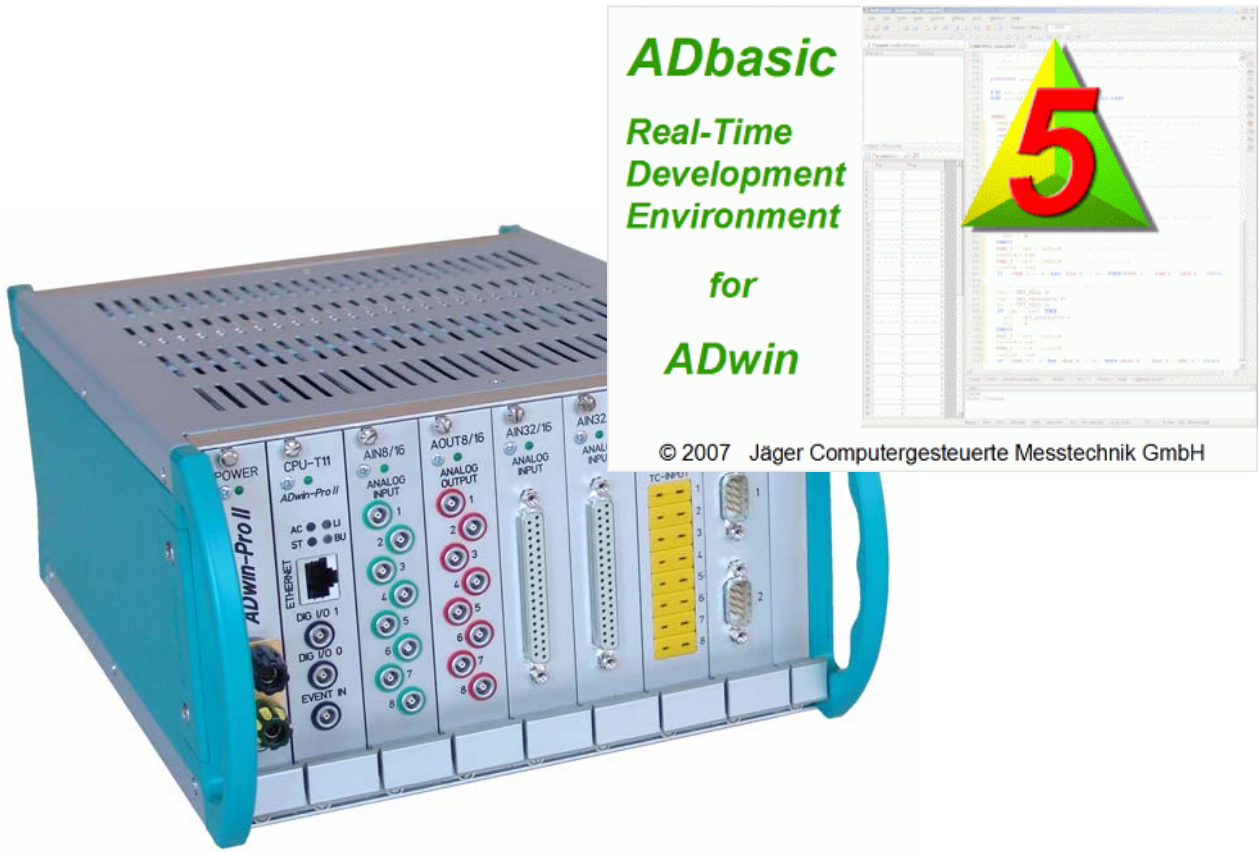
JÄGER
Computergesteuerte
Messtechnik GmbH

ADwin-Pro II

System specifications
Programming in *ADbasic*



ADwin-Pro II software, Manual March 2014

For any questions, please don't hesitate to contact us:

Hotline:	+49 6251 96320
Fax:	+49 6251 5 68 19
E-Mail:	info@ADwin.de
Internet	www.ADwin.de

JÄGER
Computergesteuerte
Messtechnik GmbH

Jäger Computergesteuerte
Messtechnik GmbH
Rheinstraße 2-4
D-64653 Lorsch
Germany

Table of contents

Table of contents III
Typographical Conventions IV
1 Introduction 1
2 The Program ADpro.exe 3
3 *ADbasic* instruction for *ADwin-Pro II* modules 4
3.1 Pro II: All Modules 4
3.2 Pro II: Digital Channels of CPU Modules 21
3.3 Pro II: Multi-I/O 26
3.4 Pro II: Analog Input Modules 35
3.5 Pro II: Output Modules 127
3.6 Pro II: Digital I/O Modules 148
3.7 Pro II: Counter Modules 187
3.8 Pro II: PWM Output Modules 214
3.9 Pro II: Temperature Measuring Modules 225
3.10 Pro II: CAN bus Modules 243
3.11 Pro II: RSxxx Modules 265
3.12 Pro II: LIN bus Interface 279
3.13 Pro II: Profibus interface 292
3.14 Pro II: MIL-STD-1553 bus Interface 296
3.15 Pro II: ARINC-429 bus Interface 306
3.16 Pro II: EtherCAT interface 321
3.17 Pro II: FlexRay 331
3.18 Pro II: SENT Interface 339
3.19 Pro II: SPI Interface 384
4 Program Examples 427
4.1 Online Evaluation of Measurement Data 427
4.2 Digital Proportional Controller 427
4.3 Data Exchange with DATA arrays 428
4.4 Digital PID Controller 428
4.5 Examples for RS232 and RS485 (Pro II) 431
4.6 Continuous signal conversion 435
Instruction Lists A-1
A.1 Alphabetic Instruction List A-1
A.2 Instruction List sorted by Module Types A-4
A.3 Thematic Instruction List A-19

Typographical Conventions

"Warning" stands for information, which indicate damages of hardware or software, test setup or injury to persons caused by incorrect handling.

You find a "note" next to

– information, which absolutely have to be considered in order to guarantee an error free operation.

– advice for efficient operation.

"Information" refers to further information in this documentation or to other sources such as manuals, data sheets, literature, etc.

<C:\ADwin\ …>

File names and paths are placed in <angle brackets> and characterized in the font `Courier New`.

`Program text`

Program commands and user inputs are characterized by the font `Courier New`.

`Var_1`

Source code elements such as commands, variables, comments and other text are characterized by the font `Courier New` and are printed in color.

Bits in data (here: 16 bit) are referred to as follows:

Bit No.	15	14	13	…	01	00
Bit value	2^{15}	2^{14}	2^{13}	…	2^1=2	2^0=1
Synonym	MSB	-	-	-	-	LSB

1 Introduction

The real-time development tool *ADbasic* is a software that on the one hand is a means for easy programming of the *ADwin-Pro II* processor system and on the other hand is a tool that completely uses the multi-processing capacities of the system.

This manual describes *ADbasic* instructions to access the variety of modules. (see annex for the Instruction List sorted by Module Types).
There is also the *ADbasic* manual which describes the more basic command e.g. for calculations, for program structure or for process control.

The commands for access to the *ADwin-Pro II* system with *ADbasic* are included in the include files. After installation from the *ADwin* CD-ROM the include files are available in the directory <C:\ADwin\ADbasic\inc>.

In order to get access to the *ADwin-Pro* and *ADwin-Pro II* modules you include all required include files with the following line into your *ADbasic* program.

```
#INCLUDE ADwinPro_All.inc
```

If you have already written *ADbasic* programs, you have used a separate include file for each module group. Delete all these include lines completely and insert the upper line instead.

Please note:

For *ADwin* systems to function correctly, adhere strictly to the information provided in this documentation and in other mentioned manuals.

Qualified personnel

Programming, start-up and operation, as well as the modification of program parameters must be performed only by appropriately qualified personnel.

> *Qualified personnel are persons who, due to their education, experience and training as well as their knowledge of applicable technical standards, guidelines, accident prevention regulations and operating conditions, have been authorized by a quality assurance representative at the site to perform the necessary acivities, while recognizing and avoiding any possible dangers.*
> *(Definition of qualified personnel as per VDE 105 and ICE 364).*

Availability of the documents

This product documentation and all documents referred to, have always to be available and to be strictly observed. For damages caused by disregarding the information in this documentation or in all other additional documentations, no liability is assumed by the company *Jäger Computergesteuerte Messtechnik GmbH*, Lorsch, Germany.

Legal information

This documentation, including all pictures is protected by copyright. Reproduction, translation as well as electronical and photographical archiving and modification require a written permission by the company *Jäger Computergesteuerte Messtechnik GmbH*, Lorsch, Germany.

OEM products are mentioned without referring to possible patent rights, the existence of which, may not be excluded.

Subject to change.

Hotline address: see inner side of cover page.

The Program ADpro.exe

2 The Program ADpro.exe

The program tool <ADpro.exe> has several taks:

– Show the modules on an *ADwin-Pro II* system a well as information on the modules.

– Set the module address for Pro II modules (see hardware manual).

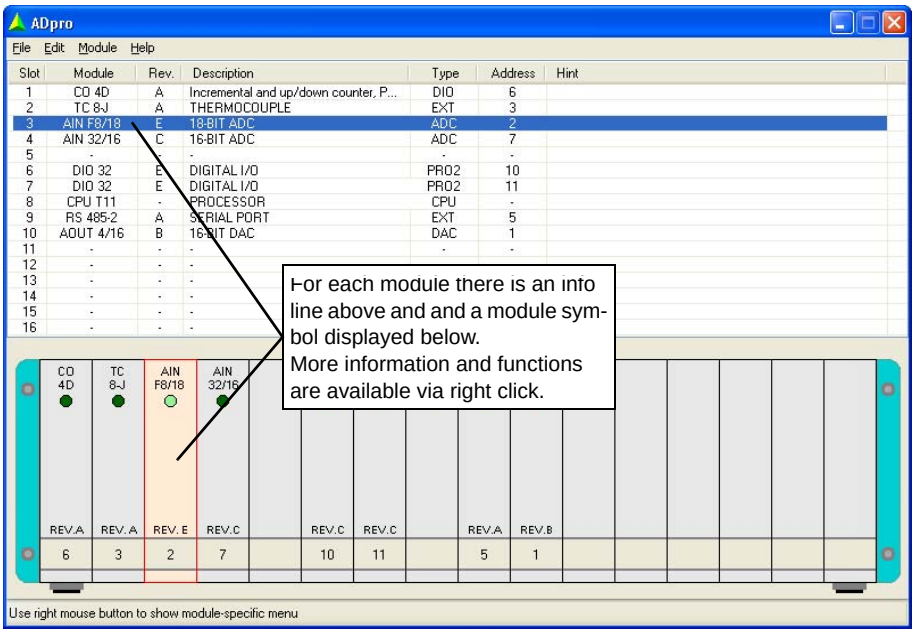
  With Pro I modules the module address is set manually; here the address can be shown only.

– Check the function of Pro I and Pro II modules: analog input / output modules, digital and counter modules, some bus modules.

– Calibrate Pro I and Pro II modules (analog input / output modules).

  The calibration meets lower demands only.

The use of the program ADpro is self-explanatory; some functions are available via context menu (right mouse click). Please note the accompanying text and follow the hints given.



Notes

ADpro.exe initializes the *ADwin* system, thus ending and deleting still running processes.

If there is an error upon start-up of the program, please check if the software packet <Microsoft .NET Framework 2.0> is installed on the PC.

The recognition of module types, one of the tasks of ADpro.exe, is sometimes requested as function in *ADbasic*, too. But it appeared that the user's organisational effort exceeds the benefits by far: the required module information would have to be regularly updated, evaluated, and added manually to the *ADbasic* source code. Therefore, the function „recognition of module types" is only available in ADpro.exe, but not in *ADbasic*.

3 *ADbasic* instruction for *ADwin-Pro II* modules

This section contains all instructions to access *ADwin-Pro II* modules. The instructions are sorted according to module groups and then alphabetically.

In the annex you find furthermore the following sorted instruction lists:

– Alphabetic Instruction List (see annex A.1)

– Instruction List sorted by Module Types (see annex A.2)

  Use the module's list of valid instructions to learn about the functions of a module.

– Thematic Instruction List (see annex A.3)

Instructions for *ADwin-Pro I* and *ADwin-Pro II* modules often are quite similar. For distinction, Pro II instructions have the prefix **P2_**.

To use an instruction you have to include the following line into your *ADbasic* program:

```
#Include ADwinPro_All.Inc
```

The description for each instruction includes:

– syntax and passed parameter.

– notes about specific features.

– a list of related instructions.

– a list of modules where the instruction is applicable.

– often an example.

  The examples (mostly) assume the module address to be set to the number 1.

All Pro II modules, which are accessed via an *ADbasic* instruction, must be plugged-in correctly. Otherwise the processor workload rises, even the communication to the PC may be interrupted.

Unlike the Pro I modules an access attempt to an non-accessible Pro II module lasts longer than if the module is accessible. This may happen for example, when a Pro II module is unplugged. The elongated access time increases the workload of the CPU module and changes the process timing.

3.1 Pro II: All Modules

This section describes instructions which apply to all or most of the Pro II modules:

– P2_Check_LED (page 5)
– P2_Set_LED (page 6)
– P2_Event_Enable (page 7)
– P2_Event_Config (page 9)
– P2_Event2_Config (page 10)
– P2_Event_Read (page 12)
– P2_Sync_All (page 13)
– P2_Sync_Enable (page 15)
– P2_Sync_Mode (page 17)
– P2_Sync_Stat (page 19)

P2_Check_LED

P2_Check_LED returns the status of the LED (on top of the front panel) of the module.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = P2_Check_LED(module)
```

Parameters

module	Module address (0…15): 0: CPU module. 1…15: Set module address.	LONG
ret_val	0: LED off (default). 1: LED on.	LONG

Notes

- / -

See also

P2_Set_LED

Valid for

AIn-16/18-8B Rev. E, AIn-16/18-C Rev. E, AIn-32/18 Rev. E, AIn-8/18 Rev. E, AIn-8/18-8B Rev. E, AIn-F-4/14 Rev. E, AIn-F-4/16 Rev. E, AIn-F-4/18 Rev. E, AIn-F-8/14 Rev. E, AIn-F-8/16 Rev. E, AIn-F-8/18 Rev. E, AOut-1/16 Rev. E, AOut-4/16 Rev. E, AOut-4/16-TiCo Rev. E, AOut-8/16 Rev. E, AOut-8/16-TiCo Rev. E, ARINC-429 Rev. E, CAN-2 Rev. E, CNT-D Rev. E, CNT-I Rev. E, CNT-T Rev. E, CPU-T11, DIO-32 Rev. E, DIO-32-TiCo Rev. E, DIO-32-TiCo2 Rev. E, EtherCAT-SL Rev. E, FlexRay-2 Rev. E, LIN-2 Rev. E, LS-2 Rev. E, MIL-STD-1553 Rev. E, MIO-4 Rev. E, MIO-4-ET1 Rev. E, MIO-D12 Rev. E, OPT-16 Rev. E, OPT-32 Rev. E, Profi-SL Rev. E, PWM-16(-I) Rev. E, REL-16 Rev. E, RS422-4 Rev. E, RSxxx-2 Rev. E, RSxxx-4 Rev. E, RTD-8 Rev. E, SENT-4 Rev. E, SENT-4-Out Rev. E, SENT-6 Rev. E, SPI-2-D Rev. E, SPI-2-T Rev. E, TRA-16 Rev. E

Example

```
#Include ADwinPro_All.Inc

Init:
  If (P2_Check_LED(1)=0) Then 'If LED is off …
    P2_Set_LED(1,1)          '… switch LED on
  EndIf
```

P2_Set_LED

P2_Set_LED switches the LED (on top of the front panel) on or off.

Syntax

```
#Include ADwinPro_All.Inc
P2_Set_LED(module,pattern)
```

Parameters

module	Module address (0…15): 0: CPU module. 1…15: Selected module address.	LONG
pattern	Status of the LED: 0: switch off. 1: switch on.	LONG

Notes

Some modules provide additional LEDs. The status of these LEDs is set with separate instructions.

See also

P2_Check_LED, P2_CAN_Set_LED, P2_LIN_Set_LED, P2_RS_Set_LED, P2_MIL_Set_LED, P2_FlexRay_Set_LED

Valid for

AIn-16/18-8B Rev. E, AIn-16/18-C Rev. E, AIn-32/18 Rev. E, AIn-8/18 Rev. E, AIn-8/18-8B Rev. E, AIn-F-4/14 Rev. E, AIn-F-4/16 Rev. E, AIn-F-4/18 Rev. E, AIn-F-8/14 Rev. E, AIn-F-8/16 Rev. E, AIn-F-8/18 Rev. E, AOut-1/16 Rev. E, AOut-4/16 Rev. E, AOut-4/16-TiCo Rev. E, AOut-8/16 Rev. E, AOut-8/16-TiCo Rev. E, ARINC-429 Rev. E, CAN-2 Rev. E, CNT-D Rev. E, CNT-I Rev. E, CNT-T Rev. E, CPU-T11, DIO-32 Rev. E, DIO-32-TiCo Rev. E, DIO-32-TiCo2 Rev. E, EtherCAT-SL Rev. E, FlexRay-2 Rev. E, LIN-2 Rev. E, LS-2 Rev. E, MIL-STD-1553 Rev. E, MIO-4 Rev. E, MIO-4-ET1 Rev. E, MIO-D12 Rev. E, OPT-16 Rev. E, OPT-32 Rev. E, Profi-SL Rev. E, PWM-16(-I) Rev. E, REL-16 Rev. E, RS422-4 Rev. E, RSxxx-2 Rev. E, RSxxx-4 Rev. E, RTD-8 Rev. E, SENT-4 Rev. E, SENT-4-Out Rev. E, SENT-6 Rev. E, SPI-2-D Rev. E, SPI-2-T Rev. E, TRA-16 Rev. E

Example

```
#Include ADwinPro_All.Inc
Init:
  P2_Set_LED(1,1)           'Switch on LED of module 1

Event:
  Rem …

Finish:
  P2_Set_LED(1,0)           'Switch off LED of module 1
  Rem …
```

P2_Event_Enable

P2_Event_Enable enables or disables an external event input on the specified module.

With a signal at this input a cycle of an *ADbasic* process can be controlled.

Syntax

```
#Include ADwinPro_All.Inc
P2_Event_Enable(module,enable)
```

Parameters

module	Specified module address (1…15).	LONG
enable	0 : disable external event signal (default). 1 : enable external event signal.	LONG

Notes

One high-priority *ADbasic* process (that is its cyclic section **Event:**), may be called by an external event signal, e.g. to synchronize it with an external process (see *ADbasic* manual).

One event input

Most of the modules have an event input. First configure the event input using **P2_Event_Config**. As soon as you have enabled the event input with **P2_Event_Enable**, the input signal will be forwarded to the processor module. The processor module recognizes the selected type of edge (rising or falling) as event signal and the specified process responds.

Several event inputs

For modules with several event inputs note the settings done with **P2_Event2_Config**. The settings of **P2_Event_Config** will then refer to the resulting event signal.

The event input of a processor module is always active and cannot be disabled with this instruction. The event input of the other modules is disabled after power-up.

In a system only one event input may be active, in addition to a processor module, that is you have to disable an actually active event input, before you enable the event input of another module.

See also

P2_Event_Config, P2_Event2_Config, P2_Event_Read

Valid for

AIn-16/18-8B Rev. E, AIn-16/18-C Rev. E, AIn-32/18 Rev. E, AIn-8/18 Rev. E, AIn-8/18-8B Rev. E, AIn-F-4/14 Rev. E, AIn-F-4/16 Rev. E, AIn-F-4/18 Rev. E, AIn-F-8/14 Rev. E, AIn-F-8/16 Rev. E, AIn-F-8/18 Rev. E, AOut-1/16 Rev. E, AOut-4/16 Rev. E, AOut-4/16-TiCo Rev. E, AOut-8/16 Rev. E, AOut-8/16-TiCo Rev. E, CAN-2 Rev. E, CNT-D Rev. E, CNT-I Rev. E, CNT-T Rev. E, DIO-32 Rev. E, DIO-32-TiCo Rev. E, DIO-32-TiCo2 Rev. E, MIO-4 Rev. E, MIO-4-ET1 Rev. E, MIO-D12 Rev. E, OPT-16 Rev. E, OPT-32 Rev. E, PWM-16(-I) Rev. E, REL-16 Rev. E, RS422-4 Rev. E, RSxxx-2 Rev. E, RSxxx-4 Rev. E, SPI-2-D Rev. E, SPI-2-T Rev. E, TRA-16 Rev. E

Example

```
#Include ADwinPro_All.Inc
Init:
  REM Configure event input for mimimum time of 15 ns,
  REM falling edge, 4 edges
  P2_Event_Config(1,0,2,4)
  'Enable an external event at the module 1
  P2_Event_Enable(1,1)
```

P2_Event_Config

P2_Event_Config configures the external event input of the specified module.

Syntax

```
#Include ADwinPro_All.Inc
P2_Event_Config(module,min_hold,edge,prescale)
```

Parameters

module	Specified module address (1…15).	LONG
min_hold	Minimum time, which an edge must be held to be accepted: 0: 15ns (Default). 1: 50ns.	LONG
edge	Type of edge which is accepted: 1: rising edge (Default). 2: fallig edge. 3: rising and falling edge.	LONG
prescale	Number (1…255) of edges, after which an event signal is triggered (Default: 1).	LONG

Notes

An event input must be enabled with **P2_Event_Enable** to have a present signal processed. First configure the event input with **P2_Event_Config** and enable the input then.

For modules with several event inputs note the settings done with **P2_Event2_Config**. The settings of **P2_Event_Config** will then refer to the resulting event signal. The setting min_hold is valid for all event inputs.

See also

P2_Event_Enable, P2_Event2_Config, P2_Event_Read

Valid for

AIn-16/18-8B Rev. E, AIn-16/18-C Rev. E, AIn-32/18 Rev. E, AIn-8/18 Rev. E, AIn-8/18-8B Rev. E, AIn-F-4/14 Rev. E, AIn-F-4/16 Rev. E, AIn-F-4/18 Rev. E, AIn-F-8/14 Rev. E, AIn-F-8/16 Rev. E, AIn-F-8/18 Rev. E, AOut-1/16 Rev. E, AOut-4/16 Rev. E, AOut-4/16-TiCo Rev. E, AOut-8/16 Rev. E, AOut-8/16-TiCo Rev. E, CAN-2 Rev. E, CNT-D Rev. E, CNT-I Rev. E, CNT-T Rev. E, DIO-32 Rev. E, DIO-32-TiCo Rev. E, DIO-32-TiCo2 Rev. E, MIO-4 Rev. E, MIO-4-ET1 Rev. E, MIO-D12 Rev. E, OPT-16 Rev. E, OPT-32 Rev. E, PWM-16(-I) Rev. E, REL-16 Rev. E, RS422-4 Rev. E, RSxxx-2 Rev. E, RSxxx-4 Rev. E, SPI-2-D Rev. E, SPI-2-T Rev. E, TRA-16 Rev. E

Example

```
#Include ADwinPro_All.Inc

Init:
  REM Configure event input for mimimum time of 15 ns,
  REM falling edge, 4 edges
  P2_Event_Config(1,0,2,4)
  'Enable an external event at the module 1
  P2_Event_Enable(1,1)
```

P2_Event2_Config

P2_Event2_Config configures the pre-processing of event signals on the specified module.

Syntax

```
#Include ADwinPro_All.Inc
P2_Event2_Config(module,mode,edge)
```

Parameters

module	Specified module address (1…15).	LONG
mode	Mode of event signal pre-processing: 0: No pre-processing (default). 1: Signal after clearance at the input ENABLE. 2: Signal from AB mode. 3: Signal from AB mode after clearance at the input ENABLE.	LONG
edge	For mode=1 or mode=3 only; type of edge, that is accepted at the input ENABLE: 1: rising edge (default). 2: falling edge. 3: rising and falling edge.	LONG

Notes

The instruction works only for modules with more than one event input. The module processes incoming event signals to the resulting event signal, which controls the event process on the processor module.

The resulting event signal is configured with **P2_Event_Config** and enabled with **P2_Event_Enable**.

According to the module different modes mode are available:

Module name	Available modes
Pro-AIn-F-8/14-D	0, 1, 2, 3
Pro-AIn-F-8/16-D	0, 1, 2, 3
Pro-AIn-F-8/18-D	0, 1

No pre-processing

The module passes the signal at input EVENT / A as resulting event signal. The parameter edge is of no importance.

Signal after clearance

The input EVENT / A is disabled first, the resulting event signal is TTL level low. As soon as an edge of type edge arrives at ENABLE, the module enables input EVENT / A and passes its signal as resulting event signal.

The input EVENT / A is disabled again by setting mode 0.

Signal from AB mode

In AB mode, the module evaluates two rectangular signals at the inputs EVENT / A and ENABLE, which are phase-shifted by 90 degrees (typical for incremental encoders): If an edge of type edge arrives at one of the inputs, a resulting event signal is triggered.

The maximum input frequency is 5 MHz; in combination with the 4 edges per signal cycle the maximum frequency of the resulting event signal ensues to 20 MHz.

The time between an edge at EVENT / A and an edge at ENABLE must not be shorter than 50 ns. Impulse widths or pause durations shorter than 100 ns are not processed.

Changing the phase-shift has an effect on the maximum input frequency because of the minimum time betwenn the edges. If the phase-shift dif-

fers from 90 degrees, the maximum input frequency of 5 MHz decreases for instance to 45 degrees at 2.5 MHz

As soon as an edge of type edge arrives at input ENABLE, the module enables the resulting event signal evaluated from the two rectangular signals of the inputs EVENT / A and ENABLE (see Signal from AB mode).

In order to use a clearance signal again, set mode = 0 to disable the inputs EVENT / A and B then set mode = 3.

Signal from AB mode after clearance

See also

P2_Event_Enable, P2_Event_Config, P2_Event_Read

Valid for

AIn-F-4/14 Rev. E, AIn-F-4/16 Rev. E, AIn-F-4/18 Rev. E, AIn-F-8/14 Rev. E, AIn-F-8/16 Rev. E, AIn-F-8/18 Rev. E

Example

```
#Include ADwinPro_All.Inc

Init:
  REM Configure event input for mimimum time of 15 ns,
  REM falling edge, 4 edges
  P2_Event_Config(1,0,2,4)
  Rem set event pre-processing to clearance and
  Rem negative edge
  P2_Event2_Config(1,1,2)
  REM Enable an external event at the module 1
  P2_Event_Enable(1,1)
```

P2_Event_Read

P2_Event_Read returns the current TTL level at the event inputs of the specified module.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = P2_Event_Read(module)
```

Parameters

`module` Module address (0…15): LONG
1…15: specified module address.

`ret_val` Bit pattern representing the current TTL levels; LONG
mapping of bits and event inputs see table.
Bit = 0: TTL level low.
Bit = 1: TTL level high.

Bits in ret_val	31:03	02	01	00
Input	–	B	Enable	Event / A

Notes

The inputs A, B, and Enable are not present on any module.

See also

P2_Event_Enable, P2_Event_Config, P2_Event2_Config

Valid for

AIn-16/18-8B Rev. E, AIn-16/18-C Rev. E, AIn-32/18 Rev. E, AIn-8/18 Rev. E, AIn-8/18-8B Rev. E, AIn-F-4/14 Rev. E, AIn-F-4/16 Rev. E, AIn-F-4/18 Rev. E, AIn-F-8/14 Rev. E, AIn-F-8/16 Rev. E, AIn-F-8/18 Rev. E, AOut-1/16 Rev. E, AOut-4/16 Rev. E, AOut-4/16-TiCo Rev. E, AOut-8/16 Rev. E, AOut-8/16-TiCo Rev. E, CAN-2 Rev. E, CNT-D Rev. E, CNT-I Rev. E, CNT-T Rev. E, DIO-32 Rev. E, DIO-32-TiCo Rev. E, DIO-32-TiCo2 Rev. E, MIO-4 Rev. E, MIO-4-ET1 Rev. E, MIO-D12 Rev. E, OPT-16 Rev. E, OPT-32 Rev. E, PWM-16(-I) Rev. E, REL-16 Rev. E, RS422-4 Rev. E, RSxxx-2 Rev. E, RSxxx-4 Rev. E, SPI-2-D Rev. E, SPI-2-T Rev. E, TRA-16 Rev. E

Example

```
#Include ADwinPro_All.Inc

Init:
  Rem Configure event input of module 1 (AIn-F-8/14)
  Rem with min. time 15 ns, neg. edge, 4 edges
  P2_Event_Config(1,0,2,4)
  Rem enable event signal of module 1
  P2_Event_Enable(1,1)

Event:
  Par_1 = P2_Event_Read(1)
```

P2_Sync_All

P2_Sync_All starts a specified action synchronically on the selected modules.

Syntax

```
#Include ADwinPro_All.Inc
P2_Sync_All(pattern)
```

Parameters

pattern — Bit pattern selecting the addresses of the modules which start an action: LONG
Bit = 0: Ignore module.
Bit = 1: Start module action synchronously.

Bits in pattern	31:16	14	13	…	01	00
Module address	–	15	14	…	2	1

Notes

The action starting on the selected modules depends on the module types and is similar to a standard instruction. Configurations being made before do apply e.g. for the multiplexer, output value, or burst mode.

Module type	Action	similar to
Analog input	Start A/D conversion on all enabled ADCs. The conversion is a single conversion. For burst sequences see **P2_Sync_Mode**.	**P2_Start_Conv** / **P2_Start_ConvF**
	AIn-F-x/14: the currently converted values of all channels are buffered.	- / -
Analog output	Start D/A conversion on all enabled DACs with the value of the DAC register.	**P2_Start_DAC**
Digital input	Transfer current status of all inputs into the input latch register. Read values: **P2_Dig_Read_Latch**.	**P2_Dig_Latch**
Digital output	Read the values from the output latch register and transfer to all digital outputs. Write latch: P2_Dig_Write_Latch or **P2_PWM_Write_Latch**.	**P2_Dig_Latch** / **P2_PWM_Latch**.
Counter	Copy current counter values into counter latches. Read values e.g. **P2_Cnt_Read_Latch** or **P2_Cnt_Get_PW**.	**P2_Cnt_Latch**
Temperature	Transfer current status of the inputs into the input latch register. Read values e.g. **P2_TC_Read_Latch**.	**P2_TC_Latch**
Multi I/O	The synchonizing signal refers to all enabled groups: – analog inputs – analog outputs – digital channels – counters The started actions are described above.	**P2_Start_Conv** **P2_Start_DAC** **P2_MIO_Dig_Latch** **P2_Cnt_Latch**

As default all inputs / outputs of the selected modules participate in the action. Using **P2_Sync_Enable** you may disable or enable one or more inputs / outputs or function groups of a module for synchronization.

Only for AIn-F-4/14 and AIn-F-8/14: All buffered values must be read at the same time with a single instruction: **P2_Read_ADCF_4**, **P2_Read_ADCF_8**, **P2_Read_ADCF_4_Packed**, **P2_Read_ADCF_8_Packed**.

The following instructions do a synchronous action, too:

- **P2_Sync_Mode**: An external event signal triggers a conversion simultaneously on all channels. The conversion can be part of a single measurement (**P2_ADCF_Mode**) or of a burst-measurement sequence (**P2_Burst_Init**).
- **P2_Burst_Start**: By software, burst sequences are started on several modules.

See also

P2_Sync_Enable, P2_Sync_Mode, P2_Sync_Stat

P2_Start_Conv, P2_Start_ConvF, P2_Read_ADCF4, P2_Read_ADCF8, P2_Start_DAC, P2_Burst_Start

P2_Dig_Latch, P2_PWM_Latch, P2_TC_Latch

Valid for

AIn-16/18-8B Rev. E, AIn-16/18-C Rev. E, AIn-32/18 Rev. E, AIn-8/18 Rev. E, AIn-8/18-8B Rev. E, AIn-F-4/14 Rev. E, AIn-F-4/16 Rev. E, AIn-F-4/18 Rev. E, AIn-F-8/14 Rev. E, AIn-F-8/16 Rev. E, AIn-F-8/18 Rev. E, AOut-1/16 Rev. E, AOut-4/16 Rev. E, AOut-4/16-TiCo Rev. E, AOut-8/16 Rev. E, AOut-8/16-TiCo Rev. E, CNT-D Rev. E, CNT-I Rev. E, CNT-T Rev. E, DIO-32 Rev. E, DIO-32-TiCo Rev. E, DIO-32-TiCo2 Rev. E, MIO-4 Rev. E, MIO-4-ET1 Rev. E, MIO-D12 Rev. E, OPT-16 Rev. E, OPT-32 Rev. E, PWM-16(-I) Rev. E, REL-16 Rev. E, TC-8-ISO Rev. E, TRA-16 Rev. E

Example

```
#Include ADwinPro_All.Inc
Dim i As Long
Dim Data_1[1000], Data_2[1000], Data_3[1000] As Long
Dim Data_5[1000] As Long

Init:
  REM Enable channel synchronization on the modules 1,2,4,5
  P2_Sync_Enable(1,11b)
  P2_Sync_Enable(2,11b)
  P2_Sync_Enable(4,11b)
  P2_Sync_Enable(5,100b)
  P2_Write_DAC(5,1,0)       'initialize output
  i=1                       'initialize index

Event:
  REM Start conversion of modules 1, 2, 4 and 5 synchronously
  P2_Sync_All(11011b)
  P2_Wait_EOC(1)            'Wait for end of conversion
  Rem Read A/D converter 1 of modules 1,2,4
  Data_1[i]=P2_Read_ADCF(1,1)
  Data_2[i]=P2_Read_ADCF(2,1)
  Data_3[i]=P2_Read_ADCF(4,1)

  REM write value into output register of D/A module 5
  P2_Write_DAC(5,1,Data_5[i])
  If (i=1000) Then End      'End process after 1000 repetitions
  Inc(i)                    'Increment index
```

P2_Sync_Enable

P2_Sync_Enable enables or disables the synchronizing option for selected inputs, outputs or function groups on the specified module.

Syntax

```
#Include ADwinPro_All.Inc
P2_Sync_Enable(module,channel)
```

Parameters

module	Specified module address (1…15).	LONG
channel	Bit pattern for selection of inputs, outputs or function groups that are enabled or disabled: 0 : disable. 1: enable.	LONG

Analog modules: AIn-F-x/16 Rev. E, AIn-F-x/18 Rev. E, AOut-x/16 Rev. E

Bits in channel	31:08	07	06	05	04	03	02	01	00
Channel no.	–	8	7	6	5	4	3	2	1

Digital modules: DIO-32 Rev. E, DIO-32-TiCo Rev. E, DIO-32-TiCo2 Rev. E OPT-16 Rev. E, OPT-32 Rev. E, PWM-16 Rev. E, REL-16 Rev. E, TRA-16 Rev. E

Bits in channel	31:02	00
Function group	–	digital channels

Multi I/O modules: MIO-4 Rev. E, MIO-4-ET1 Rev. E, MIO-D12 Rev. E

Bits in channel	31:04	03	02	01	00
Function group	–	counters	analog inputs	analog outputs	digital channels

Notes

The default setting after start-up for all modules is the value **0FFFFh**, that is all inputs, outputs or function groups are enabled.

The instruction sets all channels or function groups of a module at the same time. If you want to disable or enable a single channel, you have to indicate the settings of the other channels as well; the same applies to function groups.
With digital modules you cannot select single channels, but only all channels at the same time.

The synchronizing signal is triggered by **P2_Sync_All**.

See also

P2_Sync_All, P2_Sync_Mode, P2_Sync_Stat

Valid for

AIn-F-4/16 Rev. E, AIn-F-4/18 Rev. E, AIn-F-8/16 Rev. E, AIn-F-8/18 Rev. E, AOut-4/16 Rev. E, AOut-4/16-TiCo Rev. E, AOut-8/16 Rev. E, AOut-8/16-TiCo Rev. E, DIO-32 Rev. E, DIO-32-TiCo Rev. E, DIO-32-TiCo2 Rev. E, MIO-4 Rev. E, MIO-4-ET1 Rev. E, MIO-D12 Rev. E, OPT-16 Rev. E, OPT-32 Rev. E, PWM-16(-I) Rev. E, REL-16 Rev. E, TRA-16 Rev. E

Example

```
#Include ADwinPro_All.Inc
Dim i As Long
Dim Data_1[1000], Data_2[1000], Data_3[1000] As Long
Dim Data_5[1000] As Long

Init:
  REM Enable channel synchronization on the modules 1,2,4,5
  P2_Sync_Enable(1,11b)
  P2_Sync_Enable(2,11b)
  P2_Sync_Enable(4,11b)
  P2_Sync_Enable(5,100b)
  P2_Write_DAC(5,1,0)       'initialize output
  i=1                       'initialize index

Event:
  REM Start conversion of modules 1, 2, 4 and 5 synchronously
  P2_Sync_All(11011b)
  P2_Wait_EOC(1)            'Wait for end of conversion
  Rem Read A/D converter 1 of modules 1,2,4
  Data_1[i]=P2_Read_ADCF(1,1)
  Data_2[i]=P2_Read_ADCF(2,1)
  Data_3[i]=P2_Read_ADCF(4,1)

  REM write value into output register of D/A module 5
  P2_Write_DAC(5,1,Data_5[i])
  If (i=1000) Then End      'End process after 1000 repetitions
  Inc(i)                    'Increment index
```

P2_Sync_Mode

P2_Sync_Mode enables or disables the synchronization (of conversions) with other modules as master or as slave.

Syntax

```
#Include ADwinPro_All.Inc
P2_Sync_Mode(module,mode)
```

Parameters

module	Specified module address (1…15).	LONG
mode	Synchronization mode of the module (0…2): 0: no synchronization (default setting). 1: Synchronization as master module. 2: Synchronization as slave module.	LONG

Notes

The synchronization allows to have a signal at the event input of the master module simultaneously start conversions on all slave modules. The master module will forward the event signal to the slave modules.

The settings for pre-processing the event signal (**P2_Event_Config**, **P2_Event2_Config**) may be different for each module.

Only one master module is allowed.

As soon as synchronized slave modules (mode 2) receive the forwarded event signal, they start a conversion (like **P2_Start_ConvF** does) simultaneously on all channels. The conversion can be part of a single measurement or of a burst-measurement sequence.

If you synchronize burst-measurement sequences on several modules, you should initialize the modules to the same number of measurements with **P2_Burst_Init.** This refers especially to the master / slave synchronization: The number of measurements on the master module must be equal or greater than the number on the slave modules. If not, on the latter modules the burst-measurement sequence will not be ended with the last signal from the master module.

See also

P2_Burst_Init, P2_Burst_Read, P2_Burst_Start, P2_Burst_Status, P2_Event_Enable, P2_Event_Config, P2_Event2_Config, P2_Start_ConvF, P2_Sync_All, P2_Sync_Enable, P2_Sync_Stat

Valid for

AIn-F-4/16 Rev. E, AIn-F-4/18 Rev. E, AIn-F-8/16 Rev. E, AIn-F-8/18 Rev. E

Example

```
#Include ADwinPRO_ALL.Inc
#Define count 10000
#Define module 1

Dim i As Long
Dim Data_1[count], Data_2[count], Data_3[count] As Long
Dim Data_4[count], Data_5[count], Data_6[count] As Long
Dim Data_7[count], Data_8[count], Data_9[count] As Long
Dim Data_10[count], Data_11[count], Data_12[count] As Long

Init:
  P2_Sync_Mode(module,1)    'master module
  P2_Sync_Mode(module+1,2) 'slave module 1
  P2_Sync_Mode(module+2,2) 'slave module 2
  Rem initialize and start burst measurement for 4 channels
  P2_Burst_Init(module,15,0,count,1,100b)
  P2_Burst_Init(module+1,15,0,count,1,100b)
  P2_Burst_Init(module+2,15,0,count,1,100b)
  P2_Burst_Start(111b)      'start burst measurement on module 1-3
  Processdelay=800          'get trigger point with 50 kHz

Event:
  Par_1=P2_Burst_Status(module)'number of remaining measurements
  If (Par_1=0) Then End     'burst sequence finished, go to FINISH

Finish:
  Rem copy the last converted data of all 4 channels
  P2_Burst_Read_Unpacked4(module,count,0,
     Data_1,Data_2,Data_3,Data_4,1,3)
  P2_Burst_Read_Unpacked4(module+1,count,0,
     Data_5,Data_6,Data_7,Data_8,1,3)
  P2_Burst_Read_Unpacked4(module+2,count,0,
     Data_10,Data_10,Data_11,Data_12,1,3)
```

P2_Sync_Stat

P2_Sync_Stat returns the settings of the synchronizing option of the specified module.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = P2_Sync_Stat(module)
```

Parameters

`module` Specified module address (1…15). LONG

`ret_val` Setting of the synchronizing option at the inputs / outputs: LONG
0 : disabled.
1 : enabled.

Analog modules: AIn-F-x/x Rev. E, AOut-x/16 Rev. E

Bits in channel	31:08	07	06	05	04	03	02	01	00
Channel no.	–	8	7	6	5	4	3	2	1

Digital modules: DIO-32 Rev. E, DIO-32-TiCo Rev. E, DIO-32-TiCo2 Rev. E OPT-16 Rev. E, OPT-32 Rev. E, PWM-16 Rev. E, REL-16 Rev. E, TRA-16 Rev. E

Bits in channel	31:02	00
Function group	–	digital channels

Multi I/O modules: MIO-4 Rev. E, MIO-4-ET1 Rev. E, MIO-D12 Rev. E

Bits in channel	31:04	03	02	01	00
Function group	–	counters	analog inputs	analog outputs	digital channels

Notes

You set the synchronizing option of the channels or function groups with **P2_Sync_Enable**.

See also

P2_Sync_All, P2_Sync_Enable, P2_Sync_Mode

Valid for

AIn-F-4/16 Rev. E, AIn-F-4/18 Rev. E, AIn-F-8/16 Rev. E, AIn-F-8/18 Rev. E, AOut-4/16 Rev. E, AOut-4/16-TiCo Rev. E, AOut-8/16 Rev. E, AOut-8/16-TiCo Rev. E, MIO-4 Rev. E, MIO-4-ET1 Rev. E, MIO-D12 Rev. E, PWM-16(-I) Rev. E

Example

```
#Include ADwinPro_All.Inc
Dim i As Long
Dim Data_1[1000], Data_2[1000] As Long

Init:
  REM Is channel 1 on module 1 still disabled?
  If (P2_Sync_Stat(1) And 1 = 0) Then
    REM Enable channel 1 on D/A modules 1+2
    REM disable all other channels
    P2_Sync_Enable(1,1)
    P2_Sync_Enable(2,1)
  EndIf
  i=1                       'Initialize index

Event:
  REM write values into output registers
  P2_Write_DAC(1,1,Data_1[i])
  P2_Write_DAC(2,1,Data_2[i])
  REM Start output on modules 1+2 synchronously
  P2_Sync_All(11b)
  If (i=1000) Then End      'End process after 1000 repetitions
  Inc(i)                    'Increment index
```

3.2 Pro II: Digital Channels of CPU Modules

This section describes instructions which apply to digital channels of Pro II CPU modules:

– CPU_Digin (page 22)
– CPU_Digout (page 23)
– CPU_Dig_IO_Config (page 24)
– CPU_Event_Config (page 25)

CPU_Digin

Processor T11 only. **CPU_Digin** returns, whether an edge was detected at the input DIG I/O of the processor module since the previous call.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = CPU_Digin(channel)
```

Parameters

channel	Number of the DIG I/O input on the module: 0: DIG I/O 0. 1: DIG I/O 1.	LONG
ret_val	Flag, if a falling edge has been detected at the DIG I/O input: 0: No falling edge detected. 1: Falling edge has been detected at least once.	LONG

Notes

CPU_Digin is active only if the selected DIG I/O channel is configured as input with **CPU_Dig_IO_Config**.
Using **CPU_Dig_IO_Config** you set, whether **CPU_Digin** reacts on a rising or a falling edge. After startup the DIG I/O channels are configured as inputs and for falling edges.

CPU_Digin reads the module's internal flag for falling edges; doing so, the flag will be automatically reset to the value 0.

The inputs DIG I/O work with TTL signals only.

See also

CPU_Digout, CPU_Dig_IO_Config

Valid for

CPU-T11

Example

```
#Include ADwinPro_All.Inc
Dim dummy As Long

Init:
  REM Set both DIG I/O channels as input with rising edge
  CPU_Dig_IO_Config(100010b)
  REM Read and thus reset status signal on DIG I/O 1
  dummy = CPU_Digin(1)

Event:
  Rem …
  If (CPU_Digin(1) = 1) Then'If falling edge has been detected …
    End                     '… end this program
  EndIf
  Rem …
```

CPU_Digout

CPU_Digout sets a DIG I/O output of the processor module to the selected TTL level.

Syntax

```
#Include ADwinPro_All.Inc
CPU_Digout(channel,level)
```

Parameters

channel	Number (0, 1) of the DIG I/O output on the processor module.	LONG
level	TTL level of the output: 0: TTL level low. 1: TTL level high.	LONG

Notes

CPU_Digout is active only if the selected DIG I/O channel is configured as output with **CPU_Dig_IO_Config**.

See also

CPU_Digin, CPU_Dig_IO_Config

Valid for

CPU-T11

Example

```
#Include ADwinPro_All.Inc

Init:
  Rem set DIG I/O-0 and DIG I/O-1 as outputs
  CPU_Dig_IO_Config(110011b)

Event:
  Rem …
  CPU_Digout(1,0)           'Set DIG I/O 1 to TTL level low
  Rem …
```

CPU_Dig_IO_Config

CPU_Dig_IO_Config configures all DIG I/O channels of the processor module.

Syntax

```
#Include ADwinPro_All.Inc
CPU_Dig_IO_Config(pattern)
```

Parameters

pattern — Bit pattern to configure the DIG I/O channels. LONG
2 bits each configure a channel.
The other bits have no function.

Bits in pattern	31:06	05:04	03:02	01:00
configure channel	–	DIG I/O-1	–	DIG I/O-0

Bit combination for one channel	Configuration
00	Channel works as input for falling edges.
10	Channel works as input for rising edges.
01 or **11**	Channel works as output.

Notes

The type of edge may be set for inputs only.

The DIG I/O channels are equipped with pull-up resistors; in order to detect a rising edge, the input has therefore to be actively pulled to TTL level low.

After startup the DIG I/O channels are configured as inputs and for falling edges.

See also

CPU_Digin, CPU_Digout, CPU_Event_Config

Valid for

CPU-T11

Example

```
#Include ADwinPro_All.Inc
Dim dummy As Long

Init:
  REM Set DIG I/O-0 as input with rising edge,
  REM Set DIG I/O-1 as output
  CPU_Dig_IO_Config(110010b)
  REM Read and thus reset status signal on DIG I/O-0
  dummy = CPU_Digin(0)
  Rem …
```

CPU_Event_Config

CPU_Event_Config configures the Event In channel of the processor module.

Syntax

```
#Include ADwinPro_All.Inc
CPU_Event_Config(min_hold,edge,prescale)
```

Parameters

min_hold	Minimum time, which an edge must be held to be accepted: 0: 15ns (Default). 1: 50ns.	LONG
edge	Type of edge which is accepted: 1: rising edge (Default). 2: fallig edge. 3: ising and falling edge.	LONG
prescale	Number (1…15) of edges, after which an event signal is triggered (Default: 1).	LONG

Notes

The input Event In works with TTL signals only.

If input signals contain glitches - as far as can't be avoided - you may do the following:

- Set parameter min_hold to 1, to filter glitches.
- Redirect the input signal via an opto couple first.
- Connect the input signal t the module Pro-OPT-16 and enable the module's external event input with **EventEnable**.

See also

P2_Event_Enable, P2_Event_Read

Valid for

CPU-T11

Example

```
#Include ADwinPro_All.Inc

Init:
  REM Configure input EVENT IN for mimimum time of 15 ns,
  REM falling edge, 4 edges
  CPU_Event_Config(0,2,4)

Event:
  REM Externally controlled process starts each time, when
  REM 4 falling edges have reached the input EVENT IN.
  Rem …
```

3.3 Pro II: Multi-I/O

This section describes instructions which apply to Pro II multi I/O modules:

– P2_MIO_Dig_Latch (page 27)
– P2_MIO_Dig_Read_Latch (page 28)
– P2_MIO_Dig_Write_Latch (page 29)
– P2_MIO_Digin_Long (page 30)
– P2_MIO_Digout (page 31)
– P2_MIO_Digout_Long (page 32)
– P2_MIO_DigProg (page 33)
– P2_MIO_Get_Digout_Long (page 34)

In the Instruction List sorted by Module Types (annex A.2) you will find overviews of the instructions corresponding to the *ADwin-Pro* modules.

It is presumed that application examples use the module address 1 for D/A modules.

P2_MIO_Dig_Latch

P2_MIO_Dig_Latch transfers digital information from the inputs to the input latches and/or from the output latches to the outputs on the specified module.

Syntax

```
#Include ADwinPro_All.inc
P2_MIO_Dig_Latch(module)
```

Parameters

module	Specified module address (1…15).	LONG

Notes

It is recommended to first program the channels as inputs or outputs using **P2_MIO_DigProg**, except for TRA and OPT channels.

With digital inputs the instructions reads the input signals into the input latches. With digital outputs the instruction passes the values of the output latches to the outputs.

If the module is released for synchronization by **P2_Sync_Enable**, **P2_Sync_All** has the same functions as **P2_MIO_Dig_Latch**.

See also

P2_MIO_Dig_Read_Latch, P2_MIO_Dig_Write_Latch, P2_MIO_DigProg, P2_MIO_Digin_Long, P2_MIO_Digout, P2_MIO_Digout_Long, P2_MIO_Get_Digout_Long, P2_Sync_All

Valid for

MIO-4 Rev. E, MIO-4-ET1 Rev. E, MIO-D12 Rev. E

Example

```
#Include ADwinPro_All.inc
Rem example for MIO-4 / MIO-4-ET1

Init:
  Rem set channels 0…3 as outputs, 4…7 as inputs
  P2_MIO_DigProg(1,01b)
  P2_MIO_Dig_Write_Latch(1,0)'set output bits to 0

Event:
  Rem latch inputs and output the content of output latches
  P2_MIO_Dig_Latch(1)
  Par_1 = P2_MIO_Dig_Read_Latch(1) 'read input bits and …
  P2_MIO_Dig_Write_Latch(1,Par_1)'output with next event
```

P2_MIO_Dig_Read_Latch

P2_MIO_Dig_Read_Latch returns the bits from the latch register for the digital inputs of the specified module.

Syntax

```
#Include ADwinPro_All.inc
ret_val = P2_MIO_Dig_Read_Latch(module)
```

Parameters

module	Specified module address (1…15).	LONG
ret_val	Bit pattern of the latch register. Each bit corresponds to a digital input (see table).	LONG

MIO-4 Rev. E, MIO-4-ET1 Rev. E

Bit no.	31…20	19 … 16	15…8	7 6 … 1 0
Input	–	OPT4 … OPT1	–	7 6 … 1 0

MIO-D12 Rev. E

Bit no.	31…12	11 … 0
Input	–	OPT11 … OPT0

Notes

It is recommended to first program the specified channels as inputs using **P2_MIO_Digprog**, except for OPT inputs.

The current status of the digital inputs can be transferred to the latch register with the following instructions:

- **P2_MIO_Dig_Latch**
- **P2_Sync_All** (when activated for the module)

See also

P2_MIO_Dig_Latch, P2_MIO_Dig_Write_Latch, P2_MIO_DigProg, P2_MIO_Digin_Long, P2_Sync_All

Valid for

MIO-4 Rev. E, MIO-4-ET1 Rev. E, MIO-D12 Rev. E

Example

```
#Include ADwinPro_All.inc
Dim value As Long

Init:
  Rem set channels 07:00 of modules 1+2 as inputs
  P2_MIO_Digprog(1,00b)
  P2_MIO_Digprog(2,00b)

Event:
  Rem copy digital output levels of both modules synchronously
  Rem into the latches
  P2_Sync_All(11b)
  Par_1 = P2_MIO_Dig_Read_Latch(1)'read latch of module 1
  Par_2 = P2_MIO_Dig_Read_Latch(2)'read latch of module 2
```

P2_MIO_Dig_Write_Latch

P2_MIO_Dig_Write_Latch writes a 32 bit value into the latch register for the digital outputs on the specified module.

Syntax

```
#Include ADwinPro_All.inc
P2_MIO_Dig_Write_Latch(module,pattern)
```

Parameters

module	Specified module address (1…15).	LONG
pattern	Bit pattern. Each bit corresponds to a digital output (see table).	LONG

MIO-4 Rev. E, MIO-4-ET1 Rev. E

Bit no.	31…28	27 … 24	23…8	7 6 … 1 0
Output	–	TRA4 … TRA1	–	7 6 … 1 0

MIO-D12 Rev. E

Bit no.	31…12	27 … 16	15…0
Output	–	TRA11 … TRA0	–

Notes

The specified channels must first be programmed as outputs using **P2_MIO_Digprog**, except for TRA outputs.

You may set the value of the latch register for digital outputs with **P2_MIO_Digout**.

See also

P2_MIO_Dig_Latch, P2_MIO_Digout, P2_MIO_DigProg, P2_MIO_Get_Digout_Long

Valid for

MIO-4 Rev. E, MIO-4-ET1 Rev. E, MIO-D12 Rev. E

Example

```
#Include ADwinPro_All.inc

Init:
  P2_MIO_Digprog(1,11b)    'module channels 07:00 as outputs

Event:
  Rem output information of output latch to the pins
  P2_MIO_Dig_Latch(1)
  Rem write a long word to the output latch
  P2_MIO_Dig_Write_Latch(1,Par_1)
```

P2_MIO_Digin_Long

P2_MIO_Digin_Long returns the status of the inputs (bits 7…0) of the specified module as bit pattern

Syntax

```
#Include ADwinPro_All.inc
ret_val = P2_MIO_Digin_Long(module)
```

Parameters

module	Specified module address (1…15).	LONG
ret_val	Bit pattern. Each bit (7…0) corresponds to the input status of a digital input (see table). Bit = 0: Input has low level. Bit = 1: Input has high level.	LONG

MIO-4 Rev. E, MIO-4-ET1 Rev. E

Bit no.	31…20	19 … 16	15…8	7 6 … 1 0
Input	–	OPT4 … OPT1	–	7 6 … 1 0

MIO-D12 Rev. E

Bit no.	31…12	11 … 0
Input	–	OPT11 … OPT0

Notes

It is recommended to first program the specified channels as inputs using **P2_MIO_Digprog**, except for OPT inputs.

See also

P2_MIO_Dig_Latch, P2_MIO_DigProg, P2_MIO_Digout_Long

Valid for

MIO-4 Rev. E, MIO-4-ET1 Rev. E, MIO-D12 Rev. E

Example

```
#Include ADwinPro_All.inc

Init:
  P2_MIO_Digprog(1,00b)     'channels 07:00 as inputs

Event:
  Par_1 = P2_MIO_Digin_Long(1)'read all inputs
```

P2_MIO_Digout

P2_MIO_Digout sets a single output of the specified module to the level "high" or "low". All other outputs remain unchanged.

Syntax

```
#Include ADwinPro_All.inc
P2_MIO_Digout(module,output,value)
```

Parameters

module	Specified module address (1…15).	LONG
output	Number of the output to be set. MIO-4 Rev. E, MIO-4-ET1 Rev. E 0…7: DIO channels. 24…27: TRA outputs. MIO-D12 Rev. E 16…27: TRA outputs.	LONG
value	New status of the selected output: 0: Low level. 1: High level.	LONG

Notes

The specified channels must be first programmed as outputs using **P2_MIO_Digprog**; except for TRA outputs.

With **P2_MIO_Digout**, you can set or clear any output without changing the status of the remaining outputs.

See also

P2_MIO_Digout_Long P2_MIO_DigProg

Valid for

MIO-4 Rev. E, MIO-4-ET1 Rev. E, MIO-D12 Rev. E

Example

```
#Include ADwinPro_All.inc

Init:
  Rem channels 0…3 as inputs, 4…7 as outputs
  P2_MIO_Digprog(1,10b)

Event:
  Rem read input bits and check if channel 3 is set
  If (P2_MIO_Digin_Long(1) And 1000b = 1) Then
    P2_MIO_Digout(1,5,0)    'channel 3 is set: clear bit 5
  Else
    P2_MIO_Digout(1,5,1)    'channel 3 is set: set bit 5
  EndIf
```

P2_MIO_Digout_Long

P2_MIO_Digout_Long sets or clears all outputs on the specified module.

Syntax

```
#Include ADwinPro_All.inc
P2_MIO_Digout_Long(module,pattern)
```

Parameters

module	Specified module address (1…15).	LONG
pattern	Bit pattern that sets the digital outputs: Bit = 0: Set output to level "low". Bit = 1: Set output to level "high".	LONG

MIO-4 Rev. E, MIO-4-ET1 Rev. E

Bit no.	31…28	27 … 24	23…8	7 6 … 1 0
Output	–	TRA4 … TRA1	–	7 6 … 1 0

MIO-D12 Rev. E

Bit no.	31…12	27 … 16	15…0
Output	–	TRA11 … TRA0	–

Notes

The specified channels must be first programmed as outputs using **P2_MIO_Digprog**, except for TRA outputs.

See also

P2_MIO_Digout, P2_MIO_DigProg

Valid for

MIO-4 Rev. E, MIO-4-ET1 Rev. E, MIO-D12 Rev. E

Example

```
#Include ADwinPro_All.inc

Init:
  P2_MIO_Digprog(1,01111b) 'channels 07:00 as outputs

Event:
  P2_MIO_Digout_Long(1,128)'Output 128 as binary value on the
                           'channels
```

P2_MIO_DigProg

P2_MIO_DigProg programs the digital channels 0…7 of the specified module as inputs or outputs in groups of 4.

Syntax

```
#Include ADwinPro_All.inc
P2_MIO_Digprog(module,pattern)
```

Parameters

module	Specified module address (1…15).	LONG
pattern	Bit pattern that sets the channels as inputs or outputs: Bit = 0: Set channel as input. Bit = 1: Set channel as output.	LONG

Bit no.	31…2	1	0
channel no.	–	07:04	03:00

Notes

After power-up of the system the channels 0…7 are configured as inputs.

Channels can only be set as inputs or outputs in groups of 4, 2 relevant bits only, the other bits are ignored.
OPT channels are statically set as inputs and TRA channels as outputs; they cannot be changed.

See also

P2_MIO_Dig_Latch, P2_MIO_Dig_Read_Latch, P2_MIO_Dig_Write_Latch

P2_MIO_Digin_Long, P2_MIO_Digout, P2_MIO_Digout_Long, P2_MIO_Get_Digout_Long

Valid for

MIO-4 Rev. E, MIO-4-ET1 Rev. E, MIO-D12 Rev. E

Example

```
#Include ADwinPro_All.inc

Init:
  Rem Configure channels 0…3 of module no. 1 as inputs
  Rem and channels 4…7 as outputs
  P2_MIO_Digprog(1, 10b)
```

P2_MIO_Get_Digout_Long

P2_MIO_Get_Digout_Long returns the contents of the output latch (register for digital outputs) on the specified module.

Syntax

```
#Include ADwinPro_All.inc
ret_val = P2_MIO_Get_Digout_Long(module)
```

Parameters

module	Specified module address (1…15).	LONG
ret_val	Contents of the output latch. Each bit corresponds to the status of a digital output (see table): Bit = 0: Output has low level. Bit = 1: Output has high level.	LONG

MIO-4 Rev. E, MIO-4-ET1 Rev. E

Bit no.	31…28	27 … 24	23…8	7 6 … 1 0
Output	–	TRA4 … TRA1	–	7 6 … 1 0

MIO-D12 Rev. E

Bit no.	31…12	27 … 16	15…0
Output	–	TRA11 … TRA0	–

Notes

Returning the current status of the outputs instead of the output latch is technically impossible.

See also

P2_MIO_Dig_Latch, P2_MIO_Dig_Read_Latch, P2_MIO_Dig_Write_Latch

P2_MIO_DigProg, P2_MIO_Digin_Long, P2_MIO_Digout, P2_MIO_Digout_Long

Valid for

MIO-4 Rev. E, MIO-4-ET1 Rev. E, MIO-D12 Rev. E

Example

```
#Include ADwinPro_All.inc

Event:
  Rem read return bits 31:00 from the latch
  Par_1 = P2_MIO_Get_Digout_Long(1)
```

3.4 Pro II: Analog Input Modules

This section describes instructions which apply to Pro II modules with analog inputs:

– P2_ADC (page 37)
– P2_ADC24 (page 38)
– P2_ADC_Read_Limit (page 39)
– P2_ADC_Set_Limit (page 41)
– P2_Read_ADC (page 42)
– P2_Read_ADC24 (page 43)
– P2_Read_ADC_SConv (page 44)
– P2_Read_ADC_SConv24 (page 45)
– P2_SE_Diff (page 46)
– P2_Seq_Init (page 47)
– P2_Seq_Read (page 50)
– P2_Seq_Read24 (page 51)
– P2_Seq_Read_Packed (page 53)
– P2_Seq_Start (page 54)
– P2_Seq_Wait (page 55)
– P2_Set_Mux (page 56)
– P2_Start_Conv (page 57)
– P2_Wait_EOC (page 58)
– P2_Wait_Mux (page 59)
– P2_Burst_CRead_Unpacked1 (page 60)
– P2_Burst_CRead_Unpacked2 (page 62)
– P2_Burst_CRead_Unpacked4 (page 64)
– P2_Burst_CRead_Unpacked8 (page 66)
– P2_Burst_Init (page 76)
– P2_Burst_Read_Index (page 80)
– P2_Burst_Read (page 82)
– P2_Burst_Read_Unpacked1 (page 84)
– P2_Burst_Read_Unpacked2 (page 86)
– P2_Burst_Read_Unpacked4 (page 88)
– P2_Burst_Read_Unpacked8 (page 90)
– P2_Burst_Reset (page 92)
– P2_Burst_Start (page 94)
– P2_Burst_Status (page 95)
– P2_Burst_Stop (page 97)
– P2_Set_Average_Filter (page 99)
– P2_ADCF (page 100)
– P2_ADCF24 (page 101)
– P2_ADCF_Mode (page 102)
– P2_ADCF_Read_Limit (page 105)

– P2_ADCF_Set_Limit (page 106)
– P2_ADCF_Reset_Min_Max (page 107)
– P2_ADCF_Read_Min_Max4 (page 108)
– P2_ADCF_Read_Min_Max8 (page 110)
– P2_Read_ADCF (page 112)
– P2_Read_ADCF24 (page 113)
– P2_Read_ADCF4 (page 114)
– P2_Read_ADCF4_24B (page 115)
– P2_Read_ADCF8 (page 116)
– P2_Read_ADCF8_24B (page 117)
– P2_Read_ADCF4_Packed (page 118)
– P2_Read_ADCF8_Packed (page 119)
– P2_Read_ADCF32 (page 120)
– P2_Read_ADCF_SConv (page 121)
– P2_Read_ADCF_SConv24 (page 122)
– P2_Read_ADCF_SConv32 (page 123)
– P2_Set_Gain (page 124)
– P2_Start_ConvF (page 125)
– P2_Wait_EOCF (page 126)

In the Instruction List sorted by Module Types (annex A.2) you will find which of the functions corresponds to the *ADwin-Pro II* modules.

P2_ADC

P2_ADC runs a complete conversion on an ADC of the specified module. The return value has a resolution of 16 bit.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = P2_ADC(module, input_no)
```

Parameters

module	Selected module address (1…15).	LONG
input_no	Number of the analog input (1…8 or 1…32).	LONG
ret_val	Result of the conversion (0…65535).	LONG

Notes

P2_ADC is characterized by a sequence of several instructions:

P2_Set_Mux	→ …	→ P2_Start_Conv	→ P2_Wait_EOC	→ P2_Read_ADC
set multiplexer to an input channel	wait for multiplexer settling	Start A/D conversion	Wait for end of conversion	Read out the converted value

If the multiplexer is set to the same channel as the previous conversion, the settling time is skipped automatically.

If you want to set the gain, use **P2_Set_Mux**.

See also

P2_ADC24, P2_ADC_Read_Limit, P2_ADC_Set_Limit, P2_Read_ADC, P2_Start_Conv, P2_Wait_EOC

Valid for

AIn-16/18-8B Rev. E, AIn-16/18-C Rev. E, AIn-32/18 Rev. E, AIn-8/18 Rev. E, AIn-8/18-8B Rev. E, MIO-4 Rev. E, MIO-4-ET1 Rev. E

Example

```
#Include ADwinPro_All.Inc
Dim value As Long

Event:
  value = P2_ADC(1, 4)      'maesure 16Bit value at analog input 4
```

P2_ADC24

P2_ADC24 runs a complete conversion on an ADC of the specified module. The return value is formatted to 24 bit.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = P2_ADC24(module, input_no)
```

Parameters

module	Selected module address (1…15).	LONG
input_no	Number of the analog input (1…8 or 1…32).	LONG
ret_val	Result of the conversion (0…16777215 = 2^{24}-1).	LONG

Notes

P2_ADC24 is characterized by a sequence of several instructions:

P2_Set_Mux	→ …	→ P2_Start_Conv	→ P2_Wait_EOC	→ P2_Read_ADC24
set multiplexer to an input channel	wait for multiplexer settling	Start A/D conversion	Wait for end of conversion	Read out the converted value

If the multiplexer is set to the same channel as the previous conversion, the settling time is skipped automatically.

If you want to set the gain, use **P2_Set_Mux**.

If a measurement value has a resolution less than 24 bit, the "missing" bits in the return value „fehlenden“ are filled with zeros.
As an example, the measurement value of an 18 bit ADC is located in the bits 6…23 of the return value; the measurement value is shifted to the left by 6 bits and the bits 0…5 are zeros.

Bit no.	31…24	23…6	05…00
Content	0	18 bit meas. value	0

See also

P2_ADC, P2_ADC_Read_Limit, P2_ADC_Set_Limit, P2_Read_ADC24, P2_Start_Conv, P2_Wait_EOC

Valid for

AIn-16/18-8B Rev. E, AIn-16/18-C Rev. E, AIn-32/18 Rev. E, AIn-8/18 Rev. E, AIn-8/18-8B Rev. E, MIO-4 Rev. E, MIO-4-ET1 Rev. E

Example

```
#Include ADwinPro_All.Inc
Dim value As Long

Event:
  REM measure 24Bit value at analog input 4
  value = P2_ADC24(1, 4)
```

P2_ADC_Read_Limit

P2_ADC_Read_Limit returns the flags of limit-overrun and -underrun from 16 ADCs of the specified module.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = P2_ADC_Read_Limit(module,ch_group)
```

Parameters

module	Selected module address (1…15).	LONG
ch_group	Group of 16 channels each: 1: Channels 1…16 2: Channels 17…32	
ret_val	Bit pattern representing the limit-overrun and -underrun flags:	LONG

overrun of upper limit								
ch_group	Bit No.	31	30	29	…	18	17	16
1	Channel no.	16	15	14	…	3	2	1
2	Channel no.	32	31	30	…	19	18	17

underrun of lower limit								
ch_group	Bit No.	15	14	13	…	2	1	0
1	Channel no.	16	15	14	…	3	2	1
2	Channel no.	32	31	30	…	19	18	17

Notes

The limits are set with **P2_ADC_Set_Limit**.

Reading the flags resets all flags of a channel group to zero.

We recommend to read the flags in the **Init:** section once, as to ensure all flags be reset. This is even more important with an externally controlled process.

See also

P2_ADC, P2_ADC24, P2_ADC_Set_Limit

Valid for

AIn-16/18-8B Rev. E, AIn-16/18-C Rev. E, AIn-32/18 Rev. E, AIn-8/18 Rev. E, AIn-8/18-8B Rev. E, MIO-4 Rev. E, MIO-4-ET1 Rev. E

Example

```
#Include ADwinPro_All.Inc
Dim flags As Long

Init:
  P2_SE_Diff(1,1)           'Differential inputs
  P2_ADC_Set_Limit(1, 2, 42768,256) 'Set limits for channel 2
  P2_Seq_Init(1,3,0,10b,0) 'continuous max mode, Kanal 2
  P2_Seq_Start(1)           'Start sequence control
  P2_Seq_Wait(1)
  flags = P2_ADC_Read_Limit(1,1) 'Reset flags by reading

Event:
  flags = P2_ADC_Read_Limit(1,1) 'read flags of channels 1…16
  If ((flags And 10b) = 10b) Then
    REM limit-underrun on channel 2
    Inc Par_1
  EndIf
  If ((flags And 20000h) = 20000h) Then
    REM limit-overrun on channel 2
    Inc Par_2
  EndIf
```

P2_ADC_Set_Limit

P2_ADC_Set_Limit sets the upper and lower limit for one ADC of the specified module.

Syntax

```
#Include ADwinPro_All.Inc
P2_ADC_Set_Limit(module, input_no, high, low)
```

Parameters

module	Selected module address (1…15).	LONG
input_no	Number of the analog input (1…8 or 1…32).	LONG
high	Upper limit (0…65535) of the channel. Default: 65535.	LONG
low	Lower limit (0…65535) of the channel. Default: 0.	LONG

Notes

If a converted value exceeds the upper limit, the channel's flag is set. **P2_ADC_Read_Limit** reads and thus resets the flags.

The same way a channel's flag is set for a converted value falling below the lower limit.

A limit-overrun or -underrun does not trigger an event signal.

See also

P2_ADC, P2_ADC24, P2_ADC_Read_Limit

Valid for

AIn-16/18-8B Rev. E, AIn-16/18-C Rev. E, AIn-32/18 Rev. E, AIn-8/18 Rev. E, AIn-8/18-8B Rev. E, MIO-4 Rev. E, MIO-4-ET1 Rev. E

Example

```
#Include ADwinPro_All.Inc
Dim flags As Long

Init:
  P2_SE_Diff(1,1)           'Differential inputs
  P2_ADC_Set_Limit(1, 2, 42768,256) 'Set limits for channel 2
  P2_Seq_Init(1,3,0,10b,0) 'continuous max mode, Kanal 2
  P2_Seq_Start(1)           'Start sequence control
  P2_Seq_Wait(1)
  flags = P2_ADC_Read_Limit(1,1) 'Reset flags by reading

Event:
  flags = P2_ADC_Read_Limit(1,1) 'read flags of channels 1…16
  If ((flags And 10b) = 10b) Then
    REM limit-underrun on channel 2
    Inc Par_1
  EndIf
  If ((flags And 20000h) = 20000h) Then
    REM limit-overrun on channel 2
    Inc Par_2
  EndIf
```

P2_Read_ADC

P2_Read_ADC returns the conversion result from an ADC of the specified module. The return value has a resolution of 16 bit.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = P2_Read_ADC(module)
```

Parameters

module	Selected module address (1…15).	LONG
ret_val	Value from the ADC (0…65535).	LONG

Notes

-/-

See also

P2_ADC, P2_ADC24, P2_Start_Conv, P2_Wait_EOC, P2_ADC_Read_Limit, P2_ADC_Set_Limit, P2_Read_ADC_SConv

Valid for

AIn-16/18-8B Rev. E, AIn-16/18-C Rev. E, AIn-32/18 Rev. E, AIn-8/18 Rev. E, AIn-8/18-8B Rev. E, MIO-4 Rev. E, MIO-4-ET1 Rev. E

Example

```
#Include ADwinPro_All.Inc
Dim value1 As Long          'Declaration

Event:
  P2_Set_Mux(1,0100000010b)'set MUX to input 3, gain 2
  Rem wait for multiplexer settling, here 4 µs
  P2_Sleep(400)
  P2_Start_Conv(1)          'Start AD conversion
  P2_Wait_EOC(1)            'Wait for end of conversion
  value1 = P2_Read_ADC(1)   'Read value from ADC
```

P2_Read_ADC24

P2_Read_ADC24 returns the conversion result from an ADC of the specified module. The return value has a resolution of 24 bit.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = P2_Read_ADC24(module)
```

Parameters

module	Selected module address (1…15).	LONG
ret_val	Value from the ADC (0…16777215 = 2^{24}-1).	LONG

Notes

If a measurement value has a resolution less than 24 bit, the "missing" bits in the return value „fehlenden“ are filled with zeros.
As an example, the measurement value of an 18 bit ADC is located in the bits 6…23 of the return value; the measurement value is shifted to the left by 6 bits and the bits 0…5 are zeros.

Bit no.	31…24	23…6	05…00
Content	0	18 bit meas. value	0

See also

P2_ADC24, P2_Start_Conv, P2_Wait_EOC, P2_ADC_Read_Limit, P2_ADC_Set_Limit, P2_Read_ADC_SConv24

Valid for

AIn-16/18-8B Rev. E, AIn-16/18-C Rev. E, AIn-32/18 Rev. E, AIn-8/18 Rev. E, AIn-8/18-8B Rev. E, MIO-4 Rev. E, MIO-4-ET1 Rev. E

Example

```
#Include ADwinPro_All.Inc
Dim value1 As Long          'Declaration

Event:
  P2_Set_Mux(1,0100000010b)'set MUX to input 3, gain 2
  Rem wait for multiplexer settling, here 4 µs
  P2_Sleep(400)
  P2_Start_Conv(1)          'Start AD conversion
  P2_Wait_EOC(1)            'Wait for end of conversion
  value1 = P2_Read_ADC24(1)'Read 24 bit value from the ADC
```

P2_Read_ADC_SConv

P2_Read_ADC_SConv reads out the conversion result from an ADC of the specified module and immediately starts a new conversion.

The return value has a resolution of 16 bit.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = P2_Read_ADC_SConv(module)
```

Parameters

module	Selected module address (1…15).	LONG
ret_val	Value from the ADC (0…65535).	LONG

See also

P2_ADC, P2_ADC24, P2_Read_ADC, P2_Read_ADC_SConv24, P2_Start_Conv, P2_Wait_EOC

Valid for

AIn-16/18-8B Rev. E, AIn-16/18-C Rev. E, AIn-32/18 Rev. E, AIn-8/18 Rev. E, AIn-8/18-8B Rev. E, MIO-4 Rev. E, MIO-4-ET1 Rev. E

Example

```
#Include ADwinPro_All.Inc
Dim i As Long
Dim Data_1[1000] As Long  'Declaration

Init:
  i=1
  P2_Set_Mux(1,0100000010b)'set MUX to input 3, gain 2
  Rem wait for multiplexer settling, here 4 µs
  P2_Sleep(400)
  P2_Start_Conv(1)         'start A/D conversion

Event:
  P2_Wait_EOC(1)
  Data_1[i] = P2_Read_ADC_SConv(1)'read and start A/D converter
  Inc(i)                   'increment index
  If (i=1001) Then End     'End process after 1000 values
```

P2_Read_ADC_SConv24

P2_Read_ADC_SConv24 reads the conversion result from an ADC of the specified module and immediately starts a new conversion.

The return value has a resolution of 24 bit.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = P2_Read_ADC_SConv24(module)
```

Parameters

module	Selected module address (1…15).	LONG
ret_val	Value from the ADC (0…16777215 = 2^{24}-1).	LONG

See also

P2_ADC, P2_ADC24, P2_Read_ADC, P2_Read_ADC_SConv, P2_Start_Conv, P2_Wait_EOC

Valid for

AIn-16/18-8B Rev. E, AIn-16/18-C Rev. E, AIn-32/18 Rev. E, AIn-8/18 Rev. E, AIn-8/18-8B Rev. E, MIO-4 Rev. E, MIO-4-ET1 Rev. E

Example

```
#Include ADwinPro_All.Inc
Dim i As Long
Dim Data_1[1000] As Long  'Declaration

Init:
  i=1
  P2_Set_Mux(1,0100000010b)'set MUX to input 3, gain 2
  Rem wait for multiplexer settling, here 4 µs
  P2_Sleep(400)
  P2_Start_Conv(1)         'start A/D conversion

Event:
  P2_Wait_EOC(1)
  Data_1[i] = P2_Read_ADC_SConv24(1)'Read + start A/D converter
  Inc(i)                   'increment index
  If (i=1001) Then End     'End process after 1000 values
```

P2_SE_Diff

P2_SE_Diff sets the operating mode single ended or differential for all analog inputs of the specified module.

Syntax

```
#Include ADwinPro_All.Inc
P2_SE_Diff(module,mode)
```

Parameters

module	Selected module address (1…15).	LONG
mode	Operating mode of analog inputs. 0: single ended. 1: differential (default).	LONG

Notes

The operating mode single ended provides 32 inputs, in differential mode there are 16 inputs. After power-up of the device all inputs are set to differential mode.

See also

P2_ADC

Valid for

AIn-16/18-8B Rev. E, AIn-16/18-C Rev. E, AIn-32/18 Rev. E, MIO-4 Rev. E, MIO-4-ET1 Rev. E

Example

```
#Include ADwinPro_All.Inc

Init:
  P2_SE_Diff(1,0)           'set module address 1 to s.e.

  P2_SE_Diff(2,1)           'set module address 2 to diff.
```

P2_Seq_Init

P2_Seq_Init initializes the sequential control of the specified module.

These settings are done: Operating mode, gain factor, channel selection and multiplexer settling time (similar for all channels).

Syntax

```
#Include ADwinPro_All.Inc
P2_Seq_Init(module, mode, gain, channels, mux_time)
```

Parameters

module — Selected module address (1…15). LONG

mode — Operating mode of the sequential control: LONG
0: Single conversion (default), no sequential control.
1: Mode "single shot", single conversion cycle.
2: Mode "continuous", continuous conversion.
3: Mode "continuous max" conversion using max. speed.

gain — Gain factor (modes 1 … 3 only): LONG
0 factor = 1, voltage range -10V…+10V.
1 factor = 2, voltage range -5V…+5V.
2 factor = 4, voltage range -2.5V…+2.5V.
3 factor = 8, voltage range 1.25V…+1.25V.

channels — Bit pattern to select the channels for conversion. LONG
Bit = 0: Don't convert.
Bit = 1: Do conversion.

Bit no.	31	…	7	…	2	1	0
Channel no.	32	…	8	…	3	2	1

mux_time — Number of time units, which sets the settling time of the sequential control: LONG
0: Standard waiting time (125 = 2.5µs).
125…2^31: Waiting time in units of 20ns.

Notes

After power-up mode 0 is active.

Modes 1…3 activate the sequential control of the module, single conversions with **P2_ADC** are disabled then. The sequential control consecutively runs a conversion on several channels. The sequential control is always related to those channels being selected by **channels**.

The modes differ in the following items:

Mode	Kind of conversion
0 Standard:	Single conversion at one channel without sequential control, see **P2_ADC**.
1 single shot:	The sequential control is started by **P2_Seq_Start**; it ends as soon as each of the selected channels is converted once. The end of the sequential control is queried with **P2_Seq_Wait** and measurement values are read with **P2_Seq_Read**.

2 continuous:	The sequential control converts all selected channels for each process cycle. The conversion is started with **P2_Seq_Start** as last instruction in section **Init:**. The end of conversion (for all channels) is automatically synchronized with the beginning of the next process cycle. Therefore all measurement values can–and should be–read with **Seq_Read** at the beginning of each process cycle .
3 continuous max:	The sequential control converts the selected channels continuously, providing new measurement values all the time. That is, conversion and process cycle run non-synchronously. The conversion is started with **P2_Seq_Start**. Inside a process cycle **P2_Seq_Read** just reads the newest measurement value.

Please note for mode 2 (continuous): The synchronization happens only once and is only valid for the set cycle time (**Processdelay**). If the process timing changes, e.g. by changing the cycle time, the synchronization is lost. The consequence is, measurement values are being read to early and thus multiple, or measurement values are lost, because they are already overwritten by new values before reading.

In a channel group any module channel may be selected. The channels of a group are automatically sorted in ascending order of channel numbers, that is the sequential control converts the channel with the smallest number first.

The status and read instructions always and solely refer to the group of selected channels.

With 32-input modules the inputs must be set as single ended or differential with **P2_SE_Diff**.

If the internal resistance of the signal's voltage source is too great, the pre-set multiplexer settling time is too short for an accurate measurement. You can change the multiplexer settling time with the parameter `mux_time`.

The settling time influences the accuracy of the measurement at a high rate. Shorter settling time tends to result in less accurate measurement values and longer settling time in more accurate values.

The settling time is calculated according to the following formula:

$$\text{settling time} = \texttt{muxtime} \cdot 20\,\text{ns} + \text{conversion time}$$

The values of conversion time and pre-set settling time are given in the hardware documentation of the module.

See also

P2_ADC, P2_Seq_Read, P2_Seq_Read24, P2_Seq_Read_Packed, P2_Seq_Start, P2_Seq_Wait

Valid for

AIn-16/18-8B Rev. E, AIn-16/18-C Rev. E, AIn-32/18 Rev. E, AIn-8/18 Rev. E, AIn-8/18-8B Rev. E, MIO-4 Rev. E, MIO-4-ET1 Rev. E

Example

```
#Define module 1
#Include ADwinPro_All.Inc

Dim Data_1[16] As Long At DM_Local

Init:
  P2_SE_Diff(module,0)      'set inputs to single ended
  REM sequential control: continuous Mode, gain 1
  REM odd-numbered channels of module AIN-32
  REM standard settling time
  P2_Seq_Init(module,3,0,55555555h,0)
  REM start measuring sequence on modules 1 and 3
  P2_Seq_Start(Shift_Left(1,module-1))
  P2_Seq_Wait(module)       'wait until all selected channels
                            'are converted once

Event:
  REM read values and copy into Data_1
  P2_Seq_Read(module,16,Data_1,1)
```

P2_Seq_Read

P2_Seq_Read reads a given number of values (16 Bit) from the specified module and copies them into a destination array.

Each array element holds 1 measurement value.

Syntax

```
#Include ADwinPro_All.Inc
P2_Seq_Read(module,count,array[],array_idx)
```

Parameters

module	Selected module address (1…15).	LONG
count	Even number (2…32) of read measurement values. An odd number is not allowed.	LONG
array[]	Destination array to store the measurement values.	ARRAY LONG ~~FLOAT~~
array_idx	Destination start index: first array element to store a value in (1…n).	LONG

Notes

You can only reasonably use this instruction if the sequential control of the module has been activated before with **P2_Seq_Init**.

The measurement values of the channel group are copied into the destination array starting ascending from the smallest channel number.

See also

P2_Seq_Init, P2_Seq_Read24, P2_Seq_Read_Packed, P2_Seq_Start, P2_Seq_Wait

Valid for

AIn-16/18-8B Rev. E, AIn-16/18-C Rev. E, AIn-32/18 Rev. E, AIn-8/18 Rev. E, AIn-8/18-8B Rev. E, MIO-4 Rev. E, MIO-4-ET1 Rev. E

Example

```
#Include ADwinPro_All.Inc
#Define module 1
Dim Data_1[16] As Long At DM_Local

Init:
  P2_SE_Diff(module,0)      'set inputs to single ended
  REM sequential control: continuous Mode, gain 1
  REM odd-numbered channels, standard settling time
  P2_Seq_Init(module,3,0,55555555h,0)
  P2_Seq_Start(Shift_Left(1, module-1))'start sequential control
  P2_Seq_Wait(module)       'wait until all selected channels
                            'are converted once

Event:
  Rem copy current values from the module into Data_1
  P2_Seq_Read(module,16,Data_1,1)
```

P2_Seq_Read24

P2_Seq_Read24 reads a given number of values (18 Bit) from the specified module and copies them into a destination array.

Each array element holds 1 measurement value.

Syntax

```
#Include ADwinPro_All.Inc
P2_Seq_Read24(module,count,array[],array_idx)
```

Parameters

module	Selected module address (1…15).	LONG
count	Number (1…32) of read measurement values.	LONG
array[]	Destination array to store the measurement values.	ARRAY LONG ~~FLOAT~~
array_idx	Destination start index: first array element to store a value in (1…n).	LONG

Notes

You can only reasonably use this instruction if the sequential control of the module has been activated before with **P2_Seq_Init**.

The measurement values of the channel group are copied into the destination array starting ascending from the smallest channel number.

If a measurement value has a resolution less than 24 bit, the "missing" bits in the return value are filled with zeros.
As an example, the measurement value of an 18 bit ADC is located in the bits 6…23 of the return value; the measurement value is shifted to the left by 6 bits and the bits 0…5 are zeros.

Bit no.	31…24	23…6	05…00
Content	0	18 bit meas. value	0

See also

P2_Seq_Init, P2_Seq_Read, P2_Seq_Read_Packed, P2_Seq_Start, P2_Seq_Wait

Valid for

AIn-16/18-8B Rev. E, AIn-16/18-C Rev. E, AIn-32/18 Rev. E, AIn-8/18 Rev. E, AIn-8/18-8B Rev. E, MIO-4 Rev. E, MIO-4-ET1 Rev. E

Example

```
#Include ADwinPro_All.Inc
#Define module 1
Dim Data_1[16] As Long At DM_Local

Init:
  P2_SE_Diff(module,0)      'set inputs to single ended
  REM sequential control: continuous Mode, gain 1
  REM odd-numbered channels, standard settling time
  P2_Seq_Init(module,3,0,55555555h,0)
  P2_Seq_Start(Shift_Left(1, module-1))'start sequential control
  REM wait until all selected channels are converted once
  P2_Seq_Wait(module)       '

Event:
  Rem Copy current values from the module into Data_1
  P2_Seq_Read24(module,16,Data_1,1)
```

P2_Seq_Read_Packed

P2_Seq_Read_Packed reads an even number of value pairs (16 Bit) from the specified module and copies them into a destination array.

Each array element holds 2 measurement values.

Syntax

```
#Include ADwinPro_All.Inc
P2_Seq_Read_Packed(module,count,array[],array_idx)
```

Parameters

module	Selected module address (1…15).	LONG
count	Number of value pairs to read (1…16).	LONG
array[]	Destination array to store the measurement values.	ARRAY LONG ~~FLOAT~~
array_idx	Destination start index: first array element to store a value in (1…n).	LONG

Notes

You can only reasonably use this instruction if the sequential control of the module has been activated before with **P2_Seq_Init**.

The measurement values of the channel group are copied into the destination array starting ascending from the smallest channel number and in pairs. An array element contains the value of the channel with the smaller number in the lower word, the value of the higher channel number in the upper word.

See also

P2_Seq_Init, P2_Seq_Read, P2_Seq_Read24, P2_Seq_Start, P2_Seq_Wait

Valid for

AIn-16/18-8B Rev. E, AIn-16/18-C Rev. E, AIn-32/18 Rev. E, AIn-8/18 Rev. E, AIn-8/18-8B Rev. E, MIO-4 Rev. E, MIO-4-ET1 Rev. E

Example

```
#Include ADwinPro_All.Inc

Dim Data_1[8], Data_2[8] As Long At DM_Local

Init:
  P2_SE_Diff(1,0)           'set module 1 inputs to single ended
  P2_SE_Diff(5,0)           'set module 5 inputs to single ended
  REM modules 1+5: Sequential control continuous mode, gain 1,
  REM even-numbered channels (2…16), standard settling time
  P2_Seq_Init(1,3,0,0AAAAh,0)
  P2_Seq_Init(5,3,0,0AAAAh,0)
  P2_Seq_Start(10001b)      'start sequence control on modules 1+5
  P2_Seq_Wait(1)            'wait until all selected channels
                            'are converted once

Event:
  REM read 16 values and copy into Data_1 and Data_2
  P2_Seq_Read_Packed(1,8,Data_1,1)
  P2_Seq_Read_Packed(5,8,Data_2,1)
```

P2_Seq_Start

P2_Seq_Start starts the sequence control on all selected modules at the same time.

Syntax

```
#Include ADwinPro_All.Inc
P2_Seq_Start(module_pattern)
```

Parameters

module_pattern	Bit pattern to set the module addresses: Bit = 0: Ignore module address. Bit = 1: Select module address.	LONG

Bit pattern	31:15	14	13	...	01	00
Module address	–	15	14	...	2	1

Notes

If a module without sequence control is selected, the instruction may cause unpredictable consequences.

See also

P2_Seq_Init, P2_Seq_Read, P2_Seq_Read24, P2_Seq_Read_Packed, P2_Seq_Wait

Valid for

AIn-16/18-8B Rev. E, AIn-16/18-C Rev. E, AIn-32/18 Rev. E, AIn-8/18 Rev. E, AIn-8/18-8B Rev. E, MIO-4 Rev. E, MIO-4-ET1 Rev. E

Example

```
#Include ADwinPro_All.Inc
#Define module 4           'Module address

Dim Data_1[32] As Long At DM_Local
Dim i As Long

Init:
  P2_SE_Diff(module,0)      'set inputs to single ended
  REM set sequential control to single shot,
  REM gain 1, all channels,
  REM standard settling time
  P2_Seq_Init(module,1,0,0FFFFFFFFh,0)
  P2_Seq_Start(Shift_Left(1,module-1)) 'start sequential control

Event:
  P2_Seq_Wait(module)       'wait for end of conversion
  P2_Seq_Read(module,32,Data_1,1) 'read 32 channels ...
  For i=1 To 32
    REM convert digit to Volt and store
    Data_1[i] = (Data_1[i]-32768)*20/65536
  Next i
  P2_Seq_Start(Shift_Left(1,module-1)) 'start next sequence
```

P2_Seq_Wait

P2_Seq_Wait waits until the sequence control has converted and stored all channels of the channel group on the specified module.

Syntax

```
#Include ADwinPro_All.Inc
P2_Seq_Wait(module)
```

Parameters

module	Selected module address (1…15).	LONG

Notes

If sequence controls have been started on several modules at the same time (and with the same parameter values), they will end at the same time, too.

See also

P2_Seq_Init, P2_Seq_Read, P2_Seq_Read24, P2_Seq_Read_Packed, P2_Seq_Start

Valid for

AIn-16/18-8B Rev. E, AIn-16/18-C Rev. E, AIn-32/18 Rev. E, AIn-8/18 Rev. E, AIn-8/18-8B Rev. E, MIO-4 Rev. E, MIO-4-ET1 Rev. E

Example

```
#Include ADwinPro_All.Inc
#Define module 4           'Module address

Dim Data_1[32] As Long At DM_Local
Dim Data_2[32] As Float At DM_Local
Dim i As Long

Init:
  Processdelay=100000
  P2_SE_Diff(module,0)      'set inputs to single ended
  REM set sequential control to single shot,
  REM gain 1, all channels,
  REM settling time 0,5 µs
  P2_Seq_Init(module,3,0,0FFFFFFFFh,50)
  P2_Seq_Start(Shift_Left(1,module-1)) 'start sequence control

Event:
  P2_Seq_Wait(module)       'wait for end of conversion
  P2_Seq_Read(module,32,Data_1,1) 'read 32 channels ...
  For i=1 To 32
    REM convert digit to Volt and store
    Data_2[i] = (Data_1[i]-32768)*20/65536
  Next i
  P2_Seq_Start(Shift_Left(1,module-1)) 'start next sequence
```

P2_Set_Mux

P2_Set_Mux sets the multiplexer of the specified module to the selected input and to the selected gain.

Syntax

```
#Include ADwinPro_All.Inc
P2_Set_Mux(module,pattern)
```

Parameters

module	Selected module address (1…15).	LONG
pattern	Bit pattern for multiplexer settings (see table); bits 8…9 do gain settings, bits 0…4 set the number of the input channel.	LONG

Bit no.								
31:7	9	8	7…5	4	3	2	1	0
no function	gain		–	Multiplexer input				
	1 = 00b		–	input 1: 00000b				
	2 = 01b			input 2: 00001b				
	4 = 10b			…				
	8 = 11b			input 32: 11111b				

Notes

Combine the adequate bit combinations for gain and multiplexer input to find the wanted multiplexer settings.

You may set the bits of parameter pattern in binary format or convert them into hexadecimal or decimal format. For hex or binary formats, please note the character suffix **h** and **b**.

Please note the required multiplexer settling time (see hardware documentation). Make sure that this settling time passes at minimum between setting the multiplexer and start of conversion.

See also

P2_ADC, P2_Start_Conv, P2_Wait_EOC, P2_Read_ADC

Valid for

AIn-16/18-8B Rev. E, AIn-16/18-C Rev. E, AIn-32/18 Rev. E, AIn-8/18 Rev. E, AIn-8/18-8B Rev. E, MIO-4 Rev. E, MIO-4-ET1 Rev. E

Example

```
#Include ADwinPro_All.Inc
Dim value1 As Long          'Declaration

Event:
  P2_Set_Mux(1,0100000010b)'set MUX to input 3, gain 2
  Rem wait for multiplexer settling, here 4 µs
  P2_Sleep(400)
  P2_Start_Conv(1)          'start AD conversion
  P2_Wait_EOC(1)            'Wait for end of conversion
  value1 = P2_Read_ADC(1)   'Read value from ADC
```

P2_Start_Conv

P2_Start_Conv starts the conversion on the specified module.

Syntax

```
#Include ADwinPro_All.Inc
P2_Start_Conv(module)
```

Parameters

module	Selected module address (1…15).	LONG

Notes

The conversion can be started synchronously to actions on other modules with the instruction **P2_Sync_All**.

See also

P2_ADC, P2_ADC24, P2_Read_ADC, P2_Wait_EOC, P2_Sync_All

Valid for

AIn-16/18-8B Rev. E, AIn-16/18-C Rev. E, AIn-32/18 Rev. E, AIn-8/18 Rev. E, AIn-8/18-8B Rev. E, MIO-4 Rev. E, MIO-4-ET1 Rev. E

Example

```
#Include ADwinPro_All.Inc
Dim value As Long          'Declaration

Event:
  P2_Set_Mux(1,0100000010b)'set MUX to input 3, gain 2
  Rem wait for multiplexer settling, here 4 µs
  P2_Sleep(400)
  P2_Start_Conv(1)         'start AD conversion
  P2_Wait_EOC(1)           'Wait for end of conversion
  value = P2_Read_ADC(1)   'Read value from ADC
```

P2_Wait_EOC

P2_Wait_EOC waits for the end of conversion on the specified module.

Syntax

```
#Include ADwinPro_All.Inc
P2_Wait_EOC(module)
```

Parameters

module	Selected module address (1…15).	LONG

Notes

- / -

See also

P2_ADC, P2_ADC24, P2_Start_Conv, P2_Read_ADC

Valid for

AIn-16/18-8B Rev. E, AIn-16/18-C Rev. E, AIn-32/18 Rev. E, AIn-8/18 Rev. E, AIn-8/18-8B Rev. E, MIO-4 Rev. E, MIO-4-ET1 Rev. E

Example

```
#Include ADwinPro_All.Inc
Dim value As Long          'Declaration

Event:
  P2_Set_Mux(1,0100000010b)'set MUX to input 3, gain 2
  Rem wait for multiplexer settling, here 4 µs
  P2_Sleep(400)
  P2_Start_Conv(1)         'start AD conversion
  P2_Wait_EOC(1)           'Wait for end of conversion
  value = P2_Read_ADC(1)   'Read value from ADC
```

P2_Wait_Mux

P2_Wait_Mux waits for the end of the multiplexer settling on the specified module.

Syntax

```
#Include ADwinPro_All.Inc
P2_Wait_Mux(module)
```

Parameters

module	Selected module address (1…15).	LONG

Notes

If you set the multiplexer to a different channel using **P2_Set_Mux** it takes a defined time until the multiplexer is settled. The instruction **P2_Wait_Mux** waits until this moment, so immediately afterwards you can start a A/D conversion.

If the multiplexer is set to the same channel as the previous conversion, or if the multiplexer has already settled, the waiting time is skipped automatically.

See also

P2_ADC, P2_ADC24, P2_Set_Mux, P2_Start_Conv, P2_Read_ADC

Valid for

AIn-16/18-8B Rev. E, AIn-16/18-C Rev. E, AIn-32/18 Rev. E, AIn-8/18 Rev. E, AIn-8/18-8B Rev. E, MIO-4 Rev. E, MIO-4-ET1 Rev. E

Example

```
#Include ADwinPro_All.Inc
#Define module 1

Init:
  P2_Seq_Init(module,0,0,0,0) 'switch off sequential control mode
  P2_Set_Mux(module,0b)       'set multiplexer to input 1, gain 1
  P2_Wait_Mux(module)
  P2_Start_Conv(module)       'start AD conversion
  Processdelay=30000          'cycle-time 0.1 ms

Event:
  P2_Set_Mux(module,0100000001b) 'set MUX to input 2, gain 2
  P2_Wait_EOC(module)         'wait for end of conversion
  Par_1 = P2_Read_ADC(module) 'read channel value 1 from the ADC
  P2_Wait_Mux(module)         'wait for end of settling time
  P2_Start_Conv(module)       'start AD conversion
  P2_Set_Mux(module,0b)       'set MUX to input 1, gain 1
  P2_Wait_EOC(module)         'wait for end of conversion
  Par_2 = P2_Read_ADC(module) 'read channel value 2 from the ADC

  P2_Wait_Mux(module)         'wait for end of settling time
  P2_Start_Conv(module)       'start AD conversion
```

P2_Burst_CRead_Unpacked1

P2_Burst_CRead_Unpacked1 copies an amount of the last measured values of a channel from the memory of the specified module into an array.

Syntax

```
#Include ADwinPro_All.Inc
P2_Burst_CRead_Unpacked1(module, count, array[],
    array_idx, flowrate)
```

Parameters

module	Specified module address (1…15).	LONG
count	Amount of measurement values to be transferred. The amount must be divisible by 8.	LONG
array[]	Destination array, where the measurement values are transferred. No float type and no FIFO array allowed.	ARRAY LONG ~~FLOAT~~
array_idx	Destination start index: Array element from which measurement values are stored.	LONG
flowrate	Evaluation for low-priority processes only: Parameter for data throughput. 1: slow. 2: medium. 3: fast.	LONG

Notes

On modules AIn-F-x-16, the instruction is available since revision E04.

The instruction be used if a continuous burst sequence was initialized with 1 channel (see **P2_Burst_Init**, parameters mode, channels).

The instruction reads the amount of count measurement values that are stored in the module memory. count should be lower by the factor 2 than the number of measurements (buffer_count), specified in **P2_Burst_Init**

The instruction stores the measurement values one after the other in the elements of the destination array.

In high-priority processes the maximum data throughput is used automatically; the parameter flowrate must be indicated all the same.

The higher the data troughput–in a low priority process only–is selected, the more it may happen that a process of higher priority is delayed.

See also

P2_Burst_Init, P2_Burst_CRead_Unpacked2, P2_Burst_CRead_Unpacked4, P2_Burst_CRead_Unpacked8, P2_Burst_Start, P2_Burst_Status

Valid for

AIn-F-4/14 Rev. E, AIn-F-4/16 Rev. E, AIn-F-8/14 Rev. E, AIn-F-8/16 Rev. E

Example

```
#Include ADwinPro_All.Inc
#Define module 4

Dim Data_1[1000] As Long
Dim pattern As Long

Init:
  REM Initiate cont. burst sequence for channel 1 using 20ns
  REM period duration, save 2^26 samples from address 0.
  P2_Burst_Init (module,1,0,67108864,1,010b)
  REM Start burst sequence
  pattern = Shift_Left(1,module-1) 'access single module only
  P2_Burst_Start(pattern)
  Processdelay=10000000

Event:
  REM Read last 1000 samples from channel (slowly) and store
  REM in Data_1
  P2_Burst_CRead_Unpacked1(module,1000,Data_1,1,1)
```

P2_Burst_CRead_Unpacked2

P2_Burst_CRead_Unpacked2 copies an amount of the last measurement values of 2 channels from the memory of the specified module into 2 arrays.

Syntax

```
#Include ADwinPro_All.Inc
P2_Burst_CRead_Unpacked2(module, count, array1[],
    array2[], array_idx, flowrate)
```

Parameters

module	Specified module address (1…15).	LONG
count	Amount of measurement values per channel to be transferred. The amount must be divisible by 4.	LONG
arrayx[]	Destination arrays for the measurement values of channels 1 and 2. No float type and no FIFO array allowed.	ARRAY LONG ~~FLOAT~~
array_idx	Destination start index: Array element from which measurement values are stored.	LONG
flowrate	Evaluation for low-priority processes only: Parameter for data throughput. 1: slow. 2: medium. 3: fast.	LONG

Notes

On modules AIn-F-x-16, the instruction is available since revision E04.

The instruction be used if a continuous burst sequence was initialized with 2 channels (see **P2_Burst_Init**, parameters mode, channels).

The instruction reads the amount of count measurement values that are stored in the module memory. count should be lower by the factor 2 than the number of measurements (buffer_count), specified in **P2_Burst_Init**

The instruction stores the measurement values one after the other in the elements of the destination array.

In high-priority processes the maximum data throughput is used automatically; the parameter flowrate must be indicated all the same.

The higher the data troughput–in a low priority process only–is selected, the more it may happen that a process of higher priority is delayed.

See also

P2_Burst_Init, P2_Burst_CRead_Unpacked1, P2_Burst_CRead_Unpacked4, P2_Burst_CRead_Unpacked8, P2_Burst_Start, P2_Burst_Status

Valid for

AIn-F-4/14 Rev. E, AIn-F-4/16 Rev. E, AIn-F-8/14 Rev. E, AIn-F-8/16 Rev. E

Example

```
#Include ADwinPro_All.INC
#Define module 4

Dim Data_1[1000], Data_2[1000] As Long
Dim pattern As Long

Init:
  REM Initiate cont. burst sequence for channels 1…2 using 60ns
  REM period duration, save 2^26 samples per channel starting
  REM from address 0.
  P2_Burst_Init (module,3,0,67108860,3,010b)
  REM Start burst sequence
  pattern = Shift_Left(1,module-1) 'access single module only
  P2_Burst_Start(pattern)
  P2_Set_LED(module,1)
  Processdelay=10000000

Event:
  REM Read last 1000 samples per channel (slowly) and store
  REM in Data_1 and Data_2
  P2_Burst_CRead_Unpacked2(module,1000,Data_1,Data_2,1,1)
```

P2_Burst_CRead_Unpacked4

P2_Burst_CRead_Unpacked4 copies an amount of the last measurement values of 4 channels from the memory of the specified module into 4 arrays.

Syntax

```
#Include ADwinPro_All.Inc
P2_Burst_CRead_Unpacked4(module, count, array1[],
    array2[], array3[], array4[], array_idx,
    flowrate)
```

Parameters

module	Specified module address (1…15).	LONG
count	Amount of measurement values per channel to be transferred. The amount must be divisible by 2.	LONG
arrayx[]	Destination arrays for the measurement values of channels 1…4. No float type and no FIFO array allowed.	ARRAY LONG ~~FLOAT~~
array_idx	Destination start index: Array element from which measurement values are stored.	LONG
flowrate	Evaluation for low-priority processes only: Parameter for data throughput. 1: slow. 2: medium. 3: fast.	LONG

Notes

On modules AIn-F-x-16, the instruction is available since revision E04.

The instruction be used if a continuous burst sequence was initialized with 4 channels (see **P2_Burst_Init**, parameters mode, channels).

The instruction reads the number of count measurement values that are stored in the module memory. count should be lower by the factor 2 than the number of measurements (buffer_count), specified in **P2_Burst_Init**.

The instruction stores the measurement values one after the other in the elements of the destination array.

In high-priority processes the maximum data throughput is used automatically; the parameter flowrate must be indicated all the same.

The higher the data troughput–in a low priority process only–is selected, the more it may happen that a process of higher priority is delayed.

See also

P2_Burst_Init, P2_Burst_CRead_Unpacked1, P2_Burst_CRead_Unpacked2, P2_Burst_CRead_Unpacked8, P2_Burst_Start, P2_Burst_Status

Valid for

AIn-F-4/14 Rev. E, AIn-F-4/16 Rev. E, AIn-F-8/14 Rev. E, AIn-F-8/16 Rev. E

Example

```
#Include ADwinPro_All.Inc
#Define module 4

Dim Data_1[1000], Data_2[1000] As Long
Dim Data_3[1000], Data_4[1000] As Long
Dim pattern As Long

Init:
  REM Initiate cont. burst sequence for channels 1…4 using 40ns
  REM period duration, save 2^25 samples per channel starting
  REM from address 0.
  P2_Burst_Init (module,15,0,3355444,2,010b)
  REM Start burst sequence
  pattern = Shift_Left(1,module-1) 'access single module only
  P2_Burst_Start(pattern)
  Processdelay=50000000

Event:
  REM Read last 1000 samples per channel (fast) and store
  REM in Data_1 to Data_4
  P2_Burst_CRead_Unpacked4(module,1000,Data_1,Data_2,Data_3,
    Data_4,1,3)
```

P2_Burst_CRead_Unpacked8

P2_Burst_CRead_Unpacked8 copies an amount of the last measurement values of 8 channels from the memory of the specified module into 8 arrays.

Syntax

```
#Include ADwinPro_All.Inc
P2_Burst_CRead_Unpacked8(module, count, array1[],
    array2[], array3[], array4[], array5[], array6[],
    array7[], array8[], array_idx, flowrate)
```

Parameters

module	Specified module address (1…15).	LONG
count	Amount of measurement values per channel to be transferred. The amount of values must be divisible by 4.	LONG
arrayx[]	Destination arrays for the measurement values of channels 1…8. No float type and no FIFO array allowed.	ARRAY LONG ~~FLOAT~~
array_idx	Destination start index: Array element from which measurement values are stored.	LONG
flowrate	Evaluation for low-priority processes only: Parameter for data throughput. 1: slow. 2: medium. 3: fast.	LONG

Notes

The instruction be used if a continuous burst sequence was initialized with 8 channels (see **P2_Burst_Init**, parameters mode, channels).

The instruction reads the number of count measurement values that are stored in the module memory. count should be lower by the factor 2 than the number of measurements (buffer_count), specified in **P2_Burst_Init**.

The instruction stores the measurement values one after the other in the elements of the destination array.

In high-priority processes the maximum data throughput is used automatically; the parameter flowrate must be indicated all the same.

The higher the data troughput–in a low priority process only–is selected, the more it may happen that a process of higher priority is delayed.

See also

P2_Burst_Init, P2_Burst_CRead_Unpacked1, P2_Burst_CRead_Unpacked2, P2_Burst_CRead_Unpacked4, P2_Burst_Start, P2_Burst_Status

Valid for

AIn-F-8/14 Rev. E, AIn-F-8/16 Rev. E

Example

```
#Include ADwinPro_All.Inc
#Define module 4

Dim Data_1[1000], Data_2[1000] As Long At DM_Local
Dim Data_3[1000], Data_4[1000] As Long At DM_Local
Dim Data_5[1000], Data_6[1000] As Long At DM_Local
Dim Data_7[1000], Data_8[1000] As Long At DM_Local
Dim pattern As Long

Init:
  REM Initiate cont. burst sequence for channels 1…4 using 40ns
  REM period duration, save 2^25 samples per channel starting
  REM from address 0.
  P2_Burst_Init (module,255,100,1000000,2,010b)
  REM Start burst sequence
  pattern = Shift_Left(1,module-1) 'access single module only
  P2_Burst_Start(pattern)
  Processdelay=10000000

Event:
  REM Read last 10000 samples per channel (fast) and store
  REM in Data_1 to Data_8
  P2_Burst_CRead_Unpacked8(module,1000,Data_1,Data_2,Data_3,
    Data_4,Data_5,Data_6,Data_7,Data_8,1,3)
```

P2_Burst_CRead_Pos_Unpacked1

P2_Burst_CRead_Pos_Unpacked1 copies an amount of measurement values of 1channel from the given address of the module memory into one array.

If the end of the memory buffer is reached while reading measurement values, further values are read from the start of the memory buffer.

Syntax

```
#Include ADwinPro_All.Inc

P2_Burst_CRead_Pos_Unpacked1(module, buffer_start,
    buffer_count, count, startadr, array1[],
    array_idx, flowrate)
```

Parameters

module	Specified module address (1…15).	LONG
buffer_start	Start address of the memory buffer which is to be read from.	LONG
buffer_count	Number of measurement values per channel to be stored in the memory buffer which is to be read from.	LONG
count	Amount of measurement values per channel to be transferred. The amount of values must be divisible by 4.	LONG
startadr	Start address within the memory buffer where the first measurement value is to be read.	LONG
array1[]	Destination array for the measurement values.	ARRAY
	No float data type and no FIFO array allowed.	LONG
		~~FLOAT~~
array_idx	Destination start index: Array element from which measurement values are stored.	LONG
flowrate	Evaluation for low-priority processes only: Parameter for data throughput. 1: slow. 2: medium. 3: fast.	LONG

Notes

On modules AIn-F-x-16, the instruction is available since revision E04.

While developing a program, we recommend to enable the debug mode. You will receive hints about programming errors.

The instruction be used if a continuous burst sequence was initialized with 1 channel (see **P2_Burst_Init**, parameters mode, channels).

The parameters buffer_start and buffer_count concerning the memory buffer support dividing the module memory into several buffers. Enter the same values as with intialiszation of the memory buffer with **P2_Burst_Init**–even if you use the module memory without dividing.

The instruction reads the number of count measurement values starting from the address startadr. If the burst sequence still continues the count should be lower at least by the factor 10 than the number of measurements (buffer_count), specified in **P2_Burst_Init**.

The instruction stores the measurement values one after the other in the elements of the destination array.

In high-priority processes the maximum data throughput is used automatically; the parameter `flowrate` must be indicated all the same.

The higher the data troughput–in a low priority process only–is selected, the more it may happen that a process of higher priority is delayed.

See also

P2_Burst_Init, P2_Burst_CRead_Unpacked1, P2_Burst_CRead_Pos_Unpacked1, P2_Burst_CRead_Pos_Unpacked2, P2_Burst_CRead_Pos_Unpacked4, P2_Burst_CRead_Pos_Unpacked8, P2_Burst_Start, P2_Burst_Status

Valid for

AIn-F-4/14 Rev. E, AIn-F-4/16 Rev. E, AIn-F-8/14 Rev. E, AIn-F-8/16 Rev. E

Example

- / -

P2_Burst_CRead_Pos_Unpacked2

P2_Burst_CRead_Pos_Unpacked2 copies an amount of measurement values of 2 channels from the given address of the module memory into 2 arrays.

If the end of the memory buffer is reached while reading measurement values, further values are read from the start of the memory buffer.

Syntax

```
#Include ADwinPro_All.Inc

P2_Burst_CRead_Pos_Unpacked2(module, buffer_start,
    buffer_count, count, startadr, array1[],
    array2[], array_idx, flowrate)
```

Parameters

module	Specified module address (1…15).	LONG
buffer_start	Start address of the memory buffer which is to be read from.	LONG
buffer_count	Number of measurement values per channel to be stored in the memory buffer which is to be read from.	LONG
count	Amount of measurement values per channel to be transferred. The amount of values must be divisible by 4.	LONG
startadr	Start address within the memory buffer where the first measurement value is to be read.	LONG
arrayx[]	Destination arrays for the measurement values of channels 1…2. No float type and no FIFO array allowed.	ARRAY LONG ~~FLOAT~~
array_idx	Destination start index: Array element from which measurement values are stored.	LONG
flowrate	Evaluation for low-priority processes only: Parameter for data throughput. 1: slow. 2: medium. 3: fast.	LONG

Notes

On modules AIn-F-x-16, the instruction is available since revision E04.

While developing a program, we recommend to enable the debug mode. You will receive hints about programming errors.

The instruction be used if a continuous burst sequence was initialized with 2 channels (see **P2_Burst_Init**, parameters mode, channels).

The parameters buffer_start and buffer_count concerning the memory buffer support dividing the module memory into several buffers. Enter the same values as with intialiszation of the memory buffer with **P2_Burst_Init**–even if you use the module memory without dividing.

The instruction reads the number of count measurement values starting from the address startadr. If the burst sequence still continues the count should be lower at least by the factor 2 than the number of measurements (buffer_count), specified in **P2_Burst_Init**.

The instruction stores the measurement values one after the other in the elements of the destination array.

In high-priority processes the maximum data throughput is used automatically; the parameter `flowrate` must be indicated all the same.

The higher the data troughput–in a low priority process only–is selected, the more it may happen that a process of higher priority is delayed.

See also

P2_Burst_Init, P2_Burst_CRead_Unpacked2, P2_Burst_CRead_Pos_Unpacked1, P2_Burst_CRead_Pos_Unpacked4, P2_Burst_CRead_Pos_Unpacked8, P2_Burst_Start, P2_Burst_Status

Valid for

AIn-F-4/14 Rev. E, AIn-F-4/16 Rev. E, AIn-F-8/14 Rev. E, AIn-F-8/16 Rev. E

Example

- / -

P2_Burst_CRead_Pos_Unpacked4

P2_Burst_CRead_Pos_Unpacked4 copies an amount of measurement values of 4 channels from the given address of the module memory into 4 arrays.

If the end of the memory buffer is reached while reading measurement values, further values are read from the start of the memory buffer.

Syntax

```
#Include ADwinPro_All.Inc

P2_Burst_CRead_Pos_Unpacked4(module, buffer_start,
    buffer_count, count, startadr, array1[],
    array2[], array3[], array4[],
    array_idx, flowrate)
```

Parameters

module	Specified module address (1…15).	LONG
buffer_start	Start address of the memory buffer which is to be read from.	LONG
buffer_count	Number of measurement values per channel to be stored in the memory buffer which is to be read from.	LONG
count	Amount of measurement values per channel to be transferred. The amount of values must be divisible by 4.	LONG
startadr	Start address within the memory buffer where the first measurement value is to be read.	LONG
arrayx[]	Destination arrays for the measurement values of channels 1…4. No float type and no FIFO array allowed.	ARRAY LONG ~~FLOAT~~
array_idx	Destination start index: Array element from which measurement values are stored.	LONG
flowrate	Evaluation for low-priority processes only: Parameter for data throughput. 1: slow. 2: medium. 3: fast.	LONG

Notes

On modules AIn-F-x-16, the instruction is available since revision E04.

While developing a program, we recommend to enable the debug mode. You will receive hints about programming errors.

The instruction be used if a continuous burst sequence was initialized with 4 channels (see **P2_Burst_Init**, parameters mode, channels).

The parameters buffer_start and buffer_count concerning the memory buffer support dividing the module memory into several buffers. Enter the same values as with intialiszation of the memory buffer with **P2_Burst_Init**–even if you use the module memory without dividing.

The instruction reads the number of count measurement values starting from the address startadr. If the burst sequence still continues the count should be lower at least by the factor 2 than the number of measurements (buffer_count), specified in **P2_Burst_Init**.

The instruction stores the measurement values one after the other in the elements of the destination array.

In high-priority processes the maximum data throughput is used automatically; the parameter `flowrate` must be indicated all the same.

The higher the data troughput–in a low priority process only–is selected, the more it may happen that a process of higher priority is delayed.

See also

P2_Burst_Init, P2_Burst_CRead_Unpacked4, P2_Burst_CRead_Pos_Unpacked1, P2_Burst_CRead_Pos_Unpacked2, P2_Burst_CRead_Pos_Unpacked8, P2_Burst_Start, P2_Burst_Status

Valid for

AIn-F-4/14 Rev. E, AIn-F-4/16 Rev. E, AIn-F-8/14 Rev. E, AIn-F-8/16 Rev. E

Example

- / -

P2_Burst_CRead_Pos_Unpacked8

P2_Burst_CRead_Pos_Unpacked8 copies an amount of measurement values of 8 channels from the given address of the module memory into 8 arrays.

If the end of the memory buffer is reached while reading measurement values, further values are read from the start of the memory buffer.

Syntax

```
#Include ADwinPro_All.Inc

P2_Burst_CRead_Pos_Unpacked8(module, buffer_start,
    buffer_count, count, startadr, array1[],
    array2[], array3[], array4[], array5[], array6[],
    array7[], array8[], array_idx, flowrate)
```

Parameters

module	Specified module address (1…15).	LONG
buffer_start	Start address of the memory buffer which is to be read from.	LONG
buffer_count	Number of measurement values per channel to be stored in the memory buffer which is to be read from.	LONG
count	Amount of measurement values per channel to be transferred. The amount of values must be divisible by 4.	LONG
startadr	Start address within the memory buffer where the first measurement value is to be read.	LONG
arrayx[]	Destination arrays for the measurement values of channels 1…8. No float type and no FIFO array allowed.	ARRAY LONG ~~FLOAT~~
array_idx	Destination start index: Array element from which measurement values are stored.	LONG
flowrate	Evaluation for low-priority processes only: Parameter for data throughput. 1: slow. 2: medium. 3: fast.	LONG

Notes

While developing a program, we recommend to enable the debug mode. You will receive hints about programming errors.

The instruction be used if a continuous burst sequence was initialized with 8 channels (see **P2_Burst_Init**, parameters mode, channels).

The parameters buffer_start and buffer_count concerning the memory buffer support dividing the module memory into several buffers. Enter the same values as with initialization of the memory buffer with **P2_Burst_Init**–even if you use the module memory without dividing.

The instruction reads the number of count measurement values starting from the address startadr. If the burst sequence still continues the count should be lower at least by the factor 2 than the number of measurements (buffer_count), specified in **P2_Burst_Init**.

The instruction stores the measurement values one after the other in the elements of the destination array.

In high-priority processes the maximum data throughput is used automatically; the parameter `flowrate` must be indicated all the same.

The higher the data troughput–in a low priority process only–is selected, the more it may happen that a process of higher priority is delayed.

See also

P2_Burst_Init, P2_Burst_CRead_Unpacked8, P2_Burst_CRead_Pos_Unpacked1, P2_Burst_CRead_Pos_Unpacked2, P2_Burst_CRead_Pos_Unpacked4, P2_Burst_Start, P2_Burst_Status

Valid for

AIn-F-8/14 Rev. E, AIn-F-8/16 Rev. E

Example

- / -

P2_Burst_Init

P2_Burst_Init sets the parameters for a burst-measurement sequence on the specified module.

These are: Measurement mode (amount and numbers of measurement channels), start address in module memory, period duration of measurement sequence and number of measurements to be executed.

Syntax

```
#Include ADwinPro_All.Inc
P2_Burst_Init (module, channels, buffer_start,
    buffer_count, pulses, mode)
```

Parameters

`module` Specified module address (1…15). LONG

`channels` specifies amount and numbers of measurement channels. Only given values are allowed. LONG

In combination with the memory size the maximum amount of measurement values per channel is set:

channels	Channel no.	max. amount of values per channel for buffer_start=0:
1	1	134217720 = 7FFFFF0h
3	1…2	67108860 = 3FFFFFCh
15	1…4	33554428 = 1FFFFFCh
255	1…8	16777212 = 0FFFFFCh

Alternatively the last channel may be used as time channel with the following values:

channels	channel no.	time ch.	max. amount of values per channel for buffer_start=0:
131	1	2	67108860
143	1…3	4	33554428
127	1…7	8	16777212

`buffer_start` Start address (0…67108860 = 2^{26} - 4 Longs) of the memory buffer in the module memory. Address is given in longs and must be divisible by 4. LONG

`buffer_count` Amount of measurements per channel to be stored in the memory buffer; the amount determines the size of the memory buffer. LONG

The maximum amount of `buffer_count` is determined by `channels` (and the start address). The amount must be divisible by 4; if `channels`=1 (1 channel) it must be divisible by 8.

`pulses` determines the period duration of a measurement sequence as number of time intervals; valid only with timer-controlled sequence (see `mode`): LONG

AIn-F-x/14: period duration = `pulses` * 20ns.

AIn-F-x/16: period duration = `pulses` * 10ns.

The value range is 1…65535; with 8 channels (`channels`=255 or 127) the range starts with 2.

The period duration is the time from the beginning of a measurement until the beginning of the next measurement.

AIn-F-x/16 only: The smallest value of `pulses` depends on the parameter `mode` of **`P2_Set_Average_Filter`**:

`mode` (**`P2_Set_Average_Filter`**)	$pulses_{min}$
0	25
1	67
2	154
3	313
4	645
5	1333

`mode` Bit pattern, defining the mode of burst sequence: LONG

Bit no.	03…31	02	01	00
Function	–	Type of clock speed: Bit = 0: Timer controlled (`pulses`). Bit = 1: Externally controlled (event input).	Operating mode of burst sequence: Bit = 0: Single. Bit = 1: Continuous.	–

Notes

On modules AIn-F-x-16, the instruction is available since revision E04.

You can read the current measurement value of a channel even with **`P2_Read_ADCF`** if it is not saved, for instance for testing a trigger condition.

A burst sequence cannot be combined with the operating modes Timer or Event of the instruction **`P2_ADCF_Mode`**.

The time channel stores with each burst measurement the current value of the module timer. Thus, the values can be exactly allocated on a timeline e.g. with externally controlled speed. One time unit of the 16 bit-timer relates to 20ns.

The number of storable measurement values per channel depends on the given start address and the module's memory size.

Operating mode

There are 2 operating modes:

- Single burst sequence: The module converts and stores a fixed number of values (`buffer_count`). As soon as all values are

stored, the burst sequence stops. Values be read using **P2_Burst_Read_Unpacked…** .

- Continuous burst sequence: The module converts continuously with the defined cycle duration. The burst sequence be stopped with **P2_Burst_Stop** and values be read using **P2_Burst_CRead_…** . The module stores values in the allocated memory buffer which is used as ring buffer. Therefore the youngest value will each overwrite the eldest value.

Dividing memory in several buffers

You can virtually divide the module memory into several buffers for different burst sequences. Only one of the memory buffers can be active at a time , i.e. be initialized with **P2_Burst_Init** and store measurement values.

The length of a memory buffer in longs is

$$\text{buffer_length} = \text{buffer_count} \times \text{channel number} \div 2$$

Clock speed

With timer controlled speed the clock rate of the burst sequence is set with `pulses`.

With externally controlled speed each event signal starts a burst measurement; please note the settings of **P2_Event_Config**. The maximum clock rate is 50 MHz. Burst sequences on several modules may be synchronized using **P2_Sync_Mode**.

See also

P2_Burst_CRead_Pos_Unpacked1, P2_Burst_CRead_Pos_Unpacked2, P2_Burst_CRead_Pos_Unpacked4

P2_Burst_CRead_Unpacked1, P2_Burst_CRead_Unpacked2, P2_Burst_CRead_Unpacked4, P2_Burst_CRead_Unpacked8, P2_Set_Average_Filter

P2_Burst_Read_Index, P2_Burst_Read_Unpacked1, P2_Burst_Read_Unpacked2, P2_Burst_Read_Unpacked4, P2_Burst_Read_Unpacked8

P2_Burst_Start, P2_Burst_Status, P2_Burst_Stop, P2_Read_ADCF, P2_Sync_Mode, P2_Set_Average_Filter

Valid for

AIn-F-4/14 Rev. E, AIn-F-4/16 Rev. E, AIn-F-8/14 Rev. E, AIn-F-8/16 Rev. E

Example

See also example "Continuous signal conversion" on page 1078.

```
#Include ADwinPro_All.Inc
#Define module 4

Dim Data_1[1000] As Long
Dim pattern As Long

Init:
  REM Initiate continuous burst sequence for channel 1 using 20ns
  REM period duration, save 2^26 samples from address 0.
  P2_Burst_Init (module,1,0,67108864,1,010b)
  REM Start burst sequence
  pattern = Shift_Left(1,module-1) 'access single module only
  P2_Burst_Start(pattern)
  Processdelay=10000000

Event:
  REM Read last 1000 samples from channel (slowly) and store
  REM in Data_1
  P2_Burst_CRead_Unpacked1(module,1000,Data_1,1,1)
```

P2_Burst_Read_Index

P2_Burst_Read_Index returns the address in the module memory, where the last measurement values have been stored.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = P2_Burst_Read_Index(module)
```

Parameters

module	Specified module address (1…15).	LONG
ret_val	Address (0…67108860 = 2^{26} - 4) in the module memory.. The address is divisible by 4.	LONG

Notes

On modules AIn-F-x-16, the instruction is available since revision E04.

P2_Burst_Read_Index is an elementary instruction enabling special solutions in combination with **P2_Burst_Read**, but requires particular accuracy and knowledge of programming. The more simple alternative is to use the instructions **P2_Burst_Read_Unpacked…** or **P2_Burst_CRead_Unpacked…** .

Starting from the returned address ret_val the number n of saved values may be calculated. The start address and the number of channels are set with **P2_Burst_Init:**

$$n = (\text{ret_val} - \text{startadr}) \cdot \frac{2}{\text{no. of channels}}$$

The module memory is always addressed in steps of 4 (4 times 32 bits). After the instructions **P2_Burst_Init** and **P2_Burst_Reset** the address pointer is set to the last possible address of the reserved module memory, which is buffer_start + buffer_count * (no. of channels) - 4.

It depends on the measurement mode which channels provide the last measurement values in the memory. More see **P2_Burst_Read**.

See also

P2_Burst_Init, P2_Burst_Read_Unpacked1, P2_Burst_Read_Unpacked2, P2_Burst_Read_Unpacked4, P2_Burst_Read_Unpacked8, P2_Burst_Reset, P2_Burst_Start, P2_Burst_Status, P2_Burst_Stop

Valid for

AIn-F-4/14 Rev. E, AIn-F-4/16 Rev. E, AIn-F-8/14 Rev. E, AIn-F-8/16 Rev. E

Example

See also example "Continuous signal conversion" on page 1078.

```
#Include ADwinPro_All.Inc

#Define module 4
#Define buffer_count 500000
#Define channels 4
#Define frq_Hz 5000
#Define mem_idx Par_1
#Define count Par_2
#Define overflow Par_3

Dim Data_1[buffer_count], Data_2[buffer_count] As Long
Dim Data_3[buffer_count], Data_4[buffer_count] As Long
Dim i, prev_mem_idx, start_idx As Long

LowInit:
  For i = 1 To buffer_count
    Data_1[i] = 0 : Data_2[i] = 0 : Data_3[i] = 0 : Data_4[i] = 0
  Next i

Init:
  Processdelay = 300000000 / frq_Hz
  P2_Set_LED(module, 1)     'switch on LED
  REM Continuous burst sequence for channels 1…4 using 100ns
  REM period duration
  P2_Burst_Init(module, 15, 0, buffer_count, 5, 2)
  P2_Burst_Start(Shift_Left(1, module - 1))
  start_idx = 1
  prev_mem_idx = 0
  overflow = 0

Event:
  REM current memory address
  mem_idx = P2_Burst_Read_Index(module)
  REM no. of new samples per channel since last cycle
  count = (mem_idx - prev_mem_idx) * 2 / channels

  If (count > 0) Then
    REM read samples from F8/14 module
    P2_Burst_Read_Unpacked4(module, count, prev_mem_idx,
     Data_1, Data_2,  Data_3, Data_4, start_idx, 0)
    REM Start index for next cycle
    start_idx = start_idx + count
    REM store index of F8/14 module
    prev_mem_idx = mem_idx
  EndIf

  If (count < 0) Then
    REM No. of samples until end of DATA
    count = buffer_count - prev_mem_idx * 2 / channels
    REM read samples from F8/14 module
    P2_Burst_Read_Unpacked4(module, count, prev_mem_idx,
     Data_1, Data_2,  Data_3, Data_4, start_idx, 0)
    REM Start index for next cycle
    start_idx = 1
    REM store index of F8/14 module for next cycle
    prev_mem_idx = 0
    Inc(overflow)           'increase overflow counter
  EndIf

Finish:
  P2_Set_LED(module, 0)     'switch off LED
```

P2_Burst_Read

P2_Burst_Read copies 32-bit values from the memory of the specified module into a specified array.

Syntax

```
#Include ADwinPro_All.Inc
P2_Burst_Read(module, count, startadr, array[],
    array_idx, flowrate)
```

Parameters

module	Specified module address (1…15).	LONG
count	Amount of 32-bit values to be transferred. The amount must be divisible by 4.	LONG
startadr	Start address (0…67108860 = 2^{26} - 4) in the module memory: Address from which the measurement values are read. The address must be divisible by 4.	LONG
array[]	Destination array, where the measurement values are transferred. No float type and no FIFO array allowed.	ARRAY LONG ~~FLOAT~~
array_idx	Destination start index: Array element from which measurement values are stored.	LONG
flowrate	Evaluation for low-priority processes only: Parameter for data throughput. 1: slow. 2: medium. 3: fast.	LONG

Notes

On modules AIn-F-x-16, the instruction is available since revision E04.

P2_Burst_Read is an elementary instruction enabling special solutions in combination with **P2_Burst_Read_Index**, but requires particular accuracy and knowledge of programming. The more simple alternative is to use the instructions **P2_Burst_Read_Unpacked…** or **P2_Burst_CRead_Unpacked…** .

P2_Burst_Read copies the 32 bit values from the memory without changes; any 32 bit value holds 2 measurement data of 16 bit. It depends on the set number of channels (see **P2_Burst_Init**, parameter channels) which channels the measurement data are assigned to. The following tables show the assignment of 16 bit data D to channel numbers C:

address	Bits 31:16	Bits 15:00
startadr	C1 / D2	C1 / D1
startadr+1	C1 / D4	C1 / D3
startadr+2	C1 / D6	C1 / D5
…	…	…

No. of channels: 1

address	Bits 31:16	Bits 15:00
startadr	C2 / D1	C1 / D1
startadr+1	C2 / D2	C1 / D2
startadr+2	C2 / D3	C1 / D3
…	…	…

No. of channels: 2

address	Bits 31:16	Bits 15:00
startadr	C2 / D1	C1 / D1
startadr+1	C4 / D1	C3 / D1
startadr+2	C2 / D2	C1 / D2
startadr+3	C4 / D2	C3 / D2
startadr+4	C2 / D3	C1 / D3
...	...	...

No. of channels: 4

address	Bits 31:16	Bits 15:00
startadr	C2 / D1	C1 / D1
startadr+1	C4 / D1	C3 / D1
startadr+2	C6 / D1	C5 / D1
startadr+3	C8 / D1	C7 / D1
startadr+4	C2 / D2	C1 / D2
...	...	...

No. of channels: 8

In high-priority processes the maximum data throughput is used automatically; the parameter `flowrate` must be indicated all the same.

The higher the data troughput–in a low priority process only–is selected, the more it may happen that a process of higher priority is delayed.

See also

P2_Burst_Init, P2_Burst_Read_Index, P2_Burst_Read_Unpacked1, P2_Burst_Read_Unpacked2, P2_Burst_Read_Unpacked4, P2_Burst_Read_Unpacked8, P2_Burst_Start, P2_Burst_Status, P2_Burst_Stop, P2_Read_ADCF, P2_Set_Average_Filter

Valid for

AIn-F-4/14 Rev. E, AIn-F-4/16 Rev. E, AIn-F-8/14 Rev. E, AIn-F-8/16 Rev. E

Example

see P2_Burst_Read_Index

P2_Burst_Read_Unpacked1

P2_Burst_Read_Unpacked1 copies the measurement values of a channel into a specified array.

Syntax

```
#Include ADwinPro_All.Inc
P2_Burst_Read_Unpacked1(module, count, startadr,
    array[], array_idx, flowrate)
```

Parameters

module	Specified module address (1…15).	LONG
count	Amount of measurement values to be transferred. The amount must be divisible by 8.	LONG
startadr	Start address (0…67108860 = 2^{26} - 4) in the module memory: Address from which the measurement values are read. The address must be divisible by 4.	LONG
array[]	Destination array, where the measurement values are transferred. No float type and no FIFO array allowed.	ARRAY LONG ~~FLOAT~~
array_idx	Destination start index: Array element from which measurement values are stored.	LONG
flowrate	Evaluation for low-priority processes only: Parameter for data throughput. 1: slow. 2: medium. 3: fast.	LONG

Notes

On modules AIn-F-x-16, the instruction is available since revision E04.

The instruction be used if the burst sequence was initialized with 1 channel (see **P2_Burst_Init**, parameter channels).

The instruction stores the measurement values one after the other in the elements of the destination array.

In high-priority processes the maximum data throughput is used automatically; the parameter flowrate must be indicated all the same.

The higher the data troughput–in a low priority process only–is selected, the more it may happen that a process of higher priority is delayed.

See also

P2_Burst_Init, P2_Burst_Read_Unpacked2, P2_Burst_Read_Unpacked4, P2_Burst_Read_Unpacked8, P2_Burst_Start, P2_Burst_Status

Valid for

AIn-F-4/14 Rev. E, AIn-F-4/16 Rev. E, AIn-F-8/14 Rev. E, AIn-F-8/16 Rev. E

Example

See also examples for Continuous signal conversion on page 435.

```
#Include ADwinPro_All.Inc
#Define module 1

Dim Data_1[1000] As Long
Dim state As Long
Dim rest As Long
Dim pattern As Long

Init:
  REM Initiate single burst sequence for channel 1 using 20ns
  REM period duration, save 1000 samples from address 0.
  P2_Burst_Init (module,1,0,1000,1,0)
  REM Start burst sequence
  pattern = Shift_Left(1,module-1) 'access single module only
  P2_Burst_Start(pattern)
  Processdelay=10000000
  REM State: Burst_sequence is running
  state=0

Event:
  REM Get number of samples still to do
  rest=P2_Burst_Status(module)
  REM All samples done: change state
  If (rest=0) Then state=1
  If (state=1) Then
    REM All samples done: Read 1000 samples (fast) and store
    REM in Data_1
    P2_Burst_Read_Unpacked1(module,1000,0,Data_1,1,3)
    REM Start next burst sequence
    state=0
    P2_Burst_Reset(pattern)
    P2_Burst_Start(pattern)
  EndIf
```

P2_Burst_Read_Unpacked2

P2_Burst_Read_Unpacked2 copies the measurement values of 2 channels from the memory of the specified module into 2 arrays.

Syntax

```
#Include ADwinPro_All.Inc
P2_Burst_Read_Unpacked2(module, count, startadr,
    array1[], array2[], array_idx, flowrate)
```

Parameters

module	Specified module address (1…15).	LONG
count	Amount of measurement values to be transferred. The amount must be divisible by 4.	LONG
startadr	Start address (0…67108860 = 2^{26} - 4) in the module memory: Address from which the measurement values.are read. The address must be divisible by 4.	LONG
arrayx[]	Destination arrays for the measurement values of channels 1 and 2. No float type and no FIFO array allowed.	ARRAY LONG ~~FLOAT~~
array_idx	Destination start index: Array element from which measurement values are stored.	LONG
flowrate	Evaluation for low-priority processes only: Parameter for data throughput. 1: slow. 2: medium. 3: fast.	LONG

Notes

On modules AIn-F-x-16, the instruction is available since revision E04.

The instruction be used if the burst sequence was initialized with 2 channels (see **P2_Burst_Init**, parameter channels).

The instruction stores the measurement values one after the other in the elements of the destination array: Channel 1 in array1, channel 2 in array2.

In high-priority processes the maximum data throughput is used automatically; the parameter flowrate must be indicated all the same.

The higher the data troughput–in a low priority process only–is selected, the more it may happen that a process of higher priority is delayed.

See also

P2_Burst_Init, P2_Burst_Read_Unpacked1, P2_Burst_Read_Unpacked4, P2_Burst_Read_Unpacked8, P2_Burst_Start, P2_Burst_Status

Valid for

AIn-F-4/14 Rev. E, AIn-F-4/16 Rev. E, AIn-F-8/14 Rev. E, AIn-F-8/16 Rev. E

Example

See also examples for Continuous signal conversion on page 435.

```
#Include ADwinPro_All.Inc
#Define module 4

Dim Data_1[1000], Data_2[1000] As Long
Dim state As Long
Dim rest As Long
Dim pattern As Long

Init:
  REM Initiate single burst sequence for channels 1 and 2 using
  REM 40ns period duration, save 1000 samples from address 0.
  P2_Burst_Init (module,3,0,1000,2,0)
  REM Start burst sequence
  pattern = Shift_Left(1,module-1) 'access single module only
  P2_Burst_Start(pattern)
  Processdelay=10000000
  REM State: Burst_sequence is running
  state=0

Event:
  REM Get number of samples still to do
  rest=P2_Burst_Status(module)
  If (rest=0) Then state=1
  If (state=1) Then
    REM All samples done: Read 1000 samples for each channel (fast)
    REM and store in Data_1 and Data_2
    P2_Burst_Read_Unpacked2(module,1000,0,Data_1,Data_2,1,3)
    REM Start next burst sequence
    state=0
    P2_Burst_Reset(pattern)
    P2_Burst_Start(pattern)
  EndIf
```

P2_Burst_Read_Unpacked4

P2_Burst_Read_Unpacked4 copies the measurement values of 4 channels from the memory of the specified module into 4 arrays.

Syntax

```
#Include ADwinPro_All.Inc

P2_Burst_Read_Unpacked4(module, count, startadr,
    array1[], array2[], array3[], array4[],
    array_idx, flowrate)
```

Parameters

module	Specified module address (1…15).	LONG
count	Amount of measurement values to be transferred. The amount must be divisible by 2.	LONG
startadr	Start address (0…67108860 = 2^{26} - 4) in the module memory: Address from which the measurement values.are read. The address must be divisible by 4.	LONG
arrayx[]	Destination arrays for the measurement values of channels 1…4. No float type and no FIFO array allowed.	ARRAY LONG ~~FLOAT~~
array_idx	Destination start index: Array element from which measurement values are stored.	LONG
flowrate	Evaluation for low-priority processes only: Parameter for data throughput. 1: slow. 2: medium. 3: fast.	LONG

Notes

On modules AIn-F-x-16, the instruction is available since revision E04.

The instruction be used if the burst sequence was initialized with 4 channels (see **P2_Burst_Init**, parameter channels).

The instruction stores the measurement values one after the other in the elements of the destination array: Channel 1 in array1, channel 2 in array2 etc.

In high-priority processes the maximum data throughput is used automatically; the parameter flowrate must be indicated all the same.

The higher the data troughput–in a low priority process only–is selected, the more it may happen that a process of higher priority is delayed.

See also

P2_Burst_Init, P2_Burst_Read_Unpacked1, P2_Burst_Read_Unpacked2, P2_Burst_Read_Unpacked8, P2_Burst_Start, P2_Burst_Status

Valid for

AIn-F-4/14 Rev. E, AIn-F-4/16 Rev. E, AIn-F-8/14 Rev. E, AIn-F-8/16 Rev. E

Example

See also examples for Continuous signal conversion on page 435.

```
#Include ADwinPro_All.Inc
#Define module 4

Dim Data_1[1000], Data_2[1000] As Long
Dim Data_3[1000], Data_4[1000] As Long
Dim state As Long
Dim rest As Long
Dim pattern As Long

Init:
  REM Initiate single burst sequence for channels 1…4 using 40ns
  REM period duration, save 3000 samples per channel from addr. 0
  P2_Burst_Init (module,15,0,3000,2,0)
  REM Start burst sequence
  pattern = Shift_Left(1,module-1) 'access single module only
  P2_Burst_Start(pattern)
  Processdelay=10000000
  REM State: Burst_sequence is running
  state=0

Event:
  REM Get number of samples still to do
  rest=P2_Burst_Status(module)
  If (rest=0) Then state=1
  If (state=1) Then
    REM All samples done: Read 3000 samples (fast) per channel and
    REM store in Data_1 to Data_4
    P2_Burst_Read_Unpacked4(module,1000,0,Data_1,Data_2,Data_3,
     Data_4,1,3)
    REM Start next burst sequence
    state=0
    P2_Burst_Reset(pattern)
    P2_Burst_Start(pattern)
  EndIf
```

P2_Burst_Read_Unpacked8

P2_Burst_Read_Unpacked8 copies the measurement values of 8 channels from the memory of the specified module into 8 arrays.

Syntax

```
#Include ADwinPro_All.Inc
P2_Burst_Read_Unpacked8(module, count, startadr,
    array1[], array2[], array3[], array4[], array5[],
    array6[], array7[], array8[], array_idx,
    flowrate)
```

Parameters

module	Specified module address (1…15).	LONG
count	Amount of measurement values to be transferred.	LONG
startadr	Start address (0…67108860 = 2^{26} - 4) in the module memory: Address from which the measurement values.are read. The address must be divisible by 4.	LONG
arrayx[]	Destination arrays for the measurement values of channels 1…8. No float type and no FIFO array allowed.	ARRAY LONG ~~FLOAT~~
array_idx	Destination start index: Array element from which measurement values are stored.	LONG
flowrate	Evaluation for low-priority processes only: Parameter for data throughput. 1: slow. 2: medium. 3: fast.	LONG

Notes

The instruction be used if the burst sequence was initialized with 1 channel (see **P2_Burst_Init**, parameter channels).

The instruction stores the measurement values one after the other in the elements of the destination array: Channel 1 in array1, channel 2 in array2 etc.

In high-priority processes the maximum data throughput is used automatically; the parameter flowrate must be indicated all the same.

The higher the data troughput–in a low priority process only–is selected, the more it may happen that a process of higher priority is delayed.

See also

P2_Burst_Init, P2_Burst_Read_Unpacked1, P2_Burst_Read_Unpacked2, P2_Burst_Read_Unpacked4, P2_Burst_Start, P2_Burst_Status

Valid for

AIn-F-8/14 Rev. E, AIn-F-8/16 Rev. E

Example

See also examples for Continuous signal conversion on page 435.

```
#Include ADwinPro_All.Inc
#Define module 4

Dim Data_1[1000], Data_2[1000] As Long
Dim Data_3[1000], Data_4[1000] As Long
Dim Data_5[1000], Data_6[1000] As Long
Dim Data_7[1000], Data_8[1000] As Long
Dim state As Long
Dim rest As Long
Dim pattern As Long

Init:
  REM Initiate single burst sequence for channels 1…8 using 40ns
  REM period duration, save 1000 samples from address 0.
  P2_Burst_Init (module,255,0,1000,2,0)
  REM Start burst sequence
  pattern = Shift_Left(1,module-1) 'access single module only
  P2_Burst_Start(pattern)
  Processdelay=10000000
  REM State: Burst_sequence is running
  state=0

Event:
  REM Get number of samples still to do
  rest=P2_Burst_Status(module)
  If (rest=0) Then state=1
  If (state=1) Then
    REM All samples done: Read 1000 samples (fast) per channel and
    REM store in Data_1 to Data_4
    P2_Burst_Read_Unpacked4(module,1000,0,Data_1,Data_2,Data_3,
     Data_4,1,3)
    REM Start next burst sequence
    state=0
    P2_Burst_Reset(pattern)
    P2_Burst_Start(pattern)
  EndIf
```

P2_Burst_Reset

P2_Burst_Reset resets the data pointer of burst sequences on all specified modules.

Syntax

```
#Include ADwinPro_All.Inc
P2_Burst_Reset(module_pattern)
```

Parameters

module_pattern	Bit pattern to access the module addresses: Bit = 0: Ignore module. Bit = 1: Access module.	LONG

Bit no.	31:15	14	13	…	01	00
Module address	–	15	14	…	2	1

Notes

On modules AIn-F-x-16, the instruction is available since revision E04.

The instruction addresses all set modules at the same time. If the instruction is not valid for a set module unexpected results may occur.

The data pointer refers to the address in the module memory where the previous values have been stored. Resetting the data pointer will have the next values stored at the start address set by **P2_Burst_Init**. The data pointer may be read with **P2_Burst_Read_Index**.

See also

P2_Burst_Init, P2_Burst_Read_Index, P2_Burst_CRead_Unpacked1, P2_Burst_CRead_Unpacked2, P2_Burst_CRead_Unpacked4, P2_Burst_CRead_Unpacked8, P2_Burst_Read, P2_Burst_Stop

Valid for

AIn-F-4/14 Rev. E, AIn-F-4/16 Rev. E, AIn-F-8/14 Rev. E, AIn-F-8/16 Rev. E

Example

```
#Include ADwinPro_All.Inc
#Define module 4

Dim Data_1[1000] As Long
Dim state As Long
Dim remaining As Long
Dim pattern As Long

Init:
  REM Initiate single burst sequence for channel 1 using 20ns
  REM period duration, save 1000 samples from address 0.
  P2_Burst_Init (module,1,0,1000,1,0)
  REM start burst sequence
  pattern = Shift_Left(1,module-1) 'access module 4 only
  P2_Burst_Start(pattern)
  Processdelay=10000000
  state=0

Event:
  REM get number of remaining burst measurements
  remaining = P2_Burst_Status(module)
  REM all measurements are done: change status
  If (remaining = 0) Then state = 1
  If (state = 1) Then
    REM all burst measurements are done: read 1000 values fast
    REM and store into Data_1
    P2_Burst_Read_Unpacked1(module,1000,0,Data_1,1,3)
    REM start next burst sequence
    state=0
    P2_Burst_Reset(pattern)
    P2_Burst_Start(pattern)
  EndIf
```

P2_Burst_Start

P2_Burst_Start starts the burst measurement sequence on all specified modules at the same time.

Syntax

```
#Include ADwinPro_All.Inc
P2_Burst_Start(module_pattern)
```

Parameters

module_pattern	Bit pattern to set the module addresses: Bit = 0: Ignore module address. Bit = 1: Select module address.	LONG

Bit pattern	31:15	14	13	...	01	00
Module address	–	15	14	...	2	1

Notes

On modules AIn-F-x-16, the instruction is available since revision E04.

The instruction addresses all set modules at the same time. If the instruction is not valid for a set module unexpected results may occur.

See also

P2_Burst_Init, P2_Burst_Read_Index, P2_Burst_CRead_Unpacked1, P2_Burst_CRead_Unpacked2, P2_Burst_CRead_Unpacked4, P2_Burst_CRead_Unpacked8, P2_Burst_Status, P2_Burst_Stop

Valid for

AIn-F-4/14 Rev. E, AIn-F-4/16 Rev. E, AIn-F-8/14 Rev. E, AIn-F-8/16 Rev. E

Example

See also example "Continuous signal conversion" on page 1078.

```
#Include ADwinPro_All.Inc
#Define module 4

Dim Data_1[1000] As Long
Dim pattern As Long

Init:
  REM Initiate cont. burst sequence for channel 1 using 20ns
  REM period duration, save 2^26 samples from address 0.
  P2_Burst_Init (module,1,0,67108864,1,010b)
  REM Start burst sequence
  pattern = Shift_Left(1,module-1) 'one module only
  P2_Burst_Start(pattern)
  Processdelay=10000000

Event:
  REM Read previous 1000 samples from channel (slowly) and store
  REM in Data_1
  P2_Burst_CRead_Unpacked1(module,1000,Data_1,1,1)
```

P2_Burst_Status

P2_Burst_Status determines the number of burst measurements which are still to execute on the specified module.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = P2_Burst_Status(module)
```

Parameters

module	Specified module address (1…15).	LONG
ret_val	Number of measurements which are to execute.	LONG

Notes

On modules AIn-F-x-16, the instruction is available since revision E04.

The instruction be used for a single burst sequence only (see **P2_Burst_Init**).

If a burst measurement sequence is already finished, the function retuns 0 (zero).

See also

P2_Burst_Init, P2_Burst_Read_Index, P2_Burst_Read_Unpacked1, P2_Burst_Read_Unpacked2, P2_Burst_Read_Unpacked4, P2_Burst_Read_Unpacked8, P2_Burst_Start, P2_Burst_Stop, P2_Read_ADCF

Valid for

AIn-F-4/14 Rev. E, AIn-F-4/16 Rev. E, AIn-F-8/14 Rev. E, AIn-F-8/16 Rev. E

Example

```
#Include ADwinPro_All.Inc
#Define module 1

Dim Data_1[1000] As Long
Dim state As Long
Dim rest As Long
Dim pattern As Long

Init:
  REM Initiate single burst sequence for channel 1 using 20ns
  REM period duration, save 1000 samples from address 0.
  P2_Burst_Init (module,1,0,1000,1,0)
  REM Start burst sequence
  pattern = Shift_Left(1,module-1) 'one module only
  P2_Burst_Start(pattern)
  Processdelay=10000000
  REM State: Burst_sequence is running
  state=0

Event:
  REM Get number of samples still to do
  rest=P2_Burst_Status(module)
  REM All samples done: change state
  If (rest=0) Then state=1
  If (state=1) Then
    REM All samples done: Read 1000 samples (fast) and store
    REM in Data_1
    P2_Burst_Read_Unpacked1(module,1000,0,Data_1,1,3)
    REM Start next burst sequence
    state=0
    P2_Burst_Reset(pattern)
    P2_Burst_Start(pattern)
  EndIf
```

P2_Burst_Stop

P2_Burst_Stop stops a running burst-measurement sequence on all specified modules at the same time.

Syntax

```
#Include ADwinPro_All.Inc
P2_Burst_Stop(module_pattern)
```

Parameters

module_pattern	Bit pattern for accessing the module addresses: Bit = 0: ignore module. Bit = 1: access module.	LONG

Bit no.	31:15	14	13	…	01	00
Module address	–	15	14	…	2	1

Notes

On modules AIn-F-x-16, the instruction is available since revision E04.

An internal data pointer refers to the address in the module memory where the previous values have been stored. The data pointer may be read with **P2_Burst_Read_Index**.

A burst sequence being stopped can be resumed with **Burst_Start**.

P2_Burst_Reset resets the data pointer to the start address set by **P2_Burst_Init**. Thus, new values will overwrite previously saved values.

See also

P2_Burst_Init, P2_Burst_CRead_Unpacked1, P2_Burst_Read_Unpacked1, P2_Burst_Read, P2_Burst_Status

Valid for

AIn-F-4/14 Rev. E, AIn-F-4/16 Rev. E, AIn-F-8/14 Rev. E, AIn-F-8/16 Rev. E

Example

```
#Include ADwinPro_All.Inc
#Define module 4

Dim Data_1[1000] As Long
Dim i As Long
Dim pattern As Long

Init:
  REM Initiate cont. burst sequence for channel 1 using 20ns
  REM period duration, save 2^26 samples from address 0.
  P2_Burst_Init (module,1,0,67108864,1,0)
  REM Initiate single burst sequence for channels 1…8 using 40ns
  REM period duration, save 1000 samples from address 0.
  P2_Burst_Init (module,255,0,1000,2,0)
  REM Start burst sequence
  pattern = Shift_Left(1,module-1) 'one module only
  P2_Burst_Start(pattern)
  Processdelay=10000000

Event:
  REM Read last 1000 samples from channel (slowly) and store
  REM in Data_1
  P2_Burst_CRead_Unpacked1(module,1000,Data_1,1,1)
  REM Abort Burst sequence, if limit is exceeded
  For i = 1 To 1000
    If (Data_1[i]>5) Then
      P2_Burst_Stop(pattern)
    EndIf
  Next
```

P2_Set_Average_Filter

P2_Set_Average_Filter determines if the module calculates an average and of how many values the average is calculated.

Syntax

```
#Include ADwinPro_All.Inc
P2_Set_Average_Filter(module, mode)
```

Parameters

module	Specified module address (1…15).	LONG
mode	Filter mode (0…5): 0: Filter off = Original measurement values. 1: Average over 2^1 = 2 values. 2: Average over 2^2 = 4 values. 3: Average over 2^3 = 8 values. 4: Average over 2^4 = 16 values. 5: Average over 2^5 = 32 values.	LONG

Notes

The filter mode applies likewise for single value measurements and burst-measurements.

The average is always calculated from the previously converted measurement values. The calculation method is different for the module types:

- AIn-F-x/14: moving average.
  With each new measurement value, the average is calculated anew (from the last n values).
- AIn-F-x/16: arithmetic mean.
  For each arithmetic mean the selected number of measurement values is newly converted and calculated.

See also

P2_Burst_Init, P2_Burst_CRead_Unpacked1, P2_Burst_Read_Unpacked1, P2_Read_ADCF

Valid for

AIn-F-4/14 Rev. E, AIn-F-4/16 Rev. E, AIn-F-8/14 Rev. E, AIn-F-8/16 Rev. E

Example

```
#Include ADwinPro_All.Inc

Init:
  REM Initiate filter to average over 2 samples
  P2_Set_Average_Filter(1,1)
```

P2_ADCF

P2_ADCF executes a complete measurement on a Fast-ADC. The return value has a resolution of 16 bit.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = P2_ADCF(module, adc_no)
```

Parameters

module	Specified module address (1…15).	LONG
adc_no	Number (1…4 or 1…8) of the F-ADC.	LONG
ret_val	Result of the conversion (0…65535).	LONG

Notes

The function **P2_ADCF** is characterized by a sequence of several instructions:

P2_Start_ConvF	→	P2_Wait_EOCF	→	P2_Read_ADCF
Start A/D conversion		Wait for end of conversion		Read out the converted value

See also

P2_ADCF24, P2_ADCF_Mode, P2_ADCF_Read_Limit, P2_ADCF_Set_Limit, P2_Read_ADCF, P2_Start_ConvF, P2_Wait_EOCF

Valid for

AIn-F-4/14 Rev. E, AIn-F-4/16 Rev. E, AIn-F-4/18 Rev. E, AIn-F-8/14 Rev. E, AIn-F-8/16 Rev. E, AIn-F-8/18 Rev. E

Example

```
#Include ADwinPro_All.Inc
Dim value As Long

Event:
  value = P2_ADCF(1, 4)     'measure 16Bit value at analog input 4
```

P2_ADCF24

P2_ADCF24 executes a complete measurement on a Fast-ADC. The return value has a resolution of 24 bit.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = P2_ADCF24(module, adc_no)
```

Parameters

module	Specified module address (1…15).	LONG
adc_no	Number (1…4 or 1…8) of the F-ADC.	LONG
ret_val	Result of the conversion (0…16777215 = 2^{24}-1).	LONG

Notes

The function **P2_ADCF24** is characterized by a sequence of several instructions:

P2_Start_ConvF	→	P2_Wait_EOCF	→	P2_Read_ADCF24
Start A/D conversion		Wait for end of conversion		Read out the converted value

See also

P2_ADCF, P2_ADCF_Mode, P2_ADCF_Read_Limit, P2_ADCF_Set_Limit

Valid for

AIn-F-4/18 Rev. E, AIn-F-8/18 Rev. E

Example

```
#Include ADwinPro_All.Inc
Dim value As Long

Event:
REM Measure 24 bit value at analog input 4
  value = P2_ADCF24(1, 4)
```

P2_ADCF_Mode

P2_ADCF_Mode sets the working mode for all channels of the selected modules.

Syntax

```
#Include ADwinPro_All.Inc
P2_ADCF_Mode(module_pattern, mode)
```

Parameters

`module` Bit pattern to set the module addresses: LONG
Bit = 0: Ignore module address.
Bit = 1: Select module address.

Bit pattern	31:15	14	13	...	01	00
Module address	–	15	14	...	2	1

`mode` Working mode of the module: LONG

mode	Mode
0	Standard mode (default)
1	Timer Mode
3	Timer Mode with Multiplex option[a]
4	Event Mode
6	Event Mode with Multiplex option[a]

a. not available for AIn-F-4/16 and AIn-F-8/16

Notes

The instruction addresses all selected modules at the same time. If the instruction is not valid for a selected module unexpected results may occur.

Standard Mode

In standard mode, the processor module starts each conversion for each channel separately, e.g. using **P2_Start_Conv**.

Timer Mode

In timer mode, the module converts all channels independently and cyclic. Thus, the processor module is disburdened, furthermore only reading and processing the already converted values in the process. Each conversion on the module runs synchronously to the **Processdelay** of the process.

The timer mode can be enabled in the **Init:** section only; the instruction should be be placed at the section end.

The processor module should read the converted value in the **Event:** section first.

The following describes the action in detail:

P2_ADCF_Mode passes the currently set **Processdelay** of the process to the module. A certain time later, the module independently starts conversion on all channels. The on-module timer periodically restarts the conversion and – using the passed **Processdelay** – in synchrony to the process cycle; the maximum conversion rate is given in the module's hardware description.

In timer mode the end of conversion is regularly being reached, when the processor module starts its process cycle. If the processor reads the value somewhat later–e.g. because the process cycle starts retarded or the read instruction is not first of the process cycle–the next conversion may be starting already or even be completed. Thus, the processor module may omit single values or read them more than once.

Timer mode should be used in combination with a single high priority process only.

Timer Mode with Multiplex option

In timer mode with multiplex option the module runs with double speed than in normal timer mode, but can use only half of the channels. The processor module reads and processes a pair of converted values in each process cycle.

The measurement values can be read in pairs only, using the instructions **P2_Read_ADCF32**, **P2_Read_ADCF4_Packed**, **P2_Read_ADCF8_Packed**. The prior of both values is returned in the upper word, the subsequent value in the lower word.

Each analog signal must be connected to a pair of inputs: 1+2, 3+4, 5+6, 7+8; other pairing combinations are not allowed.

The **Processdelay** of the process must be even to synchronously clock the conversions.

Event Mode

In Event Mode each Event signal at the module's event input starts a conversion on all channels.

If the event input of the module is enabled with **P2_Event_Enable**, the module will send an event signal to the processor module. The event signal starts the (externally controlled) process cycle at the moment, when the end of conversion is reached. Thus, the processor module is disburdened, since it will only read and process the already converted values.

Event Mode with Multiplex option

In Event Mode with Multiplex option the module can run with double speed than in normal event mode, but can use only half of the channels.

Each analog signal msut be connected to a pair of inputs: 1+2, 3+4, 5+6, 7+8; other pairing combinations are not allowed.

If the event input of the module is enabled with **P2_Event_Enable**, the module will send an event signal to the processor module with the end of every second conversion.

The processor module must read a pair of values in every process cycle; suitable instructions are **P2_Read_ADCF32**, **P2_Read_ADCF4_Packed**, **P2_Read_ADCF8_Packed**. The prior of both values is returned in the upper word, the subsequent value in the lower word.

There is an event input only on modules with Sub-D plugs.

See also

P2_ADCF, P2_ADCF24, P2_ADCF_Read_Limit, P2_ADCF_Set_Limit, P2_Start_ConvF, P2_Wait_EOCF, P2_Read_ADCF, P2_Read_ADCF32, P2_Read_ADCF4_Packed, P2_Read_ADCF8_Packed, P2_Read_ADCF4_24B, P2_Read_ADCF8_24B

Valid for

AIn-F-4/16 Rev. E, AIn-F-4/18 Rev. E, AIn-F-8/16 Rev. E, AIn-F-8/18 Rev. E

Example

```
#Include ADwinPro_All.Inc
Dim value[4] As Long

Init:
  Rem …
  P2_ADCF_Mode(1,1)          'Switch on timer mode
                             'Last instruction of INIT section!

Event:
  P2_Read_ADCF4(1, value, 1) 'read values of ADC 1-4
  REM process values
```

P2_ADCF_Read_Limit

P2_ADCF_Read_Limit reads the limit-overrun and -underrun flags of all F-ADCs on the specified module.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = P2_ADCF_Read_Limit(module)
```

Parameters

module	Specified module address (1…15).	LONG
ret_val	Bit pattern representing the limit-overrun and -underrun flags.	LONG

Bit No.	31:24	23	22	21	20	19	18	17	16
	overrun of upper limit								
F-ADC no.	–	8	7	6	5	4	3	2	1

Bit No.	15:08	7	6	5	4	3	2	1	0
	underrun of lower limit								
F-ADC no.	–	8	7	6	5	4	3	2	1

Notes

The limits are set with **P2_ADCF_Set_Limit**.

Reading the flags resets all flags to zero.

We recommend to read the flags in the **Init:** section once, as to ensure all flags be reset. This is more important with an externally controlled process.

See also

P2_ADCF, P2_ADCF24, P2_ADCF_Mode, P2_ADCF_Set_Limit

Valid for

AIn-F-4/14 Rev. E, AIn-F-4/16 Rev. E, AIn-F-4/18 Rev. E, AIn-F-8/14 Rev. E, AIn-F-8/16 Rev. E, AIn-F-8/18 Rev. E

Example

```
#Include ADwinPro_All.Inc
Dim flags As Long

Init:
  P2_ADCF_Set_Limit(1, 2, 32768,256) 'set limits for channel 2
  flags = P2_ADCF_Read_Limit(1) 'read and reset flags

Event:
  flags = P2_ADCF_Read_Limit(1) 'read and reset flags
  If (flags And 10b = 10b) Then
    REM limit-overrun
    Rem …
  EndIf
  If (flags And 2000h = 2000h) Then
    REM limit-underrun
    Rem …
  EndIf
```

P2_ADCF_Set_Limit

P2_ADCF_Set_Limit sets the upper and lower limit for one F-ADC of the specified module.

Syntax

```
#Include ADwinPro_All.Inc
P2_ADCF_Set_Limit(module, adc_no, high, low)
```

Parameters

module	Specified module address (1…15).	LONG
adc_no	Number (1…4 or 1…8) of the F-ADC.	LONG
high	Upper limit (0…65535) of the channel. Default: 65535.	LONG
low	Lower limit (0…65535) of the channel. Default: 0.	LONG

Notes

This instruction will run as expected only, if the module does not run in standard working mode (see **P2_ADCF_Mode**).

If a converted value exceeds the upper limit, the channel's flag is set. **P2_ADC_Read_Limit** reads and thus resets the flags.

The same way a channel's flag is set for a converted value falling below the lower limit.

A limit-overrun or -underrun does not trigger an event signal.

See also

P2_ADCF, P2_ADCF24, P2_ADCF_Mode, P2_ADCF_Read_Limit

Valid for

AIn-F-4/14 Rev. E, AIn-F-4/16 Rev. E, AIn-F-4/18 Rev. E, AIn-F-8/14 Rev. E, AIn-F-8/16 Rev. E, AIn-F-8/18 Rev. E

Example

```
#Include ADwinPro_All.Inc
Dim flags As Long

Init:
  P2_ADCF_Set_Limit(1, 2, 32768,256) 'Set limits for channel 2
  flags = P2_ADCF_Read_Limit(1) 'read and reset flags

Event:
  flags = P2_ADCF_Read_Limit(1) 'read and reset flags
  If (flags And 10b = 10b) Then
    REM limit-overrun
    Rem …
  EndIf
  If (flags And 20000h = 20000h) Then
    REM limit-underrun
    Rem …
  EndIf
```

P2_ADCF_Reset_Min_Max

P2_ADCF_Reset_Min_Max resets the minimum and maximum values of selected channels of the specified module.

Syntax

```
#Include ADwinPro_All.Inc
P2_ADCF_Reset_Min_Max(module, channel_pattern)
```

Parameters

module	Specified module address (1…15).	LONG
channel_pattern	Bit pattern to select channels, where extreme minimum and maximum values are to be reset.	LONG

Bit No.	15:08	7	6	5	4	3	2	1	0
F-ADC No.	–	8	7	6	5	4	3	2	1

Notes

The maximum values are reset to zero, the minimum values to **0FFFFh**.

See also

P2_ADCF_Read_Min_Max4, P2_ADCF_Read_Min_Max8, P2_ADCF_Read_Limit, P2_ADCF_Set_Limit

Valid for

AIn-F-4/16 Rev. E, AIn-F-8/16 Rev. E

Example

see P2_ADCF_Read_Min_Max8

P2_ADCF_Read_Min_Max4

P2_ADCF_Read_Min_Max4 returns the minimum and maximum values of F-ADC 1…4 of the specified module in an array.

Syntax

```
#Include ADwinPro_All.Inc
P2_ADCF_Read_Min_Max4(module, array[], array_index)
```

Parameters

module	Specified module address (1…15).	LONG
array[]	Destination array, where extremal values are stored. The array must have at least array_index + 7 elements.	ARRAY LONG
array_index	Index of the destination array element, where the first extremal value is stored.	LONG

Notes

Extremal values are not reset by reading. For reset, use the instruction **P2_ADCF_Reset_Min_Max**.

The extremal values are stored in array[] in the following order (with array_index = n):

Array element	value, channel
array[n]	Min. channel 1
array[n+1]	Max. channel 1
array[n+2]	Min. channel 2
array[n+3]	Max. channel 2
array[n+4]	Min. channel 3
array[n+5]	Max. channel 3
array[n+6]	Min. channel 4
array[n+7]	Max. channel 4

See also

P2_ADCF_Read_Min_Max8, P2_ADCF_Reset_Min_Max, P2_ADCF_Read_Limit, P2_ADCF_Set_Limit

Valid for

AIn-F-4/16 Rev. E, AIn-F-8/16 Rev. E

Example

```
#Include ADwinPro_All.Inc
Dim Data_10[8] As Long
Dim i As Long

Init:
  P2_ADCF_Reset_Min_Max(1,1111b)'reset all 4 F-ADC

Event:
  Rem read high and low values of F-ADC 1…4
  P2_ADCF_Read_Min_Max4(1,Data_10,1)
  For i = 1 To 8 Step 2
    If (Data_10[i] < 2500) Then
      Rem minimum is below limit
      Rem …
      P2_ADCF_Reset_Min_Max(1,1111b)'reset all 4 F-ADC
    EndIf

    If (Data_10[i+1] > 50000) Then
      Rem value is above limit
      Rem …
    EndIf
  Next i
```

P2_ADCF_Read_Min_Max8

P2_ADCF_Read_Min_Max8 returns the minimum and maximum values of F-ADC 1…8 of the specified module in an array.

Syntax

```
#Include ADwinPro_All.Inc
P2_ADCF_Read_Min_Max8(module, array[], array_index)
```

Parameters

module	Specified module address (1…15).	LONG
array[]	Destination array, where extremal values are stored. The array must have at least array_index + 15 elements.	ARRAY LONG
array_index	Index of the destination array element, where the first extremal value is stored.	LONG

Notes

Extremal values are not reset by reading. For reset, use the instruction **P2_ADCF_Reset_Min_Max**.

The extremal values are stored in array[] in the following order (with array_index = n):

Array element	value, channel
array[n]	Min. channel 1
array[n+1]	Max. channel 1
array[n+2]	Min. channel 2
array[n+3]	Max. channel 2
array[n+4]	Min. channel 3
array[n+5]	Max. channel 3
array[n+6]	Min. channel 4
array[n+7]	Max. channel 4

Array element	value, channel
array[n+8]	Min. channel 5
array[n+9]	Max. channel 5
array[n+10]	Min. channel 6
array[n+11]	Max. channel 6
array[n+12]	Min. channel 7
array[n+13]	Max. channel 7
array[n+14]	Min. channel 8
array[n+15]	Max. channel 8

See also

P2_ADCF_Read_Min_Max4, P2_ADCF_Reset_Min_Max, P2_ADCF_Read_Limit, P2_ADCF_Set_Limit

Valid for

AIn-F-4/16 Rev. E, AIn-F-8/16 Rev. E

Example

```
#Include ADwinPro_All.Inc
Dim Data_4[16] As Long
Dim i As Long

Init:
  P2_ADCF_Reset_Min_Max(1,11111111b)'reset all 8 F-ADC

Event:
  Rem read high and low values of F-ADC 1…8
  P2_ADCF_Read_Min_Max8(1,Data_4,1)
  For i = 1 To 16 Step 2
    If (Data_4[i] < 2500) Then
      Rem minimum is below limit
      Rem …
      P2_ADCF_Reset_Min_Max(1,11111111b)'reset all 8 F-ADC
    EndIf

    If (Data_4[i+1] > 50000) Then
      Rem value is above limit
      Rem …
    EndIf
  Next i
```

P2_Read_ADCF

P2_Read_ADCF reads out the conversion result from an F-ADC of the specified module. The return value has a resolution of 16 bit.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = P2_Read_ADCF(module, adc_no)
```

Parameters

module	Specified module address (1…15).	LONG
adc_no	Number (1…4 or 1…8) of the F-ADC.	LONG
ret_val	Measurement value in the F-ADC register (0…65535).	LONG

Notes

The instructions **P2_Read_ADCF4**, **P2_Read_ADCF8**, **P2_Read_ADCF4_Packed**, **P2_Read_ADCF8_Packed** read conversion results very fast.

See also

P2_ADCF, P2_ADCF24, P2_Start_ConvF, P2_Wait_EOCF, P2_ADCF_Mode, P2_ADCF_Read_Limit, P2_ADCF_Set_Limit, P2_Read_ADCF32, P2_Read_ADCF_SConv, P2_Read_ADCF_SConv32, P2_Read_ADCF4, P2_Read_ADCF8, P2_Read_ADCF4_Packed, P2_Read_ADCF8_Packed

Valid for

AIn-F-4/14 Rev. E, AIn-F-4/16 Rev. E, AIn-F-4/18 Rev. E, AIn-F-8/14 Rev. E, AIn-F-8/16 Rev. E, AIn-F-8/18 Rev. E

Example

```
#Include ADwinPro_All.Inc
Dim value1 As Long          'Declaration

Event:
  P2_Start_ConvF(1,1)       'Start AD conversion
  P2_Wait_EOCF(1,1)         'Wait for end of conversion
  value1 = P2_Read_ADCF(1,1)'Read value from ADC
```

P2_Read_ADCF24

P2_Read_ADCF24 reads out the conversion result from an F-ADC of the specified module. The return value has a resolution of 24 bit.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = P2_Read_ADCF24(module, adc_no)
```

Parameters

module	Specified module address (1…15).	LONG
adc_no	Number (1…4 or 1…8) of the F-ADC.	LONG
ret_val	Measurement value in the F-ADC register (0…16777215 = 2^{24}-1).	LONG

Notes

The instructions **Read_ADCF4_24B**, **Read_ADCF8_24B** read conversion results very fast.

See also

P2_ADCF24, P2_Start_ConvF, P2_Wait_EOCF, P2_ADCF_Mode, P2_ADCF_Read_Limit, P2_ADCF_Set_Limit, P2_Read_ADCF_SConv24, P2_Read_ADCF4_24B, P2_Read_ADCF8_24B

Valid for

AIn-F-4/18 Rev. E, AIn-F-8/18 Rev. E

Example

```
#Include ADwinPro_All.Inc
Dim value1 As Long          'Declaration

Event:
  P2_Start_ConvF(1,1)       'Start AD conversion
  P2_Wait_EOCF(1,1)         'Wait for end of conversion
  value1 = P2_Read_ADCF24(1,1)'Read 24 bit value from the ADC
```

P2_Read_ADCF4

P2_Read_ADCF4 reads out the conversion results from the first 4 F-ADC of the specified module.

Syntax

```
#Include ADwinPro_All.Inc
P2_Read_ADCF4(module, array[], index)
```

Parameters

module	Specified module address (1…15).	LONG
array[]	Destination array, where conversion results are saved.	ARRAY LONG ~~FLOAT~~
index	Element index in the destination array, where the first conversion result is saved.	LONG

Notes

In any case the conversion results of the module's F-ADC 1…4 are read. The conversion results are saved subsequentially in the destination array array[], starting with element index.

The instruction is much faster than repeated use of **P2_Read_ADCF**.

See also

P2_ADCF, P2_ADCF24, P2_Start_ConvF, P2_Wait_EOCF, P2_Read_ADCF, P2_Read_ADCF8, P2_Read_ADCF4_Packed, P2_Read_ADCF8_Packed, P2_Read_ADCF_SConv

Valid for

AIn-F-4/14 Rev. E, AIn-F-4/16 Rev. E, AIn-F-4/18 Rev. E, AIn-F-8/14 Rev. E, AIn-F-8/16 Rev. E, AIn-F-8/18 Rev. E

Example

```
#Include ADwinPro_All.Inc
Dim value[4] As Long       'Array for conversion results

Init:
  P2_Start_ConvF(1,0Fh)    'Start AD conversion channels 1…4

Event:
  P2_Wait_EOCF(1,0Fh)      'Wait for end of conversion
  P2_Read_ADCF4(1,value,1) 'Read values of ADC 1…4
  P2_Start_ConvF(1,0Fh)    'Start new AD conversion
```

P2_Read_ADCF4_24B

P2_Read_ADCF4_24B reads out the conversion results from the first 4 F-ADC of the specified module. The return values have a resolution of 24 bits.

Syntax

```
#Include ADwinPro_All.Inc
P2_Read_ADCF4_24B(module, array[], index)
```

Parameters

module	Specified module address (1…15).	LONG
array[]	Destination array, where conversion results (24 bit) are saved.	ARRAY LONG ~~FLOAT~~
index	Element index in the destination array, where the first conversion result is saved.	LONG

Notes

In any case the conversion results of the module's F-ADC 1…4 are read. The conversion results are saved subsequentially in the destination array array[], starting with element index.

The instruction is much faster than repeated use of **P2_Read_ADCF24**.

See also

P2_ADCF24, P2_Start_ConvF, P2_Wait_EOCF, P2_Read_ADCF, P2_Read_ADCF_SConv24, P2_Read_ADCF8, P2_Read_ADCF4_Packed, P2_Read_ADCF8_Packed, P2_Read_ADCF8_24B

Valid for

AIn-F-4/18 Rev. E, AIn-F-8/18 Rev. E

Example

```
#Include ADwinPro_All.Inc
Dim value[4] As Long       'Array for conversion results

Init:
  P2_Start_ConvF(1,0Fh)    'Start AD conversion channels 1…4

Event:
  P2_Wait_EOCF(1,0Fh)      'Wait for end of conversion
  P2_Read_ADCF4_24B(1,value,1)'Read values of ADC 1…4
  P2_Start_ConvF(1,0Fh)    'Start new AD conversion
```

P2_Read_ADCF8

P2_Read_ADCF8 reads out the conversion results from all 8 F-ADCs of the specified module.

Syntax

```
#Include ADwinPro_All.Inc
P2_Read_ADCF8(module, array[], index)
```

Parameters

module	Specified module address (1…15).	LONG
array[]	Destination array, where conversion results are saved.	ARRAY LONG ~~FLOAT~~
index	Element index in the destination array, where the first conversion result is saved.	LONG

Notes

In any case the conversion results of the module's F-ADC 1…8 are read. The conversion results are saved subsequentially in the destination array array[], starting with element index.

The instruction is much faster than repeated use of **P2_Read_ADCF**.

See also

P2_ADCF, P2_ADCF24, P2_Start_ConvF, P2_Wait_EOCF, P2_Read_ADCF4, P2_Read_ADCF4_Packed, P2_Read_ADCF8_Packed, P2_Read_ADCF_SConv

Valid for

AIn-F-8/14 Rev. E, AIn-F-8/16 Rev. E, AIn-F-8/18 Rev. E

Example

```
#Include ADwinPro_All.Inc
Dim value[8] As Long       'Array for conversion results

Init:
  P2_Start_ConvF(1,0FFh)   'Start AD conversion channels 1…8

Event:
  P2_Wait_EOCF(1,0FFh)     'Wait for end of conversion
  P2_Read_ADCF8(1,value,1) 'Read values of ADC 1…8
  P2_Start_ConvF(1,0FFh)   'Start new AD conversion
```

P2_Read_ADCF8_24B

P2_Read_ADCF8_24B reads out the conversion results from all 8 F-ADC of the specified module. The return values have a resolution of 24 bits.

Syntax

```
#Include ADwinPro_All.Inc
P2_Read_ADCF8_24B(module, array[], index)
```

Parameters

module	Specified module address (1…15).	LONG
array[]	Destination array, where conversion results (24 bit) are saved.	ARRAY LONG ~~FLOAT~~
index	Element index in the destination array, where the first conversion result is saved.	LONG

Notes

In any case the conversion results of the module's F-ADC 1…8 are read. The conversion results are saved subsequentially in the destination array array[], starting with element index.

The instruction is much faster than repeated use of **P2_Read_ADCF24**.

See also

P2_ADCF, P2_ADCF24, P2_Start_ConvF, P2_Wait_EOCF, P2_Read_ADCF8, P2_Read_ADCF4_Packed, P2_Read_ADCF8_Packed, P2_Read_ADCF_SConv24

Valid for

AIn-F-8/18 Rev. E

Example

```
#Include ADwinPro_All.Inc
Dim value[8] As Long       'Array for conversion results

Init:
  P2_Start_ConvF(1,0FFh)   'Start AD conversion channels 1…8

Event:
  P2_Wait_EOCF(1,0FFh)     'Wait for end of conversion
  P2_Read_ADCF8_24B(1,value,1)'Read values of ADC 1…8
  P2_Start_ConvF(1,0FFh)   'Start new AD conversion
```

P2_Read_ADCF4_Packed

P2_Read_ADCF4_Packed reads out the conversion results from the first 4 F-ADC of the specified module.

Every 2 consecutive F-ADC results are returned in a single 32-bit value.

Syntax

```
#Include ADwinPro_All.Inc
P2_Read_ADCF4_Packed(module, array[], index)
```

Parameters

module	Specified module address (1…15).	LONG
array[]	Destination array, where conversion results are saved.	ARRAY LONG ~~FLOAT~~
index	Element index in the destination array, where the first conversion result is saved.	LONG

Notes

In any case the conversion results of the module's F-ADC 1…4 are read. The conversion result of an F-ADC with odd number is written into the lower word, of an F-ADC with even number into the higher word. The values are saved into the destination array array[] as follows:

Array element no.	Bit no.	
	31…16	15…0
index	F-ADC 2	F-ADC 1
index+1	F-ADC 4	F-ADC 3

See also

P2_ADCF, P2_ADCF24, P2_Start_ConvF, P2_Wait_EOCF, P2_Read_ADCF_SConv, P2_Read_ADCF4, P2_Read_ADCF8, P2_Read_ADCF8_Packed

Valid for

AIn-F-4/14 Rev. E, AIn-F-4/16 Rev. E, AIn-F-4/18 Rev. E, AIn-F-8/14 Rev. E, AIn-F-8/16 Rev. E, AIn-F-8/18 Rev. E

Example

```
#Include ADwinPro_All.Inc
Dim value[4] As Long       'Array for conversion results

Init:
  P2_Start_ConvF(1,0Fh)    'Start AD conversion channels 1…4

Event:
  P2_Wait_EOCF(1,0Fh)      'Wait for end of conversion
  P2_Read_ADCF4_Packed(1,value,1) 'Read values of ADC 1…4
  P2_Start_ConvF(1,0Fh)    'Start new AD conversion
```

P2_Read_ADCF8_Packed

P2_Read_ADCF8_Packed reads out the conversion results from all 8 F-ADC of the specified module.

Every 2 consecutive F-ADC results are returned in a single 32-bit value.

Syntax

```
#Include ADwinPro_All.Inc
P2_Read_ADCF8_Packed(module, array[], index)
```

Parameters

module	Specified module address (1…15).	LONG
array[]	Destination array, where conversion results are saved.	ARRAY LONG ~~FLOAT~~
index	Element index in the destination array, where the first conversion result is saved.	LONG

Notes

In any case the conversion results of the module's F-ADC 1…8 are read. The conversion result of an F-ADC with odd number is written into the lower word, of an F-ADC with even number into the higher word. The values are saved into the destination array array[] as follows:

Array element no.	Bit no.	
	31…16	15…0
index	F-ADC 2	F-ADC 1
index+1	F-ADC 4	F-ADC 3
index+2	F-ADC 6	F-ADC 5
index+3	F-ADC 8	F-ADC 7

See also

P2_ADCF, P2_ADCF24, P2_Start_ConvF, P2_Wait_EOCF, P2_Read_ADCF_SConv, P2_Read_ADCF4, P2_Read_ADCF8, P2_Read_ADCF4_Packed

Valid for

AIn-F-8/14 Rev. E, AIn-F-8/16 Rev. E, AIn-F-8/18 Rev. E

Example

```
#Include ADwinPro_All.Inc
Dim value[8] As Long       'Array for conversion results

Init:
  P2_Start_ConvF(1,0Fh)    'Start AD conversion channels 1…8

Event:
  P2_Wait_EOCF(1,0Fh)      'Wait for end of conversion
  P2_Read_ADCF8_Packed(1,value,1) 'Read values of ADC 1…8
  P2_Start_ConvF(1,0Fh)    'Start new AD conversion
```

P2_Read_ADCF32

P2_Read_ADCF32 reads the conversion result from 2 consecutive F-ADCs of the specified module and returns them in a single 32-bit value.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = P2_Read_ADCF32(module, adc_pair)
```

Parameters

`module` Specified module address (1…15). LONG

`adc_pair` Number (1…2 or 1…4) of the F-ADC pair to read: LONG

adc_pair	1	2	3	4
F-ADC-no.	1, 2	3, 4	5, 6	7, 8

`ret_val` The measurement values in the F-ADC registers (0…65535 each); one measurement value in the lower and one in the higher word. LONG

Notes

The conversion result of the odd numbered ADC (`adc_pair*2-1`) is written into the low-word, of the even numbered ADC (`adc_pair*2`) into the high-word.

The number of the first F-ADC must be odd. Therefore it is for instance not possible to read out the conversion results of the F-ADCs 2 and 3 with one instruction.

See also

P2_ADCF, P2_ADCF24, P2_Start_ConvF, P2_Wait_EOCF, P2_Read_ADCF, P2_Read_ADCF_SConv, P2_Read_ADCF_SConv32, P2_Read_ADCF4, P2_Read_ADCF8, P2_Read_ADCF4_Packed, P2_Read_ADCF8_Packed

Valid for

AIn-F-4/14 Rev. E, AIn-F-4/16 Rev. E, AIn-F-4/18 Rev. E, AIn-F-8/14 Rev. E, AIn-F-8/16 Rev. E, AIn-F-8/18 Rev. E

Example

```
#Include ADwinPro_All.Inc
Dim value1 As Long          'Declaration

Event:
  P2_Start_ConvF(1,3)       'Start AD conversion on ADC1 and ADC2
  P2_Wait_EOCF(1,3)         'Wait for the end of conversions
  value1 = P2_Read_ADCF32(1,1)'Read values of ADC1 and ADC2
```

P2_Read_ADCF_SConv

P2_Read_ADCF_SConv reads the conversion result from an F-ADC of the specified module and immediately starts a new conversion.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = P2_Read_ADCF_SConv(module, adc_no)
```

Parameters

module	Specified module address (1…15).	LONG
adc_no	Number (1…4 or 1…8) of the F-ADC.	LONG
ret_val	Measurement value in the F-ADC register (0…65535).	LONG

See also

P2_ADCF, P2_ADCF24, P2_Start_ConvF, P2_Wait_EOCF, P2_Read_ADCF32, P2_Read_ADCF_SConv32, P2_Read_ADCF4, P2_Read_ADCF8, P2_Read_ADCF4_Packed, P2_Read_ADCF8_Packed

Valid for

AIn-F-4/16 Rev. E, AIn-F-4/18 Rev. E, AIn-F-8/16 Rev. E, AIn-F-8/18 Rev. E

Example

```
#Include ADwinPro_All.Inc
Dim i As Long
Dim Data_1[1000] As Long  'Declaration

Init:
  i=1
  P2_Start_ConvF(1,1)      'Start A/D converter

Event:
  P2_Wait_EOCF(1,1)
  Data_1[i] = P2_Read_ADCF_SConv(1,1)'Read and start ADC
  Inc(i)                   'Increment index
  If (i=1001) Then End     'End process after 1000 measurement
                           ''values
```

P2_Read_ADCF_SConv24

P2_Read_ADCF_SConv24 reads the conversion result from an F-ADC of the specified module and immediately starts a new conversion.

The return value has a resolution of 24 bit.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = P2_Read_ADCF_SConv24(module, adc_no)
```

Parameters

module	Specified module address (1…15).	LONG
adc_no	Number (1…4 or 1…8) of the F-ADC.	LONG
ret_val	Measurement value in the F-ADC register (0…16777215 = 2^{24}-1).	LONG

See also

P2_ADCF, P2_ADCF24, P2_Read_ADCF, P2_Read_ADCF4, P2_Read_ADCF8, P2_Read_ADCF4_Packed, P2_Read_ADCF8_Packed, P2_Read_ADCF32, P2_Read_ADCF_SConv32, P2_Start_ConvF, P2_Wait_EOCF

Valid for

AIn-F-4/18 Rev. E, AIn-F-8/18 Rev. E

Example

```
#Include ADwinPro_All.Inc
Dim i As Long
Dim Data_1[1000] As Long   'Declaration

Init:
  i=1
  P2_Start_ConvF(1,1)      'start A/D conversion

Event:
  P2_Wait_EOCF(1,1)
  Data_1[i] = P2_Read_ADCF_SConv24(1,1)'Read out + start
                           'AD converter 24 bit
  Inc(i)                   'Increment index
  If (i=1001) Then End     'End process after 1000 measurement
                           ''values
```

P2_Read_ADCF_SConv32

P2_Read_ADCF_SConv32 reads the conversion results from 2 F-ADCs of the specified module and returns them in a 32-bit value.

Then a new conversion is started immediately.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = P2_Read_ADCF_SConv32(module, adc_pair)
```

Parameters

module	Specified module address (1…15).	LONG
adc_pair	Number of the F-ADC pair to read: 1…2 or 1…4.	LONG
ret_val	The return value (32-bit) contains the measurement data of 2 consecutive F-ADCs (16-bit each: 0…65535); one measurement value is in the lower word and one in the upper word.	LONG

adc_pair	1	2	3	4
F-ADC-no.	1, 2	3, 4	5, 6	7, 8

Notes

The conversion result of the odd numbered ADC (adc_pair*2-1) is written into the low-word, of the even numbered ADC (adc_pair*2) into the high-word.

The number of the first F-ADC must be odd. Therefore it is for instance not possible to read out the conversion results of the F-ADCs 2 and 3 with one instruction.

See also

P2_ADCF, P2_ADCF24, P2_Read_ADCF, P2_Read_ADCF4, P2_Read_ADCF8, P2_Read_ADCF4_Packed, P2_Read_ADCF8_Packed, P2_Read_ADCF32, P2_Read_ADCF_SConv, P2_Start_ConvF, P2_Wait_EOCF

Valid for

AIn-F-4/16 Rev. E, AIn-F-4/18 Rev. E, AIn-F-8/16 Rev. E, AIn-F-8/18 Rev. E

Example

```
#Include ADwinPro_All.Inc
Dim value As Long          'Declaration

Init:
  P2_Start_ConvF(1,3)      'Start AD conversion

Event:
  P2_Wait_EOCF(1,3)        'Wait for end of conversion
  value = P2_Read_ADCF_SConv32(1,1)'read values from ADC1 and
                           'ADC2, start conversion of both ADCs
```

P2_Set_Gain

P2_Set_Gain sets the operating mode of a channel on the selected module and thus the gain and measurement range.

Syntax

```
#Include ADwinPro_All.Inc
P2_Set_Gain(module, channel, mode)
```

Parameters

module	Specified module address (1…15).	LONG
channel	Number (1…4 or 1…8) of the F-ADC.	LONG
mode	Operating mode (0…3) of the ADC: Sets the gain of the input signal. With gain, the measurement range of input signals changes proprtionally.	LONG

Operating mode	Gain	Measure-ment range
mode	2^n	±10V / 2^n
0	1	±10V
1	2	±5V
2	4	±2.5V
3	8	±1.25V

See also

P2_ADCF24, P2_Read_ADCF, P2_Start_ConvF, P2_Wait_EOCF

Valid for

- / -

Example

```
#Include ADwinPro_All.Inc
#Define ainadr 1            'Module address AIN module

Init:
  Rem Set voltage range of channel 4 to mode 1
  Rem Measuring range: +5V…-5V
  P2_Set_Gain(ainadr,4,1)

Event:
  Par_1 = P2_ADCF(1,4)      'Measure analog input 4
```

P2_Start_ConvF

P2_Start_ConvF starts the conversion on one or more F-ADCs of the specified module.

Syntax

```
#Include ADwinPro_All.Inc
P2_Start_ConvF(module, adc_pattern)
```

Parameters

module	Specified module address (1…15).	LONG
adc_pattern	Bit pattern that determines the ADCs where conversion is to be started (see table).	LONG

Bit no.	31:8	7	6	5	4	3	2	1	0
Converter no.	–	8	7	6	5	4	3	2	1

Notes

Determining the ADCs is made with a bit pattern, so that the conversion of several converters can be started at the same time. For instance, when starting the A/D converters 1 and 3 the bit pattern **101b** (decimal 5) has to be transferred.

With module AIn-F-x/14, **P2_Start_ConvF** is not required, since the ADCs run continuously with fixed conversion rate. Attention: If the instruction is used yet, the current measurement value is copied into the latch register; a following **P2_Read_ADCF** will read the copied (not the current) value, even if this happens much later.

You can synchronously start conversions on several modules with **P2_Sync_All**. You can then disable selected channels for the synchronization with **P2_Sync_Enable**.

Conversions on several modules can also be executed synchronously, if you have released the corresponding modules with **P2_Sync_Mode** for synchronization.
As soon as you start a conversion on the master module, you start simultaneously conversions on all channels of the slave modules. You will have the same effect with event-controlled modules, as soon as a signal is provided at the event-input.

See also

P2_ADCF, P2_ADCF24, P2_Read_ADCF, P2_Read_ADCF4, P2_Read_ADCF8, P2_Read_ADCF4_Packed, P2_Read_ADCF8_Packed, P2_Wait_EOCF, P2_Sync_All, P2_Sync_Enable, P2_Sync_Mode

Valid for

AIn-F-4/16 Rev. E, AIn-F-4/18 Rev. E, AIn-F-8/16 Rev. E, AIn-F-8/18 Rev. E

Example

```
#Include ADwinPro_All.Inc
Dim value As Long          'Declaration

Event:
  P2_Start_ConvF(1,1)      'Start AD conversion channel 1
  P2_Wait_EOCF(1,1)        'Wait for end of conversion
  value = P2_Read_ADCF(1,1)'Read value from ADC
```

P2_Wait_EOCF

P2_Wait_EOCF waits until the end of conversion on all F-ADCs of the specified module.

Syntax

```
#Include ADwinPro_All.Inc
P2_Wait_EOCF(module, adc_pattern)
```

Parameters

module	Specified module address (1…15).	LONG
adc_pattern	Bit pattern that determines the ADCs, whose end of conversion shall be awaited (see table).	LONG

Bit no.	31:8	7	6	5	4	3	2	1	0
Converter no.	–	8	7	6	5	4	3	2	1

Notes

Determining the ADCs is made bit by bit, so that the conversion can be started from several converters at the same time. For instance, when starting the A/D converters 1 and 3 the bit pattern **101b** (decimal 5) has to be transferred.

With module AIn-F-x/14, **P2_Wait_ConvF** is not required, since the ADCs run continuously with fixed conversion rate. The instruction has no effect except for loss of processor time.

See also

P2_ADCF, P2_ADCF24, P2_Start_ConvF, P2_Read_ADCF, P2_Read_ADCF4, P2_Read_ADCF8, P2_Read_ADCF4_Packed, P2_Read_ADCF8_Packed

Valid for

AIn-F-4/16 Rev. E, AIn-F-4/18 Rev. E, AIn-F-8/16 Rev. E, AIn-F-8/18 Rev. E

Example

```
#Include ADwinPro_All.Inc
Dim value As Long          'Declaration

Event:
  P2_Start_ConvF(1,1)      'Start AD conversion
  P2_Wait_EOCF(1,1)        'Wait for end of conversion
  value = P2_Read_ADCF(1,1)'Read value from ADC
```

3.5 Pro II: Output Modules

This section describes instructions which apply to Pro II modules with Analog Outputs:

– P2_DAC (page 128)
– P2_DAC4 (page 129)
– P2_DAC4_Packed (page 130)
– P2_DAC8 (page 132)
– P2_DAC8_Packed (page 133)
– P2_Start_DAC (page 134)
– P2_Write_DAC (page 135)
– P2_Write_DAC4 (page 136)
– P2_Write_DAC4_Packed (page 137)
– P2_Write_DAC8 (page 138)
– P2_Write_DAC8_Packed (page 139)
– P2_Write_DAC32 (page 140)
– P2_DAC1_DIO (page 141)
– P2_DAC_Ramp_Write (page 142)
– P2_DAC_Ramp_Status (page 144)
– P2_DAC_Ramp_Buffer_Free (page 146)
– P2_DAC_Ramp_Stop (page 147)

In the Instruction List sorted by Module Types (annex A.2) you will find which of the functions corresponds to the *ADwin-Pro II* modules.

P2_DAC

P2_DAC outputs an (analog) voltage on the channel of the specified module, that corresponds to the indicated digital value.

Syntax

```
#Include ADwinPro_All.Inc
P2_DAC(module,dac_no,value)
```

Parameters

module	Specified module address (1…15).	LONG
dac_no	Number (1…4 or 1…8) of the output.	LONG
value	value to output (0…65535).	LONG

Notes

We recommend to use the instructions **P2_DAC4** or **P2_DAC8** instead, since they can output more values than **P2_DAC** in the same time.

P2_DAC is characterized by a sequence of several commands:

P2_Write_DAC	→ P2_Start_DAC
Transfer digital value into DAC register	Start D/A conversion

See also

P2_DAC4, P2_DAC4_Packed, P2_DAC8, P2_DAC8_Packed, P2_Start_DAC, P2_Write_DAC, P2_Write_DAC4, P2_Write_DAC4_Packed, P2_Write_DAC8, P2_Write_DAC8_Packed, P2_Write_DAC32

Valid for

AOut-1/16 Rev. E, AOut-4/16 Rev. E, AOut-4/16-TiCo Rev. E, AOut-8/16 Rev. E, AOut-8/16-TiCo Rev. E, MIO-4 Rev. E, MIO-4-ET1 Rev. E

Example

```
REM Digital proportionl controller
#Include ADwinPro_All.Inc
#Define setpoint Par_1     'set point in digits
#Define gain FPar_2
Dim deviation, actuate As Long

Event:
  deviation = setpoint - P2_ADC(1,1)'calculate control deviation
  actuate = deviation * gain 'calculate actuating value
  P2_DAC(1,1,actuate)       'output actuating value
```

P2_DAC4

P2_DAC4 outputs 4 digital values from an array on the DAC 1…4 of the specified module as (analog) voltages.

Syntax

```
#Include ADwinPro_All.Inc
P2_DAC4(module,array[],index)
```

Parameters

module	Specified module address (1…15).	LONG
array[]	Array with values (0…65535) to be output.	ARRAY LONG ~~FLOAT~~
index	Index of the first array element to be output.	LONG

Notes

P2_DAC4 is characterized by a sequence of several commands:

P2_Write_DAC4	→ P2_Start_DAC
Transfer digital value into DAC register	Start D/A conversion.

See also

P2_DAC, P2_DAC4_Packed, P2_DAC8, P2_DAC8_Packed, P2_Start_DAC, P2_Write_DAC, P2_Write_DAC4, P2_Write_DAC4_Packed, P2_Write_DAC8, P2_Write_DAC8_Packed, P2_Write_DAC32

Valid for

AOut-4/16 Rev. E, AOut-4/16-TiCo Rev. E, AOut-8/16 Rev. E, AOut-8/16-TiCo Rev. E, MIO-4 Rev. E, MIO-4-ET1 Rev. E

Example

```
REM Digital proportionl controller for 4 channels
#Include ADwinPro_All.Inc
#Define setpoint Par_1     'set point in digits
#Define gain FPar_2
Dim i, deviation As Long
Dim input[4], actuate[4] As Long

Event:
  P2_Read_ADCF4(1,input,1) 'read 4 input values
  For i = 1 To 4
    deviation = setpoint - input[i] 'calculate control deviation
    actuate[i] = deviation * gain 'calculate actuating value
  Next i
  P2_DAC4(2,actuate,1)      'output 4 actuating values
```

P2_DAC4_Packed

P2_DAC4_Packed outputs 4 packed digital values from an array on the DAC 1…4 of the specified module as (analog) voltages.

Syntax

```
#Include ADwinPro_All.Inc
P2_DAC4_Packed(module,array[],index)
```

Parameters

module	Specified module address (1…15).	LONG
array[]	Array with values (0…65535) to be output as packed data: Each 32 Bit array element holds 2 values of 16 Bit.	ARRAY, LONG, ~~FLOAT~~
index	Index (³1) of the first array element to be output.	LONG

Notes

P2_DAC4_Packed is characterized by a sequence of two commands:

P2_Write_DAC4_Packed	→	**P2_Start_DAC**
Transfer 4 digital values into DAC registers.		Start D/A conversion.

Every 2 array elements of 32 Bit hold 4 digital values of 16 Bit in the following order:

Array element	array[n+1]		array[n]	
Bit no.	31:16	15:00	31:16	15:00
Digital value for	DAC4	DAC3	DAC2	DAC1

See also

P2_DAC, P2_DAC4, P2_DAC8, P2_DAC8_Packed, P2_Start_DAC, P2_Write_DAC, P2_Write_DAC4, P2_Write_DAC4_Packed, P2_Write_DAC8, P2_Write_DAC8_Packed, P2_Write_DAC32

Valid for

AOut-4/16 Rev. E, AOut-4/16-TiCo Rev. E, AOut-8/16 Rev. E, AOut-8/16-TiCo Rev. E, MIO-4 Rev. E, MIO-4-ET1 Rev. E

Example

```
REM Digital proportionl controller for 4 channels
#Include ADwinPro_All.Inc
#Define setpoint Par_1     'set point in digits
#Define gain FPar_2
Dim i, deviation1, deviation2 As Long
Dim input[4], actuate[4] As Long

Event:
  P2_Read_ADCF4_Packed(1,input,1)'read 4 input values
  For i = 1 To 2
    REM Calculare control deviations
    deviation1 = setpoint - (input[i] And 0FFh)
    deviation2 = setpoint - (Shift_Right(input[i],16) And 0FFh)
    REM Calculate actuating values and store
    actuate[i] = Shift_Left(deviation2*gain,16) + deviation1*gain
  Next i
  P2_DAC4_Packed(2,actuate,1)'output 4 actuating values
```

P2_DAC8

P2_DAC8 outputs 8 digital values from an array on the DAC 1…8 of the specified module as (analog) voltages.

Syntax

```
#Include ADwinPro_All.Inc
P2_DAC8(module,array[],index)
```

Parameters

module	Specified module address (1…15).	LONG
array[]	Array with values (0…65535) to be output.	ARRAY LONG ~~FLOAT~~
index	Index (³1) of the first array element to be written.	LONG

Notes

P2_DAC8 is characterized by a sequence of several commands:

P2_Write_DAC8	→ P2_Start_DAC
transfer 8 digital values into the DAC register.	Start D/A conversion.

See also

P2_DAC, P2_DAC4, P2_DAC4_Packed, P2_DAC8_Packed, P2_Start_DAC, P2_Write_DAC, P2_Write_DAC4, P2_Write_DAC4_Packed, P2_Write_DAC8, P2_Write_DAC8_Packed, P2_Write_DAC32

Valid for

AOut-8/16 Rev. E, AOut-8/16-TiCo Rev. E

Example

```
REM Digital proportionl controller for 8 channels
#Include ADwinPro_All.Inc
#Define setpoint Par_1     'set point in digits
#Define gain FPar_2
Dim i, deviation As Long
Dim input[8], actuate[8] As Long

Event:
  P2_Read_ADCF8(1,input,1) 'read 8 input values
  For i = 1 To 8
    deviation = setpoint - input[i] 'calculate control deviation
    actuate[i] = deviation * gain 'calculate actuating value
  Next i
  P2_DAC8(2,actuate,1)     'output 8 actuating values
```

P2_DAC8_Packed

P2_DAC8_Packed outputs 8 packed digital values from an array on the DAC 1…8 of the specified module as (analog) voltages.

Syntax

```
#Include ADwinPro_All.Inc
P2_DAC8_Packed(module,array[],index)
```

Parameters

module	Specified module address (1…15).	LONG
array[]	Array with values (0…65535) to be output as packed data: Each 32 bit array element holds 2 values of 16 bit.	ARRAY LONG ~~FLOAT~~
index	Index (³1) of the first array element to be written.	LONG

Notes

P2_DAC8_Packed is characterized by a sequence of two commands:

P2_Write_DAC8 →	**P2_Start_DAC**
Transfer digital value into DAC register	Start D/A conversion.

Every 4 array elements of 32 Bit hold 8 digital values of 16 Bit in the following order:

Array element	array[n+3]		array[n+2]		array[n+1]		array[n]	
Bit no.	31:16	15:00	31:16	15:00	31:16	15:00	31:16	15:00
Digital value for	DAC8	DAC7	DAC6	DAC5	DAC4	DAC3	DAC2	DAC1

See also

P2_DAC, P2_DAC4, P2_DAC4_Packed, P2_DAC8, P2_Start_DAC, P2_Write_DAC, P2_Write_DAC4, P2_Write_DAC4_Packed, P2_Write_DAC8, P2_Write_DAC8_Packed, P2_Write_DAC32

Valid for

AOut-8/16 Rev. E, AOut-8/16-TiCo Rev. E

Example

```
REM Digital proportionl controller for 8 channels
#Include ADwinPro_All.Inc
#Define setpoint Par_1     'set point in digits
#Define gain FPar_2
Dim i, deviation1, deviation2 As Long
Dim input[8], actuate[8] As Long

Event:
  P2_Read_ADCF8_Packed(1,input,1)'read 8 input values
  For i = 1 To 4
    REM Calculare control deviations
    deviation1 = setpoint - (input[i] And 0FFh)
    deviation2 = setpoint - (Shift_Right(input[i],16) And 0FFh)
    REM Calculate actuating values and store
    actuate[i] = Shift_Left(deviation2*gain,16) + deviation1*gain
  Next i
  P2_DAC8_Packed(2,actuate,1)'output 8 actuating values
```

P2_Start_DAC

P2_Start_DAC starts the conversion or output of all DAC on the specified module

Syntax

```
#Include ADwinPro_All.Inc
P2_Start_DAC(module)
```

Parameters

module	Specified module address (1…15).	LONG

Notes

You can synchronously start conversions on several modules with the instruction **P2_Sync_All**. You can disable selected channels for synchonization using **P2_Sync_Enable**.

See also

P2_DAC, P2_DAC4, P2_DAC4_Packed, P2_DAC8, P2_DAC8_Packed, P2_Write_DAC, P2_Write_DAC4, P2_Write_DAC4_Packed, P2_Write_DAC8, P2_Write_DAC8_Packed, P2_Write_DAC32, P2_Sync_All

Valid for

AOut-1/16 Rev. E, AOut-4/16 Rev. E, AOut-4/16-TiCo Rev. E, AOut-8/16 Rev. E, AOut-8/16-TiCo Rev. E, MIO-4 Rev. E, MIO-4-ET1 Rev. E

Example

```
REM Simultaneous output of two different signals
REM on the outputs 1 and 2 of a D/A module.
#Include ADwinPro_All.Inc
Dim i As Long
Init:
  i = 0

Event:
  P2_Write_DAC(1,1,i)       'Set output register DAC1
  P2_Write_DAC(1,2,65535-i)'Set output register DAC2
  P2_Start_DAC(1)           'Start output on all DAC
  Inc(i)
  If (i=65535) Then i=0
```

P2_Write_DAC

P2_Write_DAC writes a digital value into the output register of a DAC on the specified module.

Syntax

```
#Include ADwinPro_All.Inc
P2_Write_DAC(module,dac_no,value)
```

Parameters

module	Specified module address (1…15).	LONG
dac_no	Number (1…4 or 1…8) of the output.	LONG
value	value to output (0…65535).	LONG

Notes

P2_Start_DAC starts the conversion of the digital value into an output voltage.

We recommend to use the instructions **P2_Write_DAC4** or **P2_Write_DAC8** instead, since they can output more values than **P2_Write_DAC** in the same time.

See also

P2_DAC, P2_DAC4, P2_DAC4_Packed, P2_DAC8, P2_DAC8_Packed, P2_Start_DAC, P2_Write_DAC4, P2_Write_DAC4_Packed, P2_Write_DAC8, P2_Write_DAC8_Packed, P2_Write_DAC32

Valid for

AOut-1/16 Rev. E, AOut-4/16 Rev. E, AOut-4/16-TiCo Rev. E, AOut-8/16 Rev. E, AOut-8/16-TiCo Rev. E, MIO-4 Rev. E, MIO-4-ET1 Rev. E

Example

```
REM Simultaneous output of four different signals
REM on the output channels 1, 2, 3 and 4 of a D/A module
REM The signals are filed in four DATA arrays and
REM can be transferred from the PC before program start
#Include ADwinPro_All.Inc
Dim i As Long
Dim Data_1[1000], Data_2[1000], Data_3[1000] As Long
Dim Data_4[1000] As Long

Init:
  i = 1

Event:
  P2_Write_DAC(1,1,Data_1[i])'Set output register DAC1
  P2_Write_DAC(1,2,Data_2[i])'Set output register DAC2
  P2_Write_DAC(1,3,Data_3[i])'Set output register DAC3
  P2_Write_DAC(1,4,Data_4[i])'Set output register DAC4
  P2_Start_DAC(1)           'Start output on all DAC
  Inc(i)
  If (i>1000) Then i = 1
```

P2_Write_DAC4

P2_Write_DAC4 writes 4 digital values from an array into the output registers of the DAC 1…4 of the specified module.

P2_Start_DAC starts the conversion of the digital values into the output voltages.

Syntax

```
#Include ADwinPro_All.Inc
P2_Write_DAC4(module,array[],index)
```

Parameters

module	Specified module address (1…15).	LONG
array[]	Array with values (0…65535) to be output.	ARRAY LONG ~~FLOAT~~
index	Index of the first array element to be output.	LONG

See also

P2_DAC, P2_DAC4, P2_DAC4_Packed, P2_DAC8, P2_DAC8_Packed, P2_Start_DAC, P2_Write_DAC, P2_Write_DAC4_Packed, P2_Write_DAC8, P2_Write_DAC8_Packed, P2_Write_DAC32

Valid for

AOut-4/16 Rev. E, AOut-4/16-TiCo Rev. E, AOut-8/16 Rev. E, AOut-8/16-TiCo Rev. E, MIO-4 Rev. E, MIO-4-ET1 Rev. E

Example

```
REM Simultaneous output of four different signals
REM on the output channels 1, 2, 3 and 4 of a D/A module
REM The signals are stored sequentially in the array Data_1
REM and may be transferred before program start from the PC

#Include ADwinPro_All.Inc
Dim i As Long
Dim Data_1[4000] As Long

Init:
  i = 1

Event:
  REM Set output registers DAC1...DAC4
  P2_Write_DAC4(1,Data_1,(i-1)*4+i)
  P2_Start_DAC(1)          'Start output on all DAC
  Inc(i)
  If (i>1000) Then i = 1
```

P2_Write_DAC4_Packed

P2_Write_DAC4_Packed writes 4 packed digital values from an array into the output registers of the DAC 1…4 of the specified module.

P2_Start_DAC starts the conversion of the digital values into the output voltages.

Syntax

```
#Include ADwinPro_All.Inc
P2_Write_DAC4_Packed(module,array[],index)
```

Parameters

module	Specified module address (1…15).	LONG
array[]	Array with values (0…65535) to be output as packed data: Each 32 bit array element holds 2 values of 16 bit.	ARRAY LONG ~~FLOAT~~
index	Index of the first array element to be output.	LONG

Notes

Every 2 array elements of 32 Bit hold 4 digital values of 16 Bit in the following order:

Array element	array[n+1]		array[n]	
Bit no.	31:16	15:00	31:16	15:00
Digital value for	DAC4	DAC3	DAC2	DAC1

See also

P2_DAC, P2_DAC4, P2_DAC4_Packed, P2_DAC8, P2_DAC8_Packed, P2_Start_DAC, P2_Write_DAC, P2_Write_DAC4, P2_Write_DAC8, P2_Write_DAC8_Packed, P2_Write_DAC32

Valid for

AOut-4/16 Rev. E, AOut-4/16-TiCo Rev. E, AOut-8/16 Rev. E, AOut-8/16-TiCo Rev. E, MIO-4 Rev. E, MIO-4-ET1 Rev. E

Example

```
REM Simultaneous output of four different signals
REM on the output channels 1, 2, 3 and 4 of a D/A module
REM The signals are stored sequentially in the array Data_1
REM packed and can be transferred from the PC before program start

#Include ADwinPro_All.Inc
Dim i As Long
Dim Data_1[4000] As Long

Init:
  i = 1

Event:
REM Set output registers DAC1...DAC4
  P2_Write_DAC4_Packed(1,Data_1,(i-1)*2+i)
  P2_Start_DAC(1)           'Start output on all DAC
  Inc(i)
  If (i>1000) Then i = 1
```

P2_Write_DAC8

P2_Write_DAC8 writes 8 digital values from an array into the output registers of the DAC 1…8 of the specified module.

P2_Start_DAC starts the conversion of the digital values into the output voltages.

Syntax

```
#Include ADwinPro_All.Inc
P2_Write_DAC8(module,array[],index)
```

Parameters

module	Specified module address (1…15).	LONG
array[]	Array with values (0…65535) to be output.	ARRAY LONG ~~FLOAT~~
index	Index of the first array element to be output.	LONG

See also

P2_DAC, P2_DAC4, P2_DAC4_Packed, P2_DAC8, P2_DAC8_Packed, P2_Start_DAC, P2_Write_DAC, P2_Write_DAC4, P2_Write_DAC4_Packed, P2_Write_DAC8_Packed, P2_Write_DAC32

Valid for

AOut-8/16 Rev. E, AOut-8/16-TiCo Rev. E

Example

Example

```
REM Simultaneous output of four different signals
REM on the outputs 1...8 of a D/A module.
REM The signals are sequentially stored into a DATA array
REM and may be transferred before program start from the PC.

#Include ADwinPro_All.Inc
Dim i As Long
Dim Data_1[8000] As Long

Init:
  i = 1

Event:
  REM Set output registers DAC1...DAC8
  P2_Write_DAC8(1,Data_1,(i-1)*8+i)
  P2_Start_DAC(1)          'Start output on all DAC
  Inc(i)
If (i>1000) Then i = 1
```

P2_Write_DAC8_Packed

P2_Write_DAC8_Packed writes 8 packed digital values from an array into the output registers of the DAC 1…8 of the specified module.

P2_Start_DAC starts the conversion of the digital values into the output voltages.

Syntax

```
#Include ADwinPro_All.Inc
P2_Write_DAC8_Packed(module,array[],index)
```

Parameters

module	Specified module address (1…15).	LONG
array[]	Array with values (0…65535) to be output as packed data: Each 32 bit array element holds 2 values of 16 bit.	**ARRAY** LONG ~~FLOAT~~
index	Index of the first array element to be output.	LONG

Notes

Every 4 array elements of 32 Bit hold 8 digital values of 16 Bit in the following order:

Array element	array[n+3]		array[n+2]		array[n+1]		array[n]	
Bit no.	31:16	15:00	31:16	15:00	31:16	15:00	31:16	15:00
Digital value for	DAC8	DAC7	DAC6	DAC5	DAC4	DAC3	DAC2	DAC1

See also

P2_DAC, P2_DAC4, P2_DAC4_Packed, P2_DAC8, P2_DAC8_Packed, P2_Start_DAC, P2_Write_DAC, P2_Write_DAC4, P2_Write_DAC4_Packed, P2_Write_DAC8, P2_Write_DAC32

Valid for

AOut-8/16 Rev. E, AOut-8/16-TiCo Rev. E

Example

Example

```
REM Simultaneous output of four different signals
REM on the outputs 1...8 of a D/A module.
REM The signals are sequentially stored into a DATA array
REM packed and can be transferred from the PC before program start
REM übergeben werden.
#Include ADwinPro_All.Inc
Dim i As Long
Dim Data_1[8000] As Long

Init:
  i = 1

Event:
  REM Set output registers DAC1...DAC8
  P2_Write_DAC8_Packed(1,Data_1,(i-1)*4+i)
  P2_Start_DAC(1) 'Start output on all DAC
  Inc(i)
  If (i>1000) Then i = 1
```

P2_Write_DAC32

P2_Write_DAC32 copies two 16 Bit values from a 32 Bit value into the output registers of a DAC pair of the specified module.

The conversion into an output voltage is done using **Start_DAC**.

Syntax

```
#Include ADwinPro_All.Inc
P2_Write_DAC32(module,dac_pair,value32)
```

Parameter

module	Specified module address (1…15).	LONG
dac_pair	Selection of DAC pair: 0: DAC 1 and 2 1: DAC 3 and 4 2: DAC 5 and 6 3: DAC 7 and 8	LONG
value32	Auszugebender Wert (0h…0FFFFFFFFh).	LONG

See also

The lower word (bits 0…15) of the digital value32 is written into the DAC with odd number, the upper word (bits 16…31) into the DAC with even number.

See also

P2_DAC, P2_DAC4, P2_DAC4_Packed, P2_DAC8, P2_DAC8_Packed, P2_Start_DAC, P2_Write_DAC, P2_Write_DAC4, P2_Write_DAC4_Packed, P2_Write_DAC8, P2_Write_DAC8_Packed

Valid for

AOut-4/16 Rev. E, AOut-4/16-TiCo Rev. E, AOut-8/16 Rev. E, AOut-8/16-TiCo Rev. E, MIO-4 Rev. E, MIO-4-ET1 Rev. E

Example

```
REM Simultaneous output of two different signals
REM on the outputs 3 and 4 of a D/A module.
REM The signals are filed in two DATA arrays and
REM can be transferred from the PC before program start
#Include ADwinPro_All.Inc
Dim i As Long                'Declaration
Dim Data_1[1000], Data_2[1000] As Long
Dim array[1000] As Long

Init:
  For i = 1 To 1000
    array[i] = Shift_Left(Data_2[i],16) + Data_1[i]
Next i
  i = 1

Event:
  P2_Write_DAC32(1,2,array[i])'Set output register DAC 3+4
  P2_Start_DAC(1) 'Start output on all DAC
  Inc(i)
  If (i>1000) Then i=1
```

P2_DAC1_DIO

P2_DAC1_DIO outputs an (analog) voltage on the DAC channel 1 and sets or clears the digital outputs of the specified module.

Syntax

```
#Include ADwinPro_All.Inc
P2_DAC1_DIO(module, value)
```

Parameters

module	Specified module address (1…255).	LONG
value	Combined output value for DAC and digital outputs (assignment of outputs see table). Bits 0…15: DAC output value (0…65535). Bits 16…31: Bit pattern. Each bit corresponds to a digital output (see table). Bit = 0: Set output to level low. Bit = 1: Set output to level high.	LONG

Bit no.	31	…	16	15…0
Dig. output	31	…	16	–

Notes

The output on DAC1 and on the digital outputs happens at the same time.

See also

P2_DAC, P2_Digout_Long

Valid for

AOut-1/16 Rev. E

Example

```
#Include ADwinPro_All.Inc
#Define module 5
Dim value As Long

Event:
  Rem DAC output 0V (32768) and set dig. outputs 16..19
  P2_DAC1_DIO(module, Join_DAC_DIO(32768, 01111b))

Function Join_DAC_DIO(dac_val, dio_val) As Long
  Join_DAC_DIO = dac_val And Shift_Left(dio_val, 16)
EndFunction
```

P2_DAC_Ramp_Write

P2_DAC_Ramp_Write defines the parameters for the output of the next voltage ramp and starts the DAC output.

Syntax

```
#Include ADwinPro_All.Inc
P2_DAC_Ramp_Write(module, dac_no, start_value,
    end_value, dio_start, dio_end, time, out_mode)
```

Parameters

module	Specified module address (1…255).	LONG
dac_no	Number (1) of the output.	LONG
start_value	Start value (0…65535) of the DAC ramp.	LONG
end_value	End value (0…65535) of the DAC ramp.	LONG
dio_start	Output value (10000h…0FFF0000h) for digital outputs at the begin of the ramp.	LONG
dio_end	Output value (10000h…0FFF0000h) for digital outputs at the end of the ramp.	LONG
time	Duration (20, 40, 60 … 65520) of the ramp in ns.	LONG
out_mode	Bit pattern to set the output mode: Bit 0: Output of DAC ramp. Bit 1: Output of digital value dio_start. Bit 2: Output of digital value dio_end. Bit = 0: Output disabled. Bit = 1: Output enabled.	LONG

Bit no. in dio_start, dio_end	31	30	…	17	16	0…15
Digital output	31	30	…	17	16	–

Notes

A ramp is defined by start_value, end_value and the duration time. After outputting the start value, the module increases (or decreases) the output voltage every 20ns by a constant amout until the end value is reached.
According to the data the real ramp duration may slightly deviate from the time value.

If start value and end value are identical, the output value at the DAC is held constantly for the duration given by time.

Combining several ramps consecutively you can output a freely defined voltage curve. While one ramp is being output the next ramp can be written into a buffer using **P2_DAC_Ramp_Write**. Only the data of one ramp can be buffered at a time.

First query with **P2_DAC_Ramp_Buffer_Free** if the buffer is free and only then write the next ramp data into the buffer using **P2_DAC_Ramp_Write**.

With **P2_DAC_Ramp_Status** you can check if a ramp is being output.

You can immediately stop a ramp output with **P2_DAC_Ramp_Stop**.

As an alternative to the output of ramps you can also output single voltage values or several voltage values via output fifo. A direct link bet-

ween different output methods (e.g. ramp directly followed by output fifo) is not supported.

See also

P2_DAC, P2_DAC_Ramp_Status, P2_DAC_Ramp_Buffer_Free, P2_DAC_Ramp_Stop, P2_DAC1_DIO, P2_Dig_Fifo_Mode

Valid for

AOut-1/16 Rev. E

Example

```
#Include ADwinPro_All.inc
#Define module 5
#Define dac_no 1
Dim array[5] As Long
Dim ramp_no, v_start, v_end As Long

Init:
  Rem stop possibly running ramp
  P2_DAC_Ramp_Stop(module, dac_no)
  Rem define values for 4 ramps
  array[1] = 0              '-2.0 V
  array[2] = 16384          '-1.0 V
  array[3] = 57344          '+1.5 V
  array[4] = 32768          ' 0.0 V
  array[5] = 49152          '+1.0 V
  ramp_no = 1

Event:
  Rem check if buffer has free space
  If (P2_DAC_Ramp_Buffer_Free(module,dac_no) >= 0) Then
    v_start = array[ramp_no]
    v_end = array[ramp_no+1]
    Rem write ramp, set ramp time 1.5µs, no digital values
    P2_DAC_Ramp_Write(module,dac_no,v_start,v_end,0,0,1500,001b)
    Inc ramp_no
    If (ramp_no > 4) Then ramp_no = 1
  EndIf
```

P2_DAC_Ramp_Status

P2_DAC_Ramp_Status returns if a voltage ramp is being output.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = P2_DAC_Ramp_Status(module, dac_no)
```

Parameters

module	Specified module address (1…255).	LONG
dac_no	Number (1) of the output.	LONG
ret_val	Status of ramp output: 0: no ramp output active. 1: ramp is being output.	LONG

Notes

You can immediately stop a ramp output with **P2_DAC_Ramp_Stop**.

As an alternative to the output of ramps you can also output single voltage values or several voltage values via output fifo. First query with **P2_DAC_Ramp_Status** if the ramp output is already finished before using a different output method.

See also

P2_DAC, P2_DAC_Ramp_Write, P2_DAC_Ramp_Buffer_Free, P2_DAC_Ramp_Stop, P2_DAC1_DIO, P2_Dig_Fifo_Mode

Valid for

AOut-1/16 Rev. E

Example

```
#Include ADwinPro_All.inc
#Define module 5
#Define dac_no 1
Dim value[4] As Long
Dim ramp_active As Long

Init:
  Processdelay = 6000       '6000 x 3.3 ns = 20µs

  Rem stop possibly running ramp
  P2_DAC_Ramp_Stop(module, dac_no)
  Rem write ramp, set ramp -2V..0V, time 10.5µs, no digital values
  P2_DAC_Ramp_Write(module,dac_no,0,32768,0,0,10500,001b)
  ramp_active = 1

  Rem Do settings for output fifo (start output later)
  value[1] = 00001C000h     'output 1V, dig. output 16
  value[2] = 500            ' with output time 5 µs (relative)
  value[3] = 00002FFFFh     'output 2V, dig. output 17
  value[4] = 700            ' with output time 7 µs (relative)
  P2_Dig_Fifo_Mode(module,3) 'Set FIFO as relative output
  P2_Digout_Fifo_Clear(module) 'clear FIFO
  P2_Digout_Fifo_Enable(module,11b) 'Enable output channels 0+1
  Rem write 2 value pairs into output FIFO
  P2_Digout_Fifo_Write(module,2,value,1)

Event:
  If (ramp_active = 1) Then
    Rem As long as ramp was running: check status
    ramp_active = P2_DAC_Ramp_Status(module, dac_no)
  Else
    Rem ramp has finished -> start output FIFO
    P2_Digout_Fifo_Start(Shift_Left(1,module-1))

    Rem write new value pairs into FIFO, if possible
    If (P2_Digout_Fifo_Empty(module) >= 2) Then
      P2_Digout_Fifo_Write(module,2,value,1)
    EndIf
  EndIf
```

P2_DAC_Ramp_Buffer_Free

P2_DAC_Ramp_Buffer_Free returns if the buffer for ramp output is free.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = P2_DAC_Ramp_Buffer_Free(module, dac_no)
```

Parameters

module	Specified module address (1…255).	LONG
dac_no	Number (1) of the output.	LONG
ret_val	Status of buffer for ramp output: >=0: buffer is empty. <0: buffer is full.	LONG

Notes

Combining several ramps consecutively you can output a freely defined voltage curve. While one ramp is being output the next ramp can be written into a buffer using **P2_DAC_Ramp_Write**. Only the data of one ramp can be buffered at a time.

First query with **P2_DAC_Ramp_Buffer_Free** if the buffer is free and only then write the next ramp data into the buffer using **P2_DAC_Ramp_Write**.

With **P2_DAC_Ramp_Status** you can check if a ramp is being output.

See also

P2_DAC, P2_DAC_Ramp_Write, P2_DAC_Ramp_Status, P2_DAC_Ramp_Stop, P2_DAC1_DIO, P2_Dig_Fifo_Mode

Valid for

AOut-1/16 Rev. E

Example

see P2_DAC_Ramp_Write

P2_DAC_Ramp_Stop

P2_DAC_Ramp_Stop stops a ramp output immediately.

Syntax

```
#Include ADwinPro_All.Inc
P2_DAC_Ramp_Stop(module, dac_no)
```

Parameters

module	Specified module address (1…255).	LONG
dac_no	Number (1) of the output.	LONG

Notes

P2_DAC_Ramp_Stop stops the ramp output as well as it clears the buffer for ramp output.

We recommend to initialize ramp outputs with **P2_DAC_Ramp_Stop** in the **Init:** section.

See also

P2_DAC, P2_DAC_Ramp_Write, P2_DAC_Ramp_Status, P2_DAC_Ramp_Buffer_Free, P2_DAC1_DIO, P2_Dig_Fifo_Mode

Valid for

AOut-1/16 Rev. E

Example

see P2_DAC_Ramp_Write

3.6 Pro II: Digital I/O Modules

This section describes instructions which apply to Pro II modules with digital inputs / outputs:

– P2_Comp_Init (page 149)
– P2_Comp_Filter_Init (page 151)
– P2_Comp_Set (page 152)
– P2_Dig_Fifo_Mode (page 153)
– P2_Dig_Latch (page 155)
– P2_Dig_Read_Latch (page 156)
– P2_Dig_Write_Latch (page 157)
– P2_Digin_Edge (page 158)
– P2_Digin_Fifo_Clear (page 159)
– P2_Digin_Fifo_Enable (page 160)
– P2_Digin_Fifo_Full (page 162)
– P2_Digin_Fifo_Read (page 163)
– P2_Digin_Fifo_Read_Fast (page 165)
– P2_Digin_Fifo_Read_Timer (page 167)
– P2_Digin_Filter_Init (page 168)
– P2_Digin_Long (page 169)
– P2_Digout (page 170)
– P2_Digout_Bits (page 171)
– P2_Digout_Fifo_Clear (page 173)
– P2_Digout_Fifo_Empty (page 174)
– P2_Digout_Fifo_Enable (page 175)
– P2_Digout_Fifo_Read_Timer (page 176)
– P2_Digout_Fifo_Start (page 177)
– P2_Digout_Fifo_Write (page 178)
– P2_Digout_Long (page 180)
– P2_Digout_Reset (page 181)
– P2_Digout_Set (page 182)
– P2_DigProg (page 183)
– P2_DigProg_Bits (page 184)
– P2_Digprog_Set_IO_Level (page 185)
– P2_Get_Digout_Long (page 186)

In the Instruction List sorted by Module Types (annex A.2) you will find which of the functions corresponds to the *ADwin-Pro II* modules.

P2_Comp_Init

P2_Comp_Init sets the operating mode for the comparators of a channel group on the specified module.

Syntax

```
#Include ADwinPro_All.inc
P2_Comp_Init(module, ch_group, mode)
```

Parameters

module	Specified module address (1…15).	LONG
ch_group	Channel group (1…4): 1: Channels 1, 2, 3, 4 2: Channels 5, 6, 7, 8 3: Channels 9, 10, 11, 12 4: Channels 13, 14, 15, 16	LONG
mode	Operating mode (0…6) of the comparators: 0: Neither hysteresis nor hold. 1: Hold 1µs. 2: Hold 10µs. 3: Hold 100µs. 4: Hysteresis ±17mV around the threshold. 5: Hysteresis ±55mV around the threshold. 6: Hysteresis ±100mV around the threshold.	LONG

Notes

The operating mode refers to 2 channels. The setting is still valid if the channel pair is used for free hysteresis.

See also

P2_Comp_Set, P2_Dig_Latch, P2_Dig_Read_Latch, P2_Digin_Edge, P2_Digin_Fifo_Enable, P2_Digin_Long

Valid for

Comp-16 Rev. E

Example

```
#Include ADwinPro_All.inc
#Define module 2
Dim Bit_Ch_1, Bit_Ch_1_2 As Long

Init:
  P2_Comp_Init(module,1,3) 'set channels 1,2 To hold 100µs
  Rem Set thresholds: channel group 1 To 0V, channel group 2 To 10V
  P2_Comp_Set(module,1,Volt2Digits(0.0))
  P2_Comp_Set(module,1,Volt2Digits(10.0))

Event:
  Rem check comparator bits
  Par_1 = P2_Digin_Long(module)
  Rem channel 1
  Bit_Ch_1 = Par_1 And 1b  'threshold of channel 1
  Rem free hysteresis determined by channels 1 + 2
  Bit_Ch_1_2 = Shift_Right(Par_1,16) And 1

  If (Bit_Ch_1 = 1) Then
    Rem channels 1 > 0 V
  Else ' (Bit_Ch_1 = 0)
    Rem channels 1 < 0 V
  EndIf

  If (Bit_Ch_1_2 = 1) Then
    Rem both channels 1, 2 > 10 V
  Else ' (Bit_Ch_1_2 = 0)
    Rem both channels 1, 2 < 0 V
  EndIf


Function Volt2Digits(volt) As Long
  Dim volt_min, volt_max As Float
  Dim digit_min, digit_max, value As Long
  volt_min = -1.0
  volt_max = 30.0
  digit_min = 31675
  digit_max = 65535

  Volt2Digits = (volt - volt_min) / (volt_max - volt_min)
    * (digit_max - digit_min) + digit_min
EndFunction
```

P2_Comp_Filter_Init

P2_Comp_Filter_Init sets the filter duration for all comparators on the specified module.

Syntax

```
#Include ADwinPro_All.inc
P2_Comp_Filter_Init(module, filter_value)
```

Parameters

module	Specified module address (1…15).	LONG
filter_value	Filter duration, given in units (1…65535) of 20 ns. The value 0 (zero) disables the filter.	LONG

Notes

The filter suppresses spikes of a signal. The number of spikes should be small compared to the pulse width of the signal. The filter duration should be somewhat longer than the expected width of spikes.

The filter settings apply to all channels. Each channel has its own filter. After power-up all filters are disabled.

Please note: The filter delays edges of the resulting signal by the set filter duration. If spikes occur, edges will delay slightly in addtion .

The filter can be combined with standard hysteresis and free hysteresis. The filter is not suitable for combination with the comparator mode "Hold".

See also

P2_Comp_Init, P2_Comp_Filter_Init, P2_Dig_Latch, P2_Dig_Read_Latch, P2_Digin_Edge, P2_Digin_Fifo_Enable, P2_Digin_Long

Valid for

Comp-16 Rev. E

Example

see P2_Comp_Init

P2_Comp_Set

P2_Comp_Set sets the comparator thresholds of a channel group on the specified module.

Syntax

```
#Include ADwinPro_All.inc
P2_Comp_Set(module, ch_group, value)
```

Parameters

module	Specified module address (1…15).	LONG
ch_group	Channel group(1…4): 1: Channels 1, 3, 5, 7. 2: Channels 2, 4, 6, 8. 3: Channels 9, 11, 13, 15. 4: Channels 10, 12, 14, 16.	LONG
value	Value (31676…65535) in digits to be output for threshold -1V…30V.	LONG

Notes

The conversion between digits and volts is as follows:

$$\text{digits} = \text{volts} \cdot \frac{65536}{60} + 32768$$

$$\text{volts} = (\text{digits} - 32768) \cdot \frac{60}{65536}$$

You read comparator bits with instructions for digital inputs as e.g. **P2_Digin_Long**.

With modes hold / standard hysteresis the bits are assigned to single channels as follows:

Bit no.	31…16	15	14	…	2	1	0
Channel no.	–	16	15	…	3	2	1

The comparator bits for the channel pairs of free hysteresis are the bits 16…23; the assignment is as follows:

Bit no.	31…24	23	22	…	17	16	15…0
Channel pair	–	15, 16	13, 14	…	3, 4	1, 2	–

See also

P2_Comp_Init, P2_Dig_Latch, P2_Dig_Read_Latch, P2_Digin_Edge, P2_Digin_Fifo_Enable, P2_Digin_Long

Valid for

Comp-16 Rev. E

Example

see P2_Comp_Init

P2_Dig_Fifo_Mode

P2_Dig_Fifo_Mode sets the FIFO operation mode on the specified module, input with edge detection or edge output.

Syntax

```
#Include ADwinPro_All.inc
P2_Dig_Fifo_Mode(module, mode)
```

Parameters

module	Specified module address (1…15).	LONG
mode	FIFO operation mode: 0: Input FIFO with edge detection. Default. 1: Output FIFO with edge output, time values with absolute reference, single output. 3: Output FIFO with edge output, time values with relative reference, single output. 5: TiCo2 only Output FIFO with edge output, time values with absolute reference, continuous output. 7: TiCo2 only Output FIFO with edge output, time values with relative reference, continuous output.	LONG

Notes

With DIO-32-TiCo, the output FIFO is available since revision E03.

Time stamps of an output FIFO set the time when an edge is output. The time stamp value can be defined with absolute or relative reference:

- Absolute value: The time stamp refers to the starting time 0 of the module counter (**P2_Digout_Fifo_Start**).
  Using this mode, the current counter value can be read with **P2_Digout_Fifo_Read_Timer**.
- Relative value: The time stamp is counted relative to the previous time stamp.

With processor TiCo2 the values in the output FIFO can be output once or continuously:

- single output: The list of value pairs in the FIFO is output only once, afterwards the output stops.
- continuous output: The list of value pairs in the FIFO is output and repeated until the output is stopped with **P2_Digout_Fifo_Clear**.

With single output (modes 1 and 3), the list of value pairs can be filled up–as long as there are any value pairs in the FIFO.

With module AOut-1/16, only modes 1 and 3 can be set.

See also

P2_Digin_Fifo_Enable, P2_Digout_Fifo_Read_Timer, P2_Digout_Fifo_Start, P2_Digout_Fifo_Write, P2_Digprog_Set_IO_Level

Valid for

AOut-1/16 Rev. E, DIO-32-TiCo Rev. E, DIO-32-TiCo2 Rev. E, MIO-D12 Rev. E, SPI-2-D Rev. E, SPI-2-T Rev. E

Example

```
#Include ADwinPro_All.inc
#Define module 2
Dim value[4] As Long

Init:
  Processdelay = 6000       '6000 x 3.3 ns = 20µs
  value[1] = 01b            'output value n
  value[2] = 5000           ' with output time 50 µs (relative)
  value[3] = 10b            'output value n+1
  value[4] = 7000           ' with output time 70 µs (relative)
  REM With AOUT-1/16: delete line using P2_DigProg
  P2_DigProg(module,0Fh)    'set all channels as output
  P2_Dig_Fifo_Mode(module,3) 'Set FIFO as relative output
  P2_Digout_Fifo_Clear(module) 'clear FIFO
  P2_Digout_Fifo_Enable(module,11b) 'Enable output channels 0+1
  Rem write 2 value pairs into output FIFO and start output
  P2_Digout_Fifo_Write(module,2,value,1)
  P2_Digout_Fifo_Start(Shift_Left(1,module-1))

  Rem DIO-32-TiCo2 only: set voltage level
  Rem P2_DigProg_Set_IO_Level(module, 0, 160)

Event:
  Rem write new value pairs into FIFO, if possible
  If (P2_Digout_Fifo_Empty(module) > 2) Then
    P2_Digout_Fifo_Write(module,2,value,1)
  EndIf
```

P2_Dig_Latch

P2_Dig_Latch transfers digital information from the inputs to the input latches and from the output latches to the outputs on the specified module.

Syntax

```
#Include ADwinPro_All.Inc
P2_Dig_Latch(module)
```

Parameters

module	Specified module address (1…15).	LONG

Notes

With digital inputs the instructions reads the input signals into the input latches. Read the values with **P2_Dig_Read_Latch**.
With digital outputs the instruction passes the values of the output latches to the outputs. Write values into the latch register with **P2_Dig_Write_Latch**.

Latching can be started synchronously to actions on other modules with **P2_Sync_All**.

See also

P2_Dig_Read_Latch, P2_Dig_Write_Latch, P2_DigProg, P2_Digin_Long, P2_Digout_Bits, P2_Digout, P2_Digout_Long, P2_Get_Digout_Long, P2_Sync_All

Valid for

AOut-1/16 Rev. E, DIO-32 Rev. E, DIO-32-TiCo Rev. E, DIO-32-TiCo2 Rev. E, OPT-16 Rev. E, OPT-32 Rev. E, REL-16 Rev. E, SPI-2-D Rev. E, SPI-2-T Rev. E, TRA-16 Rev. E

Example

```
#Include ADwinPro_All.Inc

Init:
  REM With AOUT-1/16: delete line using P2_DigProg
  REM Set channels 0…15 as outputs, 16…31 as inputs
  P2_DigProg(1,0011h)
  P2_Dig_Write_Latch(1,0)  'Set all output bits to 0

Event:
  P2_Dig_Latch(1)          'latch inputs, output content of
                           'output latches
  Rem further program
  Par_1 = P2_Dig_Read_Latch(1) 'read input bits and …
  P2_Dig_Write_Latch(1,Par_1)'output in next event cycle
```

P2_Dig_Read_Latch

P2_Dig_Read_Latch returns the bits from the latch register for the digital inputs of the specified module.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = P2_Dig_Read_Latch(module)
```

Parameters

module	Specified module address (1…15).	LONG
ret_val	Bit pattern. Each bit corresponds to a digital input (see table).	LONG

Module AOut-1/16:

Bit no.	31…16	15	…	0
Input	–	15	…	0

Other modules:

Bit no.	31	30	29	…	2	1	0
Input	31	30	29	…	2	1	0

Notes

It is recommended to first program the specified channels as inputs using **P2_DigProg**.

The current status of the digital inputs can be transferred to the latch register with the following instructions:

- **P2_Dig_Latch**
- **P2_Sync_All** (when activated)

See also

P2_Dig_Latch, P2_Dig_Write_Latch, P2_DigProg, P2_Digin_Long, P2_Sync_All, P2_Sync_Mode

Valid for

AOut-1/16 Rev. E, DIO-32 Rev. E, DIO-32-TiCo Rev. E, DIO-32-TiCo2 Rev. E, OPT-16 Rev. E, OPT-32 Rev. E, SPI-2-D Rev. E, SPI-2-T Rev. E

Example

```
#Include ADwinPro_All.Inc
Dim value As Long

Init:
  REM With AOUT-1/16: delete lines using P2_DigProg
  REM Set DIO15:00 of modules 1+2 as inputs
  P2_DigProg(1,0000b)
  P2_DigProg(2,0000b)

Event:
  REM synchronously transfer the logic levels at the digital
  REM inputs of both modules to the latch register
  P2_Sync_All(11b)
  Par_1 = P2_Dig_Read_Latch(1)'Read latch register of module 1
  Par_2 = P2_Dig_Read_Latch(2)'Read latch register of module 2
```

P2_Dig_Write_Latch

P2_Dig_Write_Latch writes a 32 bit value into the latch register for the digital outputs on the specified module.

Syntax

```
#Include ADwinPro_All.Inc
P2_Dig_Write_Latch(module,pattern)
```

Parameters

module	Specified module address (1…15).	LONG
pattern	Bit pattern. Each bit corresponds to a digital output (see table).	LONG

Module AOut-1/16:

Bit no.	31…16	15	…	0
Input	–	15	…	0

Other modules:

Bit no.	31	30	29	…	2	1	0
Input	31	30	29	…	2	1	0

Notes

The specified channels must first be programmed as outputs using **P2_DigProg**.

You may set the value of the latch register for digital outputs with **P2_Digout**.

With module version TRA-16-G Rev. E, high level switches to ground, not to V_{CC}.

See also

P2_Dig_Latch, P2_Digout, P2_DigProg, P2_Get_Digout_Long

Valid for

AOut-1/16 Rev. E, DIO-32 Rev. E, DIO-32-TiCo Rev. E, DIO-32-TiCo2 Rev. E, REL-16 Rev. E, SPI-2-D Rev. E, SPI-2-T Rev. E, TRA-16 Rev. E

Example

```
#Include ADwinPro_All.Inc

Init:
  REM With AOUT-1/16: delete line using P2_DigProg
  P2_DigProg(1,1111b)       'Set DIO31:00 as outputs

Event:
  P2_Dig_Latch(1)           'Output values from the latch
                            'registers
  P2_Dig_Write_Latch(1,Par_1)'write long word into the output
                            'latch
```

P2_Digin_Edge

P2_Digin_Edge returns whether a positive or negative edge has occurred on digital inputs of the specified module.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = P2_Digin_Edge(module,edge)
```

Parameters

module	Specified module address (1…15).	LONG
edge	Kind of detected edge: 1: Detect positive edge. 0: Detect negative edge.	LONG
ret_val	Bit pattern where each bits represent an edge occurred at an input. The mapping of bits to inputs is shown below. Bit = 1: An edge has occurred. Bit = 0: No edge occurred.	LONG

Bit no.	31	30	29	…	2	1	0
Input	31	30	29	…	2	1	0

Notes

A set bit in ret_val means, that a selected edge has been occurred at least once at the digital input since the previous query. Bit for output channels always return zero.

A query with **P2_Digin_Edge** resets all bits to zero.

With module Pro II-MIO-D12 Rev. E, only bits 0…11 (refers to OPT0 … OPT11) can be used.

See also

P2_Digin_Fifo_Clear, P2_Digin_Fifo_Enable, P2_Digin_Fifo_Full, P2_Digin_Fifo_Read, P2_Digin_Fifo_Read_Timer

Valid for

AOut-1/16 Rev. E, DIO-32 Rev. E, DIO-32-TiCo Rev. E, DIO-32-TiCo2 Rev. E, MIO-D12 Rev. E, OPT-16 Rev. E, OPT-32 Rev. E, SPI-2-D Rev. E, SPI-2-T Rev. E

Example

```
#Include ADwinPro_All.Inc

Init:
  P2_DigProg(1,1100b)       'channels 0:15 as inputs

Event:
  REM check positive and negative edges, mask out outputs
  Par_1 = P2_Digin_Edge(1,1) And 0Fh
  Par_2 = P2_Digin_Edge(1,0) And 0Fh

  REM output edge changes
  If (Par_1 + Par_2 > 0)Then
    P2_Digout_Bits(1,Shift_Left(Par_1,16),Shift_Left(Par_2,16))
  EndIf
```

P2_Digin_Fifo_Clear

P2_Digin_Fifo_Clear clears the edge detection unit on the specified module.

Syntax

```
#Include ADwinPro_All.Inc
P2_Digin_Fifo_Clear(module)
```

Parameters

module	Specified module address (1…15).	LONG

Notes

- / -

See also

P2_Digin_Fifo_Enable, P2_Digin_Fifo_Full, P2_Digin_Fifo_Read, P2_Digin_Fifo_Read_Timer, P2_Digin_Edge

Valid for

DIO-32 Rev. E, DIO-32-TiCo Rev. E, DIO-32-TiCo2 Rev. E, MIO-D12 Rev. E, OPT-16 Rev. E, OPT-32 Rev. E, SPI-2-D Rev. E, SPI-2-T Rev. E

Example

```
#Include ADwinPro_All.Inc

Dim Data_1[10000], Data_2[10000] As Long
Dim num, index As Long

Init:
  P2_DigProg(1,1100b)       'channels 0:15 as inputs
  P2_Digin_Fifo_Enable(1,0)'output edge edge detection off
  P2_Digin_Fifo_Clear(1)    'clear FIFO
  P2_Digin_Fifo_Enable(1,10011b)'edge detection channels 1,2,5
  index = 1


Event:
  num = P2_Digin_Fifo_Full(1)'number of value pairs
  If (num>50) Then
    If (index+num>10000) Then index = 1
    Rem read value pairs
    P2_Digin_Fifo_Read(1, num, Data_1, Data_2, index)
    index=index + num
  EndIf
```

P2_Digin_Fifo_Enable

P2_Digin_Fifo_Enable determines, which input channels of the specified module the edge detection unit will monitor.

Syntax

```
#Include ADwinPro_All.Inc
P2_Digin_Fifo_Enable(module,channels)
```

Parameters

module	Specified module address (1…15).	LONG
channels	Bit pattern which determines the input channels to be monitored.	LONG

Bit no.	31	30	29	…	2	1	0
Input	31	30	29	…	2	1	0

Notes

Only input channels can be monitored. The channels are programmed as inputs or outputs with **P2_DigProg**; exceptions are OPT channels.

With module Pro II-MIO-D12 Rev. E, only bits 0…11 (refers to OPT0 … OPT11) can be used.

The FIFO should be set to edge detection mode with **P2_Dig_Fifo_Mode**.

The edge detection unit checks with each counter tick (each 10 ns / 5 ns), if an edge has occurred at the selected input channels or if a level has been changed. If an edge has occurred, a pair of values is copied into an internal FIFO array:

- Value 1 contains the level status of all channels as bit pattern. With outputs the status is undefined.
- Value 2 contains a time stamp, the current timer value.
  DIO-32-TiCo2: clock rate 200 MHz
  all other modules: clock rate 100 MHz

The FIFO array may contain 511 or 2047 value pairs (level status and time stamp) in maximum. If and as long as the FIFO array is filled completely, any additional value pair cannot be saved and will thus be lost.

See also

P2_Dig_Fifo_Mode, P2_Digin_Fifo_Clear, P2_Digin_Fifo_Full, P2_Digin_Fifo_Read, P2_Digin_Fifo_Read_Timer, P2_Digin_Edge, P2_DigProg

Valid for

DIO-32 Rev. E, DIO-32-TiCo Rev. E, DIO-32-TiCo2 Rev. E, MIO-D12 Rev. E, OPT-16 Rev. E, OPT-32 Rev. E, SPI-2-D Rev. E, SPI-2-T Rev. E

Example

```
#Include ADwinPro_All.Inc

Dim Data_1[10000], Data_2[10000] As Long
Dim num, index As Long

Init:
  P2_DigProg(1,1100b)       'channels 0:15 as inputs
  P2_Digin_Fifo_Enable(1,0)'output edge edge detection off
  P2_Digin_Fifo_Clear(1)    'clear FIFO
  P2_Digin_Fifo_Enable(1,10011b)'edge detection channels 1,2,5
  index = 1


Event:
  num = P2_Digin_Fifo_Full(1)'number of value pairs
  If (num>50) Then
    Rem read value pairs
    P2_Digin_Fifo_Read(1, num, Data_1, Data_2, index)
    index=index + num
    If (index>10000) Then index = 1
  EndIf
```

P2_Digin_Fifo_Full

P2_Digin_Fifo_Full returns the number of saved value pairs in the FIFO of the edge detection unit.

Syntax

```
#Include ADwinPro_All.Inc
ret_value = P2_Digin_Fifo_Full(module)
```

Parameters

module	Specified module address (1…15).	LONG
ret_value	Number of saved value pairs in the FIFO: DIO-32-TiCo2: 0…2047 all other modules: 0…511	LONG

Notes

The FIFO array may contain 511 or 2047 value pairs (level status and time stamp) in maximum. If and as long as the FIFO array is filled completely, any additional value pair cannot be saved and will thus be lost.

See also

P2_Digin_Fifo_Clear, P2_Digin_Fifo_Enable, P2_Digin_Fifo_Read, P2_Digin_Fifo_Read_Timer, P2_Digin_Edge

Valid for

DIO-32 Rev. E, DIO-32-TiCo Rev. E, DIO-32-TiCo2 Rev. E, MIO-D12 Rev. E, OPT-16 Rev. E, OPT-32 Rev. E, SPI-2-D Rev. E, SPI-2-T Rev. E

Example

```
#Include ADwinPro_All.Inc

Dim Data_1[10000], Data_2[10000] As Long
Dim num, index As Long

Init:
  P2_DigProg(1,1100b)       'channels 0:15 as inputs
  P2_Digin_Fifo_Enable(1,0)'output edge edge detection off
  P2_Digin_Fifo_Clear(1)    'clear FIFO
  P2_Digin_Fifo_Enable(1,10011b)'edge detection channels 1,2,5
  index = 1


Event:
  num = P2_Digin_Fifo_Full(1)'number of value pairs
  If (num>50) Then
    Rem read value pairs
    P2_Digin_Fifo_Read(1, num, Data_1, Data_2, index)
    index=index + num
    If (index>10000) Then index = 1
  EndIf
```

P2_Digin_Fifo_Read

P2_Digin_Fifo_Read reads the value pairs from the FIFO of the edge detection unit and writes them into 2 arrays.

Syntax

```
#Include ADwinPro_All.Inc
P2_Digin_Fifo_Read(module, count, value[],
    timestamp[], start_index)
```

Parameters

module	Specified module address (1…15).	LONG
count	Number of value pairs to be read: DIO-32-TiCo2: 0…2047 all other modules: 0…511	LONG
value[]	Array where the level status bit patterns are written. Each level status bit corresponds to a digital input (see table below).	LONG ARRAY
timestamp[]	Array where time stamps are written.	LONG ARRAY
start_index	Start index for both arrays value[] and timestamp[], where the first value are written.	LONG

Bit no.	31	30	29	…	2	1	0
Input	31	30	29	…	2	1	0

Notes

With module Pro II-MIO-D12 Rev. E, only bits 0…11 (refers to OPT0 … OPT11) can be used.

No more value pairs may be read than are saved in the FIFO. Thus, before reading there must be a check with **P2_Digin_Fifo_Full**, how much value pairs are saved in the FIFO.

The arrays must be dimensioned with sufficient size, so that all value pairs may be written into the arrays.

The time difference between 2 level status patterns is the difference of the appropriate time stamps, measured in timer ticks. The length of a timer tick depends on the module type.

$$\Delta t = (stamp_1 - stamp_2) \cdot \text{tick length}$$

Module type	Clock rate	Tick length
DIO-32-TiCo2	200 MHz	5 ns
all other modules	100 MHz	10 ns

See also

P2_Digin_Fifo_Clear, P2_Digin_Fifo_Enable, P2_Digin_Fifo_Full, P2_Digin_Fifo_Read_Timer, P2_Digin_Edge

Valid for

DIO-32 Rev. E, DIO-32-TiCo Rev. E, DIO-32-TiCo2 Rev. E, MIO-D12 Rev. E, OPT-16 Rev. E, OPT-32 Rev. E, SPI-2-D Rev. E, SPI-2-T Rev. E

Example

```
#Include ADwinPro_All.Inc

Dim Data_1[10000], Data_2[10000] As Long
Dim num, index As Long

Init:
  P2_DigProg(1,1100b)       'channels 0:15 as inputs
  P2_Digin_Fifo_Enable(1,0)'output edge edge detection off
  P2_Digin_Fifo_Clear(1)    'clear FIFO
  P2_Digin_Fifo_Enable(1,10011b)'edge detection channels 1,2,5
  index = 1


Event:
  num = P2_Digin_Fifo_Full(1)'number of value pairs
  If (num>50) Then
    Rem read value pairs
    P2_Digin_Fifo_Read(1, num, Data_1, Data_2, index)
    index=index + num
    If (index>10000) Then index = 1
  EndIf
```

P2_Digin_Fifo_Read_Fast

P2_Digin_Fifo_Read_Fast reads the value pairs from the FIFO of the edge detection unit and writes them into a single array.

Syntax

```
#Include ADwinPro_All.Inc
P2_Digin_Fifo_Read_Fast(module, count, valuepairs[],
    start_index)
```

Parameters

module	Specified module address (1…15).	LONG
count	Number of value pairs to be read: DIO-32-TiCo2: 0…2047 all other modules: 0…511	LONG
valuepairs[]	Array where the value pairs are written to, alternately a level status bit pattern and a time stamp. Each level status bit corresponds to a digital input (see table below).	LONG ARRAY
start_index	Start index for both arrays valuepairs[] and timestamp[], where the first value are written.	LONG

Bit no.	31	30	29	…	2	1	0
Input	31	30	29	…	2	1	0

Notes

With module Pro II-MIO-D12 Rev. E, only bits 0…11 (refers to OPT0 … OPT11) can be used.

No more value pairs may be read than are saved in the FIFO. Thus, before reading there must be a check with **P2_Digin_Fifo_Full**, how much value pairs are saved in the FIFO.

The arrays must be dimensioned with sufficient size, so that all value pairs may be written into the arrays.

The array valuepairs[] contains value pairs of level status and appropriate time stamp:

- One array element holds the level staus of input channels as bit pattern.
- The next array element hold a time stamp (absolute or relative, see **P2_Dig_Fifo_Mode**).

The time difference between 2 level status patterns is the difference of the appropriate time stamps, measured in timer ticks. The length of a timer tick depends on the module type.

$$\Delta t = (stamp_1 - stamp_2) \cdot tick\ length$$

Module type	Clock rate	Tick length
DIO-32-TiCo2	200 MHz	5 ns
all other modules	100 MHz	10 ns

See also

P2_Digin_Fifo_Clear, P2_Digin_Fifo_Enable, P2_Digin_Fifo_Full, P2_Digin_Fifo_Read_Timer, P2_Digin_Edge

Valid for

DIO-32 Rev. E, DIO-32-TiCo Rev. E, DIO-32-TiCo2 Rev. E, MIO-D12 Rev. E, OPT-16 Rev. E, OPT-32 Rev. E, SPI-2-D Rev. E, SPI-2-T Rev. E

Example

```
#Include ADwinPro_All.inc

Dim Data_1[20000] As Long
Dim num, index As Long

Init:
  P2_DigProg(1,1100b)       'channels 0:15 as inputs
  P2_Digin_Fifo_Enable(1,0)'output edge edge detection off
  P2_Digin_Fifo_Clear(1)    'clear FIFO
  P2_Digin_Fifo_Enable(1,10011b)'edge detection channels 1,2,5
  index = 1


Event:
  num = P2_Digin_Fifo_Full(1)'number of value pairs
  If (num > 50) Then
    Rem read value pairs
    P2_Digin_Fifo_Read_Fast(1, num, Data_1, index)
    index = index + num
    If (index > 10000) Then index = 1
  EndIf
```

P2_Digin_Fifo_Read_Timer

P2_Digin_Fifo_Read_Timer returns the current value of the timer on the specified module.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = P2_Digin_Fifo_Read_Timer(module)
```

Parameters

module	Specified module address (1…15).	LONG
ret_val	Current value (-2^{31}-1 … 2^{31}) of the timer.	LONG

Notes

The module timer is used to provide time stamps for the edge detection unit, see **P2_Digin_Fifo_Enable**.

The timer value is regularly increased by 1, so the timer will reach the original timer value after 2^{32} ticks. For comparison of time this "overflow" must be considered, so the timer value must be queried regularly in the program before a overflow has happened. The counter runs with different clock rates according to the module type:

Module	Clock rate	Tick length	overflow after
DIO-32-TiCo2	200 MHz	5 ns	21 s = 5 ns × 2^{32}
all other modules	100 MHz	10 ns	43 s = 10 ns × 2^{32}

See also

P2_Digin_Fifo_Enable, P2_Digin_Fifo_Read

Valid for

DIO-32 Rev. E, DIO-32-TiCo Rev. E, DIO-32-TiCo2 Rev. E, MIO-D12 Rev. E, OPT-16 Rev. E, OPT-32 Rev. E, SPI-2-D Rev. E, SPI-2-T Rev. E

Example

```
#Include ADwinPro_All.inc
#Define ticks 10000

Dim Data_1[10000], Data_2[10000] As Long
Dim num, index As Long

Init:
  P2_DigProg(1,1100b)       'channels 0:15 as inputs
  P2_Digin_Fifo_Enable(1,0)'output edge edge detection off
  P2_Digin_Fifo_Clear(1)    'clear FIFO
  P2_Digin_Fifo_Enable(1,10011b)'edge detection channels 1,2,5
  index = 1

Event:
  num = P2_Digin_Fifo_Full(1)'number of value pairs
  If (num > 50) Then
    Rem read value pairs
    P2_Digin_Fifo_Read(1, num, Data_1, Data_2, index)
    index = index + num
    If (index > 10000) Then index = 1
  EndIf
```

P2_Digin_Filter_Init

P2_Digin_Filter_Init sets the filter duration for all digital inputs on the specified module.

Syntax

```
#Include ADwinPro_All.inc
P2_Digin_Filter_Init(module, filter_value)
```

Parameters

module	Specified module address (1…15).	LONG
filter_value	Filter duration, given in units (1…65535) of 20ns. The value 0 (zero) disables the filter.	LONG

Notes

The filter suppresses spikes of a signal. The number of spikes should be small compared to the pulse width of the signal. The filter duration should be somewhat longer than the expected width of spikes.

The filter settings apply to all channels. Each channel has its own filter. After power-up all filters are disabled.

Please note: The filter delays edges of the resulting signal by the set filter duration. If spikes occur, edges will delay slightly in addition .

See also

P2_Comp_Init, P2_Comp_Filter_Init, P2_Dig_Latch, P2_Dig_Read_Latch, P2_Digin_Edge, P2_Digin_Fifo_Enable, P2_Digin_Long

Valid for

DIO-32-TiCo Rev. E

Example

```
#Include ADwinPro_All.Inc
#Define module 2

Init:
  P2_DigProg(module,0)      'Set DIO31:00 as inputs
  P2_Digin_Filter_Init(module, 5)'set spike filter to 100ns

Event:
  Par_1 = P2_Digin_Long(module)'Read all inputs
```

P2_Digin_Long

P2_Digin_Long returns the status of the inputs (bits 31…00) of the specified module as bit pattern.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = P2_Digin_Long(module)
```

Parameters

module	Specified module address (1…15).	LONG
ret_val	Bit pattern. Each bit (31…0) corresponds to the input status of a digital input (see table). Bit = 0: Input has low level. Bit = 1: Input has high level.	LONG

Module AOut-1/16:

Bit no.	31…16	15	…	0
Input	–	15	…	0

Other modules:

Bit no.	31	30	29	…	2	1	0
Input	31	30	29	…	2	1	0

Notes

It is recommended to first program the specified channels as inputs using **P2_DigProg**. This is not required for module AOut-1/16.

See also

P2_Dig_Latch, P2_DigProg, P2_Digout_Long

Valid for

AOut-1/16 Rev. E, DIO-32 Rev. E, DIO-32-TiCo Rev. E, DIO-32-TiCo2 Rev. E, OPT-16 Rev. E, OPT-32 Rev. E, SPI-2-D Rev. E, SPI-2-T Rev. E

Example

```
#Include ADwinPro_All.Inc

Init:
  REM With AOUT-1/16: delete line using P2_DigProg
  P2_DigProg(1,0)           'Set DIO31:00 as inputs

Event:
  Par_1 = P2_Digin_Long(1) 'Read all inputs
```

P2_Digout

P2_Digout sets a single output of the specified module to the level "high" or "low". All other outputs remain unchanged.

Syntax

```
#Include ADwinPro_All.Inc
P2_Digout(module,output,value)
```

Parameters

module	Specified module address (1…15).	LONG
output	Number of the output to be set (0…31).	LONG
value	New status of the selected output: 0: Low level. 1: High level.	LONG

Notes

The specified channels must be first programmed as outputs using **P2_DigProg**. This is not required for module AOut-1/16.

You can set or clear any output without changing the status of the remaining outputs.

With module version TRA-16-G Rev. E, high level switches to ground, not to V_{CC}.

See also

P2_Digout_Long, P2_Digout_Bits, P2_DigProg

Valid for

AOut-1/16 Rev. E, DIO-32 Rev. E, DIO-32-TiCo Rev. E, DIO-32-TiCo2 Rev. E, REL-16 Rev. E, SPI-2-D Rev. E, SPI-2-T Rev. E, TRA-16 Rev. E

Example

```
#Include ADwinPro_All.Inc

Init:
  REM With AOUT-1/16: delete line using P2_DigProg
  REM Set channels 0…15 as inputs, 16…31 as outputs
  P2_DigProg(1,01100b)

Event:
  If (P2_Digin_Long(1) And 8000h = 1) Then
    P2_Digout(1,31,0)       'channel 15 is set: clear bit 31
  Else
    P2_Digout(1,31,1)       'channel 15 is cleared: set bit 31
  EndIf
```

P2_Digout_Bits

P2_Digout_Bits sets the specified outputs of the specified module to the levels "high" or "low".

Syntax

```
#Include ADwinPro_All.Inc
P2_Digout_Bits(module,set,clear)
```

Parameters

module	Specified module address (1…15).	LONG
set	Bit pattern that sets specified digital outputs to the level "high". Bit = 0: Output remains unchanged. Bit = 1: Set output to level "high".	LONG
clear	Bit pattern that sets specified digital outputs to the level "low". Bit = 0: Output remains unchanged. Bit = 1: Set output to level "low".	LONG

Module AOut-1/16:

Bit no.	31	…	16	15…0
Output	31	…	16	–

Other modules:

Bit no.	31	30	…	2	1	0
Output	31	30	…	2	1	0

Notes

The specified channels must be first programmed as outputs using **P2_DigProg**. This is not required for module AOut-1/16.

You can set or clear any required outputs without changing the status of the remaining outputs.

For clarity reasons please note that the bits in `set` must not be set in the bit pattern `clear` at the same time, and vice versa.

With module version TRA-16-G Rev. E, high level switches to ground, not to V_{CC}.

See also

P2_Dig_Latch, P2_Digout_Reset, P2_Digout_Set, P2_DigProg, P2_Digout_Long

Valid for

AOut-1/16 Rev. E, DIO-32 Rev. E, DIO-32-TiCo Rev. E, DIO-32-TiCo2 Rev. E, REL-16 Rev. E, SPI-2-D Rev. E, SPI-2-T Rev. E, TRA-16 Rev. E

Example

```
#Include ADwinPro_All.Inc

Init:
  REM With AOUT-1/16: delete line using P2_DigProg
  REM Set channels 0…31 as outputs
  P2_DigProg(1,01111b)

Event:
  If (Par_1 = 1) Then       'Get condition
    REM lower word: Set byte MSBs, clear all other bits
    P2_Digout_Bits(1,8080h,7F7Fh)
  Else
    REM lower word: set odd-numbered bits, clear even-numbered
    P2_Digout_Bits(1,5555h,0AAAAh)
  EndIf
```

P2_Digout_Fifo_Clear

P2_Digout_Fifo_Clear stops the edge output and clears the edge output FIFO on the specified module.

Syntax

```
#Include ADwinPro_All.inc
P2_Digout_Fifo_Clear(module)
```

Parameters

module	Specified module address (1…15).	LONG

Notes

On the module DIO-32-TiCo, the output is available since revision E03.

Before first use, the FIFO must be cleared. Then, the FIFO can be filled with data using **P2_Digout_Fifo_Write**.

If the edge output has been stopped with **P2_Digout_Fifo_Clear**, it can only be started with **P2_Digout_Fifo_Start** again.

See also

P2_Digout_Fifo_Enable, P2_Dig_Fifo_Mode, P2_Digout_Fifo_Start, P2_Digout_Fifo_Write, P2_DigProg

Valid for

AOut-1/16 Rev. E, DIO-32-TiCo Rev. E, DIO-32-TiCo2 Rev. E, MIO-D12 Rev. E, SPI-2-D Rev. E, SPI-2-T Rev. E

Example

see P2_Dig_Fifo_Mode

P2_Digout_Fifo_Empty

P2_Digout_Fifo_Empty returns the number of free value pairs in the edge output FIFO.

Syntax

```
#Include ADwinPro_All.inc
ret_value = P2_Digout_Fifo_Empty(module)
```

Parameters

module	Specified module address (1…15).	LONG
ret_value	Number of free value pairs in the FIFO: DIO-32-TiCo2: 0…2047 all other modules: 0…511	LONG

Notes

The FIFO array can hold up to 511 or 2047 value pairs (level status and time stamp), according to the module type.

See also

P2_Dig_Fifo_Mode, P2_Digout_Fifo_Read_Timer, P2_Digout_Fifo_Start, P2_Digout_Fifo_Write

Valid for

AOut-1/16 Rev. E, DIO-32-TiCo Rev. E, DIO-32-TiCo2 Rev. E, MIO-D12 Rev. E, SPI-2-D Rev. E, SPI-2-T Rev. E

Example

see P2_Dig_Fifo_Mode

P2_Digout_Fifo_Enable

P2_Digout_Fifo_Enable sets the output channels of the specified module where edges are output.

Syntax

```
#Include ADwinPro_All.inc
P2_Digout_Fifo_Enable(module,channels)
```

Parameters

module	Specified module address (1…15).	LONG
channels	Bitt pattern to set the selected output channels.	LONG

Module AOut-1/16:

Bit no.	31	30	29	…	16	15…0
Output	–	–	29	…	16	–

Other modules:

Bit no.	31	30	…	2	1	0
Output	31	30	…	2	1	0

Notes

On the module DIO-32-TiCo, the output is available since revision E03.

Edges can only be output to output channels. The specified channels must be first programmed as outputs using **P2_DigProg**. This is not required for module AOut-1/16.

The FIFO must be set as output FIFO with **P2_Dig_Fifo_Mode**.

P2_Digout_Fifo_Enable selects channels for edge output via output FIFO. The levels of the other output channels–and only of these–can be set with instructions like **P2_Digout_Long**.

The levels and points of time of egde output are set with **P2_Digout_Fifo_Write**.

On a module with TiCo2 processor, the previously set voltage levels are active after power-up. For safety reasons you should set the channel levels again in the **Init:** section with **P2_DigProg_Set_IO_Level** before starting the output.

See also

P2_Digout_Fifo_Clear, P2_Dig_Fifo_Mode, P2_Digout_Fifo_Start, P2_Digout_Fifo_Write, P2_DigProg, P2_Digprog_Set_IO_Level, P2_Digout_Long

Valid for

AOut-1/16 Rev. E, DIO-32-TiCo Rev. E, DIO-32-TiCo2 Rev. E, MIO-D12 Rev. E, SPI-2-D Rev. E, SPI-2-T Rev. E

Example

see P2_Dig_Fifo_Mode

P2_Digout_Fifo_Read_Timer

P2_Digout_Fifo_Read_Timer returns the current counter value on the specified module.

Syntax

```
#Include ADwinPro_All.inc
ret_val = P2_Digout_Fifo_Read_Timer(module)
```

Parameters

module	Specified module address (1…15).	LONG
ret_val	Current value (-2^{31}-1 … 2^{31}) of the counter.	LONG

Notes

The module counter is used for exact edge output timing at predefined points of time, see **P2_Digout_Fifo_Write**.

The counter value can only be used in the FIFO operation mode with absolute time values–and on modules with TiCo2 only with single output–i.e. parameter mode = 1 in **P2_Dig_Fifo_Mode**

The timer value is regularly increased by 1, so the timer will reach the original timer value after 2^{32} ticks. A "missed" edge output is done only after the timer has happened this "overflow". The counter runs with different clock rates according to the module type:

Module	Clock rate	Tick length	overflow after
DIO-32-TiCo since Rev. E03 MIO-D12, AOut-1/16	100 MHz	10 ns	43 s = 10 ns × 2^{32}
DIO-32-TiCo2	200 MHz	5 ns	21 s = 5 ns × 2^{32}

The counter is set to zero with **P2_Digout_Fifo_Clear**.

See also

P2_Dig_Fifo_Mode, P2_Digout_Fifo_Empty, P2_Digout_Fifo_Start, P2_Digout_Fifo_Write, P2_DigProg

Valid for

AOut-1/16 Rev. E, DIO-32-TiCo Rev. E, DIO-32-TiCo2 Rev. E, MIO-D12 Rev. E, SPI-2-D Rev. E, SPI-2-T Rev. E

Example

- / -

P2_Digout_Fifo_Start

P2_Digout_Fifo_Start starts the edge output on the specified modules.

Syntax

```
#Include ADwinPro_All.inc
P2_Digout_Fifo_Start(module_pattern)
```

Parameters

module_pattern	Bit pattern to access the modules: Bit = 0: Ignore module. Bit = 1: Start edge output on the module.	LONG

Bits in module_pattern	31:15	14	13	…	01	00
Module address	–	15	14	…	2	1

Notes

On the module DIO-32-TiCo, the output is available since revision E03.

On a module with TiCo2 processor, the previously set voltage levels are active after power-up. For safety reasons you should set the channel levels again in the **Init:** section with **P2_DigProg_Set_IO_Level** before starting the output.

After start, the module counter starts to count with 0. The module counter is used to do exact output timing, see **P2_Digout_Fifo_Write**.

The timer value is regularly increased by 1, so the timer will reach the original timer value after 2^{32} ticks. A "missed" edge output is done only after the timer has happened this "overflow". The counter runs with different clock rates according to the module type:

Module	Clock rate	Tick length	overflow after
DIO-32-TiCo since Rev. E03 MIO-D12, AOut-1/16	100 MHz	10 ns	43 s = 10 ns × 2^{32}
DIO-32-TiCo2	200 MHz	5 ns	21 s = 5 ns × 2^{32}

See also

P2_Digout_Fifo_Clear, P2_Digout_Fifo_Enable, P2_Dig_Fifo_Mode, P2_Digout_Fifo_Read_Timer, P2_Digout_Fifo_Write, P2_DigProg

Valid for

AOut-1/16 Rev. E, DIO-32-TiCo Rev. E, DIO-32-TiCo2 Rev. E, MIO-D12 Rev. E, SPI-2-D Rev. E, SPI-2-T Rev. E

Example

see P2_Dig_Fifo_Mode

P2_Digout_Fifo_Write

P2_Digout_Fifo_Write writes value pairs into the output edge FIFO.

Syntax

```
#Include ADwinPro_All.inc
P2_Digout_Fifo_Write(module, count, values[],
    start_index)
```

Parameters

module	Specified module address (1…15).	LONG
count	Number of value pairs to write: DIO32-TiCo2: 1…2047 all other modules: 1…511	LONG
values[]	Array containing bit patterns of level status and time stamps for output timing. Each bit corresponds to a digital output or a specific function (see table).	LONG ARRAY
start_index	Start index of the array values[], where the first value is read.	LONG

Module AOut-1/16:

Bit no.	31	30	29	…	16	15…0
Output / function	status TTL output	status DAC output	29	…	16	DAC value

Other modules:

Bit no.	31	30	…	2	1	0
Output	31	30	…	2	1	0

Notes

You must not write more value pairs into the FIFO than are free. The number of free values in the FIFO is returned with **P2_Digout_FIFO_Empty**.

The FIFO array may contain 511 or 2047 value pairs (level status and time stamp) in maximum. If and as long as the FIFO array is filled completely, no more value pair can be written into.

The array values[] has to contain value pairs of level status and appropriate time stamp. Double time values refer (in this order) to the modules a) DIO32-TiCo, MIO-D12, and AOut-1/16, and b) DIO32-TiCo2:

- Odd numbered array elements hold the level status of output channels 0…31 as bit pattern.

  With module AOut-1/16, any array element contains two output values and 2 status bits:
  Bits 0…15 contain a digital value for DAC output.
  Bits 16…29 contain the levels of the outputs 16…29 as bit pattern.
  Bits 30…31 determine, which values are output. With bit = 1 a value is being output, with bit = 0 the previous value remains unchanged:
  - Bit 30: bits 0…15 are output as DAC value.
  - Bit 31: bits 16…29 are output to TTL outputs.
- Even numbered array elements hold a time stamp (absolute or relative, see **P2_Dig_Fifo_Mode**). The difference between two output times must be at least 20ns/10ns.

The value of a time stamp is counted in processor clocks i.e. in units of 10ns / 5ns.
- Please note: If the continuous output is enabled on a module with TiCo2 processor, the first time stamp in the array `values[]` must have a value greater than or equal 2 (=10ns).
  As long as the continuous output is running, no more values can be written into the output FIFO.

The output runs like follows:
- The module counter is increased every 10ns / 5ns by 1.
- If the counter value equals the time stamp of the current value pair in the FIFO, the bit pattern are output to the specified output channels.
- If a bit pattern has been output, the value pair is deleted from the FIFO.
- The value pairs are processed in the order as they were written in to the FIFO.

Therefore:

A time stamp defines the exact output time, and in time units of 10ns / 5ns. The value can be given in two ways:
- As absolute value in relation to the starting time of the module counter using **`P2_Digout_Fifo_Start`**.
  A time stamp of 152 would have the appropriate bit pattern be output exactly at 1.52µs/0.76µs after the module counter has started.
- As relative value which is relative to the previous time stamp.
  A time stamp of 152 would have the appropriate bit pattern be output exactly at 1.52µs/0.76µs after the previous pattern was output.

Time stamps must be stored in ascending order.

The FIFO must be filled with data early enough, so that the next output time is located in the future. But if the FIFO runs empty please note:
- With absolute values, the time stamp must be greater than the current timer value. Otherweise the edge output is "missed" and executed only after the timer has run once around. (about 43 / 21 seconds).
- With relative values, the time stamp must be greater than the time period since the previous pattern output (when the FIFO ran empty). If this fails, the bit pattern is output immediately (but obviously with delay); the next time stamp will then be relative to the delayed output time.

See also

P2_Digout_Fifo_Empty, P2_Digout_Fifo_Enable, P2_Dig_Fifo_Mode, P2_Digout_Fifo_Read_Timer, P2_Digout_Fifo_Start, P2_Digout_Long, P2_DigProg

Valid for

AOut-1/16 Rev. E, DIO-32-TiCo Rev. E, DIO-32-TiCo2 Rev. E, MIO-D12 Rev. E, SPI-2-D Rev. E, SPI-2-T Rev. E

Example

see P2_Dig_Fifo_Mode

P2_Digout_Long

P2_Digout_Long sets or clears all outputs on the specified module via the passed 32 bit value.

Syntax

```
#Include ADwinPro_All.Inc
P2_Digout_Long(module,pattern)
```

Parameters

module	Specified module address (1…15).	LONG
pattern	Bit pattern that sets the digital outputs: Bit = 0: Set output to level "low". Bit = 1: Set output to level "high".	LONG

Module AOut-1/16:

Bit no.	31	…	16	15…0
Output	31	…	16	–

Other modules:

Bit no.	31	30	…	2	1	0
Output	31	30	…	2	1	0

Notes

The specified channels must be first programmed as outputs using **P2_DigProg**. This is not required for module AOut-1/16.

With module version TRA-16-G Rev. E, high level switches to ground, not to V_{CC}.

See also

P2_Digout, P2_Digout_Bits, P2_DigProg

Valid for

AOut-1/16 Rev. E, DIO-32 Rev. E, DIO-32-TiCo Rev. E, DIO-32-TiCo2 Rev. E, REL-16 Rev. E, SPI-2-D Rev. E, SPI-2-T Rev. E, TRA-16 Rev. E

Example

```
#Include ADwinPro_All.Inc

Init:
  REM With AOUT-1/16: delete line using P2_DigProg
  P2_DigProg(1,0FFFFh)      'DIO31:00 as outputs

Event:
  P2_Digout_Long(1,1000000)'Output the value 1 million as binary
                           'value on the DIOs
```

P2_Digout_Reset

P2_Digout_Reset sets the specified outputs of the specified module to the level "low".

Syntax

```
#Include ADwinPro_All.Inc
P2_Digout_Reset(module,clear)
```

Parameters

module	Specified module address (1…15).	LONG
clear	Bit pattern that sets specified digital outputs to the level "low". Bit = 0: Output remains unchanged. Bit = 1: Set output to level "low".	LONG

Module AOut-1/16:

Bit no.	31	…	16	15…0
Output	29	…	16	–

Other modules:

Bit no.	31	30	…	2	1	0
Output	31	30	…	2	1	0

Notes

The specified channels must be first programmed as outputs using **P2_DigProg**. This is not required for module AOut-1/16.

You can clear any required outputs without changing the status of the remaining outputs.

With module version TRA-16-G Rev. E, high level switches to ground, not to V_{CC}.

See also

P2_Digout, P2_Digout_Bits, P2_Digout_Long, P2_Digout_Set, P2_DigProg

Valid for

AOut-1/16 Rev. E, DIO-32 Rev. E, DIO-32-TiCo Rev. E, DIO-32-TiCo2 Rev. E, REL-16 Rev. E, SPI-2-D Rev. E, SPI-2-T Rev. E, TRA-16 Rev. E

Example

```
#Include ADwinPro_All.Inc

Init:
  REM With AOUT-1/16: delete line using P2_DigProg
  REM Set channels 0…31 as outputs
  P2_DigProg(1,01111b)

Event:
  If (Par_1 = 1) Then       'Get condition
    REM lower word: clear even-numbered bits
    P2_Digout_Reset(1,0AAAAh)
  EndIf
```

P2_Digout_Set

P2_Digout_Set sets the specified outputs of the specified module to the level "high".

Syntax

```
#Include ADwinPro_All.Inc
P2_Digout_Set(module,set)
```

Parameters

module	Specified module address (1…15).	LONG
set	Bit pattern that sets specified digital outputs to the level "high". Bit = 0: Output remains unchanged. Bit = 1: Set output to level "high".	LONG

Module AOut-1/16:

Bit no.	31	…	16	15…0
Output	29	…	16	–

Other modules:

Bit no.	31	30	…	2	1	0
Output	31	30	…	2	1	0

Notes

The specified channels must be first programmed as outputs using **P2_DigProg**. This is not required for module AOut-1/16.

P2_Digout_Set can set any required outputs without changing the status of the remaining outputs.

With module version TRA-16-G Rev. E, high level switches to ground, not to V_{CC}.

See also

P2_Digout, P2_Digout_Bits, P2_Digout_Long, P2_Digout_Reset, P2_DigProg

Valid for

AOut-1/16 Rev. E, DIO-32 Rev. E, DIO-32-TiCo Rev. E, DIO-32-TiCo2 Rev. E, REL-16 Rev. E, SPI-2-D Rev. E, SPI-2-T Rev. E, TRA-16 Rev. E

Example

```
#Include ADwinPro_All.Inc

Init:
  REM With AOUT-1/16: delete line using P2_DigProg
  REM Set channels 0…31 as outputs
  P2_DigProg(1,1111b)

Event:
  If (Par_1 = 1) Then       'Get condition
    REM lower word: Set byte MSBs
    P2_Digout_Set(1,8080h)
  EndIf
```

P2_DigProg

P2_DigProg programs the digital channels 0…31 of the specified module as inputs or outputs in groups of 8.

Syntax

```
#Include ADwinPro_All.Inc
P2_DigProg(module,pattern)
```

Parameters

module	Specified module address (1…15).	LONG
pattern	Bit pattern that sets the channels as inputs or outputs: Bit = 0: Set group of channels as inputs. Bit = 1: Set group of channels as outputs.	LONG

Bit no.	31…4	3	2	1	0
channel no.	–	31:24	23:16	15:08	07:00

Notes

After power-up of the system all channels are configured as inputs.

Channels can only be set as inputs or outputs in groups of 8, 4 relevant bits only, the other bits are ignored.

See also

P2_DigProg_Bits, P2_Dig_Latch, P2_Dig_Read_Latch, P2_Dig_Write_Latch

P2_Digin_Long, P2_Digout_Bits, P2_Digout, P2_Digout_Long, P2_Get_Digout_Long

Valid for

DIO-32 Rev. E, DIO-32-TiCo Rev. E, DIO-32-TiCo2 Rev. E, SPI-2-D Rev. E, SPI-2-T Rev. E

Example

```
#Include ADwinPro_All.Inc

Init:
  REM Configure channels 0…7 of the DIO module no. 1 as inputs and
  REM channels 8…31 as inputs
  P2_DigProg(1, 1110b)
```

P2_DigProg_Bits

P2_DigProg_Bits programs the digital channels 0…11 of the specified module as inputs or outputs.

Syntax

```
#Include ADwinPro_All.Inc
P2_DigProg_Bits(module,pattern)
```

Parameters

module	Specified module address (1…15).	LONG
pattern	Bit pattern that sets the channels as inputs or outputs: Bit = 0: Set channel as input. Bit = 1: Set channel as output.	LONG

Bit no.	31…12	11	10	…	1	0
channel no.	–	11	10	…	1	0

Notes

After power-up of the system all channels are configured as inputs.

See also

P2_DigProg, P2_Dig_Latch, P2_Dig_Read_Latch, P2_Dig_Write_Latch

P2_Digin_Long, P2_Digout_Bits, P2_Digout, P2_Digout_Long, P2_Get_Digout_Long

Valid for

SPI-2-D Rev. E

Example

```
#Include ADwinPro_All.Inc

Init:
  REM Configure channels 0…11 of the DIO module no. 1 as inputs and
  REM channels 12…31 as inputs
  P2_DigProg_Bits(1, 0FFFFFF00h)
```

P2_Digprog_Set_IO_Level

P2_Digprog_Set_IO_Level sets the voltage level of the digital channels in groups of 8 to a defined value.

Syntax

```
#Include ADwinPro_All.Inc
P2_Digprog_Set_IO_Level(module, group, level)
```

Parameters

module	Specified module address (1…15).	LONG
group	Number (0…3) of the channel group the voltage level of which is to be set: 0: Channels 0…7 1: Channels 8…15 2: Channels 16…23 3: Channels 24…31	LONG
level	Value in digits (0…255) to set the voltage level of the digital channels. Assignment see table.	LONG

Digits	Voltage level
0	1.6V
100	2.2V
193	3.3V
240	4.4V
255	4.7V

Notes

After power-up, the previously set voltage levels are active. For safety reasons you should set the channel levels again in the **Init:** section with **P2_DigProg_Set_IO_Level** before starting the output.

Channels can only be set as inputs or outputs in groups of 8, 4 relevant bits only, the other bits are ignored.

The described voltage levels are high levels and are achieved with a tolerance of ±0.1V. The correlation of digital value and voltage level is non-linear.

Please note: Setting a different voltage level may take–according to voltage difference and change direction–up to four milliseconds. Start the output with **P2_Digout_Fifo_Start** only after an appropriate waiting period.

See also

P2_Digin_Long, P2_Digout, P2_Digout_Long, P2_Get_Digout_Long

Valid for

DIO-32-TiCo2 Rev. E

Example

```
#Include ADwinPro_All.Inc

Init:
  Rem set voltage level 2.8 V for channels 0..15
  P2_DigProg_Set_IO_Level(1, 0, 160)
  P2_DigProg_Set_IO_Level(1, 1, 160)
```

P2_Get_Digout_Long

P2_Get_Digout_Long returns the contents of the output latch (register for digital outputs) on the specified module.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = P2_Get_Digout_Long(module)
```

Parameters

module	Specified module address (1…15).	LONG
ret_val	Contents of the output latch (bits 31:00).	LONG

Notes

Returning the current status of the outputs instead of the output latch is technically impossible.

With module version TRA-16-G Rev. E, high level switches to ground, not to V_{CC}.

See also

P2_Dig_Latch, P2_Dig_Read_Latch, P2_Dig_Write_Latch

P2_DigProg, P2_Digin_Long, P2_Digout_Bits, P2_Digout, P2_Digout_Long

Valid for

AOut-1/16 Rev. E, DIO-32 Rev. E, DIO-32-TiCo Rev. E, DIO-32-TiCo2 Rev. E, REL-16 Rev. E, SPI-2-D Rev. E, SPI-2-T Rev. E, TRA-16 Rev. E

Example

```
#Include ADwinPro_All.Inc

Event:
Par_1 = P2_Get_Digout_Long(1)'return bits 31:00 from the latch
```

3.7 Pro II: Counter Modules

This section describes instructions which apply to Pro II modules with counters:

– P2_Cnt_Clear (page 188)
– P2_Cnt_Enable (page 189)
– P2_Cnt_Get_Status (page 190)
– P2_Cnt_Get_PW (page 191)
– P2_Cnt_Get_PW_HL (page 192)
– P2_Cnt_Latch (page 193)
– P2_Cnt_Mode (page 194)
– P2_Cnt_PW_Enable (page 196)
– P2_Cnt_PW_Latch (page 197)
– P2_Cnt_Read (page 198)
– P2_Cnt_Read4 (page 199)
– P2_Cnt_Read_Int_Register (page 200)
– P2_Cnt_Read_Latch (page 201)
– P2_Cnt_Read_Latch4 (page 202)
– P2_Cnt_Sync_Latch (page 203)
– P2_SSI_Mode (page 205)
– P2_SSI_Read (page 206)
– P2_SSI_Read2 (page 208)
– P2_SSI_Set_Bits (page 209)
– P2_SSI_Set_Clock (page 210)
– P2_SSI_Set_Delay (page 211)
– P2_SSI_Start (page 212)
– P2_SSI_Status (page 213)

In the Instruction List sorted by Module Types (annex A.2) you will find which of the functions corresponds to the *ADwin-Pro II* modules.

P2_Cnt_Clear

P2_Cnt_Clear sets the counter values of one or more counters to 0 (zero), according to the given bit pattern.

Syntax

```
#Include ADwinPro_All.Inc
P2_Cnt_Clear(module, pattern)
```

Parameters

module	Specified module address (1…15).	LONG
pattern	Bit pattern, assignment to the counters, see table. Bit = 0: No function. Bit = 1: Reset counter.	LONG

Bit no.	31…4	3	2	1	0
Counter no.	–	4	3	2	1

Notes

After **P2_Cnt_Clear** has been executed the bit pattern is automatically reset to 0 (zero), so the counters start counting from 0.

See also

P2_Cnt_Enable, P2_Cnt_Get_Status, P2_Cnt_Latch, P2_Cnt_Mode, P2_Cnt_Read, P2_Cnt_Read4, P2_Cnt_Read_Latch, P2_Cnt_Read_Latch4

Valid for

Example

```
#Include ADwinPro_All.Inc
Dim old_1, new_1 As Long
Dim old_2, new_2 As Long

Init:
  old_1 = 0                    'initialize
  old_2 = 0
  Rem Counter 1: clock-dir, enable CLR input
  P2_Cnt_Mode(1,1,10000b)
  Rem Counter 2: clock-dir, enable LATCH input
  P2_Cnt_Mode(1,2,00000b)
  P2_Cnt_Clear(1,11b)          'reset counters 1+2 to 0
  P2_Cnt_Enable(1,11b)         'start counters 1+2, stop
counters 3+4

Event:
  P2_Cnt_Latch(1,11b)          'latch both counters 1+2
  new_1 = P2_Cnt_Read_Latch(1,1)'read latch A counter 1 and…
  new_2 = P2_Cnt_Read_Latch(1,2)'latch A counter 2
  Par_1 = new_1 - old_1'get difference (f = impulses / time)
  Par_2 = new_2 - old_2'-"-
  old_1 = new_1                'store new counter value as old
  old_2 = new_2                '-"-
```

P2_Cnt_Enable

P2_Cnt_Enable enables or disables the counters selected by pattern.

Syntax

```
#Include ADwinPro_All.Inc
P2_Cnt_Enable(module, pattern)
```

Parameters

module	Specified module address (1…15).	LONG
pattern	Bit pattern Bit = 0: Disable counter. Bit = 1: Enable counter.	LONG

Bit no.	31…4	3	2	1	0
Counter no.	–	VR4	VR3	VR2	VR1

Notes

PWM counters are started or stopped with **P2_Cnt_PW_Enable**.

See also

P2_Cnt_Clear, P2_Cnt_PW_Enable, P2_Cnt_Get_Status, P2_Cnt_Latch, P2_Cnt_Mode, P2_Cnt_Read, P2_Cnt_Read4, P2_Cnt_Read_Latch, P2_Cnt_Read_Latch4

Valid for

CNT-D Rev. E, CNT-I Rev. E, CNT-T Rev. E, MIO-4-ET1 Rev. E, MIO-D12 Rev. E

Example

```
#Include ADwinPro_All.Inc
Dim old_1, new_1 As Long
Dim old_2, new_2 As Long

Init:
  old_1 = 0                    'initialize…
  old_2 = 0
  Rem Counter 1: clock-dir, enable CLR input
  P2_Cnt_Mode(1,1,10000b)
  Rem Counter 2: clock-dir, enable LATCH input
  P2_Cnt_Mode(1,2,00000b)
  P2_Cnt_Clear(1,11b)          'reset counters 1+2 to 0
  P2_Cnt_Enable(1,11b)         'start counters 1+2, stop
counters 3+4

Event:
  P2_Cnt_Latch(1,11b)          'latch both counters 1+2
  new_1 = P2_Cnt_Read_Latch(1,1)'read latch A counter 1 and…
  new_2 = P2_Cnt_Read_Latch(1,2)'latch A counter 2
  Par_1 = new_1 - old_1'get difference (f = impulses / time)
  Par_2 = new_2 - old_2'-"-
  old_1 = new_1                'store new counter value as old
  old_2 = new_2                '-"-
```

P2_Cnt_Get_Status

P2_Cnt_Get_Status returns the counter status register of one counter.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = P2_Cnt_Get_Status(module, counter_no)
```

Parameters

module	Specified module address (1…15).	LONG
counter_no	Counter number: 1…4.	LONG
ret_val	Content of the counter status register: Hints for potential error sources. Meaning of bits 0…4 see table.	LONG

Bit no.	31…5	4	3	2	1	0
Signal	–	C	L	N	B	A

- : don't care (mask with **01Fh**).
A: Signal A (static).
B: Signal B(static).
N: CLR/Latch input (static).
L: Line error (cable not connected or the line is broken).
C: Correlation error* (signals A and B are identical, they are not phase-shifted by approx. 90°).

Notes

A line error (L) can only be detected with differential inputs! For TTL inputs these bits are always 0.

The status register is automatically reset by reading.

See also

P2_Cnt_Enable, P2_Cnt_PW_Enable, P2_Cnt_Get_PW, P2_Cnt_Mode, P2_Cnt_Read

Valid for

CNT-D Rev. E, CNT-I Rev. E, CNT-T Rev. E, MIO-4-ET1 Rev. E, MIO-D12 Rev. E

Example

- / -

P2_Cnt_Get_PW

P2_Cnt_Get_PW returns frequency and duty cycle of a PWM counter.

Syntax

```
#Include ADwinPro_All.Inc
P2_Cnt_Get_PW(module, pwm_no,frequency,dutycycle)
```

Parameters

module	Specified module address (1…15).	LONG
pwm_no	Number (1…4) of PWM counter.	LONG
frequency	Frequency in Hertz: 0.023 Hz …100 MHz.	FLOAT CONST
dutycycle	Duty cycle in percent: 0.0…100.0.	FLOAT CONST

Notes

The return values are given in the parameters `frequency` and `dutycycle`.

See also

P2_Cnt_PW_Enable, P2_Cnt_Get_Status, P2_Cnt_Get_PW_HL, P2_Cnt_Mode, P2_Cnt_PW_Latch, P2_Cnt_Sync_Latch

Valid for

CNT-D Rev. E, CNT-I Rev. E, CNT-T Rev. E, MIO-4-ET1 Rev. E, MIO-D12 Rev. E

Example

```
#Include ADwinPro_All.Inc
Dim old_1, new_1 As Long
Dim old_2, new_2 As Long

Init:
  old_1 = 0                    'initialize
  old_2 = 0
  P2_Cnt_Mode(1,1,0)           'Counter 1: PWM at input CLK
  P2_Cnt_Mode(1,2,0)           'Counter 2: PWM at input CLK
  P2_Cnt_PW_Enable(1,0011b) 'start PWM counters 1+2, stop 3+4

Event:
  P2_Cnt_PW_Latch(1,11b)       'latch both counters 1+2
  P2_Cnt_Get_PW_HL(1,1,Par_1,Par_2) 'read high/low time
  P2_Cnt_Get_PW(1,1,FPar_1,FPar_2) 'read frequency/duty cycle
```

P2_Cnt_Get_PW_HL

P2_Cnt_Get_PW_HL returns a stored high and low time of a PWM counter.

Syntax

```
#Include ADwinPro_All.Inc
P2_Cnt_Get_PW_HL(module, counter_no, hightime,
          lowtime)
```

Parameters

module	Specified module address (1…15).	LONG
counter_no	Number (1…4) of PWM counter.	LONG
hightime	Pulse duration in units of 10ns: high level time of PWM signal.	LONG CONST
lowtime	Pulse period in units of 10ns: low level time of PWM signal.	LONG CONST

Notes

The return values are given in the parameters hightime and lowtime.

See also

P2_Cnt_PW_Enable, P2_Cnt_Get_Status, P2_Cnt_Get_PW, P2_Cnt_Mode, P2_Cnt_PW_Latch, P2_Cnt_Sync_Latch

Valid for

CNT-D Rev. E, CNT-I Rev. E, CNT-T Rev. E, MIO-4-ET1 Rev. E, MIO-D12 Rev. E

Example

```
#Include ADwinPro_All.Inc
Dim old_1, new_1 As Long
Dim old_2, new_2 As Long

Init:
  old_1 = 0                    'initialize
  old_2 = 0
  P2_Cnt_Mode(1,1,0)           'Counter 1: PWM at input CLK
  P2_Cnt_Mode(1,2,0)           'Counter 2: PWM at input CLK
  P2_Cnt_PW_Enable(1,0011b) 'start PWM counters 1+2, stop 3+4

Event:
  P2_Cnt_PW_Latch(1,11b)       'latch both counters 1+2
  P2_Cnt_Get_PW_HL(1,1,Par_1,Par_2) 'read high/low time
  P2_Cnt_Get_PW(1,1,FPar_1,FPar_2) 'read frequency / duty
cycle
```

P2_Cnt_Latch

P2_Cnt_Latch transfers the current counter values of one or more counters into the relevant Latch A, depending on the bit pattern.

Syntax

```
#Include ADwinPro_All.Inc
P2_Cnt_Latch(module, pattern)
```

Parameters

module	Specified module address (1…15).	LONG
pattern	Bit pattern. Bit = 0: no function. Bit = 1: transfer counter values into Latch A .	LONG

Bit no.	31…4	3	2	1	0
Counter no.	–	4	3	2	1

Notes

After **Cnt_Latch** has been executed the bit pattern is automatically reset to 0 (zero).

Latch A is read out into a variable with **Cnt_Read_Latch** command.

Transferring counter values can be started synchronously to actions on other modules with **P2_Sync_All**. You can disable selected counters for synchonization using **P2_Sync_Enable**.

See also

P2_Cnt_Clear, P2_Cnt_Enable, P2_Cnt_Get_Status, P2_Cnt_Mode, P2_Cnt_Read, P2_Cnt_Read4, P2_Cnt_Read_Latch, P2_Cnt_Read_Latch4, P2_Sync_All, P2_Sync_Enable

Valid for

CNT-D Rev. E, CNT-I Rev. E, CNT-T Rev. E, MIO-4-ET1 Rev. E, MIO-D12 Rev. E

Example

```
#Include ADwinPro_All.Inc
Dim old_1, new_1 As Long
Dim old_2, new_2 As Long

Init:
  old_1 = 0                    'initialize
  old_2 = 0
  Rem Counter 1: clock-dir, enable CLR input
  P2_Cnt_Mode(1,1,10000b)
  Rem Counter 2: clock-dir, enable LATCH input
  P2_Cnt_Mode(1,2,00000b)
  P2_Cnt_Clear(1,11b)          'reset counters 1+2 to 0
  P2_Cnt_Enable(1,11b)         'start counters 1+2, stop
counters 3+4

Event:
  P2_Cnt_Latch(1,11b)          'latch both counters 1+2
  new_1 = P2_Cnt_Read_Latch(1,1)'read latch A counter 1 and…
  new_2 = P2_Cnt_Read_Latch(1,2)'latch A counter 2
  Par_1 = new_1 - old_1'get difference (f = impulses / time)
  Par_2 = new_2 - old_2'-"-
  old_1 = new_1                'store new counter value as old
  old_2 = new_2                '-"-
```

P2_Cnt_Mode

P2_Cnt_Mode defines the operating mode of one counter.

Syntax

```
#Include ADwinPro_All.Inc
P2_Cnt_Mode(module, counter_no, pattern)
```

Parameters

module	Specified module address (1…15).	LONG
counter_no	Counter number: 1…4.	LONG
pattern	Bit pattern to set the operating mode of a counter.	LONG

Bit no.	Meaning
Bit 0	Counter mode: Bit = 0: clock/direction mode. Bit = 1: four edge evaluation mode (A-B).
Bit 1	Clear mode. Signal condition which clears the counter: Bit = 0: TTL level high at input CLR. Bit = 1: TTL level high at all inputs A, B, CLR. Available with four edge evaluation mode only.
Bit 2	Revert input A / CLK in mode clock/direction: Bit = 0: Input is not reverted. Bit = 1: Input is reverted.
Bit 3	Revert input B / DIR in mode clock/direction: Bit = 0: Input is not reverted. Bit = 1: Input is reverted.
Bit 4	Set use of input CLR / LATCH. Bit = 0: CLR input: clear counter. Bit = 1: LATCH input: latch counter.
Bit 5	Enable input CLR / LATCH. Bit = 0: Input CLR / LATCH is disabled. Bit = 1: Input CLR / LATCH is enabled.
Bit 6	Select edge for PWM analysis. Bit = 0: rising edge. Bit = 1: falling edge.
Bit 7,8	Select input for PWM analysis. 00b: Input A / CLK 01b: Input B / DIR 10b: Input CLR / LATCH With CNT-T note the pin junction to the counter mode (see below).
Bits 9…31	reserved

Notes

Please use **P2_Cnt_Mode** only when the counter is disabled, see **P2_Cnt_Enable**.

With standard clear mode (bit 1=0), the counter value is reset to zero as long as TTL level high is given at the input. In order to clear the counter, the input CLR must be enabled with bit 5=1.

On module CNT-T Rev. E, the PWM input pins A and B are usable only in conjunction with four edge evaluation mode and the PWM input pins CLK and DIR in conjunction with clock/direction mode.

See also

P2_Cnt_Clear, P2_Cnt_Enable, P2_Cnt_PW_Enable, P2_Cnt_Get_Status

Valid for

CNT-D Rev. E, CNT-I Rev. E, CNT-T Rev. E, MIO-4-ET1 Rev. E, MIO-D12 Rev. E

Example

```
#Include ADwinPro_All.Inc
Dim old_1, new_1 As Long
Dim old_2, new_2 As Long

Init:
  old_1 = 0                   'initialize
  old_2 = 0
  Rem Counter 1: clock-dir, set and enable LATCH input
  P2_Cnt_Mode(1,1,10000b)
  Rem Counter 2: clock-dir, set and enable CLR input
  P2_Cnt_Mode(1,2,00000b)
  P2_Cnt_Clear(1,11b)         'reset counters 1+2 to 0
  Rem start counters 1+2 and stop counters 3+4
  P2_Cnt_Enable(1,11b)

Event:
  P2_Cnt_Latch(1,11b)         'latch both counters 1+2
  new_1 = P2_Cnt_Read_Latch(1,1)'read latch A counter 1 and
  new_2 = P2_Cnt_Read_Latch(1,2)'latch A counter 2
  Par_1 = new_1 - old_1 'get differences (f = impulses / time)
  Par_2 = new_2 - old_2
  old_1 = new_1               'store new counter values as old
  old_2 = new_2
```

P2_Cnt_PW_Enable

P2_Cnt_PW_Enable enables or disables the counters selected by pattern.

Syntax

```
#Include ADwinPro_All.Inc
P2_Cnt_PW_Enable(module, pattern)
```

Parameters

module	Specified module address (1…15).	LONG
pattern	Bit pattern Bit = 0: Disable counter. Bit = 1: Enable counter.	LONG

Bit no.	31…4	3	2	1	0
Counter no.	–	PW 4	PW 3	PW 2	PW 1

Notes

Standard counters are started or stopped with **P2_Cnt_Enable**.

See also

P2_Cnt_Enable, P2_Cnt_Get_Status, P2_Cnt_Get_PW, P2_Cnt_Get_PW_HL, P2_Cnt_Mode, P2_Cnt_PW_Latch

Valid for

CNT-D Rev. E, CNT-I Rev. E, CNT-T Rev. E, MIO-4-ET1 Rev. E, MIO-D12 Rev. E

Example

```
#Include ADwinPro_All.Inc
Dim old_1, new_1 As Long
Dim old_2, new_2 As Long

Init:
  old_1 = 0                    'initialize…
  old_2 = 0
  Rem Counter 1: clock-dir, enable CLR input
  P2_Cnt_Mode(1,1,10000b)
  Rem Counter 2: clock-dir, enable LATCH input
  P2_Cnt_Mode(1,2,00000b)
  P2_Cnt_Clear(1,11b)          'reset counters 1+2 to 0
  P2_Cnt_PW_Enable(1,0011b) 'start PWM counters 1+2, stop 3+4

Event:
  P2_Cnt_PW_Latch(1,11b)       'latch both counters 1+2
  P2_Cnt_Get_PW_HL(1,1,Par_1,Par_2) 'read high/low time
  P2_Cnt_Get_PW(1,1,FPar_1,FPar_2) 'read frequency / duty
cycle
```

P2_Cnt_PW_Latch

P2_Cnt_PW_Latch copies the value of one or more PWM counters into a buffer.

Syntax

```
#Include ADwinPro_All.Inc
P2_Cnt_PW_Latch(module, pattern)
```

Parameters

module	Specified module address (1…15).	LONG
pattern	Bit pattern. Bit = 0: no function. Bit = 1: transfer PWM counter value into a buffer.	LONG

Bit no.	31…4	3	2	1	0
Counter no.	–	4	3	2	1

Notes

The buffer is to be read with **P2_Cnt_Get_PW** or **P2_Cnt_Get_PW_HL**.

See also

P2_Cnt_PW_Enable, P2_Cnt_Get_Status, P2_Cnt_Get_PW, P2_Cnt_Get_PW_HL, P2_Cnt_Mode

Valid for

CNT-D Rev. E, CNT-I Rev. E, CNT-T Rev. E, MIO-4-ET1 Rev. E, MIO-D12 Rev. E

Example

```
#Include ADwinPro_All.Inc
Dim old_1, new_1 As Long
Dim old_2, new_2 As Long

Init:
  old_1 = 0                    'initialize
  old_2 = 0
  P2_Cnt_Mode(1,1,0)           'Counter 1: PWM at input CLK
  P2_Cnt_Mode(1,2,0)           'Counter 2: PWM at input CLK
  P2_Cnt_PW_Enable(1,0011b) 'start PWM counters 1+2, stop 3+4

Event:
  P2_Cnt_PW_Latch(1,11b)       'latch both counters 1+2
  P2_Cnt_Get_PW_HL(1,1,Par_1,Par_2) 'read high/low time
  P2_Cnt_Get_PW(1,1,FPar_1,FPar_2) 'read frequency/duty cycle
```

P2_Cnt_Read

P2_Cnt_Read transfers a current counter value into Latch A and returns the value.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = P2_Cnt_Read(module, counter_no)
```

Parameters

module	Specified module address (1…15).	LONG
counter_no	Counter number: 1…4.	LONG
ret_val	Counter values.	LONG

Notes

Use the return value in calculations only with variables of the type **Long** (e.g. differences or count direction).

See also

P2_Cnt_Clear, P2_Cnt_Enable, P2_Cnt_Get_Status, P2_Cnt_Latch, P2_Cnt_Mode, P2_Cnt_Read4, P2_Cnt_Read_Latch, P2_Cnt_Read_Latch4

Valid for

CNT-D Rev. E, CNT-I Rev. E, CNT-T Rev. E, MIO-4-ET1 Rev. E, MIO-D12 Rev. E

Example

```
#Include ADwinPro_All.Inc
Dim old_1, new_1 As Long
Dim old_2, new_2 As Long

Init:
  old_1 = 0                    'initialize
  old_2 = 0
  Rem Counter 1: clock-dir, enable CLR input
  P2_Cnt_Mode(1,1,10000b)
  Rem Counter 2: clock-dir, enable LATCH input
  P2_Cnt_Mode(1,2,00000b)
  P2_Cnt_Clear(1,11b)          'reset counters 1+2 to 0
  P2_Cnt_Enable(1,11b)         'start counters 1+2, stop
counters 3+4

Event:
  P2_Cnt_Latch(1,11b)          'latch both counters 1+2
  new_1 = P2_Cnt_Read_Latch(1,1)'read latch A counter 1 and…
  new_2 = P2_Cnt_Read_Latch(1,2)'latch A counter 2
  Par_1 = new_1 - old_1'get difference (f = impulses / time)
  Par_2 = new_2 - old_2'-"-
  old_1 = new_1                'store new counter value as old
  old_2 = new_2                '-"-
```

P2_Cnt_Read4

P2_Cnt_Read4 transfers a current counter value into Latch A and returns the value.

Syntax

```
#Include ADwinPro_All.Inc
P2_Cnt_Read4(module, array[], index)
```

Parameters

module	Specified module address (1…15).	LONG
array[]	Destination array to store counter values.	ARRAY
		LONG
index	First element in array[] to be written into.	LONG

Notes

Use the array[] values in calculations only with variables of the type **Long** (e.g. differences or count direction).

See also

P2_Cnt_Clear, P2_Cnt_Enable, P2_Cnt_Get_Status, P2_Cnt_Latch, P2_Cnt_Mode, P2_Cnt_Read, P2_Cnt_Read_Latch, P2_Cnt_Read_Latch4

Valid for

CNT-D Rev. E, CNT-I Rev. E, CNT-T Rev. E

Example

```
#Include ADwinPro_All.Inc
Dim Data_1[4] AS LONG
Dim old[4], new[4] As Long
Dim i As Long

Init:
  P2_Cnt_Enable(1,0)          'stop counters
  Rem Counter 1: clock-dir, enable CLR input
  P2_Cnt_Mode(1,1,10000b)
  P2_Cnt_Mode(1,2,0)          'all counters use external
trigger
  Rem Counter 3: clock-dir, enable CLR input
  P2_Cnt_Mode(1,3,10000b)
  P2_Cnt_Mode(1,4,0)          'all counters use external
trigger
  P2_Cnt_Clear(1,1111b)       'reset counters to 0
  P2_Cnt_Enable(1,1111b)'start counters

Event:
  P2_Cnt_Read4(1,new,1)       'read counter values into array
new
  For i = 1 To 4
    Data_1[i] = new[i]-old[i] 'get difference (f = impulses /
time)
    old[i] = new[i]           'store new counter value as old
  Next i
```

P2_Cnt_Read_Int_Register

T11 TiCo

P2_Cnt_Read_Int_Register returns the content of a counter register.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
ret_val = P2_Cnt_Read_Int_Register(module,
          counter_no, reg_no)
```

Parameters

module	Specified module address (1…15).	LONG
counter_no	Counter number: 1…4.	LONG
reg_no	Key number (0…15) for a counter register, see below.	LONG
ret_val	Content of the counter register.	LONG

reg_no	Register
0	Latch 1 for positive edges.
1	Latch 2 for positive edges.
2	Latch 3 for positive edges.
3	Latch 1 for negative edges.
4	Latch 2 for negative edges.
5	Latch 3 for negative edges.
6	Software latch for VR counter.
7	Software latch for PWM counter.
8	Shadow register for Latch 1, positive edges.
9	Shadow register for Latch 2, positive edges.
10	Shadow register for Latch 3, positive edges.
11	Shadow register for Latch 1, negative edges.
12	Shadow register for Latch 2, negative edges.
13	Shadow register for Latch 3, negative edges.
14	Shadow register for software latch, VR counter.
15	Counter status.

Notes

- / -

See also

P2_Cnt_Sync_Latch

Valid for

CNT-D Rev. E, CNT-I Rev. E, CNT-T Rev. E, MIO-4-ET1 Rev. E, MIO-D12 Rev. E

P2_Cnt_Read_Latch

P2_Cnt_Read_Latch returns the value of a counter's Latch A.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = P2_Cnt_Read_Latch(module, counter_no)
```

Parameters

module	Specified module address (1…15).	LONG
counter_no	Counter number: 1…4.	LONG
ret_val	Contents of Latch A .	LONG

Notes

Use the return value in calculations only with variables of the type Long (e.g. differences or count direction).

See also

P2_Cnt_Clear, P2_Cnt_Enable, P2_Cnt_Get_Status, P2_Cnt_Latch, P2_Cnt_Mode, P2_Cnt_Read, P2_Cnt_Read4, P2_Cnt_Read_Latch4

Valid for

CNT-D Rev. E, CNT-I Rev. E, CNT-T Rev. E, MIO-4-ET1 Rev. E, MIO-D12 Rev. E

Example

```
#Include ADwinPro_All.Inc
Dim old_1, new_1 As Long
Dim old_2, new_2 As Long

Init:
  old_1 = 0                    'initialize
  old_2 = 0
  Rem Counter 1: clock-dir, enable CLR input
  P2_Cnt_Mode(1,1,10000b)
  P2_Cnt_Mode(1,2,0)           'all counters use external
trigger
  P2_Cnt_Clear(1,11b)          'reset counters 1+2 to 0
  P2_Cnt_Enable(1,11b)         'start counters 1+2, stop
counters 3+4

Event:
  P2_Cnt_Latch(1,11b)          'latch both counters 1+2
  new_1 = P2_Cnt_Read_Latch(1,1)'read latch A counter 1 and…
  new_2 = P2_Cnt_Read_Latch(1,2)'latch A counter 2
  Par_1 = new_1 - old_1'get difference (f = impulses / time)
  Par_2 = new_2 - old_2'-"-
  old_1 = new_1                'store new counter value as old
  old_2 = new_2                '-"-
```

P2_Cnt_Read_Latch4

P2_Cnt_Read_Latch4 returns the values of 4 counter latches in an array.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = P2_Cnt_Read_Latch4(module, array[], index)
```

Parameters

module	Specified module address (1…15).	LONG
array[]	Destination array to store counter values.	ARRAY LONG
index	First element in array[] to be written into.	LONG

Notes

Use the array[] values in calculations only with variables of the type Long (e.g. differences or count direction).

See also

P2_Cnt_Clear, P2_Cnt_Enable, P2_Cnt_Get_Status, P2_Cnt_Latch, P2_Cnt_Mode, P2_Cnt_Read, P2_Cnt_Read4, P2_Cnt_Read_Latch

Valid for

CNT-D Rev. E, CNT-I Rev. E, CNT-T Rev. E

Example

```
#Include ADwinPro_All.Inc
Dim Data_1[4] AS LONG
Dim old[4], new[4] As Long
Dim i As Long

Init:
  P2_Cnt_Enable(1,0)           'stop counters
  Rem Counter 1: clock-dir, enable CLR input
  P2_Cnt_Mode(1,1,10000b)
  P2_Cnt_Mode(1,2,0)           'all counters use external
trigger
  Rem Counter 3: clock-dir, enable CLR input
  P2_Cnt_Mode(1,3,10000b)
  P2_Cnt_Mode(1,4,0)           'all counters use external
trigger
  P2_Cnt_Clear(1,1111b)        'reset counters to 0
  P2_Cnt_Enable(1,1111b)'start counters

Event:
  P2_Cnt_Latch(1,1111b)        'latch counters
  P2_Cnt_Read_Latch4(1,new,1) 'read counter values into array
new
  For i = 1 To 4
    Data_1[i] = new[i]-old[i] 'get difference (f = impulses /
time)
    old[i] = new[i]            'store new counter value as old
  Next i
```

P2_Cnt_Sync_Latch

P2_Cnt_Sync_Latch copies the contents of selected counters and PWM counters into latches.

Syntax

```
#Include ADwinPro_All.inc
P2_Cnt_Sync_Latch(module, pattern)
```

Parameters

module	Specified module address (1…15).	LONG
pattern	Bit pattern to select pairs of counters. Bit = 0: No counter selected. Bit = 1: Copy counter content into latch.	LONG

Bit no.	31…4	3	2	1	0
Counter no.	–	4	3	2	1

Notes

Each bit is associated either to a standard counter and a PWM counter. Both counter contents are latched at the same time. Therefore, the instruction has the same function as both **P2_Cnt_Latch** and **P2_Cnt_PW_Latch**.

The latches can be read e.g. with **P2_Cnt_Read_Latch** or **P2_Cnt_Get_PW**.

See also

P2_Cnt_Get_PW, P2_Cnt_Latch, P2_Cnt_Mode, P2_Cnt_PW_Latch, P2_Cnt_Read_Latch, P2_Sync_All

Valid for

CNT-D Rev. E, CNT-I Rev. E, CNT-T Rev. E, MIO-4-ET1 Rev. E, MIO-D12 Rev. E

Example

```
#Include ADwinPro_All.inc
#Define frequency FPar_1
Dim time, edges As Long
Dim rest As Float
Dim oldpw, oldcnt, newpw, newcnt As Long
Dim pw_cnt As Long

Init:
  Processdelay = 3000000      '100Hz
  P2_Cnt_Enable(1,0001b)
  P2_Cnt_Mode(1,1,00000000b) 'mode: clock/dir
  P2_Cnt_Clear(1,0Fh)
  P2_Cnt_Enable(1,0Fh)        'enable standard counters
  P2_Cnt_PW_Enable(1,0Fh)     'enable PW counters
  P2_Cnt_PW_Latch(1,0Fh)      'copy all counter values
  oldpw = 0
  oldcnt = 0
  frequency = 0

Event:
  P2_Cnt_Sync_Latch(1,0001b) 'latch all counter values
  newcnt = P2_Cnt_Read_Latch(1,1)    'vr-cnt
  edges = (newcnt-oldcnt)    'number of edges between events
  If (edges <> 0) Then
    PW_cnt = P2_Cnt_Read_Int_Register(1,1,8)
    time = PW_cnt - oldpw     'calculate timebase
    frequency = edges*100000000/time  'frequency
                              '(100000000->frequency of
P2-CNT-Module)
    oldcnt=newcnt             'store VR-counter value
    oldpw =newpw              'store PW-counter value
  EndIf
```

P2_SSI_Mode

P2_SSI_Mode sets the modes of all SSI decoders on the specified module, either "single shot" (read out once) or "continous" (read out continously).

Syntax

```
#Include ADwinPro_All.Inc
P2_SSI_Mode(module,pattern)
```

Parameters

module	Specified module address (1…15).	LONG
pattern	Operation mode of the SSI decoders, indicated as bit pattern. A bit is assigned to each of the decoders (see table). Bit = 0: "Single shot" mode, the encoder is read out once. Bit = 1: "Continuous" mode, the encoder is read out continously.	LONG

Bit no.	31:2	1	0
SSI decoder	–	2	1

Notes

If you select the mode "continuous", reading the encoder is started immediately. **P2_SSI_Start** is not necessary for this. With **P2_SSI_Set_Delay**, you set the time delay between reading two encoder values.

Using the "continuous" mode, some encoder types occasionally return the wrong counter value 0 (zero) instead of the corrct counter value. This error does not occur with the "single shot" mode.

See also

P2_SSI_Read, P2_SSI_Read2, P2_SSI_Set_Bits, P2_SSI_Set_Clock, P2_SSI_Set_Delay, P2_SSI_Start, P2_SSI_Status

Valid for

CNT-D Rev. E, MIO-4-ET1 Rev. E, MIO-D12 Rev. E

Example

```
#Include ADwinPro_All.Inc

Init:
  P2_SSI_Set_Clock(1,200)   'CLK (clock rate) = 250 kHz
  P2_SSI_Set_Delay(1,1,250)'Waiting delay decoder 1: 5µs
  P2_SSI_Set_Delay(1,2,500)'Waiting delay decoder 2: 10µs
  P2_SSI_Set_Bits(1,1,23)   'Amount of bits=23 (decoder 1)
  P2_SSI_Set_Bits(1,2,23)   'Amount of bits=23 (decoder 2)
  P2_SSI_Mode(1,3)          'continuous-mode for both decoders

Event:
  Par_1 = P2_SSI_Read(1,1) 'Read position value decoder 1
  Par_2 = P2_SSI_Read(1,2) 'Read position value decoder 2
```

P2_SSI_Read

P2_SSI_Read returns the last saved counter value of a specified SSI counter on the specified module.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = P2_SSI_Read(module,dcdr_no)
```

Parameters

module	Specified module address (1…15).	LONG
dcdr_no	Number (1, 2) of the SSI decoder whose counter value is to be read.	LONG
ret_val	Previous counter value of the SSI counter (= absolute value position of the encoder).	LONG

Notes

An encoder value is saved when the bits indicated by **P2_SSI_Set_Bits** are read.

Always the amount of bits is returned that is set before by **P2_SSI_Set_Bits**, even if this does not correspond to the resolution of the encoder.
In this case the returned counter value depends on the encoder (see documentation of the manufacturer). Normally there are the following rules:

- If the encoder has a higher resolution, its exceeding least-significant bits are not used.
- If the encoder has a lower resolution as indicated, a 0 (zero) is read for each missing most-significant bit.

See also

P2_SSI_Mode, P2_SSI_Read2, P2_SSI_Set_Bits, P2_SSI_Set_Clock, P2_SSI_Set_Delay, P2_SSI_Start, P2_SSI_Status

Valid for

CNT-D Rev. E, MIO-4-ET1 Rev. E, MIO-D12 Rev. E

Example

```
#Include ADwinPro_All.Inc
Dim m, n, y As Long

Init:
  P2_SSI_Set_Clock(1,50)    'CLK (clock rate) = 1 MHz
  P2_SSI_Set_Delay(1,1,250)'Waiting delay decoder: 5µs
  P2_SSI_Set_Bits(1,1,23)   'Amount of bits=23 (decoder 1)
  P2_SSI_Mode(1,1)          'Set continuous-mode (decoder 1)

Event:
  Par_1 = P2_SSI_Read(1,1) 'Read out and display position
                           'value (decoder 1)
  REM If you have an encoder with Gray-code:
  m = 0                    'delete value of the last conversion
  y = 0                    ' -"-
  For n = 1 To 32          'Check all 32 possible bits
    m = (Shift_Right(Par_1,(32 - n)) And 1) XOr m
    y = (Shift_Left(m,(32 - n))) Or y
  Next n
  Par_9 = y                'The result of the Gray/binary
                           'conversion in Par_9
```

P2_SSI_Read2

P2_SSI_Read2 returns the last saved counter values of both SSI counters on the specified module.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = P2_SSI_Read2(module,array[],index)
```

Parameters

module	Specified module address (1…15).	LONG
array[]	Destination array to store counter values.	ARRAY
		LONG
index	First element in array[] to be written into.	LONG

Notes

An encoder value is saved when the bits indicated by **P2_SSI_Set_Bits** are read.

Always the amount of bits is returned that is set before by **P2_SSI_Set_Bits**, even if this does not correspond to the resolution of the encoder.
In this case the returned counter value depends on the encoder (see documentation of the manufacturer). Normally there are the following rules:

- If the encoder has a higher resolution, its exceeding least-significant bits are not used.
- If the encoder has a lower resolution as indicated, a 0 (zero) is read for each missing most-significant bit.

See also

P2_SSI_Mode, P2_SSI_Read, P2_SSI_Set_Bits, P2_SSI_Set_Clock, P2_SSI_Set_Delay, P2_SSI_Start, P2_SSI_Status

Valid for

CNT-D Rev. E

Example

```
#Include ADwinPro_All.Inc
Dim Data_1[2000] As Long

Init:
  P2_SSI_Set_Clock(1,50)    'CLK (clock rate) = 1 MHz
  P2_SSI_Set_Delay(1,1,250)'Waiting delay decoder 1: 5µs
  P2_SSI_Set_Delay(1,2,500)'Waiting delay decoder 2: 10µs
  P2_SSI_Set_Bits(1,1,10)   '10 bits for decoder 1
  P2_SSI_Set_Bits(1,2,25)   '25 bits for decoder 2
  P2_SSI_Mode(1,3)          'Set continuous-mode (both decoders)
  Par_1 = 0

Event:
  Inc Par_1
  If (Par_1 > 1000) Then Par_1 = 1
  P2_SSI_Read2(1,Data_1,Par_1*2)'Read both position values
```

P2_SSI_Set_Bits

P2_SSI_Set_Bits sets for an SSI counter on the specified module the amount of bits which generate a complete encoder value.

The number of bits should be similar to the resolution of the encoder.

Syntax

```
#Include ADwinPro_All.Inc
P2_SSI_Set_Bits(module,dcdr_no,bit_no)
```

Parameters

module	Specified module address (1…15).	LONG
dcdr_no	Number (1, 2) of the SSI decoder whose resolution is to be set.	LONG
bit_no	Amount of bits (1…32) to be read for the encoder value (corresponds to encoder resolution).	LONG

Notes

The resolution (amount of bits) of the SSI encoder should be similar to the amount of transferred bits.

It is always expected to get that certain amount of bits for an encoder value that was indicated before by **P2_SSI_Set_Bits**, even if this does not correspond to the resolution of the encoder.
In this case the returned counter value depends on the encoder (see documentation of the manufacturer). Normally there are the following rules:

- If the encoder has a higher resolution, its exceeding least-significant bits are not used.
- If the encoder has a lower resolution as indicated, a 0 (zero) is read for each missing most-significant bit.

See also

P2_SSI_Mode, P2_SSI_Read, P2_SSI_Read2, P2_SSI_Set_Clock, P2_SSI_Set_Delay, P2_SSI_Start, P2_SSI_Status

Valid for

CNT-D Rev. E, MIO-4-ET1 Rev. E, MIO-D12 Rev. E

Example

```
#Include ADwinPro_All.Inc

Init:
  P2_SSI_Set_Clock(1,50)    'CLK (clock rate) = 1 MHz
  P2_SSI_Set_Delay(1,1,250)'Waiting delay decoder 1: 5µs
  P2_SSI_Set_Delay(1,2,500)'Waiting delay decoder 2: 10µs
  P2_SSI_Mode(1,3)          'Set continuous-mode (both decoders)
  P2_SSI_Set_Bits(1,1,10)   '10 bits for decoder 1
  P2_SSI_Set_Bits(1,2,25)   '25 bits for decoder 2

Event:
  Par_1 = P2_SSI_Read(1,1) 'Read position value decoder 1
  Par_2 = P2_SSI_Read(1,2) 'Read position value decoder 2
```

P2_SSI_Set_Clock

P2_SSI_Set_Clock sets the clock rate (approx. 6.1 kHz to 12.5 MHz) on the specified module, with which the decoder is clocked.

Syntax

```
#Include ADwinPro_All.Inc
P2_SSI_Set_Clock(module,prescale)
```

Parameters

module	Specified module address (1…15).	LONG
prescale	scale factor (2…4095) for setting the clock rate according to the equation: Clock rate = 25 MHz / prescale.	LONG

Notes

The setting of the clock rate is always identical for both encoders, which are connected to the module, and cannot be set separately. If necessary, the clock has to consider the clock rate of the slowest encoder.

After start-up of the module the default scale factor of 100 is used, corresponding to 250 kHz.

For scale factors > 255 only the least significant 12 bits are used as scale factor.

The possible clock frequency depends on the length of the cable, cable type, and the send and receive components of the encoder or decoder. Basically the following rule applies: The higher the clock frequency the shorter the cable length.

See also

P2_SSI_Mode, P2_SSI_Read, P2_SSI_Read2, P2_SSI_Set_Bits, P2_SSI_Set_Delay, P2_SSI_Start, P2_SSI_Status

Valid for

CNT-D Rev. E, MIO-4-ET1 Rev. E, MIO-D12 Rev. E

Example

```
#Include ADwinPro_All.Inc

Init:
  P2_SSI_Set_Clock(1,10)    'CLK (clock rate) = 2.5 MHz
  P2_SSI_Set_Delay(1,1,250)'Waiting delay decoder 1: 5µs
  P2_SSI_Set_Delay(1,2,500)'Waiting delay decoder 2: 10µs
  P2_SSI_Mode(1,3)          'Set continuous-mode
                            '(for both decoders)
  P2_SSI_Set_Bits(1,1,10)   'Amount of bits=10 (decoder 1)
  P2_SSI_Set_Bits(1,2,25)   'Amount of bits=25 (decoder 2)

Event:
  Par_1 = P2_SSI_Read(1,1) 'Read position value decoder 1
  Par_2 = P2_SSI_Read(1,2) 'Read position value decoder 2
```

P2_SSI_Set_Delay

P2_SSI_Set_Delay sets the waiting time between reading two encoder values for one SSI-decoder on the specified module.

Syntax

```
#Include ADwinPro_All.Inc
P2_SSI_Set_Delay(module,dcdr_no,delay)
```

Parameters

module	Specified module address (1…15).	LONG
dcdr_no	Number (1, 2) of the SSI decoder whose waiting time is to be set.	LONG
delay	Waiting time (1…65535) in units of 20ns; the selectable range is 20ns…1310.7µs	LONG

Notes

The waiting time delay starts after an encoder value is read completely and ends when the next encoder value starts being read.

After start-up of the module the default value of 1250 is used, corresponding to 25µs.

See also

P2_SSI_Mode, P2_SSI_Read, P2_SSI_Read2, P2_SSI_Set_Bits, P2_SSI_Set_Clock, P2_SSI_Start, P2_SSI_Status

Valid for

CNT-D Rev. E, MIO-4-ET1 Rev. E, MIO-D12 Rev. E

Example

```
#Include ADwinPro_All.Inc

Init:
  P2_SSI_Set_Clock(1,50)    'CLK (clock rate) = 1 MHz
  P2_SSI_Set_Delay(1,1,400)'waiting time 8µs for decoder 1
  P2_SSI_Set_Delay(1,2,200)'waiting time 4µs for decoder 2
  P2_SSI_Set_Bits(1,1,10)   '10 bits for decoder 1
  P2_SSI_Set_Bits(1,2,25)   '25 bits for decoder 2
  P2_SSI_Mode(1,3)          'Set continuous-mode (for both
                            'decoders)

Event:
  Par_1 = P2_SSI_Read(1,1) 'Read position value decoder 1
  Par_2 = P2_SSI_Read(1,2) 'Read position value decoder 2
```

P2_SSI_Start

P2_SSI_Start starts the reading of one or both SSI decoders on the specified module (only in mode "single shot").

Syntax

```
#Include ADwinPro_All.Inc
P2_SSI_Start(module,pattern)
```

Parameters

module	Specified module address (1…15).	LONG
pattern	Bit pattern for selecting the SSI decoders which are to be started: Bit = 0: No function. Bit = 1: Start reading of the SSI decoder.	LONG

Bit no.	31:2	1	0
SSI decoder	–	2	1

Notes

In the continous mode this instruction has no function, because the encoder values are nevertheless read out continously.

An encoder value will be saved only when the amount of bits is read which is set by **P2_SSI_Set_Bits**.

A complete encoder value is always transferred, even if the operation mode is changing meanwhile.

See also

P2_SSI_Mode, P2_SSI_Read, P2_SSI_Read2, P2_SSI_Set_Bits, P2_SSI_Set_Clock, P2_SSI_Set_Delay, P2_SSI_Status

Valid for

CNT-D Rev. E, MIO-4-ET1 Rev. E, MIO-D12 Rev. E

Example

```
#Include ADwinPro_All.Inc

Init:
  P2_SSI_Set_Clock(1,250)   'CLK (clock rate) = 200 kHz
  P2_SSI_Set_Delay(1,1,250)'Waiting delay decoder 1: 5µs
  P2_SSI_Set_Delay(1,2,500)'Waiting delay decoder 2: 10µs
  P2_SSI_Mode(1,0)          'Set single shot-mode
                            '(both counters)
  P2_SSI_Set_Bits(1,1,23)   'Amount of bits=23 (decoder 1)
  P2_SSI_Set_Bits(1,2,23)   'Amount of bits=23 (decoder 2)

Event:
  P2_SSI_Start(1,3)         'Read position value of decoders 1 & 2
  Do                        'for decoder 1:
  Until (P2_SSI_Status(1,1) = 0)
  REM If position value is read completely, then …
  Par_1 = P2_SSI_Read(1,1) 'read out and display position value
  Do                        'For decoder 2:
  Until (P2_SSI_Status(1,2) = 0)
  REM If position value is read completely, then …
  Par_1 = P2_SSI_Read(1,2) 'read out and display position value
```

P2_SSI_Status

P2_SSI_Status returns the current read-status on the specified module for a specified decoder.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = P2_SSI_Status(module,dcdr_no)
```

Parameters

module	Specified module address (1…15).	LONG
dcdr_no	Number (1, 2) of the SSI decoder whose status is to be queried.	LONG
ret_val	Read-status of the decoder: 0: Decoder is ready, that is a complete value was has been read. 1: Decoder is reading an encoder value.	LONG

Notes

Use the status query only in the SSI mode "single shot". In the mode "continous" querying the status is not useful.

See also

P2_SSI_Mode, P2_SSI_Read, P2_SSI_Read2, P2_SSI_Set_Bits, P2_SSI_Set_Clock, P2_SSI_Set_Delay, P2_SSI_Start

Valid for

CNT-D Rev. E, MIO-4-ET1 Rev. E, MIO-D12 Rev. E

Example

```
#Include ADwinPro_All.Inc

Init:
  P2_SSI_Set_Clock(1,250)   'CLK (clock rate) = 200 kHz
  P2_SSI_Mode(1,0)          'Set single shot-mode
                            '(both counters)
  P2_SSI_Set_Bits(1,1,23)   'Amount of bits=23 (decoder 1)
  P2_SSI_Set_Bits(1,2,23)   'Amount of bits=23 (decoder 2)

Event:
  P2_SSI_Start(1,3)         'Read position value of decoders 1 & 2
  Do                        'For decoder 1:
  Until (P2_SSI_Status(1,1) = 0)
  REM If position value is read completely, then …
  Par_1 = P2_SSI_Read(1,1) 'Read out and display position value
  Do                        'For decoder 2:
  Until (P2_SSI_Status(1,2) = 0)
  REM If position value is read completely, then …
  Par_1 = P2_SSI_Read(1,2) 'Read out and display position value
```

3.8 Pro II: PWM Output Modules

This section describes instructions which apply to Pro II modules with PWM outputs:

– P2_PWM_Enable (page 215)
– P2_PWM_Get_Status (page 216)
– P2_PWM_Init (page 217)
– P2_PWM_Latch (page 219)
– P2_PWM_Reset (page 220)
– P2_PWM_Standby_Value (page 221)
– P2_PWM_Write_Latch (page 222)
– P2_PWM_Write_Latch_Block (page 223)

In the Instruction List sorted by Module Types (annex A.2) you will find which of the functions corresponds to the *ADwin-Pro II* modules.

P2_PWM_Enable

P2_PWM_Enable enables or disables one or more PWM outputs.

Syntax

```
#Include ADwinPro_All.Inc
P2_PWM_Enable(module,pattern)
```

Parameters

module	Specified module address (1…15).	LONG
pattern	Bit pattern for selection of PWM outputs. Bit = 0: Disable PWM output. Bit = 1: Enable PWM output.	LONG

Bit no.	31…16	15	14	…	1	0
PWM output	–	16	15	…	2	1

Notes

Before enabling a PWM output, you have to–especially after power-up of the hardware–set the defaults with **P2_PWM_Init** as well as set frequency and duty cycle for output with **P2_PWM_Write_Latch** and **P2_PWM_Latch**.

The time, when the PWM outputs are disabled–at once or after the next end of period–depends on the setting which was done with **P2_PWM_Init** (parameter mode).

See also

P2_PWM_Get_Status, P2_PWM_Init, P2_PWM_Latch, P2_PWM_Reset, P2_PWM_Standby_Value, P2_PWM_Write_Latch, P2_PWM_Write_Latch_Block

Valid for

PWM-16(-I) Rev. E

Example

see P2_PWM_Init (page 217)

P2_PWM_Get_Status

P2_PWM_Get_Status returns the operation status of all PWM outputs.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = P2_PWM_Get_Status(module)
```

Parameters

module	Specified module address (1…15).	LONG
ret_val	Bit pattern with status bits of all PWM outputs. Bit = 0: PWM output is disabled. Bit = 1: PWM output is enabled.	LONG

Bit no.	31…16	15	14	…	1	0
PWM output	–	16	15	…	2	1

Notes

- / -

See also

P2_PWM_Enable, P2_PWM_Init, P2_PWM_Latch, P2_PWM_Reset, P2_PWM_Standby_Value, P2_PWM_Write_Latch

Valid for

PWM-16(-I) Rev. E

Example

- / -

P2_PWM_Init

P2_PWM_Init sets the defaults for one PWM output.

Syntax

```
#Include ADwinPro_All.Inc
P2_PWM_Init(module, pwm_output, startdelay,
    startvalue, mode, count)
```

Parameters

module	Specified module address (1…15).	LONG
pwm_output	Number (1…16) of PWM output.	LONG
startdelay	Start delay in units of 10ns.	LONG
startvalue	Start level of PWM output: 0: TTL-level low. 1: TTL-level high.	LONG
mode	Operating mode of PWM output as bit pattern (bits 0…2 only). Bit 0: Moment to take over a new PW frequency: • Bit = 0: Take over at end of period. • Bit = 1: Take over immediately. Bit 1: Number of pulses: • Bit = 0: infinite number of periods. • Bit = 1: number of periods is count. Bit 2: Moment to stop after stop instruction: • Bit = 0: Stop at end of period. • Bit = 1: Stop immediately.	LONG
count	Number of periods (1…32768), which are processed during an output cycle. Only relevant, if mode, bit 1 = 1.	LONG

Notes

The defaults get active as soon as the PWM outputs are enabled with **P2_PWM_Enable**.

A change of defaults during PWM output is not possible. Instead, the PWM outputs must be stopped with **P2_PWM_Reset** or disabled with **P2_PWM_Enable** in order to change defaults. Afterwards the PWM outputs are enabled again.

See also

P2_PWM_Enable, P2_PWM_Get_Status, P2_PWM_Latch, P2_PWM_Reset, P2_PWM_Standby_Value, P2_PWM_Write_Latch

Valid for

PWM-16(-I) Rev. E

Example

```
#Include ADwinPro_All.inc
#Define module 4
#Define freq1 FPar_1
#Define freq2 FPar_2
#Define pw1 FPar_3
#Define pw2 FPar_4
Dim channel As Long

Init:
  freq1 = 1000                  '1000 Hz
  freq2 = 2000                  '2000 Hz
  pw1 = 50                      '50 %
  pw2 = 70                      '70 %
  P2_PWM_Reset(module,011B) 'stop channels 1 and 2

  For channel = 1 To 2
    P2_PWM_Init(module,channel,0,0,0,0)
  Next

  P2_PWM_Write_Latch(module,1,pw1,freq1)
  P2_PWM_Write_Latch(module,2,pw2,freq2)
  P2_PWM_Latch(module,11B)
  P2_PWM_Enable(module,011B)'start output

Event:
  P2_PWM_Write_Latch(module,1,pw1,freq1)
  P2_PWM_Write_Latch(module,2,pw2,freq2)

  P2_PWM_Latch(module,11B)
```

P2_PWM_Latch

P2_PWM_Latch enables frequency and duty cycle of one or more PWM outputs to be output.

Syntax

```
#Include ADwinPro_All.Inc
P2_PWM_Latch(module,pattern)
```

Parameters

module	Specified module address (1…15).	LONG
pattern	Bit pattern to select PWM outputs: Bit = 0: No influence. Bit = 1: latch = enable for output.	LONG

Bit no.	31…16	15	14	…	1	0
PWM output	–	16	15	…	2	1

Notes

P2_PWM_Write_Latch writes frequency and duty cycle into the latch register. Only when **P2_PWM_Latch** is processed the latch values are started to be output.

The time, when the output of the new values starts–at once or after the next end of period–depends on the setting which was done with **P2_PWM_Init** (parameter mode).

You can output latch values synchronously with actions on other modules with **P2_Sync_All**.

See also

P2_PWM_Enable, P2_PWM_Get_Status, P2_PWM_Init, P2_PWM_Reset, P2_PWM_Standby_Value, P2_PWM_Write_Latch, P2_PWM_Write_Latch_Block, P2_Sync_All

Valid for

PWM-16(-I) Rev. E

Example

see P2_PWM_Init (page 217)

P2_PWM_Reset

P2_PWM_Reset stops the output of one or more PWM outputs immediately..

Syntax

```
#Include ADwinPro_All.Inc
P2_PWM_Reset(module,pattern)
```

Parameters

module	Specified module address (1…15).	LONG
pattern	Bit pattern to select the PWM outputs: Bit = 0: No influence Bit = 1: Stop output immediately.	LONG

Bit no.	31…16	15	14	…	1	0
PWM output	–	16	15	…	2	1

Notes

The output will be stopped immediately even when **P2_PWM_Init** has set a different stop mode.

See also

P2_PWM_Enable, P2_PWM_Get_Status, P2_PWM_Init, P2_PWM_Latch, P2_PWM_Standby_Value, P2_PWM_Write_Latch

Valid for

PWM-16(-I) Rev. E

Example

see P2_PWM_Init (page 217)

P2_PWM_Standby_Value

P2_PWM_Standby_Value sets the default TTL levels for all PWM outputs.

Syntax

```
#Include ADwinPro_All.Inc
P2_PWM_Standby_Value(module,pattern)
```

Parameters

module	Specified module address (1…15).	LONG
pattern	Bit pattern to select the default TTL level of the PWM outputs: Bit = 0: TTL-level low Bit = 1: TTL-level high	LONG

Bit no.	31…16	15	14	…	1	0
PWM output	–	16	15	…	2	1

Notes

Using **P2_PWM_Standby_Value**, PWM outputs may be used as simple TTL outputs.

If the PWM output is disabled with **P2_PWM_Enable**, the output is set to default level from pattern. The default level will also be set after the PWM output has stopped.
After power-up the outputs are set to TTL-level low.

See also

P2_PWM_Enable, P2_PWM_Get_Status, P2_PWM_Init, P2_PWM_Latch, P2_PWM_Reset, P2_PWM_Write_Latch

Valid for

PWM-16(-I) Rev. E

Example

- / -

P2_PWM_Write_Latch

P2_PWM_Write_Latch writes frequency and duty cycle into the latch register.

Syntax

```
#Include ADwinPro_All.Inc
P2_PWM_Write_Latch(module,pwm_output, dutycycle,
    frequency)
```

Parameters

module	Specified module address (1…15).	LONG
pwm_output	Number (1…16) of PWM output.	LONG
dutycycle	Duty cycle / inverse duty cycle in percent between 0.0 and 100.0 (do not use 0.0 or 100.0).	FLOAT
frequency	Frequency in Hertz: 0.025 Hz …50 MHz.	FLOAT

Notes

The value of dutycycle depends on the setting of the parameter startvalue from the instruction **PWM_Init:**

- startvalue = 1: Set dutycycle to the value of the duty cycle.
- startvalue = 0: Set dutycycle to the "inverse duty cycle": dutycycle = 100% - duty cycle

The highest output frequency where the duty cycle can be still defined in 1%-steps, is 1000 kHz.

See also

P2_PWM_Enable, P2_PWM_Get_Status, P2_PWM_Init, P2_PWM_Latch, P2_PWM_Reset, P2_PWM_Standby_Value

Valid for

PWM-16(-I) Rev. E

Example

see P2_PWM_Init (page 217)

P2_PWM_Write_Latch_Block

P2_PWM_Write_Latch_Block writes frequency and duty cycle for several PWM outputs into the latch registers.

Syntax

```
#Include ADwinPRO_ALL.inc
P2_PWM_Write_Latch_Block(module, dutycycle[],
    frequency[], channel_count)
```

Parameters

module	Specified module address (1…15).	LONG
dutycycle[]	Duty cycle / inverse duty cycle in percent between 0.0 and 100.0 (do not use 0.0 or 100.0).	FLOAT
frequency[]	Frequency in Hertz: 0.025 Hz …50 MHz.	FLOAT
channel_count	Number (1…16) of output channels, where output data is to be set.	LONG

Notes

The value of dutycycle depends on the setting of the parameter startvalue from the instruction **PWM_Init:**

- startvalue = 1: Set dutycycle to the value of the duty cycle.
- startvalue = 0: Set dutycycle to the "inverse duty cycle": dutycycle = 100% - duty cycle

The output data are set for the PWM outputs 1...channel_count.

Using **P2_PWM_Write_Latch_Block**, frequency and duty cycle are only written into the latch registers. **P2_PWM_Latch** has to be used to activate the values for PWM output.

The highest output frequency where the duty cycle can be still defined in 1%-steps, is 1000 kHz.

See also

P2_PWM_Enable, P2_PWM_Get_Status, P2_PWM_Init, P2_PWM_Latch, P2_PWM_Reset, P2_PWM_Standby_Value

Valid for

PWM-16(-I) Rev. E

Example

```
#Include ADwinPro_All.inc
#Define module 4
#Define freq Data_1
#Define pw Data_2
Dim freq[16] As Float
Dim pw[16] As Float
Dim channel As Long

Init:
  For channel = 1 To 16
    freq[channel] = 1000 * channel 'channel 1: 1 kHz, channel
16: 16 KHz
    pw[channel] = 50           'all channels 50 %
  Next
  P2_PWM_Reset(module,0FFFFh)'stop all channels
  For channel = 1 To 16
    P2_PWM_Init(module,channel,0,0,0,0)
  Next
  P2_PWM_Write_Latch_Block(module, pw, freq, 3)
  P2_PWM_Latch(module,0FFFFh)
  P2_PWM_Enable(module,0FFFFh)'start output

Event:
  P2_PWM_Write_Latch_Block(module, pw, freq, 3)
  P2_PWM_Latch(module,11b)
```

3.9 Pro II: Temperature Measuring Modules

This section describes instructions for Pro II modules for temperature measurement:

– P2_RTD_Channel_Config (page 226)
– P2_RTD_Config (page 228)
– P2_RTD_Convert (page 229)
– P2_RTD_Read (page 230)
– P2_RTD_Read8 (page 231)
– P2_RTD_Start (page 232)
– P2_RTD_Status (page 234)
– P2_TC_Latch (page 235)
– P2_TC_Read_Latch (page 236)
– P2_TC_Read_Latch4 (page 238)
– P2_TC_Read_Latch8 (page 240)
– P2_TC_Set_Rate (page 242)

In the Instruction List sorted by Module Types (annex A.2) you will find which of the functions corresponds to the *ADwin-Pro II* modules.

P2_RTD_ Channel_Config

P2_RTD_Channel_Config sets the temperature measuring mode for a certain channel on the specified module.

Syntax

```
#Include ADwinPro_All.inc
P2_RTD_Channel_Config(module, channel, active, type,
    element, filter, sample_period)
```

Parameters

module	Specified module address (1…15).	LONG
channel	Number (1…8) of the measuring channel.	LONG
active	Activation of the measuring channel: 0: Channel is disabled. 1: Channel is enabled.	LONG
type	Measuring method (0…2): 0: 2 wire 1: 4 wire 2: 3 wire	LONG
element	Type (0…2) of thermo couple: 0: PT100 1: PT500 2: PT1000 3: Ni100	LONG
filter	Filter quality (0…7); in order to filter periodic disturbances, a number of measurements is averaged to a measuring value: 0: 1 measurement (filter off). 1: 2 measurements (= 2^1). 2: 4 measurements (= 2^2). 3: 8 measurements (= 2^3). 4: 16 measurements (= 2^4). 5: 32 measurements (= 2^5). 6: 64 measurements (= 2^6). 7: 128 measurements (= 2^7).	LONG
sample_ period	Sampling interval in microseconds (3…65535) between each 2 measurements for a measurement value (which is not the interval between 2 measurement values).	LONG

Notes

After power-up of the hardware all measurement channels are disabled. Enabled measuring channels are processed with ascending channel number in the measuring cycle.

You may only enable those channels, which are connected to a temperature sensor. Otherweise there may be disturbances on the other channels–being connected to temperature sensors–which falsify the measurement values.

A measuring cycle comprises all enabled measuring channels. With "continuous" mode you can even enable or disable measuring channels during a measuring cycle.

The measuring methods are described in the hardware manual. 4 wire measurement is recommended because it is the most precise method.

A high filter quality `filter` will raise the accuray of the measurement value, but also the duration to provide a measurement value.
Using the fitting value for the sampling interval `sample_period` you can optimize the filter for a specific disturbance frequency. Calculate the sampling interval (in microseconds) as follows:

$$\text{sample_period} = 10^6 / (\text{frequency} \cdot 2^{\text{filter}})$$

By setting the sampling interval all measurements are equidistantly arranged along the period duration of the disturbance frequency, which therefore will be filtered from the measurement value.

The duration T for one measurement value (of a 2 or 4 wire measurement) is T = `sample_period` x 2^`filter`.
The duration T of a 3 wire measurement takes double as long, since it requires double the number of measurements.

See also

P2_RTD_Config, P2_RTD_Convert, P2_RTD_Read, P2_RTD_Read8, P2_RTD_Start, P2_RTD_Status

Valid for

RTD-8 Rev. E

Example

see P2_RTD_Start

P2_RTD_Config

P2_RTD_Config initializes the temperature measurement on the specified module.

Syntax

```
#Include ADwinPro_All.inc
P2_RTD_Config(module, mode, muxtime)
```

Parameters

module	Specified module address (1…15).	LONG
mode	Operating mode of temperature measurement: 0: Mode "single shot", single measurement cycle. 1: Mode "continuous", continuous measurement cycle.	LONG
muxtime	Settling time after switching to a different measuring channel: 0: Standard setting (5000 = 100µs). 125…2^31: Time in units of 20ns.	LONG

Notes

You configure the operating mode of each temperature channel separately using **P2_RTD_Config_Channel**.

A measuring cycle comprises all enabled measuring channels. With "continuous" mode you can even enable or disable measuring channels during a measuring cycle.

The longer the measuring cable to the temperature sensor the higher you should set the settling time.

The duration T_{total} of a measurement cycle is the sum of the measuring duration of the enabled temperature channels and the settling times:

$$T_{total} = \sum_{1}^{channels} (T_{channel} + settling\ time)$$

See also

P2_RTD_Channel_Config, P2_RTD_Convert, P2_RTD_Read, P2_RTD_Read8, P2_RTD_Start, P2_RTD_Status

Valid for

RTD-8 Rev. E

Example

see P2_RTD_Start

P2_RTD_Convert

P2_RTD_Convert calculates the resistance or the temperature in degrees Celsius or Fahrenheit from the measured digital value of a temperature sensor.

Syntax

```
#Include ADwinPro_All.inc
ret_val = P2_RTD_Convert(dig_val, element, ret_type)
```

Parameters

dig_val	Digital value (24 bit).	LONG
element	Type (0…2) of thermo couple: 0: PT100 1: PT500 2: PT1000 3: Ni100	LONG
ret_type	Type of return value: 0: Resistance in Ohm. 1: Temperature in degrees Celsius. 2: Temperature in degrees Fahrenheit.	LONG
ret_val	Resistance in Ohm or temperature in degrees Celsius or in degrees Fahrenheit.	FLOAT

Notes

The temperature values in °C and °F apply only for standard sensors according to the norms IEC 751 (Pt) and IEC 43760 (Ni).

See also

P2_RTD_Channel_Config, P2_RTD_Config, P2_RTD_Read, P2_RTD_Read8, P2_RTD_Start, P2_RTD_Status

Valid for

RTD-8 Rev. E

Example

see P2_RTD_Start

P2_RTD_Read

P2_RTD_Read returns the current digital measurement value of a temperature sensor at the specified channel on the module.

Syntax

```
#Include ADwinPro_All.inc
ret_val = P2_RTD_Read(module, channel)
```

Parameters

module	Specified module address (1…15).	LONG
channel	Number (1…8) of the measuring channel.	LONG
ret_val	Current temperature value (24 Bit) in digits.	LONG

Notes

With "single shot" mode a measurement value may only be read, if the measurement cycle is completed (see **P2_RTD_Status**).

See also

P2_RTD_Channel_Config, P2_RTD_Config, P2_RTD_Convert, P2_RTD_Read8, P2_RTD_Start, P2_RTD_Status

Valid for

RTD-8 Rev. E

Example

- / -

P2_RTD_Read8

P2_RTD_Read8 returns the current digital measurement values of temperature sensors at all channels on the module.

Syntax

```
#Include ADwinPro_All.inc
P2_RTD_Read8(module, array[], index)
```

Parameters

module	Specified module address (1…15).	LONG
array[]	Destination array, where measurement values are stored.	ARRAY LONG
index	Index of the array element where the first measurement value is stored.	LONG

Notes

8 measurement values will be stored in the destination array even when you have enabled less than 8 measurement channels. The measurement values are stored with ascending channel number.

See also

P2_RTD_Channel_Config, P2_RTD_Config, P2_RTD_Convert, P2_RTD_Read, P2_RTD_Start, P2_RTD_Status

Valid for

RTD-8 Rev. E

Example

see P2_RTD_Start

P2_RTD_Start

P2_RTD_Start starts the temperature measurement cycle on all specified modules at the same time.

Syntax

```
#Include ADwinPro_All.inc
P2_RTD_Start(module_pattern)
```

Parameters

module_pattern	Bit pattern to select the module addresses, where a measurement cycle is started: Bit = 0: Ignore module address. Bit = 1: Select module address.	LONG

Bits in pattern	31:15	14	13	…	01	00
Module address	–	15	14	…	2	1

Notes

Before starting a measurement cycle you have to set the operating modes of the modules with **P2_RTD_Config** and the operating modes of each channel separately with **P2_RTD_Config_Channel**.

If a module without temperature measurement is selected, the instruction may cause unpredictable consequences.

See also

P2_RTD_Channel_Config, P2_RTD_Config, P2_RTD_Convert, P2_RTD_Read, P2_RTD_Read8, P2_RTD_Status

Valid for

RTD-8 Rev. E

Example

```
#Include ADwinPro_All.inc
#Define module 2

Dim values24[8] As Long
Dim channel As Long
Dim i As Long
Dim run_state As Long
Dim status As Long

Init:
  P2_RTD_Config(module, 0, 0) 'single shot mode
  Rem use channels 1…6
  For i = 1 To 6
    Rem do channel settings: 4 wire, PT100, 50 Hz filter
    P2_RTD_Channel_Config(module, i, 1, 1, 0, 7, 156)
  Next
  Processdelay=50000000
  run_state = 0

Event:
  SelectCase run_state
    Case 0
      Rem start measurement cycle
      P2_RTD_start(Shift_Left(1, module-1))
      run_state = 1
    Case 1
      Rem check for end of measurement cycle
      status = P2_RTD_status(module)
      If (status = 0) Then run_state = 2
    Case 2
      Rem read measured values and prepare start of next cycle
      P2_RTD_read8(module, values24, 1) 'messwerte  lesen
      For i = 1 To 6
        Rem convert measurement values
        fpar[i] = P2_RTD_convert(values24[i], 0, 1)
      Next
      run_state = 0
  EndSelect
```

P2_RTD_Status

P2_RTD_Status returns the status of temperature measurement cycle in "single shot" mode on the specified module.

Syntax

```
#Include ADwinPro_All.inc
ret_val = P2_RTD_Status(module)
```

Parameters

module	Specified module address (1…15).	LONG
ret_val	Status of measurement cycle: 0: Measurement cycle is completed. 1: Measurement cycle is being processed.	LONG

Notes

The instruction **P2_RTD_Status** is only useful with operating mode "single shot".

See also

P2_RTD_Channel_Config, P2_RTD_Config, P2_RTD_Convert, P2_RTD_Read, P2_RTD_Read8, P2_RTD_Start

Valid for

RTD-8 Rev. E

Example

see P2_RTD_Start

P2_TC_Latch

P2_TC_Latch copies the current voltage values at the inputs into latches.

Syntax

```
#Include ADwinPro_All.Inc
P2_TC_Latch(module)
```

Parameters

module	Specified module address (1…15).	LONG

Notes

The values in the latches will be calculated into the desired return value (°C, °F, thermo voltage) when read with **…Read_Latch**.

Copying values to latches can be started synchronously to actions on other modules with **P2_Sync_All**.

See also

P2_TC_Read_Latch, P2_TC_Read_Latch4, P2_TC_Read_Latch8, P2_TC_Set_Rate, P2_Sync_All

Valid for

TC-8-ISO Rev. E

Example

```
#Include ADwinPro_All.Inc

Init:
  Rem Set sampling rate to 27.5 Hz
  P2_TC_Set_Rate(1,8)

Event:
  Rem copy values to latches
  P2_TC_Latch(1)
  Rem Read temperature from channel 5, thermo couple K in °C
  FPar_1 = P2_TC_Read_Latch(1,5,1,1)
```

P2_TC_Read_Latch

P2_TC_Read_Latch returns the thermo voltage (µV) or temperature (°C / °F) of the selected channel of the module.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = P2_TC_Read_Latch(module, channel,
    tc_element, ret_type)
```

Parameters

module	Specified module address (1…15).	LONG
channel	Number (1…8) of the channel being read.	LONG
tc_element	Type of thermo couple: -1: no conversion, thus current thermo voltage without cold junction compensation. 0: Thermo couple type J 1: Thermo couple type K 2: Thermo couple type N 3: Thermo couple type S 4: Thermo couple type T 5: Thermo couple type R 6: Thermo couple type E 7: Thermo couple type B	LONG
ret_type	Type of return value ret_val: 0: Thermo voltage in µV; with tc_element = -1 without cold junction compensation. 1: Temperature in °C. 2: Temperature in °F.	LONG
ret_val	Measurement value of the channel, depending on ret_type and tc_element.	FLOAT

Notes

The module reglarly samples the channels (setting the sample rate see **P2_TC_Set_Rate**). The instruction **P2_TC_Read_Latch** returns the most recently sampled value.

If you want to read several channels using the same thermo couple type, the instructions **P2_TC_Read_Latch4** and **P2_TC_Read_Latch8** are a lot faster.

We recommend to use a constant for tc_element. Using a variable the instruction requires much more program memory.

If you set tc_element = -1, the thermo voltage value will be left unchanged, i.e. neither a cold junction correction is performed nor the thermo couple characteristics are considered.
The value range of ret_val will then be -80000µV…80000µV.

The thermoelectric voltage and the temperature values in °C and °F apply only for standard thermocouples according to the norm IEC 584-1. The value ranges are:

Type	Temperature range [°C]	Temperature range [°F]	Thermo voltage [µV]
B	250…1820	482…3329.6	291…13820
E	-200…1000	-328…1832	-8825…76373
J	-210…1200	-346…2192	-8095…69553

Type	Temperature range [°C]	Temperature range [°F]	Thermo voltage [µV]
K	-200…1372	-328…2501.6	-5891…54886
N	-200…1300	-328…2372	-3990…47513
R	-50…1768	-58…3214.4	-226…21101
S	-50…1768	-58…3214.4	-236…18693
T	-200…400	-454…752	-5603…20872

See also

P2_TC_Latch, P2_TC_Read_Latch4, P2_TC_Read_Latch8, P2_TC_Set_Rate

Valid for

TC-8-ISO Rev. E

Example

```
#Include ADwinPro_All.Inc

Init:
  Rem Set sampling rate to 27.5 Hz
  P2_TC_Set_Rate(1,8)

Event:
  Rem copy values to latches
  P2_TC_Latch(1)
  Rem Read temperature from channel 5, thermo couple K in °C
  FPar_1 = P2_TC_Read_Latch(1,5,1,1)
```

P2_TC_Read_Latch4

P2_TC_Read_Latch4 returns the thermo voltage (µV) or temperature (°C / °F) of the channels 1…4 of the module.

Syntax

```
#Include ADwinPro_All.Inc
P2_TC_Read_Latch4(module, tc_element, ret_type,
    array[], array_start_index)
```

Parameters

module	Specified module address (1…15).	LONG
tc_element	Type of thermo couple: -1: no conversion, thus current thermo voltage without cold junction compensation. 0: Thermo couple type J 1: Thermo couple type K 2: Thermo couple type N 3: Thermo couple type S 4: Thermo couple type T 5: Thermo couple type R 6: Thermo couple type E 7: Thermo couple type B	LONG
ret_type	Type of return value ret_val: 0: Thermo voltage in µV; with tc_element = -1 without cold junction compensation. 1: Temperature in °C. 2: Temperature in °F.	LONG
array[]	Destination array to store measurement value of channels 1…4; measurement values depending on ret_type and tc_element.	ARRAY FLOAT
array_start_index	Index of the first array element in array[] being written.	LONG

Notes

The module reglarly samples the channels (setting the sample rate see **P2_TC_Set_Rate**). The instruction **P2_TC_Read_Latch4** returns the most recently sampled values of the channels 1…4.

We recommend to use a constant for tc_element. Using a variable the instruction requires much more program memory.

If you set tc_element = -1, the thermo voltage value will be left unchanged, i. e. neither a cold junction correction is performed nor the thermo couple characteristics are considered.
The value range of ret_val will then be -80000 µV…80000 µV.

The thermoelectric voltage and the temperature values in °C and °F apply only for standard thermocouples according to the norm IEC 584-1. The value ranges are:

Type	Temperature range [°C]	Temperature range [°F]	Thermo voltage [µV]
B	250…1820	482…3329.6	291…13820
E	-200…1000	-328…1832	-8825…76373
J	-210…1200	-346…2192	-8095…69553

Type	Temperature range [°C]	Temperature range [°F]	Thermo voltage [µV]
K	-200…1372	-328…2501.6	-5891…54886
N	-200…1300	-328…2372	-3990…47513
R	-50…1768	-58…3214.4	-226…21101
S	-50…1768	-58…3214.4	-236…18693
T	-200…400	-454…752	-5603…20872

See also

P2_TC_Latch, P2_TC_Read_Latch, P2_TC_Read_Latch8, P2_TC_Set_Rate

Valid for

TC-8-ISO Rev. E

Example

```
#Include ADwinPro_All.Inc
Dim cnt As Long
Dim values[1000] As Float

Init:
  Rem Set sampling rate to 27.5 Hz
  P2_TC_Set_Rate(1,8)
  cnt = 1

Event:
  Rem copy values to latches
  P2_TC_Latch(1)
  Rem Read temperature from channels 1..4, thermo couple J in °F
  P2_TC_Read_Latch4(1,0,2,values,cnt)
  Rem increase counter
  cnt = cnt + 4 : If (cnt > 1000) Then cnt = 1
```

P2_TC_Read_Latch8

P2_TC_Read_Latch8 returns the thermo voltage (µV) or temperature (°C / °F) of the channels 1…8 of the module.

Syntax

```
#Include ADwinPro_All.Inc
P2_TC_Read_Latch8(module, tc_element, ret_type,
    array[], array_start_index)
```

Parameters

module	Specified module address (1…15).	LONG
tc_element	Type of thermo couple: -1: no conversion, thus current thermo voltage without cold junction compensation. 0: Thermo couple type J 1: Thermo couple type K 2: Thermo couple type N 3: Thermo couple type S 4: Thermo couple type T 5: Thermo couple type R 6: Thermo couple type E 7: Thermo couple type B	LONG
ret_type	Type of return value ret_val: 0: Thermo voltage in µV; with tc_element = -1 without cold junction compensation. 1: Temperature in °C. 2: Temperature in °F.	LONG
array[]	Destination array to store measurement value of channels 1…8; measurement values depending on ret_type and tc_element.	ARRAY FLOAT
array_start_index	Index of the first array element in array[] being written.	LONG

Notes

The module reglarly samples the channels (setting the sample rate see **P2_TC_Set_Rate**). The instruction **P2_TC_Read_Latch8** returns the most recently sampled values of the channels 1…8.

We recommend to use a constant for tc_element. Using a variable the instruction requires much more program memory.

If you set tc_element = -1, the thermo voltage value will be left unchanged, i.e. neither a cold junction correction is performed nor the thermo couple characteristics are considered.
The value range of ret_val will then be -80000µV…80000µV.

The thermoelectric voltage and the temperature values in °C and °F apply only for standard thermocouples according to the norm IEC 584-1. The value ranges are:

Type	Temperature range [°C]	Temperature range [°F]	Thermo voltage [µV]
B	250…1820	482…3329.6	291…13820
E	-200…1000	-328…1832	-8825…76373
J	-210…1200	-346…2192	-8095…69553

Type	Temperature range [°C]	Temperature range [°F]	Thermo voltage [µV]
K	-200…1372	-328…2501.6	-5891…54886
N	-200…1300	-328…2372	-3990…47513
R	-50…1768	-58…3214.4	-226…21101
S	-50…1768	-58…3214.4	-236…18693
T	-200…400	-454…752	-5603…20872

See also

P2_TC_Latch, P2_TC_Read_Latch, P2_TC_Read_Latch4, P2_TC_Set_Rate

Valid for

TC-8-ISO Rev. E

Example

```
#Include ADwinPro_All.Inc
Dim cnt As Long
Dim values[1000] As Float

Init:
  Rem Set sampling rate to 27.5 Hz
  P2_TC_Set_Rate(1,8)
  cnt = 1

Event:
  Rem copy values to latches
  P2_TC_Latch(1)
  Rem Read temperature from channels 1..8, thermo couple J in °F
  P2_TC_Read_Latch8(1,0,2,values,cnt)
  Rem increase counter
  cnt = cnt + 8 : If (cnt > 1000) Then cnt = 1
```

P2_TC_Set_Rate

P2_TC_Set_Rate sets the sampling rate of the selected module.

Syntax

```
#Include ADwinPro_All.Inc
P2_TC_Set_Rate(module, rate)
```

Parameters

module	Specified module address (1…15).	LONG
rate	Key figure of the specified sample rate (see table); default: 15.	LONG

Key figure	Sample rate [Hz]	ADC noise [nV]
1	3520	23000
2	1760	3500
3	880	2000
4	440	1400
5	220	1000
6	110	750
7	55	510
8	27.5	375
9	13.75	250
15	6.875	200

Notes

The sample rate is valid for all channels in similar.

A higher sample rate refers to a higher noise signal at the ADC of the channel. The noise signal superposes the sampled signal (see table).

See also

P2_TC_Latch, P2_TC_Read_Latch, P2_TC_Read_Latch4, P2_TC_Read_Latch8

Valid for

TC-8-ISO Rev. E

Example

```
#Include ADwinPro_All.Inc

Init:
  Rem Set sampling rate to 27.5 Hz
  P2_TC_Set_Rate(1,8)
```

3.10 Pro II: CAN bus Modules

This section describes instructions which apply to Pro II CAN bus modules:

– CAN_Msg (page 244)
– P2_CAN_Interrupt_Source (page 246)
– P2_CAN_Set_LED (page 248)
– P2_En_Interrupt (page 249)
– P2_En_Receive (page 251)
– P2_En_Transmit (page 252)
– P2_Get_CAN_Reg (page 253)
– P2_Init_CAN (page 254)
– P2_Read_Msg (page 255)
– P2_Read_Msg_Con (page 257)
– P2_Set_CAN_Baudrate (page 259)
– P2_Set_CAN_Reg (page 263)
– P2_Transmit (page 264)

In the Instruction List sorted by Module Types (annex A.2) you will find which of the functions corresponds to the *ADwin-Pro II* modules.

CAN_Msg

CAN_Msg is a one-dimensional array consisting of 9 elements, where the message objects of the CAN bus are saved during sending and receiving.

Syntax

```
#Include ADwinPro_All.Inc
CAN_Msg[n] = value
```

or

```
ret_val = CAN_Msg[n]
```

Parameters

n	Number of an array element (1… 9).	LONG
value	Expression whose value (0...256) is written into the message object.	LONG
ret_val	Value (0…256), which is read from the message object.	LONG

Notes

The first 8 elements of the array contain the data bytes 1...8 and the 9th array element the amount of valid data bytes of the message. Here a value from 0 to 8 must be entered.

The data bytes use only the bits 7...0 in the array elements, bits 31...8 are ignored.

The values in the array **CAN_Msg**[] must be entered before executing **P2_Transmit**.

See also

P2_En_Receive, P2_En_Transmit, P2_Read_Msg, P2_Transmit

Valid for

CAN-2 Rev. E

Example

```
REM Sends a 32 Bit FLOAT-value (here: Pi) as sequence of
REM 4 bytes in a message object
REM (Receiving of a float value see example at P2_Read_Msg)

#Include ADwinPro_All.Inc
#Define pi 3.14159265
Dim i As Long

Init:
  P2_Init_CAN(1,1)          'Initialize CAN controller 1

  REM Enable message object 6 of controller 1
  REM for sending with the identifier 40 (11 bit)
  P2_En_Transmit(1,1,6,40,0)

  REM Create bit pattern of Pi with data type Long
  Par_1 = Cast_FloatToLong(pi)

  REM divide bit pattern (32 Bit) into 4 bytes
  CAN_Msg[4] = Par_1 And 0FFh 'assign LSB
  For i = 1 To 3
    CAN_Msg[4-i] = Shift_Right(Par_1,8*i) And 0FFh
  Next i
  CAN_Msg[9] = 4            'message length in bytes

Event:
  P2_Transmit(1,1,6)        'Send the message object 6
```

P2_CAN_ Interrupt_Source

P2_CAN_Interrupt_Source returns the CAN channels which have generated an event signal (interrupt).

Syntax

```
#INCLUDE ADwinPro_All.inc
ret_val = P2_CAN_Interrupt_Source(module)
```

Parameters

module	Selected module address (1…15).	LONG
ret_val	Bit pattern which defines the interrupt source.	LONG

Bit no.	31…2	1	0
CAN channel	–	2	1

Notes

The instruction is useful only if generating event signals is enabled with **P2_En_Interrupt**.

P2_CAN_Interrupt_Source runs much faster than reading the interrupt register on a CAN controller.

After an event signal having been generated the message of the generating message object must be read with P2_Read_Msg, thus enabling the message object to generate a new event signal. In the meanwhile the CAN controller ignores incoming messages for this message object.

See also

P2_En_Interrupt, P2_Init_CAN, P2_Read_Msg

Valid for

CAN-2 Rev. E

Example

```
#Include ADwinPRO_ALL.INC

Init:
  P2_Init_CAN(1,1)          'initialize channel 1
  P2_En_Receive(1,1,3,1,0) 'configure msg objects 3 and 15
  P2_En_Receive(1,1,15,385,0) 'for read
  P2_En_Interrupt(1,1,3)    'configure msg objects 3 and 15
  P2_En_Interrupt(1,1,15)   'for interrupt
  P2_Event_Enable(1,1)      'enable event interrupt

Event:
  Par_13 = P2_CAN_Interrupt_Source(1) 'check for interrupt
  If (Par_13 And 01b = 1) Then
    Par_14 = CAN_Interrupt_Msg(1,1) 'get interrupting msg object
    Rem get msg object = enable new interrupt
    Par_15 = P2_Read_Msg(1,1,CAN_Interrupt_Msg(1,1))
  EndIf

Function CAN_Interrupt_Msg(module,channel) As Long
  REM read interrupt register and change value to objekt no.
  CAN_Interrupt_Msg = P2_Get_CAN_Reg(module,channel,5fh)
  If (CAN_Interrupt_Msg = 2) Then
    CAN_Interrupt_Msg = 15
  Else
    CAN_Interrupt_Msg = CAN_Interrupt_Msg - 2
  EndIf
EndFunction
```

The value of the interrupt register **5Fh** refers to a message object according to the following table:

Register value	2	3	4	…	16
No. of message object	15	1	2	…	14

P2_CAN_Set_LED

P2_CAN_Set_LED switches the additional LED of a CAN channel on (with color) or off.

Syntax

```
#Include ADwinPro_All.inc
P2_CAN_Set_LED(module,channel,led_col)
```

Parameters

module	Selected module address (1…15).	LONG
channel	Number (1, 2) of the CAN channel that determines the CAN controller.	LONG
led_col	Status and Farbe of the additional LED: 0: LED off. 1: LED on, color red. 2: LED on, color green. 3: LED on, color orange.	LONG

Notes

- / -

See also

P2_Set_LED

Valid for

CAN-2 Rev. E

Example

```
#Include ADwinPro_All.inc
Init:
  P2_Init_CAN(1,1)          'initialize CAN controller
  P2_CAN_Set_LED(1,1,2)     'switch on channel 1 LED, color green
```

P2_En_Interrupt

P2_En_Interrupt configures a message object of the specified module to generate an event signal (interrupt) when a message arrives.

Syntax

```
#Include ADwinPro_All.inc
P2_En_Interrupt(module,channel,msg_no)
```

Parameters

module	Selected module address (1…15).	LONG
channel	Number (1, 2) of the CAN channel that determines the CAN controller.	LONG
msg_no	Number (1…15) of the message object in the CAN controller.	LONG

Notes

A generated event signal will be forwarded to the processor module only, when the event signal is enabled with **P2_Event_Enable**. The specified message objects must be configured at first before the event signal is enabled at last.

In a system only one event input may be active, in addition to a processor module, that is you have to disable an actually active event input, before you enable the event input of another module.

After an event signal having been generated the message of the generating message object must be read with **P2_Read_Msg**, thus enabling the message object to generate a new event signal. In the meanwhile the CAN controller ignores incoming messages for this message object.

See also

P2_CAN_Interrupt_Source, P2_En_Receive, P2_Event_Enable, P2_Event_Read, P2_Get_CAN_Reg, P2_Init_CAN

Valid for

CAN-2 Rev. E

Example

```
#Include ADwinPro_All.Inc

Init:
  REM Initialize CAN controller 1 on the CAN module 1
  P2_Init_CAN(1,1)
  REM Configure message objects 3 and 15 for read
  P2_En_Receive(1,1,3,1,0)
  P2_En_Receive(1,1,15,385,0)
  REM Configure interrupt for message objects 3 and 15
  P2_En_Interrupt(1,1,3)
  P2_En_Interrupt(1,1,15)
  REM Enable event signal
  P2_Event_Enable(1,1)
Event:
  REM Read interrupt register (see below)
  Par_13 = P2_Get_CAN_Reg(1,1,5Fh)
  REM Convert register value into no. of object
  If (Par_13 = 2) Then
    Par_13 = 15
  Else
    Par_13 = Par_13 - 2
  EndIf
  Rem get msg object = enable new interrupt
  Par_15 = P2_Read_Msg(1,1,Par_13)
```

The value of the interrupt register **5Fh** refers to a message object according to the following table:

Register value	2	3	4	…	16
No. of message object	15	1	2	…	14

P2_En_Receive

P2_En_Receive enables a message object on the specified module to receive messages.

For the message object the CAN channel, the length of the message identifier and the identifier itself are determined.

Syntax

```
#Include ADwinPro_All.inc
P2_En_Receive(module,channel,msg_no,id,id_extend)
```

Parameters

module	Selected module address (1…15).	LONG
channel	Number (1, 2) of the CAN channel that determines the CAN controller.	LONG
msg_no	Number (1…15) of the message object in the CAN controller.	LONG
id	Identifier of the messages which are to be received in this message object (0…2^{11} or 0…2^{29}).	LONG
id_extend	Marker for the length of the identifier: 0: 11 bit identifier. 1: 29 bit identifier.	LONG

Notes

A message object is only able to receive messages from the CAN bus, when it has been enabled before by **P2_En_Receive**.

See also

CAN_Msg, P2_En_Transmit, P2_Init_CAN, P2_Read_Msg, P2_Transmit

Valid for

CAN-2 Rev. E

Example

```
#Include ADwinPro_All.inc

Init:
  REM Initialize CAN controller 1 on CAN module 1
  P2_Init_CAN(1,1)
  REM Enable message object 1 to receive CAN messages
  REM with the 11 bit-identifier 200
  P2_En_Receive(1,1,1,200,0)
```

P2_En_Transmit

P2_En_Transmit enables a message object on the specified module to transmit messages.

The CAN channel, the length of the message identifier and the identifier itself are determined for the message object.

Syntax

```
#Include ADwinPro_All.inc
P2_En_Transmit(module,channel,msg_no,id,id_extend)
```

Parameters

module	Selected module address (1…15).	LONG
channel	Number (1, 2) of the CAN channel that specifies the CAN controller.	LONG
msg_no	Number (1…14) of the message object in the CAN controller.	LONG
id	Identifier of the messages that are sent in this message object (0…2^{11} or 0…2^{29}).	LONG
id_extend	Marker for the length of the identifier: 0: 11 bit identifier. 1: 29 bit identifier.	LONG

Notes

A message object can only transmit messages to the CAN bus when it has been enabled before by **P2_En_Transmit**.

See also

CAN_Msg, P2_En_Receive, P2_Init_CAN, P2_Read_Msg, P2_Transmit

Valid for

CAN-2 Rev. E

Example

```
#Include ADwinPro_All.inc
Init:
  REM Initialize CAN controller 1 on CAN module 1
  P2_Init_CAN(1,1)
  REM Enable message object 6 for sending of CAN messages
  REM with the 11 bit-identifier 40
  P2_En_Transmit(1,1,6,40,0)
```

P2_Get_CAN_Reg

P2_Get_CAN_Reg returns the contents of a specified register on a CAN controller on the specified module.

Syntax

```
#Include ADwinPro_All.inc
ret_val = P2_Get_CAN_Reg(module,channel,regno)
```

Parameters

module	Selected module address (1…15).	LONG
channel	Number (1, 2) of the CAN channel that determines the CAN controller.	LONG
regno	Register number (0…255) of the CAN controller.	LONG
ret_val	Content (0…255) of the CAN controller register.	LONG

Notes

The register number corresponds to the register number of the CAN controller (see data-sheet AN82527 from Intel®), e.g.:

- address **00h**: controll register
- address **01h**: status register
- address **5fh**: interrupt register

See also

P2_En_Interrupt, P2_Init_CAN, P2_Set_CAN_Baudrate, P2_Set_CAN_Reg

Valid for

CAN-2 Rev. E

Example

```
#Include ADwinPro_All.Inc
Init:
  REM Initialize CAN controller 1 on CAN module 1
  P2_Init_CAN(1,1)
  REM Read out the control register of CAN controller 1, module 1
  Par_1 = P2_Get_CAN_Reg(1,1,0)
```

P2_Init_CAN

P2_Init_CAN initializes one of the CAN controllers on the specified module and sets it into an initial status.

Syntax

```
#Include ADwinPro_All.inc
P2_Init_CAN(module,channel)
```

Parameters

module	Selected module address (1…15).	LONG
channel	Number (1, 2) of the CAN channel that determines the CAN controller.	LONG

Notes:

The instruction executes the following actions:

- Reset (hardware reset of the CAN controller).
- All filters are set to "must match".
- Set clockout register to 0 (= external frequency will not be divided).
- Set bus configuraton register to 0
- Set transfer rate for the CAN bus to 1 MBit/s.
- Disable all message objects.

This instruction must be executed at the beginning of the process (if possible in the process sections **LowInit:** or **Init:**) before other instructions access the CAN controller.

With low speed CAN the maximum transfer rate is 125 kBit/s and therefore must be newly set with **P2_Set_CAN_Baudrate**.

See also

P2_En_Receive, P2_En_Transmit, P2_Get_CAN_Reg, P2_Set_CAN_Baudrate, P2_Set_CAN_Reg

Valid for

CAN-2 Rev. E

Example

```
#Include ADwinPro_All.inc
Init:
  REM Initialize CAN controller 1 on CAN module 1
  P2_Init_CAN(1,1)
```

P2_Read_Msg

P2_Read_Msg returns the information if a new message in a message object of one of the CAN controllers on the module has been received.

If yes, the message is copied to the array **CAN_Msg**[] and the identifier is returned.

Syntax

```
#Include ADwinPro_All.inc
ret_val = P2_Read_Msg(module,channel,msg_no)
```

Parameters

module	Selected module address (1…15).	LONG
channel	Number (1, 2) of the CAN channel that determines the CAN controller.	LONG
msg_no	Number (1… 15) of the message object in the CAN controller.	LONG
ret_val	>-1: A new message has arrived, the value is the identifier of the message object. -1: No new message has arrived.	LONG

Notes

To receive a message you have to follow the correct order:

- Once: Enable the message object with **P2_En_Receive** for receiving.
- As often as needed: Check for a received message and save to **CAN_Msg** with **P2_Read_Msg**.

You can read a received message only once.

See also

CAN_Msg, P2_En_Receive, P2_En_Transmit, P2_Init_CAN, P2_Transmit

Valid for

CAN-2 Rev. E

Example

```
REM If a new message with the correct identifier is received
REM the data is read out. The first 4 bytes of the message are
REM combined to a float value of length 32 bit. (Sending a
REM float value see example of P2_Transmit).
#Include ADwinPro_All.Inc
Dim n As Long

Init:
Par_1 = 0
  P2_Init_CAN(1,1)          'Initialize CAN controller 1
  P2_En_Receive(1,1,8,40,0)'Initialize the message object 8
                           'to receive CAN messages with
                           'identifier 40

Event:
  REM If the message is changed, read out the received data
  REM from object 8 and save the identifier to parameter 9.
  REM The data bytes are in the array CAN_MSG[].
  Par_9 = Read_Msg(1,1,8)

  If (Par_9 = 40) Then
    REM New message for message object with the identifier 40
    REM has arrived
    Par_1 = CAN_Msg[1]      'Read out high-byte
    For n = 2 To 4          'Combine with remaining 3 bytes to
     Par_1 = Shift_Left(Par_1,8) + CAN_Msg[n]'a 32-bit value
    Next n
    REM Convert the bit pattern in Par_1 to data type FLOAT and
    REM assign to the variable FPar_1.
    FPar_1 = Cast_LongToFloat(Par_1)
  EndIf
```

P2_Read_Msg_Con

P2_Read_Msg_Con returns the information if a new message in a message object of one of the CAN controllers on the module has been received.

If yes, the message is copied to the array **CAN_Msg**[] and the identifier is returned.

Syntax

```
#Include ADwinPro_All.inc
ret_val = P2_Read_Msg_Con(module,channel,msg_no)
```

Parameters

module	Selected module address (1…15).	LONG
channel	Number (1, 2) of the CAN channel that determines the CAN controller.	LONG
msg_no	Number (1… 15) of the message object in the CAN controller.	LONG
ret_val	≥-1: A new message has arrived, the value is the identifier of the message object. -1: No new message has arrived.	LONG

Notes

Different to **P2_Read_Msg**, **P2_Read_Msg** makes sure the message is consistent: If a new message arrives while reading, the newer message will be returned.

To receive a message you have to follow the correct order:

- Once: Enable the message object with **P2_En_Receive** for receiving.
- As often as needed: Check for a received message and save to **CAN_Msg** with **P2_Read_Msg_Con**.

You can read a received message only once.

See also

CAN_Msg, P2_En_Receive, P2_En_Transmit, P2_Init_CAN, P2_Transmit

Valid for

CAN-2 Rev. E

Example

```
REM If a new message with the correct identifier is received
REM the data is read out. The first 4 bytes of the message are
REM combined to a float value of length 32 bit. (Sending a
REM float value see example of P2_Transmit).
#Include ADwinPro_All.Inc
Dim n As Long

Init:
Par_1 = 0
  P2_Init_CAN(1,1)          'Initialize CAN controller 1
  P2_En_Receive(1,1,8,40,0)'Initialize the message object 8
                            'to receive CAN messages with
                            'identifier 40

Event:
  REM If the message is changed, read out the received data
  REM from object 8 and save the identifier to parameter 9.
  REM The data bytes are in the array CAN_MSG[].
  Par_9 = Read_Msg_Con(1,1,8)

  If (Par_9 = 40) Then
    REM New message for message object with the identifier 40
    REM has arrived
    Par_1 = CAN_Msg[1]      'Read out high-byte
    For n = 2 To 4          'Combine with remaining 3 bytes to
     Par_1 = Shift_Left(Par_1,8) + CAN_Msg[n]'a 32-bit value
    Next n
    REM Convert the bit pattern in Par_1 to data type FLOAT and
    REM assign to the variable FPar_1.
    FPar_1 = Cast_LongToFloat(Par_1)
  EndIf
```

P2_Set_CAN_Baudrate

P2_Set_CAN_Baudrate sets the baud rate on one of the controllers on the specified module and returns the status information.

Syntax

```
#Include ADwinPro_All.inc
ret_val = P2_Set_CAN_Baudrate(module,channel,rate)
```

Parameters

module	Selected module address (1…15).	LONG
channel	Number (1, 2) of the CAN channel, that determines the CAN controller.	LONG
rate	Baud rate of the CAN controller: High speed CAN: 5000…1000000 Bit/s. Low speed CAN: 5000 … 125000 Bit/s.	FLOAT
ret_val	Status of the instruction: 0: Baud rate is set. 1: Baud rate is not allowed and cannot be set.	LONG

Notes

The available baud rates (bus frequencies) are given in the table "Available Baud rates". Please use the table's notation exactly, i.e. non-integer baud rates with 4 decimal places; values with different notation will be rejected as not allowed.

The instruction executes the following actions:

- Checks if the transferred Baud rate is allowed. If not then set the return value to 1 and stop processing.
- Set the registers of the CAN controller for the Baud rate.
- Set sampling mode to 0: One sample per bit.
- Select the settings in such a way that the sample point is always between 60% and 72% of the total bit length.
- Set the jump width for synchroniziation to 1.

In special cases it may be of interest to set a baud rate in a different way than the instruction works. The manual "Pro hardware" gives an explanation how to do this.

The instruction should be called in the program sections **LowInit:** or **Init:**, after the instruction **P2_Init_CAN**, because otherwise the set Baud rate will be overwritten by the default setting (1MBit/s).

See also

P2_Get_CAN_Reg, P2_Init_CAN, P2_Set_CAN_Reg

Valid for

CAN-2 Rev. E

Example

```
#Include ADwinPro_All.inc
Dim status As Long
Init:
  P2_Init_CAN(1,1)          'Initialize CAN controller
  status = P2_Set_CAN_Baudrate(1,1,125000) 'set rate 125 kBit/s
```

Available Baud rates

Available Baud rates [Bit/s]				
1000000.0000	888888.8889	800000.0000	727272.7273	666666.6667
615384.6154	571428.5714	533333.3333	500000.0000	470588.2353
444444.4444	421052.6316	400000.0000	380952.3810	363636.3636
347826.0870	333333.3333	320000.0000	307692.3077	296296.2963
285714.2857	266666.6667	250000.0000	242424.2424	235294.1176
222222.2222	210526.3158	205128.2051	200000.0000	190476.1905
181818.1818	177777.7778	173913.0435	166666.6667	160000.0000
156862.7451	153846.1538	148148.1481	145454.5455	142857.1429
140350.8772	133333.3333	126984.1270	125000.0000	123076.9231
121212.1212	117647.0588	115942.0290	114285.7143	111111.1111
106666.6667	105263.1579	103896.1039	102564.1026	100000.0000
98765.4321	95238.0952	94117.6471	90909.0909	88888.8889
87912.0879	86956.5217	84210.5263	83333.3333	81632.6531
80808.0808	80000.0000	78431.3725	76923.0769	76190.4762
74074.0741	72727.2727	71428.5714	70175.4386	69565.2174
68376.0684	67226.8908	66666.6667	66115.7025	64000.0000
63492.0635	62500.0000	61538.4615	60606.0606	60150.3759
59259.2593	58823.5294	57971.0145	57142.8571	55944.0559
55555.5556	54421.7687	53333.3333	52631.5789	52287.5817
51948.0519	51282.0513	50000.0000	49689.4410	49382.7160
48484.8485	47619.0476	47337.2781	47058.8235	46783.6257
45714.2857	45454.5455	44444.4444	43956.0440	43478.2609
42780.7487	42328.0423	42105.2632	41666.6667	41025.6410
40816.3265	40404.0404	40000.0000	39215.6863	38647.3430
38461.5385	38277.5120	38095.2381	37037.0370	36363.6364
36199.0950	35714.2857	35555.5556	35087.7193	34782.6087
34632.0346	34482.7586	34188.0342	33613.4454	33333.3333
33057.8512	32921.8107	32388.6640	32258.0645	32000.0000
31746.0317	31620.5534	31372.5490	31250.0000	30769.2308
30651.3410	30303.0303	30075.1880	29629.6296	29411.7647
29304.0293	29090.9091	28985.5072	28673.8351	28571.4286
28070.1754	27972.0280	27777.7778	27681.6609	27586.2069
27210.8844	27027.0270	26936.0269	26755.8528	26666.6667
26315.7895	26143.7908	25974.0260	25806.4516	25641.0256
25396.8254	25078.3699	25000.0000	24844.7205	24767.8019
24691.3580	24615.3846	24390.2439	24242.4242	24024.0240
23809.5238	23668.6391	23529.4118	23460.4106	23391.8129
23255.8140	23188.4058	22988.5057	22857.1429	22792.0228
22727.2727	22408.9636	22222.2222	22160.6648	22038.5675
21978.0220	21739.1304	21680.2168	21621.6216	21505.3763
21390.3743	21333.3333	21276.5957	21220.1592	21164.0212
21052.6316	20833.3333	20779.2208	20671.8346	20512.8205
20460.3581	20408.1633	20202.0202	20050.1253	20000.0000
19851.1166	19753.0864	19704.4335	19656.0197	19607.8431
19512.1951	19323.6715	19230.7692	19138.7560	19047.6190

Available Baud rates [Bit/s]				
18912.5296	18867.9245	18823.5294	18648.0186	18604.6512
18518.5185	18433.1797	18390.8046	18306.6362	18181.8182
18140.5896	18099.5475	18018.0180	17857.1429	17777.7778
17738.3592	17582.4176	17543.8596	17429.1939	17391.3043
17316.0173	17241.3793	17204.3011	17094.0171	17021.2766
16949.1525	16913.3192	16842.1053	16806.7227	16771.4885
16666.6667	16632.0166	16563.1470	16528.9256	16460.9053
16393.4426	16326.5306	16260.1626	16227.1805	16194.3320
16161.6162	16129.0323	16000.0000	15873.0159	15810.2767
15779.0927	15686.2745	15625.0000	15594.5419	15503.8760
15473.8878	15444.0154	15384.6154	15325.6705	15238.0952
15180.2657	15151.5152	15122.8733	15094.3396	15065.9134
15037.5940	15009.3809	14842.3006	14814.8148	14705.8824
14652.0147	14571.9490	14545.4545	14519.0563	14492.7536
14414.4144	14336.9176	14311.2701	14285.7143	14260.2496
14184.3972	14109.3474	14035.0877	13986.0140	13937.2822
13913.0435	13888.8889	13840.8304	13793.1034	13722.1269
13675.2137	13605.4422	13582.3430	13559.3220	13513.5135
13468.0135	13445.3782	13377.9264	13333.3333	13289.0365
13223.1405	13157.8947	13136.2890	13114.7541	13093.2897
13071.8954	13008.1301	12987.0130	12903.2258	12882.4477
12820.5128	12800.0000	12759.1707	12718.6010	12698.4127
12578.6164	12558.8697	12539.1850	12500.0000	12422.3602
12403.1008	12383.9009	12345.6790	12326.6564	12307.6923
12288.7865	12195.1220	12158.0547	12121.2121	12066.3650
12030.0752	12012.0120	11994.0030	11922.5037	11904.7619
11851.8519	11834.3195	11764.7059	11730.2053	11695.9064
11661.8076	11627.9070	11611.0305	11594.2029	11544.0115
11494.2529	11477.7618	11428.5714	11396.0114	11379.8009
11363.6364	11347.5177	11299.4350	11220.1964	11204.4818
11188.8112	11111.1111	11080.3324	11034.4828	11019.2837
10989.0110	10943.9124	10928.9617	10884.3537	10869.5652
10840.1084	10810.8108	10796.2213	10781.6712	10752.6882
10695.1872	10666.6667	10638.2979	10610.0796	10582.0106
10540.1845	10526.3158	10457.5163	10430.2477	10416.6667
10389.6104	10335.9173	10322.5806	10296.0103	10269.5764
10256.4103	10230.1790	10204.0816	10101.0101	10088.2724
10062.8931	10025.0627	10012.5156	10000.0000	9937.8882
9925.5583	9876.5432	9852.2167	9828.0098	9803.9216
9791.9217	9768.0098	9756.0976	9696.9697	9685.2300
9661.8357	9615.3846	9603.8415	9569.3780	9523.8095
9456.2648	9433.9623	9411.7647	9400.7051	9367.6815
9356.7251	9324.0093	9302.3256	9291.5215	9259.2593
9227.2203	9216.5899	9195.4023	9153.3181	9142.8571
9090.9091	9070.2948	9049.7738	9039.5480	9009.0090
8958.5666	8928.5714	8918.6176	8888.8889	8879.0233

Available Baud rates [Bit/s]				
8869.1796	8859.3577	8771.9298	8743.1694	8714.5969
8695.6522	8658.0087	8648.6486	8620.6897	8602.1505
8592.9108	8556.1497	8547.0085	8510.6383	8483.5631
8474.5763	8465.6085	8456.6596	8421.0526	8403.3613
8385.7442	8333.3333	8281.5735	8264.4628	8255.9340
8230.4527	8205.1282	8196.7213	8163.2653	8130.0813
8113.5903	8105.3698	8097.1660	8088.9788	8080.8081
8064.5161	8000.0000	7976.0718	7944.3893	7936.5079
7905.1383	7843.1373	7812.5000	7804.8780	7797.2710
7774.5384	7751.9380	7736.9439	7729.4686	7714.5612
7692.3077	7662.8352	7655.5024	7619.0476	7590.1328
7575.7576	7561.4367	7547.1698	7532.9567	7518.7970
7469.6545	7441.8605	7421.1503	7407.4074	7400.5550
7386.8883	7352.9412	7326.0073	7285.9745	7272.7273
7259.5281	7246.3768	7187.7808	7168.4588	7142.8571
7136.4853	7130.1248	7111.1111	7098.4916	7092.1986
7054.6737	7017.5439	6993.0070	6956.5217	6944.4444
6926.4069	6902.5022	6896.5517	6861.0635	6820.1194
6808.5106	6802.7211	6791.1715	6779.6610	6734.0067
6688.9632	6683.3751	6666.6667	6611.5702	6578.9474
6568.1445	6562.7564	6557.3770	6535.9477	6530.6122
6493.5065	6456.8200	6451.6129	6441.2238	6410.2564
6400.0000	6379.5853	6349.2063	6324.1107	6289.3082
6274.5098	6269.5925	6250.0000	6245.1210	6211.1801
6172.8395	6163.3282	6153.8462	6144.3932	6102.2121
6060.6061	6046.8632	6037.7358	5997.0015	5961.2519
5952.3810	5925.9259	5895.3574	5865.1026	5847.9532
5818.1818	5797.1014	5772.0058	5747.1264	5714.2857
5702.0670	5681.8182	5649.7175	5614.0351	5610.0982
5555.5556	5521.0490	5517.2414	5464.4809	5434.7826
5423.7288	5376.3441	5333.3333	5291.0053	5245.9016
5208.3333	5161.2903	5079.3651	5000.0000	

P2_Set_CAN_Reg

P2_Set_CAN_Reg writes a value in a register of the selected CAN controller on the specified module.

Syntax

```
#Include ADwinPro_All.inc
P2_Set_CAN_Reg(module,channel,regno,value)
```

Parameters

module	Selected module address (1…15).	LONG
channel	Number (1, 2) of the CAN channel that determines the CAN controller.	LONG
regno	Register number (0…255) of the CAN controller.	LONG
value	Value (0…255), written into the controller register.	LONG

Notes

The register number which has to be indicated corresponds to the register number of the CAN controller (see data-sheet AN82527 from Intel®). For instance the controll register has the address 0 and the status register the address 1.

See also

P2_Init_CAN, P2_Set_CAN_Baudrate, P2_Get_CAN_Reg

Valid for

CAN-2 Rev. E

Example

```
#Include ADwinPro_All.inc
Init:
  P2_Init_CAN(1,1)          'Initialize CAN controller
  P2_Set_CAN_Reg(1,1,0,1)   'Set control register to the value 1
```

P2_Transmit

P2_Transmit reads the data from the array **CAN_Msg**. As soon as the message object in one of the CAN controllers has access rights to the CAN bus, the message is sent.

Syntax

```
#Include ADwinPro_All.inc
P2_Transmit(module,channel,msg_no)
```

Parameters

module	Selected module address (1…15).	LONG
channel	Number (1, 2) of the CAN channel, that determines the CAN controller.	LONG
msg_no	Number (1… 14) of the message object in the CAN controller.	LONG

Notes

This instruction can only be executed when the corresponding message object has been configured before to send with the **P2_En_Transmit**.

The message data must be entered in **CAN_Msg**[], before executing **P2_Transmit**.

See also

CAN_Msg, P2_En_Receive, P2_En_Transmit, P2_Read_Msg

Valid for

CAN-2 Rev. E

Example

```
REM Sends a 32 Bit FLOAT-value (here: Pi) as sequence of
REM 4 bytes in a message object
REM (Receiving of a float value see example at P2_Read_Msg)

#Include ADwinPro_All.Inc
#Define pi 3.14159265
Dim i As Long

Init:
  P2_Init_CAN(1,1)          'Initialize CAN controller 1

  REM Enable message object 6 of controller 1
  REM for sending with the identifier 40 (11 bit)
  P2_En_Transmit(1,1,6,40,0)

  REM Create bit pattern of Pi with data type Long
  Par_1 = Cast_FloatToLong(pi)

  REM divide bit pattern (32 Bit) into 4 bytes
  CAN_Msg[4] = Par_1 And 0FFh 'assign LSB
  For i = 1 To 3
    CAN_Msg[4-i] = Shift_Right(Par_1,8*i) And 0FFh
  Next i
  CAN_Msg[9] = 4            'message length in bytes

Event:
  P2_Transmit(1,1,6)        'Send the message object 6
```

3.11 Pro II: RSxxx Modules

This section describes instructions which apply to Pro II RSxxx modules:

– P2_Check_Shift_Reg (page 266)
– P2_Get_RS (page 267)
– P2_Read_FIFO (page 268)
– P2_RS_Init (page 269)
– P2_RS_Reset (page 271)
– P2_RS485_Send (page 272)
– P2_RS_Set_LED (page 273)
– P2_Set_RS (page 274)
– P2_Write_Fifo (page 275)
– P2_Write_Fifo_Full (page 276)

In the Instruction List sorted by Module Types (annex A.2) you will find which of the functions corresponds to the *ADwin-Pro II* modules.

P2_Check_Shift_Reg

P2_Check_Shift_Reg returns, if all data has been sent, which was written into the send-Fifo of the channel on the specified module.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = P2_Check_Shift_Reg(module,channel)
```

Parameters

module	Selected module address (1…15).	LONG
channel	number of the channel that is to be read out (1, 2 or 1…4).	LONG
ret_val	Sending status: 0: Data has been sent (= no more data in the send-Fifo). 1: Not yet all data sent (= the send-Fifo still contains data).	LONG

Notes

With the return value 0 both the send Fifo and the output shift register are empty. With the return value 1 there is at least one bit to be sent.

We recommend to use this instruction only after you have more experience about how the controller operates (data-sheet of the manufacturer Texas Instruments). For more common applications more comfortable instructions are availabe in the include file.

See also

P2_Get_RS, P2_RS_Init, P2_RS_Reset, P2_RS485_Send, P2_Set_RS, P2_Write_Fifo, P2_Write_Fifo_Full

Valid for

RS422-4 Rev. E, RSxxx-2 Rev. E, RSxxx-4 Rev. E

Example

```
#Include ADwinPro_All.Inc

Event:
  …
  Par_1 = P2_Check_Shift_Reg(1,1)'Check if channel 1 still
                                 'has data to send
  …
```

P2_Get_RS

P2_Get_RS reads out the controller register on the specified module.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = P2_Get_RS(module,register)
```

Parameters

module	Selected module address (1…15).	LONG
register	Address of the controller register to read.	LONG
ret_val	Contents of the controller register.	LONG

Notes

We recommend to use this instruction only after you have more experience about how the controller operates (data-sheet of the manufacturer Texas Instruments). For more common applications more comfortable instructions are availabe in the include file.

See also

P2_Check_Shift_Reg, P2_RS_Init, P2_RS_Reset, P2_RS485_Send, P2_Set_RS, P2_Write_Fifo, P2_Write_Fifo_Full

Valid for

RS422-4 Rev. E, RSxxx-2 Rev. E, RSxxx-4 Rev. E

Example

- / -

P2_Read_FIFO

P2_Read_Fifo reads a value from the input Fifo of a specified channel on the specified module.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = P2_Read_Fifo(module,channel)
```

Parameters

module	Selected module address (1…15).	LONG
channel	number of the channel that is to be read out (1, 2 or 1…4).	LONG
ret_val	Contents of the input Fifo: -1: Fifo is empty. ≥0:Transferred value.	LONG

Notes

- / -

See also

P2_Check_Shift_Reg, P2_Get_RS, P2_RS_Init, P2_RS_Reset, P2_RS485_Send, P2_Set_RS, P2_Write_Fifo, P2_Write_Fifo_Full

Valid for

RS422-4 Rev. E, RSxxx-2 Rev. E, RSxxx-4 Rev. E

Example

```
#Include ADwinPro_All.Inc

Init:
  P2_RS_Reset(1)
  P2_RS_Init(1,1,9600,0,8,0,1)'Initialize channel 1 on module
                              '1 with 9600 Baud, without parity,
                              '8 data bits, 1 stop bit and
                              'hardware handshake (RS232 only).

Event:
  Par_1 = P2_Read_Fifo(1,1)'Get a value from the Fifo. If
                           'the Fifo is empty, -1 is returned.
```

See also further Examples for RS232 and RS485 (Pro II) on page 431.

P2_RS_Init

P2_RS_Init initializes one channel on the specified module.

The following parameters are set:

- Transfer rate in Baud
- Use of test bits
- Data length
- Amount of stop bits

Transfer protocol (handshake)

Syntax

```
#Include ADwinPro_All.Inc
P2_RS_Init(module, channel, baud_rate, parity, bits,
    stop, handshake)
```

Parameters

module	Selected module address (1…15).	LONG
channel	Channel, which is to be initialized (1, 2 or 1…4).	LONG
baud_rate	Transfer rate in Baud: RS232: 35 … 115,200 Baud. RS485: 35…2,304,000 Baud. RS422: 35…2,304,000 Baud.	LONG
parity	Use of test bits: 0: without parity bit. 1: even parity. 2: odd parity.	LONG
bits	Amount of data bits (5, 6, 7 or 8).	LONG
stop_bits	Amount of stop bits. 0: 1 stop bit. 1: 1½ stop bits at 5 data bits; 2 stop bits at 6, 7 or 8 data bits.	LONG
handshake	Transfer protocol: 0: RS232, no handshake. 1: RS232, hardware handshake (RTS/CTS). 2: RS232, software handshake (Xon/Xoff). 3: RS485, no handshake. 4: RS422, no handshake. 6: RS422, software handshake (Xon/Xoff).	LONG

Notes

This instruction is necessary before working first with the selected channel, in order to set the interface parameters. Parameters must be identical to the remote station, in order to verify a correct transfer.

The initialization is necessary after you have executed a hardware reset with **P2_RS_Reset**.

The baud rates are derived from the basic clock rate of 2.304 MHz by dividing the basic clock rate by an integer. The divisor range is 1…0FFFFh resulting into a band width of 35…2,304,000 Bit/s. According to its specification, the RS-232 interface is limited to 115,200 Bit/s. The following list shows some common baud rates.

Common baud rates [Bit/s]		
2,304,000	57,600	2,400
1,152,000	38,400	1,200

Common baud rates [Bit/s]		
460,800	19,200	600
230,400	9,600	300
115,200	4,800	

See also

P2_Check_Shift_Reg, P2_Get_RS, P2_RS_Reset, P2_RS485_Send, P2_Set_RS, P2_Write_Fifo, P2_Write_Fifo_Full

Valid for

RS422-4 Rev. E, RSxxx-2 Rev. E, RSxxx-4 Rev. E

Example

```
#Include ADwinPro_All.Inc

Init:
  P2_RS_Reset(1)            'Reset RS-module
  P2_RS_Init(1,1,9600,0,8,0,1)'Initialize channel 1 on
                            'module 1 with 9600 Baud, without,
                            'parity, 8 data bits, 1 stop bit and
                            'hardware handshake (RS232 only).
```

See also further Examples for RS232 and RS485 (Pro II) on page 431.

P2_RS_Reset

P2_RS_Reset executes a hardware reset on the specified module and deletes the settings for all channels.

Syntax

```
#Include ADwinPro_All.Inc
P2_RS_Reset(module)
```

Parameters

module	Selected module address (1…15).	LONG

Notes

The instruction sends a reset impulse to the input of the controller TL16C754. In the data-sheet of the controller 16C754 from Texas Instruments it is described, to which values the registers have been set after the hardware reset.

After a hardware reset an initialization with **P2_RS_Init** must follow, in order to initialize the controller and to set the interface parameters.

P2_RS_Init sets the same registers as a hardware reset does. Nevertheless, **P2_RS_Reset** should be used for the case the controller has crashed.

See also

P2_Check_Shift_Reg, P2_Get_RS, P2_RS_Init, P2_RS485_Send, P2_Set_RS, P2_Write_Fifo, P2_Write_Fifo_Full

Valid for

RS422-4 Rev. E, RSxxx-2 Rev. E, RSxxx-4 Rev. E

Example

```
#Include ADwinPro_All.Inc

Init:
  P2_RS_Reset(1)            'Reset RS-module
  P2_RS_Init(1,1,9600,0,8,0,1)'Initialize channel 1 of
                            'module 1 with 9600 Baud, without
                            'parity, 8 data bits, 1 stop bit and
                            'hardware handshake (RS232 only).
```

P2_RS485_Send

P2_RS485_Send determines the transfer direction for a specified channel on the specified module.

Syntax

```
#Include ADwinPro_All.Inc
P2_RS485_Send(module,channel,dir)
```

Parameters

module	Selected module address (1…15).	LONG
channel	Channel to be set (1, 2 or 1…4).	LONG
dir	Transfer direction of the channel: 0: Set channel to receive. 1: Set channel to send. 2: Set channel to send and to receive its sent data.	LONG

Notes

Setting the transfer direction means:

- Receiver: The channel can only read data, even if data are in the output Fifo of the controller for this channel.
- Sender: The channel transfers data to the bus which are read by other devices.
- Sender/receiver: The channel can transfer data to the bus and back at the same time. Thus, the sent data can be checked.

See also

P2_Check_Shift_Reg, P2_Get_RS, P2_RS_Init, P2_RS_Reset, P2_Set_RS, P2_Write_Fifo, P2_Write_Fifo_Full

Valid for

RS422-4 Rev. E, RSxxx-2 Rev. E, RSxxx-4 Rev. E

Example

See example "RS485: Receive And send“ on page 1074.

P2_RS_Set_LED

P2_RS_Set_LED switches the additional LED of a RSxxx channel on (with color) or off.

Syntax

```
#INCLUDE ADwinPro_All.inc
P2_RS_Set_LED(module,channel,led_col)
```

Parameters

module	Selected module address (1…15).	LONG
channel	Channel to be set (1, 2 or 1…4).	LONG
led_col	Status and Farbe of the additional LED: 0: LED off. 1: LED on, color red. 2: LED on, color green. 3: LED on, color orange.	LONG

Notes

- / -

See also

P2_Set_LED

Valid for

RS422-4 Rev. E, RSxxx-2 Rev. E, RSxxx-4 Rev. E

Example

```
#INCLUDE ADwinPro_All.inc

Init:
  P2_RS_Set_LED(1,1,3)      'switch on channel 1 LED, color orange
```

P2_Set_RS

P2_Set_RS writes a value into a specified register on the specified module.

Syntax

```
#Include ADwinPro_All.Inc
P2_Set_RS(module, register, value)
```

Parameters

module	Selected module address (1…15).	LONG
register	Number of the register, into which data are written.	LONG
value	Value to be written into the register.	LONG

Notes

We recommend to use this instruction only after you have more experience about how the controller operates (data-sheet of the manufacturer: TL16C754 from Texas Instruments). For more common applications more comfortable instructions are availabe in the include file.

See also

P2_Check_Shift_Reg, P2_Get_RS, P2_RS_Init, P2_RS_Reset, P2_RS485_Send, P2_Write_Fifo, P2_Write_Fifo_Full

Valid for

RS422-4 Rev. E, RSxxx-2 Rev. E, RSxxx-4 Rev. E

Example

- / -

P2_Write_Fifo

P2_Write_Fifo writes a value into the send-Fifo of a specified channel on the specified module.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = P2_Write_Fifo(module,channel,value)
```

Parameters

module	Selected module address (1…15).	LONG
channel	Channel number to whose send-Fifo data are transferred (1, 2 or 1…4).	LONG
value	Value to be written into the send-Fifo.	LONG
ret_val	Status message: 0: Data are transferred successfully. 1: Data were not transferred, send-Fifo is full.	LONG

Notes

The instruction checks first if there is at least one memory space in the send-Fifo. If this is so, the transferred value is written into the Fifo (return value 0); otherwise a 1 is returned, indicating that the Fifo is full and writing is not possible.

The value to be transferred may be a single ASCII character or an ASCII instruction (chars are internally similar to Long data type). The hardware documentation contains an example for sending a string.

See also

P2_Check_Shift_Reg, P2_Get_RS, P2_RS_Init, P2_RS_Reset, P2_RS485_Send, P2_Set_RS, P2_Write_Fifo_Full

Valid for

RS422-4 Rev. E, RSxxx-2 Rev. E, RSxxx-4 Rev. E

Example

```
#Include ADwinPro_All.Inc
Dim val As Long

Init:
  P2_RS_Reset(1)
  P2_RS_Init(1,1,9600,0,8,0,1)'Initialize channel 1 of
                              'module 1 with 9600 Baud, no parity,
                              '8 data bits, 1 stop bit and
                              'hardware handshake (RS232 only).

Event:
  Par_1 = Write_Fifo(1,1,val)'If the Fifo is not full, [val]
                             'is written into the Fifo. Otherwise
                             'a 1 in Par_1 indicates that writing
                             'into the Fifo ist not possible
                             '(Fifo full).
```

See also further Examples for RS232 and RS485 (Pro II) on page 431.

P2_Write_Fifo_Full

P2_Write_Fifo_Full returns if there is at least one free element in the send-Fifo of a specified channel on the specified module.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = P2_Write_Fifo_Full(module, channel)
```

Parameters

module	Selected module address (1…15).	LONG
channel	Channel number to whose send-Fifo data are transferred (1, 2 or 1…4).	LONG
ret_val	Status message: 0: At least one element is free in send-Fifo. 1: Send-Fifo is full.	LONG

Notes

The return value is the same as with **P2_Write_Fifo**.

See also

P2_Check_Shift_Reg, P2_Get_RS, P2_RS_Init, P2_RS_Reset, P2_RS485_Send, P2_Set_RS, P2_Write_Fifo

Valid for

RS422-4 Rev. E, RSxxx-2 Rev. E, RSxxx-4 Rev. E

Example

```
Rem sending data to and receiving data from the PC while using
Rem a Fifo in ADwin-Pro II
#Include ADwinPro_All.inc

#Define outfifo Data_1
#Define infifo Data_2
#Define rs_adr 5
#Define rs_channel 1

Dim outfifo[1000] As Long As Fifo
Dim infifo[1000] As Long As Fifo
Dim value, dummy, check As Long

Rem use LED as signal: red = sending, green = receiving,
Rem orange (red+green) = sending + receiving
Dim red_led, green_led As Long
Dim green_led_time As Long
Dim led_time As Long

Init:
  Rem reset and initialize interface
  P2_RS_Reset(rs_adr)
  P2_RS_Init(rs_adr, 1, 9600, 0, 8, 0, 0)
  Fifo_Clear(1)
  Fifo_Clear(2)
  green_led = 0
  red_led = 0

Event:
  Rem sending
  If (Fifo_Full(1) > 0) Then 'any data present?
    If (P2_Write_Fifo_Full(rs_adr, rs_channel) = 0) Then
      Rem send Fifo empty?
      value = outFifo        'read value from Fifo
      dummy = P2_Write_Fifo(rs_adr, rs_channel, value)
      Rem dummy is not to be checked, since Write_Fifo_Full has
      Rem proved that Fifo has empty elements.

      'do LED settings
      If (red_led = 0) Then
        red_led = 1
        led_time = Read_Timer()
      EndIf

    EndIf
  EndIf

  Rem receiving
  If (Fifo_Empty(2) > 0) Then 'are there empty elements?
    check = P2_Read_Fifo(rs_adr, rs_channel)

    If (check <> -1) Then   'is a value in the receiving buffer?
      inFifo = check        'get value into inFifo

      'do LED settings
      If (green_led = 0) Then
        green_led = 1
        led_time = Read_Timer()
      EndIf
    EndIf
  EndIf

  'output LED settings
  dummy = (red_led And 1) Or (Shift_Left((green_led And 1), 1))
```

```
P2_RS_Set_LED(rs_adr, rs_channel, dummy)
If ((red_led > 0) Or (green_led > 0)) Then
 If ((Read_Timer() - led_time) > 20000000) Then
   If (red_led > 0) Then Inc red_led
   If (green_led > 0) Then Inc green_led
   led_time = Read_Timer()
  EndIf
 EndIf

If ((red_led = 3) Or (green_led = 3)) Then
 red_led = 0
 green_led = 0
EndIf
```

See also further Examples for RS232 and RS485 (Pro II) on page 431.

3.12 Pro II: LIN bus Interface

This section describes instructions which apply to Pro II LIN bus modules:

– P2_LIN_Init (page 280)
– P2_LIN_Init_Write (page 282)
– P2_LIN_Init_Apply (page 283)
– P2_LIN_Reset (page 284)
– P2_LIN_Get_Version (page 285)
– P2_LIN_Read_Dat (page 286)
– P2_LIN_Msg_Write (page 288)
– P2_LIN_Msg_Transmit (page 290)
– P2_LIN_Set_LED (page 291)

In the Instruction List sorted by Module Types (annex A.2) you will find which of the functions corresponds to the *ADwin-Pro II* modules.

P2_LIN_Init

P2_LIN_Init initializes the data transfer between *ADwin* CPU and the LIN interface on a specified module.

Syntax

```
#Include ADwinPro_All.Inc
REM define LIN settings array
Dim lin_datatable[150] As Long
ret_val = P2_LIN_Init(module, lin_datatable[])
```

Parameters

module	Selected module address (1…15).	LONG
lin_data-table[]	Array which contains settings for data transfer between *ADwin* CPU and the LIN module.	ARRAY LONG
ret_val	Status of initialization: 0: initialization was successful. 1: Error: no Pro II module at this address. 2: Error: no LIN interface on the module.	LONG

Notes

P2_LIN_Init is to be executed before data transfer between *ADwin* CPU and LIN interface. The instruction should be used in the **Init:** section.

Before initialization, an array lin_datatable[] with 150 elements must be declared for each module.

Valid for

LIN-2 Rev. E

See also

P2_LIN_Init_Write, P2_LIN_Init_Apply, P2_LIN_Reset, P2_LIN_Get_Version, P2_LIN_Read_Dat, P2_LIN_Msg_Write, P2_LIN_Msg_Transmit

Example

```
#Include ADwinPro_All.Inc

#Define mod_adr 4
Dim lin_datatable[150] As Long
Dim Data_1[20] As Long
Dim Data_2[20] As Long
Dim state As Long

Init:
  Processdelay = 30000000     '10 Hz
  'Initialize communication ADwin CPU - LIN module
 Par_1 = P2_LIN_Init(mod_adr, lin_datatable)
 If (Par_1 <> 0) Then Exit'error
 Rem Interface 1, 9600 baud, LIN master
 P2_LIN_Init_Write(lin_datatable, 1, 9600, 1, 0)
 Rem Interface 2, 9600 baud, LIN slave
 P2_LIN_Init_Write(lin_datatable, 2, 9600, 0, 0)
 Rem message box 1 for receive on interface 2, msg id 1
 P2_LIN_Msg_Write(lin_datatable, 2, 1, 1, Data_2, 8, 0)
 P2_LIN_Init_Apply(lin_datatable)
 state = 1

Event:
 SelectCase state
  Case 1  'msg transmit
   Data_1[1] = 1
   Data_1[2] = 2
   Data_1[3] = 3
   Data_1[4] = 4
   Data_1[5] = 5
   Data_1[6] = 6
   Data_1[7] = 7
   Data_1[8] = 8
   Rem message box 1 for write on interface 1, msg id 1
   P2_LIN_Msg_Write(lin_datatable, 1, 1, 1, Data_1, 8, 1)
   Rem send header and message (interface 1 = LIN master)
   P2_LIN_Msg_Transmit(lin_datatable, 1, 1) 'msg tx
   state = 2
  Case 2                  'check for msg receive, msg id 1
   P2_LIN_Read_Dat(lin_datatable, 2, 1, Data_2)
   If (Data_2[20] = 1) Then 'new msg rx
    Par_11 = Data_2[3]    'ID
    Par_12 = Data_2[4]    'Byte 1
    Par_13 = Data_2[5]
    Par_14 = Data_2[6]
    Par_15 = Data_2[7]
    Par_16 = Data_2[8]
    Par_17 = Data_2[9]
    Par_18 = Data_2[10]
    Par_19 = Data_2[11]   'Byte 8
    Par_20 = Data_2[12]   'checksum
    Par_21 = Data_2[13]   'length
    Inc Par_10
    state = 1             'new Msg tx
   EndIf
  EndSelect
```

P2_LIN_Init_Write

P2_LIN_Init_Write sets baudrate and operating mode for a specified LIN interface.

Syntax

```
#Include ADwinPro_All.Inc
P2_LIN_Init_Write(lin_datatable[], channel,
    baudrate, mode, chs_enh)
```

Parameters

lin_data-table[]	Array with settings for data transfer between *ADwin* CPU and the LIN module.	ARRAY LONG
channel	Number (1…2) of LIN interface.	LONG
baudrate	Baud rate (2400…19200) to run the interface with.	LONG
mode	Operating mode of the LIN interface: 0: Operation as LIN Slave node. 1: Operation as LIN Master node.	LONG
chs_enh	Type of checksum: 0: classic. 1. enhanced.	LONG

Notes

Writing initialization data is required for each LIN interface separately. Afterwards, the initialization data must be activated for the LIN bus using **P2_LIN_Init_Apply**.

If **P2_LIN_Init_Write** is not processed, all interfaces run with standard settiings:

- Baud rate 9600 Baud.
- Slave mode.
- Checksum type „classic“.

Valid for

LIN-2 Rev. E

See also

P2_LIN_Init, P2_LIN_Init_Apply, P2_LIN_Reset, P2_LIN_Get_Version, P2_LIN_Read_Dat, P2_LIN_Msg_Write, P2_LIN_Msg_Transmit

Example

see P2_LIN_Init

P2_LIN_Init_Apply

P2_LIN_Init_Apply activates the initialization data given with **P2_LIN_Init_Write** for all LIN interfaces.

Syntax

```
#Include ADwinPro_All.Inc
P2_LIN_Init_Apply(lin_datatable[])
```

Parameters

lin_data-table[]	Array with settings for data transfer between *ADwin* CPU and the LIN module.	ARRAY LONG

Notes

P2_LIN_Init_Apply does not change initialization data of the LIN bus interfaces.

Valid for

LIN-2 Rev. E

See also

P2_LIN_Init, P2_LIN_Init_Write, P2_LIN_Reset, P2_LIN_Get_Version, P2_LIN_Read_Dat, P2_LIN_Msg_Write, P2_LIN_Msg_Transmit

Example

see P2_LIN_Init

P2_LIN_Reset

P2_LIN_Reset resets all LIN interfaces, either all settings (start-up status) or LIN-internal counters only.

Syntax

```
#Include ADwinPro_All.Inc
P2_LIN_Reset(lin_datatable[], reset_mode)
```

Parameters

lin_data-table[]	Array with settings for data transfer between *ADwin* CPU and the LIN module.	ARRAY LONG
reset_mode	State to set the interfaces to: 1: all settings to start-up status. 2: reset LIN counters to 0.	LONG

Notes

If resetting to start-up status, the following settings are set for all LIN interfaces:

- Baud rate 9600 Baud
- Slave mode
- Set internal counters to 0 (message, timeout).

Valid for

LIN-2 Rev. E

See also

P2_LIN_Init, P2_LIN_Init_Write, P2_LIN_Init_Apply, P2_LIN_Get_Version, P2_LIN_Read_Dat, P2_LIN_Msg_Write, P2_LIN_Msg_Transmit

Example

- / -

P2_LIN_Get_Version

P2_LIN_Get_Version returns the version number of the LIN interface.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = P2_LIN_Get_Version(lin_datatable[])
```

Parameters

lin_data-table[]	Array with settings for data transfer between *ADwin* CPU and the LIN module.	ARRAY LONG
ret_val	Version number (0…9999) of LIN interface.	LONG

Notes

The version number is needed only, if you query our support about programming the LIN bus.

Valid for

LIN-2 Rev. E

See also

P2_LIN_Init, P2_LIN_Init_Write, P2_LIN_Init_Apply, P2_LIN_Reset, P2_LIN_Read_Dat, P2_LIN_Msg_Write, P2_LIN_Msg_Transmit

Example

- / -

P2_LIN_Read_Dat

P2_LIN_Read_Dat reads the data of a message box or the status of a LIN interface and writes the result into an array.

Syntax

```
#Include ADwinPro_All.Inc
P2_LIN_Read_Dat(lin_datatable[], channel, msgbox,
    rd_dat[])
```

Parameters

lin_data-table[]	Array with settings for data transfer between *ADwin* CPU and the LIN module.	ARRAY LONG
channel	Number (1…2) of LIN interface.	LONG
msgbox	Code number to select either a message box or LIN interface status: 1…64: Number of LIN message box, where data is read. 65: Read LIN interface status.	LONG
rd_dat[]	Destination array where data is returned.	ARRAY LONG

Notes

If you provide an invalid code number msgbox the module may get into an instable mode. If so, you have to switch off the Pro system and restart it.

A message box / the interface status has 20 elements. The data is stored in the array elements rd_dat[1] ... rd_dat[20]. The following table shows the element's meaning:

Element	Message box (membox=1…64)	interface status (membox=65)
1	Kanalnummer (1…2)	Kanalnummer (1…2)
2	Number (1…64) of message box.	Identifier (65) for interface status.
3	Identifier (0…63) of the message box	Previously used identifier (0…63).
4	Data bytes 1…8	Number of transferred messages since xxx.
5…11		–
12	Checksum for data bytes	–
13	Number of valid data bytes	–
14	Transfer direction of message box: 0: receive 1: send	–
15	Time of LIN header in µs.	–
16	Time of LIN response in µs.	–
17	Total time of a LIN message in µs (= 15+16).	–
18	Pause time in µs between 2 data bytes. Standard: 0.	–

Ele-ment	Message box (membox=1…64)	interface status (membox=65)
19	Number of timeout errors for this message.	–
20	1: New message has been received / sent. -1: No new message -2: Message is being received. -3: Message has timeout error. -4: Check sum error while receiving.	–

Valid for

LIN-2 Rev. E

See also

P2_LIN_Init, P2_LIN_Init_Write, P2_LIN_Init_Apply, P2_LIN_Reset, P2_LIN_Get_Version, P2_LIN_Msg_Write, P2_LIN_Msg_Transmit

Example

see P2_LIN_Init

P2_LIN_Msg_Write

P2_LIN_Msg_Write configures a message box of a LIN interface for send or receive.

Syntax

```
#Include ADwinPro_All.Inc
P2_LIN_Msg_Write(lin_datatable[], channel, membox,
    msg_id, msg_dat[], msg_len, msg_send)
```

Parameters

lin_data-table[]	Array with settings for data transfer between *ADwin* CPU and the LIN module.	ARRAY LONG
channel	Number (1…2) of LIN interface.	LONG
membox	Number (1…64) of the LIN message box to be configured.	LONG
msg_id	Identifier (0…63) of the message box.	LONG
msg_dat[]	Source array, from which data bytes are transferred into the message box.	LONG
msg_len	Number (1…8) of data bytes of msg_dat[] to be transferred.	LONG
msg_send	Transfer status of the message box: 0: receive 1: send	LONG

Notes

The array msg_dat[] must be dimensioned to 8 elements at least. With message box transfer status "receive", the data of array msg_dat[] will not be used.

After configuring, the message box is immediately active on the LIN bus, i.e. data can be received or sent.

If you want to change the data bytes for a message box with transfer status "send", use **P2_LIN_Msg_Write** again.

The message box of a LIN master node operates different from a LIN slave node:

- **Master node, send**: The LIN master sends both the header (see **P2_LIN_Msg_Transmit**) and then the data packet of the message box.
- **Master node, receive**: The LIN master sends the header (see **P2_LIN_Msg_Transmit**) on the LIN Bus and waits for the response of the appropriate slave node. The received data packet is stored into the message box.
- **Slave node, send**: The LIN slave waits until the master sends the header with the identifier which fits to the identifier of the message box. Only then the slave node will send its data packet.
- **Slave node, receive**: The slave node waits until the master sends the header with the identifier which fits to the identifier of the message box. Then the slave receives the data packet and stores it into the message box.

Valid for

LIN-2 Rev. E

See also

P2_LIN_Init, P2_LIN_Init_Write, P2_LIN_Init_Apply, P2_LIN_Reset, P2_LIN_Get_Version, P2_LIN_Read_Dat, P2_LIN_Msg_Transmit

Example

see P2_LIN_Init

P2_LIN_Msg_Transmit

P2_LIN_Msg_Transmit sends a header and the identifier of a message box to the LIN bus. To use only with operating mode LIN master.

Syntax

```
#Include ADwinPro_All.Inc
P2_LIN_Msg_Transmit(lin_datatable[], channel,
    membox)
```

Parameters

lin_data-table[]	Array with settings for data transfer between *ADwin* CPU and the LIN module.	ARRAY LONG
channel	Number (1…2) of LIN interface.	LONG
membox	Number (1…64) of the message box, which is enabled for data transfer.	LONG

Notes

P2_LIN_Msg_Transmit is valid only for an interface with operating mode LIN master (see **P2_LIN_Init_Write**), because only a LIN master can send a header.

After sending a header to the LIN bus, only this bus node will react which manages a message box with the identifier msg_id. Thus, this node will send a data packet to or receive a data packet from the LIN bus. The identifier msg_id of message box
membox is set with **P2_LIN_Msg_Write**.

Valid for

LIN-2 Rev. E

See also

P2_LIN_Init, P2_LIN_Init_Write, P2_LIN_Init_Apply, P2_LIN_Reset, P2_LIN_Get_Version, P2_LIN_Read_Dat, P2_LIN_Msg_Write

Example

see P2_LIN_Init

P2_LIN_Set_LED

P2_LIN_Set_LED switches the additional LED of a LIN interface on (with color) or off.

Syntax

```
#Include ADwinPro_All.Inc
P2_LIN_Set_LED(module,channel,led_col)
```

Parameters

module	Selected module address (1…15).	LONG
channel	Number (1…2) of LIN interface.	LONG
led_col	Status and color of the additional LED: 0: LED off. 1: LED on, color red. 2: LED on, color green. 3: LED on, color orange.	LONG

Notes

You set the LED on top of the front panel with **P2_Set_LED**.

See also

P2_Set_LED

Valid for

LIN-2 Rev. E

Example

```
#Include ADwinPro_All.Inc
Dim lin_datatable[150] As Long
Dim ret_val As Long

Init:
  Rem initialize LIN controller
  ret_val = P2_LIN_Init(1, lin_datatable)
  P2_LIN_Set_LED(1,1,3)    'set LED 1 to orange
```

3.13 Pro II: Profibus interface

This section contains instructions to access a Profibus interface of *ADwin-Pro II*.

– P2_Init_Profibus (page 293)
– P2_Run_Profibus (page 295)

In the Instruction List sorted by Module Types (annex A.2) you will find which of the functions corresponds to the *ADwin-Pro II* modules.

P2_Init_Profibus

P2_Init_Profibus initializes the Profibus Slave.

Syntax

```
#Include ADwinGoldII.Inc
ret_val = P2_Init_Profibus(module, dev_adr,
    in_mod_cnt, in_mod_type, out_mod_cnt,
    out_mod_type, work_arr[], info[])
```

Parameters

module	Selected module address (1…15).	LONG
dev_adr	Slave node address (1…125) on the Profibus.	LONG
in_mod_cnt	Number (0…76) of input areas in the Profibus Slave. The max. number depends on in_mod_type.	LONG
in_mod_type	Key number (1…3, 16) for the length of input areas: 1: 1 Byte; max. value for in_mod_cnt: 76. 2: 2 Byte; max. value for in_mod_cnt: 38. 3: 4 Byte; max. value for in_mod_cnt: 19. 16:8 Byte; max. value for in_mod_cnt: 9.	LONG
out_mod_cnt	Number (0…76) of output areas in the Profibus Slave. The max. number depends on out_mod_type.	LONG
out_mod_type	Key number (1…3, 16) for the length of output areas: 1: 1 Byte; max. value for out_mod_type: 76. 2: 2 Byte; max. value for out_mod_type: 38. 3: 4 Byte; max. value for out_mod_type: 19. 16:8 Byte; max. value for out_mod_type: 9.	LONG
work_arr[]	Array to store data for operation of the Profibus Slave. The array must have at least 200 elements.	ARRAY LONG
info[]	Array holding data about the Profibus Slave. The array must have at least 10 elements. The elements info[1] and info[2] contain the production type of the Profibus Slave: info[1]=1, info[2]=4	ARRAY LONG
ret_val	State of initialization: 0: no error. ≠0: Error; please contact the support of Jäger Messtechnik.	LONG

Notes

This instruction must be processed before working with Profibus Slave.

P2_Init_Profibus should be processed in a program section with low priority, because of the long processing time (about 2-3 seconds). Using the instruction in a (non-interruptable) high priority process, the communication between PC and *ADwin* system would be interrupted too long and thus produce an error message (timeout).

Station address, number and length of areas must equal the project settings of the profibus. For projecting, the area length is also given in words: 1 word = 2 byte.

The total number of data bytes in an area is reuired for the instruction **P2_Run_Profibus**: the total number is the product of the number of input/output areas times the length of the area in bytes.
Example: for in_mod_cnt = 7 and in_mod_type = 3 the total number is 7 x 4 Bytes = 28 Bytes.

Valid for

Profi-SL Rev. E

See also

P2_Run_Profibus

Example

```
#Include ADwinPro_All.INC
#Define module 5           'module address
#Define node 2             'slave node address
#Define info Data_1         'info array
#Define out_arr Data_2
#Define in_arr Data_3

Dim out_arr[76] As Long At DM_Local
Dim in_arr[76] As Long At DM_Local
Dim conf_arr[200] As Long At DM_Local
Dim info[10] As Long At DM_Local
Dim i As Long
Dim error As Long

Init:
  Processdelay = 3000000    'set to 100 Hz
  For i = 1 To 10           'initialize info array
    info[i] = 0
  Next i
  Rem initialize profibus interface: 38 input data areas of 2 byte
  Rem and 76 output data bytes of 1 Byte
  error = P2_Init_Profibus(module,node,38,2,76,1,conf_arr,info)
  If (error <> 0) Then      'initialization error
    Par_1 = error
    Exit
  EndIf

Event:
  Rem set data in out_arr[] to be transferred
  For i = 1 To 76
    out_arr[i] = (out_arr[i] + i) And 0FFh
  Next i

  Rem send and read data. data bytes input: 38x2=76;
  Rem data bytes output areas: 38x1=76)
  error = P2_Run_Profibus(module,out_arr,76,in_arr,76,conf_arr)
  error = error And 7h
  Par_2 = error

  Rem here the received data in in_arr[] can be processed
```

P2_Run_Profibus

P2_Run_Profibus exchanges data with the Profibus Slave.

Syntax

```
#Include ADwinGoldII.Inc
ret_val = P2_Run_Profibus(module, out_pd_arr[],
    out_pd_arr_len, in_pd_arr[], in_pd_arr_len,
    work_arr[])
```

Parameters

module	Selected module address (1…15).	LONG
out_pd_arr[]	Array from which the Profibus Slave reads data and writes them to the Profibus.	ARRAY LONG
out_pd_arr_len	Number of output areas (1…76), the data of which are read from the array out_pd_arr[]. The number may not be greater than given in out_mod_cnt with **P2_Init_Profibus**.	LONG
in_pd_arr[]	Array into which the Profibus-Slave writes data, which are read by the Profibus..	ARRAY LONG
in_pd_arr_len	Number of input areas (1…76), the data of which are returned in the array in_pd_arr[]. The number may not be greater than given in in_mod_cnt with **P2_Init_Profibus**.	LONG
work_arr[]	Array holding data for operation of the Profibus Slave, see **P2_Init_Profibus**.	ARRAY LONG
ret_val	Bit pattern holding the state of operation of the Profibus Slave. Only bits Bits 0…2 are important: **100b**: Slave is active and runs correcty. **010b**: Profibus inactive, Slave is waiting. **110b**, **111b**: Error.	LONG

Notes

P2_Run_Profibus should be processed in a program section with low priority, because of the long processing time. Using the instruction in a (non-interruptable) high priority process, the communication between PC and *ADwin* system would be interrupted too long and thus produce an error message (timeout).

Each array element in in_pd_arr[] and out_pd_arr[] contains a single data byte only (bits 0…7). Data areas of more than one byte length are saved in the appropriate number of consecutive array elements. Example: 5 data areas of 4 byte length are stored in 5x4=20 array elements.

Valid for

Profi-SL Rev. E

See also

P2_Init_Profibus

Example

see P2_Init_Profibus

3.14 Pro II: MIL-STD-1553 bus Interface

This section describes instructions which apply to Pro II MIL-STD-1553 bus modules:

– P2_MIL_Reset (page 297)
– P2_MIL_SMT_Init (page 298)
– P2_MIL_SMT_Message_Read (page 299)
– P2_MIL_SMT_Set_All_Filters (page 301)
– P2_MIL_SMT_Set_Filter (page 302)
– P2_MIL_Set_LED (page 303)
– P2_MIL_Set_Register (page 304)
– P2_MIL_Get_Register (page 305)

P2_MIL_Reset

P2_MIL_Reset initializes the MIL interface on the specified module and resets all registers to the default value.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = P2_MIL_Reset(module)
```

Parameters

module	Selected module address (1…15).	LONG
ret_val	Status of initialization: 0: initialization was successful. <>0: Error.	LONG

Notes

P2_MIL_Reset is to be executed before the MIL interface can be accessed. The instruction should be used in the **Init:** section.

Valid for

MIL-1553 Rev. E

See also

P2_MIL_SMT_Init, P2_MIL_SMT_Message_Read, P2_MIL_SMT_Set_All_Filters, P2_MIL_SMT_Set_Filter, P2_MIL_Set_LED

Example

```
#Include ADwinPro_All.Inc
#Define mod_adr 4
#Define cmd_dat Data_1
#Define msg_dat Data_2

Dim cmd_dat[20] As Long
Dim msg_dat[20] As Long
Dim state As Long

Init:
  Rem Initialize MIL module
  Par_1 = P2_MIL_Reset(mod_adr)
  If (Par_1 <> 0) Then Exit'error
  Rem initialize SMT 16 bit, time tag counter 64µs
  P2_MIL_SMT_Init(mod_adr, 1, 7)
  Rem disable all subaddresses for read and write
  P2_MIL_SMT_Set_All_Filters(mod_adr, 1, 1)
  Rem record RT 8, all subaddresses receive and transmit
  Par_2 = P2_MIL_SMT_Set_Filter(mod_adr, 8, 0FFh, 0FFh)

Event:
  Rem check for new message
  Par_1 = P2_MIL_SMT_Message_Read(mod_adr, cmd_dat, msg_dat)
  If (Par_1 >= 0) Then 'new message found
    Rem process message
    Rem …
  EndIf
```

P2_MIL_SMT_Init

P2_MIL_SMT_Init initializes the 16-bit simple monitoring terminal mode (SMT) for both buses A and B on the specified module.

Syntax

```
#Include ADwinPro_All.Inc
P2_MIL_SMT_Init(module, timetag_mode, clock_source)
```

Parameters

module	Selected module address (1…15).	LONG
timetag_mode	Time tag mode: 1: SMT 16-bit time tag. Other modes are not supported yet in *ADbasic*.	LONG
clock_source	Clock source for time tag counter: 0: Time tag counter disabled. 2: Internally generated 2µs clock. 3: Internally generated 4µs clock. 4: Internally generated 8µs clock. 5: Internally generated 16µs clock. 6: Internally generated 32µs clock. 7: Internally generated 64µs clock. 8: Internally generated 100µs clock.	LONG

Notes

The SMT monitors both buses A and B. The module's bus controller or remote terminal can use one or both buses in parallel.

If the time tag counter is disabled, the time tag will always be 0 (zero). The external clock source is not supported.

Other monitor modes (SMT 48 bit, IRIG Monitor Terminal IMT) can be used setting the appropriate registers.

Valid for

MIL-1553 Rev. E

See also

P2_MIL_Reset, P2_MIL_SMT_Message_Read, P2_MIL_SMT_Set_All_Filters, P2_MIL_SMT_Set_Filter, P2_MIL_Set_LED

Example

see P2_MIL_Reset

P2_MIL_SMT_Message_Read

P2_MIL_SMT_Message_Read reads Command Buffer and Data Buffer Block of the recently recorded MIL message on the specified module.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = P2_MIL_SMT_Message_Read(module, cmd_dat[], msg_dat[])
```

Parameters

module	Selected module address (1…15).	LONG
cmd_dat[]	Array where 4 command buffer words (16 bit) of a single message are stored. cmd_dat[1] = Block Status Word cmd_dat[2] = Message Time Stamp cmd_dat[3] = Data Block Pointer cmd_dat[4] = Message Command Word The array size must be dimensioned to 4 at least.	ARRAY LONG
msg_dat[]	Array, where data words (16 bit) are stored. The number of stored data words is given in ret_val. The array size must be dimensioned to 35 at least.	ARRAY LONG
ret_val	Reading status: -1: No message found. 0…n: Number of words stored in msg_dat[].	LONG

Notes

You can only read the most recent message information. As soon as a new MIL message is completely received and stored, any previous message information is lost.

Use **P2_MIL_SMT_Init** to set the time tag mode for the message time stamp.

The data words in msg_dat[] are stored in ascending order as they were sent via the MIL bus, starting from array index 1.
The data block pointer (cmd_dat[3]) is returned for technical reasons only, but is of no use in *ADbasic*.

The block status word (cmd_dat[1]) contains additional information regarding message status, the bus on which the message occurred, and occurred errors. If any, RT status words are stored in msg_dat[].
The bits of the block status word have the following meaning:

Bit no.	Bit name	Function
15	EOM	End of message: 0: message is incomplete. 1: message is complete. SOM is reset concurrently.
14	SOM	Start of message: 0: message was completed. 1: message started = a valid command word was completed before.
13	BID	Bus ID: 0: Bus A. 1: Bus B.

Bit no.	Bit name	Function
12	EO	Error flag: One of the error bits (0…5, 9, 10) is set, or an unfinished message is superseded by another valid command.
11	RR	0: command with a single command word. 1: RT-to-RT transfer, i.e. message begins with 2 contiguous command words.
10	IGE	1: Illegal gap error.
9	TM	1: Response timeout.
8	GDB	0: error occurred (in a completed message). 1: good data block transfer.
7	DSR	1: Data buffer rollover.
6	SFS	1: Status flag set.
5	LE	1: Word count error.
4	SE	1: Sync type error.
3	WE	1: Invalid word error.
2	RRGSA	1: RT-to-RT gap/sync/address error.
1	RRCW2	1: RT-to-RT command word 2 error.
0	CWCE	1: Command word content error.

You find more information about errors in the separate documentation "HI-6310 / MIL-STD-1553 / BC/MT/RT Multi-Terminal Device" by Holt Integrated Circuits Inc.

Valid for

MIL-1553 Rev. E

See also

P2_MIL_Reset, P2_MIL_SMT_Init, , P2_MIL_SMT_Set_All_Filters, P2_MIL_SMT_Set_Filter, P2_MIL_Set_LED

Example

see P2_MIL_Reset

P2_MIL_SMT_Set_All_Filters

P2_MIL_SMT_Set_All_Filters enables or disables filtering of all receive and transmit subaddresses of all remote terminals for the MIL interface on the specified module.

Syntax

```
#Include ADwinPro_All.Inc
P2_MIL_SMT_Set_All_Filters(module, disable_receive, disable_transmit)
```

Parameters

module	Selected module address (1…15).	LONG
disable_receive	Filter setting of all receive subaddresses 0…31 of all 32 RT addresses: 0: enable all receive subaddresses. 1: disable all receive subaddresses.	LONG
disable_transmit	Filter setting of all transmit subaddresses 0…31 of all 32 RT addresses: 0: enable all transmit subaddresses. 1: disable all transmit subaddresses.	LONG

Notes

Use **P2_MIL_SMT_Set_Filter** to set the filtering of a single remote terminal.

After power-up, all addresses and subaddresses for receive and transmit are enabled. I.e. the SMT records every message on the MIL bus.

While filtering MIL messages, the SMT refers to each command word's RT address and subaddress, and the transmit/receive bit status. You get the command word using **P2_MIL_SMT_Message_Read** in the array element cmd_dat[4].

Valid for

MIL-1553 Rev. E

See also

P2_MIL_Reset, P2_MIL_SMT_Init, P2_MIL_SMT_Message_Read, P2_MIL_SMT_Set_Filter, P2_MIL_Set_LED

Example

see P2_MIL_Reset

P2_MIL_SMT_Set_Filter

P2_MIL_SMT_Set_Filter enables or disables filtering of all receive and transmit subaddresses of a single remote terminal for the MIL interface on the specified module.

Syntax

```
#Include ADwinPro_All.inc
ret_val = P2_MIL_SMT_Set_Filter(module, rt_addr,
    rx_subaddr, tx_subaddr)
```

Parameters

module	Selected module address (1…15).	LONG
rt_addr	RT address (0…31) to filter.	LONG
rx_subaddr	Bit pattern as filter setting of the receive sub addresses 0…31 (see table): Bit = 0: Enable subaddress to be monitored. Bit = 1: Disable subaddress from monitoring.	LONG
tx_subaddr	Bit pattern as filter setting of the transmit sub addresses 0…31 (see table): Bit = 0: Enable subaddress to be monitored. Bit = 1: Disable subaddress from monitoring.	LONG
ret_val	Instruction execution status: 0: OK 1: Error, address of remote terminal out of range	LONG

Bit no.	31	30	…	1	0
Subaddress	31	30	…	1	0

Notes

Use **P2_MIL_SMT_Set_All_Filters** to enable or disable the filtering of all remote terminals at once.

After power-up, all addresses and subaddresses for receive and transmit are enabled. I.e. the SMT records every message on the MIL bus.

While filtering MIL messages, the SMT refers to each command word's RT address and subaddress, and the transmit/receive bit status. You get the command word using **P2_MIL_SMT_Message_Read** in the array element cmd_dat[4].

Valid for

MIL-1553 Rev. E

See also

P2_MIL_Reset, P2_MIL_SMT_Init, P2_MIL_SMT_Message_Read, P2_MIL_SMT_Set_All_Filters, P2_MIL_Set_LED

Example

see P2_MIL_Reset

P2_MIL_Set_LED

P2_MIL_Set_LED switches the additional LEDs of the MIL interface on the specified module on or off.

Syntax

```
#Include ADwinPro_All.Inc
P2_MIL_Set_LED(module,pattern)
```

Parameters

module	Selected module address (1…15).	LONG
pattern	Bit pattern to set the additional LEDs. Bit = 0: Switch LED off. Bit = 1: Switch LED on.	LONG

Bit no.	31:4	3	2	1	0
LED	–	Bus A, LED 2	Bus A, LED 1	Bus B, LED 2	Bus B, LED 1

Notes

You set the LED on top of the front panel with **P2_Set_LED**.

See also

P2_MIL_Reset, P2_MIL_SMT_Init, P2_MIL_SMT_Message_Read, P2_MIL_SMT_Set_All_Filters, P2_MIL_SMT_Set_Filter, P2_Set_LED

Valid for

MIL-1553 Rev. E

Example

```
#Include ADwinPro_All.Inc
#Define mod_adr 4

Init:
  Rem initialize MIL controller
  Par_1 = P2_MIL_Reset(mod_adr)
  If (Par_1 <> 0) Then Exit'error
  P2_MIL_Set_LED(mod_adr,1100b)'set both bus A LEDs
```

P2_MIL_Set_Register

P2_MIL_Set_Register sets a register value in the MIL interface on the specified module.

Syntax

```
#Include ADwinPro_All.Inc
P2_MIL_Set_Register(module, reg_addr, value)
```

Parameters

module	Selected module address (1…15).	LONG
reg_addr	Address (0…65535) of the value to be set.	LONG
value	Value (0…65535) to be set.	LONG

Notes

You find information about MIL interface registers in the separate documentation "HI-6310 / MIL-STD-1553 / BC/MT/RT Multi-Terminal Device" by Holt Integrated Circuits Inc.

See also

P2_MIL_Get_Register, P2_MIL_Reset, P2_MIL_SMT_Init, P2_MIL_SMT_Message_Read, P2_MIL_SMT_Set_All_Filters, P2_MIL_SMT_Set_Filter, P2_MIL_Set_LED

Valid for

MIL-1553 Rev. E

Example

```
#Include ADwinPro_All.Inc
#Define mod_adr 4

Init:
  Rem initialize MIL controller
  Par_1 = P2_MIL_Reset(mod_adr)
  If (Par_1 <> 0) Then Exit'error
  Rem set register 1000h
  P2_MIL_Set_Register(mod_adr, 1000h, 3)
```

P2_MIL_Get_Register

P2_MIL_Get_Register returns a register value from the MIL interface on the specified module.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = P2_MIL_Get_Register(module, reg_addr)
```

Parameters

module	Selected module address (1…15).	LONG
reg_addr	Address (0…65535) of the value to be read.	LONG
ret_val	Value (0…65535) which was read.	LONG

Notes

You find information about MIL interface registers in the separate documentation "HI-6310 / MIL-STD-1553 / BC/MT/RT Multi-Terminal Device" by Holt Integrated Circuits Inc.

See also

P2_MIL_Set_Register, P2_MIL_Reset, P2_MIL_SMT_Init, P2_MIL_SMT_Message_Read, P2_MIL_SMT_Set_All_Filters, P2_MIL_SMT_Set_Filter, P2_MIL_Set_LED

Valid for

MIL-1553 Rev. E

Example

```
#Include ADwinPro_All.Inc
#Define mod_adr 4

Init:
  Rem initialize MIL controller
  Par_1 = P2_MIL_Reset(mod_adr)
  If (Par_1 <> 0) Then Exit'error
  Rem read register 1000h
  Par_10 = P2_MIL_Get_Register(mod_adr, 1000h)
```

3.15 Pro II: ARINC-429 bus Interface

This section describes instructions which apply to Pro II ARINC bus modules:

– P2_ARINC_Reset (page 307)
– P2_ARINC_Config_Transmit (page 308)
– P2_ARINC_Config_Receive (page 310)
– P2_ARINC_Transmit_Fifo_Full (page 312)
– P2_ARINC_Transmit_Fifo_Empty (page 313)
– P2_ARINC_Write_Transmit_Fifo (page 314)
– ARINC_Create_Value32 (page 315)
– P2_ARINC_Transmit_Enable (page 316)
– P2_ARINC_Receive_Fifo_Empty (page 317)
– P2_ARINC_Read_Receive_Fifo (page 318)
– ARINC_Split_Value32 (page 319)
– P2_ARINC_Set_Labels (page 320)

P2_ARINC_Reset

P2_ARINC_Reset runs a master reset on the ARINC interface of the specified module.

Syntax

```
#Include ADwinPro_All.Inc
P2_ARINC_Reset(module)
```

Parameters

module	Selected module address (1…15).	LONG

Notes

P2_ARINC_Reset is to be executed before the ARINC interface can accessed. The instruction should be used in the **Init:** section. Afterwards, configure the transmitter and/or receiver settings.

On a master reset, data transmission and reception are immediately terminated, the transmit and receive FIFOs are cleared.

Valid for

ARINC-429 Rev. E

See also

P2_ARINC_Config_Transmit, P2_ARINC_Config_Receive, P2_ARINC_Write_Transmit_Fifo, ARINC_Create_Value32, P2_ARINC_Transmit_Enable, P2_ARINC_Transmit_Fifo_Full, P2_ARINC_Read_Receive_Fifo, P2_ARINC_Set_Labels

Example

see P2_ARINC_Config_Transmit or P2_ARINC_Config_Receive

P2_ARINC_ Config_Transmit

`P2_ARINC_Config_Transmit` configures the transmitter settings on the specified ARINC module.

Syntax

```
#Include ADwinPro_All.Inc
P2_ARINC_Config_Transmit(module, clk_speed,
      parity_enable, parity)
```

Parameters

`module`	Selected module address (1…15).	LONG
`clk_speed`	Transfer bit rate: 0: 100kHz (high speed), O/P slope = 1.5µs 1: 12.5kHz (low speed), O/P slope = 10µs	LONG
`parity_ enable`	Enable or disable parity check. 0: No parity check, parity bit is part of transmitted data. 1: Check parity and set parity bit.	LONG
`parity`	Set type of parity check; only useful if `parity_ enable` is set to 1. 0: Odd parity. 1: Even parity.	LONG

Notes

The instruction should be used in the `Init:` section.

Note, that the module receivers will always check for parity. Receiver parity check cannot be disabled.

Valid for

ARINC-429 Rev. E

See also

P2_ARINC_Reset, P2_ARINC_Config_Receive, P2_ARINC_Write_ Transmit_Fifo, ARINC_Create_Value32, P2_ARINC_Transmit_Enable

Example

```
#Include ADwinPro_All.Inc
#Define mod_adr 4
#Define number Par_10
#Define value Par_11
Dim arinc_label As Long

Init:
  Rem Initialize ARINC module
  P2_ARINC_Reset(mod_adr)
  Rem configure transmitter to 100kHz and even parity
  P2_ARINC_Config_Transmit(mod_adr, 0, 1, 1)
  Rem enable transmitter
  P2_ARINC_Transmit_Enable(mod_adr, 1)
  number = 0
  arinc_label = 10010001b  'set label

Event:
  Rem check for space in transmitter fifo
  If (P2_ARINC_Transmit_Fifo_Full(mod_adr) = 0) Then
    Inc number              'increase number to be sent
    If (number > 07FFFFh) Then number = 1
    Rem create value to be sent (with SSM=11b and SDI=01b)
    value = ARINC_Create_Value32(arinc_label, 11b, 01b, number)
    Rem Write value to transmitter fifo
    P2_ARINC_Write_Transmit_Fifo(mod_adr, value)
  EndIf
```

P2_ARINC_ Config_Receive

P2_ARINC_Config_Receive configures the receive settings on the specified ARINC module.

Syntax

```
#Include ADwinPro_All.Inc
P2_ARINC_Config_Receive(module, channel,
      clk_speed, enable_label, enable_decoder,
      decoder_value)
```

Parameters

module	Selected module address (1…15).	LONG
channel	Number (1, 2) of receive channel on the module.	LONG
clk_speed	Transfer bit rate: 0: 100kHz (high speed), O/P slope = 1.5µs 1: 12.5kHz (low speed), O/P slope = 10µs	LONG
enable_ label	Enable or disable label matching: 0: disable label matching, all incoming values are read into the receiver FIFO. 1: enable label matching, only values matching the label are read into the receiver FIFO.	LONG
enable_ decoder	Enable or disable receiver SDI decoding: 0: disable receiver decoding. 1: enable receiver decoding, the SDI bits of incoming values must match the pattern decoder_value.	LONG
decoder_ value	Bit pattern (00b…11b) for SDI decoding.	LONG

Notes

The instruction should be used in the **Init:** section.

Labels are set with **P2_ARINC_Set_Labels**.

If receiver decoding is enabled, the receiver will only copy ARINC messages to the receiver Fifo which match the bit pattern decoder_value in the SDI bits. The receiver will omit any other ARINC message.

If both, label matching and receiver decoding are enabled, an incoming message must match both conditions. If not, the receiver will omit the message.

Note, that the module receivers will always check for parity. Receiver parity check cannot be disabled.

See also

P2_ARINC_Reset, P2_ARINC_Config_Transmit, P2_ARINC_Read_Receive_Fifo, P2_ARINC_Set_Labels

Valid for

ARINC-429 Rev. E

Example

```
#Include ADwinPro_All.Inc
#Define mod_adr 4
#Define labels_1 Data_1
#Define labels_2 Data_2
Dim i As Long
Dim labels_1[16] As Long  'labels array for channel 1
Dim labels_2[16] As Long  'labels array for channel 2

Init:
  Rem Initialize ARINC module
  P2_ARINC_Reset(mod_adr)

  Rem Initialize labels arrays
  For i = 1 To 16
    labels_1[i] = 0
    labels_2[i] = 3
  Next i
  Rem add some more labels
  labels_1[2] = 15
  labels_1[3] = 143        'channel 1 uses labels 0, 15, 143
  P2_ARINC_Set_Labels(mod_adr, 1, labels_1, 1)
  labels_2[2] = 18         'channel 2 uses labels 3, 18
  P2_ARINC_Set_Labels(mod_adr, 1, labels_2, 1)

  Rem configure receiver 1 to 12.5kHz, enable label matching,
  Rem enable decoding with pattern 01b
  P2_ARINC_Config_Receive(mod_adr, 1, 1, 1, 1, 01b)
  Rem configure receiver 2 to 100kHz, enable label matching,
  Rem disable decoding (provide 00b as dummy value)
  P2_ARINC_Config_Receive(mod_adr, 2, 0, 1, 0, 00b)

Event:
  Rem check for messages in receiver fifo 1
  If (P2_ARINC_Receive_Fifo_Empty(mod_adr, 1) = 0) Then
    Rem Read message from receiver fifo 1 into Par_10
    Par_10 = P2_ARINC_Read_Receive_Fifo(mod_adr, 1)
    Rem split message into label, ssm, sdi, data, and parity
    ARINC_Split_Value32(Par_10, Par_11, Par_12, Par_13,
      Par_14, Par_15)
  EndIf
  Rem same procedure for receiver fifo 2
  If (P2_ARINC_Receive_Fifo_Empty(mod_adr, 2) = 0) Then
    Rem Read message from receiver fifo 1 into Par_20
    Par_20 = P2_ARINC_Read_Receive_Fifo(mod_adr, 2)
    Rem split message into label, ssm, sdi, data, and parity
    ARINC_Split_Value32(Par_20, Par_21, Par_22, Par_23,
      Par_24, Par_25)
  EndIf
```

P2_ARINC_Transmit_Fifo_Full

P2_ARINC_Transmit_Fifo_Full returns if the transmitter Fifo of the ARINC interface on the specified module is full.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = P2_ARINC_Transmit_Fifo_Full(module)
```

Parameters

module	Selected module address (1…15).	LONG
ret_val	Flag, if the transmitter Fifo is full: 0: Transmitter Fifo is not full. 1: Transmitter Fifo is full.	LONG

Notes

The transmitter's FIFO can store 32 words maximum.

Use **P2_ARINC_Transmit_Fifo_Full** to check for free space in the transmitter Fifo before writing new data into with **P2_ARINC_Write_Transmit_Fifo**.

Valid for

ARINC-429 Rev. E

See also

P2_ARINC_Config_Transmit, P2_ARINC_Transmit_Fifo_Empty, P2_ARINC_Write_Transmit_Fifo, ARINC_Create_Value32, P2_ARINC_Transmit_Enable

Example

see P2_ARINC_Config_Transmit

P2_ARINC_Transmit_Fifo_Empty

P2_ARINC_Transmit_Fifo_Empty returns if the transmitter Fifo of the ARINC interface on the specified module is empty.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = P2_ARINC_Transmit_Fifo_Empty(module)
```

Parameters

module	Selected module address (1…15).	LONG
ret_val	Flag, if the transmitter Fifo is empty: 0: Transmitter Fifo contains data. 1: Transmitter Fifo is empty.	LONG

Notes

The transmitter's FIFO can store 32 words maximum.

Use **P2_ARINC_Transmit_Fifo_Empty** to check if all data has been sent from the transmitter Fifo.

Valid for

ARINC-429 Rev. E

See also

P2_ARINC_Config_Transmit, P2_ARINC_Transmit_Fifo_Full, P2_ARINC_Write_Transmit_Fifo, ARINC_Create_Value32, P2_ARINC_Transmit_Enable

Example

see P2_ARINC_Config_Transmit

P2_ARINC_Write_Transmit_Fifo

P2_ARINC_Write_Transmit_Fifo writes a 32 bit value into the transmitter Fifo of the ARINC interface on the specified module.

Syntax

```
#Include ADwinPro_All.inc
P2_ARINC_Write_Transmit_Fifo(module, value)
```

Parameters

module	Selected module address (1…15).	LONG
value	32 bit value to be sent, format see below.	LONG

Notes

Use **P2_ARINC_Transmit_Fifo_Full** to check for free space in the transmitter Fifo before writing a new value into.

The transmitter Fifo can store 32 words maximum and ignores attempts to load additional data if full.

Use **ARINC_Create_Value32** to create a 32 bit value correctly. If you want to set the value's bits on your own, set bits in the following order:

	MSB										LSB
Bit	31	30	…	14	13	12	11	10	9	8	7…0
	MSB	Data			LSB	SDI		SSM		P	Label

If parity check is enabled with **P2_ARINC_Config_Transmit**, the parity bit (P) is automatically calculated and set, after writing the value to the transmitter Fifo. If parity is disabled, the 32 bit value is sent unchanged.

If the transmitter is enabled, the values in the transmitter fifo are sent to the ARINC bus as soon as possible.

Valid for

ARINC-429 Rev. E

See also

P2_ARINC_Config_Transmit, P2_ARINC_Transmit_Fifo_Full, P2_ARINC_Transmit_Fifo_Empty, P2_ARINC_Transmit_Enable, ARINC_Create_Value32, P2_ARINC_Read_Receive_Fifo

Example

see P2_ARINC_Config_Transmit

ARINC_Create_Value32

ARINC_Create_Value32 creates a 32 bit value from the components SSM, SDI, data value, and label.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = ARINC_Create_Value32(arinc_label, ssm, sdi,
      data)
```

Parameters

arinc_label	ARINC label, 8 bits.	LONG
ssm	Sign/status matrix, 2 bits.	LONG
sdi	Source/destination identifier, 2 bits.	LONG
data	Data value, 19 bits.	LONG
ret_val	32 bit value, format see below.	LONG

Notes

Be sure that unused bits in all passed parameters are set to zero, else the created value will be false.

The format of the created value refers to the following table.

	MSB										LSB
Bit	31	30	…	14	13	12	11	10	9	8	7…0
	MSB		Data		LSB	SDI		SSM		P	Label

The parity bit (P) is set to zero. When you write the value to the transmitter Fifo with **P2_ARINC_Write_Transmit_Fifo**, the parity bit is automatically calculated and set (only if parity check is enabled, see **P2_ARINC_Config_Transmit**).

To split a value into its components, use **ARINC_Split_Value32**.

See also

P2_ARINC_Reset, P2_ARINC_Config_Transmit, P2_ARINC_Transmit_Fifo_Full, P2_ARINC_Transmit_Fifo_Empty, P2_ARINC_Write_Transmit_Fifo, ARINC_Split_Value32

Valid for

ARINC-429 Rev. E

Example

see P2_ARINC_Config_Transmit

P2_ARINC_ Transmit_Enable

P2_ARINC_Transmit_Enable enables or disables transmitting from the transmitter Fifo of the ARINC interface on a specified module.

Syntax

```
#Include ADwinPro_All.Inc
P2_ARINC_Transmit_Enable(module, enable)
```

Parameters

module	Selected module address (1…15).	LONG
enable	Flag (0,1) if transmitting data is enabled or disabled: 0: Transmitting is disabled. 1: Transmitting is enabled.	LONG

Notes

If the transmitter is enabled, the values in the transmitter fifo are sent to the ARINC bus as soon as possible.

See also

P2_ARINC_Reset, P2_ARINC_Config_Transmit, P2_ARINC_Transmit_Fifo_Full, P2_ARINC_Transmit_Fifo_Empty, P2_ARINC_Write_Transmit_Fifo

Valid for

ARINC-429 Rev. E

Example

see P2_ARINC_Config_Transmit

P2_ARINC_Receive_Fifo_Empty

P2_ARINC_Receive_Fifo_Empty returns if the receiver Fifo of the ARINC interface on the specified module is empty.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = P2_ARINC_Receive_Fifo_Empty(module,
      channel)
```

Parameters

module	Selected module address (1…15).	LONG
channel	Number (1, 2) of the receiver channel.	LONG
ret_val	Flag, if the receiver Fifo is empty: 0: Receiver Fifo contains data. 1: Receiver Fifo is empty.	LONG

Notes

Use **P2_ARINC_Receive_Fifo_Empty** to check for data in the transmitter Fifo before reading a new value.

See also

P2_ARINC_Config_Receive, P2_ARINC_Transmit_Fifo_Empty, P2_ARINC_Read_Receive_Fifo, P2_ARINC_Set_Labels

Valid for

ARINC-429 Rev. E

Example

see P2_ARINC_Config_Receive

P2_ARINC_Read_Receive_Fifo

P2_ARINC_Read_Receive_Fifo returns a 32 bit value from a receiver Fifo of the ARINC interface on the specified module.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = P2_ARINC_Read_Receive_Fifo(module,
      channel)
```

Parameters

module	Selected module address (1…15).	LONG
channel	Number (1, 2) of the receiver channel.	LONG
ret_val	Read 32 bit value.	LONG

Notes

Use **P2_ARINC_Receive_Fifo_Empty** to check for data in the receiver Fifo before reading a new value.

A receiver Fifo can store 32 words maximum.

If the receiver Fifo is empty, **P2_ARINC_Read_Receive_Fifo** returns the most recently received 32 bit value.

Use **ARINC_Split_Value32** to split the 32 bit value into its components. If you want to evaluate the bits of ret_val on your own, refer to the following bit order:

	MSB										LSB
Bit	31	30	…	14	13	12	11	10	9	8	7…0
	MSB	Data			LSB	SDI		SSM		P	Label

See also

P2_ARINC_Config_Receive, P2_ARINC_Write_Transmit_Fifo, P2_ARINC_Receive_Fifo_Empty, ARINC_Split_Value32, P2_ARINC_Set_Labels

Valid for

ARINC-429 Rev. E

Example

see P2_ARINC_Config_Receive

ARINC_Split_Value32

ARINC_Split_Value32 splits a 32 bit ARINC value into its components label, SSM, SDI, data value, and parity bit.

Syntax

```
#Include ADwinPro_All.Inc
ARINC_Split_Value32(value, arinc_label, ssm, sdi,
      data, parity)
```

Parameters

value	32 bit ARINC value, format see below.	LONG
arinc_label	ARINC label, 8 bits.	LONG
ssm	Sign/status matrix, 2 bits.	LONG
sdi	Source/destination identifier, 2 bits.	LONG
data	Data value, 19 bits.	LONG
parity	Parity bit. 0: value has odd parity. 1: value has even parity.	LONG

Notes

The evaluation of values refers to the following bit order format:

	MSB										LSB
Bit	31	30	…	14	13	12	11	10	9	8	7…0
	MSB		Data		LSB	SDI		SSM		P	Label

The module receivers will always check for parity and change the parity bit (P) appropriately. Receiver parity check cannot be disabled.

To create a 32 bit value, use **ARINC_Create_Value32**.

See also

P2_ARINC_Config_Receive, P2_ARINC_Read_Receive_Fifo, P2_ARINC_Receive_Fifo_Empty, P2_ARINC_Set_Labels, ARINC_Create_Value32

Valid for

ARINC-429 Rev. E

Example

see P2_ARINC_Config_Receive

P2_ARINC_Set_Labels

P2_ARINC_Set_Labels sets all 16 labels of a receiver on the specified ARINC module.

Syntax

```
#Include ADwinPro_All.Inc
P2_ARINC_Set_Labels(module, channel, labels[],
      array_index)
```

Parameters

module	Selected module address (1…15).	LONG
channel	Number (1, 2) of the receiver channel.	LONG
labels[]	Source array holding the label values (8 bit length) to be set. The array size must be at least 16.	LONG
array_index	First value in labels[] to be read.	LONG

Notes

Labels are only used if you have enabled label matching with **P2_ARINC_Config_Receive**, enable_label=1.

If label matching is enabled, the receiver will only accept ARINC messages which match one of the set labels. The receiver will omit any other ARINC message.

Each label has 8 bit length. In an array element of labels[], a label is positioned in the bits 0…7.

With **P2_ARINC_Set_Labels**, you set all 16 label at the same time. In order to use less than 16 labels, fill not array elements labels with one of the used label values.
Example: If you want to use 2 labels, set the values of label 1 and 2, and repeat label 2 (or label 1) 14 times to fill up to 16 label values.

See also

P2_ARINC_Reset, P2_ARINC_Config_Receive, P2_ARINC_Receive_Fifo_Empty, P2_ARINC_Read_Receive_Fifo

Valid for

ARINC-429 Rev. E

Example

see P2_ARINC_Config_Receive

3.16 Pro II: EtherCAT interface

This section contains instructions to access a EtherCAT interface of *ADwin-Pro II*.

– P2_ECAT_Get_Version (page 322)
– P2_ECAT_Get_State (page 323)
– P2_ECAT_Init (page 324)
– P2_ECAT_Set_Mode (page 326)
– P2_ECAT_Read_Data_16L (page 327)
– P2_ECAT_Write_Data_16L (page 328)
– P2_ECAT_Read_Data_16F (page 329)
– P2_ECAT_Write_Data_16F (page 330)

P2_ECAT_Get_Version

P2_ECAT_Get_Version returns the version of the EtherCAT interface.

Syntax

```
#Include ADwinPro_All.inc
ret_val = P2_ECAT_Get_Version(ecat_datatable[])
```

Parameters

ecat_datatable[]	Array which contains settings for data transfer between *ADwin* CPU and the EtherCAT module.	ARRAY LONG
ret_val	Version number of the EtherCAT interface, to be read in hexadecimal notation.	LONG

Notes

The version number is only required if you have questions about programming the EtherCAT bus to our support.

The version number (in hexadecimal notation) has five digits, e.g. **10000h**; the first digits is the main revision number.

Valid for

EtherCAT-SL Rev. E

See also

P2_ECAT_Init

Example

see P2_ECAT_Init

P2_ECAT_Get_State

P2_ECAT_Get_State returns the operation mode of the EtherCAT interface.

Syntax

```
#Include ADwinPro_All.inc
ret_val = P2_ECAT_Get_State(ecat_datatable[])
```

Parameters

module	Selected module address (1…15).	LONG
ecat_datatable[]	Array which contains settings for data transfer between *ADwin* CPU and the EtherCAT module.	ARRAY LONG
ret_val	Operation mode of the EtherCAT interface : 1: Operation mode Init. 2: Operation mode PreOp. 3: Operation mode Boot. 4: Operation mode SafeOp. 8: Operation mode Op.	LONG

Notes

The operation mode Boot is not supported in *ADbasic*.

Valid for

EtherCAT-SL Rev. E

See also

P2_ECAT_Init, P2_ECAT_Set_Mode, P2_ECAT_Write_Data_16L

Example

see P2_ECAT_Init

P2_ECAT_Init

P2_ECAT_Init initializes the EtherCAT Slave.

Syntax

```
#Include ADwinPro_All.inc
ret_val = P2_ECAT_Init(module, ecat_datatable[])
```

Parameters

module	Selected module address (1…15).	LONG
ecat_datatable[]	Array which contains settings for data transfer between *ADwin* CPU and the EtherCAT module.	ARRAY LONG
ret_val	Status of initialization: 0: no error. 1: invalid module.	LONG

Notes

This instruction must be run before using the EtherCAT Slave.

Valid for

EtherCAT-SL Rev. E

See also

P2_ECAT_Get_Version, P2_ECAT_Get_State, P2_ECAT_Set_Mode, P2_ECAT_Write_Data_16L

Example

```
#Include ADwinPro_All.INC

#Define ECAT_MODULE_ADDRESS 5
#Define ecat_inputs Data_1
#Define ecat_outputs Data_2

Dim ecat_inputs[16] As Long
Dim ecat_outputs[16] As Long
Dim ecat_comtable[150] As Long
Dim i As Long
Dim ret As Long

Init:
  Processdelay = 300000     ' 1kHz
  Rem initialize data transfer T11 <-> TiCo
  Par_1 = P2_ECAT_Init(ECAT_MODULE_ADDRESS, ecat_comtable)
  Par_2 = P2_ECAT_Get_Version(ecat_comtable) '10000h

  For i = 1 To 16
    ecat_inputs[i] = 0
  Next
  For i = 1 To 16
    ecat_outputs[i] = i
  Next
  Par_11 = 0
  Par_12 = 0

Event:
  ret = P2_ECAT_Get_State(ecat_comtable)

  If (ret = 8) Then         'operational mode
    ret = P2_ECAT_Write_Data_16L(ecat_comtable, ecat_outputs)
    If (ret = 0) Then       'writing data was o.k.
      Inc Par_11            'increase write counter
    EndIf

    ret = P2_ECAT_Read_Data_16L(ecat_comtable, ecat_inputs)
    If (ret = 0) Then       'reading data was o.k.
      Inc Par_12            'increase read counter
    EndIf
  EndIf
```

P2_ECAT_Set_Mode

P2_ECAT_Set_Mode sets the data transfer mode of the EtherCAT slave.

Syntax

```
#Include ADwinPro_All.inc
P2_ECAT_Set_Mode(ecat_datatable[], mode)
```

Parameters

ecat_datatable[]	Array which contains settings for data transfer between *ADwin* CPU and the EtherCAT module.	ARRAY LONG
mode	Data transfer mode of the EtherCAT slave: 0: data type Long 1: data type Float 2: data types Long and Float	LONG

Notes

This function is available since firmware version 2.0.

Valid for

EtherCAT-SL Rev. E

See also

P2_ECAT_Set_Mode, P2_ECAT_Write_Data_16L, P2_ECAT_Read_Data_16F, P2_ECAT_Write_Data_16F, P2_ECAT_Get_Version

Example

see P2_ECAT_Init

P2_ECAT_Read_Data_16L

P2_ECAT_Read_Data_16L reads 16 Long values from the EtherCAT slave and returns them in an array.

Syntax

```
#Include ADwinPro_All.inc
ret_val = P2_ECAT_Read_Data_16L(ecat_datatable[],
    ecat_inputs[])
```

Parameters

ecat_datatable[]	Array which contains settings for data transfer between *ADwin* CPU and the EtherCAT module.	ARRAY LONG
ecat_inputs[]	Array where the EtherCAT slave writes the Long data.	ARRAY LONG
ret_val	Status of reading: 0: Reading was successful. ≠0: Error while reading data.	LONG

Notes

- / -

Valid for

EtherCAT-SL Rev. E

See also

P2_ECAT_Get_State, P2_ECAT_Init, P2_ECAT_Write_Data_16L

Example

see P2_ECAT_Init

P2_ECAT_Write_Data_16L

P2_ECAT_Write_Data_16L writes 16 Long values from an array to the EtherCAT slave.

Syntax

```
#Include ADwinPro_All.inc
ret_val = P2_ECAT_Write_Data_16L(ecat_datatable[],
    ecat_outputs[])
```

Parameters

ecat_datatable[]	Array which contains settings for data transfer between *ADwin* CPU and the EtherCAT module.	ARRAY LONG
ecat_outputs[]	Array from where the EtherCAT slave reads the Long data and writes them to the EtherCAT bus.	ARRAY LONG
ret_val	Status of writing: 0: Writing was successful. ≠0: Error while writing data.	LONG

Notes

- / -

Valid for

EtherCAT-SL Rev. E

See also

P2_ECAT_Get_State, P2_ECAT_Init, P2_ECAT_Set_Mode

Example

see P2_ECAT_Init

P2_ECAT_Read_Data_16F

P2_ECAT_Read_Data_16F reads 16 Float values from the EtherCAT slave and returns them in an array.

Syntax

```
#Include ADwinPro_All.inc
ret_val = P2_ECAT_Read_Data_16F(ecat_datatable[],
    ecat_inputs[])
```

Parameters

ecat_datatable[]	Array which contains settings for data transfer between *ADwin* CPU and the EtherCAT module.	ARRAY LONG
ecat_inputs[]	Array where the EtherCAT slave writes the Float data.	ARRAY LONG
ret_val	Status of reading: 0: Reading was successful. ≠0: Error while reading data.	LONG

Notes

This function is available since firmware version 2.0.

Valid for

EtherCAT-SL Rev. E

See also

P2_ECAT_Get_State, P2_ECAT_Init, P2_ECAT_Write_Data_16L

Example

see P2_ECAT_Init

P2_ECAT_Write_Data_16F

P2_ECAT_Write_Data_16F writes 16 Float values from an array to the EtherCAT slave.

Syntax

```
#Include ADwinPro_All.inc
ret_val = P2_ECAT_Write_Data_16F(ecat_datatable[],
    ecat_outputs[])
```

Parameters

ecat_datatable[]	Array which contains settings for data transfer between *ADwin* CPU and the EtherCAT module.	ARRAY LONG
ecat_outputs[]	Array from where the EtherCAT slave reads the Float data and writes them to the EtherCAT bus.	ARRAY LONG
ret_val	Status of writing: 0: Writing was successful. ≠0: Error while writing data.	LONG

Notes

This function is available since firmware version 2.0.

Valid for

EtherCAT-SL Rev. E

See also

P2_ECAT_Get_State, P2_ECAT_Init, P2_ECAT_Set_Mode

Example

see P2_ECAT_Init

3.17 Pro II: FlexRay

This section contains instructions to access a FlexRay interface of *ADwin-Pro II*:

– P2_FlexRay_Get_Version (page 332)
– P2_FlexRay_Init (page 333)
– P2_FlexRay_Read_Word (page 335)
– P2_FlexRay_Reset (page 336)
– P2_FlexRay_Set_LED (page 337)
– P2_FlexRay_Write_Word (page 338)

In the Instruction List sorted by Module Types (annex A.2) you will find which of the functions corresponds to the *ADwin-Pro II* modules.

P2_FlexRay_Get_Version

P2_FlexRay_Get_Version returns the version number of the FlexRay interface.

Syntax

```
#Include ADwinPro_All.inc
ret_val = P2_FlexRay_Get_Version(fr_datatable[],
    status)
```

Parameters

fr_datatable[]	Array which contains settings for data transfer between *ADwin* CPU and the FlexRay module.	ARRAY LONG
status	Status of access to the FlexRay module: 0: Access was successful. 1: Error: FlexRay controller was busy. 2: Error: FlexRay controller has not responded in time.	LONG
ret_val	Version number of FlexRay firmware.	LONG

Notes

The version number will only be used, if you have questions about programming the FlexRay module to our support.

Each 4 hexadecimal numerals represent the version numbers of the high level and of the low level driver. For example **01030205h** represents the versions 1.3 (high level) and 2.5 (low level).

See also

P2_FlexRay_Init

Valid for

FlexRay-2 Rev. E

Example

- / -

P2_FlexRay_Init

P2_FlexRay_Init initializes the data transfer between *ADwin* CPU and the FlexRay interface on a specified module.

Syntax

```
#Include ADwinPro_All.inc

REM communication settings array of FlexRay module
Dim fr_datatable[150] As Long

P2_FlexRay_Init(module, fr_datatable[], status)
```

Parameters

module	Selected module address (1…15).	LONG
fr_datatable[]	Array which contains settings for data transfer between *ADwin* CPU and the FlexRay module.	ARRAY LONG
status	Status of access to the FlexRay module: 0: Access was successful. 1: Error: No Pro II module at this address. 2: Error: no FlexRay interface on the module.	LONG

Notes

P2_FlexRay_Init is to be executed before data transfer between *ADwin* CPU and FlexRay interface. The instruction should be used in the **Init:** section.

Before initialization, an array fr_datatable[] with 150 elements must be declared for each module.

See also

P2_FlexRay_Read_Word, P2_FlexRay_Reset, P2_FlexRay_Set_LED, P2_FlexRay_Write_Word

Valid for

FlexRay-2 Rev. E

Example

```
#Include ADwinPro_All.inc
Dim fr_datatable[150] As Long
Dim status, value As Long

Init:
  Rem initialize communication to the FlexRay controller
  P2_FlexRay_Init(1, fr_datatable, status)
  If (status <> 0) Then Exit

Event:
  Rem read address 210h from controller 1
  value = P2_FlexRay_Read_Word(fr_datatable,1,210h,status)
  If (status <> 0) Then End
  If (value = 15) Then
    Rem read address 220h from controller 1
    value = P2_FlexRay_Read_Word(fr_datatable,1,220h,status)
  Else
    Rem write value to address 192h of controller 1
    P2_FlexRay_Write_Word(fr_datatable,1,192h,value,status)
  EndIf

Finish:
  If (status <> 0) Then
    Rem set Par_1 to error number
    Par_1 = status
  EndIf
```

P2_FlexRay_Read_Word

P2_FlexRay_Read_Word returns a 16 bit value from a FlexRay controller on the specified module.

Syntax

```
#Include ADwinPro_All.inc
ret_val = P2_FlexRay_Read_Word(fr_datatable[],
    controller, address, status)
```

Parameters

fr_datatable[]	Array which contains settings for data transfer between *ADwin* CPU and the FlexRay module.	ARRAY LONG
controller	Number (1, 2) of the FlexRay controller.	LONG
address	Address (0…1FFEh) on the FlexRay controller, whose value is read. Enter the address with 2 byte alignment.	LONG
status	Status of access to the FlexRay module: 0: Access was successful. 1: Error: FlexRay controller was busy. 2: Error: FlexRay controller has not responded in time.	LONG
ret_val	Content (16 bit value) of the address on the FlexRay controller.	LONG

Notes

- / -

See also

P2_FlexRay_Init, P2_FlexRay_Reset, P2_FlexRay_Write_Word

Valid for

FlexRay-2 Rev. E

Example

see P2_FlexRay_Init

P2_FlexRay_Reset

P2_FlexRay_Reset resets a FlexRay controller on the spcified module.

Syntax

```
#Include ADwinPro_All.inc
P2_FlexRay_Reset(fr_datatable[], controller, status)
```

Parameters

fr_datatable[]	Array which contains settings for data transfer between *ADwin* CPU and the FlexRay module.	ARRAY LONG
controller	Number (1, 2) of the FlexRay controller.	LONG
status	Status of access to the FlexRay module: 0: Access was successful. 1: Error: FlexRay controller was busy. 2: Error: FlexRay controller has not responded in time.	LONG

Notes

- / -

See also

P2_FlexRay_Init, P2_FlexRay_Read_Word, P2_FlexRay_Write_Word

Valid for

FlexRay-2 Rev. E

Example

see P2_FlexRay_Init

P2_FlexRay_Set_LED

P2_FlexRay_Set_LED switches a channel LED of a FlexRay controller on the specified module on or off.

Syntax

```
#Include ADwinPro_All.inc
P2_FlexRay_Set_LED(module, controller, channel,
    value, status)
```

Parameters

fr_datatable[]	Array which contains settings for data transfer between *ADwin* CPU and the FlexRay module.	ARRAY LONG
controller	Number (1, 2) of the FlexRay controller.	LONG
channel	Number (1, 2) of the FlexRay channel. 1: Channel A. 2: Channel B.	LONG
value	Status of the LED: 0: LED off. 1: LED on.	LONG
status	Status of access to the FlexRay module: 0: Access was successful. 1: Error: FlexRay controller was busy. 2: Error: FlexRay controller has not responded in time.	LONG

Notes

- / -

See also

P2_FlexRay_Init

Valid for

FlexRay-2 Rev. E

Example

```
#Include ADwinPro_All.inc
Dim fr_datatable[150] As Long
Dim status As Long

Init:
  Rem FlexRay-Controller initialisieren
  P2_FlexRay_Init(1, fr_datatable, status)
  Rem LED für Kanal 2, Controller 1 einschalten
  P2_FlexRay_Set_LED(fr_datatable,1,2,1,status)
```

P2_FlexRay_Write_Word

P2_FlexRay_Write_Word writes a 16 bit value to an address in a FlexRay controller of the specified module.

Syntax

```
#Include ADwinPro_All.inc
P2_FlexRay_Write_Word(fr_datatable[],
    controller, address, value, status)
```

Parameters

fr_datatable[]	Array which contains settings for data transfer between *ADwin* CPU and the FlexRay module.	ARRAY LONG
controller	Number (1, 2) of the FlexRay controller.	LONG
address	Address (0…1FFEh) on the FlexRay controller, where the value is written. Enter the address with 2 byte alignment.	LONG
value	16 bit value, which is writtten to the address in the FlexRay controller.	LONG
status	Status of access to the FlexRay module: 0: Access was successful. 1: Error: FlexRay controller was busy. 2: Error: FlexRay controller has not responded in time.	LONG

Notes

- / -

See also

P2_FlexRay_Init, P2_FlexRay_Read_Word, P2_FlexRay_Reset

Valid for

FlexRay-2 Rev. E

Example

see P2_FlexRay_Init

3.18 Pro II: SENT Interface

This section contains instructions to access a SENT interface of *ADwin-Pro II*:

SENT Inputs

- P2_SENT_Init (page 340)
- P2_SENT_Get_Msg_Counter (page 342)
- P2_SENT_Command_Ready (page 343)
- P2_SENT_Get_Version (page 341)
- P2_SENT_Get_ChannelState (page 344)
- P2_SENT_Get_ClockTick (page 345)
- P2_SENT_Get_PulseCount (page 346)
- P2_SENT_Get_Fast_Channel_CRC_OK (page 347)
- P2_SENT_Get_Fast_Channel1 (page 348)
- P2_SENT_Get_Fast_Channel2 (page 350)
- P2_SENT_Get_Serial_Message_CRC_OK (page 351)
- P2_SENT_Get_Serial_Message_Id (page 352)
- P2_SENT_Get_Serial_Message_Data (page 353)
- P2_SENT_Get_Serial_Message_Array (page 354)
- P2_SENT_Clear_Serial_Message_Array (page 356)
- P2_SENT_Set_CRC_Implementation (page 357)
- P2_SENT_Set_Detection (page 358)
- P2_SENT_Set_ClockTick (page 359)
- P2_SENT_Set_PulseCount (page 360)
- P2_SENT_Set_Sensor_Type (page 361)
- P2_SENT_Request_Latch (page 362)
- P2_SENT_Check_Latch (page 363)
- P2_SENT_Get_Latch_Data (page 364)

SENT Outputs

- P2_SENT_Set_Output_Mode (page 368)
- P2_SENT_Get_Output_Mode (page 369)
- P2_SENT_Config_Output (page 370)
- P2_SENT_Config_Serial_Messages (page 371)
- P2_SENT_Enable_Channel (page 372)
- P2_SENT_Invert_Channel (page 373)
- P2_SENT_Set_Reserved_Bits (page 374)
- P2_SENT_Set_Fast_Channel1 (page 375)
- P2_SENT_Set_Fast_Channel2 (page 376)
- P2_SENT_Set_Serial_Message_Pattern (page 377)
- P2_SENT_Set_Serial_Message_Data (page 379)
- P2_SENT_Fifo_Empty (page 380)
- P2_SENT_Fifo_Clear (page 381)
- P2_SENT_Set_Fifo (page 382)

In the Instruction List sorted by Module Types (annex A.2) you will find which of the functions corresponds to the *ADwin-Pro II* modules.

P2_SENT_Init

P2_SENT_Init initializes the data transfer between *ADwin* CPU and the SENT interface on a specified module.

Syntax

```
#Include ADwinPro_All.Inc

REM define SENT settings array
Dim sent_datatable[150] As Long

P2_SENT_Init(module, sent_datatable[])
```

Parameters

module	Selected module address (1…15).	LONG
sent_datatable[]	Array (size ≥ 150) which contains settings for data transfer between *ADwin* CPU and the SENT module.	ARRAY LONG

Notes

P2_SENT_Init is to be executed before data transfer between *ADwin* CPU and SENT interface. The instruction should be used in the **Init:** section.

Before initialization, an array sent_datatable[] with 150 elements must be declared for each module.

Valid for

SENT-4 Rev. E, SENT-4-Out Rev. E, SENT-6 Rev. E

See also

P2_SENT_Get_Serial_Message_Array, P2_SENT_Set_Serial_Message_Pattern, P2_SENT_Get_Latch_Data

Example

see P2_SENT_Get_Latch_Data

P2_SENT_Get_Version

P2_SENT_Get_Version returns the version of the SENT interface on the specified module.

Syntax

```
#Include ADwinPro_All.inc
ret_val = P2_SENT_Get_Version(module)
```

Parameters

module	Selected module address (1…15).	LONG
ret_val	Version number of the SENT interface.	LONG

Notes

The version number will only be used, if you have questions about programming the SENT module to our support.

Valid for

SENT-4 Rev. E, SENT-4-Out Rev. E, SENT-6 Rev. E

See also

P2_SENT_Get_Fast_Channel1, P2_SENT_Get_Fast_Channel2, P2_SENT_Get_Fast_Channel_CRC_OK, P2_SENT_Get_ChannelState, P2_SENT_Get_ClockTick

Example

```
#Include ADwinPro_All.inc

#Define module 2

Init:
  REM get software version
  Par_1 = P2_SENT_Get_Version(module)
```

P2_SENT_Get_Msg_Counter

P2_SENT_Get_Msg_Counter returns the number of sent / received messages on the specified SENT channel.

Syntax

```
#Include ADwinPro_All.inc
ret_val = P2_SENT_Get_Msg_Counter(module,
    sent_channel)
```

Parameters

module	Selected module address (1…15).	LONG
sent_channel	Number (1…4, 1…6) of the SENT channel.	LONG
ret_val	Number of sent / received SENT messages.	LONG

Notes

SENT input modules only: As long as new messages are being received on a SENT channel, the SENT sensor is active. The continuous change of the return value therefore serves as timeout watchdog of the SENT sensor.

Valid for

SENT-4 Rev. E, SENT-4-Out Rev. E, SENT-6 Rev. E

See also

P2_SENT_Get_ChannelState, P2_SENT_Get_ClockTick, P2_SENT_Get_PulseCount, P2_SENT_Get_Output_Mode

Example

see P2_SENT_Get_Fast_Channel1

P2_SENT_Command_Ready

P2_SENT_Command_Ready returns whether the SENT interface on the specified module is ready to process the next command.

Syntax

```
#Include ADwinPro_All.inc
ret_val = P2_SENT_Command_Ready(module)
```

Parameters

module	Selected module address (1…15).	LONG
ret_val	Status of the interface: 0: Interface is ready to process next command. ≠0: Interface is busy.	LONG

Notes

Use **P2_SENT_Command_Ready** each time before processing any of the following commands:

- **P2_SENT_Clear_Serial_Message_Array**
- **P2_SENT_Set_CRC_Implementation**
- **P2_SENT_Set_Detection**
- **P2_SENT_Set_ClockTick**
- **P2_SENT_Set_PulseCount**
- **P2_SENT_Set_Sensor_Type**
- **P2_SENT_Request_Latch**
- **P2_SENT_Config_Output**
- **P2_SENT_Config_Serial_Messages**
- **P2_SENT_Set_Serial_Message_Pattern**
- **P2_SENT_Set_Serial_Message_Data**
- **P2_SENT_Fifo_Clear**
- **P2_SENT_Set_Fifo**

If you process a command though the interface is still busy, the command will be ignored.

Valid for

SENT-4 Rev. E, SENT-4-Out Rev. E, SENT-6 Rev. E

See also

P2_SENT_Clear_Serial_Message_Array, P2_SENT_Set_CRC_Implementation, P2_SENT_Set_Detection, P2_SENT_Set_ClockTick, P2_SENT_Set_PulseCount, P2_SENT_Set_Sensor_Type, P2_SENT_Request_Latch, P2_SENT_Config_Output, P2_SENT_Config_Serial_Messages, P2_SENT_Set_Serial_Message_Pattern, P2_SENT_Set_Serial_Message_Data, P2_SENT_Fifo_Clear, P2_SENT_Set_Fifo

Example

```
#Include ADwinPro_All.inc
#Define module 2
#Define channel 1

Init:
  Rem set CRC mode to 'recommended'
  Do : Until (P2_SENT_Command_Ready(module) = 0)
  Par_1 = P2_SENT_Set_CRC_Implementation(module, channel, 1)
```

P2_SENT_Get_ChannelState

P2_SENT_Get_ChannelState returns the receive modes of the SENT channels of the specified module.

Syntax

```
#Include ADwinPro_All.inc
ret_val = P2_SENT_Get_ChannelState(module)
```

Parameters

module	Selected module address (1…15).	LONG
ret_val	Bit pattern with operating modes of the SENT channels. Bit = 0: Detection mode. Bit = 1: Read mode.	LONG

Bit no.	31…6	5	4	3	2	1	0
SENT channel	–	6	5	4	3	2	1

Notes

After power-on all input channels are set to detection mode, where incoming SENT messages are analyzed. As soon as the module detects the clock tick and the pulse number of a message, it switches the input channel to read mode.

Valid for

SENT-4 Rev. E, SENT-6 Rev. E

See also

P2_SENT_Get_Fast_Channel1, P2_SENT_Get_Fast_Channel2, P2_SENT_Get_Fast_Channel_CRC_OK, P2_SENT_Get_PulseCount, P2_SENT_Get_ClockTick

Example

see P2_SENT_Get_Fast_Channel1

P2_SENT_Get_ClockTick

P2_SENT_Get_ClockTick returns the clock tick of a SENT input channel on the specified module.

Syntax

```
#Include ADwinPro_All.inc
ret_val = P2_SENT_Get_ClockTick(module,
    sent_channel)
```

Parameters

module	Selected module address (1…15).	LONG
sent_channel	Number (1…4, 1…6) of the SENT channel.	LONG
ret_val	Clock tick (1500…90000) in ns.	LONG

Notes

The clock tick can be different for every SENT input channel.

After power-on of the module the clock tick of an incoming SENT messages is detected automatically (detection mode). Alternatively, you cn set the clock tick with **P2_SENT_Set_ClockTick** manually.

Valid for

SENT-4 Rev. E, SENT-6 Rev. E

See also

P2_SENT_Set_ClockTick, P2_SENT_Command_Ready, P2_SENT_Get_Fast_Channel1, P2_SENT_Get_Fast_Channel2, P2_SENT_Get_Fast_Channel_CRC_OK, P2_SENT_Get_ChannelState, P2_SENT_Get_PulseCount

Example

see P2_SENT_Get_Fast_Channel1

P2_SENT_Get_PulseCount

P2_SENT_Get_PulseCount returns the number of pulses in a SENT message on an input channel of the specified module.

Syntax

```
#Include ADwinPro_All.inc
ret_val = P2_SENT_Get_PulseCount(module,
    sent_channel)
```

Parameters

module	Selected module address (1…15).	LONG
sent_channel	Number (1…4, 1…6) of the SENT channel.	LONG
ret_val	Number (9, 10) of pulses in the message: 9: SENT signal without pause pulse. 10:SENT signal with pause pulse.	LONG

Notes

A SENT message consists of several pulses. The return value of **P2_SENT_Get_PulseCount** is the number of pulses in the recently received SENT message.

For SENT messages with two 12 bit values and a pause pulse the message length results to 10 pulses:

- Calibration pulse for synchronization
- 1 nibble pulse (=4 bit): Status and communication
- 3 nibble pulses: first 12 bit value (fast channel 1)
- 3 nibble pulses: second 12 bit value (fast channel 2)
- 1 nibble pulse: checksum
- pause pulse (optional)

Valid for

SENT-4 Rev. E, SENT-6 Rev. E

See also

P2_SENT_Set_PulseCount, P2_SENT_Get_Fast_Channel1, P2_SENT_Get_Fast_Channel2, P2_SENT_Get_Fast_Channel_CRC_OK, P2_SENT_Get_ChannelState, P2_SENT_Get_ClockTick

Example

see P2_SENT_Get_Fast_Channel1

P2_SENT_Get_Fast_Channel_CRC_OK

P2_SENT_Get_Fast_Channel_CRC_OK returns the result of the CRC check for the signals of a SENT message on a channel of the specified module.

Syntax

```
#Include ADwinPro_All.inc
ret_val = P2_SENT_Get_Fast_Channel_CRC_OK(
    module, sent_channel)
```

Parameters

module	Selected module address (1…15).	LONG
sent_channel	Number (1…4, 1…6) of the SENT channel.	LONG
ret_val	Result of the CRC check: 0: Data transfer was successful. 1: Checksum error, error during data transfer.	LONG

Notes

The checksum refers to a complete SENT message with two 12 bit values (fast channels).

The module calculates the CRC checksum from the two 12 bit values (fast channels) and compares the result to the CRC checksum in the SENT message. Only if both checksums are equal the return value is set to 0.

Valid for

SENT-4 Rev. E, SENT-6 Rev. E

See also

P2_SENT_Get_Fast_Channel1, P2_SENT_Get_Fast_Channel2, P2_SENT_Get_Serial_Message_CRC_OK, P2_SENT_Get_Serial_Message_Id

Example

see P2_SENT_Get_Fast_Channel1

P2_SENT_Get_Fast_Channel1

P2_SENT_Get_Fast_Channel1 reads the first 12 bit value from a SENT channel on the specified module.

Syntax

```
#Include ADwinPro_All.inc
ret_val = P2_SENT_Get_Fast_Channel1(module,
    sent_channel)
```

Parameters

module	Selected module address (1…15).	LONG
sent_channel	Number (1…4, 1…6) of the SENT channel.	LONG
ret_val	12 bit value (0…4095).	LONG

Notes

A SENT message contains two 12 bit values also named "fast channel signals". The instruction returns the first of both values.

Valid for

SENT-4 Rev. E, SENT-6 Rev. E

See also

P2_SENT_Get_Fast_Channel2, P2_SENT_Get_Fast_Channel_CRC_OK, P2_SENT_Get_Serial_Message_CRC_OK, P2_SENT_Get_Serial_Message_Id

Example

```
#Include ADwinPro_All.inc

#Define module 2
#Define channel 1
#Define ready_state Par_20
#Define returncode Par_21
#Define status Par_22
#Define value1 Par_23
#Define value2 Par_24
#Define msg_cnt Par_25
#Define prev_msg_cnt Par_25

Init:
  Processdelay = 300000     '1 kHz

  REM Wait until detection mode of SENT channel has finished and
  REM read mode is set
  Par_1 = 2^(channel - 1)
  Do
  Until (P2_SENT_Get_ChannelState(module) And Par_1 = Par_1)

  prev_msg_cnt = 0          'message counter (timeout check)

  Rem CRC mode 'recommended'
  Do : Until (P2_SENT_Command_Ready(module) = 0)
  Par_1 = P2_SENT_Set_CRC_Implementation(module, channel, 1)

  Rem read pulse count: 9 = without pause pulse,
  Rem 10 = with pause pulse
  Par_11 = P2_SENT_Get_PulseCount(module, channel)
```

```
  Par_12 = P2_SENT_Get_ClockTick(module, channel) 'read cycle
period

Event:
  Inc Par_2
  If (Par_2 = 1000) Then   'timeout check every second
    Par_2 = 0
    msg_cnt = P2_SENT_Get_Msg_Counter(module, channel)
    If (msg_cnt = prev_msg_cnt) Then
      End                  'no change = SENT sensor timeout, exit
    Else
      prev_msg_cnt = msg_cnt 'store counter for next check
    EndIf
  EndIf

  Rem read CRC status of fast channels
  If (P2_SENT_Get_Fast_Channel_CRC_OK(module, channel) = 0) Then
    Rem read fast channel values 1+2
    Par_13 = P2_SENT_Get_Fast_Channel1(module, channel)
    Par_14 = P2_SENT_Get_Fast_Channel2(module, channel)
  EndIf

  Rem read CRC status of serial message
  If (P2_SENT_Get_Serial_Message_CRC_OK(module, channel)=0) Then
    Rem read serial message ID and data
    Par_16 = P2_SENT_Get_Serial_Message_Id(module, channel)
    Par_17 = P2_SENT_Get_Serial_Message_Data(module, channel)
  EndIf

  Rem -- write commands --
  ready_state = P2_SENT_Command_Ready(module)
  If ((ready_state = 0) And (status <> 0)) Then
    If (status = 1) Then    'reset channel
      value1 = 1
      returncode = P2_SENT_Set_Detection(module, value1)
    EndIf

    If (status = 2) Then    'set clock period
      value1 = 1
      returncode = P2_SENT_Set_ClockTick(module, value1, value2)
    EndIf

    status = 0
  EndIf
```

P2_SENT_Get_Fast_Channel2

P2_SENT_Get_Fast_Channel2 reads the second 12 bit value from a SENT channel on the specified module..

Syntax

```
#Include ADwinPro_All.inc
ret_val = P2_SENT_Get_Fast_Channel2(module,
    sent_channel)
```

Parameters

module	Selected module address (1…15).	LONG
sent_channel	Number (1…4, 1…6) of the SENT channel.	LONG
ret_val	12 bit value (0…4095).	LONG

Notes

A SENT message contains two 12 bit values also named "fast channel signals". The instruction returns the second of both values.

Valid for

SENT-4 Rev. E, SENT-6 Rev. E

See also

P2_SENT_Get_Fast_Channel1, P2_SENT_Get_Fast_Channel_CRC_OK, P2_SENT_Get_Serial_Message_CRC_OK, P2_SENT_Get_Serial_Message_Id

Example

see P2_SENT_Get_Fast_Channel1

P2_SENT_Get_Serial_Message_CRC_OK

P2_SENT_Get_Serial_Message_CRC_OK returns the result of the CRC check for the serial message on a channel of the specified module.

Syntax

```
#Include ADwinPro_All.inc
ret_val = P2_SENT_Get_Serial_Message_CRC_OK(module,
    sent_channel)
```

Parameters

module	Selected module address (1…15).	LONG
sent_channel	Number (1…4, 1…6) of the SENT channel.	LONG
ret_val	Result of the CRC check: 0: Data transfer was successful. 1: Checksum error, error during data transfer.	LONG

Notes

The checksum refers to the most recently receieved, complete serial message. A serial message is being sent distributed over several SENT messages.

The following sending formats of serial messages can be recognized:

- Short Serial Message Format, length 12 bit:
  ID 4 bit and data value 8 bit.
- Enhanced Serial Message Format, 20 bit Länge:
  ID 4 bit, data value 16 bit or ID 8 bit, data value 12 bit.

Valid for

SENT-4 Rev. E, SENT-6 Rev. E

See also

P2_SENT_Get_Serial_Message_Id, P2_SENT_Get_Serial_Message_Data, P2_SENT_Get_Fast_Channel1, P2_SENT_Get_Fast_Channel2, P2_SENT_Get_Fast_Channel_CRC_OK

Example

see P2_SENT_Get_Fast_Channel1

P2_SENT_Get_Serial_Message_Id

P2_SENT_Get_Serial_Message_Id returns the ID of a serial message from a channel of the specified module.

Syntax

```
#Include ADwinPro_All.inc
ret_val = P2_SENT_Get_Serial_Message_Id(module,
    sent_channel)
```

Parameters

module	Selected module address (1…15).	LONG
sent_channel	Number (1…4, 1…6) of the SENT channel.	LONG
ret_val	ID (0…255) of the serial message.	LONG

Notes

The ID refers to the most recently receieved, complete serial message. A serial message is being sent distributed over several SENT messages.

The following sending formats of serial messages can be recognized:

- Short Serial Message Format, length 12 bit:
  ID 4 bit and data value 8 bit.
- Enhanced Serial Message Format, 20 bit Länge:
  ID 4 bit, data value 16 bit or ID 8 bit, data value 12 bit.

Valid for

SENT-4 Rev. E, SENT-6 Rev. E

See also

P2_SENT_Get_Serial_Message_CRC_OK, P2_SENT_Get_Serial_Message_Data, P2_SENT_Get_Fast_Channel1, P2_SENT_Get_Fast_Channel2, P2_SENT_Get_Fast_Channel_CRC_OK

Example

see P2_SENT_Get_Fast_Channel1

P2_SENT_Get_Serial_Message_Data

P2_SENT_Get_Serial_Message_Data returns the data value of a serial message from a channel of the specified module.

Syntax

```
#Include ADwinPro_All.inc
ret_val = P2_SENT_Get_Serial_Message_Data(module,
    sent_channel)
```

Parameters

module	Selected module address (1…15).	LONG
sent_channel	Number (1…4, 1…6) of the SENT channel.	LONG
ret_val	Data value (0…65535) of the serial message.	LONG

Notes

The data value refers to the most recently received, complete serial message. A serial message is being sent distributed over several SENT messages.

The following sending formats of serial messages can be recognized:

- Short Serial Message Format, length 12 bit:
  ID 4 bit and data value 8 bit.
- Enhanced Serial Message Format, 20 bit Länge:
  ID 4 bit, data value 16 bit or ID 8 bit, data value 12 bit.

Use **P2_SENT_Get_Serial_Message_Array** to get a complete pattern of serial messages.

Valid for

SENT-4 Rev. E, SENT-6 Rev. E

See also

P2_SENT_Get_Serial_Message_CRC_OK, P2_SENT_Get_Serial_Message_Id, P2_SENT_Get_Fast_Channel1, P2_SENT_Get_Fast_Channel2, P2_SENT_Get_Fast_Channel_CRC_OK

Example

see P2_SENT_Get_Fast_Channel1

P2_SENT_Get_Serial_Message_Array

P2_SENT_Get_Serial_Message_Array returns a complete pattern of serial messages from a channel of the specified module.

Syntax

```
#Include ADwinPro_All.inc

REM define SENT settings array
P2_SENT_Init(module, sent_datatable[])

P2_SENT_Get_Serial_Message_Array(sent_datatable[],
   sent_channel, serial_array[], serial_array_index)
```

Parameters

sent_datatable[]	Array which contains settings for data transfer between *ADwin* CPU and the SENT module.	ARRAY LONG
sent_channel	Number (1…4, 1…6) of the SENT channel.	LONG
serial_array[]	Destination array, where the buffered message pattern is stored. The array must have at least 512 elements.	ARRAY LONG
serial_array_index	First array element in serial_array[] which is written.	LONG

Notes

Call **P2_SENT_Init** first before you use **P2_SENT_Get_Serial_Message_Array**.

Use **P2_SENT_Get_Serial_Message** to get a single serial message.

After power-up, the module collects and buffers incoming serial messages of the SENT sensor as message set. A message pattern is formed by several (up to 32) serial messages. Data is only buffered for ids used by the sensor.

With **P2_SENT_Clear_Serial_Message_Array** you can set all buffered data to zero.

For each transferred ID, the recently received data value and the number of completely received serial messages are stored in the destination array; array elements related to other ids stay unchanged. If the ID has 4 bit length, only the elements [1]…[16] and [257]…[272] in serial_array[] can be used.

In the serial_array[], both the data values and the number of received messages is transferred in the following way:

	Data value	Number of messages
ID 0	serial_array[1]	serial_array[257]
ID 1	serial_array[2]	serial_array[258]
…	…	…
ID 15	serial_array[16]	serial_array[272]
…	…	…
ID 254	serial_array[255]	serial_array[511]
ID 255	serial_array[256]	serial_array[512]

The following sending formats of serial messages can be recognized:

- Short Serial Message Format, length 12 bit:
  ID 4 bit and data value 8 bit.
- Enhanced Serial Message Format, 20 bit Länge:
  ID 4 bit, data value 16 bit or ID 8 bit, data value 12 bit.

Valid for

SENT-4 Rev. E, SENT-6 Rev. E

See also

P2_SENT_Init, P2_SENT_Get_PulseCount, P2_SENT_Get_Fast_Channel1, P2_SENT_Get_Fast_Channel2, P2_SENT_Get_Fast_Channel_CRC_OK, P2_SENT_Get_ChannelState, P2_SENT_Get_ClockTick, P2_SENT_Clear_Serial_Message_Array

Example

```
#Include ADwinPro_All.inc

#Define module 2
#Define channel 1
#Define id_array Data_1
Dim senttable[150] As Long At DM_Local
Dim id_array[32] As Long At DM_Local
Dim array[512] As Long At DM_Local
Dim i, j As Long

Init:
  Processdelay = 300000     '1 kHz

  Rem initialize module, request latch for channel 1 (once)
  P2_SENT_Init(module, senttable)

  REM Wait until detection mode of SENT channel has finished and
  REM read mode is set
  Par_1 = 2^(channel - 1)
  Do : Until (P2_SENT_Get_ChannelState(module) And Par_1 = Par_1)

  Rem CRC mode 'recommended'
  Do : Until (P2_SENT_Command_Ready(module) = 0)
  Par_1 = P2_SENT_Set_CRC_Implementation(module, channel, 1)

  REM clear serial message array
  Par_2 = P2_SENT_Clear_Serial_Message_Array(module, channel)

Event:
  REM get pattern of serial messages
  P2_SENT_Get_Serial_Message_Array(senttable, channel, array, 1)

  REM get list of used IDs
  For i = 257 To 512
    REM if count > 0, the ID is used
    If (array[i] > 0) Then
      Inc j
      REM store ID
      id_array[j] = i
    EndIf
  Next i
```

P2_SENT_Clear_Serial_Message_Array

P2_SENT_Clear_Serial_Message_Array clears the buffered pattern of serial messages of a SENT channel.

Syntax

```
#Include ADwinPro_All.inc
P2_SENT_Clear_Serial_Message_Array(module,
   sent_channel)
```

Parameters

module	Selected module address (1…15).	LONG
sent_channel	Number (1…4, 1…6) of the SENT channel.	LONG

Notes

First, check with **P2_SENT_Command_Ready**, if the SENT interface is ready to process the next command before using **P2_SENT_Clear_Serial_Message_Array**.

After power-up, the module collects and buffers incoming serial messages of the SENT sensor as message set. A message pattern is formed by several (up to 32) serial messages.

The query of single serial messages with **P2_SENT_Get_Serial_Message_Data** is not affected by the reset.

Valid for

SENT-4 Rev. E, SENT-6 Rev. E

See also

P2_SENT_Command_Ready, P2_SENT_Get_PulseCount, P2_SENT_Get_Fast_Channel1, P2_SENT_Get_Fast_Channel2, P2_SENT_Get_Fast_Channel_CRC_OK, P2_SENT_Get_ChannelState, P2_SENT_Get_ClockTick, P2_SENT_Get_Serial_Message_Array

Example

see P2_SENT_Get_Serial_Message_Array

P2_SENT_Set_CRC_Implementation

P2_SENT_Set_CRC_Implementation sets the checksum implementation type of a SENT channel on the specified module.

Syntax

```
#Include ADwinPro_All.inc
ret_val = P2_SENT_Set_CRC_Implementation(module,
    sent_channel, crc_implement)
```

Parameters

module	Selected module address (1…15).	LONG
sent_channel	Number (1…4, 1…6) of the SENT channel.	LONG
crc_implement	Implementation type of the CRC checksum: 0: Legacy (default). 1: Recommended.	LONG
ret_val	Status of command processing: 0: Command was processed successfully.	LONG

Notes

First, check with **P2_SENT_Command_Ready**, if the SENT interface is ready to process the next command before using **P2_SENT_Set_CRC_Implementation**.

After power up the implementation type "Legacy" is set. The implementation type refers to both signals (fast channels) and serial messages.

Valid for

SENT-4 Rev. E, SENT-6 Rev. E

See also

P2_SENT_Set_ClockTick, P2_SENT_Get_ChannelState P2_SENT_Command_Ready

Example

see P2_SENT_Get_Fast_Channel1

P2_SENT_Set_Detection

P2_SENT_Set_Detection sets a SENT channel of the specified module to detection mode.

Syntax

```
#Include ADwinPro_All.inc
ret_val = P2_SENT_Set_Detection(module,
    sent_channel)
```

Parameters

module	Selected module address (1…15).	LONG
sent_channel	Number (1…4, 1…6) of the SENT channel.	LONG
ret_val	Status of command processing: 0: Command was processed successfully.	LONG

Notes

First, check with **P2_SENT_Command_Ready**, if the SENT interface is ready to process the next command before using **P2_SENT_Set_Detection**.

In detection mode incoming SENT messages are analyzed. As soon as the module detects the clock tick and number of pulses of a message it switches the input channel to read mode.

Alternatively, you can manually set clock tick and number of pulses with **P2_SENT_Set_ClockTick** and **P2_SENT_Set_PulseCount**.

Valid for

SENT-4 Rev. E, SENT-6 Rev. E

See also

P2_SENT_Set_ClockTick, P2_SENT_Get_ChannelState P2_SENT_Command_Ready

Example

see P2_SENT_Get_Fast_Channel1

P2_SENT_Set_ClockTick

P2_SENT_Set_ClockTick switches a SENT channel on the specified module to read mode with the specified clock tick.

Syntax

```
#Include ADwinPro_All.inc
ret_val = P2_SENT_Set_ClockTick(module,
    sent_channel, period)
```

Parameters

module	Selected module address (1…15).	LONG
sent_channel	Number (1…4, 1…6) of the SENT channel.	LONG
period	SENT clock tick (1500…90000) in nanoseconds. A too small clock tick will be automatically corrected.	LONG
ret_val	Status of command processing: 0: Command was processed successfully.	LONG

Notes

First, check with **P2_SENT_Command_Ready**, if the SENT interface is ready to process the next command before using **P2_SENT_Set_ClockTick**.

The input channel is switched to read mode without checking and evaluates SENT messages according to the specified clock tick.

Alternatively, the module can detect the clock tick of SENT messages in detection mode automatically, see **P2_SENT_Set_Detection**.

Valid for

SENT-4 Rev. E, SENT-6 Rev. E

See also

P2_SENT_Get_ClockTick, P2_SENT_Set_Detection, P2_SENT_Get_ChannelState, P2_SENT_Command_Ready

Example

see P2_SENT_Get_Fast_Channel1

P2_SENT_Set_PulseCount

P2_SENT_Set_PulseCount sets if a pause pulse is expected in SENT messages of an input channel of the specified module.

Syntax

```
#Include ADwinPro_All.inc
P2_SENT_Set_PulseCount(module, sent_channel,
   pulse_count)
```

Parameters

module	Selected module address (1…15).	LONG
sent_channel	Number (1…4, 1…6) of the SENT channel.	LONG
pulse_count	Key value for pause pulse: 9: SENT message without pause pulse. 10:SENT message with pause pulse.	LONG

Notes

First, check with **P2_SENT_Command_Ready**, if the SENT interface is ready to process the next command before using **P2_SENT_Set_PulseCount**.

A SENT message consists of several pulses. pulse_count determines whether the module expects a pause pulse in the SENT message or not.

For SENT messages with two 12 bit values and pause pulse the message length results to 10 pulses:

- Calibration pulse for synchronization
- 1 nibble pulse (=4 bit): Status and communication
- 3 nibble pulses: erster 12 bit value (fast channel 1)
- 3 nibble pulses: zweiter 12 bit value (fast channel 2)
- 1 nibble pulse: Prüfsumme
- pause pulse (optional)

Alternatively, the module can detect the pause pulse of SENT messages in detection mode automatically, see **P2_SENT_Set_Detection**.

Valid for

SENT-4 Rev. E, SENT-6 Rev. E

See also

P2_SENT_Command_Ready, P2_SENT_Get_PulseCount, P2_SENT_Get_Fast_Channel1, P2_SENT_Get_Fast_Channel2, P2_SENT_Get_Fast_Channel_CRC_OK, P2_SENT_Get_ChannelState, P2_SENT_Get_ClockTick

Example

see P2_SENT_Get_Fast_Channel1

P2_SENT_Set_Sensor_Type

P2_SENT_Set_Sensor_Type sets the expected sensor type for the SENT messages on an input channel of the specified module.

Syntax

```
#Include ADwinPro_All.inc
P2_SENT_Set_Sensor_Type(module, sent_channel,
   sensor_type)
```

Parameters

module	Selected module address (1…15).	LONG
sent_channel	Number (1…4, 1…6) of the SENT channel.	LONG
sensor_type	Key value for the sensor type: 0: no sensor type selected (default). 1: throttle position sensor. 2: single secure sensor.	LONG

Notes

First, check with **P2_SENT_Command_Ready**, if the SENT interface is ready to process the next command before using **P2_SENT_Set_Sensor_Type**.

Some sensor types send messages with a specified combination and function of nibbles. If you select a sensor type with **P2_SENT_Set_Sensor_Type**, the module checks is the received SENT data also corresponds to the specifications of the sensor type.

If the received SENT data does not fit to the specified sensor type, an error code is provided. You read the error code with **P2_SENT_Get_Latch_Data**.

Valid for

SENT-4 Rev. E, SENT-6 Rev. E

See also

P2_SENT_Command_Ready, P2_SENT_Get_Latch_Data, P2_SENT_Get_PulseCount, P2_SENT_Get_Fast_Channel1, P2_SENT_Get_Fast_Channel2, P2_SENT_Get_Fast_Channel_CRC_OK, P2_SENT_Get_ChannelState, P2_SENT_Get_ClockTick

Example

- / -

P2_SENT_ Request_Latch

P2_SENT_Request_Latch requests to buffer the data of selected SENT channels on the specified once in a latch buffer.

Syntax

```
#Include ADwinPro_All.inc
P2_SENT_Request_Latch(module, pattern)
```

Parameters

module	Selected module address (1…15).	LONG
pattern	Bit pattern selecting SENT channels to be buffered (see table). Bit = 0: Don't buffer channel data. Bit = 1: Do buffer channel data.	LONG

Bit no.	31…6	5	4	3	2	1	0
SENT channel	–	6	5	4	3	2	1

Notes

Data of SENT messages are collected in the latch buffered until all nibbles of the message have been received completely. Only then the message can be read from the latch using **P2_SENT_Get_Latch_Data**.

You can request buffering of SENT data only at a time for the next SENT message, but not as a permanent feature.

P2_SENT_Request_Latch does settings for all SENT channels at once. If you want to set only one SENT channel, read the current status with **P2_SENT_Check_Latch**, change the bit of the appropriate channel and start a new request (see example).

Buffering via latch does not affect reading single information as with **P2_SENT_Get_Fast_Channel1** or **P2_SENT_Get_Serial_Message_Data**.

Valid for

SENT-4 Rev. E, SENT-6 Rev. E

See also

P2_SENT_Init, P2_SENT_Check_Latch, P2_SENT_Get_Latch_Data

Example

see P2_SENT_Get_Latch_Data

P2_SENT_Check_Latch

P2_SENT_Check_Latch returns whether the latch contains data of the requested SENT channels.

Syntax

```
#Include ADwinPro_All.inc
ret_val = P2_SENT_Check_Latch(module)
```

Parameters

module	Selected module address (1…15).	LONG
ret_val	Bit pattern, which determines whether latch data is ready to be read (see table): Bit = 0: Data is ready to be read. Bit = 1: The data is incomplete.	LONG

Bit no.	31…6	5	4	3	2	1	0
SENT channel	–	6	5	4	3	2	1

Notes

P2_SENT_Check_Latch is only useful for SENT channels, where you have requested buffering data in the latch with **P2_Request_Latch**.

Valid for

SENT-4 Rev. E, SENT-6 Rev. E

See also

P2_SENT_Init, P2_SENT_Request_Latch, P2_SENT_Get_Latch_Data

Example

see P2_SENT_Get_Latch_Data

P2_SENT_Get_Latch_Data

P2_SENT_Get_Latch_Data reads data of a SENT message of a SENT channel from the latch buffer.

Syntax

```
#Include ADwinPro_All.inc

REM define SENT settings array
P2_SENT_Init(module, sent_datatable[])

P2_SENT_Get_Latch_Data(sent_datatable[],
   sent_channel, data_array[], data_array_index)
```

Parameters

sent_datatable[]	Array which contains settings for data transfer between *ADwin* CPU and the SENT module.	ARRAY LONG
sent_channel	Number (1…4, 1…6) of the SENT channel.	LONG
data_array[]	Destination array, where SENT data is stored. The array must have at least 32 elements. The meaning of the array elements is described below.	ARRAY LONG
data_array_index	First array element in data_array[], which is written.	LONG

Notes

First, check with **P2_SENT_Check_Latch**, whether data is ready to be read. If you read data from the latch while data is still incomplete, you receive the data of the previous SENT message.

A data set consists of 32 array elements which are stored starting from index data_array_index. A data set contains all 8 nibbles of a SENT message as well as some already evaluated data.

The data of the serial message (elements 7…11) are only contained in the data set if the current SENT message terminates the serial message. If no serial message is contained in the data set element 7 has the value zero (0).

The content of array elements is described below. To simplify the illustration, data_array_index is assumed to be 1.

Index	Meaning
[1]	Number of received messages on the SENT channel.
[2][3]	Time of receipt of the SENT message, given as 64 bit counter value. The value is split to upper [2] and lower word [3]. The counter runs at a frequency of 100 MHz, i.e. the timer value is given in units of 10 ns.
[4]	Fast channel 1: First 12 bit value of the message.
[5]	Fast channel 2: Second 12 bit value of the message.
[6]	Fast channels: Result of CRC check: 0: Data transfer was successful. 1: Checksum error, error during data transfer.

Index	Meaning
[7]	Serial message, key value for sending format: 0: No serial message data available. 1: Short Serial Message Format, 12 bit length: ID 4 bit, data value 8 bit. 2: Enhanced Serial Message format, 20 bit length: ID 4 bit, data value 16 bit 3: Enhanced Serial Message format, 20 bit length: ID 8 bit, data value 12 bit
[8]	Serial message ID.
[9]	Serial message, data value.
[10]	Serial message, CRC checksum.
[11]	Serial message, result of CRC check: 0: Data transfer was successful. -1: Checksum error, error during data transfer.
[12]	Error bit pattern. If no error occurred the value of the bit pattern equals 0. Meaning of set bits see below.
[13]… [19]	Reserved
[20]	Length of calibration pulse in units of 10ns.
[21]… [28]	All 8 nibbles of the SENT message in the order: Status nibble, data nibbles 1…6, CRC nibble.
[29]… [32]	reserved

Set error bits in the array element 12 have the following meaning:

Bit no.	Meaning
0	Stat Reserved 0
1	Stat Reserved 1
2	Invalid nibble value: < 0 or > 15
3	Invalid calibration puls length: <56 clock ticks - 1.5625% or >56 clock ticks + 1.5625%
4	Abs Sync Size Fail
5	2 or more consecutive synchronization pulses. This may happen if pause pulse and synchronization pulse have same length.
6	Wrong number of pulses between two synchronization pulses.
7	Rolling count fail
8	Inverted nibble fail
9	Serial message fail
10…31	reserved

Valid for

SENT-4 Rev. E, SENT-6 Rev. E

See also

P2_SENT_Init, P2_SENT_Set_Sensor_Type, P2_SENT_Check_Latch, P2_SENT_Check_Latch

- / -

Example

```
#Include ADwinPro_All.inc
#Define module 4
Rem set SENT channel and appropriate bit pattern
#Define channel 1
#Define ch_pattern Shift_Left(1, channel-1)
#Define msg_count Par_2    'number of received messages
#Define error_count Par_7  'number of CRC errors
#Define lost_msg Par_9     'number of lost messages
Dim senttable[150] As Long At DM_Local
Dim sent_data[100] As Long At DM_Local
Dim msg_no, msg_no_old As Long
Dim i As Long

Init:
  Rem Initialize receive buffer for SENT data
  For i = 1 To 100
    sent_data[i] = 0
  Next i
  Processdelay = 60000      '60000=5kHz / 30000=10kHz
  Rem initialize module, request latch for channel 1 (once)
  P2_SENT_Init(module, senttable)
  Par_5 = P2_SENT_Request_Latch(module, ch_pattern)
  Do : Until (P2_SENT_Command_Ready(module) = 0)
  Par_4 = P2_SENT_Set_CRC_Implementation(module, channel, 1)
  msg_count = 0             'number of received messages
  msg_no_old = -1           'init msg number
  lost_msg = -1 'number of lost messages, skip first check

Event:
  Rem read latch status of all channels
  Par_1 = P2_SENT_Check_Latch(module)
  Rem Any data for SENT channel 1?
  If ((Par_1 And ch_pattern) = 0) Then
    Rem read latch data into sent_data[]
    P2_SENT_Get_Latch_Data(senttable, channel, sent_data, 1)
    Rem request latch for channel 1 again
    Par_5 = P2_SENT_Request_Latch(module, Par_1 XOr ch_pattern)
    Inc msg_count           'number of received messages

    Rem check for serial message
    If (sent_data[7] <> 0) Then
      Rem serial message data are given in sent_data[7]..[11]
    EndIf

    Rem check for lost messages
    msg_no = sent_data[1]
    If ((msg_no - msg_no_old) <> 1) Then Inc lost_msg
    msg_no_old = msg_no      'store for next check
    Rem check for CRC errors (index 6)
    If (sent_data[6] <> 0) Then Inc error_count
  EndIf
```

P2_SENT_Set_Output_Mode

P2_SENT_Set_Output_Mode sets the output mode for a SENT channel on the specified module.

Syntax

```
#Include ADwinPro_All.inc
ret_val = P2_SENT_Set_Output_Mode(module,
    sent_channel, mode)
```

Parameters

module	Selected module address (1…15).	LONG
sent_channel	Number (1…4) of the SENT channel.	LONG
mode	Output mode of the SENT channel: 0: Continuous output (default). 1: FIFO output.	LONG

Notes

First, check with **P2_SENT_Command_Ready**, if the SENT interface is ready to process the next command before using **P2_SENT_Set_Output_Mode**.

Disable the SENT channel using **P2_SENT_Enable_Channel** before you change the output mode. The output starts as soon as you enable the SENT channel again.

The output mode determines where the module gets the SENT messages from:

- Continuous output: The currently defined SENT message is sent continuously.
  You set the content of the SENT message with **P2_SENT_Set_Fast_Channel1** / **_Channel2** **P2_SENT_Set_Serial_Message_Pattern** **P2_SENT_Set_Reserved_Bits**.
- FIFO output: You fill a FIFO continuously with output values, see **P2_SENT_Set_Fifo**. The module reads the values from the FIFO and outputs the values consecutively.
  If the Fifo runs empty, the output stops and the SENT channel is disabled automatically.

Valid for

SENT-4-Out Rev. E

See also

P2_SENT_Init, P2_SENT_Enable_Channel, P2_SENT_Get_Output_Mode, P2_SENT_Set_Fast_Channel1, P2_SENT_Set_Fast_Channel2, P2_SENT_Set_Serial_Message_Pattern, P2_SENT_Set_Reserved_Bits, P2_SENT_Set_Fifo

Example

- / -

P2_SENT_Get_Output_Mode

P2_SENT_Get_Output_Mode returns the output mode of all SENT channels on the specified module.

Syntax

```
#Include ADwinPro_All.inc
ret_val = P2_SENT_Get_Output_Mode(module)
```

Parameters

module	Selected module address (1…15).	LONG
ret_val	Bit pattern with the output mode of all SENT channels: Bit = 0: Continuous output. Bit = 1: FIFO output.	LONG

Bit no.	31…4	3	2	1	0
SENT channel	–	4	3	2	1

Notes

- / -

Valid for

SENT-4-Out Rev. E

See also

P2_SENT_Init, P2_SENT_Set_Output_Mode

Example

- / -

P2_SENT_Config_Output

P2_SENT_Config_Output does the basic settings for a SENT output channel on the specified module.

Syntax

```
#Include ADwinPro_All.inc
ret_val = P2_SENT_Config_Output(module,
    sent_channel, baseclock, length, lowticks, crc)
```

Parameters

module	Selected module address (1…15).	LONG
sent_channel	Number (1…4) of the SENT channel.	LONG
baseclock	Clock tick (1500…90000) for SENT messages, given in ns.	LONG
length	Fixed total length (0; 166…922) of the SENT message, given in baseclock units. 0: Variable length without pause pulse. >0: Total length of the message.	LONG
lowticks	Length (normally 4) of the low level at the start of calibration pulse or nibble pulse, given in baseclock units.	LONG
crc	Implementation type for CRC checksum: 0: Legacy. 1: Recommended.	LONG
ret_val	Status of command processing: 0: Command was processed successfully.	LONG

Notes

First, check with **P2_SENT_Command_Ready**, if the SENT interface is ready to process the next command before using **P2_SENT_Config_Output**.

A SENT message without pause pulse has a variable length of 154 … 270 clock ticks. With length you can set the total length of a message to a fixed value. This is acheived by an additional pause pulse with the appropriate length.
Make sure, that length is always 12 units greater than the actual length of the SENT message.

The implementation type for the CRC checksum refers to both, signals (fast channels) and serial message.

Valid for

SENT-4-Out Rev. E

See also

P2_SENT_Init, P2_SENT_Command_Ready, P2_SENT_Enable_Channel, P2_SENT_Set_Fast_Channel1, P2_SENT_Set_Fast_Channel2, P2_SENT_Set_Fifo

Example

- / -

P2_SENT_Config_Serial_Messages

P2_SENT_Config_Serial_Messages configures the sending format for serial messages on a SENT channel of the specified module.

Syntax

```
#Include ADwinPro_All.inc
ret_val = P2_SENT_Config_Serial_Messages(
    module, sent_channel, format)
```

Parameters

module	Selected module address (1…15).	LONG
sent_channel	Number (1…4) of the SENT channel.	LONG
format	Sending format of serial messages: 0: No serial message. 1: Format "Standard", 4 bit ID, 8 bit data value. 2: Format "Enhanced", 4 bit ID, 16 bit data value. 3: Format "Enhanced", 8 bit ID, 12 bit data value.	LONG
ret_val	Status of command processing: 0: Command was processed successfully.	LONG

Notes

The instruction is only useful if the module runs in continuous mode, see **P2_SENT_Set_Output_Mode**.

First, check with **P2_SENT_Command_Ready**, if the SENT interface is ready to process the next command before using **P2_SENT_Config_Serial_Messages**.

Valid for

SENT-4-Out Rev. E

See also

P2_SENT_Init, P2_SENT_Command_Ready, P2_SENT_Set_Serial_Message_Pattern, P2_SENT_Set_Serial_Message_Data

Example

- / -

P2_SENT_ Enable_Channel

P2_SENT_Enable_Channel enables or disables a SENT channel on the specified module.

Syntax

```
#Include ADwinPro_All.inc
P2_SENT_Enable_Channel(module, enable)
```

Parameters

module	Selected module address (1…15).	LONG
enable	Bit pattern to enable or disable SENT channels: Bit=0: Disable SENT channel. Bit=1: Enable SENT channel.	LONG

Bit no.	31…4	3	2	1	0
SENT channel	–	4	3	2	1

Notes

The SENT channel must first be configured with **P2_SENT_Config_Output** before you enable it for sending.

If you disable a SENT channel, the current SENT message will be completely sent, afterwards the output is stopped.
After enabling a channel the output starts at once. In continuous mode the defined SENT message is sent; in Fifo mode the next available SENT message in the Fifo is read and sent.

Disabling a channel interrupts sending of the serial message. After enabling, sending of the serial message is resumed at the next bit.

In Fifo mode, the SENT channel is automatically disabled (and the output is stopped) if the Fifo runs empty. Therefore please make sure that–after enabling the SENT channel–to write data into the Fifo fast enough.

Valid for

SENT-4-Out Rev. E

See also

P2_SENT_Init, P2_SENT_Config_Output, P2_SENT_Invert_Channel, P2_SENT_Set_Serial_Message_Pattern

Example

- / -

P2_SENT_Invert_Channel

P2_SENT_Invert_Channel inverts all output levels of the SENT channels of the specified module.

Syntax

```
#Include ADwinPro_All.inc
P2_SENT_Invert_Channel(module, invert)
```

Parameters

module	Selected module address (1…15).	LONG
invert	Bit pattern to invert the output levels of SENT channels. Bit = 0: Normal output level. Bit = 1: Inverted output level.	LONG

Bit no.	31…4	3	2	1	0
SENT channel	–	4	3	2	1

Notes

If a SENT channel is disabled, the normal output level is a low level, if inverted it is high level.

Level inverting with **P2_SENT_Invert_Channel** enables to toggle the output level even with disabled SENT channel.

Valid for

SENT-4-Out Rev. E

See also

P2_SENT_Init, P2_SENT_Config_Output, P2_SENT_Enable_Channel

Example

- / -

P2_SENT_Set_Reserved_Bits

P2_SENT_Set_Reserved_Bits sets the reserved bits of the status nibble for a SENT channel on the specified module.

Syntax

```
#Include ADwinPro_All.inc
P2_SENT_Set_Reserved_Bits(module, sent_channel,
    value)
```

Parameters

module	Selected module address (1…15).	LONG
sent_channel	Number (1…4) of the SENT channel.	LONG
value	Bit pattern (0…11b), which is transferred to the reserved bits of the status nibble.	LONG

Notes

The instruction is only useful if the module runs in continuous mode, see **P2_SENT_Set_Output_Mode**.

In the status nibble, bits 0 and 1 are reserved for special applications.

Valid for

SENT-4-Out Rev. E

See also

P2_SENT_Init, P2_SENT_Enable_Channel, P2_SENT_Config_Output, P2_SENT_Config_Serial_Messages, P2_SENT_Set_Serial_Message_Pattern, P2_SENT_Set_Serial_Message_Data, P2_SENT_Set_Fast_Channel1, P2_SENT_Set_Fast_Channel2

Example

- / -

P2_SENT_Set_Fast_Channel1

P2_SENT_Set_Fast_Channel1 sets the first 12 bit value in a SENT message for a SENT channel on the specified module.

Syntax

```
#Include ADwinPro_All.inc
P2_SENT_Set_Fast_Channel1(module, sent_channel,
    value)
```

Parameters

module	Selected module address (1…15).	LONG
sent_channel	Number (1…4) of the SENT channel.	LONG
value	First 12 bit value (0…4095) in the SENT message.	LONG

Notes

The instruction is only useful if the module runs in continuous mode, see **P2_SENT_Set_Output_Mode**.

A SENT message contains two 12 bit values, also named "fast channel signals". The instruction sets the first of both values.

The CRC checksum is automatically calculated (implementation type see **P2_SENT_Config_Output**) and set in the SENT message.

Valid for

SENT-4-Out Rev. E

See also

P2_SENT_Init, P2_SENT_Enable_Channel, P2_SENT_Config_Output, P2_SENT_Config_Serial_Messages, P2_SENT_Set_Serial_Message_Pattern, P2_SENT_Set_Serial_Message_Data, P2_SENT_Set_Fast_Channel2, P2_SENT_Set_Reserved_Bits

Example

- / -

P2_SENT_Set_Fast_Channel2

P2_SENT_Set_Fast_Channel2 sets the second 12 bit value in a SENT message for a SENT channel on the specified module.

Syntax

```
#Include ADwinPro_All.inc
P2_SENT_Set_Fast_Channel2(module, sent_channel,
    value)
```

Parameters

module	Selected module address (1…15).	LONG
sent_channel	Number (1…4) of the SENT channel.	LONG
ret_val	Second 12 bit value (0…4095) in the SENT message.	LONG

Notes

The instruction is only useful if the module runs in continuous mode, see **P2_SENT_Set_Output_Mode**.

A SENT message contains two 12 bit values, also named "fast channel signals". The instruction sets the second of both values.

The CRC checksum is automatically calculated (implementation type see **P2_SENT_Config_Output**) and set in the SENT message.

Valid for

SENT-4-Out Rev. E

See also

P2_SENT_Init, P2_SENT_Enable_Channel, P2_SENT_Config_Output, P2_SENT_Config_Serial_Messages, P2_SENT_Set_Serial_Message_Pattern, P2_SENT_Set_Serial_Message_Data, P2_SENT_Set_Fast_Channel1, P2_SENT_Set_Reserved_Bits

Example

- / -

P2_SENT_Set_Serial_Message_Pattern

P2_SENT_Set_Serial_Message_Pattern defines a pattern of serial messages for a SENT channel.

Syntax

```
#Include ADwinPro_All.inc

REM define SENT settings array
P2_SENT_Init(module, sent_datatable[])

P2_SENT_Set_Serial_Message_Pattern(module,
    sent_datatable[], sent_channel, sm_id_array[],
    sm_value_array[], sm_len)
```

Parameters

module	Selected module address (1…15).	LONG
sent_datatable[]	Array which contains settings for data transfer between *ADwin* CPU and the SENT module.	ARRAY LONG
sent_channel	Number (1…4) of the SENT channel.	LONG
sm_id_array[]	Array with up to 32 IDs for the pattern of serial messages.	ARRAY LONG
sm_value_array[]	Array with up to 32 data values for the pattern of serial messages, referring to IDs of sm_id_array[].	ARRAY LONG
sm_len	Number (1…32) of serial messages in the message set.	LONG

Notes

The instruction is only useful if the module runs in continuous mode, see **P2_SENT_Set_Output_Mode**.

First, check with **P2_SENT_Command_Ready**, if the SENT interface is ready to process the next command before using **P2_SENT_Set_Serial_Message_Pattern**.

Disable the output channel with **P2_SENT_Enable_Channel** before you define a message set.

The value range of ids and data values depends on the set snding format of serial messages. You set the sending format (standard, enhanced) with **P2_SENT_Config_Serial_Messages**.

The message pattern in sm_id_array and sm_value_array is transferred consecutively in the SENT messages. If the end of list (sm_len) is reached, the data transfer starts again.

You can change single data values afterwards with **P2_SENT_Set_Serial_Message_Data**. To change ids you have to re-define the complete message set.

The appropriate CRC checksum is calculated and transferred automatically (implementation type see **P2_SENT_Config_Output**).

Valid for

SENT-4-Out Rev. E

See also

P2_SENT_Init, P2_SENT_Command_Ready, P2_SENT_Enable_Channel, P2_SENT_Config_Output, P2_SENT_Config_Serial_Messages, P2_SENT_Set_Serial_Message_Data, P2_SENT_Set_Fast_Channel1, P2_SENT_Set_Fast_Channel2, P2_SENT_Set_Reserved_Bits

Example

- / -

P2_SENT_Set_Serial_Message_Data

P2_SENT_Set_Serial_Message_Data changes a data value in the defined message pattern for serial messages on a SENT channel.

Syntax

```
#Include ADwinPro_All.inc
P2_SENT_Set_Serial_Message_Data(module,
    sent_channel, sm_id_idx, sm_value)
```

Parameters

module	Selected module address (1…15).	LONG
sent_channel	Number (1…4) of the SENT channel.	LONG
sm_id_idx	Index in the defined message pattern array.	LONG
sm_value	Data value for the serial message.	LONG

Notes

The instruction is only useful if the module runs in continuous mode, see **P2_SENT_Set_Output_Mode**.

First, check with **P2_SENT_Command_Ready**, if the SENT interface is ready to process the next command before using **P2_SENT_Set_Serial_Message_Data**.

The instruction changes a data value in the message pattern you have defined with **P2_SENT_Set_Serial_Message_Pattern**, at the position sm_id_idx. You can only change data values but not the defined ids.

The appropriate CRC checksum is calculated and transferred automatically (implementation type see **P2_SENT_Config_Output**).

Valid for

SENT-4-Out Rev. E

See also

P2_SENT_Init, P2_SENT_Command_Ready, P2_SENT_Config_Output, P2_SENT_Config_Serial_Messages, P2_SENT_Set_Serial_Message_Pattern, P2_SENT_Set_Fast_Channel1, P2_SENT_Set_Fast_Channel2, P2_SENT_Set_Reserved_Bits

Example

- / -

P2_SENT_Fifo_Empty

P2_SENT_Fifo_Empty returns the number of free elements in the output Fifo of a SENT channel.

Syntax

```
#Include ADwinPro_All.inc
ret_val = P2_SENT_Fifo_Empty(module, sent_channel)
```

Parameters

module	Selected module address (1…15).	LONG
sent_channel	Number (1…4) of the SENT channel.	LONG
ret_val	Number (0…500) of free elements in the output Fifo of the SENT channel.	LONG

Notes

The instruction is only useful if the module runs in continuous mode, see **P2_SENT_Set_Output_Mode**.

Valid for

SENT-4-Out Rev. E

See also

P2_SENT_Init, P2_SENT_Set_Output_Mode, P2_SENT_Fifo_Clear, P2_SENT_Set_Fifo

Example

- / -

P2_SENT_Fifo_Clear

P2_SENT_Fifo_Clear initializes the write and read pointer of the output Fifo of a SENT channel.

Syntax

```
#Include ADwinPro_All.inc
P2_SENT_Fifo_Clear(module, sent_channel)
```

Parameters

module	Selected module address (1…15).	LONG
sent_channel	Number (1…4) of the SENT channel.	LONG

Notes

The instruction is only useful if the module runs in continuous mode, see **P2_SENT_Set_Output_Mode**.

First, check with **P2_SENT_Command_Ready**, if the SENT interface is ready to process the next command before using **P2_SENT_Fifo_Clear**.

Valid for

SENT-4-Out Rev. E

See also

P2_SENT_Init, P2_SENT_Command_Ready, P2_SENT_Set_Output_Mode, P2_SENT_Fifo_Empty, P2_SENT_Set_Fifo

Example

- / -

P2_SENT_Set_Fifo

P2_SENT_Set_Fifo writes new data into the output Fifo of a SENT channel on the specified module.

Syntax

```
#Include ADwinPro_All.inc

REM define SENT settings array
Dim sent_datatable[150] As Long

ret_val = P2_SENT_Set_Fifo(module, sent_datatable[],
    sent_channel, array[], array_count)
```

Parameters

module	Selected module address (1…15).	LONG
sent_datatable[]	Array which contains settings for data transfer between *ADwin* CPU and the SENT module.	ARRAY LONG
sent_channel	Number (1…4) of the SENT channel.	LONG
array[]	Source array with output values. The array may not exceed 100 elements.	ARRAY LONG
array_count	Number of output values to be transferred.	LONG
ret_val	Status of command processing: 0: Command was processed successfully.	LONG

Notes

The instruction is only useful if the module runs in continuous mode, see **P2_SENT_Set_Output_Mode**.

Run **P2_SENT_Init** once before you use **P2_SENT_Get_Serial_Message_Array**.

First, check with **P2_SENT_Command_Ready**, if the SENT interface is ready to process the next command before using **P2_SENT_Set_Fifo**.

Check with **P2_SENT_Fifo_Empty**, if there is enough space in the Fifo before writing new data.

Each array element in array[] contains a SENT message. You have to put together the 32 bit of the SENT messages on your own. The bits are expected in the following order:

Bit no.	31…28	27…16	15…4	3…0
Content	checksum	signal 2 bits 11…0	signal 1 bits 11…0	status nibble

If the Fifo runs empty, the output is stopped automatically and the SENT channel is disabled. Therefore please make sure that–after enabling the SENT channel–to write data into the Fifo fast enough.

Valid for

SENT-4-Out Rev. E

See also

P2_SENT_Init, P2_SENT_Command_Ready, P2_SENT_Set_Output_Mode, P2_SENT_Fifo_Empty, P2_SENT_Fifo_Clear

Example

- / -

3.19 Pro II: SPI Interface

This section describes instructions which apply to Pro II SPI-2 bus modules:

– P2_SPI_Mode (page 385)
– P2_SPI_Config (page 386)
– P2_SPI_Master_Config (page 388)
– P2_SPI_Master_Set_Value32 (page 392)
– P2_SPI_Master_Set_Value64 (page 393)
– P2_SPI_Master_Start (page 394)
– P2_SPI_Master_Status (page 395)
– P2_SPI_Master_Get_Value32 (page 397)
– P2_SPI_Master_Get_Value64 (page 398)
– P2_SPI_Master_Get_Static_Input (page 399)
– P2_SPI_Master_Set_Clk_Wait (page 400)
– P2_SPI_Slave_Config (page 402)
– P2_SPI_Slave_OutFifo_Write (page 403)
– P2_SPI_Slave_OutFifo_Empty (page 405)
– P2_SPI_Slave_InFifo_Full (page 406)
– P2_SPI_Slave_InFifo_Read (page 407)
– P2_SPI_Slave_Clear_Fifo (page 409)

There are more instructions available for the module SPI-2-T Rev. E, see Instruction List sorted by Module Types (annex A.2).

P2_SPI_Mode

P2_SPI_Mode sets the operating mode (SPI master / SPI slave / digital module) of the specified module.

Syntax

```
#Include ADwinPro_All.inc
P2_SPI_Mode(module, mode)
```

Parameters

module	Selected module address (1…15).	LONG
mode	Code number for the module operating mode: 1: Digital module; corresponds to the module Pro II-DIO-32-TiCo, but without DRAM. 2: SPI module with 2 slave interfaces. 3: SPI module; channel 1 is master interface, channel 2 is slave interface. 4: SPI module with 2 master interfaces. 5: SPI module with 2 slave interfaces, common CLK input SCLK1.	LONG

Notes

The pin assignment is different for each operating mode. With SPI modes (mode = 2…5) some pins not being used for SPI can be used as simple digital channels. Pin assignment see hardware manual.

The operating mode can be changed while running.

See also

P2_SPI_Config, P2_SPI_Master_Config, P2_SPI_Master_Set_Clk_Wait, P2_SPI_Slave_Config

Valid for

SPI-2-D Rev. E, SPI-2-T Rev. E

Example

```
#Include ADwinPro_All.inc

Rem set operation mode to 2 SPI slave interfaces
P2_SPI_Mode(1, 2)
```

P2_SPI_Config

P2_SPI_Config configures an SPI interface of the specified module, for both master and slave.

Syntax

```
#Include ADwinPro_All.inc
P2_SPI_Config(module, channel, mode, bitlength,
     data_order, ss_select)
```

Parameters

module	Selected module address (1…15).	LONG
channel	Number (1, 2) of the SPI channel.	LONG
mode	Mode of SPI master for CPOL (clock polarity) and CPHA (clock phase): 0: CPOL = 0, CPHA = 0 1: CPOL = 0, CPHA = 1 2: CPOL = 1, CPHA = 0 3: CPOL = 1, CPHA = 1	LONG
bitlength	Number (1…64) of transferred bits in a SPI message.	LONG
data_order	Bit order for transfer of a SPI message: 0: most significant bit (MSB) first. 1: least significant bit (LSB) first.	LONG
ss_select	Level which defines slave select lines as active: 0: Level active low. 1: Level active high.	LONG

Notes

If the slave select signal is set automatically (see **P2_SPI_Master_Config**), make sure that the slave select line of the SPI master is deactivated (query with **P2_SPI_Master_Status**) before re-configuring the SPI master. Otherwise, while configuring the SPI master spikes can occur which will be misinterpreted by connected slaves and therefore disturb the data transfer.

The clock signals are present at the pins SCLK1 / SCLK2, pin assignment see hardware manual hardware manual. If **SPI_Mode** was set with parameter mode=5, use the SPI slave's pin SCLK1 as common clock input; in this case, SCLK2 has no function.

The bit length refers the lines SCLK, DATAIN and DATAOUT likewise. Please note, that *ADbasic* variables have a length of 32 bits, therefore SPI messages of 33…64 bits length must be split and stored in 2 variables.

See also

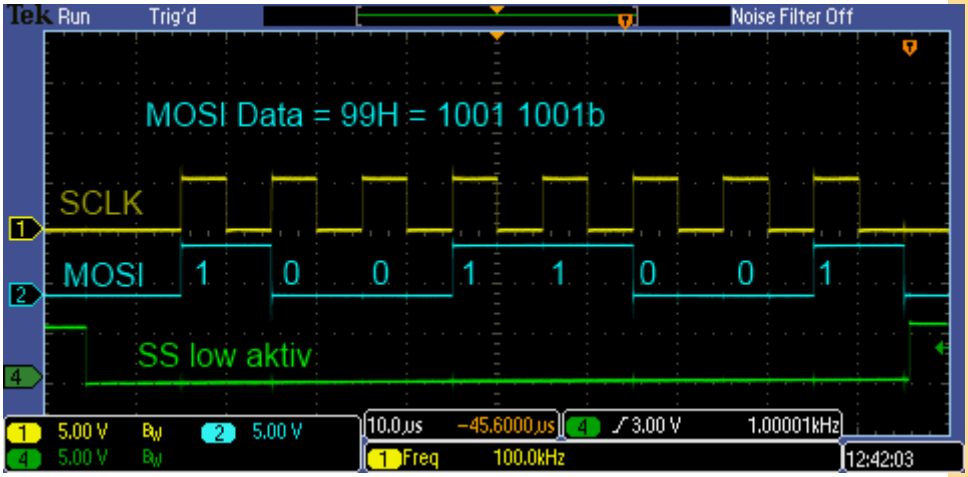
P2_SPI_Mode, P2_SPI_Master_Config, P2_SPI_Master_Set_Clk_Wait, P2_SPI_Slave_Config, P2_SPI_Master_Start, P2_SPI_Master_Status

Valid for

SPI-2-D Rev. E, SPI-2-T Rev. E

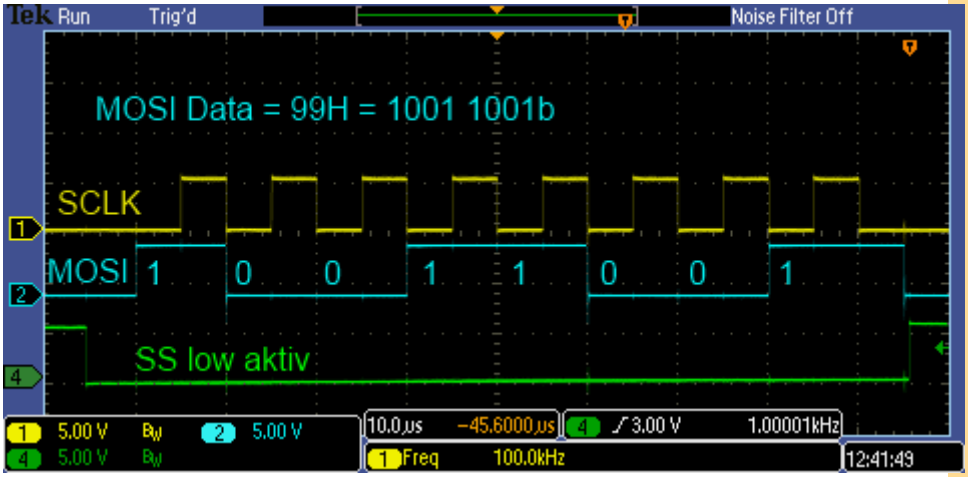
Example

```
#Include ADwinPro_All.inc
#Define module 1
Init:
  P2_SPI_Mode(module, 4)      '2 master interfaces
  Rem Configure master 1:
  Rem CPOL = 0, CPHA = 1; message length 8 Bit
  Rem MSB first; slave select active high
  P2_SPI_Config(module, 1, 1, 8, 0, 0)
  Rem set clock frequence 1 MHz etc.
  P2_SPI_Master_Config(module, 1, 250, 75, 250, 0)
```



MOSI output with rising edge of SCLK with 8 bit value 99h

```
#Include ADwinPro_All.inc
#Define module 1
Init:
  P2_SPI_Mode(module, 4)      '2 master interfaces
  Rem Configure master 1:
  Rem CPOL = 1, CPHA = 0; message length 8 Bit
  Rem MSB first; slave select active high
  P2_SPI_Config(module, 1, 2, 8, 0, 0)
  Rem set clock frequence 1 MHz etc.
  P2_SPI_Master_Config(module, 1, 250, 75, 250, 0)
```



MOSI output with falling edge of SCLK

P2_SPI_Master_Config

P2_SPI_Master_Config sets (additional) properties of an SPI interface of the specified module.

- clk_factor sets the frequency of the SPI clock signal f_SPI.
- miso_delay delays reading of SPI messages (relative to MOSI signal).
- ss_manual sets whether slave slect signal ist set automatically or manually.
- ss_time extends the (automatically activated) slave select signal with reference to the clock signal.

Syntax

```
#Include ADwinPro_All.inc
P2_SPI_Master_Config(module, channel, clk_factor,
     miso_delay, ss_manual, ss_time)
```

Parameters

module	Selected module address (1…15).	LONG
channel	Number (1, 2) of the interface, which is configured as SPI master.	LONG
clk_factor	Factor (2…2500) to determine the clock frequency f_SPI from the base frequency of the timer: f_SPI = 25 MHz / clk_factor	LONG
miso_delay	Delay time in units (0…255) of 20 ns for reading of SPI messages. The delay time can be set in a range of 0µs…5.1µs.	LONG
ss_manual	Setting of slave select operating mode of the SPI master: 0: activate slave select line automatically. 1: activate slave select line manually.	LONG
ss_time	Only with ss_manual=0, i.e. with automatic activation: Additional time offset for the slave select signal in units (0…255) of 20 ns. Default setting is 25 (= 0.5µs). The time offset can be set in a range of 0µs … 5.1µs. The time offset T between slave select signal and clock signal is calculated as follows: T = 0.5 / f_SPI + ss_time x 20 ns	LONG

Notes

Clock frequency

With the factor clk_factor you set the signal's frequency f_SPI, with which the MISO and MOSI signals are output and read at the pins SCLK1 or SCLK2. Pin assignment see hardware manual.

Clock frequency can be set in a range of 100Hz …. 12.500Hz.

A newly set frequency will only be used after **SPI_Master_Start** has been called.

Delayed reading

Incoming MISO data will be read delayed (compared to the MOSI signal) by the set delay time miso_delay. Thus, signal run times can be compensated, which can occur by signal conditioning.

The set delay time is in effect until a new delay time is set. Using the value 0 the delay time is deactivated.

MISO data (line 2) read delayed by 1.5 µs (`miso_delay` = 75), compared to MOSI data (line 4)

Slave select mode

With `ss_manual`=0 the appropriate slave select line SS `out` of the master is automatically activated upon start of data transfer with **`P2_SPI_Master_Start`** and afterwards deactivated. This is useful if there is only one slave present on the SPI bus.

With manual activation (`ss_manual`=1) you activate each slave select line individually with one of the **`P2_Digout_…`** instructions, before starting the data transfer with **`P2_SPI_Master_Start`**.

All available digital outputs can be used as manually operated slave select lines, also SS1 `out` or SS2 `out` (pin assignment see hardware manual).
Please note: After power-up, digital channels are configured as inputs and can be configured as outputs using **`P2_DigProg`**.

With **`P2_SPI_Config`** you set which voltage level sets the slave select lines as active.

Extended slave select signal

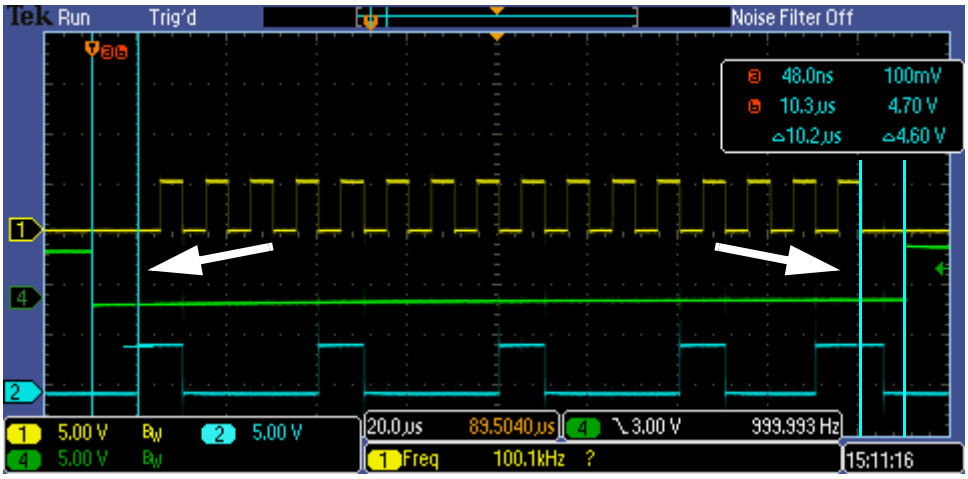
This option is only available, if `ss_manual`=0 sets the automatic activation of the slave select line.

Generally, the slave select signal ends a defined time before the start of the clock signal and ends by the same time after the end of the clock signal. The time interval results to a half clock period plus the additional time from the setting of `ss_time`.

You set the time interval with the parameter `ss_time`. The extension effects an additional time before the clock signal–which means a delayed start of the clock signal–as well as after the clock signal.

The set additional time remains active until a new setting is done.

Example: f_SPI = 100 kHz, `ss_time` = 250
T = 0.5 / 100 kHz + 250 x 20ns = 5µs + 5µs = 10µs



Signal select signal (line 4) starts 10 µs earlier and ends 10 µs later as the clock signal (line 1)

See also

P2_SPI_Mode, P2_SPI_Config, P2_SPI_Master_Set_Clk_Wait, P2_SPI_Slave_Config, P2_SPI_Master_Set_Value32, P2_SPI_Master_Get_Value32, P2_SPI_Master_Start, P2_SPI_Master_Status, P2_SPI_Master_Get_Static_Input

P2_Digout_Long, P2_Digout, P2_Digout_Bits, P2_Digout_Reset, P2_Digout_Set

Valid for

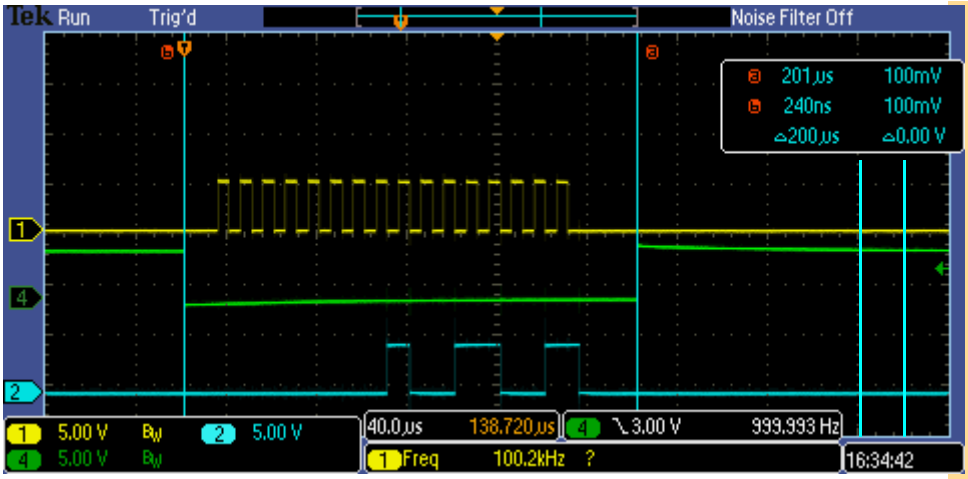
SPI-2-D Rev. E, SPI-2-T Rev. E

Example

```
#Include ADwinPro_All.inc
#Define mod_no 4
#Define master_no 1

Init:
  P2_SPI_Mode(mod_no, 4)      '2 master interfaces
  Rem Configure master:
  Rem CPOL = 0, CPHA = 0; message length 16 Bit
  Rem MSB first; slave select active low
  P2_SPI_Config(mod_no, master_no, 0, 16, 0, 0)
  Rem clk_factor 250: clock frequency 100 kHz
  Rem miso_delay 75: read MISO signal 1.5µs delayed
  Rem ss_time 250: extend slave select signal to 10µs
  Rem activate slave select line manually
  P2_SPI_Master_Config(mod_no, master_no, 250, 75, 250, 1)
  Rem provide SPI message for output
  P2_SPI_Master_Set_Value32(mod_no, master_no, 99h)

Event:
  Rem activate slave select via DIO25, start data transfer
  P2_Digout(mod_no, 25, 0)
  P2_SPI_Master_Start(mod_no, master_no)
  Rem wait 200µs;
  Rem signal time T=16Bits*10µs=160µs + waiting time
  P2_Sleep(20000)
  Rem deactivate slave select line DIO25 of the master
  P2_Digout(mod_no, 25, 1)
```



Slave select (line 4) is activated for 200µs

P2_SPI_Master_Set_Value32

P2_SPI_Master_Set_Value32 provides an SPI message of up to 32 bit length at the master output.

Syntax

```
#Include ADwinPro_All.inc
P2_SPI_Master_Set_Value32(module,channel,mosi_data)
```

Parameters

module	Selected module address (1…15).	LONG
channel	Number (1, 2) of the SPI master.	LONG
mosi_data	SPI message to be sent, length up to 32 bits.	LONG

Notes

You find the pin assignments of the pins DataOut for the output of MOSI data signals in the hardware manual.

The SPI message is only provided to be sent. You start data transfer with the instruction **P2_SPI_Master_Start**.

With **P2_SPI_Master_Set_Value32** you can provide in any case only the lower word (bits 1…32) of an SPI message, even when the SPI bit length is greater than 32 bits. In order to provide both the upper and the lower word, use **P2_SPI_Master_Set_Value64**.

Use **P2_SPI_Config** to configure several parameters for data transfer like bit order and message length.

See also

P2_SPI_Config, P2_SPI_Master_Config, P2_SPI_Master_Set_Value64, P2_SPI_Master_Start, P2_SPI_Master_Status, P2_SPI_Master_Get_Value32

Valid for

SPI-2-D Rev. E, SPI-2-T Rev. E

Example

```
#Include ADwinPro_All.inc
#Define mod_no 4
#Define master_no 2

Init:
  P2_SPI_Mode(mod_no, 4)      '2 master interfaces
  Rem CPOL = 0, CPHA = 0; message length 16 Bit
  P2_SPI_Config(mod_no, master_no, 0, 16, 0, 0)
  Rem set clock frequency 1 MHz etc.
  P2_SPI_Master_Config(mod_no, master_no, 250, 75, 250, 0)

Event:
  Rem provide SPI message for output and
  Rem start data transfer
  P2_SPI_Master_Set_Value32(mod_no, master_no, 12345678h)
  P2_SPI_Master_Start(mod_no, master_no)
```

P2_SPI_Master_Set_Value64

P2_SPI_Master_Set_Value64 provides an SPI message of up to 64 bit length at the master output.

Syntax

```
#Include ADwinPro_All.inc
P2_SPI_Master_Set_Value64(module, channel,
     mosi_high, mosi_low)
```

Parameters

module	Selected module address (1…15).	LONG
channel	Number (1, 2) of the SPI master.	LONG
mosi_data	Upper 32 bits of the SPI message to be sent.	LONG
mosi_low	Lower 32 bits of the SPI message to be sent	LONG

Notes

You find the pin assignments of the pins DataOut for the output of MOSI data signals in the hardware manual.

The SPI message is only provided to be sent. You start data transfer with the instruction **P2_SPI_Master_Start**.

If the SPI bit length is less or equal 32 bits, **P2_SPI_Master_Set_Value32** is the faster instruction.

Use **P2_SPI_Config** to configure several parameters for data transfer like bit order and message length.

See also

P2_SPI_Config, P2_SPI_Master_Config, P2_SPI_Master_Set_Value32, P2_SPI_Master_Start, P2_SPI_Master_Status, P2_SPI_Master_Get_Value32

Valid for

SPI-2-D Rev. E, SPI-2-T Rev. E

Example

```
#Include ADwinPro_All.inc
#Define mod_no 4
#Define master_no 2

Init:
  P2_SPI_Mode(mod_no, 4)      '2 master interfaces
  Rem CPOL = 0, CPHA = 0; message length 64 Bit
  P2_SPI_Config(mod_no, master_no, 0, 64, 0, 0)
  Rem set clock frequency 1 MHz etc.
  P2_SPI_Master_Config(mod_no, master_no, 250, 75, 250, 0)

Event:
  Rem provide 64 bit SPI message for output
  P2_SPI_Master_Set_Value64(mod_no, master_no, 0F678h, 5678h)
  Rem start data transfer
  P2_SPI_Master_Start(mod_no, master_no)
```

P2_SPI_Master_Start

P2_SPI_Master_Start starts the data transfer via SPI bus. If configureed the slave select line of the SPI master is activated automatically.

Syntax

```
#Include ADwinPro_All.inc
P2_SPI_Master_Start(module, channel)
```

Parameters

module	Selected module address (1…15).	LONG
channel	Number (1, 2) of the SPI master.	LONG

Notes

The data transfer generally works in both directions, i.e. the master sends the SPI message, which was recently provided (MOSI). Normally, the addressed slave answers during the same data transfer (MISO).

You can query the end of a data transfer with **P2_SPI_Master_Status**.

P2_SPI_Master_Config configures, if the slave select line of an SPI master is activated manually or automatically. In manual mode you have to activate the slave select line before data transfer and deactivate it afterwards. In automatic mode the master sets its line SS `out` automatically.

Use **P2_SPI_Config** to configure which level sets the slave select line to active.

See also

P2_SPI_Config, P2_SPI_Master_Config, P2_SPI_Master_Set_Value32, P2_SPI_Master_Set_Value64, P2_SPI_Master_Status, P2_SPI_Master_Get_Value32

Valid for

SPI-2-D Rev. E, SPI-2-T Rev. E

Example

```
#Include ADwinPro_All.inc
#Define mod_no 4
#Define master_no 2

Init:
  P2_SPI_Mode(mod_no, 4)      '2 master interfaces
  Rem CPOL = 0, CPHA = 0; message length 64 Bit
  P2_SPI_Config(mod_no, master_no, 0, 64, 0, 0)
  Rem clock frequency 100 kHz
  Rem read MISO signal delayed by 1.5µs
  Rem extend slave select signal by 5µs to 10µs
  Rem activate slave select line automatically
  P2_SPI_Master_Config(mod_no, master_no, 250, 75, 250, 0)

Event:
  Rem provide SPI message for output
  P2_SPI_Master_Set_Value64(mod_no, master_no, 0F678h, 5678h)
  Rem activate slave select line SS out and
  Rem start data transfer
  P2_SPI_Master_Start(mod_no, master_no)
```

P2_SPI_Master_Status

P2_SPI_Master_Status returns if the data transfer of a SPI master is active or inactive.

Syntax

```
#Include ADwinPro_All.inc
ret_val = P2_SPI_Master_Status(module, channel)
```

Parameters

module	Selected module address (1…15).	LONG
channel	Number (1, 2) of the SPI master.	LONG
ret_val	Status of the SPI master: 0: Data transfer is finished; with automatic mode also: slave select line is inactive. 1: Data transfer is still running; with automatic mode also: slave select line is active.	LONG

Notes

As long as the data transfer is active (return value = 1) no other SPI instruction may be processed.

If automatic activation of the slave select line is configured with **P2_SPI_Master_Config**, **P2_SPI_Master_Status** also returns, if the slave select line SS `out` is active or inactive.

Use **P2_SPI_Config** to configure which level sets the slave select line to active.

See also

P2_SPI_Master_Set_Value32, P2_SPI_Master_Set_Value64, P2_SPI_Master_Start, P2_SPI_Master_Get_Value32, P2_SPI_Master_Get_Value64

Valid for

SPI-2-D Rev. E, SPI-2-T Rev. E

Example

```
#Include ADwinPro_All.inc
#Define mod_no 4
#Define master_no 2

Init:
  P2_SPI_Mode(mod_no, 4)      '2 master interfaces
  Rem CPOL = 0, CPHA = 0; message length 64 Bit
  P2_SPI_Config(mod_no, master_no, 0, 64, 0, 0)
  Rem clock frequency 100 kHz
  Rem read MISO signal delayed by 1.5µs
  Rem extend slave select signal by 5µs to 10µs
  Rem activate slave select line automatically
  P2_SPI_Master_Config(mod_no, master_no, 250, 75, 250, 0)

  Rem provide 64 bit SPI message for output
  P2_SPI_Master_Set_Value64(mod_no, master_no, 0F678h, 5678h)

Event:
  Rem activate slave select line SS out and
  Rem start data transfer
  P2_SPI_Master_Start(mod_no, master_no)
  Rem query master status until inactive
  Do
    Par_80 = P2_SPI_Master_Status(mod_no, master_no)
  Until(Par_80 <> 1)
```

P2_SPI_Master_Get_Value32

P2_SPI_Master_Get_Value32 reads a (already received) SPI message of up to 32 bits from the input of the SPI interface.

Syntax

```
#Include ADwinPro_All.inc
ret_val = P2_SPI_Master_Get_Value32(module, channel)
```

Parameters

module	Selected module address (1…15).	LONG
channel	Number (1, 2) of the SPI master.	LONG
ret_val	Received SPI message with up to 32 bits.	LONG

Notes

You find the pin assignments of the pins DataIn for the input of MISO data signals in the hardware manual.

You set the number of transferred bits in an SPI message with **P2_SPI_Config**. If the bit length is less than 32 bits, unused bits in ret_val are set to 0.

You can only read the SPI message when the data transfer has been completed. Query the status with **SPI_Master_Status**.

With **P2_SPI_Master_Get_Value32** you can in any case only read the lower word (bits 1…32) of the SPI message, even when the SPI bit length is greater than 32 bits. In order to read both the upper and the lower word, use **P2_SPI_Master_Get_Value64**.

See also

P2_SPI_Config, P2_SPI_Master_Config, P2_SPI_Master_Set_Value32, P2_SPI_Master_Status, P2_SPI_Master_Get_Value64

Valid for

SPI-2-D Rev. E, SPI-2-T Rev. E

Example

```
#Include ADwinPro_All.inc
#Define mod_no 4
#Define master_no 1

Init:
  P2_SPI_Mode(mod_no, 4)      '2 master interfaces
  Rem CPOL = 0, CPHA = 0; message length 18 Bit
  P2_SPI_Config(mod_no, master_no, 0, 18, 0, 0)
  Rem set clock frequency 1 MHz etc.
  P2_SPI_Master_Config(mod_no, master_no, 250, 75, 250, 0)

Event:
  Rem query the master status until inactive
  Do
    Par_80 = P2_SPI_Master_Status(mod_no, master_no)
  Until(Par_80 <> 1)
  Rem read SPI message
  Rem Par_13 contains a 32 bit value where bits 0..18 contain
  Rem the SPI message while bits 19..31 are 0.
  Par_13 = P2_SPI_Master_Get_Value32(mod_no, master_no)
```

P2_SPI_Master_Get_Value64

P2_SPI_Master_Get_Value64 reads a (already received) SPI message of up to 64 bits from the input of the SPI interface.

Syntax

```
#Include ADwinPro_All.inc
P2_SPI_Master_Get_Value64(module, channel,
     mosi_high, mosi_low)
```

Parameters

module	Selected module address (1…15).	LONG
channel	Number (1, 2) of the SPI master.	LONG
mosi_high	Upper 32 bits of the received SPI message.	LONG
mosi_low	Lower 32 bits of the received SPI message.	LONG

Notes

You find the pin assignments of the pins DataIn for the input of MISO data signals in the hardware manual.

You set the number of transferred bits in an SPI message with **P2_SPI_Config**. If the bit length is less than 64 bits, unused bits in mosi_high (and in mosi_low, if applicable) are set to 0.

You can only read the SPI message when the data transfer has been completed. Query the status with **SPI_Master_Status**.

See also

P2_SPI_Config, P2_SPI_Master_Config, P2_SPI_Master_Set_Value64, P2_SPI_Master_Status, P2_SPI_Master_Get_Value32

Valid for

SPI-2-D Rev. E, SPI-2-T Rev. E

Example

```
#Include ADwinPro_All.inc
#Define mod_no 4
#Define master_no 1

Init:
  P2_SPI_Mode(mod_no, 4)      '2 master interfaces
  Rem CPOL = 0, CPHA = 0; message length 48 Bit
  P2_SPI_Config(mod_no, master_no, 0, 48, 0, 0)
  Rem set clock frequency 1 MHz etc.
  P2_SPI_Master_Config(mod_no, master_no, 250, 75, 250, 0)

Event:
  Rem query the master status until inactive
  Do
    Par_80 = P2_SPI_Master_Status(mod_no, master_no)
  Until(Par_80 <> 1)
  Rem read SPI message
  Rem Par_12 contains the lower 32 bits, Par_13 contains the
  Rem remaining upper 16 bits of the SPI message. The bits
  Rem 16..31 of Par_13 are 0.
  P2_SPI_Master_Get_Value64(mod_no, master_no, Par_13, Par_12)
```

P2_SPI_Master_Get_Static_Input

P2_SPI_Master_Get_Static_Input returns the level of the data line of the SPI bus.

Syntax

```
#Include ADwinPro_All.inc
ret_val = P2_SPI_Master_Get_Static_Input(module,
     channel)
```

Parameters

module	Selected module address (1…15).	LONG
channel	Number (1, 2) of the SPI master.	LONG
ret_val	Level of the data line 0: Level low. 1: Level high.	LONG

Notes

Some SPI slaves use the data line not for data transfer only but also transfer signals to the SPI master. In this case you can use **P2_SPI_Master_Get_Static_Input** to read the level of the data line and react appropriately.

See also

P2_SPI_Mode, P2_SPI_Config, P2_SPI_Master_Config, P2_SPI_Master_Status, P2_SPI_Master_Get_Value32, P2_SPI_Master_Get_Value64

Valid for

SPI-2-D Rev. E, SPI-2-T Rev. E

Example

- / -

P2_SPI_Master_Set_Clk_Wait

P2_SPI_Master_Set_Clk_Wait inserts several waiting times after a selectable number of clocks into the clock signal of the specified SPI master.

Syntax

```
#Include ADwinPro_All.inc
P2_SPI_Master_Set_Clk_Wait(module, channel, clocks,
    half_clk_wait[])
```

Parameters

module	Selected module address (1…15).	LONG
channel	Number (1, 2) of the SPI master.	LONG
clocks	Number (0…16) of periods in the clock signal, after which a waiting time is inserted. Use 0 to deactivate all waiting times.	LONG
half_clk_wait[]	Array with 5 waiting times, given in half periods (1…16). A waiting time is calculated like: t_wait = 0.5 x half_clk_wait[n] / f_SPI	ARRAY LONG

Notes

The period length results from the clock frequency f_SPI, which is set with **P2_SPI_Master_Config**.

Please note, that a waiting time has a length of at least a half period.

Waiting times will not be inserted after the last period of the clock signal.

If SPI bit length and the number of clocks result to 6 or more waiting times to be inserted into the clock signal, the waiting time half_clk_wait[5] will be repeated accordingly.

See also

P2_SPI_Mode, P2_SPI_Config, P2_SPI_Master_Config, P2_SPI_Slave_Config

Valid for

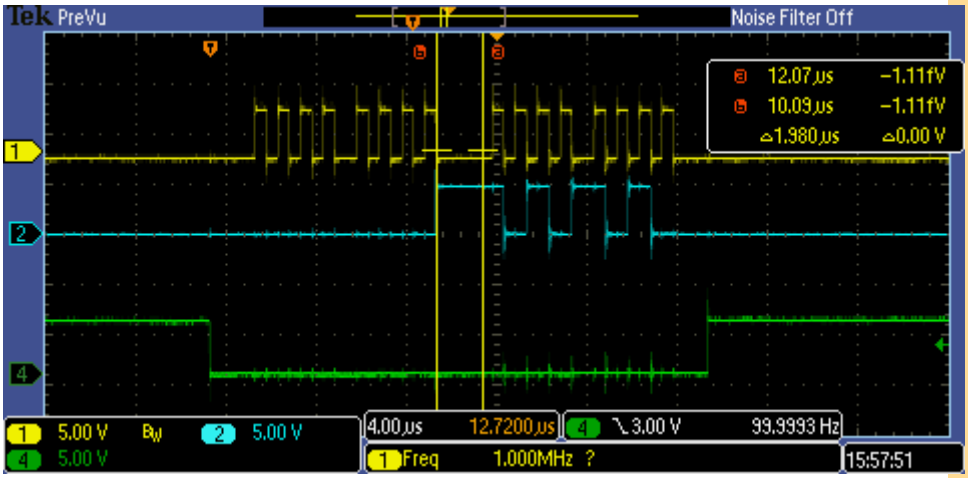
SPI-2-D Rev. E, SPI-2-T Rev. E

Example

```
#Include ADwinPro_All.inc
#Define mod_no 4
#Define master_no 1
Dim clk_wait_arr[5] As Long

Init:
  P2_SPI_Mode(mod_no, 4)      '2 master interfaces
  Rem CPOL = 0, CPHA = 0; message length 16 Bit
  P2_SPI_Config(mod_no, master_no, 0, 16, 0, 0)
  Rem set clock frequency to 1 MHz etc.
  P2_SPI_Master_Config(mod_no, master_no, 250, 75, 250, 0)
  Rem define waiting times
  clk_wait_arr[1] = 1         'waiting time 1: 0.5µs
  clk_wait_arr[2] = 4         'waiting time 2: 2.0µs
  clk_wait_arr[3] = 1         'waiting time 3: 0.5µs
  clk_wait_arr[4] = 2         'waiting time 4: 1.0µs
  clk_wait_arr[5] = 1         'waiting time 5, 6, ..: 0.5µs
  Rem insert waiting time after every 4th period into
  Rem clock signal
  P2_SPI_Master_Set_Clk_Wait(mod_no,master_no,4,clk_wait_arr)

Event:
  Rem provide SPI message for output
  P2_SPI_Master_Set_Value64(mod_no, master_no, 0, 0AAh)
  Rem start data transfer
  P2_SPI_Master_Start(mod_no, master_no)
```



Waiting times in clock signal (yellow) after 4 periods each;
waiting time 2 with 2.0 µs is highlighted

P2_SPI_Slave_Config

P2_SPI_Slave_Config sets (additional) properties of an SPI slave interface of the module.

Syntax

```
#Include ADwinPro_All.inc
P2_SPI_Slave_Config(module, channel, mode)
```

Parameters

module	Selected module address (1…15).	LONG
channel	Number (1, 2) of the interface being configured as SPI slave.	LONG
mode	Set mode "store number of bits": 0: store only SPI message in input Fifo. 1: Store number of received bits and SPI message in input Fifo.	LONG

Notes

As soon as the master deactivates the slave select line, the SPI slave stops receiving data, even when the SPI message has not been transferred completely yet.

The number of received bits is stored–if the option is enabled–as 32 bit value before the appropriate SPI message in the input Fifo.

See also

P2_SPI_Mode, P2_SPI_Config, P2_SPI_Master_Config, P2_SPI_Slave_OutFifo_Write, P2_SPI_Slave_OutFifo_Empty, P2_SPI_Slave_InFifo_Full, P2_SPI_Slave_InFifo_Read, P2_SPI_Slave_Clear_Fifo

Valid for

SPI-2-D Rev. E, SPI-2-T Rev. E

Example

- / -

P2_SPI_Slave_OutFifo_Write

P2_SPI_Slave_OutFifo_Write writes several 32 bit values as SPI messages into the output Fifo of an SPI slave.

Syntax

```
#Include ADwinPro_All.inc
P2_SPI_Slave_OutFifo_Write(module, channel, count,
    array[], array_idx)
```

Parameters

module	Selected module address (1…15).	LONG
channel	Number (1, 2) of the SPI slave.	LONG
count	Number (1…18) of 32 bit values to write.	LONG
array[]	Array which contains the SPI messages to be written.	ARRAY LONG
array_idx	Index (> 0) of the first 32 bit value in array[] to be written.	LONG

Notes

The SPI slave buffers SPI messages in an output Fifo for output. Thus the SPI slave can immediately react to signals of the SPI master.

First check with **P2_SPI_Slave_OutFifo_Empty**, if there are enough unused values in the output Fifo before you write new SPI messages into.

The output Fifo can contain up to 18 values of 32 bit length. According to the SPI bit length either 18 SPI messages of up to 32 bit length or 9 SPI messages of up to 64 bit length can be buffered in the output Fifo.

In array[], the SPI messages must be arranged as follows:

Index in array[]	Bit length 1…32	Bit length 33…64
array_idx	message 1	message 1, lower word
array_idx + 1	message 2	message 1, upper word
array_idx + 2	message 3	message 2, lower word
array_idx + 3	message 4	message 2, upper word
…	…	…
array_idx + 14	message 15	message 8, lower word
array_idx + 15	message 16	message 8, upper word
array_idx + 16	message 17	message 9, lower word
array_idx + 17	message 18	message 9, upper word

With a bit length greater than 32 bit please consider that

- in an SPI message 32 bit values are always used in pairs. Normally `count` will have even values and `array_idx` only odd values.
- in `array[]` the lower word of an SPI message is expected to precede the upper word.

The `array[]` must be dimensioned with at least `array_idx` + `count` - 1 elements.

If the output Fifo runs empty, the recently written SPI message is repeated until you write a new SPI message into the output Fifo.

See also

P2_SPI_Config, P2_SPI_Slave_Config, P2_SPI_Slave_OutFifo_Empty, P2_SPI_Slave_InFifo_Full, P2_SPI_Slave_InFifo_Read, P2_SPI_Slave_Clear_Fifo

Valid for

SPI-2-D Rev. E, SPI-2-T Rev. E

Example

```
#Include ADwinPro_All.inc
#Define mod_no 4
#Define slave_no 1
Dim array[100] As Long

Init:
  P2_SPI_Mode(mod_no, 2)      '2 slave interfaces
  Rem CPOL = 0, CPHA = 0; message length 16 Bit
  P2_SPI_Config(mod_no, slave_no, 0, 16, 0, 0)
  P2_SPI_Slave_Config(mod_no, slave_no, 0)

Event:
  Rem if unused values are available ..
  If (P2_SPI_Slave_OutFifo_Empty(mod_no, slave_no) > 0) Then
    Rem .. provide one SPI message for output
    array[1] = 50
    P2_SPI_Slave_OutFifo_Write(mod_no, slave_no, 1, array, 1)
  EndIf
```

P2_SPI_Slave_OutFifo_Empty

P2_SPI_Slave_OutFifo_Empty returns the number of unused values in the output Fifo of an SPI slave.

Syntax

```
#Include ADwinPro_All.inc
ret_val=P2_SPI_Slave_OutFifo_Empty(module, channel)
```

Parameters

module	Selected module address (1…15).	LONG
channel	Number (1, 2) of the SPI slave.	LONG
ret_val	Number of unused values in the output Fifo.	LONG

Notes

If you want to write data into the output Fifo, you should first check with **P2_SPI_Slave_OutFifo_Empty** if there are enough unused values in the output Fifo.

A value in the output Fifo has a length of 32 bit. With an SPI bit length of less than or equal 32 bit a Fifo value can contain a complete SPI message; for an SPI bit length greater than 32 bit each SPI message requires 2 Fifo values.

See also

P2_SPI_Config, P2_SPI_Slave_Config, P2_SPI_Slave_OutFifo_Write, P2_SPI_Slave_InFifo_Full, P2_SPI_Slave_InFifo_Read, P2_SPI_Slave_Clear_Fifo

Valid for

SPI-2-D Rev. E, SPI-2-T Rev. E

Example

```
#Include ADwinPro_All.inc
#Define mod_no 4
#Define slave_no 1
Dim array[100] As Long

Init:
  P2_SPI_Mode(mod_no, 2)      '2 slave interfaces
  Rem CPOL = 0, CPHA = 0; message length 16 Bit
  P2_SPI_Config(mod_no, slave_no, 0, 16, 0, 0)
  P2_SPI_Slave_Config(mod_no, slave_no, 0)

Event:
  Rem if unused values are available ..
  If (P2_SPI_Slave_OutFifo_Empty(mod_no, slave_no)> 0) Then
    array[1] = 50
    Rem .. provide an SPI message for output
    P2_SPI_Slave_OutFifo_Write(mod_no, slave_no, 1, array, 1)
  EndIf
```

P2_SPI_Slave_InFifo_Full

P2_SPI_Slave_InFifo_Full returns the number of used values (=received 32 bit values) in the input Fifo.

Syntax

```
#Include ADwinPro_All.inc
ret_val = P2_SPI_Slave_InFifo_Full(module, channel)
```

Parameters

module	Selected module address (1…15).	LONG
channel	Number (1, 2) of the SPI slave.	LONG
ret_val	Number of used values in the input Fifo.	LONG

Notes

If you want to read data from the input Fifo, you should first check with **P2_SPI_Slave_OutFifo_Full** if there are any data in the input Fifo. If there is no more data in, an undefined value is returned.

A value in the input Fifo has a length of 32 bit. With an SPI bit length of less than or equal 32 bit a Fifo value contains a complete SPI message; for an SPI bit length greater than 32 bit each SPI message requires 2 Fifo values.
If also the number of received bits (see **P2_SPI_Config**) is buffered in the input Fifo, each SPI message requires 3 Fifo values.

See also

P2_SPI_Config, P2_SPI_Slave_Config, P2_SPI_Slave_OutFifo_Write, P2_SPI_Slave_OutFifo_Empty, P2_SPI_Slave_InFifo_Read, P2_SPI_Slave_Clear_Fifo

Valid for

SPI-2-D Rev. E, SPI-2-T Rev. E

Example

```
#Include ADwinPro_All.inc
#Define mod_no 4
#Define slave_no 1
Dim array[100] As Long
Dim index As Long

Init:
  P2_SPI_Mode(mod_no, 2)      '2 slave interfaces
  Rem Configure slave 1:
  Rem CPOL = 0, CPHA = 1; message length 8 Bit
  Rem MSB first; slave select active high
  P2_SPI_Config(mod_no, slave_no, 1, 8, 0, 0)
  P2_SPI_Slave_Config(mod_no, slave_no, 0)
  index = 1

Event:
  Rem if any data was received ..
  Par_1 = P2_SPI_Slave_InFifo_Full(mod_no, slave_no)
  If (Par_1 > 0) Then
    Rem .. read SPI messages
    P2_SPI_Slave_InFifo_Read(mod_no, slave_no, Par_1, array,
     index)
    index = index + Par_1
    If (index > 100) Then index = 1
  EndIf
```

P2_SPI_Slave_InFifo_Read

P2_SPI_Slave_InFifo_Read reads several 32 bit values as SPI messages from the input Fifo of an SPI slave.

Syntax

```
#Include ADwinPro_All.inc
P2_SPI_Slave_InFifo_Read(module, channel, count,
    array[], array_idx)
```

Parameters

module	Selected module address (1…15).	LONG
channel	Number (1, 2) of the SPI slave.	LONG
count	Number (1…18) of 32 bit values to read.	LONG
array[]	Destination array where read SPI messages are stored.	ARRAY LONG
array_idx	Indes of the first 32 bit value in array[], which was read.	LONG

Notes

The SPI slave collects incoming SPI messages in an input Fifo. A message is stored in the input Fifo as soon as the set bit length is reached or if the slave select line was deactivated before.

If the appropriate option was enabled with **P2_SPI_Slave_Config**, the number of received bits is stored in the input Fifo in addition to each SPI message.

You should first check with **P2_SPI_Slave_OutFifo_Full** if there are any data in the input Fifo before reading new SPI messages.

The input Fifo can contain up to 18 values of 32 bit length. According to the SPI bit length either 18 SPI messages of up to 32 bit length or 9 SPI messages of up to 64 bit length can be buffered in the input Fifo. If also the number of received bits (see **P2_SPI_Config**) is buffered in the input Fifo, either 9 SPI messages of up to 32 bit length or 6 SPI messages of up to 64 bit length can be buffered.

The following table shows how data is stored in the array[]. The array[] must be dimensioned with at least array_idx + count - 1 elements.

Index in array[]	Without number of received bits		With number of received bits	
	bit length 1…32	bit length 33…64	bit length 1…32	bit length 33…64
array_idx	message 1	message 1, lower word	number of bits 1	number of bits 1
array_idx + 1	message 2	message 1, upper word	message 1	message 1, lower word
array_idx + 2	message 3	message 2, lower word	number of bits 2	message 1, upper word
array_idx + 3	message 4	message 2, upper word	message 2	number of bits 2
…	…	…	…	…
array_idx + 14	message 15	message 8, lower word	number of bits 8	message 5, upper word

Index in array[]	Without number of received bits		With number of received bits	
	bit length 1…32	bit length 33…64	bit length 1…32	bit length 33…64
array_idx + 15	message 16	message 8, upper word	SPI-message 8	number of bits 6
array_idx + 16	message 17	message 9, lower word	number of bits 9	message 6, lower word
array_idx + 17	message 18	message 9, upper word	SPI-message 9	message 6, upper word

With a bit length greater than 32 bit please consider that

- in an SPI message 32 bit values are always used in pairs.
- in array[] the lower word of an SPI message is expected to precede the upper word.

If the input Fifo is full, newly incoming SPI messages are not stored and are lost.

See also

P2_SPI_Config, P2_SPI_Slave_Config, P2_SPI_Slave_OutFifo_Write, P2_SPI_Slave_OutFifo_Empty, P2_SPI_Slave_InFifo_Full, P2_SPI_Slave_Clear_Fifo

Valid for

SPI-2-D Rev. E, SPI-2-T Rev. E

Example

```
#Include ADwinPro_All.inc
#Define mod_no 4
#Define slave_no 1
Dim array[100] As Long
Dim index As Long

Init:
  P2_SPI_Mode(mod_no, 2)      '2 slave interfaces
  index = 1
  Rem Configure slave 1:
  Rem CPOL = 0, CPHA = 1; message length 8 Bit
  Rem MSB first; slave select active high
  P2_SPI_Config(mod_no, slave_no, 1, 8, 0, 0)
  P2_SPI_Slave_Config(mod_no, slave_no, 0)
  index = 1

Event:
  Rem if any data was received ..
  Par_1 = P2_SPI_Slave_InFifo_Full(mod_no, slave_no)
  If (Par_1 > 0) Then
    Rem .. read SPI messages
    P2_SPI_Slave_InFifo_Read(mod_no, slave_no, Par_1, array,
     index)
    index = index + Par_1
    If (index > 100) Then index = 1
  EndIf
```

P2_SPI_Slave_Clear_Fifo

P2_SPI_Slave_Clear_Fifo clears the input Fifo and/or the output Fifo of an SPI slave.

Syntax

```
#Include ADwinPro_All.inc
P2_SPI_Slave_Clear_Fifo(module, channel, pattern)
```

Parameters

module	Selected module address (1…15).	LONG
channel	Number (1, 2) of the SPI slave.	LONG
pattern	Bit pattern to select the Fifo of the SPI slave being cleared: Bit 0: Input Fifo (MOSI) Bit = 0: remain Fifo unchanged. Bit = 1: clear Fifo content. Bit 1: Output Fifo (MISO) Bit = 0: remain Fifo unchanged. Bit = 1: clear Fifo content. Bits 2…31: no function.	LONG

Notes

- / -

See also

P2_SPI_Config, P2_SPI_Slave_Config, P2_SPI_Slave_OutFifo_Write, P2_SPI_Slave_OutFifo_Empty, P2_SPI_Slave_InFifo_Full, P2_SPI_Slave_InFifo_Read

Valid for

SPI-2-D Rev. E, SPI-2-T Rev. E

Example

```
#Include ADwinPro_All.inc
#Define mod_no 4
#Define slave_no 1

Init:
  P2_SPI_Mode(mod_no, 2)      '2 slave interfaces
  Rem CPOL = 0, CPHA = 1; message length 8 Bit
  P2_SPI_Config(mod_no, slave_no, 1, 8, 0, 0)
  P2_SPI_Slave_Config(mod_no, slave_no, 0)

  Rem clear both Fifos
  P2_SPI_Slave_Clear_Fifo(mod_no, slave_no, 11b)
```

3.19.1 LS-Bus + Pro II

This section describes instructions which apply to Pro II LS bus modules:

– P2_LS_DIO_Init (page 411)
– P2_LS_DigProg (page 413)
– P2_LS_Dig_IO (page 415)
– P2_LS_Digout_Long (page 417)
– P2_LS_Digin_Long (page 419)
– P2_LS_Get_Output_Status (page 421)
– P2_LS_Watchdog_Init (page 423)
– P2_LS_Watchdog_Reset (page 425)

P2_LS_DIO_Init

P2_LS_DIO_Init initializes the specified module of type HSM-24V on the LS bus via an interface of the Pro II module and returns the error status

Syntax

```
#Include ADwinPro_All.Inc
ret_val = P2_LS_DIO_Init(module,channel,ls-module)
```

Parameters

module	Specified module address (1…15).	LONG
channel	number (1, 2) of the LS bus interface on the Pro II module.	LONG
ls-module	Specified module address on the LS bus (1…15).	LONG
ret_val	Error status. -1: No communication possible with the module. >0: Bit pattern representing the error status. Bit = 0: No error. Bit = 1: Error occurred.	LONG

Bit no.	31…8	7	6	5…4	3	2	1	0
Status	–	Temp2	Temp1	–	WD	Time	Ovr	Par

- :don't care (mask with 0CFh).
Par:Parity error during data transfer on the LS bus.
Ovr:Overrun error during data transfer on the LS bus.
Time:Timeout error during data transfer on the LS bus.
WD:Watchdog was released. The channel drivers are deactivated.
Temp1:Superheating on driver for channels 1…16. Driver is deactivated.
Temp2:Superheating on driver for channels 17…32. Driver is deactivated.

Notes

The instruction only be used in section **Init:**, since it takes long processing time.

The initialization does the following settings:

- All DIO channels are set as inputs.
  Other settings see **P2_LS_DigProg**.
- The over-current status (> ca. 500 mA) is reset.
- The error status for superheating is reset.
- The error status for timeout on the LS bus is reset.

The module stores occurring errors independently from the *ADwin* system. Error bits in the return value therefore may refer to an error which has occurred at an earlier point of time.

The error "superheating" of a driver may only occur, if over-currrent in the range of 150…500 mA is present on several channels at the same time. Irrespective of this, an over-current of mor than 500 mA automatically switches off the concerned channel.

The channels of the module HSM-24V may only be operated in the range of 0…150 mA. This ensures that the module HSM-24V can run continuously without interruption even when all channels are active at the same time.

Valid for

LS-2 Rev. E

See also

P2_LS_DigProg, P2_LS_Dig_IO, P2_LS_Digout_Long, P2_LS_Digin_Long, P2_LS_Get_Output_Status, P2_LS_Watchdog_Init, P2_LS_Watchdog_Reset

Example

```
REM Example process for one module HSM-24V and ADwin-Pro II-LS2
#Include ADwinPro_All.Inc

Init:
  Processdelay = 4000000   '10Hz HP
  Par_1 = P2_LS_DIO_Init(1,2,1)
  Par_2 = P2_LS_DigProg(1,2,1,0Fh) 'channels 1…32 as output
  Par_3= P2_LS_Watchdog_Init(1,2,1,1,1100) 'watchdog time 1.1 sec

Event:
  REM set one channel to high, rotating from 1 to 32
  Inc Par_10
  If (Par_10 >= 32) Then Par_10 = 0
  Par_11 = Shift_Left(1,Par_10)
  REM set channels and read back real state
  Par_12 = P2_LS_Dig_IO(1,2,Par_11)
  Rem reset watchdog
  P2_LS_Watchdog_Reset(1,1)
```

P2_LS_DigProg

P2_LS_DigProg sets the digital channels 1…32 of the specified module of type HSM-24V on the LS bus as inputs or outputs in groups of 8 via an interface of the Pro II module.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = P2_LS_DigProg(module,channel,ls-module,
    pattern)
```

Parameters

module	Specified module address (1…15).	LONG
channel	number (1, 2) of the LS bus interface on the Pro II module.	LONG
ls-module	Specified module address on the LS bus (1…15).	LONG
pattern	Bit pattern, setting the channels as inputs or outputs: Bit = 0: Set channels as inputs. Bit = 1: Set channels as outputs.	LONG

Bit No.	31…4	3	2	1	0
Channel no.	–	32:25	24:17	16:9	8:1

ret_val	Error status. -1: No communication possible with the module. >0: Bit pattern representing the error status. Bit = 0: No error. Bit = 1: Error occurred.	LONG

Bit no.	31…8	7	6	5…4	3	2	1	0
Status	–	Temp2	Temp1	–	WD	Time	Ovr	Par

- :don't care (mask with 0CFh).
Par:Parity error during data transfer on the LS bus.
Ovr:Overrun error during data transfer on the LS bus.
Time:Timeout error during data transfer on the LS bus.
WD:Watchdog was released. The channel drivers are deactivated.
Temp1:Superheating on driver for channels 1…16. Driver is deactivated.
Temp2:Superheating on driver for channels 17…32. Driver is deactivated.

Notes

The instruction only be used in section **Init:**, since it takes long processing time.

After initialization with **P2_LS_DIO_Init** all channels are set as inputs.

The channels may be set as inputs or outputs in groups of 8 only (4 relevant bits only, other bits are ignored).

Valid for

LS-2 Rev. E

See also

P2_LS_DIO_Init, P2_LS_Dig_IO, P2_LS_Digout_Long, P2_LS_Digin_Long, P2_LS_Get_Output_Status, P2_LS_Watchdog_Init, P2_LS_Watchdog_Reset

Example

```
REM Example process for one module HSM-24V and ADwin-Pro II-LS2
#Include ADwinPro_All.Inc

Init:
  Processdelay = 4000000   '10Hz HP
  Par_1 = P2_LS_DIO_Init(1,2,1)
  Par_2 = P2_LS_DigProg(1,2,1,0Fh) 'channels 1…32 as output
  Par_3= P2_LS_Watchdog_Init(1,2,1,1,1100) 'watchdog time 1.1 sec

Event:
  REM set one channel to high, rotating from 1 to 32
  Inc Par_10
  If (Par_10 >= 32) Then Par_10 = 0
  Par_11 = Shift_Left(1,Par_10)
  REM set channels and read back real state
  Par_12 = P2_LS_Dig_IO(1,2,Par_11)
  Rem reset watchdog
  P2_LS_Watchdog_Reset(1,1)
```

P2_LS_Dig_IO

P2_LS_Dig_IO sets all digital outputs of the specified module HSM-24V on the LS bus to the level High or Low and returns the status of all channels as bit pattern.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = P2_LS_Dig_IO(module, channel, pattern)
```

Parameters

module	Specified module address (1…15).	LONG
channel	number (1, 2) of the LS bus interface on the Pro II module.	LONG
pattern	Bit pattern, setting the digital outputs (see table). Bit = 0: Set output to level Low. Bit = 1: Set output to level High.	LONG
ret_val	Bit pattern representing the real state of all digital channels (see table). Bit = 0: Channel has level Low. Bit = 1: Channel has level High.	LONG

Bit No.	31	30	29	…	2	1	0
Channel no.	32	31	30	…	3	2	1

Notes

P2_LS_Dig_IO only runs correctly, if the following conditions are given:

- There is only one module on the LS bus.
- The module is of type HSM-24V.
- The module's address is set to 1.
- After calling **P2_LS_Dig_IO**, no other LS bus instruction will be used.

The channels are set as inputs or outputs using **P2_LS_Digprog**.

The pattern is applied to those channels only, which are set as outputs. Bits for input channels are ignored.

The return value contains the real state of both inputs and outputs. The inputs have a filter causing about 12µs signal delay.

P2_LS_Dig_IO resets the watchdog counter of the module to the start value. The counter remains enabled. The start value is set using **P2_LS_Watchdog_Init**.

Reset the active watchdog timer at least once to the start value within the counting interval, in order to keep the module working.

Valid for

LS-2 Rev. E

See also

P2_LS_DIO_Init, P2_LS_DigProg, P2_LS_Digout_Long, P2_LS_Digin_Long, P2_LS_Get_Output_Status, P2_LS_Watchdog_Init, P2_LS_Watchdog_Reset

Example

```
REM Example process for one module HSM-24V and ADwin-Pro II-LS2
#Include ADwinPro_All.Inc

Init:
  Processdelay = 4000000   '10Hz HP
  Par_1 = P2_LS_DIO_Init(1,2,1)
  Par_2 = P2_LS_DigProg(1,2,1,0Fh) 'channels 1…32 as output
  Par_3= P2_LS_Watchdog_Init(1,2,1,1,1100) 'watchdog time 1.1 sec

Event:
  Rem check for over-current
  Par_5 = P2_LS_Get_Output_Status(1,2,1)
  If (Par_5>0) Then End    'over-current: Exit program

  Rem set one channel to high, rotating from 1 to 32
  Inc Par_10
  If (Par_10 >= 32) Then Par_10 = 0
  Par_11 = Shift_Left(1,Par_10)
  Rem set channels and read back real state
  Par_12 = P2_LS_Dig_IO(1,2,Par_11)
  Rem reset watchdog
  P2_LS_Watchdog_Reset(1,2)
```

P2_LS_Digout_Long

P2_LS_Digout_Long sets or clears all digital outputs of the specified module HSM-24V on the LS bus according to the transferred 32 bit value.

Syntax

```
#Include ADwinPro_All.Inc
P2_LS_Digout_Long(module,channel,ls-module,pattern)
```

Parameters

module	Specified module address (1…15).	LONG
channel	number (1, 2) of the LS bus interface on the Pro II module.	LONG
ls-module	Specified module address on the LS bus (1…15).	LONG
pattern	Bit pattern, setting the digital outputs (see table). Bit = 0: Set outputs to level Low. Bit = 1: Set outputs to level High.	LONG

Bit No.	31	30	…	2	1	0
Channel no.	32	31	…	3	2	1

Notes

The channels are set as inputs or outputs using **P2_LS_DigProg**.

The pattern is applied to those channels only, which are set as outputs. Bits for input channels are ignored.

Valid for

LS-2 Rev. E

See also

P2_LS_DIO_Init, P2_LS_DigProg, P2_LS_Dig_IO, P2_LS_Digin_Long, P2_LS_Get_Output_Status, P2_LS_Watchdog_Init, P2_LS_Watchdog_Reset

Example

```
REM Example process for ADwin-Pro and 2 modules HSM-24V
REM Set process to low priority!
#Include ADwinPro_All.Inc

Init:
  Processdelay = 4000000   '10Hz HP
  REM Settings for LS module 2 via Pro module 3, channel 1
  Par_1 = P2_LS_DIO_Init(3,1,2)
  Par_2 = P2_LS_DigProg(3,1,2,01111b) 'channels 1…32 as output
  Par_3 = P2_LS_Watchdog_Init(3,1,2, 1, 1100) 'watchdog time 1.1
sec

  REM Settings for LS module 4 via Pro module 3, channel 1
  Par_11 = P2_LS_DIO_Init(3,1,4)
  Par_12 = P2_LS_DigProg(3,1,4, 0h) 'channels 1…32 as input
  Par_13 = P2_LS_Watchdog_Init(3,1,4, 1, 1100) 'watchdog time 1.1
sec

Event:
  REM set one channel to high, rotating from 1 to 32
  Inc Par_10
  If (Par_10 >= 32) Then Par_10 = 0
  Par_11 = Shift_Left(1,Par_10)
  REM set channels of LS module 2
  P2_LS_Digout_Long(3,1,2,Par_11)
  REM read channels of LS module 4
  Par_15 = P2_LS_Digin_Long(3,1,4)
  REM reset watchdog
  P2_LS_Watchdog_Reset(3,1)
```

P2_LS_Digin_Long

P2_LS_Digin_Long returns the status of all channels of the specified module HSM-24V on the LS bus as bit pattern.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = P2_LS_Digin_Long(module,channel,ls-module)
```

Parameter

module	Specified module address (1…15).	LONG
channel	number (1, 2) of the LS bus interface on the Pro II module.	LONG
ls-module	Specified module address on the LS bus (1…15).	LONG
ret_val	Bit pattern representing the real state of all digital channels (see table). Bit = 0: Channel has level Low. Bit = 1: Channel has level High.	LONG

Bit no.	31	30	…	2	1	0
Channel no.	32	31	…	3	2	1

Notes

We recommend to set the used channels as inputs with **P2_LS_DigProg** before use.

The return value contains the real state of both inputs and outputs. The inputs have a filter causing about 12µs signal delay.

Valid for

LS-2 Rev. E

See also

P2_LS_DIO_Init, P2_LS_DigProg, P2_LS_Dig_IO, P2_LS_Digout_Long, P2_LS_Get_Output_Status, P2_LS_Watchdog_Init, P2_LS_Watchdog_Reset

Example

```
REM Example process for ADwin-Pro and 2 modules HSM-24V
REM Set process to low priority!
#Include ADwinPro_All.Inc

Init:
  Processdelay = 4000000   '10Hz HP
  REM Settings for LS module 2 via Pro module 3, channel 1
  Par_1 = P2_LS_DIO_Init(3,1,2)
  Par_2 = P2_LS_DigProg(3,1,2,01111b) 'channels 1…32 as output
  Par_3= P2_LS_Watchdog_Init(3,1,2,1,1100) 'watchdog time 1.1 sec

  REM Settings for LS module 4 via Pro module 3, channel 1
  Par_11 = P2_LS_DIO_Init(3,1,4)
  Par_12 = P2_LS_DigProg(3,1,4, 0h) 'channels 1…32 as input
  Par_13 = P2_LS_Watchdog_Init(3,1,4,1,1100) 'watchdog time 1.1s

Event:
  REM set one channel to high, rotating from 1 to 32
  Inc Par_10
  If (Par_10 >= 32) Then Par_10 = 0
  Par_11 = Shift_Left(1,Par_10)
  REM set channels of LS module 2
  P2_LS_Digout_Long(3,1,2,Par_11)
  REM read channels of LS module 4
  Par_15 = P2_LS_Digin_Long(3,1,4)
  REM reset watchdog
  P2_LS_Watchdog_Reset(3,1)
```

P2_LS_Get_Output_Status

P2_LS_Get_Output_Status returns the over-current status of outputs of the specified module HSM-24V on the LS bus as bit pattern.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = P2_LS_Get_Output_Status(module,channel,
    ls-module)
```

Parameters

module	Specified module address (1…15).	LONG
channel	number (1, 2) of the LS bus interface on the Pro II module.	LONG
ls-module	Specified module address on the LS bus (1…15).	LONG
ret_val	Bit pattern. Each bit (31…0) represents the over-current status of a digital output (see table). Bit = 0: Standard status. Bit = 1: Over-current occurred, output disabled.	LONG

Bit no.	31	30	…	2	1	0
Channel no.	32	31	…	3	2	1

Notes

A "superheating" error of a driver may only occur, if over-currrent in the range of 150…500mA is present on several channels at the same time. Irrespective of this, an over-current of more than 500mA automatically switches off the concerned channel.

After a "superheating" errorthe module is reset with **P2_LS_DIO_Init**.

The channels of the module HSM-24V may only be operated in the range of 0…150mA. This ensures the module HSM-24V is working permanently without interruption even if all channels are used in parallel.

Valid for

LS-2 Rev. E

See also

P2_LS_DIO_Init, P2_LS_DigProg, P2_LS_Dig_IO, P2_LS_Digout_Long, P2_LS_Digin_Long, P2_LS_Watchdog_Init, P2_LS_Watchdog_Reset

Example

```
REM Example process for ADwin-Pro and 2 modules HSM-24V
REM Set process to low priority!
#Include ADwinPro_All.Inc

Init:
  Processdelay = 4000000   '10Hz HP
  REM Settings for LS module 2 via Pro module 3, channel 1
  Par_1 = P2_LS_DIO_Init(3,1,2)
  Par_2 = P2_LS_DigProg(3,1,2,01111b) 'channels 1…32 as output
  Par_3= P2_LS_Watchdog_Init(3,1,2,1,1100) 'watchdog time 1.1 sec

  REM Settings for LS module 4 via Pro module 3, channel 1
  Par_11 = P2_LS_DIO_Init(3,1,4)
  Par_12 = P2_LS_DigProg(3,1,4, 0h) 'channels 1…32 as input
  Par_13 = P2_LS_Watchdog_Init(3,1,4,1,1100) 'watchdog time 1.1s

Event:
  Rem check for over-current
  Par_5 = P2_LS_Get_Output_Status(3,1,2)
  Par_5 = Par_5 + P2_LS_Get_Output_Status(3,1,4)
  If (Par_5>0) Then End    'over-current: Exit program

  REM set one channel to high, rotating from 1 to 32
  Inc Par_10
  If (Par_10 >= 32) Then Par_10 = 0
  Par_11 = Shift_Left(1,Par_10)
  REM set channels of LS module 2
  P2_LS_Digout_Long(3,1,2,Par_11)
  REM read channels of LS module 4
  Par_15 = P2_LS_Digin_Long(3,1,4)
  REM reset watchdog
  P2_LS_Watchdog_Reset(3,1)
```

P2_LS_Watchdog_Init

P2_LS_Watchdog_Init enables or disables the watchdog counter of a specified module on the LS bus via an interface of the Pro II module.

If enabled, the counter is set to the start value and is started.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = P2_LS_Watchdog_Init(module, channel,
    ls-module, enable, time)
```

Parameters

module	Specified module address (1…15).	LONG
channel	number (1, 2) of the LS bus interface on the Pro II module.	LONG
ls-module	Specified module address on the LS bus (1…15).	LONG
enable	Set status of watchdog counter: 0 : Disable watchdog counter. 1 : Enable watchdog counter.	LONG
time	Release time (0…107374) of the counter in milliseconds.	LONG
ret_val	Error status. -1: No communication possible with the module. >0: Bit pattern representing the error status. Bit = 0: No error. Bit = 1: Error occurred.	LONG

Bit no.	31…8	7	6	5…4	3	2	1	0
Status	–	Temp2	Temp1	–	WD	Time	Ovr	Par

- :don't care (mask with **0CFh**)
Par:Parity error during data transfer on the LS bus.
Ovr: Overrun error during data transfer on the LS bus.
Time: Timeout error during data transfer on the LS bus.
WD: Watchdog was released. The channel drivers are deactivated.
Temp1: Superheating on driver for channels 1…16. Driver is deactivated.
Temp2: Superheating on driver for channels 17…32. Driver is deactivated.

Notes

The instruction only be used in section **Init:**, since it takes long processing time.

As long as the watchdog counter is enabled, it decrements the counter value continuously. After the set release time the counter value reaches 0 (zero). If so, the module assumes a malfunction and stops; thus, all output signals are reset.

After power-up of the module the counter is set to the start value 10 ms and the watchdog counter is enabled.

Reset the active watchdog timer at least once to the start value within the counting interval, in order to keep the module working. To reset the module use any module specific instruction or **P2_LS_Watchdog_Reset**.

The watchdog function is used as to monitor the connection between *ADwin* system and LS bus module.

Valid for

LS-2 Rev. E

See also

P2_LS_DIO_Init, P2_LS_DigProg, P2_LS_Dig_IO, P2_LS_Digout_Long, P2_LS_Digin_Long, P2_LS_Get_Output_Status, P2_LS_Watchdog_Reset

Example

```
REM Example process for one module HSM-24V and ADwin-Pro II-LS2
#Include ADwinPro_All.Inc

Init:
  Processdelay = 4000000   '10Hz HP
  Par_1 = P2_LS_DIO_Init(1,2,1)
  Par_2 = P2_LS_DigProg(1,2,1,0Fh) 'channels 1…32 as output
  Par_3= P2_LS_Watchdog_Init(1,2,1,1,1100) 'watchdog time 1.1 sec

Event:
  REM set one channel to high, rotating from 1 to 32
  Inc Par_10
  If (Par_10 >= 32) Then Par_10 = 0
  Par_11 = Shift_Left(1,Par_10)
  REM set channels and read back real state
  Par_12 = P2_LS_Dig_IO(1,2,Par_11)
  REM reset watchdog
  P2_LS_Watchdog_Reset(1,2)
```

P2_LS_Watchdog_Reset

P2_LS_Watchdog_Reset resets the watchdog counters of all modules on the LS bus to the appropriate start value. The counters remain enabled.

Syntax

```
#Include ADwinPro_All.Inc
P2_LS_Watchdog_Reset(module, channel)
```

Parameters

module	Specified module address (1…15).	LONG
channel	number (1, 2) of the LS bus interface on the Pro II module.	LONG

Notes

As long as a watchdog counter is enabled, it decrements the counter value continuously. After the set release time the counter value reaches 0 (zero). If so, the module assumes a malfunction and stops; thus, all output signals are reset.

Reset the active watchdog timer at least once to the start value within the counting interval, in order to keep the module working. To reset the module you may also use any module specific instruction.

The watchdog function is used as to monitor the connection between *ADwin* system and LS bus module.

Valid for

LS-2 Rev. E

See also

P2_LS_DIO_Init, P2_LS_DigProg, P2_LS_Dig_IO, P2_LS_Digout_Long, P2_LS_Digin_Long, P2_LS_Get_Output_Status, P2_LS_Watchdog_Init

Example

```
REM Example process for ADwin-Pro and 2 modules HSM-24V
REM Set process to low priority!
#Include ADwinPro_All.Inc

Init:
  Processdelay = 4000000   '10Hz HP
  REM Settings for LS module 2 via Pro module 3, channel 1
  Par_1 = P2_LS_DIO_Init(3,1,2)
  Par_2 = P2_LS_DigProg(3,1,2,01111b) 'channels 1…32 as output
  Par_3= P2_LS_Watchdog_Init(3,1,2,1,1100) 'watchdog time 1.1 sec

  REM Settings for LS module 4 via Pro module 3, channel 1
  Par_11 = P2_LS_DIO_Init(3,1,4)
  Par_12 = P2_LS_DigProg(3,1,4, 0h) 'channels 1…32 as input
  Par_13 = P2_LS_Watchdog_Init(3,1,4,1,1100) 'watchdog time 1.1s

Event:
  REM set one channel to high, rotating from 1 to 32
  Inc Par_10
  If (Par_10 >= 32) Then Par_10 = 0
  Par_11 = Shift_Left(1,Par_10)
  REM set channels of LS module 2
  P2_LS_Digout_Long(3,1,2,Par_11)
  REM read channels of LS module 4
  Par_15 = P2_LS_Digin_Long(3,1,4)
  REM reset watchdog
  P2_LS_Watchdog_Reset(3,1)
```

4 Program Examples

The following examples are available:

– Online Evaluation of Measurement Data, page 427
– Digital Proportional Controller, page 427
– Data Exchange with DATA arrays, page 428
– Digital PID Controller, page 428
– Data exchange with fieldbus, page 383
– Examples for RS232 and RS485 (Pro II):
  • RS232: Send and receive, page 431
  • RS232: Send string instruction, page 432
  • RS232: Receive string instruction, page 432
  • RS485: Receive and send, page 433
– Continuous signal conversion: Convert 1 channel, page 435

Most examples are stored as program files in the directory <C:\ADwin\ADbasic\samples_ADwin_ProII>.

4.1 Online Evaluation of Measurement Data

The program <ProII_DMO1.BAS> searches for the maximum and minimum value out of 1000 measurements of ADC1 and writes the result to the variables Par_1 and Par_2.

You require an A/D module Pro II-AIn-x/x with module address 1 and an input signal at channel 1 of the module.

```
#Include ADwinPro_All.Inc  'Include file
#Define limit 65535       'max. 16 bit ADC value
#Define module 1          'module number
#Define input 1           'input number
Dim i1, iw, max, min As Long

Init:
  i1 = 1                  7'reset sample counter
  max = 0                 'initial maximum value
  min = limit             'initial minimum value
  Par_10 = 0              'init End-Flag
  Processdelay = 1 * 3E5  'cycle-time of 1ms

Event:
  iw = P2_ADC(module, 1)   'get sample
  Rem for modules Pro II-AIn-F-x/x, delete the previous line and
  Rem use the following line instead (without comment char ')
  'iw = P2_ADCF(module, 1)
  If (iw > max) Then max = iw 'new maximum sample?
  If (iw < min) Then min = iw 'new minimum sample?
  Inc i1                   'increment index
  If (i1 > 1000) Then      '1000 samples done?
    i1 = 1                 'reset index
    Par_1 = min            'write minimum value
    Par_2 = max            'write maximum value
    max = 0                'reset minimum value
    min = 65535            'reset maximum value
    Par_10 = 1             'set End-Flag
  EndIf
```

4.2 Digital Proportional Controller

The program <ProII_DMO2.BAS> is a digital proportional controller. The setpoint is specified by Par_1, the gain by Par_2.

You require:

– A/D module Pro II-AIn-x/x with module address 1.
– D/A module Pro II-AOut-x/x with module address 2.
– An external controlled system that receives the signal from output 1 of the D/A module and returns a signal to input1 of the A/D module.

```
#Include ADwinPro_All.Inc
#Define offset 32768       '0V for 16 bit ADC/DAC-systems

REM cd: control deviation; av: actuating value
Dim cd, av As Integer

Init:
  Par_1 = offset           'initial setpoint
  Par_2 = 10               'initial gain
  Processdelay = 40000     'cycle-time of 1ms (T9)

Event:
 Rem Read signal from input 1 of A/D module
 cd = Par_1 - P2_ADC(1, 1)'compute control deviation (cd)
 Rem for modules Pro II-AIn-F-x/x, delete the previous line and
 Rem use the following line instead (without comment char ')
 'cd = Par_1 - P2_ADCF(1, 1)
 av = cd * Par_2 + offset'compute actuating value (av)
 Rem Write actuating value to output 1 of D/A module
 P2_DAC(2, 1, av)         'output actuating value on DAC #1
```

4.3 Data Exchange with DATA arrays

The program <ProII_DM03.BAS> measures the analog input 1 of the A/D module with address 1 and sets an end flag after 1000 measurements to indicate that the computer can now get the measurement data. The data are transferred by using the array Data_1.

You require an A/D module Pro II-AIn-x/x with module address 1.

```
#Include ADwinPro_All.Inc
Dim Data_1[1000] As Integer
Dim index As Integer

Init:
 Par_10 = 0
 index = 0                'reset array pointer
 Processdelay = 40000     'cycle-time of 1ms (T9)

Event:
 index = index + 1        'increment array pointer
 If (index > 1000) Then   '1000 samples done?
'  ACTIVATE_PC            'set ACTIVATE_PC flag (only necessary
                          'for TestPoint)
   Par_10 = 1             'set End-Flag
   End                    'terminate process
 EndIf
 Data_1[index] = P2_ADC(1, 1) 'acquire sample and save in array
 Rem for modules Pro II-AIn-F-x/x, delete the previous line and
 Rem use the following line instead (without comment char ')
 'Data_1[index] = P2_ADCF(1, 1)
```

4.4 Digital PID Controller

The program <ProII_DM06.BAS> is a digital PID controller.

Before starting the PID controller the global variables must be set to the controller's values.

You require:

– A/D module Pro II-AIn-x/x with module address 1.
– D/A module Pro II-AOut-x/x with module address 2.
– An external controlled system that receives the signal from output 1 of the D/A module and returns a signal to input1 of the A/D module.

Calculation on the PC:

The control coefficients are calculated on the computer and transferred as global variables to the processor of the *ADwin-Pro* system. Vice versa the information is returned from the program to the PC.

Controller parameter settings	
`FPar_2`	gain of the controller
`FPar_3`	integration time of the controller
`FPar_4`	differentation time of the controller
`Par_1`	Setpoint in digits
`Par_6`	controller sampling rate in units of 3.3ns
Information from the program	
`Par_5`	array index (of `Data_1`) of control deviation
`Par_9`	mean value of control deviation
`Par_10`	Flag: All samples are done
`Data_1[]`	Array holding all control deviations

***ADbasic* Program:**

The address of the A/D module is set to 1 in this example, the address of the analog output module is set to 2.

Please note that you will get a time saving effect, when calculation and output of the actuating value is executed during the necessary waiting period during reading the control deviation (after **`P2_Set_Mux`** and **`P2_Start_Conv`**).

The consequence is that the output actuating value is calculated from the control deviation of the previous process call.

```
#Include ADwinPro_All.Inc
#Define offset 32768       '0V output
#Define module_ad 1        'module number
#Define module_da 2        'module number

Dim Data_1[4000] As Long
Dim av, cd, cdo, sum As Long
Dim diff As Float

Init:
  sum = 0                  'initial value of integral part
  cd = P2_ADC(module_ad, 1)'initial value of control deviation
                           '(cd) & MUX to Ch #1
  Par_5 = 1                'set array index
  If (FPar_3<75E2) Then FPar_3 = 75E2 'check min. integration time
  If (Par_6<3E5) Then Par_6 = 3E5'allow only cycle times >= 1ms
  Processdelay = Par_6     'set cycle-time

Event:
  Rem compute actuating value
  av = FPar_2 * (cd + sum / FPar_3 + diff * FPar_4)
  P2_Start_Conv(module_ad) 'start conversion ADC #1
  Rem while conversion is running …
  P2_DAC(module_da,1,av+offset)'output actuating value at DAC #1
  cdo = cd                 'keep control deviation in mind
  P2_Wait_EOC(module_ad)   'wait until end-of-conversion of ADC
  cd = Par_1 - P2_Read_ADC(module_ad)'compute control deviation
  FPar_9 = FPar_9 * 0.99 + cd * 0.01'mean value of control
                           'deviation
  sum = sum + cd           'calculate integral
  If (sum > 2E6) Then sum = 2E6 'positive limit of integral
  If (sum < -2E6) Then sum = -2E6 'negative limit of integral
  diff = (cd - cdo)        'calculate deviation difference
  Data_1[Par_5] = cd       'write control deviation in a buffer
  Inc Par_5                'increment buffer index
  If (Par_5 >= 4000) Then  '4000 samples done?
    Par_10 = 1             'set End-flag
    Par_5 = 1              'reset array index
  EndIf

Finish:
  P2_DAC(module_da,1,av+offset)'analog output #1 to 0V

  Rem Note: For modules Pro II-AIn-F-x/x, the A/D instructions
  Rem must be renamed, parameters are left unchanged:
  Rem * P2_ADC -> P2_ADCF
  Rem * P2_Start_Conv -> P2_Start_ConvF
  Rem * P2_Wait_EOC -> P2_Wait_EOCF
  Rem * P2_Read_ADC -> P2_Read_ADCF
```

4.5 Examples for RS232 and RS485 (Pro II)

The following examples are complete programs for sending and receivingof data and strings with RS232 or RS485.

You require a module Pro II-RSxxx with module address 1.

RS232:
Send and receive

The following program illustrates the initialization of the serial RS232 interface in the **Init:** section and cyclic data read and write in the **Event:** section. The process is timer-controlled.

```
Rem The program initializes the serial interfaces in
Rem the INIT: section.
Rem In the EVENT: section data are exchanged between interfaces
Rem 1 & 2 of the RSxxx module.
Rem With this program both interfaces can be tested.
Rem To do so, connect the interfaces with each other before
Rem starting the program.

#Include ADwinPro_All.Inc
#Define num_data 1000      'number of send and receive data
#Define module 1           'Module address
Dim Data_1[num_data] As Long'send data
Dim Data_2[num_data] As Long'receive data
Dim i As Long               'count variable

Init:
  P2_RS_Reset(module)
  For i = 1 To num_data     'initialize send data
    Data_1[i] = i And 0FFh
  Next i
  Rem Initialize interfaces 1 and 2:
  Rem 9600 Baud, no parity bit, 8 data bits, 2 stop bits,
  Rem no handshake
  P2_RS_Init(module, 1, 9600, 0, 8, 1, 0)
  P2_RS_Init(module, 2, 9600, 0, 8, 1, 0)
  Par_1 = 1
  Par_4 = 1

Event:
  Rem read and write a data set
  If (Par_1 <= num_data) Then'send data
    Par_2 = P2_Write_FIFO(module, 1, Data_1[Par_1])
    If (Par_2 = 0) Then Inc Par_1
  EndIf

  Par_3 = P2_Read_FIFO(module, 2)'read data
  If (Par_3 <> -1) Then
    Data_2[Par_4] = Par_3
    Inc Par_4
  EndIf
  If (Par_4 > num_data) Then End 'all data are transferred
```

RS232:
Send string instruction

You require a module Pro II-RSxxx with module address 1.

Many devices with an RS232 interface can be controlled using string instructions. The following 2 programs show how to send a string in one process and how to receive the string with another process. Both programs are available on the *ADwin* CDROM.

The programs can be used on the same module but with different interfaces. Please pay attention to the remarks in the programs.

The program `RS232_send_string.BAS` first initializes interface 1. In the **Event** section the interface 1 sends a string char by char. In the **Finish** section the character "#" is used as an end marker. It may be replaced by any other character.

```
' Process for RS232 communication: sending a string
' ++++++++++++++++++++++++++++++++++++++++++++++++++++++++
' The program may run together with RS232_receive_string.BAS
' on the same module.  If so, please follow these instructions:
' - connect the interfaces with each other
' - compile and start RS232_receive_string.BAS
' - compile and start RS232_send_string.BAS

#Include ADwinPro_All.Inc

Rem import string library
Import string.lib

#Define rs_adr 1           'module address
#Define rs_no 1            'interface number
#Define s_endchar "#"      'end marker "#"
#Define s_send Data_1
#Define str_len 50         'length of send string

Dim s_send[str_len] As String 'send string
Dim s_temp[1] As String    'single char
Dim sp As Long             'send pointer

Init:
  Rem 0.25 s
  Processdelay = 75E6
  Rem A reset is allowed only once on a module!
  'P2_RS_Reset(rs_adr)     'reset RS module
  P2_RS_Init(rs_adr,rs_no,9600,0,8,0,0) 'init RS interface
  sp=1                     'initialize pointer
  s_send = „This is a TESTSTRING“ 'send string

Event:
  StrMid(s_send, sp, 1, s_temp) 'read next char of string
  Par_11 = Asc(s_temp)     'get ascii code of char
  If (Par_11 = 0) Then End 'quit when all chars are sent
  Par_12 = P2_Write_FIFO(rs_adr, rs_no, Par_11) 'send code
  Rem increase pointer, else send again
  If (Par_12 = 0) Then Inc sp
  Rem quit when all chars are sent
  If (sp > str_len) Then End

Finish:
  Do                       'send End marker "#"
    Par_11 = Asc(s_endchar) 'get ascii code
    Par_12 = P2_Write_FIFO(rs_adr, rs_no, Par_11) 'send code
  Until (Par_12 = 0)
```

RS232:
Receive string instruction

You require a module Pro II-RSxxx with module address 1.

The program RS232_receive_string.BAS first initializes interface 2. In the **Event** section the interface 2 receives a string until the end marker char is received (or the receiving string is full)

```
'' Process for RS232-communication: Receiving a string.
' ++++++++++++++++++++++++++++++++++++++++++++++++++++++
' The program may run together with RS232_send_string.BAS
' on the same module. If so, please follow these instructions:
' - connect the interfaces with each other
' - compile and start RS232_receive_string.BAS
' - compile and start RS232_send_string.BAS
#Include ADwinPro_All.Inc

Rem import string library
Import string.lib

#Define rs_adr 1           'module address
#Define rs_no 2            'interface number
#Define s_receive Data_2
#Define str_len 50         'max. length of received string

Dim s_receive[str_len] As String 'received string
Dim s_temp[1] As String    'single char
Dim s_endchar[1] As String 'end marker
Dim endflag As Long
Dim rp As Long             'receive pointer

Init:
  Rem 0.25 s
  Processdelay = 75000000

  Rem A reset is allowed only once on a module!
  P2_RS_Reset(rs_adr)      'reset RS module
  P2_RS_Init(rs_adr,rs_no,9600,0,8,0,0) 'init RS interface
  rp = 0                   'initialize receive pointer
  s_receive = ""           'initialize receive string
  s_endchar = "#"          'end marker

Event:
  Par_21 = P2_Read_FIFO(rs_adr, rs_no) 'receive status / char
  If (Par_21 <> -1) Then
    Chr(Par_21,s_temp)     'get char from ascii value
    Inc rp                 'increase receive pointer
    Rem End marker received or string full?
    endflag = StrComp(s_temp, s_endchar)
    If ((endflag=0) Or (rp>str_len)) Then End
    s_receive = s_receive + s_temp 'save char to string
  EndIf
```

You require a module Pro II-RSxxx with module address 1.

RS485:
Receive and send

In this example the RS485 interface 2 is a passive participant, which reads data coming from the bus. If a specified byte (55) is received, the interface becomes active and starts sending the value 44.

```
#Include ADwinPro_All.Inc
#Define rs_adr 1

Dim ret_val As Long
Dim val As Long

Init:
  P2_RS_Reset(rs_adr)
  P2_RS_Init(rs_adr,2,38400,0,8,0,3)
  P2_RS485_Send(rs_adr,2,0) 'set channel 2 as receiving

Event:
   val = P2_Read_FIFO(rs_adr,2) 'read data
   If (val = 55) Then
    P2_RS485_Send(rs_adr,2,1) 'set channel 2 as sending
    ret_val = P2_Write_FIFO(rs_adr,2,44) 'write data
   EndIf
```

4.6 Continuous signal conversion

The modules Pro II AIn-F-4/14 Rev. E and Pro II AIn-F-8/14 Rev. E allow for very fast, continuous signal conversion. In parallel to conversion, the data must also be read and if need be processed.

Hereafter there are examples for continuous signal conversion.

– Convert 1 channel

Example files are stored in the directory <C:\ADwin\ADbasic\samples_ADwin_ProII>.

Convert 1 channel

You need

– one module Pro II AIn-F-x/14 Rev. E with module address 1.

– an analog signal at input channel 1.

The program <PROII-F-x-14-CONT-1CH.BAS> does a continuous burst sequence on input channel 1 with a clock rate 25MHz. The memory is set to hold 20,000 measurement values.

During the running burst sequence the measurement values are read into the global array `Data_1` (a FIFO array is not available). The parallel conversion and read-out calls for an adjustment to each other. For this the memory area is divided into 4 ranges of 5,000 values. That range will only be read, which has just been written completely.

In the same way, an adjustment of conversion rate and read-out rate is necessary. The process cycle is set by **`Processdelay`** = **`20000`** (= 15kHz) in a way, so the read-out rate in average is a multiple of the conversion rate 25MHz:

$$15\text{kHz} \cdot 5000 \;=\; \text{Read-out rate } 75 \text{ kHz} > \text{Conversion rate } 25\text{MHz}$$

For changes of the example please note: If the processing time of the **`Event:`** section rises, e.g. by processing measurement values, the read-out rate may be too low to read all converted values. In this case measurement values will be lost, because they are overwritten. Thus, you have to adjust conversion rate and read-out rate anew.

```
#Include ADwinPro_All.Inc  'include file
#Define module 1           'module no.
#Define d1 Data_1          'holds values of channel 1
#Define mem_idx Par_1      'mem position of last written value
#Define max_val 20000      'no. of values
#Define seg1 max_val/8     'end of segment 1
#Define seg2 max_val/4     'end of segment 2
#Define seg3 max_val/8*3   'end of segment 3
#Define blk max_val/4      'read block size

Dim d1[max_val] As Long    'destination array
Dim pattern As Long        'bit pattern to address one module
Dim segment As Long        'segment that is currently written

Init:
  REM 1 channel continuous, mem for max_val values, 25 MHz
  P2_Burst_Init(module,1,0,max_val,2,010b)
  pattern = Shift_Left(1,module-1)'address this module only
  P2_Burst_Start(pattern)
  segment = 1              'start with memory segment 1
  Processdelay = 20000     'cycle time 66.6 µs -> 15 kHz

Event:
  mem_idx = P2_Burst_Read_Index(module) 'get current mem index
  If (segment = 1) Then     'read 1. segment
    If ((mem_idx > seg1) And (mem_idx < seg3)) Then
   REM memory index is in segments 2 or 3: read segment 1
     P2_Burst_Read_Unpacked1(module,blk,0,Data_1,1,3)
     segment = 2
    EndIf
  EndIf

  If (segment = 2) Then     'read 2. segment
    If (mem_idx > seg2) Then
     REM memory index is in segments 3 or 4: read segment 2
     P2_Burst_Read_Unpacked1(module,blk,seg1,Data_1,blk+1,3)
     segment = 3
    EndIf
  EndIf

  If (segment = 3) Then     'read 3. segment
    If ((mem_idx > seg3) Or (mem_idx < seg1)) Then
     REM memory index is in segments 4 or 1: read segment 3
     P2_Burst_Read_Unpacked1(module,blk,seg2,Data_1,blk*2+1,3)
     segment = 4
    EndIf
  EndIf

  If (segment = 4) Then    'read 4. segment
    If (mem_idx < seg2) Then
     REM memory index is in segments 1 or 2: read segment 4
     P2_Burst_Read_Unpacked1(module,blk,seg3,Data_1,blk*3+1,3)
     segment = 1
    EndIf
  EndIf
```

Instruction Lists

A.1 Alphabetic Instruction List

A

P2_ADC · 37
P2_ADC24 · 38
P2_ADCF · 100
P2_ADCF24 · 101
P2_ADCF_Mode · 102
P2_ADCF_Read_Limit · 105
P2_ADCF_Read_Min_Max4 · 108
P2_ADCF_Read_Min_Max8 · 110
P2_ADCF_Reset_Min_Max · 107
P2_ADCF_Set_Limit · 106
P2_ADC_Read_Limit · 39
P2_ADC_Set_Limit · 41
P2_ARINC_Config_Receive · 310
ARINC_Create_Value32 · 315
P2_ARINC_Read_Receive_Fifo · 318
P2_ARINC_Receive_Fifo_Empty · 317
P2_ARINC_Reset · 307
P2_ARINC_Set_Labels · 320
ARINC_Split_Value32 · 319
P2_ARINC_Transmit_Enable · 316
P2_ARINC_Transmit_Fifo_Empty · 313
P2_ARINC_Transmit_Fifo_Full · 312
P2_ARINC_Write_Transmit_Fifo · 314

B

P2_Burst_CRead_Pos_Unpacked1 · 68
P2_Burst_CRead_Pos_Unpacked2 · 70
P2_Burst_CRead_Pos_Unpacked4 · 72
P2_Burst_CRead_Pos_Unpacked8 · 74
P2_Burst_CRead_Unpacked1 · 60
P2_Burst_CRead_Unpacked2 · 62
P2_Burst_CRead_Unpacked4 · 64
P2_Burst_CRead_Unpacked8 · 66
P2_Burst_Init · 76
P2_Burst_Read · 82
P2_Burst_Read_Index · 80
P2_Burst_Read_Unpacked1 · 84
P2_Burst_Read_Unpacked2 · 86
P2_Burst_Read_Unpacked4 · 88
P2_Burst_Read_Unpacked8 · 90
P2_Burst_Reset · 92
P2_Burst_Start · 94
P2_Burst_Status · 95
P2_Burst_Stop · 97

C

P2_CAN_Interrupt_Source · 246
CAN_Msg (Pro II) · 244
P2_CAN_Set_LED · 248
P2_Check_LED · 5
P2_Check_Shift_Reg · 266
P2_Cnt_Clear · 188
P2_Cnt_Enable · 189
P2_Cnt_Get_PW · 191
P2_Cnt_Get_PW_HL · 192
P2_Cnt_Get_Status · 190
P2_Cnt_Latch · 193
P2_Cnt_Mode · 194
P2_Cnt_PW_Enable · 196
P2_Cnt_PW_Latch · 197
P2_Cnt_Read · 198
P2_Cnt_Read4 · 199
P2_Cnt_Read_Int_Register · 200
P2_Cnt_Read_Latch · 201
P2_Cnt_Read_Latch4 · 202
P2_Cnt_Sync_Latch · 203
P2_Comp_Filter_Init · 151
P2_Comp_Init · 149
P2_Comp_Set · 152
CPU_Digin (T11) · 22
CPU_Digout · 23
CPU_Dig_IO_Config · 24
CPU_Event_Config · 25

D

P2_DAC · 128
P2_DAC1_DIO · 141
P2_DAC4 · 129
P2_DAC4_Packed · 130
P2_DAC8 · 132
P2_DAC8_Packed · 133
P2_DAC_Ramp_Buffer_Free · 146
P2_DAC_Ramp_Status · 144
P2_DAC_Ramp_Stop · 147
P2_DAC_Ramp_Write · 142
P2_Digin_Edge · 158
P2_Digin_Fifo_Clear · 159
P2_Digin_Fifo_Enable · 160
P2_Digin_Fifo_Full · 162
P2_Digin_Fifo_Read · 163
P2_Digin_Fifo_Read_Fast · 165
P2_Digin_Fifo_Read_Timer · 167
P2_Digin_Filter_Init · 168
P2_Digin_Long · 169
P2_Digout · 170
P2_Digout_Bits · 171
P2_Digout_Fifo_Clear · 173
P2_Digout_Fifo_Empty · 174
P2_Digout_Fifo_Enable · 175
P2_Digout_Fifo_Read_Timer · 176
P2_Digout_Fifo_Start · 177
P2_Digout_Fifo_Write · 178
P2_Digout_Long · 180
P2_Digout_Reset · 181
P2_Digout_Set · 182
P2_DigProg · 183
P2_DigProg_Bits · 184
P2_Digprog_Set_IO_Level · 185

P2_Dig_Fifo_Mode · 153
P2_Dig_Latch · 155
P2_Dig_Read_Latch · 156
P2_Dig_Write_Latch · 157

E

P2_ECAT_Get_State · 323
P2_ECAT_Get_Version · 322
P2_ECAT_Init · 324
P2_ECAT_Read_Data_16F · 329
P2_ECAT_Read_Data_16L · 327
P2_ECAT_Set_Mode · 326
P2_ECAT_Write_Data_16F · 330
P2_ECAT_Write_Data_16L · 328
P2_En_Interrupt · 249
P2_En_Receive · 251
P2_En_Transmit · 252
P2_Event2_Config · 10
P2_Event_Config · 9
P2_Event_Enable · 7
P2_Event_Read · 12

F

P2_FlexRay_Get_Version · 332
P2_FlexRay_Init · 333
P2_FlexRay_Read_Word · 335
P2_FlexRay_Reset · 336
P2_FlexRay_Set_LED · 337
P2_FlexRay_Write_Word · 338

G

P2_Get_CAN_Reg · 253
P2_Get_Digout_Long · 186
P2_Get_RS · 267

I

P2_Init_CAN · 254
P2_Init_Profibus · 293

L

P2_LIN_Get_Version · 285
P2_LIN_Init · 280
P2_LIN_Init_Apply · 283
P2_LIN_Init_Write · 282
P2_LIN_Msg_Transmit · 290
P2_LIN_Msg_Write · 288
P2_LIN_Read_Dat · 286
P2_LIN_Reset · 284
P2_LIN_Set_LED · 291

M

P2_MIL_Get_Register · 305
P2_MIL_Reset · 297
P2_MIL_Set_LED · 303
P2_MIL_Set_Register · 304
P2_MIL_SMT_Init · 298
P2_MIL_SMT_Set_All_Filters · 301
P2_MIL_SMT_Set_Filter · 302
P2_MIO_Digin_Long · 30
P2_MIO_Digout · 31
P2_MIO_Digout_Long · 32
P2_MIO_DigProg · 33
P2_MIO_Dig_Latch · 27
P2_MIO_Dig_Read_Latch · 28
P2_MIO_Dig_Write_Latch · 29

P

P2_LS_Digin_Long · 419
P2_LS_Digout_Long · 417
P2_LS_DigProg · 413
P2_LS_Dig_IO · 415
P2_LS_DIO_Init · 411
P2_LS_Get_Output_Status · 421
P2_LS_Watchdog_Init · 423
P2_LS_Watchdog_Reset · 425
P2_MIO_Get_Digout_Long · 34
P2_PWM_Enable · 215
P2_PWM_Get_Status · 216
P2_PWM_Init · 217
P2_PWM_Latch · 219
P2_PWM_Reset · 220
P2_PWM_Standby_Value · 221
P2_PWM_Write_Latch · 222
P2_PWM_Write_Latch_Block · 223

R

P2_Read_ADC · 42
P2_Read_ADC24 · 43
P2_Read_ADCF · 112
P2_Read_ADCF32 · 120
P2_Read_ADCF4 · 114
P2_Read_ADCF4_24B · 115
P2_Read_ADCF4_Packed · 118
P2_Read_ADCF8 · 116
P2_Read_ADCF8_24B · 117
P2_Read_ADCF8_Packed · 119
P2_Read_ADCF_24 · 113
P2_Read_ADCF_SConv · 121
P2_Read_ADCF_SConv24 · 122
P2_Read_ADCF_SConv32 · 123
P2_Read_ADC_SConv · 44
P2_Read_ADC_SConv24 · 45
P2_Read_Fifo · 268
P2_TC_Read_Latch8 · 240
P2_Read_Msg · 255
P2_Read_Msg_Con · 257
P2_RS485_Send · 272
P2_RS_Init · 269
P2_RS_Reset · 271
P2_RS_Set_LED · 273
P2_RTD_Channel_Config · 226
P2_RTD_Config · 228
P2_RTD_Convert · 229
P2_RTD_Read · 230
P2_RTD_Read8 · 231
P2_RTD_Start · 232

P2_RTD_Status · 234
P2_Run_Profibus · 295

S

P2_SENT_Clear_Serial_Message_Array · 356
P2_SENT_Command_Ready · 343
P2_SENT_Get_ChannelState · 344
P2_SENT_Get_ClockTick · 345
P2_SENT_Get_Fast_Channel1 · 348
P2_SENT_Get_Fast_Channel2 · 350
P2_SENT_Get_Fast_Channel_CRC_OK · 347
P2_SENT_Get_Msg_Counter · 342
P2_SENT_Get_PulseCount · 346
P2_SENT_Get_Serial_Message_Array · 354
P2_SENT_Get_Serial_Message_CRC_OK · 351
P2_SENT_Get_Serial_Message_Data · 353
P2_SENT_Get_Serial_Message_Id · 352
P2_SENT_Get_Version · 341
P2_SENT_Init · 340
P2_SENT_Set_ClockTick · 359
P2_SENT_Set_CRC_Implementation · 357
P2_SENT_Set_Detection · 358
P2_SENT_Set_PulseCount · 360
P2_SENT_Check_Latch · 363
P2_SENT_Config_Output · 370
P2_SENT_Config_Serial_Messages · 371
P2_SENT_Enable_Channel · 372
P2_SENT_Fifo_Clear · 381
P2_SENT_Fifo_Empty · 380
P2_SENT_Get_Latch_Data · 364
P2_SENT_Get_Output_Mode · 369
P2_SENT_Invert_Channel · 373
P2_SENT_Request_Latch · 362
P2_SENT_Set_Fast_Channel1 · 375
P2_SENT_Set_Fast_Channel2 · 376
P2_SENT_Set_Fifo · 382
P2_SENT_Set_Output_Mode · 368
P2_SENT_Set_Reserved_Bits · 374
P2_SENT_Set_Sensor_Type · 361
P2_SENT_Set_Serial_Message_Data · 379
P2_SENT_Set_Serial_Message_Pattern · 377
P2_Seq_Init · 47
P2_Seq_Read · 50
P2_Seq_Read24 · 51
P2_Seq_Read_Packed · 53
P2_Seq_Start · 54
P2_Seq_Wait · 55
P2_Set_Average_Filter · 99
P2_Set_CAN_Baudrate · 259
P2_Set_CAN_Reg · 263
P2_Set_Gain · 124
P2_Set_LED · 6
P2_Set_Mux · 56
P2_Set_RS · 274
P2_SE_Diff · 46
P2_ARINC_Config_Transmit · 308
P2_MIL_SMT_Message_Read · 299
P2_SPI_Config · 386
P2_SPI_Master_Config · 388
P2_SPI_Master_Get_Static_Input · 399
P2_SPI_Master_Get_Value32 · 397
P2_SPI_Master_Get_Value64 · 398
P2_SPI_Master_Set_Clk_Wait · 400
P2_SPI_Master_Set_Value32 · 392
P2_SPI_Master_Set_Value64 · 393
P2_SPI_Master_Start · 394
P2_SPI_Master_Status · 395
P2_SPI_Mode · 385
P2_SPI_Slave_Clear_Fifo · 409
P2_SPI_Slave_Config · 402
P2_SPI_Slave_InFifo_Full · 406
P2_SPI_Slave_InFifo_Read · 407
P2_SPI_Slave_OutFifo_Empty · 405
P2_SPI_Slave_OutFifo_Write · 403
P2_SSI_Mode · 205
P2_SSI_Read · 206
P2_SSI_Read2 · 208
P2_SSI_Set_Bits · 209
P2_SSI_Set_Clock · 210
P2_SSI_Set_Delay · 211
P2_SSI_Start · 212
P2_SSI_Status · 213
P2_Start_Conv · 57
P2_Start_ConvF · 125
P2_Start_DAC · 134
P2_Sync_All · 13
P2_Sync_Enable · 15
P2_Sync_Mode · 17
P2_Sync_Stat · 19

T

P2_TC_Latch · 235
P2_TC_Read_Latch · 236
P2_TC_Read_Latch4 · 238
P2_TC_Set_Rate · 242
P2_Transmit · 264

W

P2_Wait_EOC · 58
P2_Wait_EOCF · 126
P2_Wait_Mux · 59
P2_Write_DAC · 135
P2_Write_DAC32 · 140
P2_Write_DAC4 · 136
P2_Write_DAC4_Packed · 137
P2_Write_DAC8 · 138
P2_Write_DAC8_Packed · 139
P2_Write_Fifo · 275
P2_Write_Fifo_Full · 276

A.2 Instruction List sorted by Module Types

You find the instruction lists of the modules on these pages:

Module name	Page
AIn-16/18-8B Rev. E	A-4
AIn-16/18-C Rev. E	A-5
AIn-32/18 Rev. E	A-5
AIn-8/18 Rev. E	A-5
AIn-8/18-8B Rev. E	A-5
AIn-F-4/14 Rev. E	A-6
AIn-F-4/16 Rev. E	A-6
AIn-F-4/18 Rev. E	A-7
AIn-F-8/14 Rev. E	A-7
AIn-F-8/16 Rev. E	A-8
AIn-F-8/18 Rev. E	A-8
AOut-1/16 Rev. E	A-9
AOut-4/16 Rev. E	A-9
AOut-4/16-TiCo Rev. E	A-9
AOut-8/16 Rev. E	A-9
AOut-8/16-TiCo Rev. E	A-10
ARINC-429 Rev. E	A-10
CAN-2 Rev. E	A-10
CNT-D Rev. E	A-10
CNT-I Rev. E	A-11
CNT-T Rev. E	A-11
Comp-16 Rev. E	A-11
CPU-T11	A-11
DIO-32 Rev. E	A-11
DIO-32-TiCo Rev. E	A-12
DIO-32-TiCo2 Rev. E	A-12
EtherCAT-SL Rev. E	A-12
FlexRay-2 Rev. E	A-12
LIN-2 Rev. E	A-13
LS-2 Rev. E	A-13
MIL-1553 Rev. E	A-13
MIL-STD-1553 Rev. E	A-13
MIO-4 Rev. E	A-13
MIO-4-ET1 Rev. E	A-14
MIO-D12 Rev. E	A-14
OPT-16 Rev. E	A-15
OPT-32 Rev. E	A-15
Profi-SL Rev. E	A-15
PWM-16(-I) Rev. E	A-15
REL-16 Rev. E	A-15
RS422-4 Rev. E	A-15
RSxxx-2 Rev. E	A-16
RSxxx-4 Rev. E	A-16
RTD-8 Rev. E	A-16
SENT-4 Rev. E	A-16
SENT-4-Out Rev. E	A-16
SENT-6 Rev. E	A-17
SPI-2-D Rev. E	A-17
SPI-2-T Rev. E	A-18
TC-8-ISO Rev. E	A-18
TRA-16 Rev. E	A-18

AIn-16/18-8B Rev. E

A: P2_ADC · 37
P2_ADC24 · 38
P2_ADC_Read_Limit · 39
P2_ADC_Set_Limit · 41
C: P2_Check_LED · 5
E: P2_Event_Config · 9
P2_Event_Enable · 7
P2_Event_Read · 12
R: P2_ADC_SConv · 44
P2_ADC_SConv24 · 45
P2_Read_ADC · 42
P2_Read_ADC24 · 43
S: P2_Seq_Init · 47
P2_Seq_Read · 50
P2_Seq_Read24 · 51
P2_Seq_Read_Packed · 53
P2_Seq_Start · 54
P2_Seq_Wait · 55
P2_Set_LED · 6
P2_Set_Mux · 56
P2_SE_Diff · 46
P2_Start_Conv · 57
P2_Sync_All · 13
W: P2_Wait_EOC · 58
P2_Wait_Mux · 59

AIn-16/18-C Rev. E

A: P2_ADC · 37
P2_ADC24 · 38
P2_ADC_Read_Limit · 39
P2_ADC_Set_Limit · 41
C: P2_Check_LED · 5
E: P2_Event_Config · 9
P2_Event_Enable · 7
P2_Event_Read · 12
R: P2_Read_ADC · 42
P2_Read_ADC24 · 43
P2_Read_ADC_SConv · 44
P2_Read_ADC_SConv24 · 45
S: P2_Seq_Init · 47
P2_Seq_Read · 50
P2_Seq_Read24 · 51
P2_Seq_Read_Packed · 53
P2_Seq_Start · 54
P2_Seq_Wait · 55
P2_Set_LED · 6
P2_Set_Mux · 56
P2_SE_Diff · 46
P2_Start_Conv · 57
P2_Sync_All · 13
W: P2_Wait_EOC · 58
P2_Wait_Mux · 59

AIn-32/18 Rev. E

A: P2_ADC · 37
P2_ADC24 · 38
P2_ADC_Read_Limit · 39
P2_ADC_Set_Limit · 41
C: P2_Check_LED · 5
E: P2_Event_Config · 9
P2_Event_Enable · 7
P2_Event_Read · 12
R: P2_Read_ADC · 42
P2_Read_ADC24 · 43
P2_Read_ADC_SConv · 44
P2_Read_ADC_SConv24 · 45
S: P2_Seq_Init · 47
P2_Seq_Read · 50
P2_Seq_Read24 · 51
P2_Seq_Read_Packed · 53
P2_Seq_Start · 54
P2_Seq_Wait · 55
P2_Set_LED · 6
P2_Set_Mux · 56
P2_SE_Diff · 46
P2_Start_Conv · 57
P2_Sync_All · 13
W: P2_Wait_EOC · 58
P2_Wait_Mux · 59

AIn-8/18 Rev. E

A: P2_ADC · 37
P2_ADC24 · 38
P2_ADC_Read_Limit · 39
P2_ADC_Set_Limit · 41
C: P2_Check_LED · 5
E: P2_Event_Config · 9
P2_Event_Enable · 7
P2_Event_Read · 12
R: P2_Read_ADC · 42
P2_Read_ADC24 · 43
P2_Read_ADC_SConv · 44
P2_Read_ADC_SConv24 · 45
S: P2_Seq_Init · 47
P2_Seq_Read · 50
P2_Seq_Read24 · 51
P2_Seq_Read_Packed · 53
P2_Seq_Start · 54
P2_Seq_Wait · 55
P2_Set_LED · 6
P2_Set_Mux · 56
P2_Start_Conv · 57
P2_Sync_All · 13
W: P2_Wait_EOC · 58
P2_Wait_Mux · 59

AIn-8/18-8B Rev. E

A: P2_ADC · 37
P2_ADC24 · 38
P2_ADC_Read_Limit · 39
P2_ADC_Set_Limit · 41
C: P2_Check_LED · 5
E: P2_Event_Config · 9
P2_Event_Enable · 7
P2_Event_Read · 12
R: P2_ADC_SConv · 44
P2_ADC_SConv24 · 45
P2_Read_ADC · 42
P2_Read_ADC24 · 43
S: P2_Seq_Init · 47
P2_Seq_Read · 50
P2_Seq_Read24 · 51
P2_Seq_Read_Packed · 53
P2_Seq_Start · 54
P2_Seq_Wait · 55
P2_Set_LED · 6
P2_Set_Mux · 56
P2_Start_Conv · 57
P2_Sync_All · 13
W: P2_Wait_EOC · 58
P2_Wait_Mux · 59

AIn-F-4/14 Rev. E

A: P2_ADCF · 100
P2_ADCF_Read_Limit · 105
P2_ADCF_Set_Limit · 106
B: P2_Burst_CRead_Pos_Unpacked1 · 68
P2_Burst_CRead_Pos_Unpacked2 · 70
P2_Burst_CRead_Pos_Unpacked4 · 72
P2_Burst_CRead_Unpacked1 · 60
P2_Burst_CRead_Unpacked2 · 62
P2_Burst_CRead_Unpacked4 · 64
P2_Burst_Init · 76
P2_Burst_Read · 82
P2_Burst_Read_Index · 80
P2_Burst_Read_Unpacked1 · 84
P2_Burst_Read_Unpacked2 · 86
P2_Burst_Read_Unpacked4 · 88
P2_Burst_Reset · 92
P2_Burst_Start · 94
P2_Burst_Status · 95
P2_Burst_Stop · 97
C: P2_Check_LED · 5
E: P2_Event2_Config · 10
P2_Event_Config · 9
P2_Event_Enable · 7
P2_Event_Read · 12
R: P2_Read_ADCF · 112
P2_Read_ADCF32 · 120
P2_Read_ADCF4 · 114
P2_Read_ADCF4_Packed · 118
S: P2_Set_Average_Filter · 99
P2_Set_LED · 6
P2_Sync_All · 13

AIn-F-4/16 Rev. E

A: P2_ADCF · 100
P2_ADCF_Mode · 102
P2_ADCF_Read_Limit · 105
P2_ADCF_Read_Min_Max4 · 108
P2_ADCF_Read_Min_Max8 · 110
P2_ADCF_Reset_Min_Max · 107
P2_ADCF_Set_Limit · 106
B: P2_Burst_CRead_Pos_Unpacked1 · 68
P2_Burst_CRead_Pos_Unpacked2 · 70
P2_Burst_CRead_Pos_Unpacked4 · 72
P2_Burst_CRead_Unpacked1 · 60
P2_Burst_CRead_Unpacked2 · 62
P2_Burst_CRead_Unpacked4 · 64
P2_Burst_Init · 76
P2_Burst_Read · 82
P2_Burst_Read_Index · 80
P2_Burst_Read_Unpacked1 · 84
P2_Burst_Read_Unpacked2 · 86
P2_Burst_Read_Unpacked4 · 88
P2_Burst_Reset · 92
P2_Burst_Start · 94
P2_Burst_Status · 95
P2_Burst_Stop · 97
C: P2_Check_LED · 5
E: P2_Event2_Config · 10
P2_Event_Config · 9
P2_Event_Enable · 7
P2_Event_Read · 12
R: P2_Read_ADCF · 112
P2_Read_ADCF32 · 120
P2_Read_ADCF4 · 114
P2_Read_ADCF4_Packed · 118
P2_Read_ADCF_SConv · 121
P2_Read_ADCF_SConv32 · 123
S: P2_Set_Average_Filter · 99
P2_Set_LED · 6
P2_Start_ConvF · 125
P2_Sync_All · 13
P2_Sync_Enable · 15
P2_Sync_Mode · 17
P2_Sync_Stat · 19
W: P2_Wait_EOCF · 126

AIn-F-4/18 Rev. E

A: P2_ADCF · 100
P2_ADCF24 · 101
P2_ADCF_Mode · 102
P2_ADCF_Read_Limit · 105
P2_ADCF_Set_Limit · 106
C: P2_Check_LED · 5
E: P2_Event2_Config · 10
P2_Event_Config · 9
P2_Event_Enable · 7
P2_Event_Read · 12
R: P2_Read_ADCF · 112
P2_Read_ADCF24 · 113
P2_Read_ADCF32 · 120
P2_Read_ADCF4 · 114
P2_Read_ADCF4_24B · 115
P2_Read_ADCF4_Packed · 118
P2_Read_ADCF_SConv · 121
P2_Read_ADCF_SConv24 · 122
P2_Read_ADCF_SConv32 · 123
S: P2_Set_LED · 6
P2_Start_ConvF · 125
P2_Sync_All · 13
P2_Sync_Enable · 15
P2_Sync_Mode · 17
P2_Sync_Stat · 19
W: P2_Wait_EOCF · 126

AIn-F-8/14 Rev. E

A: P2_ADCF · 100
P2_ADCF_Read_Limit · 105
P2_ADCF_Set_Limit · 106
B: P2_Burst_CRead_Pos_Unpacked1 · 68
P2_Burst_CRead_Pos_Unpacked2 · 70
P2_Burst_CRead_Pos_Unpacked4 · 72
P2_Burst_CRead_Pos_Unpacked8 · 74
P2_Burst_CRead_Unpacked1 · 60
P2_Burst_CRead_Unpacked2 · 62
P2_Burst_CRead_Unpacked4 · 64
P2_Burst_CRead_Unpacked8 · 66
P2_Burst_Init · 76
P2_Burst_Read · 82
P2_Burst_Read_Index · 80
P2_Burst_Read_Unpacked1 · 84
P2_Burst_Read_Unpacked2 · 86
P2_Burst_Read_Unpacked4 · 88
P2_Burst_Read_Unpacked8 · 90
P2_Burst_Reset · 92
P2_Burst_Start · 94
P2_Burst_Status · 95
P2_Burst_Stop · 97
C: P2_Check_LED · 5
E: P2_Event2_Config · 10
P2_Event_Config · 9
P2_Event_Enable · 7
P2_Event_Read · 12
R: P2_Read_ADCF · 112
P2_Read_ADCF32 · 120
P2_Read_ADCF4 · 114
P2_Read_ADCF4_Packed · 118
P2_Read_ADCF8 · 116
P2_Read_ADCF8_Packed · 119
S: P2_Set_Average_Filter · 99
P2_Set_LED · 6
P2_Sync_All · 13

AIn-F-8/16 Rev. E

A: P2_ADCF · 100
P2_ADCF_Mode · 102
P2_ADCF_Read_Limit · 105
P2_ADCF_Read_Min_Max4 · 108
P2_ADCF_Read_Min_Max8 · 110
P2_ADCF_Reset_Min_Max · 107
P2_ADCF_Set_Limit · 106
B: P2_Burst_CRead_Pos_Unpacked1 · 68
P2_Burst_CRead_Pos_Unpacked2 · 70
P2_Burst_CRead_Pos_Unpacked4 · 72
P2_Burst_CRead_Pos_Unpacked8 · 74
P2_Burst_CRead_Unpacked1 · 60
P2_Burst_CRead_Unpacked2 · 62
P2_Burst_CRead_Unpacked4 · 64
P2_Burst_CRead_Unpacked8 · 66
P2_Burst_Init · 76
P2_Burst_Read · 82
P2_Burst_Read_Index · 80
P2_Burst_Read_Unpacked1 · 84
P2_Burst_Read_Unpacked2 · 86
P2_Burst_Read_Unpacked4 · 88
P2_Burst_Read_Unpacked8 · 90
P2_Burst_Reset · 92
P2_Burst_Start · 94
P2_Burst_Status · 95
P2_Burst_Stop · 97
C: P2_Check_LED · 5
E: P2_Event2_Config · 10
P2_Event_Config · 9
P2_Event_Enable · 7
P2_Event_Read · 12
R: P2_Read_ADCF · 112
P2_Read_ADCF32 · 120
P2_Read_ADCF4 · 114
P2_Read_ADCF4_Packed · 118
P2_Read_ADCF8 · 116
P2_Read_ADCF8_Packed · 119
P2_Read_ADCF_SConv · 121
P2_Read_ADCF_SConv32 · 123
S: P2_Set_Average_Filter · 99
P2_Set_LED · 6
P2_Start_ConvF · 125
P2_Sync_All · 13
P2_Sync_Enable · 15
P2_Sync_Mode · 17
P2_Sync_Stat · 19
W: P2_Wait_EOCF · 126

AIn-F-8/18 Rev. E

A: P2_ADCF · 100
P2_ADCF24 · 101
P2_ADCF_Mode · 102
P2_ADCF_Read_Limit · 105
P2_ADCF_Set_Limit · 106
C: P2_Check_LED · 5
E: P2_Event2_Config · 10
P2_Event_Config · 9
P2_Event_Enable · 7
P2_Event_Read · 12
R: P2_Read_ADCF · 112
P2_Read_ADCF32 · 120
P2_Read_ADCF4 · 114
P2_Read_ADCF4_24B · 115
P2_Read_ADCF4_Packed · 118
P2_Read_ADCF8 · 116
P2_Read_ADCF8_24B · 117
P2_Read_ADCF8_Packed · 119
P2_Read_ADCF_24 · 113
P2_Read_ADCF_SConv · 121
P2_Read_ADCF_SConv24 · 122
P2_Read_ADCF_SConv32 · 123
S: P2_Set_LED · 6
P2_Start_ConvF · 125
P2_Sync_All · 13
P2_Sync_Enable · 15
P2_Sync_Mode · 17
P2_Sync_Stat · 19
W: P2_Wait_EOCF · 126

AOut-1/16 Rev. E

C: P2_Check_LED · 5
D: P2_DAC · 128
P2_DAC1_DIO · 141
P2_DAC_Ramp_Buffer_Free · 146
P2_DAC_Ramp_Status · 144
P2_DAC_Ramp_Stop · 147
P2_DAC_Ramp_Write · 142
P2_Digin_Edge · 158
P2_Digin_Long · 169
P2_Digout · 170
P2_Digout_Bits · 171
P2_Digout_Fifo_Clear · 173
P2_Digout_Fifo_Empty · 174
P2_Digout_Fifo_Enable · 175
P2_Digout_Fifo_Read_Timer · 176
P2_Digout_Fifo_Start · 177
P2_Digout_Fifo_Write · 178
P2_Digout_Long · 180
P2_Digout_Reset · 181
P2_Digout_Set · 182
P2_Dig_Fifo_Mode · 153
P2_Dig_Latch · 155
P2_Dig_Read_Latch · 156
P2_Dig_Write_Latch · 157
E: P2_Event_Config · 9
P2_Event_Enable · 7
P2_Event_Read · 12
G: P2_Get_Digout_Long · 186
S: P2_Set_LED · 6
P2_Start_DAC · 134
P2_Sync_All · 13
W: P2_Write_DAC · 135

AOut-4/16 Rev. E

C: P2_Check_LED · 5
D: P2_DAC · 128
P2_DAC4 · 129
P2_DAC4_Packed · 130
E: P2_Event_Config · 9
P2_Event_Enable · 7
P2_Event_Read · 12
S: P2_Set_LED · 6
P2_Start_DAC · 134
P2_Sync_All · 13
P2_Sync_Enable · 15
P2_Sync_Stat · 19
W: P2_Write_DAC · 135
P2_Write_DAC32 · 140
P2_Write_DAC4 · 136
P2_Write_DAC4_Packed · 137

AOut-4/16-TiCo Rev. E

C: P2_Check_LED · 5
D: P2_DAC · 128
P2_DAC4 · 129
P2_DAC4_Packed · 130
E: P2_Event_Config · 9
P2_Event_Enable · 7
P2_Event_Read · 12
S: P2_Set_LED · 6
P2_Start_DAC · 134
P2_Sync_All · 13
P2_Sync_Enable · 15
P2_Sync_Stat · 19
W: P2_Write_DAC · 135
P2_Write_DAC32 · 140
P2_Write_DAC4 · 136
P2_Write_DAC4_Packed · 137

AOut-8/16 Rev. E

C: P2_Check_LED · 5
D: P2_DAC · 128
P2_DAC4 · 129
P2_DAC4_Packed · 130
P2_DAC8 · 132
P2_DAC8_Packed · 133
E: P2_Event_Config · 9
P2_Event_Enable · 7
P2_Event_Read · 12
S: P2_Set_LED · 6
P2_Start_DAC · 134
P2_Sync_All · 13
P2_Sync_Enable · 15
P2_Sync_Stat · 19
W: P2_Write_DAC · 135
P2_Write_DAC32 · 140
P2_Write_DAC4 · 136
P2_Write_DAC4_Packed · 137
P2_Write_DAC8 · 138
P2_Write_DAC8_Packed · 139

AOut-8/16-TiCo Rev. E

C: P2_Check_LED · 5
D: P2_DAC · 128
P2_DAC4 · 129
P2_DAC4_Packed · 130
P2_DAC8 · 132
P2_DAC8_Packed · 133
E: P2_Event_Config · 9
P2_Event_Enable · 7
P2_Event_Read · 12
S: P2_Set_LED · 6
P2_Start_DAC · 134
P2_Sync_All · 13
P2_Sync_Enable · 15
P2_Sync_Stat · 19
W: P2_Write_DAC · 135
P2_Write_DAC32 · 140
P2_Write_DAC4 · 136
P2_Write_DAC4_Packed · 137
P2_Write_DAC8 · 138
P2_Write_DAC8_Packed · 139

ARINC-429 Rev. E

A: ARINC_Create_Value32 · 315
ARINC_Split_Value32 · 319
P2_ARINC_Config_Receive · 310
P2_ARINC_Config_Transmit · 308
P2_ARINC_Read_Receive_Fifo · 318
P2_ARINC_Receive_Fifo_Empty · 317
P2_ARINC_Reset · 307
P2_ARINC_Set_Labels · 320
P2_ARINC_Transmit_Enable · 316
P2_ARINC_Transmit_Fifo_Empty · 313
P2_ARINC_Transmit_Fifo_Full · 312
P2_ARINC_Write_Transmit_Fifo · 314
C: P2_Check_LED · 5
S: P2_Set_LED · 6

CAN-2 Rev. E

C: CAN_Msg · 244
P2_CAN_Set_LED · 248
P2_Check_LED · 5
E: P2_CAN_Interrupt_Source · 246
P2_En_Interrupt · 249
P2_En_Receive · 251
P2_En_Transmit · 252
P2_Event_Config · 9
P2_Event_Enable · 7
P2_Event_Read · 12
G: P2_Get_CAN_Reg · 253
I: P2_Init_CAN · 254
R: P2_Read_Msg · 255
P2_Read_Msg_Con · 257
S: P2_Set_CAN_Baudrate · 259
P2_Set_CAN_Reg · 263
P2_Set_LED · 6
T: P2_Transmit · 264

CNT-D Rev. E

C: P2_Check_LED · 5
P2_Cnt_Clear · 188
P2_Cnt_Enable · 189
P2_Cnt_Get_PW · 191
P2_Cnt_Get_PW_HL · 192
P2_Cnt_Get_Status · 190
P2_Cnt_Latch · 193
P2_Cnt_Mode · 194
P2_Cnt_PW_Enable · 196
P2_Cnt_PW_Latch · 197
P2_Cnt_Read · 198
P2_Cnt_Read4 · 199
P2_Cnt_Read_Int_Register · 200
P2_Cnt_Read_Latch · 201
P2_Cnt_Read_Latch4 · 202
P2_Cnt_Sync_Latch · 203
E: P2_Event_Config · 9
P2_Event_Enable · 7
P2_Event_Read · 12
S: P2_Set_LED · 6
P2_SSI_Mode · 205
P2_SSI_Read · 206
P2_SSI_Read2 · 208
P2_SSI_Set_Bits · 209
P2_SSI_Set_Clock · 210
P2_SSI_Set_Delay · 211
P2_SSI_Start · 212
P2_SSI_Status · 213
P2_Sync_All · 13

CNT-I Rev. E

C: P2_Check_LED · 5
P2_Cnt_Clear · 188
P2_Cnt_Enable · 189
P2_Cnt_Get_PW · 191
P2_Cnt_Get_PW_HL · 192
P2_Cnt_Get_Status · 190
P2_Cnt_Latch · 193
P2_Cnt_Mode · 194
P2_Cnt_PW_Enable · 196
P2_Cnt_PW_Latch · 197
P2_Cnt_Read · 198
P2_Cnt_Read4 · 199
P2_Cnt_Read_Int_Register · 200
P2_Cnt_Read_Latch · 201
P2_Cnt_Read_Latch4 · 202
P2_Cnt_Sync_Latch · 203
E: P2_Event_Config · 9
P2_Event_Enable · 7
P2_Event_Read · 12
S: P2_Set_LED · 6
P2_Sync_All · 13

CNT-T Rev. E

C: P2_Check_LED · 5
P2_Cnt_Clear · 188
P2_Cnt_Enable · 189
P2_Cnt_Get_PW · 191
P2_Cnt_Get_PW_HL · 192
P2_Cnt_Get_Status · 190
P2_Cnt_Latch · 193
P2_Cnt_Mode · 194
P2_Cnt_PW_Enable · 196
P2_Cnt_PW_Latch · 197
P2_Cnt_Read · 198
P2_Cnt_Read4 · 199
P2_Cnt_Read_Int_Register · 200
P2_Cnt_Read_Latch · 201
P2_Cnt_Read_Latch4 · 202
P2_Cnt_Sync_Latch · 203
E: P2_Event_Config · 9
P2_Event_Enable · 7
P2_Event_Read · 12
S: P2_Set_LED · 6
P2_Sync_All · 13

Comp-16 Rev. E

C: P2_Comp_Filter_Init · 151
P2_Comp_Init · 149
P2_Comp_Set · 152

CPU-T11

C: CPU_Digin · 22
CPU_Digout · 23
CPU_Dig_IO_Config · 24
CPU_Event_Config · 25
P2_Check_LED · 5
S: P2_Set_LED · 6

DIO-32 Rev. E

C: P2_Check_LED · 5
D: P2_Digin_Edge · 158
P2_Digin_Fifo_Clear · 159
P2_Digin_Fifo_Enable · 160
P2_Digin_Fifo_Full · 162
P2_Digin_Fifo_Read · 163
P2_Digin_Fifo_Read_Fast · 165
P2_Digin_Fifo_Read_Timer · 167
P2_Digin_Long · 169
P2_Digout · 170
P2_Digout_Bits · 171
P2_Digout_Long · 180
P2_Digout_Reset · 181
P2_Digout_Set · 182
P2_DigProg · 183
P2_Dig_Latch · 155
P2_Dig_Read_Latch · 156
P2_Dig_Write_Latch · 157
E: P2_Event_Config · 9
P2_Event_Enable · 7
P2_Event_Read · 12
G: P2_Get_Digout_Long · 186
S: P2_Set_LED · 6
P2_Sync_All · 13, · 15

DIO-32-TiCo Rev. E

C: P2_Check_LED · 5
D: P2_Digin_Edge · 158
P2_Digin_Fifo_Clear · 159
P2_Digin_Fifo_Enable · 160
P2_Digin_Fifo_Full · 162
P2_Digin_Fifo_Read · 163
P2_Digin_Fifo_Read_Fast · 165
P2_Digin_Fifo_Read_Timer · 167
P2_Digin_Filter_Init · 168
P2_Digin_Long · 169
P2_Digout · 170
P2_Digout_Bits · 171
P2_Digout_Fifo_Clear (Rev. E03) · 173
P2_Digout_Fifo_Empty (Rev. E03) · 174
P2_Digout_Fifo_Enable (Rev. E03) · 175
P2_Digout_Fifo_Read_Timer (Rev. E03) · 176
P2_Digout_Fifo_Start (Rev. E03) · 177
P2_Digout_Fifo_Write (Rev. E03) · 178
P2_Digout_Long · 180
P2_Digout_Reset · 181
P2_Digout_Set · 182
P2_DigProg · 183
P2_Dig_Fifo_Mode (Rev. E03) · 153
P2_Dig_Latch · 155
P2_Dig_Read_Latch · 156
P2_Dig_Write_Latch · 157
E: P2_Event_Config · 9
P2_Event_Enable · 7
P2_Event_Read · 12
G: P2_Get_Digout_Long · 186
S: P2_Set_LED · 6
P2_Sync_All · 13, · 15

DIO-32-TiCo2 Rev. E

C: P2_Check_LED · 5
D: P2_Digin_Edge · 158
P2_Digin_Fifo_Clear · 159
P2_Digin_Fifo_Enable · 160
P2_Digin_Fifo_Full · 162
P2_Digin_Fifo_Read · 163
P2_Digin_Fifo_Read_Fast · 165
P2_Digin_Fifo_Read_Timer · 167
P2_Digin_Long · 169
P2_Digout · 170
P2_Digout_Bits · 171
P2_Digout_Fifo_Clear · 173
P2_Digout_Fifo_Empty · 174
P2_Digout_Fifo_Enable · 175
P2_Digout_Fifo_Read_Timer · 176
P2_Digout_Fifo_Start · 177
P2_Digout_Fifo_Write · 178
P2_Digout_Long · 180
P2_Digout_Reset · 181
P2_Digout_Set · 182
P2_DigProg · 183
P2_Digprog_Set_IO_Level · 185
P2_Dig_Fifo_Mode · 153
P2_Dig_Latch · 155
P2_Dig_Read_Latch · 156
P2_Dig_Write_Latch · 157
E: P2_Event_Config · 9
P2_Event_Enable · 7
P2_Event_Read · 12
G: P2_Get_Digout_Long · 186
S: P2_Set_LED · 6
P2_Sync_All · 13, · 15

EtherCAT-SL Rev. E

C: P2_Check_LED · 5
E: P2_ECAT_Get_State · 323
P2_ECAT_Get_Version · 322
P2_ECAT_Read_Data_16F · 329
P2_ECAT_Read_Data_16L · 327
P2_ECAT_Set_Mode · 326
P2_ECAT_Write_Data_16F · 330
P2_ECAT_Write_Data_16L · 328
I: P2_ECAT_Init · 324
S: P2_Set_LED · 6

FlexRay-2 Rev. E

C: P2_Check_LED · 5
F: P2_FlexRay_Get_Version · 332
P2_FlexRay_Init · 333
P2_FlexRay_Read_Word · 335
P2_FlexRay_Reset · 336
P2_FlexRay_Set_LED · 337
P2_FlexRay_Write_Word · 338
S: P2_Set_LED · 6

LIN-2 Rev. E

C: P2_Check_LED · 5
L: P2_LIN_Get_Version · 285
P2_LIN_Init · 280
P2_LIN_Init_Apply · 283
P2_LIN_Init_Write · 282
P2_LIN_Msg_Transmit · 290
P2_LIN_Msg_Write · 288
P2_LIN_Read_Dat · 286
P2_LIN_Reset · 284
P2_LIN_Set_LED · 291
S: P2_Set_LED · 6

LS-2 Rev. E

C: P2_Check_LED · 5
L: P2_LS_Digout_Long · 417, · 419, · 421, · 425
P2_LS_DigProg · 413
P2_LS_Dig_IO · 415
P2_LS_DIO_Init · 411
P2_LS_Watchdog_Init · 423
S: P2_Set_LED · 6

MIL-1553 Rev. E

M: P2_MIL_Get_Register · 305
P2_MIL_Reset · 297
P2_MIL_Set_LED · 303
P2_MIL_Set_Register · 304
P2_MIL_SMT_Init · 298
P2_MIL_SMT_Message_Read · 299
P2_MIL_SMT_Set_All_Filters · 301
P2_MIL_SMT_Set_Filter · 302

MIL-STD-1553 Rev. E

C: P2_Check_LED · 5
S: P2_Set_LED · 6

MIO-4 Rev. E

A: P2_ADC · 37
P2_ADC24 · 38
P2_ADC_Read_Limit · 39
P2_ADC_Set_Limit · 41
C: P2_Check_LED · 5
D: P2_DAC · 128
P2_DAC4 · 129
P2_DAC4_Packed · 130
E: P2_Event_Config · 9
P2_Event_Enable · 7
P2_Event_Read · 12
M: P2_MIO_Digin_Long · 30
P2_MIO_Digout · 31
P2_MIO_Digout_Long · 32
P2_MIO_DigProg · 33
P2_MIO_Dig_Latch · 27
P2_MIO_Dig_Read_Latch · 28
P2_MIO_Dig_Write_Latch · 29
P2_MIO_Get_Digout_Long · 34
R: P2_Read_ADC · 42
P2_Read_ADC24 · 43
P2_Read_ADC_SConv · 44
P2_Read_ADC_SConv24 · 45
S: P2_Seq_Init · 47
P2_Seq_Read · 50
P2_Seq_Read24 · 51
P2_Seq_Read_Packed · 53
P2_Seq_Start · 54
P2_Seq_Wait · 55
P2_Set_LED · 6
P2_Set_Mux · 56
P2_SE_Diff · 46
P2_Start_Conv · 57
P2_Start_DAC · 134
P2_Sync_All · 13, · 15
P2_Sync_Stat · 19
W: P2_Wait_EOC · 58
P2_Wait_Mux · 59
P2_Write_DAC · 135
P2_Write_DAC32 · 140
P2_Write_DAC4 · 136
P2_Write_DAC4_Packed · 137

MIO-4-ET1 Rev. E

A: P2_ADC · 37
P2_ADC24 · 38
P2_ADC_Read_Limit · 39
P2_ADC_Set_Limit · 41
C: P2_Check_LED · 5
P2_Cnt_Clear · 188
P2_Cnt_Enable · 189
P2_Cnt_Get_PW · 191
P2_Cnt_Get_PW_HL · 192
P2_Cnt_Get_Status · 190
P2_Cnt_Latch · 193
P2_Cnt_Mode · 194
P2_Cnt_PW_Enable · 196
P2_Cnt_PW_Latch · 197
P2_Cnt_Read · 198
P2_Cnt_Read_Int_Register · 200
P2_Cnt_Read_Latch · 201
P2_Cnt_Sync_Latch · 203
D: P2_DAC · 128
P2_DAC4 · 129
P2_DAC4_Packed · 130
E: P2_Event_Config · 9
P2_Event_Enable · 7
P2_Event_Read · 12
M: P2_MIO_Digin_Long · 30
P2_MIO_Digout · 31
P2_MIO_Digout_Long · 32
P2_MIO_DigProg · 33
P2_MIO_Dig_Latch · 27
P2_MIO_Dig_Read_Latch · 28
P2_MIO_Dig_Write_Latch · 29
P2_MIO_Get_Digout_Long · 34
R: P2_Read_ADC · 42
P2_Read_ADC24 · 43
P2_Read_ADC_SConv · 44
P2_Read_ADC_SConv24 · 45
S: P2_Seq_Init · 47
P2_Seq_Read · 50
P2_Seq_Read24 · 51
P2_Seq_Read_Packed · 53
P2_Seq_Start · 54
P2_Seq_Wait · 55
P2_Set_LED · 6
P2_Set_Mux · 56
P2_SE_Diff · 46
P2_SSI_Mode · 205
P2_SSI_Read · 206
P2_SSI_Set_Bits · 209
P2_SSI_Set_Clock · 210
P2_SSI_Set_Delay · 211
P2_SSI_Start · 212
P2_SSI_Status · 213
P2_Start_Conv · 57
P2_Start_DAC · 134
P2_Sync_All · 13, · 15
P2_Sync_Stat · 19
W: P2_Wait_EOC · 58
P2_Wait_Mux · 59
P2_Write_DAC · 135
P2_Write_DAC32 · 140
P2_Write_DAC4 · 136
P2_Write_DAC4_Packed · 137

MIO-D12 Rev. E

C: P2_Check_LED · 5
P2_Cnt_Clear · 188
P2_Cnt_Enable · 189
P2_Cnt_Get_PW · 191
P2_Cnt_Get_PW_HL · 192
P2_Cnt_Get_Status · 190
P2_Cnt_Latch · 193
P2_Cnt_Mode · 194
P2_Cnt_PW_Enable · 196
P2_Cnt_PW_Latch · 197
P2_Cnt_Read · 198
P2_Cnt_Read_Int_Register · 200
P2_Cnt_Read_Latch · 201
P2_Cnt_Sync_Latch · 203
D: P2_Digin_Edge · 158
P2_Digin_Fifo_Clear · 159
P2_Digin_Fifo_Enable · 160
P2_Digin_Fifo_Full · 162
P2_Digin_Fifo_Read · 163
P2_Digin_Fifo_Read_Fast · 165
P2_Digin_Fifo_Read_Timer · 167
P2_Digout_Fifo_Clear · 173
P2_Digout_Fifo_Empty · 174
P2_Digout_Fifo_Enable · 175
P2_Digout_Fifo_Read_Timer · 176
P2_Digout_Fifo_Start · 177
P2_Digout_Fifo_Write · 178
P2_Dig_Fifo_Mode · 153
E: P2_Event_Config · 9
P2_Event_Enable · 7
P2_Event_Read · 12
M: P2_MIO_Digin_Long · 30
P2_MIO_Digout · 31
P2_MIO_Digout_Long · 32
P2_MIO_DigProg · 33
P2_MIO_Dig_Latch · 27
P2_MIO_Dig_Read_Latch · 28
P2_MIO_Dig_Write_Latch · 29
P2_MIO_Get_Digout_Long · 34
S: P2_Set_LED · 6
P2_SSI_Mode · 205
P2_SSI_Read · 206
P2_SSI_Set_Bits · 209
P2_SSI_Set_Clock · 210
P2_SSI_Set_Delay · 211
P2_SSI_Start · 212
P2_SSI_Status · 213
P2_Sync_All · 13, · 15
P2_Sync_Stat · 19

OPT-16 Rev. E

C: P2_Check_LED · 5
D: P2_Digin_Edge · 158
P2_Digin_Fifo_Clear · 159
P2_Digin_Fifo_Enable · 160
P2_Digin_Fifo_Full · 162
P2_Digin_Fifo_Read · 163
P2_Digin_Fifo_Read_Fast · 165
P2_Digin_Fifo_Read_Timer · 167
P2_Digin_Long · 169
P2_Dig_Latch · 155
P2_Dig_Read_Latch · 156
E: P2_Event_Config · 9
P2_Event_Enable · 7
P2_Event_Read · 12
S: P2_Set_LED · 6
P2_Sync_All · 13, · 15

OPT-32 Rev. E

C: P2_Check_LED · 5
D: P2_Digin_Edge · 158
P2_Digin_Fifo_Clear · 159
P2_Digin_Fifo_Enable · 160
P2_Digin_Fifo_Full · 162
P2_Digin_Fifo_Read · 163
P2_Digin_Fifo_Read_Fast · 165
P2_Digin_Fifo_Read_Timer · 167
P2_Digin_Long · 169
P2_Dig_Latch · 155
P2_Dig_Read_Latch · 156
E: P2_Event_Config · 9
P2_Event_Enable · 7
P2_Event_Read · 12
S: P2_Set_LED · 6
P2_Sync_All · 13, · 15

Profi-SL Rev. E

C: P2_Check_LED · 5
I: P2_Init_Profibus · 293
R: P2_Run_Profibus · 295
S: P2_Set_LED · 6

PWM-16(-I) Rev. E

C: P2_Check_LED · 5
E: P2_Event_Config · 9
P2_Event_Enable · 7
P2_Event_Read · 12
P: P2_PWM_Enable · 215
P2_PWM_Get_Status · 216
P2_PWM_Init · 217
P2_PWM_Latch · 219
P2_PWM_Reset · 220
P2_PWM_Standby_Value · 221
P2_PWM_Write_Latch · 222
P2_PWM_Write_Latch_Block · 223
S: P2_Set_LED · 6
P2_Sync_All · 13
P2_Sync_Enable · 15, · 19

REL-16 Rev. E

C: P2_Check_LED · 5
D: P2_Digout · 170
P2_Digout_Bits · 171
P2_Digout_Long · 180
P2_Digout_Reset · 181
P2_Digout_Set · 182
P2_Dig_Latch · 155
P2_Dig_Write_Latch · 157
E: P2_Event_Config · 9
P2_Event_Enable · 7
P2_Event_Read · 12
G: P2_Get_Digout_Long · 186
S: P2_Set_LED · 6
P2_Sync_All · 13, · 15

RS422-4 Rev. E

C: P2_Check_LED · 5
P2_Check_Shift_Reg · 266
E: P2_Event_Config · 9
P2_Event_Enable · 7
P2_Event_Read · 12
G: P2_Get_RS · 267
R: P2_Read_Fifo · 268
P2_RS485_Send · 272
P2_RS_Init · 269
P2_RS_Reset · 271
P2_RS_Set_LED · 273
S: P2_Set_LED · 6
P2_Set_RS · 274
W: P2_Write_Fifo · 275
P2_Write_Fifo_Full · 276

RSxxx-2 Rev. E

C: P2_Check_LED · 5
P2_Check_Shift_Reg · 266
E: P2_Event_Config · 9
P2_Event_Enable · 7
P2_Event_Read · 12
G: P2_Get_RS · 267
R: P2_Read_Fifo · 268
P2_RS485_Send · 272
P2_RS_Init · 269
P2_RS_Reset · 271
P2_RS_Set_LED · 273
S: P2_Set_LED · 6
P2_Set_RS · 274
W: P2_Write_Fifo · 275
P2_Write_Fifo_Full · 276

RSxxx-4 Rev. E

C: P2_Check_LED · 5
P2_Check_Shift_Reg · 266
E: P2_Event_Config · 9
P2_Event_Enable · 7
P2_Event_Read · 12
G: P2_Get_RS · 267
R: P2_Read_Fifo · 268
P2_RS485_Send · 272
P2_RS_Init · 269
P2_RS_Reset · 271
P2_RS_Set_LED · 273
S: P2_Set_LED · 6
P2_Set_RS · 274
W: P2_Write_Fifo · 275
P2_Write_Fifo_Full · 276

RTD-8 Rev. E

C: P2_Check_LED · 5
R: P2_RTD_Channel_Config · 226
P2_RTD_Config · 228
P2_RTD_Convert · 229
P2_RTD_Read · 230
P2_RTD_Read8 · 231
P2_RTD_Start · 232
P2_RTD_Status · 234
S: P2_Set_LED · 6

SENT-4 Rev. E

C: P2_Check_LED · 5
S: P2_SENT_Check_Latch · 363
P2_SENT_Clear_Serial_Message_Array · 356
P2_SENT_Command_Ready · 343
P2_SENT_Get_ChannelState · 344
P2_SENT_Get_ClockTick · 345
P2_SENT_Get_Fast_Channel1 · 348
P2_SENT_Get_Fast_Channel2 · 350
P2_SENT_Get_Fast_Channel_CRC_OK · 347
P2_SENT_Get_Latch_Data · 364
P2_SENT_Get_Msg_Counter · 342
P2_SENT_Get_PulseCount · 346
P2_SENT_Get_Serial_Message_Array · 354
P2_SENT_Get_Serial_Message_CRC_OK · 351
P2_SENT_Get_Serial_Message_Data · 353
P2_SENT_Get_Serial_Message_Id · 352
P2_SENT_Get_Version · 341
P2_SENT_Init · 340
P2_SENT_Request_Latch · 362
P2_SENT_Set_ClockTick · 359
P2_SENT_Set_CRC_Implementation · 357
P2_SENT_Set_Detection · 358
P2_SENT_Set_PulseCount · 360
P2_SENT_Set_Sensor_Type · 361
P2_Set_LED · 6

SENT-4-Out Rev. E

C: P2_Check_LED · 5
S: P2_SENT_Command_Ready · 343
P2_SENT_Config_Output · 370
P2_SENT_Config_Serial_Messages · 371
P2_SENT_Enable_Channel · 372
P2_SENT_Fifo_Clear · 381
P2_SENT_Fifo_Empty · 380
P2_SENT_Get_Msg_Counter · 342
P2_SENT_Get_Output_Mode · 369
P2_SENT_Get_Version · 341
P2_SENT_Init · 340
P2_SENT_Invert_Channel · 373
P2_SENT_Set_Fast_Channel1 · 375
P2_SENT_Set_Fast_Channel2 · 376
P2_SENT_Set_Fifo · 382
P2_SENT_Set_Output_Mode · 368
P2_SENT_Set_Reserved_Bits · 374
P2_SENT_Set_Serial_Message_Data · 379
P2_SENT_Set_Serial_Message_Pattern · 377
P2_Set_LED · 6

SENT-6 Rev. E

C: P2_Check_LED · 5
S: P2_SENT_Check_Latch · 363
P2_SENT_Clear_Serial_Message_Array · 356
P2_SENT_Command_Ready · 343
P2_SENT_Get_ChannelState · 344
P2_SENT_Get_ClockTick · 345
P2_SENT_Get_Fast_Channel1 · 348
P2_SENT_Get_Fast_Channel2 · 350
P2_SENT_Get_Fast_Channel_CRC_OK · 347
P2_SENT_Get_Latch_Data · 364
P2_SENT_Get_Msg_Counter · 342
P2_SENT_Get_PulseCount · 346
P2_SENT_Get_Serial_Message_Array · 354
P2_SENT_Get_Serial_Message_CRC_OK · 351
P2_SENT_Get_Serial_Message_Data · 353
P2_SENT_Get_Serial_Message_Id · 352
P2_SENT_Get_Version · 341
P2_SENT_Init · 340
P2_SENT_Request_Latch · 362
P2_SENT_Set_ClockTick · 359
P2_SENT_Set_CRC_Implementation · 357
P2_SENT_Set_Detection · 358
P2_SENT_Set_PulseCount · 360
P2_SENT_Set_Sensor_Type · 361
P2_Set_LED · 6

SPI-2-D Rev. E

C: P2_Check_LED · 5
D: P2_Digin_Edge · 158
P2_Digin_Fifo_Clear · 159
P2_Digin_Fifo_Enable · 160
P2_Digin_Fifo_Full · 162
P2_Digin_Fifo_Read · 163
P2_Digin_Fifo_Read_Fast · 165
P2_Digin_Fifo_Read_Timer · 167
P2_Digin_Long · 169
P2_Digout · 170
P2_Digout_Bits · 171
P2_Digout_Fifo_Clear · 173
P2_Digout_Fifo_Empty · 174
P2_Digout_Fifo_Enable · 175
P2_Digout_Fifo_Read_Timer · 176
P2_Digout_Fifo_Start · 177
P2_Digout_Fifo_Write · 178
P2_Digout_Long · 180
P2_Digout_Reset · 181
P2_Digout_Set · 182
P2_DigProg · 183
P2_DigProg_Bits · 184
P2_Dig_Fifo_Mode · 153
P2_Dig_Latch · 155
P2_Dig_Read_Latch · 156
P2_Dig_Write_Latch · 157
E: P2_Event_Config · 9
P2_Event_Enable · 7
P2_Event_Read · 12
G: P2_Get_Digout_Long · 186
S: P2_Set_LED · 6
P2_SPI_Config · 386
P2_SPI_Master_Config · 388
P2_SPI_Master_Get_Static_Input · 399
P2_SPI_Master_Get_Value32 · 397
P2_SPI_Master_Get_Value64 · 398
P2_SPI_Master_Set_Clk_Wait · 400
P2_SPI_Master_Set_Value32 · 392
P2_SPI_Master_Set_Value64 · 393
P2_SPI_Master_Start · 394
P2_SPI_Master_Status · 395
P2_SPI_Mode · 385
P2_SPI_Slave_Clear_Fifo · 409
P2_SPI_Slave_Config · 402
P2_SPI_Slave_InFifo_Full · 406
P2_SPI_Slave_InFifo_Read · 407
P2_SPI_Slave_OutFifo_Empty · 405
P2_SPI_Slave_OutFifo_Write · 403

SPI-2-T Rev. E

C: P2_Check_LED · 5
D: P2_Digin_Edge · 158
P2_Digin_Fifo_Clear · 159
P2_Digin_Fifo_Enable · 160
P2_Digin_Fifo_Full · 162
P2_Digin_Fifo_Read · 163
P2_Digin_Fifo_Read_Fast · 165
P2_Digin_Fifo_Read_Timer · 167
P2_Digin_Long · 169
P2_Digout · 170
P2_Digout_Bits · 171
P2_Digout_Fifo_Clear · 173
P2_Digout_Fifo_Empty · 174
P2_Digout_Fifo_Enable · 175
P2_Digout_Fifo_Read_Timer · 176
P2_Digout_Fifo_Start · 177
P2_Digout_Fifo_Write · 178
P2_Digout_Long · 180
P2_Digout_Reset · 181
P2_Digout_Set · 182
P2_DigProg · 183
P2_Dig_Fifo_Mode · 153
P2_Dig_Latch · 155
P2_Dig_Read_Latch · 156
P2_Dig_Write_Latch · 157
E: P2_Event_Config · 9
P2_Event_Enable · 7
P2_Event_Read · 12
G: P2_Get_Digout_Long · 186
S: P2_Set_LED · 6
P2_SPI_Config · 386
P2_SPI_Master_Config · 388
P2_SPI_Master_Get_Static_Input · 399
P2_SPI_Master_Get_Value32 · 397
P2_SPI_Master_Get_Value64 · 398
P2_SPI_Master_Set_Clk_Wait · 400
P2_SPI_Master_Set_Value32 · 392
P2_SPI_Master_Set_Value64 · 393
P2_SPI_Master_Start · 394
P2_SPI_Master_Status · 395
P2_SPI_Mode · 385
P2_SPI_Slave_Clear_Fifo · 409
P2_SPI_Slave_Config · 402
P2_SPI_Slave_InFifo_Full · 406
P2_SPI_Slave_InFifo_Read · 407
P2_SPI_Slave_OutFifo_Empty · 405
P2_SPI_Slave_OutFifo_Write · 403

TC-8-ISO Rev. E

S: P2_Sync_All · 13
T: P2_TC_Latch · 235
P2_TC_Read_Latch · 236
P2_TC_Read_Latch4 · 238
P2_TC_Read_Latch8 · 240
P2_TC_Set_Rate · 242

TRA-16 Rev. E

C: P2_Check_LED · 5
D: P2_Digout · 170
P2_Digout_Bits · 171
P2_Digout_Long · 180
P2_Digout_Reset · 181
P2_Digout_Set · 182
P2_Dig_Latch · 155
P2_Dig_Write_Latch · 157
E: P2_Event_Config · 9
P2_Event_Enable · 7
P2_Event_Read · 12
G: P2_Get_Digout_Long · 186
S: P2_Set_LED · 6
P2_Sync_All · 13, · 15

A.3 Thematic Instruction List

The instructions are divided into the following groups. Inside a group the instructions are sorted alphabetically.

Analog Inputs (fast ADC): page A-19
Analog Inputs (multiplexer): page A-20
Analog Outputs: page A-21
Analoge Eingänge (Fast-ADC): page A-21
Bus type: ARINC 429: page A-22
Bus type: CAN: page A-22
Bus type: EtherCAT: page A-22
Bus type: FlexRay: page A-23
Bus type: LIN: page A-23
Bus type: MIL STD 1553: page A-23
Bus type: Profibus: page A-23
Bus type: RSxxx: page A-23
Bus type: SENT: page A-24
Bus type: SPI: page A-25
Communication interface: page A-25
Counters: page A-25
CPU digital channels: page A-26
Digital Inputs/Outputs: page A-26
Multi-I/O: page A-27
PWM outputs: page A-27
SSI decoder: page A-27
System: page A-27
Temperature Inputs: page A-28

Analog Inputs (fast ADC)

P2_ADCF	executes a complete measurement on a Fast-ADC. The return value has a resolution of 16 bit.
P2_ADCF24	executes a complete measurement on a Fast-ADC. The return value has a resolution of 24 bit.
P2_ADCF_Mode	sets the working mode for all channels of the selected modules.
P2_ADCF_Read_Limit	reads the limit-overrun and -underrun flags of all F-ADCs on the specified module.
P2_ADCF_Read_Min_Max4	returns the minimum and maximum values of F-ADC 1…4 of the specified module in an array.
P2_ADCF_Read_Min_Max8	returns the minimum and maximum values of F-ADC 1…8 of the specified module in an array.
P2_ADCF_Reset_Min_Max	resets the minimum and maximum values of selected channels of the specified module.
P2_ADCF_Set_Limit	sets the upper and lower limit for one F-ADC of the specified module.
P2_Burst_CRead_Unpacked1	copies an amount of the last measured values of a channel from the memory of the specified module into an array.
P2_Burst_CRead_Unpacked2	copies an amount of the last measurement values of 2 channels from the memory of the specified module into 2 arrays.
P2_Burst_CRead_Unpacked4	copies an amount of the last measurement values of 4 channels from the memory of the specified module into 4 arrays.
P2_Burst_CRead_Unpacked8	copies an amount of the last measurement values of 8 channels from the memory of the specified module into 8 arrays.
P2_Burst_Init	sets the parameters for a burst-measurement sequence on the specified module.
P2_Burst_Read	copies 32-bit values from the memory of the specified module into a specified array.

P2_Burst_Read_Index returns the address in the module memory, where the last measurement values have been stored.

P2_Burst_Read_Unpacked1 copies the measurement values of a channel into a specified array.

P2_Burst_Read_Unpacked2 copies the measurement values of 2 channels from the memory of the specified module into 2 arrays.

P2_Burst_Read_Unpacked4 copies the measurement values of 4 channels from the memory of the specified module into 4 arrays.

P2_Burst_Read_Unpacked8 copies the measurement values of 8 channels from the memory of the specified module into 8 arrays.

P2_Burst_Reset resets the data pointer of burst sequences on all specified modules.

P2_Burst_Start starts the burst measurement sequence on all specified modules at the same time.

P2_Burst_Status determines the number of burst measurements which are still to execute on the specified module.

P2_Burst_Stop stops a running burst-measurement sequence on all specified modules at the same time.

P2_Read_ADCF reads out the conversion result from an F-ADC of the specified module. The return value has a resolution of 16 bit.

P2_Read_ADCF24 reads out the conversion result from an F-ADC of the specified module. The return value has a resolution of 24 bit.

P2_Read_ADCF32 reads the conversion result from 2 consecutive F-ADCs of the specified module and returns them in a single 32-bit value.

P2_Read_ADCF4 reads out the conversion results from the first 4 F-ADC of the specified module.

P2_Read_ADCF4_24B reads out the conversion results from the first 4 F-ADC of the specified module. The return values have a resolution of 24 bits.

P2_Read_ADCF4_Packed reads out the conversion results from the first 4 F-ADC of the specified module.

P2_Read_ADCF8 reads out the conversion results from all 8 F-ADCs of the specified module.

P2_Read_ADCF8_24B reads out the conversion results from all 8 F-ADC of the specified module. The return values have a resolution of 24 bits.

P2_Read_ADCF8_Packed reads out the conversion results from all 8 F-ADC of the specified module.

P2_Read_ADCF_SConv reads the conversion result from an F-ADC of the specified module and immediately starts a new conversion.

P2_Read_ADCF_SConv24 reads the conversion result from an F-ADC of the specified module and immediately starts a new conversion.

P2_Read_ADCF_SConv32 reads the conversion results from 2 F-ADCs of the specified module and returns them in a 32-bit value.

P2_Set_Average_Filter determines if the module calculates an average and of how many values the average is calculated.

P2_Set_Gain sets the operating mode of a channel on the selected module and thus the gain and measurement range.

P2_Start_ConvF starts the conversion on one or more F-ADCs of the specified module.

P2_Wait_EOCF waits until the end of conversion on all F-ADCs of the specified module.

Analog Inputs (multiplexer)

P2_ADC runs a complete conversion on an ADC of the specified module. The return value has a resolution of 16 bit.

P2_ADC24 runs a complete conversion on an ADC of the specified module. The return value is formatted to 24 bit.

P2_ADC_Read_Limit returns the flags of limit-overrun and -underrun from 16 ADCs of the specified module.

P2_ADC_Set_Limit sets the upper and lower limit for one ADC of the specified module.

P2_Read_ADC returns the conversion result from an ADC of the specified module. The return value has a resolution of 16 bit.

P2_Read_ADC24 returns the conversion result from an ADC of the specified module. The return value has a resolution of 24 bit.

P2_Read_ADC_SConv reads out the conversion result from an ADC of the specified module and immediately starts a new conversion.

P2_Read_ADC_SConv24 reads the conversion result from an ADC of the specified module and immediately starts a new conversion.

P2_Seq_Init	initializes the sequential control of the specified module.
P2_Seq_Read	reads a given number of values (16 Bit) from the specified module and copies them into a destination array.
P2_Seq_Read24	reads a given number of values (18 Bit) from the specified module and copies them into a destination array.
P2_Seq_Read_Packed	reads an even number of value pairs (16 Bit) from the specified module and copies them into a destination array.
P2_Seq_Start	starts the sequence control on all selected modules at the same time.
P2_Seq_Wait	waits until the sequence control has converted and stored all channels of the channel group on the specified module.
P2_Set_Mux	sets the multiplexer of the specified module to the selected input and to the selected gain.
P2_SE_Diff	sets the operating mode single ended or differential for all analog inputs of the specified module.
P2_Start_Conv	starts the conversion on the specified module.
P2_Wait_EOC	waits for the end of conversion on the specified module.
P2_Wait_Mux	waits for the end of the multiplexer settling on the specified module.

Analog Outputs

P2_DAC	outputs an (analog) voltage on the channel of the specified module, that corresponds to the indicated digital value.
P2_DAC1_DIO	outputs an (analog) voltage on the DAC channel 1 and sets or clears the digital outputs of the specified module.
P2_DAC4	outputs 4 digital values from an array on the DAC 1…4 of the specified module as (analog) voltages.
P2_DAC4_Packed	outputs 4 packed digital values from an array on the DAC 1…4 of the specified module as (analog) voltages.
P2_DAC8	outputs 8 digital values from an array on the DAC 1…8 of the specified module as (analog) voltages.
P2_DAC8_Packed	outputs 8 packed digital values from an array on the DAC 1…8 of the specified module as (analog) voltages.
P2_DAC_Ramp_Buffer_Free	returns if the buffer for ramp output is free.
P2_DAC_Ramp_Status	returns if a voltage ramp is being output.
P2_DAC_Ramp_Stop	stops a ramp output immediately.
P2_DAC_Ramp_Write	defines the parameters for the output of the next voltage ramp and starts the DAC output.
P2_Start_DAC	starts the conversion or output of all DAC on the specified module
P2_Write_DAC	writes a digital value into the output register of a DAC on the specified module.
P2_Write_DAC32	copies two 16 Bit values from a 32 Bit value into the output registers of a DAC pair of the specified module.
P2_Write_DAC4	writes 4 digital values from an array into the output registers of the DAC 1…4 of the specified module.
P2_Write_DAC4_Packed	writes 4 packed digital values from an array into the output registers of the DAC 1…4 of the specified module.
P2_Write_DAC8	writes 8 digital values from an array into the output registers of the DAC 1…8 of the specified module.
P2_Write_DAC8_Packed	writes 8 packed digital values from an array into the output registers of the DAC 1…8 of the specified module.

Analoge Eingänge (Fast-ADC)

P2_Burst_CRead_Pos_Unpacked1	P2_Burst_CRead_Pos_Unpacked8 copies an amount of measurement values of 8 channels from the given address of the module memory into 8 arrays.
P2_Burst_CRead_Pos_Unpacked2	P2_Burst_CRead_Pos_Unpacked8 copies an amount of measurement values of 8 channels from the given address of the module memory into 8 arrays.
P2_Burst_CRead_Pos_Unpacked4	P2_Burst_CRead_Pos_Unpacked8 copies an amount of measurement values of 8 channels from the given address of the module memory into 8 arrays.

P2_Burst_CRead_Pos_Unpacked8 copies an amount of measurement values of 8 channels from the given address of the module memory into 8 arrays.

Bus type: ARINC 429

ARINC_Create_Value32 creates a 32 bit value from the components SSM, SDI, data value, and label.

ARINC_Split_Value32 splits a 32 bit ARINC value into its components label, SSM, SDI, data value, and parity bit.

P2_ARINC_Config_Receive configures the receive settings on the specified ARINC module.

P2_ARINC_Config_Transmit configures the transmitter settings on the specified ARINC module.

P2_ARINC_Read_Receive_FifoP2_ARINC_Receive_Fifo_Empty returns if the receiver Fifo of the ARINC interface on the specified module is empty.

P2_ARINC_Receive_Fifo_Empty returns if the receiver Fifo of the ARINC interface on the specified module is empty.

P2_ARINC_Reset runs a master reset on the ARINC interface of the specified module.

P2_ARINC_Set_Labels P2_ARINC_Receive_Fifo_Empty returns if the receiver Fifo of the ARINC interface on the specified module is empty.

P2_ARINC_Transmit_Enable enables or disables transmitting from the transmitter Fifo of the ARINC interface on a specified module.

P2_ARINC_Transmit_Fifo_Empty returns if the transmitter Fifo of the ARINC interface on the specified module is empty.

P2_ARINC_Transmit_Fifo_Full returns if the transmitter Fifo of the ARINC interface on the specified module is full.

P2_ARINC_Write_Transmit_Fifo writes a 32 bit value into the transmitter Fifo of the ARINC interface on the specified module.

Bus type: CAN

CAN_Msg is a one-dimensional array consisting of 9 elements, where the message objects of the CAN bus are saved during sending and receiving.

P2_CAN_Interrupt_Source returns the CAN channels which have generated an event signal (interrupt).

P2_CAN_Set_LED switches the additional LED of a CAN channel on (with color) or off.

P2_En_Interrupt configures a message object of the specified module to generate an event signal (interrupt) when a message arrives.

P2_En_Receive enables a message object on the specified module to receive messages.

P2_En_Transmit enables a message object on the specified module to transmit messages.

P2_Get_CAN_Reg returns the contents of a specified register on a CAN controller on the specified module.

P2_Init_CAN initializes one of the CAN controllers on the specified module and sets it into an initial status.

P2_Read_Msg returns the information if a new message in a message object of one of the CAN controllers on the module has been received.

P2_Read_Msg_Con returns the information if a new message in a message object of one of the CAN controllers on the module has been received.

P2_Set_CAN_Baudrate sets the baud rate on one of the controllers on the specified module and returns the status information.

P2_Set_CAN_Reg writes a value in a register of the selected CAN controller on the specified module.

P2_Transmit reads the data from the array CAN_Msg. As soon as the message object in one of the CAN controllers has access rights to the CAN bus, the message is sent.

Bus type: EtherCAT

P2_ECAT_Get_State returns the operation mode of the EtherCAT interface.

P2_ECAT_Get_Version returns the version of the EtherCAT interface.

P2_ECAT_Init initializes the EtherCAT Slave.

P2_ECAT_Read_Data_16F reads 16 Float values from the EtherCAT slave and returns them in an array.

P2_ECAT_Read_Data_16L reads 16 Long values from the EtherCAT slave and returns them in an array.

P2_ECAT_Set_Mode sets the data transfer mode of the EtherCAT slave.

P2_ECAT_Write_Data_16F writes 16 Float values from an array to the EtherCAT slave.

P2_ECAT_Write_Data_16L writes 16 Long values from an array to the EtherCAT slave.

Bus type: FlexRay

P2_FlexRay_Get_Version returns the version number of the FlexRay interface.

P2_FlexRay_Init initializes the data transfer between ADwin CPU and the FlexRay interface on a specified module.

P2_FlexRay_Read_Word returns a 16 bit value from a FlexRay controller on the specified module.

P2_FlexRay_Reset resets a FlexRay controller on the spcified module.

P2_FlexRay_Set_LED switches a channel LED of a FlexRay controller on the specified module on or off.

P2_FlexRay_Write_Word writes a 16 bit value to an address in a FlexRay controller of the specified module.

Bus type: LIN

P2_LIN_Get_Version returns the version number of the LIN interface.

P2_LIN_Init initializes the data transfer between ADwin CPU and the LIN interface on a specified module.

P2_LIN_Init_Apply activates the initialization data given with P2_LIN_Init_Write for all LIN interfaces.

P2_LIN_Init_Write sets baudrate and operating mode for a specified LIN interface.

P2_LIN_Msg_Transmit sends a header and the identifier of a message box to the LIN bus. To use only with operating mode LIN master.

P2_LIN_Msg_Write configures a message box of a LIN interface for send or receive.

P2_LIN_Read_Dat reads the data of a message box or the status of a LIN interface and writes the result into an array.

P2_LIN_Reset resets all LIN interfaces, either all settings (start-up status) or LIN-internal counters only.

P2_LIN_Set_LED switches the additional LED of a LIN interface on (with color) or off.

Bus type: MIL STD 1553

P2_MIL_Get_Register returns a register value from the MIL interface on the specified module.

P2_MIL_Reset initializes the MIL interface on the specified module and resets all registers to the default value.

P2_MIL_Set_LED switches the additional LEDs of the MIL interface on the specified module on or off.

P2_MIL_Set_Register sets a register value in the MIL interface on the specified module.

P2_MIL_SMT_Init initializes the 16-bit simple monitoring terminal mode (SMT) for both buses A and B on the specified module.

P2_MIL_SMT_Message_Read reads Command Buffer and Data Buffer Block of the recently recorded MIL message on the specified module.

P2_MIL_SMT_Set_All_Filters enables or disables filtering of all receive and transmit subaddresses of all remote terminals for the MIL interface on the specified module.

P2_MIL_SMT_Set_Filter enables or disables filtering of all receive and transmit subaddresses of a single remote terminal for the MIL interface on the specified module.

Bus type: Profibus

P2_Init_Profibus initializes the Profibus Slave.

P2_Run_Profibus exchanges data with the Profibus Slave.

Bus type: RSxxx

P2_Check_Shift_Reg returns, if all data has been sent, which was written into the send-Fifo of the channel on the specified module.

P2_Get_RS reads out the controller register on the specified module.

P2_Read_Fifo reads a value from the input Fifo of a specified channel on the specified module.

P2_RS485_Send determines the transfer direction for a specified channel on the specified module.

P2_RS_Init initializes one channel on the specified module.

P2_RS_Reset executes a hardware reset on the specified module and deletes the settings for all channels.

P2_RS_Set_LED switches the additional LED of a RSxxx channel on (with color) or off.

P2_Set_RS writes a value into a specified register on the specified module.

P2_Write_Fifo writes a value into the send-Fifo of a specified channel on the specified module.

P2_Write_Fifo_Full returns if there is at least one free element in the send-Fifo of a specified channel on the specified module.

Bus type: SENT

P2_SENT_Check_Latch returns whether the latch contains data of the requested SENT channels.

P2_SENT_Clear_Serial_Message_Array clears the buffered pattern of serial messages of a SENT channel.

P2_SENT_Command_Ready returns whether the SENT interface on the specified module is ready to process the next command.

P2_SENT_Config_Output does the basic settings for a SENT output channel on the specified module.

P2_SENT_Config_Serial_Messages configures the sending format for serial messages on a SENT channel of the specified module.

P2_SENT_Enable_Channel enables or disables a SENT channel on the specified module.

P2_SENT_Fifo_Clear initializes the write and read pointer of the output Fifo of a SENT channel.

P2_SENT_Fifo_Empty returns the number of free elements in the output Fifo of a SENT channel.

P2_SENT_Get_ChannelState returns the receive modes of the SENT channels of the specified module.

P2_SENT_Get_ClockTick returns the clock tick of a SENT input channel on the specified module.

P2_SENT_Get_Fast_Channel1 reads the first 12 bit value from a SENT channel on the specified module.

P2_SENT_Get_Fast_Channel2 reads the second 12 bit value from a SENT channel on the specified module..

P2_SENT_Get_Fast_Channel_CRC_OK returns the result of the CRC check for the signals of a SENT message on a channel of the specified module.

P2_SENT_Get_Latch_Data reads data of a SENT message of a SENT channel from the latch buffer.

P2_SENT_Get_Msg_Counter returns the number of sent / received messages on the specified SENT channel.

P2_SENT_Get_Output_Mode returns the output mode of all SENT channels on the specified module.

P2_SENT_Get_PulseCount returns the number of pulses in a SENT message on an input channel of the specified module.

P2_SENT_Get_Serial_Message_Array returns a complete pattern of serial messages from a channel of the specified module.

P2_SENT_Get_Serial_Message_CRC_OK returns the result of the CRC check for the serial message on a channel of the specified module.

P2_SENT_Get_Serial_Message_Data returns the data value of a serial message from a channel of the specified module.

P2_SENT_Get_Serial_Message_Id returns the ID of a serial message from a channel of the specified module.

P2_SENT_Get_Version returns the version of the SENT interface on the specified module.

P2_SENT_Init initializes the data transfer between ADwin CPU and the SENT interface on a specified module.

P2_SENT_Invert_Channel inverts all output levels of the SENT channels of the specified module.

P2_SENT_Request_Latch requests to buffer the data of selected SENT channels on the specified once in a latch buffer.

P2_SENT_Set_ClockTick switches a SENT channel on the specified module to read mode with the specified clock tick.

P2_SENT_Set_CRC_Implementation sets the checksum implementation type of a SENT channel on the specified module.

P2_SENT_Set_Detection sets a SENT channel of the specified module to detection mode.

P2_SENT_Set_Fast_Channel1 sets the first 12 bit value in a SENT message for a SENT channel on the specified module.

P2_SENT_Set_Fast_Channel2 sets the second 12 bit value in a SENT message for a SENT channel on the specified module.

P2_SENT_Set_Fifo writes new data into the output Fifo of a SENT channel on the specified module.

P2_SENT_Set_Output_Mode sets the output mode for a SENT channel on the specified module.

P2_SENT_Set_PulseCount sets if a pause pulse is expected in SENT messages of an input channel of the specified module.

P2_SENT_Set_Reserved_Bits sets the reserved bits of the status nibble for a SENT channel on the specified module.

P2_SENT_Set_Sensor_Type sets the expected sensor type for the SENT messages on an input channel of the specified module.

P2_SENT_Set_Serial_Message_Data changes a data value in the defined message pattern for serial messages on a SENT channel.

P2_SENT_Set_Serial_Message_Pattern defines a pattern of serial messages for a SENT channel.

Bus type: SPI

P2_SPI_Config configures an SPI interface of the specified module, for both master and slave.

P2_SPI_Master_Config sets (additional) properties of an SPI interface of the specified module.

P2_SPI_Master_Get_Static_Input returns the level of the data line of the SPI bus.

P2_SPI_Master_Get_Value32 reads a (already received) SPI message of up to 32 bits from the input of the SPI interface.

P2_SPI_Master_Get_Value64 reads a (already received) SPI message of up to 64 bits from the input of the SPI interface.

P2_SPI_Master_Set_Clk_Wait inserts several waiting times after a selectable number of clocks into the clock signal of the specified SPI master.

P2_SPI_Master_Set_Value32 provides an SPI message of up to 32 bit length at the master output.

P2_SPI_Master_Set_Value64 provides an SPI message of up to 64 bit length at the master output.

P2_SPI_Master_Start starts the data transfer via SPI bus. If configureed the slave select line of the SPI master is activated automatically.

P2_SPI_Master_Status returns if the data transfer of a SPI master is active or inactive.

P2_SPI_Mode sets the operating mode (SPI master / SPI slave / digital module) of the specified module.

P2_SPI_Slave_Clear_Fifo clears the input Fifo and/or the output Fifo of an SPI slave.

P2_SPI_Slave_Config sets (additional) properties of an SPI slave interface of the module.

P2_SPI_Slave_InFifo_Full returns the number of used values (=received 32 bit values) in the input Fifo.

P2_SPI_Slave_InFifo_Read reads several 32 bit values as SPI messages from the input Fifo of an SPI slave.

P2_SPI_Slave_OutFifo_Empty returns the number of unused values in the output Fifo of an SPI slave.

P2_SPI_Slave_OutFifo_Write writes several 32 bit values as SPI messages into the output Fifo of an SPI slave.

Communication interface

LS-BUS

P2_LS_Digout_Long sets or clears all digital outputs of the specified module HSM-24V on the LS bus according to the transferred 32 bit value.

P2_LS_DigProg sets the digital channels 1…32 of the specified module of type HSM-24V on the LS bus as inputs or outputs in groups of 8 via an interface of the Pro II module.

P2_LS_Dig_IO sets all digital outputs of the specified module HSM-24V on the LS bus to the level High or Low and returns the status of all channels as bit pattern.

P2_LS_DIO_Init initializes the specified module of type HSM-24V on the LS bus via an interface of the Pro II module and returns the error status

P2_LS_Watchdog_Init enables or disables the watchdog counter of a specified module on the LS bus via an interface of the Pro II module.

P2_LS_Digin_Long returns the status of all channels of the specified module HSM-24V on the LS bus as bit pattern.

P2_LS_Get_Output_Status returns the over-current status of outputs of the specified module HSM-24V on the LS bus as bit pattern.

P2_LS_Watchdog_Reset resets the watchdog counters of all modules on the LS bus to the appropriate start value. The counters remain enabled.

Counters

P2_Cnt_Clear sets the counter values of one or more counters to 0 (zero), according to the given bit pattern.

P2_Cnt_Enable enables or disables the counters selected by pattern.

P2_Cnt_Get_PW returns frequency and duty cycle of a PWM counter.

P2_Cnt_Get_PW_HL returns a stored high and low time of a PWM counter.

P2_Cnt_Get_Status returns the counter status register of one counter.

P2_Cnt_Latch transfers the current counter values of one or more counters into the relevant Latch A, depending on the bit pattern.

P2_Cnt_Mode	defines the operating mode of one counter.
P2_Cnt_PW_Enable	enables or disables the counters selected by pattern.
P2_Cnt_PW_Latch	copies the value of one or more PWM counters into a buffer.
P2_Cnt_Read4	transfers a current counter value into Latch A and returns the value.
P2_Cnt_Read	transfers a current counter value into Latch A and returns the value.
P2_Cnt_Read_Int_Register	returns the content of a counter register.
P2_Cnt_Read_Latch4	returns the values of 4 counter latches in an array.
P2_Cnt_Read_Latch	returns the value of a counter's Latch A.
P2_Cnt_Sync_Latch	copies the contents of selected counters and PWM counters into latches.

CPU digital channels

CPU_Digin (T11)	Processor T11 only. CPU_Digin returns, whether an edge was detected at the input DIG I/O of the processor module since the previous call.
CPU_Digout	sets a DIG I/O output of the processor module to the selected TTL level.
CPU_Dig_IO_Config	configures all DIG I/O channels of the processor module.
CPU_Event_Config	configures the Event In channel of the processor module.

Digital Inputs/Outputs

P2_Digin_Long	returns the status of the inputs (bits 31…00) of the specified module as bit pattern.
P2_Digout	sets a single output of the specified module to the level "high" or "low". All other outputs remain unchanged.
P2_Digout_Bits	sets the specified outputs of the specified module to the levels "high" or "low".
P2_Digout_Long	sets or clears all outputs on the specified module via the passed 32 bit value.
P2_Digout_Reset	sets the specified outputs of the specified module to the level "low".
P2_Digout_Set	sets the specified outputs of the specified module to the level "high".
P2_DigProg	programs the digital channels 0…31 of the specified module as inputs or outputs in groups of 8.
P2_DigProg_Bits	programs the digital channels 0…11 of the specified module as inputs or outputs.
P2_Digprog_Set_IO_Level	P2_Digprog_Get_IO_Level returns the set voltage level of a digital channel group.
P2_Dig_Latch	transfers digital information from the inputs to the input latches and from the output latches to the outputs on the specified module.
P2_Dig_Read_Latch	returns the bits from the latch register for the digital inputs of the specified module.
P2_Dig_Write_Latch	writes a 32 bit value into the latch register for the digital outputs on the specified module.
P2_Get_Digout_Long	returns the contents of the output latch (register for digital outputs) on the specified module.
P2_Comp_Filter_Init	sets the filter duration for all comparators on the specified module.
P2_Comp_Init	sets the operating mode for the comparators of a channel group on the specified module.
P2_Comp_Set	sets the comparator thresholds of a channel group on the specified module.
P2_Digin_Edge	returns whether a positive or negative edge has occurred on digital inputs of the specified module.
P2_Digin_Fifo_Clear	clears the edge detection unit on the specified module.
P2_Digin_Fifo_Enable	determines, which input channels of the specified module the edge detection unit will monitor.
P2_Digin_Fifo_Full	returns the number of saved value pairs in the FIFO of the edge detection unit.
P2_Digin_Fifo_Read	reads the value pairs from the FIFO of the edge detection unit and writes them into 2 arrays.
P2_Digin_Fifo_Read_Fast	P2_Digin_Fifo_Read reads the value pairs from the FIFO of the edge detection unit and writes them into 2 arrays.
P2_Digin_Fifo_Read_Timer	returns the current value of the timer on the specified module.
P2_Digin_Filter_Init	sets the filter duration for all digital inputs on the specified module.
P2_Digout_Fifo_Clear	stops the edge output and clears the edge output FIFO on the specified module.
P2_Digout_Fifo_Empty	returns the number of free value pairs in the edge output FIFO.
P2_Digout_Fifo_Enable	sets the output channels of the specified module where edges are output.
P2_Digout_Fifo_Read_Timer	returns the current counter value on the specified module.
P2_Digout_Fifo_Start	starts the edge output on the specified modules.

P2_Digout_Fifo_Write	writes value pairs into the output edge FIFO.
P2_Dig_Fifo_Mode	sets the FIFO operation mode on the specified module, input with edge detection or edge output.

Multi-I/O

P2_MIO_Digin_Long	returns the status of the inputs (bits 7…0) of the specified module as bit pattern
P2_MIO_Digout	sets a single output of the specified module to the level "high" or "low". All other outputs remain unchanged.
P2_MIO_Digout_Long	sets or clears all outputs on the specified module.
P2_MIO_DigProg	programs the digital channels 0…7 of the specified module as inputs or outputs in groups of 4.
P2_MIO_Dig_Latch	transfers digital information from the inputs to the input latches and/or from the output latches to the outputs on the specified module.
P2_MIO_Dig_Read_Latch	returns the bits from the latch register for the digital inputs of the specified module.
P2_MIO_Dig_Write_Latch	writes a 32 bit value into the latch register for the digital outputs on the specified module.
P2_MIO_Get_Digout_Long	returns the contents of the output latch (register for digital outputs) on the specified module.

PWM outputs

P2_PWM_Enable	enables or disables one or more PWM outputs.
P2_PWM_Get_Status	returns the operation status of all PWM outputs.
P2_PWM_Init	sets the defaults for one PWM output.
P2_PWM_Latch	enables frequency and duty cycle of one or more PWM outputs to be output.
P2_PWM_Reset	stops the output of one or more PWM outputs immediately..
P2_PWM_Standby_Value	sets the default TTL levels for all PWM outputs.
P2_PWM_Write_Latch	writes frequency and duty cycle into the latch register.
P2_PWM_Write_Latch_Block	writes frequency and duty cycle for several PWM outputs into the latch registers.

SSI decoder

P2_SSI_Mode	sets the modes of all SSI decoders on the specified module, either "single shot" (read out once) or "continous" (read out continously).
P2_SSI_Read	returns the last saved counter value of a specified SSI counter on the specified module.
P2_SSI_Read2	returns the last saved counter values of both SSI counters on the specified module.
P2_SSI_Set_Bits	sets for an SSI counter on the specified module the amount of bits which generate a complete encoder value.
P2_SSI_Set_Clock	sets the clock rate (approx. 6.1 kHz to 12.5 MHz) on the specified module, with which the decoder is clocked.
P2_SSI_Set_Delay	sets the waiting time between reading two encoder values for one SSI-decoder on the specified module.
P2_SSI_Start	starts the reading of one or both SSI decoders on the specified module (only in mode "single shot").
P2_SSI_Status	returns the current read-status on the specified module for a specified decoder.

System

P2_Check_LED	returns the status of the LED (on top of the front panel) of the module.
P2_Event2_Config	configures the pre-processing of event signals on the specified module.
P2_Event_Config	configures the external event input of the specified module.
P2_Event_Enable	enables or disables an external event input on the specified module.
P2_Event_Read	returns the current TTL level at the event inputs of the specified module.
P2_Sync_Mode	enables or disables the synchronization (of conversions) with other modules as master or as slave.
P2_Set_LED	switches the LED (on top of the front panel) on or off.
P2_Sync_All	starts a specified action synchronically on the selected modules.
P2_Sync_Enable	enables or disables the synchronizing option for selected inputs, outputs or function groups on the specified module.
P2_Sync_Stat	returns the settings of the synchronizing option of the specified module.

Temperature Inputs

P2_RTD_Channel_Config	sets the temperature measuring mode for a certain channel on the specified module.
P2_RTD_Config	initializes the temperature measurement on the specified module.
P2_RTD_Convert	calculates the resistance or the temperature in degrees Celsius or Fahrenheit from the measured digital value of a temperature sensor.
P2_RTD_Read	returns the current digital measurement value of a temperature sensor at the specified channel on the module.
P2_RTD_Read8	returns the current digital measurement values of temperature sensors at all channels on the module.
P2_RTD_Start	starts the temperature measurement cycle on all specified modules at the same time.
P2_RTD_Status	returns the status of temperature measurement cycle in "single shot" mode on the specified module.
P2_TC_Latch	copies the current voltage values at the inputs into latches.
P2_TC_Read_Latch	returns the thermo voltage (µV) or temperature (°C / °F) of the selected channel of the module.
P2_TC_Read_Latch4	returns the thermo voltage (µV) or temperature (°C / °F) of the channels 1…4 of the module.
P2_TC_Read_Latch8	returns the thermo voltage (µV) or temperature (°C / °F) of the channels 1…8 of the module.
P2_TC_Set_Rate	sets the sampling rate of the selected module.