JÄGER
Computergesteuerte
Messtechnik GmbH

ADwin-Gold-
USB / -ENET

Manual

ADwin-Gold USB / ENET, manual version 4.2, November 2008

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

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

JÄGER
Computergesteuerte
Messtechnik GmbH

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

Table of contents

Typographical Conventions

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

You find a "note" next to

– information, which have absolutely to be considered in order to guarantee an operation without any errors
– advice for efficient operation

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

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

<C:\ADwin\ …>

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

Program text

ADbasic source code elements such as **INSTRUCTIONS**, *variables*, `comments` and `other text` are characterized by the font `Courier New` and are printed in color (see also the editor of the ***ADbasic*** development environment).

Var_1

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

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

1 Information about this Manual

This manual contains complex information about the operation of the *ADwin-Gold* 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-Gold* system.
- the manual *ADbasic*, which explains basic instructions for the compiler *ADbasic* and the functional layout of the *ADwin* system.
- the manuals for all current development environments containing the description of installation and instructions.

Please note:

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

Qualified personnel

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

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

Availability of the documents

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

Legal information

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 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 continously running communication process enables a continous 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.

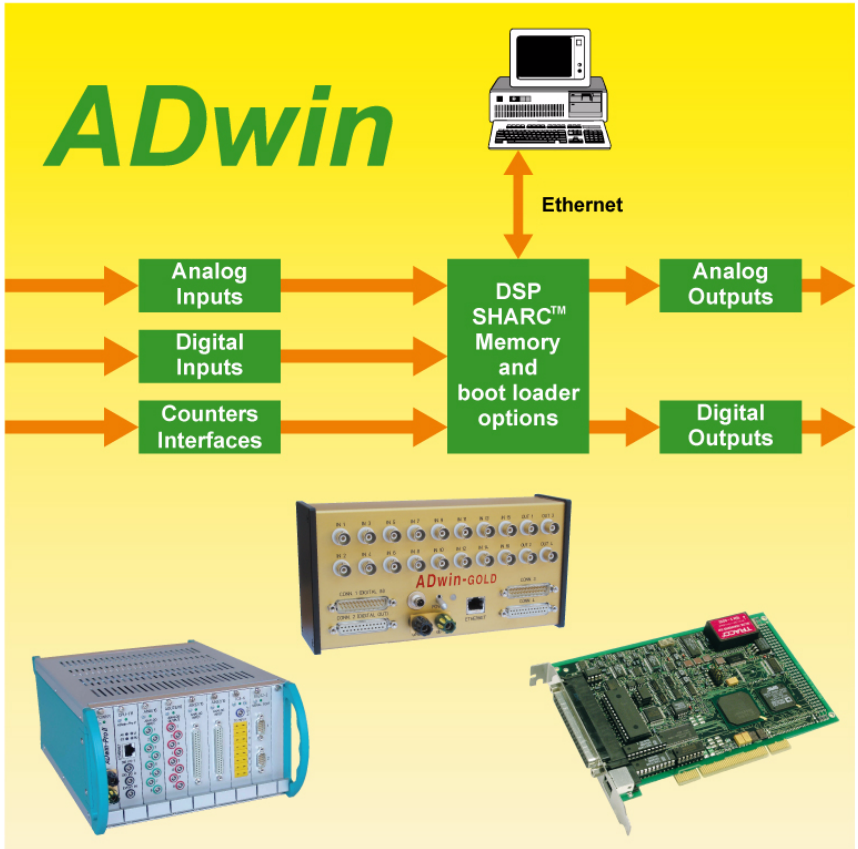


Fig. 1 – Concept of the ***ADwin*** systems

2.2 The *ADwin-Gold* System

Processor and memory

The *ADwin-Gold* system is equipped with the digital 32 bit signal processor T9 (SHARC ADSP 21062) from Analog Devices with floating point and integer processing. It is responsible for the complete measurement data acquisition, online processing, and signal output, and makes it possible to process instantaneously sample rates of up to several 100 kHz.

The on-chip memory with 256 KiB has a very short access time of 25 ns and is large enough to hold the complete *ADwin* operating system, the *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. For buffering larger quantities of data the DSP uses an external memory of 16 MiB (DRAM; optional 64 MiB).

Analog inputs

The system has 16 analog inputs with BNC plugs (alternatively: DSub connectors), which are divided into two groups each being connected to one multiplexer. These two outputs are optionally converted by a 14-bit or 16-bit analog-to-digital converter (ADC), (see Fig. 2 "Block diagram of the ADwin-Gold). With the 14-bit ADCs it is possible to sample very fast, with the 16-bit ADCs highly accurately.

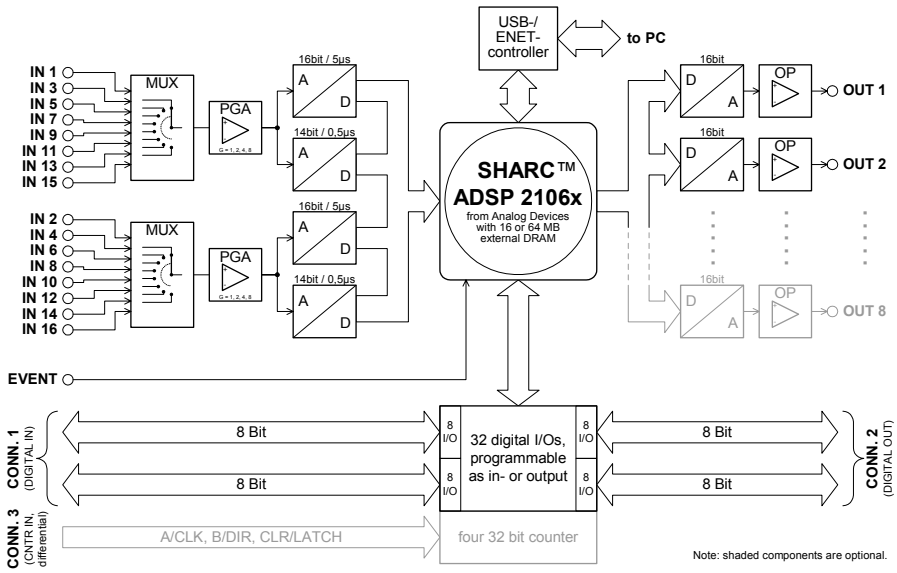


Fig. 2 – Block diagram of the *ADwin-Gold*

Analog outputs

The standard version of the *ADwin-Gold* system is equipped with 2 analog outputs (optional 8) with an output voltage range of -10 V … +10 V and a 16-bit resolution. You can synchronize the output of the voltage of all DACs per software.

Digital inputs and outputs

32 digital inputs or outputs are available on two 25-pin D-Sub connectors. They can be programmed in groups of 8 as inputs or outputs. The inputs or outputs are TTL-compatible.

Trigger input (EVENT)

The *ADwin-Gold* has a trigger input (EVENT, see also chapter 5.2 "Digital Inputs and Outputs"). Processes can be triggered by a signal and are completely processed afterwards. (see *ADbasic* manual, chapter "Structure of the ADbasic Program").

All analog data inputs and outputs of the system are differential.

The connection between *ADwin-Gold* system and computer is made via the USB or Ethernet interface (depending on the version you have purchased).

Standard delivery

The standard delivery items for the *ADwin-Gold* system:

- the *ADwin-Gold* system with USB or Ethernet interface,
- a USB cable or a cross-over Ethernet cable from the PC to the Gold device (length about 1.8 m).
- the power adapter: a three-pin power supply cable, which prevents the possibility of mismatch, at a slot metal sheet with socket connector,
- the power supply cable from the power adapter to the system,
- the ADwin CDROM,
- the manual "Driver Installation",
- this hardware manual.

Options

2.2.1 Options (no upgrades possible)

The following options are available:

- *Gold-D*: All inputs and outputs have DSub-connectors, including the analoge inputs (instead of BNC plugs).
- *Gold-DA*: 6 additional analog outputs (differential) with a 16-bit DAC each.
- *Gold-CO1*: counter option with four 32 bit counters, which can optionally be used for period width measurement, as impulse counters or as up/down counters with clock/direction or four edge evaluation for quadrature encoders.
- *Gold-CAN*: 4 decoders for use with incremental encoders with SSI interface, 2 CAN interfaces (both either high speed *or* low speed) and 2 RSxxx interfaces (RS232, RS485). This option is available in combination with the option Gold-D only.
- *GOLD-MEM-64*: external memory with 64 MiB instead of 16 MiB and 512 KiB internal CPU memory instead of 256 KiB.
- *Gold-Boot*: Flash-EPROM boot loader for stand-alone operation without PC (only in combination with the *Gold-ENET*).

If not excluded above, all additional options can be combined with each other.

Accessories

2.2.2 Accessories

- *ADbasic*, real-time development tool for all *ADwin* systems
- *ADwin-Gold*-pow: external power supply (necessary for notebook operation)
- *Gold-Mount*: kit for installation of the *ADwin-Gold* system on a DIN rail.
- Single cable-connector for a self-made external power supply cable.

3 Operating Environment

The *ADwin-Gold* electronic is installed in a closed aluminum enclosure and it is only allowed to operate it in this enclosure. With the necessary accessories the system can be operated in 19-inch-enclosures or as a mobile system (e.g. in cars). See also chapter 2.2.2 "Accessories").

Earth protection

The *ADwin-Gold* 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 aluminum enclosure, via a short low-impedance solid-type cable to the central earth connection point of your device.

Galvanic connection

The power supply cable is the galvanic connection between the computer and the *ADwin-Gold*.

The version with USB interface has a galvanic connection to the computer or where appropriate also via the power supply.

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 *ADwin-Gold* aluminum enclosure by using cable shield ground clamps.

BNC cables

The shielding of BNC cables is normally used as differential ground and looses therefore the shielding effect. So BNC cables are influenced by interferences when differential measurements are executed. For signal and data transfer outside of an enclosure it is necessary to use twisted pair data transfer cables, whose channels are shielded, too.

Protection low voltage

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

Ambient temperature

The *ADwin-Gold* 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-Gold* before you have executed the **following steps**:

– Carry out completely the installation of the drivers and the power supply at the computer or notebook (see manual: "ADwin Driver Installation").

– connect the *ADwin-Gold* only with the computer or notebook (s.b.).

– Read chapter 5 "Inputs and Outputs" in this manual.

– Begin now with the connection of the inputs and outputs.

Please take into account that there is a galvanic isolation between the *ADwin-Gold* system and the computer via power supply cable, USB and Ethernet lines (see chapter 3, section "Galvanic connection").

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.

Power supply

The power supply connection of the *ADwin-Gold* with 12V (see Annex, Technical Data) is made via the built-in connector, at left next to the power switch or above the GND plug (see Fig. 4). Connect the 3-pin subminiature connector there. For the pin assignment see the following picture:

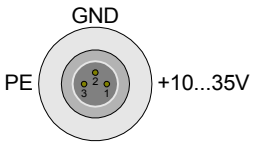


Fig. 3 – Power supply connector (male)

For using the system with an external power supply unit you need the subminiature connector described above. The connector is provided by the following manufacturer under the article number 712 299-0406-00-03 (Series 712):

Franz Binder GmbH + Co. elektrische Bauelemente KG
Rötelstrasse 27
74172 Neckarsulm,
Phone: ++49-7132 / 325-0
www.binder-connector.de

When using the system with a notebook, power has to be supplied by a separate power supply, (see chapter 2.2.2 on page 5). Please pay attention to the fact that it is sufficiently dimensioned.

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.

Connection

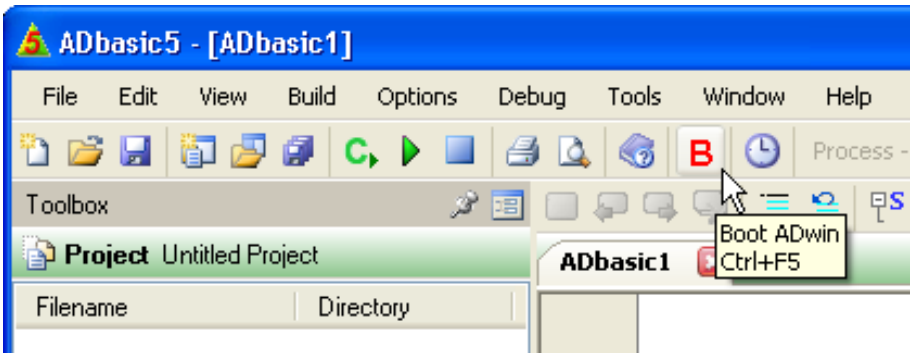
If you have completed the installation of the *ADwin* drivers and the configurations in the *ADbasic* menu "`Options\Compiler`", then connect the USB or Ethernet data transfer cables and the power supply cable. Then start the computer.

Power-up

In order to avoid switching off the system inadvertently, the switch is equipped with a blocking device. Pull the switch a little bit, then pull it into the direction "Power". Now the device is switched on and the LED lights up in red.

Booting

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



The flashing LED (green colored now) and the display in the 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).

Programming the *ADwin* systems is described more detailed in the *ADbasic* manual. Instructions for access to *ADwin-Gold* I/Os are described in chapter 12 on page 44.

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

Programs with *ADbasic*

5 Inputs and Outputs

Connectors

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 the *ADwin-Gold* system.

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.
An exception is the event input, which has already an internal pull-up resistance (10 kΩ).

The inputs and outputs of the ADwin-Gold II basic version is decscribed on the following pages:

– 16 analog inputs via 2 multiplexers (page 10)
– 2 analog outputs (page 11)
– 32 digital inputs/outputs (page 15)

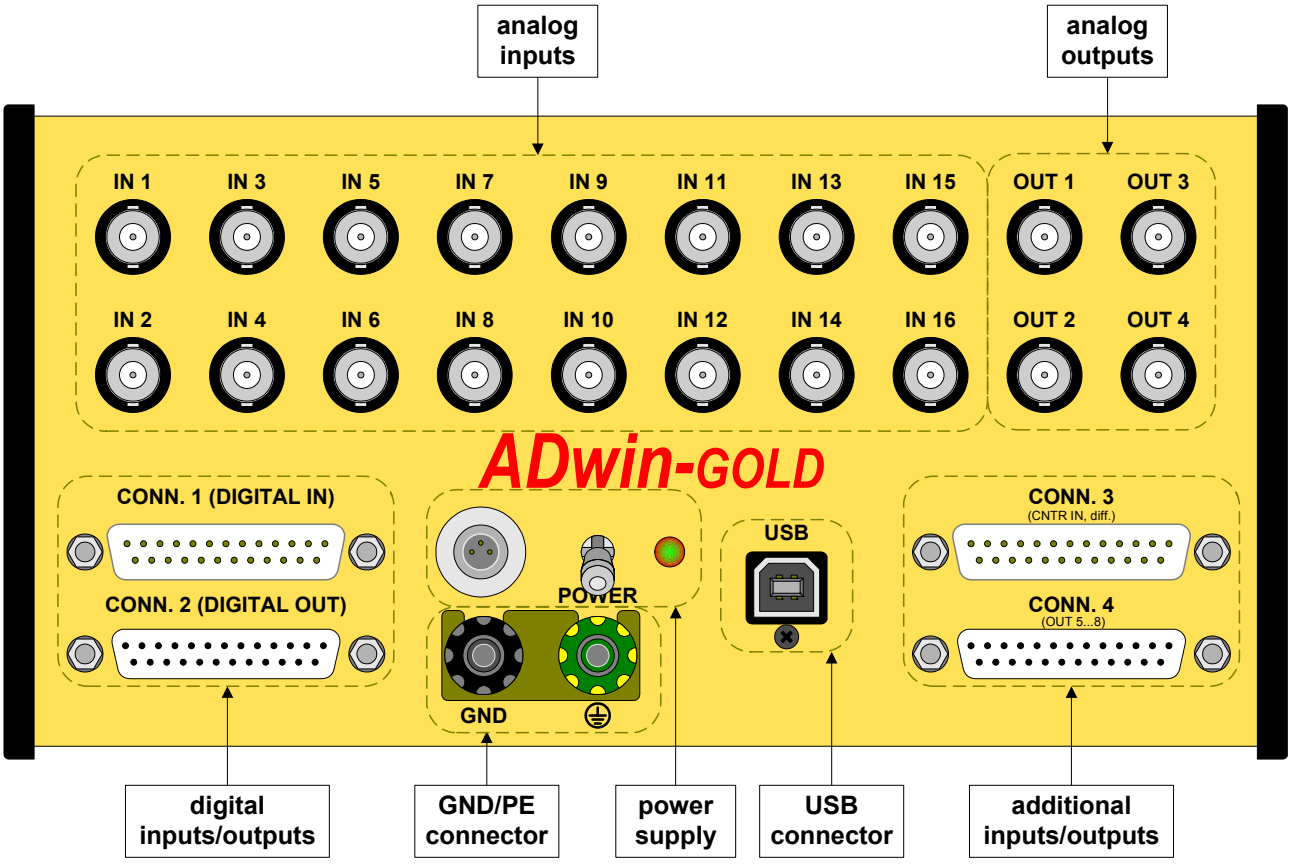


Fig. 4 – Schematic of *ADwin-Gold* (USB version)

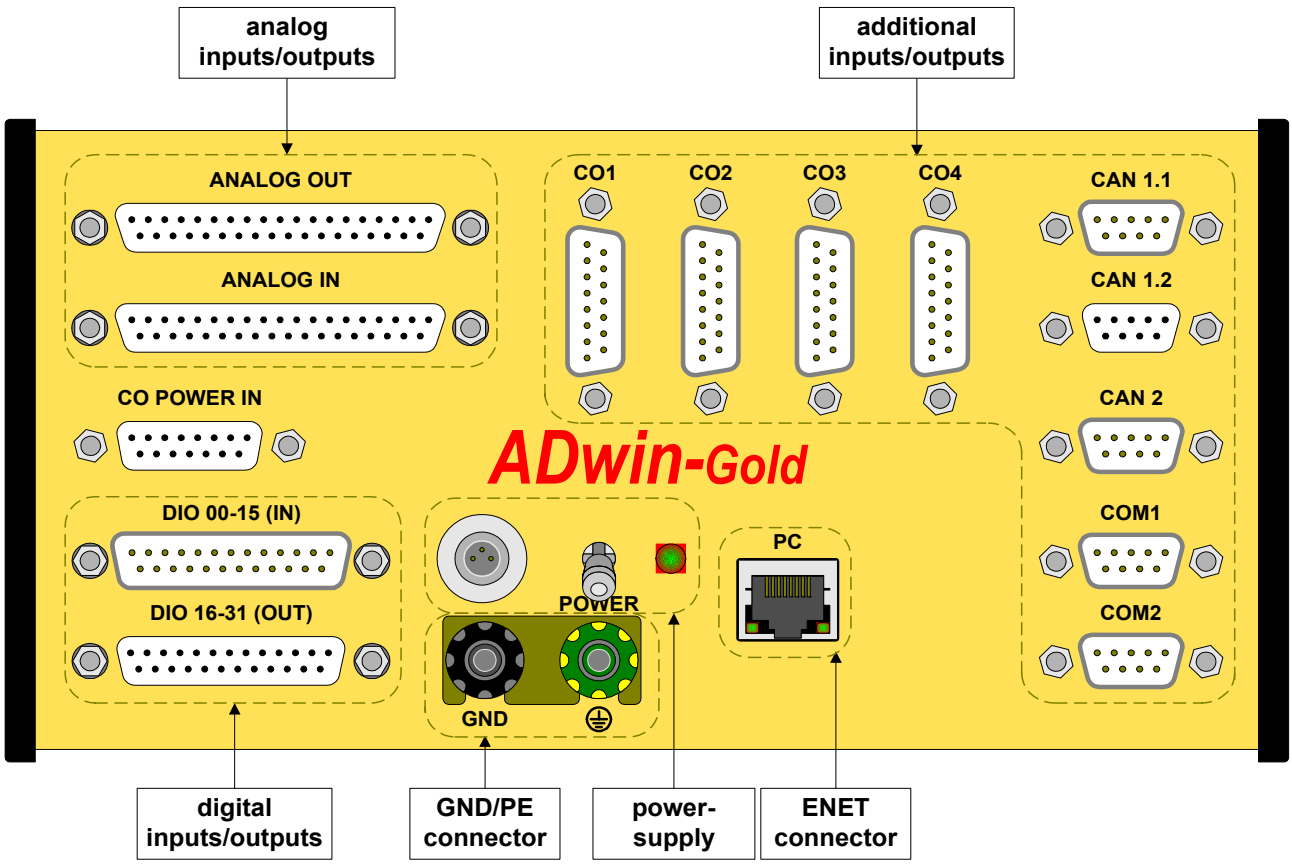


Fig. 5 – Schematic of *ADwin-Gold-D* (ENET version)

5.1 Analog Inputs and Outputs

In order to operate the system without any interferences, isolated BNC connectors are necessary. Otherwise there will be the danger of damages caused by ESD or short circuits at the inputs. This will be the case when using not isolated BNC T-pieces.

The *ADwin-Gold* device has to be connected to earth, in order to execute measurement tasks without any interferences. Connect the GND plug via a low-impedance solid-type cable with the central earth connection point of your device.

The power supply from the power adapter at the computer also connects the earth of the *ADwin-Gold* system with the earth of the computer. If you do not operate the PC and the *ADwin-Gold* system in the same place, you should not use the power supplied by the PC but an external power supply unit which is earth-free, in order to avoid influences by different ground reference potentials.

In addition to the description of the inputs and outputs you will find notes below for the conversion of digits into voltage values and for the input settings of the analog inputs.

Standard instructions

For fast and easy programming there are standard instructions available in the compiler *ADbasic*, which enable a user to easily measure or output data; see ADC (page 47) and DAC (page 46). Use other instructions only if extremely time-critical or special tasks require to do so.

5.1.1 Analog Inputs

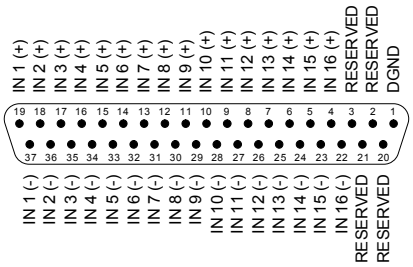
Multiplexer

The system has 16 analog inputs IN1 … IN16. The inputs with odd numbers (1, 3, … 15) are allocated to multiplexer 1, those with even numbers (2, 4, … 16) to multiplexer 2. The output of each multiplexer is connected to both a 14 bit-ADC and a 16 bit-ADC (see also "Block diagram of the ADwin-Gold", page 4).

Differential

The analog inputs are differential. For each of the measurement channels 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.

The inputs are equipped with male BNC-plugs, which are arranged in 2 rows; the Gold-D option has the inputs connected to the DSub-connector `ANALOG IN`. At the BNC-plugs, the positive input is the inner conductor, the negative input is the outer conductor.



ANALOG IN

Fig. 6 – Pin assignment of analog channels with Gold-D option

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.

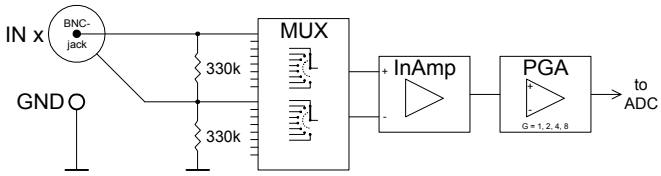


Fig. 7 – Input circuitry of an analog input

16-bit and 14-bit measurements

You can convert the signals at the multiplexer outputs optionally with a 14-bit or a 16-bit analog-to-digital-converter (ADC), (see Fig. 2 "Block diagram of the ADwin-Gold"). You are measuring with

– the 14-bit ADC very fast (max. 0.5µs, resolution 1.221mV)

– the 16-bit ADC very accurately (max. 5µs, resolution 305µV).

ADC instruction

The instructions **`ADC`**`()` for the 16-bit ADC and **`ADC12`**`()` for the 14-bit ADC execute a complete measurement with one of the ADCs on the analog input. The ADC instructions consider for instance the settling of the multiplexer and assure perfect measurements (see page 47).

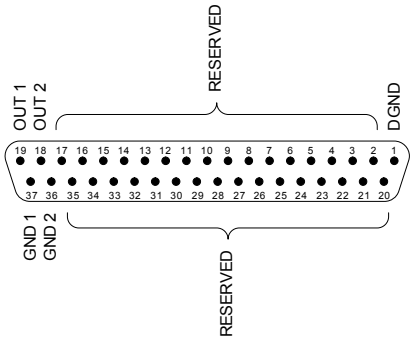
Please pay attention to a low internal resistance of the power supply unit (of the input signals), because it may have influence on the measuring accuracy. If this is not possible:

– Depending on the output resistance a linear error is caused.
  You can compensate this by multiplying the measurement value with a corresponding factor and get a sort of recalibration.

– From approx. 3kΩ upwards the multiplexer settling time extends.
  The waiting time defined in the standard instructions **`ADC`** and **`ADC12`** is then too short, so that imprecise values are recalled. In this case please use the instructions described in chapter 5.3.1.

5.1.2 Analog Outputs

The system has 2 analog outputs (OUT1, OUT2) with BNC-plugs; with Gold-D option the outputs are located on the DSub connector ANALOG OUT (see Fig. 6). A digital-to-analog converter (DAC) is allocated to each of the outputs.



ANALOG OUT

Additional outputs see chapter 7 "DA Add-On".

DAC instruction

The standard instruction **DAC** (`number`, `value`) (see page 46) checks each of the values if it exceeds or falls below of the 16-bit value range (0...65535). If the value is in the 16-bit value range, the indicated value is output on the output `number`. If it is not in the value range the maximum or minimum values are output.

5.1.3 Calculation Basis

Voltage range

The voltage range of the *ADwin-Gold* at the analog inputs and outputs is between –10V to +10V (bipolar 10V).

Allocation of digits to voltage

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

– 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 the 16-bit conversion and a voltage value of 9.998779V for the 14-bit conversion.

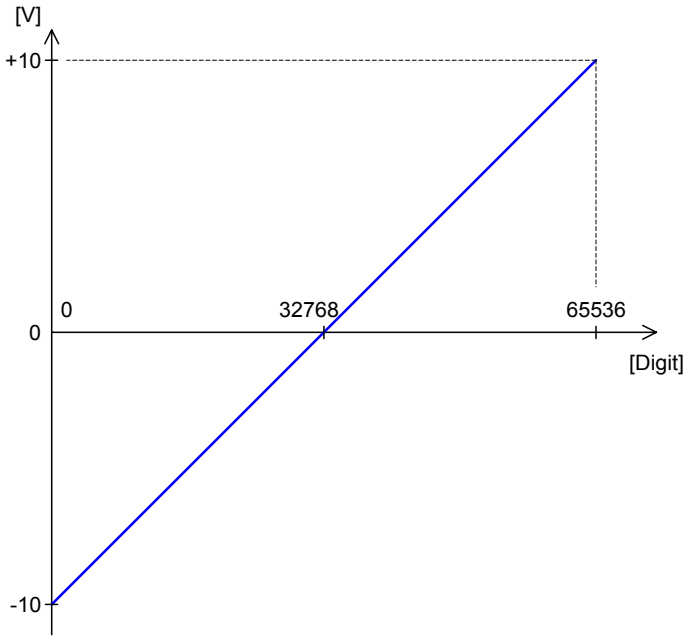


Fig. 8 – Zero offset in the standard setting of bipolar 10 Volt

Zero offset U_{OFF}

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

ADwin-Gold has a programmable gain (PGA), with which you can amplify the input voltage by the factors 1, 2, 4, and 8. At the same time the measurement range gets smaller by the corresponding gain factor k_V (see Annex "Technical Data").

Please note that upon applications with k_V >1 the 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). It is different for the two ADCs:

– 16-bit ADC: $U_{LSB} = 20V / 2^{16} = 305.175\mu V$

– 14-bit ADC: $U_{LSB} = 20V / 2^{14} = 1220.7\mu V$

The measured 16-bit value of the ADC is returned in the lower word of the register. A DAC value, which is to be output, has to be available there.

Bit No.	31…16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
32-bit-memory	0	16-bit value of the 16-bit ADC / DAC in the lower word															
	0	14-bit value of the 14-bit ADC in the lower word														0	0

Fig. 9 – Storage of the ADC/DAC bits in the memory

In order to compare the measurement values of the 14-bit ADC with the values of the 16-bit ADC, the converted value is written left-aligned into the lower word of the register at the 14-bit ADC. Therefore the lower 2 bits are always 0 (zero).

The 16384 digits of the 14-bit ADC are mapped to the 65536 digits of the 16-bit ADC. Thus 4 digits of the 16-bit ADC are equivalent to one digit of the 14-bit ADC.

Therefore the following equations can be used for both ADC types:

Conversion Digit to Voltage

DAC

For a DAC:

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

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

ADC

For an ADC (14-bit and 16-bit):

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

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

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.

DNL

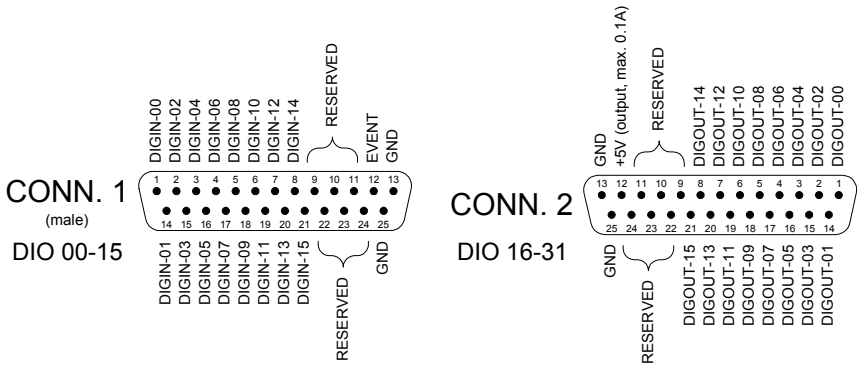
– The differential non-linearity (DNL) defines the maximum deviation from the ideal quantization level.

5.2 Digital Inputs and Outputs

Digital inputs/outputs

On two 25-pin D-SUB sockets (DIO 00…DIO 31) there are 32 digital inputs or outputs. They are programmable in groups of 8 as inputs or outputs.

After power-up of the device, all 4 groups are configured as inputs.



DIO-00…DIO-15 DIO-16…DIO-31

Fig. 10 – Pin assignment digital IOs

The digital inputs are TTL-compatible and not protected against over voltage.
Do not use pins marked as "reserved". They are planned for changes and expansions and can cause damages to your system if you do not pay attention to this fact.

Trigger input (EVENT)

The *ADwin-Gold* is equipped with an external trigger input (EVENT). With this trigger input processes are triggered by an external signal (trigger) with rising edge and can completely and immediately be processed, (see also *ADbasic* manual, chapter: "Program Structure").

Programming

Instructions to program analog inputs are described starting from page 56. The instructions are defined in the include file `<ADwinGoldII.inc>` and are described in the online help, too.

Function	Instructions
Configure	`CONF_DIO`
Read input values	`DIGIN`, `DIGIN_WORD`
Set output values	`DIGOUT_WORD`, `DIGOUT_WORD` `CLEAR_DIGOUT`

The instruction `CONF_DIO(12)` configures DIO 15:00 as digital inputs and DIO 31:16 as digital outputs.

`CONF_DIO`(12)

Only in this configuration will you be able to totally access the inputs and outputs with the above instructions. About programming under other configurations the following chapter will give you more detailed information: chapter 5.3 "Time-Critical Tasks" (see also tutorial).

5.3 Time-Critical Tasks

For extremely time-critical tasks you can use instructions with which you have direct access to the **control and data registers of the ADC and DAC** (see *ADbasic* manual). These registers can be found in the memory address area of the ADSP (memory mapped). These instructions also allow to optimize the program structure (s.b.).

Contrary to the standard instructions `ADC()`, `ADC12()` and `DAC()` the instructions for direct access **do not have any test routines.** Before you use them we recommend to learn more about time sequences, program structures and functions sequences in an ADC.

5.3.1 Analog Inputs and Outputs

`ADC()` and `ADC12()`

The standard instructions `ADC()` and `ADC12()` consist of a sequence of several instructions (see below). They need a certain time for execution. The execution time is mostly determined by the settling time of the multiplexer and the conversion time.

```
SET_MUX()
...                             'wait for settling of the
                                'multiplexer
START_CONV()
WAIT_EOC()                      'wait for end of conversion
READADC()                       'or READADC12() at ADC12()
```

Program structure

You can use (or extend) the waiting times caused by the standard instructions for other purposes. If you apply these instructions skillfully you may be able to execute faster measurements.

It is important to set the **START_CONV()** instruction (page 55) in a sufficient time-delay from the **SET_MUX()** instruction (page 53), in order to consider the multiplexer settling time.

Use the waiting times for instance for arithmetic operations and save CPU time:

- Settling time of the multiplexer: At a maximum voltage jump of 20 Volt it is 6.5 µs (max.) for the 16-bit ADC and 2.5 µs for the 14-bit ADC.
- Conversion time of the ADC: Its is 0.5 µs for the 14-bit ADC and 5 µs for the 16-bit ADC.

ADC

DAC

Direct Register Access

A measurement can be executed very fast, when you directly access the control and data registers of the ADC.

If you have made sure that at the analog outputs the values are within the range limits, you can write very quickly into one or more DAC registers with direct access to the hardware registers, and you can synchronously start the output, (see instructions **PEEK** and **POKE** in the *ADbasic* manual).

The hardware addresses for direct access to control and data registers are described in the annex.

5.3.2 Digital Inputs and Outputs

After power-up of the device all 4 connection groups are configured as inputs; this corresponds to the instruction **CONF_DIO(0)**. The following table shows how the inputs and outputs (IN, OUT) are configured when you use the value of the first column as instruction argument.

CONF_DIO()	DIO31:24	DIO23:16	DIO15:08	DIO07:00
0	IN	IN	IN	IN
1	IN	IN	IN	OUT
2	IN	IN	OUT	IN
3	IN	IN	OUT	OUT
4	IN	OUT	IN	IN
5	IN	OUT	IN	OUT
6	IN	OUT	OUT	IN
7	IN	OUT	OUT	OUT
8	OUT	IN	IN	IN
9	OUT	IN	IN	OUT
19	OUT	IN	OUT	IN
11	OUT	IN	OUT	OUT
12	OUT	OUT	IN	IN
13	OUT	OUT	IN	OUT
14	OUT	OUT	OUT	IN
15	OUT	OUT	OUT	OUT
Applicable instructions:	**DIGOUT_WORD**, **CLEAR_DIGOUT**, **SET_DIGOUT**		**DIGIN_WORD**, **DIGIN**	

CONF_DIO()	DIO31:24	DIO23:16	DIO15:08	DIO07:00
Instruction is applicable for DIOnn, at	Configuration "OUT"		Configuration "IN" At configuration "OUT" the register contents of this byte is returned	

Fig. 11 – Overview of the configuration with CONF_DIO

Please pay attention to the following restriction:

Only if the inputs/outputs are configured with CONF_DIO(12) (see pin assignmenton page 14) will you be able to fully access the inputs/outputs with the instructions DIGOUT_WORD, SET_DIGOUT, CLEAR_DIGOUT, DIGIN_WORD, DIGIN.

For any other configuration you have to read out or write into the corresponding hardware register (see instructions PEEK and POKE in the *ADbasic* manual). The hardware addresses for direct register access are described in the annex.

6 Calibration

6.1 General Information

The 2 digital-to-analog (DAC, optional 8) and the 4 analog-to-digital (ADC) converters of the *ADwin-Gold* systems have been calibrated in factory. In accordance with the regulations for keeping the measurement accuracy in your field of application, the systems must be calibrated in regular time intervals.

You calibrate the system with the program <GoldCalib.exe>; at standard installation the path is <C:\ADwin\Tools\ADwin-Gold>.

Tools

The following tools are necessary for the calibration:

– A digital multimeter (DMM) with an accuracy of
  - 30 µV when using 16-bit converters
  - 120 µV when using 14-bit converters
– A reference voltage source with an accuracy of
  - 30 µV when using 16-bit converters
  - 120 µV when using 14-bit converters
– connecting cables from the inputs/outputs to the reference voltage source and to the measurement device (recommended: BNC cables).

6.2 Calibrating

Connect your *ADwin-Gold* system with the computer and configure it according to the program <ADconfig.exe>.

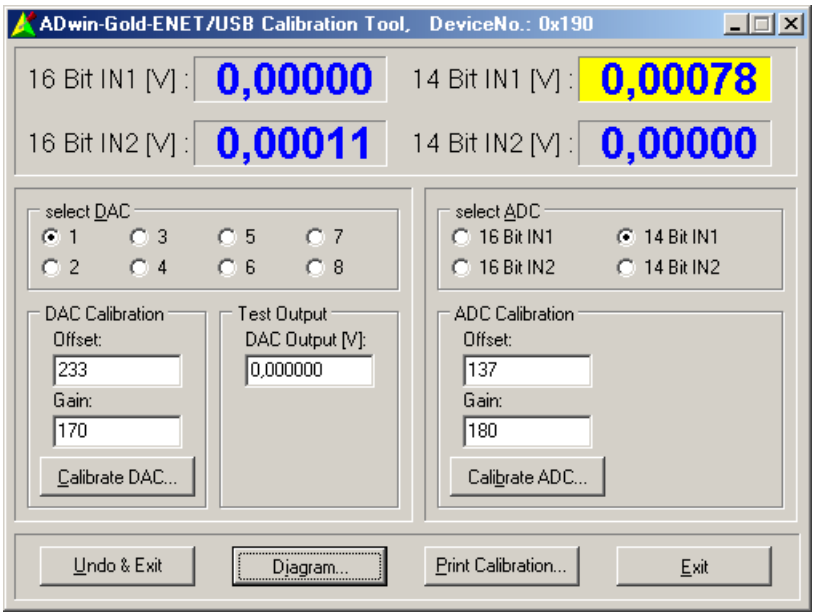
Calibration has to be made when the *ADwin-Gold* system reaches its operating temperature. With a power-up temperature of the device of approx. 20 to 25 degrees Celsius (room temperature), the system reaches the operating temperature approx. 30 minutes after power-up.

Step 1

Start the **calibration program** <GoldCalib.exe>. The window "ADwin-Gold-ENET/USB Calibration Tool" appears.

Choose the device number of the system you want to calibrate and confirm by clicking on "OK".
You will get a warning when you haven't chosen an *ADwin-Gold* system or one of an older firmware version. You can ignore the warning with "Yes" or return to the previous window with "No".

An **overview window** appears. In the header line the device number you have selected is shown.



The upper field shows the current measurement values at the inputs IN1 and IN2, measured each by the 16-bit and the 14-bit ADC.

Select in the lower field left the DAC which you wish to calibrate and at right the ADC. The measurement value at the selected ADC is highlighted. Below you will find the calibration settings for `Offset` and `Gain` of the DAC and the ADC. There you can directly enter values. With `"Calibrate DAC"` or `"Calibrate ADC"` you start the calibration of the selected converter.
In the dialog box `"Test Output"` you can enter a voltage value, which is automatically output at the converter/output you have selected earlier.

Every input you make is immediately transferred to the *ADwin-Gold* system. If you close the program with `"Exit"`, the new settings remain. With `"Undo&Exit"` you undo all inputs and you exit the calibration program (that means the original settings are transferred to the *ADwin-Gold* system).
`"Diagram"` displays in a graph the accuracy of the current calibration setting. You print a protocol of the settings with `"Print Calibration"`.

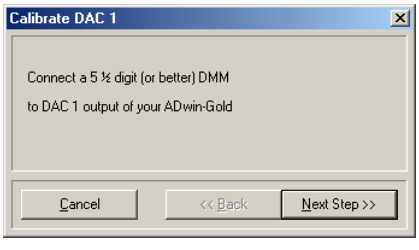
Calibrate the converters in the order you like (only with reference voltage source). The calibration of a converter is effected in 3 steps; you can switch between the windows of the steps by using the forward/backward buttons.

Calibration is also possible without reference voltage source, but it will not be so precise. Calibrate first the DACs and then the ADCs.

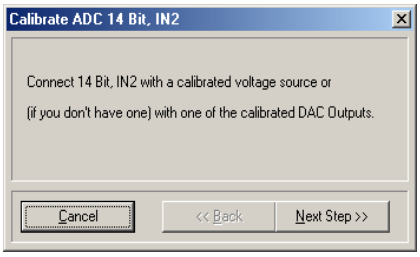
Step 2

The 3 levels for **calibrating a converter** are described below, for the DAC in the left column and for the ADC in the right column.

1. Connect the external device (DMM / voltage source):
   Select the corresponding key `"Calibrate ..."` for calibrating a converter; the first window appears.:



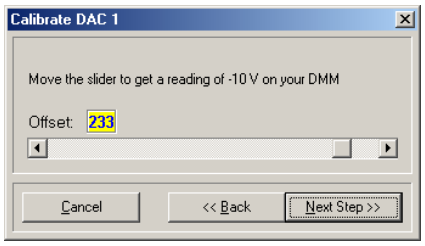
Connect a DMM to the selected output.



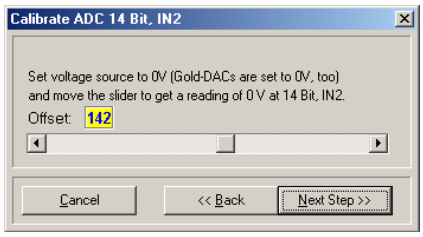
Connect the voltage source (or a calibrated DAC output) to the selected input.

Please note Fig. 4 "Schematic of ADwin-Gold (USB version)".
Select "Next Step >>".

2. Setting the offset



Adjust the offset value at the scrollbar in such a manner that your digital multimeter displays -10 V.

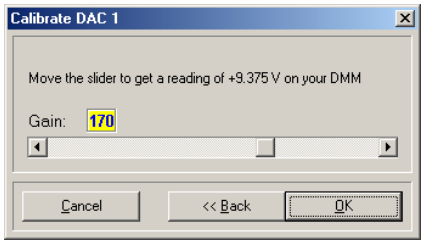


Set your voltage source to 0 V.The setting of the ADC to this value is made automatically.
Adjust the offset value at the scrollbar in such a manner that the setpoint at the ADC is displayed in the overview window.

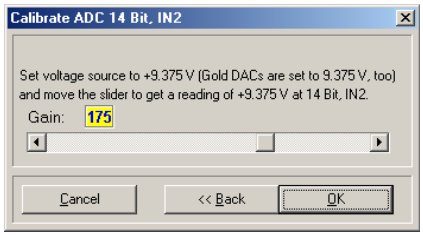
Select "Next Step >>".

3. Setting the gain

:



Adjust the offset value at the scrollbar in such a manner that your digital multimeter displays 9.375V.

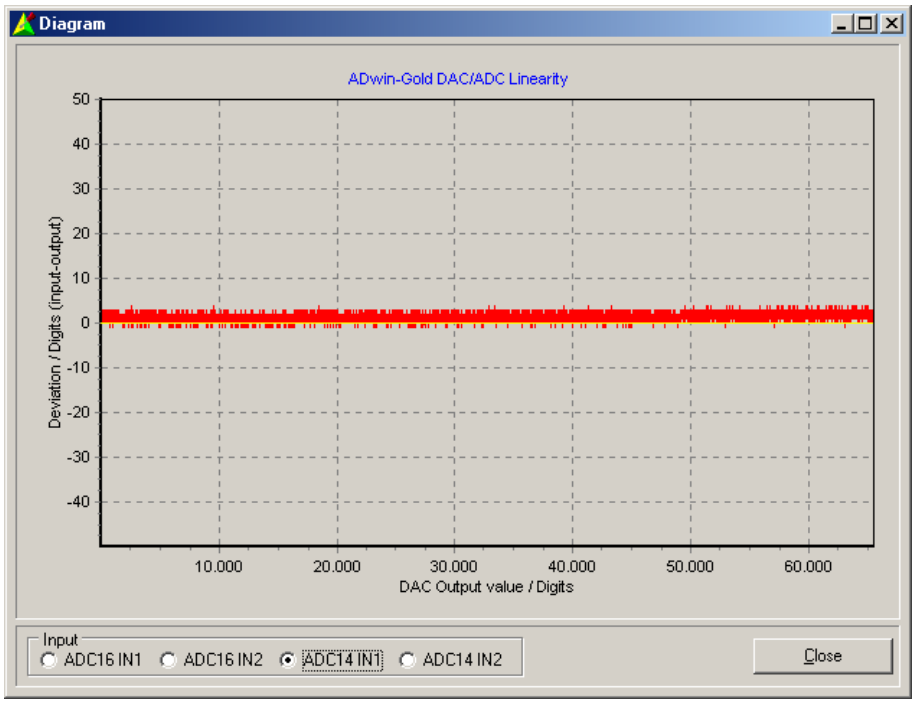


Set your voltage source to 9.375 V (setpoint). The setting of the ADC to this value is made automatically.
Adjust the offset value at the scrollbar in such a manner that the setpoint at the ADC is displayed in the overview window.

The calibration for this converter has finished. Select "OK". Repeat step 2 for the other converters if necessary.

Step 3

With a diagram (button Diagram in the overview window) you can check the **accuracy** of the calibration. First connect any 2 outputs with the inputs IN1 and IN2. Select in the diagram one of the inputs and the corresponding converter.



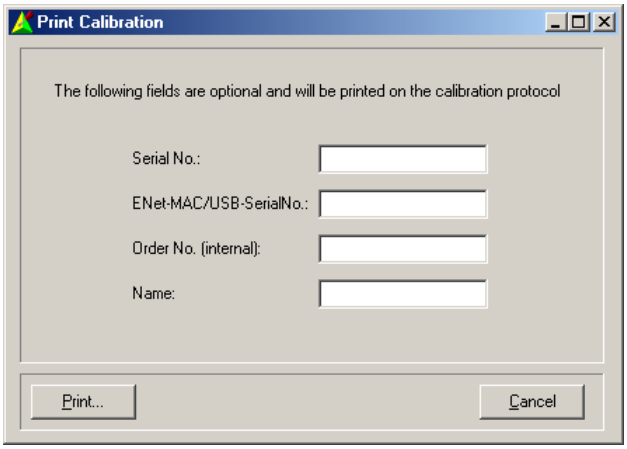
The program outputs the values 0...65535 digits on both DACs, compares them to the measured input values and displays the deviation in graphs.
The deviation should be less than 5 digits.

With `Close` you return to the overview window.

Step 4

With "`Print Calibration`" you can print a **protocol** of the specified calibration data.

In the open window you can enter different information which will be presented in your printout (for a later allocation to the protocol). With "`Print`" you start printing; the program automatically returns to the overview window.
In the protocol you will find the calibration settings of all inputs and outputs for gain and offset as well as the date of print.



The calibration is finished.

Step 5

7 DA Add-On

Connectors

With the DA add-on you have **8 analog outputs** in total with a resolution of 16 bit (and a DAC each).

In the standard version two of these outputs go from DAC 3 and DAC 4 to the BNC plugs OUT 3 and OUT 4. The other 4 outputs connect DAC 5...DAC 8 to the pins 1...4 and 14...17 of the 25-pin D-SUB socket CONN4 (see figure).

With the Gold-D option all additional outputs are connected to the pins of the DSub socket ANALOG OUT.

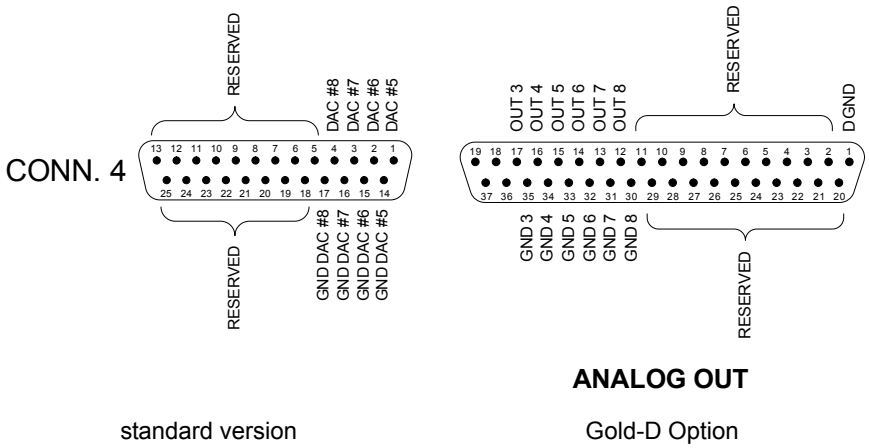


Fig. 12 – Pin assignment of the DA add-on

Programming and calibration

You program and calibrate the additional DACs like the DAC 1 and DAC 2 (see chapter 5.1, chapter 6 and chapter 12).

8 CO1 Counter Add-On

The counter add-on *Gold-CO1* (ordering option) has four 32-bit up/down counters with four-edge-evaluation.

The technical data of the counter add-on *CO1* is described in the annex A.1.

8.1 Hardware

The counter add-on *Gold-CO1* has four 32-bit up/down counters with four-edge-evaluation. You can configure and read out the counters individually as well as all together. (The block diagram shows the design of a single counter).

Latch A and B

The counters can be internally or externally clocked and are read out via accompanying latches. All counters have each a Latch A and a Latch B. The counter values can be cleared or transferred in a latch by using programming commands or (if configured) when there is an external signal at CLR/LATCH.

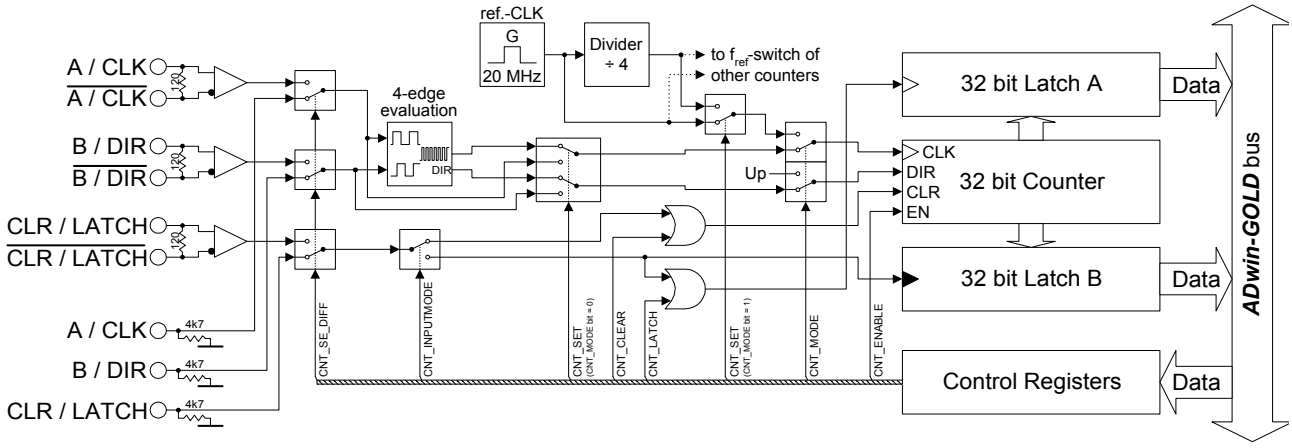


Fig. 13 – Block diagram of the *Gold-CO1* counter add-on

There are the following operating modes: event counting (external clock) and pulse width measurement (internal clock); see also chapter 8.3 / 8.4:

External clock input

a) **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).

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:** 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.

Internal clock input

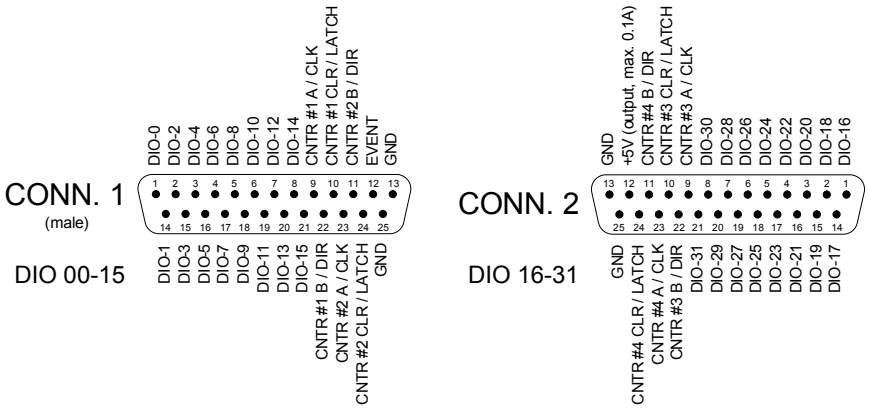
b) **Pulse width measurement:** Incrementing/decrementing of the counter is caused by an internal reference clock generator; a signal frequency of 5 MHz or 20 MHz can be used. The square-wave signal at CLR/LATCH is evaluated: With every positive edge the counter values are written to latch A, with a negative edge to latch B.

You can calculate:

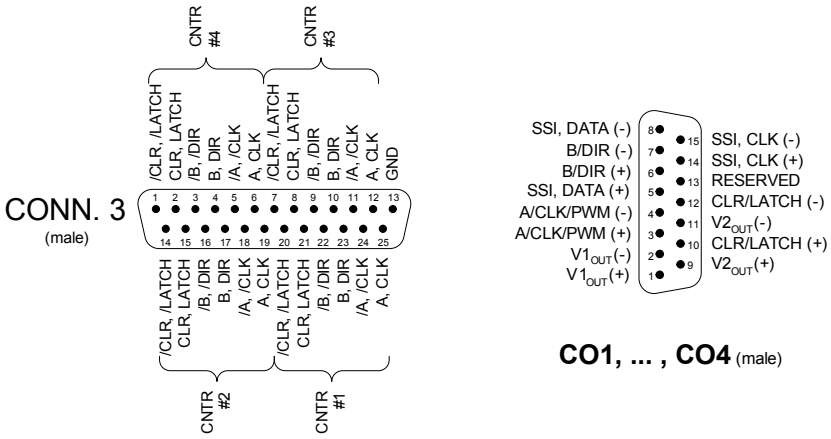
1. the **period duration** of the input signal at CLR/LATCH from the values in latch A or latch B.
2. the **pulse width** and **pause time** from the values in latch A and latch B

The counters are controlled by *ADbasic* instructions via a control register (instructions, see table in chapter 8.2).

At the inputs A/CLK, B/DIR and CLR/LATCH TTL-alike signals are necessary. More details and limit values can be found in the "Technical Data".



Counter inputs with TTL operation mode for Gold / Gold-D
(single-ended; mode not available with Rev. B2)



with Gold only

with Gold-D only

Counter inputs with differential operation mode

Fig. 14 – Pin assignment of the CO1 add-on

In any case you have to set the input operation mode with the instruction **CNT_SE_DIFF**. This is done in pairs, i.e. the counters 1 and 2 together and the counters 3 and 4 together (see page 79).
With Rev. B2 differential operation mode can be set only, TTL operation mode (single ended) is available from Rev. B3.

Although all inputs for the *CO1* add-on have a pull-down resistor, not-connected inputs can cause errors in an environment which is not protected against interferences. If you do not use a counter input, connect for safety reasons both lines of the (differential) input to a specified potential: Connect the positive input to +5V and the negative input to GND.

On the option Gold-D you can – via the connector `CO Power in` – supply a voltage, which is then available at the connectors `CO1`...`CO4`, e.g. for external increment encoder.
Please note: All minus inputs V1in (-) are galvanically connected to GND via a common line; the minus inputs V2in (-) have such a common connection, too.

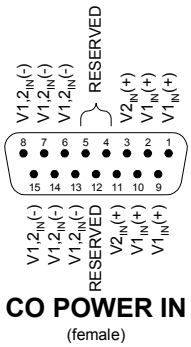


Fig. 15 – Pin assignment counter voltage supply (Gold-D)

8.2 Software

The functions necessary for accessing the counters can be found in the include file:

```
<ADWGCNT.INC>
```

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 12, starting from page 65.

Instruction	Function
CNT_CLEAR() *	Clear counter
CNT_ENABLE()	Disable or enable counter (please note the already running counters)
CNT_GETSTATUS(#)	Read out status register (# = counter no. 1…4)
CNT_RESETSTATUS()	Clear status register of all counters.
CNT_INPUTMODE()	Set CLR/LATCH input to CLR or LATCH mode
CNT_LATCH() *	Latch counter values into Latch A
CNT_MODE()	Use external clock input or internal reference clock
CNT_SE_DIFF()	Set clock inputs to differential or single-ended (as pairs)
CNT_SET()	In combination with **CNT_MODE**(): Set counter mode or length of the internal reference clock
CNT_READ(#)	Read out counter values and transfer them to Latch A (# = counter no. 1…4)
CNT_READLATCH(#)	Read out Latch A (triggered by positive edge), (# = counter no. 1…4)
CNT_READFLATCH(#)	Read out Latch B (triggered by negative edge), (# = counter no. 1…4)

* These functions are reset after they have been executed. All other functions are reset by opposing functions.

Fig. 16 – Instructions of the *Gold-CO1* counter add-on

With the instructions in the table matrix you are always effecting *all* counters (except **CNT_READ**…). 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**, **CNT_SET**, **CNT_INPUTMODE**, **CNT_SE_DIFF**)
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.

Please pay attention to the fact that the **CNT_SET** instruction depends on the **CNT_MODE** instruction.

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.

8.2.1 Evaluation of the Counter Contents

The binary counters of the CO1 add-on 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 (231-1) follows the highest negative number (-231) and the lowest positive number (0) follows the highest negative number (-1).

Circle

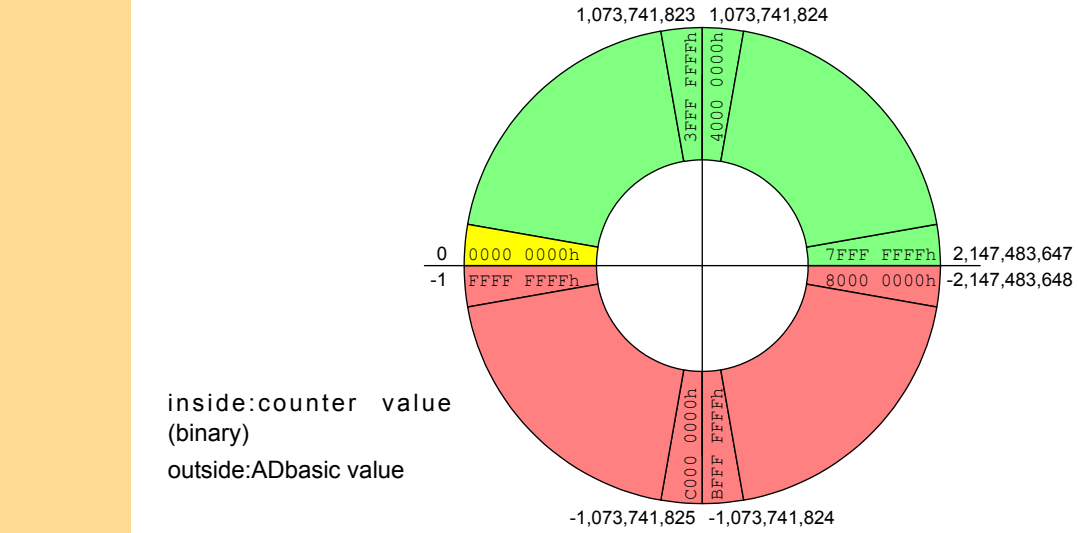


Fig. 17 – Circle for the interpretation of counter values

Please pay attention to the following rules for programming:

a) Process the read 32-bit value only with variables of the type **INTEGER** or **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 3½ minutes with an input frequency of 20 MHz or after more than 14 minutes with 5 MHz.

Example programs

You will find several example programs for the *CO1* add-on in the directory <C:\ADwin\ADbasic3\samples_ADwin_Gold> (standard installation).

8.3 Operating Mode Impulse/Event Counting

External square-wave signals at the inputs A/CLK and B/DIR clock the counters in this mode. With **CNT_SET** you either activate the mode for determining the clock frequency and direction or the four edge evaluation.

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

Clear

– clear the counter (CLR)

Latch

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

8.3.1 Clock and Direction

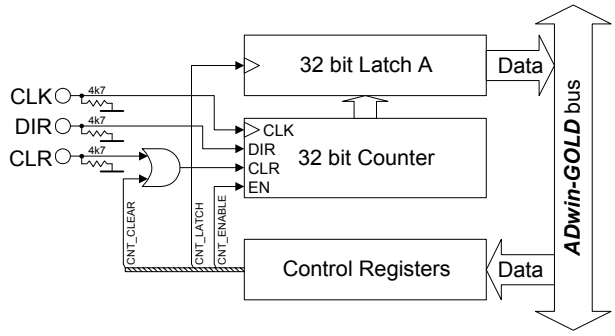


Fig. 18 – Block diagram of the CO1 add-on in the mode "clock and direction"

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.

Programming example

```
INIT:                    initialization ...
. . .
CNT_ENABLE(0)            disable counters
CNT_CLEAR(1)             clear counter(s)
CNT_MODE(0)              external clock input (CLK)
CNT_SET(1)               activate mode "clock and direction"
CNT_INPUTMODE(0)         input CLR/LATCH as CLR input
CNT_SE_DIFF(0)           select single-ended inputs
CNT_ENABLE(1)            enable counter(s)
EVENT:                   event loop ...
CNT_LATCH(1)             current counter value to latch A
CNT_READLATCH(1)         read out latch A
. . .                    evaluate counter value in process
```

8.3.2 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.

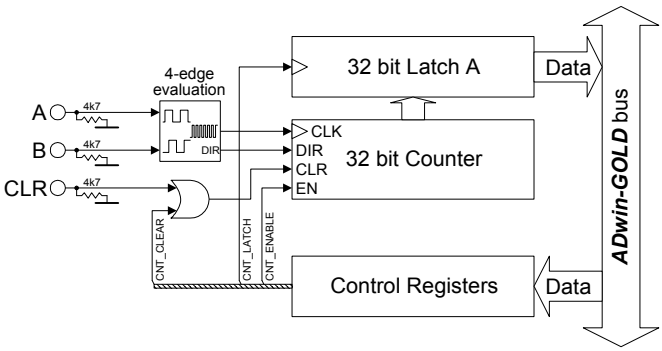


Fig. 19 – Block diagram of the *CO1* add-on in the mode "four edge evaluation"

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 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 MHz decreases for instance to 45 degrees at 2.5 MHz.

Programming example

`INIT:`	initialization ...
`. . .`	
`CNT_ENABLE(0)`	disable counters
`CNT_CLEAR(1)`	clear counter(s)
`CNT_MODE(0)`	external clock input (CLK)
`CNT_SET(0)`	activate mode "four edge evaluation"
`CNT_INPUTMODE(0)`	input CLR/LATCH as CLR input
`CNT_SE_DIFF(0)`	select single-ended inputs
`CNT_ENABLE(1)`	enable counter(s)
`EVENT:`	event loop ...
`CNT_LATCH(1)`	current counter value to latch A
`CNT_READLATCH(1)`	read out latch A
`. . .`	evaluate counter value in process

8.4 Operating Mode Impulse Width and Period Width Measurement

Reference clock generator

In this operating mode an internal reference clock generator clocks the counter with a signal frequency of 20 MHz or (after a prescaler) 5 MHz. All counters have a switch in order to change the signal frequency. The period duration or pulse width of a square-wave signal at input CLR/LATCH can be measured.

In this mode you have to consider at high frequencies that your `GLOBALDELAY` remains smaller than a signal period, in order to acquire a cycle.

8.4.1 Period Duration Measurement

All four counters can execute period duration measurements.

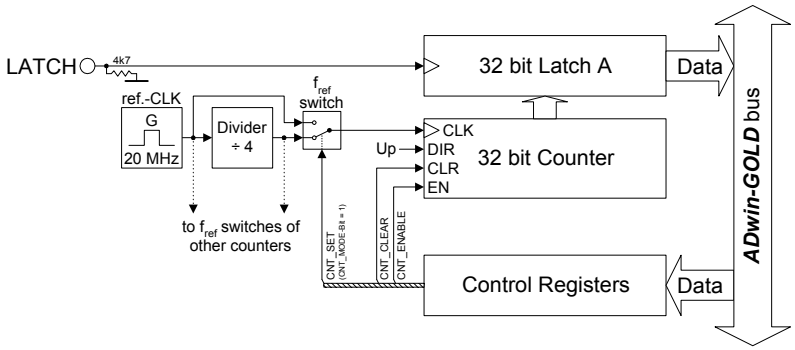


Fig. 20 – Block diagram of the *CO1* add-on in the mode "period duration measurement"

In this mode, the counter values are latched into Latch A at every positive edge, and the previous data are overwritten. The pulse width will be derived from the counter value difference multiplied by the period duration of the reference clock.

Programming example

```
INIT:                          initialization ...
. . .
CNT_ENABLE(0)                  disable counters
CNT_CLEAR(1)                   clear counter(s)
CNT_MODE(1)                    mode "internal reference clock" with ...
CNT_SET(0)                     ... 20 MHz or
CNT_SET(1)                     ... 5 MHz reference frequency
CNT_INPUTMODE(1)               input CLR/LATCH as LATCH input
CNT_SE_DIFF(0)                 select single-ended inputs
CNT_ENABLE(1)                  enable counter(s)

EVENT:                         event loop ...
CNT_READLATCH(1)               read out latch A
. . .                          evaluate counter value in process
```

8.4.2 Impulse Width and Pause Duration Measurements

All 4 counters measure impulse width and pause duration.

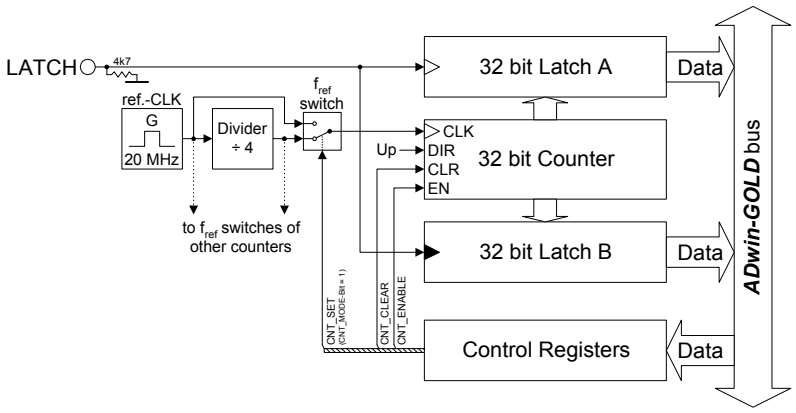


Fig. 21 – Block diagram of the *CO1* add-on mode
"impulse width/pause duration"

The counters 1 and 2 have two latches for positive (Latch A) and negative edges (Latch B). Thus, pulse and pause duration can be evaluated separately by calculating the differences of the latches.

Programming example

```
INIT:                          initialization ...
. . .
CNT_ENABLE(0)                  disable counters
CNT_CLEAR(1)                   clear counter(s)
CNT_MODE(1)                    mode "internal reference clock" with ...
CNT_SET(0)                     ... 20 MHz or
  CNT_SET(1)                   ... 5 MHz reference frequency
CNT_INPUTMODE(1)               input CLR/LATCH as LATCH input
CNT_SE_DIFF(0)                 select single-ended inputs
CNT_ENABLE(1)                  enable counter(s)

EVENT:                         event loop ...
CNT_READLATCH(1)               read out latch A
CNT_READFLATCH(1)              read out latch B
. . .                          evaluate counter value in process
```

8.4.3 Hardware addresses (*CO1*-add-on)

A process can be executed very quickly if you access directly the control and data register (see chapter 5.3 and instructions **PEEK** and **POKE** in the *ADbasic* manual).

The hardware addresses of the CO1 add-on can be found in the annex (compare to list of instructions in chapter 8.2).

9 CAN add-on

The add-on *Gold-CAN* is equipped with several additional interfaces that are configured and operated individually:

– 4 SSI decoders (page 32)

  The decoders can be used for the connection of incremental encoders with SSI interface. All inputs are differential and designed for RS422/485 level (5V).

  The decoder inputs are located on the connectors CO1...CO4, where the inputs of the CO1 add-on can also be found.

– 2 CAN interfaces (page 34)

  Depending on your requirements, you can order both interfaces either as high-speed or low-speed version. Switching in operation is not possible.

  The inputs of the CAN 1 interface are located on the connectors CAN 1.1 and CAN 1.2, those of the CAN 2 interface on the connector CAN 2.

– 2 RSxxx interfaces (page 37)

  Both interfaces can be configured separately per software to be operated as RS232 or RS485.

  The interface inputs are located on the connectors COM1 and COM2.

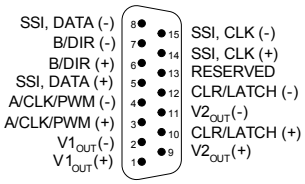
The add-on *Gold-CAN* is only available in combination with the Gold-D option.

9.1 SSI Decoder

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

The decoders either read out an individual value (on request) or they continously provide the current value.

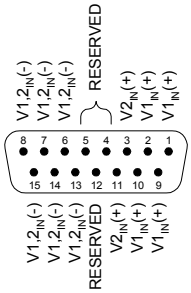
The connections of the 4 decoders are on the connectors CO1...CO4 (15-pin, DSUB), on the pins 5, 8, 14 and 15 (see fig. 22). If the device is equipped with the CO1 add-on the remaining pins are reserved for the counter connections.



CO1, ... , CO4 (male)

Fig. 22 – Pin assignment SSI decoder

A voltage input to the connection `CO Power in` is supplied at the connectors CO1...CO4, for example for an external incremental encoder.
Please note: The negative inputs $U1_{in}$ (-) are galvanically connected with GND via a common line, the negative inputs $U2_{in}$ (-) have such a connection, too.



CO POWER IN
(female)

Setting the properties

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

- Clock rates: With `SSI_SET_CLOCK` clock rates of approx. 40 kHz up to 1 MHz are possible with a pre-scaler.
- 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. 23 – Listing: Conversion of Gray code into binary code

Programming

The functionality of the decoders is easily programmed with ***ADbasic*** instructions:

Range	Instructions
Initialization	SSI_MODE SSI_SET_BITS SSI_SET_CLOCK
Receiving of data	SSI_READ SSI_START SSI_STATUS

The instructions are in the include file <ADWGCAN.INC>. More information can be found in the ***ADbasic*** manual and the online help.

9.2 CAN Interface

The CAN interfaces 1 and 2 can be operated individually. Depending on your requirements, you can order both interfaces either as high-speed or low-speed version. Switching in operation is not possible.

9.2.1 Hardware Description

The connections of the interfaces 1 and 2 are located on the 9-pin DSUB connector:

– Interface 1: Connector (male) CAN 1.1 and connector (female) CAN 1.2. The pins of the connectors are internally connected with each other.

– Interface 2: Connector CAN 2.

The pinouts for CAN "High speed" and "Low speed" are different.

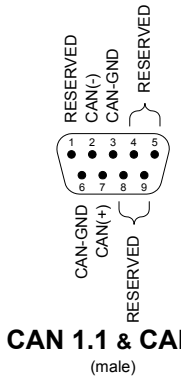
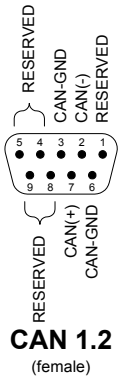
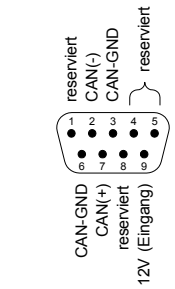
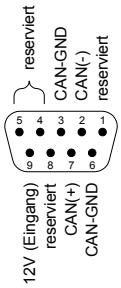
CAN Bus	Male connectors CAN 1.1, CAN 2	Female connector CAN 1.2
High speed		
Low speed		

Fig. 24 – CAN: Pinbelegungen

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

Power supply (Low speed only)

The low speed version requires an external power supply of 12 V DC to run the CAN controller. The power must be supplied for each interface separately.

Bus Termination (High speed only)

If the CAN interface functions as the physical termination of a high-speed CAN bus, it must be terminated with a 120 Ω 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 120 Ω.

9.2.2 Description of the CAN interface

The CAN bus interface is equipped with the Intel® CAN controller AN82527 which 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.

Message

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 15 message objects.

The registers are used for configuration and status display of the CAN controller. Here the bus speed and interrupt handling, etc. are set (see separate documentation "82527 - Serial Communications Controller, Architectural Overview" by Intel®)

The CAN bus can be set to frequencies of up to 1 MHz and is usually operated with 1 MHz; with low speed CAN the max. frequency is 125 kHz. The CAN bus is galvanically isolated by optocouplers from the ***ADwin*** system.

An arriving message can trigger an interrupt which instantaneously generates an event at the processor. Therefore an immediate processing of messages is guaranteed.

Message Management

Identifier

The CAN controller identifies messages by an identifier; these are parameters in a defined bit length. The parameters 0...2^{11}-1 or 0...2^{29}-1 result from the bit length.

Message objects

The controller stores each message (incoming or outgoing) in one out of 15 message objects. The message objects can either be configured to send or to receive messages. Message object 15 can only be used to receive messages.
After initializing the CAN controller all message objects are not configured.

Each message object has an identifier, which enables the user to assign a message to a message object.

Transferring messages

In ***ADbasic*** a message is transferred to a message object using the array `can_msg[][]`, which can receive 8 data bytes plus the amount of data bytes (9 elements). When reading a message from the message object it can also be transferred to the array `can_msg[][]`.

Sending messages

Sending a message is made as follows:

- You configure a message object to send and define the identifier of the object (instruction **`EN_TRANSMIT`**).
- Save the message in `can_msg[]`.
- Send the message (instruction **`TRANSMIT`**). The message in the array `can_msg[]` is transferred to the message object. As soon as the bus is ready, the message is sent (with the identifier of the message object).

Receiving messages

Receiving a message is made as follows:

- You configure a message object to receive and define the identifier of the object (instruction **`EN_RECEIVE`**).
- The controller monitors the CAN bus if there are incoming messages and saves messages with the right identifier in the message object.
- Transfer the message from the message object into the array `can_msg[]` (instruction **`READ_MSG`**) and read out the corresponding identifier.

An arriving message overwrites the old data in the message object, which will be definitely lost. Therefore pay attention to reading out the data faster than you are receiving them. A data loss is indicated by a flag.

The message object 15 has an additional buffer, so that 2 messages can be stored there.

Assigning messages

The allocation of an arriving message to a message object is automatically controlled by comparing its identifiers. The global mask (CAN registers 6...7 or 6...9) controls this comparison as follows:

- The identifier of the message is bit by bit compared to the identifier of the message object. If the relevant bits are identical, the message is transferred to the message object. Not relevant bits are not compared to each other, that is, the message is transferred to the object (if it depends on this bit).
- Relevant bits are set in the global mask.

Global mask

With the global mask a message object is used for receiving messages with **different identifiers** (ID). The following example shows the assignment of the message IDs 1...4 to the message object IDs 1...4, when all bits of the global mask are set, except the two least-significant bits (if you have an 11-bit identifier it is `11111111100b`).

Message ID	ID of the message object			
	1 …001b	2 …010b	3 …011b	4 …100b
1 (…001b)	x	x	x	0
2 (…010b)	x	x	x	0
3 (…011b)	x	x	x	0
4 (…100b)	0	0	0	x

x: Message is admitted
0: Message is not admitted

In this example the comparison of bit 2 is responsible for the assignment of the messages, because the bits 3...10 of the compared identifiers are identical (= 0) and the bits 0 and 1 are not compared, because they are set to zero in the global mask (= not relevant).

Setting the bus frequency

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

The initialization with **INIT_CAN** configures the controller automatically to a CAN bus frequency of 1 MHz. If the CAN bus is to operate with a different frequency, just use the instruction **SET_CAN_BAUDRATE**.

With low speed CAN the maximum bus frequency is 125 kBit/s.

Bus frequency for special cases

In some special cases it may be better to select configurations other than those set with **SET_CAN_BAUDRATE**. For this purpose specified registers have to be set with the instruction **POKE**. The structure of the register is described in the controller documentation.

Enable Interrupt / Trigger Event

A message object can be enabled to trigger an interrupt when a message arrives. The interrupt output of the CAN controller is connected to the event input of the processor. The processor reacts immediately to incoming messages without having to control the message input (polling).

You can enable the interrupts of several message objects. Which object has caused the interrupt can be seen in the interrupt register (5Fh): It contains the number of the message object that caused the interrupt. If the interrupt flag (new message flag) is reset in the message object, the interrupt register will be updated. If there is no interrupt the register is set to 0. If another interrupt occurs during working with the first interrupt its source will be shown in the interrupt register. An additional interrupt does not occur in this case.

Programming

The interface is easily programmed using ***ADbasic*** instructions:

Range	Instructions
Initialization	**INIT_CAN** **EN_CAN_INTERRUPT** **SET_CAN_BAUDRATE**
Receiving and sending of data	CAN_Msg **EN_RECEIVE**, **EN_TRANSMIT** **READ_MSG**, **READ_MSG_CON**, **TRANSMIT**
Write / read access to the controller register	**SET_CAN_REG** **GET_CAN_REG**

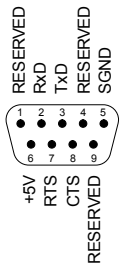
The instructions are in the include file <ADWGCAN.INC>. More information can also be found in the ***ADbasic*** manual and the online help.

9.3 RSxxx Interfaces

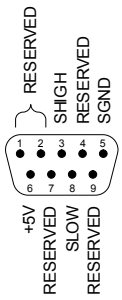
Each of the 2 RSxxx interfaces is equipped with the "Quad Universal Asynchronous Receiver/Transmitter" controller (UART), type TL16C754 from Texas Instruments®. Functionality and programming of the interfaces are based on this controller.

Both interfaces can be operated individually with the RS232 or RS485 protocol. The physical difference between the protocols is the level of the signals, which are generated by special driver components on the bus.

Pin assignment



COM1, COM2
(RS232) *(male)*



COM1, COM2
(RS485) (male)

Bus termination (RS485 only)

If an RS485 interface functions as the physical bus termination, the terminator must be a resistor (only the first or last RS485 participant). RS485 participants, which are not positioned in an end-location, must not be terminated.

For the termination there is–if required for the chosen circuit type–a voltage of +5 V provided at pin 6. For bus termination please note, that the voltage line is equipped with a 330 Ω resistor.

9.3.1 Setting the interface parameters

Each interface has an input and an output FIFO with a length of 64 bytes each.

The settings of the interface parameters are made separately for each channel, using the controller register. Below the settings are described more detailed:

Handshake

– Handshake: The interface is operated in 4 modes:

1. RS232 without handshake
2. RS232 with software handshake (Xon/Xoff)
3. RS232 hardware handshake (RTS/CTS). The signals RTS and CTS must be connected.
4. RS485

Parity

– 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

– Data bits: the active data to be transferred may be 5...8 bits long.

Stop bits

– Stop bits: The number of stop bits can be set to 1, 1½ or 2. Here the number of stop bits depends on the number of data bits:
  - 5 data bits: 1 or 1½ stop bits.
  - 6…8 data bits: 1 or 2 stop bits.

Baud rate

– Baud rate: The physical data are between 35 Baud and 2.304 MBaud; when using an RS-232 interface the maximum Baud rate is 115.2 kBaud.

  The Baud Rates are derived by the clock rate of the module; the basic clock rate has a frequency of 2.304 MHz. Based on this fact, every Baud rate is possible that can be derived by an integer division of the basic frequency. The divisor can have values between `1...0FFFFh`. The following table shows some common Baud rates and their divisors:

Baud rate	Divisor	
	dez.	hex.
2304000	1	0001h
1152000	2	0002h
460800	5	0005h
230400	10	000Ah
115200	20	0014h
57600	40	0028h
38400	60	003Ch

Baud rate	Divisor	
	dez.	hex.
19200	120	0078h
9600	240	00F0h
4800	480	01E0h
2400	960	03C0h
1200	1920	0780h
600	3840	0F00h
300	7680	1E00h

Fig. 25 – RS-xxx: Baud rates

Special features of RS485

Via a RS485 interface more than 2 participants can communicate with each other. (Contrary to the RS232 interface). With RS485 interfaces a bus can be set up.

Consider the following:

– There is no handshake, because a handshake is only possible between 2 participants.

– The interface must know if it should write to the bus or get data from the bus (`RS485_SEND`).

9.3.2 Programming

Functionality and programming of the interface depend on this controller. The controller is easily programmed with ***ADbasic*** instructions:

Range	Instructions
Initialization	`RS_INIT`, `RS_RESET`
Receiving and transmitting of data	`RS485_SEND`, `READ_FIFO`, `WRITE_FIFO`
Write and read access to the controller register	`GET_RS`, `SET_RS`

The instructions are in the include file `<ADWGCAN.INC>`. More information can also be found in the ***ADbasic*** manual and the online help.

RS232

Example programs

The following program illustrates the initialization of the serial RS232 interface in the INIT: section and the cyclic reading and writing of data in the EVENT:. section. The process is timer-controlled:

```
REM The program initializes the serial interface
REM in the Init: section.
REM In the Event: section data is exchanged between
REM the interfaces 1 & 2 of the RS module.
REM The interfaces are tested with this program.
REM For this connect the interfaces with each other
REM befor starting the program.

#INCLUDE adwpext.inc
DIM DATA_1[1000] AS LONG'Transmitted data
DIM DATA_2[1000] AS LONG'Received data
DIM lauf AS LONG           'Control variable

INIT:
  FOR run = 1 TO 1000      'Initialization of the transmit-
                           'ted data
    DATA_1[run] = run AND 0FFh
  NEXT run
  REM Initialization of the interfaces:
  REM 9600 Baud, not parity bit, 8 data bits,
  REM 2 stop bits, RS232 witout handshake
  RS_INIT(19600.0.8,1,0)
  RS_INIT(2,9600,0,8,1,0)
  PAR_1 = 1
  PAR_4 = 1

EVENT:
  REM Read and write a data set
  IF (PAR_1 <= 1000) THEN'Send data
    PAR_2 = WRITE_FIFO(1,DATA_1[PAR_1])
    IF (PAR_2 = 0) THEN INC PAR_1
  ENDIF

  PAR_3 = READ_FIFO(2)     'Read data
  IF (PAR_3 <> -1) THEN
    DATA_2[PAR_4] = PAR_3
    INC PAR_4
  ENDIF
  IF (PAR_4 > 1000) THEN END'All data are transmitted
```

RS485

In this example the RS485 interface is a passive participant, which reads data coming from the input. If a specified value (55) is received, the interface starts to send. It sends continuously the value 44.

```
REM Interface 2 reads all data coming from the bus
REM until it receives the value 55. Now the interface
REM becomes active and sends the value 44.

#INCLUDE adwgcan.inc
DIM ret_val, val AS LONG

INIT:
  RS_RESET()
  REM Initialization of the interfaces:
  REM 38400 Baud, no parity bit, 8 data bits,
  REM 1 stop bit, RS485 software handshake
  RS_INIT(1,38400,0,8,0,3)
  RS_INIT(2,38400,0,8,0,3)
  RS485_SEND(1,1)           'Send interface 1
  RS485_SEND(2,0)           'Receive interface 2

EVENT:
   val = READ_FIFO(2)       'Read data from interface 2

   IF (val = 55) THEN
    RS485_SEND(2,1)         'Send interface 2
    ret_val = WRITE_FIFO(2,44) 'Write data
   ENDIF
```

10 *ADwin-Gold*-Boot

This option is only available in an ***ADwin-Gold-ENET***.

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

With ***ADwin-Gold-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 boot loader option:

Disable boot loader

– Boot the system after power-up and the previously saved processes are disabled.

– After switching off and powering up anew, the boot loader option is enabled again.

By programming the Flash-EEPROM without processes and only with the file `<ADwin9.btl>` the system will only be booted after power-up, but no processes can be executed.

With the installation of the ***ADwin*** Developer-Software from the supplied ***ADwin*** CD-ROM, the utility program for the boot loader (ADethflash) is automatically copied. You should have a CDROM version 3.00.2735 or a later version.

Use the program `<ADethflash.exe>` for an ***ADwin-Gold*** system with Ethernet interface.

At standard installation you will find the program in the directory

`<C:\ADwin\Tools\Ethernet Interface\…>`.

You will find information about the boot loader with Ethernet interface in the ***ADwin*** Driver Installation manual.

Help for Ethernet interface

In combination with the Ethernet interface and boot loader you can write or read out 2000 long or float values à 32-bit via ***ADbasic*** processes into/from the Flash- EEPROM. A more detailed description can be found in the program `<ADethflash.exe>` by clicking on `"Info about eeprom support"`.

2000 values you can freely dispose of

11 Accessories

The following accessories are available for the ***ADwin-Gold***:

– ***ADwin-Gold***-pow: external 12V power supply unit (necessary for notebook operation).

  On the secondary side ***ADwin-Gold-pow*** provides 12 Volt at a maximum load of 2 Ampere. The power supply unit is rated for the highest load and maximum expansions of the ***ADwin-Gold***.

  Please pay attention to the fact that the USB, Ethernet cables are sufficiently shielded, in order to avoid interferences in the data lines. Interferences have to be passed before entering the chassis via GND (ground). (See also chapter 3 "Operating Environment").

– various lengths of power supply and USB or Ethernet cable

– *Gold-Mount*: kit for installation of the *ADwin-Gold* system on a DIN rail.

– cable connector for an external power supply

12 Software

You are programming ADwin-Gold - all add-ons included - with simple ADbasic instructions. Basic instructions are described in the ADbasic manual.

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

12.1 Analog Inputs and Outputs

This section describes the following instructions:

DAC

DAC outputs a defined voltage on a specified analog output.

Syntax

```
DAC(dac_no,value)
```

Parameters

dac_no	Number of the analog output (1…8).	LONG
val	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.

See also

ADC

Valid for

Gold, Gold-DA

Example

```
REM Digital proportional controller
DIM set_to, gain, diff, Out AS LONG 'Declaration

EVENT:
  set_to = Par_1          'Setpoint
  gain = Par_2            'Dimension
  diff = set_to - ADC(1)'Calculate control deviation
  Out = diff * gain       'Calculate actuating value
  DAC(1, Out)             'Output of the actuating value
```

ADC

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

If specified, the return value is multiplied by a gain factor.

For the 12-/14-bit converter use the instruction **ADC12**.

Syntax

```
ret_val = ADC(channel{,gain})
```

Parameters

channel	Number (1…16) of the analog input channel.	LONG
gain	Optional: gain factor (1, 2, 4, 8).	LONG CONST
ret_val	Measurement value in digits (0…65535).	LONG

Notes

ADC is a combination of consecutive functions:

- **SET_MUX**: Set the multiplexer to the specified input channel.
- Wait for settling of the multiplexer.
- **START_CONV**: Start measurement: Convert analog signal-considering the gain factor-to a digital value.
- **WAIT_EOC**: Wait for the end of conversion.
- **READADC**: Read out digital value from the register and return it.

Multiplexer settling time and conversion time are given on page 14.

If you indicate a non-existing input channel the measurement result will be undefined.

The execution time for the instruction depends on the system you use. You will find Information about the multiplexer settling time and the conversion time in the hardware documentation of your system.

If you set the process cycle time (**PROCESSDELAY**) to a value less than 20 µs, the execution time of the instruction is only half as long. This is possible, because the compiler skips the waiting time for the settling of the multiplexer. It is assumed that you want to execute a measurement without setting the multiplexer. If (at such short cycle times) you require the first measurement to be correct, you have to set the multiplexer to the specified input channel prior to using the instruction **ADC** with **SET_MUX** for the first time. This time has to be at least as long as the multiplexer settling time.

In the following examples the instructions **SET_MUX**, **START_CONV**, **WAIT_EOC** and **READADC** should be used instead of **ADC** in the following cases:

- Very short cycle times: **PROCESSDELAY** < 240 (s.a.).
- 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.

The measurement range depends on the gain factor:

Gain factor	Input voltage range	Measurement range
1	-10V … 10V	20V
2	-5V … 5V	10V
4	-2.5V … 2.5V	5V
8	-1.25V … 1.25V	2.5V

With the following formula you can calculate the measured voltage from the returned digital value.

$$\text{Voltage} = (\text{Digits} - 32768_{\text{bipolar}}) \cdot \frac{\text{measurement range}}{65536}$$

The following values, shown in the table below, apply in case you have chosen a gain of 1 (measurement range of 20 Volt):

Measurement range	Return value of ADC			1 Digit is
	0	32768	65535	
20V	-10V	0V	+9.999695V	305.175µV

See also

ADC12, ReadADC, Set_Mux, Start_Conv,Wait_EOC, DAC

Valid for

Gold

Example

```
DIM iw AS LONG          'Declaration

EVENT:
  'Measure analog input 1 with gain of 4
  iw = ADC(1,4)
  'Write measurement value into global variable, so
  'that the computer can read it
  Par_1 = iw
```

ADC12

ADC12 measures the voltage of an analog input via 12-bit (rev. A) or 14-bit converter (rev. B).

The measurement value is returned in digits, multiplied by a gain factor if specified.

For the 16-bit converter use the instruction **ADC**.

Syntax

```
ret_val = ADC12(channel{,gain})
```

Parameters

channel	Number (1…16) of the analog input channel.	LONG
gain	Optional: gain factor(1, 2, 4, 8).	LONG CONST
ret_val	Measurement result in digits: 12-bit: 0, 16, 32, …, 65520 14-bit: 0, 4, 8, …, 65532.	LONG

Notes

ADC12 is a combination of consecutive functions:

- **SET_MUX**: Set the multiplexer to the specified input channel.
- Wait for settling of the multiplexer.
- **START_CONV**: Start measurement: Convert analog signal-considering the gain factor-to a digital value. If specified, the digital value is multiplied by a gain factor.
- **WAIT_EOC**: Wait for the end of conversion.
- **READADC12**: Read out digital value from the register and return it.

Multiplexer settling time and conversion time are given on page 14.

If you indicate a non-existing input channel the measurement result will be undefined.

The execution time for the instruction depends on the system you use. You will find Information about the multiplexer settling time and the conversion time in the hardware documentation of your system.

The steps of 16 and 4 of the returned measurement values result from the fact that the 12-bit and 14-bit conversion results are returned each as a 16-bit value: The bits 0 to 3 are always 0 (zero) with 12-bit converters and bits 0 and 1 with 14-bit converters.

In the following examples you should use the instructions **SET_MUX**, **START_CONV**, **WAIT_EOC** and **READADC12** instead of **ADC** in the following cases:

- Very short cycle times: **PROCESSDELAY** < 200: **ADC12** cannot be executed during the cycle time.
- High internal resistance (>3k Ω) 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.

The measurement range depends on the gain factor.

Gain	Input voltage range	Meas. range
1	-10 V … 10 V	20V
2	-5 V … 5 V	10V
4	-2.5 V … 2.5 V	5V
8	-1.25 V … 1.25 V	2.5V

With the following formula you can calculate the measured voltage from the returned digital value:

$$\text{Voltage} = (\text{Digits} - 32768_{\text{bipolar}}) \cdot \frac{\text{measurement range}}{65536}$$

The following values, shown in the table below, apply in case you have chosen a gain of 1 (measurement range of 20 Volt):

Measurement range	Return value of ADC12			1 Digit is
	0	32768	65535	
20V	-10V	0V	+9.99512V	4.88mV

See also

ADC, ReadADC12, Set_Mux, Start_Conv,Wait_EOC

Valid for

Gold

Example

```
DIM iw AS LONG          'Declaration

EVENT:
  'Measure analog input 1 with a gain of 4
  iw = ADC12(1,4)
  'Write measurement value into global variable so that
  'the computer can read it.
  Par_1 = iw
```

ReadADC

READADC returns a converted value from a 16-bit A/D-converter.

Syntax

```
ret_val = READADC(adc_no)
```

Parameters

adc_no	Number (1, 2) of the 16-bit converter to read.	LONG
ret_val	Measurement value in digits which corresponds to the voltage at the converter's input.	LONG

Notes

READADC12 reads the converted values of the 12-bit or 14-bit A/D converter.

See also

ADC, ReadADC12, Set_Mux, Start_Conv,Wait_EOC

Valid for

Gold

Example

```
EVENT:
  'Set multiplexer: ADC1 to channel 3, ADC2
  'to channel 4 (without gain)
  SET_MUX(1001b)
  …                         'Wait for MUX settling time
  START_CONV(11b)           'Start conversion for both ADCs
  WAIT_EOC(11b)             'Wait for end of conversion
  Par_1 = READADC(1)        'Read value of ADC1
  Par_2 = READADC(2)        'Read value of ADC2
```

ReadADC12

READADC12 returns a converted value from one of the two 12-bit/14-bit A/D converters.

Syntax

```
ret_val = READADC12(adc_no)
```

Parameters

adc_no	Number (1, 2) of the 12-bit converter to read.	LONG
ret_val	Measurement value in digits, which corresponds to the voltage at the converter's input.	LONG

Notes

READADC reads the converted value of the 16-bit A/D converter.

The A/D converters (ADC) divide the measurement range of 20 Volts into equal steps (digits), these are 4096 digits with 12-bit ADC and 16384 with 14-bit ADC.

In order to make comparing these values to the measurement values of the 16-bit ADC's easier, the instruction **READADC12** returns the result "left-aligned" descending from bit 31; the bits 3…0 (12-bit ADC) or 1…0 (14-bit ADC) have always the value 0.
Therefore using the instructions **READADC** and **READADC12** to measure the same voltage always return the same result in bits 31…4 or 31…2.

See also

ADC12, Set_Mux, Start_Conv,Wait_EOC

Valid for

Gold

Example

```
DIM val1, val2 AS LONG

EVENT:
  'Set multiplexer: ADC12-1 to channel 3, ADC12-2
  'to channel 4 (without gain)
  SET_MUX(1001b)
  …                       'Wait for MUX settling time
  START_CONV(11000b)      'Start conversion for both ADCs
  WAIT_EOC(11000b)        'Wait for end of conversion
  val1 = READADC12(1)     'Read value of ADC12-1
  val2 = READADC12(2)     'Read value of ADC12-2
```

Set_Mux

SET_MUX sets one or more A/D input multiplexers and the corresponding gain for the specified measurement channel.

Syntax

```
SET_MUX(pattern)
```

Parameters

`pattern` Bit pattern for the allocation of measurement channels and gain. LONG

Bit no.	9 8	7 6	5 4 3	2 1 0
	PGA 2	PGA 1	MUX 2	MUX 1

PGA 1 / 2 — 2 bits (6…7 / 8…9) each determine the gain factor of the multiplexer:

2 Bits	PGA 1 / PGA 2
00:	Factor 1
01:	Factor 2
10:	Factor 4
11:	Factor 8

MUX 1 / 2 — 3 bits each (0…2 / 3…5) determine the channel to which the multiplexer is set:

3 bits	MUX 2	MUX 1
000:	channel 2	channel 1
001:	channel 4	channel 3
010:	channel 6	channel 5
011:	channel 8	channel 7
100:	channel 10	channel 9
101:	channel 12	channel 11
110:	channel 14	channel 13
111:	channel 16	channel 15

Notes

Please consider that when setting the multiplexer to another channel a specified settling time is required. You should only start the conversion after this settling time has elapsed.

Multiplexer settling time and conversion time are given on page 14.

It is preferable to use a binary code (suffix "b") for the bit pattern. This will make it easier to display the bit pattern than if you use a decimal or hexadecimal representation although it is still possible to use these.

See also

ADC, ADC12, ReadADC, ReadADC12, Start_Conv,Wait_EOC

Valid for

Gold

Example

To set the multiplexer of ADC1 to channel 5 and to gain 8 and at the same time the multiplexer of ADC2 to channel 10 and gain 2, you need the bit pattern: `0111100010b` (decimal: 482).

```
DIM val AS LONG

EVENT:
  SET_MUX(0111100010b)'Set multiplexer (s.a.)
  'Wait here for the settling time of the multiplexer
  'by inserting some instructions.
  START_CONV(1)          'Start AD-conversion ADC1
  WAIT_EOC(1)            'Wait for end of conversion of ADC1
  val = READADC(1)       'Read value of ADC1
```

Start_Conv

START_CONV can start the conversion of one or more A/D converters as well as of all the D/A converters.

Syntax

```
START_CONV(pattern)
```

Parameters

pattern — Bit pattern that specifies which converters should be started (only bits 0…4 can be used). CONST, LONG

Bit no.	31…5	4	3	2	1	0
ADC1, 16-bit	–	–	–	–	–	x
ADC2, 16-bit	–	–	–	–	x	–
all DACs	–	–	–	x	–	–
ADC1, 12-bit ADC1, 14-bit	–	–	x	–	–	–
ADC2, 12-bit ADC2, 14-bit	–	x	–	–	–	–

Notes

ADC1 and ADC2 can either be 12-bit, 14-bit or 16-bit analog-to-digital converters. For more information see page 10.

You can only use constants as parameters, variables are not allowed.

It is preferable to use a binary code (suffix "b") for the bit pattern. This will make it easier to display the bit pattern than if you use a decimal or hexadecimal representation although it is still possible to use these.

See also

ADC, ADC12, ReadADC, ReadADC12, Set_Mux,Wait_EOC

Valid for

Gold

Example

```
DIM val1 AS LONG

EVENT:
  SET_MUX(0)              'Set multiplexer to channel 1
  'Bypass the settling time with command lines
  START_CONV(1)           'Start ADC1 A/D-conversion
  WAIT_EOC(1)             'Wait for end of conversion
  val1 = READADC(1)       'Read out value
```

Multiplexer settling time see page 14.

Wait_EOC

WAIT_EOC waits for the end of the conversion cycle of a specified A/D-converter.

Syntax

```
WAIT_EOC(pattern)
```

Parameters

pattern — Bit pattern that specifies which converters are to be waited for (only bits 0…4 can be used). CONST, LONG

Bit no.	31…5	4	3	2	1	0
ADC1, 16-bit	–	–	–	–	–	x
ADC2, 16-bit	–	–	–	–	x	–
ADC1, 12/14-bit	–	–	x	–	–	–
ADC2, 12/14-bit	–	x	–	–	–	–

Notes

If you set more than one of the bits, you have to wait for the conversion to finished for all of the relevant ADCs.

Always select the bits of existing ADCs. Otherwise the communication in a high-priority process between *ADwin* system and computer will be interrupted.

See also

ADC, ADC12, ReadADC, ReadADC12, Set_Mux, Start_Conv

Valid for

Gold

Example

```
DIM val AS LONG

EVENT:
  SET_MUX(001000b)      'Set MUX of ADC2 to channel 4
  'Bypass the settling time of the multiplexer with
  'command lines
  START_CONV(2)         'Start A/D-conversion ADC2
  WAIT_EOC(2)           'Wait for end of conversion at 'ADC2
  val = READADC(2)      'Read out value
```

Multiplexer settling time see page 14.

12.2 Digital Inputs and Outputs

This section describes the following instructions:

Clear_Digout

CLEAR_DIGOUT sets one of the digital outputs to 0 (TTL low).

Syntax

```
CLEAR_DIGOUT(bit_no)
```

Parameters

bit_no — Bit number (0…15) which specifies the output (see table). CONST, LONG

bit_no	0	1	…	14	15
Output	DIO16	DIO17	…	DIO30	DIO31

Notes

CLEAR_DIGOUT accepts only constants as parameter. If you want to specify the output to be deleted using a variable, use **DIGOUT_WORD**.

You have to configure the relevant channel as output, otherwise **CLEAR_DIGOUT** has no effect.
With **CONF_DIO** you can configure the digital channels in groups of 8 inputs or outputs. We recommend the digital channels to be configured with **CONF_DIO**(1100b): Channels 0…15 as inputs, channels 16…31 as outputs.

CLEAR_DIGOUT clears a bit in the output register of the channels DIO16…DIO31. Therefore a TTL low is set at the corresponding channel, as long as it has been defined as output.

If you want to set one of the channels 0...15 to 0, clear the corresponding bit in the output register of the channels DIO0…DIO15 (note: Configure the channel as output first). Follow these steps (see example below):

- Read out the register with **PEEK**.
- Clear the bit belonging to the channel (**AND** masking).
- Write the value back into the register with **POKE**.

You will find the register number in the table in the annex, chapter A.2.

See also

Clear_Digout, Conf_DIO, Digout_Word, Set_Digout, Peek, Poke, And

Valid for

Gold

Example

```
DIM val AS LONG          'Declaration

INIT:
  SET_DIGOUT(0)          'Set digital output DIO16 to 1

EVENT:
  val = ADC(1)           'Measurement data acquisition
  IF (val > 3000) THEN
    CLEAR_DIGOUT(0)      'Clear dig. output DIO16
  ENDIF
```

A subroutine which sets a single bit of the DIO lines 0...15 to 0 could be as follows:

```
SUB CLEAR_DIGOUT_CONN1(bitno)
  POKE(204001C0h, PEEK(204001C0h) AND NOT(SHIFT_LEFT(1,bitno)) )
ENDSUB
```

Conf_DIO

CONF_DIO configures the 32 digital channels in groups of 8 as inputs or outputs.

Syntax

```
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.	CONST LONG

Bitno. in pattern	15…4	3	2	1	0
Channels	–	DIO31 … DIO24	DIO23 … DIO16	DIO15 … DIO08	DIO07 … DIO00

Notes

CONF_DIO accepts only a constant as parameter pattern.

The digital channels are initially configured as inputs after power-up (and cannot be used as outputs). They can only be configured in groups of 8 as inputs or outputs.

It is recommended that you use 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.

We recommend the use of the configuration **CONF_DIO**(1100b), which specifies DIO00…DIO15 as inputs and DIO16…DIO31 as outputs.
The instructions **CLEAR_DIGOUT**, **SET_DIGOUT**, **DIGIN_WORD**, **DIGOUT_WORD**, **DIGIN** are dependent on this configuration; a different configuration can interfere with or prevent the proper operation of these commands.

If you use a configuration other than the recommend configuration, you can only set and process the digital channels if you read out or write into the corresponding hardware registers with **PEEK** and **POKE** commands (see table in the annex, chapter A.2).

See also

Clear_Digout, Digin, Digin_Word, Digout_Word, Set_Digout, Peek, Poke

Valid for

Gold

Example

```
'Configure DIO00…DIO15 as inputs
'and DIO16…DIO31 as outputs
CONF_DIO(1100b)
```

Digin

DIGIN returns the value of one of the digital inputs DIO00…DIO15.

Syntax

```
ret_val = DIGIN(channel_no)
```

Parameters

channel_no	Number which specifies the input to be queried (see table below).	LONG, CONST
ret_val	1: TTL-level high. 0: TTL-level low.	LONG

channel_no	0	1	…	14	15
Input No.	DIO00	DIO01	…	DIO14	DIO15

Notes

DIGIN accepts only a constant as parameter channel_no.

DIGIN fits best for the reading of few bits. If several bits are to be read (e.g. in a loop), the usage of the instruction **DIGIN_WORD** is definitely quicker. Please remember this for time-critical applications in particular.

The instruction requires that you configure the relevant channel as input. If the channel is configured as output it will return an irrelevant value.

CONF_DIO can be used to configure the digital channels as inputs or outputs in groups of 8. We recommend that you configure using **CONF_DIO**(1100b) which specifies: Channels 0…15 as inputs and channels 16…31 as outputs.

If you need the value of one of the channels DIO16…DIO31, then read out the corresponding bit from the input register of these channels. These channels must be configured as inputs first. Follow these steps (see 2nd example DIGIN_CONN2):

- Read out the register with **PEEK**.
- Clear all bits except the one belonging to the channel (**AND**-masking).

You will find the register number in the table in the annex, chapter A.2.

See also

Conf_DIO, Digin, Digin_Word, Digout_Word, Peek, And

Valid for

Gold

Example

```
DIM Data_1[10000] AS LONG AS FIFO

EVENT:
  'Is digital input 0 set?
  IF (DIGIN(0) = 1) THEN
    Data_1 = ADC(1)      'Measurement data acquisition
  ENDIF
```

A function returning the value of one of the channels DIO16...DIO31 could be as follows:

```
FUNCTION DIGIN_CONN2(bitno) AS LONG
  DIGIN_CONN2=SHIFT_RIGHT(PEEK(204001B0h), bitno) AND 1
ENDFUNCTION
```

Digin_Word

DIGIN_WORD returns the values of all digital inputs at the same time.

Syntax

```
ret_val = DIGIN_WORD()
```

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 ... 16	15	14	...	1	0
Input No.	–	DIO15	DIO14	...	DIO01	DIO00

Notes

DIGIN_WORD requires that you have configured the channels DIO00...DIO15 as inputs. If these channels are configured as output channels, no useful value is returned.
With **CONF_DIO** you can configure the digital channels as inputs or outputs in groups of 8. We recommend that you configure them using **CONF_DIO**(1100b) which specifies: Channels 0…15 as inputs, channels 16…31 as outputs.

If you need the values of the channels DIO16…DIO31, read out the input register of these channels (please note: Configure the channels as inputs first); see also 2nd example DIGIN_WORD_CONN2. You will find the register number in the annex, chapter A.2. The bits in this return value are allocated to the channels as follows:

Bit No.	31 … 16	15	…	1	0
Input No.	–	DIO31	…	DIO17	DIO16

See also

Conf_DIO, Digout_Word, Peek

Valid for

Gold

Example

```
DIM Data_1[10000] AS LONG AS FIFO

EVENT:
  'Querying if the inputs 0 and 1 are set
  IF ((DIGIN_WORD() AND 11b) = 11b) THEN
    Data_1 = ADC(1)      'Measurement data acquisition
  ENDIF
```

A function which returns the value of the channels DIO16...DIO31, could be as follows:

```
FUNCTION DIGIN_WORD_CONN2() AS LONG
  DIGIN_WORD_CONN2=PEEK(204001B0h)
ENDFUNCTION
```

Digout_Word

DIGOUT_WORD sets with a bit pattern all digital outputs to defined TTL-levels.

Syntax

```
DIGOUT_WORD(pattern)
```

Parameters

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

Bit-Nr. in pattern	31…16	15	…	1	0
Ausgang Nr.	–	DIO31	…	DIO17	DIO16

Notes

DIGOUT_WORD requires that you have configured the channels DIO16...DIO31 as outputs. Otherwise it has no effect.

With **CONF_DIO** you can configure the digital channels as inputs our outputs in groups of 8. We recommend that you configure using **CONF_DIO**(1100b) which specifies: Channels 0…15 as inputs, channels 16…31 as outputs.

If you want to set the outputs of the channels DIO00…DIO15, write the corresponding bit pattern to the output register of these channels (please note: Configure channels as outputs first); see also 2nd example DIGOUT_WORD_CONN1. You will find the register number in the annex, chapter A.2.

See also

Conf_DIO, Digin_Word, Clear_Digout, Set_Digout, Poke

Valid for

Gold

Example

```
DIM value AS LONG

INIT:
  REM Configure inputs and output (for ADwin-Gold only)
  CONF_DIO(1100b)

EVENT:
  value = ADC(1)          'Measurement data acquisition
  IF (value > 3000) THEN'Is the limit value exceeded?
    DIGOUT_WORD(101b)     'Set outputs 0 and 2,
                          'clear all other outputs
  ENDIF
```

A program setting TTL-levels of channels DIO00 … DIO15, could be as follows:

```
INIT:
  CONF_DIO(1111b)         'configure all channels as outputs

EVENT:
  IF (ADC(1) > 3000) THEN'value limit exceeded?
    DIGOUT_WORD_CONN1(0Fh)'set outputs 0…15
  ENDIF

SUB DIGOUT_WORD_CONN1(value)
  POKE(204001C0h,value)
ENDSUB
```

Set_Digout

SET_DIGOUT sets one of the digital outputs to 1 (TTL-level high).

Syntax

```
SET_DIGOUT(bit_no)
```

Parameters

bit_no — Bit number (0…15) which specifies the output (see table). CONST, LONG

bit_no	0	1	…	5	…	15
Output in *ADwin-Gold*	DIO16	DIO17	…	DIO21	…	DIO31
Output in *ADwin-light-16*	0	1	…	5	–	–

Notes

SET_DIGOUT accepts only a constant as parameter bit_no.

SET_DIGOUT fits best for the setting of few bits. If several bits are to be set (e.g. in a loop), the usage of the instruction **DIGOUT_WORD** is definitely quicker. Please remember this for time-critical applications in particular.

If you want to set the output using a variable, use the instruction **DIGOUT_WORD**.

SET_DIGOUT requires that you have previously configured the corresponding channel as an output. Otherwise it performs no action.

With **CONF_DIO** you can configure the digital channels as inputs or outputs in groups of 8. We recommend that you configure them using **CONF_DIO**(1100b) which specifies: Channels 0…15 as inputs, channels 16…31 as outputs.

SET_DIGOUT sets one bit in the output register of the channels DIO16…DIO31. If you have set the corresponding channel as output it will generate a TTL-level high.

If you want to set one of the channels 0...15 to 1, set the corresponding bit in the output register of the channels DIO0…DIO15 using the **POKE** command (note: Configure the channel as output first). Follow these steps (see 2nd example SET_DIGOUT_CONN1):

- Read out the register with **PEEK**.
- Set the bit belonging to the channel (**OR**-masking).
- Write the value with **POKE** into the register.

You will find the register number in the annex, chapter A.2.

See also

Clear_Digout, Conf_DIO, Digout_Word, Peek, Poke, And

Valid for

Gold

Example

```
DIM val AS LONG

INIT:
  'Configure digital inputs/output (ADwin-Gold only)
  CONF_DIO(1100b)

EVENT:
  val = ADC(1)          'Measurement data acquisition
  IF (val > 3000) THEN
    SET_DIGOUT(0)       'Set digital output DIO16 to 1
  ENDIF
```

A subroutine which sets a single bit of the DIO-lines 0...15 to 1 could be as follows:

```
SUB SET_DIGOUT_CONN1(bitno)
  POKE(204001C0h, PEEK(204001C0h) OR SHIFT_LEFT(1,bitno) )
ENDSUB
```

12.3 Counter

This section describes the following instructions:

Cnt_Clear

CNT_CLEAR sets one or more counters to zero, according to the bit pattern in pattern.

Syntax

```
#INCLUDE ADWGCNT.Inc
CNT_CLEAR(pattern)
```

Parameters

pattern — Bit pattern. LONG
Bit = 0: no influence.
Bit = 1: set counter to zero.

Bit no.	31…4	3	2	1	0
Counter no.	–	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.

See also

Cnt_Enable, Cnt_GetStatus, Cnt_InputMode, Cnt_Latch, Cnt_Mode, Cnt_Read, Cnt_ReadLatch, Cnt_ReadFLatch, Cnt_ResetStatus, Cnt_Set, Cnt_SE_Diff

Valid for

Gold-CO1

Example

```
#INCLUDE ADWGCNT.Inc
DIM old_1, new_1 AS LONG'Dimension
DIM old_2, new_2 AS LONG'the variables

INIT:
  old_1 = 0             'Initialize
  old_2 = 0             'the variables
  CNT_SE_DIFF(11b)      'All counter inputs differential
  CNT_MODE(0)           'All counters on external clock input
  CNT_SET(11b)          'counters 1+2 with clock (CLK) and
                        'direction (DIR) input
  CNT_INPUTMODE(0)      'Determine functionality CLR/LATCH:
                        'All as CLR input
  CNT_CLEAR(11b)        'Reset counters 1+2 to 0
  CNT_ENABLE(11b)       'Start counters 1+2

EVENT:
  CNT_LATCH(11b)        'Latch counters 1+2 simultaneously
  new_1 = CNT_READLATCH(1)'read out Latch A counter 1 and...
  new_2 = CNT_READLATCH(2)'Latch A counter 2.
  Par_1 = new_1 - old_1'Calculate the difference (f = impulses / time)
  Par_2 = new_2 - old_2'   -"-
  old_1 = new_1         'Save new counter values
  old_2 = new_2         '  -"-
```

Cnt_Enable

CNT_ENABLE disables or enables the counters set by pattern, to count incoming impulses.

Syntax

```
#INCLUDE ADWGCNT.Inc

CNT_ENABLE(pattern)
```

Parameters

pattern	Bit pattern. Bit = 0: stop counter. Bit = 1: enable counter.	LONG

Bit no.	31…4	3	2	1	0
Counter no.	–	4	3	2	1

See also

Cnt_Clear, Cnt_GetStatus, Cnt_InputMode, Cnt_Latch, Cnt_Mode, Cnt_Read, Cnt_ReadLatch, Cnt_ReadFLatch, Cnt_ResetStatus, Cnt_Set, Cnt_SE_Diff

Valid for

Gold-CO1

Example

```
#INCLUDE ADWGCNT.Inc
DIM old_1, new_1 AS LONG ' Dimension
DIM old_2, new_2 AS LONG ' the variables

INIT:
  old_1 = 0              'Initialize
  old_2 = 0              ' the variables
  CNT_SE_DIFF(11b)       'All counter inputs differential
  CNT_MODE(0)            'All counters on external clock input
  CNT_SET(11b)           'Counters 1+2 with clock (CLK) and
                         'direction (DIR) inputs
  CNT_INPUTMODE(0)       'Determine functionality: At all
                         'counters as CLR-input
  CNT_CLEAR(11b)         'Reset counters 1+2 to 0
  CNT_ENABLE(11b)        'Start counters 1+2

EVENT:
  CNT_LATCH(11b)         'Latch counters 1+2 simultaneously
  new_1 = CNT_READLATCH(1)'read out Latch A counter 1 and...
  new_2 = CNT_READLATCH(2)'Latch A counter 2.
  Par_1 = new_1 - old_1 'Calculate the difference (f = impulses / time)
  Par_2 = new_2 - old_2 '-"-
  old_1 = new_1          'Save new counter values as old
  old_2 = new_2          '-"-
```

Cnt_GetStatus

CNT_GETSTATUS reads out and returns the counter status register.

Syntax

```
#INCLUDE ADWGCNT.Inc
ret_val = CNT_GETSTATUS()
```

Parameters

ret_val — Contents of the status register: In case of error, refer to the table for the meaning of the individual bits. LONG

Bit Nr.	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
Signal	-	-	-	-	-	-	-	-	N4	N3	N2	N1	-	-	-	-
Bit Nr.	**31**	**30**	**29**	**28**	**27**	**26**	**25**	**24**	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**
Signal	L4	C4	L3	C3	L2	C2	L1	C1	B4	A4	B3	A3	B2	A2	B1	A1

- :don't care (signal status is not defined, mask out with FF FF 00 F0h)
Ax:Signal A (signal is not changing states)
Bx: Signal B (signal is not changing states)
Cx:Correlation error (signals A and B are identical, they are not phase-shifted by approx. 90°)
Lx: Line error (cable not connected or the line is broken)
Nx:CLR-/LATCH-input (signal is not changing state)
x:Counter number (1…4)

Notes

A line error (Lx) can only be detected at differential inputs! For TTL-inputs these bits are always 0.

The status register is not reset by reading it; use **CNT_RESETSTATUS** instead.

See also

Cnt_Clear, Cnt_Enable, Cnt_InputMode, Cnt_Latch, Cnt_Mode, Cnt_Read, Cnt_ReadLatch, Cnt_ReadFLatch, Cnt_ResetStatus, Cnt_Set, Cnt_SE_Diff

Valid for

Gold-CO1

Example

- / -

Cnt_InputMode

CNT_INPUTMODE sets the function of the CLR/LATCH input of one or more counters.

Syntax

```
#INCLUDE ADWGCNT.Inc
CNT_INPUTMODE(pattern)
```

Parameters

pattern — Bit pattern. LONG
Bit = 0: Set CLR-mode.
Bit = 1: Set LATCH-mode.

Bit no.	31…4	3	2	1	0
Counter no.	–	4	3	2	1

Notes

Use this instruction only when the counter is not enabled.

See also

Cnt_Clear, Cnt_Enable, Cnt_GetStatus, Cnt_Latch, Cnt_Mode, Cnt_Read, Cnt_ReadLatch, Cnt_ReadFLatch, Cnt_ResetStatus, Cnt_Set, Cnt_SE_Diff

Valid for

Gold-CO1

Example

```
#INCLUDE ADWGCNT.Inc
DIM old_1, new_1 AS LONG'Dimension...
DIM old_2, new_2 AS LONG'variables

INIT:
  old_1 = 0              'Initialize...
  old_2 = 0              'variables
  CNT_SE_DIFF(11b)       'All counter inputs differential
  CNT_MODE(0)            'All counters on external clock input
  CNT_SET(11b)           'Counters 1+2 with clock (CLK) and
                         'direction (DIR) input
  CNT_INPUTMODE(0)       'Determine functionality CLR/LATCH: As
                         'CLR-input at all counters
  CNT_CLEAR(11b)         'Reset counters 1+2 to 0
  CNT_ENABLE(11b)        'Start counters 1+2

EVENT:
  CNT_LATCH(11b)         'Latch counters 1+2 simultaneously
  new_1 = CNT_READLATCH(1)'Read out latch A counter 1 and...
  new_2 = CNT_READLATCH(2)'latch A counter 2.
  Par_1 = new_1 - old_1 'Calculate the difference (f = impulses / time)
  Par_2 = new_2 - old_2 ' -"-
  old_1 = new_1          'Save new counter values as old
  old_2 = new_2          ' -"-
```

Cnt_Latch

CNT_LATCH transfers the current counter values of one or more counters into the relevant Latch A, depending on the bit pattern in pattern.

Syntax

```
#INCLUDE ADWGCNT.Inc
CNT_LATCH(pattern)
```

Parameters

pattern — Bit pattern. LONG
Bit = 0: no function.
Bit = 1: transfer counter values into Latch A .

Bit no.	31…4	3	2	1	0
Counter no.	–	4	3	2	1

Notes

After the instruction has been executed the bit pattern is automatically reset to 0 (zero).

Latch A is read out into a variable with **CNT_READLATCH** command.

Valid for

Gold-CO1

See also

Cnt_Clear, Cnt_Enable, Cnt_GetStatus, Cnt_InputMode, Cnt_Mode, Cnt_Read, Cnt_ReadLatch, Cnt_ReadFLatch, Cnt_ResetStatus, Cnt_Set, Cnt_SE_Diff

Example

```
#INCLUDE ADWGCNT.Inc
DIM old_1, new_1 AS LONG'Dimension...
DIM old_2, new_2 AS LONG'the variables

INIT:
  old_1 = 0              'Initialize
  old_2 = 0              'the variables
  CNT_SE_DIFF(11b)       'All counter inputs differential
  CNT_MODE(0)            'All counters on external clock input
  CNT_SET(11b)           'Counters 1+2 with clock (CLK) and
                         'direction (DIR) input
  CNT_INPUTMODE(0)       'Determine functionality CLR/LATCH: As
                         'CLR-input at all counters
  CNT_CLEAR(11b)         'Reset counters 1+2 to 0
  CNT_ENABLE(11b)        'Start counters 1+2

EVENT:
  CNT_LATCH(11b)         'Latch counters 1+2 simultaneously and then...
  new_1 = CNT_READLATCH(1)'read out Latch A counter 1 and...
  new_2 = CNT_READLATCH(2)'Latch A counter 2.
  Par_1 = new_1 - old_1 'Calculate the difference (f = impulses / time)
  Par_2 = new_2 - old_2 '-"-
  old_1 = new_1          'Save new counter values as old
  old_2 = new_2          '-"-
```

Cnt_Mode

CNT_MODE defines the operating mode of all counters by selecting which clock input they use according to the bit pattern in pattern.

Syntax

```
#INCLUDE ADWGCNT.Inc
CNT_MODE(pattern)
```

Parameters

pattern — Bit pattern. LONG
Bit = 0: external clock input (CLK/DIR or A/B).
Bit = 1: internal clock input (5 MHz or 20 MHz).

Bit no.	31…4	3	2	1	0
Counter no.	–	4	3	2	1

Notes

CNT_SET determines the mode of the selected clock input.

Please use CNT_MODE only when the counter is disabled.

See also

Cnt_Clear, Cnt_Enable, Cnt_GetStatus, Cnt_InputMode, Cnt_Latch, Cnt_Read, Cnt_ReadLatch, Cnt_ReadFLatch, Cnt_ResetStatus, Cnt_Set, Cnt_SE_Diff

Valid for

Gold-CO1

Example

```
#INCLUDE ADWGCNT.Inc
DIM old_1, new_1 AS LONG'Dimension
DIM old_2, new_2 AS LONG'the variables

INIT:
  old_1 = 0              'Initialize
  old_2 = 0              'the variables
  CNT_SE_DIFF(11b)       'All counter inputs differential
  CNT_MODE(0)            'All counters on external clock input
  CNT_SET(1)             'Counter 1 with 20 MHz
  CNT_INPUTMODE(0)       'Determine the functionality CLR/LATCH
                         ' As CLR-input at all counters
  CNT_CLEAR(11b)         'Reset counters 1+2 to 0
  CNT_ENABLE(11b)        'Start counters 1+2

EVENT:
  CNT_LATCH(11b)         'Latch counters 1+2 simultaneously and then...
  new_1 = CNT_READLATCH(1)'Read out Latch A counter 1 and...
  new_2 = CNT_READLATCH(2)'Latch A counter 2.
  Par_1 = new_1 - old_1 'Calculate the difference (f = impulses / time)
  Par_2 = new_2 - old_2 '-"-
  old_1 = new_1          'Save new counter values as old
  old_2 = new_2          '-"-
```

Cnt_Read

CNT_READ transfers the current counter value into Latch A and returns it as return value.

Syntax

```
#INCLUDE ADWGCNT.Inc
ret_val = CNT_READ(CounterNo)
```

Parameters

CounterNo	Counter number: 1…4.	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_GetStatus, Cnt_InputMode, Cnt_Latch, Cnt_Mode, Cnt_ReadLatch, Cnt_ReadFLatch, Cnt_ResetStatus, Cnt_Set, Cnt_SE_Diff

Valid for

Gold-CO1

Example

```
#INCLUDE ADWGCNT.Inc
DIM old_1, new_1 AS LONG'Dimension...
DIM old_2, new_2 AS LONG'the variables

INIT:
  old_1 = 0              'Initialize...
  old_2 = 0              'the variables
  CNT_SE_DIFF(11b)       'All counter inputs differential
  CNT_MODE(0)            'All counters on external clock input
  CNT_SET(11b)           'Counters 1+2 with clock (CLK) and
                         'direction (DIR) inputs
  CNT_INPUTMODE(0)       'Determine functionality CLR/LATCH: At
                         'all as CLR-input
  CNT_CLEAR(11b)         'Reset counters 1+2 to 0
  CNT_ENABLE(11b)        'Start counters 1+2

EVENT:
  new_1 = CNT_READ(1)    'Latch counter 1 and read out Latch A afterward
  new_2 = CNT_READ(2)    'Latch counter 2 and read out Latch A afterward
  Par_1 = new_1 - old_1 'Calculate the difference (f = impulses / time)
  Par_2 = new_2 - old_2 ' -"-
  old_1 = new_1          'Save new counter values as old
  old_2 = new_2          ' -"-
```

Cnt_ReadLatch

CNT_READLATCH returns the value of a counter previously stored in Latch A.

Syntax

```
#INCLUDE ADWGCNT.Inc
ret_val = CNT_READLATCH(CounterNo)
```

Parameters

CounterNo	Counter number: 1…4.	LONG
ret_val	Contents of Latch A .	LONG

Notes

Use the return value in calculations only with variables of the type **LONG** (e.g. differences or count direction).

The point of time when the current counter value is latched depends on the **CNT_MODE** settings:

- External clock input (**CNT_MODE** bit = 0): Only the instruction **CNT_LATCH** latches the counter.
- Internal clock input (**CNT_MODE** bit = 1): Any edge of the external measurement signal latches the counter.
  At a positive edge of the input signal the counter values are latched into Latch A, whereas at a negative edge of the input signal the counter values are latched into Latch B.

See also

Cnt_Clear, Cnt_Enable, Cnt_GetStatus, Cnt_InputMode, Cnt_Latch, Cnt_Mode, Cnt_Read, Cnt_ReadFLatch, Cnt_ResetStatus, Cnt_Set, Cnt_SE_Diff

Valid for

Gold-CO1

Example

```
#INCLUDE ADWGCNT.Inc

DIM rise, rise_old, fall, fall_old AS LONG
#DEFINE high Par_1
#DEFINE low Par_2
#DEFINE T Par_9
#DEFINE f Par_10

INIT:
  rise_old = 0          'Initialize the variables
  fall_old = 0
  CNT_SE_DIFF(11b)      'All counter inputs differential
  CNT_MODE(11b)         'Counters 1+2 on internal clock input
  CNT_SET(0)            'All counters with 20 MHz internal
                        'reference clock
  CNT_INPUTMODE(11b)    'Determine functionality CLR/LATCH: At
                        'counters 1+2 as LATCH input
  CNT_CLEAR(11b)        'Reset counters 1+2 to 0
  CNT_ENABLE(1)         'Start couner 1

EVENT:
  rise = CNT_READLATCH(1)'Read out Latch A counter 1
  fall = CNT_READFLATCH(1)'Read out Latch B counter 1
  IF (rise <> rise_old) THEN'Is a rising edge detected?
    T = rise - rise_old 'Period duration in nanoseconds
    f = 1E9 / T         'Frequency in Hertz
    IF (fall <> fall_old) THEN'Is a falling edge detected?
      high = (fall - rise) * 25'Impulse duration in nanoseconds
      low = (rise - fall_old) * 25'Pause duration in nanoseconds
    ELSE                'No falling edge is detected
      high = (fall - rise_old) * 25'Impulse duration in nanoseconds
      low = (rise - fall) * 25'Pause duration in nanoseconds
    ENDIF
  ENDIF
  rise_old = rise       'Save contents of the latch
  fall_old = fall       'Save contents of the latch
```

Cnt_ReadFLatch

CNT_READFLATCH returns the value of a counter previously stored in Latch B.

Syntax

```
#INCLUDE ADWGCNT.Inc
ret_val = CNT_READFLATCH(CounterNo)
```

Parameters

CounterNo	Counter number: 1…4.	LONG
ret_val	Contents of Latch B.	LONG

Notes

Use the return value in calculations only with variables of the type **LONG** (e.g. differences or count direction).

The point of time when the current counter value is latched depends on the **CNT_MODE** settings:

- External clock input (**CNT_MODE** bit = 0): Only the instruction **CNT_LATCH** latches the counter.
- Internal clock input (**CNT_MODE** bit = 1): Any edge of the external measurement signal latches the counter.
  At a positive edge of the input signal the counter values are latched into Latch A, whereas at a negative edge of the input signal the counter values are latched into Latch B.

See also

Cnt_Clear, Cnt_Enable, Cnt_GetStatus, Cnt_InputMode, Cnt_Latch, Cnt_Mode, Cnt_Read, Cnt_ReadLatch, Cnt_ResetStatus, Cnt_Set, Cnt_SE_Diff

Valid for

Gold-CO1

Example

```
#INCLUDE ADWGCNT.Inc
DIM rise, rise_old, fall, fall_old AS LONG
#DEFINE high Par_1
#DEFINE low Par_2
#DEFINE T Par_9
#DEFINE f Par_10

INIT:
  rise_old = 0          'Initialize...
  fall_old = 0          ' the variables
  CNT_SE_DIFF(11b)      'All counter inputs differential
  CNT_MODE(11b)         'Counters 1+2 on internal clock input
  CNT_SET(0)            'All counters with 20 MHz internal
                        'clock reference
  CNT_INPUTMODE(11b)    'Determine functionality CLR/LATCH: At
                        'counters 1+2 as LATCH inputs
  CNT_CLEAR(11b)        'Reset counters 1+2 to 0
  CNT_ENABLE(1)         'Start counter 1

EVENT:
  rise = CNT_READLATCH(1)'Read out Latch A counter 1
  fall = CNT_READFLATCH(1)'Read out Latch B counter 1
  IF (rise <> rise_old) THEN'Is a rising edge detected?
    T = rise - rise_old 'Period duration in nanoseconds
    f = 1E9 / T         'Frequency in Hertz
    IF (fall <> fall_old) THEN'Is a falling edge detected?
     high = (fall - rise) * 25'Impulse duration in nanoseconds
     low = (rise - fall_old) * 25'Pause duration in nanoseconds
    ELSE                'No falling edge detected
     high = (fall - rise_old) * 25'Impulse duration in nanoseconds
     low = (rise - fall) * 25'Pause duration in nanoseconds
    ENDIF
  ENDIF
  rise_old = rise       'Save contents of the latch
  fall_old = fall       'Save contents of the latch
```

Cnt_ResetStatus

CNT_RESETSTATUS clears the status register of all four 32 bit-counters.

Syntax

```
#INCLUDE ADWGCNT.Inc

CNT_RESETSTATUS()
```

Parameters

- / -

Notes

The status register is read out with the instruction CNT_GETSTATUS.

See also

Cnt_Clear, Cnt_Enable, Cnt_GetStatus, Cnt_InputMode, Cnt_Latch, Cnt_Mode, Cnt_Read, Cnt_ReadLatch, Cnt_ReadFLatch, Cnt_Set, Cnt_SE_Diff

Valid for

Gold-CO1

Example

```
#INCLUDE ADWGCNT.Inc

DIM error AS LONG

DIM old_1, new_1 AS LONG 'Dimensioning…
DIM old_2, new_2 AS LONG ' variables

INIT:
  CNT_ENABLE(0)          'Stop all counters
  CNT_CLEAR(1111b)       'Clear all counters
  CNT_SE_DIFF(11b)       'Set all counters to diff. inputs
  CNT_MODE(0)            'Set external event input
  CNT_SET(0)             'Set mode 4 edge evaluation
  CNT_INPUTMODE(0)       'Enable CLR counter input
  CNT_ENABLE(1111b)      'Start all counters
  old_1 = 0              'Initialize…
  old_2 = 0              ' variables
  error = 0              'Initialize error flag

EVENT:
  Par_1 = CNT_READ(1)    'Read counter 1
  Par_2 = CNT_GETSTATUS(1) AND 0FFFF00F0h 'Read out and mask
                         'status register counter 1
  IF (Par_2 AND 2000000h = 2000000h) THEN'Line or cable error
                         'counter 1?
    INC Par_3            'Number of line or cable errors until now...
    error = 1            'Set error flag
  ENDIF
  IF (Par_2 AND 1000000h = 1000000h) THEN'Correlation error cnt 1?
    INC Par_4            'Number of correlation errors until now...
    error = 1            'Set error flag
  ENDIF
  CNT_RESETSTATUS()      'Clear bits of line and correlation errors
  Par_5 = SHIFT_RIGHT(Par_2 AND 10h,4) 'status of CLR-input
  Par_6 = SHIFT_RIGHT(Par_2 AND 10000h,16) 'status of input A
  Par_7 = SHIFT_RIGHT(Par_2 AND 20000h,17) 'status of input B
```

Cnt_Set

CNT_SET defines the operating mode for all counters (depending on CNT_MODE) according to the given bit pattern.

Syntax

```
#INCLUDE ADWGCNT.Inc
CNT_SET(pattern)
```

Parameters

pattern — Bit pattern, for the meaning of the bits see table below. LONG

Bit value in pattern	External clock input Bit = 0 in CNT_MODE	Internal clock input Bit = 1 in CNT_MODE
Bit = 0	4-edge evaluation	Reference clock 20 MHz
Bit = 1	Clock and direction input	Reference clock 5 MHz

Bit no.	31…4	3	2	1	0
Counter no.	–	4	3	2	1

Notes

Please use this instruction only when the counter is disabled.

See also

Cnt_Clear, Cnt_Enable, Cnt_GetStatus, Cnt_InputMode, Cnt_Latch, Cnt_Mode, Cnt_Read, Cnt_ReadLatch, Cnt_ReadFLatch, Cnt_ResetStatus, Cnt_SE_Diff

Valid for

Gold-CO1

Example

```
#INCLUDE ADWGCNT.Inc

INIT:
  CNT_SE_DIFF(11b)      'All counter inputs differential
  CNT_MODE(0)           'All counters on external clock input
  CNT_SET(1100b)        'Counters 3+4 with clock/direction evaluation,
                        'Counters 1+2 with 4 edge evaluation
  CNT_CLEAR(1100b)      'Set counters 3+4 to 0
  CNT_ENABLE(1100b)     'Enable counters 3+4, disable counters 1+2
```

Cnt_SE_Diff

CNT_SE_DIFF sets counter inputs to the input mode single-ended or differential as pairs.

Syntax

```
#INCLUDE ADWGCNT.Inc
CNT_SE_DIFF(CounterNo)
```

Parameter

CounterNo — Bit pattern to choose the counter pairs (see table) and set the input mode: LONG
Bit = 0: Run inputs single-ended.
Bit = 1: Run inputs differential.

Bit no. in pattern	31 … 2	1	0
Inputs of counters no.	–	3 + 4	1 + 2

Notes

After start-up, the operating mode of the counter inputs is undefined; all of the counter inputs have to be set to the desired operating mode.

See also

Cnt_Clear, Cnt_Enable, Cnt_GetStatus, Cnt_InputMode, Cnt_Latch, Cnt_Mode, Cnt_Read, Cnt_ReadLatch, Cnt_ReadFLatch, Cnt_ResetStatus, Cnt_Set

Valid for

Gold-CO1

Example

```
#INCLUDE ADWGCNT.Inc
DIM error AS LONG       'Dimensioning…
DIM old_1, new_1 AS LONG' variables
DIM old_2, new_2 AS LONG

INIT:
  CNT_ENABLE(0)         'Stop all counters
  CNT_CLEAR(1111b)      'Clear all counters
  CNT_SE_DIFF(11b)      'Set all counters to diff. inputs
  CNT_MODE(0)           'Set external event input
  CNT_SET(0)            'Set mode 4 edge evaluation
  CNT_INPUTMODE(0)      'Enable CLR counter input
  CNT_ENABLE(1111b)     'Start all counters
  old_1 = 0             'Initialize…
  old_2 = 0             ' variables
  error = 0             'Initialize error flag

EVENT:
  Par_1 = CNT_READ(1)   'Read out counter 1
  Par_2 = CNT_GETSTATUS(1) AND 0FFFF00F0h'Read out and mask
                        'status register counter 1
  IF (Par_2 AND 2000000h = 2000000h) THEN'Line or cable error cnt 1?
    INC Par_3           'Number of line or cable errors until now...
    error = 1           'Set error flag
  ENDIF
  IF (Par_2 AND 1000000h = 1000000h) THEN'Correlation error cnt 1?
    INC Par_4           'Number of correlation errors until now...
    error = 1           'Set error flag
  ENDIF
  CNT_RESETSTATUS()     'Clear bits of line and correlation errors
  Par_5 = SHIFT_RIGHT(Par_2 AND 10h,4) 'status of CLR-input
  Par_6 = SHIFT_RIGHT(Par_2 AND 10000h,16) 'status of input A
  Par_7 = SHIFT_RIGHT(Par_2 AND 20000h,17) 'status of input B
```

12.4 CAN interface

This section describes the following instructions:

CAN_Msg

CAN_Msg[] is a one-dimensional array of 9 elements, where CAN message objects are stored.

Syntax

```
#INCLUDE ADWGCAN.Inc
CAN_Msg[n] = value
```

or

```
value = CAN_Msg[n]
```

Parameters

n	Element number in the field CAN_Msg (1…9).	LONG
value	Value (8 bit), which is to be written into or read from the message object.	LONG

Notes

The elements of the array CAN_Msg[] have the following functions:

Element no.	1…8	9
Contents	Message objects = data bytes	Number (0…8) of allocated data bytes

Enter the values to be transferred into the field CAN_Msg[], *before* transferring them with **TRANSMIT**.

See also

Init_CAN, Read_Msg, Read_Msg_Con, Transmit

Valid for

Gold-CAN

Example

```
#INCLUDE ADWGCAN.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 Read_Msg)

#DEFINE pi 3.14159265
DIM i AS LONG

INIT:
  INIT_CAN(1)           'Initialize CAN controller 1

  REM Enable message object 6 of controller 1 with the
  REM for sending with the identifier 40 (11 bit)
  EN_TRANSMIT(1, 6,40,0)

  REM Create bit pattern of Pi with data type Long
  Par_1 = CAST_FLOATTOLONG(pi)

  REM divide bit pattern (32 Bit) into 4 bytes
  CAN_Msg[4] = Par_1 AND 0FFh 'assign LSB
  FOR i = 1 TO 3
    CAN_Msg[4-i] = SHIFT_RIGHT(Par_1,8*i) AND 0FFh
  NEXT i
  CAN_Msg[9] = 4        'message length in bytes

EVENT:
  TRANSMIT(1,6)         'Send message object 6
```

En_CAN_Interrupt

EN_CAN_INTERRUPT configures a specified message object of a CAN interface to generate an external event when a message arrives.

Syntax

```
#INCLUDE ADWGCAN.Inc
EN_CAN_INTERRUPT(can_no, msg_no)
```

Parameters

can_no	Number (1, 2) of CAN interface.	LONG
msg_no	Number (1…15) of message object.	LONG

Notes

- / -

See also

CAN_Msg, En_Receive, Get_CAN_Reg, Set_CAN_Reg

Valid for

Gold-CAN

Example

```
#INCLUDE ADWGCAN.Inc

INIT:
  INIT_CAN(1)           'Initialization of CAN controller 1
  EN_RECEIVE(1,1,200,0) 'Initialize the message object 1 of
                        'controller 1 to receive CAN messages
                        'with the identifier 200
  EN_CAN_INTERRUPT(1,1) 'Enables the triggering of interrupts
                        '(ext. EVENT) when receiving the
                        'message object 1
```

En_Receive

EN_RECEIVE enables a specified message object of a CAN inteface to receive messages.

Syntax

```
#INCLUDE ADWGCAN.Inc
EN_RECEIVE(can_no, msg_no, id, id_extend)
```

Parameters

can_no	Number (1, 2) of CAN interface.	LONG
msg_no	Number (1…15) of message object.	LONG
id	Identifier (0…2^{11} or 0…2^{29}) of the messages, which can be received in this message object.	LONG
id_extend	Length of the identifer: 0: 11 bits. 1: 29 bits.	LONG

See also

CAN_Msg, En_Transmit, Read_Msg, Read_Msg_Con

Notes

A message object can only receive messages from the CAN bus when you have previously enabled it to receive with **EN_RECEIVE.**

The message object only receives messages with the identifier you have specified.

Valid for

Gold-CAN

Example

```
#INCLUDE ADWGCAN.Inc

INIT:
  INIT_CAN(1)           'Initialization of CAN controller 1
  EN_RECEIVE(1,1,200,0) 'Initialize the message object 1 of
                        'controller 1 to receive CAN messages
                        'with the identifier 200
```

En_Transmit

EN_TRANSMIT enables a specified message object of a CAN inteface to send messages.

Syntax

```
#INCLUDE ADWGCAN.Inc

EN_TRANSMIT(can_no, msg_no, id, id_extend)
```

Parameters

can_no	Number (1, 2) of CAN interface.	LONG
msg_no	Number (1…14) of message object.	LONG
id	Identifier which is sent with the messages of this message object.	LONG
id_extend	Length of the identifier: 0: 11 bits. 1: 29 bits.	LONG

Notes

A message object can only send messages to the CAN bus when you have it previously enabled to send with **EN_TRANSMIT.**

See also

CAN_Msg, En_Receive, Transmit

Valid for

Gold-CAN

Example

```
#INCLUDE ADWGCAN.Inc

INIT:
  INIT_CAN(1)           'Initialization of CAN controller 1
  REM Initialize message objects 6 of controller 1:
  REM Object 1 to receive with identifier 200
  REM Object 1 to send with identifier 40
  EN_RECEIVE(1,1,200,0)
  EN_TRANSMIT(1,6,40,0)
```

Get_CAN_Reg

GET_CAN_REG reads the value of a specified register in one of the CAN controllers.

Syntax

```
#INCLUDE ADWGCAN.Inc
ret_val = GET_CAN_REG(can_no, regno)
```

Parameters

can_no	Number (1, 2) of CAN interface.	LONG
regno	Register number in the CAN controller (0…255).	LONG
ret_val	Contents of the register (transferred to the lower 8 bits).	LONG

Notes

You will find the register list of the CAN controller in the Intel® AN82527 data sheet.

See also

Init_CAN, Set_CAN_Baudrate, Set_CAN_Reg

Valid for

Gold-CAN

Example

```
#INCLUDE ADWGCAN.Inc
INIT:
  INIT_CAN(1)           'Initialization of CAN controller 1
  Par_1 = GET_CAN_REG(1,0)'Read out the control register
```

Init_CAN

INIT_CAN initializes one of the CAN controllers.

Syntax

```
#INCLUDE ADWGCAN.Inc
INIT_CAN(can_no)
```

Parameters

can_no	Number (1, 2) of CAN interface.	LONG

Notes

The instruction carries out the following steps:

- Reset (hardware reset of the CAN controller)
- All filters are set to "must match".
- Clockout register is set to 0 (= the external frequency is not divided).
- The register "Bus Configuration" is set to 0.
- The transfer rate for the CAN bus is set to 1 MBit/s.
- All message objects are disabled.

You have to execute this instruction before you access the CAN controller with other instructions. We recommend you place this instruction in the process section **LOWINIT:** or **INIT:**

See also

CAN_Msg, En_CAN_Interrupt, En_Receive, En_Transmit, Get_CAN_Reg, Set_CAN_Baudrate, Set_CAN_Reg

Valid for

Gold-CAN

Example

```
#INCLUDE ADWGCAN.Inc

INIT:
  INIT_CAN(1)           'Initialize CAN controller 1
```

Read_Msg

READ_MSG checks if new messages have been received in a specified message object of CAN interface.

If so, the message is saved in CAN_Msg and the identifier of the message is returned.

Syntax

```
#INCLUDE ADWGCAN.Inc
ret_val = READ_MSG(can_no, msg_no)
```

Parameters

can_no	Number (1, 2) of CAN interface.	LONG
msg_no	Number (1…15) of message object.	LONG
ret_val	-1: No new message. >0: New message received; value = identifier of the message.	LONG

Notes

To receive a message you have to follow the correct order:

- Once: Enable the message object with EN_RECEIVE for receiving.
- As often as needed: Check for a received message and save to CAN_Msg with READ_MSG.

You can read a received message only once.

See also

CAN_Msg, En_CAN_Interrupt, En_Receive, En_Transmit, Read_Msg

Valid for

Gold-CAN

Example

```
#INCLUDE ADWGCAN.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.
DIM n AS LONG

INIT:
Par_1 = 0
  INIT_CAN(1)           'Initialize CAN controller 1
  EN_RECEIVE(1,1,40,0)  'Initialize the message object 1 of
                        'controller 1 to receive CAN messages
                        'with identifier 40

EVENT:
  REM If the message is changed, read out the received data
  REM from object 1 and save the identifier to parameter 9.
  REM The data bytes are in the array CAN_MSG[].
  Par_9 = READ_MSG(1,1)

  IF (Par_9 = 40) THEN
    REM New message for message object with the identifier 40
    REM has arrived
    Par_1 = CAN_Msg[1]   'Read out high-byte
    FOR n = 2 TO 4       'Combine with remaining 3 bytes to
     Par_1 = SHIFT_LEFT(Par_1,8) + CAN_Msg[n]'a 32-bit value
    NEXT n
    REM Convert the bit pattern in PAR_1 to data type FLOAT and
    REM assign to the variable FPAR_1.
    FPar_1 = CAST_LONGTOFLOAT(Par_1)
  ENDIF
```

Sending a float value see example at Transmit.

Read_Msg_Con

READ_MSG_CON checks if a complete new message has been received in a specified message object.

If so, the message is saved in CAN_Msg and the identifier of the message is returned.

Syntax

```
#INCLUDE ADWGCAN.Inc
ret_val = READ_MSG_CON(can_no, msg_no)
```

Parameters

can_no	Number (1, 2) of CAN interface.	LONG
msg_no	Number (1…15) of message object.	LONG
ret_val	-1: no new message arrived. >0: new message; ret_val = message identifier.	LONG

Notes

In contrary to READ_MSG, READ_MSG_CON makes sure the message is consistent: If a new message arrives while reading an old message, there is no mixture of old and new message.

To receive a message, follow these steps:

- Enable the message object for receive with EN_RECEIVE.
- Check for a new message, and if, store the message in CAN_Msg with READ_MSG.

You can read a received message only once.

See also

CAN_Msg, En_CAN_Interrupt, En_Receive, En_Transmit, Read_Msg

Valid for

Gold-CAN

Example

```
#INCLUDE ADWGCAN.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.
DIM n AS LONG

INIT:
Par_1 = 0
  INIT_CAN(1)           'Initialize CAN controller 1
  EN_RECEIVE(1,1,40,0)  'Initialize the message object 1
                        'to receive CAN messages with
                        'identifier 40

EVENT:
  REM If the message is changed, read out the received data
  REM from object 1 and transfer the identifier to parameter 9.
  REM The data bytes are in the array CAN_MSG[].
  Par_9 = READ_MSG_CON(1,1)

  IF (Par_9 = 40) THEN
    REM New message for message object with the identifier 40
    REM has arrived
    Par_1 = CAN_Msg[1]   'Read out high-byte
    FOR n = 2 TO 4       'Combine with remaining 3 bytes to
     Par_1 = SHIFT_LEFT(Par_1,8) + CAN_Msg[n]'a 32-bit value
    NEXT n
    REM Convert the bit pattern in PAR_1 to data type FLOAT and
    REM assign to the variable FPAR_1.
    FPar_1 = CAST_LONGTOFLOAT(Par_1)
  ENDIF
```

Sending a float value see example at Transmit.

Set_CAN_Baudrate

SET_CAN_BAUDRATE sets the Baud rate of the specified CAN controller.

Syntax

```
#INCLUDE ADWGCAN.Inc
ret_val = SET_CAN_BAUDRATE(can_no, rate)
```

Parameters

can_no	Number (1, 2) of CAN interface.	LONG
rate	Baud rate in bits/second.	FLOAT
ret_val	0: Baud rate is set. 1: Baud rate invalid.	LONG

Notes

The available baud rates (bus frequencies) are given in the table "Baudrates for the CAN bus" (Annex, page A-8). Please use the table's notation exactly, i.e. non-integer baud rates with 4 decimal places; values with different notation will be rejected as not allowed.

SET_CAN_BAUDRATE executes the following actions:

- Checks if the transferred Baud rate is allowed. If not then set the return value to 1 and stop processing.
- Set the registers of the CAN controller for the Baud rate.
- Set sampling mode to 0: One sample per bit.
- Select the settings in such a way that the sample point is always between 60% and 72% of the total bit length.
- Set the jump width for synchroniziation to 1.

In special cases it may be of interest to set a baud rate in a different way than the instruction works. The hardware manual gives an explanation how to do this.

The instruction should be called in the program sections **LOWINIT:** or **INIT:**, after the instruction **INIT_CAN**, because otherwise the set Baud rate will be overwritten by the default setting (1 MBit/s).

See also

Get_CAN_Reg, Init_CAN, Set_CAN_Reg

Valid for

Gold-CAN

Example

```
#INCLUDE ADWGCAN.Inc

INIT:
  INIT_CAN(1)            'Initialize CAN controller 1
  SET_CAN_BAUDRATE(1,125000)'Set the Baud rate to 125 kBit/s
```

Set_CAN_Reg

SET_CAN_REG writes a value into a specified register of one of the CAN controllers.

Syntax

```
#INCLUDE ADWGCAN.Inc
SET_CAN_REG(can_no, regno, value)
```

Parameters

can_no	Number (1, 2) of CAN interface.	LONG
regno	Register number in the CAN controller (0…255).	LONG
value	Value (8 bits), which is written into the register.	LONG

Notes

The register list of the CAN controller can be found in the Intel® AN82527 datasheet.

See also

Get_CAN_Reg, Init_CAN, Set_CAN_Baudrate

Valid for

Gold-CAN

Example

```
#INCLUDE ADWGCAN.Inc

INIT:
  INIT_CAN(1)           'Initialization of CAN controller 1
  SET_CAN_REG(1,0,1)    'Set control register to the value 1
```

Transmit

TRANSMIT sends the message in CAN_Msg via the specified message object of a CAN controller.

Syntax

```
#INCLUDE ADWGCAN.Inc
TRANSMIT(can_no, msg_no)
```

Parameters

can_no	Number (1, 2) of CAN interface.	LONG
msg_no	Number (1…14) of message object.	LONG

Notes

To send a message you have to follow the correct order:

- Enable the message object with **EN_TRANSMIT** for sending (only once).
- Enter the message into the array CAN_Msg: Data bytes and number of data bytes.
- Send the message with **TRANSMIT**.

CAN interface will send the message as soon as the message object has received access rights to the CAN bus.

See also

CAN_Msg, En_Transmit, Init_CAN, Set_CAN_Baudrate

Valid for

Gold-CAN

Example

```
#INCLUDE ADWGCAN.Inc
REM Sends a 32 bit FLOAT value (here: Pi) as sequence of
REM 4 bytes in a message object
#DEFINE pi 3.14159265
DIM i AS LONG

INIT:
  INIT_CAN(2)           'Initialize CAN-Controller 2

  REM Initialize message object 6 of controller 2
  REM for sending of CAN messages with the identifier 40
  EN_TRANSMIT(2, 6,40,0)

  REM Create bit pattern of Pi with data type Long
  Par_1 = CAST_FLOATTOLONG(pi)

  REM divide bit pattern (32 Bit) into 4 bytes
  CAN_Msg[4] = Par_1 AND 0FFh 'assign LSB
  FOR i = 1 TO 3
    CAN_Msg[4-i] = SHIFT_RIGHT(Par_1,8*i) AND 0FFh
  NEXT i
  CAN_Msg[9] = 4        'message length in bytes

EVENT:
  TRANSMIT(2,6)         'Sends the message object 6
```

Receiving of a float value see example at Read_Msg.

12.5 RSxxx interface

This section describes the following instructions:

Check_Shift_Reg

CHECK_SHIFT_REG returns, if all data has been sent, which was written into the send-FIFO of the RSxxx interface.

Syntax

```
#INCLUDE ADWGCAN.Inc
ret_val = CHECK_SHIFT_REG(interface)
```

Parameters

interface	Number (1, 2) of RSxxx interface that is to be read.	LONG
ret_val	Sending status: 0: Data has been sent (= no more data in the send-FIFO). 1: Not yet all data sent (= the send-FIFO still contains data).	LONG

Notes

With return value 0 both the send FIFO and the output shift register are empty. With the return value 1 there is at least one bit to be sent.

We recommend to use this instruction only after you have more experience about how the controller operates (data-sheet of the manufacturer Texas Instruments). For more common applications more comfortable instructions are availabe in the include file.

See also

GET_RS, RS_INIT, RS_RESET, WRITE_FIFO

Valid for

Gold-CAN

Example

```
#INCLUDE ADWGCAN.Inc

EVENT:
  …
  Par_1 = CHECK_SHIFT_REG(1)'Check if RSxxx interface 1 still
                            'has data to send
  …
```

Get_RS

GET_RS reads out a specified controller register.

Syntax

```
#INCLUDE ADWGCAN.Inc
ret_val = GET_RS(reg_addr)
```

Parameters

reg_addr	Address of the controller register to read.	LONG
ret_val	Contents of the controller register.	LONG

Notes

We recommend to use this instruction only after you have more experience about how the controller operates (data-sheet of the manufacturer Texas Instruments). For more common applications more comfortable instructions are availabe in the include file.

See also

CHECK_SHIFT_REG, RS_INIT, RS_RESET, SET_RS

Valid for

Gold-CAN

Example

-/-

Read_FIFO

READ_FIFO reads a value from the input FIFO of a specified RSxxx interface.

Syntax

```
#INCLUDE ADWGCAN.Inc
ret_val = READ_FIFO(interface)
```

Parameters

interface	number (1, 2) of the RSxxx interface that is to be read out.	LONG
ret_val	Contents of the input FIFO: -1: FIFO is empty. ≥0: Transferred value.	LONG

Notes

-/-

See also

RS_INIT, RS_RESET, RS485_SEND, WRITE_FIFO

Valid for

Gold-CAN

Example

```
#INCLUDE ADWGCAN.Inc

INIT:
  RS_RESET()
  Rem Initialize RSxxx interface 1: 9600 Baud, without parity,
  Rem 8 data bits, 1 stop bit and hardware handshake.
  RS_INIT(1,9600,0,8,0,1)

EVENT:
  Rem Get a value from the FIFO. If the FIFO is empty, -1 is returned.
  Par_1 = READ_FIFO(1)
```

RS485_Send

RS485_SEND determines the transfer direction for a specified RSxxx interface.

Syntax

```
#INCLUDE ADWGCAN.Inc
RS485_SEND(interface,dir)
```

Parameters

interface	RSxxx interface to be set (1, 2).	LONG
dir	Tranfer direction of the RSxxx interface: 0: Set RSxxx interface to receive. 1: Set RSxxx interface to send. 2: Set RSxxx interface to send and to receive its sent data. 3: Mute RSxxx interface, i.e. the interface works as receiver but doesn't put data into the input FIFO.	LONG

Notes

Setting the transfer direction means:

- Receiver: The RSxxx interface can only read data, even if data are in the output FIFO of the controller for this RSxxx interface.
- Sender: The RSxxx interface transfers data to the bus which are read by other devices.
- Sender/receiver: The RSxxx interface can transfer data to the bus and back at the same time. Thus, the sent data can be checked.

See also

CHECK_SHIFT_REG, GET_RS, RS_INIT, RS_RESET, SET_RS

Valid for

Gold-CAN

Example

-/-

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 ADWGCAN.Inc
RS_INIT(interface,baud,parity,bits,stop,handshake)
```

Parameters

interface	Number of RSxxx interface (1, 2), which is to be initialized.	LONG
baud	Transfer rate in Baud.	LONG
parity	Use of test bits: 0: without parity bit. 1: even parity. 2: odd parity.	LONG
bits	Amount of data bits (5, 6, 7 or 8).	LONG
stop	Amount of stop bits. 0: 1 stop bit. 1: 1½ stop bits at 5 data bits; 2 stop bits at 6, 7 or 8 data bits.	LONG
handshake	Transfer protocol: 0: RS232, No handshake. 1: RS232, Hardware handshake (RTS/CTS). 2: RS232, Software handshake (Xon/Xoff). 3: RS485 (default).	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.

The initialization is necessary after you have executed a hardware reset with the instruction **RS_RESET**.

If transfer protocol RS485 is set, the transfer direction must be set, too (with **RS485_SEND**).

See also

CHECK_SHIFT_REG, GET_RS, RS485_SEND, RS_RESET, SET_RS

Valid for

Gold-CAN

Example

```
#INCLUDE ADWGCAN.Inc

INIT:
  RS_RESET()              'Reset RSxxx controller
  RS_INIT(1,9600,0,8,0,1)'Initialization of RSxxx interface 1
                          'with 9600 Baud, without parity,
                          '8 data bits, 1 stop bit and
                          'hardware handshake.
```

RS_Reset

RS_RESET executes a hardware reset and deletes the settings for all RSxxx interfaces.

Syntax

```
#INCLUDE ADWGCAN.Inc

RS_RESET()
```

Parameters

- / -

Notes

RS_RESET sends a reset impulse to the input of the controller TL16C754. In the data-sheet of the controller 16C754 from Texas Instruments it is described, to which values the registers have been set after the hardware reset.

After a hardware reset an initialization with **RS_INIT** must follow, in order to initialize the controller and to set the interface parameters.

See also

CHECK_SHIFT_REG, GET_RS, RS_INIT, SET_RS

Valid for

Gold-CAN

Example

```
#INCLUDE ADWGCAN.Inc

INIT:
  RS_RESET()              'Reset RSxxx controller
  RS_INIT(1,9600,0,8,0,1)'Initialization of RSxxx interface 1
                         'with 9600 Baud, without parity,
                         '8 data bits, 1 stop bit and
                         'hardware handshake.
```

Set_RS

SET_RS writes a value into a specified register of the controller.

Syntax

```
#INCLUDE ADWGCAN.Inc
SET_RS(reg_addr,value)
```

Parameters

reg_addr	Number of the register, into which data are written.	LONG
value	Value to be written into the register.	LONG

Notes

We recommend to use this instruction only after you have more experience about how the controller operates (data-sheet of the manufacturer: TL16C754 from Texas Instruments). For more common applications more comfortable instructions are availabe in the include file.

See also

GET_RS, RS_INIT, RS_RESET

Valid for

Gold-CAN

Example

-/-

Write_FIFO

WRITE_FIFO writes a value into the send-FIFO of a specified RSxxx interface.

Syntax

```
#INCLUDE ADWGCAN.Inc
ret_val = WRITE_FIFO(interface,value)
```

Parameters

interface	RSxxx interface number (1, 2) to whose send-FIFO data are transferred.	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.

See also

CHECK_SHIFT_REG, READ_FIFO, RS_INIT, RS_RESET, RS485_SEND

Valid for

Gold-CAN

Example

```
#INCLUDE ADWGCAN.Inc
DIM val AS LONG

INIT:
  RS_RESET()
  RS_INIT(1,9600,0,8,0,1)'Initialization of RSxxx interface 1
                         'with 9600 Baud, no parity,
                         '8 data bits, 1 stop bit and
                         'hardware handshake.

EVENT:
  Par_1 = WRITE_FIFO(1,val)'If the FIFO is not full, [val]
                           'is written into the FIFO. Otherwise
                           'a 1 in PAR_1 indicates that writing
                           'into the FIFO ist not possible
                           '(FIFO full).
```

12.6 SSI interface

This section describes the following instructions:

SSI_Mode

SSI_MODE sets the modes of all SSI decoders, either "single shot" (read out once) or "continuous" (read out continuously).

Syntax

```
#INCLUDE ADWGCNT.Inc
SSI_MODE(pattern)
```

Parameters

pattern — Operation mode of the SSI decoders, indicated as bit pattern. A bit is assigned to each of the decoders (see table). LONG

Bit = 0: "Single shot" mode, the encoder is read out once.

Bit = 1: "Continuous" mode, the encoder is read out continuously.

Bit no.	31:2	3	2	1	0
SSI decoder	–	4	3	2	1

Notes

If you select "continuous" mode, reading the encoder is started immediately. **SSI_START** is not necessary then.

Using the "continuous" mode, some encoder types occasionally return the wrong counter value 0 (zero) instead of the corrct counter value. This error does not occur with the "single shot" mode.

See also

SSI_Read, SSI_Set_Bits, SSI_Set_Clock, SSI_Start, SSI_Status

Valid for

Gold-CAN

Example

```
#INCLUDE ADWGCAN.Inc
Rem Decoder 1 runs 1.0 MHz, Decoder 2 runs 0.4 MHz
INIT:
  SSI_SET_CLOCK(1,10)   'clock rate for decoder 1
  SSI_SET_CLOCK(2,25)   'clock rate for decoder 2
  SSI_MODE(11b)         'Set continuous-mode for encoders 1+2
  SSI_SET_BITS(1,10)    '10 encoder bits for encoder 1
  SSI_SET_BITS(2,25)    '25 encoder bits for encoder 2

EVENT:
  Par_1 = SSI_READ(1)   'Read out position value (encoder 1)
  Par_2 = SSI_READ(2)   'Read out position value (encoder 2)
```

SSI_Read

SSI_READ returns the last saved counter value of a specified SSI counter.

Syntax

```
#INCLUDE ADWGCNT.Inc
ret_val = SSI_READ(dcdr_no)
```

Parameters

dcdr_no	Number (1…4) of the SSI decoder whose counter value is to be read.	LONG
ret_val	Last counter value of the SSI counter (= absolute value position of the encoder).	LONG

Notes

An encoder value is saved when the bits indicated by **SSI_SET_BITS** are read.

See also

SSI_Mode, SSI_Set_Bits, SSI_Set_Clock, SSI_Start, SSI_Status

Valid for

Gold-CAN

Example

```
#INCLUDE ADWGCAN.Inc
Rem Decoder runs 200 kHz
DIM m, n, y AS LONG

INIT:
  SSI_SET_CLOCK(1,50)   'clock rate for decoder 1
  SSI_MODE(1)           'Set continuous-mode (encoder 1)
  SSI_SET_BITS(1,23)    '23 encoder bits for encoder 1

EVENT:
  Par_1 = SSI_READ(1)   'Read out position value (encoder 1)

  REM Change value from Gray-code into a binary value:
  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 for an SSI counter the amount of bits which generate a complete encoder value.

The number of bits should be equal to the resolution of the encoder.

Syntax

```
#INCLUDE ADWGCNT.Inc
SSI_SET_BITS(dcdr_no,bit_count)
```

Parameters

dcdr_no	Number (1…4) of the SSI decoder whose resolution is to be set.	LONG
bit_count	Amount of bits (1…32) of the 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 which are transferred.

See also

SSI_Mode, SSI_Read, SSI_Set_Clock, SSI_Start, SSI_Status

Valid for

Gold-CAN

Example

```
#INCLUDE ADWGCAN.Inc
Rem Decoder 1 runs 1.0 MHz, Decoder 2 runs 0.4 MHz
INIT:
  SSI_SET_CLOCK(1,50)   'clock rate for decoder 1
  SSI_SET_CLOCK(2,50)   'clock rate for decoder 2
  SSI_MODE(11b)         'Set continuous-mode (encoders 1+2)
  SSI_SET_BITS(1,10)    '10 encoder bits for encoder 1
  SSI_SET_BITS(2,25)    '25 encoder bits for encoder 2

EVENT:
  Par_1 = SSI_READ(1)   'Read out position value (encoder 1)
  Par_2 = SSI_READ(2)   'Read out position value (encoder 2)
```

SSI_Set_Clock

SSI_SET_CLOCK sets the clock rate (approx. 40 kHz to 1 MHz) , with which the encoder is clocked.

Syntax

```
#INCLUDE ADWGCNT.Inc
SSI_SET_CLOCK(dcdr_no,prescale)
```

Parameters

dcdr_no	Number (1…4) of the SSI decoder whose clock rate is to be set.	LONG
prescale	Scale factor (10…255) for setting the clock rate according to the equation: Clock rate = 10 MHz / prescale.	LONG

Notes

Scale factors < 10 are automatically corrected to the value 10; from values > 255 only the least significant 8 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_Start, SSI_Status

Valid for

Gold-CAN

Example

```
#INCLUDE ADWGCAN.Inc
Rem Decoder 1 runs 1.0 MHz, Decoder 2 runs 0.4 MHz
INIT:
  SSI_SET_CLOCK(1,10)   'clock rate for decoder 1
  SSI_SET_CLOCK(2,20)   'clock rate for decoder 2
  SSI_MODE(11b)         'Set continuous-mode for encoder 1+2
  SSI_SET_BITS(1,10)    '10 encoder bits for encoder 1
  SSI_SET_BITS(2,25)    '25 encoder bits for encoder 2

EVENT:
  Par_1 = SSI_READ(1)   'Read out position value (encoder 1)
  Par_2 = SSI_READ(2)   'Read out position value (encoder 2)
```

SSI_Start

SSI_START starts the reading of one or both SSI encoders (only in mode "single shot").

Syntax

```
#INCLUDE ADWGCNT.Inc
SSI_START(pattern)
```

Parameters

pattern	Bit pattern for selecting the SSI decoders which are to be started: Bit = 0: No function. Bit = 1: Start reading of the SSI decoder.	LONG

Bit no.	31:2	3	2	1	0
SSI decoder	–	4	3	2	1

Notes

In continuous mode **SSI_START** 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.

See also

SSI_Mode, SSI_Read, SSI_Set_Bits, SSI_Set_Clock, SSI_Status

Valid for

Gold-CAN

Example

```
#INCLUDE ADWGCAN.Inc
Rem Both decoders run 40 kHz
INIT:
  SSI_SET_CLOCK(1,250)  'clock rate for decoder 1
  SSI_SET_CLOCK(2,250)  'clock rate for decoder 2
  SSI_MODE(0)           'Set single shot-mode (all counters)
  SSI_SET_BITS(1,23)    '23 encoder bits for encoder 1
  SSI_SET_BITS(2,23)    '23 encoder bits for encoder 2

EVENT:
  SSI_START(11b)        'Read position value of encoders 1 & 2
  DO                    'for encoder 1:
  UNTIL (SSI_STATUS(1) = 0) 'If position value is read completely …
  Par_1 = SSI_READ(1)   'read out and display position value
  DO                    'For encoder 2:
  UNTIL (SSI_STATUS(2) = 0) 'If position value is read completely …
  Par_1 = SSI_READ(2)   'read out and display position value
```

SSI_Status

SSI_STATUS returns the current read-status on the speicified module for a specified decoder.

Syntax

```
#INCLUDE ADWGCNT.Inc
ret_val = SSI_STATUS(dcdr_no)
```

Parameters

dcdr_no	Number (1…4) of the SSI decoder whose status is to be queried.	LONG
ret_val	Read-status of the decoder: 0: Decoder is ready, that is a complete value was has been read. 1: Decoder is reading an encoder value.	LONG

Notes

Use the status query only in the SSI mode "single shot". In the mode "continuous" querying the status is not useful.

See also

SSI_Mode, SSI_Read, SSI_Set_Bits, SSI_Set_Clock, SSI_Start

Valid for

Gold-CAN

Example

```
#INCLUDE ADWGCAN.Inc
Rem Both decoders run 40 kHz
INIT:
  SSI_SET_CLOCK(1,250)  'clock rate for decoder 1
  SSI_SET_CLOCK(2,250)  'clock rate for decoder 2
  SSI_MODE(0)           'Set single shot-mode (all counters)
  SSI_SET_BITS(1,23)    '23 encoder bits for encoder 1
  SSI_SET_BITS(2,23)    '23 encoder bits for encoder 2

EVENT:
  SSI_START(11b)        'Read position value of encoders 1 & 2
  DO                    'For encoder 1:
  UNTIL (SSI_STATUS(1) = 0) 'If position value is read completely …
  Par_1 = SSI_READ(1)   'Read out and display position value
  DO                    'For encoder 2:
  UNTIL (SSI_STATUS(2) = 0) 'If position value is read completely …
  Par_1 = SSI_READ(2)   'Read out and display position value
```

Annex

A.1 Technical Data

All technical data refer to a powered-up ***ADwin-Gold*** system.

General Data/Limit Values						
	Symbol	Conditions	min.	typ.	max.	Unit
Supply Voltage/Supply Current						
Voltage	U_b		10	12	35	V
Idle current, USB Interface	$I_{idle, USB}$	U_b=10V		1.1		A
		U_b=12V[a]		0.9		
		U_b=35V		0.3		
		U_b=12V; *Gold-DA*		1.4		
Power-up current, USB Interface	$I_{power-on, USB}$	U_b=12V[a]	1.7			
		U_b=12V; *Gold-DA*	2.9			
Idle current, Ethernet Interface	$I_{idle, USB}$	U_b=10V		1.3		A
		U_b=12V[a]		1.1		
		U_b=35V		0.4		
		U_b=12V; *Gold-DA*		1.5		
Power-up current, Ethernet Interface	$I_{power-on, Enet}$	U_b=12V[a]	2.1			
		U_b=12V; *Gold-DA*	3.1			
Valid operation ranges						
Temperature	$T_{chassis}$		+5		+60	°C
Relative humidity	F_{rel}	no condensation	0		80	%
Storage						
Temperature	T		-20		+70	°C
Connectors						
DSUB connectors	Metric ISO threads; UNC threads available as ordering option					
Dimensions						
Width × height × depth	W × H × D	*Gold-USB*, *Gold-ENET*	214 × 67 × 109			mm
		with *CAN* Add-On	Height: +20			
Net weight						
Weight	m_{Net}	*Gold-USB*, *Gold-ENET*	1320			g
		with *CAN* Add-On	1760			
		Clips[b]	32			

a. applies to Gold-CO1, too

b. Accessories for DIN rail mounting: *Gold-Mount*

Digital Inputs/Outputs						
Parameters	Symbol	Conditions	min.	typ.	max.	Unit
I/O-lines						
Number	DIO00:DIO31	32 (programmable in groups of 8 as inputs or outputs)				
	EVENT	ext. trigger input (positive TTL logic)				
Inputs						
Max. input voltage		V_{CC} = 5V	-0.5		+5.5	V
Logic input voltage	V_{IH} (High)	V_{CC} = 5V	2.4			
	V_{IL} (Low)	V_{CC} = 5V			0.8	
Logic input current	I_I	V_{CC} = 5V		±0.01	±2	µA
Outputs						
Logic output voltage	V_{OH} (High)	I_{OH}= -6mA	3.84	4.3		V
	V_{OL} (Low)	I_{OL}= +6mA		0.17	0.33	
Logic output current	I_O	per DIO line			±35	mA
	I_{TOTAL}	all DIGIN or. all DIGOUT via V_{CC} / GND			±70	
EVENT Input						
Edge recognition, pos.	V_{T+} (Low)	V_{CC} = 5V	1.65	1.9	2.15	V
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	

Analog Inputs/Outputs						
Parameters	Symbol	Conditions	min.	typ.	max.	Unit
Inputs						
Number	2 × 8 via multiplexer, differential					
Input resistance	R_i		323.4	330	336.6	kΩ
Overvoltage	$U_{in\ max.}$	ON & OFF			±35	V
Multiplexer settling time	t_{MUX}	1 LSB 14-bit		2.5		µs
		1 LSB 16-bit		6.5		µs
ADC 14-bit						
Conversion time	t_{conv}				0.5	µs
Measurement range	U_{in}	F_V=1	-10		+9.999695	V
		F_V=2	-5		+4.999847	
		F_V=4	-2.5		+2.499924	
		F_V=8	-1.25		+1.249962	
Diff. common mode voltage.					±2.5	
Integral non-linearity	INL			±1	±3	LSB
Differential non-linearity	DNL			±0.25	±0.5	

Analog Inputs/Outputs						
Parameters	Symbol	Conditions	min.	typ.	max.	Unit
Offset	Drift			±2		ppm/K
	Error	adjustable				
Gain	Drift			±5		ppm/K
	Error	adjustable				
ADC 16-bit						
Conversion time	t_{conv}				5	µs
Measurement range	U_{in}	F_V=1	-10		+9.999695	V
		F_V=2	-5		+4.999847	
		F_V=4	-2.5		+2.499924	
		F_V=8	-1.25		+1.249962	
Diff. common mode voltage					±2.5	
Integral non-linearity	INL			±1	±3	LSB
Differential non-linearity	DNL			±0.25	±0.5	
Offset	Drift			±2		ppm/K
	Error	Adjustable				
Gain	Drift			±5		ppm/K
	Error	Adjustable				
Outputs: DAC 16-bit						
Number	2 (with DA add-on: 8)					
Output voltage	U_{out}		-10		+9.999695	V
Settling time	t_{settle}	2V jump		3		µs
		FSR[a] (20V)		10		
Permissible current					±25	mA
Integral non-linearity	INL				±2	LSB
Differential non-linearity	DNL				±1	
Offset	Drift			±1		ppm/K
	Error	Adjustable				
Gain	Drift			±3		ppm/K
	Error	Adjustable				

a. Full Scale Range

Processor						
Parameters	Symbol	Conditions	min.	typ.	max.	Unit
Type	ADSP21062 (SHARC™)					
Manufacturer	Analog Devices					
Clock frequency	f_{CLK}			40		MHz
Register width				32		Bit
Internal memory	SRAM	for programs		128	256 [a]	kByte
		for data		128	256 [a]	
External memory	SDRAM			16	64 [a]	MByte

a. combined memory expansion G-MEM-64

CO1 Add-On						
Parameters	Symbol	Conditions	min.	typ.	max.	Unit
Counter						
Number	4 counters (CNTR1 … CNTR4)					
Inputs	For each counter 3 differential inputs (A/CLK, B/DIR, CLR/LATCH); counter inputs programmable in pairs for differential or TTL mode (single-ended)					
Counter resolution				32		Bit
Count frequency	f_{CLK}	Input CLK		20		MHz
		Input A/B		5		
Latch width	LATCH			32		Bit
Reference quartz oscillator						
Reference frequency	f_{ref}			20		MHz
Prescaler by 4	f_{ref} / 4			5		
Accuracy and Drift					100	ppm
Counter inputs differential[a]						
Differential input threshold voltage	V_{TH}	-10V ≤ V_{CM} ≤ 13.2V	-200		+200	mV
Input hysteresis	ΔV_{TH}	-10V ≤ V_{CM} ≤ 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
Counter inputs single ended[b] (with Schmitt trigger)						
Edge recognition, pos.	V_{T+} (Low)	V_{CC} = 5V	1.65	1.9	2.15	V
Edge recognition, neg.	V_{T-} (Low)		0.75	1.0	1.25	
Switching hysteresis	V_{T+} - V_{T-}		0.4	0.9		
Input current	I_H	V_I = 2.7V			20	µA
	I_L	V_I = 0.4V			-50	

a. see also data sheet MAX3098 from MAXIM

b. see also data sheet 74LS19 from Texas Instruments

A.2 Hardware Addresses - General Overview

Hardware addresses for ADCs

Address [HEX]	Function	Bit 31:16	15:10	9	8	7	6	5	4	3	2	1	0	Commentary
20400000	Set MUX 1: channels 1, 3, 5, …, 15	-	-	-	-	-	-	-	-	-	n	n	n	""nnn"" binary = 0...7 decimal, selected ch. = nnn +
	Set MUX 2: channels 2, 4, 6, …, 16	-	-	-	-	-	-	n	n	n	-	-	-	""nnn"" binary = 0...7 decimal, selected ch. = 2(nnn + 1
	Gain PGA 1	-	-	-	-	g	g	-	-	-	-	-	-	""gg"" binary = 0...3 decimal, selected gain = 2gg
	Gain PGA 2	-	-	g	g	-	-	-	-	-	-	-	-	
20400010	Start conversion: ADC 1 (16-bit)	-	-	-	-	-	-	-	-	-	1	-	s	s = 0 : start conversion s = 1 : no effect
	Start conversion: ADC 2 (16-bit)	-	-	-	-	-	-	-	-	-	1	s	-	
	Start conversion: ADC 1 (14-bit)	-	-	-	-	-	-	-	-	s	1	-	-	
	Start conversion: ADC 2 (14-bit)	-	-	-	-	-	-	-	s	-	1	-	-	
20400020	EOC status: ADC 1 (16-bit)	-	-	-	-	-	-	-	-	-	-	-	e	e = 0 : end of conversion e = 1 : conversion is running
	EOC status: ADC 2 (16-bit)	-	-	-	-	-	-	-	-	-	-	e	-	
	EOC status: ADC 1 (14-bit)	-	-	-	-	-	-	-	-	e	-	-	-	
	EOC status: ADC 2 (14-bit)	-	-	-	-	-	-	-	e	-	-	-	-	
20400030	Read out register: ADC 1 (16-bit)	-	x	x	x	x	x	x	x	x	x	x	x	x : result of conversion
20400040	Read out register: ADC 2 (16-bit)	-	x	x	x	x	x	x	x	x	x	x	x	
20400130	Read out register: ADC 1 (14-bit)	-	x	x	x	x	x	x	x	x	x	0	0	
20400140	Read out register: ADC 2 (14-bit)	-	x	x	x	x	x	x	x	x	x	0	0	
20400100	Read out register and start conversion: ADC 1 (16-bit)	-	x	x	x	x	x	x	x	x	x	x	x	
20400110	Read out register and start conversion: ADC 2 (16-bit)	-	x	x	x	x	x	x	x	x	x	x	x	
20400120	Read out register and start conversion: ADC 1 (14-bit)	-	x	x	x	x	x	x	x	x	x	x	x	
204001D0	Read out register and start conversion: ADC 2 (14-bit)	-	x	x	x	x	x	x	x	x	x	x	x	

Hardware addresses for DACs

Address [HEX]	Function	Bit 31:16	15:10	9	8	7	6	5	4	3	2	1	0	Commentary
20400010	Start conversion: All DACs synchronously	-	-	-	-	-	-	-	1	1	s	1	1	s = 0 : start conversion s = 1 : no effect
20400050	Write only to the register: DAC 1	-	x	x	x	x	x	x	x	x	x	x	x	x : digital value to be converted
20400060	Write only to the register: DAC 2	-	x	x	x	x	x	x	x	x	x	x	x	
20400070	Write only to the register: DAC 3 (*Gold-DA*)	-	x	x	x	x	x	x	x	x	x	x	x	
20400080	Write only to the register: DAC 4 (*Gold-DA*)	-	x	x	x	x	x	x	x	x	x	x	x	
20400090	Write only to the register: DAC 5 (*Gold-DA*)	-	x	x	x	x	x	x	x	x	x	x	x	
204000A0	Write only to the register: DAC 6 (*Gold-DA*)	-	x	x	x	x	x	x	x	x	x	x	x	
20400190	Write only to the register: DAC 7 (*Gold-DA*)	-	x	x	x	x	x	x	x	x	x	x	x	
204001A0	Write only to the register: DAC 8 (*Gold-DA*)	-	x	x	x	x	x	x	x	x	x	x	x	
20400200	Write to the register and start conversion immediately: DAC 1	-	x	x	x	x	x	x	x	x	x	x	x	x : digital value to be converted
20400210	Write to the register and start conversion immediately: DAC 2	-	x	x	x	x	x	x	x	x	x	x	x	
20400220	Write to the register and start conversion immediately: DAC 3 (*Gold-DA*)	-	x	x	x	x	x	x	x	x	x	x	x	
20400230	Write to the register and start conversion immediately: DAC 4 (*Gold-DA*)	-	x	x	x	x	x	x	x	x	x	x	x	
20400240	Write to the register and start conversion immediately: DAC 5 (*Gold-DA*)	-	x	x	x	x	x	x	x	x	x	x	x	
20400250	Write to the register and start conversion immediately: DAC 6 (*Gold-DA*)	-	x	x	x	x	x	x	x	x	x	x	x	
20400260	Write to the register and start conversion immediately: DAC 7 (*Gold-DA*)	-	x	x	x	x	x	x	x	x	x	x	x	
20400270	Write to the register and start conversion immediately: DAC 8 (*Gold-DA*)	-	x	x	x	x	x	x	x	x	x	x	x	

Hardware addresses for digital inputs / outputs

Address [HEX]	Function	Bit 31:16	15:10	9	8	7	6	5	4	3	2	1	0	Commentary
204000B0	Input registers DIO15:00	-	x	x	x	x	x	x	x	x	x	x	x	x : digital value read in
204001B0	Input registers DIO31:16	-	x	x	x	x	x	x	x	x	x	x	x	
204001C0	Output registers DIO15:00	-	x	x	x	x	x	x	x	x	x	x	x	x : digital value to be output
204000C0	Output registers DIO31:16	-	x	x	x	x	x	x	x	x	x	x	x	

Hardware addresses for CO1 counter add-on

Address [HEX]	Function	Bit 31:04	3	2	1	0	Commentary
20400204	Read out Latch A: Counter 1	x	x	x	x	x	x : Contents of the latch
20400208	Read out Latch B: Counter 1	x	x	x	x	x	
20400214	Read out Latch A: Counter 2	x	x	x	x	x	
20400218	Read out Latch B: Counter 2	x	x	x	x	x	
20400224	Read out Latch A: Counter 3	x	x	x	x	x	
20400238	Read out Latch B: Counter 3	x	x	x	x	x	
20400234	Read out Latch A: Counter 4	x	x	x	x	x	
20400238	Read out Latch B: Counter 4	x	x	x	x	x	
20400300	Enable counter	-	x	x	x	x	x = 0 : Disable counter x = 1 : Enable counter
20400304	Set counter inputs to TTL or differential mode (in pairs only)	-	-	-	y	x	x: counter inputs 1+2 y: counter inputs 3+4 x,y = 0: TTL (single-ended) x,y = 1: differential
20400310	Clear counter	-	x	x	x	x	x = 0 : No influence x = 1 : Clear counter
20400320	Latch counter	-	x	x	x	x	x = 0 : No influence x = 1 : Latch counter
20400330	Input: CLR or LATCH	-	x	x	x	x	x = 0 : CLR input x = 1 : LATCH input
20400340	Impulse/event counter or impulse/pause duration measurement	-	x	x	x	x	x = 0 : External clock input x = 1 : Int. ref. clock(20/5MHz)
20400350	4 edge evaluation/CLK+DIR or 20/5MHz reference clock	-	x	x	x	x	CNT_MODE = 0: x = 0 : 4-Fl.; x = 1 : CLK+DIR CNT_MODE = 1: x = 0 : 20MHz; x = 1 : 5MHz
20400370	Counter: Error register[a]	several bits					Error bits, see CNT_GETSTATUS

aYou have to reset this register manually!

A.3 Hardware revisions

The revision of a Gold system 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	1998	First release with link data connection.
B1	Nov. 2002	Protoype (internal use only, not delivered to customers)
B2	Apr. 2003	Data connection to PC no longer via link, but via Ethernet or USB. All analog inputs and counter inputs are only available with differential operation mode.
B3	Nov. 2003	Additional TTL counter inputs for single-ended operation mode (for use as alternative to counter inputs for differential operation mode). New option Gold-D with DSUB connectors instead of BNC sockets.
B4	Dec. 2003	Several enhancements
B5	Mar. 2004	Several enhancements Layout change of printed circuit board
B6	Aug. 2004	Enhanced Ethernet interface (ENET-2) with increased data throughput. New option Gold-CAN with several communication interfaces.

A.4 RoHS Declaration of Conformity

The directive 2002/95/EG 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 1st July, 2006.

The following substances are involved:

– Lead (Pb)
– Cadmium (Cd)
– Hexavalent chromium (Cr VI)
– Polybrominated biphenyls (PBB)
– Polybrominated diphenyl ethers (PBDE)
– Mercury (Hg)

The product line ***ADwin-Gold*** complies with the requirements of the RoHS directive in all delivered variants since June 2006.

A.5 Baudrates for the CAN bus

ADwin-Gold-CAN provides CAN bus interfaces, version „high speed“. The following baudrates can be set:

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
18912.5296	18867.9245	18823.5294	18648.0186	18604.6512
18518.5185	18433.1797	18390.8046	18306.6362	18181.8182

Available Baud rates [Bit/s]				
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
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

Available Baud rates [Bit/s]				
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	

A.6 Table of figures

A.7 Index