JÄGER
Computergesteuerte
Messtechnik GmbH

ADwin-X-A20

Manual

ADwin-X-A20, Manual Sep. 2019

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

Hotline:	+49 6251 96320
Fax:	+49 6251 56819
E-Mail:	info@ADwin.de
Internet	www.ADwin.de

Jäger Com-
putergesteuerte
Messtechnik GmbH
Rheinstraße 2-4
D-64653 Lorsch
Germany

Table of Contents

Table of Contents . . . III

1 Typografische Konventionen . . . V

1 Information about this Manual . . . 1

2 System description . . . 2
- 2.1 *ADwin* system concept . . . 2
- 2.2 *ADwin-X-A20* . . . 3

3 Operating Environment . . . 7

4 Initialization of the Hardware . . . 8

5 Overview Inputs and Outputs . . . 9

6 X-A20 Basic . . . 11
- 6.1 Multi-color LED . . . 11
- 6.2 Analog inputs, 18-bit . . . 11
- 6.3 Analog outputs, 12-bit . . . 14
- 6.4 Analog outputs, 16-bit . . . 14
- 6.5 TTL digital channels DIO39:DIO32 . . . 15
- 6.6 Event Input . . . 15
- 6.7 LS-Bus . . . 16
- 6.8 Synchronous Actions . . . 16

7 Option CO1 . . . 17

8 Option D . . . 18
- 8.1 Diff. digital channels DIO47:DIO40 . . . 18
- 8.2 Diff. counters 4, 5 . . . 18
- 8.3 SSI interface . . . 19

9 Option DCT . . . 21
- 9.1 TTL-digital channels DIO31:DIO00 . . . 21
- 9.2 Comparator inputs DIO59:DIO48 . . . 21
- 9.3 Edge control and Edge output . . . 22
- 9.4 TTL Counters 2, 3 . . . 22
- 9.5 Comparator Counters 6, 7 . . . 23

10 Option COM . . . 24
- 10.1 CAN interfaces . . . 24
- 10.2 RS232 interface . . . 25

11 Option Profibus 27

12 Option Profinet-IRT 30

13 Option EtherCAT 34

14 Option Boot 37

15 Counter block 38
- 15.1 Evaluation of the Counter Contents 40
- 15.2 Using Event Counter 41
- 15.3 Using PWM Counter 43

16 Software 45
- 16.1 General instructions 46
- 16.2 Analog Inputs and Outputs. 53
- 16.3 Digital Inputs and Outputs 77
- 16.4 Counter 110
- 16.5 SSI interface. 127
- 16.6 CAN interface. 135
- 16.7 RSxxx Interface 146
- 16.8 Profibus interface 152
- 16.9 Profinet interface 156
- 16.10 EtherCAT interface. 159

Annex A-1
- A.1 Technical Data. A-1
- A.2 Hardware revisions A-6
- A.3 RoHS Declaration of Conformity A-6

1 Typografische Konventionen

Das „Achtung“-Zeichen steht bei Informationen, die auf Folgeschäden durch Fehlbedienung an der Hard- oder Software, am Messaufbau oder an Personen hinweisen.

Einen „Hinweis“ finden Sie bei

– Informationen, die für einen fehlerfreien Betrieb unbedingt beachtet werden müssen.

– Tipps und Ratschlägen für einen effizienten Betrieb.

Das Zeichen „Information“ verweist auf weiterführende Informationen in dieser Dokumentation oder andere Quellen wie Handbücher, Datenblätter, Literatur etc.

`C:\ADwin\…`

Dateinamen und -verzeichnisse sind in spitzen Klammern und im Schrifttyp Courier New angeben.

`Programmtext`

Programmanweisungen und Benutzer-Eingaben sind durch den Schrifttyp Courier New gekennzeichnet.

`Var_1`

Elemente eines Quelltextes wie Befehle, Variablen, Kommentar und sonstiger Text werden im Schrifttyp Courier New und farbig dargestellt.

In einem Datenwort (hier: 16 Bit) werden die Bits wie folgt nummeriert:

Bit-Nr.	15	14	13	…	1	0
Wert des Bits	2^{15}	2^{14}	2^{13}	…	$2^1=2$	$2^0=1$
Bezeichnung	MSB	-	-	-	-	LSB

1 Information about this Manual

This manual contains complex information about the operation of the *ADwin-X-A20* system. Additional information are available in

– the manual "ADwin Installation", which describes all interface installations for the *ADwin* systems.
  With this manual, you begin your installation!
– the description of the configuration program *ADconfig*, with which you initialize the communication from the corresponding interface to your *ADwin-X-A20* system.
– the manual *ADbasic*, which explains basic instructions for the compiler *ADbasic* and the functional layout of the *ADwin* system.

  The online help of *ADbasic* contains the same information.
– the manuals for all current development environments containing the description of installation and instructions.
– installation and instruction manuals for drivers of all popular development environments
– the manual "*ADwin* HSM-24V" which describes a module for the LS bus.

Please note: This manual is still in progress, errors can be contained.

Please note:

For *ADwin* systems to function correctly, follow strictly 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.

2 System description

2.1 *ADwin* system concept

ADwin systems guarantee fast and accurate operation of measurement data acquisition and automation tasks under real-time conditions. This offers an ideal basis for applications such as:

- very fast digital closed-loop control systems
- very fast open-loop control systems
- data acquisition with very fast online analysis of the measurement data
- monitoring of complex trigger conditions and many more

ADwin systems are optimized for processes, which need **very short process cycle times** of one millisecond down to some microseconds.

System features

The ***ADwin*** system is equipped with analog and digital inputs and outputs, a fast processor (32-bit or 64-bit floating-point signal processor) and local memory. The processor is responsible for the whole real-time processing in the system. The applications run **independent** of the PC and its workload.

Processor

The processor of the ***ADwin*** system processes ***each* measurement value at once**.

In one cycle, you can acquire the status of the inputs, process the status with the help of any mathematical functions, and react to the results, even at very fast process cycle times of some microseconds. This results in a perfect and logical work sharing: The PC executes a program for visualizing of data, for input and operation of the processes, togeher with access to networks and data bases, while the processor of the ***ADwin*** system executes all tasks, which require real-time processing concurrently.

Real-time operating system

The operating system for the DSP of the ***ADwin*** system has been optimized to achieve the fastest response times possible. It manages parallel processes in a **multitasking** environment. Low priority processes are managed by time slicing. Specified high priority processes interrupt all low priority processes and are immediately and completely executed (preemptive multitasking). High priority processes are executed as time-controlled or event-controlled processes (external trigger).

Timing

The built-in **timer** is responsible for the precise scheduling of high priority processes. It has a resolution of 25 nanoseconds (3,3ns since processor T11). The ***ADwin*** systems are characterized by an extremely short response time of only 300 nanoseconds during the change from a low to a high priority process. A continuously running communication process enables a continuous data exchange between the ***ADwin*** system and the PC even while applications are active. The communication has no influence on the real-time capability of the ***ADwin*** system, even so, it is possible to exchange data at any time.

ADbasic

The real-time development tool ***ADbasic*** gives the opportunity to create time-critical programs for ***ADwin*** systems very easily and quickly. ***ADbasic*** is an **integrated development environment** under Windows with possibilities of online debugging. The familiar, easy-to-learn BASIC instruction syntax has been extended by many more functions, in order to allow direct access to inputs and outputs as well as by functions for process control and communication with the PC.

Communication between *ADwin* system and PC

Interfaces

The ***ADwin*** system is connected to the PC via an **USB or Ethernet** interface. After power-up the ***ADwin*** system is booted from the PC via this interface. Afterwards the ***ADwin*** operating system is waiting for instructions from the PC, which it will process.

Instruction processing

There are two kinds of instructions: On the one hand instructions, which transfer data from the PC to the ***ADwin*** system, for instance "load process", "start process" or "set parameter", on the other hand instructions, which wait for a response from the ***ADwin*** system, for instance "read variables" or "read data sets". Both kinds of instructions are processed immediately by the ***ADwin*** system, which means immediate and complete responses. The ***ADwin*** system never sends data to the PC without request! The data transfer to the PC is always a response to an instruction coming from the PC. Thus, embedding the ***ADwin*** system into various programming languages and standard software packages for measurements is held simple, because they have only to be able to call functions and process the return value.

Software interfaces

Under Windows 95/98/NT/ME/2000/XP/Vista, you can use a **DLL** and an **ActiveX** interface. On this basis the following drivers for **development environments** are available: .NET, Visual Basic, Visual-C, C/C++, Delphi, VBA (Excel, Access, Word), TestPoint, LabVIEW / LabWINDOWS, Agilent VEE (HP-VEE), InTouch, DIAdem, DASYLab, SciLab, MATLAB.

Versions for Linux, Mac OS and Java are available, too.

The simple, instruction-oriented communication with the ***ADwin*** system enables several Windows programs to access the same ***ADwin*** system in coordination at the same time. This is of course a great advantage when programs are being developed and installed.

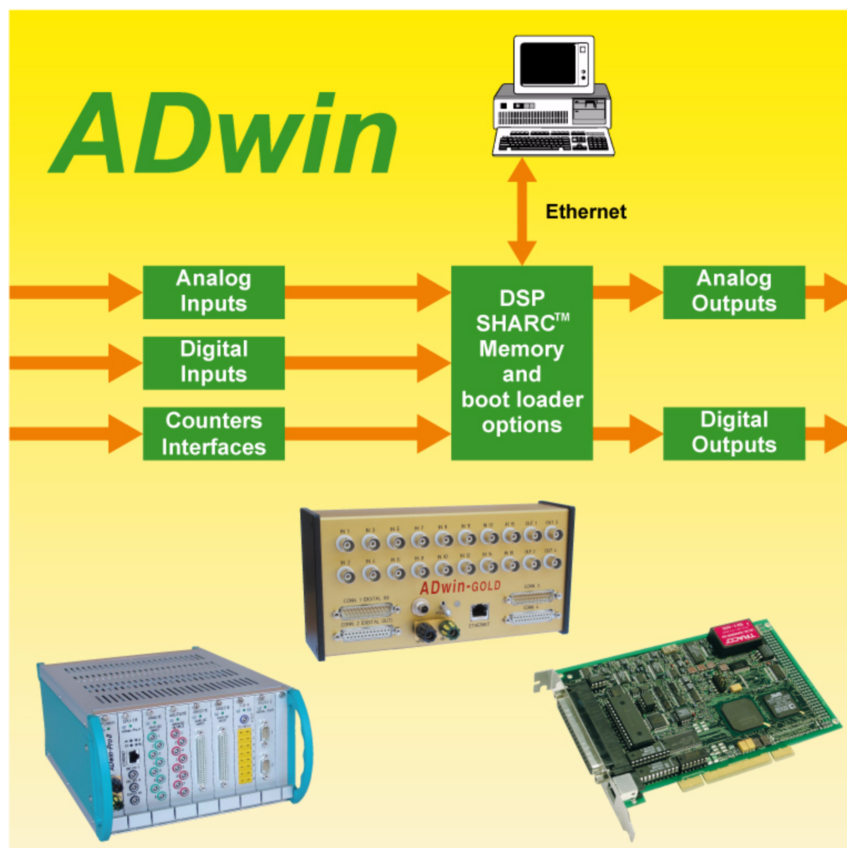


Fig. 1 – Concept of the ***ADwin*** systems

2.2 *ADwin-X-A20*

Prozessor und Speicher

ADwin-X-A20 is euqipped with the digital **signal processor** XILINX ZYNQ™ with Dual-Core ARM Cortex-A9 (666 MHz), which processes 64 bit float and 32 bit integer. It is responsible for the complete measurement data acquisition, online processing, and signal output, and makes it possible to instantaneously process sample rates in the range of 200 Kilohertz to 1 Megahertz.

The **memory 1 GiB** is large enough for all tasks and even bulk data. An integrated cache memory allows very short access time and holds the complete *ADwin* operating system, *ADbasic* processes and all variables.

In order to get maximum access times, all inputs and outputs are memory-mapped in the external memory section of the DSP.

The number and function of inputs and outputs differ according to the selected variant of X-A20. The following text describes all available functions.

Analog inputs

The system has 8 analog inputs (differential) on a DSub connectors. The input signals are converted by a 18-bit analog-to-digital converter (ADC), see Fig. 2 "Block diagram". According to the *ADwin-X* version, the sampling sequence converts the digital value of one channel (X-A20-M1) or of up to 8 channels synchronously (X-A20-F) .

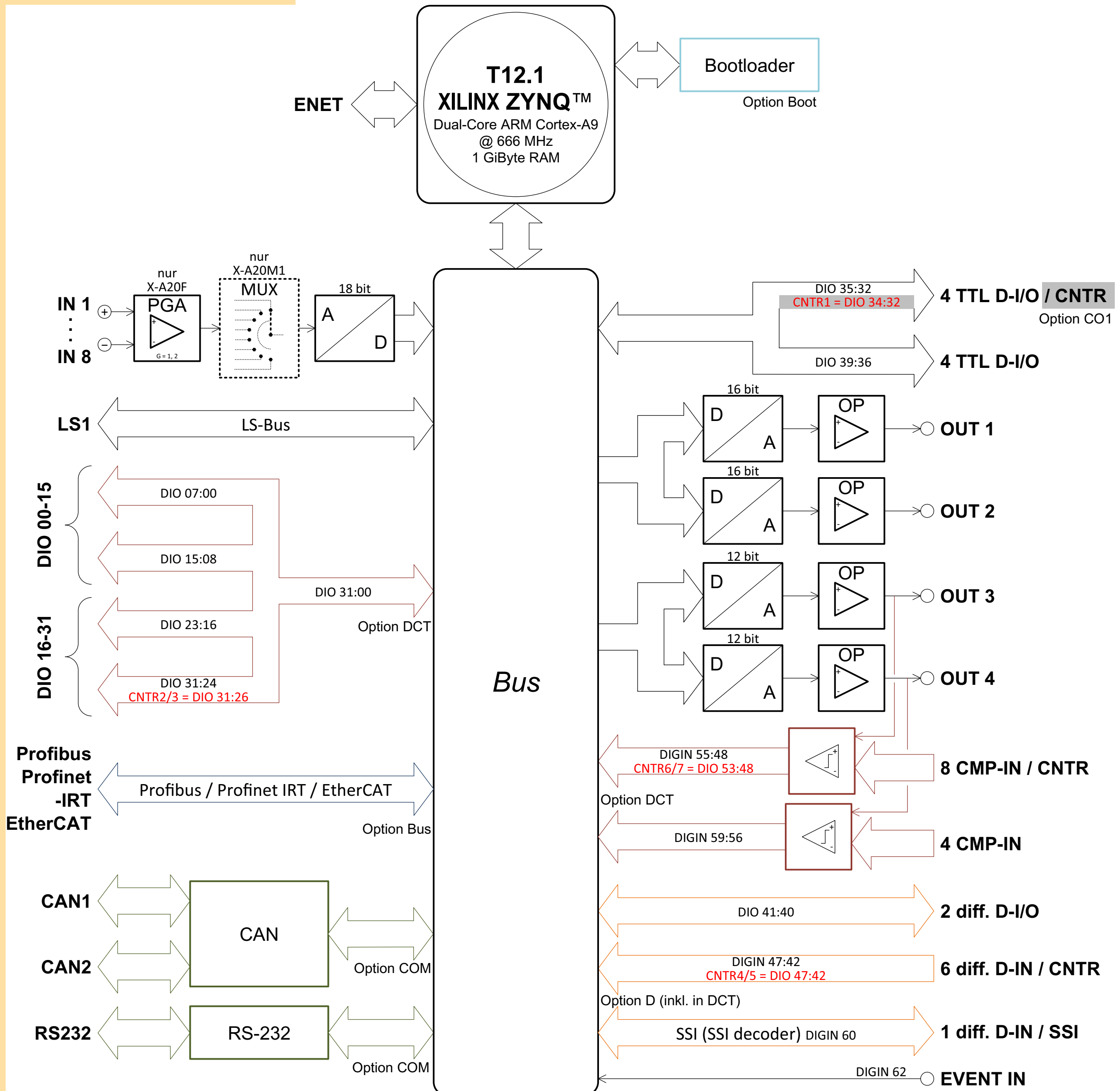


Fig. 2 – Block diagram

Analog outputs

ADwin-X-A20 is equipped with **2 analog outputs of 16-bit resolution** with an output voltage range of -10V … +10V. You can synchronize the voltage output of all DACs via software. The output signal is smoothed by a low-pass filter with a cut-off frequency of f_g = 700 kHz.

There are also **2 analog outputs with 12-bit resolution**, conversion rate 1ms. The output signals are internally used for comparator inputs and counters.

Digital inputs and outputs

32 digital inputs or outputs are available on D-Sub connectors. They can be programmed in groups of 8 as inputs or outputs. The inputs or outputs are TTL-compatible.

D-Sub connectors provide a range of 8 to 61 **digital inputs and outputs**, according to the *ADwin-X* version. Generally, the digital channels are TTL-compatible, but there are also 8 differential channels and 8 comparator inputs. Most channels can be programmed in groups as inputs or outputs, but some are already fixed. Partly, digital channels have double allocation and are also used as counter inputs.

Using a FIFO, an edge control of digital inputs is available. With digital outputs, FIFOs allow to output edges at specified points in time.

Counters

There are overall **7 counter blocks** with 32-bit; a counter block provides one up/down counter and one PWM counter. Counter blocks are equal in function but can process different inputs signals: TTL-like signals, differential signal, comparator signals.

Trigger input (EVENT)

ADwin-X-A20 has a trigger input (EVENT, see also page 15). Processes can be triggered by a signal and are completely processed afterwards. (see *ADbasic* manual, chapter "Processes in the *ADbasic* System").

24 Volt signals

Using the LS bus interface (see page 16), up to 15 LS bus modules can be addressed. The LS bus module HSM-24V provides 32 digital channels for 24 Volt signals.

Schnittstellen

There are interfaces for CAN (High Speed), RS232, SSI, Profibus, Profinet-IRT, and EtherCAT. More interfaces are available on request.

Bootloader

The Bootloader starts a previously programmed application automatically after power-up. After installation of this application, an operation without computer is possible.

Standard delivery

The standard delivery items for *ADwin-X-A20*:

- Hardware *ADwin-X-A20* (Design and Functions as ordered).
- Cross-over Ethernet cable, length 1.8 meters.
- Three-pole power supply cable with one power supply plug, length 1.0 meters.

  The open cable end is used for connection to the external power supply (self-assembly); see appendix for power supply characteristics.
- *ADwin* software package.
- Manual "Driver Installation".
- This manual.

2.2.1 Design

ADwin-X-A20 is available in the following designs:

- Standard: Metal enclosure as desktop unit.
- A20-R: Metal enclosure for installation in 19“ racks. All +Bus options (PROFIBUS, PROFINET, ECAT, see below) are not available.

2.2.2 Functions

ADwin-X-A20 as basic version is available as *X-A20-M1* or *X-A20-F*. The basic version can be combined with any of several opptions.

Options		Functions	Page
X-A20 Basic		8 analog inputs, 18-bit 2 analog outputs, 16-bit 2 analog outputs, 12-bit	page 11
		8 TTL digital channels 1 Event input	page 11
		1 LS bus interface	page 16
	M1	Analog inputs with multiplexer: 1 measurement value per conversion, 200 kHz.	
	F	Analog input, fast: up to 8 measurement values converted synchronously, 200 kHz…800 kHz; gain selectable.	
+CO1		1 TTL counter block: 32-bit up/down counter and PWM counter	page 17
+DCT (inklusive D)W		32 TTL digital channels 12 comparator inputs 2 TTL counter blocks: 32-bit up/down counter and PWM counter 2 comparator counter blocks: 32-bit up/down counter and PWM counter Input FIFO and output FIFO for digital channels	page 21
	+D	6 diff. digital inputs + 2 diff. digital channels 2 differential counter blocks: 32-bit up/down counter and PWM counter 1 SSI interface	page 18
+COM		2 CAN interfaces (high speed) 1 RS232 interface	page 24
+Bus	+PROFI-BUS	1 Profibus interface (Slave)	page 27
	+PROFI-NET	1 Profinet-IRT interface (Slave)	page 30
	+ECAT	1 EtherCAT-Schnittstelle (Slave)	page 34
+Boot		Flash EPROM bootloader for independent processing without PC	page 37

Option DCT is expansion of option D, so these two options cannot be combined.

Options PROFIBUS, PROFINET, and ECAT cannot be combined.

2.2.3 Accessories

For *ADwin-X-A20*, the supplementary accessories are availabe:

- *ADbasic*, real-time development tool for all *ADwin* systems.
- *A20-Mount*: A20 mount: Housing for DIN rail mounting in a switch cabinet with insulated clips.
- *A20*-Pow: external power supply.
- *A20*-Pow-Mount: external power supply for DIN rails.
- *HSM-24V*: DIN rail module for LS bus interface, 32 digital I/Os, 24 V level, configurable in groups of 8, screw terminals.

3 Operating Environment

With the necessary accessories, the system can be operated in 19-inch-enclosures or as a mobile system (e.g. in cars).

The *ADwin-X-A20* device **must be earth-protected**, in order to

– build a ground reference point for the electronic

– conduct interferences to earth.

Connect the GND plug, which is internally connected with the ground reference point and the enclosure, via a short low-impedance solid-type cable to the central earth connection point of your device.

Galvanic connection

The data lines at the version with Ethernet interface are optically isolated, but the ground potentials are connected, because the shielding of the Ethernet connector (RJ-45) is connected to GND.

Excluding transient currents

Transient currents, which are conducted via the aluminum enclosure or the shielding, have an influence on the measurement signal.
Please, make sure that the shielding is not reduced, for instance by taking measures for bleeding off interferences, such as connecting the shielding to the enclosure just before entering it. The more frequently you earth the shielding on its way to the machine the better the shielding will be.

Use cables with shielding on both ends for signal lines. Here too, you should reduce the bleeding off of interferences via the enclosure by using screen clips.

Protection low voltage

The *ADwin-X-A20* is externally operated with a protection low voltage of 10V to 35V; internally it is operated with a voltage of +5V and ±15V against GND. It is not life-threatening. For operation with an external power supply, the instructions of the manufacturer applies.

Ambient temperature

The *ADwin-X-A20* is designed for operation in dry rooms with a room temperature of +5°C … +50°C and a relative humidity of 0 … 80% (no condensation, see Annex).

Chassis temperature

The temperature of the chassis (surface) must not exceed +60°C, even under extreme operating conditions – e.g. in an enclosure or if the system is exposed to the sun for a longer period of time. You risk damages at the device or not-defined data (values) are output, which can cause damages at your measurement device under unfavorable circumstances.

4 Initialization of the Hardware

If you start initializing do not connect any cables to the *ADwin-X-A20* before you have executed the following steps:

1. Software installation / installation in PC or 19" enlosure

   Follow the manual "*ADwin* Driver Installation".
2. Set the operating environment, see chapter 3.
3. Read chapter 5 “Overview Inputs and Outputs“ in this manual.
4. Begin now with the connection of the inputs and outputs.

Notes

Providing the power supply

Please pay attention that reliable power source is supplied.
This concerns the computer (standard delivery). Otherwise also the external power supply, if operated in a car, the battery voltage.

If using current-limiting power supplies, please pay attention to the fact, that after power-up the current demand can be a multiple of the idle current. More detailed information can be found in the Technical Data (Annex).

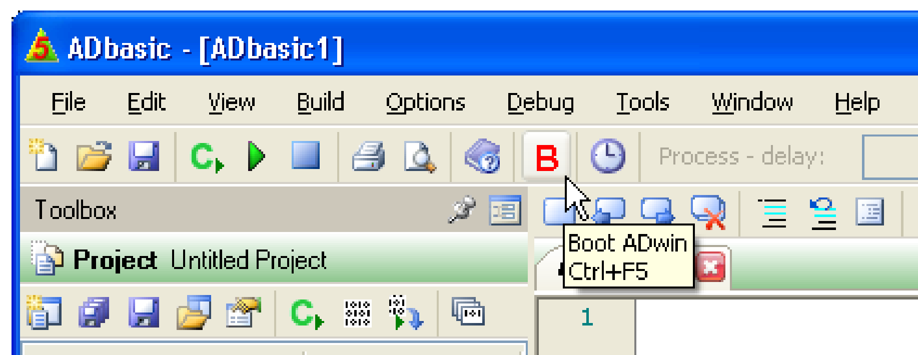
In case of a power failure, all data, which have not been saved are lost. Not-defined data (values) can under unfavorable circumstances cause damages to other equipment.

Avoid direct contact to uninsulated parts to be secure of electrostatic charging.

Checking the Connection

Booting

Start *ADbasic* and boot the *ADwin* system by clicking on the boot button B.



The flashing LED1 (green colored) of the system and the display in the *ADbasic* status line: "`ADwin is booted`" show that the operating system has been loaded and *ADbasic* can connect the *ADwin* system. (If not, please check the connectors first).

Programs with *ADbasic*

Programming the *ADwin* systems is described more detailed in the *ADbasic* manual. Instructions for access to *ADwin-X-A20* I/Os are described in chapter 16 on page 45.

Start with the programming examples in the *ADbasic* Tutorial.

5 Overview Inputs and Outputs

ADwin-X-A20 provides the following inputs and outputs (Pin assignment see next page). According to the built-in optionsm, only some of the pins may be used.

– Ethernet connector
– Power supply connector
– GND connector
– 3 D-Sub sockets, 37-pole: `Conn. 1`, `Conn. 2`, `Conn. 3`
  - analog inputs, analog outputs
  - digital inputs and outputs: TTL, differential, comparator
  - inputs for counters: TTL, differential, comparator
  - SSI interface
  - digital trigger input (Event)
  - power output +5V

  Some pins have double assignments.
– 2 D-Sub plugs 9-pole, `CAN1`, `CAN2`
– 1 D-Sub plug 9-pole, `RS232`
– 1 D-Sub socket 9-pole, `LS-BUS`

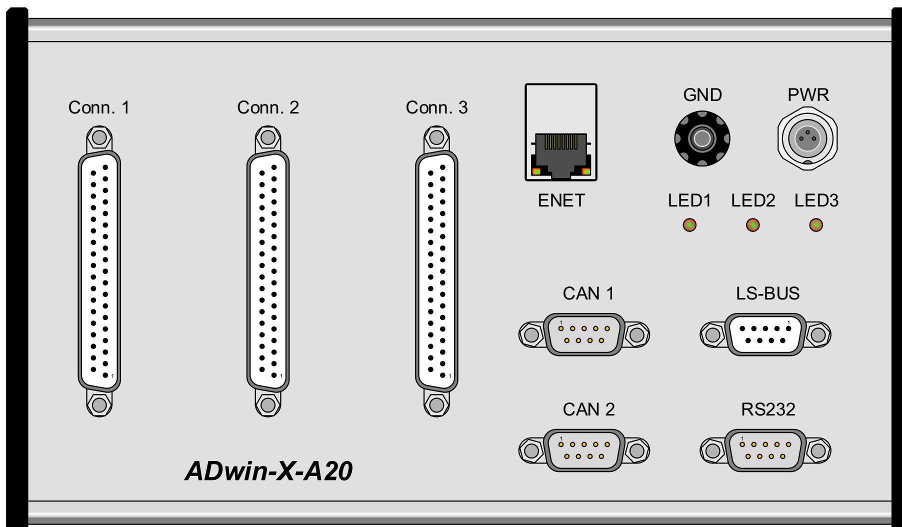


Fig. 3 – Connectors of *ADwin-X-A20*

All inputs and outputs may only be operated according to the specifications given (see Annex A.1 Technical Data). In case of doubt, ask the manufacturer of the device, to which you want to connect *ADwin-X-A20*.

Open-ended inputs can cause errors - above all in an environment where interferences may occur. For your safety, set the inputs, which you do not use to a specified level (for instance GND) and also connect them as close to the connector as possible. Don't connect open ended cables to the inputs; open ended cables may cause spikes at the inputs.
The event input is an exception, it has an internal pull-up resistor (4.7 kΩ).

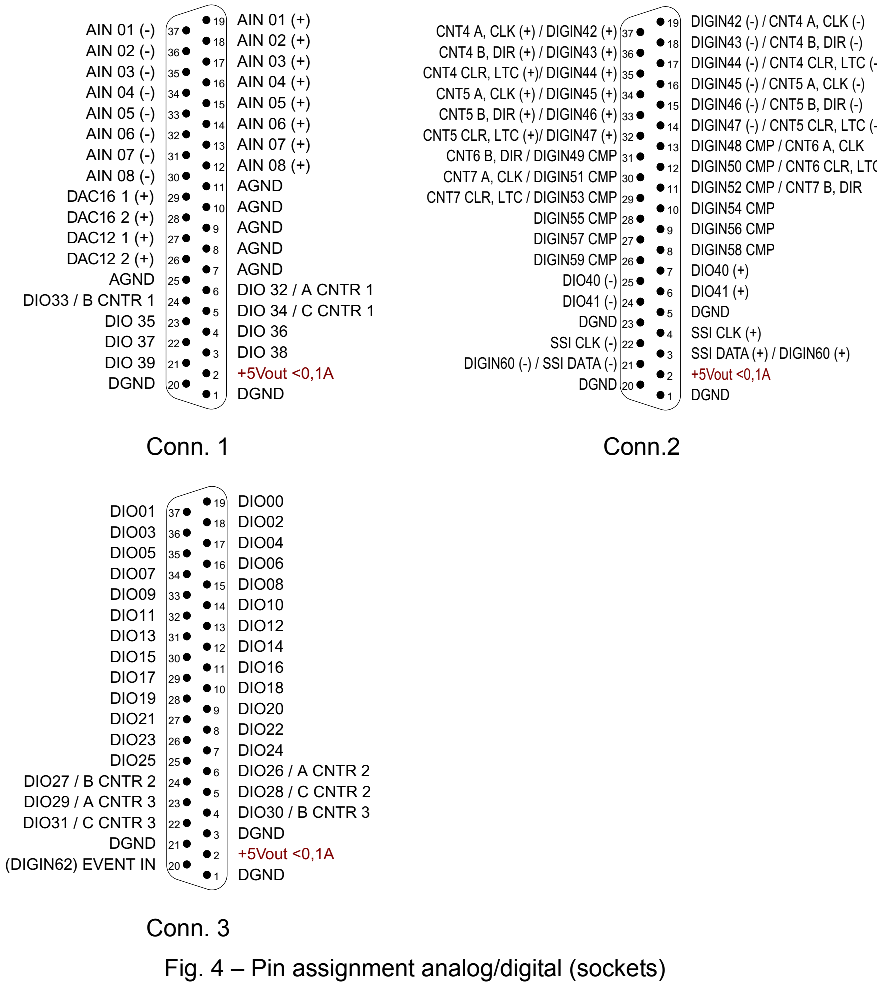
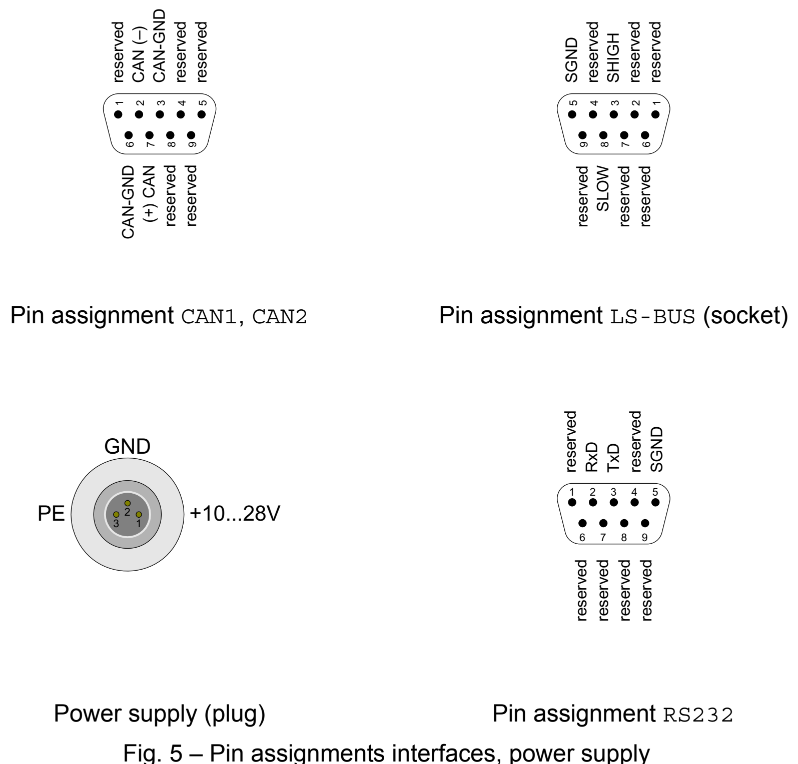


Fig. 4 – Pin assignment analog/digital (sockets)



Pin assignment `CAN1`, `CAN2`

Pin assignment `LS-BUS` (socket)

Power supply (plug)

Pin assignment `RS232`

Fig. 5 – Pin assignments interfaces, power supply

6 X-A20 Basic

The basic version of X-A20 comprises:

- 3 Multi-color LED
- 8 Analog inputs, 18-bit
  - X-A20-M1: Multiplexer
  - X-A20-F: Synchronous conversion
- 2 Analog outputs, 12-bit
- 2 Analog outputs, 16-bit
- 8 TTL digital channels DIO39:DIO32
- 1 Event Input
- 1 LS-Bus
- Synchronous Actions

6.1 Multi-color LED

X-A20 provides 3 multi-color LEDs, which you can switch on and off.

After power-up, LED 1 serves as status LED and glows red; as soon as boot processing has finished, Process 15 is running and makes LED 1 blink green.

Instructions to program LEDs are described starting from page 47 and in the online help. The instructions are defined in the include file `ADwin-X.inc`.

Function	Instructions
Check LED status	**Check_LED**
Switch LED on or off, set color	**Set_LED**

6.2 Analog inputs, 18-bit

X-A20 provides 8 differential analog measuring inputs, which run via a 18-bit analog digital converter (ADC). There are two variants:

- X-A20-M1: Multiplexer

  The 8 analog measuring inputs are allocated to a multiplexer and from there connected to a 18-bit-ADC. The conversion time is 5µs (including multiplexer settling time). The gain factor is set to 1.

- X-A20-F: Synchronous conversion

  Up to 8 analog measuring inputs are converted synchronously. The gain (PGA) is programmable to 1 or 2. The conversion time (for all channels together) depends on the number of converted channels:
  - 1 channel: max. 800kHz = 1.25µs.
  - 2 channels: max. 550kHz = 1.82µs.
  - 3 channels: max. 425kHz = 2.35µs.
  - 4 channels: max. 350kHz = 2.86µs.
  - 5 channels: max. 300kHz = 3.3µs.
  - 6 channels: max. 250kHz = 4.0µs.
  - 7 channels: max. 225kHz = 4.44µs.
  - 8 channels: max. 200kHz = 5µs.

  If you select different channels with a conversion instruction (`ADC…` / `Start_Conv`) than with the previous conversion instruction, the conversion time is extended: With the change, the conversion is processed twice, once with the previously selected channels and once with the newly selected channels.

The input voltage range is ±10V (with gain factor 1).

The analog inputs are differential.For each of the measurement inputs there is a positive and a negative input, between them the voltage difference is measured (but not free of potential). Both, the positive and negative input have to be connected, see Fig. 4 – Pin assignment analog/digital (sockets).

Please note, that the inputs do need a mass connection between the system's GND-plug and the signal source. This is in addition to the connections to the positive and negative input.

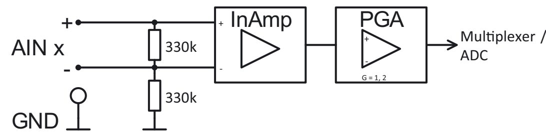


Fig. 6 – Input circuitry of an analog input

Signals are converted fast and accurately (76 µV) with the 18-bit analog-to-digital-converter (ADC). Measurement values can be returned with 16 bit or 24 bit resolution.

Please note the Calculation Basics for evaluation of measurement values.

With X-A20-F there are two alternative options to convert measurement values. With X-A20-M1 both methods are available, but only with a single channel:

- Single conversion: The conversion of one or several channels is started at a defined time and the measurement value(s) is returned in the appropriate time. Each conversion has to be started on its own.
- Continuous conversion: A sequential control continuously converts measurment values on one or several selected channels. You can read the current value without waiting, but the exact time of conversion is unknown. Thus, the processor can be discharged a lot and only has to read completely converted values from the sequential control's buffer.

Earth protection

X-A20 must be earth-protected, in order do measurements free of interference. Connect the GND plug via a short low-impedance solid-type cable to the central earth connection point of your device.

The enclosure is connected to GND via the GND wire of the power supply cable as well as via the shiedling of the Ethernet cable.

The power supply via the power adapter at the PC links the ground potential of the *ADwin-X-A20* with the PC. A voltage difference between ground potentials interferes with operation and can change measurements or cause considerable damage. Avoid disturbances by using an external power supply.

Standard instruction

The standard instruction `ADC()` processes a complete measurement with the ADC on one channel (see page 58) and returns a 16-bit value.

You get measurement values with 18-bit resolution with the instruction `ADC24` (see page 59); the value is returned as a 24-bit value (see page 16).

Both instructions use the 18-bit-ADC, only the return values have different formats.

Please pay attention to a low output resistance of the signal source (of the input signals), because it may have influence on the measuring accuracy. If this is not possible:

Depending on the output resistance a linear error is caused (about 1 digit per 10 Ω).
You can compensate this by multiplying the measurement value with a corresponding factor and get a sort of re-calibration.

Programming

Instructions to program analog inputs (both X-A20-M1 and X-A20-F) are described starting from page 58 and in the online help. The instructions are defined in the include file `ADwin-X.inc`.

Function	Instructions	-M1	-F
Do a complete conversion	`ADC`, `ADC24`	x	x

Do conversions on several channels	`ADC2`, `ADC4`, `ADC8`
	`ADC_2_24`, `ADC4_24`
	`ADC8_24`
		-	x

Do a measurement in steps, start continuous measurement	`Start_Conv`, `Wait_EOC` `Read_ADC`, `Read_ADC24`	x	x
Set gain (and start conversion)	`Start_Conv_PGA`	-	x

Read several measurement values at the same time (measurement in steps)	`Read_ADC_Packed`
	`Read_ADC8`
	`Read_ADC8_24`
		-	x

Start several functions synchronously.	`Sync_All`	x	x

6.2.1 Calculation Basics

Voltage range

The voltage range of the *ADwin-X-A20* at the analog inputs and outputs is between –10V to +10V or bipolar 10V.

Allocation of digits to voltage

The 65536 (2^{16}) digits are allocated to the corresponding voltage ranges of the ADCs and DACs so that[1]

- 0 (zero) digits correspond to the maximum negative voltage and
- 65535 digits correspond to the maximum positive voltage

The value for 65536 digits, exactly +10 Volt, is just outside the measurement range, so that you will get a maximum voltage value of +9.999695V for a 16-bit conversion.

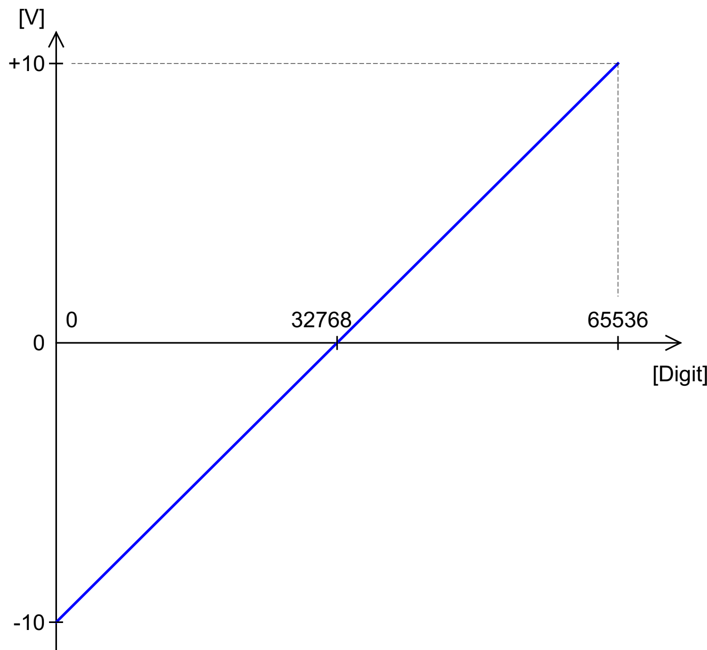


Fig. 7 – Zero offset in the standard setting of bipolar 10 Volts

In the bipolar setting, you will get a zero offset, also called offset U_{OFF} in the following text.
For the voltage range of -10V … +10V applies: U_{OFF} = -10V

Gain factor k_V

X-A20-F has a programmable gain (PGA), which can amplify the input voltage by the factors 1 (±10V) or 2 (±5V). The gain factor also changes the measurement range.

Please note that with a gain factor k_V=2, interference signals are amplified respectively.

Quantization level U_{LSB}

The quantization level (U_{LSB}) is the smallest digitally displayable voltage difference and is equivalent to the voltage of the least significant bit (LSB).

The measured value of the 18-bit ADC can be returned with 16-bit or with 24-bit format. The DACs process values with 12 bit and 16 bit:

- 24-bit format: The 24-bit value holds the 18-bit measurement value in the bits 23:6, the measurement value being shifted by 6 bits to the left, the bits 5:0 are always zero.
- $U_{LSB\ 24bit}$ = 20V / 2^{24} = 1.19µV
  16-bit format: The measurement value is given in the lower word (bits 15:0), the upper word is zero.
  $U_{LSB\ 16bit}$ = 20V / 2^{16} = 305.175µV

  The same applies for a DAC value to be output.

1. With a 24-bit value, 16777216 (2^{24}) digits are allocated to the voltage range.

A digital value for a 12-bit DAC is also given in the 16-bit format, left-aligned in the lower word, bits 3:0 are always zero.

Bit no.	31:24	23:16	15:6	5:4	3:0
Inhalt	0	18-bit value in bits 23:6		0	
	0	0	16-bit value in bits 15:0		
	0	0	12-bit value in bits 15:4		0

Fig. 8 – Storage of the ADC/DAC bits in the memory

Values in the same bit format can be added or subtracted directly, which here applies to 12-bit and 16-bit values. To calculate values with different formats, the more accurate 24-bit value must be shifted 8 bits to the right or the 16-bit value to the left.

Conversion Digit to Voltage

DAC

For a DAC (16-bit format):

$$U_{OUT} = Digits \cdot U_{LSB} + U_{OFF}$$

$$Digits = \frac{U_{OUT} - U_{OFF}}{U_{LSB}}$$

ADC

For an ADC (either 24-bit and 16-bit format):

$$Digits = \frac{k_V \cdot U_{IN} - U_{OFF}}{U_{LSB}}$$

$$U_{IN} = \frac{Digits \cdot U_{LSB} + U_{OFF}}{k_V}$$

Tolerance Ranges

Slight variations regarding the calculated values may be within the tolerance range of the individual component. Two kinds of variations are possible (in LSB), which are indicated in this hardware manual:

INL

– The integral non-linearity (INL) defines the maximum deviation from the ideal straight line over the whole input voltage range (see Fig. 8, page 13).

DNL

– The differential non-linearity (DNL) defines the maximum deviation from the ideal quantization level.

6.3 Analog outputs, 12-bit

ADwin-X-A20 has 2 analog outputs with 12-bit converters (DAC12-1, DAC12-2), see page 10. Each output has a digital analog converter (DAC).

The DAC has a max. conversion time of 1000 µs.

The 12-bit ADCs are internally connected to the comparator inputs DIO59:DIO48 (see Option DCT, page 21). The set DAC voltage serves as comparator signal (0V…5V), i.e. a given digital signal with a lower voltage is processed as level Low, with a higher voltage as level High.

Please note the Calculation Basics for processing of DAC values.

Programming

There is a single instruction to program analog outputs, as described on page 55 and in the online help. Instructions are defined in the include file `ADwin-X.inc`.

Function	Instructions
Output voltage	DAC12

6.4 Analog outputs, 16-bit

ADwin-X-A20 has 2 analog outputs with 16-bit converters (DAC16-1, DAC16-2), see page 10. Each output has its own digital analog converter (DAC).

The DAC has a conversion time of 1 µs.

Please note the Calculation Basics for processing of DAC values.

Programming

Instructions to program analog outputs are described starting from page 55 and in the online help. The instructions are defined in the include file `ADwin-X.inc`.

Function	Instructions
Output voltage	`DAC`
Output voltage in steps	`Write_DAC` `Start_DAC`
Start several functions synchronously.	`Sync_All`

6.5 TTL digital channels DIO39:DIO32

8 digital channels (DIO39:DIO32) are available on D-Sub socket `Conn. 1`, see page 10. The channels are programmable in groups of 4 as inputs or outputs.

The channels DIO34:DIO32 can be also assigned as counter inputs (see Option CO1). In this case only one of the functions (digital channel or counter input) can be used.

The digital channels are TTL-compatible and not protected against over voltage. Inputs have a pull-down-resistor (10kΩ).

Programming

Instructions to program digital channels are described starting from page 78 and in the online help. The instructions are defined in the include file `ADwin-X.inc`.

Function	Instructions
Configure channels	`Conf_DIO`
Configure input filter	`Digin_Filter_Init`
Read input values.	`Digin`, `Digin_Long2`
Control edges of digital inputs.	`Digin_Edge2`

Set digital outputs.	`Digout`
	`Digout_Long2`
	`Digout_bits2`
	`Get_Digout_Long2`

Read and set values via latch register.	`Dig_Latch`
	`Digin_Read_Latch2`
	`Digout_Write_Latch2`

Use output Fifo (with Option DCT only).	`Digout_Fifo_Read_Timer`
	`Digout_Fifo_Clear`
	`Digout_Fifo_Enable`
	`Digout_Fifo_Empty`
	`Digout_Fifo_Mode`
	`Digout_Fifo_Start`
	`Digout_Fifo_Write`

Use input Fifo (with Option DCT only).	`Digin_Fifo_Read_Timer`
	`Digin_Fifo_Clear`
	`Digin_Fifo_Enable`
	`Digin_Fifo_Full`
	`Digin_Fifo_Read`

Start several functions synchronously.	`Sync_All`

6.6 Event Input

ADwin-X-A20 provides an external trigger input (EVENT) on pin 20 of the D-Sub socket `Conn. 3`, see Fig. 4 – Pin assignment analog/digital (sockets).

An external trigger signal with rising edge at the event input can start a process cycle being completely and immediately processed, (see also *ADbasic* manual, chapter: "Processes in the ADwin System").

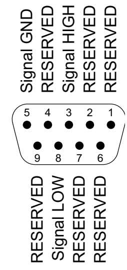
With the instruction `CPU_Event_Config`, you can configure which edges at the event input trigger a process cycle.

The event input has an internal pull-down resistance (4.7kΩ).

The level of the event input can be read via software as digital input DIGIN62.

6.7 LS-Bus

ADwin-X-A20 provides one interface for LS bus on a 9-pin DSub connector `LS-BUS`.



The LS bus is a bi-directional serial bus with 5MHz clock rate (Low Speed). The bus is a in-house-design to access external modules. The first module available is HSM-24V, which can process 24 Volt signals on 32 digital channels.

The bus is set up as line connection, i.e. the *ADwin* interface and up to 15 LS bus modules are connected to each other via two-way links. The last module of the LS bus must have the bus termination activated. The maximum bus length is 5m.

The LS bus modules are programmed with *ADbasic* instructions, which are sent via the LS bus interface of the *ADwin* system. The instructions for the LS bus module are described in the manual of the LS bus module and in the online help.

Programming

Instructions to program LS bus modules are described starting from page 164 and in the online help. The instructions are defined in the include file `ADwin-X.inc`.

Function	Instructions
Initialize LS bus module	`LS_DIO_Init`
Set and read digital channels on the LS bus module 1	`LS_Dig_IO` (no other instructions usable)
Output digital signals	`LS_Digout_Long` `LS_Digout_Long_BS`
Read digital signals	`LS_Digin_Long` `LS_Digin_Long_BS`
Read over-current status from module outputs	`LS_Get_Output`
Reset interface	`LS_Reset`
Use LS bus watchdog	`LS_Watchdog_Init` `LS_Watchdog_Reset`

6.8 Synchronous Actions

ADwin-X-A20 enables to synchronously start actions at different innputs and outputs with the instruction `Sync_All`. The instruction is desribed on page 51.

The following actions are available (according to the version): read and output analog signals, read and output digital signals, start edge output / edge detection on digital channels, copy counter values, reset counters, read SSI signal.

7 Option CO1

Option X-A20-CO1 additionally provides a TTL counter block with number 1.

The counter inputs are on the pins DIO34:DIO32 on DSub socket `Conn. 1.`, see fig. 4 on page 10.

The pins have a double allocation as digital channels, see TTL digital channels DIO39:DIO32. The channels must be configured as digital inputs with `Conf_DIO` to enable the usage as counter inputs.

The counter inputs A/CLK, B/DIR, and CLR/LATCH are TTL compatible and are not protected against over-voltage.

All functions of counter block 1 are described in chapter 15 "Counter block".

Programming

Instructions to program counters are described starting from page 111 and in the online help. The instructions are defined in the include file `ADwin-X.inc`.

Function	Instructions
Configure channels as inputs.	`Conf_DIO`
Clear counter.	`Cnt_Clear`
Disable or enable counter (please note already running counters).	`Cnt_Enable` `Cnt_PW_Enable`
Read out status register.	`Cnt_Get_Status`
Write counter value into Latch A.	`Cnt_Latch`
Write counter value into latch A and read the latch value.	`Cnt_Sync_Latch`
Set counter operation mode.	`Cnt_Mode`
Read latch value.	`Cnt_Read`
Return the content of a counter register.	`Cnt_Read_Latch`
Set counter mode to single ended / differential inputs.	`Cnt_Read_Int_Register`
Write PWM counter values into latch A.	`Cnt_PW_Latch`
Read frequency and duty cycle of a PWM counter.	`Cnt_Get_PW`
Read high time and low time of a PWM counter.	`Cnt_Get_PW_HL`
Start several functions synchronously.	`Sync_All`

8 Option D

Option X-A20-D additionally provides:

- 8 Diff. digital channels DIO47:DIO40
- 2 Diff. counters 4, 5
- 1 SSI interface

8.1 Diff. digital channels DIO47:DIO40

8 differential digital channels (DIGIN47:DIGIN42, DIO41:DIO40) are available on D-Sub socket `Conn. 2`, see fig. 4. The channels DIO41:DIO40 are each programmable as input or output, all other channels are set as inputs.

The channels DIGIN47:DIGIN42 can be also assigned as counter inputs (see Diff. counters 4, 5). In this case only one of the functions (digital channel or counter input) can be used.

The digital channels are differential and not protected against over-current. For each channel there is a positive and a negative pin, between which the voltage difference is measured (but not free of potential). Between each pair of channel pins, there is a bus termination of 120Ω..

The inputs require TTL-like signals.

Programming

Instructions to program digital channels are described starting from page 78 and in the online help. The instructions are defined in the include file `ADwin-X.inc`.

Function	Instructions
Configure channels	**Conf_DIO**
Configure input filter	**Digin_Filter_Init**
Read input values.	**Digin**, **Digin_Long2**
Control edges of digital inputs.	**Digin_Edge2**
Set digital outputs.	**Digout** **Digout_Long2** **Digout_bits2** **Get_Digout_Long2**
Read and set values via latch register.	**Dig_Latch** **Digin_Read_Latch2** **Digout_Write_Latch2**
Start several functions synchronously.	**Sync_All**

8.2 Diff. counters 4, 5

Option X-A20-D additionally provides two differential counters with numbers 4 and 5.

The counter inputs are on the pins DIGIN47:DIGIN42 on DSub socket `Conn. 2.`, see fig. 4 on page 10.

The inputs require TTL-like signals.

The pins have a double allocation as digital channels, see Diff. digital channels DIO47:DIO40. The channels must be configured as digital inputs with **Conf_DIO** to enable the usage as counter inputs.

The counter inputs A/CLK, B/DIR, and CLR/LATCH are differential and not protected against over-voltage. For each channel there is a positive and a negative pin, between which the voltage difference is measured (but not free of potential). Between each pair of channel pins, there is a bus termination of 120Ω.. Both, the positive and negative input have to be connected at each input.

All functions of counter blocks 4 or 5 are described in chapter 15 “Counter block“.

Programming

Instructions to program counters are described starting from page 111 and in the online help. The instructions are defined in the include file `ADwin-X.inc`.

Function	Instructions
Configure channels as inputs.	**Conf_DIO**
Clear counter.	**Cnt_Clear**

Function	Instructions
Disable or enable counter (please note already running counters).	Cnt_Enable Cnt_PW_Enable
Read out status register.	Cnt_Get_Status
Write counter value into Latch A.	Cnt_Latch
Write counter value into latch A and read the latch value.	Cnt_Sync_Latch
Set counter operation mode.	Cnt_Mode
Read latch value.	Cnt_Read
Return the content of a counter register.	Cnt_Read_Latch
Set counter mode to single ended / differential inputs.	Cnt_Read_Int_Register
Write PWM counter values into latch A.	Cnt_PW_Latch
Read frequency and duty cycle of a PWM counter.	Cnt_Get_PW
Read high time and low time of a PWM counter.	Cnt_Get_PW_HL
Start several functions synchronously.	Sync_All

8.3 SSI interface

An incremental encoder with SSI interface can be connected the decoder. The signals are differential and have RS422/485 levels.

A decoder either reads out an individual value (on request) or continuously provides the current value.

The decoder connections are provided on the socket `Conn. 2`, pins `SSI CLK`, `SSI DATA`. For pinouts see fig. 4 on page 10.

Pin `SSI DATA` / `DIGIN60` may also be used as digital input to evaluate the SSI input level.

Setting properties

The following properties of the decoders can be set via software:

– Clock rate: With **SSI_Set_Clock**, clock rates of approx. 100kHz up to 2.5MHz are possible.

– Timing: **SSI_Set_Delay** sets the time between reading two encoder values.

– Resolution: Can be set with **SSI_Set_Bits** up to 32 bit.

Example: Conversion of Gray code

A conversion from Gray code into binary code is made with the routine below, which you have programmed in the *ADbasic* process.

```
REM Par_1 = Gray value To be converted
REM Par_2 = Flag indicating a new Gray value
REM Par_9 = Result of the Gray-To-binary conversion

Dim m, n As Long

Event:
  If (Par_2 = 1) Then 'Start of conversion
    m = 0             'initialize value
    Par_9 = 0         ' -"-
    For n = 1 To 32   'Go through all possible 32 bits
      m = (Shift_Right(Par_1,(32-n)) And 1) XOr m
      Par_9 = (Shift_Left(m,(32-n))) Or Par_9
    Next n
  Par_2=0             'Enable next conversion
  EndIf
```

Fig. 9 – Listing: Conversion of Gray code into binary code

Programming

Instructions to program digital channels are described starting from page 128 and in the online help. The instructions are defined in the include file `ADwin-X.inc`.

Function	Instruction
Initialize decoder	SSI_Mode SSI_Set_bits SSI_Set_Clock SSI_Set_Delay
Read encoder values	SSI_Read SSI_Start SSI_Status
Start several functions synchronously.	Sync_All

9 Option DCT

Option X-A20-DCT includes all functions of option D. Option DCT provides:

- 52 digital channels
  - 32 TTL-digital channels DIO31:DIO00, page 21
  - 8 Diff. digital channels DIO47:DIO40, page 18 (see option D)
  - 12 Comparator inputs DIO59:DIO48, page 21
  - Edge control and Edge output for digital channels, page 22
- Counters
  - 2 TTL Counters 2, 3, page 22
  - 2 Diff. counters 4, 5, page 18 (see option D)
  - 2 Comparator Counters 6, 7, page 23
- 1 SSI interface, page 19 (see option D)

9.1 TTL-digital channels DIO31:DIO00

32 digital channels (DIO31:DIO00) are available on D-Sub socket `Conn. 3`, see Fig. 4 – Pin assignment analog/digital (sockets). The channels are programmable as inputs or outputs in groups of 8.

The channels DIO31:DIO26 are double assigned as counter inputs (see TTL Counters 2, 3). Only one of the two purposes (digital channel or counter input) can be used.

The channels are TTL-compatible and not protected against over-current.

Input and output FIFOs enable edge control for digital channels, see Edge control and Edge output on page 22.

Instructions to program digital channels see below (chapter 9.2 and chapter 9.3).

9.2 Comparator inputs DIO59:DIO48

12 digital inputs (DIGIN59:DIGIN48) with comparator function are available on D-Sub socket `Conn. 2`, see Fig. 4 – Pin assignment analog/digital (sockets). All channels are permanently set as inputs and cannot be used as outputs.

The channels DIGIN53:DIGIN48 can also be used as comparator counter inputs, see Comparator Counters 6, 7. Only one of the functions (digital channel or counter input) can be used at a time.

Comparator inputs are internally connected with the analog outputs of the 12-bit DAC, DAC12-1 with DIGIN55:DIGIN48 and DAC12-2 with DIGIN59:DIGIN56. The set DAC voltage (0V…5V) serves as comparator signal, i. e. an applied digital signal with a lower voltage is processed as level Low, with a higher voltage as level High.

The maximum measurement frequency depends on the comparator signal set (at the DAC) and on the voltage of the input signal. For input voltages around +24V, the maximum measurement frequency is less than 30kHz.
For an accurate measurement of the pulse width of the input signal, a measurement frequency well below the maximum must be selected.

The comparator inputs can be combined with edge control for digital channels, see Edge control and Edge output on page 22.

Programming

Instructions to program digital channels are described starting from page 78 and in the online help. The instructions are defined in the include file `ADwin-X.inc`.

Function	Instructions
Configure channels	`Conf_DIO`
Configure input filter	`Digin_Filter_Init`
Read input values.	`Digin`, `Digin_Long2`
Control edges of digital inputs.	`Digin_Edge2`

Set digital outputs.	`Digout`
	`Digout_Long2`
	`Digout_bits2`
	`Get_Digout_Long2`

Function	Instructions
Read and set values via latch register.	Dig_Latch Digin_Read_Latch2 Digout_Write_Latch2
Start several functions synchronously.	Sync_All

9.3 Edge control and Edge output

Option X-A20-D can automatically monitor the edges of digital inputs and independently output edges on digital outputs at determined points in time.

Edge detection and edge output are available for all digital channels of X-A20, not only for channels of option DCT.

There are two input FIFOs and two output FIFOs, each FIFO refers to 32 digital channels (DIO31:DIO00 or DIO60:DIO32).

Edge control

An edge detection checks every 10ns whether a level has been changed at the selected digital inputs or if an edge has has occurred ist. With every change, a value pair is copied to the appropriate input FIFO:

– Value 1 contains the level status of all 32 channels as bit pattern.

– Value 2 is a time stamp, the current value of the 100MHz timer. Each edge detection has its own timer.

Up to 511 of value pairs (level status and time stamp) can be stored in an input FIFO, building an exact logging of all changes. The FIFO data can be read and processed

The edge detections for channels DIO31:DIO00 and DIO60:DIO32 work independently from each other.

As an alternative, you can register edges (without time stamps) of selected input channels with Digin_Edge1/2. If a positive or a negative edge arrives at an input, the appropriate bit of the input channel is set in a buffer. The buffer content can be read at any time. Number and timing of the edges are not stored.

Time-controlled edge output

The edge output can independently output edges at determined points in time on the digital outputs. An output FIFO serves as buffer for the user-defined levels and points in time, at a maximum of 511 value pairs. The point in time can be set with an accuracy of 10ns.

Programming

Instructions to program digital channel FIFOs are described starting from page 97 and in the online help. The instructions are defined in the include file ADwin-X.inc.

Function	Instructions
Use output Fifo.	Digout_Fifo_Read_Timer Digout_Fifo_Clear Digout_Fifo_Enable Digout_Fifo_Empty Digout_Fifo_Mode Digout_Fifo_Start Digout_Fifo_Write
Use input Fifo.	Digin_Fifo_Read_Timer Digin_Fifo_Clear Digin_Fifo_Enable Digin_Fifo_Full Digin_Fifo_Read
Start several functions synchronously.	Sync_All

9.4 TTL Counters 2, 3

Option X-A20-D additionally provides two TTL counters with numbers 2 and 3.

The counter inputs are on the pins DIO31:DIO26 on DSub socket Conn. 3., see fig. 4 on page 10.

The pins DIO31:DIO26 have a double allocation as digital channels, see TTL-digital channels DIO31:DIO00. The channels must be configured as digital inputs with Conf_DIO to enable the usage as counter inputs.

All functions of the counter blocks are described in chapter 15 “Counter block“.

Instructions to program counters are described starting from page 111, overview see below (chapter 9.5).

9.5 Comparator Counters 6, 7

Option X-A20-D additionally provides two comparator counters with numbers 6 and 7.

The counter inputs are on the pins DIGIN53:DIGIN48 on DSub socket `Conn. 2.`, see fig. 4 on page 10.

The pins have a double allocation as digital channels, see Comparator inputs DIO59:DIO48. The channels must be configured as digital inputs with `Conf_DIO` to enable the usage as counter inputs.

The inputs DIGIN53:DIGIN48 of the comparator counters are internally connected with the analog output of the 12-bit DAC DAC12-1 (see also Comparator inputs DIO59:DIO48). The set DAC voltage (0V…5V) serves as comparator signal, i.e. an applied digital signal with a smaller voltage is processed as level Low, with a higher voltage as level High.

All functions of counter blocks 4 or 5 are described in chapter 15 “Counter block“.

Programming

Instructions to program counters are described starting from page 111 and in the online help. The instructions are defined in the include file `ADwin-X.inc`.

Function	Instructions
Configure channels as inputs.	`Conf_DIO`
Clear counter.	`Cnt_Clear`
Disable or enable counter (please note already running counters).	`Cnt_Enable` `Cnt_PW_Enable`
Read out status register.	`Cnt_Get_Status`
Write counter value into Latch A.	`Cnt_Latch`
Write counter value into latch A and read the latch value.	`Cnt_Sync_Latch`
Set counter operation mode.	`Cnt_Mode`
Read latch value.	`Cnt_Read`
Return the content of a counter register.	`Cnt_Read_Latch`
Set counter mode to single ended / differential inputs.	`Cnt_Read_Int_Register`
Write PWM counter values into latch A.	`Cnt_PW_Latch`
Read frequency and duty cycle of a PWM counter.	`Cnt_Get_PW`
Read high time and low time of a PWM counter.	`Cnt_Get_PW_HL`
Start several functions synchronously.	`Sync_All`

10 Option COM

Option X-A20-COM provides additional interfaces:

– 2 CAN interfaces "High-speed", page 24.
– 1 RS232 interface, page 25.

10.1 CAN interfaces

The CAN interfaces 1 and 2 ("High-speed") run independently from each other.

10.1.1 Hardware description

The connectors of the CAN interfaces are on 9-pole DSub plugs CAN1 and CAN2, same assignment.

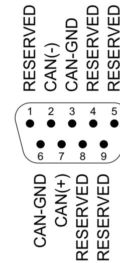


Fig. 10 – CAN: Pin assignment

Both interfaces have their individual CAN-GND potentials; the potentials are both galvanically isolated from each other as well as from the mass potential (GND) of the enclosure.

If the CAN interface functions as the physical termination of a high-speed CAN bus, it must be terminated with a 120Ω resistor (only the first or the last CAN node). CAN nodes, which are not positioned in an end-location, must not be terminated.

If termination is required for one (or both) interfaces, the pins CAN(+) and CAN(-) must be connected by a resistor of 120Ω.

10.1.2 Description of the CAN interface

The CAN bus interface works according to the specification CAN 2.0 parts A and B as well as to ISO 11898. You program the interface with *ADbasic* instructions, which are directly accessing the controller's registers.

Messages sent via CAN bus are data telegrams with up to 8 bytes, which are characterized by so-called identifiers. The CAN controller supports identifiers with a length of 11 bit and 29 bit. The communication, that means the management of bus messages, is effected by an input FIFO and an output FIFO.

The CAN bus (high speed) can be set to frequencies up to 1MHz and is normally run with 1MHz. The CAN bus is galvanically isolated from the *ADwin* system by optocouples.

Managing messages

The CAN controller distinguishes sent messages by the identifier, i.e. a code number of a defined bit length. The bit length determines the range of possible identifiers 0...2^{11}-1 and 0...2^{29}-1.

Input / output FIFO

The controller stores messages to be sent in an output FIFO, and received messages in an input FIFO. After initialization of the CAN controller, both FIFOs are not configured and are not active on the bus.

Additionally, messages can be sent with high priority. If so, messages in the output FIFO are postponed.

In *ADbasic*, you get a CAN message after receipt from the input FIFO via the **can_msg**[] array. The array contains up to 8 data bytes, the amount of data bytes, the identifier, and (only receiving) a receipt time stamp (11 elements). While sending, a message is also transferred via the **can_msg**[] array.

Sending a message

Sending a message is done as follows:

– Save the message and identifier in the `can_msg[]` array.

– Transfer the `can_msg[]` array with `CAN_Transmit` as message to the CAN interface. As soon as the bus is ready, the message is sent.
    - With normal priority, the interface stores the message in the output FIFO. The message are sent in the order as they were stored.
    - With high priority, the message is sent as soon as the CAN interface has access to the CAN bus. Messages waiting in the output FIFO to be sent are postponed.

Receiving a message

Receiving a message is done as follows:

– Set the receive filter to selected identifiers (`CAN_RX_Set_Filter`). If you do not set the receive filter, all CAN messages are accepted.

– The CAN interface checks the bus for incoming messages, filters them according to the receive filter and stores them in the input FIFO.

– You read the message from the input FIFO via the `can_msg[]` array (with `CAN_Receive`) and read the appropriate identifier.

The input FIFO can store up to 64 CAN messages. If the FIFO is full, any incoming message overwrites old data, which will be definitely lost. Therefore, pay attention to reading out data faster than you are receiving them. A data loss is indicated by a flag.

Filtering incoming messages

With `CAN_RX_Set_Filter`, you can set up to four receive filters for incoming messages. Filters cn be enabled and disabled individually. The identifier of an incoming message is compared to the active receive filters and then accepted or refused:

– If the identifier of the message is equal to the receive filter, the message is stored in th einput FIFO.

– As soon as a message has successfully passed an active receive filter it is stored in the receive FIFO.

Setting the bus frequency

The **CAN bus frequency** depends on the controller configuration.

The initilization with `CAN_Init` configures the CAN bus frequency. In some cases, different settings may be useful than available with `CAN_Init`. If so, please refer to out support.

Programming

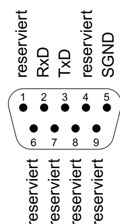
Instructions to program counters are described starting from page 136 and in the online help. The instructions are defined in the include file `ADwin-X.inc`.

Function	Instructions
Initialization	`CAN_Init`

Receving and sending data	`can_msg[]`
	`CAN_Receive`
	`CAN_RX_Set_Filter`
	`CAN_Transmit`

10.2 RS232 interface

Die RS232 interface runs without handshake. The connectors of the CAN interfaces are on the DSub plug `RS232`.



The interface provides an input FIFO and an output FIFO with a length of each 64 byte. The following interface parameters can be set:

– Parity: In order to recognize an error or incorrect data during the transfer, a parity bit can be transferred at the same time. The parity can be even or odd or you can have no parity bit at all.

– Data bits: the active data to be transferred may be 6...8 bits long.

– Stop bits: The number of stop bits can be set to 1, 1½ or 2.

– Baud rate: The physical data are between 35 Baud and 115.2 kBaud. Typical baud rates are 300, 600, 1200, 2400, …, 115200 Baud.

Programming

Instructions to program counters are described starting from page 147 and in the online help. The instructions are defined in the include file `ADwin-X.inc`.

Function	Instructions
Initialization	RS_Init
Receving and sending data	RS_Read_FIFO RS_Write_FIFO RS_Write_FIFO_Full RS_Write_FIFO_Empty

11 Option Profibus

The add-on X-A20-Profibus provides a fieldbus node with the functionality of a Profibus slave. All settings are done via software.

The option cannot be combined with Option Profinet-IRT or Option EtherCAT.

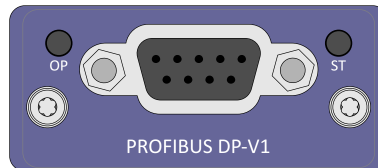
Functions description

After power-on the fieldbus node must be initialized. The initialization determines station address (slave node address) on the Profibus and the size of the input area and the output area.

There is a range each for data input and data output; the size of each range can be set individually, i.e. a size of 1, 2, 4, 8, 16, 32, or 61 double words.

Hardware

The pin assignment of the 9-pin DSUB connector refers to DIN E 19245, part 3.



The Profibus has to be terminated at its physical beginning and at the end of its segments by an active terminator. If required, you have to add the terminator yourself at the appropriate data lines of the fieldbus node or use an appropriate connector with integrated terminator.

Besides the DSUB connector, there are 2 LEDs, which display the operation status of the fieldbus node: operation mode (OP) and interface status (ST).

LED	Status	Meaning
OP	off	Offline or no power.
	green	Fieldbus node online, data exchange.
	flashing green	Fieldbus node online, status clear.
	flashing red, 1 flash	Error: Input/output configuration does not fit to master configuration.
	flashing red, 2 flashes	Error in Profibus configuration.
ST	off	Offline or no power.
	green	initialized.
	flashing green	initialized, diagnostiv event(s) present.
	red	Exception error.

Fig. 11 – Profibus: Meaning of LEDs

Projecting the Profibus

You are projecting the Profibus bus with a configuration tool suitable for the bus master. The following process description uses a Profibus master of the Siemens company and the appropriate program `SIMATIC-Manager`.

The process description is valid for other configuration tools, correspondingly. Look for the exact process description of bus projection in the documentation of the configuration tool.

Install the GSD file

– In the program `SIMATIC-Manager`, install the GSD file `hmsb1815.gsd` of the fieldbus node from `C:\ADwin\Fieldbus\Profibus`.

  The configuration tool loads all required information about the new slave from the GSD file; the file content is determined by EN 50170. Afterwards, the slave appears in the profile tree and can be accessed.

– Set the station address of the slave in `SIMATIC-Manager` to the same value as in *ADbasic* with **`Init_Profibus`**.

Configure the Slave

– Configure the size of of the data ranges for input data and output data one by one.

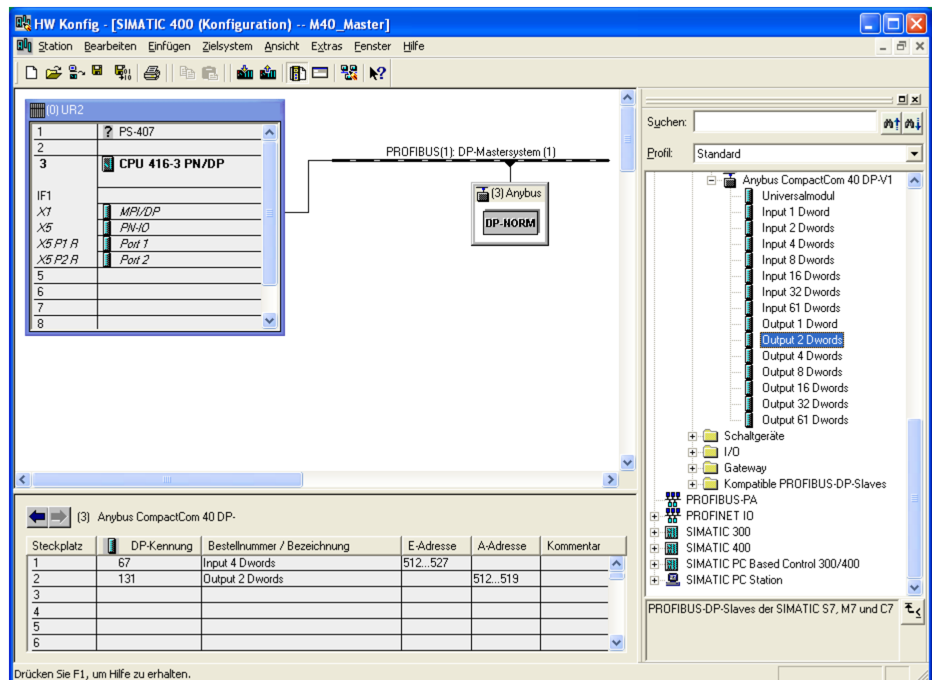
Please note the following rules:

- The terms "input" and "output" have reverse meanings in *ADbasic* (slave) and in the configuration tool (master).
  If the input size is initialized in *ADbasic*, you have to configure the output size correspondingly in the configuration tool.
- Size of data ranges must correspond to the values used for *ADbasic* initialization with `Init_Profibus`.
- The data range size for input and output can be set indidually. Data range can be configured in one of the following sizes: 1, 2, 4, 8, 16, 32, 61 double words.

The following example line in *ADbasic* configures the slave with station address 5, input size 2 double words, output size 4 double words:

```
Par_31 = Init_Profibus(5, 2, 4, conf_Arr)
```

To configure the slave correctly in the configuration tool, you have to set the input first to 4 double words and then the output to 2 double words. The graphic shows the example configuration:



Programming with *ADbasic*

The Profibus interface is programmed with *ADbasic* instructions which are described starting from page 153 and in the online help. The instructions are defined in the include file `ADwin-X.inc`.

Area	Instructions
Reset, initialize data ranges	`Init_Profibus`
Read and write data, message handling	`Run_Profibus`

Initialization must be run with low priority since it takes some time; if using high priority, the PC would stop communication after a time-out. On the other hand, reading and writing data may happen with high priority.

Specification

The fieldbus node is in agreement with the European Standard EN 50170, Volume 2. This norm is provided by the Profibus user organization:

Profibus Nutzerorganisation e.V.
Haid-und-Neu-Str. 7
76131 Karlsruhe, Germany
Phone: +49-72196-58590
Fax : +49-72196-58589
Order number: 0.042

Operating modes of fieldbus node

The following table shows the operating modes, the fieldbus node supports and its behavior:

Operating mode	Behavior
Operate	The Profibus slave is part of the cyclic data exchange. Input data are transferred to the master via bus and output data are made ready for the master to transfer them.
Clear	The inputs are updated and the outputs are set to zero.
Stop	The slave is no longer part of the bus communication.

Fig. 12 – Profibus: Operating modes

12 Option Profinet-IRT

The add-on X-A20-Profinet-IO provides a fieldbus node with the functionality of a Profinet IRT slave. All settings are done via software.

The option is available with different connectors:

– Profinet-IRT-Cu: Interface with copper cable, 2 sockets RJ-45, customary connectors.

– Profinet-IRT-FO: Interface with optical fiber, 2 duplex sockets SC-RJ (fiber optics).

The option cannot be combined with Option Profibus or Option EtherCAT.

Functions description

After power-on the fieldbus node must be initialized. The initialization determines the size of the input and output areas.

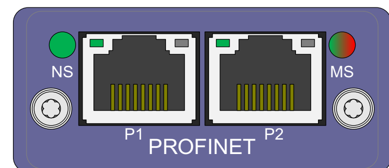
There is a range each for data input and data output; each range has a maximum size of 1280 bytes. Please note, that the terms "input" and "output" are used as the fieldbus master sees them.

During initialization, you set the number and size of input and output areas separately. Nevertheless, during operation only one size can be used.

Hardware

The connectors are connected to standard plugs:

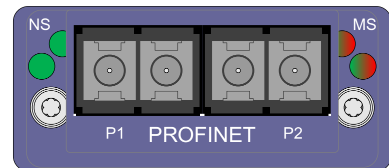
– Ethernet plugs RJ-45 (IRT-Cu)



Left and right of the RJ-45 connectors, there are two LEDs, which display the operation status of the profinet node: network status (NS) and interface status (MS).

In each connector there is a LINK LED.

– Fiber duplex plugs SC-RJ (IRT-FO)



Left and right of the connectors, there are four LEDs. The half concealed LEDs display the operation status of the profinet node: network status (NS) and interface status (MS).

The LEDs more outside display the LINK status for the plug.

LED	Status	Meaning
NS	off	Offline: No power or no connection with IO controller.
	green	Online (RUN): Field bus node online, IO Controller in RUN state.
	green, 1 flash	Online (STOP): Field bus node online, IO Controller in STOP state or IO data bad. IRT synchronization not finished.
	green, blinking	Blink: Used by engineering tools to identify the node on the network.
	red	Fatal event: Major internal error. Indication is combined with a red MS LED.
	red, 1 flash	Station name error: Station name is not set.
	red, 2 flashes	IP address error: IP address is not set.
	red, 3 flashes	Configuration error: Expected identification differs from real identification.
MS	off	Not initialized: No power or module in SETUP or NW_INIT state.
	green	Normal operation.
	green, 1 flash	Diagnostic events present.
	red	Device in state EXCEPTION.
	red	Fatal event: Major internal error. Indication is combined with a red NS LED.
	red / green alternating	Firmware update. Do not power off the device. Turning the device off during this phase could cause permanent damage.
LINK	off	No link.
	green	Ethernet link established, no communication.
	green flickering	Ethernet link established, communication present.

Fig. 13 – Profinet: Meaning of LEDs

Projecting the Profinet

You are projecting the Profinet bus with a configuration tool suitable for the bus master. The following process description uses a Profinet master of the Siemens company and the appropriate program `SIMATIC-Manager`.

The process description is valid for other configuration tools, correspondingly. Look for the exact process description of bus projection in the documentation of the configuration tool.

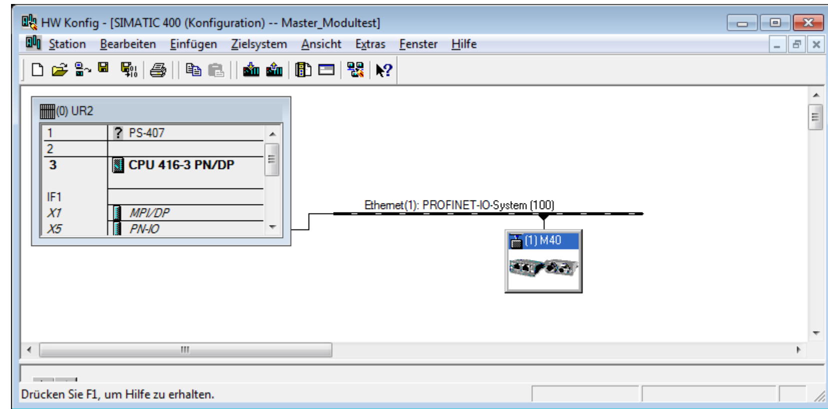
Install the GSD file

– In the program `SIMATIC-Manager`, install the GSD file `GSDML-V2.33-HMS-ABCC40-PIR-ADwin-20180620.xml` of the fieldbus node from `C:\ADwin\Fieldbus\Profinet`.

  The configuration tool loads all required information about the new slave from the appropriate XML file; the file content is determined by EN 50170. Afterwards, the slave can be accessed by any master.

– In the configuration tool, add the Slave, i.e. the fieldbus node to the Profinet.

  Afterwards the bus could be structured as below:



Configure the Slave

– Configure number and length of input and output data of input and output data in the fieldbus node memory one by one.

Please note the following rules:

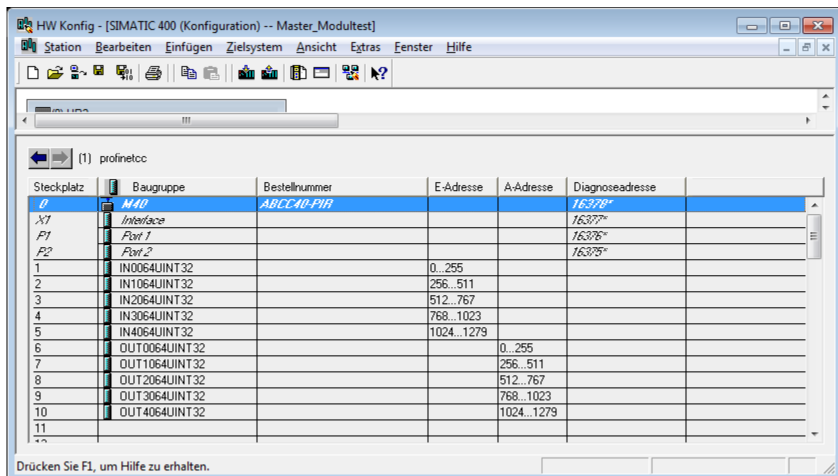
- The terms "input" and "output" have reverse meanings in *ADbasic* (slave) and in the configuration tool (master).
  If there are inputs initialized in *ADbasic*, you have to configure outputs as correspondent in the configuration tool.
- Number and length of data ranges must equal the data used for *ADbasic* initialization with `Init_ProfinetIO`.
- The data range sizes for input and output can be set indidually. Data ranges can be configured in on e of the following sizes (1 double word = 4 byte): 1, 2, 4, … 64 double words; 64, 128, … 320 double words.
  The blocks of 64 double words are numbered and may only be configured in ascending order IN0…IN5 / OUT0…OUT5.

Example configuration

The following example line in *ADbasic* configures the slave with an input range and an output range of 320 double words each (=1280 bytes):

```
Init_ProfinetIO(320, 320, work_arr)
```

To configure the slave correctly in the configuration tool, you have to set 5 blocks of 256 bytes (=64 double words) in both input range and output range. Please note the required order of block configuration:



Programming with *ADbasic*

The Profinet interface is programmed with *ADbasic* instructions which are described starting from page 190 and in the online help. The instructions are defined in the include file `ADwin-X.inc`.

Area	Instructions
Reset, initialize data ranges	`Init_ProfinetIO`
Read and write data, message handling	`Run_ProfinetIO`

Specifications

The fieldbus node is in agreement with the Standard IEC 61158 (Profinet). This norm is provided by the Profibus user organization:

ProfibusNutzerorganisation e.V.
Haid-und-Neu-Str. 7
76131 Karlsruhe, Germany
Phone: +49-72196-58590
Fax : +49-72196-58589
www.profibus.com

The following table shows the operating modes, the fieldbus node supports and its behavior:

Operating mode	Behavior
Setup	Interface initialization.
Wait	Slave waits for bus start by master.
Active	Profinet slave is part of the cyclic data exchange.
Error	Error. Profinet slave is not part of the cyclic data exchange.
Exception	Major internal error. Profinet slave is not part of the cyclic data exchange.

Fig. 14 – Profinet: Operating modes

13 Option EtherCAT

The add-on X-A20-EtherCAT provides a fieldbus node with the functionality of an EtherCAT slave. All settings are done via software.

The option cannot be combined with Option Profibus or Option Profinet-IRT.

Functions description

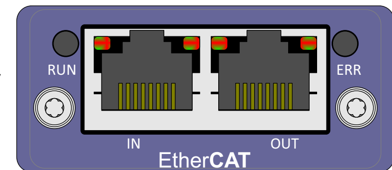
After power-on the fieldbus node must be initialized in *ADbasic*. The initialization determines the size of the input and output areas.

There is a range each for data input and data output; each range has a maximum size of 1440 bytes. Please note, that the terms "input" and "output" are used as the fieldbus master sees them.

During initialization, you set the number and size of data ranges in the input area and the output area separately. Nevertheless, during operation only one size can be used.

Hardware

The interface has two standard Ethernet plugs RJ-45 marked with `IN` and `OUT`.



Inside each plug there are two LEDs; the upper left is named `Link / Activity` and displays the node operation status on the EtherCAT bus. The upper right LED has no function.

Left and right of the RJ-45 connectors, there are two LEDs, which display the EtherCAT status (`RUN`) and the occurrence of communication errors (`ERR`).

LED	Status	Bedeutung
`Link / Activity`	off	Offline (or no power).
	green	Fieldbus node online, no data exchange.
	green, flickering	Fieldbus node online, with data exchange.
`RUN`	off	Status INIT: interface being initialized (or no power).
	blinks green	Status PRE-OP: Interface has contact to bus master.
	flashes green once	Status SAFE-OP: Interface can read data from the bus, but not send.
	green	Status OP: Interface is completely ready, inputs and outputs are active.
	red	Status EXCEPTION.
`ERR`	off	No error (or no power).
	blinks red	Invalid configuration.
	flashes red once	Local error in the interface; EtherCAT status has been changed.
	flashes red twice	Application watchdog time-out.
	red	Critical communication error.

Fig. 15 – EtherCAT: Bedeutung der LED

If both LEDs `RUN` and `ERR` turn red, a fatal event has occurred in the interface. Please do refer to the supoort of Jäger Messtechnik; you find the address on the inner side of the cover page of the manual.

Projecting the EtherCAT bus

You are projecting the EtherCAT bus with a configuration tool suitable for the bus master. The following process description uses the program "TwinCAT System Manager" of the Beckhoff company (version 3.1) as EtherCAT bus master.

The process description is valid for other configuration tools, correspondingly. Look for the exact process description of bus projection in the documentation of the configuration tool.

– Configure the *ADwin*-EtherCAT slave in an *ADbasic* program using the instruction `Init_EtherCAT`.

– Copy the description file `HMS CompactCom 40 EtherCAT 2_08.xml` of the fieldbus node from `C:\ADwin\Fieldbus\EtherCAT` into the root directory of the configuration tool.

  Upon start-up, the configuration tool loads the required information about the new slave from the appropriate description file.

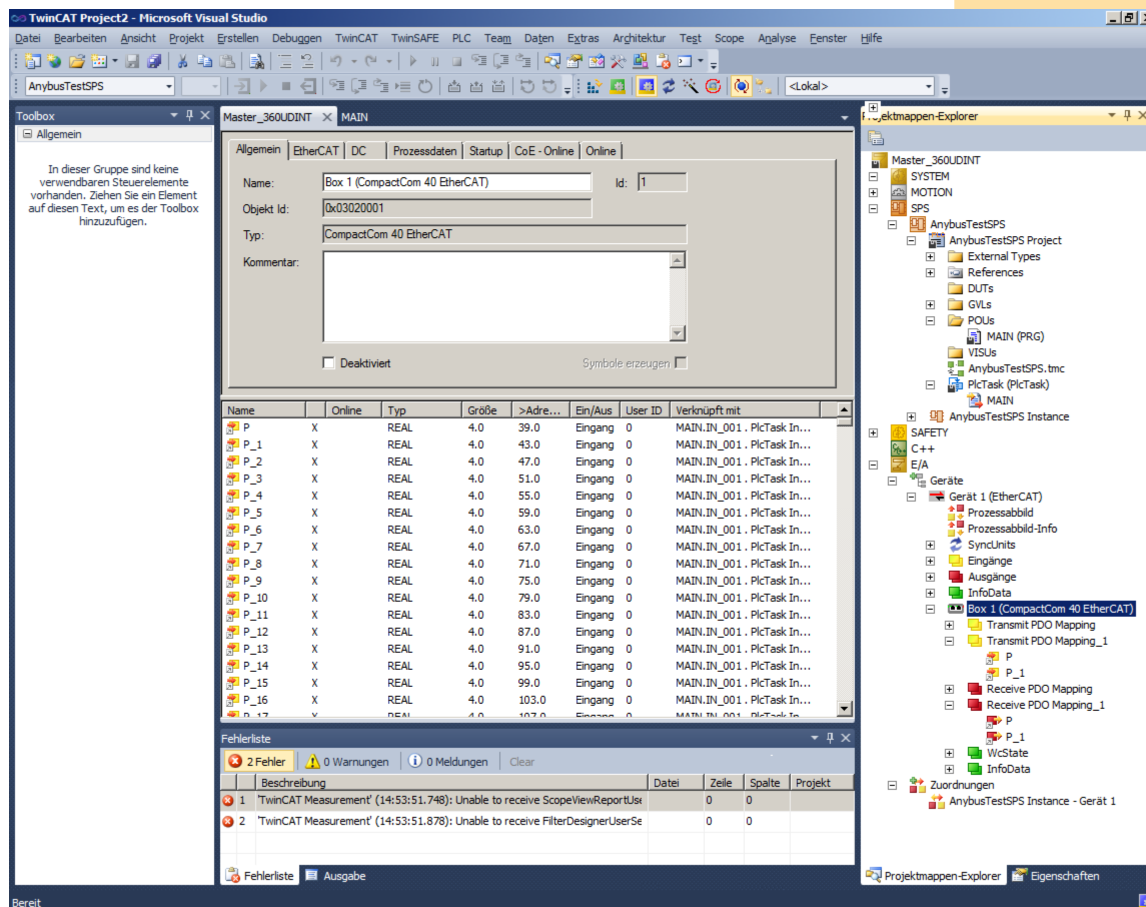
– Add the *ADwin*-EtherCAT slave as bus member to the EtherCAT bus.

  Using TwinCAT System Manager, you mark the EtherCAT master and select the menu entry `Scan` from the context menu (right mouse click).
  A list of all current bus members will be displayed.

– Select the *ADwin*-EtherCAT slave from the list; now the slave is confirmed as bus member.

– Read the configuration into the configuration tool.

  Using the TwinCAT System Manager, you mark the *ADwin*-EtherCAT slave and click the button `Load PDO Info from the device`.



– Now the bus projecting is complete and the module is ready to run.

Programming with *ADbasic*

The fieldbus node is easily programmed with *ADbasic* instructions from `ADwin-X.inc`; description see page 160 or in the online help:

Area	Instructions
Reset, initialize data ranges	`Init_EtherCAT`
Read and write data, message control	`Run_EtherCAT`

Specifications

The fieldbus node is in agreement with the international standard IEC 61158 and IEC 61784-2. More information is provided by the EtherCAT user organization:

EtherCAT Technology Group
Ostendstraße 196
D-90482 Nürnberg
Tel.: +49 9115405620
Fax : +49 9115405629
http://www.ethercat.org/

Operating modes of EtherCAT node

The following table shows the operating modes, the EtherCAT node supports and its behavior:

Operating mode	Behavior
Init	The EtherCAT slave is being initialized by the bus master.
Boot	The EtherCAT slave is in boot mode.
PreOp	The interface is part of the data exchange, inputs and outputs are not active.
SafeOp	The interface can receive data, outputs are not active.
Op	The interface is completely ready; inputs and outputs are active.

Fig. 16 – EtherCAT: Operating modes

14 Option Boot

ADwin-X-A20-Boot starts a previously programmed application automatically after power-up. After installation of this application an operation without computer is possible.

You program the bootloader in the development environment *ADbasic*, menu entry `Tools` / `Bootlader`.

With *ADwin-X-A20*-Boot, the following steps are executed after power-up:

– Loading the operating system.

– Loading of the compiled processes, compiled by *ADbasic* (max. 10).

– Automatic starting of the process no. 10. Here you have also to program the start of all other processes.

If you do not wish to work with the bootloader option:

– Disable temporarily:
  - Switch on *ADwin-X-A20*.
  - Boot the system from *ADbasic*. The previously saved processes are disabled.
  - After another power-up, the bootloader option is enabled again.

– Disable permanently:
  - Disable the bootloader in *ADbasic*, menu entry `Tools` / `Bootlader`, Tab `Enable`/`Disable`.
  - Switch off *ADwin-X-A20* and power-up again.

ADwin-X-A20-Boot does not provide an EEPROM (unlike other *ADwin* hardware).

15 Counter block

Each of the 7 counters in *ADwin-X-A20* is designed as counter block. In a counter bloc, there are two 32-bit counters running in parallel and independently:

– One up/down counter with clock/direction evaluation or four edge evaluation for quadrature encoders.

– One PWM counter for measurement of frequency and duty cycle or high time / low time.

Counters are configured via software, counter data are provided in latches to be read.

An *ADwin-X-A20* can be equipped with up to 7 counters in total. The counter numbers are assigned to the following options:

– Counter 1 (TTL): Option CO1, page 17

– Counters 2…3 (TTL): Option DCT, page 22

– Counters 4…5 (differential): Option D / Option DCT, page 18

– Counters 6…7 (comparator): Option DCT, page 23

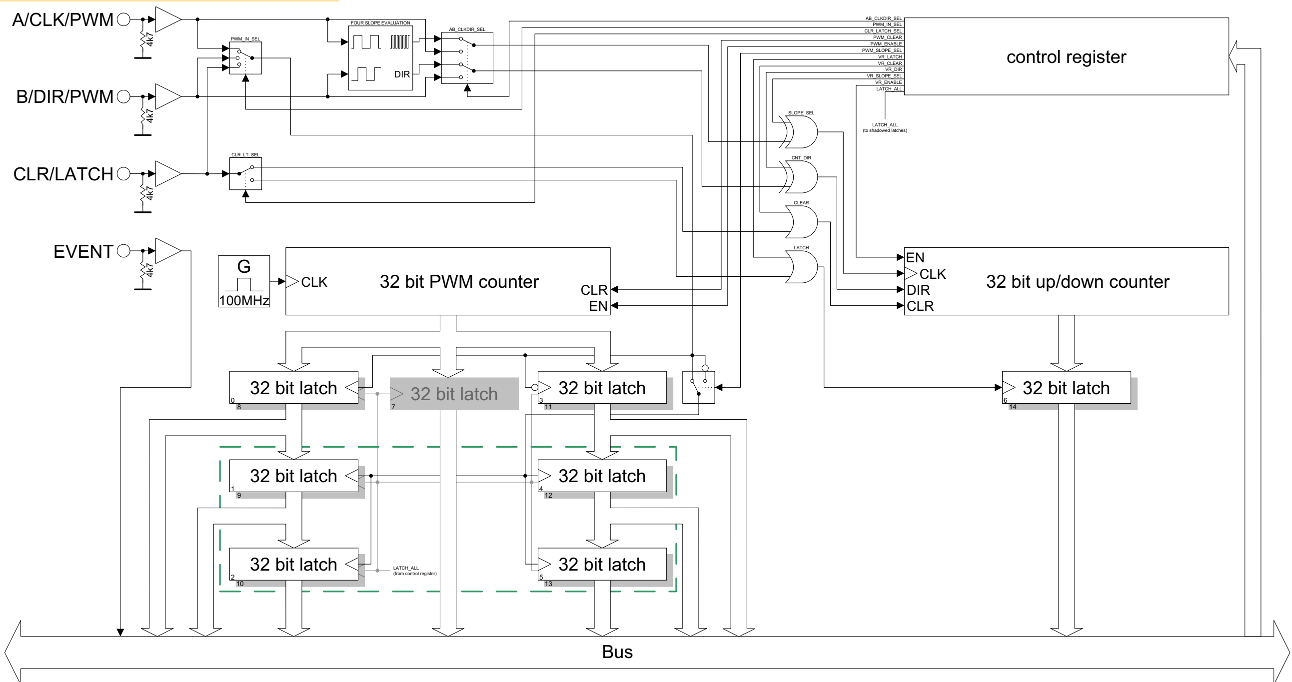


Fig. 17 – Block diagram of a counter block

Up/down counter

With event counting, incrementing/decrementing of the counter is caused by external square-wave signals at the inputs A/CLK and B/DIR.
A positive edge at CLR/LATCH either sets the counter to zero (CLR) or copies the counter values into the latch (LATCH). See also chapter 13.2.

The following modes are possible:

1. **Clock and direction**: A positive edge at CLK increments or decrements the counter values by one. The signal at DIR determines the counting direction (0 = decrement; 1 = increment).
2. **Four edge evaluation (A/B)**: Every edge of the signals (phase-shifted by 90 degrees) at A/CLK and B/DIR causes the counter to increment/decrement. The counting direction is determined by the sequence of the rising/falling edges of these signals. This mode is particularly used for quadrature encoders.

You can invert the signals at the inputs A/CLK and B/DIR via software (instruction `Cnt_Mode`) and thus change both the triggering signal and the counting direction.

PWM counter

For pulse width measurement, incrementing/decrementing of the counter is caused by an internal reference clock generator; a signal frequency of 100 MHz can be used. See also chapter 13.3.

The counter value is written into a latch register if an edge–at one's option positive or negative–occurs at the selected input (A/CLK, B/DIR or CLR/LATCH). Latching can be triggered by software, too.

From the latch register, frequency and duty cycle or high time / low time of the PWM signal can be read.

Input signals

The counters are controlled by *ADbasic* instructions via control register (instructions see below).

At the inputs A/CLK, B/DIR and CLR/LATCH TTL-alike signals are necessary.

Although all counter inputs have a pull-down resistor of 10 kΩ, not-connected inputs can cause errors in an environment, which is not protected against interferences. If you do not use a counter input, connect both lines of the (differential) input to a specified potential for safety reasons: Connect the positive input to +5 V and the negative input to GND.

Programming counters

The functions for counter access be found in the include files `ADwin-X.inc` for *ADbasic*.

Therefore, programming has to start with the include file, so that you can use the instructions in the following table. The instructions are described in chapter 7.2, starting from page 111.

Instruction	Function
`Cnt_Clear`	Clear counter.
`Cnt_Enable` `Cnt_PW_Enable`	Disable or enable counter (please note already running counters).
`Cnt_Get_Status`	Read out status register.
`Cnt_Latch`	Write counter value into Latch A.
`Cnt_Sync_Latch`	Write all counter values into latches at the same time.
`Cnt_Mode`	Set counter operation mode.
`Cnt_Read`	Write counter value into latch A and read the latch value.
`Cnt_Read_Latch`	Read latch value.
`Cnt_Read_Int_Register`	Return the content of a counter register.
`Cnt_PW_Latch`	Write PWM counter values into latch A.
`Cnt_Get_PW`	Read frequency and duty cycle of a PWM counter.
`Cnt_Get_PW_HL`	Read high time and low time of a PWM counter.
`Sync_All`	Start several functions synchronously.

Fig. 18 – Instructions for a counter block

Mostly, the instructions are effecting all counters. Therefore, pay attention to the fact, which bits you are setting or deleting. You will be able to effect every counter individually or all together.

Sequence of instructions

Please configure the counters according to the following order:

1. Disable specified counter (`Cnt_Enable`)
2. Set operating mode (`Cnt_Mode`)
3. Clear counter (`Cnt_Clear`)
4. Enable counter (`Cnt_Enable`)

For further processing of the values in the *ADbasic* program, transfer the values into the latch register and read them out there.

If you disable or enable a specified counter, then you also enable the running counters (= set bits). If you do not set the bits of these counters (unintentionally), they will be disabled.

15.1 Evaluation of the Counter Contents

The binary counters generate 32-bit values, which are interpreted by *ADbasic* as numerical values according to the model of the circle below: The most significant bit (MSB) is interpreted as a sign, the highest positive number (2^{31}-1) follows the highest negative number (-2^{31}) and the lowest positive number (0) follows the highest negative number (-1).

Circle

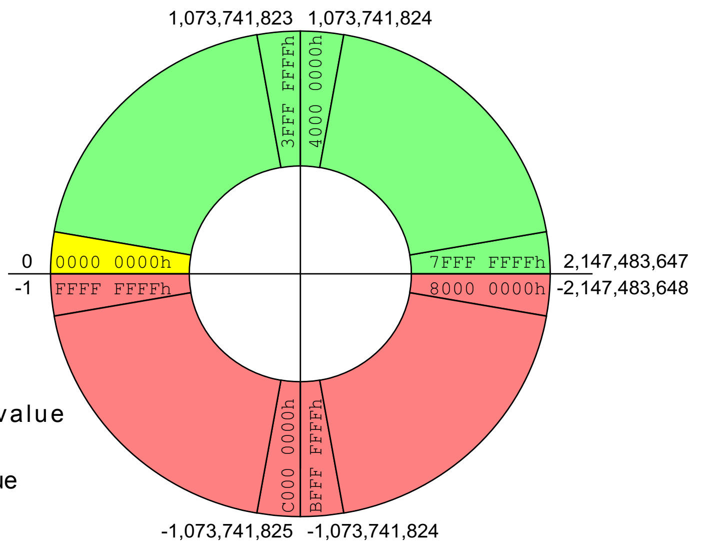


Fig. 19 – Circle for the interpretation of counter values

Please pay attention to the following rules for programming:

a) Process the read 32-bit value only with variables of the type LONG. *ADbasic* then keeps internally the read bit pattern unmodified and automatically considers the transition from the positive to the negative range of numbers. Then you get:

Count direction

b) The count direction (up or down) can reliably be derived from the Sign of the difference: [new counter value] minus [old counter value] and not from the comparison of the counter values.

"Overflow"

For programming please remember that an "overflow" between the reading out of two counts - i.e. the current counter value "laps" the last counter value, which has been read out - is not registered. Such a lap overflow occurs after some 42 seconds with an input frequency of 100 MHz.

15.2 Using Event Counter

External square-wave signals at the inputs A/CLK and B/DIR clock the counters in this mode.

The input CLR/LATCH (at high-signal) can be used to

– clear the counter (CLR)

– latch the counter values into latch register A (LATCH).

15.2.1 Clock and Direction

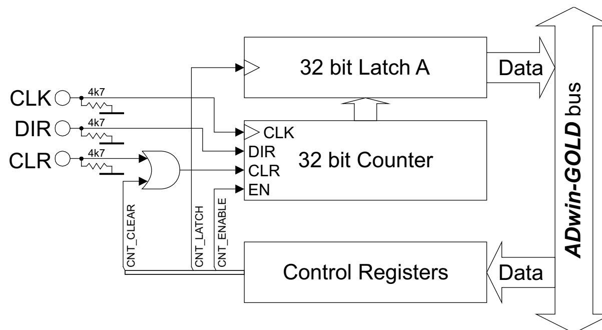


Fig. 20 – Block diagram in the mode "clock and direction"

Every positive edge of a square-wave signal at the CLK input (clock) is counted (incremented or decremented) up to a maximum frequency of 20 MHz. The direction is derived from a high signal (count up) or low signal (count down) at the DIR input (direction); This signal can be static, for a fixed count direction, or dynamic, for changing directions.

The signals at the inputs A/CLK and B/DIR can be (individually) inverted with **Cnt_Mode**.

Programming example

```
#Include ADwin-X.inc
Dim val As Long
Init:
  …
  Cnt_Enable(0)              'stop all counters
  Cnt_Clear(0001b)           'clear counter 1
  Rem set operation mode of counter 1:
  Rem Bit 0: Mode clock/direction
  Rem Bit 1: Clear mode with CLR input
  Rem Bit 2: do not invert input A/CLK
  Rem Bit 3: do not invert input B/DIR
  Rem Bit 4: set input CLR/LATCH as CLR input
  Rem Bit 5: enable input CLR/LATCH
  Cnt_Mode(1,100000b)
  Cnt_SE_Diff(0000b)         'all inputs single-ended
  Cnt_Enable(0001b)          'start counter 1
  …
Event:
  …
  Cnt_Latch(0001b)           'latch counter 1
  val = Cnt_Read_Latch(0001b)  'read latch value
```

15.2.2 Four Edge Evaluation

This mode determines clock and direction of two signals, which are phase-shifted by 90 degrees to the inputs A and B. The count direction is determined by the temporal sequence of the rising and falling edges of the two input signals.

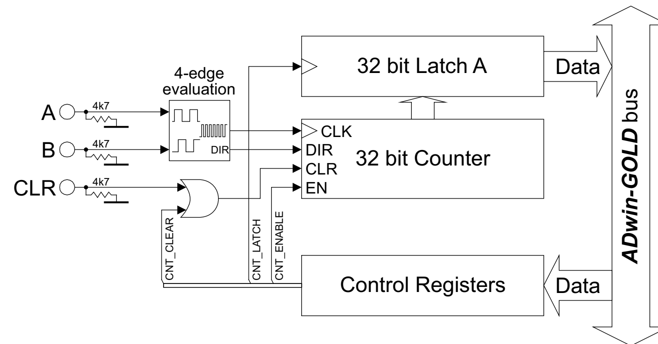


Fig. 21 – Block diagram in mode "four edge evaluation"

Please note:

– The counter counts 4 edges in one cycle of the A/B signal.

– The maximum count frequency is 20 MHz. Together with the 4 edges per cycle it will result in a maximum input frequency of 5.0 MHz.

– The time between an edge at A and an edge at B must not be shorter than 50 ns. Impulse widths or pause durations shorter than 100 ns are not incremented.

– Changing the phase-shift will have an effect on the maximum input frequency. If it differs from 90 degrees, the maximum input frequency of 5.0 MHz decreases for instance to 45 degrees at 2.5 MHz.

Programming example

```
#Include ADwin-X.inc
Dim val As Long
Init:
  …
  Cnt_Enable(0)              'stop all counters
  Cnt_Clear(0001b)           'clear counter 1
  Rem set operation mode of counter 1:
  Rem Bit 0: Mode four edge evaluation
  Rem Bit 1: Clear mode with CLR input
  Rem Bit 2: do not invert input A/CLK
  Rem Bit 3: do not invert input B/DIR
  Rem Bit 4: set input CLR/LATCH as CLR input
  Rem Bit 5: enable input CLR/LATCH
  Cnt_Mode(1,100000b)
  Cnt_SE_Diff(1111b)         'all inputs differential
  Cnt_Enable(0001b)          'start counter 1
  …
Event:
  …
  Cnt_Latch(0001b)           'latch counter 1
  val = Cnt_Read_Latch(0001b)  'read latch value
```

15.3 Using PWM Counter

Reference clock generator

In this operating mode, an internal reference clock generator clocks the counter with a signal frequency of 100 MHz. The frequency and duty cycle can be read as well as high time and low time.

Example

```
#Include ADwin-X.inc
#Define frequency FPAR_1
#Define dutycycle FPAR_2
#Define hightime PAR_1
#Define lowtime PAR_2
Init:
  …
  Cnt_PW_Enable(0)          'stop all counters
  Rem set operation mode of counter 1:
  Rem Bits 0..5: no importance
  Rem Bit 6: detect rising edge as PWM signal
  Rem Bit 7: input B/DIR as PWM input
  Cnt_Mode(1,010000000b)
  Cnt_SE_Diff(1111b)        'all inputs differential
  Cnt_PW_Enable(000000001b) 'start PWM counter 1
  …
Event:
  …
  Rem latch counter 1
  Cnt_PW_Latch(0001b)
  Rem read frequency and duty cycle
  Cnt_Get_PW(1,frequency,dutycycle)
  Rem read high time and low time
  Cnt_Get_PW_HL(1,hightime,lowtime)
```

Exception: evaluate PWM registers on your own

There are several registers assigned to each PWM counter being described below. If, like in the example above, PWM counters are evaluated with standard instructions `Cnt_Get_PW` and `Cnt_Get_PW_HL`, no further knowledge is required about PWM registers. Use the evaluation with PWM registers for special solutions only.

In order to evaluate PWM signals, the counter values of the current and the 2 preceding counter values are stored in latch registers, both for rising and falling edges. In addition, there is a "shadow register" for each of these 6 registers.

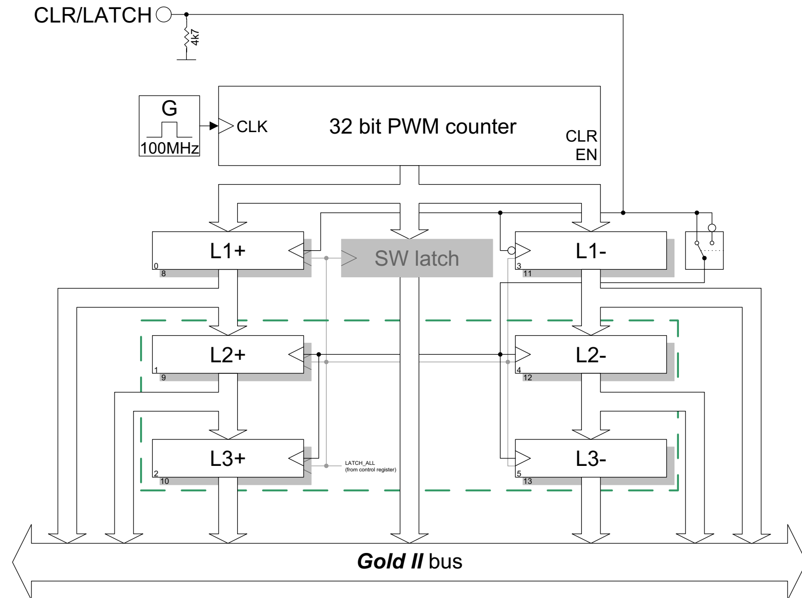
Register	Latch	Shadow register
Latch 1 for positive edges (current)	L1+	SL1+
Latch 2 for positive edges	L2+	SL2+
Latch 3 for positive edges	L3+	SL3+
Latch 1 for negative edges (current)	L1–	SL1–
Latch 2 for negative edges	L2–	SL2–
Latch 3 for negative edges	L3–	SL3–

The register values are changed with any edge like this:

- Rising edge:
  - Copy counter value to L1+
  - If rising edge is set as reference edge:
    Copy register L2+ to L3+
    Copy register L1+ to L2+
    Copy register L2– to L3–
    Copy register L1– to L2–
- Falling edge:
  - Copy counter value to L1–
  - If falling edge is set as reference edge:
    Copy register L2– to L3–
    Copy register L1– to L2–

Copy register L2+ to L3+
Copy register L1+ to L2+

In addition, there is a single latch register where the counter value is copied by software (instruction `Cnt_PW_Latch`).



How-to:
Evaluate PWM registers

For any evaluation the PWM registers of levels 2 and 3 are used. First, the register values are copied to the shadow registers with `Cnt_Sync_Latch` and then evaluated.

The calculation depends on the set reference edge:

Parameter	rising edge	falling edge
diagram	period T high time t_H High Low L3+ L2+ L1+ L3- L2-	period T high time t_H High Low L3+ L2+ L3- L2- L1-
period	T = L2+ – L3+	T = L2- – L3-
high time	t_H = L3- – L3+	t_H = L2- – L3+
low time	t_L = T – t_H = L2+ – L3-	t_L = T – t_H = L3+ – L3-
frequency	f = 1 / T = 1 / (L2+ – L3+)	f = 1 / T = 1 / (L2- – L3-)
duty cycle	g = t_H / T = (L3- – L3+) / (L2+ – L3+)	g = t_H / T = (L2- – L3+) / (L2- – L3-)

16 Software

You are programming *ADwin-X-A20* with simple *ADbasic* instructions. Basic instructions are described in the *ADbasic* manual.

Instructions for access of inputs / outputs and interfaces are found on following pages:

– page 46: General instructions
– page 53: Analog Inputs and Outputs
– page 77: Digital Inputs and Outputs
– page 110: Counter
– page 127: SSI interface
– page 135: CAN interface
– page 146: RSxxx Interface
– page 152: Profibus interface
– page 156: Profinet interface
– page 159: EtherCAT interface
– page 163: LS-Bus + ADwin-X-A20

16.1 General instructions

This section describes general instructions X-A20:

– Check_LED (page 47)
– Set_LED (page 48)
– Calc_Processdelay (page 49)
– CPU_Event_Config (page 50)
– Sync_All (page 51)

Check_LED

Check_LED returns the status of a LED.

Syntax

```
#Include ADwin-X.inc
ret_val = Check_LED(led_no)
```

Parameters

led_no	Number (1…3) of the LED.	LONG
ret_val	0: LED off. 1: LED glows green. 2: LED glows red. 3: LED glows orange.	LONG

Notes

After power-on, serves as status LED and LED 1 glows red. After booting, process 15 is running and makes LED 1 blink green. If required, you can stop the process with **Stop_Process**.

See also

Set_LED

Valid for

X-A20

Example

```
#Include ADwin-X.inc

Init:
  If (Check_LED(2)=0) Then 'if LED is off …
    Set_LED(2,3)           '… light orange LED
  EndIf
```

Set_LED

`Set_LED` switches one LED on or off.

Syntax

```
#Include ADwin-X.inc
Set_LED(led_no, color)
```

Parameters

`led_no`	Number (1…3) of the LED.	LONG
`color`	0: LED off. 1: LED glows green. 2: LED glows red. 3: LED glows orange.	LONG

Notes

After power-on, LED 1 glows red. After booting process 15 is running and makes LED 1 blink green. If required, you can stop the process with `Stop_Process`.

See also

Check_LED

Valid for

X-A20

Example

```
#Include ADwin-X.inc
Init:
  Set_LED(2,1)              'switch on LED 2, green

Event:
  Rem ...

Finish:
  Set_LED(2,0)              'switch off LED 2
```

Calc_ Processdelay

Calc_Processdelay converts a process frequency into processor ticks (processdelay or cycle time).

Syntax

```
#Include ADwin-X.Inc
ret_val = Calc_Processdelay(frequency)
```

Parameters

frequency	Process frequency in Hertz.	LONG
ret_val	Number of process cycles (= **Processdelay**).	LONG

Notes

- / -

See also

Processdelay

Valid for

X-A20

- / -

Example

```
#Include ADwin-X.Inc

Init:
  Rem set Processdelay for frequency 150kHz
  Processdelay = Calc_Processdelay(150000)
```

CPU_Event_Config

CPU_Event_Config configures the EVENT input.

Syntax

```
#Include ADwin-X.Inc
CPU_Event_Config(min_hold,edge,prescale)
```

Parameters

min_hold	Minimum time, which an event signal after 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…15) of edges, after which an event signal is triggered (default: 1).	LONG

Notes

The input Event 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.

See also

- / -

Valid for

X-A20

Example

```
#Include ADwin-X.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.
  Rem …
```

`Sync_All` starts specified actions synchronously.

Syntax

```
#Include ADwin-X.Inc
Sync_All(pattern)
```

Parameters

`pattern`	Bit pattern selecting the actions (see table below) to be started: Bit = 0: No effect. Bit = 1: Start action synchronously. Bits 31:18 are reserved.	LONG

Notes

The action starting is similar to a standard instruction (most times). Configurations being made before do apply e.g. for the multiplexer or output value.

The availability of actions refers to the X-A20 options.

	Bit no.	Action	similar to
Analog input	0	Start an ADC conversion, mode single shot.	Start_Conv
	1	Start an ADC conversion, mode continuous.	Start_Conv
Analog output	2	Start D/A conversion on DAC1 and DAC2 with the values of the DAC registers.	Start_DAC
Digital input	3	Transfer current status of inputs DIO31:DIO00 into the input latch register.	`Dig_Latch (0001b)`
	4	Transfer current status of inputs DIO63:DIO32 into the input latch register.	`Dig_Latch (0010b)`
Digital output	5	Transfer latch register to outputs DIO31:DIO00.	`Dig_Latch (0100b)`
	6	Transfer latch register to outputs DIO63:DIO32.	`Dig_Latch (1000b)`
Edge output FIFO	7	Start edge output on output FIFO 1.	Digout_ Fifo_Start
	8	Start edge output on output FIFO 2.	Digout_ Fifo_Start
	9	Stop the edge output and clear the edge output FIFO 1.	Digout_ Fifo_Clear
	10	Stop the edge output and clear the edge output FIFO 2.	Digout_ Fifo_Clear
Edge detection unit FIFO	11	Clear the FIFO 1 of the edge detection unit.	Digin_Fifo_ Clear
	12	Clear the FIFO 2 of the edge detection unit.	Digin_Fifo_ Clear
	13	Clear the reference counter of the input FIFO 1.	- / -
	14	Clear the reference counter of the input FIFO 2.	- / -
Counter	15	Set all counters 1…7 to zero.	Cnt_Clear
Sync	16	Copy contents of all counters and PWM counters into buffers.	Cnt_Sync_ Latch
SSI decoder	17	Start reading of the SSI encoder (single shot only).	SSI_Start

Bits 0 and 1 must not be set at the same time. You must run `Start_Conv` with single shot mode first, to set the used channels und the gain (if applicable).

Bits 13 and 14 reset the counters, which create the time stamps for the edge detection units of the input FIFOs; see also `Digout_Fifo_Read_Timer`.

See also

Start_Conv, Start_DAC, Dig_Latch, Digout_Fifo_Start, Digout_Fifo_Clear, Digin_Fifo_Clear, Cnt_Clear, Cnt_Sync_Latch, SSI_Start

Valid for

X-A20

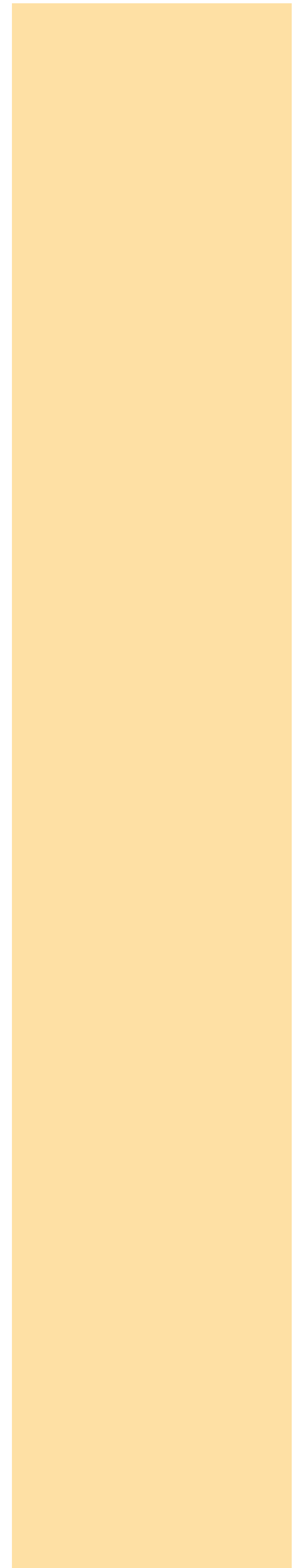
Example

```
#Include ADwin-X.Inc
Dim i As Long

Init:
  Write_DAC(1,3500)          'initialize DAC 1
  Write_DAC(2,65535)         'initialize DAC 2
  REM Set channels DIO15:DIO00 as outputs, DIO31:DIO16 as inputs
  Conf_DIO(0011b)
  Digout_Write_Latch1(0)     'Set all output bits to 0
  i=1                        'initialize index
  Rem initialize A/D conversion for Sync_All:
  Rem ADC 1, gain 1, single shot (!)
  Start_Conv(1b, 0, 1)
  Wait_EOC()                 'wait for end of conversion

Event:
  Rem start ADC (single shot), both DAC, latch digital channels
  Rem DIO31:0 synchronously
  Sync_All(101101b)
  Wait_EOC()                 'Wait for end of conversion

  Rem Read ADC
  Par_1 = Read_ADC(1)
  Write_DAC(1,Par_1)         'set DAC 1
  Write_DAC(2,Par_1 * 3.5)   'set DAC 2

  Par_2 = Digin_Read_Latch1()  'read input bits and …
  Digout_Write_Latch1(Par_1)'output in next event cycle

  If (i=1000) Then End       'End process after 1000 repetitions
  Inc(i)                     'Increment index
```

16.2 Analog Inputs and Outputs

This section describes the following instructions:

– DAC (page 54)
– DAC12 (page 55)
– Start_DAC (page 56)
– Write_DAC (page 57)
– ADC (page 58)
– ADC24 (page 59)
– ADC2 (page 61)
– ADC4 (page 62)
– ADC8 (page 63)
– ADC2_24 (page 64)
– ADC4_24 (page 65)
– ADC8_24 (page 66)
– Read_ADC (page 67)
– Read_ADC24 (page 68)
– Read_ADC_Packed (page 69)
– Read_ADC8 (page 70)
– Read_ADC8_24 (page 71)
– Start_Conv (page 72)
– Start_Conv_PGA (page 74)
– Wait_EOC (page 76)



DAC

`DAC` outputs a defined voltage on a specified analog 16 bit output.

Syntax

```
#Include ADwin-X.inc
DAC (dac_no,value)
```

Parameters

`dac_no`	Number of analog 16 bit output (1…2).	`LONG`
`value`	Value in digits, which defines the voltage to be output (0…65535).	`LONG`

Notes

If you specify `value` beyond the permissible value range, it will automatically be set to the system-specific minimum or maximum value.

The conversion time is 1 µs.

The voltage range is -10V…+10V = 20 V. With the following formula, you can calculate the measured voltage from the returned digital value.

$$\text{voltage} = \frac{\text{measurement range}}{65536} \cdot \left(\text{digits} - 32768_{\text{bipolar}}\right)$$

See also

DAC12, ADC, Start_DAC, Write_DAC

Valid for

X-A20M1, X-A20F

Example

```
#Include ADwin-X.inc
Rem Digital proportional controller
#Define set_to Par_1        'set point
#Define gain Par_2          'gain factor
#Define diff Par_3          'control deviation
#Define out Par_4           'actuating value

Init:
 Processdelay = 10000

Event:
 diff = set_to - ADC(1)     'calculate control deviation
 out = diff * gain          'calculate actuating value
 DAC(1, out)                'output actuating value
```

DAC12

`DAC12` outputs a defined voltage on a specified analog 12 bit output.

Syntax

```
#Include ADwin-X.inc
DAC12 (dac_no,value)
```

Parameters

`dac_no`	Number (1…2) of analog 12 bit output.	LONG
`value`	Value in digits (0, 16, …65520), which defines the voltage to be output.	LONG

Notes

If you specify `value` beyond the permissible value range, it will automatically be set to the system-specific minimum or maximum value.

The conversion time is in the range of 500…1000µs.
If the DAC serves the comparator signal (see below), we recommend to set the voltage in the `Init:` or `LowInit:` section and then wait with `IO_Sleep` until the voltage has been safely set.

Bits 15:4 of `value` are processed as digit value, bits 3:0 are ignored.

Bit no.	31:24	15:4	3:0
Content	0	12-bit value	–

The voltage range is -10V…+10V = 20 V. With the following formula, you can calculate the measured voltage from the returned digital value.

$$\text{voltage} = \frac{\text{measurement range}}{65536} \cdot \left(\text{digits} - 32768_{\text{bipolar}}\right)$$

The outputs of the DAC are internally connected to the comparator inputs. The set DAC voltage serves as comparator signal, i.e. a digital signal with a lower voltage is processed as level Low, with a higher voltage as level High.
The comparator signal must be in the range 0…5 Volt to have the comparator run correctly.

See also

DAC, ADC, Start_DAC, Write_DAC, IO_Sleep

Valid for

X-A20M1, X-A20F

Example

```
#Include ADwin-X.inc
Rem example for the use of comparator inputs

LowInit:
  DAC12(1, 42598)          'set +3V comp. level. channels 1..8
  DAC12(2, 39321)          'set +2V comp. level, channels 9..12
  IO_Sleep(100000)         'wait 1 ms

Event:
  Rem use comparator inputs
```

Start_DAC

Start_DAC starts the conversion or the output of all 16 bit DAC.

Syntax

```
#Include ADwin-X.inc
Start_DAC()
```

Parameters

- / -

Notes

Write_DAC sets the value in the output register of a 16 bit DAC.

You can also start a conversion with Sync_all.

See also

DAC, DAC12, Write_DAC, Sync_All

Valid for

X-A20M1, X-A20F

Example

```
REM Simultaneous output of two different signal waveforms
REM on outputs DAC 1 and 2.
#Include ADwin-X.inc
Dim i As Long

Init:
  Processdelay = 10000
  i=0
  Write_DAC(1,i)          'Set output register DAC1
  Write_DAC(2,65535-i)    'Set output register DAC2

Event:
  Start_DAC()             'Start output of all DAC
  Write_DAC(1,i)          'Set output register DAC1
  Write_DAC(2,65535-i)    'Set output register DAC2
  Inc(i)
  If (i=65535) Then i=0
```

Write_DAC

`Write_DAC` writes a digital value into the output register of a 16 bit DAC.

Syntax

```
#Include ADwin-X.inc
Write_DAC(dac_no,value)
```

Parameters

`dac_no`	Number of analog 16 bit output (1…2).	LONG
`value`	Value in digits, which defines the voltage to be output (0…65535).	LONG

Notes

The conversion into output voltage is started by `Start_DAC`.

If you specify `value` beyond the permissible value range, it will automatically be set to the system-specific minimum or maximum value.

See also

DAC, DAC12, Start_DAC

Valid for

X-A20M1, X-A20F

Example

```
REM Simultaneous output of four different signal waveforms
REM on outputs DAC 1 and 2.
REM The signal waveforms are stored in two DATA arrays and
REM can be filled before start of program from the PC.
#Include ADwin-X.inc
Dim i As Long                'Declaration
Dim Data_1[1000], Data_2[1000] As Long

Init:
 Processdelay = 10000
 i=1
 Write_DAC(1,Data_1[i])     'Set output register DAC1
 Write_DAC(2,Data_2[i])     'Set output register DAC2

Event:
 Start_DAC()                'Start output of all DAC
 Write_DAC(1,Data_1[i])     'Set output register DAC1
 Write_DAC(2,Data_2[i])     'Set output register DAC2
 Inc(i)
 If (i>1000) Then i=1
```

ADC

ADC measures the voltage of an analog input and returns the corresponding digital value.

Syntax

```
#Include ADwin-X.inc
ret_val = ADC(channel)
```

Parameters

channel	Number (1…8) of the analog input channel.	LONG
ret_val	Measurement value in digits (0…65535).	LONG

Notes

ADC24 returns digital values with 24 bit resolution.

ADC is a combination of consecutive functions:

- **Start_Conv**: Start measurement: Convert analog signal to a digital value.
- **Wait_EOC**: Wait for the end of conversion.
- **Read_ADC**: Read out digital value from the register and return it.

In the following cases, the instructions **Start_Conv**, **Wait_EOC** and **Read_ADC** should be used instead of **ADC**:

- Very short cycle times: **Processdelay** < 240 (s.a.).
- You want to use inevitable waiting times for additional program tasks. For example, several conversions can be processed faster than with ADC if you utilize the functions cleverly, see Using Waiting Times (page 157).

The measurement range is ist –10V…+10V = 20V, the gain is 1. With the following formula, you can calculate the measured voltage from the returned digital value.

$$\text{voltage} = \frac{\text{measurement range}}{65536} \cdot \left(\text{digits} - 32768_{\text{bipolar}}\right)$$

The conversion time is 5µs with X-A20M1 (including multiplexer settling time) and 1.25µs with X-A20F.

Please note with X-A20F: If you select different channels with a conversion instruction (ADC… / Start_Conv) than with the previous conversion instruction, the conversion time is extended: With the change, the conversion is processed twice, once with the previously selected channels and once with the newly selected channels.
Example: The change from **ADC8** to **ADC** takes 5µs + 1.25µs; the change from **ADC(3)** to **ADC(2)** takes 1.25µs + 1.25µs.

See also

ADC2, ADC4, ADC8, ADC24, ADC2_24, ADC4_24, ADC8_24, Read_ADC, Read_ADC24, Read_ADC_Packed, Read_ADC8, Read_ADC8_24, Start_Conv, Start_Conv_PGA, Wait_EOC

Valid for

X-A20M1, X-A20F

Example

```
#Include ADwin-X.inc
#Define in_channel 1        'input channel
#Define in_value Par_1

Event:
  Rem Measure analog input 1
  in_value = ADC(in_channel) * 10900
```

ADC24

ADC24 measures the voltage of an analog input and returns the corresponding digital value. The resolution of the return value is 24 bit.

Syntax

```
#Include ADwin-X.Inc
ret_val = ADC24(channel)
```

Parameters

channel	Number (1…8) of the analog input channel.	LONG
ret_val	Measurement value in digits (0…16777215 = 2^{24}-1).	LONG

Notes

ADC returns digital values with 16 bit resolution.

The return value of ADC24 contains an 18 bit measurement value in bits 23:6; bits 5:0 are always zero.

Bit no.	31:24	23:16	15:6	5:0
Content	0	18-bit meas. value		0

ADC24 is a combination of consecutive functions:
- Start_Conv: Start measurement: Convert analog signal–considering the gain factor–to a digital value.
- Wait_EOC: Wait for the end of conversion.
- Read_ADC24: Read out digital value from the register and return it.

If you select a non-existing input channel the measurement value is undefined.

In the following cases, you should use the instructions Start_Conv, Wait_EOC and Read_ADC24 instead of ADC24 in the following cases:
- Very short cycle times: Processdelay < 200: ADC cannot be executed during the cycle time.
- High internal resistance (>3kΩ) of the voltage source of the measurement signal: This increases the settling time of multiplexer.
- You want to use inevitable waiting times for additional program tasks.
  For example, several conversions can be processed faster than with ADC24 if you utilize the functions cleverly, see Using Waiting Times (page 157).

The measurement range is 20 Volt (input voltage range: -10V…10V). With the following formula, you can calculate the measured voltage from the returned digital value:

$$\text{voltage} = \frac{\text{measurement range}}{16777216} \cdot \left(\text{digits} - 8388608_{\text{bipolar}}\right)$$

The conversion time is 5µs with X-A20M1 (including multiplexer settling time) and 1.25µs with X-A20F.

Please note with X-A20F: If you select different channels with a conversion instruction (ADC… / Start_Conv) than with the previous conversion instruction, the conversion time is extended: With the change, the conversion is processed twice, once with the previously selected channels and once with the newly selected channels.
Example: The change from ADC8 to ADC24 takes 5µs + 1.25µs; the change from ADC(3) to ADC(2) takes 1.25µs + 1.25µs.

See also

ADC, ADC2, ADC4, ADC8, ADC2_24, ADC4_24, ADC8_24, Read_ADC, Read_ADC24, Read_ADC_Packed, Read_ADC8, Read_ADC8_24, Start_Conv, Start_Conv_PGA, Wait_EOC

Valid for

X-A20M1, X-A20F

Example

```
#Include ADwin-X.Inc
Dim iw As Long

Init:
  Processdelay = 10000

Event:
  REM Measure voltage of analog input 1
  iw = ADC24(1)
  REM Write measurement value to global variable, so it
  REM can be read from the PC.
  Par_1 = iw
```

ADC2

`ADC2` measures the voltages of 2 selected analog inputs and returns the corresponding digital values (16 bit) in an array.

Syntax

```
#Include ADwin-X.inc
ADC2(array[], array_idx, channel_group)
```

Parameters

Parameter	Description	Type
array[]	Array to hold the measurement values of the two selected input channels.	ARRAY LONG
array_idx	Array element, which holds the first measurement value.	LONG
channel_group	Selected channel group: 0: Channels 1 and 2. 1: Channels 3 and 4. 2: Channels 5 and 6. 3: Channels 7 and 8.	LONG

Notes

With input voltage range –10V…+10V = 20V and gain factor 1, you can calculate the measured voltage from the returned digital value with the formula:

$$\text{voltage} = \frac{\text{measurement range}}{65536} \cdot \left(\text{digits} - 32768_{\text{bipolar}}\right)$$

The conversion time is 1.82µs (added for both channels).

If you select different channels with a conversion instruction (ADC… / Start_Conv) than with the previous conversion instruction, the conversion time is extended: With the change, the conversion is processed twice, once with the previously selected channels and once with the newly selected channels.
Example: The change from `ADC8` to `ADC2` takes 5µs + 1.82µs; the change from `ADC2(0)` to `ADC2(3)` takes 1.82µs + 1.82µs.

See also

ADC, ADC4, ADC8, ADC24, ADC2_24, ADC4_24, ADC8_24, Read_ADC, Read_ADC24, Read_ADC_Packed, Read_ADC8, Read_ADC8_24, Start_Conv, Start_Conv_PGA, Wait_EOC

Valid for

X-A20F

Example

```
#Include ADwin-X.inc
#Define in_array Data_1

Dim in_array[2000] As Long
Dim array_idx As Long

Init:
  array_idx = 1

Event:
  Rem measure inputs 3..4
  ADC2(in_array, array_idx, 1)
  array_idx = array_idx + 2
  If (array_idx > 1992) Then array_idx = 1
```

ADC4

ADC4 measures the voltages of 4 selected analog inputs and returns the corresponding digital values (16 bit) in an array.

Syntax

```
#Include ADwin-X.inc
ADC4(array[], array_idx, channel_group)
```

Parameters

array[]	Array to hold the measurement values of the four selected input channels.	ARRAY LONG
array_idx	Array element, which holds the first measurement value.	LONG
channel_group	Selected channel group: 0: Channels 1…4. 1: Channels 5…8.	LONG

Notes

With input voltage range –10V…+10V = 20V and gain factor 1, you can calculate the measured voltage from the returned digital value with the formula:

$$\text{voltage} = \frac{\text{measurement range}}{65536} \cdot \left(\text{digits} - 32768_{\text{bipolar}}\right)$$

The conversion time is 2.86µs (added for all channels).

If you select different channels with a conversion instruction (ADC… / Start_Conv) than with the previous conversion instruction, the conversion time is extended: With the change, the conversion is processed twice, once with the previously selected channels and once with the newly selected channels.
Example: The change from **ADC8** to **ADC4** takes 5µs + 2.86µs; the change from **ADC4**(0) to **ADC4**(1) takes 2.86µs + 2.86µs.

See also

ADC, ADC2, ADC8, ADC24, ADC2_24, ADC4_24, ADC8_24, Read_ADC, Read_ADC24, Read_ADC_Packed, Read_ADC8, Read_ADC8_24, Start_Conv, Start_Conv_PGA, Wait_EOC

Valid for

X-A20F

Example

```
#Include ADwin-X.inc
#Define in_array Data_1

Dim in_array[2000] As Long
Dim array_idx As Long

Init:
  array_idx = 1

Event:
  Rem measure input 1..4
  ADC4(in_array, array_idx, 0)
  array_idx = array_idx + 4
  If (array_idx > 1992) Then array_idx = 1
```

ADC8

ADC8 measures the voltages of the analog inputs 1…8 and returns the corresponding digital values (16 bit) in an array.

Syntax

```
#Include ADwin-X.inc
ADC8(array[], array_idx)
```

Parameters

array[]	Array to hold the measurement values of the input channels 1…8.	ARRAY LONG
array_idx	Array element, which holds the first measurement value.	LONG

Notes

With input voltage range –10V…+10V = 20V and gain factor 1, you can calculate the measured voltage from the returned digital value with the formula:

$$\text{voltage} = \frac{\text{measurement range}}{65536} \cdot \left(\text{digits} - 32768_{\text{bipolar}}\right)$$

The conversion time is 5µs (added for all channels).

If you select different channels with a conversion instruction (ADC… / Start_Conv) than with the previous conversion instruction, the conversion time is extended: With the change, the conversion is processed twice, once with the previously selected channels and once with the newly selected channels.
Example: The change from **ADC** to **ADC8** takes 1.25µs + 5µs.

See also

ADC, ADC2, ADC4, ADC24, ADC2_24, ADC4_24, ADC8_24, Read_ADC, Read_ADC24, Read_ADC_Packed, Read_ADC8, Read_ADC8_24, Start_Conv, Start_Conv_PGA, Wait_EOC

Valid for

X-A20F

Example

```
#Include ADwin-X.inc
#Define in_array Data_1

Dim in_array[2000] As Long
Dim array_idx As Long

Init:
  array_idx = 1

Event:
  Rem measure inputs 1..8
  ADC8(in_array, array_idx)
  array_idx = array_idx + 8
  If (array_idx > 1992) Then array_idx = 1
```

ADC2_24

ADC2_24 measures the voltages of 2 selected analog inputs (18 bit) and returns the corresponding digital values in an array. The resolution of the return values is 24 bit.

Syntax

```
#Include ADwin-X.inc
ADC2_24(array[], array_idx)
```

Parameters

array[]	Array to hold the measurement values of the two selected input channels.	ARRAY LONG
array_idx	Array element, which holds the first measurement value.	LONG
channel_group	Selected channel group: 0: Channels 1 and 2. 1: Channels 3 and 4. 2: Channels 5 and 6. 3: Channels 7 and 8.	LONG

Notes

Each return value contains an 18 bit measurement value in bits 23:6; bits 5:0 are always zero.

Bit no.	31:24	23:16	15:6	5:0
Content	0	18-bit meas. value		0

With input voltage range –10V…+10V = 20V and gain factor 1, you can calculate the measured voltage from the returned digital value with the formula:

$$\text{voltage} = \frac{\text{measurement range}}{16777216} \cdot \left(\text{digits} - 8388608_{\text{bipolar}}\right)$$

The conversion time is 1.82µs (added for both channels).

If you select different channels with a conversion instruction (ADC… / Start_Conv) than with the previous conversion instruction, the conversion time is extended: With the change, the conversion is processed twice, once with the previously selected channels and once with the newly selected channels.
Example: The change from ADC8 to ADC2_24 takes 5µs + 1.82µs; the change from ADC2_24(0) to ADC2_24(3) takes 1.82µs + 1.82µs.

See also

ADC, ADC2, ADC4, ADC8, ADC24, ADC4_24, ADC8_24, Read_ADC, Read_ADC24, Read_ADC_Packed, Read_ADC8, Read_ADC8_24, Start_Conv, Start_Conv_PGA, Wait_EOC

Valid for

X-A20F

Example

```
#Include ADwin-X.inc
#Define in_array Data_1

Dim in_array[2000] As Long
Dim array_idx As Long

Init :
  array_idx = 1

Event :
  Rem Measure inputs 3..4
  ADC2_24(in_array, array_idx, 1)
  array_idx = array_idx + 2
  If (array_idx > 1992) Then array_idx = 1
```

ADC4_24

ADC4_24 measures the voltages of 4 selected analog inputs (18 bit) and returns the corresponding digital values in an array. The resolution of the return values is 24 bit.

Syntax

```
#Include ADwin-X.inc
ADC4_24(array[], array_idx)
```

Parameters

array[]	Array to hold the measurement values of the four selected input channels.	ARRAY LONG
array_idx	Array element, which holds the first measurement value.	LONG
channel_group	Selected channel group: 0: Channels 1…4. 1: Channels 5…8.	LONG

Notes

Each return value contains an 18 bit measurement value in bits 23:6; bits 5:0 are always zero.

Bit no.	31:24	23:16	15:6	5:0
Content	0	18-bit meas. value		0

With input voltage range –10V…+10V = 20V and gain factor 1, you can calculate the measured voltage from the returned digital value with the formula:

$$\text{voltage} = \frac{\text{measurement range}}{16777216} \cdot \left(\text{digits} - 8388608_{\text{bipolar}}\right)$$

The conversion time is 2.86µs (added for all channels).

If you select different channels with a conversion instruction (ADC… / Start_Conv) than with the previous conversion instruction, the conversion time is extended: With the change, the conversion is processed twice, once with the previously selected channels and once with the newly selected channels.
Example: The change from ADC8 to ADC4_24 takes 5µs + 2.86µs; the change from ADC4_24(0) to ADC4_24(1) takes 2.86µs + 2.86µs.

See also

ADC, ADC2, ADC4, ADC8, ADC24, ADC2_24, ADC8_24, Read_ADC, Read_ADC24, Read_ADC_Packed, Read_ADC8, Read_ADC8_24, Start_Conv, Start_Conv_PGA, Wait_EOC

Valid for

X-A20F

Example

```
#Include ADwin-X.inc
#Define in_array Data_1

Dim in_array[2000] As Long
Dim array_idx As Long

Init :
 array_idx = 1

Event :
 Rem Measure inputs 1..4
 ADC4_24(in_array, array_idx)
 array_idx = array_idx + 4
 If (array_idx > 1992) Then array_idx = 1
```

ADC8_24

ADC8_24 measures the voltages of the analog inputs 1…8 (18 bit) and returns the corresponding digital values in an array. The resolution of the return values is 24 bit.

Syntax

```
#Include ADwin-X.inc
ADC8_24(array[], array_idx)
```

Parameters

array[]	Array to hold the measurement values of the input channels 1…8.	ARRAY LONG
array_idx	Array element, which holds the first measurement value.	LONG

Notes

ADC8 returns eight digital values with 16 bit resolution.

Each return value contains an 18 bit measurement value in bits 23:6; bits 5:0 are always zero.

Bit no.	31:24	23:16	15:6	5:0
Content	0	18-bit meas. value		0

With input voltage range –10V…+10V = 20V and gain factor 1, you can calculate the measured voltage from the returned digital value with the formula:

$$\text{voltage} = \frac{\text{measurement range}}{16777216} \cdot \left(\text{digits} - 8388608_{\text{bipolar}}\right)$$

The conversion time is 5µs (added for all channels).

If you select different channels with a conversion instruction (ADC… / Start_Conv) than with the previous conversion instruction, the conversion time is extended: With the change, the conversion is processed twice, once with the previously selected channels and once with the newly selected channels. Example: The change from ADC to ADC8_24 takes 1.25µs + 5µs.

See also

ADC, ADC2, ADC4, ADC8, ADC24, ADC2_24, ADC4_24, Read_ADC, Read_ADC24, Read_ADC_Packed, Read_ADC8, Read_ADC8_24, Start_Conv, Start_Conv_PGA, Wait_EOC

Valid for

X-A20F

Example

```
#Include ADwin-X.inc
#Define in_array Data_1

Dim in_array[2000] As Long
Dim array_idx As Long

Init :
  array_idx = 1

Event :
  Rem Measure inputs 1..8
  ADC8_24(in_array, array_idx)
  array_idx = array_idx + 8
  If (array_idx > 1992) Then array_idx = 1
```

Read_ADC

`Read_ADC` returns a converted value with 16-bit resolution from an A/D-converter.

Syntax

```
#Include ADwin-X.inc
ret_val = Read_ADC(channel)
```

Parameters

channel	Number (1…8) of the converter to read.	LONG
ret_val	Measurement value in digits (0…65535).	LONG

Notes

`Read_ADC24` returns a converted value with 24-bit resolution.

With input voltage range –10V…+10V = 20V and gain factor 1, you can calculate the measured voltage from the returned digital value with the formula:

$$\text{voltage} = \frac{\text{measurement range}}{65536} \cdot \left(\text{digits} - 32768_{\text{bipolar}}\right)$$

See also

ADC, ADC2, ADC4, ADC8, ADC24, ADC2_24, ADC4_24, ADC8_24, Read_ADC24, Read_ADC_Packed, Read_ADC8, Read_ADC8_24, Start_Conv, Start_Conv_PGA, Wait_EOC

Valid for

X-A20M1, X-A20F

Example

```
#Include ADwin-X.inc
Init:
  Rem start conversion on channel 3, gain 1, single shot
  Start_Conv(100b,0,1)

Event:
  Wait_EOC()              'wait for end of conversion
  Par_1 = Read_ADC(3)     'read value from channel 3
  Rem start next conversion on channel 3
  Start_Conv(100b,0,1)
```

Read_ADC24

`Read_ADC24` returns a converted value with 24-bit resolution from an A/D-converter.

Syntax

```
#Include ADwin-X.inc
ret_val = Read_ADC24(channel)
```

Parameters

channel	Number (1…8) of the converter to read.	LONG
ret_val	Measurement value in digits (0…16777215 = 2^{24}-1).	LONG

Notes

`Read_ADC` returns a digital value with 16 bit resolution.

Each return value contains an 18-bit measurement value in bits 23:6; bits 5:0 are always zero.

Bit no.	31:24	23:16	15:6	5:0
Content	0	18-bit meas. value		0

With input voltage range –10V…+10V = 20V and gain factor 1, you can calculate the measured voltage from the returned digital value with the formula:

$$\text{voltage} = \frac{\text{measurement range}}{16777216} \cdot \left(\text{digits} - 8388608_{\text{bipolar}}\right)$$

See also

ADC, ADC2, ADC4, ADC8, ADC24, ADC2_24, ADC4_24, ADC8_24, Read_ADC, Read_ADC_Packed, Read_ADC8, Read_ADC8_24, Start_Conv, Start_Conv_PGA, Wait_EOC

Valid for

X-A20M1, X-A20F

Example

```
#Include ADwin-X.inc
Init:
  Rem start conversion on channel 5, gain 2, single shot
  Start_Conv(10000b,1,1)

Event:
  Wait_EOC()                'wait for end of conversion
  Par_1 = Read_ADC24(5)     'read value from channel 5
  Rem start next conversion on channel 5
  Start_Conv(10000b,1,1)
```

Read_ADC_Packed

Read_ADC_Packed returns the converted values of 2 channels as a packed value.

Syntax

```
#Include ADwin-X.Inc
ret_val = Read_ADC_Packed(ch_pair)
```

Parameters

ch_pair	Number (1…8) to select a channel pair: 1: Channels 1 and 2. 2: Channels 3 and 4. 3: Channels 5 and 6. 4: Channels 7 and 8.	LONG
ret_val	32 bit value holding 2 measurement values of 16 bit (0…65535) each: Bits 0…15: Value of channel n. Bits 16…31: Value of channel n+1.	LONG

Notes

With input voltage range –10V…+10V = 20V and gain factor 1, you can calculate the measured voltage from a returned digital value with the formula:

$$\text{voltage} = \frac{\text{measurement range}}{65536} \cdot \left(\text{digits} - 32768_{\text{bipolar}}\right)$$

See also

ADC, ADC2, ADC4, ADC8, ADC24, ADC2_24, ADC4_24, ADC8_24, Read_ADC, Read_ADC24, Read_ADC8, Read_ADC8_24, Start_Conv, Start_Conv_PGA, Wait_EOC

Valid for

X-A20F

Example

```
#Include ADwin-X.Inc
Init:
  Rem start conversion on channels 3+4
  Start_Conv(1100b,0,1)

Event:
  Wait_EOC()                  'wait for end of conversion
  Par_1 = Read_ADC_Packed(2) 'read value from channels 3+4
  Par_3 = Par_1 And 0FFFFh 'value channel 3
  Par_4 = Shift_Right(Par_1,16) And 0FFFFh 'value channel 4
  Rem start next conversion on channels 3+4
  Start_Conv(1100b,0,1)
```

Read_ADC8

`Read_ADC8` returns converted values with 16-bit resolution from the analog inputs 1…8 in an array.

Syntax

```
#Include ADwin-X.inc
Read_ADC8(array[],array_idx)
```

Parameters

`array[]`	Array to hold the measurement values of the input channels 1…8.	LONG
`array_idx`	Array element, which holds the first measurement value.	LONG

Notes

With input voltage range –10V…+10V = 20V and gain factor 1, you can calculate the measured voltage from a returned digital value with the formula:

$$\text{voltage} = \frac{\text{measurement range}}{65536} \cdot \left(\text{digits} - 32768_{\text{bipolar}}\right)$$

See also

ADC, ADC2, ADC4, ADC8, ADC24, ADC2_24, ADC4_24, ADC8_24, Read_ADC, Read_ADC24, Read_ADC_Packed, Read_ADC8_24, Start_Conv, Start_Conv_PGA, Wait_EOC

Valid for

X-A20F

Example

```
#Include ADwin-X.inc
Dim Data_1[2000] As Long
Dim data_idx As Long

Init:
  Rem start conversion on channels 1..8
  Start_Conv(0FFh,0,1)
  data_idx = 1

Event:
  Wait_EOC()                   'wait for end of conversion
  Read_ADC8(Data_1, data_idx) 'read values
  data_idx = data_idx + 8
  If (data_idx > 1992) Then data_idx = 1
  Rem start next conversion on channels 1..8
  Start_Conv(0FFh,0,1)
```

Read_ADC8_24

Read_ADC8_24 returns converted values with 24-bit resolution from the analog inputs 1…8 in an array.

Syntax

```
#Include ADwin-X.inc
Read_ADC8_24(array[],array_idx)
```

Parameters

array[]	Array to hold the measurement values of the input channels 1…8.	LONG
array_idx	Array element, which holds the first measurement value.	LONG

Notes

With input voltage range –10V…+10V = 20V and gain factor 1, you can calculate the measured voltage from a returned digital value with the formula:

$$\text{voltage} = \frac{\text{measurement range}}{16777216} \cdot \left(\text{digits} - 8388608_{\text{bipolar}}\right)$$

See also

ADC, ADC2, ADC4, ADC8, ADC24, ADC2_24, ADC4_24, ADC8_24, Read_ADC, Read_ADC24, Read_ADC_Packed, Read_ADC8, Start_Conv, Start_Conv_PGA, Wait_EOC

Valid for

X-A20F

Example

```
#Include ADwin-X.inc
Dim Data_1[2000] As Long
Dim data_idx As Long

Init:
  Rem start conversion on channels 1..8
  Start_Conv(0FFh,0,1)
  data_idx = 1

Event:
  Wait_EOC()                   'wait for end of conversion
  Read_ADC8_24(Data_1, data_idx)'read values
  data_idx = data_idx + 8
  If (data_idx > 1992) Then data_idx = 1
  Rem start next conversion on channels 1..8
  Start_Conv(0FFh,0,1)
```

Start_Conv

Start_Conv starts the conversion of selected A/D converters for a single shot or continuous conversion.

Syntax

```
#Include ADwin-X.inc
Start_Conv(adc_pattern, gain, mode)
```

Parameters

adc_pattern	Bit pattern that specifies, which converter/s should be started (only bits 0…7 can be used): Bit=1: start conversion. Bit=0: do not start conversion.	LONG
gain	Gain factor: 0: Factor = 1, voltage range -10V…+10V. 1: Factor = 2, voltage range -5V…+5V.	LONG
mode	Operating mode of conversion: 1: Single shot. 2: Mode "continuous", continuous conversion.	LONG

Bit no. in adc_pattern	31:8	7	6	5	4	3	2	1	0
ADC number	–	8	7	6	5	4	3	2	1

Notes

With X-A20M1, only a single bit may be set in adc_pattern; parameter gain is ignored and the gain factor is always set to 1 (±10V).

We recommend using the binary representation (suffix "b") for adc_pattern. It shows the allocation of bits to channel groups more clearly than decimal or hexadecimal representations, which can still be used if desired.

The conversion time (added for all channels) depends on th enumber of measured channels:

- 1 channel: max. 800kHz = 1.25µs.
- 2 channels: max. 550kHz = 1.82µs.
- 3 channels: max. 425kHz = 2.35µs.
- 4 channels: max. 350kHz = 2.86µs.
- 5 channels: max. 300kHz = 3.3µs.
- 6 channels: max. 250kHz = 4.0µs.
- 7 channels: max. 225kHz = 4.44µs.
- 8 channels: max. 200kHz = 5.0µs.

Please note with X-A20F: If you select different channels with a conversion instruction (ADC… / Start_Conv) than with the previous conversion instruction, the conversion time is extended: With the change, the conversion is processed twice, once with the previously selected channels and once with the newly selected channels.
Example: The change from **ADC4** to **Start_Conv**(111b,…) takes 2.86µs + 2.35µs; the change from **Start_Conv**(11b,…) to **Start_Conv**(101b,…) takes 1.82µs + 1.82µs.

Use **Start_Conv_PGA** (X-A20Fonly) to set different gain factors for all channels.

With continuous mode, the selected ADCs are continuously running conversions and providing measurement values. After the first start of continuous mode, you must check for the end of conversion with **Wait_EOC** once. Afterwards, you can read the most recent measurement values with **Read_ADC…** instructions.

All **ADC** instructions set the operating mode single shot and therefore end a continuous conversion.

You can also start a conversion with **Sync_All**.

See also

ADC, ADC2, ADC4, ADC8, ADC24, ADC2_24, ADC4_24, ADC8_24, Read_ADC, Read_ADC24, Read_ADC_Packed, Read_ADC8, Read_ADC8_24, Start_Conv_PGA, Wait_EOC, Sync_All

Valid for

X-A20M1, X-A20F

Example

```
#Include ADwin-X.inc
Dim Data_3[2000] As Long
Dim data_idx As Long

Init:
 Rem start continuous conversion with ADC 3, gain 1
 Start_Conv(00000100b, 0, 2)
 Rem check for end of conversion once
 Wait_EOC()
 data_idx = 1

Event:
 Data_3[data_idx] = Read_ADC(3)
 data_idx = data_idx + 1
 If (data_idx > 1992) Then data_idx = 1
 Rem Next conversion is started automatically
```

Start_Conv_PGA

Start_Conv_PGA starts the conversion of selected A/D converters with individual gain factors for a single shot or continuous conversion.

Syntax

```
#Include ADwin-X.inc
Start_Conv_PGA(adc_pattern, gain_pattern, mode)
```

Parameters

adc_pattern	Bit pattern that specifies, which converters should be started (only bits 0…7 can be used): 1: start conversion. 0: do not start conversion.	LONG
gain_pattern	Bit pattern that specifies the gain factors for all converters. Each 2 bits set the factor for the ADC: 00b: Factor = 1, voltage range -10V…+10V. 01b: Factor = 2, voltage range -5V…+5V. 10b, 11b: reserved.	LONG
mode	Operating mode of conversion: 1: Single shot. 2: Mode "continuous", continuous conversion.	LONG

Bit no. in gain_pattern	31:8	7	6	5	4	3	2	1	0
ADC number	–	8	7	6	5	4	3	2	1

Bit no. in gain_pattern	31:8	15:14	13:12	11:10	9:8	7:6	5:4	3:2	1:0
ADC number	–	8	7	6	5	4	3	2	1

Notes

We recommend using the binary representation (suffix "b") for the bit patterns. It shows the allocation of bits to channel groups more clearly than decimal or hexadecimal representations, which can still be used if desired.

The conversion time (added for all channels) depends on th enumber of measured channels:

- 1 channel: max. 800kHz = 1.25µs.
- 2 channels: max. 550kHz = 1.82µs.
- 3 channels: max. 425kHz = 2.35µs.
- 4 channels: max. 350kHz = 2.86µs.
- 5 channels: max. 300kHz = 3.3µs.
- 6 channels: max. 250kHz = 4.0µs.
- 7 channels: max. 225kHz = 4.44µs.
- 8 channels: max. 200kHz = 5.0µs.

If you select different channels with a conversion instruction (ADC… / Start_Conv) than with the previous conversion instruction, the conversion time is extended: With the change, the conversion is processed twice, once with the previously selected channels and once with the newly selected channels. The same applies if you change gain_pattern.
Example: The change from **ADC4** to **Start_Conv**(111b,…) takes 2.86µs + 2.35µs; the change from **Start_Conv**(11b,…) to **Start_Conv**(101b,…) takes 1.82µs + 1.82µs.

With continuous mode, the selected ADCs are continuously running conversions and providing measurement values. After the first start of continuous mode, you must check for the end of conversion with **Wait_EOC** once. Afterwards, you can read the most recent measurement values with **Read_ADC…** instructions.

All **ADC** instructions set the operating mode single shot and therefore end a continuous conversion.

See also

ADC, ADC2, ADC4, ADC8, ADC24, ADC2_24, ADC4_24, ADC8_24, Read_ADC, Read_ADC24, Read_ADC_Packed, Read_ADC8, Read_ADC8_24, Start_Conv, Wait_EOC

Valid for

X-A20F

Example

```
#Include ADwin-X.inc
Dim Data_3[2000] As Long
Dim data_idx As Long

Init:
  Rem start continuous conversion with ADC 1..8,
  Rem gain 1 for channels 1..4, gain 2 for channels 5..8
  Start_Conv_PGA(0FFh, 0101010100000000b, 2)
  Rem check for end of conversion once
  Wait_EOC()
  data_idx = 1

Event:
  Read_ADC8(Data_3, data_idx)'read 8 values
  data_idx = data_idx + 8
  If (data_idx > 1992) Then data_idx = 1
  Rem Next conversion is started automatically
```

Wait_EOC

`Wait_EOC` waits for the end of conversion on the selected A/D converters.

Syntax

```
#Include ADwin-X.inc
Wait_EOC()
```

Parameters

- / -

Notes

ADC converters are selected for conversion with `Start_Conv`.

In single shot mode, `Wait_EOC` waits until all selected ADC have finished with conversion.

In continuous mode, the end of conversion must be checked only once after having started the first conversion.

See also

ADC, ADC2, ADC4, ADC8, ADC24, ADC2_24, ADC4_24, ADC8_24, Read_ADC, Read_ADC24, Read_ADC_Packed, Read_ADC8, Read_ADC8_24, Start_Conv, Start_Conv_PGA

Valid for

X-A20M1, X-A20F

Example

```
#Include ADwin-X.inc
Dim Data_3[2000] As Long
Dim data_idx As Long

Init:
  Rem start continuous conversion with ADC 1..8, gain 1
  Start_Conv(0FFh, 0, 2)
  Rem check for end of conversion once
  Wait_EOC()
  data_idx = 1

Event:
  Read_ADC8(Data_3, data_idx) 'read 8 values
  data_idx = data_idx + 8
  If (data_idx > 1992) Then data_idx = 1
  Rem Next conversion is started automatically
```

16.3 Digital Inputs and Outputs

This section describes instructions to access digital channels:

- Conf_DIO (page 78)
- Digin_Filter_Init (page 87)
- Dig_Latch (page 79)
- Digin_Read_Latch1 (page 80)
- Digin_Read_Latch2 (page 81)
- Digout_Write_Latch1 (page 82)
- Digout_Write_Latch2 (page 83)
- Digin (page 84)
- Digin_Long1 (page 85)
- Digin_Long2 (page 86)
- Digin_Edge1 (page 88)
- Digin_Edge2 (page 89)
- Digout (page 90)
- Digout_Long1 (page 91)
- Digout_Long2 (page 92)
- Digout_Bits1 (page 93)
- Digout_Bits2 (page 94)
- Get_Digout_Long1 (page 95)
- Get_Digout_Long2 (page 96)
- Digin_Fifo_Read_Timer (page 97)
- Digin_Fifo_Clear (page 98)
- Digin_Fifo_Enable (page 99)
- Digin_Fifo_Full (page 100)
- Digin_Fifo_Read (page 101)
- Digout_Fifo_Read_Timer (page 102)
- Digout_Fifo_Clear (page 103)
- Digout_Fifo_Enable (page 104)
- Digout_Fifo_Empty (page 105)
- Digout_Fifo_Mode (page 106)
- Digout_Fifo_Start (page 107)
- Digout_Fifo_Write (page 108)

Conf_DIO

Conf_DIO configures the digital channels DIO41:DIO00 in groups as inputs or outputs.

Syntax

```
#Include ADwin-X.Inc
Conf_DIO(pattern)
```

Parameters

pattern	Bit pattern that configures the digital channels as inputs or outputs: Bit=0: Channels as inputs. Bit=1: Channels as outputs.	LONG

Bit no. in pattern	7	6	5	4	3	2	1	0
channels	DIO41	DIO40	DIO36 : DIO39	DIO32 : DIO35	DIO24 : DIO31	DIO16 : DIO23	DIO08 : DIO15	DIO00 : DIO07

Notes

The digital channels DIO41:DIO00 are initially configured as inputs after power-up (and can then not yet be used as outputs). They can only be configured in groups as inputs or outputs.

The digital channels DIO60:DIO42 are already configured as inputs and outputs and cannot be changed with Conf_DIO. The digital channels are placed on 3 connectors.

We recommend using the binary representation (suffix "b"). It shows the allocation of bits to channel groups more clearly than decimal or hexadecimal representations, which can still be used if desired.

See also

Conf_DIO, Digin_Filter_Init, Digin, Digout, Digin_Fifo_Enable, Digin_Fifo_Read, Digout_Fifo_Enable, Digout_Fifo_Write

Valid for

X-A20, X-A20+D, X-A20+DCT

Example

```
#Include ADwin-X.Inc

Init:
  Rem Configure DIO15:00 as inputs and DIO41:16 as outputs
  Conf_DIO(11111100b)
```

Dig_Latch

Dig_Latch transfers digital information from inputs to the input latches and from output latches to the outputs.

Syntax

```
#Include ADwin-X.Inc
Dig_Latch(pattern)
```

Parameters

pattern	Bit pattern to select groups of digital channels to be latched.	LONG

Bit no. in pattern	31:4	3	2	1	0
channels	–	outputs DIO63: DIO32	outputs DIO31: DIO00	inputs DIO63: DIO32	inputs DIO31: DIO00

Notes

With digital inputs, the instructions reads the input signals into the input latches. Read the values with **Digin_Read_Latch1/2**.

With digital outputs, the instruction passes the values of the output latches to the outputs. Write values into the latch register with **Digout_Write_Latch1/2**.

You can also start the transfer with **Sync_All**.

See also

Conf_DIO, Digin_Read_Latch1, Digin_Read_Latch2, Digout_Write_Latch1, Digout_Write_Latch2, Digin, Digout, Sync_All

Valid for

X-A20, X-A20+D, X-A20+DCT

Example

```
#Include ADwin-X.Inc

Init:
  REM Set channels DIO15:DIO00 as outputs, DIO31:DIO16 as inputs
  Conf_DIO(0011b)
  Digout_Write_Latch1(0)    'Set all output bits to 0

Event:
  Dig_Latch(0101b)          'latch inputs and outputs DIO31:DIO00
  Rem further program
  Par_1 = Digin_Read_Latch1() 'read input bits and …
  Digout_Write_Latch1(Par_1) 'output in next event cycle
```

Digin_Read_Latch1

Digin_Read_Latch1 returns the bits from the latch register for the digital inputs.

Syntax

```
#Include ADwin-X.Inc
ret_val = Digin_Read_Latch1()
```

Parameters

ret_val   Bit pattern. Each bit corresponds to a digital input (see table).   LONG

Bit no in ret_val	31	30	…	1	0
Input	DIO31	DIO30	…	DIO01	DIO00

Notes

We recommend first programming the specified channels as inputs using **Conf_DIO**.

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

- **Dig_Latch**
- **Sync_All**

See also

Conf_DIO, Dig_Latch, Digin_Read_Latch2, Digout_Write_Latch1, Digout_Write_Latch2, Digin, Digout, Sync_All

Valid for

X-A20+DCT

Example

```
#Include ADwin-X.Inc

Init:
  REM Set channels DIO15:DIO00 as outputs, DIO31:DIO16 as inputs
  Conf_DIO(0011b)
  Digout_Write_Latch1(0)    'Set all output bits to 0

Event:
  Dig_Latch(0101b)          'latch inputs and outputs DIO31:DIO00
  Rem further program
  Par_1 = Digin_Read_Latch1() 'read input bits and …
  Digout_Write_Latch1(Par_1) 'output in next event cycle
```

Digin_Read_Latch2

Digin_Read_Latch2 returns the bits from the latch register for the digital inputs DIO60:DIO32.

Syntax

```
#Include ADwin-X.Inc
ret_val = Digin_Read_Latch2()
```

Parameters

ret_val	Bit pattern. Each bit corresponds to a digital input (see table).	LONG

Bit no in ret_val	31:29	28	27	…	1	0
Input	–	DIO60	DIO59	…	DIO33	DIO32

Notes

We recommend first programming the specified channels as inputs using **Conf_DIO**.

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

- **Dig_Latch**
- **Sync_All**

See also

Conf_DIO, Dig_Latch, Digin_Read_Latch1, Digin_Read_Latch2, Digout_Write_Latch1, Digout_Write_Latch2, Digin, Digout, Sync_All

Valid for

X-A20, X-A20+D, X-A20+DCT

Example

```
#Include ADwin-X.Inc

Init:
  REM Set channels DIO39:DIO32 as outputs
  Conf_DIO(110000b)
  Digout_Write_Latch2(0)     'Set all output bits to 0

Event:
  Dig_Latch(1010b)           'latch inputs and outputs 32..63
  Rem further program
  Par_1 = Digin_Read_Latch2()  'read input bits and …
  Digout_Write_Latch1(Par_1)'output in next event cycle
```

Digout_Write_Latch1

`Digout_Write_Latch1` writes a 32 bit value into the latch register for the digital outputs DIO31:DIO00.

Syntax

```
#Include ADwin-X.Inc
Digout_Write_Latch1(pattern)
```

Parameters

`pattern`	Bit pattern. Each bit corresponds to a digital output (see table).	`LONG`

Bit no in `ret_val`	31	30	…	1	0
Output	DIO31	DIO30	…	DIO01	DIO00

Notes

The specified channels must first be programmed as outputs using `Conf_DIO`.

You can set a single digital output directly with `Digout`.

See also

Conf_DIO, Dig_Latch, Digin_Read_Latch1, Digin_Read_Latch2, Digout_Write_Latch1, Digout_Write_Latch2, Digin, Digout, Sync_All

Valid for

X-A20+DCT

Example

```
#Include ADwin-X.Inc

Init:
  REM Set channels DIO15:DIO00 as outputs, DIO31:DIO16 as inputs
  Conf_DIO(0011b)
  Digout_Write_Latch1(0)    'Set all output bits to 0

Event:
  Dig_Latch(0101b)          'latch inputs and outputs DIO31:DIO00
  Rem further program
  Par_1 = Digin_Read_Latch1() 'read input bits and …
  Digout_Write_Latch1(Par_1)'output in next event cycle
```

Digout_Write_Latch2

Digout_Write_Latch2 writes a 32 bit value into the latch register for the digital outputs DIO41:DIO32.

Syntax

```
#Include ADwin-X.Inc
Digout_Write_Latch2(pattern)
```

Parameters

pattern	Bit pattern. Each bit corresponds to a digital output (see table).	LONG

Bit no in ret_val	31:10	9	8	…	1	0
Output	–	DIO41	DIO40	…	DIO33	DIO32

Notes

The specified channels must first be programmed as outputs using **Conf_DIO**.

You can set a single digital output directly with **Digout**.

See also

Conf_DIO, Dig_Latch, Digin_Read_Latch1, Digin_Read_Latch2, Digout_Write_Latch1, Digout_Write_Latch2, Digin, Digout, Sync_All

Valid for

X-A20, X-A20+D, X-A20+DCT

Example

```
#Include ADwin-X.Inc

Init:
  REM Set channels DIO39:DIO32 as outputs
  Conf_DIO(110000b)
  Digout_Write_Latch2(0)    'Set all output bits to 0

Event:
  Dig_Latch(1010b)          'latch inputs and outputs 32..63
  Rem further program
  Par_1 = Digin_Read_Latch2() 'read input bits and …
  Digout_Write_Latch1(Par_1) 'output in next event cycle
```

Digin

Digin returns the TTL level of a digital input DIO60:DIO00.

Syntax

```
#Include ADwin-X.inc
ret_val = Digin(channel_no)
```

Parameters

channel_no	Number (0…60) of digital input.	LONG
ret_val	TTL level of the selected input: 1: TTL level is high. 0: TTL level is low.	LONG

Notes

For any digital channel configured as output **Digin** has no function.

Conf_DIO configures digital channels DIO41:DIO00 as inputs or outputs in groups. The channels DIO60:DIO42 are always configured as inputs.

The instruction is used to read the TTL levels of a few digital inputs. With more digital inputs, the instructions **Digin_Long1/2** are remarkably faster.

See also

Conf_DIO, Digin_Long1, Digin_Long2, Digin_Edge1, Digin_Edge2, Digout, Digin_Fifo_Enable, Digin_Fifo_Read, Digout_Fifo_Enable, Digout_Fifo_Write

Valid for

X-A20, X-A20+D, X-A20+DCT

Example

```
#Include ADwin-X.inc
Dim Data_1[10000] As Long As Fifo

Event:
  Rem Check if input 0 has TTL level high
  If (Digin(0) = 1) Then
    Data_1 = ADC(1)          'read value of ADC 1
  EndIf
```

Digin_Long1

`Digin_Long1` returns the values of the digital inputs DIO31:DIO00.

Syntax

```
#Include ADwin-X.Inc
ret_val = Digin_Long1()
```

Parameters

`ret_val` Bit pattern that corresponds to the TTL-levels at the digital inputs (see table). LONG
1: TTL-level high.
0: TTL-level low.

Bit number in `ret_val`	31	30	…	1	0
Input	DIO31	DIO30	…	DIO01	DIO00

Notes

For any digital channel configured as output `Digin_Long1` will return an undefined value.

`Conf_DIO` configures digital channels DIO31:DIO00 as inputs or outputs in groups of 8.

See also

Conf_DIO, Digin_Long2, Digin_Edge1, Digin_Edge2, Digout, Digout_Long1, Digin_Fifo_Enable, Digin_Fifo_Read, Digout_Fifo_Enable, Digout_Fifo_Write

Valid for

X-A20+DCT

Example

```
#Include ADwin-X.Inc

Init:
  Rem Configure DIO15:00 as inputs and DIO31:16 as outputs
  Conf_DIO(1100b)

Event:
  Par_1 = Digin_Long1() 'read values of inputs (DIO15:00)
```

Digin_Long2

`Digin_Long2` returns the values of the digital inputs DIO60:DIO32.

Syntax

```
#Include ADwin-X.Inc
ret_val = Digin_Long2()
```

Parameters

`ret_val`	Bit pattern that corresponds to the TTL-levels at the digital inputs (see table). 1: TTL-level high. 0: TTL-level low.	`LONG`

Bit number in `ret_val`	31:29	28	27	…	1	0
Input	–	DIO60	DIO59	…	DIO33	DIO32

Notes

For any digital channel configured as output `Digin_Long2` will return an undefined value.

`Conf_DIO` configures digital channels DIO41:DIO00 as inputs or outputs in groups. The channels DIO60:DIO42 are always configured as inputs.

See also

Conf_DIO, Digin_Long1, Digin_Edge1, Digin_Edge2, Digout, Digout_Long2, Digin_Fifo_Enable, Digin_Fifo_Read, Digout_Fifo_Enable, Digout_Fifo_Write

Valid for

X-A20, X-A20+D, X-A20+DCT

Example

```
#Include ADwin-X.Inc

Init:
  Rem Configure DIO15:00/DIO39:32 as inputs,
  Rem and DIO31:16/DIO41:40 as outputs
  Conf_DIO(11001100b)

Event:
  Par_1 = Digin_Long1() 'read inputs DIO15:00
  Par_2 = Digin_Long2() 'read inputs DIO39:32/DIO59:42
```

Digin_Filter_Init

Digin_Filter_Init sets the filter duration for all digital inputs.

Syntax

```
#Include ADwin-X.inc
Digin_Filter_Init(filter_value)
```

Parameters

filter_value	Filter duration, given in units (1…65535) of 20 ns. The value 0 (zero) disables the filter.	LONG

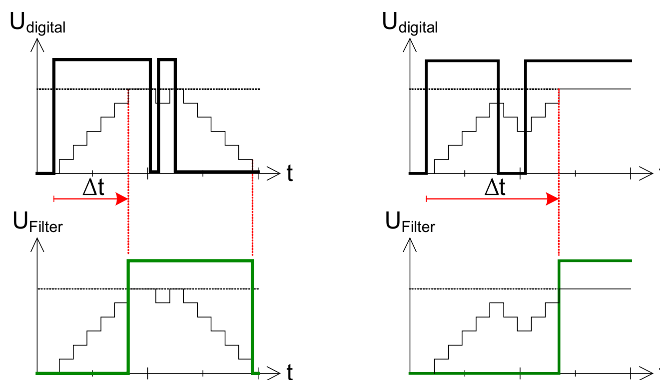
Notes

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 and also refer to the inputs of counters and SSI decoders. Each channel has its own filter. After power-up all filters are disabled.

The filter does not transfer an edge of the input signal directly to the output signal. According to the input signal, a counter is increased (High signal) or decreased every 20 ns, in the range of 0… **filter_value**. If the counter value is 0, the output signal has level Low, at **filter_value** it is level High.

Please note: The filter delays edges of the resulting signal by the set filter duration. If spikes occur, edges may delay slightly in addition.



The plot shows filtering of 2 example signals (black line, top) with spikes. The step line displays the filter counter values. In the right example, the resulting edge is delayed by the spike. The filter (using filter_value = 6) delays edges of the resulting signal; the delay Δt may increase according to the number of spikes.

Please note: The input filter affects the time stamps for the edge detection unit of an input FIFO (see **Digin_Fifo_Read_Timer**). With enabled input filter, the time distance between two time stamps is an integer multiple of 20 ns. In other words: there are either only even or only odd time stamps.

See also

Conf_DIO, Digin_Long1, Digin_Long2, Digin_Edge1, Digin_Edge2, Cnt_Mode, SSI_Mode, Digin_Fifo_Read_Timer

Valid for

X-A20, X-A20+D, X-A20+DCT

Example

```
#Include ADwin-X.Inc

Init:
 Conf_DIO(0000b)           'Set DIO31:00 as inputs
 Digin_Filter_Init(5)      'set spike filter to 100ns

Event:
 Par_1 = Digin_Long1()     'Read inputs DIO31:00
```

Digin_Edge1

`Digin_Edge1` returns whether a positive or negative edge has occurred on digital inputs DIO31:DIO00.

Syntax

```
#Include ADwin-X.inc
ret_val = Digin_Edge1(edge)
```

Parameters

`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	…	2	1	0
Input	DIO31	DIO30	…	DIO02	DIO01	DIO00

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.

`Conf_DIO` configures digital channels DIO31:DIO00 as inputs or outputs in groups of 8.

A query with `Digin_Edge1` resets all bits to zero.

See also

Conf_DIO, Digin_Long1, Digin_Long2, Digin_Edge2, Digin_Fifo_Enable, Digin_Fifo_Read, Digout_Fifo_Read_Timer

Valid for

X-A20+DCT

Example

```
#Include ADwin-X.inc

Init:
  Conf_DIO(1100b)              'channels 15:0 as inputs

Event:
  Rem check rising and falling edges, mask out outputs
  Par_1 = Digin_Edge1(1) And 0Fh
  Par_2 = Digin_Edge1(0) And 0Fh

  Rem output edge changes to outputs
  If (Par_1 + Par_2 > 0)Then
    Digout_Bits1(Shift_Left(Par_1,16),Shift_Left(Par_2,16))
  EndIf
```

Digin_Edge2

Digin_Edge2 returns whether a positive or negative edge has occurred on digital inputs DIO60:DIO32.

Syntax

```
#Include ADwin-X.inc
ret_val = Digin_Edge2(edge)
```

Parameters

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.	31:29	28	27	…	2	1	0
Input	–	DIO60	DIO59	…	DIO34	DIO33	DIO32

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.

`Conf_DIO` configures digital channels DIO41:DIO00 as inputs or outputs in groups. The channels DIO60:DIO42 are always configured as inputs.

A query with `Digin_Edge2` resets all bits to zero.

See also

Conf_DIO, Digin_Long1, Digin_Long2, Digin_Edge1, Digin_Fifo_Enable, Digin_Fifo_Read, Digout_Fifo_Read_Timer

Valid for

X-A20, X-A20+D, X-A20+DCT

Example

```
#Include ADwin-X.inc

Init:
  Rem Configure DIO15:00/DIO39:32 as inputs,
  Rem and DIO31:16/DIO41:40 as outputs
  Conf_DIO(11001100b)

Event:
  Rem check rising and falling edges, mask out outputs
  Par_1 = Digin_Edge2(1) And 0Fh
  Par_2 = Digin_Edge2(0) And 0Fh

  Rem output edge changes to outputs
  If (Par_1 + Par_2 > 0)Then
   Digout_Bits2(Shift_Left(Par_1,16),Shift_Left(Par_2,16))
  EndIf
```

Digout

`Digout` sets a single output DIO41:DIO00 to the level "high" or "low".

Syntax

```
#Include ADwin-X.Inc
Digout(channel_no,level)
```

Parameters

`channel_no`	Number (0…41) of the digital output DIO41:DIO00.	`LONG`
`level`	New status of the selected output: 0: Low level. 1: High level.	`LONG`

Notes

For any digital channel configured as input `Digout` will have no effect.

`Conf_DIO` configures digital channels DIO41:DIO00 as inputs or outputs in groups. The channels DIO60:DIO42 are always configured as inputs.

The digital channels are placed on three connectors.

The instruction is used to set the TTL levels of a few digital outputs. With more digital outputs, the instructions `Digout_Long1/2` are remarkably faster.

See also

Conf_DIO, Digin, Digout_Long1, Digout_Long2, Digout_Bits1, Digout_Bits2, Get_Digout_Long1, Get_Digout_Long2, Digin_Fifo_Enable, Digout_Fifo_Enable, Digout_Fifo_Write

Valid for

X-A20, X-A20+D, X-A20+DCT

Example

```
#Include ADwin-X.Inc

Init:
  Rem Configure DIO31:00 as outputs
  Conf_DIO(1111b)
  Par_2 = 0AAAAAAAAh         'Bit pattern for even bits

Event:
  Digout(Par_2,1)            'set outputs to level high
```

Digout_Long1

Digout_Long1 sets or clears the TTL levels of outputs DIO31:DIO00 with a bit pattern.

Syntax

```
#Include ADwin-X.Inc
Digout_Long1(pattern)
```

Parameters

pattern — Bit pattern that sets the TTL levels of digital outputs: LONG
Bit = 0: Set output to level "low".
Bit = 1: Set output to level "high".

Bit number in pattern	31	30	…	1	0
Output	DIO31	DIO30	…	DIO01	DIO00

Notes

For any digital channel configured as input, Digout_Long1 will have no effect.

Conf_DIO configures digital channels DIO31:DIO00 as inputs or outputs in groups of 8.

See also

Conf_DIO, Digin, Digout, Digout_Long2, Digout_Bits1, Digout_Bits2, Get_Digout_Long1, Get_Digout_Long2, Digin_Fifo_Enable, Digout_Fifo_Enable, Digout_Fifo_Write

Valid for

X-A20+DCT

Example

```
#Include ADwin-X.Inc

Init:
  Rem Configure DIO15:00 as outputs and DIO31:16 as inputs
  Conf_DIO(0011b)
  Rem bit pattern for odd bits
  Par_1 = 55555555h

Event:
  Digout_Long1(Par_1)        'output bits DIO15:00
```

Digout_Long2

Digout_Long2 sets or clears the TTL levels of outputs DIO41:DIO32 with a bit pattern.

Syntax

```
#Include ADwin-X.Inc
Digout_Long2(pattern)
```

Parameters

pattern	Bit pattern that sets the TTL levels of digital outputs: Bit = 0: Set output to level "low". Bit = 1: Set output to level "high".	LONG

Bit number in pattern	31:10	9	8	…	1	0
Output	–	DIO41	DIO40	…	DIO33	DIO32

Notes

For any digital channel configured as input, Digout_Long2 will have no effect.

Conf_DIO configures digital channels DIO41:DIO00 as inputs or outputs in groups. The channels DIO60:DIO42 are always configured as inputs.

See also

Conf_DIO, Digin, Digout, Digout_Long1, Digout_Bits1, Digout_Bits2, Get_Digout_Long1, Get_Digout_Long2, Digin_Fifo_Enable, Digout_Fifo_Enable, Digout_Fifo_Write

Valid for

X-A20, X-A20+D, X-A20+DCT

Example

```
#Include ADwin-X.Inc

Init:
  Rem Configure DIO39:32 as outputs
  Conf_DIO(00110000b)
  Rem bit pattern for odd bits
  Par_2 = 0AAAAAAAAh

Event:
  Digout_Long2(Par_2)          'output bits DIO39:32
```

Digout_Bits1

Digout_Bits1 sets some of the digital channels DIO31:DIO00 to a defined TTL level.

Syntax

```
#Include ADwin-X.Inc
Digout_Bits1(set,clear)
```

Parameters

`set` Bit pattern to specify outputs, which are set to TTL level High (see table). LONG
1: set to TTL level high.
0: do not change TTL level.

`clear` Bit pattern to specify outputs, which are set to TTL level Low (see table). LONG
1: set to TTL level low.
0: do not change TTL level.

Bit number in set/clear	31	30	…	1	0
Output	DIO31	DIO30	…	DIO01	DIO00

Notes

For any digital channel configured as input, **Digout_Bits1** will have no effect.

Conf_DIO configures digital channels DIO31:DIO00 as inputs or outputs in groups of 8.

See also

Conf_DIO, Digin, Digout, Digout_Long1, Digout_Long2, Digout_Bits2, Get_Digout_Long1, Get_Digout_Long2, Digin_Fifo_Enable, Digout_Fifo_Enable, Digout_Fifo_Write

Valid for

X-A20+DCT

Example

```
#Include ADwin-X.Inc

Init:
  Rem Configure DIO15:00 as outputs and DIO31:16 as inputs
  Conf_DIO(0011b)
  Rem bit pattern for odd bits
  Par_1 = 55555555h

Event:
  Rem set odd bits to level high, leave even bits unchanged
  Digout_Bits1(Par_1,0)
```

Digout_Bits2

Digout_Bits2 sets some of the digital channels DIO41:DIO32 to a defined TTL level.

Syntax

```
#Include ADwin-X.Inc
Digout_Bits2(set,clear)
```

Parameters

set	Bit pattern to specify outputs, which are set to TTL level High (see table). 1: set to TTL level high. 0: do not change TTL level.	LONG
clear	Bit pattern to specify outputs, which are set to TTL level Low (see table). 1: set to TTL level low. 0: do not change TTL level.	LONG

Bit number in set/clear	31:10	9	8	…	1	0
Output	–	DIO41	DIO40	…	DIO33	DIO32

Notes

For any digital channel configured as input, Digout_Bits2 has no effect.

Conf_DIO configures digital channels DIO41:DIO00 as inputs or outputs in groups. The channels DIO60:DIO42 are always configured as inputs.

See also

Conf_DIO, Digin, Digout, Digout_Long1, Digout_Long2, Digout_Bits1, Get_Digout_Long1, Get_Digout_Long2, Digin_Fifo_Enable, Digout_Fifo_Enable, Digout_Fifo_Write

Valid for

X-A20, X-A20+D, X-A20+DCT

Example

```
#Include ADwin-X.Inc

Init:
  Rem Configure DIO39:32 as outputs
  Conf_DIO(00110000b)
  Rem bit pattern for even bits
  Par_2 = 0AAAAAAAAh

Event:
  Rem set even bits to level high, leave odd bits unchanged
  Digout_Bits2(Par_1,0)
```

Get_Digout_Long1

Get_Digout_Long1 returns the register contents of the digital outputs DIO31:DIO00.

Syntax

```
#Include ADwin-X.inc
ret_val = Get_Digout_Long1()
```

Parameters

ret_val — Contents (bit pattern) of the output register, bit allocation to outputs see table. — LONG
1: TTL-level high.
0: TTL-level low.

Bit no. in ret_val	31	30	…	1	0
Kanal	DIO31	DIO30	…	DIO01	DIO00

Notes

The return value represents the status of the output register only. A read back of physical output status is technically impossible.

For any digital channel configured as input, **Get_Digout_Long1** will return an undefined value. **Conf_DIO** configures digital channels DIO31:DIO00 as inputs or outputs in groups of 8.

See also

Conf_DIO, Digin, Digout, Digout_Long1, Digout_Long2, Digout_Bits1, Digout_Bits2, Get_Digout_Long2, Digin_Fifo_Enable, Digout_Fifo_Enable, Digout_Fifo_Write

Valid for

X-A20+DCT

Example

```
#Include ADwin-X.inc

Init:
  REM Configure channels DIO31:00 as outputs
  Conf_DIO(1111b)
  Processdelay = 10000

Event:
  Par_1 = Get_Digout_Long1() 'read back bits 31:0
```

Get_Digout_Long2

`Get_Digout_Long2` returns the register contents of the digital outputs DIO41:DIO32.

Syntax

```
#Include ADwin-X.inc
ret_val = Get_Digout_Long2()
```

Parameters

`ret_val` Contents (bit pattern) of the output register, bit allocation to outputs see table. `LONG`
1: TTL-level high.
0: TTL-level low.

Bit number in `ret_val`	31:10	9	8	…	1	0
Output	–	DIO41	DIO40	…	DIO33	DIO32

Notes

The return value represents the status of the output register only. A read back of physical output status is technically impossible.

For any digital channel configured as input, `Get_Digout_Long2` will return an undefined value. `Conf_DIO` configures digital channels DIO41:DIO00 as inputs or outputs in groups. The channels DIO60:DIO42 are always configured as inputs.

See also

Conf_DIO, Digin, Digout, Digout_Long1, Digout_Long2, Digout_Bits1, Digout_Bits2, Get_Digout_Long1, Digin_Fifo_Enable, Digout_Fifo_Enable, Digout_Fifo_Write

Valid for

X-A20, X-A20+D, X-A20+DCT

Example

```
#Include ADwin-X.inc

Init:
  Rem Configure DIO39:32 as outputs
  Conf_DIO(00110000b)
  Processdelay = 10000

Event:
  Par_1 = Get_Digout_Long2() 'read back bits DIO39:32
```

Digin_Fifo_Read_Timer

Digin_Fifo_Read_Timer returns the current status of the 100MHz timer.

Syntax

```
#Include ADwin-X.inc
ret_val = Digin_Fifo_Read_Timer(fifo_no)
```

Parameters

fifo_no	Number (1, 2) of the input FIFO for edge detection.	LONG
ret_val	Current value ($-2^{31}-1 \ldots 2^{31}$) of the 100MHz timer.	LONG

Notes

The timer is used to provide time stamps for the edge detection unit, see **Digin_Fifo_Enable**.

The timer value is increased every 10ns by 1, so the timer will reach the original timer value after about 43 seconds (= $10\text{ns} \times 2^{32}$). 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.

Please note: If the input filter (see **Digin_Filter_Init**) is enabled, the time distance between two time stamps is an integer multiple of 20ns. In other words: there are either only even or only odd time stamps.

The Fifo timers can be reset to zero with **Sync_All**.

See also

Digin, Digin_Edge1, Digin_Edge2, Digout, Digin_Fifo_Clear, Digin_Fifo_Enable, Digin_Fifo_Full, Digin_Fifo_Read, Digout_Fifo_Read_Timer, Digout_Fifo_Enable, Digin_Filter_Init, Sync_All

Valid for

X-A20+DCT

Example

```
#Include ADwin-X.inc
Rem provide number of counter overflows
#Define count_overflow Par_1
Dim t_start, diff_new, diff_old As Long

Init:
  count_overflow = 0          'overflow occurs every 43 seconds
  t_start = Digin_Fifo_Read_Timer()
  diff_old = 0

Event:
  Rem Event section must be run at least once every 20 seconds.
  Rem Else you will miss counter overflows.

  Rem get timer difference
  diff_new = Digin_Fifo_Read_Timer() - t_start
  If ((diff_new > 0) And (diff_old < 0)) Then
    Inc(count_overflow)       'increase number of counter overflows
  EndIf
  diff_old = diff_new
```

related examples see

– *ADbasic* example `seconds_timer.bas` in folder `C:\ADwin\ADbasic\samples_ADwin: seconds_timer.bas`

Digin_Fifo_Clear

`Digin_Fifo_Clear` clears the FIFO of the edge detection unit.

Syntax

```
#Include ADwin-X.inc
Digin_Fifo_Clear(fifo_no)
```

Parameters

`fifo_no`	Number (1, 2) of the input FIFO for edge detection.	`LONG`

Notes

The input FIFO 1 refers to the digital inputs DIO31:DIO00, the input FIFO 2 refers to digital inputs DIO60:DIO32.

The input FIFOs can also be cleared with `Sync_All`.

See also

Digin, Digin_Edge1, Digin_Edge2, Digout, Digin_Fifo_Read_Timer, Digin_Fifo_Enable, Digin_Fifo_Full, Digin_Fifo_Read, Digout_Fifo_Clear, Digout_Fifo_Enable, Sync_All

Valid for

X-A20+DCT

Example

see Digin_Fifo_Enable

Digin_Fifo_Enable

Digin_Fifo_Enable determines, which input channels the edge detection unit will monitor.

Syntax

```
#Include ADwin-X.inc
Digin_Fifo_Enable(fifo_no,pattern)
```

Parameters

fifo_no	Number (1, 2) of the input FIFO for edge detection.	LONG
pattern	Bit pattern to select the input channels to be monitored.	LONG

Notes

The input FIFO 1 refers to the digital inputs DIO31:DIO00, the input FIFO 2 refers to digital inputs DIO60:DIO32.

Bit no. FIFO 1	31	30	…	2	1	0
DIO input	DIO31	DIO30	…	DIO02	DIO01	DIO00

Bit no. FIFO 2	31:29	28	27	…	1	0
DIO input	–	DIO60	DIO59	…	DIO33	DIO32

Only input channels can be monitored. The channels are programmed as inputs or outputs with **Conf_DIO**.

The edge detection unit checks every 10ns, 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.
- Value 2 contains a time stamp, which is the current value of a 100MHz timer.

The FIFO array may contain 511 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

Conf_DIO, Digin, Digin_Edge1, Digin_Edge2, Digout, Digin_Fifo_Read_Timer, Digin_Fifo_Clear, Digin_Fifo_Full, Digin_Fifo_Read, Digout_Fifo_Enable

Valid for

X-A20+DCT

Example

```
#Include ADwin-X.inc

Dim Data_1[10000], Data_2[10000] As Long
Dim i, num, index As Long

Init:
 Conf_DIO(1100b)            'channels 15:0 as inputs
 Digin_Fifo_Enable(1,0)     'edge control off
 Digin_Fifo_Clear(1)        'clear FIFO 1
 Digin_Fifo_Enable(1,101010b)'control channels 1,3,5
 index = 1

Event:
 num = Digin_Fifo_Full(1)  'get number of value pairs
 If (num > 0) Then
  If (index + num > 10000) Then index = 1
  Rem read value pairs
  For i = 1 To num
   Digin_Fifo_Read(1,Data_1[index], Data_2[index])
   index = index+1
  Next i
 EndIf
```

Digin_Fifo_Full

`Digin_Fifo_Full` returns the number of saved value pairs in the FIFO of the edge detection unit.

Syntax

```
#Include ADwin-X.inc
ret_val = Digin_Fifo_Full(fifo_no)
```

Parameters

fifo_no	Number (1, 2) of the input FIFO for edge detection.	LONG
ret_val	Number (0…511) of saved value pairs in the FIFO.	LONG

Notes

The FIFO array may contain 511 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

Digin, Digin_Edge1, Digin_Edge2, Digout, Digin_Fifo_Read_Timer, Digin_Fifo_Clear, Digin_Fifo_Enable, Digin_Fifo_Read, Digout_Fifo_Enable, Digout_Fifo_Empty

Valid for

X-A20+DCT

Example

see Digin_Fifo_Enable

Digin_Fifo_Read

Digin_Fifo_Read reads one value pair from the FIFO of the edge detection unit and returns them in 2 variables.

Syntax

```
#Include ADwin-X.inc
Digin_Fifo_Read(fifo_no, value_by_ref,
    timestamp_by_ref)
```

Parameters

fifo_no	Number (1, 2) of the input FIFO for edge detection.	LONG
value_by_ref	Variable where the level status bit patterns are written. Each level status bit corresponds to a digital input (see table below).	LONG ~~CONST~~
timestamp_by_ref	Variable where time stamps are written.	LONG ~~CONST~~

Notes

The input FIFO 1 refers to the digital inputs DIO31:DIO00, the input FIFO 2 refers to digital inputs DIO60:DIO32.

Bit no. FIFO 1	31	30	…	2	1	0
DIO input	DIO31	DIO30	…	DIO02	DIO01	DIO00

Bit no. FIFO 2	31:29	28	27	…	1	0
DIO input	–	DIO60	DIO59	…	DIO33	DIO32

Before reading you have to confirm with **Digin_Fifo_Full**, that there is at least one value pair saved in the FIFO.

The passed parameters must be variables (or array elements), not constants.

The time difference between 2 level status patterns is the difference of the appropriate time stamps, measured in units of 10ns:

$$\Delta t = 10\ \text{ns} \cdot (\text{stamp}_1 - \text{stamp}_2)$$

See also

Conf_DIO, Digin, Digin_Edge1, Digin_Edge2, Digout, Digin_Fifo_Read_Timer, Digin_Fifo_Clear, Digin_Fifo_Enable, Digin_Fifo_Full, Digout_Fifo_Enable

Valid for

X-A20+DCT

Example

```
#Include ADwin-X.inc

Dim Data_1[10000], Data_2[10000] As Long
Dim index As Long

Init:
  Conf_DIO(1100b)           'channels 15:0 as inputs
  Digin_Fifo_Enable(1,0)    'edge control off
  Digin_Fifo_Clear(1)       'clear FIFO
  Digin_Fifo_Enable(1,10011b) 'control channels 0,1,4
  index = 1

Event:
  If (Digin_Fifo_Full(1) > 0) Then
    Rem read one value pair
    Digin_Fifo_Read(1,Data_1[index], Data_2[index])
    index = index + 1
    If (index > 10000) Then index = 1
  EndIf
```

Digout_Fifo_Read_Timer

Digout_Fifo_Read_Timer returns the current value of a 100MHz counter.

Syntax

```
#Include ADwin-X.inc
ret_val = Digout_Fifo_Read_Timer(fifo_no)
```

Parameters

fifo_no	Number (1, 2) of the output FIFO.	LONG
ret_val	Current value (-2^{31}-1 … 2^{31}) of the counter.	LONG

Notes

The counter is used for exact edge output timing at predefined points of time, see **Digout_Fifo_Write**.

The output FIFO 1 refers to the digital outputs DIO31:DIO00, the output FIFO 2 refers to digital outputs DIO60:DIO32.

The counter value can only be used in the FIFO operation mode with absolute time values i.e. parameter mode = 1 in **Digout_Fifo_Mode**.

The timer value is increased by 1 every 10ns, so the timer will reach the original timer value after about 43 seconds (=10ns $\times 2^{32}$) ticks. For time comparison you have to consider this "overflow", thus the counter value must be queried regularly before an overflow happens.

The counter is set to zero with **Digout_Fifo_Clear**.

See also

Conf_DIO, Digin, Digout, Digout_Bits1, Digout_Bits2, Digin_Fifo_Read_Timer, Digout_Fifo_Clear, Digout_Fifo_Enable, Digout_Fifo_Empty, Digout_Fifo_Mode, Digout_Fifo_Start, Digout_Fifo_Write

Valid for

X-A20+DCT

Example

```
#Include ADwin-X.inc
Rem provide number of counter overflows
#Define count_overflow Par_1
Dim t_start, diff_new, diff_old As Long

Init:
  count_overflow = 0        'overflow occurs every 43 seconds
  t_start = Digout_Fifo_Read_Timer()
  diff_old = 0

Event:
  Rem Event section must be run at least once every 20 seconds.
  Rem Else you will miss counter overflows.

  Rem get timer difference
  diff_new = Digout_Fifo_Read_Timer() - t_start
  If ((diff_new > 0) And (diff_old < 0)) Then
    Inc(count_overflow)      'increase number of counter overflows
  EndIf
  diff_old = diff_new
```

other examples see

– *ADbasic* example in folder
C:\ADwin\ADbasic\samples_ADwin: seconds_timer.bas

Digout_Fifo_Clear

Digout_Fifo_Clear stops the edge output and clears the edge output FIFO.

Syntax

```
#Include ADwin-X.inc
Digout_Fifo_Clear(fifo_no)
```

Parameters

fifo_no	Number (1, 2) of the output FIFO for edge output.	LONG

Notes

Before first use, the FIFO must be cleared. Then, the FIFO can be filled with data using **Digout_Fifo_Write**.

If the edge output has been stopped with **Digout_Fifo_Clear**, it can only be started with **Digout_Fifo_Start** again.

The output FIFOs can also be cleared with **Sync_All**.

See also

Digin, Digout, Digout_Bits1, Digout_Bits2, Digin_Fifo_Clear, Digout_Fifo_Read_Timer, Digout_Fifo_Enable, Digout_Fifo_Empty, Digout_Fifo_Mode, Digout_Fifo_Start, Digout_Fifo_Write, Sync_All

Valid for

X-A20+DCT

Example

see Digout_Fifo_Mode

Digout_Fifo_Enable

Digout_Fifo_Enable sets the output channels where edges are output.

Syntax

```
#Include ADwin-X.inc
Digout_Fifo_Enable(fifo_no,pattern)
```

Parameters

fifo_no	Number (1, 2) of the output FIFO for edge output.	LONG
pattern	Bit pattern to select the output channels for edge output.	LONG

Notes

The output FIFO 1 refers to the digital outputs DIO31:DIO00, the output FIFO 2 refers to digital outputs DIO60:DIO32.

Bit no. FIFO 1	31	30	…	2	1	0
DIO output	DIO31	DIO30	…	DIO02	DIO01	DIO00

Bit no. FIFO 2	31:29	28	27	…	1	0
DIO output	–	DIO60	DIO59	…	DIO33	DIO32

Edges can only be output to output channels. The specified channels must be first programmed as outputs using **Conf_DIO**.

Conf_DIO configures digital channels DIO41:DIO00 as inputs or outputs in groups. The channels DIO60:DIO42 are always configured as inputs.

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 **Digout_Long**.

The levels and points of time of egde output are set with **Digout_Fifo_Write**.

See also

Conf_DIO, Digin, Digout, Digout_Bits1, Digout_Bits2, Digin_Fifo_Enable, Digout_Fifo_Read_Timer, Digout_Fifo_Clear, Digout_Fifo_Empty, Digout_Fifo_Mode, Digout_Fifo_Start, Digout_Fifo_Write

Valid for

X-A20+DCT

Example

```
#Include ADwin-X.inc
Dim Data_1[10000], Data_2[10000] As Long

Init:
  Conf_DIO(1111b)           'channels 0:31 as outputs
  Digout_Fifo_Clear(1)      'clear FIFO
  Digout_Fifo_Enable(1,101010b)'edge output on channels 1,3,5
  Rem write 3 value pairs into output FIFO and start output
  Rem 100ns: channels 1,3,5
  Rem 300ns: channels 1,3
  Rem 500ns: channels 3,5
  Digout_Fifo_Write(1,101010b,10)
  Digout_Fifo_Write(1,001010b,30)
  Digout_Fifo_Write(1,101000b,50)
  Digout_Fifo_Start(1)      'clear FIFO

Event:
  Rem write new value pairs into FIFO, if possible
  If (Digout_Fifo_Empty(1) > 1) Then
    Digout_Fifo_Write(1,100000b,1)
  EndIf
  If (Digout_Fifo_Empty(1) > 20) Then
    Digout_Fifo_Write(1,101010b,10)
    Digout_Fifo_Write(1,001010b,30)
    Digout_Fifo_Write(1,101000b,50)
  EndIf
```

Digout_Fifo_Empty

`Digout_Fifo_Empty` returns the number of free value pairs in the edge output FIFO.

Syntax

```
#Include ADwin-X.inc
ret_value = Digout_Fifo_Empty(fifo_no)
```

Parameters

`fifo_no`	Number (1, 2) of the output FIFO for edge output.	LONG
`ret_value`	Number (0…511) of free value pairs in the FIFO.	LONG

Notes

The output FIFO 1 refers to the digital outputs DIO31:DIO00, the output FIFO 2 refers to digital outputs DIO41:DIO32.

The FIFO may contain 511 value pairs (level status and time stamp) in maximum.

See also

Digin, Digout, Digout_Bits1, Digout_Bits2, Digin_Fifo_Full, Digout_Fifo_Read_Timer, Digout_Fifo_Clear, Digout_Fifo_Enable, Digout_Fifo_Mode, Digout_Fifo_Start, Digout_Fifo_Write

Valid for

X-A20+DCT

Example

see Digout_Fifo_Mode

Digout_Fifo_Mode

Digout_Fifo_Mode sets the FIFO operation mode of the edge output.

Syntax

```
#Include ADwin-X.inc
Digout_Fifo_Mode(fifo_no,mode)
```

Parameters

fifo_no	Number (1, 2) of the output FIFO for edge output.	LONG
mode	Operation mode of the FIFO edge output: 1: Output FIFO with edge output, time values with absolute reference. 3: Output FIFO with edge output, time values with relative reference.	LONG

Notes

The output FIFO 1 refers to the digital outputs DIO31:DIO00, the output FIFO 2 refers to digital outputs DIO41:DIO32.

Time stamps of an output FIFO set the time when an edge is output (see **Digout_Fifo_Write**). 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 100MHz counter (**Digout_Fifo_Start**).
  Using this mode, the current counter value can be read with **Digout_Fifo_Read_Timer**.
- Relative value: The time stamp is counted relative to the previous time stamp.

The list of value pairs can be filled up–as long as there are any value pairs in the FIFO.

See also

Conf_DIO, Digin, Digout, Digout_Bits1, Digout_Bits2, Digout_Fifo_Read_Timer, Digout_Fifo_Clear, Digout_Fifo_Enable, Digout_Fifo_Empty, Digout_Fifo_Start, Digout_Fifo_Write

Valid for

X-A20+DCT

Example

```
#Include ADwin-X.inc
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)
 Conf_DIO(01111b)             'set DIO31:00 as output
 Digout_Fifo_Mode(1,3)        'Set FIFO1 as relative output
 Digout_Fifo_Clear(1)         'clear FIFO
 Digout_Fifo_Enable(1,11b)    'Enable output channels 0+1
 Rem write 2 value pairs into output FIFO and start output
 Digout_Fifo_Write(1,value[1],value[2])
 Digout_Fifo_Write(1,value[3],value[4])
 Digout_Fifo_Start(01b)

Event:
 Rem write new value pairs into FIFO, if possible
 If (Digout_Fifo_Empty(1) >= 2) Then
  Digout_Fifo_Write(1,value[1],value[2])
  Digout_Fifo_Write(1,value[3],value[4])
 EndIf
```

Digout_Fifo_Start

Digout_Fifo_Start starts the edge output on selected output FIFOs.

Syntax

```
#Include ADwin-X.inc
Digout_Fifo_Start(fifo_pattern)
```

Parameters

fifo_pattern	Bit pattern to access the FIFOs: Bit = 0: Ignore FIFO. Bit = 1: Start edge output on the FIFO.	LONG

Bits in fifo_pattern	31:2	1	0
FIFO number	–	2	1

Notes

After start of the edge output, the counter starts to count with 0. The counter is used to do exact output timing, see **Digout_Fifo_Write**.

The timer value is increased by 1 every 10ns, so the timer will reach the original timer value after about 43 seconds (=10ns $\times$ 2^{32}) ticks. For time comparison you have to consider this "overflow", thus the counter value must be queried regularly before an overflow happens. The counter runs at a clock rate of 100MHz.

You can also start the edge output with **Sync_All**.

See also

Digin, Digout, Digout_Bits1, Digout_Bits2, Digout_Fifo_Read_Timer, Digout_Fifo_Clear, Digout_Fifo_Enable, Digout_Fifo_Empty, Digout_Fifo_Mode, Digout_Fifo_Write, Sync_All

Valid for

X-A20+DCT

Example

see Digout_Fifo_Mode

Digout_Fifo_Write

Digout_Fifo_Write writes one value pair into the output edge FIFO.

Syntax

```
#Include ADwin-X.inc
Digout_Fifo_Write(fifo_no, level_pattern, timestamp)
```

Parameters

fifo_no	Number (1, 2) of the output FIFO for edge output.	LONG
level_pattern	Bit pattern of level status to be output. Bit=0: TTL level low. Bit=1: TTL level high. Each bit corresponds to a digital output (see table).	LONG
timestamp	Time stamp (in steps of 10ns) referring to level_pattern, which sets the time of output.	LONG

Notes

The output FIFO 1 refers to the digital outputs DIO31:DIO00, the output FIFO 2 refers to digital outputs DIO41:DIO32.

Bit no. FIFO 1	31	30	…	2	1	0
DIO output	DIO31	DIO30	…	DIO02	DIO01	DIO00

Bit no. FIFO 2	31:10	9	8	…	1	0
DIO output	–	DIO41	DIO40	…	DIO33	DIO32

You must not write more value pairs into the FIFO than are free. The number of free values in the FIFO is returned with **Digout_Fifo_Empty**.

The FIFO array may contain 511 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 time stamp can be given with absolute or relative reference, see **Dig_Fifo_Mode**. The difference between two output times must be at least 20ns. The value of a time stamp is counted in processor clocks i.e. in units of 10ns.

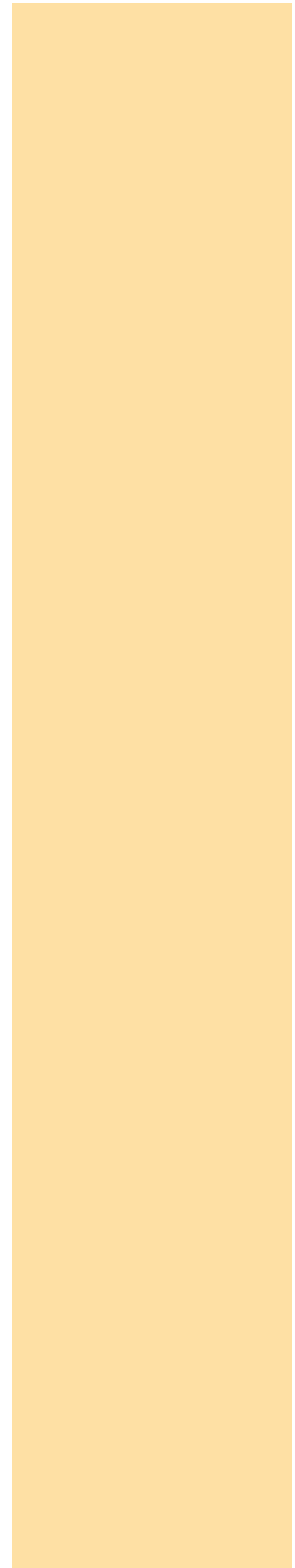
The edge output runs like follows:
- The 100MHz counter is increased by 1 every 10ns.
- If the counter value equals the time stamp of the current value pair in the FIFO, the bit pattern is output to the specified output channels.
- If a bit pattern has been output, the value pair is deleted from the FIFO.
- The value pairs are processed in the order as they were written into the FIFO.

Therefore:

A time stamp defines the exact output time, and in time units of 10ns. The value can be given in two ways:
- As absolute value in relation to the starting time of the 100MHz counter using **Digout_Fifo_Start**.
  A time stamp of 152 would have the appropriate bit pattern be output exactly at 1.52µs after the 100MHz 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 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 anyway, 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 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

Conf_DIO, Digin, Digout, Digout_Bits1, Digout_Bits2, Digin_Fifo_Enable, Digout_Fifo_Read_Timer, Digout_Fifo_Clear, Digout_Fifo_Enable, Digout_Fifo_Empty, Digout_Fifo_Mode, Digout_Fifo_Start

Valid for

X-A20+DCT

Example

see Digout_Fifo_Mode

16.4 Counter

Dieser Abschnitt beschreibt folgende Befehle:

- Cnt_Clear (page 111)
- Cnt_Enable (page 112)
- Cnt_PW_Enable (page 113)
- Cnt_Get_Status (page 114)
- Cnt_Latch (page 116)
- Cnt_Mode (page 117)
- Cnt_Read (page 119)
- Cnt_PW_Latch (page 120)
- Cnt_Read_Int_Register (page 121)
- Cnt_Get_PW (page 122)
- Cnt_Get_PW_HL (page 123)
- Cnt_Read_Latch (page 124)
- Cnt_Sync_Latch (page 125)
- SSI_Mode (page 128)
- SSI_Read (page 129)
- SSI_Set_Bits (page 130)
- SSI_Set_Clock (page 131)
- SSI_Start (page 133)
- SSI_Status (page 134)

Cnt_Clear

Cnt_Clear sets one or more up/down counters to zero, according to a bit pattern.

Syntax

```
#Include ADwin-X.inc
Cnt_Clear(pattern)
```

Parameters

pattern — Bit pattern to select counters. Bit = 0: no influence. Bit = 1: set counter to zero. — LONG

Bit no.	31:7	6	5	4	3	2	1	0
Counter no.	–	7	6	5	4	3	2	1

Notes

After **Cnt_Clear** has been executed the bit pattern is automatically reset to 0 (zero), so the counters start counting from 0.

Please pay attention to set **Cnt_Mode** parameter pattern to bit 1=0 for the appropriate counters. Else, with bit 1=1, the counter inputs A, B have also to be set to TTL level high, in order to clear the counter.

With **Sync_All**, you can set all 7 counters (and all 7 PWM counters) to zero at the same time.

See also

Cnt_Enable, Cnt_Get_Status, Cnt_Latch, Cnt_Mode, Cnt_Read, Cnt_Read_Int_Register, Cnt_Read_Latch, Cnt_Sync_Latch, Sync_All

Valid for

X-A20+D, X-A20+DCT, X-A20+CO1

Example

```
#Include ADwin-X.inc

Init:
  Cnt_Enable(0)              'stop all counters
  Cnt_Mode(2,0b)             'Counter 2 clock/direction
  Cnt_Mode(3,0b)             'Counter 3 clock/direction
  Cnt_Clear(110b)            'reset counters 2+3 to 0
  Cnt_Enable(110b)           'start counters 2+3

Event:
  Cnt_Latch(110b)            'latch counters 2+3
  Par_1 = Cnt_Read_Latch(2) 'read latch counter 2 and …
  Par_2 = Cnt_Read_Latch(3) 'latch counter 3
```

Cnt_Enable

Cnt_Enable disables or enables the up/down counters set by pattern, to count incoming impulses.

Syntax

```
#Include ADwin-X.inc
Cnt_Enable(pattern)
```

Parameters

pattern   Bit pattern.   LONG
Bit = 0: stop counter.
Bit = 1: enable counter.

Bit no.	31:7	6	5	4	3	2	1	0
Counter no.	–	7	6	5	4	3	2	1

Notes

PWM counters are enabled or disabled with **Cnt_PW_Enable**.

With counters 1, 2, and 3, you must set the inputs as digital inputs with **Conf_DIO** to make the counters work.

See also

Cnt_Clear, Cnt_Get_Status, Cnt_Latch, Cnt_Mode, Cnt_Read, Cnt_Read_Int_Register, Cnt_Read_Latch, Cnt_Sync_Latch, Cnt_Sync_Latch, Conf_DIO

Valid for

X-A20+D, X-A20+DCT, X-A20+CO1

Example

```
#Include ADwin-X.inc

Init:
 Cnt_Enable(0)              'stop all counters
 Cnt_Mode(1,0b)             'counter 1 mode clock-direction
 Cnt_Mode(2,0b)             'counter 1 mode clock-direction
 Cnt_Clear(11b)             'reset counters 1+2 to 0
 Cnt_Enable(11b)            'start counters 1+2

Event:
 Cnt_Latch(11b)             'latch counters 1+2
 Par_1 = Cnt_Read_Latch(1)  'read counter latch 1
 Par_2 = Cnt_Read_Latch(2)  'read counter latch 2
```

Cnt_PW_Enable

`Cnt_PW_Enable` enables or disables the PWM counters selected by `pattern`.

Syntax

```
#Include ADwin-X.Inc
Cnt_PW_Enable(pattern)
```

Parameters

`pattern` Bit pattern LONG
Bit = 0: Disable counter.
Bit = 1: Enable counter.

Bit no.	31:7	6	5	4	3	2	1	0
Counter no.	–	7	6	5	4	3	2	1

Notes

Up/down counters are started or stopped with `Cnt_Enable`.

The PWM counter input is set with `Cnt_Mode`.

See also

Cnt_Clear, Cnt_Get_Status, Cnt_Mode, Cnt_PW_Latch, Cnt_Read_Int_Register, Cnt_Get_PW, Cnt_Get_PW_HL, Cnt_Sync_Latch

Valid for

X-A20+D, X-A20+DCT, X-A20+CO1

Example

```
#Include ADwin_X.inc

Init:
 Cnt_PW_Enable(0)            'stop all PW counters
 Rem counters 1+2: mode clock/dir, PWM at input CLK
 Cnt_Mode(1,0) 'counter 1: mode clock/dir, PWM input CLK
 Cnt_Mode(2,0) 'counter 2: mode clock/dir, PWM input CLK
 Cnt_PW_Enable(11b)          'start PWM counters 1+2

Event:
 Cnt_PW_Latch(11b)           'latch PWM counters 1+2
 Cnt_Get_PW_HL(1,Par_1,Par_2) 'read high/low time
 Cnt_Get_PW(1,FPar_1,FPar_2)'read frequency and duty cycle
```

Cnt_Get_Status

`Cnt_Get_Status` returns the status register of one counter block.

Syntax

```
#Include ADwin-X.inc
ret_val = Cnt_Get_Status(counter_no)
```

Parameters

counter_no	Counter block number: 1…7.	LONG
ret_val	Contents of 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 (signal status is not defined, mask out 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, i.e. they are not phase-shifted by approx. 90°)

Notes

A line error (Lx) can only be detected at differential inputs! For TTL-inputs these bits are always 0.

The status register is automatically reset by reading.

See also

Cnt_Enable, Cnt_Get_PW, Cnt_Mode, Cnt_Read

Valid for

X-A20+D, X-A20+DCT, X-A20+CO1

Example

```
#Include ADwin-X.inc
Dim error As Long

Init:
 Cnt_Enable(0)                'stop counter
 Cnt_Mode(1,0)                'counter 1: mode clock/dir
 Cnt_Clear(1b)                'reset counter 1 to 0
 Cnt_Enable(1b)               'start counter 1
 error = 0                    'reset error flag

Event:
 PAR_1 = Cnt_Read(1)          'read counter 1
 PAR_2 = Cnt_GetStatus(1) And 11111b 'Status
 REM line or cable error at counter 1?
 If (PAR_2 AND 01000b = 01000b) Then
  REM number of line/cable errors
  Inc PAR_3
  error = 1                   'set error flag
 EndIf
 REM correlation error at counter 1?
 If (PAR_2 And 10000b = 10000b) Then
  Inc PAR_4                   'number correlation errors
  error = 1                   'set error flag
 EndIf
 REM status input CLR
 PAR_5 = Shift_Right(PAR_2 And 100b,2)
 REM status input A
 PAR_6 = Shift_Right(PAR_2 And 10b,1)
 REM status input B
 PAR_7 = PAR_2 And 1b
```

Cnt_Latch

Cnt_Latch transfers the current counter values of one or more up/down counters into the relevant Latch A, depending on the bit pattern.

Syntax

```
#Include ADwin-X.inc
Cnt_Latch(pattern)
```

Parameters

pattern — Bit pattern. LONG
Bit = 0: no function.
Bit = 1: transfer counter values into the latch.

Bit no.	31:6	6	5	4	3	2	1	0
Counter no.	–	7	6	5	4	3	2	1

Notes

After **Cnt_Latch** has been executed the bit pattern is automatically reset to 0 (zero).

The latch is read out into a variable with **Cnt_Read_Latch** command.

For PWM counters use **Cnt_PW_Latch**. In order to latch several counter values synchronously, use **Cnt_Sync_Latch**.

See also

Cnt_Clear, Cnt_Enable, Cnt_Get_Status, Cnt_PW_Latch, Cnt_Mode, Cnt_Read, Cnt_Read_Latch, Cnt_Sync_Latch

Valid for

X-A20+D, X-A20+DCT, X-A20+CO1

Example

```
#Include ADwin-X.inc

Init:
  Cnt_Enable(0)              'stop all counters
  Cnt_Mode(1,0b)             'counter 1 clock/direction
  Cnt_Mode(2,0b)             'counter 2 clock/direction
  Cnt_Clear(11b)             'reset counters 1+2 to 0
  Cnt_Enable(11b)            'start counters 1+2

Event:
  Cnt_Latch(11b)             'latch counters 1+2
  Par_1 = Cnt_Read_Latch(1) 'read latch counter 1
  Par_2 = Cnt_Read_Latch(2) 'read latch counter 2
```

Cnt_Mode

`Cnt_Mode` defines the operating mode of one counter block.

Syntax

```
#Include ADwin-X.inc
Cnt_Mode(cnt_no,pattern)
```

Parameters

cnt_no	Counter block number: 1…7.	LONG
pattern	Bit pattern to set the operating mode of a counter.	LONG

Bit no.	Meaning
Bit 0	Up/down counter mode: Bit = 0: mode clock/direction. Bit = 1: mode A-B.
Bit 1	Clear mode. Signal condition, which clears the up/down counter: Bit = 0: TTL level high at input CLR. Bit = 1: TTL level high at all inputs A, B, CLR. Available in mode A-B only.
Bit 2	Invert input A / CLK in mode clock/direction: Bit = 0: Input is not inverted. Bit = 1: Input is inverted.
Bit 3	Invert input B / DIR in mode clock/direction: Bit = 0: Input is not inverted. Bit = 1: Input is inverted.
Bit 4	Set use of input CLR / LTC. Bit = 0: CLR input: clear counter. Bit = 1: LTC input: latch counter.
Bit 5	Enable input CLR / LTC. Bit = 0: Input CLR / LTC is disabled. Bit = 1: Input CLR / LTC is enabled.
Bit 6	Select edge for PWM counter. Bit = 0: rising edge. Bit = 1: falling edge.
Bits 7,8	Select input for PWM counter. 00b: Input A / CLK 01b: Input B / DIR 10b: Input CLR / LTC
Bits 31:9	reserved

Notes

Please use `Cnt_Mode` only when the counter is disabled, see `Cnt_Enable` and `Cnt_PW_Enable`.

With counters 1, 2, and 3, you must set the inputs as digital inputs with `Conf_DIO` to make the counters work.

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.

If you want to clear a counter with `Cnt_Clear` set `pattern` bit 1=0. Else, with bit 1=1, the counter inputs A, B have also to be set to TTL level high, in order to clear the counter.

Plaese note: You can suppress spikes of incoming signals with `Digin_Filter_Init`.

See also

Cnt_Clear, Cnt_Enable, Cnt_Get_Status, Digin_Filter_Init, Conf_DIO

Valid for

X-A20+D, X-A20+DCT, X-A20+CO1

Example

```
#Include ADwin-X.inc

Init:
  Cnt_Enable(0)                 'stop all counters
  Cnt_Mode(1,0b)                'counter 1 clock/direction
  Cnt_Mode(2,0b)                'counter 2 clock/direction
  Cnt_Clear(11b)                'reset counters 1+2 to 0
  Cnt_Enable(11b)               'start counters 1+2

Event:
  Cnt_Latch(11b)                'latch counters 1+2
  Par_1 = Cnt_Read_Latch(1)     'read latch counter 1
  Par_2 = Cnt_Read_Latch(2)     'read latch counter 2
```

Cnt_Read

`Cnt_Read` transfers a current up/down counter value into Latch A and returns the value.

Syntax

```
#Include ADwin-X.inc
ret_val = Cnt_Read(counter_no)
```

Parameters

counter_no	Up/down counter number: 1…7.	LONG
ret_val	Counter value.	LONG

Notes

Use the return value in calculations only with variables of the type `Long` (e.g. differences or count direction).

See also

Cnt_Clear, Cnt_Enable, Cnt_Get_Status, Cnt_Latch, Cnt_Mode, Cnt_Read_Latch, Cnt_Sync_Latch

Valid for

X-A20+D, X-A20+DCT, X-A20+CO1

Example

```
#Include ADwin-X.inc

Init:
 Cnt_Enable(0)                'stop all counters
 Rem Counter 1: Mode clock/direction, enable CLR
 Cnt_Mode(1,100000b)
 Rem Counter 2: Mode clock/direction, enable LATCH
 Cnt_Mode(2,110000b)
 Cnt_Clear(11b)               'reset counters 1+2
 Cnt_Enable(11b)              'start counters 1+2

Event:
 Par_1 = Cnt_Read(1) 'read counter 1
 Par_2 = Cnt_Read(2) 'read counter 2
```

Cnt_PW_Latch

Cnt_PW_Latch copies the value of one or more PWM counters into a buffer.

Syntax

```
#Include ADwin-X.inc
Cnt_PW_Latch(pattern)
```

Parameters

pattern — Bit pattern. LONG
Bit = 0: no function.
Bit = 1: transfer PWM counter value into a buffer.

Bit no.	31:6	6	5	4	3	2	1	0
Counter no.	–	7	6	5	4	3	2	1

Notes

The buffer is to be read with **Cnt_Get_PW** or **Cnt_Get_PW_HL**.

See also

Cnt_Clear, Cnt_PW_Enable, Cnt_Get_Status, Cnt_Mode, Cnt_Read_Int_Register, Cnt_Get_PW, Cnt_Get_PW_HL, Cnt_Sync_Latch

Valid for

X-A20+D, X-A20+DCT, X-A20+CO1

Example

```
#Include ADwin-X.inc

Init:
  Cnt_PW_Enable(0)           'stop all counters
  Rem Counters 1+2: mode clock/dir, PWM input CLK
  Cnt_Mode(1,0)
  Cnt_Mode(2,0)
  Cnt_PW_Enable(11b)         'start PWM counters 1+2

Event:
  Cnt_PW_Latch(11b)          'latch counters 1+2
  REM read high/low time
  Cnt_Get_PW_HL(1,Par_1,Par_2)
  REM read frequency/duty cycle
  Cnt_Get_PW(1,FPar_1,FPar_2)
```

Cnt_Read_Int_Register

Cnt_Read_Int_Register returns the content of a counter register.

Syntax

```
#Include ADwin-X.inc
ret_val = Cnt_Read_Int_Register(counter_no,reg_no)
```

Parameters

counter_no	Counter block number: 1…7.	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

The registers above are assigned to each PWM counter. If PWM counters are evaluated with standard instructions **Cnt_Get_PW** and **Cnt_Get_PW_HL**, no further knowledge is required about PWM registers. Use the evaluation with PWM registers for special solutions only.

Register contents are set with **Cnt_PW_Latch** or **Cnt_Sync_Latch**.

See also

Cnt_Get_PW, Cnt_Get_PW_HL, Cnt_PW_Latch, Cnt_Sync_Latch

Valid for

X-A20+D, X-A20+DCT, X-A20+CO1

Example

see Cnt_Sync_Latch

Cnt_Get_PW

Cnt_Get_PW returns frequency and duty cycle of a PWM counter.

Syntax

```
#Include ADwin-X.inc
Cnt_Get_PW(pwm_no,frequency,dutycycle)
```

Parameters

pwm_no	PWM counter number: 1…7.	LONG
frequency	Frequency in Hertz: 0,023 Hz …20MHz.	FLOAT CONST
dutycycle	Duty cycle in percent: 0.0…100.0.	FLOAT CONST

Notes

Use Cnt_PW_Latch first, to receive current values.

The return values are given in the parameters frequency and dutycycle.

See also

Cnt_Clear, Cnt_PW_Enable, Cnt_Get_Status, Cnt_Get_PW_HL, Cnt_Mode, Cnt_PW_Latch, Cnt_Read_Int_Register, Cnt_Sync_Latch

Valid for

X-A20+D, X-A20+DCT, X-A20+CO1

Example

```
#Include ADwin-X.inc

Init:
  Cnt_PW_Enable(0)              'stop all counters
  Rem Counters 1+2: mode clock/dir, PWM input CLK
  Cnt_Mode(1,0)
  Cnt_Mode(2,0)
  Cnt_PW_Enable(11b)            'start PWM counters 1+2

Event:
  Cnt_PW_Latch(11b)             'latch counters 1+2
  REM read high/low time
  Cnt_Get_PW_HL(1,Par_1,Par_2)
  Cnt_Get_PW_HL(2,Par_11,Par_12)
  REM read frequency/duty cycle
  Cnt_Get_PW(1,FPar_1,FPar_2)
  Cnt_Get_PW(2,FPar_11,FPar_12)
```

Cnt_Get_PW_HL

Cnt_Get_PW_HL returns the high time and low time of a PWM counter.

Syntax

```
#Include ADwin-X.inc
Cnt_Get_PW_HL(counter_no,hightime,lowtime)
```

Parameters

counter_no	PWM counter number: 1…7.	LONG
hightime	Pulse duration in units of 10ns: PWM high level time.	LONG CONST
lowtime	Pulse period in units of 10ns: PWM low level time.	LONG CONST

Notes

Use **Cnt_PW_Latch** first, to receive current values.

The return values are given in the parameters hightime and lowtime.

See also

Cnt_Clear, Cnt_PW_Enable, Cnt_Get_Status, Cnt_Get_PW, Cnt_Mode, Cnt_PW_Latch, Cnt_Read_Int_Register, Cnt_Sync_Latch

Valid for

X-A20+D, X-A20+DCT, X-A20+CO1

Example

```
#Include ADwin-X.inc

Init:
 Cnt_PW_Enable(0)             'stop all counters
 Rem Counters 1+2: mode clock/dir, PWM input CLK
 Cnt_Mode(1,0)
 Cnt_Mode(2,0)
 Cnt_PW_Enable(11b)           'start PWM counters 1+2

Event:
 Cnt_PW_Latch(11b)            'latch counters 1+2
 REM read high/low time
 Cnt_Get_PW_HL(1,Par_1,Par_2)
 Cnt_Get_PW_HL(2,Par_11,Par_12)
 REM read frequency/duty cycle
 Cnt_Get_PW(1,FPar_1,FPar_2)
 Cnt_Get_PW(2,FPar_11,FPar_12)
```

Cnt_Read_Latch

`Cnt_Read_Latch` returns the value of a up/down counter's latch.

Syntax

```
#Include ADwin-X.inc
ret_val = Cnt_Read_Latch(counter_no)
```

Parameters

counter_no	Counter number: 1…7.	LONG
ret_val	Content of counter latch.	LONG

Notes

Use the return value in calculations only with variables of the type `Long` (e.g. differences or count direction).

See also

Cnt_Clear, Cnt_Enable, Cnt_Get_Status, Cnt_Latch, Cnt_Mode, Cnt_Read, Cnt_Sync_Latch

Valid for

X-A20+D, X-A20+DCT, X-A20+CO1

Example

```
#Include ADwin-X.inc

Init:
  Cnt_Enable(0)                 'stop all counters
  Rem counter 2 clock/dir, enable latch input
  Cnt_Mode(2,110000b)
  Cnt_Clear(10b)                'reset counter 2 to 0
  Cnt_Enable(10b)               'start counter 2

Event:
  Par_10 = Cnt_Read_Latch(2)  'read counter latch 2
```

Cnt_Sync_Latch

Cnt_Sync_Latch copies the contents of the selected up/down counters and PWM counters into buffers.

Syntax

```
#Include ADwin-X.inc
Cnt_Sync_Latch(pattern)
```

Parameters

pattern — Bit pattern — LONG
Bit = 0: No function.
Bit = 1: Copy counter content into a buffer.

Bit no.	31:6	6	5	4	3	2	1	0
Counter no.	–	7	6	5	4	3	2	1

Notes

Each bit is assigned to both an up/down counter and a PWM counter. For each set bit, both counter contents are copied simultaneously. The instruction therefore has the same function as **Cnt_Latch** and **Cnt_PW_Latch** together.

The buffers can be read e.g. with **Cnt_Read_Latch** or **Cnt_Get_PW**.

With **Sync_All**, you can latch all 7 counters and 7 PWM counters at the same time.

See also

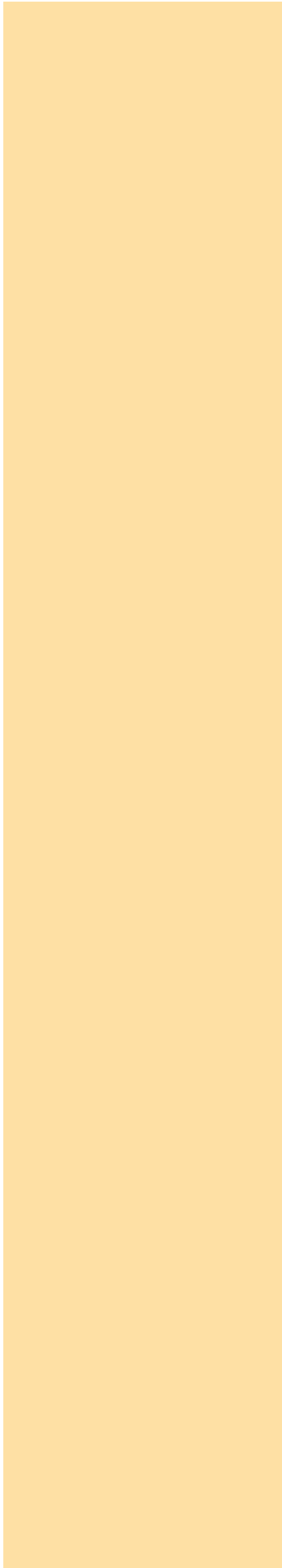
Cnt_Get_PW, Cnt_Latch, Cnt_Mode, Cnt_PW_Latch, Cnt_Read_Int_Register, Cnt_Read_Latch, Sync_All

Valid for

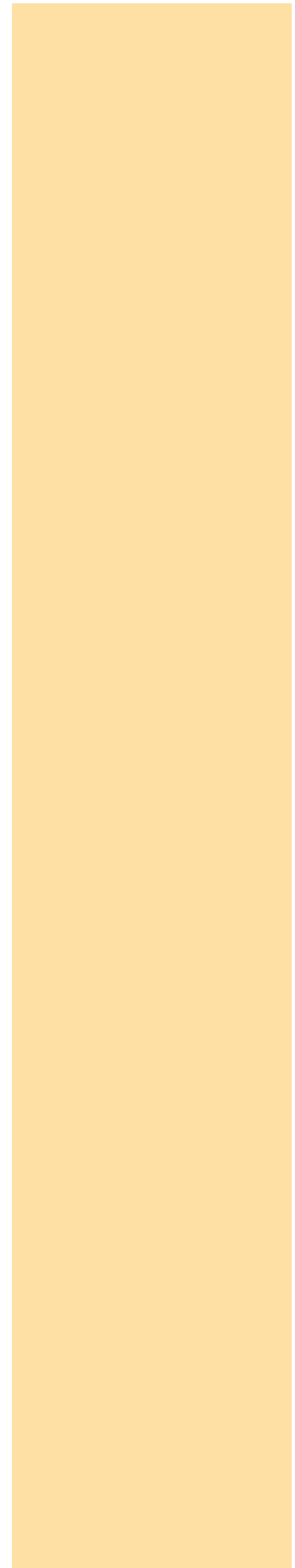
X-A20+D, X-A20+DCT, X-A20+CO1

Example

```
#Include ADwin-X.inc
#Define frequency PAR_1
Dim time, edges As Long
Dim pw, oldpw As Long
Dim vr, oldvr As Long

Init:
  Processdelay = 3000000        '200Hz with T12 processor
  Cnt_Enable(0)                 'counters off
  Cnt_PW_Enable(0)              'PWM counters off
  Cnt_Mode(1,00000000b)         'mode: clock/dir
  Cnt_Clear(0001b)              'clear counter 1
  Cnt_Enable(1b)                'enable V/R 1
  Cnt_PW_Enable(1b)             'enable PWM 1
  Cnt_Sync_Latch(0001b)  'latch counter 1 (V/R + PWM)
  oldvr = Cnt_Read_Int_Register(1,6)  'V/R counter 1
  oldpw = Cnt_Read_Int_Register(1,8)  'PWM counter 1
  frequency = 0

Event:
  Cnt_Sync_Latch(0001b)         'latch counter 1 (V/R + PWM)
  vr = Cnt_Read_Int_Register(1,6)  'V/R counter 1
  edges = Abs(vr - oldvr)       'number of edges between events
  If (edges <> 0) Then
    Rem get positive edges latch 1
    pw = Cnt_Read_Int_Register(1,8)
    time = pw - oldpw           'calculate time base
    Rem frequency: 100000000=timer frequency of CNT module
    frequency = edges * 100000000 / time
    oldvr = vr                  'store VR counter value
    oldpw = pw                  'store PW counter value
  EndIf
```



16.5 SSI interface



This section describes instructions to access SSI decoders:

– SSI_Mode (page 128)
– SSI_Read (page 129)
– SSI_Set_Bits (page 130)
– SSI_Set_Clock (page 131)
– SSI_Set_Delay (page 132)
– SSI_Start (page 133)
– SSI_Status (page 134)

SSI_Mode

`SSI_Mode` sets the operation mode of the SSI decoder, either "single shot" (read out once) or "continuous" (read out continuously).

Syntax

```
#Include ADwin-X.inc
SSI_Mode(pattern)
```

Parameters

`pattern`	Operation mode of the SSI decoder. 0: "Single shot" mode, the encoder is read out once. 1: "Continuous" mode, the encoder is read out continuously.	LONG

Notes

If you select the mode "continuous", reading the encoder is started immediately. `SSI_Start` is not necessary for this. With `SSI_Set_Delay` you set the time distance between reading two consecutive 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.

Plaese note: You can suppress spikes of incoming signals with `Digin_Filter_Init`.

See also

SSI_Read, SSI_Set_Bits, SSI_Set_Clock, SSI_Set_Delay, SSI_Start, SSI_Status, Digin_Filter_Init

Valid for

X-A20+D, X-A20+DCT

Example

```
#Include ADwin-X.inc

Init:
  SSI_Set_Clock(200)        'clock rate = 125 kHz
  SSI_Set_Bits(1,23)        'number of bits = 23
  SSI_Mode(1)               'Continuous mode

Event:
  Par_1 = SSI_Read(1)  'read position value
```

SSI_Read

`SSI_Read` returns the last saved counter value of the SSI counter.

Syntax

```
#Include ADwin-X.inc
ret_val = SSI_Read(dcdr_no)
```

Parameters

`dcdr_no`	Number (1) of the SSI decoder.	`LONG`
`ret_val`	Most recent counter value of the SSI decoder (= absolute value position of the encoder).	`LONG`

Notes

An encoder value is saved when the number of bits indicated by `SSI_Set_Bits` are read.

In any case, the amount of bits is returned that is set before by `SSI_Set_Bits`, even if this does not correspond to the resolution of the encoder.
If so, 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

SSI_Mode, SSI_Set_Bits, SSI_Set_Clock, SSI_Set_Delay, SSI_Start, SSI_Status

Valid for

X-A20+D, X-A20+DCT

Example

```
#Include ADwin-X.inc
Dim m, n, y As Long

Init:
  SSI_Set_Clock(50)          'clock rate = 500 kHz
  SSI_Set_Bits(1,23)         'number of bits = 23
  SSI_Mode(1)                'Continuous mode

Event:
  Par_1 = SSI_Read(1)        'read position value
  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
```

SSI_Set_Bits

SSI_Set_Bits sets the amount of bits of a SSI decoder, which generate a complete encoder value.

The number of bits should be similar to the resolution of the encoder.

Syntax

```
#Include ADwin-X.inc
SSI_Set_Bits(dcdr_no, no_bits)
```

Parameters

dcdr_no	Number (1) of the SSI decoder.	LONG
no_bits	Amount (1…32) of bits, which are to be read for the encoder (corresponds to the encoder resolution).	LONG

Notes

The resolution (amount of bits) of the SSI encoder should be similar to the amount of bits being transferred.

It is always expected to get that certain amount of bits for an encoder value that was indicated before by **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

SSI_Mode, SSI_Read, SSI_Set_Clock, SSI_Set_Delay, SSI_Start, SSI_Status

Valid for

X-A20+D, X-A20+DCT

Example

```
#Include ADwin-X.inc

Init:
  SSI_Set_Clock(50)          'CLK (Taktrate) = 500 kHz
  SSI_Mode(1)                'set continuous mode
  SSI_Set_Bits(1,10)         '10 bits

Event:
  Par_1 = SSI_Read(1)        'read value of decoder
```

SSI_Set_Clock

`SSI_Set_Clock` sets the clock rate (6.1 kHz to 12.5 MHz), with which the encoder is clocked.

Syntax

```
#Include ADwin-X.inc
SSI_Set_Clock(dcdr_no,prescale)
```

Parameters

`dcdr_no`	Number (1) of the SSI decoder.	`LONG`
`prescale`	Scale factor (2…4096) for setting the clock rate according to the equation: Clock rate = 25 MHz / `prescale`.	`LONG`

Notes

After start-up of the module the default scale factor of 100 is used, corresponding to 250 kHz.

With scale factors above 4095, 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

SSI_Mode, SSI_Read, SSI_Set_Bits, SSI_Set_Delay, SSI_Start, SSI_Status

Valid for

X-A20+D, X-A20+DCT

Example

```
#Include ADwin-X.inc

Init:
  SSI_Set_Clock(10)          'CLK (Taktrate) = 2.5 MHz
  SSI_Set_Bits(1,10)         '10 bits
  SSI_Mode(1)                'set continuous mode

Event:
  Par_1 = SSI_Read(1)        'read value of decoder
```

SSI_Set_Delay

SSI_Set_Delay sets the waiting time between reading two encoder values for one SSI-decoder on the specified module.

Syntax

```
#Include ADwin-X.Inc
SSI_Set_Delay(dcdr_no,delay)
```

Parameters

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

SSI_Read, SSI_Set_Bits, SSI_Set_Clock, SSI_Mode, SSI_Start, SSI_Status

Valid for

X-A20+D, X-A20+DCT

Example

```
#Include ADwin-X.Inc

Init:
  SSI_Set_Clock(50)          'CLK (clock rate) = 1 MHz
  SSI_Set_Delay(1,400)       'waiting time 8µs for decoder 1
  SSI_Set_Bits(1,10)         '10 bits for decoder 1
  SSI_Mode(1)                'Set continuous-mode

Event:
  Par_1 = SSI_Read(1)        'Read position value decoder 1
```

SSI_Start

`SSI_Start` starts the reading of the SSI encoder (only in mode "single shot").

Syntax

```
#Include ADwin-X.inc
SSI_Start(dcdr_no)
```

Parameters

dcdr_no	Number (1) of the SSI decoder.	LONG

Notes

In continuous mode, this instruction has no function, because the encoder values are nevertheless read out continuously.

An encoder value will be saved only when the amount of bits is read, which is set by `SSI_Set_Bits`.

A complete encoder value is always transferred, even if the operation mode is changing meanwhile.

You can also start reading the SSI encoder with `Sync_All`.

See also

SSI_Mode, SSI_Read, SSI_Set_Bits, SSI_Set_Clock, SSI_Set_Delay, SSI_Status, Sync_All

Valid for

X-A20+D, X-A20+DCT

Example

```
#Include ADwin-X.inc

Init:
  SSI_Set_Clock(250)          'CLK (Taktrate) = 100 kHz
  SSI_Mode(1)                 'set continuous mode
  SSI_Set_Bits(1,23)          '23 bits

Event:
  SSI_Start(1)                'Read position value
  Do : Until (SSI_Status(1) = 0)
  Rem If position value is read completely, then …
  Par_1 = SSI_Read(1)         'read position value
```

SSI_Status

`SSI_Status` returns the current read-status on the speicified module for a specified decoder.

Syntax

```
#Include ADwin-X.inc
ret_val = SSI_Status(dcdr_no)
```

Parameters

dcdr_no	Number (1) of the SSI decoder.	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 "continuous" mode, querying the status is not useful.

See also

SSI_Mode, SSI_Read, SSI_Set_Bits, SSI_Set_Clock, SSI_Set_Delay, SSI_Start

Valid for

X-A20+D, X-A20+DCT

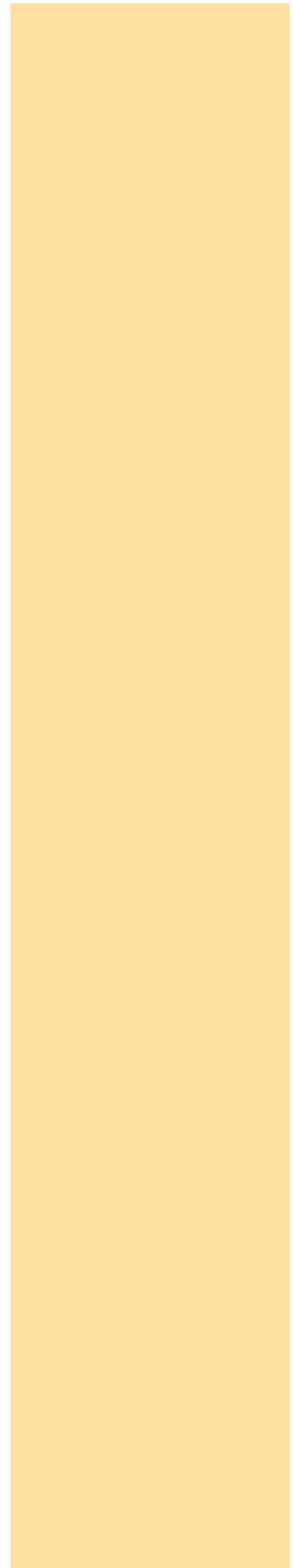
Example

```
#Include ADwin-X.inc

Init:
  SSI_Set_Clock(250)          'CLK (Taktrate) = 100 kHz
  SSI_Mode(1)                 'set continuous mode
  SSI_Set_Bits(1,23)          '23 bits

Event:
  SSI_Start(1)                'Read position value
  Do : Until (SSI_Status(1) = 0)
  Rem If position value is read completely, then …
  Par_1 = SSI_Read(1)         'read position value
```

16.6 CAN interface

This section describes instructions, which apply to *ADwin-X-A20*:

- CAN_Msg (page 136)
- CAN_Init (page 137)
- CAN_Receive (page 138)
- CAN_RX_Set_Filter (page 140)
- CAN_Transmit (page 141)



CAN_Msg

CAN_Msg is a one-dimensional array consisting of 11 elements, where a CAN message is set for sending or read after receiving.

Syntax

```
#Include ADwin-X.Inc
CAN_Msg[n] = value
```

oder

```
value = CAN_Msg[n]
```

Parameters

n	Number of an array element (1… 11).	LONG
value	Expression the value (0...256) of which is written into or read from the message object.	LONG

Notes

The array elements of CAN_Msg[] have the following function:

Element no.	Content
CAN_Msg[1…8]	CAN message from data bytes 1…8.
CAN_Msg[9]	Number (0…8) of used data bytes.
CAN_Msg[10]	ID of the CAN message (11 bit or 29 bit).
CAN_Msg[11]	Receive time stamp (16 bit).

Enter the data bytes to be transferred, the byte number, and the messsage id into the arrays Feld CAN_Msg[], *before* sending them with CAN_Transmit. The time stamp has no effect for sending.

See also

CAN_Init, CAN_Receive, CAN_RX_Set_Filter, CAN_Transmit

Valid for

X-A20+COM

Example

```
#Include ADwin-X.Inc
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 CAN_Receive)
#Define pi 3.14159265
Dim i As Long

Init:
  Rem initialize CAN Controller 1, Baud rate 10kBaud
  Par_1 = CAN_Init(1,10000)
  If (Par_1 <> 0) Then Exit

  REM Create bit pattern of Pi with data type Long
  Par_1 = Cast_Float32ToLong(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
  CAN_Msg[10] = 40           'ID

Event:
  Rem send message, low priority, 11 bit-id
  Par_2 = CAN_Transmit(1,0,0)
```

CAN_Init

CAN_Init initializes the controller of a CAN interface.

Syntax

```
#Include ADwin-X.Inc
ret_val = CAN_Init(can_no, baudrate)
```

Parameters

can_no	Number (1, 2) of CAN interface.	LONG
baudrate	Baud rate of CAN controller in bit/second.	LONG
ret_val	Status of initialization: -1: No CAN interface available. 0: Baud rate was set. 1: Invalid baud rate.	LONG

Notes

The instruction executes the following actions:

- Reset (hardware reset of the CAN controller).
- Empty receive and send Fifo.
- Disable receive filters (see **CAN_RX_Set_Filter**).
- Set baud rate.

This instruction must be executed before you can access the CAN controller with other instructions. We recommend initialization in section **LowInit:** or **Init:**.

See also

CAN_Msg, CAN_Receive, CAN_RX_Set_Filter, CAN_Transmit

Valid for

X-A20+COM

Example

```
#Include ADwin-X.Inc

Init:
 Rem initialize CAN Controller 1, Baud rate 50kBaud
 Par_1 = CAN_Init(1,50000)
 If (Par_1 <> 0) Then Exit
```

CAN_Receive

CAN_Receive returns whether a CAN message has been received in the FIFO of a CAN controller.

If yes, the oldest message of the FIFO is copied to the array **CAN_Msg**[] and the identifier is returned.

Syntax

```
#Include ADwin-X.Inc
ret_val = CAN_Receive(can_no)
```

Parameters

can_no	Number (1, 2) of CAN interface.	LONG
ret_val	-1: No new CAN message in the FIFO. >0: A new message has arrived, the value is the identifier of the message object.	LONG

Notes

You can read a received message only once, then the message is deleted from the receive FIFO pf the CAN controller.

If more than 64 messages have been recieved without reading the messages the oldest data in the input FIFO is overwritten and is lost. Therefore make sure, that CAN messages are read faster than being received. A data loss is not indicated.

The CAN id length (11 bit or 29 bit) is not returned.

With **CAN_RX_Set_Filter** you can determine to receive only CAN messages with specified identifiers.

See also

CAN_Msg, CAN_Init, CAN_RX_Set_Filter, CAN_Transmit

Valid for

X-A20+COM

Example

```
#Include ADwin-X.Inc
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 CAN_Transmit).
Dim n As Long
Dim valuePi As Float32

Init:
Par_1 = 0
 Rem initialize CAN Controller 1, Baud rate 50kBaud
 Par_1 = CAN_Init(1,50000)
 If (Par_1 <> 0) Then Exit

Event:
 REM If a message is received, the data is read and the identifier
 REM saved to Par_9.
 REM The data bytes are in the array CAN_MSG[].
 PAR_9 = CAN_Receive(1)

 If (Par_9 = 40) Then
  REM New message with identifier 40
  Par_1 = CAN_Msg[1]       'Read 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 bit pattern in Par_1 to data type Float32 and
  REM assign to the variable FPar_1.
  valuePi = Cast_LongToFloat32(Par_1)
  FPar_1 = valuePi
  Par_10 = CAN_Msg[11]     'time stamp (16 bit)
EndIf
```

Sending a float value see example of CAN_Transmit.

CAN_RX_Set_Filter

CAN_RX_Set_Filter sets a receive filter for CAN messages with a selected identifier.

Syntax

```
#Include ADwin-X.Inc
ret_val = CAN_RX_Set_Filter(channel, filter_no,
    filter_enable, id, id_extend)
```

Parameters

channel	Number (1, 2) of CAN interface..	LONG
filter_no	Number (1…4) of filter.	LONG
filter_enable	Filter status: 0: Disable filter. 1: Enable filter.	LONG
id	Identifier (0…2^{11} or 0…2^{29}) of the CAN messages which can be received.	LONG
id_extend	Length of identifier id: 0: 11 bit. 1: 29 bit.	LONG
ret_val	0: Filter has been set. <>0:Error while configuring filter.	LONG

Notes

We recommend initialization of filters in section **LowInit:** or **Init:**.

You can set and enable up to 4 receive filters for each CAN interface. As soon as a CAN message has successfully passed one of the enabled filters, the value is stored in the receive FIFO.

See also

CAN_Msg, CAN_Init, CAN_Receive, CAN_Transmit

Valid for

X-A20+COM

Example

```
#Include ADwin-X.Inc

Init:
  PAR_1 = 0
  Rem initialize CAN Controller 1, Baud rate 50kBaud
  Par_1 = CAN_Init(1,50000)
  If (Par_1 <> 0) Then Exit
  Rem Enable filter 3 of controller 1 with identifier 40
  PAR_1 = CAN_RX_Set_Filter(1,3,1,40,0)

Event:
  REM Check for a message, only identifier 40 is valid
  PAR_9 = CAN_Receive(1)
```

CAN_Transmit

CAN_Transmit transmits the message in **CAN_Msg** to a CAN interface to be sent.

Syntax

```
#Include ADwin-X.Inc
CAN_Transmit(can_no, priority, id_extend)
```

Parameters

can_no	Number (1, 2) of CAN interface.	LONG
priority	Sending priority of the message: 0: Normal priority, message is sent via the output FIFO. 1: High priority, message is to be sent next.	LONG
id_extend	Length of identifiers: 0: 11 bit. 1: 29 bit.	LONG
ret_val	0: Message is transmitted to CAN interface. -1: Sending buffer is full, transmit message again later.	LONG

Notes

In order to send a message follow these steps:

- Enter a message into array **CAN_MSG**: Data bytes, number of data bytes, and identifier. The time stamp has no function.
- Transmit the message to the CAN interface with **CAN_Transmit**.
- Check if the message has been transmitted correctly.

The CAN interface sends the message as soon as the message object has received access rights to the CAN bus.

A high priority message is sent as very next message even when other message (of normal priority) are already waiting in the output FIFO. Messages of normal priority are sent in the order as they were transmitted to the output FIFO of the CAN interface.

See also

CAN_Msg, CAN_Init, CAN_Receive, CAN_RX_Set_Filter

Valid for

X-A20+COM

Example

```
#Include ADwin-X.Inc
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 CAN_Receive)
#Define pi 3.14159265
Dim i As Long

Init:
  Rem initialize CAN Controller 2, baud rate 50kBaud
  Par_1 = CAN_Init(2,50000)
  If (Par_1 <> 0) Then Exit

  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_Float32ToLong(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
  CAN_Msg[10] = 40        'message id

Event:
  Par_2 = CAN_Transmit(2,0,40) 'send message with 11 bit id
```

Receiving a float value see example at CAN_Receive.

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**	

16.7 RSxxx Interface

This section describes instructions to access RSxxx interfaces of *ADwin-X-A20-COM*:

– RS_Read_FIFO (page 148)
– RS_Init (page 147)
– RS_Write_FIFO (page 149)
– RS_Write_FIFO_Full (page 150)
– RS_Write_FIFO_Empty (page 151)

RS_Init

RS_Init initializes one RSxxx interface.

The following parameters are set:
- Transfer rate in Baud
- Use of test bits
- Data length
- Amount of stop bits
- Transfer protocol (handshake)

Syntax

```
#Include ADwin-X.Inc
RS_Init(channel,baud,parity,bits,stop,
        handshake)
```

Parameters

channel	Number of RSxxx interface (1).	LONG
baud	Transfer rate in Baud: 35 … 115.200.	LONG
parity	Use of test bits: 0: without parity bit. 1: even parity. 2: odd parity.	LONG
bits	Amount of data bits (6…8).	LONG
stop	Amount of stop bits. 0: 1 stop bit. 1: 1½ stop bits. 2: 2 stop bits.	LONG
handshake	Transfer protocol: 0: RS232, no handshake.	LONG

Notes

RS_Init is necessary before working first with the selected RSxxx interface, in order to set the interface parameters. They must be identical to the remote station, in order to verify a correct transfer.

See also

RS_Read_FIFO, RS_Write_FIFO, RS_Write_FIFO_Full, RS_Write_FIFO_Empty

Valid for

X-A20+COM

Example

```
#Include ADwin-X.Inc

Init:
  Rem Initialize RSxxx interface 1: 9600 Baud, without parity,
  Rem 8 data bits, 1 stop bit, hardware handshake.
  RS_Init(1,9600,0,8,0,1)
```

RS_Read_FIFO

`RS_Read_Fifo` reads a value from the input FIFO of the RSxxx interface.

Syntax

```
#Include ADwin-X.Inc
ret_val = RS_Read_Fifo(channel)
```

Parameters

`channel`	Number of RSxxx interface (1).	`LONG`
`ret_val`	Contents of the input FIFO: -1: FIFO is empty. ≥0: Transferred value (0…255).	`LONG`

Notes

The input FIFO can hold up to 64 values.

See also

RS_Init, RS_Write_FIFO, RS_Write_FIFO_Full, RS_Write_FIFO_Empty

Valid for

X-A20+COM

Example

```
#Include ADwin-X.Inc

Init:
  Rem Initialize RSxxx interface 1: 9600 Baud, without parity,
  Rem 8 data bits, 1 stop bit, hardware handshake.
  RS_Init(1,9600,0,8,0,1)

Event:
  Rem Get a value from the FIFO. If the FIFO is empty,
  Rem -1 is returned.
  PAR_1 = RS_Read_Fifo(1)
```

RS_Write_FIFO

RS_Write_FIFO writes a value into the send-FIFO of the RSxxx interface.

Syntax

```
#Include ADwin-X.Inc
ret_val = RS_Write_FIFO(channel,value)
```

Parameters

channel	Number of RSxxx interface (1).	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 free memory cell in the send-FIFO. If 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.

With **RS_Write_FIFO_Full**, you can check in advance if there is space in the send FIFO. The send FIFO can hold up to 64 values.

See also

RS_Init, RS_Read_FIFO, RS_Write_FIFO_Full, RS_Write_FIFO_Empty

Valid for

X-A20+COM

Example

```
#Include ADwin-X.Inc
Dim val As Long

INIT:
 Rem Initialize RSxxx interface 1: 9600 Baud, without parity,
 Rem 8 data bits, 1 stop bit, hardware handshake.
 RS_Init(1,9600,0,8,0,1)

EVENT:
 Rem If the FIFO is not full, [val] is written into the FIFO.
 Rem Otherwise a 1 in Par_1 indicates that writing into the
 Rem FIFO ist not possible (FIFO full).
 PAR_1 = RS_Write_FIFO(1,val)
```

RS_Write_FIFO_Full

`RS_Write_FIFO_Full` checks if the send FIFO of the RSxxx interface is already full.

Syntax

```
#Include ADwin-X.Inc
ret_val = RS_Write_FIFO_Full(channel)
```

Parameters

`channel`	Number of RSxxx interface (1).	LONG
`ret_val`	Status if the send FIFO is full: 0: False, send FIFO is not full. 1: True, send FIFO is full.	LONG

Notes

The send FIFO can hold up to 64 values.

See also

RS_Init, RS_Read_FIFO, RS_Write_FIFO, RS_Write_FIFO_Empty

Valid for

X-A20+COM

Example

```
#Include ADwin-X.Inc
Dim val As Long

Init:
  Rem Initialize RSxxx interface 1: 9600 Baud, without parity,
  Rem 8 data bits, 1 stop bit, hardware handshake.
  RS_Init(1,9600,0,8,0,1)

Event:
  Rem If the FIFO is not full, write [val] into the FIFO.
  If (RS_Write_Fifo_Full(1) <> 1) Then
    PAR_1 = RS_Write_Fifo(1,val)
  EndIf
```

RS_Write_FIFO_Empty

`RS_Write_FIFO_Empty` checks if the send FIFO of the RSxxx interface is empty.

Syntax

```
#Include ADwin-X.Inc
ret_val = RS_Write_FIFO_Empty(channel)
```

Parameters

`channel`	Number of RSxxx interface (1).	LONG
`ret_val`	Status if the send FIFO is empty: 0: False, send FIFO is not empty. 1: True, send FIFO is empty.	LONG

Notes

The send FIFO can hold up to 64 values.

See also

RS_Init, RS_Read_FIFO, RS_Write_FIFO, RS_Write_FIFO_Full

Valid for

X-A20+COM

Example

```
#Include ADwin - X.Inc
#Define val Par_2

Init:
 Rem Initialize RSxxx interface 1: 9600 Baud, without parity,
 Rem 8 data bits, 1 stop bit, hardware handshake.
 RS_Init(1,9600,0,8,0,1)

Event:
 Rem If send FIFO is not full, write [val] into the FIFO.
 If (RS_Write_Fifo_Full(1) <> 1) Then
  PAR_1 = RS_Write_Fifo(1,val)
 EndIf
 Rem Wait until send FIFO is empty i.e. value was sent.
 Do : Until (RS_Write_FIFO_Empty(1) = 1)
```

16.8 Profibus interface

This section describes instructions to access a Profibus node on *ADwin-X-A20*:

– Init_Profibus (page 153)
– Run_Profibus (page 155)

Init_Profibus

Init_Profibus initializes the Profibus slave.

Syntax

```
#Include ADwin-X.inc
ret_val = Init_Profibus(dev_adr, in_mod_cnt, out_mod_cnt,
    work_arr[])
```

Parameters

Parameter	Description	Type
dev_adr	Slave node address / station address (1…125) on the Profibus.	LONG
in_mod_cnt	Size (1, 2, 4, 8, 16, 32, 61) of the input area in the Profibus slave in double words.	LONG
out_mod_cnt	Size (1, 2, 4, 8, 16, 32, 61) of the output area in the Profibus slave in double words.	LONG
work_arr[]	Array to store data for operation of the Profibus Slave. The array must have at least AB_WORK_ARR_LEN (5000) elements.	ARRAY LONG
ret_val	Status of initialization: 0: initialization successful. -1: Error, size of data ranges is wrong.	LONG

Notes

This instruction must be processed before working with Profibus slave.

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).

Smaller data ranges accelerate the data transfer via the Profibus.

Station address and size of data ranges must equal the project settings of the Profibus.

See also

Run_Profibus

Valid for

- / -

Example

```
#Include ADwin-X.inc
#Define node 2              'slave node address
#Define out_arr Data_2
#Define in_arr Data_3

Dim out_arr[1000] As Long
Dim in_arr[1000] As Long
Dim conf_arr[AB_WORK_ARR_LEN] As Long
Dim i As Long
Dim state As Long

LowInit:
  Processdelay = 3000000    'set to 100 Hz
  Rem init Profibus interface: input data area = 8 DWords
  Rem and output data area = 16 DWords
  Par_10 = Init_Profibus(node, 8, 16, conf_arr)

Event:
  Rem set data in out_arr[] to be transferred
  For i = 1 To 16
    out_arr[i] = (out_arr[i] + i)
  Next i

  Rem send and read data
  state = Run_Profibus(out_arr,in_arr,16,conf_arr)
  Par_2 = state

  Rem now process received data stored in in_arr[1..8];
  Rem in_arr[9..16] has been filled with unusable data,
  Rem in_arr[17..1000] remains unused here.
```

Run_Profibus

Run_Profibus exchanges data with the Profibus slave.

Syntax

```
#Include ADwin-X.inc
ret_val = Run_Profibus(out_pd_arr[], in_pd_arr[],
    pd_arr_len, work_arr[])
```

Parameters

Parameter	Description	Type
out_pd_arr[]	Array, from which the Profibus slave reads data (number see pd_arr_len) and writes them to the Profibus.	ARRAY LONG
in_pd_arr[]	Array, into which the Profibus Slave writes data (number see pd_arr_len) which are read from the Profibus.	ARRAY LONG
pd_arr_len	Number of double words (1, 2, 4, 8, 16, 32, 61), which are transferred in in_pd_arr[] and out_pd_arr[].	LONG
work_arr[]	Array holding data for operation of the Profibus Slave, see **Init_Profibus**.	ARRAY LONG
ret_val	State of operation of the Profibus slave: 0: Initializing. 2: Slave waits for bus start by master.. 4: Normal operation.	LONG

Notes

The number **pd_arr_len** is used for both reading and writing data, even though the input area may be initialized with a different value than the output area. The arrays in_pd_arr[] and out_pd_arr[] must be declared with at least the size **pd_arr_len**.

Each array element in in_pd_arr[] and out_pd_arr[] contains 1 double word. A double word equals a value of data type Long.

See also

Init_Profibus

Valid for

- / -

Example

see Init_Profibus

16.9 Profinet interface

This section describes instructions to access a Profinet interface of *ADwin-X-A20*:

– Init_ProfinetIO (page 157)
– Run_ProfinetIO (page 158)

Init_ProfinetIO

`Init_ProfinetIO` initializes an array for operation with the Profinet slave.

Syntax

```
#Include ADwin-X.inc
ret_val = Init_ProfinetIO(in_size, out_size, work_arr[])
```

Parameters

in_size	Size (1, 2, 4, 8, 16, 32, 64, 128, 196, 256, 320) of the input area in the Profinet slave, in double words; 1 double word = 4 byte.	LONG
out_size	Size (1, 2, 4, 8, 16, 32, 64, 128, 196, 256, 320) of the output area in the Profinet slave, in double words; 1 double word = 4 byte.	LONG
work_arr[]	Array which is initialized for the operation with the Profinet slave. The array must have at least AB_WORK_ARR_LEN (5000) elements.	ARRAY LONG
ret_val	Status of initialization: 0: initialization was successful. -1: Error data area sizes are wrong.	LONG

Notes

This instruction must be processed before working with Profinet slave.

The size of data areas must be the same as while projecting the Profinet. Please note while projecting, that the size of data areas may be given in other units than byte (e.g. Word, QWord).

See also

Run_ProfinetIO

Valid for

- / -

Example

```
#Include ADwin-X.inc
#Define out_arr Data_2
#Define in_arr Data_3

Dim out_arr[76] As Long
Dim in_arr[76] As Long
Dim conf_arr[AB_WORK_ARR_LEN] As Long
Dim i As Long
Dim state As Long

LowInit:
  Processdelay = 3000000    'set to 100 Hz
  Rem init Profinet interface: input data area = 128 DWords
  Rem and output data area = 256 DWords
  Par_10 = Init_ProfinetIO(128, 256, conf_arr)

Event:
  Rem set data in out_arr[] to be transferred
  For i = 1 To 76
    out_arr[i] = (out_arr[i] + i)
  Next i

  Rem send and read data
  state = Run_ProfinetIO(out_arr,in_arr,256,conf_arr)
  Par_2 = state

  Rem now process received data in in_arr[] ..
```

Run_ProfinetIO

Run_ProfinetIO exchanges data with the Profinet Slave.

Syntax

```
#Include ADwin-X.inc

ret_val = Run_ProfinetIO(out_pd_arr[], in_pd_arr[],
    pd_arr_len, work_arr[])
```

Parameters

out_pd_arr[]	Array, from which the Profinet Slave reads data (number pd_arr_len) and writes them to the Profinet bus.	ARRAY LONG
in_pd_arr[]	Array, into which the Profinet Slave writes received data (number pd_arr_len).	ARRAY LONG
pd_arr_len	Number of double words (1, 2, 4, 8, 16, 32, 64, 128, 196, 256, 320), which are transferred in in_pd_arr[] and out_pd_arr[].	LONG
work_arr[]	Array holding data for operation of the Profinet Slave, see **Init_ProfinetIO**.	ARRAY LONG
ret_val	State of operation of the Profinet Slave: 0: Initialization. 2: Slave waits for bus start by master. 4: Normal operation.	LONG

Notes

The number **pd_arr_len** of values is the same for both data transfer arrays even though the input area may be initialized with a different size than the output area.

Each array element in in_pd_arr[] and out_pd_arr[] stores 4 data bytes = 1 double word. A double word corresponds to a value of data type Long.

See also

Init_ProfinetIO

Valid for

- / -

Example

see Init_ProfinetIO

16.10 EtherCAT interface

This section describes instruction to access the EtherCAT node on *ADwin-X-A20*:

– Init_EtherCAT (page 160)
– Run_EtherCAT (page 162)

Init_EtherCAT

Init_EtherCAT initializes an array for operation with the EtherCAT slave.

Syntax

```
#Include ADwin-X.inc
ret_val = Init_EtherCAT(in_size, out_size, data_type,
    work_arr[])
```

Parameters

Parameter	Description	Type
in_size	Size (0…360) of the input area in the EtherCAT slave in double words.	LONG
out_size	Size (0…360) of the output area in the EtherCAT slave in double words.	LONG
data_type	Data type of input and output area. A value in the area Wert belegt ein Doppelwort (=4 Byte). Verfügbar sind: AB_DATA_TYPE_SINT32: Signed integer, 32 Bit. AB_DATA_TYPE_FLOAT: Floating point, 32 Bit.	LONG
work_arr[]	Array which is initialized for the operation with the EtherCAT slave. The array must have at least AB_WORK_ARR_LEN (5000) elements.	ARRAY LONG
ret_val	Status of initialization: 0: initialization was successful. -1: Error data area sizes are wrong.	LONG

Notes

This instruction must be processed before working with EtherCAT Slave.

The size of data areas must be the same as while projecting the EtherCAT bus. Please note while projecting, that the size of data areas may be given in other units than byte (e.g. Byte, Word).

See also

Run_EtherCAT

Valid for

X-A20+ECAT

Example

```
ADbasic#Include ADwin-X.inc
#Define out_arr Data_2
#Define in_arr Data_3

'Data type according to Init_EtherCAT:
'  AB_DATA_TYPE_SINT32 -> Long
'  AB_DATA_TYPE_FLOAT  -> Float32
Dim out_arr[1000] As Long
Dim in_arr[1000] As Long
Dim conf_arr[AB_WORK_ARR_LEN] As Long
Dim i As Long
Dim state As Long

LowInit:
  Processdelay = 3000000   'set to 100 Hz
  Rem init EtherCAT interface: input data area = 76 values
  Rem and output data area = 92 values
  Par_10 = Init_EtherCAT(76, 92, AB_DATA_TYPE_SINT32,
    conf_arr)

Event:
  Rem set data in out_arr[] to be transferred
  For i = 1 To 92
    out_arr[i] = (out_arr[i] + i)
  Next i

  Rem send and read data
  state = Run_EtherCAT(out_arr,in_arr,92,conf_arr)
  Par_2 = state

  Rem now process received data stored in in_arr[1..76];
  Rem in_arr[77..92] has been filled with unusable data,
  Rem in_arr[93..1000] remains unused here.
```

Run_EtherCAT

Run_EtherCAT exchanges data with the EtherCAT slave.

Syntax

```
#Include ADwin-X.inc
ret_val = Run_EtherCAT(out_pd_arr[], in_pd_arr[],
    pd_arr_len, work_arr[])
```

Parameters

Parameter	Description	Type
out_pd_arr[]	Array, from which the EtherCAT Slave reads data (number pd_arr_len) and writes them to the EtherCAT bus.	ARRAY LONG | FLOAT |
in_pd_arr[]	Array, into which the EtherCAT Slave writes received data (number pd_arr_len).	ARRAY LONG | FLOAT |
pd_arr_len	Number of values (1…360), which are transferred in in_pd_arr[] and out_pd_arr[].	LONG |
work_arr[]	Array holding data for operation of the Profinet Slave, see **Init_EtherCAT**.	ARRAY LONG |
ret_val	State of operation of the Profinet Slave: 0: Initialization. 2: Slave waits for bus start by master. 4: Normal operation.	LONG |

Notes

The number **pd_arr_len** of values is the same for both data transfer arrays even though the input area may be initialized with a different size than the output area.

The arrays in_pd_arr[] and out_pd_arr[] must be declared with the size of **pd_arr_len** at least.

Declare the arrays in_pd_arr[] and out_pd_arr[] with the data type which fits to the setting of **Init_EtherCAT**, parameter data_type:
- Long for AB_DATA_TYPE_SINT32
- Float32 for AB_DATA_TYPE_FLOAT

Each array element in in_pd_arr[] and out_pd_arr[] has a size of 4 data bytes = 1 double word.

See also

Init_EtherCAT

Valid for

X-A20+ECAT

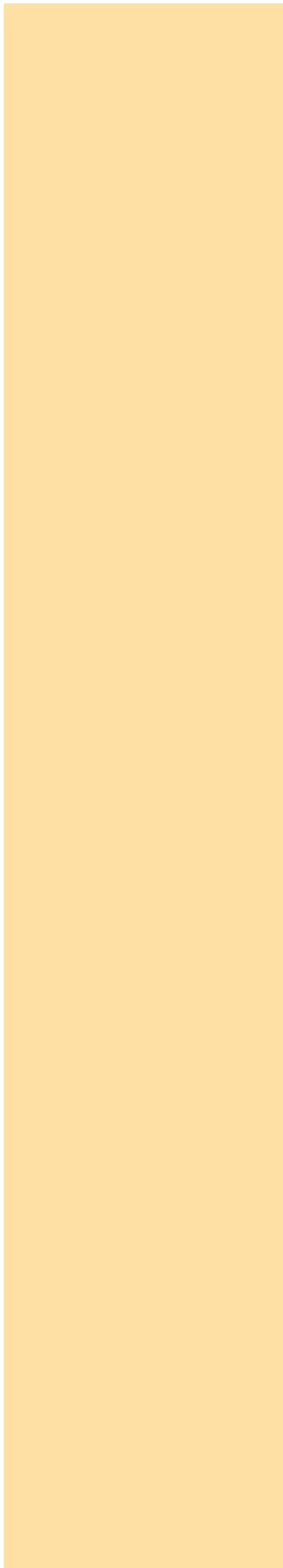
Example

see Init_EtherCAT

16.10.1 LS-Bus + *ADwin-X-A20*

The section describes instructions of the LS-bus interface of *ADwin-X-A20*:

– LS_DIO_Init (page 164)
– LS_DigProg (page 166)
– LS_Dig_IO (page 168)
– LS_Digout_Long (page 170)
– LS_Digout_Long_BS (page 171)
– LS_Digin_Long (page 173)
– LS_Digin_Long_BS (page 174)
– LS_Get_Output_Status (page 176)
– LS_Reset (page 178)
– LS_Watchdog_Init (page 179)
– LS_Watchdog_Reset (page 181)



LS_DIO_Init

`LS_DIO_Init` initializes the specified module of type HSM-24V on the LS bus and returns the error status.

Syntax

```
#Include ADwin-X.Inc
ret_val = LS_DIO_Init(ls-module)
```

Parameters

`ls-module`	Specified module address on the LS bus (1…15).	`LONG`
`ret_val`	Return value representing the error status: -1: Communication with module is impossible. >0: Bit pattern with several error bits. 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 `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. Therefore, error bits in the return value can refer to an error which has occurred some time earlier.

Please note: At program start, ignore the error bit WD until the watchdog counter is reset with `LS_Watchdog_Init`. The watchdog counter starts with power-up of the module and normally has launched an error until program start.

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 the module HSM-24V is working permanently without interruption even if all channels are used in parallel.

See also

LS_DigProg, LS_Dig_IO, LS_Digout_Long, LS_Digin_Long, LS_Get_Output_Status, LS_Reset, LS_Watchdog_Init, LS_Watchdog_Reset

Valid for

HSM-24V + X-A20

Example

```
REM Example process for one module HSM-24V and ADwin-X
#Include ADwin-X.inc

Init:
  Processdelay = 4000000   '10Hz HP
  Par_1 = LS_DIO_Init(1)
  REM enable watchdog time with 1.1 sec
  Par_2 = LS_Watchdog_Init(1, 1, 1100)
  REM LS channels 1…32 as outputs
  Par_3 = LS_Digprog(1, 0Fh)

Event:
  REM set one LS 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 reset watchdog, set LS channels, and read back real state
  REM (note: LS_Dig_IO uses LS address 1)
  Par_12 = LS_Dig_IO(Par_11)
```

LS_DigProg

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.

Syntax

```
#Include ADwin-X.Inc
ret_val = LS_Digprog(ls-module, pattern)
```

Parameters

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	Return value representing the error status: -1: Communication with module is impossible. >0: Bit pattern with several error bits. 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 **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).

See also

LS_DIO_Init, LS_Dig_IO, LS_Digout_Long, LS_Digin_Long, LS_Get_Output_Status, LS_Reset, LS_Watchdog_Init, LS_Watchdog_Reset

Valid for

HSM-24V + X-A20

Example

```
REM Example process for one module HSM-24V and ADwin-X
#Include ADwin-X.inc

Init:
 Processdelay = 4000000   '10Hz HP
 Par_1 = LS_DIO_Init(1)
 REM enable watchdog time with 1.1 sec
 Par_2 = LS_Watchdog_Init(1, 1, 1100)
 REM LS channels 1…32 as outputs
 Par_3 = LS_Digprog(1, 0Fh)

Event:
 REM set one LS 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 reset watchdog, set LS channels, and read back real state
 REM (note: LS_Dig_IO uses LS address 1)
 Par_12 = LS_Dig_IO(Par_11)
```

LS_Dig_IO

LS_Dig_IO sets all digital outputs of the specified module HSM-24V on the LS bus to the level High oder Low and returns the status of all channels as bit pattern.

Syntax

```
#Include ADwin-X.Inc
ret_val = LS_DIG_IO(pattern)
```

Parameters

pattern	Bit pattern, setting the digital outputs (see table). Bit = 0: Set outputs to level Low. Bit = 1: Set outputs 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

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 LS_Dig_IO, no other LS bus instruction will be used.

The channels are set as inputs or outputs using 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.

LS_Dig_IO resets the watchdog counter of the module to the start value. The counter remains enabled. The start value is set using 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

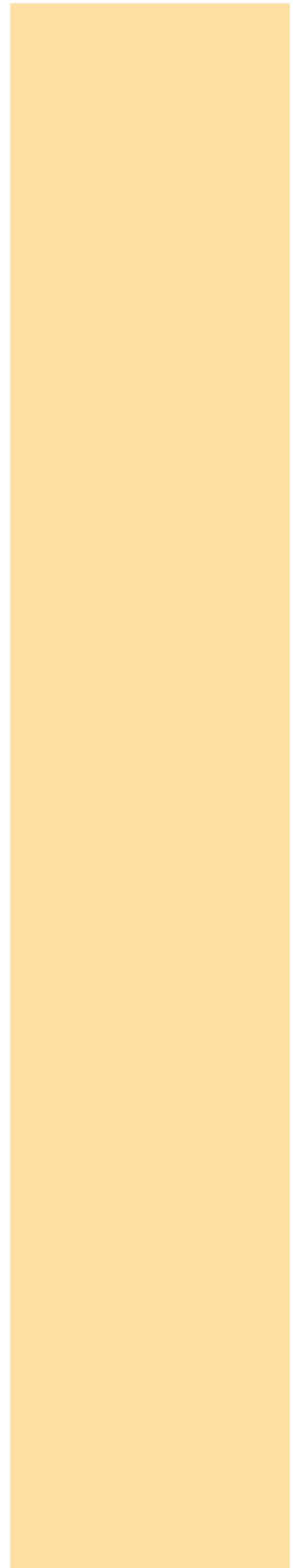
HSM-24V + X-A20

See also

LS_DIO_Init, LS_DigProg, LS_Digout_Long, LS_Digin_Long, LS_Get_Output_Status, LS_Reset, LS_Watchdog_Init, LS_Watchdog_Reset

Example

```
REM Example process for one module HSM-24V and ADwin-X
#Include ADwin-X.inc

Init:
 Processdelay = 4000000   '10Hz HP
 Par_1 = LS_DIO_Init(1)
 REM enable watchdog time with 1.1 sec
 Par_2 = LS_Watchdog_Init(1, 1, 1100)
 REM LS channels 1…32 as outputs
 Par_3 = LS_Digprog(1, 0Fh)

Event:
 REM set one LS 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 reset watchdog, set LS channels, and read back real state
 REM (note: LS_Dig_IO uses LS address 1)
 Par_12 = LS_Dig_IO(Par_11)
```



LS_Digout_Long

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 ADwin-X.Inc
LS_Digout_Long(ls-module,pattern)
```

Parameters

`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 **LS_Digprog**.

The `pattern` is applied to those channels only, which are set as outputs. Bits for input channels are ignored.

See also

LS_DIO_Init, LS_DigProg, LS_Dig_IO, LS_Digin_Long, LS_Digout_Long_BS, LS_Get_Output_Status, LS_Reset, LS_Watchdog_Init, LS_Watchdog_Reset

Valid for

HSM-24V + X-A20

Example

```
REM Example process for ADwin-X-A20 and 2 modules HSM-24V
REM Set process to low priority!
#Include ADwin-X.inc

Init:
  Processdelay = 4000000   '10Hz HP
  REM settings for LS module 1
  Par_1 = LS_DIO_Init(1)
  REM enable watchdog with 1.1 s
  Par_2 = LS_Watchdog_Init(1, 1, 1100)
  REM set LS channels 1…32 as outputs
  Par_3 = LS_Digprog(1, 01111b)

  REM settings for LS module 3
  Par_11 = LS_DIO_Init(3)
  REM enable watchdog with 1.1 s
  Par_12 = LS_Watchdog_Init(3, 1, 1100)
  REM set LS channels 1…32 as inputs
  Par_13 = LS_Digprog(3, 0h)

Event:
  REM set one LS channel to high, rotating from 1 to 32
  Inc Par_15
  If (Par_15 >= 32) Then Par_15 = 0
  Par_16 = Shift_Left(1, PAR_15)
  REM set LS channels of module 1
  LS_Digout_Long(1, PAR_16)
  REM read LS channels of module 3
  Par_17 = LS_Digin_Long(3)
  REM reset watchdog
  LS_Watchdog_Reset()
```

LS_Digout_Long_BS

LS_Digout_Long_BS sets or clears all digital outputs of the specified module HSM-24V on the LS bus according to the transferred 32-bit value and returns the error status.

Syntax

```
#Include ADwin-X.Inc
LS_Digout_Long_BS(ls_module,pattern,status)
```

Parameters

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

status	Bit pattern representing the error status: 0: O.k. -1: No communication possible. >0: Bit pattern with error bits. 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 channels are set as inputs or outputs using **LS_Digprog**.

The pattern is applied to those channels only, which are set as outputs. Bits for input channels are ignored.

See also

LS_DIO_Init, LS_DigProg, LS_Dig_IO, LS_Digin_Long_BS, LS_Digout_Long, LS_Get_Output_Status, LS_Reset, LS_Watchdog_Init, LS_Watchdog_Reset

Valid for

HSM-24V + X-A20

Example

```
REM Example process for ADwin-X-A20 and 2 modules HSM-24V
REM Set process to low priority!
#Include ADwin-X.inc

Init:
  Processdelay = 4000000   '10Hz HP
  REM settings for LS module 1
  Par_1 = LS_DIO_Init(1)
  REM enable watchdog with 1.1 s
  Par_2 = LS_Watchdog_Init(1, 1, 1100)
  REM set LS channels 1…32 as outputs
  Par_3 = LS_Digprog(1, 01111b)

  REM settings for LS module 3
  Par_11 = LS_DIO_Init(3)
  REM enable watchdog with 1.1 s
  Par_12 = LS_Watchdog_Init(3, 1, 1100)
  REM set LS channels 1…32 as inputs
  Par_13 = LS_Digprog(3, 0h)

Event:
  REM set one LS channel to high, rotating from 1 to 32
  Inc Par_15
  If (Par_15 >= 32) Then Par_15 = 0
  Par_16 = Shift_Left(1, PAR_15)
  REM set LS channels of module 1
  LS_Digout_Long_BS(1, PAR_16, Par_5)
  If (Par_5 <> 0) Then End 'exit on error)
  REM read LS channels of module 3
  Par_17 = LS_Digin_Long_BS(3, Par_6)
  If (Par_6 <> 0) Then End 'exit on error)
  REM reset watchdog
  LS_Watchdog_Reset()
```

LS_Digin_Long

LS_Digin_Long returns the status of all channels of the specified module HSM-24V on the LS bus as bit pattern.

Syntax

```
#Include ADwin-X.Inc
ret_val = LS_DIGIN_LONG(ls-module)
```

Parameters

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 **LS_Digprog** before use.

The return value contains the real state of both inputs and outputs. The inputs have a filter causing about 12 µs signal delay.

See also

LS_DIO_Init, LS_DigProg, LS_Dig_IO, LS_Digin_Long_BS, LS_Digout_Long, LS_Get_Output_Status, LS_Reset, LS_Watchdog_Init, LS_Watchdog_Reset

Valid for

HSM-24V + X-A20

Example

```
REM Example process for ADwin-X-A20 and 2 modules HSM-24V
REM Set process to low priority!
#Include ADwin-X.inc

Init:
  Processdelay = 4000000    '10Hz HP
  REM settings for LS module 1
  Par_1 = LS_DIO_Init(1)
  REM enable watchdog with 1.1 s
  Par_2 = LS_Watchdog_Init(1, 1, 1100)
  REM set LS channels 1…32 as outputs
  Par_3 = LS_Digprog(1, 01111b)

  REM settings for LS module 3
  Par_11 = LS_DIO_Init(3)
  REM enable watchdog with 1.1 s
  Par_12 = LS_Watchdog_Init(3, 1, 1100)
  REM set LS channels 1…32 as inputs
  Par_13 = LS_Digprog(3, 0h)

Event:
  REM set one LS channel to high, rotating from 1 to 32
  Inc Par_15
  If (Par_15 >= 32) Then Par_15 = 0
  Par_16 = Shift_Left(1, PAR_15)
  REM set LS channels of module 1
  LS_Digout_Long(1, PAR_16)
  REM read LS channels of module 3
  Par_17 = LS_Digin_Long(3)
  REM reset watchdog
  LS_Watchdog_Reset()
```

LS_Digin_Long_BS

LS_Digin_Long_BS returns the status of all channels of the specified module HSM-24V on the LS bus as bit pattern as well as the error status.

Syntax

```
#Include ADwin-X.Inc
ret_val = LS_Digin_Long_BS(ls_module, status)
```

Parameters

ls_module	Specified module address (1…15) on the LS bus.	LONG
status	Bit pattern representing the error status: 0: O.k. -1: No communication possible. >0: Bit pattern with error bits. 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.

ret_val	Bit pattern. Each bit (31:0) represents the state of a digital channel (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 **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.

See also

LS_DIO_Init, LS_DigProg, LS_Dig_IO, LS_Digin_Long, LS_Digout_Long_BS, LS_Get_Output_Status, LS_Reset, LS_Watchdog_Init, LS_Watchdog_Reset

Valid for

HSM-24V + X-A20

Example

```
REM Example process for ADwin-X-A20 and 2 modules HSM-24V
REM Set process to low priority!
#Include ADwin-X.inc

Init:
 Processdelay = 4000000   '10Hz HP
 REM settings for LS module 1
 Par_1 = LS_DIO_Init(1)
 REM enable watchdog with 1.1 s
 Par_2 = LS_Watchdog_Init(1, 1, 1100)
 REM set LS channels 1…32 as outputs
 Par_3 = LS_Digprog(1, 01111b)

 REM settings for LS module 3
 Par_11 = LS_DIO_Init(3)
 REM enable watchdog with 1.1 s
 Par_12 = LS_Watchdog_Init(3, 1, 1100)
 REM set LS channels 1…32 as inputs
 Par_13 = LS_Digprog(3, 0h)

Event:
 REM set one LS channel to high, rotating from 1 to 32
 Inc Par_15
 If (Par_15 >= 32) Then Par_15 = 0
 Par_16 = Shift_Left(1, PAR_15)
 REM set LS channels of module 1
 LS_Digout_Long_BS(1, PAR_16, Par_5)
 If (Par_5 <> 0) Then End 'exit on error)
 REM read LS channels of module 3
 Par_17 = LS_Digin_Long_BS(3, Par_6)
 If (Par_6 <> 0) Then End 'exit on error)
 REM reset watchdog
 LS_Watchdog_Reset()
```

LS_Get_Output_Status

`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 ADwin-X.Inc
ret_val = LS_GET_OUTPUT_STATUS(ls-module)
```

Parameters

`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…500 mA is present on several channels at the same time. Irrespective of this, an over-current of more than 500 mA automatically switches off the concerned channel.

After a "superheating" errorthe module is reset with `LS_DIO_Init`.

The channels of the module HSM-24V may only be operated in the range of 0…150 mA. This ensures the module HSM-24V is working permanently without interruption even if all channels are used in parallel.

See also

LS_DIO_Init, LS_DigProg, LS_Dig_IO, LS_Digout_Long, LS_Digin_Long, LS_Reset, LS_Watchdog_Init, LS_Watchdog_Reset

Valid for

HSM-24V + X-A20

Example

```
REM Example process for ADwin-X-A20 and 2 modules HSM-24V
REM Set process to low priority!
#Include ADwin-X.inc

Init:
 Processdelay = 4000000   '10Hz HP
 REM settings for LS module 1
 Par_1 = LS_DIO_Init(1)
 REM enable watchdog with 1.1 s
 Par_2 = LS_Watchdog_Init(1, 1, 1100)
 REM set LS channels 1…32 as outputs
 Par_3 = LS_Digprog(1, 01111b)

 REM settings for LS module 3
 Par_11 = LS_DIO_Init(3)
 REM enable watchdog with 1.1 s
 Par_12 = LS_Watchdog_Init(3, 1, 1100)
 REM set LS channels 1…32 as inputs
 Par_13 = LS_Digprog(3, 0h)

Event:
 REM check for over-current
 Par_5 = LS_Get_Output_Status(1) + LS_Get_Output_Status(3)
 If (Par_5 > 0) Then End  'over-current: Exit program

 REM set one channel to high, rotating from 1 To 32
 Inc Par_15
 If (Par_15 >= 32) Then Par_15 = 0
 Par_16 = Shift_Left(1, PAR_15)
 REM set LS channels of module 1
 LS_Digout_Long(1, Par_16)
 REM read LS channels of module 3
 Par_17 = LS_Digin_Long(3)
 REM reset watchdog
 LS_Watchdog_Reset()
```

LS_Reset

`LS_Reset` setzt die Schnittstelle zum LS-Bus zurück.

Syntax

```
#Include ADwin-X.Inc
LS_Reset()
```

Parameters

-/-

Notes

Die Anweisung soll nur im Abschnitt `INIT:` verwendet werden, weil sie eine lange Ausführungszeit hat.

See also

LS_DIO_Init, LS_DigProg, LS_Dig_IO, LS_Digout_Long, LS_Digin_Long, LS_Get_Output_Status, LS_Watchdog_Init, LS_Watchdog_Reset

Valid for

HSM-24V + X-A20

Example

```
REM Example process for one module HSM-24V and ADwin-X
#Include ADwin-X.inc

Init:
  Processdelay = 4000000   '10Hz HP
  LS_Reset()
  LS_Watchdog_Reset()
  Par_1 = LS_DIO_Init(1)
  REM enable watchdog time with 1.1 sec
  Par_2 = LS_Watchdog_Init(1, 1, 1100)
  REM LS channels 1…32 as outputs
  Par_3 = LS_Digprog(1, 0Fh)

Event:
  REM set one LS 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 reset watchdog, set LS channels, and read back real state
  REM (note: LS_Dig_IO uses LS address 1)
  Par_12 = LS_Dig_IO(Par_11)
```

LS_Watchdog_Init

LS_Watchdog_Init enables or disables the watchdog counter of a specified module on the LS bus. If enabled, the counter is set to the start value and is started.

Syntax

```
#Include ADwin-X.Inc
ret_val = LS_WATCHDOG_INIT(ls-module,enable,time)
```

Parameters

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	Return value representing the error status: -1: Communication with module is impossible. >0: Bit pattern with several error bits. 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 10ms 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 **LS_Watchdog_Reset**.

The watchdog function is used as to monitor the connection between *ADwin* system and LS bus module.

See also

LS_DIO_Init, LS_DigProg, LS_Dig_IO, LS_Digout_Long, LS_Digin_Long, LS_Get_Output_Status, LS_Reset, LS_Watchdog_Reset

Valid for

HSM-24V + X-A20

Example

```
REM Example process for one module HSM-24V and ADwin-X
#Include ADwin-X.inc

Init:
 Processdelay = 4000000   '10Hz HP
 Par_1 = LS_DIO_Init(1)
 REM enable watchdog time with 1.1 sec
 Par_2 = LS_Watchdog_Init(1, 1, 1100)
 REM LS channels 1…32 as outputs
 Par_3 = LS_Digprog(1, 0Fh)

Event:
 REM set one LS 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 reset watchdog, set LS channels, and read back real state
 REM (note: LS_Dig_IO uses LS address 1)
 Par_12 = LS_Dig_IO(Par_11)
```

LS_Watchdog_Reset

`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 ADwin-X.Inc
LS_WATCHDOG_RESET()
```

Parameters

-/-

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.

See also

LS_DIO_Init, LS_DigProg, LS_Dig_IO, LS_Digout_Long, LS_Digin_Long, LS_Get_Output_Status, LS_Reset, LS_Watchdog_Init

Valid for

HSM-24V + X-A20

Example

```
REM Example process for ADwin-X-A20 and 2 modules HSM-24V
REM Set process to low priority!
#Include ADwin-X.inc

Init:
 Processdelay = 4000000   '10Hz HP
 REM settings for LS module 1
 Par_1 = LS_DIO_Init(1)
 REM enable watchdog with 1.1 s
 Par_2 = LS_Watchdog_Init(1, 1, 1100)
 REM set LS channels 1…32 as outputs
 Par_3 = LS_Digprog(1, 01111b)

 REM settings for LS module 3
 Par_11 = LS_DIO_Init(3)
 REM enable watchdog with 1.1 s
 Par_12 = LS_Watchdog_Init(3, 1, 1100)
 REM set LS channels 1…32 as inputs
 Par_13 = LS_Digprog(3, 0h)

Event:
 REM set one LS channel to high, rotating from 1 to 32
 Inc Par_15
 If (Par_15 >= 32) Then Par_15 = 0
 Par_16 = Shift_Left(1, PAR_15)
 REM set LS channels of module 1
 LS_Digout_Long(1, PAR_16)
 REM read LS channels of module 3
 Par_17 = LS_Digin_Long(3)
 REM reset watchdog
 LS_Watchdog_Reset()
```

Annex

A.1 Technical Data

General Data/Limit Values						
	Symbol	Conditions	min.	typ.	max.	Unit
Supply Voltage/Supply Current						
Voltage	U_b		10	12	28	V
Idle current	I_{idle}		0.6	0.8	1.5	A
Power-up current	$I_{power-on}$			3.0		
Valid operation ranges						
Temperature	$T_{environment}$		+5		+50	°C
	$T_{chassis}$		+5		+55	
Relative humidity	F_{rel}	no condensation	0		80	%
Storage						
Temperature	T		-20		+70	°C
Humidity	R_H	no condensation or aggressive atmosphere				
Dimensions						
Width x height x depth	W x H x D		215 x 125 x 47			mm
		Variant -R	42 HP x 3 U 213,5 x 129 x 29,5			mm
Net weight						
Weight	m_{Net}		760			g
		Variant -R	580			g
Connectors						
DSUB connectors	Metric ISO threads; UNC threads available as ordering option					
Installation						
optional	DNI rail mounting and wall mounting					

Processor T12.1						
Parameter	Symbol	Conditions	min.	typ.	max.	Unit
Type	ZYNQ™ with Dual-Core ARM Cortex-A9					
Manufacturer	XILINX					
Clock frequency	f_{CLK}			666		MHz
Register width		Floating point Integer		64 32		Bit
Memory	DRAM			1		GByte

Basic version						
Parameter	Symbol	Conditions	min.	typ.	max.	Unit
Analog Inputs (ADC 18-bit)						
Number	8, differential. Version M1: Conversion via multiplexer, Version F: Conversion synchronously.					
Input voltage	U_{out}	Version M1/F, k_V=1	-10		+9.999695	V
		Version F, k_V=2	-5		+4.999847	V
Input resistance	R_i		323.4	330	336.6	kΩ
Overvoltage	$U_{in\ max.}$	ON & OFF			±35	V
Conversion time	t_{Conv}	Version M1		5		µs
		Version F, refers to number of channels	1.25		5	µs
Integral non-linearity	INL	k_V=1		±1	±3.5	LSB
Differential non-linearity	DNL			±0.2	±0.9	
Outputs: DAC 16-bit						
Number	2					
Output voltage	U_{out}		-10		+9.999695	V
Update time	t_{update}			1		µs
Settling time	t_{settle}	2V jump		2		µs
		FSR[a] (20V)		4		
Permissible current					±5	mA
Integral non-linearity	INL				±2	LSB
Differential non-linearity	DNL				±1	
Offset	Error	Adjustable with ADtest.exe				
Gain	Error	Adjustable with ADtest.exe				
Outputs: DAC 12-bit						
Number	2					
Output voltage	U_{out}		-10		+9.995117	V
Update time	t_{update}		500		1000	µs
Permissible current					±5	mA
Integral non-linearity	INL				±2	LSB
Differential non-linearity	DNL				±1	
Offset	Error	Adjustable with ADtest.exe				
Gain	Error	Adjustable with ADtest.exe				
Digital Inputs/Outputs						
Number	DIO39:32	8 (TTL level), programmable as inputs / outputs in groups of 4				
	EVENT (Digin 62)	1 ext. trigger input				
Circuitry see "Circuitry of digital inputs / outputs", TTL inputs/outputs, Event input, page A-4						
LS bus						
	1 serial interface for up to 15 LS bus modules.					

[a] Full Scale Range

Option CO1						
Parameter	Symbol	Conditions	min.	typ.	max.	Unit
Counter						
Number and function	1 counter block in-/decremental counter (event counter with clock/direction or four edge evaluation) and PWM counter (pulse width measurement). Pins with TTL level, double assignment. Properties see "Counter", page A-5					

Option D						
Parameter	Symbol	Conditions	min.	typ.	max.	Unit
Digital Inputs/Outputs						
Number	DIO41:DIO40	2 programmable inputs or outputs, differential				
	DIGIN47: DIGIN42	6 inputs, differential, double assignment.				
	DIGIN60	1 input, differential, double assignment.				
Circuitry see "Circuitry of digital inputs / outputs", differential, page A-4						
Counter						
Number and function	2 counter blocks in-/decremental counter (event counter with clock/direction or four edge evaluation) and PWM counter (pulse width measurement). Pins differential, double assignment. Properties see "Counter", page A-5					
Interfaces						
SSI	1 SSI decoder, pins differential, double assignment.					

Option DCT						
Parameter	Symbol	Conditions	min.	typ.	max.	Unit
Digital Inputs/Outputs						
Number	DIO41:DIO40	2 programmable inputs or outputs, differential				
	DIGIN47: DIGIN42	6 inputs, differential, double assignment.				
	DIGIN60	1 input, differential, double assignment.				
	DIO31:DIO00	32 inputs or outputs, TTL level, programmable as inputs / outputs in groups of 8, DIO31:DIO26 have double assignment.				
	DIGIN55: DIGIN48	8 inputs, comparator levels set with DAC12-1, double assignment.				
	DIGIN59: DIGIN55	4 inputs, comparator levels set with DAC12-2.				
Circuitry see "Circuitry of digital inputs / outputs", TTL inputs/outputs, differential, page A-4						
Counter						
Number and function	2 counter blocks, inputs differential, double assignment. 2 counter blocks, inputs with TTL levels, double assignment. 2 comparator counter blocks, inputs with TTL level, double assignment. Each counter block with in-/decremental counter (event counter with clock/direction or four edge evaluation) and PWM counter (pulse width measurement). Properties see "Counter", page A-5					
Interfaces						
SSI	SSI decoder, 1 interface. Pins with TTL level, double assignment.					

Option COM	
Interfaces	
CAN	CAN High speed, 2 interfaces
RS232	RS232, 1 interface

Option Profinet	
Interfaces	
Profinet	Profinet, 1 interface

Option Profibus-IRT	
Interfaces	
Profibus-IRT	Profibus-IRT, 1 interface 2 plugs RJ-45 (copper) or 2 duplex plugs SC-RJ (fiber optics)

Option EtherCat	
Interfaces	
EtherCat	EtherCat, 1 interface

Circuitry of digital inputs / outputs						
Parameter	Symbol	Conditions	min.	typ.	max.	Unit
EVENT input						
Edge detection, pos.	V_{T+} (Low)	V_{CC} = 5V	1.65	1.9	2.15	V
Edge detection, neg.	V_{T-} (High)	V_{CC} = 5V	0.75	1.0	1.25	
Switching hysteresis	V_{T+} - V_{T-}		0.4	0.9		
Input current	I_{IH}	V_I = 2.7V			20	µA
	I_{IL}	V_I = 0.4V			-50	
TTL inputs						
Input voltage			-0.5		+6.0	V
Logic input voltage	V_{IH} (High)	V_{CC} = 5V	3.5			
	V_{IL} (Low)	V_{CC} = 5V			1.5	
Logic input current	I_I	10kΩ Pull-down				
Comparator inputs						
Input voltage			-0.1		+30	V
Logic switching treshold	V_{IH} (High)	$U_{out\ DAC12}$ + 0,4V	0.4…5			
	V_{IL} (Low)	$U_{out\ DAC12}$ - 0,4V			0…4.6	
Switching hysteresis	V_{IH} - V_{IL}	depends on $U_{out\ DAC12}$	±30		±70	mV
max. measurement frequency	f	depends on $U_{out\ DAC12}$ and V_I	10	30	200	kHz
Logic input current	I_I	14,7kΩ Pull-up (to +5V)				
Differential channels DIO41:DIO40						
Differential input threshold voltage	V_{TH}	-7V ≤ V_{CM} ≤ 12V	-300		+300	mV
Input hysteresis	ΔV_{TH}	-7V ≤ V_{CM} ≤ 12V		25		mV
Range of common mode voltage	V_{CM}		-7		+12	V
Differential output voltage	V_{OD1}		1.5		5	V

Circuitry of digital inputs / outputs						
Differential inputs DIGIN47:DIGIN42						
Differential input thresh-old voltage	V_{TH}	$-10V \leq V_{CM} \leq 13.2V$	-200		+200	mV
Input hysteresis	ΔV_{TH}	$-10V \leq V_{CM} \leq 13.2V$		40		mV
Range of common mode voltage	V_{CM}		-10		+13.2	V
Differential slew rate			0.33			V/µs
Permissible differential input voltage		for each input			±3.9	V
Bus termination				120		Ω
TTL outputs						
Logic output voltage	V_{OH} (High)	I_{OH}= -32mA	3.8			V
	V_{OL} (Low)	I_{OL}= +32mA, V_{CC} = 4.5V			0.55	
Logic output current	I_O	je DIO-Leitung			±32	mA
	I_{TOTAL}	je DIO-Gruppe (8) über V_{CC} / GND			±100	

Counter						
Parameter	Symbol	Conditions	min.	typ.	max.	Unit
Reference oscillator						
Reference frequency	f_{ref}			33,3		MHz
Accuracy and Drift					±50	ppm
Function	Counter block with – up/down counter for measurement of duty cycle, pulkse and pause width and event counting with clock / direction or four-edge-evaluation. – PWM counter with internal clock for pulse width evaluation.					
Counter inputs	Each counter has 3 inputs (A/CLK, B/DIR, CLR/LATCH). Input circuitry see "Circuitry of digital inputs / outputs"					
Counter resolution				32		Bit
Count frequency counter 1…5	f_{CLK}	Input CLK		20		MHz
		Input A/B		5		
Count frequency comparator counter 6, 7	f_{CLK}	Input CLK		200		kHz
		Input A/B		50		
PWM count frequency	f_{CLK}	internal		100		MHz

A.2 Hardware revisions

The revision is marked on the bottom of the casing. The differences of the revision status' are shown below.

Revision	First release	Changes to previous revision status
A	1 / 2019	First release.

A.3 RoHS Declaration of Conformity

The RoHS directive 2011/65/EU of the European Union on the restriction of the use of certain hazardous substances in electrical und electronic equipment (RoHS directive) has become operative as from 3rd January, 2013.

The following substances are involved:

– Lead (Pb)

– Cadmium (Cd)

– Hexavalent chromium (Cr VI)

– Polybrominated biphenyls (PBB)

– Polybrominated diphenyl ethers (PBDE)

– Mercury (Hg)

– Bis(2-ethylhexyl) phthalate (DEHP)

– Benzyl butyl phthalate (BBP)

– Dibutyl phthalate (DBP)

– Diisobutyl phthalate (DIBP)

The product line *ADwin-X-A20* complies with the requirements of the RoHS directive in all delivered variants.