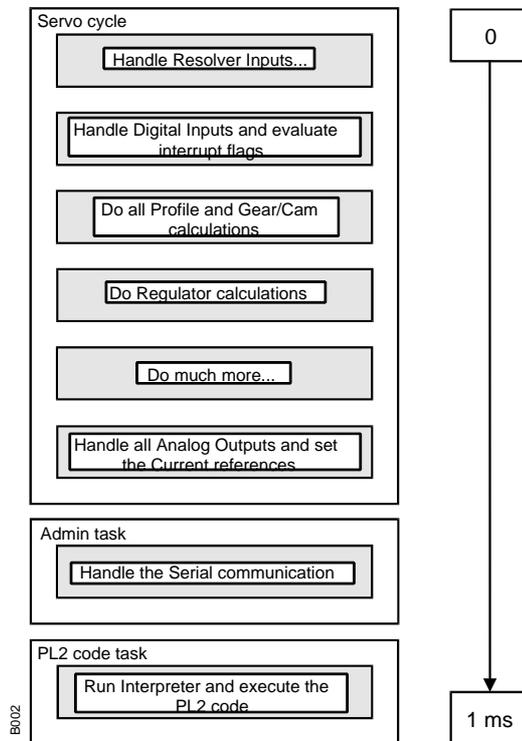


Figure 2. DMC servo cycle.

The firmware executes in two ways:

- One is clock dependent, executing once each system clock cycle. This is called a servo cycle. (Figure 2.). It handles all real time critical tasks, such as velocity and position measurement, regulators, motion profiles, cams and gear functions. It also updates real time outputs, i.e. analog outputs for monitoring purposes.



These functions are always performed, regardless of whether an application is running or not.

- The rest of the firmware execution is done in background. One task is handling the serial communication. Another task is interpreting and executing the PL2 code.

It is important to realize that most real-time dependent functions are completely executing in FW. Consider the profile generator. It produces new values every ms, but it does not require any PL2 involvement once the initial profile statement has been executed. From this point onwards, the FW is executing all related calculations within the servo cycle and the PL2 code may perform other tasks concurrently. Another example is the output of real time data to the analog outputs. Once the PL2 code has established a "connection", for example, from actual speed to an analog output, the FW performs the real time update of the analog output.

EVENT TRIGGERED PROGRAMMING

DMC² PL2 SW FLOW

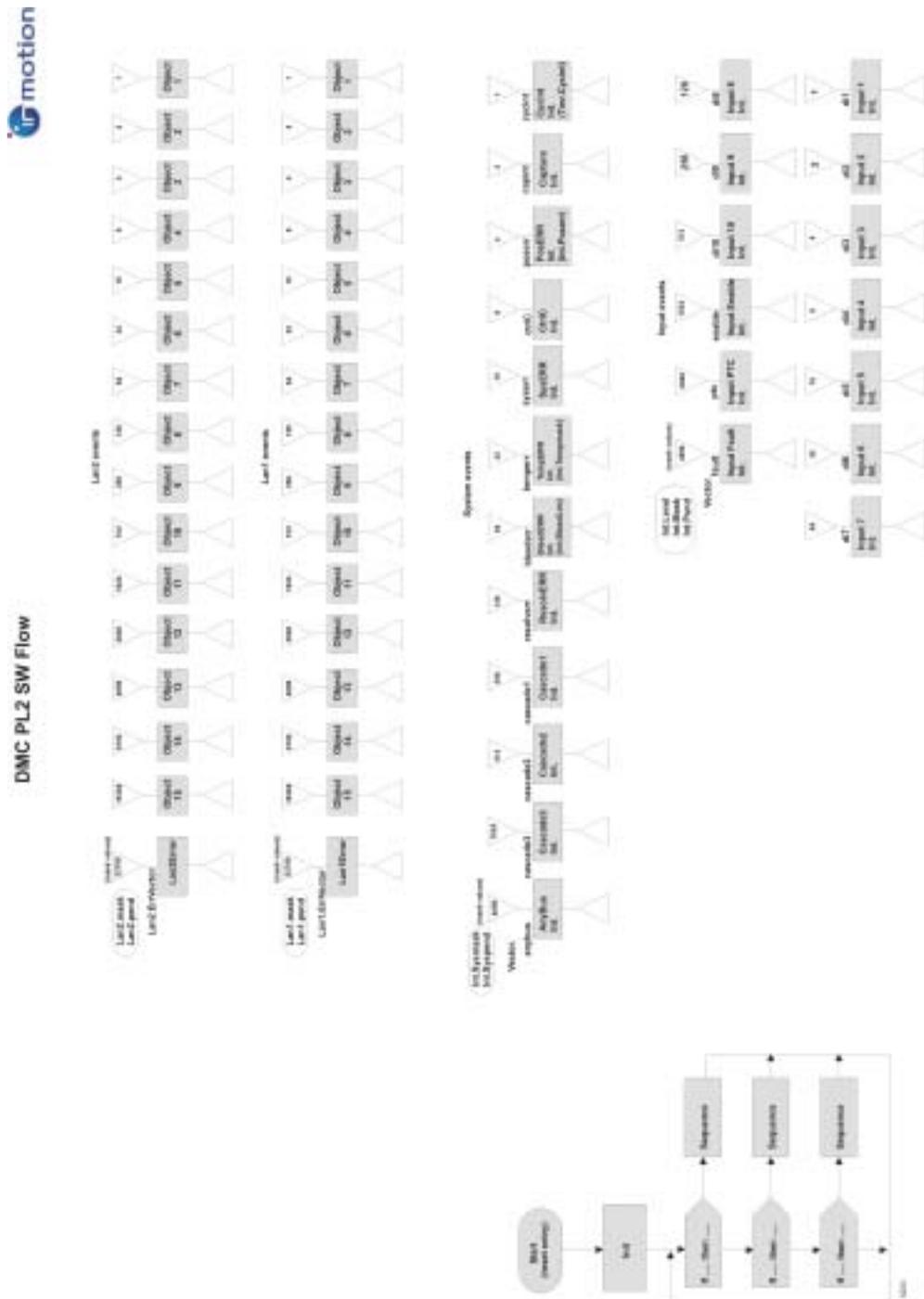


Figure 3. DMC PL2 Software flow.

Function Block Diagram

When programming a PL2 application an event triggered programming technique is recommended. [Figure 3](#) illustrates the principle.

- After startup of the system when the PL2 code initially gains access to the processor, a number of initializations normally take place. These may include setting up motor and resolver parameters, regulator and interrupt system.
- After setup program execution enters some main loop. The main loop may contain nothing. Instead all action to be taken may be triggered by interrupts.
- Special interrupt routines are executed for each detected interrupt. In this way the program assumes a function oriented structure which is easier to maintain. There are justified exceptions to this principal.
- For example, operator interface functions may be handled in the main loop.
- It is important to realize the meaning of interrupt in the PL2 environment. An interrupt does alter the natural flow of PL2 code execution, but it is not the direct result of a change of sequence flow estate of the hardware.
- Most PL2 interrupts are generated by the FW, sometimes in reaction to a hardware interrupt, but more often by polling the hardware status each servo cycle.
- A PL2 interrupt response time is therefore always approximately 1ms.
- The predictable interrupt response time is yet another advantage of event triggered programming.
- The rate of PL2 statements executed varies with the complexity of the statements and the number of real time functions active in the servo cycle. It is not advisable to base any real time related functions on the execution times of PL2 code.

FUNCTION BLOCK DIAGRAM

Each function blocks in [Figure 4](#). is explained in the following tables. For each function block the related register set, the input signals or "trigger" to the block and the resulting output (or what is affected) from the block are listed.

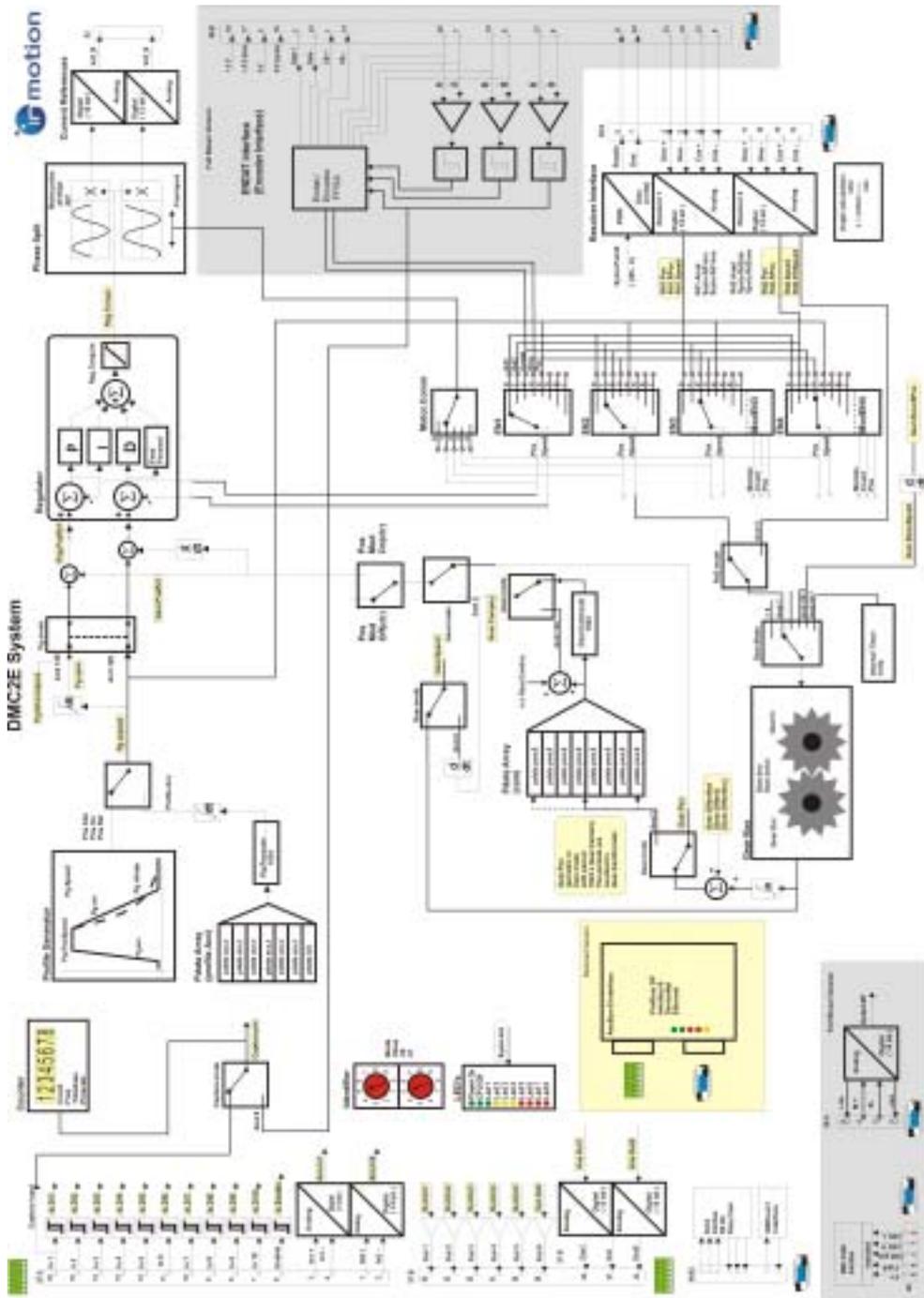


Figure 4. DMC functional block diagram.

Function block	Register set	Input/ Trigger	Output/ Affecting	Description
Digital inputs	In	Physical input	PL2 code execution flow.	Input handling to the DMC ² .
Digital outputs	Out	PL2 statement	Physical outputs.	Digital output handling from the DMC ² unit.
Profile generator	Pg	PL2 statement	Speed and position set commands.	Calculates the momentary desired values of speed and position.
Regulator	Reg	Speed and position set points and resolver input	Current set point, (momentary demanded torque).	Controls the motor speed and position according to the momentary set points and the measured speed and position. Compensates for any speed or position error.
Motor control	Motor	Register values	Motor drive adaptation.	Adapts the motor drive to the electrical characteristics of the motor.
System inputs	SysIo	Alarm and safety	PL2 code execution.	Allows the PL2 code to detect and respond to alarm and safety signals generated in hardware protection systems.
System outputs	SysIo	PL2 statement	HW affected in the unit	Allows certain hardware functions (i.e. enabling the power stage or activating brake relay or LEDs) to be controlled by the PL2 code.
Analog inputs	Ana	Analog signals	PL2 accessible values	Allows the PL2 code to monitor analog signals either by direct reading or by "connecting" it to an internal variable.
Analog outputs	Ana	PL2 statement	Value on analog output pins	Allows the PL2 code to set analog outputs to specific values or "connect" it to an internal variable.
Resolver	RD1 , RD2	Physical resolver input	Velocity and position of motor or auxiliary resolver	Allows the FW system and the PL2 code to access position and velocity, not only of the controlled motor but also of one auxiliary resolver.
Gear box	Gear	Auxiliary resolver	Set position	Allows the motor set position to be controlled by the auxiliary resolver speed and position, in such a way that an electronic gearing is achieved.
Pdata acc	None	Timer ticks	Set acceleration	Pdata acc is a data array that allows the PL2 programmer to

Function Block Diagram

Function block	Register set	Input/ Trigger	Output/ Affecting	Description
			and velocity	create his own acc / decel profiles. Each cell of the Pdata array contains the desired acc and the number of ticks during which this should be maintained.
Pdata cam	None	Auxiliary resolver or timer ticks	Set position and velocity	Pdata pos allows the PL2 programmer to create electronic cams. For each position of the aux. resolver, the desired position of the motor is maintained.

The following table explains block functions merely providing support to the PL2 programmer. These function blocks are less related to hardware functions:

Function block	Register set	Input/ Trigger	“Output/ Affecting	Description
Stack	Stack			Provides a temporary numbers storage facility to the PL2 programmer. This is a complement to conventional register storage.
Timer system	TMR			The timer system provides timing and delay facilities to the PL2 programmer.
Interrupt system	Int			The interrupt system allows the PL2 programmer to enable or mask certain interrupt sources and control interrupt trigger criteria, i.e. positive or negative edge of an input signal.
Interrupt vectors	Vector			The interrupt vectors direct program execution to the desired interrupt service routine, upon interrupt detection.
Input capture	Capture			The input capture function is allowing the PL2 code to measure the exact time, when an edge was detected on the high speed input.
Parameter storage	EEProm			For non-volatile storage of PL2 software parameters, EEPROM hardware is provided.
Communication	Comm			Allows the PL2 programmer to customize the serial communication parameters.

Function block	Register set	Input/ Trigger	“Output/ Affecting	Description
Resolver correction	RD1Corr			The RD1Corr register set is used to compensate for physical resolver imperfections. The calculation of these values may be done automatically.
Local Area Network 1	LAN1			Interface to the Local Area Network 1.
Can message descriptor 1	MsgObjLAN1			Can message descriptor temporary storage.
Local Area Network 2	LAN2			Interface to the Local Area Network 2.
Can message descriptor 2	MsgObjLAN2			Can message descriptor temporary storage.
	Muldiv			Math scaling with 64 bit protocol
	FlashMem			Protocol
	AnyBus			Fieldbus interface
	DStore			Data storage
	Par area			Non violated parameter storage
	Counter			
	RDPDATA			
	Identifier			
	EN1- EN4			
	XENDAT			
	IENC			

LOAD A NEW FIRMWARE RELEASE

The PL command 'BOOT' can be used to enter boot mode, when an existing copy of firmware already resides in the flash.

1. Turn power to the drive off.
2. Short pin 2 and pin3 on connector X5 or X4 (serial communication).
(This will echo content sent by the drive back to the drive).
3. Apply power to the drive.
4. Remove short applied at step 2.

Load a new firmware release

5. Without removing power to the drive connect to the ECT terminal emulator.
Warning: Make sure that both the DMC² and the PC is earth grounded failure, to do so may result in damage the DMC² unit and/or the PC!
6. The BOOT monitor should have been entered.
First, the current program must be erased, to do that type,
 >EA
or
 >EF
EA - Erase all.
EF - Erase firmware (currently same as EA).
H - Help.
When command has finished then next step.
7. Select CONTROLLER: DOWNLOAD from the menu and select the new firmware file
(firmware extension is *.hex).
8. Wait for download to finish
9. The new firmware should now be stored into the flash unless reported otherwise, turn the power off or type,
 >RF
RF - Run firmware
10. Do the command NEW before downloading the application program again.
Note. Any stored PL program will have been erased from the flash.

BOOT MODE COMMAND

With the addition of the 'boot' command a user can enter the boot mode from the terminal. This simplifies the above procedure to,

1. Turn power to the drive on
2. Type 'boot' when the '>' prompt is seen.
 >boot<ENTER>
3. The unit has entered 'BOOT MODE'. Continue from step 6 in the above list.

STARTUP MESSAGE

When power is applied to the DMC² the following startup message is displayed,

```
ACC DMC / Inmotion Technology AB v05.02.00 Node#:1 Baud: 9600
Mode: 0
```

This line is always sent using 9600 baud independent of the settings in the eeprom related to the COMM group or any programming of the COMM group. The line gives information about the content of the EEPROM used for initialization of the serial communication (COMM group).

The assignments made to the COMM group by the firmware at startup is:

Node	COMM.Node = EEPROM.6 and 15.
Baud	If EEPROM.4 = 0 then COMM.Baud = BaudTable[EEPROM.6 >> 8] else COMM.Baud = EEPROM.4

Mode	<code>COMM.Mode = EEPROM.7 and 15</code> <code>COMM.TLines = EEPROM.7 >> 8</code> BaudTable is a table of 8 baudrate values, 50..9600
------	---

If there appears to be a problem communicating with the DMC², then check the startup message first to verify that the initial settings are satisfactory.

PL2 Native position language

INTRODUCTION

This manual pertains to programming DMC² motion controller products and contains a language reference that explains instructions, extended registers, language syntax and compiler functions.

Typefaces used in this manual:

<i>Mono spaced</i>	This typeface represents text as it appears on the screen (in ECT) or in a program and is also used to indicate items the programmer may type.
<i>Italics</i>	Italics are used to emphasize certain words, such as new terms.
Bold	PL2 reserved word

GENERAL

Program A Program is comprised of a collection of Program Statements.



The DMC² product family allows a Program to be stored in volatile memory,(RAM) or in nonvolatile memory (FLASHPROM). The program can also be a file that is stored on a computer when the development environment (ECT) is used.

Program Statement

A program statement is one line of text that contains a mnemonic for an instruction. Each instruction has its own syntax. (The compiler checks for the correct syntax. The controller has a built in one-line compiler.)

Line

The sequential number where a program instruction is stored. (1-8191)

Mnemonic

A mnemonic is a text synonym for an instruction. The DMC² family of products directly interprets Mnemonics.
Note: Occasionally, instructions refer to the Mnemonic for the instruction.

Mnemonic Operator

An operator used together with a Mnemonic to define the Instruction.

Argument

One or several arguments are used with a Mnemonic to define the Instruction.

Expression Operator

Used with a Mnemonic and Argument(s) to define an action that should take place during execution of the Instruction.

Routine

A part of a program, usually designated to perform a certain function, is called a Routine.
A Program is typically made up of many Routines.

Parameter stack

Part of the memory organized as a FILO for parameter storage.

[Mnemonic]

Designates an optional Mnemonic.
Example: The Let Mnemonic can usually be omitted.

Syntax

There are different ways to combine a Mnemonic together with arguments and an operator. This will yield different

Argument types

executing results.

The typical form for a PL program Statement is:

```
100      Let      R7 = Pg.speed / 112
```

[Line] Mnemonic [Mnemonic Operator] [Argument] [[Operator] [Argument]]

ARGUMENT TYPES

<i>SConst</i>	Short Constant, 16-bit. Can have any value between -32768 and 32767. However, not all statements use all bits.												
<i>LConst</i>	Long Constant, 32-bit. Can have any value between -2147483648 and 2147483647. However, not all statements use all bits.												
<i>Reg</i>	Internal 32-bit, ordinary register. Can be accessed directly as Rnnn or indirectly as R(Rnnn) (where nnn is the register number). Example: <pre>[Let] R220 = 17 [Let] R(R220) = 4711 ;This will set R17 =4711</pre>												
<i>XReg</i>	Designates an Extended Register. The XReg is usually an ordinary register or a system variable. Some system variables are read-only. An attempt to write to a read-only system variable has no effect. System variables are formed into groups such as, <table> <tr> <td><i>Pg</i></td> <td>Profile Generator</td> </tr> <tr> <td><i>Motor</i></td> <td>The motor interface</td> </tr> </table> <p>Each group has members such as:</p> <table> <tr> <td><i>Motor.Mode</i></td> <td>Type of motor.</td> </tr> <tr> <td><i>Motor.Comm</i></td> <td>Commutation source.</td> </tr> <tr> <td><i>Motor.Poles</i></td> <td>Number of motor poles.</td> </tr> <tr> <td><i>Motor.PPR</i></td> <td>Pulses per revolution.</td> </tr> </table>	<i>Pg</i>	Profile Generator	<i>Motor</i>	The motor interface	<i>Motor.Mode</i>	Type of motor.	<i>Motor.Comm</i>	Commutation source.	<i>Motor.Poles</i>	Number of motor poles.	<i>Motor.PPR</i>	Pulses per revolution.
<i>Pg</i>	Profile Generator												
<i>Motor</i>	The motor interface												
<i>Motor.Mode</i>	Type of motor.												
<i>Motor.Comm</i>	Commutation source.												
<i>Motor.Poles</i>	Number of motor poles.												
<i>Motor.PPR</i>	Pulses per revolution.												
<i>Line</i>	Designates a line number. The range is 1 to 8191.												

ABBREVIATED ARGUMENT TYPES

<i>LRval</i>	Either LConst or Reg.
<i>Lval</i>	Either LConst or Xreg.
<i>RLine</i>	Either a Line or a Reg.
<i>SRval</i>	Either SConst or Reg.
<i>Sval</i>	Either SConst or Xreg.
<i>[Argument]</i>	Denotes optional argument.

EXPRESSION OPERATORS

The expression operator indicates the action to be performed on the two arguments.

The typical form of an expression is:

Argument1 Expression Operator Argument2

Where the Expression Operator can be:

- An arithmetic operator; performs an arithmetic operation.
- An equality and relational operator; performs a test operation.
- A relational circular operator; performs a test operation.
- A binary operator; performs a binary operation.

ARITHMETIC OPERATORS; PERFORMS AN ARITHMETIC OPERATION (32 BIT SIGNED OPERATIONS)

*	Multiply Argument1 with Argument2.
+	Add Argument1 to Argument2.
-	Subtract Argument2 from Argument1.
/	Divide Argument1 by Argument2
<<	Shift arithmetic Argument1 to the left by Argument2 steps
>>	Shift arithmetic Argument1 to the right by Argument2 steps. If a negative number is shifted this way it will always be negative.

EQUALITY AND RELATIONAL OPERATORS; PERFORM A TEST OPERATION

<	TRUE if Argument1 is less than Argument2.
<=	TRUE if Argument1 is less than or equal to Argument2.
<>	TRUE if Argument1 is not equal to Argument2.
=	TRUE if Argument1 is equal to Argument2. NOTE: If no operator is specified for the mnemonics IF and WAIT and Argument1 is not equal to 0, then this is evaluated TRUE. Ex: <code>WAIT 7</code> will always be TRUE
>	TRUE if Argument1 is greater than Argument2.
>=	TRUE if Argument1 is greater than or equal to Argument2.

RELATIONAL CIRCULAR OPERATORS; PERFORM A TEST OPERATION

Circular comparison eliminates the problem of a variable's value wrapping from positive to negative or from negative to positive.

Consider an 8-bit arithmetic with a possible range of -128 to +127. If you add 10 to +127 the result is +137, but because of the limited range the result is -119.

Therefore, if you have register R0 = +127, the comparison: `R0+10 > R0` would evaluate to FALSE.

Circular comparison can be thought of as a "number" circle where the value +127 is placed next to the value -128.

The circular comparison will determine whether clockwise (+) or counterclockwise (-) is the shortest way between +127 and -119.

Clockwise (>) evaluates TRUE, and counterclockwise (<) evaluates TRUE.

The possible circular comparisons are:

Mnemonic Operators

- > TRUE if shortest way from Argument1 to Argument2 is clockwise.
- < TRUE if shortest way from Argument1 to Argument2 is counterclockwise.
- >= TRUE if Argument1 equals Argument2 or if shortest way from Argument1 to Argument2 is clockwise.
- <= TRUE if Argument1 equals Argument2 or if shortest way from Argument1 to Argument2 is counter clockwise.

BINARY OPERATORS, PERFORM A BINARY OPERATION

- |< Shift binary Argument1 to the left by Argument2 steps.
- |> Shift binary Argument1 to the right by Argument2 steps.
- AND Bit wise AND of arguments.
For IF or WAIT mnemonics: TRUE if result is non-zero.
- EXOR Bit-wise EXCLUSIVE OR of ARGUMENTS.
NOTE: EXOR can be used to perform a NOT function using:
Let R10 = Argument1 EXOR -1.
- MOD Take Argument1 modulo Argument2. Remainder of the division Argument1/Argument2.
Example: R0 = 107 mod 10; R0 will be assigned the value 7.
- OR Bit wise OR of arguments.
For IF or WAIT mnemonics, TRUE if result is non-zero.

MNEMONIC OPERATORS

- Op* Operator; each statement has its own supported operators (see following).
- IFcOp* One of:
<= >= < > =
- IfOp* One of:
OR AND <= >= <> = < > THEN
- PosOp* One of:
ABORT ABS INC REL 'MOD ON CLR' 'MOD OFF CLR' 'MOD ON' 'MOD OFF'
- ProfOp* One of:
ACC
- THEN* Either THEN or , (comma)
- WcOp* One of:
<= >= < > =
- WOOp* One of:
OR AND <= >= <> = < > (none)
- LetOp* One of:
EXOR OR AND << >> |< |> MOD /
* - + MULDIV DIVMUL

SCALING MNEMONIC OPERATORS

The operation uses full 64-bit precision in the multiplication and division. In the case of an overflow in the result will be $\pm\text{MAXINT}$. (2147483647)

DIVMUL LET <Arg1> = <Arg2> DIVMUL <Arg3>

$$\text{Arg1} = \frac{65536 * \text{Arg2}}{\text{Arg3}}$$

MULDIV LET <Arg1> = <Arg2> MULDIV <Arg3>

$$\text{Arg1} = \frac{\text{Arg2} * \text{Arg3}}{65536}$$

Internal calculations are made with 64 bit arithmetic. The user must handle overflow.

Example usage of the MULDIV and DIVMUL operators:

```
; Assume a 4 pole motor
; If we want to convert one motor revolution into a
; position of 1000
; calculate the conversion factor
  r100 = 1000 DIVMUL Motor.PPR
  <other lines>
  r200 = rd1.pos MULDIV r100
  disp r200
```

COMPILER EXTENSIONS

[] Angle brackets are meta symbols implying that text is optional.

addr_of @

An address of operator can be used to obtain the line number for the start of a profile or Program. The '@' symbol designates the address of an operator. Example:

```
[Let] R10 = @Profile           ;Ex 1.
Profile Acc @Profile         ;Ex 2
Vector.CycInt = @MyTimProg   ;Ex 3.
```

Ex1. Load R10 with the line number where the profile "Profile" is defined.

Ex2. Start the profile "Profile".

Ex3. Load the vector for cyclic interrupts to point to the interrupt routine "MyTimProg".

cond_expr

A conditional expression used with conditional compilation directives using the form:

```
.if const_expr cond_operator const_expr.
```

const

A constant value of type LConst or Sconst

const_expr

Any expression which result in a constant value. Constant expressions consist of VSymbols, Operator[s] and const. Parentheses can be used to alter the normal operator precedence and associativity rules.

```
.define FOO = 10 / 2 + 1 ; Evaluates to 6
.define BAR = 10 / (2 + 1) ; Evaluates to 3
```

dyn_expr

A dynamic expression which is evaluated during run time. The DMC² has the capability of embedding dynamic expressions in its instruction set. Dynamic expression consists of const_expr,

Compiler Symbols

Reg, XReg, or instruction-dependent operators. Use parentheses to force the compiler to parse the expression if the expression is a const_expr. Format:

```
[Let] Reg = Reg let_operator SConst
[Let] XReg = XReg let_operator SConst
```

Example:

```
[Let] MyRegister = MyRegister * MyValue + 3
MyValue + 3 is a const_expr and is evaluated during
compilation.
```

indirect_reg Designates an internal 32-bit register and is accessed as R(Rnnn).

COMPILER SYMBOLS

A symbol can contain the letters “A” to “Z” and “a” to “z”, the underscore ‘_’ character, and digits “0” to “9”. With the following restrictions:

- The first character must be a letter or an underscore.
- By default, the compiler recognizes only the first 64 characters as significant.

VSymbol Alias for a value. A VSymbol can be used as an alias for const or const_expr. VSymbols are constant values and are defined using .DEFINE directives.

Example:

```
.DEFINE ENDPOS = 1000
.DEFINE STAPOS = ENDPOS + 2000
```

RSymbol Alias for a register. RSymbols that can be used as an alias for Reg. RSymbols are constant values and are defined using .DEFINE directives.

Example:

```
.DEFINE REG12 = R12
.DEFINE MAXSPD = REG12
```

XSymbol Alias for an extended register. An XSymbol can be used as an alias for XReg. XSymbols are constant values and are defined using .DEFINE directives.

Example:

```
.DEFINE MPOLES = Motor.Poles
.DEFINE MPOL = MPOLES
```

ISymbol Alias for an indirect register address. An ISymbol can be used as an alias for an indirect_reg. ISymbols are constant values and are defined using .DEFINE directives.

Example:

```
.DEFINE IDXREG = R(R40)
.DEFINE IDX = IDXREG
.DEFINE IDXFOO = R(REG12)
```

ISymbol Alias for an indirect register address. An ISymbol can be used as an alias for an indirect_reg. ISymbols are constant values and are defined using .DEFINE directives.

Example:

```
.DEFINE IDXREG = R(R40)
.DEFINE IDX = IDXREG
.DEFINE IDXFOO = R(REG12)
```

<i>LSymbol</i>	Alias for line number. An LSymbol can be used as an alias for a line_number. Lsymbols are used when a line_number is referenced, such as: <code>GOTO Start</code> <code>Vector.CycInt, @Cyclic</code> <code>Vector.PosErr, @Error</code>
<i>line_number</i>	Line numbers can be specified in two forms: Absolute and Relative. The typical form is: <i>label [:] program statement</i> A label becomes an LSymbol if it starts with a character and ends with a colon symbol; this is the relative line_number form. Example: <code>Start: Pg.Speed = 10000</code> The label can be a number, in which case the line number is absolute. <code>10 PG.Speed = 10000</code> The last definition has the side effect to define the location counter to 10, just as if it was preceded by the line: <code>.ORG 10</code> The reason for this is that the PL compiler also should be able to compile programs uploaded from a motion controller.
<i>space_char</i>	The space character has significance in the DMC compiler. Example: <code>L E T R 1 0 = 5 0 LETR10=50</code> The above statements are not equivalent, it must be typed as, <code>LETR10=50</code> Or <code>LETR10 = 50</code>

PREDEFINED COMPILER SYMBOLS

<code>_DMC_</code>	Is defined when the DMC compiler is used.
<code>_V0500_</code>	Is defined if version 5 of the compiler is used

SPECIAL COMPILER FUNCTIONS

The PL2 Language compiler has the following special mathematical functions:

Sin (a ,b ,c)

Cos (a ,b ,c)

The Sine and Cosine can be used to calculate constant values for cam tables or similar applications.

The functions are evaluated as:

Sin (angle, period, amplitude) = amplitude * Sine(angle/period), angle/period is in radians

Cos (angle ,period, amplitude) = amplitude * Cosine(angle/period)

Compiler directives

Ex: Define a PDATA table holding Sine values for 0 to 9 degrees normalized to an amplitude of 65000

```
.define c_Amplitude = 65000
.define c_period = 180*1000*996/3129      ; 180*1000/pi

pdata sin( 0 * 1000, c_period, c_Amplitude),0 ;0 degrees
pdata sin( 1 * 1000, c_period, c_Amplitude),1 ;1 degrees
pdata sin( 2 * 1000, c_period, c_Amplitude),2 ;2 degrees
pdata sin( 3 * 1000, c_period, c_Amplitude),3 ;3 degrees
pdata sin( 4 * 1000, c_period, c_Amplitude),4 ;4 degrees
pdata sin( 5 * 1000, c_period, c_Amplitude),5 ;5 degrees
pdata sin( 6 * 1000, c_period, c_Amplitude),6 ;6 degrees
pdata sin( 7 * 1000, c_period, c_Amplitude),7 ;7 degrees
pdata sin( 8 * 1000, c_period, c_Amplitude),8 ;8 degrees
pdata sin( 9 * 1000, c_period, c_Amplitude),9 ;9 degrees
```

This will compile to:

```
1 PData 0 , 0
2 PData 1134 , 1
3 PData 2268 , 2
4 PData 3402 , 3
5 PData 4534 , 4
6 PData 5665 , 5
7 PData 6794 , 6
8 PData 7921 , 7
9 PData 9046 , 8
10 PData 10168 , 9
```

This can be very useful for creating motion profiles or look-up tables.

The RDPdata mechanism (see [Extended register groups](#) on page 53) can be used to read Pdata tables for any purpose.

COMPILER DIRECTIVES

The PL2 Language compiler has the following directives:

.LIST	Format:	.LIST ON OFF	[; Comment]
	Function:	Disable/enable the generation of list file output.	
	Default:	On	
.LISTMACRO	Format:	.LISTMACRO ON OFF	[; Comment]
	Function:	Disable/enable the generation of macro expansions in the list file output	
	Default:	On	
.NOTE	Format:	.NOTE "string", Vsymbol	[; Comment]
	Function:	Print "string" and the value of symbol to the list file output.	
.ERROR	Format:	.ERROR "string"	[; Comment]

	Function:	The error string is printed to the list file output.
.ORG	Format:	<code>.ORG const [; Comment]</code>
	Function:	Sets the location counter to the value of the const so that next output record is generated at line const.
	Warnings:	If code is overwritten because of the .org directive, a warning is generated.
	Error:	If const results in a value that is not within the memory range for this controller configuration, an error is generated.
.LINESTEP	Format:	<code>.LINESTEP const [; Comment]</code>
	Function:	Sets the incremented location counter values between each generated line; the default is 1. Its purpose is to simplify debugging by allowing space between the lines. For maximum execution speed, use <code>'LINESTEP 1'</code> .
	Warnings:	None
	Error:	If, during code generation, all memory for this controller configuration is used, an error is generated.
.DEFINE	Format:	<code>.DEFINE Vsymbol = const [; Comment]</code> <code>.DEFINE Vsymbol = const_expr [; Comment]</code> <code>.DEFINE Rsymbol = reg [; Comment]</code> <code>.DEFINE Xsymbol = xreg [; Comment]</code> <code>.DEFINE Isymbol = indirect_reg [; Comment]</code>
	Function:	Defines a symbol as an alias for a constant expression, constant, register, extended register or indirect register. A constant expression is first evaluated to a const. The symbol has the same value and type as the right-hand side.
	Warnings:	None
	Error:	If the right-hand side cannot be evaluated, an error is generated.
.UNDEFINE	Format:	<code>.UNDEFINE XSYMBOL</code>
	Function:	Undefine a symbol in the compiler so that it can be redefined.
.BYTE	Format:	<code>.BYTE const1, const2., const5 [; Comment]</code>
	Function:	Primarily intended to allow an older version of the compiler to generate code for unimplemented program statements. <code>.MACRO NEWINSTR a, b, c</code> <code>.BYTE 123, 34, a, b, c</code> <code>.ENDMACRO</code>

Compiler directives

	Warnings:	None
	Error:	None
.PAGE	Format:	.PAGE Const [; Comment] .PAGE [; Comment]
	Function:	If Const is > 0, the page length is set to Const. If Const is zero or omitted, a new page is ejected. If Const is negative, a new page is ejected when less than Const lines remain on the current page.
	Warnings:	None
	Error:	None
.INCLUDE	Format:	.INCLUDE "filename"
	Function:	This directive retrieves a named file, known as an included file or header file, into the source code. The file name specified is included for compilation in the source file. The number of included files that can be opened or the nesting depth of include files, is limited only by memory or file handle limitations. Include file recursion is not allowed and generates an error message when encountered.
	Warnings:	None
	Error:	If the file could not be found, or an attempt is made to do include file recursion, an error is generated.
.REGISTER	Format:	.REGISTER RSym1<, Rsym2><, [; Comment] Rsymn
	Function:	The symbols are automatically assigned to a free register number.
	Warnings:	None
	Error:	If there are no free registers, an error message is generated.
.EPROM	Format:	.EPROM [; Comment]
	Function:	Informs the compiler to generate code suitable for direct transfer to an EPROM programmer.(DMC1 only)
	Warnings:	Code generated with this switch can not be downloaded to the DMC ² drive, and code generated without this switch can not be downloaded to an EPROM programmer.
	Error:	None
.NOFILL	Format:	.NOFILL

	Function:	Normally the compiler generates a full image for the targets program memory with empty lines where lines are not used, up to the highest line number. This directive informs the compiler to generate only the used lines. This can be used for partial download of PL2 programs to a DMC, for instance CAM-tables.(See COMM.MODE for information on that)
.MaxLines	Format:	.MaxLines const
	Function:	Allows the compiler to generate up to the given amount of lines (DMC2 supports 8191 lines)
.MaxRegisters	Format:	.MaxRegisters const
	Function:	Allows the compiler to utilize more than 256 registers(older versions of DMC) max number is 4096 for DMC2
.ConvertDispToNop	Format:	.ConvertDispToNop
	Function:	With this directive the compiler will replace all occurrences of the DISP statement with a NOP instruction. Since DISP statements are very dangerous to have in time critical parts of a PL2 program it is highly recommended to take them away in a final version.

Spline function compilation directives

SPLINE FUNCTION COMPILATION DIRECTIVES

The compiler can automatically generate a PData array using linear spline interpolation from a few waypoints.

.SplineDef	Format:	.SplineDef <Name>,<number of slots in between>
	Function:	Defines the name for the profile and the resolution in the table. If 2 slots is defined it means that 1 extra point is inserted.
.SplinePoint	Format:	.SplinePoint <waypoint>,[timescale]
	Function:	Defines a waypoint in the profile.
.SplineDefEndt	Format:	.SplineDefEndt
	Function:	Terminates the definition of the profile.

The compiler function length (<Name>) calculates the number of Pdata lines generated.

EX:

```
.Splinedef Pro_cam1,5
.SplinePoint 0
.SplinePoint 200
.SplinePoint 600
.SplinePoint 300
.SplinePoint 27
.SplineDefEndt
R100 = length(Pro_cam1)
```

This will compile to:

```
PData 0 , 0
PData 20 , 0
PData 45 , 0
PData 80 , 0
PData 130 , 0
PData 200 , 0
PData 291 , 0
PData 392 , 0
PData 488 , 0
PData 562 , 0
PData 600 , 0
PData 590 , 0
PData 542 , 0
PData 469 , 0
PData 384 , 0
PData 300 , 0
PData 227 , 0
PData 167 , 0
PData 115 , 0
PData 69 , 0
PData 26 , 0
Let R100 = 21
```

CONDITIONAL COMPILATION DIRECTIVES

Replacing the appropriate source code line with a blank line is supported by conditional compilation.

Lines between an enclosed pair of `.ifdef` and `.endif` directives will be include if the condition is true, else excluded.

All conditional compilation directives must be completed in the source or include file in which they originate.

<code>.IFDEF</code>	Format:	<code>.IFDEF Vsymbol</code>	<code>[: Comment]</code>
<code>.IFNDEF</code>		<code>.IFDEF Rsymbol</code>	<code>[: Comment]</code>
		<code>.IFDEF Xsymbol</code>	<code>[: Comment]</code>
		<code>.IFDEF Isymbol</code>	<code>[: Comment]</code>
	Function:	If the symbol is defined, the conditional directive results in a TRUE, and the next lines are going to be compiled until an <code>.if</code> , <code>.ifdef</code> , <code>.ifndef</code> , and <code>.endif</code> , directive is encountered.	
		If symbol is not defined this directive results in a FALSE and the next lines are going to be replaced with blanks until an <code>.if</code> , <code>.ifdef</code> , <code>.ifndef</code> , and <code>.endif</code> , directive is encountered.	
		Each <code>.IFDEF</code> directive must be carefully balanced with a closing <code>.endif</code> directive.	
	Warnings:	<code>.IFDEF symbol [code]</code>	<code>[: Comment]</code>
		<code>.ENDIF</code>	
	Error:	<code>.ifdef</code> without closing <code>endif</code> will generate an error.	
<code>.ENDIF</code>	Format:	<code>.ENDIF</code>	<code>[: Comment]</code>
	Function:	Closes a conditional directive.	
	Error:	None	

MULTILINE MACRO

Macros provide a mechanism for token replacement, with or without a set of formal, function-like parameters. Each occurrence of the “name” in the source code is replaced by the “macro body”.

A macro is a “new instruction” and it must be defined before it is used. A macro can have any number of arguments, but is limited by a maximum line length of 255. The definition instructs the compiler what type of argument is allowed and what type of function to perform. Each argument type must be individually specified. An argument type is any of the compiler-supported types.

Multiline Macro

MACRO DEFINITION

A macro definition needs information regarding the type of the argument, with an arg-specifier placed after each argument.

Format:

```
.MACRO
.MACRO      foo [,sym:arg-specifier] [,nn:as]           [; Comment]
            [lab:] [macro body]
.ENDMACRO                                         [; Comment]
```

The macro body specifies what operation the “new instruction” will perform when invoked (macro expansion). Standard native instructions or other macros can be used and the nesting depth of a macro calling another macro is limited only by the amount of available memory. Macro recursion is not allowed and generates an error message.

Names of labels within the macro body can be used freely. Each macro maintains its own symbol table. This means that symbols defined in the program scope are not accessible within a macro definition.

Example:

```
.DEFINE foo = 4711           ; The symbol has program scope
.MACRO bar, foobar:v
LET r0, foo                 ; This will generate an error, foo is undefined
.ENDMACRO
```

In the list file, any label used within a macro definition ends with the (\$) symbol instead of the standard colon (:) symbol.

NOTE: It is possible to use the absolute label format in a macro definition, but there can only be one within the macro. For a second occurrence within the macro, the compiler generates a “previous line overwritten” error message.

MACRO ARG SPECIFIERS

Arg specifiers are divided into 8 categories

No	Symbol	Description	NOTE
1	r	General register type, Reg, XReg, or IRreg.	
2	v	General value type, i32 or i16.	
3	lrf	General line reference.	
4	i32	Long value.	
5	i16	Short value.	Not impl.
6	reg	Ordinarily register.	
7	ireg	Index register.	
8	xreg	Extended register.	

Valid arg-specifiers for DMC² compilers are all those that are implemented.

MACRO CALL (EXPANSION)

Format: [lab:] foo<,sym><,nn> [; Comment]

The macro must be defined before using. The “new instruction” must be given the same number and type of arguments as defined for the macro.

PL2 Mnemonics

GENERAL

The general form for a *Mnemonic* is:

Mnemonic [Mnemonic Operator]

Mnemonic operators are used with a *Mnemonic* to fully define an *Instruction*.

STANDARD SET MNEMONICS

Mnemonic	Mnemonic Operator and Argument[s]	Description
<code>Nop</code>		Has no function.
<code>Stop</code>		Stops execution. Execution may later resume at the following line using the CONT command.
<code>End</code>		Ends execution. Informative message is sent to the terminal.
<code>End</code>	<code>SILENT</code>	Ends execution. No message is sent.
<code>Disp</code>	<code>Lval</code>	Displays <i>XReg</i> , <i>Reg</i> or <i>Const</i> on the terminal.
<code>Goto</code>	<code>Line</code>	Execution proceeds at the indicated line.
<code>Gosub</code>	<code>Line</code>	Calls a subroutine at indicated line.
<code>IdxGoto</code>	<code>Rval,Line</code>	Execution proceeds at the indicated line number stated in the Register + the Line number
<code>IdxGosub</code>	<code>Rvat,Line</code>	Calls a subroutine at the indicated line number stated in the Register + the Line number
<code>IdxGoto</code>	<code>Rval</code>	Execution proceeds at the indicated line number stated in the Register
<code>IdxGosub</code>		Calls a subroutine at the indicated line number stated in the Register
<code>Return</code>		Return from a subroutine to the line immediately following the GOSUB line.
<code>IReturn</code>	<code>SYS Sval</code>	Returns from a system interrupt routine and enable the specified interrupts. Binary OR <i>Sval</i> to <code>Int.SysMask</code> .
<code>IReturn</code>	<code>IN Sval</code>	Returns from an I/O interrupt routine and enable the specified interrupts. Binary OR <i>Sval</i> to <code>Int.Mask</code> .
<code>IReturn</code>	<code>CAS_n Sval</code>	Returns from a cascaded interrupt routine and enable the specified cascaded interrupts. Binary OR <i>Sval</i> to the mask register of the cascade handler, the <code>Int.SysMask</code> is automatically re-enabled. Where <u>n</u> is the cascade handler number.
<code>Loop</code>	<code>Reg, Line</code>	Decrement <i>Reg</i> by one. If <i>Reg</i> is > 0, then goto the indicated <i>Line</i> ; else proceed with the following line.

Standard set Mnemonics

WaitC	<i>Sval WcOp Lval</i>	Wait for the expression to become TRUE, but use “circular” comparison instead of the standard one.
Wait	<i>Sval WOp Lval</i>	Wait for the expression to become TRUE or non-zero.
IfC	<i>Sval IfcOp Sval THEN Line</i>	If “circular” expression is TRUE, then goto Line.
IfCNot	<i>Sval IfcOp Sval THEN Line</i>	If “circular” expression is FALSE, then goto Line.
If	<i>Sval IfOp Sval THEN Line</i>	If expression is TRUE or non-zero, then goto Line.
IfNot	<i>Sval IfOp Sval THEN Line</i>	If expression is FALSE or non-zero, then goto Line.
IfAbs	<i>Sval IfOp Sval THEN Line</i>	Convert both arguments to absolute values before evaluating the expression. If expression is TRUE or non-zero, then goto Line.
IfAbsNot	<i>Sval IfOp Sval THEN Line</i>	Convert both arguments to absolute values before evaluating the expression. If expression is FALSE or non-zero, then goto Line
Let	<i>XReg = Sval [LetOp Sval]</i>	Calculate the value of the expression and assign it to XReg.
Let	<i>XReg , Lval</i>	Assign the value Lval to XReg. NOTE: Use this for Lconsts, larger than 16-bit, 32767.
Add	<i>XReg , Lval</i>	Add value Lval to Xreg.
Sub	<i>XReg , Lval</i>	Subtract value Lval from Xreg.
Abs	<i>Xreg , Lval</i>	If Lval is positive, then Xreg := +ABS(Xreg). If Lval is negative, then Xreg := -ABS(Xreg).
ISqr	<i>XReg , Lval</i>	Calculate the integer square root of Lval and assign it to Xreg. Xreg := ISQR(Lval)
Clr	<i>Xreg</i>	Zero a register.
BClr	<i>Xreg, Lval</i>	Bit-Clear of register; the same as AND with complemented argument.
And	<i>Xreg, Lval</i>	Binary AND register with value.
Or	<i>Xreg, Lval</i>	Binary OR registers with value. NOTE: None of the instructions DISABLE interrupts; therefore, if one does BCLR Int.Pend, 4 to remove an interrupt, it is possible to miss an interrupt that is arriving just as the instruction executes.
Push	<i>Lval</i>	Push Lval on the parameter stack.
Pop	<i>Xreg</i>	Pop value from the parameter stack and assign it to Xreg.
Pos	ABORT	Terminate the current position or profile

		statement. Equivalent to: <code>[LET] Pg.Mode = 0</code>
Pos	ABS <i>Lval</i>	Position absolute to Lval. Equivalent to: <code>[LET] Pg.Mode = 0</code> <code>[LET] Pg.DPos = Lval</code> <code>IF Pg.PosSpeed = 0 THEN LBL1</code> <code>[LET] Pg.Speed = Pg.PosSpeed</code> <code>LBL1: [LET] Pg.Mode = 1</code>
Pos	INC <i>Lval</i>	Incremental positioning Lval from the last positioning statement. Equivalent to: <code>[LET] Pg.Mode = 0</code> <code>ADD Pg.DPos, Lval</code> <code>IF Pg.POS Speed = 0 THEN LBL1</code> <code>[LET] Pg.Speed = Pg.PosSpeed</code> <code>LBL1: [LET] Pg.Mode = 1</code>
Pos	REL <i>Lval</i>	Position Lval relative to the current position, of the profile, not the motor. Equivalent to: <code>[LET] Pg.Mode = 0</code> <code>[LET] Pg.DPos = Pg.APos + Lval</code> <code>IF Pg.PosSpeed = 0 THEN LBL1</code> <code>[LET] Pg.Speed = Pg.PosSpeed</code> <code>LBL1: [LET] Pg.Mode = 1</code>
Pos	MOD ON [CLR]	Allow the set-position to be modified by the cam/gearbox routines.
Pos	MOD OFF[CLR]	Disallow the set-position to be modified by the cam / gearbox routines.If CLR is specified, the internal position, Pg.APos, is changed so there is no momentary change in position. If CLR is not specified, the motor slews toward the new set-position using a speed that is determined by the regulator settings.
PData	<i>LRval , SRval</i>	Specifies data for the PROFILE statement. The LRval argument is the desired acceleration, and SRval is the number of servo cycles this acceleration will use. A value of Zero indicates the end of the profile. The variable Pg.ProScale can be used to scale the acceleration. A value of 1024 means no scaling is being done, and the LRval corresponds to increments/sec ² .
Profile	ACC <i>RLine</i>	Argument RLine specifies the line number of the first PDATA to use. See PDATA.
RefPos	POS <i>Lval</i>	Sets the reference position to Lval. This sets the current position reference relative to RD1.Pos so that the positioning statements, POS ABS etc., refer to this reference position. The RD1.RPos returns the "referenced" position. Pg.APos, Pg.DPos are also affected. The REF POS statement is implemented with the aid of a position offset from the resolver position. The offset is Pg.PosOffs. To remove the effect of

Standard set Mnemonics

<p>Connect</p> <pre>Ana.IN1 TO Sval Ana.IN2 TO Sval Ana.OUT1 TO Sval Ana.OUT2 TO Sval Dstore.In1 TO Sval Dstore.In2 TO Sva Dstore.Peek1 TO Sva Dstore.Peek2 TO Sva</pre>	<p>the statement REF POS, set Pg.PosOffs = 0.</p> <p>The connection is enabled by connecting the analog I/O to a Reg or XReg, it is disabled by connecting it to a Const.</p> <p>The update rate is set in the Ana.ConnTMR register. The maximum number of active connections at one time is four. However, running four connections at 1 ms update rate will take a considerable amount of CPU-time from the execution of the program.</p> <p>Example:</p> <pre>Ana.ConnTMR = 1 ; Want 1 ms update rate. Connect Ana.In1 TO 0 ; Turn off the connection to Ana.In1 Connect Ana.In2 TO ; Set speed from Ana.In2 Pg.Speed ; POS error to Ana.Out1 Connect Ana.Out1 TO Reg.PosErr</pre>
<p>EEStore</p>	<p>Store the contents of the EEPROM extended register group in nonvolatile memory.</p> <p>NOTE: Because this is a time consuming process, verify the operation has completed before continuing with the next instruction.</p> <pre>10 EESTore 20 wait SysIo.MemStat and 1 30</pre>
<p>EELoad</p>	<p>Load the information from the nonvolatile memory into the EEPROM extended register group.</p> <p>NOTE: Because this is a time consuming process, verify the operation has completed before continuing with the next instruction.</p> <pre>10 EELoad 20 wait SysIo.MemStat and 1 30</pre>
<p>Peek</p>	<p>Debug use only</p>
<p>Poke</p>	<p>Debug use only</p>
<p>RESET SYSTEM</p>	<p>Reset CPU-board. Recommended way to warm start system</p>
<p>FHbit</p>	<p><i>Xreg, Xreg, Sva</i></p> <p>Find highest bit in second argument. Reports bit number in first argument.</p> <p>Ex: R100 = 255</p> <p>Fhbit R10, R100, 32 will return R10 = 7 since bit7 is set in R100, The Sval is a limiter of how many bits to search for. If there are bits higher than the search limiter it will return the limiter value.</p>
<p>RegEncode</p>	<p><i>Xreg, SRval, SRval</i></p> <p>Calculates the internal address to a Xreg similar to the computer mode specification</p> <p>Ex: RegEncode R100,4,6 will return R100 =33798</p>

Used in combination with SET and GET instructions to simplify indexing into extended registers.

Regencode R100,4,5 would give the address to the register
Pg.Posspeed.(32768+256*group+member)

TRACE RELATED MNEMONICS

The Trace function allows the user to trace the execution of the PL program. The trace will save a time stamp and the current line number where code is executing. There is room for 512 entries in the buffer. If the PL program is modified after/during the trace, it will make the trace invalid. To conserve space in the buffer, the tracing can be limited to interesting parts, by use of the Trace OFF/Trace CONT command or a snapshot that fills the buffer, Trace ONCE. To inspect the trace buffer content, use the TLIST command. When the Trace is active it slows down the PL2 rate to about 70 % of the normal. That means if Trace is used for measuring execution speed that has to be compensated for. It also states that leaving trace active during normal operation is a waste of resources!!! To debug an error situation the Trace can be used like this:

Turn on the Trace function with TRACE ON at the beginning of the program. Put TRACE OFF at the end of the error handling routine. When the error occurs there will be a log of the last 512 lines of PL2 code that lead up to the error.

Trace	ON	Initializes the trace buffer and starts trace.
Trace	OFF	Stops the trace.
Trace	CONT	Continue trace without initializing the buffer.
Trace	ONCE	Trace until buffer is full.(Single shot trace)

LAN1/ LAN1 RELATED MNEMONICS

SetObj Lan1	Sval	Map the content in MsgObjLan1 to the priority level specified in Sval. For a detailed description see the LAN1 group.
GetObj Lan1	Sval	Fill in the MsgObjLan1 with the message object at priority level Sval. For a detailed description see the LAN1 group.
Read Lan1	Reg, len, level	Read len bytes and put in register Reg from the buffer for message object at level. Where len and level are Sval. For a detailed description see the LAN1 group. If len is specified as negative the data will be byte swapped.
Write Lan1	Reg, len, level	Write len bytes to the buffer for message object at level from register Reg. For a detailed description see the LAN1 group. If len is specified as negative the data will be byte swapped.

Anybus-S Related Mnemonics

`SendObj Lan1 Sval` Send the content of the buffer for the message object at Sval on to the CAN bus.
For a detailed description see the LAN1 group.

ANYBUS-S RELATED MNEMONICS

<code>AnybusIn</code>		Modifies the input buffer to the Anybus-S module. Data in the input buffer is to be transmitted onto the Fieldbus.
<code>AnybusIn putDWORD</code>	<code><reg>, <offs></code>	Write 32 bits, unsigned, from <code><reg></code> to the buffer at offset <code><offs></code> (byte offset)
<code>AnybusIn putWORD</code>	<code><reg>, <offs></code>	Write 16 bits, unsigned, from <code><reg></code> to the buffer at offset <code><offs></code> (byte offset)
<code>AnybusIn putBYTE</code>	<code><reg>, <offs></code>	Write 8 bits, unsigned, from <code><reg></code> to the buffer at offset <code><offs></code> (byte offset)
<code>AnybusIn putLONG</code>	<code><reg>, <offs></code>	Write 32 bits, signed, from <code><reg></code> to the buffer at offset <code><offs></code> (byte offset)
<code>AnybusIn putINT</code>	<code><reg>, <offs></code>	Write 16 bits, signed, from <code><reg></code> to the buffer at offset <code><offs></code> (byte offset)
<code>AnybusIn putSCHAR</code>	<code><reg>, <offs></code>	Write 8 bits, signed, from <code><reg></code> to the buffer at offset <code><offs></code> (byte offset)
<code>AnybusIn getDWORD</code>	<code><reg>, <offs></code>	Reads 32 bits, unsigned, from the buffer into <code><reg></code> at offset <code><offs></code> (byte offset)
<code>AnybusIn getWORD</code>	<code><reg>, <offs></code>	Reads 16 bits, unsigned, from the buffer into <code><reg></code> at offset <code><offs></code> (byte offset)
<code>AnybusIn getBYTE</code>	<code><reg>, <offs></code>	Reads 8 bits, unsigned, from the buffer into <code><reg></code> at offset <code><offs></code> (byte offset)
<code>AnybusIn getLONG</code>	<code><reg>, <offs></code>	Reads 32 bits, signed, from the buffer into <code><reg></code> at offset <code><offs></code> (byte offset)
<code>AnybusIn getINT</code>	<code><reg>, <offs></code>	Reads 16 bits, signed, from the buffer into <code><reg></code> at offset <code><offs></code> (byte offset)
<code>AnybusIn getSCHAR</code>	<code><reg>, <offs></code>	Reads 8 bits, signed, from the buffer into <code><reg></code> at offset <code><offs></code> (byte offset)
<code>AnybusOut</code>		Modifies the output buffer from

			the Anybus-S module. Data in the output buffer has been recieved from the Fieldbus
AnybusOut	putDWORD	<reg>, <offs>	Write 32 bits, unsigned, from <reg> to the buffer at offset <offs>(byte offset)
AnybusOut	putWORD	<reg>, <offs>	Write 16 bits, unsigned, from <reg> to the buffer at offset <offs>(byte offset)
AnybusOut	putBYTE	<reg>, <offs>	Write 8 bits, unsigned, from <reg> to the buffer at offset <offs>(byte offset)
AnybusOut	putLONG	<reg>, <offs>	Write 32 bits, signed, from <reg> to the buffer at offset <offs>(byte offset)
AnybusOut	putINT	<reg>, <offs>	Write 16 bits, signed, from <reg> to the buffer at offset <offs>(byte offset)
AnybusOut	putSCHAR	<reg>, <offs>	Write 8 bits, signed, from <reg> to the buffer at offset <offs>(byte offset)
AnybusOut	getDWORD	<reg>, <offs>	Reads 32 bits, unsigned, from the buffer into <reg>at offset <offs>(byte offset)
AnybusOut	getWORD	<reg>, <offs>	Reads 16 bits, unsigned, from the buffer into <reg>at offset <offs>(byte offset)
AnybusOut	getBYTE	<reg>, <offs>	Reads 8 bits, unsigned, from the buffer into <reg>at offset <offs>(byte offset)
AnybusOut	getLONG	<reg>, <offs>	Reads 32 bits,signed, from the buffer into <reg>at offset <offs>(byte offset)
AnybusOut	getINT	<reg>, <offs>	Reads 16 bits, signed, from the buffer into <reg>at offset <offs>(byte offset)
AnybusOut	getSCHAR	<reg>, <offs>	Reads 8 bits, signed, from the buffer into <reg>at offset <offs>(byte offset)
AnybusIO		Send	Transfers the content of the SanyBus INPUT buffer to the SanyBus module and issues a field bus send request.

INDEXED ADDRESSING MNEMONICS

Set *Xreg# IndexValue=Sval*

Set *Xreg #++ IndexValue=Xreg*

Indexed Addressing Mnemonics

Set *Xreg #-- IndexValue=Xre*

Get *Xreg = Xreg # IndexValue*

Get *Xreg = Xreg #++ IndexValue*

Get *Xreg = Xreg #- IndexValue*

The IndexValue is of type Sval or Xreg is used to index into the Xreg group. The index value is used as an offset to the member number given in Xreg. The operator # means the index-value is just used, #++ means the index-value is incremented after use and #- means the index-value is decrements after use. There is no check that the resulting member number does exist, Read and or Writes to non-existent members will be ignored with one exception. If the member 255 is read, and non-existent, the value returned will be 80 000 000h + the max allowed address of groups for the system.

The function of the Get and Set can be modified by setting `Sysio.Compatible bit0`.

If this bit is set the register holds the address to the Xreg calculated with the RegEncode instruction.

Ex. To get the value of Pg.Posspeed int R100 with the GET instruction can now be done in two ways:

Pg.Posspeed is denoted: 4,5 as in group, member.

With <code>sysio.compatible = 0</code>	normal function of Get
Get R100 = Pg.mode#5	read value of Pg.Posspeed

With <code>sysio.compatible = 1</code>	sets new function of Get
Regencode R200,4,5	calculates the address of Pg.Posspeed
Get R100 = R200#0	read value of Pg.Posspeed

This allows for designing protocol mechanisms in PL2 in relation to fieldbus usage, similar to the embedded computer mode protocol.

Consider a system where a master sends parameters to a slave on using the LAN1 network.

Interrupt service routine:

```

Read lan1 r_Group,1,1           ;read group pointer
Read lan1 r_Member,1,-1        ;read member pointer
Read lan1 r_Data,4,-1          ;read data
RegEncode r_Pointer,r_Group,0   ;decode address to group
Set r_Pointer# r_Member = r_Data ;write data to target

```

It might be necessary to check if the target really exists by reading the length of the group on index 255.

```

RegEncode r_Pointer,r_Group,0   ;decode address to group
Get r_length = r_Pointer #255   ;read length
And r_length,255                ;mask out length
If r_length = 0 then NoTarget   ;nonexistent group

```

```
If r_length < r_Member then NoTarget;nonexistent member
Set r_Pointer# r_Member = r_Data      ;write data to target
Notarget: return
```

Examples:

1	Copy EEprom.10 .. Eeprom.20 to EEprom.30 .. Eeprom.40	
100	R0 = 11	; 11 registers to copy.
110	Get R1 = EEprom.9 # R0	; Get EEprom.(9+R0)
111	Set EEprom.29 # R0 = R1	; and write to EEprom.(29+R0)
112	Loop R0, 110	; Decrement R0 and loop until Zero.

2	Push all the Motor. variables on the stack.	
100	Get R0 = Motor.0 # 255	; This gives the number of entries
110	AND R0, 255	; Remove extra info.
111	Get R1 = Motor.0 #-- R0	; Get motor.R0 and do R0 = R0-1
112	PUSH R1	
113	IF R0 > 0 THEN 111	; Loop until all done.

TEXT MODE

The text handling in for the PL environment is implemented as follows:

1. You need a format descriptor string, similar to the print in the C- language.
2. You need an instruction to specify the data that should be displayed.
3. You need a way to control the standard line editor, so it won't interfere with your printout if you are using cursor addressing or multiple print statements.

The following is now implemented:

`Image` takes string argument.

`IPrint` takes three <Sval> arguments.

`TRead` takes some modifiers and optionally one <Xreg> argument.

TEXT OUTPUT

`Image` is used to specify the format string for the output.

`IPrint` is used to specify what string, and then it can send up to two arguments to be printed according to format in the `Image`.

```
10 Image "This is a string"
20 IPrint
```

This will insert a CR/LF sequence both before and after "Hello!", numbers are always three digit decimal, but you may enter less than that if the number is terminated by a non-numerical character.

```
10 IMAGE "The result is:\013\010units ok: %d:9\13\10units
failed: %d:5\13\10"
```

Text mode

```

20   R1=102
30   R2=17
40   IPrint 10, R1, R2

```

The output result is:

```

Units OK      102
Units failed  17

```

Note that the numbers are aligned, this is accomplished by specifying the field-width of the display with the `:<digit>` modifier.

If you list the program, you will also notice that the IMAGE will take many PL lines. Since only seven characters of an image will fit into a single PL-code line, the logical IMAGE statement can span many pl-code lines. (In the same way a motion profile with P DATA statements spans multiple PL lines.) But for convenience, the firmware one-line compiler allows entry of a longer image.

You should also note the `”; End”` comment after the last image statement in the group. It shows where the logical image actually ends.

A logical image is terminated by alternately a NUL (`\000`) character, i.e. the physical image statement has less than seven characters in it. If the last physical IMAGE statement has seven characters, but the next statement is NOT an IMAGE, the logical IMAGE is also terminated.

You need to look out for this, so you don't continue an IMAGE by mistake.

```

10   Image "This is"
11   Image "an ima"
12   Image "ge.\013\010"; End
20   IPrint 10           Will produce: "This is an image"
22   IPrint 11          Will produce: "an image"
20   IPrint 10

```

The length when entering an image is limited by the input buffer size, in practice about 100 to 120 characters, depending on how many escape sequences that are entered.

The length of a logical IMAGE is only limited to the amount of PL code space that is available. Also note that in IPRINT you can use a register to specify what line the image resides on.

A cursor addressing example, this will only work if you have an ANSI or VT100 compatible terminal connected, you must also have turned off the monitor (code `\002`, see below) to get the desired effect.

```

10   Image "\027[%D%;%DH"
20   IPrint 10,R1,40

```

For a VT100/ANSI compatible terminal, this will send a cursor addressing sequence to row in R1 and column 40. Note the `"%"` sign after the first `"D"` in the image, it is to delimit, the `”;` so it is not interpreted as a format modifier for the `"D"` format.

The following escape and control sequences are of interest in an image:

```

\002   Start of TEXT, this character is used to indicate start of text mode
        printouts, it will disable the normal line-editor control-T and DISP
        statements from sending characters, to allow the PL program to have
        full control over what is sent.

\003   End of TEXT, allow standard line editor etc. to send characters.

```

`\000` Internally used to signal end of image, this code can therefore not be presenting the image.

These codes are required if you want to use the TREAD statement, or have printout using more than one IPRINT statement and they are not sent to the terminal. If you would like to send these codes, use the %C or %S format as described below.

- `\\` Insert one backslash into the string.
- `\"` Insert one quote (") into the string.
- `%%` Print one percent sign.
- `%B` Print argument as an 8-bit byte in hexadecimal.
- `%W` Print argument as a 16-bit word in hexadecimal.
- `%L` Print argument as a 32-bit long-word in hexadecimal.
- `%C` Print argument as a character.
- `%S` Print argument as a NUL terminated string, i.e. 0 to 4 chars, LSB being printed first.
- `%D` Print argument signed decimal with minimum number of spaces.
- `%D:n` Print argument signed decimal with minimum n positions.
- `%D;n` Print argument signed decimal with minimum n positions, and pad unused places "0" i.e., 12 in format D: 5 is printed as "00012".
Note: The number "n" is a ONE digit HEX number.
- `%D%` If you want a ":" to follow directly after the number, and not being interpreted as a format modifier.
- `%D% ;` Same as %D%
- `%T` TAB to position in argument, this functions does not work if you are using direct cursor addressing, since the system has no knowledge of these sequences.

For convenience, a CR/LF sequence is automatically sent if the text mode is exited when the current image is completed, thus the first example will work without the CR/LF sequence.

TEXT INPUT

To input text/numbers you will use the TREAD statement, you also need to use the IMAGE and IPRINT to output the \002 and \003 codes to control the command monitor.

TREAD has the following modifiers:

	Operator code
<code>TRead LINE</code>	4
<code>TRead LINE CLR UCH <Xreg></code>	5
<code>TRead LINE CLR NUM <Xreg></code>	6
<code>TRead NUM <Xreg></code>	1
<code>TRead CH <Xreg></code>	2
<code>TRead UCH <Xreg></code>	3

Text mode

<code>TRead RAW CH <Xreg></code>	7
<code>TRead LINE</code>	Will read a line of data to the input buffer.
<code>TRead NUM</code>	Can then be used to read a numerical value from the buffer.
<code>TRead CH</code>	Can be used to read a character from the buffer.
<code>TRead UCH</code>	Does also read a character from the buffer, but it skips all leading spaces and then converts the character to uppercase format. (Uppercase convert does only work for 7-bit characters.) This is more convenient if you will accept both "Y" and "y" as a positive answer to a question.

To be able to use TREAD NUM, TREAD CH or TREAD UCH, you must first execute TREAD LINE to get data into the input buffer and set the read-pointer to the start of the buffer.

<code>TRead LINE CLR UCH</code>	Will clear the input buffer, read a line into the input buffer and then skip all the leading spaces, get the first character and convert it to uppercase.
<code>TRead LINE CLR NUM</code>	Will clear the input buffer, read a line into the input buffer, and then read a numerical value.
<code>TRead NUM/CH/UCH</code>	Can be used to read additional characters/numbers entered on the same line.

All these TREAD statements work with ECHO enabled, and the line-editor is also active, in the same way as it is in the command monitor.

The only exception to this is the:

<code>TRead RAW CH</code>	This statement will read the first character it finds, directly from the input buffer, this is useful if you want to control the terminal completely by the PL code, if no character is available it will return the code 0.
---------------------------	--

Extended register groups

INTRODUCTION

The DMC² hardware and software, such as the resolver or the regulator, are divided into groups. Each group has members where the various values and/or bits can be manipulated by a PL2 program. Hardware and software functions are accessible via a group's members. Group members are implemented as extended registers, Xreg, thereby allowing arithmetic to be performed on them.

Group name	Number	Description	Page
Stack	1	Stack Handling	55
RD1	2	Resolver/Digital Converter #1.	57
RD2	3	Resolver/Digital Converter #2	63
Pg	4	Profile Generator	68
Motor	5	Motor Interface.	73
Reg	6	PID Regulator	79
Gear	7	Electronic Gearbox	86
Tmr	8	System Timers	93
Syslo	9	System I/O	96
Int	10	Interrupt Control.	103
In	11	Digital Input.	109
Out	12	Digital Output.	112
Vector	13	Interrupt Vectors.	114
Capture	14	Capture a Precise Time and Position.	118
Ana	15	Analog I/O.	124
EEprom	16	Non Volatile parameter storage.	127
Comm	17	Serial Communication.	130
RD1Corr	18	Position Correction	133
OptAD	22	Optional A/D Conversion Option M	135
LAN1	28	Local Area Network 1	139
MsgObjLAN1	29	Helper to LAN1	149
LAN2	30	Local Area Network 2	152
MsgObjLAN2	31	Helper to LAN2	153
MultDiv	49	Math, scaling with 64 bit protocol	154
FlashMem	50	Flash memory interface	156
ABIn	52	Input buffer to the Anybus-S modules	157
ABOut	53	Output buffer from the Anybus-S modules	158
DStore	54	Data storage buffers	159
ParArea	55	Non-violated parameter storage.	161

Stack handling

Group name	Number	Description	Page
XENDAT	56	Feedback interface for ENDAT sensor.	163
Counter	57	Count an external hardware event.	172
Identifier	58	Identify each DMC in a group.	175
RDPDATA	59	Generell access to PDATA tables	176
SAnyBus	60	Anybus-S interface (HMS modules)	178
ABInMail	61	Mail message handling with the Anybus-S modules	188
ABOutMail	62	Mails received from the Anybus-S modules	189
ABFBus	63	Fieldbus specific information	190
EN1	64	Connection points for feedback sensors in the DMC2 system	191
EN2	65	See EN1	191
EN3	66	See EN1	191
EN4	67	See EN1	191
IENC	69	Incremental encoder interface	193
ModEN3	72	Extension of EN3 with modula calculation	195
ModEN4	73	Extension of EN4 with modula calculation	195

GROUP MEMBERS. (GROUP XX)

Each group is started with the following heading:

Group	Group No.	Description

Group is the PL name, Group number can be used when communicating in computer mode. The first Group.Member in each group is started with the following heading:

Group.Member	Member No.	Range	Ability	Default

Group.Member is the PL name, Member number can be used when communicating in computer mode.

The range indicate the members numerical range and the Ability is read/write ability according to:

- R = Readable
- W = Write able
- RW = Read- and Write able.
- (W) = Write able under certain conditions
- NYI = Not Yet Implemented

STACK HANDLING

Group	Group No.	Description
Stack	1	The register bank, Stack, is a circular stack that can accommodate 64 register values, 32 bits wide. Since the stack is circular, there is no error trap when pushing the 65th number on the stack; however, the 1st number is overwritten.

GENERAL

The stack group consists of 64 registers (32 bit) that can be used by the application program for temporary storage of data. The function is circular when data is taken (popped) from the stack and linear when data is pushed onto the stack. The system uses the stack for temporary storage of interrupt masks when Int.Mode = 1.

FUNCTION

Data is placed onto the stack with the PUSH statement where the argument can be a register or a constant. Data is taken from the stack with the POP statement where the argument must be a register.

The value in Stack.63 is lost when a PUSH statement is executed.

	Reset status:	After one entry: (PUSH Data1)	After two entries: (PUSH Data2)	After one recall: (POP <reg>)
Stack.63	0	0	0	Data2
.
Stack.5	0	0	0	0
Stack.4	0	0	0	0
Stack.3	0	0	0	0
Stack.2	0	0	0	0
Stack.1	0	0	Data 1	0
Stack.0	0	Data 1	Data 2	Data 1

RELATED ITEMS

PUSH <Xreg> or <lreg>

PUSH <Lval> or <lreg>

POP <Xreg>

Int.Mode = 1 See Group number 10 (Int) for information.

EXAMPLE USAGE

Transfer data to a subroutine for conversion.

Stack handling

```

    PUSH <InData>           ;put data on the stack
    Gosub Convert           ;call conversion routine
    POP <OutData>           ;take converted data from the stack
;Conversion routine
    .register CalcData      ;temporary register
    Define Scale = 123     ;scale factor
Convert:
    POP CalcData           ;take input from the stack
    CalcData = CalcData * Scale ;modify...
    PUSH CalcData          ;put output on stack
    Return                 done....

```

GROUP MEMBERS (GROUP 1)

Group.Member	Member No.	Range	Ability
Stack.0-63	0-63	$-2^{31}..2^{31}-1$	RW

RD1, RESOLVER/DIGITAL CONVERTER #1.

Group	Group No.	Description
RD1	2	Resolver/digital converter #1 that is used for main motor communication and regulation.

GENERAL

The RD1 group contains registers for control and monitoring of the Resolver 1 input, which is, used for primary feedback of motor position and speed. The resolver is supplied with a excitation frequency from the DMC² and outputs two amplitude modulated signals which represents the sine and cosine of the angular position at all times. These analog signals are sampled and converted at a constant rate (4 kHz). From the sine and cosine values, the angle can be calculated as $\arctan(\text{sine}/\text{cosine})$. From this periodic angle measurement, the speed and position of the motor can be derived. The resolver gives an absolute position over one turn. The position unit is Increments (incs), where a two pole (single speed) resolver gives 8192 incs/turn, a four pole resolver gives 16384 incs/turn and so on. The speed unit then becomes Increments/second (incs/s) and the acceleration unit becomes: Increments/second² (incs/s²) In high resolution mode these numbers are multiplied by 8.

Resolver	Resolver speed	High resolution Mode	Standard Mode
		PPR	PPR
2-Pole	1-Speed resolver	65536	8192
4-Pole	2-Speed resolver	131072	16384
6-Pole	3-Speed resolver	196608	24576
8-Pole	4-Speed resolver	262144	32768

FUNCTION

Resolver 1 (RD1) must be set up properly in order to run the motor. The commutation of motor currents depends on proper operation of RD1.

<code>RD1.Mode</code>	Selects the mode of operation for the resolver interface. RD1.Mode = 0 turns off the resolver and resets RD1.Pos.
<code>RD1.Mode = 1</code>	Is the normal setting for standard resolvers. Other values enable special functions.
<code>RD1.SHAdj</code>	Compensates for phase shift in the resolver and cables. The synchronous demodulation principle of the resolver interface requires that this parameter is set to the proper value. A span of +/- 150 is sufficient for all types of resolvers. A measuring routine (PL2 coded) for this parameter is included in the SW package
<code>RD1.RPos</code>	Is used as the reference for all movements in the DMC ² . This relative axis position can be set to any value by the statement Ref Pos <Lval>.

RD1, Resolver/Digital converter #1.

<code>RD1.Pos</code>	Is an absolute position value related to the absolute position the resolver had upon FW initialization.
<code>RD1.AmplAdj</code>	Holds the sign of the error that causes a ResolvErr interrupt.
<code>RD1.Mode = 16</code>	High-resolution mode active.
<code>RD1.Mode = 32</code>	Automatic amplitude compensation active.
<code>RD1.SinGain</code>	Compensation for gain differences between the sine and cosine input amplifier.
<code>RD1.ChkLowLim</code>	When the amplitude falls below this value, a resolver interrupt is generated.
<code>RD1.ChkErr</code>	Last resolver error code.
<code>RD1.ChkNom</code>	Nominal value for resolver amplitude.
<code>RD1.ChkThreshold</code>	The difference between <code>RD1.Ampl</code> and <code>RD1.ChkNom</code> must exceed this value before any action is taken.

RELATED ITEMS

<code>SysIo.PWM0</code>	Controls the amplitude of the excitation signal. An adjustment routine (PL2 coded) for this parameter is included in the SW package. The excitation amplitude is inversely proportional to this value.
<code>Vector.ResolvErr</code>	System interrupts vector for resolver monitoring.
<code>Int.SysMask</code>	Bit mask for system interrupts, bit 7 (bit value 128), enables the monitoring.
<code>Int.SysPend</code>	Bit mask for pending events, bit 7 (bit value 128), indicates the ResolvErr event.
<code>Ref Pos <Lval></code>	Statement for relocation of the relative position of the axis.

EXAMPLE USAGE

```

.
RD1.Mode = 1           ;normal operation
gosub ResCal          ;adjust resolver parameters
.
Ref Pos 0              ;Sets the current reported position to 0

```

GROUP MEMBERS (GROUP 2)

Group Member	Member No	Range	Ability	Default
<code>RD1.Mode</code>	0	0..256	RW	0
	Mode selects whether the <code>RD1</code> is active or not.			
<code>RD1.Mode=0</code>	Inactive			
<code>RD1.Mode=1</code>	Active			

RD1.Mode=3	Active and High-speed mode selected. This mode is recommended for use above 500 Hz (or 30.000 rpm 2-pole speed). It is also suitable for already demodulated resolvers and transducers giving a similar signal, for sensors ranging from analog hall sensors to laser interferometers. The maximum internally supported speed is 32 MHz.
RD1.Mode=5 RD1.Mode=6 RD1.Mode=7	Use RD1 correction table (see RD1CORR group). The uncorrected RD1 position is located in the table (there is 16 values/turn), and the result is added to the RD1 position before the value is used for commutation and regulation.
RD1.Mode=13 RD1.Mode=14 RD1.Mode=15	Use and update RD1 correction table (see group RD1CORR). For the update mode to work, the motor should be running with no load at a recommended minimum high speed of 256000. The system then assumes that all torque fluctuations in the regulator are due to imperfections in the resolver, and adjusts the table to minimize these fluctuations. This adjustment may take several seconds. While the update mode is active, no speed changes are allowed. This automatic mode is most suitable for low-accuracy resolvers such as analog hall-sensors etc. To use the correction for a DMC ² standard resolver, a PL-code routine must be written and the adjustment time increased to minutes or no improvement will occur.
RD1.Mode=16	High resolution mode. Pulses-per-revolution has been increased to accommodate the 14-bit A/D converter capability. High resolution is a new operating mode in RD1 and RD2 separately. One revolution of a 2 pole resolver yields 65536 increments, this is eight times better than the low resolution mode.
i	Several group members must be scaled up or down by 8 when using this mode.
	Pg.Speed = multiply by 8 to achieve the same speed as for low resolution. Pg.Acc/Pg.Ret = multiply by 8 to achieve the same acceleration/deceleration as for low resolution. Reg.Xgain= divide by 8 all gains. Where X is P, I or D for low resolution. Motor.PPR= multiply by 8 for low resolution.
i	When this mode is activated ,the RD2 supervision is also activated, regardless of the mode setting in RD2.Mode. That is, if the RD2 resolver is deactivated, no check is performed. Currently the RD2 can not be compensated. If RD2 is enabled then in the case the RD2.Amp1 falls below RD2.ChkLowLim then a resolver error interrupt is generated and a reason code is stored in RD2.ChkErr. The standard method will generate interrupts as soon as the interrupt vector has been defined and the interrupt has been enabled. This version will start after this mode bit is activated. When you run code that is designed to find values for RD1.SHAdj you should make sure that the interrupt is not enable and that this mode is disabled. And before you enter this mode then clear any pending resolver

RD1, Resolver/Digital converter #1.

error interrupt. Also when running routines in order to adjust the Rdx.singain this mode must be turned off.			
<code>RD1.Mode=32</code>	<p>Automatic temperature compensation using <code>SysIo.PWM0</code>. When this mode is activated the resolver amplitude <code>RD1.Ampl</code> is monitored with the rate specified in <code>SysIo.ChkTmr</code>. The resolver excitation voltage (<code>SysIo.PWM0</code>) is changed in order to maintain a <code>RD1.Ampl</code> value as specified in <code>RD1.ChkNom</code>. No change is made until the difference exceeds the value in <code>RD1.ChkThreshold</code>. In the case the <code>RD1.Ampl</code> falls below <code>RD1.ChkLowLim</code> or we have reached the max/min adjustment then a resolver error interrupt is generated and a reason code is stored in <code>RD1.ChkErrr</code>.</p>		
<p>♦ Warning. If you use this mode make sure that any PL code that manipulates <code>SysIo.PWM0</code> is removed.</p>			
<p>Additional bits: b7 (128) Set CCW rotation.</p> <p>Note. When used as the commutation source, in order to change the motor rotation, you also need to change the rotation definition on the motor. See Rotation definition change.</p>			
<code>RD1.Pos</code>	1	$-2^{31}..2^{31}-1$	R
<p>The 32-bit extended position in "increments", where an electrical turn on the resolver (2-pole) equals 8192 (or 65536) increments. Resolution is approx. 2000-4000 increments/turn or 2-4 increments.</p>			
<code>RD1.Speed</code>	2	$-2^{31}..2^{31}-1$	R
<p>Speed of the resolver in increments-per-second. Resolution is approximately 2000-4000 increments-per-second.</p>			
<code>RD1.Ampl</code>	3	0..32767	R
<p>The combined amplitude of SIN and COS signals from the resolver. This value is used to adjust the phase of the sample/hold signal so that the SIN and COS signals are sampled at their maximum (10000 to 16000). If this value exceeds 16000, the A/D inputs are overloaded and the accuracy drastically decreases. If the value is less than 15000, the accuracy decreases proportional to the signal level.</p>			
<p style="font-size: 2em; margin: 0;">i</p> <p>The content of the <code>SysIo.PWM0</code> register affects both resolver #1 and #2; since both resolvers are driven from the same excitation circuit.</p>			
<code>RD1.SHAdj</code>	4	-150..150	RW
<p>Sample/Hold phase adjustment. (Time offset for sample/hold signal in microseconds.) This value is used to adjust the phase of the sample/hold signal so that the SIN and COS signals are sampled at their maximum, and thereby the <code>RD1.Ampl</code> reaches its maximum value. At this adjustment, the RD1 is tuned to</p>			

RD1, Resolver/Digital converter #1.

				perform accurately up to 30000 rpm for a 2-pole (single speed) resolver. If the <code>RD1.Ampl</code> maximum exceeds 16000 and the corresponding resolver is to be used at low speeds only (maximum of 500-1000 rpm for a 2-pole resolver), this value may be detuned to decrease <code>RD1.Ampl</code> below 16000.
<code>RD1.FiltSpeed</code>	5			Not implemented
<code>RD1.RPos</code>	6	$-2^{31}..2^{31}-1$	R	Relative position counter. $Rd1.Rpos = Rd1.pos - Rd1.PosOffs$
<code>RD1.SinOffs</code>	7	-16384..16383	RW	In High-speed mode, these values are the RD-electronics offset calibration values. The offset values are subtracted from the A/D input values to give <code>SysIo.RD1Sin</code> and <code>SysIo.RD1Cos</code> .
<code>RD1.CosOffs</code>	8	-16384..16383	RW	In High-speed mode, these values are the RD-electronics offset calibration values. The offset values are subtracted from the A/D input values to give <code>SysIo.RD1Sin</code> and <code>SysIo.RD1Cos</code> .
<code>RD1.AmplAdj</code>	9	-1..1	R	!! This is done more efficiently by using automatic correction!! The result from the resolver monitoring performed with the rate set by <code>SYSIO.ChkTMR</code> . If <code>RD1.Ampl < 15500</code> then <code>RD1.AmplAdj = -1</code> If <code>RD1.Ampl > 16000</code> then <code>RD1.AmplAdj = 1</code> If <code>RD2.Ampl < 8000</code> then <code>RD1.AmplAdj = 0</code> If the resolver interrupt is enabled, the <code>RD1.AmplAdj</code> can be used to modify the <code>SysIo.PWM0</code> in order to keep the <code>RD1.Ampl</code> within the range 15500 to 16000 by just adding the value to <code>SysIo.PWM0</code> .
<code>RD1.SinGain</code>	10	-4096 .. +4096		Sine/cosine gain compensation. To be used in high-resolution mode. This is aprox. +-12.5 % adjustment. -4096 -12.5 % less 0 unity (100%) 4096 12.5 % larger This member is used to compensate gain errors between the sine and the cosine amplifier. The value can be found by rotating the motor while searching the <code>SysIo.RD1Sin</code> maximum and <code>SysIo.RD1Cos</code> maximum (preferably in a PL program), then calculate as; $RD1.SinGain = 32768 - \left(\frac{\sin \max * 32768}{\cos \max} \right)$

RD1, Resolver/Digital converter #1.

RD1.ChkLowLim	11	0..16384	RW	Low limit of resolver amplitude, when the amplitude falls below this value a resolver error interrupt is generated. Default setting is normally good enough. (10000)
RD1.ChkErr	12	0..4	RW	The reason code for the last resolver error. 0 = No error. 1 = Excitation voltage has reached it's highest output. 2 = Excitation voltage has reached it's lowest output. 4 = Resolver amplitude has fallen below RD1.ChkLowLim.
i	The error code will not be removed by the system. A user written PL code interrupt routine may clear this error after it has been read. The system will overwrite on next error.			
RD1.ChkNom	13	0..16384	RW	Nominal value for the resolver amplitude. Default setting is normally good enough. (15500)
RD1.ChkThreshold	14	0..512	RW	The difference between the RD1.Ampl and RD1.ChkNom must exceed this value before any action is taken. Default setting is normally good enough. (100)
RD1.PosOffs	15	$-2^{31}..2^{31}-1$	RW	The position offset as a result of a Refpos xx instruction. This is the same as PG.PosOffs.

RD2, RESOLVER/DIGITAL CONVERTER #2

Group	Group No	Description
RD2	3	Resolver/Digital converter #2. Normally used as the master position in the gearbox mode, otherwise free to use by the PL program.

RD2, GENERAL

The RD2 group contains registers for control and monitoring of the Resolver 2 input, which is used primary for gearbox functions. The second resolver (RD2) can be an external resolver mounted on a machine axis or it can also be the resolver on another DMC²-controlled motor. In that case the clocks of the two DMC²'s must be synchronized.

An external resolver must have equal or lower transformation ratio than RD1. The external resolver is supplied with the same excitation frequency as the RD1. In both cases the outputs are two AM signals which represents the sine and cosine of the angular position at all times. These analog signals are sampled and converted at a constant rate (4 kHz). From the sine and cosine values, the angle can be calculated as $\arctan(\text{sine}/\text{cosine})$.

From this periodic angle measurement, the speed and position of the RD2 can be derived.

Resolver	Resolver speed	High resolution Mode	Standard Mode
		PPR	PPR
2-Pole	1-Speed resolver	65536	8192
4-Pole	2-Speed resolver	131072	16384
6-Pole	3-Speed resolver	196608	24576
8-Pole	4-Speed resolver	262144	32768

FUNCTION

RD2.Mode	Selects mode of operation for the resolver interface.
RD2.Mode = 0	Turns off the resolver and reset RD2.Pos.
RD2.Mode = 1	The normal setting for standard resolvers. Other value enables special functions.
RD2.SHAdj	Compensates for phase shift in the resolver and cables. The synchronous demodulation principle of the resolver interface requires that this parameter is set to a proper value. A span of +/- 150 is sufficient for all types of resolvers. A measuring routine (PL2 coded) for this parameter is included in the SW package.
RD2.Pos	Can be used as the input for gearbox functions. The RD2.Pos is an absolute position axis related to the position the resolver had when it was initialized.
RD2.Mode = 16	High-resolution mode.

RD2, Resolver/Digital converter #2

<code>RD2.SinGain</code>	Compensate gain errors between the sine and cosine amplifier.
<code>RD2.ChkLowLim</code>	When the amplitude falls below this value, a resolver interrupt is generated. Default value is normally good (10000)
<code>RD2.ChkErr</code>	Last resolver error code.

RELATED ITEMS

<code>SysIo.PWM0</code>	Controls the amplitude of the excitation signal. An adjustment routine (PL2 coded) for this parameter is included in the SW package.
<code>Vector.ResolvErr</code>	Systems interrupt vector for resolver monitoring.
<code>Int.SysMask</code>	Bit mask for system interrupts, bit 7 (bit value 128), enables the monitoring.
<code>Int.SysPend</code>	Bit mask for pending events, bit 7 (bit value 128), indicates the ResolvErr.
<code>Ireturn sys 128</code>	Return statement.
<code>RD1.AmplAdj</code>	Holds the sign of the error that caused the ResolvErr interrupt.
<code>SysIo.Sync</code>	Flag for synchronization of multiple DMC ² 's.
<code>SysIo.SyncMode</code>	Selects synchronization principle.

EXAMPLE USAGE

```
.
RD2.Mode = 1 ;normal operation
gosub ResCal ;adjust resolver parameters
.
```

GROUP MEMBERS (GROUP 3)

Group member	Member No	Range	Ability	Default
<code>RD2.Mode</code>	0	0..1 8 bit	RW	0 (CW Rotation)
	Mode selects whether the <code>RD2</code> is active or not.			
<code>RD2.Mode=0</code>	Inactive			
<code>RD2.Mode=1</code>	Active			
<code>RD2.Mode=2</code>	Not used			
<code>RD2.Mode=3</code>	Active and High-speed mode selected. This mode is recommended for use above 500 Hz (or 30.000 rpm 2-pole speed). It is also suitable for already demodulated resolvers and transducers giving a similar signal, for sensors ranging from analog hall sensors to laser interferometers. The maximum internally supported speed is 32 MHz, equivalent to 4 kHz			

	externally. (240.000 rpm on a 2 pole motor or 1.2 mm/s for a laser interferometer. The DMC ² resolution is about 0.16 nanometer with a He-Ne laser interferometer.)		
RD2.Mode=5(1+4)	The content of RD2.Speed is filtered before used as input to the gearbox. The filtered speed is available in RD2.FiltSpeed		
RD2.Mode=7(3+4)	The content of RD2.Speed is filtered before used as input to the gearbox. The filtered speed is available in RD2.FiltSpeed.		
RD2.Mode=16	<p>High-resolution mode. Pulses-per-revolution has been increased to accommodate the 14-bit A/D converter capability. High resolution is a new operating mode in RD1 and RD2 separately. One revolution of a 2 pole resolver yields 65536 increments, this is eight times better than the low resolution mode.</p>		
i	<p>Several group members must be scaled up or down by 8 when using this mode.</p>		
	<p>Pg.Speed = multiply by 8 to achieve the same speed as for low resolution. Pg.Acc/Pg.Ret = multiply by 8 to achieve the same acceleration/deceleration as for low resolution. Reg.Xgain= divide by 8 all gains. Where X is P, I or D. Motor.PPR= multiply by 8.</p>		
	♦		
	<p>Additional bits: b7 (128) Set CCW rotation. Note. When used as the commutation source, in order to change the motor rotation, you also need to change the rotation definition on the motor.</p>		
RD2.Pos	1	$-2^{31}..2^{31}-1$	R
	The 32-bit extended position in "increments", where an electrical turn on the resolver equals 8192 increments. Resolution is approximately 2000-4000 increments/turn, or 2-4 increments.		
RD2.Speed	2	$-2^{31}..2^{31}-1$	R
	The resolver speed in increments-per-second. Resolution is approximately 2000-4000 increments-per-second.		
RD2.Ampl	3	0..32767	R
	<p>The combined amplitude of sin and cosCOS signals from the resolver. This value is used to adjust the phase of the sample/hold signal so that the SIN and COS signals are sampled at their maximum (10000 to 16000). If this value exceeds 16000, the A/D inputs are overloaded and the accuracy drastically decreases. If the value is less than 15000, the accuracy decreases in proportion to the signal level. The SysIo.PWM0 value is adjusted so that RDx.Ampl reaches its maximum value. At this adjustment the resolver is</p>		

RD2, Resolver/Digital converter #2

				tuned to perform accurately at up to 30000 rpm for a 2-pole (single speed) resolver. Note. The content of the <code>SysIo.PWM0</code> register affects both resolver #1 and #2; since both resolvers are driven from the same excitation circuit.
<code>RD2.SHAdj</code>	4	-150..150	RW	Sample/Hold phase adjustment (the time offset for sample/hold signal is in microseconds.) This value is used to adjust the phase of the sample/hold signal so that the SIN and COS signals are sampled at their maximum, and thereby the <code>RD2.Ampl</code> reaches its maximum value. At this adjustment, the RD2 is tuned to perform accurately at up to 30000 rpm for a 2-pole (single-speed) resolver. If the <code>RD2.Ampl</code> maximum exceeds 16000, and the corresponding resolver is only used at low speeds (maximum 500-1000 rpm for a 2-pole resolver), this value may be detuned to decrease the <code>RD2.Ampl</code> below 16000.
<code>RD2.FiltSpeed</code>	5	$-2^{31}..2^{31}-1$	R	The filtered result from <code>RD2.Speed</code> when <code>RD2.Mode</code> is 5 or 7. It is calculated as the mean value over the last 4 servo cycle samples of <code>RD2.Speed</code> .
<code>RD2.RPos</code>	6	$-2^{31}..2^{31}-1$	R	Resulting position when the value in <code>RD2.PosOffs</code> is applied.
<code>RD2.SinOffs</code>	7	-16384..16383	RW	In High-speed mode, these values are the RD-electronics offset calibration values. The offset values are subtracted from the A/D input values to give <code>SysIo.RD2Sin</code> and <code>SysIo.RD2Cos</code> .
<code>RD2.CosOffs</code>	8	-16384..16383	RW	In High-speed mode, these values are the RD-electronics offset calibration values. The offset values are subtracted from the A/D input values to give <code>SysIo.RD2Sin</code> and <code>SysIo.RD2Cos</code> .
<code>RD2.AmplAdj</code>	9	-1..1	NIU	NIU.
<code>RD2.SinGain</code>	10	-4096..+4096		Sine/cosine gain compensation. To be used in high-resolution mode. Adjustment: $\pm 12.5\%$ -4096 -12.5 % less 0 unity (100%) 4096 12.5 % larger This member is used to compensate gain errors between the sine and the cosine amplifier. The value can be found by rotating the motor while searching the <code>SysIo.RD1Sin</code> maximum and <code>SysIo.RD1Cos</code> maximum (preferely in a PL program), then calculate as;

	$RD1.SinGain = 32768 - \left(\frac{\sin \max * 32768}{\cos \max} \right)$		
RD2.ChkLowLim	11	0..16384	RW
	Low limit of resolver amplitude, when the amplitude falls below this value a resolver error interrupt is generated.		
RD2.ChkErr	12	0..4	RW
	The reason code for the last resolver error. 0 = No error. 1 = Excitation voltage has reached it's highest output. 2 = Excitation voltage has reached it's lowest output. 4 = Resolver amplitude has fallen below RD2.ChkLowLim.		
i	The error code will not be removed by the system. A user written PL code interrupt routine may clear this error after it has been read. The system will overwrite on next error.		
Reserved	13		
Reserved	14		
RD2.PosOffs	15	$-2^{31} .. 2^{31} - 1$	RW
	The position offset that is used when RD2.RPos is read.		
RD2.FiltLen	16	0..3	RW
	The length of the accumulating speed filter. The filter has variable length. Value Number of samples 0 1 1 2 2 4 3 8 The RD2.FiltLen consists of the number of samples used in the mean value calculation. The resolver should not rotate when the length is changed.		

PG, PROFILE GENERATOR

Group	Group No	Description
Pg	4	Profile Generator. The profile generator calculates the velocity profiles used for trapezoidal movements.

GENERAL

The Profile generator (Pg) group controls the basic motion function. Any movement can be defined as segments of acceleration, constant speed and deceleration and the Pg function accomplishes this.

This function produces so-called Trapezoidal profiles which are movements with three phases, acceleration, constant speed and deceleration. The target position is reached with the speed = 0.

When the Connect instruction is used on this group and for instance the regulator, the values are taken from the same servo cycle. The old DMC² showed values one servo cycle ahead for the Profile group.

FUNCTION

Pg.APos
Pg.ASpeed The profile generator outputs command values for speed and position with 1 ms update rate. These values are used by the regulator as input values. The outputs are **PG.APos** and **Pg.Aspeed** where **PG.APos** is the integrated value of **Pg.ASpeed**.

Pg.Acc To get any output, the Pg.Acc must be set to some value > 0 and that value will be used as the acceleration limit.

Pg.Ret
Pg.Decel Pg.Ret (or. Decel) is used only to determine when it is time to start the deceleration phase and from that point on the actual deceleration is calculated each ms so that the speed will be zero when the position target is reached.

Pg.RSlope The deceleration phase can be smoothed by setting the value of Pg.RSlope > 0. This will provide an exponential velocity change instead of a linear one during deceleration. When Pg.RSlope > 0 the deceleration will initially be higher than the value stated in the parameter Pg.Ret.

Pg.PosSpeed The Pg.PosSpeed determines the top speed of the profiles.

Pg.Rdy The bit register Pg.Rdy reports when the profile is finished.

The profile generator can also take values from an array specifying segments of acceleration and duration time. This makes it possible to define customized profiles. In this case none of the other parameters are involved. The array can have up to 1000 segments.

RELATED ITEMS

Pos Abs <Lval> Move to the absolute position <Lval>.

Pos Rel <Lval> Move to the absolute position (<Lval> + Pg.Apos)

<code>Pos Inc</code>	<code><Lval></code>	Move to the absolute position (<code><Lval> + Pg.DPos</code>)".
<code>Profile Acc</code>	<code><Rline></code>	Generate profile based on array defined at line <code><Rline></code> .
<code>PData</code>	<code><Lrval></code> , <code><Srval></code>	Defines one segment of customised profile.
<code>Pos Mod On</code>	<code>(clr)</code>	Connects additional command values from gearbox function.

EXAMPLE USAGE

```

.
.
Pg.Acc      ,1 000 000      ;set up
Pg.Ret      ,500 000      ;the profile generator
Pg.PosSpeed ,200 000      ;for a simple movement
Pg.RSlope   ,16           ;
pos abs 81920             ;
wait Pg.RDY              ;move ...
.                          ;wait for completion
.
profile acc @Custom
wait Pg.RDY              ;start the custom profile
.                          ;wait for completion
.

; Array of acc. segments defining a profile
Custom:
  pdata acc1 ,time1      ;first segment (acceleration [in/s2];time [ms])
  pdata acc2 ,time2      ;second segment
  pdata acc3 ,time3      ;third segment
.
  pdata 0 0              ;end of array

```

IMPROVEMENTS TO 4.0

The instruction 'Wait `Pg.RDY`' will wait for the profile generator to be finished, this is not the same as saying that the position has been physically reached. The profile generator generates set point values to the regulator and it's up to the regulator to follow the profile based on the regulator settings.

To test these examples a 'stiff' regulator setting is required.

The code examples below, executed on the DMC², will behave as expected. That was not the case on the old DMC. All examples use the resolver high-resolution mode.

MODIFY DESTINATION POSITION WHILE POSITIONING.

```

Let Pg.Acc, 18784200      ; Setup the profile generator
  Let Pg.Decel, Pg.Acc
Let Pg.PosSpeed, 3276800

Pos Abs 1000000          ; Start positioning.
Wait Pg.APos > 500000    ; Wait until we reached a certain position.
Let Pg.DPos, 200000      ; Now, modify the destination position.

```

Pg, Profile generator

```
Wait Pg.RDY           ; And wait until we are there.
Stop
```

MOVE TO A TARGET POSITION 'BEHIND' OUR CURRENT POSITION.

```
Let Pg.Acc, 18784200   ; Setup the profile generator.
Let Pg.Decel, Pg.Acc
Let Pg.PosSpeed, 3276800

Let Pg.Speed, 100000   ; Start movement.
Wait Pg.ASpeed = Pg.Speed ; Wait until speed has been reached.
Wait In.DI1 = 1       ; Wait for external input active.
RefPos 0              ; Set new logical position.
Pos Abs 0             ; Move to the logical zero position.
Wait Pg.RDY          ; And wait until we are there.
Stop
```

VERY SHORT MOTION PROFILES AND/OR VERY HIGH DECELERATION RATES.

```
Let Pg.Acc, 18784200   ; Setup the profile generator.
Let Pg.Decel, Pg.Acc
Let Pg.PosSpeed, 3276800

loop: Clr Tmr.T0       ; Clear timer for time measurement.
Pos Rel 1000          ; Do a small step (modify this even smaller)
Wait Pg.RDY          ; and wait until we are there.
Disp Tmr.T0          ; Display the time required to do the step.

Clr Tmr.T0           ; Do this once every second.
Wait Tmr.T0 > 1000
Goto loop
```

GROUP MEMBERS (GROUP 4)

Group member	Member No	Range	Ability	Default
Pg.Mode	0	0..7	RW	
	Specifies the working mode for the profile generator. This is normally manipulated by the POS statement and is not usually user-modified.			
Bit0 (1) =1	Standard positioning active.			
Bit1 (2) =1	Positioning is now "locked on target", i.e. the deceleration phase has begun.			
Bit2 (4) =1	Acceleration profile is active.			
Bit3..6	NYI			
Bit7 (128) = 1	The outputs are disconnected from the regulator. Intended for external use of the Pg, i.e. virtual master.			
Pg.Acc	1	$0..2^{31}-1$	RW	
	The allowed acceleration in increments-per-second ² .			
Pg.APos	2	$-2^{31}..2^{31}-1$	RW	
	The actual set-position in increments.			

Pg.ASpeed	3	-32767000.. 32767000	RW
	The actual set-speed (velocity) in increments-per-second.		
Pg.Speed	4	-32767000.. 327670000	RW
	The destination speed (velocity) in increments-per-second.		
Pg.PosSpeed	5	0..32767000	RW
	The maximum speed to use during positioning. If set to zero, the Pg.Speed value is used.		
Pg.Decel	6	$0..2^{31}-1$	RW
	The same variable as Pg.Ret. The deceleration rate used for braking when completing a move in increments-per-second ² .		
Pg.Ret	7	$0..2^{31}-1$	RW
	The same variable as Pg.Decel. The deceleration rate used for braking when completing a move.		
Pg.ADecel	8	$0..2^{31}-1$	R
	The actual deceleration used. Differs from Pg.Decel especially if Pg.RSlope is not zero.		
Pg.Dpos	9	$-2^{31}..2^{31}-1$	RW
	The destination position for a positioning. This variable is manipulated by the POS xxx program statements, but can also be manipulated manually.		
Pg.RSlope	10	0..127	RW
	Creates a RC-like slope on the stop ramp at the end of a move. Used to allow a softer stop of the move. Pg.RSlope = 0 gives a straight line and Pg.RSlope = 127 gives a maximum soft stop. NOTE: Using large values of Pg.RSlope so the stop is very soft makes the move's completion time uncertain. Thus, in some cases, it may be more advantageous to wait for Pg.ASpeed to get below a predetermined low value than to wait for Pg.RDY to return TRUE. When Pg.RSlope is used then the initial deceleration will be larger than the setting of Pg.Decel. For Pg.RSlope = 127 the initial deceleration will be $2 * Pg.Decel$.		
Pg.RDY	11	0 1	R
	Used to see if a move Pos xxx or Profile has completed. Returns to 1 when complete and 0 when incomplete. The Pg.Mode can also be used for this, but gives more details.		
Pg.ProScale	12	$-2^{31}..2^{31}-1$	RW
	The scale factor for profiles. Pg.ProScale multiplies the data obtained from the profile and the result is then divided by 1024. If the value in the $PDATA * Pg.ProScale$ is greater than 2^{47} , the profile generator gets an overflow and the profile is aborted.		

Pg, Profile generator

i		NO error message is generated in this situation.	
<code>Pg.PosOffs</code>	13	$-2^{31} \rightarrow 2^{31}-1$	RW
<p>The offset for positions in the profile-generator and R/D conversion to set the "ZERO" position. The REF POS statement generally sets this.</p> <p>This statement refers the position to RD1. If this is not desired, the <code>Pg.PosOffs</code> can be set directly from the PL language.</p> <p>The <code>Pg.PosOffs</code> affects the <code>Pg.APos</code> and <code>Pg.Dpos</code> and <code>RD1.RPos</code> in the following way:</p> <p>ReportedPosition: = ActualPosition - <code>Pg.PosOffs</code> WrittenPosition: = RequestedWrite + <code>Pg.PosOffs</code></p>			
<code>Pg.SRmode</code>	14	0..1	RW
<p>Enables fractional integration of profile speed. Speed settings below 1000 are handled correctly.</p>			
<code>Pg.DConnAPos</code>	15	$-2^{31} \dots 2^{31}-1$	RW
<p>Virtual PG.Apos when the profile generator is disconnected from the regulator. This value is the value used as setpoint for the regulator in that case.</p>			

MOTOR, MOTOR INTERFACE

Group	Group No	Description
Motor	5	The motor interface group

GENERAL

The motor group holds information about the motor and resolver combination used. It is vital that these registers are set up correctly to get maximum performance from the system.

Induction motor definitions	
Rated current	I_N
Power factor	$\cos\varphi$
Line frequency	f_N
Rated speed	n_N
Number of poles	p
DMC rated peak current	$I_{DMCpeak}$

FUNCTION

`Motor.Poles` Sets the number of electrical poles within the motor. Four and six poles are most common. A negative value indicates compensation for the phase order.

`Motor.PPR` The commutation logic needs information about the commutation source resolution (incs/turn), and this should be set here. A two pole resolver gives 8192 incs/turn.

`Motor.PhAlign` Represents the mechanical alignment between the resolver and the motor. The manufacturer can normally define this value. If not, it can be measured with a measuring routine (PL2 coded) included in the SW package.

`Motor.IcalR`
`Motor.IcalS` Since the current control part of the system is analog it needs offset compensation. The values for phase R (`Motor.IcalR`) and phase S (`Motor.IcalS`) can NOT be defined and must be measured at every power up of the system. A measuring routine (PL2 coded) for these parameters is included in the SW package.

RELATED ITEMS

EXAMPLE USAGE

```
Motor.Poles , - 6           ;6 pole motor
Motor.PPR , 24576         ;6 pole resolver
Motor.PhAlign , -19400    ;normal value for ELMO motors
```

Motor, Motor interface

```

gosub Ical
.
Motor.Poles , - 6 ;offset calibrate
Motor.Ppr , 8192 ;6 pole motor
Motor.PhAlign , 18800 ;2 pole resolver
gosub Ical ;normal value for SEM motors

;offset calibrate
    
```

GROUP MEMBERS (GROUP 5)

Group.member	Member No	Range	Ability	Default
Motor.Mode	0	0 – 255	RW	0
Motor.Mode=0	PM-synchronous motor commutation			
Mode=8	Induction motor commutation. Also disables current regulator integration. (see Sysio.Pout for details)			
Mode=8+16	Induction motor commutation with disabled slip compensation in field weakening region. This mode is used for motor parameter tuning only.			
	Additional bits:			
	b0 (1) change rotation definition Note. In order to use this you also need to change the rotation definition on the feedback, commutation source. See Rotation definition change.			
Motor.Comm	1	0-4		
	Commutation source			
.Comm = 0	No source selected			
.Comm = 1	Commutation is taken from the EN1 signal switch (default)			
.Comm = 2	Commutation is taken from the EN2 signal switch			
.Comm = 3	Commutation is taken from the EN3 signal switch			
.Comm = 4	Commutation is taken from the EN4 signal switch			
Motor.Poles	2	(-16385..16385)*2	RW	
	Sets the number of poles on the motor. Use a negative number if the motor rotates in the wrong direction with respect to the resolver. The number of poles should generally be less than <code>Motor.PPR / 128</code> and greater than or equal to <code>Motor.PPR / 4096</code> . If commutation position alignment can be performed at startup by rotating the motor, the <code>Motor.Poles</code> value may be less than <code>Motor.PPR / 4096</code> .			
Motor.PPR	3	(-32768..32767)*2 ⁿ ; n=(0..8)	RW	
	Defines the Pulses-per-revolution the commutation logic has to work with. For the standard resolver:			

			2-Pole: 8192 1 Speed Resolver 4-Pole: 16384 2 Speed Resolver 6-Pole: 24576 3 Speed Resolver 8-Pole: 32768 4 Speed Resolver
<code>Motor.PhAlign</code>	4	-32768..32767	RW
	Defines the Commutation angle alignment. Used to align resolver/encoder to motor 16384 <--> 90 electrical degrees. If motor rotates in the wrong direction when closing the feedback-loop, add 32768 to the used value. (ELMO: -19400, SEM: -14000 (18800))		
<code>Motor.PhDelay</code>	5	-32768..32767	RW
	Compensates for the delay from reading the resolver / encoder position until the output of the commutation angle to the motor, including delay in the drive. Resolution is approximately 1ms (1024 <--> 1000 ms) The optimal value is 1750,(default)		
<code>Motor.IcalR</code>	6	-32768..32767	RW
	Phase R current calibration offset. Used to adjust offsets in the drive electronics. Makes real current = 0 when commanded current = 0.		
<code>Motor.IcalS</code>	7	-32768..32767	RW
	Phase S current calibration offset. Used to adjust offsets in the drive electronics. Makes real current = 0 when commanded current = 0.		
i	The <code>Motor.IcalR</code> and <code>Motor.IcalS</code> steals dynamics from the <code>Reg.TorqPLim</code> and <code>Reg.TorqNLim</code> variables.		
	Maximum <code>Reg.Torq(P/N)Lim</code> for synchronous motor $\text{Reg.TorqxLim} = 8191 - (\text{MAX}(\text{ABS}(\text{Motor.IcalR}), \text{ABS}(\text{Motor.IcalS})) / 4.$		
	Maximum <code>Reg.Torq(P/N)Lim</code> for induction motor $\text{Reg.TorqxLim} = \text{Sqrt}((8191 - (\text{MAX}(\text{ABS}(\text{Motor.IcalR}), \text{ABS}(\text{Motor.IcalS}))/4)^2 - \text{Motor.MagCur}2.$		
	The firmware has no internal check to verify that this condition is met.		
<code>Motor.CommPos</code>	8	-32768..32767	R(W)
	The actual commutation position of the PM-synchronous motor. Set to Zero (or other predefined value) in commutation alignment procedure.		

Motor, Motor interface

INDUCTION MOTOR SPECIFIC MEMBERS

Group.member	Member No	Range	Ability
<code>Motor.Slip</code>	9	-32768..32767	R
	Commanded slip $\frac{2 * Re g.Torque * Motor.ASlipGain}{65536}$		
<code>Motor.SlipGain</code>	10	0..32767	RW
	The <code>Motor.SlipGain</code> is calculated in Formula $Motor.Slip = \frac{I_{DMCpeak}}{\sqrt{2} * I_{2r}} * \omega_{2s} * \frac{10.43}{2}$		
i	Slip gain is temperature dependant and is generally about 25 - 35 % higher than calculated.		
<code>Motor.SlipAngl</code>	11	-2 ³¹ ..2 ³¹ -1	R
<code>Motor.MagCur</code>	12	0..13570	RW
	Magnetization current when motor is running at speeds lower than <code>Motor.BaseSpeed</code> , is calculated as shown in Formula.		
i	A larger value then 13570 may result in internal overflow. If the value is larger than 13570 , a larger DMC² drive must be selected.		
i	<code>Motor.MagCur</code> has to be zero when calibrating current offsets		
<code>Motor.ASlipGain</code>	13	0..32762	R
	The actual slippgain used in commutation. When $abs(RD1.Speed) < Motor.BaseSpeed$ then $Motor.ASlipGain = Motor.SlipGain$ or when $abs(RD1.Speed) > MOTOR.BaseSpeed$ then (Field weakening). $Motor.ASlipGain = \frac{Motor.SlipGain * Abs(RD1.Speed)}{Motor.BaseSpeed}$		
<code>Motor.AMagCur</code>	14	0..13570	R
	The actual magnetization current used in commutation When $abs(RD1.Speed) < Motor.BaseSpeed$ then or when $abs(RD1.Speed) > Motor.BaseSpeed$ then (Field weakening) $Motor.AMagCur = \frac{Motor.MagCur * Motor.BaseSpeed}{Abs(RD1.Speed)}$		

<code>Motor.BaseSpeed</code>	15	0..32767000	RW	The motor speed in inc/sec where field weakening starts.
<code>Motor.MedSpeed</code>	16	0..32767000	RW	When <code>abs(RD1.Speed)</code> is above <code>Motor.MedSpeed</code> then a linear reduction of <code>Motor.ATorqPLim</code> and <code>Motor.ATorqNLim</code> is performed. For induction motors the <code>Motor.ATorqPLim</code> and <code>Motor.ATorqNLim</code> is used instead of <code>REG.TorqPLim</code> and <code>REG.TorqNLim</code> .
<code>Motor.HighSpeed</code>	17	0..32767000	RW	When <code>abs(RD1.Speed) >= Motor.HighSpeed</code> . Both <code>Motor.ATorqPLim</code> and <code>Motor.ATorqNLim</code> is set to zero.
<code>Motor.ATorqPLim</code>	18	-8192 8191	RW	The actual positive torque limitation used in the induction motor regulator. See <code>Motor.HighSpeed</code> , <code>Motor.MedSpeed</code> and <code>Motor.BaseSpeed</code> . To set this limit, use <code>REG.TorqPLim</code> .
<code>Motor.ATorqNLim</code>	19	-8192 8191	R	The actual negative torque limitation used in the induction motor regulator. See <code>Motor.HighSpeed</code> , <code>Motor.MedSpeed</code> and <code>Motor.BaseSpeed</code> . To set this limit, use <code>REG.TorqNLim</code> .
<code>Motor.Temp</code>	20	-32768 32767	RW	The value from an <code>ANA.Inx</code> analog input. The value in <code>Motor.Temp</code> is the motor winding temperature and is used for slippgain temperature compensation. The selected analog input must be adjusted, using <code>ANA.InxRange</code> and <code>ANA.InxOffs</code> , so that the temperature, for which the <code>Motor.SlipGain</code> was given as motor parameter, result in a 0 reading at <code>Motor.Temp</code> . The <code>ANA.Inx</code> should be copied to the <code>Motor.Temp</code> at least a few times but that depends on the thermal time constant of the motor. The Slipgain will be adjusted according to the formula: $Motor.Aslipgain = Motor.Aslipgain * (1 + \frac{Motor.Temp * Motor.TempK}{16384 * 65536})$ Two ways of doing this are: 1. Connect <code>Ana.Inx</code> to <code>Motor.Temp</code> <code>Ana.ConnTMR, 200</code> 2. Let <code>Motor.Temp, Ana.Inx</code>
<div style="display: flex; align-items: center; justify-content: center;"> <div style="font-size: 2em; margin-right: 10px;">i</div> <p>If any more <code>Connect</code> statement is to be used, and with much lower <code>Ana.ConnTMR</code> setting the second way is preferred. But be sure that the code is executed often enough.</p> </div>				
<code>Motor.TempK</code>	21	0..32767	RW	A scale factor use for the temperature compensated slippgain,

Motor, Motor interface

	see header Formula.		
<code>Motor.WeakA</code>	22	0..8192	RW
	Fieldweakening scale factor, when this is 16 field weakening is done as described in <code>Motor.AMagCur</code> above, if greater than 16 the magnetization is reduced faster than $1/ABS(RD1.Speed)$.		
<code>Motor.WeakTm</code>	23	0..255	RW
	Fieldweakening time setting. The calculations are scheduled to save some time in the regulator.		
	The calculation intervall is set by the following values:		
	31 : 32 mS intervall (default value)		
	15 : 16 mS		
	7 : 8 mS		
	3 : 4 mS		
	1 : 2 mS		
	0 : 1 mS		

Reg, PID regulator

GENERAL

The regulator performs a central function in the system. It determines the torque needed to make the motor follow the speed and position commands at all times. It is a parallel PID type of regulator with several feed forward functions. It can be used in several configurations dependant on the applications characteristics. It operates with 1 kHz update rate.

The output is a normalized value of the torque calculated each servo cycle. This output is then used by the commutation logic to create two sinusoidal currents.

FUNCTION

`Reg.PosErr`
`Reg.SpdErr`

The inputs to the regulator are the command values for position, speed and acceleration. Each ms the difference between the command values and the actual values are calculated.

The acceleration is used only for the feed forward parts.

`Reg.IErr`
`Reg.SpdErr`

The position error is used in the Proportional and the Integral part, and the speed error is used in the Derivative part. The feed forward parts are Inertia compensation, Viscous friction, Static friction and Torque.

`Reg.TorqLim`
`Reg.TorqPLim`
`Reg.TorqNLim`

These parts makes up the output which is passed trough a limiting function. There are three different limits, maximum positive torque (`Reg.TorqPLim`), maximum negative torque (`Reg.TorqNLim`) and maximum continuous torque (`Reg.TorqLim`). The output torque is always limited to the range bounded by `Reg.TorqPLim` and `Reg.TorqNLim`.

`Reg.TorqTime`
`Reg.TorqCLim`

If enabled, an additional function will limit the output to `Reg.TorqLim`. If the output has been larger than this limit for more than `Reg.TorqTime` [ms]. This can be seen as a dynamic torque limit.

By adjusting these limits to proper values, the motor can be protected from excessive overheating, permitting the thermal sensor in the motor windings to be able to react fast enough.

`Reg.PGain`
`Reg.IGain`
`Reg.DGain`

The gain and other tuning parameters must be calculated and/or established by testing for each application.

RELATED ITEMS

`Pg.Apos` Command value for position.

`Pg.ASpeed` Command value for speed.

EXAMPLE USAGE

```
.
Reg.TorqLim,8000      ;set maximum torque limit
Reg.TorqCLim,2000    ;set continuos limit to 25 %
Reg.TorqTime,500     ;allow 500 ms of max. torque
or Reg.Mode,64       ;activate dynamic torque limit
.
```

GROUP MEMBERS (GROUP 6)

Group.Member	Member No	Range	Ability
Reg.Mode	0	0..255	R(W)
	<p>Bit-defined, 1. Defines the integration modes of the regulator. Bit 0 and 1 are status bits, showing the internal regulator modes. When Bit3 is activated, the <code>Reg.PGain</code> value must be decreased or set to zero. If both Bit2 and Bit3 are activated (and for simplicity <code>Reg.PGain = 0</code>), we have a Position regulator with integration of position error and speed error. This has the effect of reducing the speed, which with the regulator returns the motor to the correct position after a large position error.</p> <p>The maximum return speed is calculated as (FS = SampleFrequency = 1000 Hz).</p> $Reg.PErrLim = \frac{Reg.SpdErr}{FS * 2^{Reg.SiScale}}$ <p>Thus, the maximum return speed can be calculated to:</p> $FS \frac{Reg.PErrLim}{2^{Reg.SiScale}}$		
Bit0 (1)	Integrate up enabled.		R
Bit1 (2)	Integrate down enabled.		R
Bit2 (4)	Integrate from limited Reg.PosErr.		RW
Bit3 (8)	Integrate from unlimited $Reg.SpdErr / FS * 2^{Reg.SiScale}$		RW
Bit4 (16)	Enable simple digital filter.		RW
Bit5 (32)	Activates the Torque cam, Gear.Campos is used as SetTorq		
Bit6 (64)	This bit is set to enable the use of <code>Reg.TorqCLim</code> .		
Bit7 (128)	When this bit is set the result of the regulator is to be discarded.		
Bit2 = 0 Bit3 = 0 Reg.Pgain<> 0	Position regulator without integration. When both Bit2 and Bit3 are Zero and <code>Reg.PGain</code> is non-zero, the result is a position regulator without integration.		
Bit2 = 0 Bit3 = 0 Reg.Pgain = 0	Speed regulator without integration. When both Bit2 and Bit3 are Zero and <code>Reg.PGain</code> is zero, the result is a speed regulator without integration.		
Bit2 = 1 Bit3 = 0 Reg.PGain<>0	Bit2 is activated when a position regulator with integration of position error is desired.		
Bit2 = 0 Bit3 = 1 Reg.PGain = 0	Bit3 is activated when a velocity regulator with integration of speed error is desired.		
Reg.PGain	1	$-2^{31} .. 2^{31} - 1$	RW
	<p>The Proportional or Position feedback gain of the regulator. A value of 1024 corresponds in 1 unit of Torque generated for 1 unit of position error. Limited by Firmware, not alarmed if a value is written, only a smaller value is read.</p>		

Reg, PID regulator

Reg.IGain	2	$-2^{31}..2^{31}-1$	RW	Integral (Speed or Position or Both) feedback gain of the regulator. A value of 1024 corresponds in 1 unit of Torque generated for 1 unit of integrated position and/or speed error. Limited by Firmware, not alarmed if a value is written, only a smaller value is read.
Reg.DGain	3	$-2^{31}..2^{31}-1$	RW	Derivative or Speed feedback gain of the regulator. A value of 1024 corresponds in 1 unit of Torque generated for FS units of speed error. Limited by Firmware, not alarmed if a value is written, only a smaller value is read.
Reg.PosErr	4	$-2^{31}..2^{31}-1$	RW	The actual positioning error for the system (in increments).
Reg.IErr	5	$-2^{31}..2^{31}-1$	RW	The actual integrated position and/or speed error for the system.
Reg.SpdErr	6	$-32768*FS..32767*FS$	RW	The actual speed error for the system (in increments/seconds).
Reg.PErrLim	7	$0..2^{31}-1$	RW	Proportional or Position Error limit. This limits the maximum error signal that is allowed into the regulator. The user can set the upper level; the lower level is determined by the gain of the regulator.
Reg.IErrLim	8	$0..2^{31}-1$	RW	Integral sum error limit. This limits the maximum error signal that is allowed into the regulator. The user can set the upper level; the lower level is determined by the gain of the regulator.
Reg.DErrLim	9	$0..2^{31}-1$	RW	Derivative or Speed Error limit. This limits the maximum error signal that is allowed into the regulator. The user can set the upper level; the lower level is determined by the gain of the regulator.
Reg.Torque	10	$-32768..32767$	R(W)	The generated Torque. For PM-synchronous motors, this is proportional to motor current. 100% torque = 10V DC current command to the drive = 32767. Note:
i	The value will be 4 times the value of Reg.TorqLim. when operating in limit.			
Reg.TorqLim	11	$0..8191$	RW	Torque or (for PM-synchronous-motors: current) limit. 100% Torque = 8191. This is also torque limitation for the induction motor regulator, to read, use the Motor . group.
Reg.TorqPLim	12	$-8191..8191$	RW	

			Positive Torque or (for PM-synchronous-motors: current) limit.100% Torque = 7800. A negative value means that the motor is forced to generate torque even when at correct position or speed. Positive torque limitation in the induction motor regulator, use <code>Motor.ATorqPLim</code> to read the actual positive torque limit (speed dependend reduction).
<code>Reg.TorqNLim</code>	13	-8191..8191	RW
			Negative Torque or (for PM-synchronous-motors: current) limit.-100% Torque = -7800. A positive value means that the motor is forced to generate torque even when at correct position or speed. This is one way to generate motor-current when performing resolver alignment. Negative torque limitation in the induction motor regulator, use <code>Motor.ATorqNLim</code> to read the actual negative torque limit (speed dependend reduction).
<code>Reg.SiScale</code>	14	0..10	RW
			Speed Integrate Shift Scale factor. The integrated Speed Error is multiplied by $2^{\text{Reg.SiScale}}$. In applications where the regulator sometimes has to operate with very large position errors, a common problem is that the motor appears to be running “at uncontrollable speed” towards the correct position, once conditions allow it. To be able to limit the speed, with which the motor goes to the desired position, the DMC ² regulator has the ability to integrate both the speed error and the limited position error. Since the position and speed errors are of opposite signs when the motor goes towards the correct position, we can control the maximum velocity by limiting the position error that is allowed into the integrator and then summing that error with the speed error (properly scaled). These values cancel when: <i>Position error limit=Reg.Speed / FS * 2^{Reg.SiScale}</i> . Thus, by adjusting <code>Reg.SiScale</code> and <code>Reg.PErrLim</code> , the maximum velocity, which the motor goes towards the correct position, can be set. Since the regulator does not leave the linear mode, this also has a stabilizing effect on conditionally stable systems.
<code>Reg.Pole</code>	15	0..255	RW
			The pole of the digital filter will be placed at <code>Reg.Pole/256</code> . The transfer function of the filter is: $H(z) = b_0 * \frac{z - \frac{\text{Reg.Zero}}{256}}{z - \frac{\text{Reg.Pole}}{256}}$
			b_0 is a scale factor calculated by firmware to make DC gain equal to one. $b_{(0)} = \frac{1 - \frac{\text{Reg.Pole}}{256}}{1 - \frac{\text{Reg.Zero}}{256}}$
			When using the filter as lead link i.e. when then <code>Reg.Zero > Reg.Pole</code> , noise and signal clipping can occur at higher

Reg, PID regulator

	frequencies due to then DC gain adjustment. To view the transfer function: $plot : 20 * \log H \left(e^{2\pi * i * \frac{f}{1000}} \right) $		
Reg.Zero	16	0..255	RW
	The zero of the digital filter will be placed at <code>Reg.Zero/256</code> . see <code>Reg.Pole</code>		
Reg.SetTorq	17	-8191..8191	RW
	Torque feed forward. The content of this variable is directly added to the generated torque. It can be used to add a torque offset, or to be a direct torque setpoint input, in case a torque control mode is desired.		
	Note. A torque control mode is achieved by setting all gains to zero.		
Reg.InertiaP	18	0..32767	RW
	Acceleration feed forward constant in positive direction.		
Reg.InertiaN	19	0..32767	RW
	Acceleration feed forward constant in negative direction.		
Reg.InertSF	20	0..255	RW
	Shiftfactor for acceleration feedforward. Where 8192 is 100% torque. The feed forward torque is calculated as: $\frac{Pg.Acc}{2^{Reg.InertSF}} * Reg.InertiaP$ 65563		
	The <code>Reg.InertSF</code> must be set so that $Acc/2^{Reg.InertSF}$ is within 16 bits (32767) to avoid overflow (internally clamped to 16 bits). This means that one must know the maximum value of the acceleration for the particular application to properly use acceleration feed forward. Example: For an application with <code>Pg.Acc</code> = 3678986 and a feed forward of 2293 (28% torque) in both directions. 1. Select <code>Reg.InertSF</code> to 7 $3678986 \div 2^7 = 28742$ 2. Calculate <code>Reg.InertiaP</code> $\{2293 \times 65536\} \div 28742 = 5228 = REG.InertiaN$		
Reg.StatFric	21	-8191..8191	RW
	Compensate for static friction. Produces a constant feed forward torque, with the sign of the speed reference. This has effect only when speed is non-zero.		
	◆ Not to be used together with gearbox functions!		
Reg.ViscFric	22	0..8191	RW

		Compensate for viscous friction. It produces a feed forward torque proportional to speed reference.	
<code>Reg.ViscSF</code>	23	0..255	RW
		Scale factor for viscous friction compensation.	
<code>Reg.TorqCLim</code>	24	0..8191	RW
		Continuous torque or (for PM-synchronous-motors: current) limit. 100% Torque = 8191. This limit will be used after <code>Reg.TorqTime</code> has expired.	
<code>Reg.TorqTime</code>	25	0..8191	RW
		Timer for activation of <code>REG.TorqCLim</code> .	
<code>Reg.DerrFltSF</code>	26	0..7	RW
		Low pass filter limiting the noise at the SpdErr signal. <code>Reg.DerrFltSF = 0</code> , $y(n) = x(n)$. Default at startup, which means no filter. <code>Reg.DerrFltSF = 1</code> , $y(n) = 0.5*y(n-1)+0.5*x(n)$, time const ~ 2,5 ms. <code>Reg.DerrFltSF = 2</code> , $y(n) = 0.75*y(n-1)+0.25*x(n)$ time const ~ 8 ms. <code>Reg.DerrFltSF = 3</code> , $y(n) = 0.875*y(n-1)+0.125x(n)$ time const ~ 18 ms. <code>Reg.DerrFltSF = 4</code> , ~ 40 ms. <code>Reg.DerrFltSF = 5</code> , ~ 75 ms.	
<code>Reg.TrqLimFlgs</code>	27	0..3	R
		Bit information on Torqlimit status. Bit 0 (1) indicates if the output torque is limited by <code>Reg.TorqLim</code> . Bit 1(2) indicates if the dynamic limit has been activated, that is if the output is limited by <code>Reg.TorqCLim</code> .	
<code>Reg.PosRef</code>	28	$-2^{31}..2^{31}-1$	R
		The actual reference value used by the regulator. Normally this is the sum of <code>PG.Apos</code> and <code>Gear.PosRef</code> . In the case of disconnected Profile Generator, the <code>Pg.DconnApos</code> is used instead.	

GEAR, ELECTRONIC GEARBOX

Group	Group No	Description
Gear	7	Electronic gearbox. The EN2 is used as master position input to generate scaled set speed and position that is added to the set value generated from the profile generator. The position is calculated as: <i>OutputPosition := En2-position * InGear / Outgear</i>

GENERAL

The gearbox function makes it possible to set up an electrical gearing between a DMC² and an external position sensor (resolver or encoder) or another DMC². This gearing can be used for electrical axis applications or for camshaft emulation. In both cases, the input can be the En2 input (En2.Speed) or an internal constant frequency of 1 kHz. It operates with 1 kHz update rate.

Be aware that if the filter on Rd2 is activated, the source for the gearbox will be forced as Rd2. To use EN2 as source, turn off the filter.

Gear.In is the number of cogs on the input gear and *Gear.Out* is the number of cogs on the output gear.

If $\frac{GEAR.In}{GEAR.Out} > 1$ The gains of the regulator may have to be reduced by a corresponding factor because the RD conversion circuit (in the analog system) has a noise level of approx. ± 4 increments. If this noise level is amplified by, for example, a factor of 100 due to the gearing, the result would be a very "jumpy" run.

FUNCTION

<i>Gear.In</i>	The number of cogs on the input gear.
<i>Gear.Out</i>	The number of cogs on the output gear.
<i>Gear.Speed</i>	The input value is multiplied by the ratio <i>Gear.In</i> / <i>Gear.Out</i> . The result after each sample. If <i>Gear.In</i> / <i>Gear.Out</i> > 1 the Gains of the regulator may have to be reduced by a similar factor because the RD-conversion circuits (that is an analog system) has a noise level of around 4 increments. If this noise level is amplified by, for example, a factor of 100 due to the gearing, the result would be a very "jumpy" run. When operating in CAM-mode the <i>Gear.Speed</i> represents the speed of the Cam function.
<i>Gear.Pos</i>	The <i>Gear.Speed</i> value is then added to the sum of previous values. Since this is an incremental function it is possible to change the ratio during runtime. There is also a ramp function for smooth activation so that the gearing can be turned on when the external master is rotating. The output, <i>Gear.Pos</i> , can also be used as an index

position, pointing to values in an array of waypoints in the camshaft emulation. The way points are defined in a Pdata array with up to 1000 rows

Since the camshaft is a repetitive function, the array must be defined with continuous segments. The camshaft profile between waypoints is determined by linear interpolation between adjacent waypoints.

<code>Gear.Incr</code>	Sets the number of "cogs" that Gear.In increases with each cycle (mS) The momentary value can be observed in Gear.InAct.
<code>Gear.Offset</code>	The resulting Gear.Pos can also be modified directly with Gear.Offset.
<code>Gear.CamPos</code>	The resulting position, Gear.CamPos, is the current profile value (interpolated) times Gear.CamScale.
<code>Gear.CamCurLine</code>	Indicate relative position in cam.

RELATED ITEMS

PData <Lrval> Defines one waypoint of cam profile.

Pos Mod On (clr) Connects additional command values from gearbox function.

EXAMPLE USAGE

```

Gear.CamLine , @CamProf ;pointer to profile array
Gear.CamLen , 100 ;length of profile
Gear.CamScale , 1024 ;scale of profile
Gear.In , 1000 ;set up gearing
Gear.Out,1000 ;
Pos Mod On clr ;
Gear.Incr , 1 ;
;this will give 1000 ms ramp time of
;gearing
.
; Array of waypoints defining a cam profile
CamProf:
pdata 0 ;first point
pdata <pos1> ;next point
pdata <pos2> next point
pdata <pos3> next point
pdata <pos4> next point
pdata <pos99> last point

```

POSITION LOCK CAM

The gearbox is also used to scale the input signal for the "Position lock CAM". In this mode, the system emulates a cam-wheel. The CAM profile is defined by a number of PDATA statements and the gearbox is used to create the index into that table.

The start of the table is set by the Gear.CamLine variable and the length (in

Gear, Electronic gearbox

PDATA statements), is set by Gear.CamLen variable. Each PDATA statement corresponds to 1024 increments.

If there is a 2-pole resolver on RD2 and you want 2 turns on RD2 equal one turn on the CAM, and 17 PDATA statements are desired.

The PDATA vector length is $17 * 1024 = 17408$ increments.

Two turns on RD2 is $8192 * 2 = 16384$ increments.

To map 16384 to 17408, the gearbox must be setup like this:

Set *Gear.In* = 17 and *Gear.Out* = 16

to accomplish this.

Fill the PDATA statements with the positions for the corresponding index values. The positions given in the PDATA statements are scaled by the Gear.CamScale, which has the same function as the PG.ProScale.

In this version, only linear interpolation between index points is possible. The scaling is disabled when incremental cam is used.

TIME LOCKED CAM

The same as Position Lock cam but RD2 is replaced by a time function to generate the index into the table.

MASTER/SLAVE

In this mode the gearbox is used to follow a value given by a master. The master can be as simple as an analog input, in this case a CONNECT command can be used to connect an analog input to Gear.SlaveSPos. The master can also be another DMC² unit over the LAN, in that case the distributed position can be written to Gear.SlaveSPos by a user written PL program.

In both cases the Gear.SlaveSPos must be written in a timely manner and the firmware will automatically calculate the time difference between the writes and store that value in Gear.SlaveUTime.

The simple interpolation is done internally as,

$$Gear.SlaveSpeed = \frac{Gear.SlavePos - Gear.SlaveAPos}{Tmr.Abs - Gear.SlaveUTime}$$

If *Gear.SlaveFixTm* <> 0 the calculation will use the fixed value instead.

INCREMENTAL CAM

This mode adds an indexing function to the cam. Each time the index to the cam table wraps, the Gear.CamInc is added to the output. Gear.CamLen must be set to the "number of Pdata files minus one". Gear.CamInc should normally be set to the last value in the Pdata table. The scaling with Gear.CamScale is disabled in this mode.

GROUP MEMBERS (GROUP 7)

Group.Member	Member No	Range	Ability
<code>Gear.</code>	0	0..255	RW
		Set the operating mode of the Electronic Gearbox.	
Bit0 (1) <code>Gear.Mode=0</code>		Gearing is not active. In this mode the <code>Gear.Pos</code> may be used as a set-position offset value by the PL program.	
<code>Gear.Mode=1</code>		Gearing is active.	
Bit1 (2) <code>Gear.Mode=2+1</code>		Gearing into index of "POSITION LOCK CAM".	
Bit2 (4) <code>Gear.Mode=4+1</code>		EN2 is not used as source for the Gear box. Instead, the source POS is incremented by one for each servo cycle. i.e. 1000 tics/second. This allows the POS LOCK CAM to be used as a TIME LOCK CAM.	
Bit3 (8)		Reserved.	
Bit4 (16)		Reserved.	
Bit5 (32) <code>Gear.Mode=3+2+1</code>		Master/slave mode with simple interpolator. The master position is given in <code>Gear.SlaveSPos</code>	
Bit6 (64)		Reserved, Master/slave mode, interpolator with speed feed forward.	
Bit7 (128)		Incremental cam active. (<code>Gear.CamLen</code> must be <number of lines-1>).	
<code>Gear.In</code>	1	-32767..32767	RW
		Specifies the number of cogs on the input gear wheel. It may be a negative value. It is possible to clamp the maximum value independent of the size of assigned value.	
<code>Gear.Out</code>	2	1..32767	RW
		Specifies the number of cogs on the output gear. It may NOT be a negative value. It is possible to clamp the maximum value independent of the size of assigned value.	
		The <code>Gear.In</code> and <code>Gear.Out</code> members has been corrected to clamp to the maximum value independent of the size of the value assigned.	
<code>Gear.Speed</code>	3	-32767 000 .. 32767 000	R
		This is the geared speed from En2 or from the CAM profile	
<code>Gear.Pos</code>	4	$-2^{31}..2^{31}-1$	R(W)
		This is the geared position from En2. Because the gearing is incremental, the position starts from the current value when <code>Gear.Mode</code> is set to 1. Writing <code>Gear.Pos</code> when gearing is active may result in a "lost position". The incremental gearing also allows changes of <code>Gear.In</code> and <code>Gear.Out</code> anytime during Gear operation. However, this is not recommended during CAM operation.	

Gear, Electronic gearbox

i	By using RD2.Mode 5 or 7 the input to the gearing is filtered.		
Gear.CamLine	5	1..8191	RW
Specifies start of CAM profile. The Gear.CamLine points to the first PDATA statement that defines the profile.			
Gear.CamLen	6	2..4000	RW
Specifies the length of the CAM profile in PDATA statements. Each PDATA statement corresponds to an index-position of 1024 increments. The profile wraps from the last statement to the first. The normal positioning and speed commands (POS ABS etc.) are usable during CAM operation, since both GEAR and CAM modes generate an offset that is fed together with the normal position into the regulator. For incremental cam, Gear.CamLen should be set to <length -1>.			
Gear.CamPos	7	$-2^{31}..2^{31}-1$	R(W)
The resulting CAM PROFILE position.			
Gear.CamScale	8	$-2^{31}..2^{31}-1$	R(W)
Scale factor for CAM profiles. The data obtained from the profile is multiplied by GEAR.CamScale and the result is then a divided by 1024. If the value in the PDATA * Gear.CamScale is greater than 2^{47} , the generator gets an overflow and the CAM motion is suspended while the overflow is present. The scaling of the cam is disabled when the incremental cam is used.			
i	The POS LOCK CAM does currently only use linear interpolation. Therefore, if the segment length (the time each PDATA is active) exceeds a few milliseconds, the motor may not run as smooth as desired.		
Gear.Offset	9	$-2^{31}..2^{31}-1$	R(W)
<p><u>Gear mode.</u> The offset is calculated circularly so that continues modification of Gear.Offset is possible</p> <p><u>Cam mode.</u> The offset is calculated circularly but not in respect to the length of the CAM. In order to move within a single CAM turn a user can calculate the offset as: <i>Gear.Offset = offs modulo cam_length.</i></p> <p>Note. When the 'POS MOD ON xxx' instruction is executed this member is cleared, to avoid startup movement. This may change in a future release so that the Gear.Offset is used when the start position is calculated.</p>			
Gear.Incr	10	0.. 32767	RW
Ramp constant for the gearing. The actual value of Gear.In changes with this value every servo cycle. It can be used for			

			soft activation of the gearbox. Set to 8191 for no ramp.
<code>Gear.InAct</code>	11	-32767..32767	R
			Actual value for <code>Gear.In</code> , after the ramp function.
<code>Gear.OffsetAct</code>	12	$-2^{31}..2^{31}-1$	R(W)
			Actual value of the offset, after the ramp function, added to the result of the gear or cam function.
<code>Gear.OffsetInc</code>	13	0..32767	RW
			Ramp constant for the offset. The actual value of <code>Gear.OffsetAct</code> changes with this value every ms. It can be used for soft phase shifting of the gearbox. Set to 32767 for no ramp.
<code>Gear.SlaveSpeed</code>	14	$-2^{31}..2^{31}-1$	R(W)
			Master/Slave, The Slave's speed.
<code>Gear.SlaveSPos</code>	15	$-2^{31}..2^{31}-1$	RW
			Master/Slave, The Slave's set position. For the simple interpolator, when writing to <code>Gear.SlaveSPos</code> the firmware will calculate the speed required to reach the new position based on the current time and the value of <code>Gear.SlaveUTime</code> and store this speed in <code>Gear.SlaveSpeed</code> .
<code>Gear.SlaveAPos</code>	16	$-2^{31}..2^{31}-1$	R(W)
			Master/Slave, The Slave's actual position
<code>Gear.SlaveUTime</code>	17	$-2^{31}..2^{31}-1$	R(W)
			Master/Slave, The time for the last write to <code>Gear.SlaveSPos</code> . This is the lower 16 bits of <code>Tmr.Abs</code> .
<code>Gear.CamInc</code>	18	$-2^{31}..2^{31}-1$	RW
			Incremental cam period. Should be set to the last value in the Pdata array used as cam.
<code>Gear.CamCurLine</code>	19	0..65535	R
			Indicate relative position in cam. By adding <code>Gear.CamLine</code> to this member the current PData statement line is achieved.
<code>Gear.CamSpeed</code>	20	$-32767*FS..32767*FS$	R
			The speed of the 'CAM'. Only valid when <code>Gear.CamInterp</code> \neq 0.
<code>Gear.CamAccel</code>	21	$0..2^{31}-1$	R
			The acceleration of the 'CAM'. Due to the relatively low resolution in speed, this signal is not very useful. Only valid when <code>Gear.CamInterp</code> \neq 0. Currently not used for feed-forward.

Gear, Electronic gearbox

<code>Gear.CamInterp</code>	22	0..2	RW
0=	Linear interpolate between two points as STD.		
1=	Cubic trigonometric interpolation (4 points). The interpolator interpolates through each specified point. This may introduce undershoots, but by carefully selecting points a very nice smooth cam can be achieved.		
2=	Cubic BSpline interpolation (4 points). Does not interpolate through the control points, and does not undershoot. Normal PDATA lines are used as for the linear interpolation and it is possible to switch between the interpolation modes in real time, and also modify points in real time.		
i	Be aware that the motor will move to the new position with no speed limit! This is of the same nature as modifying the point we are currently interpolating.		
<code>Gear.SlaveFixTm</code>	23	0..255	
	Fix time base for virtual master mode of gearbox		
<code>Gear.PreCamLine</code>	24	See Gear.Camline	
	Preset value for Gear.Camline		
<code>Gear.PreCamLen</code>	25	See Gear.Camlen	
	Preset value for Gear.Camlen		
<code>Gear.PreCamScale</code>	26	See Gear.CamScale	
	Preset value for Gear.CamScale		
<code>Gear.PreCamInc</code>	27	See Gear.CamInc	
	Preset value for Gear.CamInc		
<code>Gear.PreCamSet</code>	28		
	Force setting of preset values by writing to Gear.PreCamSet when written the above Preset values are copied into the corresponding gearbox parameters. This allowing for dynamic switching of cam parameters		
<code>Gear.CamPeriods</code>	29	$-2^{31}..2^{31}-1$	RW
	Number of periods of the cam. This value increments for each period of the cam, (signed).		
<code>Gear.PosRef</code>	30	$-2^{31}..2^{31}-1$	R
	Actual reference out from the Gear/Cam function that are used by the regulator. Valid only if Pos Mod On has been activated. Shows Gear.Pos in the case of gearing and shows Gear.CamPos in the case of cam usage.		

TMR, SYSTEM TIMERS

Group	Group No	Description
Tmr	8	System timers for timekeeping, etc.

GENERAL

The timer system is based on the cycle time of the DMC² software. A crystal - controlled frequency (40 MHz) is divided down to a 1 kHz cycle using the CPU's internal interrupt system. This 1 ms cycle time is then used for all timer functions. Due to crystal tolerances, 1 ms in one DMC² is not exactly the same as 1 ms in another DMC². Time measurements must therefore be non-critical in application usage. To achieve simultaneous cycles in several DMC²'s the 1 kHz clock can be synchronized between units.

FUNCTION

`Tmr.Abs` The timer system has one absolute measurement in the `Tmr.Abs`. This counter holds the number of cycles since the last power up (or reset). The timers are all 32 bit wide, except for the word wide `Tmr.CycInt`, which means that they will wrap around after 596.5 hours (24.9 days).

`Tmr.CycInt` The `Tmr.CycInt` is intended to be used for generating a cyclical event (timer interrupt) within the application software. The rest of the timers (`Tmr.T0` through `Tmr.T3`) are free for application usage

RELATED ITEMS

`Vector.CycInt` System interrupt vector for timer interrupt.
`Int.SysMask` Bit mask for system interrupts, bit 0 (bit value 1) enables the cyclical event.
`Int.SysPend` Bit mask for pending events, bit 0 (bit value 1) indicates the cyclical event.
`Ireturn sys 1` Return statement.

EXAMPLE USAGE

```

    .
    Vector.CycInt , @CycEvent      ;pointer to service routine
    Tmr.CycInt , 100                ;100 ms event rate
    or Int.SysMask , 1              ;enable the event
    .
    clr Tmr.T2                      ;reset timer T2
    wait Tmr.T2 > 186               ;wait here for 186 ms
    .

; Cyclical event
CycEvent:
    Ireturn sys 1                  ;return from event

```

Tmr, System timers

GROUP MEMBERS (GROUP 8)

Group.Member	Member No	Range	Ability	Default
<code>Tmr.Abs</code>	0	$-2^{31}..2^{31}-1$	RW	
System absolute time (in milliseconds) since startup, wraps to negative after about 24.8 days				
<code>Tmr.CycInt</code>	1	0..65535	RW	
<p>To generate a cyclical timer interrupt, use <code>Tmr.CycInt</code> to setup the interrupt frequency. The time between the interrupts is specified in servo cycles, (at FS) where each servo cycle is currently 1 ms.</p> <p>When <code>Tmr.CycInt</code> is set, the internal interrupt-timer is also set to the same value.</p> <p>Therefore, the first interrupt occurs after <code>Tmr.CycInt</code> servo cycles.</p> <p>To generate a single-shot interrupt, setup <code>Tmr.CycInt</code> and enable the interrupt;</p>				
i Do not re-enable the interrupt when exiting the interrupt routine.				
<code>Tmr.T0</code>	2	$-2^{31}..2^{31}-1$	RW	
<p>This is the time-value for timer 0. The timers are implemented by using an adjustment offset from an absolute time. By writing to <code>Tmr.T0</code> the offset, <code>Tmr.A0</code> changes so <code>Tmr.T0</code> shows the desired time.</p> <p>To increment or decrement this time, by a fixed amount without risking that the timer will change values during the operation, add or subtract from <code>Tmr.A0</code> instead of <code>Tmr.T0</code>.</p>				
<code>Tmr.A0</code>	3	$-2^{31}..2^{31}-1$	RW	
This is the offset adjustment value for timer 0.				
<code>Tmr.T1</code>	4	$-2^{31}..2^{31}-1$	RW	
<p>This is the time-values for timer 1. The timers are implemented by using an adjustment offset from an absolute time. By writing to <code>Tmr.T1</code> the offset, <code>Tmr.A1</code> is changed so <code>Tmr.T1</code> shows the desired time.</p> <p>To increment or decrement this time, by a fixed amount without risking timer value changes during the operation, add or subtract from <code>Tmr.A1</code> instead of <code>Tmr.T1</code>.</p>				
<code>Tmr.A1</code>	5	$-2^{31}..2^{31}-1$	RW	
This is the offset adjustment value for timer 1.				
<code>Tmr.T2</code>	6	$-2^{31}..2^{31}-1$	RW	
<p>This is the time-values for timer 2. The timers are implemented by using an adjustment offset from an absolute time. By writing to <code>Tmr.T2</code> the offset, <code>Tmr.A2</code> is changed so <code>Tmr.T2</code> shows the desired time.</p>				

	To increment or decrement this time, by a fixed amount without risking timer value changes during the operation, add or subtract from <code>Tmr.A2</code> instead of <code>Tmr.T2</code> .			
<code>Tmr.A2</code>	7	$-2^{31}..2^{31}-1$	RW	
	This is the offset adjustment values for timer 2.			
<code>Tmr.T3</code>	8	$-2^{31}..2^{31}-1$	RW	
	This is the time-values for timer 3. The timers are implemented by using an adjustment offset from an absolute time. By writing to <code>Tmr.T3</code> the offset, <code>Tmr.A3</code> is changed so <code>Tmr.T3</code> shows the desired time. To increment or decrement this time, by a fixed amount without risking timer value changes during the operation, add or subtract from <code>Tmr.A3</code> instead of <code>Tmr.T3</code> .			
<code>Tmr.A3</code>	9	$-2^{31}..2^{31}-1$	RW	
	This is the offset adjustment value for timer 3.			
<code>Tmr.Nudge</code>	10	± 16 bit	RW	0
	This member makes it possible to adjust the time for the next timer interrupt. Reading this member will always return a zero. Writing this signed value will adjust the next timer interrupt either ahead or behind. The adjustment is done immediately and will then continue with the value programmed in <code>Tmr.CycInt</code> .			
i	In the case an adjustment will result in an immediate timeout then the <code>Tmr.CycInt</code> interrupt pending bit will be set. This can result in very high interrupt frequency.			

SysIo, System I/O

SysIo, SYSTEM I/O

Group	Group No.	Description
SysIo	9	System I/O for internal supervisory functions etc.

GENERAL

The SysIo group holds information mainly for monitoring and control of the hardware system. There are also some members in this group for adjusting the software flow.

Most of the registers in this group are normally not used in application programs.

FUNCTION

SysIo.Pin
SysIo.Pout

The logical interface between the software system and the digital hardware. In these registers individual bits can be manipulated to reset and enable the output power stage and also monitor error events in the hardware.

SysIo.Led
SysIo.Pout

The LED's on the DMC² front plate are manipulated as bits in the SysIo.Led register except for CPU Ok, which is handled from the SysIo.POut byte.

SysIo.PWM0

SysIo.PWM0 controls the amplitude of the resolver supply signal, the range of 3 to 13 V is mapped to 255 - 0 in this parameter. A measuring routine (PL2 coded) for this parameter is included in the SW package.

NOTE! Not fully backward compatible from Ver.4.0. or later.

SysIo.ChkTmr

SysIo.ChkTmr sets the rate for system monitoring of PosError, Bleederload, Overtemperature and Resolver errors.

SysIo.Sync
SysIo.SyncMode

SysIo.Sync and SysIo.SyncMode controls the synchronization functions needed for some gearing applications.

SysIo.ADC8

SinCos encoder (sin).

SysIo.ADC9

SinCos encoder (cos).

SysIo.ADC10

Strain gauge input # 1.

SysIo.ADC11

SysIo.SysTime

Indicate system time usage in 100 ns.

SysIo.FBCTime

Indicate field bus communication time usage in 100 ns.

SysIo.Servorate

Servo cycle frequency in Hz.

SysIo.SyncRate

The synchronization signal frequency.

RELATED ITEMS

<code>Vector.ResolvErr</code>	System interrupt vector for resolver monitoring.
<code>Vector.BleedErr</code>	System interrupt vector for bleeder monitoring.
<code>Vector.TempErr</code>	System interrupt vector for temperature monitoring.
<code>Vector.PosErr</code>	System interrupt vector for position monitoring.
<code>Int.SysMask</code>	Bit mask for system interrupts.
<code>Int.SysPend</code>	Bit mask for pending events.

EXAMPLE USAGE

```

SysIo.ChkTmr , 25      ;25 ms check rate
.
or SysIo.POut , 1      ;set CPU Ok led alive
or SysIo.POut , 4      ;reset the power stage
bclr SysIo.POut , 4    ;with a short pulse
or SysIo.POut , 2      ;enable the power stage
or SysIo.POut , 8      ;activate the brake relay
or SysIo.Led , 1+8     ;set LED1 and LED 4 alive
.

```

GROUP MEMBERS (GROUP 9)

Group.Member	Member No.	Range	Ability	Default
<code>SysIo.PIn</code>	0	0..255	R	
System parallel input port (on CPU board).				
<code>Bit0 (1)</code>	High voltage level	0 = Active	R	
<code>Bit1 (2)</code>	Comp. current sign V	2 = Active	R	
<code>Bit2 (4)</code>	Comp. current sign U	4 = Active	R	
<code>Bit3 (8)</code>	Motor Temp. High	0 = Active	R	
<code>Bit4 (16)</code>	Drive Temp. High	0 = Active	R	
<code>Bit5 (32)</code>	Current Regulator fault	0 = Active	R	
<code>Bit6 (64)</code>	Shunt regulator active (on/off) (Dynamic Brake)	0 = Active	R	
<code>Bit7 (128)</code>	Short circuit, power stage	0 = Active	R	
<code>SysIo.POut</code>	1	0..255	RW	
System parallel output port (on CPU board).				
<code>Bit0 (1)</code>	Green LED on front panel CPU OK.	1 = Active	RW (CPUA)	
<code>Bit1 (2)</code>	Software enables.	2 = Active	RW (SWEN)	
<code>Bit2 (4)</code>	Power reset.	4 = Active	RW (PRES)	

SysIo, System I/O

	Pulse duration. 100 μ s.	(Pulse)	
Bit3 (8)	Motor Brake release	8 = Active	RW (BRRL)
Bit4 (16)	Master/Slave Direction	16 = Master	RW (MASL)
Bit5 (32)	Disable current regulator hardware integrator. This bit controls the behavior of the drive's motor current regulator. With this bit set the integrator of the motor current regulator is disabled. Also affected by setting Motor.Mode = 8 (Induction motor)		
Bit6 (64)	Reserved.		
Bit7 (128)	Reserved.		
<code>SysIo.Led</code>	2	0..255	RW
	LED control port on I/O board, visible on front panel. LD3 to LD10 are user-programmable.		
i	The CPU OK led is controlled by <code>SysIo.POut</code> bit 0.		
Bit(0) 1	LED1 (GREEN)	0,1	RW
Bit1 (2)	LED2 (YELLOW)	0,2	RW
Bit2 (4)	LED3 (YELLOW)	0,4	RW
Bit3 (8)	LED4 (YELLOW)	0,8	RW
Bit4 (16)	LED5 (RED)	0,16	RW
Bit5 (32)	LED6 (RED)	0,32	RW
Bit6 (64)	LED7 (RED)	0,64	RW
Bit7 (128)	LED8 (RED)	0,128	RW
<code>SysIo.RD1Sin</code>	3	-32767..32767	R
	Demodulated sinus input from resolver #1.		
<code>SysIo.RD1Cos</code>	4	-32767..32767	R
	Demodulated cosinus input from resolver #1.		
<code>SysIo.RD2Sin</code>	5	-32767..32767	R
	Demodulated sinus input from resolver #2.		
<code>SysIo.RD2Cos</code>	6	-32767..32767	R
	Demodulated cosinus input from resolver #2.		
<code>SysIo.ADC0</code>	7	0..16383	R
	System analog input channel #0. Used for resolver #1 sin.		

SysIo.ADC1	8	0..16383	R
	System analog input channel #1. Used for resolver #1 cos.		
SysIo.ADC2	9	0..16383	R
	System analog input channel #2. Normally used for resolver #2 sin.		
SysIo.ADC3	10	0..16383	R
	System analog input channel #2. Normally used for resolver #2 cos.		
SysIo.ADC4	11	0..16383	R
	User analog input ANA.In1 raw value.		
SysIo.ADC5	12	0..16383	R
	User analog input ANA.IN2 raw value.		
SysIo.ADC6	13	0..16383	R
	System Filtered shunt regulator (Dynamic Brake).		
SysIo.ADC7	14	0..16383	R
	DC-bus level, raw value. Only valid for 4-10A.		
SysIo.PWM0	15	0..255	RW
	<p>This PWM output is used to set the resolver excitation amplitude. Note. The content of the SysIo.PWM0 register affects both resolver #1 and #2; since both resolvers are driven from the same excitation circuitry.</p> <p>NOTE! Not fully backward compatible from Ver.4.0. or later.</p>		
SysIo.PWM1	16	0..255	RW
	NIU		
SysIo.PWM2	17	0..255	RW
	NIU		
SysIo.Sync	18	0..1	RW
	<p>Adjustment range: 950..1050 Hz Time step: ± 400 ns.</p> <p>Note. Normally the nominal servo cycle is 1000Hz, when the master is another DMC². Using the servo cycle synchronization mechanism with other master devices running at other frequencies then remember that all groups with members where the servo cycle time is used, like speed and acceleration, will be inaccurate. In other words, speed and such are calculated assuming a fixed 1000 Hz servo cycle rate. Enables servo cycle synchronization for slave if 1. Master should use 0 here.</p>		

SysIo, System I/O

Writing here affects bit 4 of `SysIo.POut` to set the direction of the external RS422 transceiver properly. Writing to `SysIo.POut` does not affect the `SysIo.Sync` setting! Getting into synchronization to the master after enabling slave synchronization may take several seconds. DMC² units that share resolvers must be synchronized and remain synchronized while the shared resolvers are used. A resolver is considered shared if a DMC² unit that is not driving the excitation for said resolver reads it. Synchronization must also take effect before the resolver `RD1.SHAdj` and `RD2.SHAdj` parameters are calibrated. If a slave wants to desynchronize, it must first set: `SysIo.Sync = 0` To become master, and then immediately turn OFF the synchronization-line driver by clear bit 4 in `SysIo.POut = 0`. (Write `BCLR SysIo.POut, 16`) to prevent jamming the synchronization-line for the other units.

<code>SysIo.MemStat</code>	19	0..65536	RW
	Status of system memory. Support two blocks of EEPROM (32 bit values.)		
<code>Bit0 (1)=1</code>	EEPROM activity completed. (EELoad/EEStore done.)		
<code>Bit1 (2)=1</code>	Checksum error in firmware PROM.		
<code>Bit2 (4)=1</code>	Block0 (EEPROM.0 – EEPROM.63) Checksum error in configuration EEPROM variables. (EEPROM.0 – EEPROM.28) Set by the EELoad instruction. Must be cleared by the PL program as the EELoad instruction does not clear this bit.		
<code>Bit3 (8)=1</code>	Block0 (EEPROM.0 – EEPROM.63) Checksum error in user EEPROM variables. (EEPROM.30 – EEPROM.62) Set by the EELoad instruction. Must be cleared by the PL program as the EELoad instruction does not clear this bit.		
<code>Bit4 (16)</code>	EEPROM size, 0 = EEPROM size is 1024 bit (93C46) 1 = EEPROM size is 4096 bit (93C66) If the EEPROM size is 1024bit then only the lower 64 EEPROM registers will be saved.		
<code>Bit5 (32)=1</code>	Block1 (EEPROM.64 – EEPROM.126) Checksum error in configuration EEPROM variables. (EEPROM.64 – EEPROM.92) See Bit2.		
<code>Bit6 (64)=1</code>	Block1 (EEPROM.64 – EEPROM.126) Checksum error in user EEPROM variables. (EEPROM.94 – EEPROM.126) See Bit3		
<code>Bit7 (128)</code>	1 = EEPROM size is 16384 bit (93C86)		
<code>Bit8 (256)</code>	Reserved		
<code>Bit9 (512)</code>	Reserved.		

Bit10 (1024)	Reserved.		
Bit11 (2048)	Error in loading FPGA circuitry. Will indicate if FPGA is present and can be used to select configurations.		
SysIo.RB	20	25..400	RW
	NIU		
SysIo.ChkTMR	21	0..255	RW
	Determines the check-rate for auxiliary regulator code. Such as the position error interrupt generator, motor thermal models, resolver monitorin The default value is 20 ms.		
SysIo.RevNo	22	0..32767	R
	Firmware revision number= (MajorVersion * 100 + MinorVersion). This must match the <code>EEProm.0</code> value, if the startup options set in the EEPROM group are to be used by the system. If <code>SysIo.RevNo <= EEPROM.0</code> , the startup setting are not used		
SysIo.SyncMode	23	0..255	RW
	Only default mode is available in DMC ² =0 Synchronization of DMC ² s. (Default).		
SysIo.ADC8	24		R
	SinCos encoder (sin)		
SysIo.ADC9	25		R
	SinCos encoder (cos)		
SysIo.ADC10	26		R
	Strain gauge #1,		
SysIo.ADC11	27		R
	Not used		
SysIo.SysTime	28	0..4096	R
	Indicate system time usage in 100ns.The actual system load can be observed. It shows the amount of time spent by the firmware to do all calculatiobns before any PL2 sw is executed.		
SysIo.FBCTime	29	0..4096	R
	Indicate field bus communication time usage in 100ns		
SysIo.ServoRate	30	950..1055	R
	The servo cycle frequency in Hz. Nominal rate is 1000 Hz.		
SysIo.SynchRate	31	950..1055	R
	The synchronization signal frequency in Hz. Shows the signal rate from the synchronization master when <code>SysIo.Sync</code> is 1.		

SysIo, System I/O

	For a master, when <code>SysIo.Sync</code> is 0, then this member is 0.			
<code>SysIo.PowerStage</code>	32	0..7	R	
	<p>Indicate the power stage type. Old power stages types will return 0.</p> <p>0 = No such information available.</p> <p>1 = DMC50412, 12Amp peak.</p> <p>2 = DMC30515, 15Amp peak.</p> <p>3 = DMC50720, 20Amp peak.</p> <p>4 = DMC31025, 25Amp peak. (NYI)</p> <p>5 = DMC51540, 40Amp peak. (NYI)</p> <p>6 = DMC32050, 50Amp peak. (NYI)</p> <p>7 = DMC53080 or DMC34080, 80Amp peak. (NYI)</p>			
<code>SysIo.DCBUS</code>	33	0..1000	R	
	<p>Indicate the DC-bus voltage in Volts.</p> <p>Note. Only newer power parts support this, 4-10 A.</p>			
<code>SysIo.Compatible</code>	34	0..1	RW	0
	Change the standard functionality of system functions. b0 (1) = Get and Set instruction, added functionality			
<code>Sysio.SyncStat</code>	35	0..31	R	
	<p>Status on external synch, only valid if <code>Sysio.Sync = 1</code>.</p> <p>Bit 2,bit 1 and bit0 holds a counter for incoming synch pulses. Normally these 3 bits shows the combined value of 6 when synchronization is stable if the external signal disappears, the value becomes 0 after 6 mS. In this case of missing synch signal the variable <code>Sysio.SyncRate</code> also becomes 0. Bit 3, (8), indicates if the incoming synch signal is higher than 1050 Hz (outside locking range). Bit4, (16), indicates if the incoming synch signal is lower than 950 Hz (outside locking range). Checking <code>Sysio.SyncStat</code> for these values may do a monitoring of the external synch.</p> <p>Ex: If <code>Sysio.SyncStat = 6</code> then SyncOK</p>			
<code>Sysio.EditNo</code>	36	0..99	R	
	Edit level of the firmware. If <code>EditNo</code> is nonzero the firmware is a Beta release. Together with <code>Sysio.RevNo</code> the complete revision number can be formed.			

INT, INTERRUPT CONTROL

Group	Group No.	Description
Int	10	Interrupt handling.

GENERAL

The interrupt system in the DMC² application software recognizes two different types of events, input related events and system related events. The interrupts are controlled with a bit mask for enabling (or inhibiting) each event and an interrupt pending register that indicates events waiting to be recognized.

FUNCTION

<code>Int.Mask</code>	Int.Mask enables the input related events. Bit 0 controls the event for digital input 1, bit 1.digital input 2 and so on.
<code>Int.SysMask</code>	Int.SysMask enables the system related events. Bit 0 controls the cyclical timer event and so on, according to the Vector group
<code>Int.Level</code>	For input events, the logical level for the interrupts can be set individually for each input in the Int.Level register.
<code>Int.PosErr</code>	The Int.PosErr parameter sets the "window" used for checking of position error. 0 means no checking.
<code>Int.TempMask</code>	The Int.TempMask register holds bit information on which sources should cause the TempErr interrupt. Initialised from EEProm.5 at startup
<code>Int.BleedLim</code>	The limit for bleeder supervision is set in the Int.BleedLim parameter.

RELATED ITEMS

<code>SysIo.ChkTmr</code>	Rate for system monitoring of PosError, Bleederload, Overtemperature and Resolver errors.
<code>Vector.Dil..Di10</code>	Vectors for input related events.
<code>Vector.Fault</code>	Vectors for input related events.
<code>Vector.Enable</code>	Vectors for input related events.
<code>Vector.Ptc</code>	Vectors for input related events.
<code>Vector.ResolveErr</code>	System interrupts vector for resolver monitoring.
<code>Vector.BleedErr</code>	System interrupts vector for bleeder monitoring.
<code>Vector.TempErr</code>	System interrupts vector for temperature monitoring.
<code>Vector.SysErr</code>	System interrupts vector for interpreter monitoring.
<code>Vector.CtrlC</code>	System interrupt vector for "control C" handling.
<code>Vector.PosErr</code>	System interrupt vector for positions monitoring.

Int, Interrupt control

`Vector.CapInt` System interrupts vector for captures event.
`Vector.CycInt` System interrupts vector for cyclical event.
`Ireturn in 8` Return statement.

EXAMPLE USAGE

```

      .
      Vector.DI4 , @InDi4Event
      clr Int.Pend                               ;remove pending events
      clr Int.SysPend                           ;remove pending events
      Int.Level , 8                             ;positive edge for Di4
      Int.Mask , 8                             ;enable Di4 event

;   Input event
InDi4Event

      .
      ireturn in 8                               ;return and enable interrupt again
  
```

GROUP MEMBERS (GROUP 10)

Group.Member	Member No.	Range	Ability	Default
<code>Int.Pend</code>	0	0..65535	RW	
	Pending interrupts from inputs.			
<code>Bit0 (1) = 1</code>	Pending interrupt from <code>In.Di1</code> .			
<code>Bit1 (2) = 1</code>	Pending interrupt from <code>In.Di2</code> .			
<code>Bit2 (4) = 1</code>	Pending interrupt from <code>In.Di3</code> .			
<code>Bit3 (8) = 1</code>	Pending interrupt from <code>In.Di4</code> .			
<code>Bit4 (16) = 1</code>	Pending interrupt from <code>In.Di5</code> .			
<code>Bit5 (32) = 1</code>	Pending interrupt from <code>In.Di6</code> .			
<code>Bit6 (64) = 1</code>	Pending interrupt from <code>In.Di7</code> .			
<code>Bit7 (128) = 1</code>	Pending interrupt from <code>In.Di8</code> .			
<code>Bit8 (256) = 1</code>	Pending interrupt from <code>In.Di9</code> .			
<code>Bit9 (512) = 1</code>	Pending interrupt from <code>In.Di10</code> .			
<code>Bit10 (1024) = 1</code>	Pending interrupt from <code>In.Enable</code> .			
<code>Bit11 (2048) = 1</code>	Pending interrupt from <code>In.PTC</code> .			
<code>Bit12 (4096) = 1</code>	Pending interrupt from <code>In.Fault</code> .			
<code>Int.Mask</code>	1	0..65535	RW	
	Enables interrupts from inputs.			
<code>Bit0 (1) = 1</code>	Enables interrupt from <code>In.Di1</code> .			
<code>Bit1 (2) = 1</code>	Enables interrupt from <code>In.Di2</code> .			

Bit2 (4) = 1	Enables interrupt from <code>In.Di3</code> .		
Bit3 (8) = 1	Enables interrupt from <code>In.Di4</code> .		
Bit4 (16) = 1	Enables interrupt from <code>In.Di5</code> .		
Bit5 (32) = 1	Enables interrupt from <code>In.Di6</code> .		
Bit6 (64) = 1	Enables interrupt from <code>In.Di7</code> .		
Bit7 (128) = 1	Enables interrupt from <code>In.Di8</code> .		
Bit8 (256) = 1	Enables interrupt from <code>In.Di9</code> .		
Bit9 (512) = 1	Enables interrupt from <code>In.Di10</code> .		
Bit10 (1024) = 1	Enables interrupt from <code>In.Enable</code> .		
Bit11 (2048) = 1	Enables interrupt from <code>In.PTC</code> .		
Bit12 (4096) = 1	Enables interrupt from <code>In.Fault</code> .		
<code>Int.Level</code>	2	0..65535	RW
	Sets active edge for input interrupts. (Level sampled after possible changes caused by the <code>In.Level</code> variable.)		
Bit0 (1) = 1	<code>In.Di1</code> interrupt active high else active low.		
Bit1 (2) = 1	<code>In.Di2</code> interrupt active high else active low.		
Bit2 (4) = 1	<code>In.Di3</code> interrupt active high else active low.		
Bit3 (8) = 1	<code>In.Di4</code> interrupt active high else active low.		
Bit4 (16) = 1	<code>In.Di5</code> interrupt active high else active low.		
Bit5 (32) = 1	<code>In.Di6</code> interrupt active high else active low.		
Bit6 (64) = 1	<code>In.Di7</code> interrupt active high else active low.		
Bit7 (128) = 1	<code>In.Di8</code> interrupt active high else active low.		
Bit8 (256) = 1	<code>In.Di9</code> interrupt active high else active low.		
Bit9 (512) = 1	<code>In.Di10</code> interrupt active high else active low.		
Bit10 (1024) = 1	<code>In.Enable</code> interrupt active high else active low.		
Bit11 (2048) = 1	<code>In.PTC</code> interrupt active high else active low.		
Bit12 (4096) = 1	<code>In.Fault</code> interrupt active high else active low.		
<code>Int.SysPend</code>	3	0..65535	RW
	Pending system interrupts.		
Bit0 (1) = 1	Pending <code>CyclInt</code> interrupt.		
Bit1 (2) = 1	Pending <code>CapInt</code> interrupt.		
Bit2 (4) = 1	Pending <code>PosErr</code> interrupt.		
Bit3 (8) = 1	Pending <code>Ctrl C</code> interrupt.		
Bit4 (16) = 1	Pending <code>Syserr</code> interrupt.		
Bit5 (32) = 1	Pending <code>Temperr</code> interrupt.		

Int, Interrupt control

Bit6 (64) = 1	Pending Bleederr interrupt.		
Bit7 (128) = 1	Pending Resloverr interrupt.		
Bit8 (256) = 1	Pending Cascade1 interupt.		
Bit9 (512) = 1	Pending Cascade2 interupt		
Bit10 (1024) = 1	Pending Cascade3 interupt		
Bit11	Reserved		
Bit12 (4096) = 1	Pending AnyBus-S interupt		
Bit13	Reserved		
Bit14	Reserved		
Bit15	Reserved		
Int.SysMask	4	0..65535	RW
	Enables system interrupts.		
Bit0 (1) = 1	Enable Cyclnt interrupt.		
Bit1 (2) = 1	Enable Caplnt interrupt.		
Bit2 (4) = 1	Enable PosErr interrupt.		
Bit3 (8) = 1	Enable Ctrl C interrupt.		
Bit4 (16) = 1	Enable Syserr interrupt.		
Bit5 (32) = 1	Enable Temperr interrupt.		
Bit6 (64) = 1	Enable Bleederr interrupt.		
Bit7 (128) = 1	Enable Resloverr interrupt.		
Bit8 (256) =1	Enable Cascade1 interrupt.		
Bit9 (512) = 1	Enable Cascade2 interrupt		
Bit10 1024) = 1	Enable Cascade3 interrupt		
Bit11	Reserved		
Bit12 (4096) = 1	Enable AnyBus-S interrupt		
Bit13	Reserved		
Bit14	Reserved		
Bit15	Reserved		
Int.PosErr	5	32 bit	RW 65536
	Setting this to zero will prevent detection and generation of position error interrupt.		
Int.TempMask	6	0..28	RW
	To prevent thermal overload of the system when there is no PL-program present (or running), there is a possibility within the firmware to get thermal protection. Internal temperature too high event Bit 4 (bitvalue = 16), should be set if automatic trip on Drive temp high is wanted.		

	<p>Termistor input in Motor connector event Bit 3 (bitvalue =8), should be set if automatic trip on Motor temp high is wanted.</p> <p><code>In.PTC</code> eventBit 2 (bitvalue = 4), should be set if automatic trip on <code>In.PTC</code> "low" is wanted.</p> <p>If no PL program is running when an event occurs, the power stage will be shut off on the conditions specified in Bit2, 3 and 4.</p> <p>If a program is running and <code>Vector.TempErr</code> = 0 (default value), the power stage will be shut of on the conditions specified in Bit2, 3 and 4.</p> <p>If <code>Vector.TempErr</code> <> 0, the program must handle this event by using the system interrupt on this vector, (enable by setting bit 5 in <code>Int.SysMask</code>). These conditions are checked with the rate set in <code>SysIo.ChkTmr</code>.</p> <p><code>Int.TempMask</code> is set to a default value defined in <code>EEprom.5</code>, bit 4, 3 and 2 at startup, If <code>EEprom.0</code> matches the revision number of the firmware.</p>		
<code>Bit2 (4) = 1</code>	Enable <code>In.PTC</code> interrupt.		
<code>Bit3 (8) = 1</code>	Enable interrupt on motor over temperature.		
<code>Bit4 (16) = 1</code>	Enable interrupt on drive over temperature.		
<code>Int.BleedLim</code>	7	0 8191	RW
	<p>Limit for bleeder monitoring. The default value for this 0. The function is disabled.</p> <p>If set > 0 the function monitors the bleeder load, <code>SysIo.ADC6</code>, with the rate set in <code>SysIo.ChkTmr</code>. A value of 400 allows a load equal to 0.5 seconds of continuous bleeding of the DC-bus. A value of 8000 allows a load equal to 10 seconds of continuous bleeding of the DC-bus.</p> <p>If the program (interpreter) is stopped, the power stage will be shut of when the bleeder load reaches this limit. If the program (interpreter) is running and the <code>Vector.BLeedErr</code> = 0 (the default value), the power stage will be shut of when the bleeder load reaches this limit. If the <code>Vector.BLeedErr</code> is <> 0 then the program is supposed to handle the condition by using the system interrupt on this vector, (enabled by setting bit 6 in <code>Int.SysMask</code>).</p> <p>The shunt regulator resistors can withstand a defined amount of energy from the regenerative power stage. This amount can be translated to a time of constant operation. A value in this parameter of 400 is equal to 500 ms of continuous operation. The internal resistors in the DMC² can be used for 500 ms continuous shunting.</p>		
<code>Int.Mode</code>	8	0..1	RW
=0	Normal mode. This is the default at startup.		
=1	When an interrupt is detected the firmware will push the content of <code>INT.IntMask</code> and <code>INT.SysMask</code> on the stack and then clear them, (disabling all interrupts). On leaving an		

Int, Interrupt control

	interrupt, (using ireturn) the previously pushed values of <code>INT.IntMask</code> and <code>INT.SysMask</code> will be popped from the stack and then combined with the arguments given with the ireturn statement.			
<code>Int.Trap</code>	9	0..255	R	0
	<p>Last system errors that caused the system interrupt</p> <p>1 = Control-C character detected in terminal mode.</p> <p>2 = END instruction is executed.</p> <p>3 = STOP instruction is executed.</p> <p>4 = Illegal instruction.</p> <p>5 = Line not found, goto, gosub, idxgoto and idxgosub.</p> <p>6 = Invalid module (not supported).</p> <p>7 = Unknown group.</p> <p>8 = Unsupported baud rate given to comm.baud.</p> <p>9 = LAN, invalid message object.</p> <p>10 = LAN, zero pointer.</p> <p>11 = Obsolete instruction.</p> <p>12 = CSend, constant values no supported.</p> <p>13 = CSend, must be in computer mode.</p> <p>-1 = Other error.</p>			

IN, DIGITAL INPUTS

Group	Group No.	Description
In	11	Digital Inputs to the DMC controller.

GENERAL

The digital inputs are polled by the system at 1 ms interval. This means that a change of state on input must be present during two samples in order to be detected as an edge. The interrupt system operates on these polled events only. The capture event uses digital input Di1 or the Encoder reference input for a special function parallel to the normal input function.

FUNCTION

Each input, In.Di1 through In.Fault, can be observed as an individual bit register. All inputs can also be read in parallel in the In.pdi register where each bit corresponds to one input. Di1 is bit 0, Di2 is bit 1 and so on.

The inputs can be individually inverted by setting the corresponding bit in the In.Level register. This capability permits easy adaptation to transducers with negative logic since it does not affect the interrupt settings. The In.Level function should be thought of as, if an external inverter is connected to the input.

RELATED ITEMS

`Vector.Di1..` Vectors for input related events.

`Vector.Di10`

`Vector.Enable`

`Vector.PTC`

`Vector.Fault`

`Int.Mask` Bit mask for input interrupts, bit 0 (bit value 1) enables the event on .Di1.

`Int.Pend` Bit mask for pending events, bit 0 (bit value 1) indicates the .Di1 event.

`Ireturn in 1` Return statement.

EXAMPLE USAGE

```

        if In.Di2 = 1 then Labl1      ;conditional jump on input
        "Else Do this"
        goto Labl2

Labl:
        "Do this"
Labl2:
        . . .

```

In, Digital inputs

```

register InData                ;defines a register
.
InData = In.PDI AND 15        ;read Di4,Di3,Di2 and Di1 as a nibble
.

```

GROUP MEMBER (GROUP 11)

Group.Member	Member No.	Range	Ability	Default
In.Di1	0	0..1	R	
Represent a Boolean value of the digital input at terminal X7A:16.				
In.Di2	1	0..1	R	
Represent a Boolean value of the digital input at terminal X7A:15.				
In.Di3	2	0..1	R	
Represent a Boolean value of the digital input at terminal X7A:14.				
In.Di4	3	0..1	R	
Represent a Boolean value of the digital input at terminal X7A:13.				
In.Di5	4	0..1	R	
Represent a Boolean value of the digital input at terminal X7A:12.				
In.Di6	5	0..1	R	
Represent a Boolean value of the digital input at terminal X7A:11.				
In.Di7	6	0..1	R	
Represent a Boolean value of the digital input at terminal X7A:10.				
In.Di8	7	0..1	R	
Represent a Boolean value of the digital input at terminal X7A:9.				
In.Di9	8	0..1	R	
Represent a Boolean value of the digital input at terminal X7A:8.				
In.Di10	9	0..1	R	
Represent a Boolean value of the digital input at terminal X7A:7.				
In.Enable	10	0..1	R	
Represent a Boolean value of the digital input at terminal				

X7A:6.			
In.PTC	11	0..1	R
In.Fault	12	0..1	R
The <code>Out.Fault</code> signal is binary AND'd with all internal fault signals and then brought to the Ready relay output. The result of the binary AND can be read in <code>In.Fault</code>			
(Reserved)	13	0..1	R
NIU			
(Reserved)	14	0..1	R
NIU			
(Reserved)	15	0..1	R
NIU			
In.Level	16	0..65535	RW
Sets the active level for each bit of the I/O ports. This variable can be used to adjust the input logic when an inverting input buffer board occurs, etc. Normally is not changed. Bit = 0: Active level is High. Do not invert data from input (default). Bit = 1: Active level is Low. Invert data from input.			
Bit0 (1) = 0 Bit0 (1) = 1	Active level is High for DI1. Do not invert data from input. Active level is Low for DI1. Invert data from input.		
Bit1 (2) = 0	Active level is High for DI2. Do not invert data from input.		
Bit2 (4) = 0	Active level is High for DI3. Do not invert data from input.		
Bit1 (8) = 0	Active level is High for DI4. Do not invert data from input.		
Bit1 (16) = 0	Active level is High for DI5. Do not invert data from input.		
Bit1 (32) = 0	Active level is High for DI6. Do not invert data from input.		
Bit1 (64) = 0	Active level is High for DI7. Do not invert data from input.		
Bit1 (128) = 0	Active level is High for DI8. Do not invert data from input.		
Bit1 (256) = 0	Active level is High for DI9. Do not invert data from input.		
Bit1 (512) = 0	Active level is High for DI10. Do not invert data from input.		
In.PDI	17	0..4095	R
Parallel read of above I/O ports. Used to read more than one I/O at a time. Bit0 corresponds to Di1 Bit1 corresponds to Di2 etc. b13 (8192) - User output short circuit indication. This bit indicates that one or several digital outputs are shorted to ground. (Outputs are short circuit protected).			

Out, Digital outputs

OUT, DIGITAL OUTPUTS

Group	Group No.	Description
Out	12	Digital Outputs from the DMC controller.

GENERAL

The digital outputs are normally activated by the system whenever an instruction to do so is executed. This function can be modified such that the actual timing of output changes is synchronized to a precise time in the servo cycle. This provides the capability to synchronize outputs between several DMC²s.

FUNCTION

- `Out.Do1...Do6` Each output, Do1 through Fault, can be handled individually via bit registers. They can also be accessed in parallel with the Out.PDO register.
- `Out.JamRS` Can be handled individually via bit registers.
- `Out.Fault` Can be handled individually via bit registers.
- `Out.Level` The outputs can be individually inverted by setting the corresponding bit in the Out.Level register. This capability can be used for easy adaptation to external actuators such as magnetic valves or similar devices.
- `Out.SyncMask` The Out.SyncMask enables each output to be synchronized to the servo cycle.
- `Out.PDO` Output word

RELATED ITEMS

EXAMPLE USAGE

```
.
Out.Do1, 1           ;activate output1
.bclr Out.PDO,13    ;clr outputs Do4, Do3 and Do1
.
```

GROUP MEMBERS (GROUP 12)

Group.Member	Member No.	Range	Ability	Default
<code>Out.Do1</code>	0	0..1	RW	
Represent a Boolean value of the digital output at terminal X7B:34.				
<code>Out.Do2</code>	1	0..1	RW	
Represent a Boolean value of the digital output at terminal				

	X7B:33.		
<code>Out.Do3</code>	2	0..1	RW
	Represent a Boolean value of the digital output at terminal X7B:32.		
<code>Out.Do4</code>	3	0..1	RW
	Represent a Boolean value of the digital output at terminal X7B:31.		
<code>Out.Do5</code>	4	0..1	RW
	Represent a Boolean value of the digital output at terminal X7B:30.		
<code>Out.Do6</code>	5	0..1	RW
	Represent a Boolean value of the digital output at terminal X7B:29.		
<code>Out.JamRS</code>	6	0..1	RW
	Not used		
<code>Out.Fault</code>	7	0..1	W
	This signal is binary AND'd with all internal fault signals and then brought to the Ready relay output. Terminal X7B:35 and X7B:36. NOTE: The result of the binary AND can be read in <code>In.Fault</code> .		
<code>Out.Level</code>	8	0..255	RW
	Sets the active level of the user outputs. Bit 0 = 1 --> DO1 is active high, 0 --> Active low. Bit 1 = 1 --> DO2 is active high, etc.		
<code>Out.SyncMask</code>	9	0..255	RW
	Set the synchronization mask for the user outputs. Bit 0 = 1 --> DO1 is in synchronous mode. Bit 1 = 1 --> DO2 is in synchronous mode etc. If an output is in synchronous mode, the hardware is updated at a specific time in the servo-cycle. Further, this time is set so that if multiple synchronized DMC ² units are connected to this output, the signal on the same servo-cycle will be recognized. This can be used to simultaneously start a motion on many DMC ² s. For this function to work properly, it is essential that the input/output filter for the participating units have a combined delay of 0.1 ms or less.		
<code>Out.PDO</code>	10	0..255	RW
	Parallel write of user outputs. Can also be used to inspect the result of one or more writes to individual bits using <code>Out.DOx</code> . Bit0 <--> DO1 Bit1 <--> DO2, etc.		

VECTOR, INTERRUPT VECTORS

Group	Group No.	Description
Vector	13	Vector table for various interrupts. Holds the start address for an interrupt routine associated with the member. To enable an interrupt in this group, the appropriate bit must be set in <code>Int.SysMask</code> or <code>Int.Mask</code> .

GENERAL

The registers in the vector group are the pointers to all interrupt routines. The registers contain line numbers within the application program.

FUNCTION

When an event occurs, system or input related the interpreter would set the pending bit for that event. If the mask bit for the event is set, the program will start to execute on the line pointed out by the related vector provided that it is not zero.

i

The `Ireturn` statement causes program execution to resume at the instruction where normal program flow was interrupted. If a normal return statement is used the interrupt will be executed only once.

DO NOT USE NORMAL RETURN IF INT:MODE = 1.

Use `Ireturn` with argument 0.

RELATED ITEMS

<code>Int.SysMask</code>	Bit mask for system interrupts.
<code>Int.SysPend</code>	Bit mask for pending events.
<code>Int.Mask</code>	Bit mask for input interrupts; bit 0 (bit value 1) enables the event on Di1.
<code>Int.Pend</code>	Bit mask for pending events, bit 0 (bit value 1) indicates the Di1 event.
<code>Ireturn sys 1</code>	Return statement for system events.
<code>Ireturn in 1</code>	Return statement for input events.

EXAMPLE USAGE

```

Vector.CapInt , @Caplabl ;set the vector to the line number of Caplabl
Vector.DI7 , @Inp7 ;set the vector to the line number of Inp7

Caplabl:
    Ireturn sys 2

Inp7:
    Ireturn in 64

```

GROUP MEMBERS (GROUP 13)

Group.Member	Member No.	Range	Ability	Default
<code>Vector.CycInt</code>	0	0..32767	RW	
	Each time the cyclic timer timesout, the routine pointed to by <code>Vector.CycInt</code> is executed. To enable this interrupt: OR <code>Int.SysMask, 1</code> .			
<code>Vector.CapInt</code>	1	0.. 32767	RW	0
	To enable this interrupt: OR <code>Int.SysMask, 2</code> .			
<code>Vector.PosErr</code>	2	0.. 8192	RW	
	The position error is bigger than <code>Int.poserr</code> or if enabled , the serial communication with the Endat sensor has timed out. (see <code>Xendat .mode</code> for more info) To enable this interrupt: OR <code>Int.SysMask, 4</code> .			
<code>Vector.CtrlC</code>	3	0..32767	RW	
	To enable this interrupt: OR <code>Int.SysMask, 8</code> .			
<code>Vector.SysErr</code>	4	0..32767	RW	
	To enable this interrupt: OR <code>Int.SysMask, 16</code> .			
<code>Vector.TempErr</code>	5	0..32767	RW	
	Temperature is high in motor or drive. To enable this interrupt: OR <code>Int.SysMask, 32</code> .			
<code>Vector.BLeedErr</code>	6	0..32767	RW	
	Bleeder load exceeds <code>Int.BLeedLim</code> . To enable this interrupt: OR <code>Int.SysMask, 64</code> .			
<code>Vector.ResolvErr</code>	7	0..32767	RW	
	Resolver amplitudes out of limits. To enable this interrupt: OR <code>Int.SysMask, 128</code> .			
<code>Vector.DI1</code>	8	0..32767	RW	
	Defines interrupt service routines for respective input. The input with higher number has higher priority. The first instruction of the interrupt routine is always executed when the interrupt occurs. By setting the <code>Int.Mask</code> , lower priority interrupts can be masked.			
<code>Vector.DI2</code>	9	0..32767	RW	
<code>Vector.DI3</code>	10	0..32767	RW	
<code>Vector.DI4</code>	11	0..32767	RW	

Vector, Interrupt vectors

<code>Vector.DI5</code>	12	0..32767	RW
<code>Vector.DI6</code>	13	0..32767	RW
<code>Vector.DI7</code>	14	0..32767	RW
<code>Vector.DI8</code>	15	0..32767	RW
<code>Vector.DI9</code>	16	0..32767	RW
<code>Vector.DI10</code>	17	0..32767	RW
<code>Vector.Enable</code>	18	0..32767	RW
<code>Vector.PTC</code>	19	0..32767	RW
<code>Vector.Fault</code>	20	0..32767	RW
(Reserved)	21	0..32767	RW
(Reserved)	22	0..32767	RW
(Reserved)	23	0..32767	RW
<code>Vector.Cascade1</code>	24	0..32767	RW
	<p>Cascaded interrupt. To enable this interrupt: OR <code>Int.SysMask, 256</code>. A cascaded interrupt increases the available interrupt sources, and needs a group handler that is specific for a particular group implementation. Example: <pre>Vector.Cascade1 = LAN1.Handler; Use the LAN1's handler or Int.SysMask, 256 ; Enable the cascade1 Vector or LAN1.Mask, xxxxx ; Enable LAN1's interrupts.</pre></p>		
<code>Vector.Cascade2</code>	25	0..32767	RW
	Added cascaded irq.		

	<p>Cascaded interrupt. To enable this interrupt: OR <code>Int.SysMask, 512</code>.</p> <p>A cascaded interrupt increases the available interrupts sources, and needs a group handler that is specific for a particular group implementation.</p> <p>Example:</p> <pre>Vector.Cascade1 = LAN1.Handler; Use the LAN1's handler or Int.SysMask, 512 ; Enable the cascade1 Vector or LAN1.Mask, xxxx ; Enable LAN1's interrupts.</pre>		
<code>Vector.Cascade3</code>	26	0..32767	RW
	<p>Added cascaded irq.</p> <p>Cascaded interrupt. To enable this interrupt: OR <code>Int.SysMask, 1024</code>.</p> <p>A cascaded interrupt increases the available interrupt sources, and needs a group handler that is specific for a particular group implementation.</p> <p>Example:</p> <pre>Vector.Cascade1 = LAN1.Handler; Use the LAN1's handler or Int.SysMask, 1024 ; Enable the cascade1 Vector or LAN1.Mask, xxxx ; Enable LAN1's interrupts.</pre>		
<code>Vector.IDO</code>	27	0..32767	RW
Not supported	CANopen, extensions interrupt. To enable this interrupt: OR <code>Int.SysMask, 2048</code> .		
<code>Vector.SANYBUS</code>	28	0..32767	RW
	Anybus-S option interrupt. To enable this interrupt: OR <code>Int.SysMask, 4096</code> .		
<code>Vector.29-31</code>	29..31	0..32767	RW
	NIU		

CAPTURE, Capture exact time of external events

CAPTURE, CAPTURE EXACT TIME OF EXTERNAL EVENTS

Group	Group No.	Description
Capture	14	<p>The Capture group is used to capture a precise time for an external event. For the DMC² the capture signal is parallel to the In.DI1 input pin.</p> <p>It is possible to reroute the capture function to use the Encoder reference input, by setting capture.mode = 8. However, since the response time of this signal is typically in the microsecond region, it is not certain that an event seen on the capture input is seen on the In.DI1.</p> <p>The Capture function is <u>not</u> affected by the In.Level parameter.</p>

GENERAL

The capture function is intended for fast detection of an external event. The time, speed and position are sampled and saved for later use when this event occurs. [Figure 7](#). The capture uses a real HW interrupt within the processor. This can be used for "flying calibration of movements" to increase the accuracy.

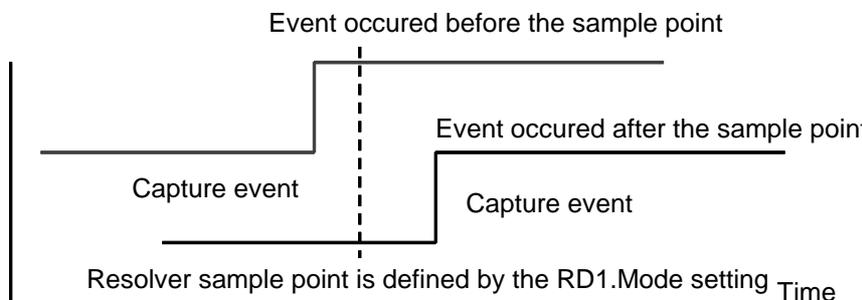


Figure 7. Timing chart for Capture event.

COMPATIBILITY DMC1 TO DMC²

The captured Pg values, Pg.APos and Pg.ASpeed are now sampled when the capture event is handled. The old DMC used a one servo cycle newer value. This may affect applications that used Capture.APos or Capture.ASpeed. The Capture.Enable mode bit 1, 'Auto apply SetTorque and SetPhDelay' for rapidly stopping the motor, is implemented but does not respond quickly.

FUNCTION

The capture uses the digital input 1 (or the Encoder reference input) as trigger source for the event. The electrical time constant for this input is much shorter when used by this function, than it is for the normal In.Di1 function. This means that the noise immunity is lower for this function.

<code>Capture.Mode</code>	The capture function can be set up to occur on positive, negative or both edges of the input signal. This setting is made in the Capture.Mode register and is not affected by the in.level bits.
<code>Capture.Time</code>	When the Capture.Enable register has been set, the first condition met will cause the HW interrupt to sample the absolute time and store this in Capture.Time.
<code>Capture.FTime</code>	The Capture.FTime register will hold the time within the cycle for the interrupt in microseconds.
<code>Capture.Pos</code> <code>Capture.RPos</code>	The Capture.Pos and Capture.RPos will be extrapolated with that time from the real sampled values of RD1.Pos and RD1.RPos.
<code>Capture.Speed</code>	The Capture.Speed will be the sampled value of RD1.Speed. When this interrupt is serviced, the Capture.Enable will be cleared.

If the Vector.CapInt is defined and the Int.SysMask bit 1 is set, the program will start to execute at the line defined.

HARDWARE CHANGE

The old DMC used only IN.DI1 as the high-speed input. In theDMC² a Encoder reference input can be used by setting capture.mode bit3.

1 Connector X6B has the balanced input at X6B:8 and X6B:21. THIS IS A BALANCED 5 V INPUT.

The DMC² does not support the prescaler on the capture input that the DMC1 has.

RELATED ITEMS

<code>Vector.CapInt</code>	System interrupts vector for captures event.
<code>Int.SysMask</code>	Bit mask for system interrupts.
<code>Int.SysPend</code>	Bit mask for pending events.

EXAMPLE USAGE

```

Vector.CapInt , @Caplab1           ;set up the pointer
Capture.Mode,1                     ;positive edge
Capture.Enable , 1                 ;activate
or Int.SysMask , 2                 ;enable the PL2 interrupt
wait Capture.Enable = 0           ;wait for capture
:
.register Target
Caplab1:

```

CAPTURE, Capture exact time of external events

```

Target = Capture.RPos + 25000 ;corrected target
pos abs Target                ;move to captured position+25000
wait PG.Rdy                    ;
return                          ;
                                ;one shot wanted

```

GROUP MEMBERS (GROUP 14)

Group.Member	Member No.	Range	Ability	Default
Capture.Mode	0	0..3	RW	
Bit0, Bit1	Capture mode setting. DMC1 had mode 0 with a prescaler on the capture input (divide by 8). THIS MODE IS NOT SUPPORTED IN DMC ² . Capture mode AND 1 = Capture positive edge. Capture mode AND 2 = Capture positive edge. Capture mode AND 3 = Capture every edge.			
Bit0(1)	Capture positive edges.			
Bit1(2)	Capture negative edges.			
Bit2 (4)	Added mode bit2, this bit indicate how the Capture.FTime is to be displayed.			
Bit2 (4) = 0	Standard display as old DMC, in microseconds. Capture.FTime is displayed as the fractional part of Tmr.Abs.			
Bit2 (4) = 1	Display Capture.FTime as a signed difference between the event and the RD sample point, with 100ns resolution. When the value is negative the event actually occurred in the previous servo cycle. This value is better suited for your own interpolations written in PL-code.			
i	All Tmr.Abs and position interpolation is done as in standard display mode. The display mode can be modified and the Capture.FTime member reread.			
Bit 3	This bit selects the input that triggers a capture.			
Bit 3 = 0 (0)	Select In.DI1 to trigger a capture. Default at startup.			
Bit 3 = 1 (8)	Select ENC_REF to trigger a capture. (X6B connector)			
Capture.Time	1	-2 ³¹ ..2 ³¹ -1	R	
The captured Tmr.Abs time is adjusted depending on whether the event occurred before or after the resolver sampling point (Rd1).				
Capture.FTime	2	0..FS-1	R	
The captured Tmr.Abs time is adjusted depending on whether the event occurred before or after the resolver sampling point (Rd1).				
Capture.Pos	3	-2 ³¹ ..2 ³¹ -1	R	

CAPTURE, Capture exact time of external events

			The <code>RD1.Pos</code> at the time of the event. The <code>RD1.Pos</code> is sampled with the FS (currently 1000 Hz) frequency, and the resulting position is then interpolated from that position and speed. Captured and interpolated <code>RD1.Pos</code> .
<code>Capture.RPos</code>	4	$-2^{31}..2^{31}-1$	R
			Same as <code>Capture.Pos</code> , but affected by <code>Pg.PosOffs</code>
<code>Capture.Enable</code>	5	0..3	RW
			Control and indicate operation of the CAPTURE mechanism. <code>Capture.Enable</code> is cleared when the event occurs and therefore, must be re-enabled every time a new capture is desired.
<code>Bit0 (1) = 1</code>			Enable this function and PL-code captures interrupt.
<code>Bit0 (1) = 1</code> <code>Bit1 (2) = 1</code>			The mode for rapidly stopping the motor.
<code>Capture.Speed</code>	6	$-32767000 .. 32767000$	R
			The current value of <code>RD1.Speed</code> when the event occurred. Captured <code>RD1.Speed</code> .
<code>Capture.SetTorque</code>	7	$-32767 .. 32767$	RW
			<code>Capture.SetTorque</code> and <code>Capture.SetPHDelay</code> values are used during/after the HSI capture interrupt is executed. The HSI capture interrupt can be used if a minimum delay start or stop of the motor is desired. The interrupt does the following:
			<pre> OR Reg.Mode, 128 ; Turn off regulator output. LET Reg.Torque = Capture.SetTorque ; Set user specified torque LET Motor.PhDelay = Capture.SetPhDelay ; Set optimum commutation for fast ACC/DECEL. ; The user Capture interrupt routine is then called to do the remaining control. </pre>
<code>Capture.SetPHDelay</code>	8	$-32767 * FS .. 32767 * FS$	R
			See <code>Capture.SetTorque</code> for description.
<code>Capture.GPos</code>	9	$-2^{31}..2^{31}-1$	R
			Captured the <code>Gear.Pos</code> interpolated to the time of the event. The <code>Gear.Pos</code> is sampled with the FS (currently 1000 Hz) frequency, and the resulting position is then interpolated from that position and speed.
<code>Capture.GSpeed</code>	10	$-32767 * FS .. 32767 * FS$	R
			Captured the <code>Gear.Speed</code> to the time of the event.
<code>Capture.APos</code>	11	$-2^{31}..2^{31}-1$	R
			Captured the <code>Pg.APos</code> at the time of the event. See "Compatibility issue".
<code>Capture.ASpeed</code>	12	$-32767 * FS .. 32767 * FS$	R

CAPTURE, Capture exact time of external events

			Captured the <code>Pg.ASpeed</code> at the time of the event. See "Compatibility issue".
<code>Capture.CamCurrLine</code>	13	0..65535	R
			Captured value of <code>Gear.CamCurLine</code> .at the time of th event
<code>Capture.CamScale</code>	14	$-2^{31}..2^{31}-1$	R
			Captured value of <code>Gear.CamScale</code> .
<code>Capture.CamPos</code>	15	$-2^{31}..2^{31}-1$	R
			Captured value of <code>Gear.CamPos</code> . Un-interpolated.
<code>Capture.CamSpeed</code>	16	$-32767 * FS .. 32767 * FS$	R
			Captured value of <code>Gear.CamSpeed</code> .
<code>Capture.PosErr</code>	17	$-2^{31}..2^{31}-1$	R
			Captured value of <code>Reg.PosErr</code> , Indicate how close we follow PG.APos. When comparing the Capture.Pos (RD.Pos) and the Capture.APos (PG.APos) the Capture.PosErr indicate the position error at the time of the event. All verification uses Capture.FTime difference display, see Capture.Mode, and constant speed
			Verification of Capture.Pos interpolation. $Capture.FTime = \frac{Capture.PosErr + Capture.Pos - Capture.APos}{Capture.Speed * 10^7}$
			Verification of Capture.GPos interpolation (When running gear box). $Capture.Pos = Capture.GPos + Capture.APos - Capture.PosErr$ When the Gear group is used for CAM generation the content of Capture.GPos is the offset into the cam table. $Offset = \frac{Capture.GPos}{1024} - Capture.CamCurline$
			Note. Capture.Gpos is interpolated.
			Verification of Capture.CamPos. $Capture.PosErr + Capture.Pos - Capture.APos - Capture.CAmPos = 0$ Currently the Capture.CamPos is a raw value, the value found at the servo cycle closest to the event.
<code>Capture.Pin</code>	18	0..1	R
			This signal is not captured, it indicates the state of the capture input source.
<code>Capture.RD2Pos</code>	19	0..1	R
			Captured and interpolated value of <code>_RD2.Pos_</code> .
<code>Capture.RD2RPos</code>	20	$-2^{31}..2^{31}-1$	R
			Captured and interpolated value of <code>RD2.RPos</code> .

CAPTURE, Capture exact time of external events

<code>Capture.RD2Spd</code>	21	$-2^{31}..2^{31}-1$	R
	Captured value of <code>RD2.Speed</code>		
<code>Capture.RD2Time</code>	22	$-2^{31}..2^{31}-1$	R
	The captured <code>Tmr.Abs</code> time is adjusted depending on whether the event occurred before or after the resolver sampling point (Rd2).		
<code>Capture.RD2FTime</code>	23	$-2^{31}..2^{31}-1$	R
	The captured <code>Tmr.Abs</code> time is adjusted depending on whether the event occurred before or after the resolver sampling point (Rd2).		
<code>Capture.SAPos</code>	24	$-2^{31}..2^{31}-1$	R
	Captured value of <code>Gear.SlaveAPos</code>		
<code>Capture.SSpd</code>	25	$-2^{31}..2^{31}-1$	R
	Captured value of <code>Gear.SlaveSpeed</code>		
<code>Capture.SFTime</code>	26	$-2^{31}..2^{31}-1$	R
	Captured value of update point of virtual master in the gearbox function. Can be used for interpolating the captured value of <code>Gear.SlaveApos</code> .		

ANA, ANALOG I/O

Group	Group No.	Description
Ana	15	<p>The Ana group is the analog user input/output interface.</p> <p>Inputs: The standard user analog inputs have 14 bits resolution and approximate 12 bits of linearity. The result from the inputs can be both scaled and offset-adjusted to be directly usable as set-speed, set-torque, etc. in the user program.</p> <p>The RANGE parameter sets the numerical range for the input and the OFFS parameter sets the offset. The result is calculated as: $(RAW-RESULT / 16384) * RANGE + OFFSET$ For example: RANGE = 10000 OFFSET = -2000. This results in a range from -2000 to 8000.</p> <p>Outputs: The RAW range for the analog outputs is -32768 to 32767 with 12 bits of resolution.</p> <p>The RAW_OUTPUT is calculated as: $RAW_OUTPUT := Ana.Out * 2^{Ana.OutSF} + Ana.OutOffs$</p> <p>There is no range-check to clamp the RAW_OUTPUT to -32768 .. 32767. In case of overflow, the analog voltage will wrap from full positive to full negative, and vice versa.</p>

GENERAL

The analog interface in the DMC² consists of two 14 bit inputs and two 12 bits outputs. These analog channels have no secondary functions and can be used freely by the application program. The inputs can be scaled to any 32 bit range and the outputs can be scaled with a shift factor (binary scale). The outputs are limited to 1mA of current (10 kOhm) load by the means of a 1 kOhm output impedance. The outputs are short circuit proof.

The scale of the outputs are set so that full logical output (16bit) gives 10.6 V. This means that a +/- 10 V output scale is mapped to +/- 31000 in the output value.

FUNCTION

`Ana.In1..Ana.In2`
`Ana.In1Range`
`Ana.In1Offs`
`Ana.ConnTMR.`

The Ana.In1Range and Ana.In1Offs scale the value in Ana.In1 (Ana.In2 is handled in the same way). The values of the inputs can be read from the program whenever needed and treated as any values or the inputs can be "connected" to registers with an update rate set with Ana.ConnTMR.

`Ana.Out1..Ana.Out2`

The output is scaled with Ana.Out1Sf for Ana.Out1 and Ana.Out2Sf for Ana.Out2.

`Ana.Out1Sign..`
`Ana.Out2Sign`

The sign of the outputs can be set in the Ana.Out1Sign and Ana.Out2Sign registers

and Ana.Out2Sign registers.

Ana.ConnTMR

The outputs can be written from the program or connected to any register with the rate set in Ana.ConnTMR.

RELATED ITEMS

Connect Statement for defining cyclical updates of analog functions.

EXAMPLE USAGE

```
.
Ana.In1Range , 100 000 ;+/- 10 V is mapped to +/- 50 000
Ana.In1Offs , -50000 ;
Ana.In2Range , -16000 ;+/- 10 V is mapped to +/- 8000
Ana.In2Offs , 8000 ;
.
Ana.Out1SF , -2
Ana.Out1Offs,0 ;+/- 131068 is mapped to +/- 10 V
Ana.Out1Sign , 1 ;
.
Ana.Out2SF , 2 ;
Ana.Out2Offs,0
Ana.Out2Sign , -1 ;+/- 8191 is mapped to +/- 10 V
.
connect Ana.In1 to Pg.Speed ;
Ana.ConnTMR,1 ;
. ;connect analog input 1 to the speed command
;1 ms update rate
```

GROUP MEMBERS (GROUP 15)

Group.Member	Member No.	Range	Ability	Default
Ana.In1	0	$-2^{31}..2^{31}-1$	R	
Scaled and offset-adjusted value for user analog input 1. (For raw data, see SysIoO.ADC4) Terminal: X7A:3,4				
Ana.In2	1	$-2^{31}..2^{31}-1$	R	
Scaled and offset-adjusted value for user analog input 2. (For raw data, see SysIo.ADC5) Terminal: X7A:1,2				
Ana.In1Range	2	$-2^{31}..2^{31}-1$	RW	
Range for Ana.In1				
Ana.In2Range	3	$-2^{31}..2^{31}-1$	RW	
Range for Ana.In2				
Ana.In1Offs	4	$-2^{31}..2^{31}-1$	RW	
Offset for Ana.In1				
Ana.In2Offs	5	$-2^{31}..2^{31}-1$	RW	

Ana, Analog I/O

	Offset for <code>Ana.In2</code>		
<code>Ana.Out1</code>	6	$-2^{31}..2^{31}-1$	RW
	Scaled and offset-adjusted Analog output value. Terminal: X7A:22		
<code>Ana.Out2</code>	7	$-2^{31}..2^{31}-1$	RW
	Scaled and offset-adjusted analog output value. Terminal: X7A:23		
<code>Ana.Out1SF</code>	8	-31..31	RW
	Scale factor for <code>Ana.Out1</code>		
<code>Ana.Out2SF</code>	9	-31..31	RW
	Scale factor for <code>Ana.Out2</code>		
<code>Ana.Out1Offs</code>	10	-32768..32767	RW
	Offset for <code>Ana.Out1</code>		
<code>Ana.Out2Offs</code>	11	-32768..32767	RW
	Offset for <code>Ana.Out2</code>		
<code>Ana.ConnTMR</code>	12	1..255,0	RW
	Interval in Servo Cycles between updates setup by the CONNECT command. The default value is 0 (connection disabled). If no connection is used set this value to 0 since this generates less software overhead than connecting all the channels to 0. When looking at registers with the CONNECT command, there is approximately a 1 ms delay between the internal value and the Analog Out. When looking at extended registers such as <code>RD1.Speed</code> , there is approximately a 1ms delay from the value to the Analog Out.		
<code>Ana.Out1Sign</code>	13	-32768..32767	RW
	Sign of analog output #1. A negative value will electrically invert the output.		
<code>Ana.Out2Sign</code>	14	-32768..32767	RW
	Sign of analog output #2. A negative value will electrically invert the output.		

EEPROM

Group	Group No.	Description
EEprom	16	Non-volatile parameter storage.

GENERAL

The EEprom group has 128 registers. Each EEprom register is 32 bit wide. The registers EEprom.29, 63, 93, 127 are reserved for storage of partial checksums of the data

NOTE: At startup, the DMC² unit will always print the sign on text (and eventual error messages) at 9600 baud. The EEprom variables are then used to setup other addresses, etc. This allows a standard terminal to be used to verify that the system is working properly. It also allows firmware errors to be reported prior to a system crash.

The initial printout (done without enabling the interrupt system) may look something like this:

```
1: DMC Ver 5.00<1>
2: ?FirmWare checksum error: SUM = xxxx<2>
3: <3>
```

If line 2: is printed, the System Firmware is damaged or the CPU board may be malfunctioning.

If the cursor stops at <2> instead of going to <3>, the system locked (crashed) when the interrupt system was enabled.

If only line 1 is printed and the cursor stops at position <1>, the firmware checksum was ok but the system crashed when the interrupt system was initiated. If the cursor stops at <3>, i.e. at the beginning of a new line, the problem may be one of the following:

A baud rate different from 9600 is setup by code in the EEprom.

A baud rate different from 9600 is setup by auto starting PL-code.

A comm-mode indicating computer mode, deselected unit, or xoff status bit set is setup in the EEprom.

FUNCTION

In the DMC² there is a 16 kbit serial EEprom which may be used for nonvolatile storage of application parameters. Some system parameters are taken from the EEprom at power up. Only 4 kbit are used for the 128 EEProm registers. The rest is used by the ParArea resource.

At power up (reset), or when forced by the EEload statement, the contents of the EEprom device is read by the processor and put in RAM memory (in the EEprom register group). The consistency is checked using the checksum calculated at the previous storage.

The EEstore statement forces a store procedure of the data in the EEprom group to the EEprom device.

It is important to understand that when an EEProm register is read or written, it is affecting only the image held in RAM memory. To actually store the contents, an EEstore instruction has to be executed.

EEprom

RELATED ITEMS

- `Eeload` Statement for unpacking the contents of the EEprom.
- `Eestore` Statement for storing the data into the EEprom.
- `SysIo.MemStat` Status of memories.
- `Pararea.load...` Parameter storage in Eeprom with windowing.

EXAMPLE USAGE

```

.  

EEprom.36 , 278 ;write a value to the group member (in RAM)  

Eestore ;force a storage to the serial EEprom device  

wait SysIo.MemStat and 1 ;wait for this to complete (takes a while)  

.  

Eeload  

wait SysIo.MemStat and 1 ;force a unpacking of the serial EEprom device  

;wait for this to complete (takes a while)  

.
    
```

GROUP MEMBERS (GROUP 16)

Group.Member	Member No.	Range	Ability	Default
<code>EEprom.0</code>	0	$-2^{31}..2^{31}-1$	RW	
This entry must be equal to the SysIo.RevNo for the other EEProm settings to take effect. This is to prevent an uninitialized or uninstalled EEProm to set abnormal parameters. In addition, if the firmware is updated, existing parameters may have modified values while new parameters may be present.				
<code>EEprom.1-3</code>	1..3	$-2^{31}..2^{31}-1$	RW	
Reserved for system use.				
<code>EEprom.4</code>	4	75..19200	RW	
This member is used to specify serial baud rate. If this member is zero the baud rate will be taken from EEprom.6.				
The value is set directly in baud rate values.				
<code>EEprom.5</code>	5	0..31	RW	
The default value for <code>Int.TempMask</code> . Normal value is 24.				
<code>EEprom.6</code>	6	$-2^{31}..2^{31}-1$	RW	
System usage. Bit mask for various startup settings.To change the node number without affecting any other bits do:				
<pre> >BCLR EEprom.6, 15 ; Reset all bits, mask out >OR EEprom.6, 3 ; Set node number to 3 >EESTore ; Store it for later use </pre>				
NOTE: To do this in a program, wait for the EESTore instruction to complete (see EESTore instruction).				

Bit(0..3) 1+2+4+8=15	Daisy Chain Node number assigned to this unit.		
Bit(4..6) 16+32+64=112	The default baud rate if Eeprom.4 = 0. Currently 0 = 75, 1 = 150, 2 = 300, 3 = 600, 4 = 1200, 5 = 2400, 6 = 4800, 7 = 9600 baud.		
Bit(7) = 128			
Bit(8..15)	Reserved for future expansion.		
EEprom.7	7	$-2^{31}..2^{31}-1$	RW
	System usage.		
Bit(0..7)	The Comm.Mode setting.		
Bit(8..15)	The terminal line length. (for LIST etc).		
EEprom.8	8	$-2^{31}..2^{31}-1$	RW
	LAN1, Lan1 communication frequency. See LAN1.Init		
EEprom.9-12	9..12	$-2^{31}..2^{31}-1$	RW
	Reserved. CAN1 ACC protocol CAN TxID and RxID.		
EEprom.13	13	$-2^{31}..2^{31}-1$	RW
	LAN2, Lan2 communication frequency. See LAN2.Init		
EEprom.14-17	14..17	$-2^{31}..2^{31}-1$	RW
	Reserved CAN2.		
EEprom.18-28	18..28	$-2^{31}..2^{31}-1$	RW
	Reserved for system use; is defined later.		
EEprom.29	29	$-2^{31}..2^{31}-1$	RW
	Checksum of entries 0..28 (LSW of 2's complement of sum).		
EEprom.30-62	30..62	$-2^{31}..2^{31}-1$	RW
	Free for USER parameters.		
EEprom.63	63	$-2^{31}..2^{31}-1$	RW
	Checksum of entries 30..62 (LSW of 2's complement of sum).		
EEprom.64-92	64..92	$-2^{31}..2^{31}-1$	RW
	Reserved for system use; is defined later.		
EEprom.93	93	$-2^{31}..2^{31}-1$	RW
	Checksum of entries 64..92 (LSW of 2's complement of sum).		
EEprom.94-126	94..126	$-2^{31}..2^{31}-1$	RW
	Free for USER parameters.		
EEprom.127	127	$-2^{31}..2^{31}-1$	RW
	Checksum of entries 94..126 (LSW of 2's complement of sum).		

COMM, SERIAL COMMUNICATION

Group	Group No.	Description
Comm	17	Serial Communication Interface.

GENERAL

The DMC² has a serial interface for both programming purposes and for application use. The function of the communication is defined in the Comm group.

FUNCTION

- `Comm.Node` Several DMC²'s can be connected in a "Daisy Chain" configuration. Each unit must then have a unique address set in the Comm.Node register. The number will be shown in the prompter when using terminal mode.
- `Comm.Baud` The speed of the communication can be set in Comm.Baud. From Version 4.x support 19200 baud
- `Comm.TLines` The number of lines shown in the terminal window is set in Comm.TLines where 0 means no limit at all.
- `Comm.Mode` The DMC² can operate in different communication modes. Standard terminal mode is set with Comm.Mode = 0.

RELATED ITEMS

- `EEprom.0` Version number of system software.
- `EEprom.6` Baud rate and node number.
Forced run + baud + Comm.Node
(baud = BaudMask: 0=75, 1=150, 2=300 aso)
- `EEprom.7` Number of lines and communication mode.
256*Comm.Tlines +Comm.Mode

EXAMPLE USAGE

- `EEprom.6` Forced run
Baud rate =9600
Node = 8
`EEprom.6` = 128 + 16*5+8
- `EEprom.7` Comm.Tlines = 27
Mode = 3
`EEprom.7` = 256*27 + 3

GROUP MEMBERS (GROUP 17)

Group.Member	Member No.	Range	Ability	Default
Comm.Mode	0		RW	
	The <code>Comm.Mode</code> at startup is determined by the LSB of EEPROM.7. Useful bits to set are 0, 1, 4 and 7. Set Bit 7 for all nodes in a daisy chain, except for one node. This setting determines the default node to communicate with at startup (i.e. before the first select sequence is sent.)			
Bit0 (1)	Output from PL program disabled. All DISP, error messages, etc. are thrown away.			
Bit1 (2)	Computer mode enabled on serial channel and terminal mode disabled.			
Bit2 (4)	Echo off. Special communication mode is enabled. The unit will not echo characters when typed and the DISP statement will only print the values, not the register name.			
Bit3 (8)	Enables special behaviour of the computer mode protocol for downloading application SW. Normally the contents of the application memory is cleared by the download process prior to the actual transfer of new PL2 SW. This bit overrides that behaviour so that partial download of PL2 SW is possible. Typically this can be used to speed up the download of say a Camtable or another part of an application. The part of SW handled this way must be set to a specific part of the memory by using the .ORG directive.			
Bit4 (16)	Xon/Xoff protocol is enabled. Xoff (Control-S) stops output; Xon (Control-Q) resumes output. If the node is deselected, the output is stopped when Xon/Xoff is enabled; it is thrown away otherwise.			
Bit5 (32)	Output is disabled due to a received Xoff.			
Bit6 (64)	A start of Select is detected (Control-Z) and the controller is now waiting for the next character in the sequence (the node number).			
Bit7 (128)	Node is deselected. No text output can be made at the moment. If an output is attempted (DISP, etc.), the result is thrown away if Bit 4 = 0. If Bit 4 = 1, the result stays in the output buffer. If the output buffer becomes full, the executing DISP statement waits until the node is selected again.			
Comm.Baud	1	75..19200	RW	
	Supported baud rates are: 75,150, 300, 600, 1200, 2400, 4800, 9600 and 19200. (Only from Version 4.x. The baud rate value is now checked for validity before accepted.)			
Comm.Node	2	0..15	RW	
	The node number to use when using the daisy-chain option. The initial value used after power up is stored in EEPROM.6.Setting			

Comm, Serial communication

	<p><code>Comm.Node</code> takes effect immediately. If an <code>EEStore</code> is done after, the new setting is valid after a power up; otherwise, the old value remains valid.</p> <p>To change the node number immediately to 3:</p> <pre>>Comm.Node, 3</pre> <p>To change the node number used when the DMC² is powered up to 3:</p> <pre>>BCLR EEprom.6, 15 >OR EEprom.6, 3 >EEStore</pre>		
<code>Comm.TLines</code>	3	0..256	RW
	<p>The number of lines displayed sequentially when many lines will follow. After <code>Comm.TLines</code> is displayed, the system prompts the user to continue. The user has the option of pressing the space bar for another screen page, or the ENTER key for one line at the time. Pressing Q terminates the listing.</p>		
<code>Comm.Rdy</code>	4	0..1	R
	<p>Indicates if the serial communication is busy handling a <code>Csend</code> instruction and if a computer mode session is active.</p> <p>Bit 0: 1: <code>Csend</code> is Idle 0: <code>Csend</code> is Busy</p> <p>Bit 1: 1: Computer mode Idle 0: Computer mode Busy (start of record received)</p> <p>Typical use:</p> <pre>Wait Comm.Rdy and 2 ;wait for idle condition Csend R7 ;send the message Wait Comm.Rdy and 1 ;wait for message to be sent</pre>		

RD1CORR, POSITION CORRECTOR

Group	Group No.	Description
RD1Corr	18	Position correction values for RD1. This table can be used to increase the accuracy of RD1. This is useful if RD1 is a low accuracy type, such as a HALL sensor, etc. Turning ON bit 2 of RD1.Mode enables use of this table.
To swap to alternate table, use the following:		
<pre> ; Assume R19 is 1 for forward and 0 for reverse direction. 10 IF RD1Corr.34 = R19 then 12 ; Check if right direction. 11 RD1Corr.34 = R19 ; Nope, swap table 12 ... </pre>		

GENERAL

The DMC² is intended for use with resolvers as feedback devices. If other devices are to be used, or the resolver is of low quality, it may be necessary to correct the values from the feedback device before the angle is calculated.

The RD1Corr group is a table of such correction values.



**The correction table does not operate in high-resolution mode.
Version 4.x**

FUNCTION

RD1Corr.

The DMC² can measure and create the values in the correction table using a special mode of the RD1.Interface. The motor must run with high and constant speed for several seconds to be able to find the values.

Bit 3 in RD1.Mode activates the measurement procedure and bit 2 activates the correction.

RELATED ITEMS

RD1.Mode Resolver 1 mode register.

EXAMPLE USAGE

```

RD1.Mode,13 ;1+4+8 use and create the correction table
Reg.SetTorq , 500 ;run with torque control
.
clr Tmr.T0
wait Tmr.T0 > 20000
. ;run a while
RD1.Mode,5
. ;1+4 use the correction table

```

RD1Corr, Position corrector

GROUP MEMBERS (GROUP 18)

Group.Member	Member No.	Range	Ability	Default
RD1Corr.0-15	0-15	-32767..32767	RW	
	This is the correction factor in increments for each 1/16 of a resolver turn.			
RD1Corr.16-31	16-31	-32767..32767	RW	
	This is currently only used as storage for alternate table.			
RD1Corr.32	32	-32767..32767	RW	
	This is the Integrated speed error. This is only valid when bit 3 of RD1.Mode is set.			
RD1Corr.33	33	-32767..32767	RW	
	This is the Integrates position error. This is only valid when bit 3 of RD1.Mode is set.			
RD1Corr.34	34	-32767..32767	RW	
	When entry 34 is written, the entries 0..15 are swapped with entries 16..31. This is useful since the optimal table may differ between varying speeds or rotation direction.			

OPTAD, ANALOG TO DIGITAL CONVERTER.

Group	Group No.	Description
OptAD	22	Optional analog to digital conversion.

GENERAL

The DMC equipped with this option has an analog input named M1. This channel has an individual gain setting. The gain setting is controlled with the OptAD.GainM1 member. The input is intended for a strain gauges transducer i.e. a load cell or a torque transducer. The connector also incorporates a balanced supply voltage of +/- 5 VDC.

Channel	Measuring range	Purpose	
M1	±50 mVDC	Strain gauge transducer	Extern measurement bridge is required.

CONVERSION RESOLUTION

The AD resolution is 14 bit, and the result is sign extended to utilize the 32 bit PL register set.

CALIBRATION

The M1 channel has two resistances, 390 kΩ and 47kΩ, that can be used to calibrate an externally provided measurement bridge. When the value in the table is written to OptAD.Cal then the selected calibration resistance will be connected in parallel with one of the four legs in the Measurement Bridge.

Value	Channel	Resulting calibration resistance.
0	none	Disconnect all calibration resistances.
8	M1	390 kΩ
9	M1	47 kΩ

EXAMPLE USAGE

```

; On startup.
OptAD.Cal = 0      ; disconnect all calibration resistances.
.. <other initializations>
                ; Start calibration
OptAD.Cal = 8      ; M1, calibration resistance = 390 kOhm
R200 = OptAD.M2    ; Conversion is done within 200 us.
.. <measure and store values>
OptAD.Cal = 0      ; disconnect all calibration resistances.
<other code>

; Start measuring
R100 = OptAD.M1    ; and the result is stored in R100.
; The calibration value in R200 is application dependent and
; bridge dependent, and it can be used in several different
ways.
;
; 1. Value may indicate what type of measurement

```

OptAD, analog to digital converter.

```
; bridge that is connected.
; 2. Value may indicate if a measurement bridge is connected.
; 3. Value is a 32bit integer representation of a the
; bridge manufacturer's engineering unit calibration
value.
```



After power on of DMC². Set OptAD.Cal=0 before measuring.

AMPLIFIER GAIN SETTING

Setting a value in the OptAD.GainM1 member controls the gain of the input amplifier. The following values are possible:

Value	Resulting gain
0	internal_signal = 1 * external_signal
1	internal_signal = 1.5 * external_signal
2	internal_signal = 3 * external_signal
3	internal_signal = 6 * external_signal
4	internal_signal = 12 * external_signal
>4	internal_signal = 1 * external_signal

EXAMPLE USAGE

```
OptAD.GainM1 = 1 ; internal_signal = 1 * external_signal
```

GROUP MEMBERS (GROUP 22)

Group.Member	Member No.	Range	Ability	Default
OptAD.Mode	0	0..4	RW	
	Reserved			
OptAD.Cal	1	0 or 8..13	RW	
	Select calibration resistance. See Group description.			
OptAD.GainM1	2	0..4	RW	
	Amplifier gain for the M1 channel. See Group description			
OptAD.GainM2	3	0..4	RW	
	Not used in DMC2			
OptAD.4	4			
	NIU			
OptAD.5	5			
	NIU			
OptAD.6	6			

OptAD, analog to digital converter.

	NIU		
OptAD.7	7		
	NIU		
OptAD.8	8		
	NIU		
OptAD.M1	9	$-2^{31}-1..2^{31}-1$	R
	When this member is read, a conversion on channel M1 is started. Measured value input range ± 50 mVDC.		
OptAD.M2	10	$-2^{31}-1..2^{31}-1$	R
	Not used		
OptAD.11	11		
	NIU		
OptAD.12	12		
	NIU		
OptAD.13	13		
	NIU		
OptAD.14	14		
	NIU		
OptAD.15	15		
	NIU		
OptAD.OffsetM1	16	$-32767..32767$	
	Offset adjustment of M1 signal. This offset affects both OptAD.M1 and OptAD.sM1. Set the offset using: $OptAD.OffsetM1 = OptAD.OffsetM1 + OptAD.M1$		
OptAD.OffsetM2	17	$-32767..32767$	R
	NIU		
OptAD.ScaleM1	18	$-2^{31}-1..2^{31}-1$	RW
	Scale factor for the M1 signal. The scale factor is calculated as: $OptAD.ScaleM1 = DispValue \text{ DIVMUL } maxvalue$ Example: We have an OptAD.M1 range of ± 760 increments and we want to display that as 7.400 Nm. $OptAD.ScaleM1 = 7400 \text{ DIVMUL } 760$		
i	DIVMUL, In the divmul function the parameter 7400 must currently be ± 32767 only to avoid overflow. When the OptAD.sM1 is read we get values in the range ± 7400.		
OptAD.ScaleM2	19	$-2^{31}-1..2^{31}-1$	RW

OptAD, analog to digital converter.

	NIU		
OptAD.sM1	20	$-2^{31}-1..2^{31}-1$	R
	<p>Scaled and offset adjusted measured value. The calculation done by the system is:</p> $OptAD.sM1 = \frac{(OptAD.M1 - OptAD.Offset) * OptAD.ScaleM}{65536}$ <p>Note. The numerator product must be restricted to 48 bit. In case an overflow is detected then OptAD.OvfM1 is non-zero after OptAS.sM1 is read. To detect overflow a user should read OptAD.sM1 first then OptAD.OvfM1.</p>		
OptAD.sM2	21	$-2^{31}-1..2^{31}-1$	R
	NIU		
OptAD.OvfM1	22	0..1	R
	If non-zero indicates an overflow condition after OptAD.sM1 was read.		
OptAD.OvfM2	23	0..1	R
	NIU		

LAN1, LOCAL AREA NETWORK 1

Group	Group No.	Description
LAN1	28	Local area network group. LAN channel number 1. This group is the interface to the Local Area Network. The OSI reference model for LAN communication: Application layer Presentation layer Session layer Transport layer Network layer Datalink layer Physical layer

LAN1, INTERRUPT HANDLING

There are two types of interrupts that can be generated by the LAN1 group.

1. Net Error

When the LAN1 low level protocol (the CAN chip) detects an error it will go bus off. For this event the system will generate a PL interrupt and execute the code specified at `LAN1.ErrVector`.

This interrupt is always enabled but if a user does not specify any line where to execute PL code, leaving `LAN1.ErrVector = 0`, then the CAN communication line will only go bus off and no other action will be done by the firmware. A user can specify what action to take, when an error occurs in a PL written interrupt routine.
2. Message Object interrupt

A message object that has either been received or transmitted can generate an interrupt. The PL service routine is specified in the `MsgObjLan1.Vector` when the object is defined. The following must be done before any LAN1 related interrupts will be generated:

 1. A `MsgObjLan1.xx` must be initialized and mapped to a priority level. The `MsgObjLan1.Vector` member must have a valid PL code line # at the time when the `SetObjLan1` instruction is used to program the priority level.
 2. The corresponding `LAN1.Mask` bit for the level used must be set.
 3. The `Vector.Cascade1` member must be set to `LAN1.Handler`
 4. The `Int.SysMask` must be set to enable the `Vector.Cascade1`. Before enabling `Vector.Cascade1`, a user should clear all bits in `LAN1.Pend` and the bit for the `Vector.Cascade1` in `Int.SysPend`, to avoid generating interrupt on old events.

LAN1, Local area network 1

LAN1, DOUBLE BUFFERING

The firmware uses a double buffering method to communicate between the CAN lower protocol and the PL interpreter. The Read and Write instructions are used for manipulating the buffers.

LAN1, SPECIFIC INSTRUCTIONS

`SetObjLAN1` `<level>`

This instruction will map a previously defined `MsgObjLan1` to a `<level>`, priority level, in the CAN low protocol, thereby activating the content of the `MsgObjLAN1` (make it alive). Lower numbers yield higher priority. A `MsgObjLAN1` is mapped to a priority level. The number of usable priority levels depends on the setting of `LANx.lowprot` as:

Standard = 8 priority levels.

Extended = 15 priority levels where level 15 has special possibilities.

Note. Priority level 13, 14 and 15 are reserved for system usage and may not be available in future releases.

It is possible to remap an already activated object, just issue a new `SetObjLAN1` instruction with the same priority level.

Example:

```
<Activate LAN1 ..>
...
MsgObjLAN1.Id, 1234
MsgObjLAN1.Type = 1      ; receive
MsgObjLAN1.....
...
SetObjLAN1 1
```

The Message Object with the ID 1234 is made active and mapped to priority level 1.

For each message object received with ID 1234 the content will be stored in the buffer related to priority level 1.

To read the content into a register, the following code should be executed:

```
ReadLAN1 r45, 4, 1
```

This code is typically put into a interrupt service routine.

Here is how to deactivate a message object:

```
MsgObjLAN1.Type = 0
SetObjLAN1 1
```

A previous defined Message Object is deactivated.

`GetObjLAN1` `<level>`

Fill in the `MsgObjLAN1` with the message object at `<level>`.

`ReadLAN1` `<reg>`, Read `<len>` bytes and put in register `<reg>` from the
 `<len>`, buffer for message object at `<level>`.
 `<level>` Multiple reads after the initial first, can be done by

specifying - **<level>**.

An internal offset is maintained that allows a user to pack the data in the 8 available bytes in a message.

Example:

```
Read LAN1 R5, 4, 1
; Read 4 bytes of data into register R5
; from level #1
Read LAN1 R6, 1, -1
; Read the fifth byte into register R6
Read LAN1 R7, 2, -1
; Read two bytes beginning at the sixth
; byte into register R7
...
```

CAUTION:

Lan1 interrupt routines, automatically disables other Lan1 interrupts until the Ireturn statement is executed. To enable another Lan1 interrupt the corresponding bit in Int.sysmask has to be set.

- WriteLAN1** `<reg>`, `<len>`, `<level>` Write **<len>** bytes to the buffer for message object at **<level>** from register **<reg>**. Multiple writes, after the initial first, can be done by specifying - **<level>**. See **ReadLAN1**.
- SendObjLAN1** `<level>` Send the buffer content for the message object at **<level>** on to the CAN bus. The data size sent will be the length that was previously defined when the object was defined. If the len member of the **MsgObjLAN1** was zero when the **SetObjLAN1** instruction was executed then a message object with no data will be transmitted. The content of the data should be filled in using the **WriteLAN1** instruction prior to this instruction.
- IReturnCAS1** `<priority>` `-level>` A return from a user written PL interrupt service routine should end with this instruction. It will behave as the normal **IReturn**, but affect the individual message object interrupts. The **<priority-level>** value is binary added (OR) to **LAN1.Mask**. The **Int.SysMask** is automatically re-enabled.

LAN1, REMOTE FRAMES IN CAN

The CAN 'Remote Frame' concept is implemented in hardware by the low-level communication protocol. The name of this mechanism, 'Remote Frame', is unfortunate it would have been better with 'Respond Frame', because the receiver of a 'Remote Frame' shall respond with it's contents. The receiver here is the transmit descriptor that owns the requested data and the sender is a receive descriptor that wants this data.

To generate a 'Remote Frame' in PL a user can send, use the instruction **SendObjLAN1**, on a descriptor that was defined as a receive type. The low level communication protocol will in this case send a CAN 'Remote Frame' so that somewhere on the net a transmit object with the same ID will respond and send the content of its descriptor.

LAN1, Local area network 1

A user controls how to handle 'Remote Frame' responses on transmit descriptors. See `MsgObjLAN1.Status` and `MsgObjLAN1.Frame`.

LAN1, POWER UP

On power up the low-level protocol initializes itself and sets up CAN communications rate at the content of `EEprom.8`. A user can change the can frequency. See `LAN1.Ini`.

LAN1, HIGH LEVEL COMMUNICATION PROTOCOLS.

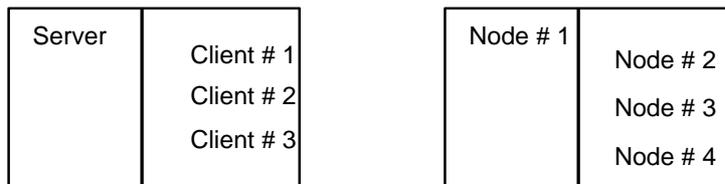
No higher-level communication protocols are implemented that require their own groups.

A higher protocol uses some user defined "live" message objects and the underlying protocol uses these to implement communication services, as:

- On-demand data, (server/client relation ship).
- Broadcast data.
- Sporadic data.
- Periodic data.
- Data synchronization.
- Large data messages, up/download of application code.
- User defined services. (User defined protocol).
- Multi data services, uses a communication area, when more that one parameter is required before a service can be performed.

LAN1 COMMUNICATION SCENARIOS.

Server uses periodic receive objects to trigger the slave owning the object to transmit its data. A slave transmits data on demand from the server.



All nodes use periodic receives objects to trigger the owner of the object to transmit its data. Any node can send or receive sporadic 'alert' message objects.

i

All message objects must have a unique object ID. Make sure that each and every node/server use unique transmits object IDs. In other words, there cannot be more than one unit that transmit an object with the same ID, at the same time. Sporadic message objects have the disadvantage that the user must be sure that these messages will not clogging the network. And the firmware cannot handle back-to-back messages. The time between two messages sent back-to-back

is approx. 50us when 1Mbit/sec CAN frequency is used. Conceptually, a particular node owns a message object. But any node can listen to this message object by defining a received message object with the same ID.

If multiple messages are sent on a channel without any confirmation the some data will be overwritten in the receiving ends.

There is no buffer for incoming messages so if a new message comes in (on the same channel) before the previous message was read, the old message is lost !!!!

GROUP MEMBERS (GROUP 28)

Group.Member	Member No.	Range	Ability	Default
LAN1.Mode	0	0..255	R(W)	
Bit0 = 0	Bit 0 is the only write able bit. All other bits are reserved and Read only.			
Bit0 = 1	Normal operation. This bit should be low for normal operation. When this bit is high the CAN controller is disconnected from the net (bus off). If this bit is set high then the CAN controller will be forced bus off. It will reset the internal error handling and start a recovery sequence where the CAN low level protocol expects to see an idle bus.			
LAN1.Status	1	0..255	RW	
Bit 0-2 (0..7)	The LAN1.Status indicate the operation of the low protocol (CAN) chip.			
Bit3(8)	Last Error code, can also be read in member LAN1.LastErr.			
Bit4(16)	Transmit message successfully.			
Bit5(32)	Receive message successfully.			
Bit6(64)	Reserved.			
Bit7(128)	Warning. There is an abnormal rate of occurrences of errors on the CAN bus.			
	Bus Off. There was an abnormal rate of occurrences of errors on the CAN bus. The unit is disconnected from the CAN bus. The only way to reset this situation is to force the LAN1.Mode bit0 high and then low (or cycle power).			

LAN1, Local area network 1

Bit8(256)	Valid only For LAN2.Mode Indicates when the LAN2 bus has external power connected.		
LAN1.Init	2	0..2 ²⁴ -1	RW
	Initialize the CAN frequency, etc. The default is 125 kbit/sec can frequency. 1000 kbit/sec = 41856 500 kbit/sec = 41857 250 kbit/sec = 41859 125 kbit/sec = 41863		

Bit No.								Description
31	30	29	28	27	26	25	24	
x	x	x	x	x	x	x	x	Bit 24..31 Not used

Bit No.								Description
23	22	21	20	19	18	17	16	Bus configuration register.
0	x	–	0	–	0	–	–	x = CoBy, Compare Bypass bit. 1 = input comparator is by passed and RX0 regarded as valid bus input. 0 = normal operation RX0 and RX1 are the inputs to the input comparator. (Should never be changed).
0	–	x	0	0	0	–	–	x = Pol, Polarity bit. 1 = If the input comparator is bypassed then a logical one is interpreted as dominant and a logical zero is recessive on the RX0 input. 0 = normal operation. If the input comparator is bypassed the logical one is interpreted as recessive and a logical zero is dominant on the RX0 input. (Should never be changed).
0	–	–	0	x	0	–	–	x = DcT1 bit, disconnect TX1 output. 1 = Disables TX1 output. 0 = normal operation, enables TX1 output.
0	–	–	0	–	0	x	–	x = DcR1 bit, disconnect RX1 input. 1 = RX1 is disabled and disconnected. 0 = normal operation.
0	–	–	0	–	0	–	x	x = DcR0 bit, disconnect RX0 input. 1 = RX0 is disabled and disconnected. 0 = normal operation.

Bit No.								Description
15	14	13	12	11	10	9	8	Bit Timing Register 2
x	–	–	–	–	–	–	–	x = Spl bit, Number of samples per bit Bit24, 1 = 1 spl, 1 = 3 spl.

-	x	x	x	-	-	-	-	x = TSEG2 1-7, time segment <u>after</u> the sampling point.
-	-	-	-	x	x	x	x	x = TSEG1 2-15, time segment <u>before</u> the sampling point.

Bit No.								Description
7	6	5	4	3	2	1	0	Bit Timing Register 1.
x	x	-	-	-	-	-	-	SJW 0-3 Sync. jump width.
-	-	x	x	x	x	x	x	BRP 0-63 Baud Rate prescaler.

Group.Member	Member No.	Range	Ability
LAN1.LowProt	3	0..2	R
	Low level communication protocol. This member represents the physical and data link layers of the OSI reference model. The CAN low level protocol features:		
	Bit wise Contention, mechanism to resolve collisions based on the priority of the message id number.		
	#	Standard Data and Remote frames, 11 bit id (CAN 2.0 part A).	
	#	Extended Data and Remote frames, 29 bit id (CAN 2.0 part B).	
	#	12 "Live" Message objects. (CAN 2.0 part B).	
	#	0 to 8 byte data per message.	
	#	Programmable bit rate (higher bit rate for shorter distances). (Note. bit rate is not the same as throughput).	
This member is also used to inform an application program about the presence of the LAN. 0 = Not available 1 = Standard, CAN specification 2.0 part A is supported. 2 = Standard and Extended, CAN specification 2.0 part B is supported. The Standard CAN spec is a subset of the Extended spec. The CAN protocol uses a multi-master (contention based) bus configuration for transfer of "communication objects" between nodes of the network. This multi-master node is also referred to as CSMA/CR or Carrier Sense, Multiple Access, with Collision Resolution. As defined in ISO/DIS 11898 Road vehicles - Interchange of digital information - Controller area network (CAN) for high-speed communication.			
LAN1.HighProt	4	0	R
	High level communication protocol. This member indicate, what protocol set are used for the layers that are not defined in LAN.LowProt.0 = No high level protocol implemented.		

LAN1, Local area network 1

LAN1.StdFilter	5	0..2047	RW	Message Acceptance Filter for Standard Frames. Allows the user to globally mask, or "don't care" any identifier bits in the incoming message object. Range, 11-bit message ID. Default is that all bits must match. NOTE. This is a global filter, it affects all descriptors, and use only if you are familiar with the arbitration method of CAN.
LAN1.ExtFilter	6	0..536870911	RW	Message Acceptance Filter for Extended Frames. Allows the user to globally mask, or "don't care" any identifier bits in the incoming message object. Range, 29-bit message ID. Default is, all bits must match. NOTE. This is a global filter, it affects all descriptors, and use only if you are familiar with the arbitration method of CAN.
LAN1.Mask	7	0..65535	RW	Specify the object(s) at respective level that can generate interrupt to the PL code interpreter. Also the error vector.
	0			No message object can generate interrupt.
	Bit0(1)			Message object at level 1 will generate interrupt
	Bit1(2)			Message object at level 2 will generate interrupt
	Bit2(4)			Message object at level 3 will generate interrupt
	Bit3(8)			Message object at level 4 will generate interrupt
	Bit4(16)			Message object at level 5 will generate interrupt
	Bit5(32)			Message object at level 6 will generate interrupt
	Bit6(64)			Message object at level 7 will generate interrupt
	Bit7(128)			Message object at level 8 will generate interrupt
	Bit8(256)			Message object at level 9 will generate interrupt
	Bit9(512)			Message object at level 10 will generate interrupt
	Bit10(1024)			Message object at level 11 will generate interrupt
	Bit11(2048)			Message object at level 12 will generate interrupt
	Bit12(4096)			Message object at level 13 will generate interrupt
	Bit13(8192)			Message object at level 14 will generate interrupt
	Bit14(16384)			Message object at level 15 will generate interrupt
	Bit15(32768)			CAN low level error will generate interrupt
	 The objects at level 13, 14 and 15 are reserved for system usage, they may not be available in future versions.			
LAN1.Pend	8	0..65535	RW	Indicate a pending interrupt at the respective level. See LAN1.Mask for bit description.

LAN1.ErrVector	9	1..Max. line of PL2 program lines	RW
Address of the PL-code line to execute when a comm error occurs.			
LAN1.LastErr	10	1..7	R
The last error reported by the low level CAN protocol. This member is most useful when the <code>ErrVector</code> is used.			
<p>0. No error</p> <p>1. Stuff error More than 5 equal bits in a sequence have occurred in a part of a received message where this is not allowed.</p> <p>2. Form Error The fixed format part of a received frame has the wrong format.</p> <p>3. Acknowledgment Error The message transmitted by this device was not acknowledged by another node.</p> <p>4. Bit 1 Error During the transmission of a message, the 82527 wanted to send a recessive level, (bit of logical value 1), but the monitored CAN bus value dominant.</p> <p>5. Bit 0 Error During the transmission of a message, the 82527 wanted to send a recessive level, (bit of logical value 0), but the monitored CAN bus value was recessive. During busoff recovery, this status is set each time a recessive bit is received.</p> <p>6. CRC Error The CRC checksum was incorrect in the message received. The CRC received for an incoming message does not match with the CRC value calculated by this device for the received data.</p> <p>7. Unused.</p>			
LAN1.Handler	11	1..xx	R
<p>An interrupt handler for cascading interrupts from the LAN1 group. See cascading interrupts, Vector and Int. This member is used when installing an interrupt cascade handler for the LAN1 group. The <code>Vector.Cascade</code> member must be initialized with this handler before interrupts can be generated from the LAN1 group. Example:</p> <pre> ; Setup the cascaded interrupt handler. Vector.Cascade1 = LAN1.Handler Or Int.SysMask, 256 ; LAN1 can generate cascaded system ; interrupts LAN1.ErrVector = @isrLANERR or LAN1.Mask, 32768 ; Allow errors to generate an interrupt </pre>			

LAN1, Local area network 1

LAN1.ErrStat	12	$-2^{31}..2^{31}-1$	RW
	Error statistic. Number of errors since power on or reset.		
LAN1.RxStat	13	$-2^{31}..2^{31}-1$	RW
	Receive statistic. Number of successfully received frames since power on or reset.		
LAN1.TxStat	14	$-2^{31}..2^{31}-1$	RW
	Transmit statistic. Number of successfully transmitted frames since power on or reset.		
LAN1.RemStat	15	$-2^{31}..2^{31}-1$	RW
	Indicates number of received 'remote frame' packets.		
LAN1.ICount	16	$-2^{31}..2^{31}-1$	RW
	Debug usage. Indicates number of total service interrupts.		
LAN1.VecNum	17	0..xx	R
	Indicates the cascaded level that the LAN1.Handler is connected to 1 means that cascade 1 is used 2 means that cascade 2 is used 4 means that cascade 3 is used.		
LAN1.Port	18	0..255	RW
	Not used		
LAN1.OvrWrite	19	0..65535	R
	Pending interrupt overwrite counter. Indicates the number of times that the LAN1.Pend bit was still active at the time when the same bit was to be set as a result of a new interrupt. In other words, the PL interrupt service routine was not performed before a new interrupt was generated. Note. There is no indication on what priority level this occurred at nor does it indicate if it was a receive, or transmit interrupt.		

MSGOBJLAN1, HELPER FOR LAN1

Group	Group No.	Description
MsgObjLAN1	29	<p>CAN message descriptor temporary storage. A message object descriptor must be initialized and then activated with the <code>SetObjLAN1</code> instruction.</p> <p>It is the responsibility of the user to fill in the <code>MsgObjLAN1</code> group prior the use of the instruction <code>SetObjLAN1</code>.</p> <p>The system fills in the <code>MsgObjLAN1</code> group when the <code>GetObjLAN1</code> instruction is used.</p> <p>Example:</p>
		<pre><Do LAN initialize..> MsgObjLAN1.Id = 2000 ; ID number for this message object MsgObjLAN1.Type = 1 ; Receive type MsgObjLAN1.Frame = 0; Standard frame MsgObjLAN1.DataLen = 2 ; Two bytes in this message MsgObjLAN1.Vector = @Isr2000 ; Interrupt service routine. SetObjLAN1 3 ; Activate message object <..></pre>

GROUP MEMBERS (GROUP 29)

Group.Member	Member No.	Range	Ability	Default
MsgObjLAN1.ID	0	11-bit and 29-bit ID	RW	
	<p>The message object ID.</p> <p>If the <code>MsgObjLAN1.Frame</code> type is 'Standard' then the id is an 11-bit number.</p> <p>If <code>MsgObjLAN1.Frame</code> type is 'Extended' the id is a 29-bit number.</p> <p>Note1. If used on an existing CAN net the user must avoid the use of any reserved ID used in other protocols.</p> <p>Note2. For 'standard' frames, ids above 2030 are reserved and should not be used.</p>			
MsgObjLAN1.Type	1	0..3	RW	
	<p>The object type.</p> <ol style="list-style-type: none"> 0. Invalidate, this can be used to deactivate an existing message object. 1. Receive 2. Transmit, with automatic response to a remote frame. 3. Transmit, respond to a remote frame, but the response should be written in PL code, this must be used together with an interrupt. 			

MsgObjLAN1, Helper for LAN1

MsgObjLAN1.Frame	2	0..1	RW	<p>Indicate the frame type that this message object should use.</p> <p>0 = Standard, 11-bit Ids. 1 = Extended, 29-bit Ids.</p> <p>Can only be used when LANx.LowProt equals 2 indicating that CAN specification 2.0 part B is supported.</p> <p>Note. A descriptor that is programmed to receive 'Extended' frames will not receive 'Standard' frames nor will a message programmed to receive 'Standard' frames receive 'Extended' frames. (It is possible, if all units on the net follow CAN specification 2.0, to mix 'Standard' and 'Extended' frames on the same net).</p>
MsgObjLAN1.DataLen	3	0..8	RW	<p>The data length for this message object. Normally it is most efficient to use 8 bytes of data. It is not necessary to read or write all bytes to an object before using it.</p>
MsgObjLAN1.Vector	4	1..Max. line of PL2 program lines	RW	<p>Action vector on a reception/transmission of the ID specified.</p> <p>Specify a line # where to execute code when a reception or a transmission of an object with the MsgObjLAN1.ID occurs.</p>
MsgObjLAN1.Status	5	0..3 (when read) 1..12 (when written)	RW	<p>This member has valid data after a GetObjLAN1 instruction has been executed.</p> <p>On read, indicates the status of the descriptor. On write, the value written is the <level> that is to transmit a frame as a response to a remote frame.</p> <p>0 = Undefined 1 = The descriptor has been Transmitted. 2 = A remote frame has been received with this ID. 3 = The descriptor has been Received.</p> <p>Usually used in a PL-code interrupt service routine for a particular message object.</p> <p>Example:</p> <pre> <Setup interrupt system> MsgObjLAN1.id = 400 MsgObjLAN1.type = 3 ; Tx, with semi automatic remote frame response MsgObjLAN1.frame = 0 MsgObjLAN1.DataLen = 4 MsgObjLAN1.Vector = @isrLan setObjLAN1 5 or lan1.mask, 16 ; Enable interrupt .. other initialization ;Message object 5 interrupt service routine isrLan: GetObjLAN1 5 ; Get descriptor content </pre>

```
if MsgObjLAN1.status <> 2 then IsrDone
This was a remote frame
;request for the data ;in this descriptor,
now ;we can update the data
write LAN1 R10, 1, 5
MsgObj.status = 5
IsrDone: ; we are done with it, it will now be sent.
IreturnCAS1 16
; Re-enable this interrupt:
```

LAN2, Local area network 2

LAN2, LOCAL AREA NETWORK 2

Group	Group No.	Description
LAN2	30	Local area network group. LAN channel number 1. This group is the interface to the Local Area Network. The OSI reference model for LAN communication: Application layer Presentation layer Session layer Transport layer Network layer Datalink layer Physical layer

GENERAL

The main difference between LAN2 and LAN1 is that LAN2 has an isolated interface and the connector configuration follows the CANOPEN standard.

LAN2 COMMANDS

LAN2 have the same commands as LAN1. See chapter LAN1.

MSGOBJLAN2, HELPER FOR LAN2

Group	Group No.	Description
MsgObjLAN2	31	<p>CAN message descriptor temporary storage. A message object descriptor must be initialized and then activated with the <code>SetObjLAN2</code> instruction.</p> <p>It is the responsibility of the user to fill in the <code>MsgObjLAN2</code> group prior the use of the instruction <code>SetObjLAN2</code>.</p> <p>The system fills in the <code>MsgObjLAN2</code> group when the <code>GetObjLAN2</code> instruction is used.</p> <p>Example:</p>
		<pre><Do LAN initialize..> MsgObjLAN2.Id = 2000 ; ID number for this message object MsgObjLAN2.Type = 1 ; Receive type MsgObjLAN2.Frame = 0; Standard frame MsgObjLAN2.DataLen = 2 ; Two bytes in this message MsgObjLAN2.Vector = @Isr2000 ; Interrupt service routine. SetObjLAN2 3 ; Activate message object <..></pre>

MSGOBJLAN2 COMMANDS

MsgObjLAN2 have the same commands as MsgObjLAN1. See chapter MsgObjLAN1.

MultDiv,

MULTDIV,

Group	Group No.	Description
MultDiv	49	The operation support full 64-bit precision in the multiplication and division. Also division by zero and overflow detection has been added. In case an overflow is detected result will be +-MAXINT. The division does automatic round off.

GENERAL

This group performs the following operation,

$$\frac{MultDiv.Num1 * MultDiv.Num2}{MultDiv.Denom} = MultDiv.Result, MultDiv.Reminder$$

FUNCTION

An Internal calculation method that uses 64-bit precision.

The calculations are made when a result variable is read.

Automatic roundoff is done by adding 50% of the Denominator to the Numerator before dividing.

RELATED ITEMS

MulDiv, DivMul instructions.

EXAMPLE USAGE

For all scaling purposes where 32-bit precision is inadequate.

GROUP MEMBERS (GROUP 49)

Group.Member	Member No.	Range	Ability	Default
MultDiv.Num1	0	$-2^{31}..2^{31}-1$	RW	
Numerator #1 is a 32bit signed value, the numerator is formed by the product: <code>MultDiv.Num1 * MultDiv.Num2</code> . The product has 64-bit precision but is not available to read.				
MultDiv.Num2	1	$-2^{31}..2^{31}-1$	RW	
Numerator #2 is a 32bit signed value, the numerator is formed by the product: <code>MultDiv.Num1 * MultDiv.Num2</code> . The product has 64-bit precision but is not available to read.				
MultDiv.Denom	2	$-2^{31}..2^{31}-1$	RW	

	Denominator is a 32bit signed value.		
<code>MultDiv.Quotient</code>	3	$-2^{31}..2^{31}-1$	R
	When this member is read then the quotient (integer part of the result) is calculated and the <code>MultDiv.Flags</code> are updated.		
<code>MultDiv.Reminder</code>	4	$-2^{31}..2^{31}-1$	R
	When this member is read then the remainder (fractional part of the result) is calculated and the <code>MultDiv.Flags</code> are updated.		
<code>MultDiv.Flags</code>	5	0..7	R
b0 (1)	Indicates a division by zero.		
b1 (2)	Indicates an overflow.		

FlashMem

FLASHMEM

Group	Group No.	Description
FlashMem	50	Flash memory control

GROUP MEMBERS (GROUP 50)

Group.Member	Member No.	Range	Ability	Default
FlashMem.Mode Bit0(1)	0	0..1	RW	0
When set the download of an application SW will be automatically followed by a FSTORE command.				
FlashMem.Status	1			
NIU				

ABIn

Group	Group No.	Description
ABIn	52	Input buffer to the Anybus-S modules

GENERAL

This is an array of 8bit bytes. The group contains the first 255 bytes of the IN area.

FUNCTION

RELATED ITEMS

ABOut

EXAMPLE USAGE

GROUP MEMBERS (GROUP 52)

Group.Member	Member No.	Range	Ability	Default
ABIn.	0-254	0..255	RW	0
Data received from DMC ² .				

ABOut

ABOUT

Group	Group No.	Description
ABOut	53	Output buffer from the Anybus-S module

GENERAL

This is an array of 8bit bytes. The group contains the first 255 bytes of the OUT area.

FUNCTION

When the Anybus-S module has received and processed a fieldbus message the data will be available in this output buffer.

RELATED ITEMS

ABIn

EXAMPLE USAGE

Display incoming data from the fieldbus

```
Disp ABOut.23 ;inspect byte 23 in the output buffer
```

GROUP MEMBERS (GROUP 53)

Group.Member	Member No.	Range	Ability	Default
ABOut.	0-254	0..255	RW	0
Data received from the Anybus-S module.				

DSTORE,

Group	Group No.	Description
DStore	54	Stores 32bit signed values. The max-length of DStore is 2047. This group enables an application programmer to implement a FIFO (First In First Out) or a LIFO (Last In First Out), data structure.

GENERAL

The Dstore mechanism allows the user to store 2047 samples of any internal 32bit variable in each of the two available buffers. By using the Connect mechanism to do so the Dstore can be seen as a two channel digital oscilloscope within the DMC².

The data stored can then be uploaded over the serial channel or connected to an analog output for analysis.

This group is not affected by a GDisp terminal mode command.

FUNCTION

When a sample of data is written to a Dstore buffer a pointer is incremented so that the next data can be written. When saturated the oldest data will be lost. The number of data (if less than 2047) can be seen in Dstore.Lengthx

RELATED ITEMS

Connect statement allows the user to automatically store to or extract data from the dstore buffers

EXAMPLE USAGE

To store values to DStore

```
DStore.In1,Rd1.Speed ; or any other valid data
```

```
Connect Dstore.In1 to reg.torque
```

NOTE. If you store faster than once per servo cycle then multiple entries with the same value will be found in DStore..

To 'play' the recorded data to an analog output port do:

```
Connect Ana.Out1 to DStore.Peek1
```

To remove a value from DStore do,

```
R100, DStore.Out1 ; FIFO structure (This is a queue)
R100, DStore.In1 ; LIFO structure (This is a stack)
```

To clear DStore from all values do,

```
Dstore.Length1, 0
```

DStore,

GROUP MEMBERS (GROUP 54)

Group.Member	Member No.	Range	Ability	Default
DStore.Mode	0	$-2^{31}..2^{31}-1$	RW	
	Not used			
DStore.In1	1	$-2^{31}..2^{31}-1$	RW	
	Input to DStore. On Write: The value is inserted at the top of the storage area, and the length will be incremented by one. On Read: Return the value at the top of the storage area. The length will be decrement by one.			
DStore.Out1	2	$-2^{31}..2^{31}-1$	RW	
	Output from Dstore. On Write: Not possible. On Read: Return the value at the bottom of the storage area, and decrement the length by one.			
DStore.Peek1	3	$-2^{31}..2^{31}-1$	RW	
	Peek into the storage area. On Write: Set offset where to peek. On Read: Return the value at the peek-index, and increment the peek-index by one. The peek-index is internally kept within the value of DStore.LengthX.			
DStore.Length1	4	0..2047	RW	
	The currently used length of DStore, (not the max. length). On Write: Reset current length to zero. (The value given is ignored). On Read: Return the currently used length of DStore. (Return number of entries in Dstore).			
	X = 1 or 2			

PARAREA,

Group	Group No.	Description
ParArea	55	Non-volatile parameter storage area.

GENERAL

This group uses a window technique to select a specific parameter set. The parameters can be configured to be 16 bit or 32-bit size.

FUNCTION

This group uses the same hardware resource as the EEprom group. Therefore a user must wait for a load/store action to finish the same way as for the EELoad/EEStore instructions, but use ParaArea.Mode Bit0 instead.

RELATED ITEMS

EXAMPLE USAGE

GROUP MEMBERS (GROUP 55)

Group.Member	Member No.	Range	Ability	Default
ParArea.Mode	0	0..7	RW	
	Operation mode and status information.			
	b0 (1)	Operation done. After a Load/Store operation a wait instruction should follow to make sure that the operation has finished before executing the next instruction. This is actually an image of the bit in the <code>SysIo.MemStat</code> indicating that the physical Eeprom operation has finished. In order to be future compatible it is strongly suggested that this bit is used instead of the <code>SysIo.MemStat</code> .		
	b1 (2)	Checksum error.		
	b2 (4) =0 =1	Parameter organization. A parameter is 32 bit. A parameter is 16 bit.		
ParArea.Load	1	32bit=0..31 16bit=0..63	RW	
	Load all Raw members from the window given. After a Load/Store operation a wait instruction should follow to make sure that the operation has finished before executing the next instruction. In case the checksum calculation fails then b1 in <code>ParArea.Mode</code> is set.			
	<i>Ex.</i>			

ParArea,

	<code>ParArea.Load, 4 ; Load from window #4</code> <code>wait ParArea.Mode and 1</code> <code><check ParArea.CSum></code>		
<code>ParArea.Store</code>	2	32bit=0..31 16bit=0..63	RW
	Store all Raw members into specified window. After a Load/Store operation a wait instruction should follow to make sure that the operation has finished before executing the next instruction. The CSum member is automatically calculated by the firmware. Ex. <code>ParArea.Store, 4 ;Store from window #4</code> <code>wait ParArea.Mode and 1</code>		
<code>ParArea.Raw0..10</code>	3 - 13	$-2^{31}..2^{31}-1$	RW
	Parameter #n of the currently loaded window.		
<code>ParArea.CSum</code>	14	$-2^{31}..2^{31}-1$	RW
	Checksum value.		
<code>ParArea.S0..S10</code>	15 - 25	$-2^{31}..2^{31}-1$	RW
	Scaled RawX value. When this member is read then the following calculation is performed, $sX = \frac{RawX * ScaleX}{65536}$ The intention is to use the DIVMUL operation to calculate the scale value as, <code>ParArea.scale2 = 7400 DIVMUL 800</code> Scale s2 to return a full-scale reading of 7400 for a Raw value of 800.		
<code>ParArea.S0..S10</code>	26 - 36	$-2^{31}..2^{31}-1$	RW
	The scale value used in the calculation when a sX member is read.		

XENDAT,

Group	Group No.	Description
XENDAT	56	<p>The ENDAT transducer interface group. Purpose is to initialize, control and readout position/speed from the ENDAT sensor.</p> <p>The interface supports ENDAT sensors that conforms to the Heidenhain document D297403-00-A-02, version 2.1.</p>

GENERAL

This group is not affected by a GDisp terminal mode command.

FUNCTION

The sensor can be operated in four modes

MANUAL MODE.

In this mode the PL program controls each transmission and reception from the sensor. This is the mode used to initialize, configure and troubleshoot the sensor.

AUTOMATIC SERIAL MODE

The serial position of the sensor is read every servo cycle, or as often as the sensor/sensor clock allows, if this is slower.

This position is absolute up to the capacity of the serial capacity of the encoder, and then incrementally extended to the 32-bit position the DMC² uses. In case the encoder has a capacity of more than 32 bits, only the 32 least significant bits are used.

If this mode is used for commutating and regulating the motor, it must be ensured that the encoder is setup in such a way that it can respond every servo cycle. Also no manually transmitted commands that disturbs this is allowed. (At 2 MHz endat-clock it is possible to run both parameter reads and regulation at the same time)

ANALOG MODE ONLY

The analog signals from the sensor is counted in and up/down quadrate counter to extract a coarse position and an Arctangent calculation is performed on the sin and cos signals to extract a fine position. These positions are then combined to a total position. The extra resolution that can be extracted this way is maximum 11 bits. The 13 least significant bits of this information is absolute, all other higher order bits are incremental.

COMBINED SERIAL AND ANALOG MODE

In this mode the position from the analog mode is combined with the serial data to get an absolute position from the encoder, and to extend it with up to 11 more bits. Thus for a 25 bit encoder the position can be extended by 7 bits to 32 bits fully absolute position. Extending the position with more than 7 bits will in this case

XENDAT,

result in the most significant bits being discarded to keep the total position to 32 bits.

This is the preferred mode, since it allows running the encoder on lower clock-frequency, and allows interruptions in the serial data without affecting the regulation. Also any counting errors in the incremental signal will be corrected when next serial data arrives.

RELATED ITEMS

EXAMPLE USAGE

Before the ENDAT sensor can be used, the sensor must be initialized.

Example code to initialize:

```

;
; 1          Check that the ENDAT host interface has been initialized
              properly.
              if SYSIO.MEMSTAT and 2048 then INIT_ERR
RTmp,        ; Resets any spurious flags.
XENDAT.Status

;
; 2          Reset interface.

XENDAT.Mode, 128 ; Reset host ENDAT interface.

;
; 3          Check ENDAT power supply.
;
XENDAT.Mode, 128 ; if XENDAT.Status and 4 then PWR_ERR.

;
; 4          Synchronize communication with the sensor.
;

XENDAT.Mode, 32 ; Force CLK line low.
clr tmr.t0
wait tmr.t0 > 100 ; This time is sensor dependent.

XENDAT.Mode, 64 ; Force CLK line high.
clr tmr.t0

```

wait tmr.t0 > 2 ; This time is sensor dependent.

XENDAT.Mode, ; Automatic clock mode, and enable.
32+64+1

XENDAT.Clock, 4 ; 2MHz transfer rate Transfer rate is cable length dependent.

;

; 5. Ready to issue manual transfers.

; Read position, read/write parameters.

GROUP MEMBERS (GROUP 56)

Group.Member	Member No.	Range	Ability	Default
XENDAT.Mode	0	0..255	RW	0
ENDAT.Mode = 0.The ENDAT communications not activated.				
<p>b0 Serial communication interface active.</p> <p>b1 Analog interface active.</p> <p>b2 Standard encoder interface active. See the IENC group.</p> <p>b3 Enable ENDAT serial communication timeout interrupt See the VECTOR.PosErr member.</p> <p>b4 Serial data timeout has occurred.</p> <p>b5 Serial comm. manual mode, bit0.</p> <p>b6 Serial comm. manual mode, bit1.</p> <p>Bit 0 to Bit 6 initialize the ENDAT sensor.</p> <p>00 Normal mode.</p> <p>01 Force clock low.</p> <p>10 Force clock high.</p> <p>11 Automatic mode.</p> <p>After this mode is given then Normal mode is automatically entered.</p> <p>b7 Reset interface. When this bit is set then the ENDAT communication interface is reset.</p> <p>Mode values: 0 = ENDAT interface turned off. 1 = Serial mode active.</p>				

XENDAT,

2 = Analog mode active.
 3 = Combined serial and analog mode.
 4 + any of the above: The incremental encoder interface active. (See the IENC group)
 The following values are used for initializing the interface and sensor:
 32 = Force the ENDAT clock line LOW
 64 = Force the ENDAT clock line HIGH
 64+32 = Start the ENDAT CLOCK.
 128 = Reset the interface hardware.

<code>XENDAT.Clock</code>	1	4..99	RW	80
<p>This sets the clock frequency to the ENDAT sensor. *</p> <p>Frequency is calculated as follows: $f = 10\,000\,000 / (\text{value} + 1)$</p> <p>Max ENDAT clock is 2 MHz, which gives the value: 4 Min ENDAT clock is 0.1 MHz, which gives the value: 99</p> <p>The max allowable clock rate is also depending on the cable length to the sensor.</p> <p>*See the ENDAT specification for a more detailed description.</p> <p>** Up to 127 Accepted by hardware, but will generate an out-of-specification clock frequency.</p> <p>Set frequency for the ENDAT communication. Below is a list of example values and the resulting communication frequency,</p> <p>99 => 100 kHz 79 => 125 kHz 39 => 250 kHz 19 => 500 kHz 9 => 1.0 MHz 4 => 2.0 MHz</p>				
<code>XENDAT.</code>	2	0..65535	R	x
<p>NIU</p>				
<code>XENDAT.Status</code>	3	0..65535	R	x
<p>Shows status of hardware when (<code>XENDAT.Mode</code> and 1) = 0, otherwise gives status data returned for last manually transmitted command.</p> <p>b0 (1) = ALRMbit, sensor alarm.</p> <p>This bit is only valid after a position transfer,</p>				

	<p><code>XENDAT.Transfer = 0.</code></p> <p>b1 (2) = CRCbit, CRC compare mismatch.</p> <p>b2 (4) = PWRbit, current status of the ENDAT power supply.</p> <p>b3..b15 = reserved.</p>			
<code>XENDAT.Config</code>	4	0..63	RW	25
	<p>Specifies the number of position data bits for the particular sensor in use. This value must be set to match the actual sensor in use.</p> <p>The DMC² can handle maximum 32 bits of position information, thus if a sensor with more than 32 bits is used, the most significant bits are discarded. If the sensor used less than 32 bits, the driver will incrementally extend the ENDAT position to 32 bits, to be able to fully utilize the DMC² functionality.</p>			
<code>XENDAT.Rdy</code>	5	0..1	R	1
	<p>This member indicates when the ENDAT communications interface has completed a manually issued command.</p>			
<code>XENDAT.Data</code>	6	16 bit	RW	0
	<p>Data to/from a manually issued command.</p> <p>Read: Value received from the sensor as a result of a parameter read transfer.</p> <p>Write: Value to transmit to the sensor for a parameter writes transfer.</p>			
<code>XENDAT.Addr</code>	7	32 bit	RW	0
	<p>Address to read/write to sensor in manual mode.</p> <p>The ENDAT manual uses WORD as a notation for parameters, to convert a WORD value to an address use <code>XENDAT.Addr = rWORD</code> and 15.</p> <p>Read: Address received from the sensor as a result of a parameter read transfer.</p> <p>Write: Address to write <code>ENDAT.Data</code> to, within the ENDAT based sensor.</p>			
<code>XENDAT.Transfer</code>	8	0-7	RW	0
	<p>To manually issue a command, command type is 0 to 7.</p> <p>After sensor initialization the following transfer types are available. Before a new transfer is issued with the ENDAT interface a programmer must check that</p>			

XENDAT,

	<code>XENDAT.Rdy</code> is 1.
0	<p>Read absolute position from sensor.</p> <p>Setup: None</p> <p>Check:<code>XENDAT.Status</code> -> ALRMbit, CRCbit and PWRbit</p> <p>Result: If the check passed then, <code>ENDAT.Pos</code> = The 32bit signed position.</p>
1	<p>Select memory area. The selected memory area will be valid until a new is chosen.</p> <p>Setup: Set <code>ENDAT.Addr</code> with the wanted MRS code. For MRS-codes see the ENDAT manual.</p> <p>Check: After the transfer is completed then verify that <code>ENDAT.Addr</code> holds the wanted MRS code.</p> <p>Result: If the check passed then the wanted MRS code, memory area is selected within the sensor.</p>
2	Receive test values from sensor. NYI
3	<p>Send parameter to sensor.</p> <p>Setup: Select the appropriate memory area, MRS-code, using a type 1 transfer. <code>ENDAT.Addr</code> = address of the parameter to write to. <code>ENDAT.Data</code> = 16-bit data to write.</p> <p>Check: After the transfer is completed verify that <code>ENDAT.Addr</code> holds the address of the parameter and check <code>XENDAT.Status</code> PWRbit, CRCbit. (ALRMbit is not valid at this time and must be ignored).</p> <p>Result: If the check passed then the data has been written to the parameter.</p>
4	<p>Receive parameter from sensor.</p> <p>Setup: Select the appropriate memory area, MRS-code, using a type 1 transfer. <code>XENDAT.Addr</code> = the address of the parameter to read.</p> <p>Check:</p>

					<p>After the transfer is completed verify that <code>XENDAT.Addr</code> holds the address of the parameter that was read and check <code>XENDAT.Status</code></p> <p>PWRbit, CRCbit. (ALRMbit is not valid at this time and must be ignored).</p> <p>Result:</p> <p><code>XENDAT.Data</code> contain the parameter value.</p> <p><code>XENDAT.Addr</code> contain the parameter address.</p>
	5	Send reset to sensor.			<p>Setup:</p> <p><code>XENDAT.Addr</code> = value1</p> <p><code>XENDAT.Data</code> = value2</p> <p>Check:</p> <p>After the transfer is completed verify that <code>XENDAT.Addr</code> and <code>XENDAT.Data</code> holds the values we programmed during setup.</p> <p>Result:</p> <p>If the check passed, the sensor has been reset.</p>
	6	Send test command to sensor. NYI.			
	7	Receive test data from sensor. NYI.			
<code>XENDAT.ManPos</code>	9	32 bit	RW	0	Resulting position when a position was requested manually. Position read from the sensor after a type 0 transfer.
<code>XENDAT.Ver</code>	10	16 bit	R	Current version	FPGA code version used in the ENDAT interface.
<code>XENDAT.Pos</code>	11	±31 bit	R	0	Automatically retrieved position. This is the position used for regulation of the motor. The resolution and update rate for this value is depending on the selected operating mode.
<code>XENDAT.Speed</code>	12	±31 bit	R	0	<p>This is the speed from the ENDAT sensor.</p> <p>In the DMC² the maximum supported speed for regulation and gear box etc is -32767000 .. 32767000. Due to the possible high resolution of an ENDAT sensor this speed may be reached at a few 100 rpm. To avoid this problem the speed reading here does not have that limitation. If the ENDAT sensor is used as an input to the regulator or gearbox, the user must make sure that this lower speed is not exceeded.</p>

XENDAT,

					The speed sent to the regulator and gearbox is limited to ± 32767000 . If this speed does not allow a sufficiently high rpm, the resolution of the sensor has to be decreased. (See <code>XENDAT.PosShift</code>).
<code>XENDAT.SerABSPos</code>	13	Not A	R	0	<p>This is the serial absolute position as received from the sensor, the position wraps at the capacity of the sensor.</p> <p>The value is only updated if bit 0 in <code>XENDAT.Mode</code> is set.</p> <p>Note A. Sensor dependent, max 32bit.</p>
<code>XENDAT.SerPos</code>	14	± 31 bit	R	0	<p>This is the serial extended position from the sensor.</p> <p>The position wraps at the DMC² 32 bit position capacity.</p> <p>The <code>XENDAT.SerWrapCnt</code> member can control the difference between this position and the absolute position.</p> <p>The value is only updated if bit 0 in <code>XENDAT.Mode</code> is set.</p>
<code>XENDAT.SerErrors</code>	15	± 31 bit	RW		<p>Error counter for transmission errors from the endat sensor.</p> <p>The following errors are counted:</p> <p>Alarm from endat. CRC error on received frame. 5 Volt supply error.</p>
<code>XENDAT.IncErrors</code>	16	± 31 bit	RW	0	<p>Error counter for detected errors in the UP/DOWN counter. This quadrature counter is used to count the whole sine/cosine periods of the analog ENDAT signal. When there is a situation where both input signal change state at the same time, the discriminator in the encoder cannot determine if this should be an UP or DOWN count. In this case the I errors counter is incremented.</p>
					<p>i Since the same hardware counter is used in the IENC group, this error counter is also common to both groups.</p>
<code>XENDAT.SerIncOffs</code>	17	± 31 bit	R	0	<p>When combined serial and analog (incremental) mode is used, this value holds the difference between the incremental and the absolute position from the sensor. If there are no errors, this value will be constant, once the system is started.</p>

<code>XENDAT.RPos</code>	18	±31 bit	R	0	Referenced position, value = <code>XENDAT.Pos</code> - <code>XENDAT.PosOffs</code> .
<code>XENDAT.PosOffs</code>	19	±31 bit	RW	0	Position offset between <code>XENDAT.Pos</code> and <code>XENDAT.RPos</code> .
<code>XENDAT.SerWrapCnt</code>	20	±31 bit	RW	0	Position offset between <code>XENDAT.SerABSPos</code> and <code>XENDAT.SerPos</code> . This value can only be set to multiples of the ENDAT sensor capacity. Typical use is to set the home position with aid of external sensors if the ENDAT absolute range is not large enough.
<code>XENDAT.SinOffs</code>	21	0..16384	RW	8192	Offset calibration for ENDAT sine-signal.
<code>XENDAT.CosOffs</code>	22	0..16384	RW	8192	Offset calibration for endat cosine-signal.
<code>XENDAT.PosShift</code>	23	0..11	RW	0	In ENDAT analog mode, or combined mode, this value is the number of bits from the evaluation of the analog sine and cosine signals that is added to the serial position data.
i	This parameter does not affect the values that are used for commutation of the motor.				
	The motor is commutated on the standard (non-extended) value from the ENDAT sensor. <code>Motor.Ppr</code> shall be set to the resolution the sensor has when <code>XENDAT.PosShift</code> is set to zero, and it is not to be changed when the <code>XENDAT.PosShift</code> is changed. Since this parameter changes the magnitude of speed and position sent into the regulator for a given mechanical movement, the regulator gain settings as well as motion profiles etc will have to be changed if the resolution of the sensor is changed.				
<code>XENDAT.SerPosAge</code>	24	0..32767	R		Age(in milliseconds), of last serial position value from the Endta sensor (i
<code>XENDAT.SerTimeOut</code>	25	0..32767	RW		Timeout (in milliseconds), before a position error interrupt is set pending. (See <code>Xendat.mode</code>)

Counter

COUNTER

Group	Group No.	Description
Counter	57	The counter group can be configured to count an external hardware event, measure frequency, or to generate a high-resolution time count. The source signal can be an internal frequency of 10 Mhz or an external digital input.(Di1 or Encoder Zero pulse)

GENERAL

The DMC² can utilize an internal HW counter to count high frequencies.

FUNCTION

A hardware resource within the CPU is used to count every edge on a source signal. The frequency is derived as a number of edges counted during a specified timebase and then scaled. A prescaler is implemented to adjust the scale.

Be aware that the frequency value is only updated once every timebase period, which means that the value of `Counter.Freq` is not valid until at least one period has elapsed.

RELATED ITEMS

Capture.Mode

EXAMPLE USAGE

Measure the frequency of an external signal connected to the Incremental Encoder Interface:

```
Capture.Mode,8           ;redirect zero pulse input to counter
Counter.mode,3           ;activate with external source
Counter.Timebase,100    ;100 mS timebase for counting
```

GROUP MEMBERS (GROUP 57)

Group.Member	Member No.	Range	Ability	Default
Counter.Mode	0	0..256	R (W)	0
	b0	Enable or disable Counter group. 0 = Disabled. 1 = Enable.		
	b1	Counting source.		

					<p>0 = Internal time counter. 1 = External event counter.(selcted with capture.mode) The counter counts both the positive and the negative going external edge.</p> <p>b2-b6 Reserved.</p> <p>b7 Frequency overflow indication. During frequency calculation overflow is indicated. This occurs in the case a too high frequency is to be measured with a too long time base value</p>														
<code>Counter.Count</code>	1	±31 bit	RW	0	The count value. The <code>Counter.PreScale</code> can be used to divide the count value with a fixed value. In the case the Counter is configured for internal time measurement then only a 16 bit counter value is returned.														
<code>Counter.Freq</code>	2	±31 bit	R	0	The count frequency in Hz. The <code>Counter.TimeBase</code> indicate the time that is used for measuring the frequency.														
<code>Counter.TimeBase</code>	3	1..1000	RW	1	<p>The time interval used for frequency calculation in milliseconds.</p> <p>Note. If a time base value is too large for the given frequency then measurement overflow occurs. The overflow is indicated in <code>Counter.Mode</code>.</p> <p>The theoretical measurement range for different values of <code>Counter.TimeBase</code> are:</p> <table border="0"> <tr> <td>1000ms</td> <td>1Hz - 32kHz</td> </tr> <tr> <td>500ms</td> <td>2Hz - 65kHz</td> </tr> <tr> <td>100ms</td> <td>10Hz - 327kHz</td> </tr> <tr> <td>50ms</td> <td>20Hz - 655kHz</td> </tr> <tr> <td>10ms</td> <td>100Hz - 3MHz</td> </tr> <tr> <td>5ms</td> <td>200Hz - 6MHz</td> </tr> <tr> <td>1ms</td> <td>1000Hz - 32MHz</td> </tr> </table>	1000ms	1Hz - 32kHz	500ms	2Hz - 65kHz	100ms	10Hz - 327kHz	50ms	20Hz - 655kHz	10ms	100Hz - 3MHz	5ms	200Hz - 6MHz	1ms	1000Hz - 32MHz
1000ms	1Hz - 32kHz																		
500ms	2Hz - 65kHz																		
100ms	10Hz - 327kHz																		
50ms	20Hz - 655kHz																		
10ms	100Hz - 3MHz																		
5ms	200Hz - 6MHz																		
1ms	1000Hz - 32MHz																		
<code>Counter.PreScale</code>	4	0..7	RW	1	<p>Shift factor for Counter.Count value.</p> <p>0 = divide by 1 1 = divide by 2 2 = divide by 4 3 = divide by 8 4 = divide by 16</p>														

Counter

5 = divide by 32

6 = divide by 64

7 = divide by 128

In the case the counter is configured to measure time the resolution is given by reading `Counter.Freq`. If the frequency is 10000000 then the time resolution is 100ns.

IDENTIFIER

Group	Group No.	Description
Identifier	58	The identifier group is the interface to the front panel selector. The standard DMC ² is equipped with a decimal encoded selector, the value from one single selector will be in the range 0..9.

GENERAL

The DMC² is equipped with two rotary switches on the front panel. The switches are BCD coded (0 - 9) and have no predefined function.

The intention is to use them as address switches for serial- or fieldbus communication.

FUNCTION

The switches are read as any other digital input.

RELATED ITEMS

EXAMPLE USAGE

Set node number for serial communication at startup:

GROUP MEMBERS (GROUP 58)

Group.Member	Member No.	Range	Ability	Default
Identifier.Mode	0	0..2	R W	0
	NYI.			
Identifier.Value	1	0..255	R	0
	The decimal value of the combined selector.			
Identifier.V0	2	0..15	R	0
	The value of the least significant selector. (bottom switch)			
Identifier.V1	3	1..15	R	0
	The value of next selector. (top switch).			

RDPDATA

RDPDATA

Group	Group No.	Description
RDPDATA	59	Reading PDATA contents.

GENERAL

Data arrays can be define in the DMC² using the Pdata statement.

The standard usage of such an array is as input to the Profile Generator (Profile Acc) or as a Cam table.

FUNCTION

This group allows the user to read a Pdata array for any purpose.

RELATED ITEMS

Pdata statement

EXAMPLE USAGE

```
ptab:   PDATA 89, 2
        PDATA R3, R9
        PDATA 0,0
```

The RPDATA group can be used for reading the content of the PDATA table.

```

        R0, @ptab
loop:   RPDATA.Load, R0
        if RPDATA.Status and 1 then LoopStop

        disp RPDATA.Arg1
        disp RPDATA.Arg2
        add R0, 1
        goto loop
LoopStop:
        stop
```

Running this program will show,

```
>run
```

>RDPDATA.Arg1 = 89
 >RDPDATA.Arg2 = 2
 >RDPDATA.Arg1 = <content of R3>
 >RDPDATA.Arg2 = <content of R9>
 >RDPDATA.Arg1 = 0
 >RDPDATA.Arg2 = 0
 >

GROUP MEMBERS (GROUP 59)

Group.Member	Member No.	Range	Ability	Default
RDPDATA.Status	0	0..7	RW	0
	b0	Indicate that the value given for RDPDATA.Load member is not a PDATA instruction.		
	b1	Indicate that RPDData.Arg1 value came from a register not a constant.		
	b2	Indicate that RPDData.Arg2 value came from a register not a constant.		
RDPDATA.Load	1	0..8192	RW	0
	The line number specified must point to a PDATA instruction. If the specified line is not a PDATA instruction then a bit in RDPDATA.Status is set and the assignment is rejected.			
RDPDATA.Arg1	2	±31 bit	R	0
	The content of the first argument of the PDATA line pointed out by RDPDATA.Load. If the PDATA line is specified using a register then the value of the register is used.			
RDPDATA.Arg2	3	±16 bit	R	0
	The content of the second argument of the PDATA line pointed out by RDPDATA.Load. If the PDATA line is specified using a register then the value of the register is used.			

SANYBUS

Group	Group No.	Description
SAnyBus	60	Interface to the Anybus-S board by Hassbjer Micro Systems AB.

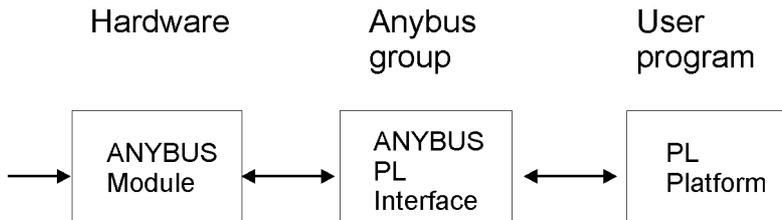


Figure 8. Blockdiagram, AnyBus support.

GENERAL

Before the SANYBUS group is operational some steps must be done to initialize the running environment. This is done by enabling the ANYBUS group and then use the mailbox communication mechanisms to initialize the ANYBUS-S module. The details can be found in the ANYBUS-S documentation from HMS Fieldbus systems AB. (<http://www.hms.se/>)

FUNCTION

This group provides an interface for a DMC2E application to the family of Anybus-S fieldbus interfaces available from HMS Fieldbus systems AB. Preferred bustypes are Profibus-DP, Interbus and Device NET. The interface is seen as an input and an output buffer where data can be read and written. Some commands are provided to control the data flow. The application has to define how the data is to be used (no defined standard protocol).

RELATED ITEMS

Vector	PL interrupts service routine vectors.
Int	PL2 interrupts masks.
SAnyBus	Anybus module control group.
ABOUT	Output buffer, from AnyBus-S module to the PL2 environment. This buffer is for debug only.
ABIN	Input buffer, to AnyBus-S module from the PL2 environment. This buffer is for debug only.

EXAMPLE USAGE

```

;
; Testing the ANYBUS-S interface to the DMC2E.
;
;
; Sets up an ECHO of 4 bytes.
; Incoming data is echoed back on the Profibus
; Note. These names are only visible at compile time.
; The two groups ABInMail and ABOutMail are arrays of 16bit integers.
;
;
; DMC -> Fieldbus
;
        .define ABInMail.MessageID      = { 61, 0 }
        .define ABInMail.MessageInfo    = { 61, 1 }
        .define ABInMail.Command        = { 61, 2 }
        .define ABInMail.DataSize       = { 61, 3 }
        .define ABInMail.FrameCount     = { 61, 4 }
        .define ABInMail.FrameNumber    = { 61, 5 }
        .define ABInMail.OffsetHigh     = { 61, 6 }
        .define ABInMail.OffsetLow      = { 61, 7 }

; Profibus
        .define PBIIn.InIOLen           = { 61, 16 }
        .define PBIIn.InDPRAMLen       = { 61, 17 }
        .define PBIIn.InTotalLen       = { 61, 18 }
        .define PBIIn.OutIOLen         = { 61, 19 }
        .define PBIIn.OutDPRAMLen      = { 61, 20 }
        .define PBIIn.OutTotalLen      = { 61, 21 }

; Generic
        .define ABInMail.ModuleStatus  = { 61, 22 }
        .define ABInMail.IrqNotify     = { 61, 23 }

;
; Fieldbus -> DMC
;
        .define ABOutMail.MessageID    = { 62, 0 }
        .define ABOutMail.MessageInfo  = { 62, 1 }
        .define ABOutMail.Command      = { 62, 2 }

        .define ABOutMail.DataSize     = { 62, 3 }
        .define ABOutMail.FrameCount   = { 62, 4 }
        .define ABOutMail.FrameNumber  = { 62, 5 }
        .define ABOutMail.OffsetHigh   = { 62, 6 }
        .define ABOutMail.OffsetLow    = { 62, 7 }
        .define ABOutMail.Error        = { 62, 15 }

        .register rOnLine
        .register rProtNumber
        .register rTmp91,rTmp92,rTmp93
        .register tmp10, tmp20

```

SAnyBus

```

.define cSTART_INIT    = 1
.define cANYBUS_INIT   = 2
.define cSET_ETN_CONFIG = 3
.define cCONNECT_TIMEOUT = 4
.define cEND_INIT      = 5

.define cNoModule      = 0
.define cProfibusDB    = 1

; Start

        gosub SRou_SAnyBusInit          ;init the module
        wait 1=2                        ;

;
; Give the fieldbus module time to start.
;
SRou_SAnyBusInit:
    SAnybus.TimeOut,50
    clr rOnLine
    clr tmr.t0
    wait SAnybus.FBType <> cNoModule
    if SAnybus.FBType = cProfibusDB then ProfibusDP
    clr tmr.t0
    wait tmr.t0 > 100
    return

;=====
;
; Profibus-DP
;
ProfibusDP:

    Vector.SAnybus, @IRou_ProfiBus      ;set vector
    or int.sysmask, 4096                 ;enable interrupt
    SAnybus.HasMail, 0                   ; Clear mailbox flag
    SAnybus.Mode, 1+2+4+8+16             ; Enable SANYBUS and all interrupts and
                                         reverse ;read

    gosub SendStartInit                  ;
    gosub SendInit64_64                  ;
    gosub SendEndInit                    ;
    return

;Interrupt entry for profibus
;Remember that the ABOut buffer is holds the OUTPUT FROM the Anybus module
;seen fromthe PL2 side.
;The ABIn buffer holds the INPUT TO the Anybus module seen from
the PL2 side.
IRou_ProfiBus:
    if SAnybus.HasOutput = 0 then isrProfibus00 ;test if any data in output
                                                ;buffer
    SAnybus.HasOutput, 0                   ; Clear flag
;

```

```

; Copy return data
anybusout getlong rTmp91,0           ;read byte 0 - 3 into reg
anybusout getint rTmp92,4           ;read byte 4 - 5 into reg
anybusout getbyte rTmp93,6         ;read byte 6 into reg

anybusin putlong rTmp91,0           ;write back first long

AnybusIO Send                        ;send it back

isrProfibus00:
    if SAnybus.HasMail = 0 then isrProfibus10

; We handle this in code. But we show it here anyway.
;    disp SAnybus.HasMail

isrProfibus10:
    if SAnybus.OnLine = rOnLine then isrProfibus20
;    disp SAnybus.OnLine
    rOnLine, SAnybus.OnLine

isrProfibus20:
    ireturn sys 4096

;
; START_INIT
;
SendStartInit
:    rProtNumber, cSTART_INIT
    SAnybus.Command, 1           ; Prepare START_INIT mail message
    wait SAnybus.Command = 0     ; Wait for it to be done
    SAnybus.Command, 7           ; Send to fieldbus module
    wait SAnybus.HasMail = 1     ; Wait for response mail
; Investigate response mail for errors here
    if ABOutMail.MessageInfo < 0 then ErrorProt
    SAnybus.HasMail, 0           ; Clear mailbox flag
    return

SendInit64_64:
    rProtNumber, cANYBUS_INIT
    SAnybus.Command, 3           ; Prepare ANYBUS_INIT mail message
    wait SAnybus.Command = 0     ; Wait for it to be done

; Modify default values here in ABInMail
; Length in bytes!
;
    PBlIn.InIOLen, 64
    PBlIn.InDPRAMLen, 64
    PBlIn.InTotalLen, 64
    PBlIn.OutIOLen, 64
    PBlIn.OutDPRAMLen, 64
    PBlIn.OutTotalLen, 64

    ABInMail.ModuleStatus, 512+2 ; default value (you must keep 512)

```

SAnyBus

```

        ABInMail.IrqNotify, 7          ; default value, do not change

        SAnybus.Command, 7           ; Send to fieldbus module
        wait SAnybus.HasMail = 1     ; Wait for response mail
; Investigate response mail for errors here
        if ABOutMail.MessageInfo < 0 then ErrorProt
        if ABOutMail.Error <> 0 then ABIErrProt

        SAnybus.HasMail, 0          ; Clear mailbox flag
        return

;
; END_INIT
;
SendEndInit
:
    rProtNumber, cEND_INIT
    SAnybus.Command, 2             ; Prepare END_INIT mail message
    wait SAnybus.Command = 0       ; Wait for it to be done
    SAnybus.Command, 7             ; Send to fieldbus module
    wait SAnybus.HasMail = 1       ; Wait for response mail
; Investigate response mail for errors here
    if ABOutMail.MessageInfo < 0 then ErrorProt

    SAnybus.HasMail, 0             ; Clear mailbox flag
    return

ErrorProt:
    tmp10 = ABOutMail.MessageInfo and 0ffh
    tmp20 = ABOutMail.MessageInfo and 0f00h
    tmp20 = tmp20 >> 8
;
    iprint @txtProtError, rProtNumber
txtProtError:    image "Error in: %d"

;
; if ABOutMail.MessageInfo and 04000h then ErrCommand
;
; Response message
;
;
    clr tmr.t0
    wait tmr.t0 > 100
    stop

; Command error
;
;
ErrCommand:
    clr tmr.t0
    wait tmr.t0 > 100
    stop
ABIErrProt:
    clr tmr.t0
    wait tmr.t0 > 100
    stop

```

GROUP MEMBERS (GROUP 60)

Group.Member	Member No.	Range	Ability	Default
SAnyBus.Mode	0	0..256	R(W)	128
Mode = <bitvalue>	<p>B0 (1) = Enable group. B1 (2) = Enable interrupt generation for received mail, ABOutMail has new data. When this bit is activated a PL interrupt Routine can be used to service this event.</p> <p>B2 (4) = Enable interrupt generation for received data, ABOut has new data. When this bit is activated a PL interrupt Routine can be used to service this event.</p> <p>B3 (8) = Enable interrupt generation for module state change, Anybus.OnLine changes. When this bit is activated a PL interrupt Routine can be used to service this event.</p> <p>B4 (16) = Byte order selection for the AnybusIn and AnybusOut read/write instructions. 0 = Big endian (or Motorola) byte order. 1 = Little endian (or Intel) byte order.</p> <p>b7 (128) = Control ANYBUS-S reset pin. When this bit is set the ANYBUS module's reset pin is active. To reset the module a user written PL routine should keep this bit active for 10ms. (Note. After reset of a slave, the field bus master may have to be restarted). 0 = Reset pin is inactive. 1 = Reset pin is active.</p>			
SAnyBus.HasMail	1	0..1	RW	0
	<p>Indicates that mail is available in the ABOutMail group. It is the users responsibility to reset this bit when the data has been acted upon</p>			
SAnyBus.HasOutput	2	0..1	RW	0
	<p>Indicates that data is available in the ABOut group. It is the users responsibility to reset this bit when data has been acted upon.</p>			
SAnyBus.Error	3	0..15	RW	0
	<p>Errors reported during access/release of the dual ported memory (DPM). b0 (1) = Timeout trying to access IN area. b1 (2) = Timeout trying to release IN area. b2 (4) = Timeout trying to access OUT area.</p>			

SAnyBus

	b(8) = Timeout trying to release OUT area. The timeout time can be set in SAnybus.Timeout			
SAnyBus.Online	4	0..1	RW	0
	Indicates that the module changed the online state. 0 = Offline. 1 = Online			
SAnyBus.State	5	0..255	R	0
	Indicate the ANYBUS-S interface state for full duplex input and output. The value actually indicate the state of two separate state machines within the group (IN and OUT), Out access/release 0 = Idle 1 = Init access to out area. 2 = Wait for access to out area. 4 = Wait for release of out area. In access/release 0 = Idle 256 = Init access to in area. 512 = Wait for access to in area. 768 = Wait for release of in area.			
SAnyBus.Command	6	0..255	RW	0
	Control the behavior of the interface. The user must wait For SANYBUS.Command member to be 0 before giving a new command. Commands: 0 - No operation. 1 – Copy default START_INIT mail data to ABInMail. Use command #7 to send mail to module. 2 – Copy default END_INIT mail data to ABInMail. Use command #7 to send mail to module. 3 - Copy default ANYBUS_INIT mail data to ABInMail. The user may modify data within AbinMail before using command #7. 4 - reserved, do not use. 5 - reserved, do not use. 6 - reserved, do not use. 7 - Send content of ABInMail to the ANYBUS-S module.			

	As a response to this command the ANYBUS.HasMail bit should be activated, indicating that response mail is available.			
<i>SAnyBus.FBType</i>	7	0..32767	R	Module dependant
	The field bus type connected to the interface. 1 = ProfiBus-DP 16 = Interbus 21 = LonWorks 32 = CANopen 8 = InterBus-S 37 = DeviceNet 64 = Modbus Plus 69 = Modbus RTU 101 = ControlNet 128 = Ethernet (Modbus/TCP) See (http://www.hms.se/)			
<i>SAnyBus.SWVer</i>	8	0..32767	R	Module dependant
	The version number of the ANYBUS module firmware. See (http://www.hms.se/)			
<i>SAnyBus.ModType</i>	9	0..32767	R	Module dependant
	The ANYBUS module type. See (http://www.hms.se/)			
<i>SAnyBus.ModuleStat</i>	10	0..32767	R	Module dependant
	Module status. b0 (1) = Fieldbus on/off line. b1 (2) = The out area is freezed/cleared when going off line. B8 (256) = The in area freezed/cleared when going off line. b9 (512) = Changed data field is active. This register indicates the way that the module has been programmed. See (http://www.hms.se/)			
<i>SAnyBus.InIOLen</i>	11	0..32767	R	0
	This register indicates the way that the module has been programmed. See (http://www.hms.se/)			
<i>SAnyBus.InDPRAMLen</i>	12	0..32767	R	0
	This register indicates the way that the module has been programmed.			

SAnyBus

	See (http://www.hms.se/)			
<code>SAnyBus.InTotLen</code>	13	0..32767	R	0
	This register indicates the way that the module has been programmed. See (http://www.hms.se/)			
<code>SAnyBus.OutIOLen</code>	14	0..32767	R	0
	This register indicates the way that the module has been programmed. See (http://www.hms.se/)			
<code>SAnyBus.OutDPRAMLen</code>	15	0..32767	R	0
	This register indicates the way that the module has been programmed. See (http://www.hms.se/)			
<code>SAnyBus.OutTotLen</code>	16	0..32767	R	0
	This register indicates the way that the module has been programmed. See (http://www.hms.se/)			
<code>SAnyBus.InCount</code>	19	32 bit	RW	0
	Number of times that the ABIn area has been given to the ANYBUS-S module.			
<code>SAnyBus.OutCount</code>	20	32 bit	RW	0
	Number of times that the ABOut area has been given to the application.			
<code>SAnyBus.InMailCount</code>	21	32 bit	RW	0
	Number of times that the ABInMail area has been given to the ANYBUS-S module.			
<code>SAnyBus.OutMailCount</code>	22	32 bit	RW	0
	Number of times that the ABOutMail area has been given to the application			
<code>SAnyBus.TimeOut</code>	23	32 bit	RW	2
	The time to wait for access/release of the dual ported memory before a timeout error is reported.			

ANYBUS RELATED PL INSTRUCTIONS

Instructions to communicate with the AnyBus module via the buffers. These instructions are doing the Intel byte order to Motorola byte order conversion. The following parameters define the data type that is passed to the instruction:

PutDWORD	Will write a 32 bit unsigned value.
PutWORD	Will write a 16 bit unsigned value.
PutBYTE	Will write an 8 bit unsigned value.

PutLONG Will write a 32 bit signed value.
 PutINT Will write a 16 bit signed value.
 PutSCHAR Will write an 8 bit signed value.
 GetDWORD Will read a 32 bit unsigned value.
 GetWORD Will read a 16 bit unsigned value.

GetBYTE Will read an 8 bit unsigned value.
 GetLONG Will read a 32 bit signed value.
 GetINT Will read a 16 bit signed value.
 GetSCHAR Will read an 8 bit signed value.

MANIPULATE THE ANYBUS INPUT BUFFER

Data in the InPut buffer is to be transmitted onto the field bus. These instructions can manipulate data in a specified location of a field bus frame. The instructions below are used to manipulate the AnyBus input buffer.

offs = 0..479 (index into data array)

AnyBusIN putDWORD <i>Reg, offs</i>	Put 32 bit unsigned data from register <i>Reg</i> at index <i>offs</i> .
AnyBusIN putWORD <i>Reg, offs</i>	Put 16 bit unsigned data from register <i>Reg</i> at index <i>offs</i> .
AnyBusIN putBYTE <i>Reg, offs</i>	Put 8 bit unsigned data from register <i>Reg</i> at index <i>offs</i> .
AnyBusIN putLONG <i>Reg, offs</i>	Put 32 bit signed data from register <i>Reg</i> at index <i>offs</i> .
AnyBusIN putINT <i>Reg, offs</i>	Put 16 bit signed data from register <i>Reg</i> at index <i>offs</i> .
AnyBusIN putSCHAR <i>Reg, offs</i>	Put 8 bit signed data from register <i>Reg</i> at index <i>offs</i> .

AnyBusIN getDWORD <i>Reg, offs</i>	Get 32 bit unsigned data from index <i>offs</i> and store in register <i>Reg</i> .
AnyBusIN getWORD <i>Reg, offs</i>	Get 16 bit unsigned data from index <i>offs</i> and store in register <i>Reg</i> .
AnyBusIN getBYTE <i>Reg, offs</i>	Get 8 bit unsigned data from index <i>offs</i> and store in register <i>Reg</i> .
AnyBusIN getLONG <i>Reg, offs</i>	Get 32 bit signed data from index <i>offs</i> and store in register <i>Reg</i> .
AnyBusIN getINT <i>Reg, offs</i>	Get 16 bit signed data from index <i>offs</i> and store in register <i>Reg</i> .
AnyBusIN getSCHAR <i>Reg, offs</i>	Get 8 bit signed data from index <i>offs</i> and store in register <i>Reg</i> .

MANIPULATE THE ANYBUS OUTPUT BUFFER

Data in the OutPut buffer is received on the field bus. When a field bus 'receive' is completed then the entire field bus frame is stored in the OUTPUT buffer and a PL interrupt is generated. The instructions below are used to manipulate the AnyBus output buffer.

offs = 0..479 (index into data array)

AnyBusOUT putDWORD <i>Reg, offs</i>	Put 32 bit unsigned data from register <i>Reg</i> at index <i>offs</i> .
AnyBusOUT putWORD <i>Reg, offs</i>	Put 16 bit unsigned data from register <i>Reg</i> at index <i>offs</i> .
AnyBusOUT putBYTE <i>Reg, offs</i>	Put 8 bit unsigned data from register <i>Reg</i> at index <i>offs</i> .
AnyBusOUT putLONG <i>Reg, offs</i>	Put 32 bit signed data from register <i>Reg</i> at index <i>offs</i> .
AnyBusOUT putINT <i>Reg, offs</i>	Put 16 bit signed data from register <i>Reg</i> at index <i>offs</i> .
AnyBusOUT putSCHAR <i>Reg, offs</i>	Put 8 bit signed data from register <i>Reg</i> at index <i>offs</i> .
AnyBusOUT getDWORD <i>Reg, offs</i>	Get 32 bit unsigned data from index <i>offs</i> and store in register <i>Reg</i> .
AnyBusOUT getWORD <i>Reg, offs</i>	Get 16 bit unsigned data from index <i>offs</i> and store in register <i>Reg</i> .
AnyBusOUT getBYTE <i>Reg, offs</i>	Get 8 bit unsigned data from index <i>offs</i> and store in register <i>Reg</i> .
AnyBusOUT getLONG <i>Reg, offs</i>	Get 32 bit signed data from index <i>offs</i> and store in register <i>Reg</i> .
AnyBusOUT getINT <i>Reg, offs</i>	Get 16 bit signed data from index <i>offs</i> and store in register <i>Reg</i> .
AnyBusOUT getSCHAR <i>Reg, offs</i>	Get 8 bit signed data from index <i>offs</i> and store in register <i>Reg</i> .

TRANSFERS THE ANYBUS INPUT BUFFER

AnyBusIO Send	Send the buffer onto the fieldbus
---------------	-----------------------------------

ABINMAIL

Group	Group No.	Description
ABInMail	61	Mail message handling with the Anybus-S modules.

GENERAL

Area where mail is build ups by the application program and transmitted to the Anybus-S module.

This is an array of 16bit words.

It is used to send commands to the Anybus-S module.

FUNCTION

RELATED ITEMS

EXAMPLE USAGE

GROUP MEMBERS (GROUP 61)

Group.Member	Member No.	Range	Ability	Default
ABINMail	0-143	16bit	RW	0
	ABInMail.0	MessageID		
	ABInMail.1	MessageInfo		
	ABInMail.2	Command		
	ABInMail.3	DataSize		
	ABInMail.4	FrameCount		
	ABInMail.5	FrameNumber		
	ABInMail.6	OffsetHigh		
	ABInMail.7	OffsetLow		
	ABInMail.8..143	Depends on context. See (2).		

ABOUTMAIL

Group	Group No.	Description
ABOutMail	62	Mails received from the Anybus-S modules

GENERAL

Area where mail is received from the Anybus-S module.

This is an array of 16bit words. It is used to hold response messages from the Anybus-S module until processed by a user program.

FUNCTION

RELATED ITEMS

EXAMPLE USAGE

GROUP MEMBERS (GROUP 62)

Group.Member	Member No.	Range	Ability	Default
ABOutMail	0-143	16bit	RW	0
	ABOutMail.0	MessageID		
	ABOutMail.1	MessageInfo		
	ABOutMail.2	Command		
	ABOutMail.3	DataSize		
	ABOutMail.4	FrameCount		
	ABOutMail.5	FrameNumber		
	ABOutMail.6	OffsetHigh		
	ABOutMail.7	OffsetLow		
	ABOutMail.8..143	Depends on context. see (2)		

ABFBus

ABFBUS

Group	Group No.	Description
ABFBus	63	Fieldbus specific information from the Anybus-S modules.

GENERAL

The content of this area is fieldbus specific. This is an array of 8bit bytes. It is the fieldbus specific area. The manual, (Anybus-S interface manual) refers to this as being located at 0x640 (hexadecimal) and also specifies positions into this area in the manual as absolute. To find the ABFBus member based on an absolute specification from the manual then simply subtract 0x640 from the specification.

Example:

The DeviceNet module has "Explicit Connection" status at 0x642, 0x642-0x640 = 0x2.

ABFBus.2 contains "Explicit Connection" status.

Note. Only the first 255 bytes are available out of 384 bytes.

Some fieldbus types does not provide any information in this area,

ProfiBus-S	Area not used.
Interbus	Area not used.
DeviceNet	Connection status information.
Ethernet	Socket information.

FUNCTION**RELATED ITEMS****EXAMPLE USAGE****GROUP MEMBERS (GROUP 63)**

Group.Member	Member No.	Range	Ability	Default
ABFbus	0-254	(bit)	R	0
	.ABFbus.0 - Location 0x640			
	.ABFbus.1 - Location 0x641			
	.ABFbus.2 - Location 0x642			

EN1-EN4, ENCODER1-4

Group	Group No.	Description
EN1	64	These groups are "place holders" or connection points for the different encoders that are available to the system
EN2	65	
EN3	66	
EN4	67	

GENERAL

In earlier versions of the DMC² firmware, there have only been two position sensors available, `RD1` and `RD2`. `RD1` has been hard connected to the regulator and the commutation of the motor. `RD2` has been connected to the Gear box/CAM input.

With the introduction of ENDAT encoders and incremental encoders in the DMC² firmware it became necessary to be able to select which encoder to use for respective function. This selection is controlled via the EN1..EN4 groups.

The output from the ENx groups are connected to the other system functions as follows:

EN No	FUNCTION
EN1	Goes to the pos/speed regulator.
EN2	Goes to the gear box/cam input.
EN3	Free for general use by PL program.
EN4	Free for general use by PL program.

For commutation selection see the `Motor.Comm` member.

BACKWARD COMPATIBILITY NOTE

Previous versions of the DMC firmware had both `Pg.PosOffs` and `RD1.PosOffs` members pointing at the same variable. Thus executing a REFPOS statement changed both `Pg.PosOffs` and `RD1.PosOffs`. This behavior is preserved if the `EN1.Source` has the value 16. It is currently not possible to set the `EN1.Source` (if changed) to 16 by other means that resetting the system.

To update an application to be compatible with the new firmware NOT using the backward compatibility mode you only need to change all references from `RD1.PosOffs` to `Pg.PosOffs`.

FUNCTION

The ENx functions act as a selector switch for redirection of various signals, mainly for feedback purposes. The default settings are set so that backward compatibility with DMC1 is achieved.

EN1-EN4, Encoder1-4

RELATED ITEMS

Gear group

lenc group

ModEn3-4

EXAMPLE USAGE**GROUP MEMBERS (GROUP 64-67)**

Group.Member	Member No.	Range	Ability	Default																		
<i>ENx.Source</i>	6x:0	0..7	RW	See below																		
Default value: EN1: 16 ¹⁾ Goes to the pos/speed regulator. ¹⁾ See Backward compatibility note. EN2: 1 Goes to the gear box/cam input. EN3: 2 Free for general use by PL program. EN4: 3 Free for general use by PL program. Value description: <table border="1"> <thead> <tr> <th>Source</th> <th>Value</th> </tr> </thead> <tbody> <tr> <td><i>RD1</i></td> <td>0</td> </tr> <tr> <td><i>RD2</i></td> <td>1</td> </tr> <tr> <td><i>ENDAT</i></td> <td>2</td> </tr> <tr> <td><i>IENC</i></td> <td>3</td> </tr> <tr> <td><i>Pg</i></td> <td>4</td> </tr> <tr> <td>Reserved.</td> <td>5</td> </tr> <tr> <td>Reserved.</td> <td>6</td> </tr> <tr> <td>Reserved.</td> <td>7</td> </tr> </tbody> </table>					Source	Value	<i>RD1</i>	0	<i>RD2</i>	1	<i>ENDAT</i>	2	<i>IENC</i>	3	<i>Pg</i>	4	Reserved.	5	Reserved.	6	Reserved.	7
Source	Value																					
<i>RD1</i>	0																					
<i>RD2</i>	1																					
<i>ENDAT</i>	2																					
<i>IENC</i>	3																					
<i>Pg</i>	4																					
Reserved.	5																					
Reserved.	6																					
Reserved.	7																					
<i>ENx.Pos</i>	6x:1	±31 bit	RW	Position from selected source.																		
This is the position from the connected sensor.																						
<i>Enx.Speed</i>	6x:2	±31 bit	R	Position from selected source.																		
This is the speed from the connected sensor.																						

IENC

Group	Group No.	Description
IENC	69	Simple incremental encoder interface. This is a simple interface to an incremental encoder.

GENERAL

The DMC² can handle a three channel Incremental Encoder (A, B and Z), with 5 Volt differential outputs. The connector also contains supply voltage for the device.

The incremental feedback can be used for commutating the motor (primary feedback) or as a master signal to the Gearbox (secondary feedback).

FUNCTION

The interface always counts every edge (quadrature counting), which means that an Encoder with 4096 pulses per turn gives 16384 counts per turn.

The Zero pulse (once per turn) triggers the counter to freeze the value on rising edge of the marker pulse. This Zero pulse can also be used to trigger the Capture mechanism, (see page 118 Capture group for details)

RELATED ITEMS

Capture function

EN1 – EN4

EXAMPLE USAGE

Use the Incremental encoder as master to Gearbox functions.

```
En2.Source,3           ;encoder as source to gearbox
Ienc.Mode,1           ;activate
```

GROUP MEMBERS (GROUP 69)

Group.Member	Member No.	Range	Ability	Default
IENC.Mode	0	0..1	RW	0
Turns on or off the incremental encoder interface. 1 = interface is active. See the XENDAT group.				
IENC.Pos	1	±31 bit	R	0
This is the incremental position count from the encoder. This counter counts every edge on the input signals thus an encoder with 5000 pulses/revolution will here give 20.000 pulses/revolution since every edge is counted.				

IENC

<code>IENC.Speed</code>	2	-32767000..32767000	R	0	
	This is the number of counts/second from the incremental encoder.				
<code>IENC.RPos</code>	3	±31 bit	R	0	
	$IENC.RPos = IENC.Pos - IENC.PosOffs$				
<code>IENC.PosOffs</code>	4	±31 bit	RW	0	
	Position offset between <code>IENC.Pos</code> and <code>IENC.RPos</code>				
<code>IENC.IdxPos</code>	5	16 bit	RW	0	
	<p>This register returns the 16 bit value of the up/down counter when the index signal from the encoder went active. The index signal from the incremental sensor shares the connector pin with the trig signal for the <code>CAPTURE</code> group. In order to use the <code>CAPTURE</code> function to detect updates of <code>IENC.IdxPos</code> a user must make sure that the pulse width is within specifications of both the trig signal for <code>CAPTURE</code> and the <code>IENC.IdxPos</code>.</p> <p>See hardware specifications.</p>				
<code>IENC.IncErrors</code>	6	±31 bit	RW	0	
	<p>Error counter for detected errors in the UP/DOWN counter. This quadrature counter is used to count the A and B signals from the incremental encoder. When there is a situation where both input signals change state at the same time, the discriminator in the encoder cannot determine if this should be an UP or DOWN count. In this case the errors counter is incremented.</p>				
i	Since the same hardware counter is used in the XENDAT group, this error counter is also common to both groups.				

MODEN3-MODEN4

Group	Group No.	Description
ModEn3-4	72-73	Modula calculation on EN3 and EN4

GENERAL

In order to make periodical systems the ModEnx can be set up map an external position signal (or the Profile generator output) to a machine period. This can be used for synchronizing purposes.

FUNCTION

Calculates a periodic position from the output of En3 and En4

It is possible to reset the counters at any time. A capture function is provided to take snapshots of the counter values based on a software trig or the hardware capture function (See the Capture group, page 118). The r

RELATED ITEMS

En3, En4

EXAMPLE USAGE

Set up matching periods from Resolver 1 and Resolver 2(Master/Slave scenario) where the machine has a period of 250 000 pulses on the slave axis and 600 000 pulses on the master axis.

```
Gear.In,250           ;this will map
Gear.Out,600         ;one period of the
Gear.Mode,1          ;master to one period
Pos Mod On Clr       ;of the slave.

En3.Source,0         ;Rd1 as source
En4.Source,1         ;Rd2 as source

ModEn3.Module,250000 ;set period on Rd1
ModEn4.Module,600000 ;set period on Rd"
ModEn3.mode,1        ;activate
ModEn4.mode,17       ;activate with ganged reset
```

GROUP MEMBERS (GROUP 72-73)

Group.Member	Member No.	Range	Ability	Default
ModEN3.Mode	0	0..255	RW	0

ModEn3-ModEn4

<p>Controls various aspects of this group.</p> <p>Bit0 (1) - Enable.</p> <p>Bit1 (2) - Enable hardware capture into CapCount and CapPos.</p> <p>On hardware capture, see Capture group, the CapPos and CapCount will be updated.</p> <p>Bit2 (4) - Reserved.</p> <p>Bit3 (8) - Reserved.</p> <p>Bit4 (16) - Ganged reset of ModEN3 and ModEN4 simultaneously.</p> <p>Bit5 (32) - Ganged capture as b6 but done for all ModEN groups simultaneously.</p> <p>Bit6 (64) - Manual capture of ModEN3.Pos and ModEN3.Count into ModEN3.CapPos and ModEN3.CapCount.</p> <p>Bit7 (128) - Reset ModEN3.Pos and ModEN3.Count to zero.</p> <p>When writing to the Mode member the bits are evaluated in this order:</p> <p>Bit6 Manual capture</p> <p>Bit5 Ganged capture</p> <p>Bit7 Reset</p> <p>Bit4 Ganged reset</p> <p>This allows for setting many bits at the same time.</p> <p>Note. Only bit b0 and b1 are present when read.</p>				
<code>ModEN3.Module</code>	1	32 bit	RW	0
<p>The position range of ModEN3.Pos. This value is always positive. (The period for this counter)</p>				
<code>ModEN3.Count</code>	2	±31 bit	RW	0
<p>Number of period since last reset.</p>				
<code>ModEN3.Pos</code>	3	32 bit	RW	0
<p>The current position within tModEn3. Module. This position is always positive. A user can write a new position and the value will automatically be fitted within the period. ModeEn3.Pos and ModEn3.count will be updated. Example: If ModEn3.Module = 10000 and ModEn3.Mode = 1 an attempt to write the value 1234561 to ModEn3.pos will read back as</p> <p>ModEn3.Pos = 3456</p> <p>ModEn3.Count = 12</p>				

ModEN3.CapCount	4	±31 bit	RW	0
	Captured value of ModEn3.Count			
ModEN3.CapPos	5	32 bit	RW	0
	Captured value of ModEn3.Pos			

Communication protocol

INTRODUCTION

THE BASIC DMC² communication protocol follows the Intel HEX-protocol with some extensions. The register number field may also contain X-REGISTERS for the read and write register routines.

The ACK/NAK characters are moved to some characters that are not part of the data character set.

The RECORD-TYPE byte high nibble values 8 -- F may be used to force unit addressing to eliminate response from a miss-selected unit.

PROTOCOL FORMAT

The general format for the protocol

```
: <Len><Addr><Type><Data #0> ... <Data #Len-1><Csum><CR>
```

DESCRIPTION

:	The start of record character.
<Len>	A two-digit hexadecimal number, indicating the length of the data field (in bytes) in the record. The maximum supported value of <len> is 6F hex.
<Addr>	A four-digit address, Register are addresses with their number (0-255). Extended registers are addresses with $[32768+256 \times group+member]$.
<Type>	Type of record and node address, if command or data record, and type of command/data. The first Hex-Digit is the node address (in daisy chain operation) and the second digit is the record type identifier.
<Data>	Is the data field.
<Csum>	Is the 2's complement of the modulo 256 sum of all bytes in the record (except <Csum>). (So that the modulo 256 sum off all bytes, including <Csum> is Zero.)
<CR>	Carriage return used to help identify the end of the record.

A record with correct checksum is acknowledged with an "Y" followed by a <CR>; a record with incorrect checksum is acknowledged with an "N" followed by a <CR>. These rules apply to both the host computer and the DMC² unit.

COMPUTER MODE

Record type	Description	Example
0	Ordinary data record Used to download/upload the internal program. <Addr> is the byte-address in the internal program area. For the DMC ² to accept this record type, a record of type 2 must have been sent to enable program downloading.	

Computer Mode

Record type	Description	Example
1	<p>End of data record. Is sent after a program has been completely up/downloaded and disables download of program code.</p>	:00000001FF<CR>
2	<p>Prepare for download. Stops Execution initializes the program area and enables download of program code.</p>	:00000002FE<CR>
3	<p>Stop program execution.</p>	:00000003FD<CR>
4	<p>Start execution at first line of program.</p>	:00000004FC<CR>
5	<p>Set Terminal Mode. The same as the statement [LET] Comm.Mode = 0. Used to exit the Computer Mode.</p>	:00000005FB<CR>
6	<p>Upload all programs Memory. The contents of the program memory are sent as type 0 records, ending with a type 1 record. Only Non-empty records are sent;.</p>	:00000006FA<CR>
7	<p>Read system tables. This record type can be used by the PL compiler or similar program to determine what commands are supported by the current version of the DMC²firmware. Current Format (Addr Contents): 0 DMC Version number (0 for all Ver 0.XX) 1 TBLROOT -- Pointer to pointer to Command name definition tables. 2 XGWPTR -- Pointer to write pointer in X-group table 3 XGROUP -- Pointer to base of X-group table 4 XGDISTAB - Pointer to Xgroup R/W dispatch table 5 1F -- To be defined later -- If the <Len> field is >= 1, then the <ADDR> field is a 16-bit address in the firmware memory. The first data byte indicates the number of bytes of firmware memory to return starting at <ADDR>. It is not possible to read addresses 0..1F.</p>	
8	<p>Get a register value. <Addr> Specifies the number of the register to get. Valid numbers are from 0 to 255 or an Xregister descriptor. The value of the register is returned in a record of type 8, with a <len> field equal to 4.</p>	:00<Addr>08<Csum><CR> >
9	<p>Set one or more registers. <Addr> Specifies the first register to set. <Len> Len * 4 specifies the number of registers to load with the following data. Four bytes of data are required for each register. If the PL program is executing time-critical code,</p>	

Record type	Description	Example
	do not load more than a few registers in each record. A longer record implies that the PL program is halted a longer time while the registers are loaded from the internal communication buffers to the register area. This is particularly critical when loading Xregisters.	

Ex: Read Member 2 in Group 15 at node 13

```

<ctrl Z> D           ; select node 13
:008F02D897          ; Addr = 32768+15 x 256+2 (8F02)
                     ; Type = D for node and 8 for record type
                     ; Csum = 97 (0 -(8F+02+D8))

```

Ex: Write 23 (dec) to register 11 at node 7

```

<ctrl Z> 7           ; select node 7
:04000B790000001761 ; Len = 4
                     ; Addr = 11 (dec) (B)
                     ; Type = 7 for node and 9 for record type
                     ; Data = 17 (hex)
                     ; Csum = 61 ( 0-(04+0B+79+17))

```


PL2 On line commands

PL2 ON LINE COMMANDS

<code>LIST</code>	<code>[Line1 [,Line2]]</code>	If no argument, lists all lines in the program. If Line1 is given as a single argument, then list Line1. If both Line1 and Line2 are given as arguments then list all lines from Line1 to Line2.
<code>HLIST</code>	<code>[Line1 [,Line2]]</code>	Same as LIST; also, list the instruction code in HEX before the line.
<code>TLIST</code>	<code>[Lines]</code>	Lists the trace buffer on to the serial communication port. If an argument line is given, the listing will include only the last line number of lines in the buffer. See TRACE ON/OFF/CONT. The first displayed line's time field is taken as reference for all subsequent lines in the buffer. The time field will indicate time passed since the first displayed PL line was executed. The time is in milliseconds.
<code>DEL</code>	<code>Line1 [,Line2]</code>	Delete Line1. If both Line1 and Line2 are given as an argument, then delete all lines from Line1 to Line2.
<code>RUN</code>	<code>[Line]</code>	Start the execution of instructions at line Line.
<code>NEW</code>		Erase all program memory.
<code>CONT</code>	<code>[Line]</code>	Continue execution after a STOP program statement or a Control-C break.
<code>GDISP</code>	<code>XReg</code>	Display all elements in the group XReg resides in. Xreg may be abbreviated to GROUP. name instead of the complete GROUP.MEMBER notation. Example: To display all elements in the group that <code>Pg.Rdy</code> resides in: <code>GDISP Pg.Rdy</code> To display the setting of the regulator (Note the dot after the group name): <code>GDISP REG.</code>
<code>HELP</code>		Show the firmware revision and lists all currently available ON LINE COMMANDS, PROGRAM STATEMENTS and X-REGISTER groups.
<code>FLoad</code>		When you download PL code to the drive, the code is not automatically stored to nonvolatile memory.
<code>FStore</code>		Must be issued manually after a download to store the code in nonvolatile memory. Note: The motor must be turned off before
<code>Boot</code>		Forces the drive into boot mode. Boot mode enables a user to download new firmware.
<code>Status</code>		Displays the active connections made with the connect statement
<code>Control+C</code>		Control key + the "C" character. Will stop the execution of instructions in the controller. Note. It will not stop any motion is taking place. Using

Command Line Editor (CLE)

Vector.CtrlC can stop motion.

Example:

```
10      Vector.CtrlC, 300
20      Int.SysMask = Int.SysMask or 4xx.. Other
code
300      Pos Abort
; Abort any motion          profile.
301      PG.Speed 0          ; Stop any motion.
302      End
```

Control+Z *Unit-Address* Control key + the “Z” character. Select the unit to communicate with. The Unit-Address is the characters “1”-“9” and “A”-“F”.

Control+T Control key + the “T” character. Display a snap shot of the controller status.

COMMAND LINE EDITOR (CLE)

User interaction in terminal mode has been improved with a full online editor. Also a circular buffer that stores the last used command line has been added. This enables a user to pick any command that already has been entered, from a list, to edit or execute. This will speed up interactive user sessions. The keys needed to navigate the circular buffer can be mapped so that the PC arrow keys can be used.

To manipulate the command line from the keyboard, commands are given to the CLE as single ASCII characters. Holding the CONTROL key down before the key is pressed can generate all these commands, this will generate a single ASCII character in the range 0 to 31. The mapping of commands to special keys are made in the Promoton.ini file in the ECT directory.

Command name	ECT mapping	Value	Control key	Description
..		0	+@	..
bol	KEY_HOME	1	+A	Goto beginning of the line.
bck	KEY_LEFT	2	+B	Backup one character.
stp		3	+C	Stop execution of PL code.
delf		4	+D	Delete one character forward.
eol	KEY_END	5	+E	Goto end of the line.
fwd	KEY_RIGHT	6	+F	Go one char forward.
..		7	+G	..
delb	BS	8	+H	Delete one char backward.
tab	KEY_INSERT	9	+I	Insert one space.
nop		10	+J	No action at all.
kill		11	+K	Delete rest of line.
..		12	+L	..
done	RETURN	13	+M	Line completed!
next	KEY_DOWN	14	+N	Display next line.
..		15	+O	..

Command name	ECT mapping	Value	Control key	Description
prev	KEY_UP	16	+P	Display previous line.
..		17	+Q	Reserved, Xon/Xoff protocol.
..		18	+R	..
..		19	+S	
ctlt		20	+T	
junk	KEY_ESCAPE	21	+U	
..		22	+V	
delwb	KEY_DELETE	23	+W	Delete word backward.
..		24	+X	..
yank		25	+Y	Yank killed data back.
..		26	+Z	..
..		27	+Esc	..
..		28	+\ +]	..
..		29	+]	..
..		30	++	..
..		31	+ -	..

ECT

INTRODUCTION

The ECT is a Windows-based application used for EDITING, COMPILING AND TESTING of user-created source code to control the DMC² Motion Controller. Each of these functions is available through the ECT application Main Menu window.

- The EDITOR is full-featured ASCII text editor that allows the user to create and edit source code. This source code can then be compiled using the ECT Compiler, and tested using ECT's Test application.
- The COMPILER is a tool that translates the source code into native DMC² executable code; also referred to as hex code. When selected, source code in the active Edit window is automatically compiled.
- TEST is a tool for debugging user-written/edited DMC² source code.

DEFINITIONS

Source code	A collection of organized DMC ² instructions that is recognized by the compiler, together with compiler directives and any library functions subsequently added and made available to the user.
Library functions and macros	Inmotion Technologies AB written or user-written code segments that perform encapsulated functions. Macros may be saved as independent text files, and 'included' through the Editor for compilation.

RUNNING ECT

When ECT starts, the user is presented with a complete environment for editing, compiling, and testing PL2 source code.

To start ECT, select the ECT icon from the Windows Program Manager screen. Once loaded, ECT displays the ECT desktop.

THE ECT DESKTOP

The following lists the major components of the ECT desktop:

Application Caption Bar:	Displays "ECT" and the name of the active Application.
Menu Bar:	Contain a list of user-selectable menus that include commands to instruct ECT to perform actions.
Speed Bar:	When used with a mouse, provide instant access to frequently used ECT commands.
Status Bar:	Displays information at the bottom of the ECT desktop about the selected menu bar command.
Edit Window:	Used for creating and/or editing source code files.
Compile Window:	A text window that reports compiler errors and/or warnings.
Test Window:	A text window that reports motion controller response.

THE ECT MAIN MENU

When ECT is loaded, the main menu appears as shown in [Figure 9](#) below.

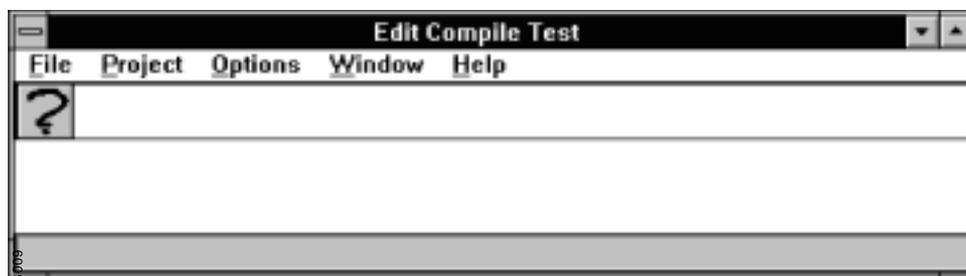


Figure 9. The ECT main menu.

The ECT Main Menu offers the following items:

FILE	PROJECT	OPTIONS	WINDOW	HELP
------	---------	---------	--------	------

The following lists the selections available within each Main Menu item.

FILE

File allows the user to work with (edit and create) PL2 source files. Menu selections include:

NEW	OPEN	SAVE	SAVE AS	PRINT	PRINTER SETUP	EXIT
-----	------	------	---------	-------	---------------	------

PROJECT

The Project function allows the user to store and/or retrieve a collection of related files and settings into a Project file. Selections within this item include:

NEW	OPEN	CLOSE	SAVE	SAVE AS	DEL ITEM	ADD ITEM	OPEN ITEM
-----	------	-------	------	---------	----------	----------	-----------

OPTIONS

The Options item allows the user to configure/setup various components of ECT for their particular application. Menu selections include:

FONT	PREFERENCES	TEST SETUP	COMPILER SETUP
------	-------------	------------	----------------

WINDOW

Window contains selections that allow the user to control the ECT desktop. It also includes selections for various ECT components. When selected, this item displays the following choices:

TILE	CASCADE	ARRANGE ICONS	NEW EDITOR	PROJECT	TERMINAL	MESSAGE
------	---------	------------------	---------------	---------	----------	---------

HELP

ECT Help is available through this item. Setup in the Windows-typical Help format, it offers the following selections:

INDEX	SYNTAX	USING HELP	ABOUT
-------	--------	------------	-------

USING THE TEXT EDITOR

The following is a list of the functions and their descriptions available when editing text within the various ECT components.

Moving within the window

Key (+ Key)	Function
Up Arrow	Moves up one line.
Down Arrow	Moves down one line.
Right Arrow	Moves right one character.
Left Arrow	Moves left one character.
Ctrl+Right Arrow	Moves right one word.
Ctrl+Left Arrow	Moves left one word.
Home	Moves to the beginning of the line.
End	Moves to the end of the line.
PgUp	Moves up one window.
PgDn	Moves down one window.
Ctrl+Home	Moves to the beginning of the document.
Ctrl+End	Moves to the end of the document.

SELECTING TEXT

Key (+ Key)	Function
Shift+Left or Right Arrow	Selects text one character at a time to the left or right. If the character is already selected cancels the selection.
Shift+Down or Up	Selects one line of text up or down. Or, if the line is already selected cancels the selection.
Shift+PgUp	Selects text up one window. Or, if the previous window is already selected cancels the selection.
Shift+PgDn	Selects text down one window. Or, if the next window is already selected cancels the selection.
Shift+Home	Selects text to the beginning of the line.
Shift+End	Selects text to the end of the line.
Ctrl+Shift+Left Arrow	Selects the previous word.
Ctrl+Shift+Right Arrow	Selects the next word.
Ctrl+Shift+Home	Selects text to the beginning of the document.
Ctrl+Shift+End	Selects text to the end of the document.

EDIT

Edit is the application that allows source code to be created and/or edited, then saved and compiled. It utilizes a fully functional ASCII text editor to assist the user

Edit

in generating this code. For a description of the text editing functions, refer to "Using the Text Editor".

ACCESSING THE EDIT APPLICATION

There are several ways in which to access the Edit application when creating new source code:

- From the ECT Main Menu select:
 - File
 - New

The ECT Main Menu Window will now display an "edit" window as shown in [Figure 10](#) below.

Note that the Menu Bar change

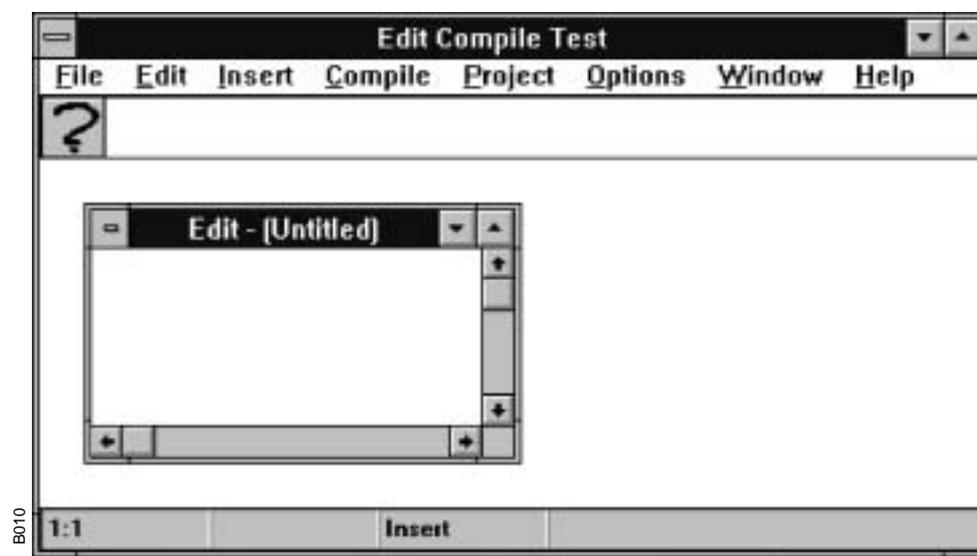


Figure 10. The ECT main menu window displays an "edit" window.

- At the ECT Main Menu window, select
 - Window
 - New Editor

This creates a new "edit" window and changes the Menu Bar as described in No. 1 and shown in [Figure 10](#). Using this method to create a new source file requires that the File/Save As. menu selections be used to save the file.

WHEN EDITING EXISTING FILES

Accessing the Edit application to work with an existing source file is accomplished from the ECT Main Menu window by selecting:

- File
- Open

A box as shown in [Figure 11](#). is displayed. Enter or select from the list, the file to be edited. Note that the content of the file can be viewed before an edit window is created.

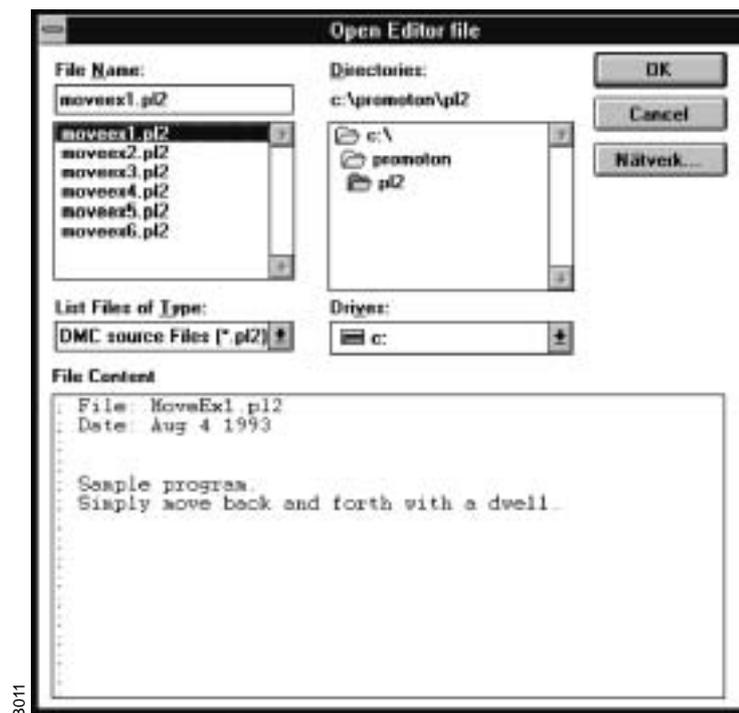


Figure 11. Open an Editor file.

Once the file is loaded, the ECT desktop will appear as shown in [Figure 12](#).

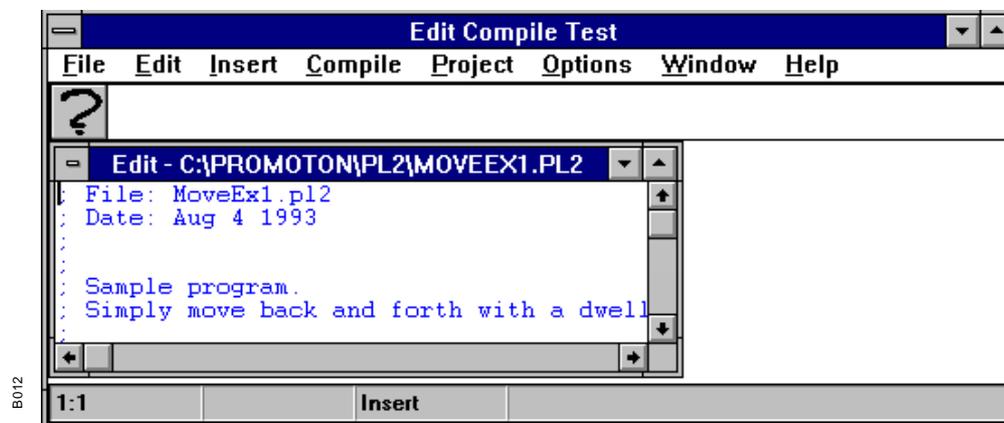


Figure 12. The ECT desktop with file to be edited.

CREATING/EDITING SOURCE CODE

While some functions within the ECT generate source code automatically for "inclusion" into a source file, user-written DMC² source code is governed by the rules, syntax, commands and statements of the PL language as described in the DMC² Language Description Manual.

Compile

EXITING EDIT

To exit Edit and return to the ECT Main Menu window, Save and Close all open editor windows.

COMPILE

INTRODUCTION

The ECT Compile application compiles the source code in the active edit window. The result of a "clean" compile is executable code for the motion controller. The compiler also generates a list file with cross-references and a list of variables and constants used in the program.

The ECT compiler performs a number of functions

- Preprocessing.
Involves include files, macro definition and expansion, and conditional compilation.
- Lexical analysis.
Recognizes different categories of word-like units, referred to as tokens.
- Phrase structure grammar.
Details the rules by which tokens can be grouped together to form expressions, statements, and other significant units.

ACCESSING COMPILE

Compile is accessed through the Edit application. When Compile is selected from the Edit application speed bar, the text in the active edit window is automatically compiled. However, prior to compiling source code, verify the compiler settings are correct. Accessing the compiler setup window is described in the following section.

SETTING UP THE COMPILER

Prior to editing and compiling source code, verify the compiler settings are correct. At the ECT Main Menu window, access the compiler setup window by selecting:

- Options.
- Compiler Setup.

The window as shown in [Figure 13](#). is displayed.

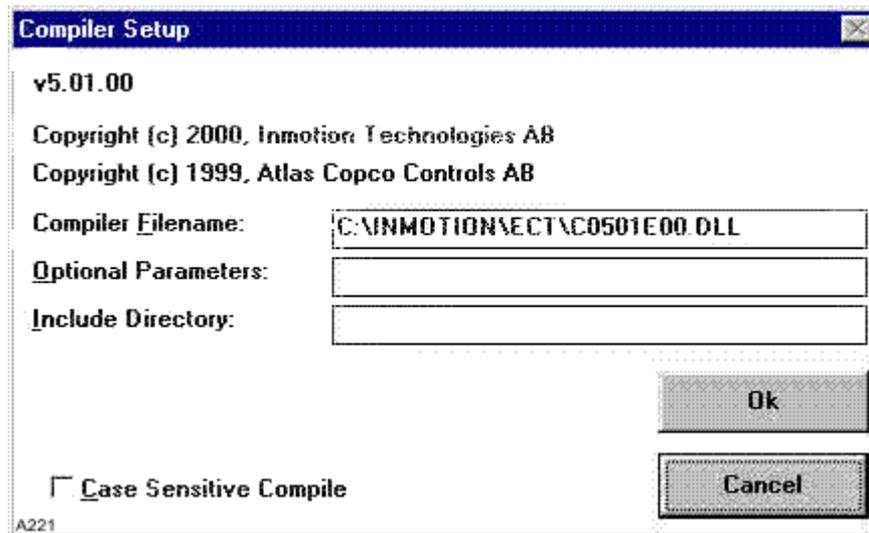


Figure 13. The Compiler setup window.

COMPILING SOURCE CODE

While in the active edit window, access the compiler as described above. While the source code is being compiled, a window such as that had shown in [Figure 14](#). is displayed. This window identifies the file being compiled and the error/warning status of the source code as it is being compiled.

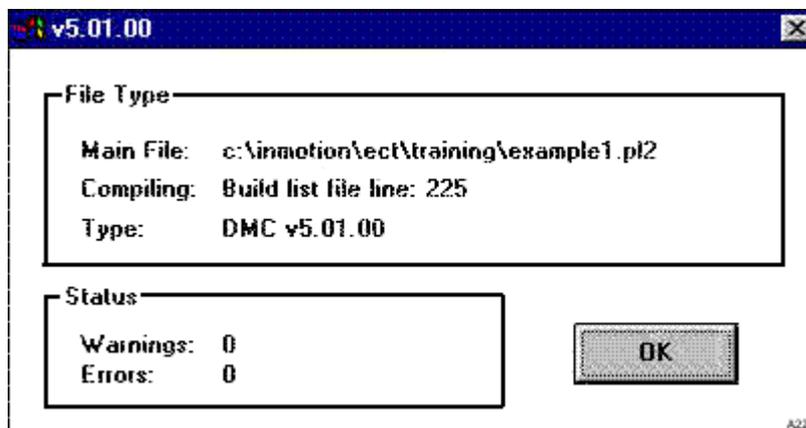


Figure 14. The compiled file.

When the compiling is complete, click on the OK button to display the output

COMPILER OUTPUT

Output, in the form of errors and/or warnings are displayed in the Compiler "Messages" window ([Figure 15](#).). Errors and warnings are reported back to the active Edit window, which then sequentially highlights the first faulty source code

Test

line, assisting the user to identify and correct syntax errors. Double-click on a message line and you will be taken to that line in the source code.

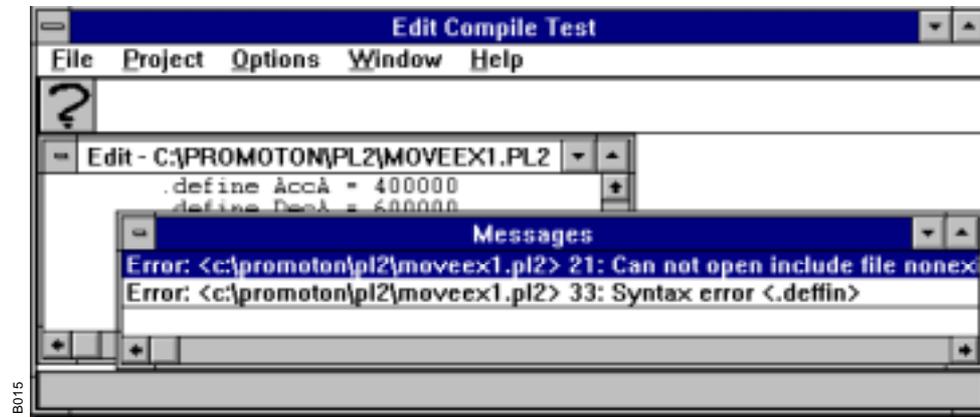


Figure 15. Compiler “message” window.

The message example shown in Fig.7. is described as follows:

<pre> Error: <C:\promoton\pl2\moveex1.pl2> 33: Syntax error Error: <C:\promoton\pl2\moveex.pl2> 33: Syntax error <.deffin> </pre>	<pre> Category: Error or Warning Drive, path and filename of compiled file. Line number of offending statement in the edit file Error type (description) Offending statement </pre>
---	---

TEST

INTRODUCTION

The Test program provides the user with a complete environment for testing PL2 source code.

TEST SETUP

Prior to testing source code, set up the test environment from the ECT main menu by selecting:

- Options
- Test Setup

Windows like that shown in [Figure 16](#). is displayed. This window includes the following setup options:

- Communication Method
- Test Device
- Function Key Setup
- Monitor Setup

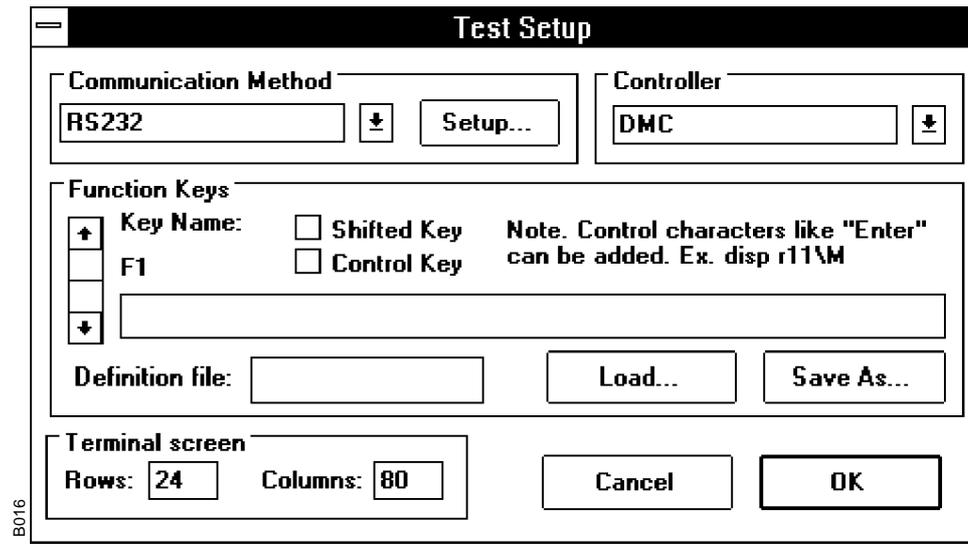


Figure 16. Test Setup window.

Clicking the Setup button in this window displays the window illustrated in [Figure 17](#). This window is used to establish communications parameters such as baud rate, port etc.

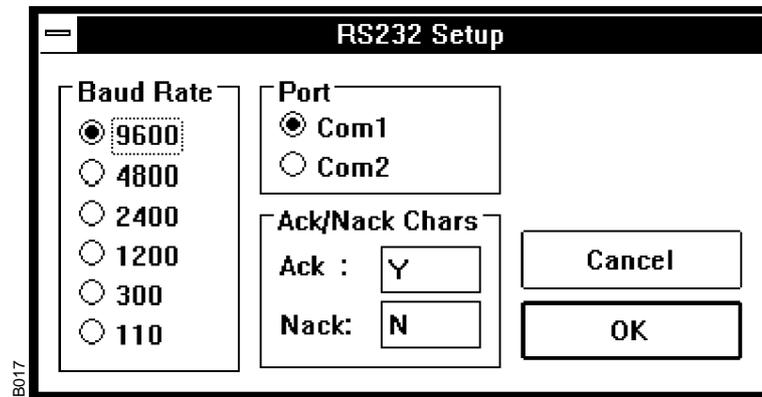


Figure 17. The communication parameters.

TEST FUNCTION

The user can connect to the DMC² controller using different hardware and communication devices:

- Serial Port.
- CAN network (future).

The communication device is selected using the method described in the previous section, Test Setup. Once the link is selected, it is active until the test session terminates. It cannot be changed from within the TEST environment. The selected communication device is responsible for managing the communications protocol,

Test

including the upload and download of information to and from the target controller. Users can:

- Change values of registers.
- Enter and patch code in the DMC² without having to recompile and download to the controller.

ACCESSING THE TEST SYSTEM

The Test system is accessed from the ECT main menu by selecting:

- Window
- Terminal

This displays a new menu bar as well as a terminal window like that shown in [Figure 18](#).

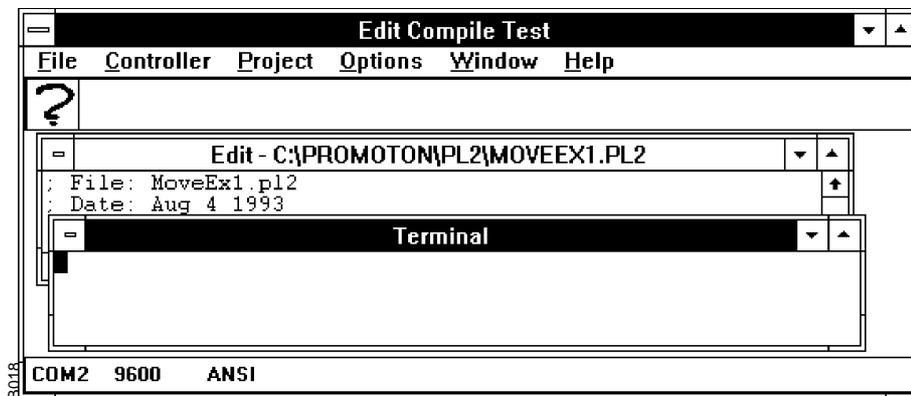


Figure 18. Terminal window.

The Test Window automatically attempts to connect with the DMC² controller through the selected communication device. If this fails, a dialog box indicating this is displayed. The file in the active Edit Window internally informs the Test Window of the current source file. If there is no active Edit Window, a file must be opened.

TEST MENU

The Test menu appears when there is an active Terminal window on the desktop. This menu includes the following selections:

FILE CONTROLLER PROJECT OPTIONS WINDOW HELP

The following menu selection descriptions are those that directly relate to the Test environment.

FILE

Create or open an editor window. See [Edit](#)

CONTROLLER

The Controller selection controls the DMC² controller. Selections within this menu item allow the user to download a program, start and stop the controller.

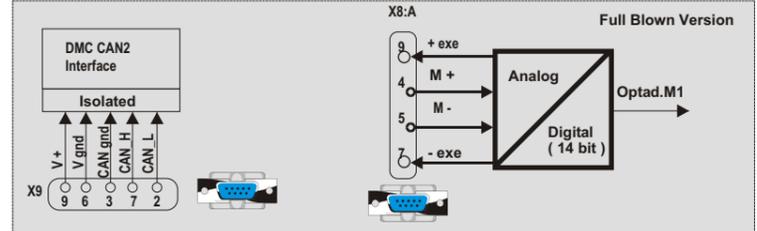
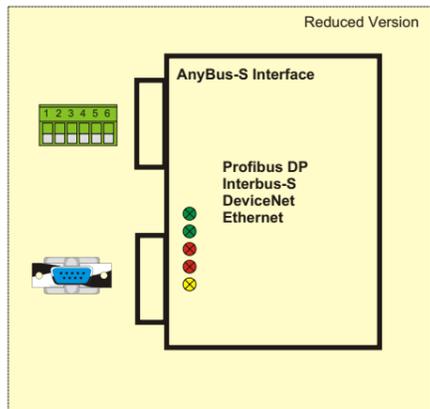
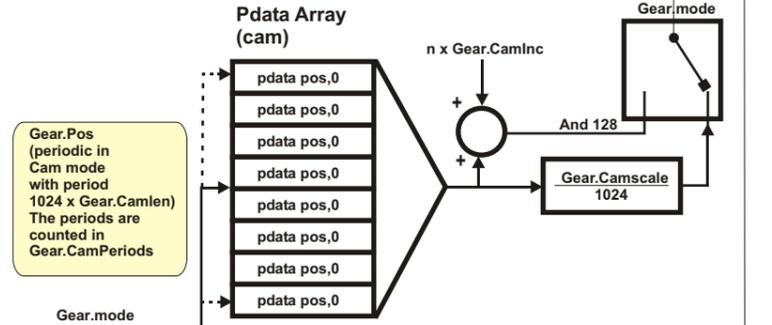
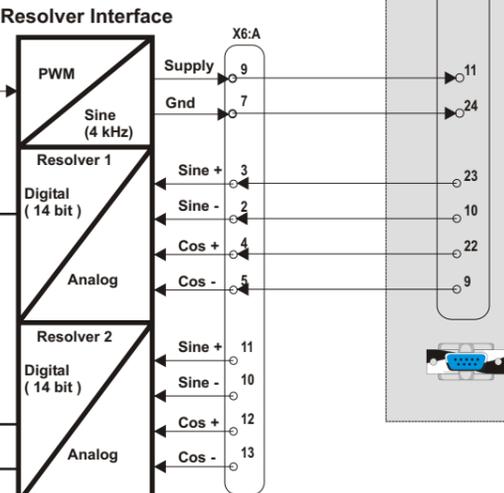
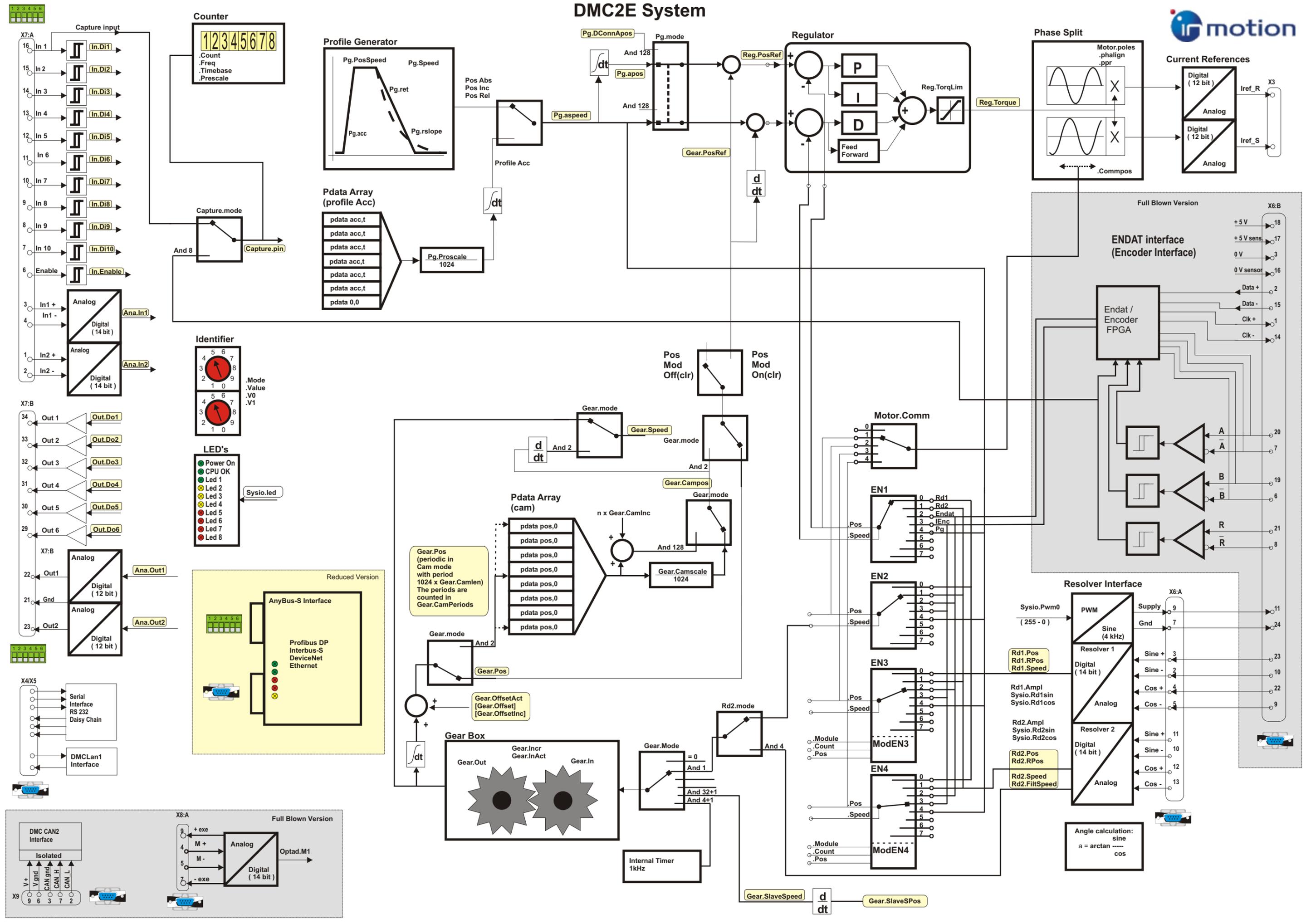
Start Starts execution of the application program that is

	resident in the DMC ² controller.
Stop	Halts execution of the application program that is resident in the DMC ² controller. CAUTION: STOP does not halt any on-going motion; only the execution of PL2 code is halted.
Download	Use to download a previously compiled HEX file to the connected DMC ² controller. Program name, version and compiled date are also loaded. The file loaded will be either the compiled version of the file in the active Edit Window, or that which was opened by the user. Note: Any application program in the DMC ² controller prior to downloading will be overwritten.
Force Terminal Mode	Force controller into terminal mode, in case the controller was left in computer mode.
Snap Shot	Display a snap shot of the current execution state within the controller.
Unit Select	Select the controller (unit address) that will be connected to the terminal for communication. Up to 15 units can use the same physical wires, but only one unit can be connected to the terminal at a time. Note: All connected units must have unique unit addresses.
Send File	For transmitting text files to a controller or an EPROM programmer.
Capture	Records received and transmitted characters in a text file. The capture file is closed when the terminal window is closed or when this menu item is activated again. The status bar indicates the capture file name when the capture function is active.

EXITING THE TEST ENVIROMENT

To exit the Test environment, close all active windows, saving them as necessary.

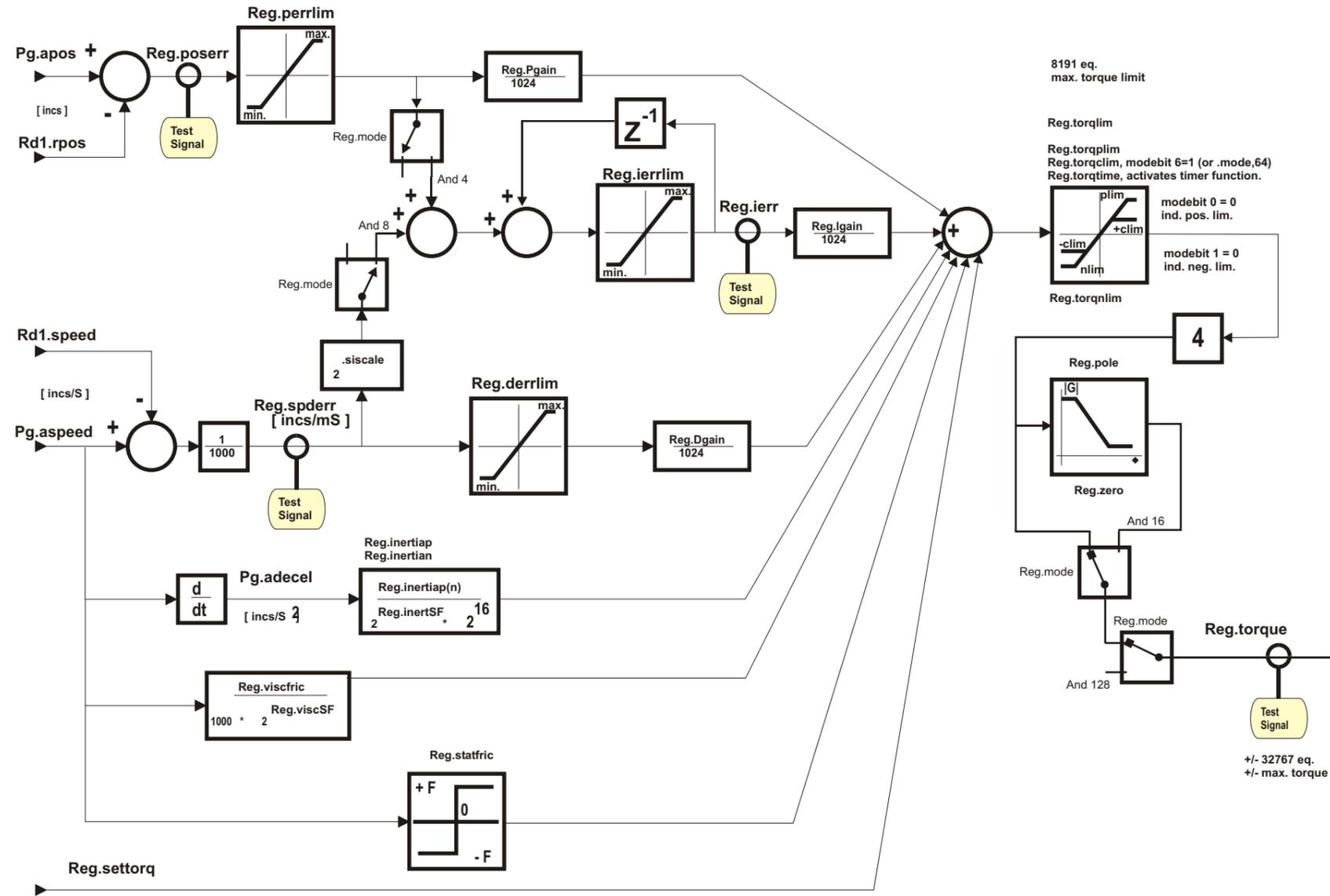
DMC2E System



Angle calculation:

$$a = \arctan \frac{\text{sine}}{\text{cos}}$$

DMC Position Controller



DMC PL2 SW Flow

