JÄGER
Computergesteuerte
Messtechnik GmbH

ADwin-Gold II

Manual

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 absolutely have to be considered in order to guarantee an error free operation.

– advice for efficient operation.

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

<C:\ADwin\ …>

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

Program text

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

Var_1

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

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

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

1 Information about this Manual

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

– the manual "ADwin Installation", which describes all interface installations for the *ADwin* systems.

  Begin the installation with this manual!

– the description of the configuration program ADconfig, with which you initialize the communication from the corresponding interface to your *ADwin* device.

– the manual *ADbasic*, which explains basic instructions for the compiler *ADbasic* and the functional layout of the *ADwin* system.

– the manual *TiCoBasic*, which explains basic instructions for the compiler *TiCoBasic* and the functional layout of the *TiCo* processor.

– 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 milliseconds down to less than one microsecond.

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 3.3 nanoseconds. The ***ADwin*** systems are characterized by an extremely short response time of only 100 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 **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 "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.

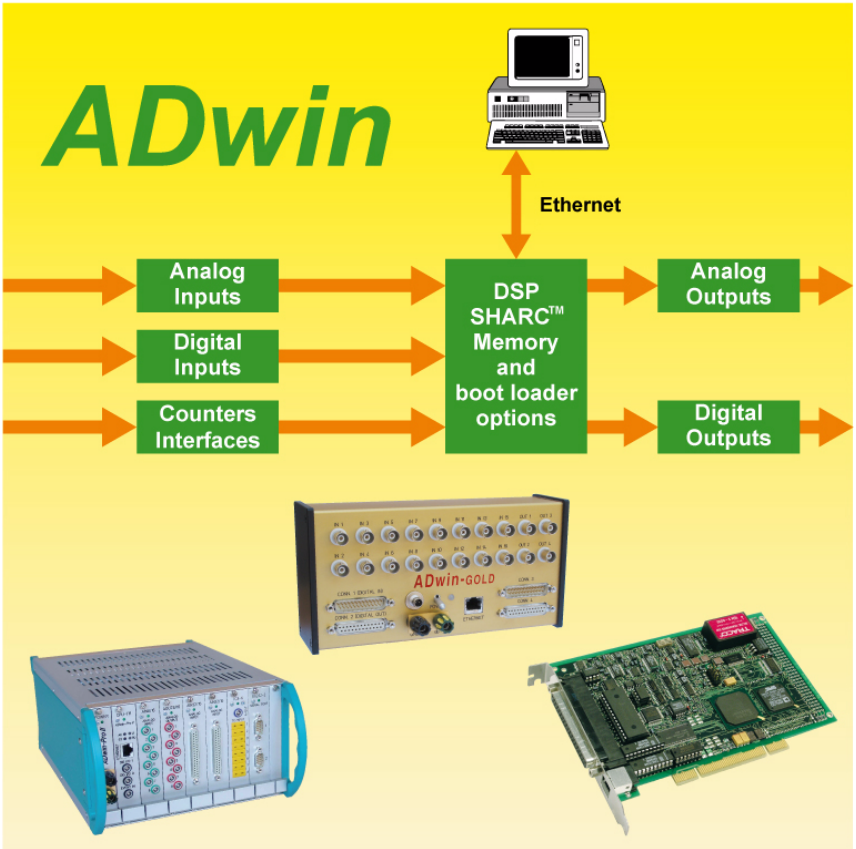


Fig. 1 – Concept of the ***ADwin*** systems

2.2 The *ADwin-Gold II* System

Processor and memory

The *ADwin-Gold II* system is equipped with the digital 32 bit signal processor T11 (DSP TS101S TigerSharc) 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 Kilohertz.

The on-chip memory with 3 × 256 KiB has a very short access time 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 256 MiB (DRAM).

Add-on processor *TiCo*

ADwin-Gold II provides an independent, freely programmable add-on processor, the *TiCo* processor (*Ti*ming *Co*ntroller). The *TiCo* processor can access all inputs and outputs and therefore perform special tasks as conversion, communication protocol (SPI), signal generator, loop control etc. According to the task the *TiCo* processor can help the processor T11 by pre-processing data, or it may perform an independent task.

The *TiCo* processor is optimized for fast response time and exact timing. The processor works with 50 MHz clock speed, a memory of von 28 KiB in PM and DM and it processes integer values of 32 bit length only (data type `LONG`).

Programming of the *TiCo* processors with *TiCoBasic* and other details are described in the manual *TiCoBasic*.

2 processors in parallel

TiCo processor and T11 use a separate bus each to access all inputs, outputs and interfaces and thus can work independently from each other. The Block diagram ADwin-Gold II shows the available inputs and outputs. As an example, the T11 can output values to the DAC while the *TiCo* processor controls the counter inputs.

A simultaneous access of *TiCo* processor and T11 to the same peripherals is not possible.

In the following cases several inputs/outputs share a single connection to each the T11 bus and the *TiCo* bus:

– DAC outputs OUT1…OUT8

– CAN1, CAN2, COM1, COM2

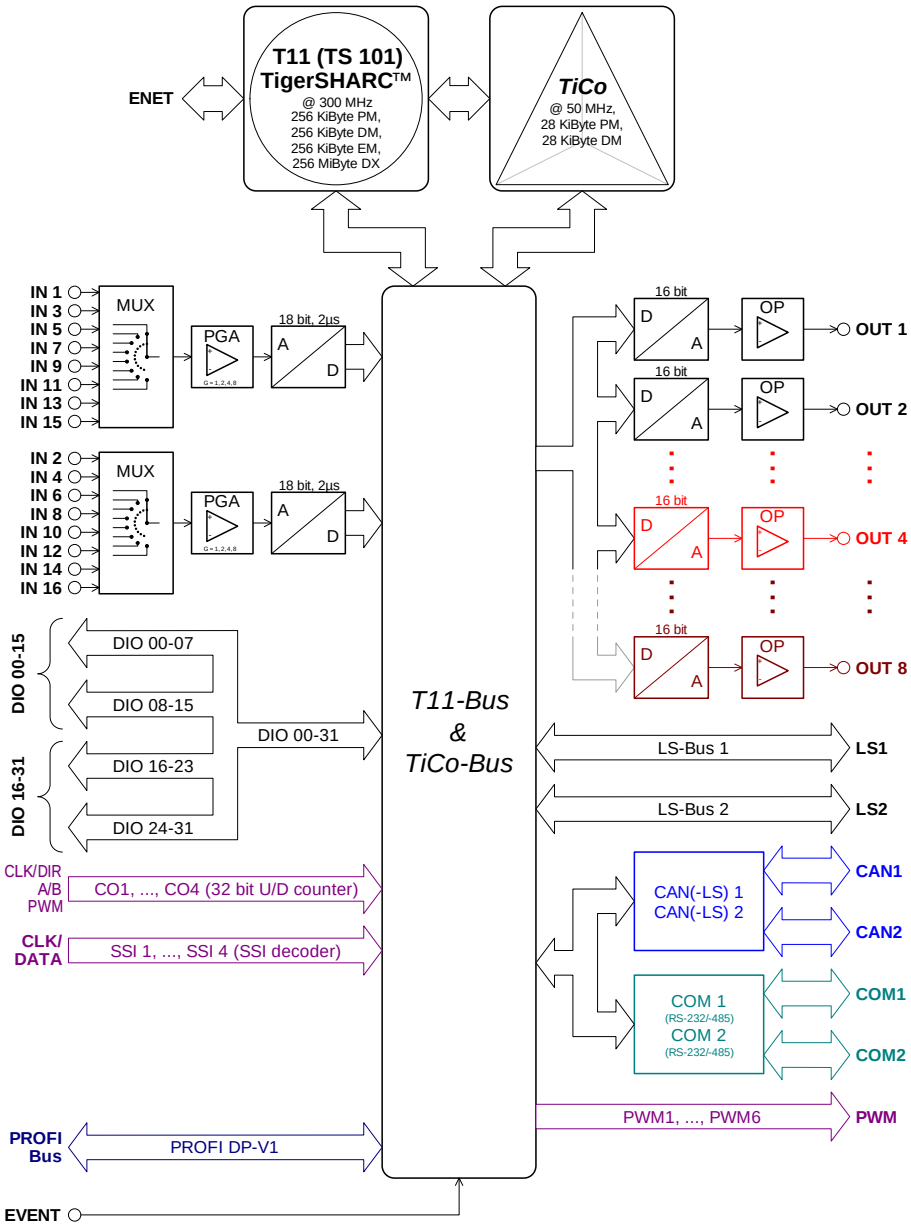


Fig. 2 – Block diagram *ADwin-Gold II*

Analog inputs

The system has 2 × 8 analog inputs with BNC plugs (alternatively: DSub connectors), which are divided into two groups each being connected to one multiplexer. The input signals are converted by a 18-bit analog-to-digital converter (ADC), see Fig. 2 "Block diagram ADwin-Gold II".

Analog outputs

The standard version of *ADwin-Gold II* is equipped with 2 analog outputs (optional 4 or 8) with an output voltage range of -10V … +10V and 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 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)

ADwin-Gold II 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").

Watchdog

The operation of *ADwin-Gold II* can be controlled with a watchdog counter. If a programmed signal is missing unexpectedly, the watchdog creates a reset signal for the T11 and/or *TiCo* processor. The reset signal can also be output on a pin as TTL level.

24 Volt signals

Using the two LS bus interfaces up to 30 LS bus modules can be addressed. The LS bus module HSM-24V provides 32 digital channels for 24 Volt signals.

Connection to the PC

The connection between *ADwin-Gold II* system and PC is made via Ethernet interface. This way, there is direct access to processes and global variables of the T11 processor. Please note, that only the T11 processor has direct access to the *TiCo* processor; a direct access from the PC is not available.

Standard delivery

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

- the *ADwin-Gold II* system with Ethernet interface,
- 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 manual.

2.2.1 Options (no upgrades possible)

The following options are available:

- *Gold II-DA4 / -DA8* (page 19): Add-on to 4 or 8 analog outputs (differential). Each output is equipped with a 16-bit DAC.
- *Gold II-CNT* (page 20): 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. 4 decoders for use with incremental encoders with SSI interface. 6 PWM outputs.
- *Gold II-CAN* (page 30): 2 CAN interfaces (both either high speed *or* low speed) and 2 RSxxx interfaces (RS232, RS485).
- *Gold II-Profibus* (page 38): Profibus interface. Excludes options DeviceNet and EtherCAT.
- *Gold II-DeviceNet* (page 41): DeviceNet interface. Excludes options Profibus and EtherCAT.
- *Gold II-EtherCAT* (page 43): EtherCAT interface. Excludes options Profibus and DeviceNet.
- *Gold II-Storage-16* (page 47): Memory card with 16 GiB memory as well as a battery buffered real-time clock.
- *Gold II-Boot* (page 48): Flash-EPROM boot loader for stand-alone operation without PC.

All additional options can be combined with each other. As an exception, the options Profibus, DeviceNet and EtherCAT cannot be combined.

2.2.2 Accessories

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

3 Operating Environment

The *ADwin-Gold II* 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 II* device **must be earth-protected**, in order to

– build a ground reference point for the electronic

– conduct interferences to earth.

Connect the GND plug via a short low-impedance solid-type cable to the central earth connection point of your device.

Galvanic connection

A galvanic connection combines links the ground potential of an *ADwin-Gold II* (and thus of your device) with an external ground potential. A voltage difference between ground potentials interfers with operation and can cause considerable damage. Avoid galvanic connection or at least minimize voltage difference.

The following components establish a galvanic connection:

Component	Connection to
Power supply via power adapter in the PC	Ground potential of the PC
Some versions of the power supply unit Gold II-Pow	Power grid supplying Gold II-Pow
Shielding of the Ethernet cable	Ground potential of the device on the opposite of the Ethernet cable

If the power supply unit Gold II-Pow really establishes a galvanic connection, you can only find out by a measurement.

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 II* aluminum enclosure by using screen clips.

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 II* 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 II* 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 II* 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 II* 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 II* 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 II* 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. 3). Connect the 3-pin subminiature connector there. For the pin assignment see the following picture:

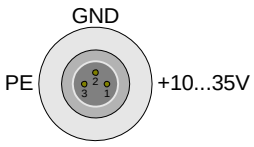


Fig. 3 – Power supply connector (male)

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 6). 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

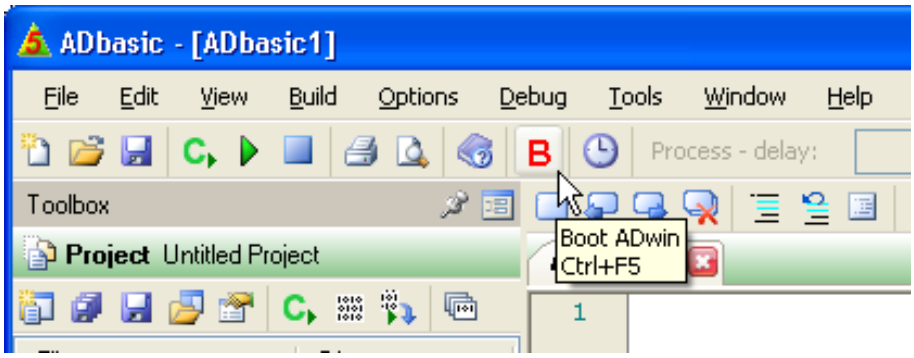
If you have completed the installation of the *ADwin* drivers and the configurations in the *ADbasic* menu "Options\Compiler", then connect the USB or Ethernet data transfer cables and the power supply cable. Then start the computer.

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 for a moment and then in green.

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).

Programs with *ADbasic*

Programming the *ADwin* systems is described more detailed in the *ADbasic* manual. Instructions for access to *ADwin-Gold II* I/Os are described in chapter 15 on page 50.

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

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 *ADwin-Gold II*.

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.

ADwin-Gold II provides several pins with a voltage of +5V. The maximum current of 100mA applies to all pins together.

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

- 16 analog inputs via 2 multiplexers (page 11)
- 2 analog outputs (page 13)
- 32 digital inputs/outputs (page 15)
- watchdog output (page 16)
- 2 LS bus interfaces (page 15); up to 30 LS bus modules can be addressed. The LS bus module HSM-24V provides 32 digital channels for 24 Volt signals.

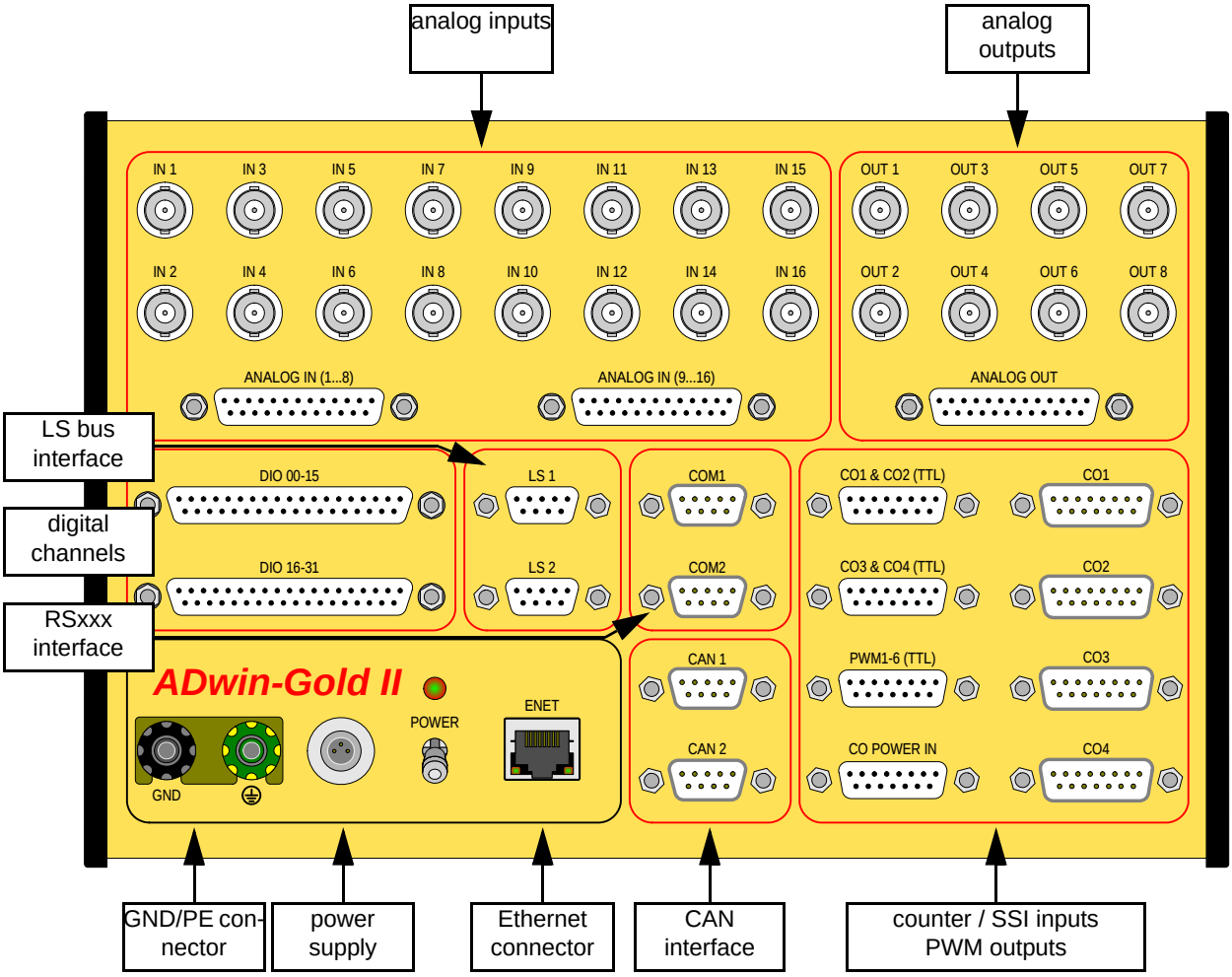


Fig. 4 – Schematic of *ADwin-Gold II*

5.1 Analog Inputs and Outputs

In order to operate the system's BNC plugs 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 II* 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 II* system with the earth of the computer. If you do not operate the PC and the *ADwin-Gold II* 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.

For fast and easy programming there are standard instructions available in the standard instructions compiler *ADbasic*, which enable a user to easily measure or output data (see **ADC** or **DAC**, page 60). Use other instructions only if extremely time-critical or special tasks require to do so. (See also *ADbasic* manual).

5.1.1 Analog Inputs

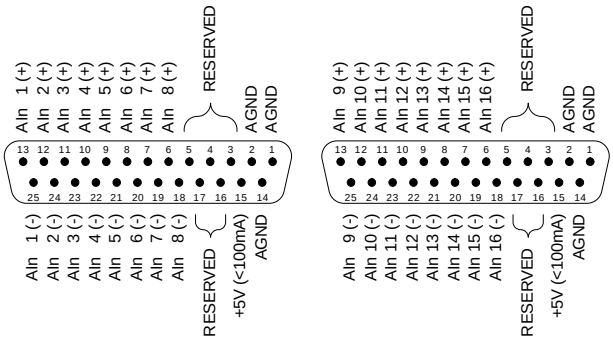
2 Multiplexers

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 ADwin-Gold II", page 5).

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; below, the inputs are 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 1…8 ANALOG IN 9…16

Fig. 5 – Pin assignment of analog inputs (DSub)

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

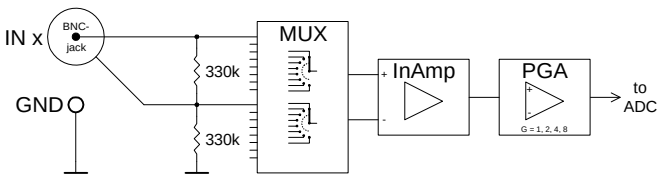


Fig. 6 – Input circuitry of an analog input

18-bit measurements

Signals at the multiplexer outputs are converted with a 18-bit analog-to-digital-converter (ADC), see Fig. 2 "Block diagram ADwin-Gold II". The 18-bit ADC works fast (max. 2µs) and very accurately (resolution 76µV).

There are two methods to convert measurement values, which are used alternatively:

– single conversion: The conversion is started at a defined time and the value is returned after the appropriate delay. The value can have a resolution of 16 bit or 18 bit.

– continuous conversion: A sequential control continuously converts 16 bit values. You can read the current value without waiting, but the exact time of conversion is unknown.

Single conversion

The instruction **ADC** executes a complete measurement with one of the ADCs on the analog input (see page 63) and returns a 16 bit value. The instruction considers for instance the settling of the multiplexer and assures perfect measurements.

Measurement values with 18 bit resolution are available with the instruction **ADC24** (see page 65); the values are returned with 24 bit format (see page 14).

Both instructions run with the 18-bit ADC, only return values have different resolution.

Continuous conversion with sequential control

ADwin-Gold II provides a sequential control for each ADC, which can consecutively read measurement values of several or all input channels of an ADC. Thus, the processor can be discharged a lot and only has to read completely converted values from the sequential control's buffer. The sequential controls work independently from each other.

Programming

Instructions to program analog inputs are described starting from page 63. The instructions are defined in the include file <ADwinGoldII.inc> and are described in the online help, too.

Function	Instructions
do a complete measurement	**ADC**, **ADC24**
Do a measurement in steps (see chapter 5.5)	**SET_MUX1**, **SET_MUX2** **Start_Conv**, **Wait_EOC** **READ_ADC**
initialize and start sequential control	**SEQ_MODE** **SEQ_SELECT** **SEQ_SET_DELAY** **SEQ_SET_GAIN** **SEQ_START**
Read data from sequential control	**SEQ_READ** **SEQ_STATUS**

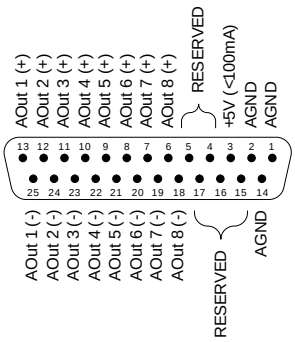
Please pay attention to a low output resistance of the signal source (of the input signals), because it may have influence on the measuring accuracy. If this is not possible:

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

– From approx. 3kΩ upwards the multiplexer settling time extends.
The waiting time defined in the standard instruction **ADC** is then too short, so that imprecise values are recalled. In this case please use the instructions described in chapter 5.5.

5.1.2 Analog Outputs

The system has 2 analog outputs (OUT1, OUT2) with BNC-plugs; below the outputs are located on the DSub connector ANALOG OUT (see Fig. 7). A digital-to-analog converter (DAC) is allocated to each of the outputs.



ANALOG OUT

Fig. 7 – Pin assignment of analog outputs (DSub)

Additional outputs see chapter 6 "DA Add-On".

Programming

Instructions to program analog outputs are described starting from page 60. The instructions are defined in the include file <ADwinGoldII.inc> and are described in the online help, too.

Function	Instructions
do a complete output of a value	**DAC**
Do an output in steps	**Write_DAC** **Start_DAC**

The standard instruction **DAC** (number, value) 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 II* 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 a 16-bit conversion.

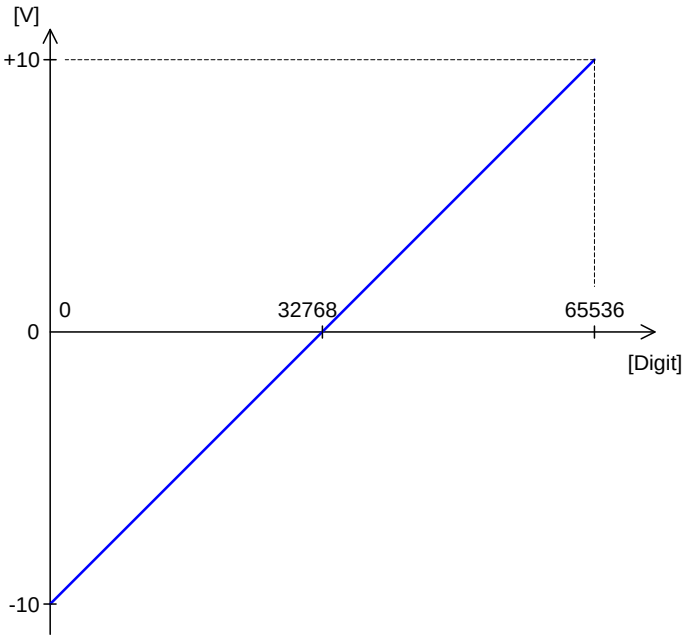


Fig. 8 – Zero offset in the standard setting of bipolar 10 Volts

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 II has a programmable gain (PGA), which can amplify the input voltage by the factors 1, 2, 4, or 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).

The measured value of the 18-bit ADC can be returned with 16 bit or with 24 bit format. The DAC processes values with 16 bit:

– 16-bit format: The measurement value is given in the lower word, the upper word is zero.
U_{LSB} = 20V / 2^{16} = 305.175µV

The same applies for a DAC value.

– 24-bit format: The 24-bit value holds the measurement value in the bits 6…23, the measurement value being shifted by 6 bits to the left, the bits 0…5 are always zero.
U_{LSB} = 20V / 2^{24} = 1.192µV

Bit no.	31…24	23…16	15…6	05…00
content	0	18-bit measurement value in bits 6…23		0
	0	0	16-bit measurement value in bits 0…15	

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

Conversion Digit to Voltage

DAC

For a DAC:

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

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

For an ADC (18-bit and 16-bit):

ADC

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

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

Tolerance Ranges

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

INL

– The integral non-linearity (INL) defines the maximum deviation from the ideal straight line over the whole input voltage range (see Fig. 8, page 14).

DNL

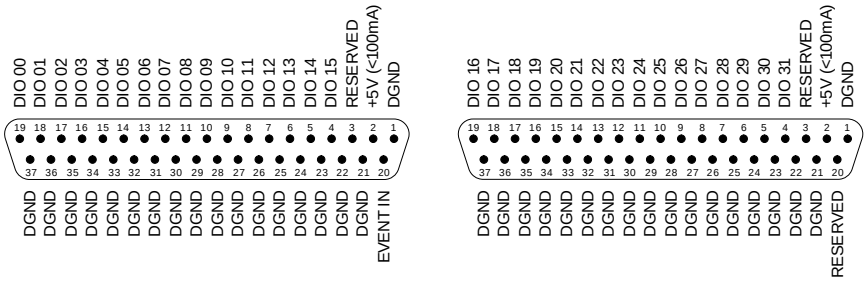
– The differential non-linearity (DNL) defines the maximum deviation from the ideal quantization level.

5.2 Digital Inputs and Outputs

Digital inputs/outputs

On two 37-pin D-SUB sockets there are 32 digital inputs or outputs (DIO 00…DIO 31). They are programmable in groups of 8 as inputs or outputs.

After power-up all 4 groups are configured as inputs.



Inputs / DIO 00-15 (IN) Outputs / DIO 16-31 (OUT)

Fig. 10 – Pin assignment of digital channels

The digital inputs are TTL-compatible and not protected against over voltage. Each digital input has an internal pull-down resistance (2.2kΩ).
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 II* is equipped with an external trigger input (EVENT) at pin 20 of the D-SUB socket DIO 00-15 (see fig. 10).

An external trigger signal with rising edge at the event input can start processes being completely and immediately processed, (see also *ADbasic* manual, chapter: "Program Structure"). The event input has an internal pull-down resistance (4.7kΩ).

In addition to the external trigger input there are more sources for event signals (see page 52). Only one of the sources may be active.

Edge detection unit

ADwin-Gold II can automatically detect edges at selected input channels. 2 options are available:

– Exact protocol of changes: The edge detection unit checks every 10ns, if an edge has occurred at the selected input channels or if a level has been changed. With every change a pair of values is copied into an internal FIFO array:
  - Value 1 contains the level status of all channels as bit pattern.
  - Value 2 contains a time stamp, which is the current value of a 100MHz timer.

The FIFO array may contain 511 value pairs (level status and time stamp) in maximum. Thus, an exact time chronology of level changes is stored. The FIFO data can be read and further processed.

– Register edges: if a positive or a negative edge has occurred at an input, the corresponding bit of the channel is set in a buffer. Number and moment of the edges are not stored.

  The bits can be queried to see which inputs have had a positive or negative edge. A query resets all bits to zero.

Programming

Instructions to program digital inputs/outputs are described starting from page 88:

Function	Instructions
Configure	**Conf_DIO**
Read input values	**Digin_Long**, **Digin_Word1**, **Digin_Word2**
Set output values	**Digout**, **Digout_Bits**, **Digout_Long**, **Digout_Word1**, **Digout_Word2**
Read back output values	**Get_Digout_Long**, **Get_Digout_Word1**, **Get_Digout_Word2**
Control edges at digital inputs	**Digin_FIFO_Clear**, **Digin_FIFO_Enable** **Digin_FIFO_Full**, **Digin_FIFO_Read** **Digin_FIFO_Read_Timer**
Query edges of input channels	**Digin_Edge**

CONF_DIO(12) configures DIO 15:00 as digital inputs and DIO 31:16 as digital outputs (pin assignment see Fig. 10).

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

Fig. 11 – Overview of configurations with **Conf_DIO**

5.3 Watchdog

The correct operation of *ADwin-Gold II* can be controlled with a watchdog counter. If the watchdog counter is enabled, it decrements the counter value continuously. As soon as

the counter value reaches 0 (zero), the system assumes a malfunction and releases a combination of previously configured emergency actions:

– Stop processor T11.

– Stop *TiCo* processor.

– Set output `Watchdog Out` to TTL level low.

  The signal of pin `Watchdog Out` serves to have an external unit controlled by the *ADwin* system to move into a safe operating point. As an example, the watchdog signal could disable the clearing of the power amplifier of the control signal.

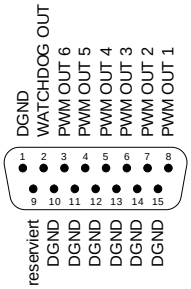
– Set outputs `OUT1`…`OUT8` and `DIO00`…`DIO32` to TTL level low.

Reset the watchdog counter to the starting value at least once during the counting procedure, in order to ensure safe operation of the system.

Watchdog output

The digital output `Watchdog Out` is provided on the 15-pin D-SUB socket `PWM1-6 (TTL)`.

After power-up the output `Watchdog Out` is set to TTL level Low. If the watchdog counter is disabled, the output `Watchdog Out` can be programmed freely.



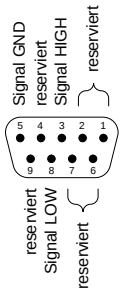
Programming

Instructions to program the watchdog counter are described starting from page 51:

Function	Instructions
Initialize	**`Watchdog_Init`**
Reset	**`Watchdog_Reset`**
Set output value	**`Watchdog_Standby_Value`**
Read watchdog status	**`Watchdog_Status`**

5.4 LS Bus

ADwin-Gold II provides two interfaces for LS bus on 9-pin DSub connectors (female) LS1 and LS2; the pin assignment is shown on the right.



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

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

The LS bus modules are programmed with *ADbasic* instructions, which are sent from the LS bus interface of the *ADwin* system. The instructions are mostly specific for the module and are described in the manual of the LS bus module (or in the online help).

5.5 Time-Critical Tasks

For time-critical tasks it can be useful to split the standard instructions **`ADC`**`()` and **`ADC24`**`()` into a series of instructions.

ADC() and ADC24()

The standard instructions **`ADC`**`()` and **`ADC24`**`()` consist of a sequence of instructions (see below or page 63). The standard instructions need a specified time for execution

which is mostly determined by the settling time of the multiplexer and the conversion time.

```
SET_MUX1()
…                              'wait for settling of the
                               'multiplexer
START_CONV()
WAIT_EOC()                     'wait for end of conversion
READADC()                      'or READADC12() at ADC12()
```

Using waiting times

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 in a sufficient time-delay from the **Set_Mux1** instruction, in order to consider the multiplexer settling time.

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

– The settling time of the multiplexer is 2µs (max.) at a maximum voltage jump of 20 Volt.

– The conversion time of the ADC is 2µs.

6 DA Add-On

Connectors

With the DA add-on there are 4 (Gold II-DA4) or 8 (Gold II-DA8) analog outputs in total with a resolution of 16 bit (and a DAC) each.

The outputs are both connected to the BNC plugs OUT1…OUT8 as to the 37-pin D-SUB socket ANALOG OUT (see figure).

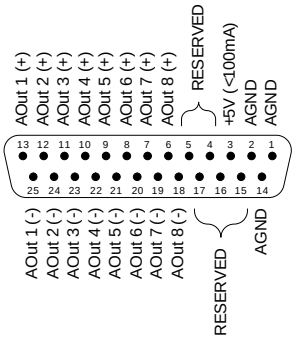


Fig. 12 – Pin assignment ANALOG OUT of DA add-on

Programming

You program the additional DACs like the DAC 1 and DAC 2 (see chapter 5.1 and chapter 15 starting from page 60).

7 CNT Add-On

The add-on Gold II-CNT provides:

– 4 counter blocks: Each counter block provides two 32-bit counters: One up/down counter with clock/direction evaluation or four edge evaluation for quadrature encoders. The other counter for measurement of frequency and duty cycle or high time / low time.

– 4 SSI decoders (page 28)

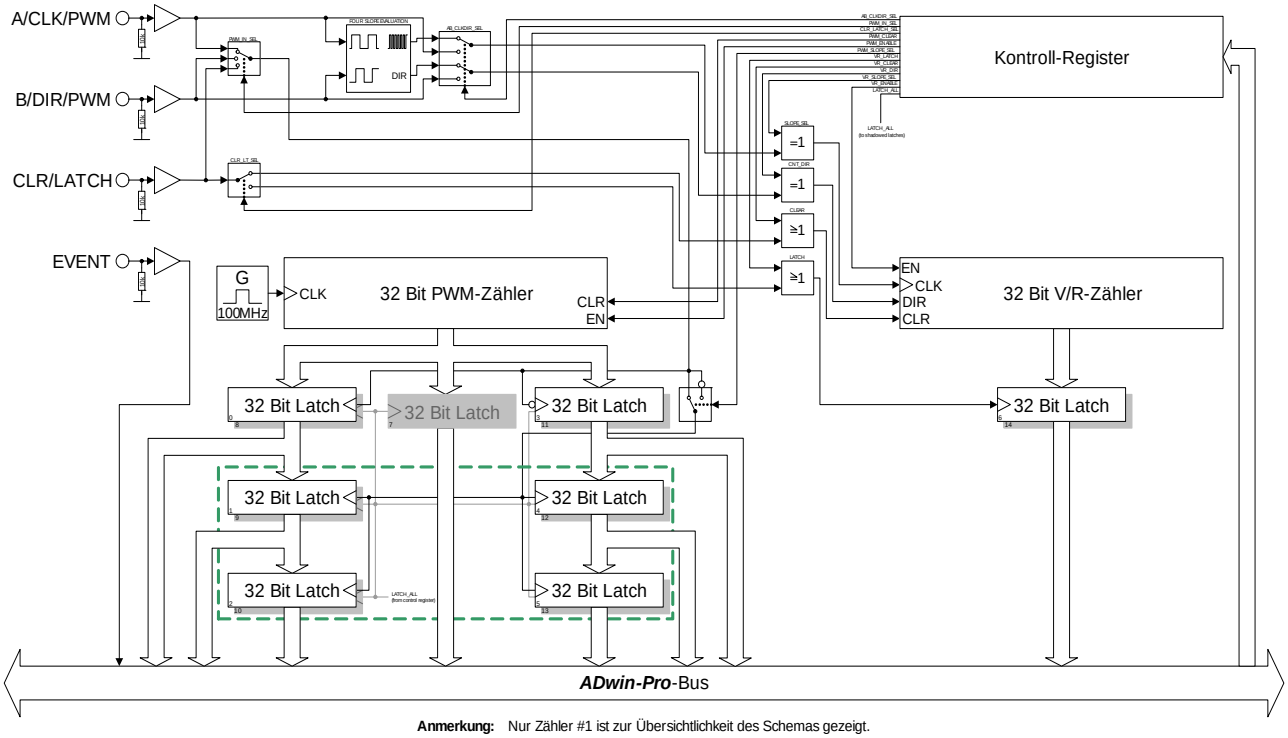
  Decoders for use with incremental encoders with SSI interface. The inputs are provided on the connectors CO1…CO4; signals are differential and designed for RS422/485 level (5V).

– 6 PWM outputs (page 29): Output of pulse-width modulated signals with selectable duty cycle.

7.1 Counter Hardware

Counter block

The four counter blocks of the add-on work with each 2 parallel 32-bit counters: up/down counter for external clock with clock/direction or four edge evaluation, and PWM counters with internal clock. You can configure and read out the counters individually as well as all together. The data of the evaluating units are provided in latches be read out.



Anmerkung: Nur Zähler #1 ist zur Übersichtlichkeit des Schemas gezeigt.

Fig. 13 – Block diagram of a counter block

Up/down counter (external clock input)

With event counting, incrementing/decrementing of the counter is caused by external square-wave signals at the inputs A/CLK and B/DIR.
A positive edge at CLR/LATCH either sets the counter to zero (CLR) or copies the counter values into the latch (LATCH). See also chapter 7.3.

The following modes are possible:

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

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

PWM counter (internal clock input)

For pulse width measurement, incrementing/decrementing of the counter is caused by an internal reference clock generator; a signal frequency of 100 MHz can be used. See also chapter 7.4.

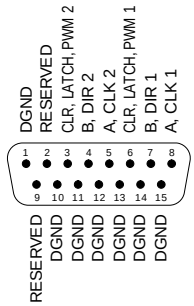
The counter value is written into a latch register if an edge–at one's option positive or negative–occurs at the selected input (A/CLK, B/DIR or CLR/LATCH). Latching can be triggered by software, too.

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

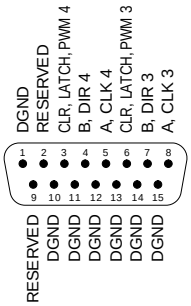
The counters are controlled by *ADbasic* instructions via control register (instructions see below).

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

In any case you have to set the input operation mode (single ended / differential) with **Cnt_SE_Diff**.

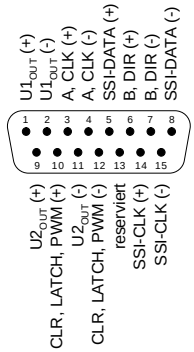


C01 & C02 (TTL)

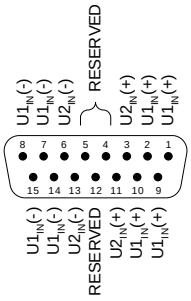


C03 & C04 (TTL)

Counters, operation mode TTL (single-ended)



Counters, operation mode differential / SSI
C01…C04



Counter power supply
C0 POWER IN

Fig. 14 – Pin assignment of CNT add-on

Although all inputs of the CNT 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 +5 V and the negative input to GND.

External power supply

Via the connector CO POWER IN, you can supply two voltages U_1 and U_2, which are then available at the pins U_{1out} and U_{2out} of the connectors CO1…CO4, e.g. for external incremental encoders. The maximum current at each of the output pins is 350mA.
Please note: All minus inputs V_{1in} (-) are galvanically connected to GND via a common line; the minus inputs V_{2in} (-) have such a common connection, too.

7.2 Counter Software

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

```
<ADwinGoldII.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 15, starting from page 111.

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

Fig. 15 – Instructions of Gold II-CNT counter add-on

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

Sequence of instructions

Please configure the counters according to the following order:

1. Disable specified counter (**Cnt_Enable**)
2. Set operating mode (**Cnt_Mode**, **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.

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.

7.2.1 Evaluation of the Counter Contents

The binary counters of the CNT 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 (2^{31}-1) follows the highest negative number (-2^{31}) and the lowest positive number (0) follows the highest negative number (-1).

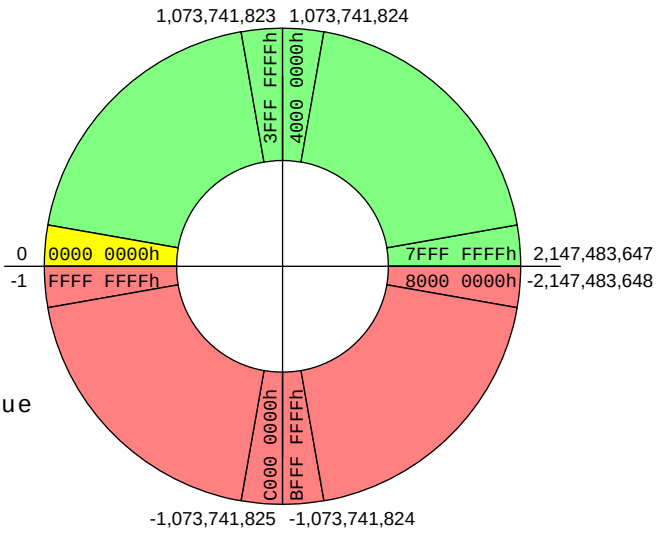


Fig. 16 – Circle for the interpretation of counter values

Circle

Please pay attention to the following rules for programming:

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

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.

Count direction

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

"Overflow"

7.3 Using Event Counter

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

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

– clear the counter (CLR)

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

7.3.1 Clock and Direction

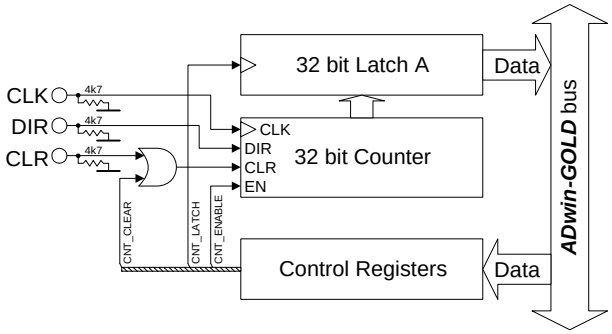


Fig. 17 – Block diagram CNT add-on in the mode "clock and direction"

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.

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

Programming example

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

7.3.2 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.

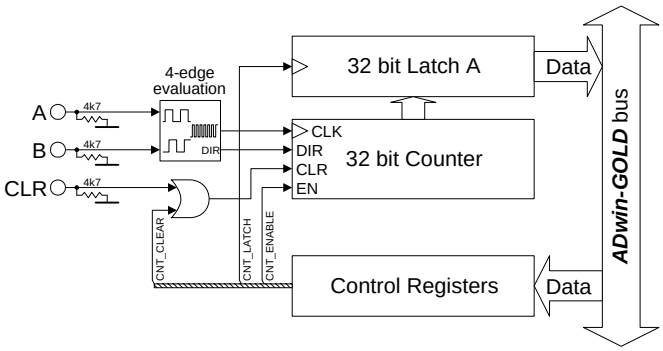


Fig. 18 – Block diagram CNT add-on in mode "four edge evaluation"

Please note:

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

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

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

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

Programming example

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

7.4 Using PWM Counter

Reference clock generator

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

Example

```
#Include ADwinGoldII.inc
#Define frequency FPAR_1
#Define dutycycle FPAR_2
#Define hightime PAR_1
#Define lowtime PAR_2
Init:
  ...
  Cnt_Enable(0)             'stop all counters
  Cnt_Clear(0001b)          'clear counter 1
  Rem set operation mode of counter 1:
  Rem Bits 0..5: no importance
  Rem Bit 6: detect rising edge as PWM signal
  Rem Bit 7: input B/DIR as PWM input
  Cnt_Mode(1,01000000b)
  Cnt_SE_Diff(1111b)        'all inputs differential
  Cnt_Enable(0001b)         'start counter 1
  ...
Event:
  ...
  Rem latch counter 1
  Cnt_PW_Latch(0001b)
  Rem read frequency and duty cycle
  Cnt_Get_PW(1,frequency,dutycycle)
  Rem read high time and low time
  Cnt_Get_PW_HL(1,hightime,lowtime)
```

Exception: evaluate PWM registers on your own

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

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

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

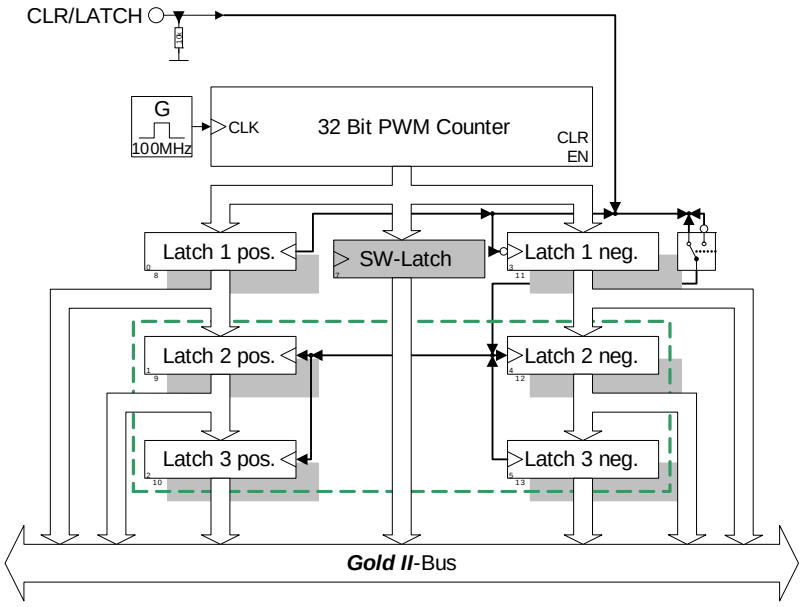
The register values are changed with any edge like this:

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

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

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

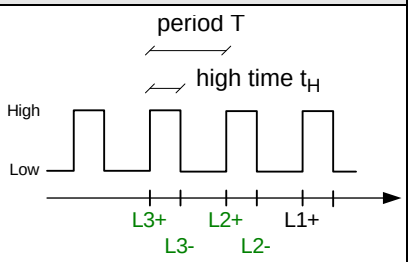
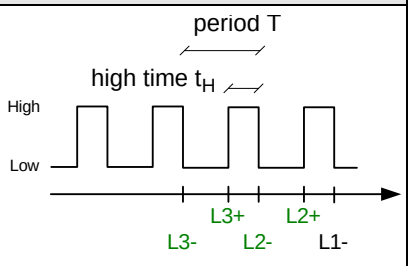
In addition, there is a single latch register where the counter value is copied by software (instruction **Cnt_PW_Latch**).



Example: Evaluation of PWM registers

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

The calculation depends on the set reference edge:

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

7.5 SSI decoder

An incremental encoder with SSI interface can be connected to each of the 4 decoders. The signals are differential and have RS422/485 levels.

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

The decoder connections are provided on the connectors CO1…CO4 (15 pins, DSub, see page 11), on the pins 5, 8, 14 and 15.

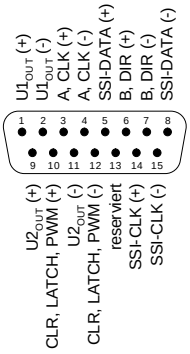


Fig. 19 – Pin assignment SSI decoders, CO1…CO4

Setting properties

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

– Clock rates: With **SSI_Set_Clock** clock rates of approx. 100 kHz up to 2.5 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. 20 – Listing: Conversion of Gray code into binary code

Programming

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

Range	Instructions
Initialize decoder	**SSI_Mode** **SSI_Set_Bits** **SSI_Set_Clock**
read encoder data	**SSI_Read** **SSI_Start** **SSI_Status**

The instructions are in the include file <ADwinGoldII.INC>. More information can be found in the online help.

7.6 PWM outputs

PWM outputs enable to output pulse-width-modulated signals with selectable duty cycle on 6 outputs. The output is clocked with 50 MHz.

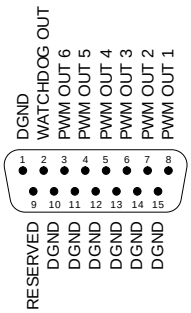


Fig. 21 – Pin assignments PWM outputs, PWM 1-8 (TTL)

Programming

The functionality of the PWM outputs is easily programmed with *ADbasic* instructions; description see chapter 15, starting from page 136.

Function	Instructions

Initialize PWM outputs
Set operation mode
	`PWM_Init`
	`PWM_Reset`
	`PWM_Standby_Value`

Start PWM output	`PWM_Enable`
Set PWM mode	`PWM_Write_Latch`
Read PWM mode and status	`PWM_Get_Status` `PWM_Latch`

The instructions The instructions are in the include file `<ADwinGoldII.INC>`. More information can be found in the online help.

8 CAN add-on

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

– 2 CAN interfaces (page 31)

  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 interfaces are located on the connectors CAN 1 and CAN 2.

– 2 RSxxx interfaces (page 34)

  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.

8.1 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.

8.1.1 Hardware Description

The connections of the interfaces 1 and 2 are located on the 9-pin DSUB connectors CAN 1 and CAN 2. The pin assignment is the same.

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

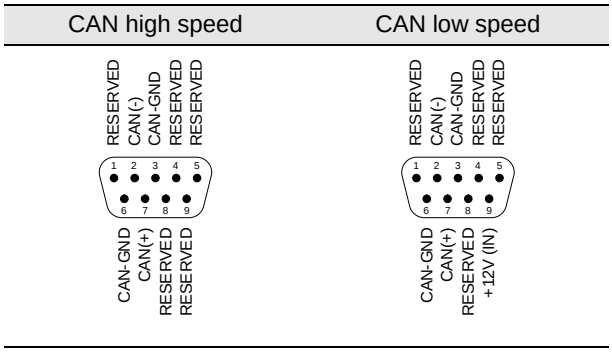


Fig. 22 – CAN: Pin assignments

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 12V DC. 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 120W 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 120W.

8.1.2 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 1MHz; with low speed CAN the max. frequency is 125kHz. 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_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**

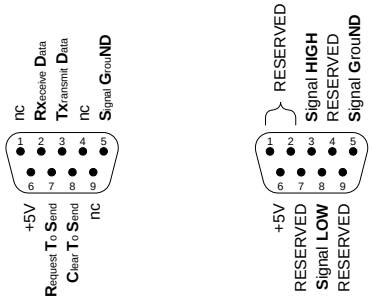
The instructions are in the include file <ADwinGoldII.INC>. More information can also be found in the ADbasic manual and the online help.

8.2 RSxxx Interface

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 and the last RS485 member). RS485 members, which are not positioned at the physical end of the bus, 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 300 Ω resistor.

8.2.1 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	
	dec.	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	
	dec.	hex.
19200	120	0078h
9600	240	00F0h
4800	480	01E0h
2400	960	03C0h
1200	1920	0780h
600	3840	0F00h
300	7680	1E00h

Fig. 23 – 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).

8.2.2 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	**Check_Shift_Reg**, **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 <ADwinGoldII.INC>. More information can be found starting from or 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
```

9 Profibus Add-on

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

Functions description

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

There is a range each for data input and data output; each range has a maximum size of 76 bytes. Please note, that the terms "input" and "output" are used as the fieldbus controller (slave) sees them.

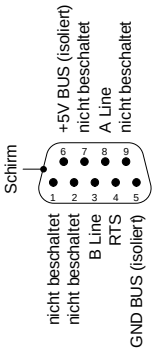
You set the number and length of input and output areas separately.

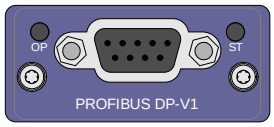
Hardware

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

The Profibus has to be terminated at its physical beginning and at the end of its segments by an active terminator.

If required, you have to add the terminator yourself at the appropriate data lines of the fieldbus node or use an appropriate connector with integrated terminator.





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

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

Fig. 24 – Profibus: Meaning of LEDs

Projecting the Profibus

You are projecting the Profibus with a configuration tool suitable for the bus master. The following process description uses a Profibus master of the Hilscher company and the appropriate program SyCon.

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

Copy the GSD file

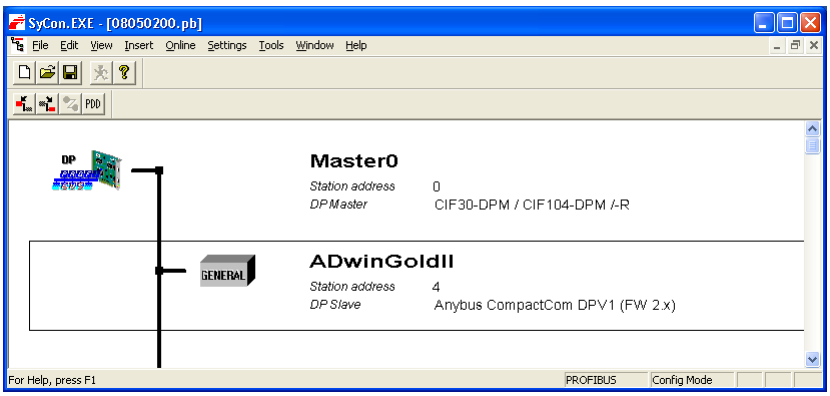
– Copy or import the GSD file hmsb1811.gsd of the fieldbus node from `C:\ADwin\Fieldbus\Profibus` into the source directory of the configuration tool.

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

Integrate the Slave

– In the configuration tool, add the Slave, i.e. the fieldbus node to the Profibus by selecting the GSD file `hmsb1811.gsd`. The station address must equal the address used for *ADbasic* initialization with **`Init_Profibus`**.

  Afterwards the bus could be structured as below:



Configure the Slave

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

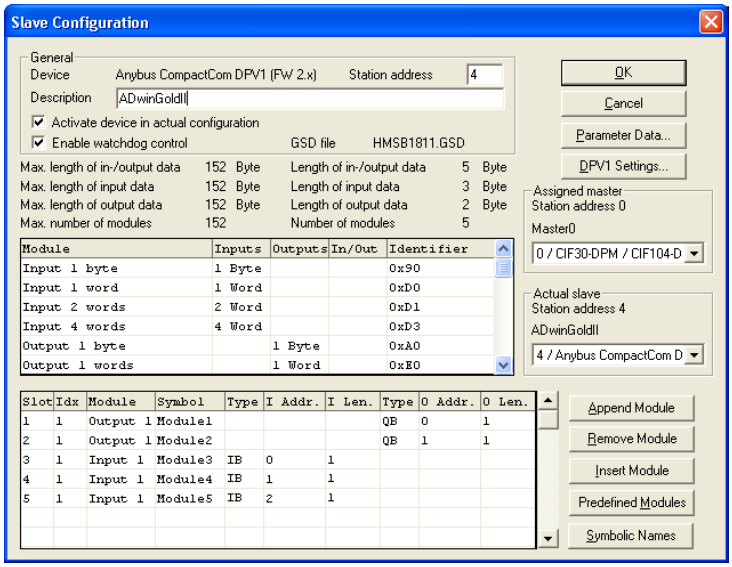
  Please note the following rules:

  - The terms "input" and "output" have reverse meanings in *ADbasic* (slave) and in the configuration tool (master).
  - Configure the outputs (as seen from the master) first and then the inputs. If there are inputs initialized in *ADbasic*, you have to configure outputs as correspondent in the configuration tool.
  - Number and length of data ranges must equal the data used for *ADbasic* initialization with **`Init_Profibus`**.
  - You may use only a single data length for each input data and output data. Input and output data can be set to a length of 1, 2, 4 or 8 Byte (2 byte = 1 word).

  The following example line in *ADbasic* configures the slave with 2 inputs of 1 byte and 3 outputs of 1 byte.

```
Par_31 = Init_Profibus(2, 2, 1, 3, 1, conf_Arr, Data_1)
```

  To configure the slave correctly in the configuration tool you have to set the 2 outputs first and the 3 inputs afterwards (1 byte each). The graphic below shows this example configuration:



Programming with *ADbasic*

The profibus interface is easily programmed with *ADbasic* instructions.

Area	Instructions
Initialize station address and data ranges	`Init_Profibus`
Read and write data	`Run_Profibus`

The instructions are described starting from page 173 or in the online help.

Initialization must be run with low priority since it takes some seconds; if it were a process with high priority, the PC interrupts the communication after a time (time-out). For the same reason, reading and writing data should be run with low priority.

Specifications

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

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

Operating modes of fieldbus node

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

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

Fig. 25 – Profibus: Operating modes

10 DeviceNet Add-on

The add-on Gold II-DeviceNet provides a fieldbus node with the functionality of a DeviceNet slave. All settings are done via software.

Functions description

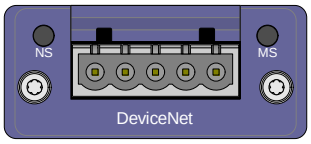
After power-on the fieldbus node must be initialized. The initialization determines the station address (slave node address) on the DeviceNet, the baudrate, and the size of the input and output areas.

There is a range each for data input and data output; each range has a maximum size of 255 bytes. Please note, that the terms "input" and "output" are used as the fieldbus controller (slave) sees them.

You set the number and length of input and output areas separately.

Hardware

The pin assignment of the DeviceNet connector refers to the specification of the Open DeviceNet Vendor Association (ODVA).



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

Left and right of the connector, there are two LEDs, which display the operation status of the node in the DeviceNet: network status (NP) and module status (MS; module stands for the node here).

LED	Status	Meaning
NS	off	Offline or no power.
	green	On-line, one or more connections are established.
	flashing green	On-line, no connections established.
	red	Error: Critical link failure.
	flashing red	Error: One or more connections timed-out.
	alternating red / green	Self test.
MS	off	No power.
	green	Operating in normal condition
	flashing green	Missing or incomplete configuration, device needs commissioning
	red	Error: Unrecoverable Fault(s).
	flashing red	Error: Recoverable Fault(s).
	alternating red / green	Self test.

Fig. 26 – DeviceNet: Meaning of LEDs

Projecting the DeviceNet

You are projecting the DeviceNet with a configuration tool suitable for the bus master. Look for the process description of bus projection in the documentation of the configuration tool.

For projecting, the file `324-8172-EDS_ABCC_DEV_V_2_3.eds` (electronic data sheet) is provided in the folder `C:\ADwin\Fieldbus\DeviceNet`.

Make sure that you use the same settings (station address, baudrate, number and length of input and output data) for projecting as you have done for initialization in *ADbasic* with the instruction **`Init_DeviceNet`**.

Please note, that the terms "input" and "output" may have reverse meanings in *ADbasic* (slave) and in the configuration tool (master). Also, the order of setting inputs and outputs may be of importance in the configuration tool.

Programming with *ADbasic*

The DeviceNet interface is easily programmed with *ADbasic* instructions.

Area	Instructions
Initialize station addresss, baudrate and data ranges	`Init_DeviceNet`
Read and write data	`Run_DeviceNet`

The instructions are described starting from page 178 or in the online help.

Initialization must be run with low priority since it takes some seconds; if it were a process with high priority, the PC interrupts the communication after a time (time-out). For the same reason, reading and writing data should be run with low priority.

11 EtherCAT Add-on

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

Functions description

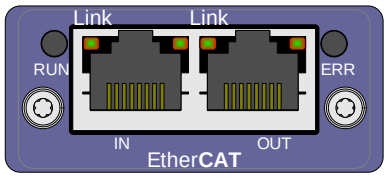
After power-on the fieldbus node must be initialized in *ADbasic*. The initialization determines the size of the input and output areas.

There is a range each for data input and data output; each range has a maximum size of 254 bytes. Please note, that the terms "input" and "output" are used as the fieldbus controller sees them.

You set the number and length of input and output areas separately.

Hardware

The interface has a plug connector of type RJ45 for both data input (IN) and data output (OUT). Each connector has a LED "Link / Activity" top left, which displays the operating status of the node in the EtherCAT bus. The two other LEDs have no function.



To the right and to the left of the connectors there are LEDs displaying the status of the EtherCAT state machine (RUN) and the occurrence of communication errors (ERR).

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

Fig. 27 – EtherCAT: Meaning of LEDs

If both LEDs RUN and ERR turn red, a serious error has occurred in the interface. Please inform the support of Jäger Messtechnik; you find the address on the inner side of the cover page of the manual.

Projecting the EtherCAT bus

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

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

– Copy the description files `*.XML` of the fieldbus node from `C:\ADwin\Fieldbus\EtherCAT` into the root directory of the configuration tool.

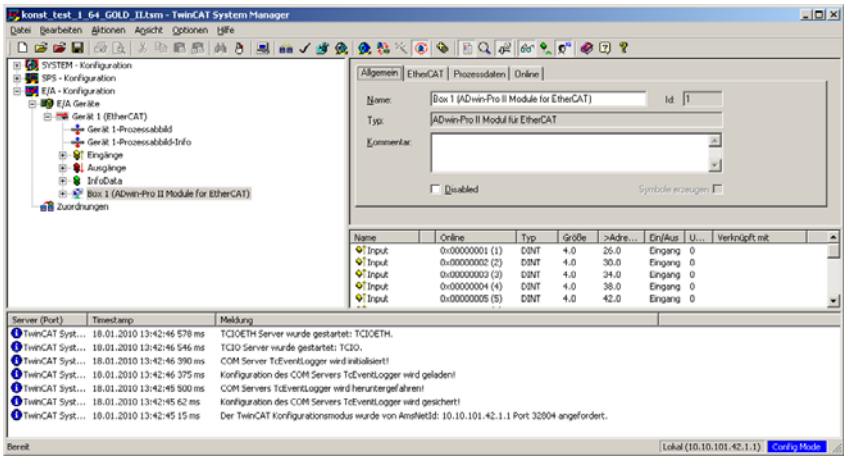
  Upon start-up, the configuration tool loads the required information about the new slave from the appropriate description file.

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

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

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

  Afterwards the bus could be structured as below:

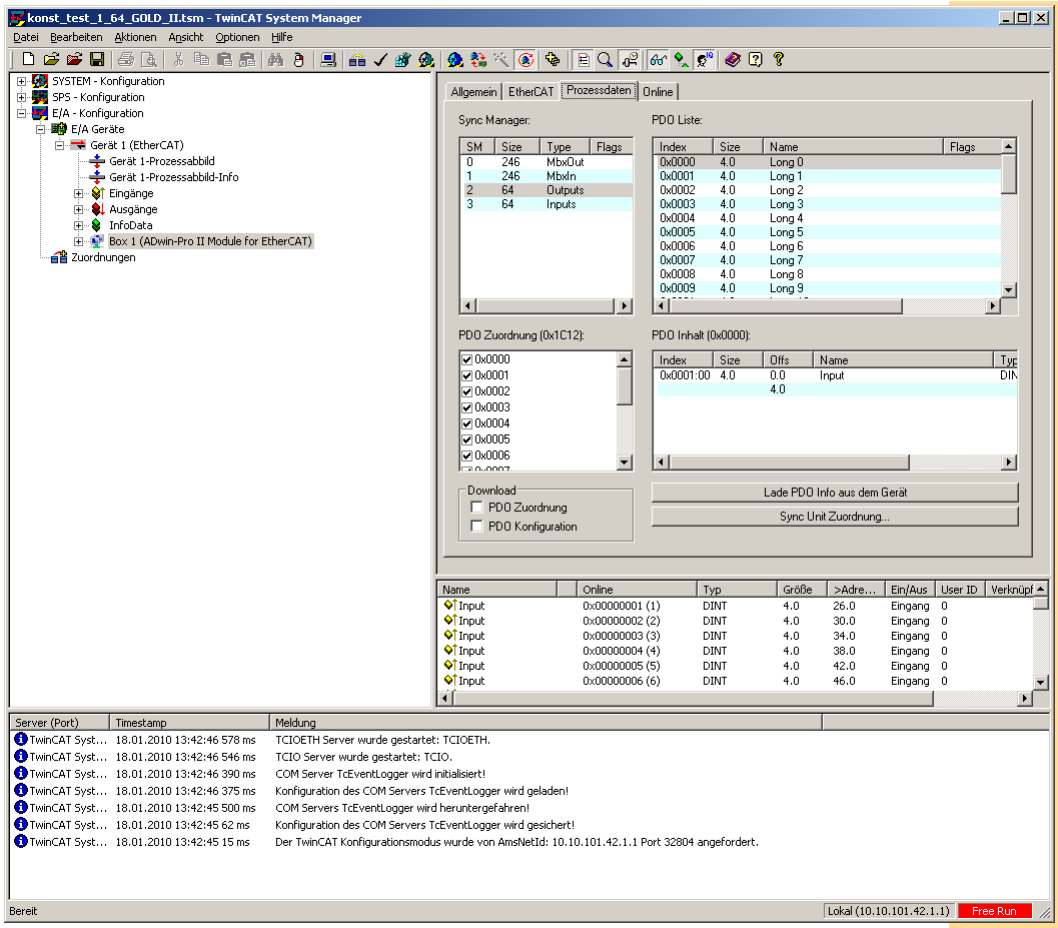


– Configure the *ADwin*-EtherCAT slave in an *ADbasic* program using the instruction **`ECAT_Init`**.

– Read the configuration into the configuration tool.

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

  Afterwards the slave configuration could be as shown as below: 2 ranges of 4 byte as inputs and 2 ranges of 4 byte as outputs.



Programming with *ADbasic*

The fieldbus node is easily programmed with *ADbasic* instructions.

Area	Instructions
Initialize station address and data ranges	**ECAT_Init**
Read and write data	**ECAT_Run**

The instructions are described starting from page 182 or in the online help.

Initialization must be run with low priority since it takes some seconds; if it were a process with high priority, the PC interrupts the communication after a time (time-out). For the same reason, reading and writing data should be run with low priority.

Specifications

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

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

Operating modes of EtherCAT node

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

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

Fig. 28 – EtherCAT: Operating modes

Storage-16 Add-on

12 Storage-16 Add-on

The add-on *Gold II-Storage-16* contains a memory card with 16 GiB storage volume and a battery-buffered real-time clock.

Memory card

You can access the memory card from *ADbasic* or *TiCoBasic*. A simultaneous access from both *ADbasic* and *TiCoBasic* is not allowed.

The memory card must always be initialized in *ADbasic*, even if all other accesses are done in *TiCoBasic*.

Initialization must be run with low priority since it takes some seconds; if it were a process with high priority, the PC interrupts the communication after a time (time-out).

Data may be read or written. Data is always stored in blocks of 128 values on the memory card. As with initialization, reading and writing of data should be done in low priority.

The data transfer rate of a read / write process increases with the amount of transferred data blocks. In any case the optimum data rate is reached with erased memory card (after running **Media_Erase**).

Real-time clock

The add-on provides a real-time clock which can be used to assign a specified time to measurements. With simple instructions, date and time are set and read out.

The time must be specified by a valid date and time of day; it has a resolution of one second. Leap years are considered.

The clock is battery-backed (type CR1632) and can remain up to 2 years without any external power supply i. e. when the system is turned off. To replace the buffer battery send *ADwin-Gold II* to the address on the inner side of the cover page.

Programming in *ADbasic* und *TiCoBasic*

The memory card is easily programmed with the following instructions. Please note that some instructions are only available in *ADbasic* but not in *TiCoBasic*:

Area	Instructions	Available in
Initialize memory card	**Media_Init**	*ADbasic*
Delete all data from memory card	**Media_Erase**	*ADbasic*
Read data	**Media_Read**	*ADbasic* *TiCoBasic*
Write data	**Media_Write**	*ADbasic* *TiCoBasic*
Read real-time clock	**RTC_Get**	*ADbasic* *TiCoBasic*
Set real-time clock	**RTC_Set**	*ADbasic*

The instructions are described starting from page 187 or in the online help.

13 ADwin-Gold II-Boot

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

With ADwin-Gold II-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.

Disable boot loader

If you do not wish to work with the boot loader option:

– 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 <ADwin11.btl> the system will only be booted after power-up, but no processes can be executed.

With the installation of the *ADwin*-Software from the supplied *ADwin* CD-ROM, the utility program for the boot loader (ADethflash) is automatically copied.

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 Installation" manual.

16000 values you can freely dispose of

In combination with the Ethernet interface and boot loader you can write or read out 16000 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".

14 Accessories

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

– *ADwin-Gold II-pow*: external 12 V power supply unit
– various lengths of power supply and USB or Ethernet cable
– cable connector for an external power supply
– mounting kits for enclosures

ADwin-Gold II-pow

On the secondary side ADwin-Gold II-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 II.

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").

15 Software

You are programming *ADwin-Gold II*- 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:

– page 51: System functions
– page 59: Analog Inputs and Outputs
– page 87: Digital Inputs and Outputs
– page 110: Counters
– page 129: SSI interface
– page 136: PWM Outputs
– page 145: CAN interface
– page 160: RSxxx interface
– page 173: Profibus interface
– page 177: DeviceNet interface
– page 182: EtherCAT interface
– page 186: Real-time clock
– page 189: Storage media (ADbasic)

– Instructions for LS bus modules like HSM24V are described in a separate manual or in the online help.

The *TiCo* processor of the *ADwin-Gold II* system can also access the inputs, outputs, and interfaces. The description of *ADbasic* instructions for T11 are therefore also valid for *TiCoBasic*. Please note, that only one of the processors *TiCo* oder T11 can access the I/Os at the same time.

Symbols in each description (see right) show, which processor (T11, *TiCo*) can use the instruction.

T11 TiCo

Please note:

– There are 2 different include files:

  ADbasic:`ADwinGoldII.inc`; *TiCoBasic*: `GoldIITiCo.inc`.

– The *TiCo* processor runs with 50 MHz clock speed (unit of Processdelay: 20 ns), while T11 runs with 300 MHz (unit of Processdelay: 3,3 ns).

Online help

All descriptions are also found in the online help of the development environments *ADbasic* and *TiCoBasic*. To show a description, move the cursor on a keyword and hit the [F1] key.

15.1 System functions

This section describes the following instructions:

This section contains instructions für system functions of *ADwin-Gold II*.

Event_Config

T11 TiCo

Event_Config configures the external event input.

Syntax

```
#INCLUDE ADwinGoldII.inc / GoldIITiCo.inc
Event_Config(min_hold,edge,prescale)
```

Parameters

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

Notes

- / -

See also

Event_Enable

Valid for

Gold II

Example

```
Rem Select the appropriate include file for ADbasic / TiCoBasic
#INCLUDE ADwinGoldII.inc / GoldIITiCo.inc

Init:
  Rem Configure event input for
  Rem minimum time 15 ns, neg. edge, event signal after 4 edges
  Event_Config(0,2,4)
```

Event_Enable

T11

Event_Enable selects the event source.

Syntax

```
#INCLUDE ADwinGoldII.inc
Event_Enable(pattern)
```

Parameters

pattern	Bit pattern, where the set bit selects the event source: Bit 0: CAN interface 1. Bit 1: CAN interface 2. Bit 3: TiCo processr with instruction **Trigger_Event**. Bit 4: Event input at the DSub socket DIO 00-15 (IN, see page 15); default. All other bits are reserved.	LONG

Notes

Only one of the bits should be set.

The instructions **Event_Enable** and **En_CAN_Interrupt** may not be used in parallel, because both instructions set the used event source.

See also

Event_Config, En_CAN_Interrupt

Valid for

Gold II

Example

```
#INCLUDE ADwinGoldII.inc

Init:
  Rem Set CAN interface 2 as event source
  Event_Enable(00010b)
```

Set_LED

T11 TiCo

Set_LED switches the LED (besides the power switch) on or off.

Syntax

```
#INCLUDE ADwinGoldII.inc / GoldIITiCo.inc
Set_LED(value)
```

Parameters

value	Status of the LED. 0: off. 1: on, green light. 2: ein, red light.	LONG

Notes

The LED is normally switched on and off by process 15. To use **Set_LED** the process should therefore be stopped.

See also

- / -

Valid for

Gold II

Example

```
Rem Select the appropriate include file for ADbasic / TiCoBasic
#INCLUDE ADwinGoldII.inc / GoldIITiCo.inc
Init:
  Stop_Process(15)
  Set_LED(1)                'set green LED

Event:
  Rem …

Finish:
  Set_LED(0)                'switch off LED
  Start_Process(15)
```

Watchdog_Init

T11 TiCo

Watchdog_Init configures and activates the watchdog counter.

Syntax

```
#INCLUDE ADwinGoldII.inc / GoldIITiCo.inc
Watchdog_Init(enable, time, mode)
```

Parameters

enable	Operating mode of watchdog counter: 0: disabled (default). 1: enabled.	LONG
time	Starting value (1…0FFFEh) of watchdog counter in units of 10µs.	LONG
mode	Bit pattern (bits 0…3) to set the function of the watchdog counter. The bit function is: Bit 0: Stop T11 (*ADwin* CPU). Bit 1: Stop *TiCo* processor. Bit 2: Set output Watchdog Out to TTL level low. Bit 3: Set outputs OUT1…OUT8 and DIO00…DIO32 to TTL level low.	LONG

Notes

As soon as the watchdog counter is enabled, it decrements the counter value continuously. When the counter value reaches 0 (zero) the system assumes a malfunction and executes the functions being set by mode.

Set the active watchdog timer at least once to the start value within the counting time, in order to keep your system working (see **Watchdog_Reset**).

If bit 2 is set (and the counter enabled), the output Watchdog Out is set to the watchdog status: TTL level high = watchdog counting, TTL level low = watchdog stopped, counter value 0 is reached.
If bit 2 is not set or the watchdog counter is disabled, the output Watchdog Out can be set with **Watchdog_Standby_Value**.

See also

Watchdog_Reset, Watchdog_Standby_Value, Watchdog_Status

Valid for

Gold II

Example

```
Rem Select the appropriate include file for ADbasic / TiCoBasic
#INCLUDE ADwinGoldII.inc / GoldIITiCo.inc
Init:
  Watchdog_Init(1,0FFFEh,1111b)'enable and configure watchdog

Event:
  Watchdog_Reset()          'reset watchdog regularly
  Rem …
```

Watchdog_Reset

T11 TiCo

Watchdog_Reset resets the watchdog counter to the start value. The counter remains enabled.

Syntax

```
#INCLUDE ADwinGoldII.inc / GoldIITiCo.inc
Watchdog_Reset()
```

Parameters

- / -

Notes

As soon as the watchdog counter is enabled, it decrements the counter values continuously. When the counter value reaches 0 (zero) the system assumes a malfunction and and executes the functions being set by **Watchdog_Init**.

Set the active watchdog timer at least once to the start value (see **Watchdog_Init**) within the counting time, in order to keep your system working. If the watchdog counter is disabled, **Watchdog_Reset** has no function.

See also

Watchdog_Init, Watchdog_Standby_Value, Watchdog_Status

Valid for

Gold II

Example

```
Rem Select the appropriate include file for ADbasic / TiCoBasic
#INCLUDE ADwinGoldII.inc / GoldIITiCo.inc

Init:
  Watchdog_Init(1,0FFFEh,1111b)'enable and configure watchdog

Event:
  Watchdog_Reset()          'reset watchdog regularly
  Rem …

Finish:
  Watchdog_Init(0,0,0)      'disable watchdog
```

Watchdog_ Standby_Value

T11 TiCo

Watchdog_Standby_Value sets the output Watchdog Out to the selected TTL level.

Syntax

```
#INCLUDE ADwinGoldII.inc / GoldIITiCo.inc
Watchdog_Standby_Value(level)
```

Parameters

level	TTL level of the output Watchdog Out: 0: TTL level low. 1: TTL level high.	LONG

Notes

The ouput can only be set with **Watchdog_Standby_Value** if the watchdog counter itself does not set the output or is disabled (see **Watchdog_Init**).

After power-up the output Watchdog Out is set to TTL level Low.

See also

Watchdog_Init, Watchdog_Reset, Watchdog_Status

Valid for

Gold II

Example

```
Rem Select the appropriate include file for ADbasic / TiCoBasic
#INCLUDE ADwinGoldII.inc / GoldIITiCo.inc

Init:
  Watchdog_Init(1,0FFFEh,0001b)'enable and configure watchdog
  Watchdog_Standby_Value(1)'set watchdog value to level high

Event:
  Watchdog_Reset()          'reset watchdog regularly
  Rem …
```

Watchdog_Status

T11 TiCo

Watchdog_Status returns the status of the watchdog counter.

Syntax

```
#INCLUDE ADwinGoldII.inc / GoldIITiCo.inc
ret_val = Watchdog_Status()
```

Parameters

ret_val	Bit pattern with the status of the watchdog counter. Bit 0: Operating mode. Bit 0 = 0: watchdog counter is disabled. Bit 0 = 1: watchdog counter is enabled. Bit 1: detection of malfunction. Bit 1 = 0: no malfunction, watchdog is counting. Bit 1 = 1: malfunction, watchdog has reached zero (0). Other bits have no function.	LONG

Notes

As soon as the watchdog counter is enabled, it decrements the counter values continuously. When the counter value reaches 0 (zero) the system assumes a malfunction and and executes the functions being set by **Watchdog_Init**.

If the watchdog counter is disabled (bit 0 = 0), bit 1 has no function.

See also

Watchdog_Init, Watchdog_Reset, Watchdog_Standby_Value

Valid for

Gold II

Example

- / -

15.2 Analog Inputs and Outputs

This section describes instructions to access analog inputs and outputs on *ADwin-Gold II*:

DAC

T11 TiCo

DAC outputs a defined voltage on a specified analog output.

Syntax

```
#INCLUDE ADwinGoldII.inc / GoldIITiCo.inc
DAC(dac_no,value)
```

Parameters

dac_no	Number of the analog output: Gold II: 1…2 Gold II-DA4: 1…4 Gold II-DA8: 1…8	LONG
value	Value in digits, which defines the voltage to be output (0…65535).	LONG

Notes

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

See also

Start_DAC, Write_DAC

Valid for

Gold II, Gold II-DA4, Gold II-DA8

Example

```
Rem Please select the appropriate include for ADbasic /
 TiCoBasic
#Include ADwinGoldII.inc / GoldIITiCo.inc
Rem Digital P-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
```

Start_DAC

T11 TiCo

Start_DAC starts the conversion or the output of all DAC.

Syntax

```
#INCLUDE ADwinGoldII.inc / GoldIITiCo.inc
Start_DAC()
```

Parameters

- / -

Notes

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

See also

DAC, Write_DAC

Valid for

Gold II, Gold II-DA4, Gold II-DA8

Example

```
REM Simultaneous output of two different signal waveforms
REM on outputs DAC 1 and 2.
Rem Please select the appropriate include for ADbasic / TiCoBasic
#Include ADwinGoldII.inc / GoldIITiCo.inc
Dim i As Long

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

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

Write_DAC

T11 TiCo

Write_DAC writes a digital value into the output register of a DAC. The conversion into output voltage is started by **Start_DAC**.

Syntax

```
#INCLUDE ADwinGoldII.inc / GoldIITiCo.inc
Write_DAC(dac_no,value)
```

Parameters

dac_no	Number of the analog output: Gold II: 1…2 Gold II-DA4: 1…4 Gold II-DA8: 1…8	LONG
value	Value in digits, which defines the voltage to be output (0…65535).	LONG

Notes

Start_DAC starts the conversion of the register value into an output voltage.

See also

DAC, Start_DAC

Valid for

Gold II, Gold II-DA4, Gold II-DA8

Example

```
REM Simultaneous output of four different signal waveforms
REM on outputs DAC 1, 2, 3, 4 (Gold II-DA4).
REM The signal waveforms are stored in four DATA arrays and
REM can be filled before start of program from the PC.
Rem Please select the appropriate include for ADbasic / TiCoBasic
#Include ADwinGoldII.inc / GoldIITiCo.inc
Dim i As Long                 'Declaration
Dim Data_1[1000], Data_2[1000], Data_3[1000] As Long
Dim Data_4[1000] As Long

Init:
  Processdelay = 10000
  i=1
  Write_DAC(1,Data_1[i])      'Set output register DAC1
  Write_DAC(2,Data_2[i])      'Set output register DAC2
  Write_DAC(3,Data_3[i])      'Set output register DAC3
  Write_DAC(4,Data_4[i])      'Set output register DAC4

Event:
  Start_DAC()                 'Start ouput of all DAC
  Write_DAC(1,Data_1[i])      'Set output register DAC1
  Write_DAC(2,Data_2[i])      'Set output register DAC2
  Write_DAC(3,Data_3[i])      'Set output register DAC3
  Write_DAC(4,Data_4[i])      'Set output register DAC4
  Inc(i)
  If (i>1000) Then i=1
```

ADC

T11 TiCo

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

Syntax

```
#INCLUDE ADwinGoldII.inc / GoldIITiCo.inc
ret_val = ADC(channel)
```

Parameters

channel	Number (1…16) of the analog input channel.	LONG
ret_val	Measurement value in digits (0…65535).	LONG

Notes

ADC24 returns digital values with 24 bit resolution.

ADC is a combination of consecutive functions:

- **Set_Mux1**, **Set_Mux2**: Set the multiplexer to the specified input channel.
- Wait for settling of the multiplexer.
- **Start_Conv**: Start measurement: Convert analog signal to a digital value.
- **Wait_EOC**: Wait for the end of conversion.
- **Read_ADC**: Read out digital value from the register and return it.

Multiplexer settling time and conversion time are given on page 17.

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

In the following examples the instructions **Set_Mux1**/**2**, **Start_Conv**, **Wait_EOC** and **Read_ADC** 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.

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 measurement range is 20 Volt here (input voltage range: -10V…10V).

See also

ADC24, Read_ADC, Set_Mux1, Set_Mux2, Start_Conv, Wait_EOC

Valid for

Gold II

Example

```
Rem Please select the appropriate include for ADbasic /
 TiCoBasic
#Include ADwinGoldII.inc 'for ADbasic
Rem #Include GoldIITiCo.inc     'for TiCoBasic
Dim iw As Long                'Declaration

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

ADC24

T11 TiCo

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

Syntax

```
#INCLUDE ADwinGoldII.inc / GoldIITiCo.inc
ret_val = ADC24(channel)
```

Parameters

channel	number of the analog input channel (1…16).	LONG
ret_val	measurement result in digits: (0…16,777,215 = 2^{24}-1).	LONG

Notes

ADC returns measurement values with 16 bit resolution.

The return value of **ADC24** contains an 18 bit measurement value in bits 6…23; bits 0…5 are always zero (see also page 14).

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

ADC24 is a combination of consecutive functions:

- **Set_Mux1**, **Set_Mux2**: 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.
- **Read_ADC24**: Read out digital value from the register and return it.

Multiplexer settling time and conversion time are given on page 17.

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

In the following examples you should use the instructions **Set_Mux1**/**2**, **Start_Conv**, **Wait_EOC** and **READADC24** instead of **ADC24** 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.

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

$$\text{Voltage} = (\text{Digits} - 8388608_{\text{bipolar}}) \cdot \frac{\text{measurement range}}{16777216}$$

The measurement range is 20 Volt here (input voltage range: -10V…10V).

See also

ADC, Read_ADC24, Set_Mux1, Set_Mux2, Start_Conv, Wait_EOC

Valid for

Gold II

Example

```
Rem Please select the appropriate include for ADbasic /
 TiCoBasic
#Include ADwinGoldII.inc 'for ADbasic
Rem #Include GoldIITiCo.inc     'for TiCoBasic
Dim iw As Long                'Declaration

Event:
  'Measure analog input 1
  iw = ADC24(1)
  'Write measurement value into global variable so that
  'the computer can read it.
  Par_1 = iw
```

Read_ADC

T11 TiCo

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

Syntax

```
#INCLUDE ADwinGoldII.inc / GoldIITiCo.inc
ret_val = Read_ADC(adc_no)
```

Parameters

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

Notes

Read_ADC24 returns a converted value with 12-bit resolution.

See also

ADC, Read_ADC24, Set_Mux1, Set_Mux2, Start_Conv, Wait_EOC

Valid for

Gold II

Example

```
Rem Please select the appropriate include for ADbasic /
 TiCoBasic
#Include ADwinGoldII.inc 'for ADbasic
Rem #Include GoldIITiCo.inc     'for TiCoBasic
Event:
  'Set multiplexer: ADC1 to channel 3, ADC2 to channel 4
  Set_Mux1(001b)
  Set_Mux2(001b)
  Rem interrupt IO access for MUX settling time
  IO_Sleep(200)
  Rem use waiting time, except for access to IOs or external
   memory
  Rem …
  Start_Conv(11b)              'Start conversion for both ADCs
  Wait_EOC(11b)                'Wait for end of conversion
  Par_1 = Read_ADC(1)          'Read value of ADC1
  Par_2 = Read_ADC(2)          'Read value of ADC2
```

Read_ADC24

T11 TiCo

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

Syntax

```
#INCLUDE ADwinGoldII.inc / GoldIITiCo.inc
ret_val = Read_ADC24(adc_no)
```

Parameters

adc_no	Number (1, 2) of the converter to read.	LONG
ret_val	Measurement value in digits (0…16,777,215 = 2^{24}-1), which corresponds to the voltage at the converter's input.	LONG

Notes

Read_ADC returns a converted value with 16-bit resolution.

See also

ADC24, Read_ADC, Set_Mux1, Set_Mux2, Start_Conv, Wait_EOC

Valid for

Gold II

Example

```
Rem Please select the appropriate include for ADbasic /
 TiCoBasic
#Include ADwinGoldII.inc 'for ADbasic
Rem #Include GoldIITiCo.inc     'for TiCoBasic
Dim val1, val2 As Long

Event:
  'Set multiplexer: ADC12-1 to channel 3, ADC12-2 to channel 4
  Set_Mux1(001b)
  Set_Mux2(001b)
  Rem interrupt IO access for MUX settling time
  IO_Sleep(200)
  Rem use waiting time, except for access to IOs or external
   memory
  Rem …
  Start_Conv(11b)              'Start conversion for both ADCs
  Wait_EOC(11b)                'Wait for end of conversion
  val1 = Read_ADC24(1)         'Read value of ADC1
  val2 = Read_ADC24(2)         'Read value of ADC2
```

Set_Mux1

T11 TiCo

Set_Mux1 sets the ADC1 multiplexer to the specified measurement channel and to the specified gain.

Syntax

```
#INCLUDE ADwinGoldII.inc / GoldIITiCo.inc
Set_Mux1(pattern)
```

Parameters

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

Bit no.	31…5	4 3	2 1 0
	–	PGA 1	MUX 1

PGA 1 — Bits 3…4 determine the gain factor:

00: factor 1	10: factor 4
01: factor 2	11: factor 8

MUX 1 — Bits 0…2 determine the channel to which the multiplexer is set:

000: channel 1	100: channel 9
001: channel 3	101: channel 11
010: channel 5	110: channel 13
011: channel 7	111: 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 17.

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, ADC24, Read_ADC, Read_ADC24, Set_Mux2, Start_Conv, Wait_EOC

Valid for

Gold II

Example

To set the multiplexer of ADC1 to channel 5 and to gain 8, you need the bit pattern: **11010b** (decimal: **26**).

```
Rem Please select the appropriate include for ADbasic /
 TiCoBasic
#Include ADwinGoldII.inc 'for ADbasic
Rem #Include GoldIITiCo.inc     'for TiCoBasic
Dim val As Long

Event:
 Set_Mux1(11010b)'Set multiplexer 1: channel and gain
 Rem interrupt IO access for MUX settling time
 IO_Sleep(200)
 Rem use waiting time, except for access to IOs or external
  memory
 Rem …
 Start_Conv(1)                'Start AD-conversion ADC1
 Wait_EOC(1)                  'Wait for end of conversion of
                              'ADC1
 val = Read_ADC(1)            'Read value of ADC1
```

Set_Mux2

T11 TiCo

Set_Mux2 sets the ADC2 multiplexer to the specified measurement channel and to the specified gain.

Syntax

```
#INCLUDE ADwinGoldII.inc / GoldIITiCo.inc
Set_Mux2(pattern)
```

Parameters

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

Bit no.	31…5	4 3	2 1 0
	–	PGA 2	MUX 2

PGA 2 — Bits 3…4 determine the gain factor:

00: factor 1
01: factor 2
10: factor 4
11: factor 8

MUX 2 — Bits 0…2 determine the channel to which the multiplexer is set:

000: channel 2
001: channel 4
010: channel 6
011: channel 8
100: channel 10
101: channel 12
110: channel 14
111: channel 16

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 17.

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, ADC24, Read_ADC, Read_ADC24, Set_Mux1, Start_Conv, Wait_EOC

Valid for

Gold II

Example

To set the multiplexer of ADC2 to channel 10 and gain 2, you need the bit pattern: **01100b** (decimal: **12**).

```
Rem Please select the appropriate include for ADbasic /
 TiCoBasic
#Include ADwinGoldII.inc 'for ADbasic
Rem #Include GoldIITiCo.inc     'for TiCoBasic
Dim val As Long

Event:
 Set_Mux2(01100b)'Set multiplexer 2: channel and gain
 Rem interrupt IO access for MUX settling time
 IO_Sleep(200)
 Rem use waiting time, except for access to IOs or external
  memory
 Rem …
 Start_Conv(2)                'Start AD-conversion ADC2
 Wait_EOC(2)                  'Wait for end of conversion of
                              'ADC2
 val = Read_ADC(2)            'Read value of ADC2
```

Start_Conv

T11 TiCo

Start_Conv can start the conversion of one or more A/D converters.

Syntax

```
#INCLUDE ADwinGoldII.inc / GoldIITiCo.inc
Start_Conv(pattern)
```

Parameters

pattern	Bit pattern that specifies which converters should be started (only bits 0…2 can be used): 1: start conversion. 0: do not start conversion.	LONG

Bit no.	1	0
ADC1	–	x
ADC2	x	–

Notes

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, ADC24, Read_ADC, Read_ADC24, Set_Mux1, Set_Mux2, Wait_EOC

Valid for

Gold II

Example

```
Rem Please select the appropriate include for ADbasic /
 TiCoBasic
#Include ADwinGoldII.inc 'for ADbasic
Rem #Include GoldIITiCo.inc     'for TiCoBasic
Dim val1 As Long

Event:
 Set_Mux1(0)                'Set multiplexer 1 to channel 1
 Rem interrupt IO access for MUX settling time
 IO_Sleep(200)
 Rem use waiting time, except for access to IOs or external
  memory
 Rem …
 Start_Conv(1)              'Start ADC1 A/D-conversion
 Wait_EOC(1)                'Wait for end of conversion
 val1 = Read_ADC(1)         'Read out value
```

Multiplexer settling time and conversion time are given on page 17.

Wait_EOC

T11 TiCo

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

Syntax

```
#INCLUDE ADwinGoldII.inc / GoldIITiCo.inc
Wait_EOC(pattern)
```

Parameters

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

Bit no.	1	0
ADC1	–	x
ADC2	x	–

Notes

- / -

See also

ADC, ADC24, Read_ADC, Read_ADC24, Set_Mux1, Set_Mux2, Start_Conv

Valid for

Gold II

Example

```
Rem Please select the appropriate include for ADbasic /
 TiCoBasic
#Include ADwinGoldII.inc 'for ADbasic
Rem #Include GoldIITiCo.inc     'for TiCoBasic
Dim val As Long

Event:
  Set_Mux2(001b)              'Set MUX 2 to channel 4
  Rem interrupt IO access for MUX settling time
  IO_Sleep(200)
  Rem use waiting time, except for access to IOs or external
   memory
  Rem …
  Start_Conv(10b)             'Start A/D-conversion ADC2
  Wait_EOC(10b)               'Wait for end of conversion at
                              ''ADC2
  val = Read_ADC(2)           'Read out value
```

Multiplexer settling time and conversion time are given on page 17.

The conversion time of the ADC is 2µs.

Seq_Mode

T11 TiCo

Seq_Mode initializes operation mode for the sequential control of an ADC.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
Seq_Mode(adc_no, mode)
```

Parameters

adc_no	Number (1, 2) of the ADC.	LONG
mode	Operating mode of the sequential control: 0: Single conversion (default), no sequential control. 1: Mode "single shot", single conversion cycle. 2: Mode "continuous" continuous conversion.	LONG

Notes

After power-up mode 0 is active.

Modes 1 and 2 activate the sequential control of the selected ADC, single conversions with **ADC** are disabled then (on this ADC). The sequential control consecutively runs a conversion on several channels. The channels are selected by **Seq_Select**.

The modes differ in the following items:

Mode	Kind of conversion
0 Standard:	Single conversion at one channel without sequential control, see **ADC**.
1 single shot:	The sequential control is started by **Seq_Start**; it ends as soon as each of the selected channels is converted once. The measurement values of the sequential control are read with **Seq_Read**.
2 continuous:	The sequential control converts the selected channels continuously, providing new measurement values all the time. That is, conversion and process cycle run non-synchronously. The conversion is started with **Seq_Start**. Inside a process cycle **Seq_Read** just reads the newest measurement value.

See also

ADC, Seq_Read, Seq_Read8, Seq_Read16, Seq_Set_Delay, Seq_Set_Gain, Seq_Select, Seq_Start, Seq_Status

Valid for

Gold II

Example

```
Rem Please select the appropriate include for ADbasic / TiCoBasic
#Include ADwinGoldII.inc / GoldIITiCo.inc
Dim Data_1[16] As Long At DM_Local
Dim i As Long

Init:
  Rem settings for sequential control of ADC 2
  Seq_Mode(2,2)             'continuous mode
  Seq_Set_Delay(2,125)      'waiting time 4.5 µs (125*20ns + 2µs)
  Seq_Set_Gain(2,0)         'gain factor 1
  Rem select all channels. selection is valid only for active
  Rem sequential control 2 = even numbered channels.
  Seq_Select(0FFFFh)
  Rem start sequential control of ADC2
  Seq_Start(10b)

Event:
  Rem read current values of even channels
  For i = 2 To 16 Step 2
    Data_1[i] = Seq_Read(i)
  Next i

Finish:
  Seq_Mode(2,0)             'reset to standard mode
```

Seq_Read

T11 TiCo

Seq_Read returns the current measurement value (16 bit) of a selected input channel of the sequential control.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
ret_val = Seq_Read(channel)
```

Parameters

channel	Input channel (1…16).	LONG
ret_val	Measurement value (16 bit).	LONG

Notes

The selection of input channels is useful only, if the sequential control has been enabled with **Seq_Mode** and **Seq_Start** and if the input channel has been selected with **Seq_Select**.

See also

Seq_Mode, Seq_Read8, Seq_Read16, Seq_Set_Delay, Seq_Set_Gain, Seq_Select, Seq_Start, Seq_Status

Valid for

Gold II

Example

```
Rem Please select the appropriate include for ADbasic / TiCoBasic
#Include ADwinGoldII.inc / GoldIITiCo.inc
Dim Data_1[16] As Long At DM_Local
Dim i As Long

Init:
  Rem settings for sequential control of ADC 2
  Seq_Mode(2,2)             'continuous mode
  Seq_Set_Delay(2,125)      'waiting time 4.5 µs (125*20ns + 2µs)
  Seq_Set_Gain(2,0)         'gain factor 1
  Rem select all channels. selection is valid only for active
  Rem sequential control 2 = even numbered channels.
  Seq_Select(0FFFFh)
  Rem start sequential control of ADC2
  Seq_Start(10b)

Event:
  Rem read current values of even channels
  For i = 2 To 16 Step 2
    Data_1[i] = Seq_Read(i)
  Next i

Finish:
  Seq_Mode(2,0)             'reset to standard mode
```

Seq_Read8

T11

Seq_Read8 returns the measurement values (16 bit) of the input channels 1…8 of the sequential control.

Syntax

```
#Include ADwinGoldII.inc
Seq_Read8(array[], array_idx)
```

Parameters

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

Notes

The selection of input channels is useful only, if the sequential control has been enabled with **Seq_Mode** and **Seq_Start**, and only for input channels which have been selected with **Seq_Select**.

If less than 4 input channels are o be read, a loop with **Seq_Read** is faster than **Seq_Read8**.

See also

Seq_Mode, Seq_Read, Seq_Read16, Seq_Set_Delay, Seq_Set_Gain, Seq_Select, Seq_Start, Seq_Status

Valid for

Gold II

Example

```
Rem Please select the appropriate include for ADbasic / TiCoBasic
#Include ADwinGoldII.inc / GoldIITiCo.inc
Dim Data_1[16] As Long At DM_Local

Init:
  Rem settings for sequential controls
  Seq_Mode(1,2) : Seq_Mode(2,2)'continuous mode
  Rem waiting time 4.5 µs (125*20ns + 2µs)
  Seq_Set_Delay(1,125) : Seq_Set_Delay(2,125)
  Seq_Set_Gain(1,0) : Seq_Set_Gain(2,0)'gain factor 1
  Seq_Select(0FFh)          'select channels 1…8
  Rem start sequential control of ADC1 and ADC2
  Seq_Start(11b)

Event:
  Rem read values of channels 1…8
  Seq_Read8(Data_1,1)

Finish:
  Seq_Mode(1,0)             'reset to standard mode
  Seq_Mode(2,0)             'reset to standard mode
```

Seq_Read16

T11

Seq_Read16 returns the measurement values (16 bit) of the input channels 1…16 of the sequential control.

Syntax

```
#Include ADwinGoldII.inc
Seq_Read16(array[], array_idx)
```

Parameters

array[]	Array to hold the measurement values of the input channels 1…16.	ARRAY LONG
array_idx	Array element which holds the first measurement value.	LONG

Notes

The selection of input channels is useful only, if the sequential control has been enabled with **Seq_Mode** and **Seq_Start**, and only for input channels which have been selected with **Seq_Select**.

See also

Seq_Mode, Seq_Read, Seq_Read8, Seq_Set_Delay, Seq_Set_Gain, Seq_Select, Seq_Start, Seq_Status

Valid for

Gold II

Example

```
#Include ADwinGoldII.inc
Dim Data_1[16] As Long At DM_Local

Init:
  Rem settings for sequential controls
  Seq_Mode(1,2) : Seq_Mode(2,2)'continuous mode
  Rem waiting time 4.5 µs (125*20ns + 2µs)
  Seq_Set_Delay(1,125) : Seq_Set_Delay(2,125)
  Seq_Set_Gain(1,0) : Seq_Set_Gain(2,0)'gain factor 1
  Seq_Select(0FFFFh)        'select channels 1…16
  Rem start sequential control of ADC1 and ADC2
  Seq_Start(11b)

Event:
  Rem read values of channels 1…16
  Seq_Read16(Data_1,1)

Finish:
  Seq_Mode(1,0)             'reset to standard mode
  Seq_Mode(2,0)             'reset to standard mode
```

Seq_Set_Delay

T11 TiCo

Seq_Set_Delay sets the waiting time (between 2 measurements) of the sequential control for one ADC.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
Seq_Set_Delay(adc_no, mux_time)
```

Parameters

adc_no	Number (1, 2) of the ADC.	LONG
mux_time	Number of time units of the variable part of the waiting time of the sequential control: 0…2^31: Time in units of 20ns.	LONG

Notes

The instruction is useful only, if the sequential control is enabled with **Seq_Mode**.

After power-up, the waiting time is equal to the multiplexer settling time of 2µs.

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

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

The waiting time is calculated according to the following formula:

$$\text{waiting time} = \texttt{mux\_time} \cdot 20\text{ns} + \text{conversion time}$$

The conversion time of the ADC is 2µs.

See also

Seq_Mode, Seq_Read, Seq_Read8, Seq_Read16, Seq_Set_Gain, Seq_Select, Seq_Start, Seq_Status

Valid for

Gold II

Example

```
Rem Please select the appropriate include for ADbasic / TiCoBasic
#Include ADwinGoldII.inc / GoldIITiCo.inc
Dim Data_1[16] As Long At DM_Local
Dim i As Long

Init:
  Rem settings for sequential control of ADC 2
  Seq_Mode(2,2)             'continuous mode
  Seq_Set_Delay(2,125)      'waiting time 4.5 µs (125*20ns + 2µs)
  Seq_Set_Gain(2,0)         'gain factor 1
  Rem select all channels. selection is valid only for active
  Rem sequential control 2 = even numbered channels.
  Seq_Select(0FFFFh)
  Rem start sequential control of ADC2
  Seq_Start(10b)

Event:
  Rem read current values of even channels
  For i = 2 To 16 Step 2
    Data_1[i] = Seq_Read(i)
  Next i

Finish:
  Seq_Mode(2,0)             'reset to standard mode
```

Seq_Set_Gain

T11 TiCo

Seq_Set_Gain sets the gain factor of the sequential control for one ADC.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
Seq_Set_Gain(adc_no, gain)
```

Parameters

adc_no	Number (1, 2) of the ADC.	LONG
gain	Gain factor (with active sequential control only): 0 factor = 1, voltage range -10V…+10V. 1 factor = 2, voltage range -5V…+5V. 2 factor = 4, voltage range -2.5V…+2.5V. 3 factor = 8, voltage range 1.25V…+1.25V.	LONG

Notes

The instruction is useful only, if the sequential control is enabled with **Seq_Mode**.

See also

Seq_Mode, Seq_Read, Seq_Read8, Seq_Read16, Seq_Set_Delay, Seq_Select, Seq_Start, Seq_Status

Valid for

Gold II

Example

```
Rem Please select the appropriate include for ADbasic / TiCoBasic
#Include ADwinGoldII.inc / GoldIITiCo.inc
Dim Data_1[16] As Long At DM_Local
Dim i As Long

Init:
  Rem settings for sequential control of ADC 2
  Seq_Mode(2,2)              'continuous mode
  Seq_Set_Delay(2,125)       'waiting time 4.5 µs (125*20ns + 2µs)
  Seq_Set_Gain(2,0)          'gain factor 1
  Rem select all channels. selection is valid only for active
  Rem sequential control 2 = even numbered channels.
  Seq_Select(0FFFFh)
  Rem start sequential control of ADC2
  Seq_Start(10b)

Event:
  Rem read current values of even channels
  For i = 2 To 16 Step 2
    Data_1[i] = Seq_Read(i)
  Next i

Finish:
  Seq_Mode(2,0)              'reset to standard mode
```

Seq_Select

T11 TiCo

Seq_Select wählt die Eingangskanäle aus, die mit den Ablaufsteuerungen der beiden ADC gewandelt werden.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
Seq_Select(pattern)
```

Parameters

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

Bit no.	31…16	15	…	2	1	0
Channel no.	–	16	…	3	2	1

Notes

The selection of input channels is useful only, if the sequential control of the appropriate sequential control is enabled with **Seq_Mode**.

Even though the input channels are selected together with **Seq_Select**, the sequential controls of the ADCs work independently. ADC 1 converts inputs with odd numbers only, ADC 2 inputs with even number.

The channels are automatically sorted in ascending order of channel numbers, that is the sequential controls convert the channel with the smallest number first.

See also

Seq_Mode, Seq_Read, Seq_Read8, Seq_Read16, Seq_Set_Delay, Seq_Set_Gain, Seq_Start, Seq_Status

Valid for

Gold II

Example

```
Rem Please select the appropriate include for ADbasic / TiCoBasic
#Include ADwinGoldII.inc / GoldIITiCo.inc
Dim Data_1[16] As Long At DM_Local
Dim i As Long

Init:
  Rem settings for sequential control of ADC 2
  Seq_Mode(2,2)              'continuous mode
  Seq_Set_Delay(2,125)       'waiting time 4.5 µs (125*20ns + 2µs)
  Seq_Set_Gain(2,0)          'gain factor 1
  Rem select all channels. selection is valid only for active
  Rem sequential control 2 = even numbered channels.
  Seq_Select(0FFFFh)
  Rem start sequential control of ADC2
  Seq_Start(10b)

Event:
  Rem read current values of even channels
  For i = 2 To 16 Step 2
    Data_1[i] = Seq_Read(i)
  Next i

Finish:
  Seq_Mode(2,0)              'reset to standard mode
```

Seq_Start

T11 TiCo

Seq_Start enables od disables the sequential controls of both ADC.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
Seq_Start(pattern)
```

Parameters

pattern	Bit pattern to select the ADC: Bit = 0: disable sequential control of this ADC. Bit = 1: enable sequential control of this ADC.	LONG

Bit	31:02	01	00
ADC no.	–	2	1

Notes

The instruction is useful only, if the sequential control is enabled with **Seq_Mode**.

If both sequential controls are to be used, they should be run with the same delay settings (**Seq_Set_Delay**) and started at the same time. Thus, you ensure measurements of both sequential controls to be done synchronously.

See also

Seq_Mode, Seq_Read, Seq_Read8, Seq_Read16, Seq_Set_Delay, Seq_Set_Gain, Seq_Select, Seq_Status

Valid for

Gold II

Example

```
Rem Please select the appropriate include for ADbasic / TiCoBasic
#Include ADwinGoldII.inc'for ADbasic
Rem #Include GoldIITiCo.inc 'for TiCoBasic
Dim Data_1[1600] As Long At DM_Local
Dim i As Long

Init:
  Rem settings for sequential controls
  Seq_Mode(1,2) : Seq_Mode(2,2)'continuous mode
  Rem waiting time 4.5 µs (125*20ns + 2µs)
  Seq_Set_Delay(1,125) : Seq_Set_Delay(1,125)
  Seq_Set_Gain(1,0)) : Seq_Set_Gain(1,0) 'gain factor 1
  Rem select channels 1..7, i.e. channels 1,3,5,7 for sequential
  Rem control 1 and channels 2,4,6 for seq. control 2
  Seq_Select(07Fh)
  Rem start both sequential controls
  Seq_Start(11b)
  i = 1

Event:
  Rem read current values of channels 1..8
  Seq_Read8(Data_1,i) : i = i + 8
  If (i > 1600) Then i = 1

Finish:
  Seq_Mode(1,0)              'reset to standard mode
  Seq_Mode(2,0)              'reset to standard mode
```

Seq_Status

T11 TiCo

Seq_Status returns if the single conversion cycle of a sequential control of an ADC is completed.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
ret_val = Seq_Status(adc_no)
```

Parameters

adc_no	Number (1, 2) of the ADC.	LONG
ret_val	Status of the sequential control for mode "single shot": 0: Single conversion cycle is completed. 1: Single conversion cycle is not yet completed.	LONG

Notes

The return value is useful only, if the sequential control has been initialized for operation mode "single shot" with **Seq_Mode**.

See also

Seq_Mode, Seq_Read, Seq_Read8, Seq_Read16, Seq_Set_Delay, Seq_Set_Gain, Seq_Select, Seq_Start

Valid for

Gold II

Example

```
Rem Please select the appropriate include for ADbasic / TiCoBasic
#Include ADwinGoldII.inc / GoldIITiCo.inc
Dim Data_1[16] As Long At DM_Local
Dim i As Long

Init:
  Rem settings for sequential control of ADC 1
  Seq_Mode(1,1)              'single shot mode
  Seq_Set_Delay(1,125)       'waiting time 4.5 µs (125*20ns + 2µs)
  Seq_Set_Gain(1,1)          'gain factor 2
  Rem select all channels. selection is valid only for active
  Rem sequential control 1 = odd numbered channels.
  Seq_Select(0FFFFh)
  Rem start sequential control of ADC1
  Seq_Start(01b)

Event:
  Do
  Until (Seq_Status(1)=0)
  Rem read values of odd channels
  For i = 1 To 15 Step 2
    Data_1[i] = Seq_Read(i)
  Next i
  Rem start sequential control of ADC1
  Seq_Start(01b)

Finish:
  Seq_Mode(2,0)              'reset to standard mode
```

15.3 Digital Inputs and Outputs

This section describes instructions to access digital inputs and outputs of *ADwin-Gold II*:

Conf_DIO

T11 TiCo

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

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
Conf_DIO(pattern)
```

Parameterss

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

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

Notes

The digital channels system 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.

We recommend the use of the configuration **Conf_DIO(1100b)**, which specifies DIO00…DIO15 as inputs and DIO16…DIO31 as outputs (see also page 15).

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.

See also

Digin, Digin_Long, Digin_Word1, Digin_Word2, Digout, Digout_Bits, Digout_Long, Digout_Word1, Digout_Word2, Get_Digout_Long, Get_Digout_Word1, Get_Digout_Word2

Valid for

Gold II

Example

```
Rem Please select the appropriate include for ADbasic / TiCoBasic
#Include ADwinGoldII.inc / GoldIITiCo.inc
Init:
  'Configure DIO00…DIO15 as inputs
  'and DIO16…DIO31 as outputs
  Conf_DIO(1100b)
```

Digin

T11 TiCo

Digin returns the TTL level of a digital input.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
ret_val = Digin(channel)
```

Parameters

channel	Number (0…31) of digital input.	LONG
ret_val	TTL level of the selected input: 1: TTL level is high. 0: TTL level is low.	LONG

Notes

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

Conf_DIO configures digital channels as inputs or outputs in groups of 8.

See also

Conf_DIO, Digin_Edge, Digin_Long, Digin_Word1, Digin_Word2, Digout

Valid for

Gold II

Example

```
Rem Please select the appropriate include for ADbasic / TiCoBasic
#Include ADwinGoldII.inc / GoldIITiCo.inc

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

Event:
  Rem input DIO00 = high?
  If (Digin(0) = 1) Then
    Rem Set DIO18 and DIO20 to values of DIO02 and DIO05
    Digout(18, Digin(2))
    Digout(20, Digin(5))
  EndIf
```

Digin_Edge

T11 TiCo

Digin_Edge returns whether a positive or negative edge has occurred on digital inputs.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
ret_val = Digin_Edge(edge)
```

Parameters

edge	Kind of detected edge: 1: Detect positive edge. 0: Detect negative edge.	LONG
ret_val	Bit pattern where each bits represent an edge occurred at an input. The mapping of bits to inputs is shown below. Bit = 1: An edge has occurred. Bit = 0: No edge occurred.	LONG

Bit no.	31	30	...	2	1	0
Input	31	30	...	2	1	0

Notes

A set bit in ret_val means, that a selected edge has been occurred at least once at the digital input since the previous query. Bit for output channels always return zero.

A query with **Digin_Edge** resets all bits to zero.

See also

Digin_FIFO_Clear, Digin_FIFO_Enable, Digin_FIFO_Full, Digin_FIFO_Read, Digin_FIFO_Read_Timer

Valid for

Gold II

Example

```
Rem Please select the appropriate include for ADbasic / TiCoBasic
#Include ADwinGoldII.inc / GoldIITiCo.inc

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

Event:
  Rem check positive and negative edges, mask out outputs
  Par_1 = Digin_Edge(1) And 0Fh
  Par_2 = Digin_Edge(0) And 0Fh

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

Digin_FIFO_Clear

T11 TiCo

Digin_FIFO_Clear clears the FIFO of the edge detection unit.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
Digin_FIFO_Clear()
```

Parameters

- / -

Notes

- / -

See also

Digin_FIFO_Enable, Digin_FIFO_Full, Digin_FIFO_Read, Digin_FIFO_Read_Timer, Digin_Edge

Valid for

Gold II

Example

```
Rem Please select the appropriate include for ADbasic / TiCoBasic
#Include ADwinGoldII.inc / GoldIITiCo.inc

Dim Data_1[10000], Data_2[10000] As Long
Dim num, i, index As Long

Init:
  Conf_DIO(1100b)           'channels 0:15 as inputs
  Digin_FIFO_Enable(0)      'edge control off
  Digin_FIFO_Clear()        'clear FIFO
  Digin_FIFO_Enable(10011b)'control channels 1,2,5
  index = 1


Event:
  num = Digin_FIFO_Full()  'number of value pairs
  If (num > 0) Then
    If (index+num > 10000) Then index = 1
    Rem read value pairs
    For i = 1 To num
      Digin_FIFO_Read(Data_1[index], Data_2[index])
      index = index+1
    Next i
  EndIf
```

Digin_FIFO_Enable

T11 TiCo

Digin_FIFO_Enable determines, which input channels the edge detection unit will monitor.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
Digin_FIFO_Enable(channels)
```

Parameters

channels — Bit pattern which determines the input channels to be monitored. LONG

Bit no.	31	30	…	2	1	0
Input	31	30	…	2	1	0

Notes

Only input channels can be monitored. The channels are programmed as inputs or outputs with **DigProg**.

The edge detection unit checks each 10 ns, if an edge has occurred at the selected input channels or if a level has been changed. If an edge has occurred, a pair of values is copied into an internal FIFO array:

- Value 1 contains the level status of all channels as bit pattern.
- Value 2 contains a time stamp, which is the current value of a 100 MHz timer.

The FIFO array may contain 511 value pairs (level status and time stamp) in maximum. If and as long as the FIFO array is filled completely, any additional value pair cannot be saved and will thus be lost.

See also

Digin_FIFO_Clear, Digin_FIFO_Full, Digin_FIFO_Read, Digin_FIFO_Read_Timer, Digin_Edge, Conf_DIO

Valid for

Gold II

Example

```
Rem Please select the appropriate include for ADbasic / TiCoBasic
#Include ADwinGoldII.inc / GoldIITiCo.inc

Dim Data_1[10000], Data_2[10000] As Long
Dim i, num, index As Long

Init:
  Conf_DIO(1100b)           'channels 0:15 as inputs
  Digin_FIFO_Enable(0)      'edge control off
  Digin_FIFO_Clear()        'clear FIFO
  Digin_FIFO_Enable(10011b)'control channels 1,2,5
  index = 1


Event:
  num = Digin_FIFO_Full()  'number of value pairs
  If (num > 0) Then
    If (index+num > 10000) Then index = 1
    Rem read value pairs
    For i = 1 To num
      Digin_FIFO_Read(Data_1[index], Data_2[index])
      index = index+1
    Next i
  EndIf
```

Digin_FIFO_Full

T11 TiCo

Digin_FIFO_Full returns the number of saved value pairs in the FIFO of the edge detection unit.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
ret_val = Digin_FIFO_Full()
```

Parameters

ret_val — Number (0…511) of saved value pairs in the FIFO. LONG

Notes

The FIFO array may contain 511 value pairs (level status and time stamp) in maximum. If and as long as the FIFO array is filled completely, any additional value pair cannot be saved and will thus be lost.

See also

Digin_FIFO_Clear, Digin_FIFO_Enable, Digin_FIFO_Read, Digin_FIFO_Read_Timer, Digin_Edge

Valid for

Gold II

Example

```
Rem Please select the appropriate include for ADbasic / TiCoBasic
#Include ADwinGoldII.inc / GoldIITiCo.inc

Dim Data_1[10000], Data_2[10000] As Long
Dim num, i, index As Long

Init:
  Conf_DIO(1100b)           'channels 0:15 as inputs
  Digin_FIFO_Enable(0)      'edge control off
  Digin_FIFO_Clear()        'clear FIFO
  Digin_FIFO_Enable(10011b)'control channels 1,2,5
  index = 1

Event:
  num = Digin_FIFO_Full()  'number of value pairs
  If (num > 0) Then
    If (index+num > 10000) Then index = 1
    Rem read value pairs
    For i = 1 To num
      Digin_FIFO_Read(Data_1[index], Data_2[index])
      index = index+1
    Next i
  EndIf
```

Digin_FIFO_Read

T11 TiCo

Digin_FIFO_Read reads the value pairs from the FIFO of the edge detection unit and writes them into 2 arrays.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
Digin_FIFO_Read(value_by_ref, timestamp_by_ref)
```

Parameters

value_by_ref	Array where the level status bit patterns are written. Each level status bit corresponds to a digital input (see table below).	LONG
timestamp_by_ref	Array where time stamps are written.	LONG

Bit no.	31	30	…	2	1	0
Input	31	30	…	2	1	0

Notes

Before reading there must be a check with **Digin_FIFO_Full**, if there is at least one value pair saved in the FIFO.

The passed parameters must be variables, not constants.

The time difference between 2 level status patterns is the difference of the appropriate time stamps, measured in units of 10ns:

$$\Delta t = (\text{stamp}_1 - \text{stamp}_2) \cdot 10\ \text{ns}$$

See also

Digin_FIFO_Clear, Digin_FIFO_Enable, Digin_FIFO_Full, Digin_FIFO_Read_Timer, Digin_Edge

Valid for

Gold II

Example

```
Rem Please select the appropriate include for ADbasic / TiCoBasic
#Include ADwinGoldII.inc / GoldIITiCo.inc
Dim Data_1[10000], Data_2[10000] As Long
Dim num, index As Long

Init:
  Conf_DIO(1100b)           'channels 0:15 as inputs
  Digin_FIFO_Enable(0)      'edge control off
  Digin_FIFO_Clear()        'clear FIFO
  Digin_FIFO_Enable(10011b)'control channels 1,2,5
  index = 1

Event:
  If (Digin_FIFO_Full()>0) Then
    Rem read one value pair
    Digin_FIFO_Read(Data_1[index], Data_2[index])
    index = index + 1
    If (index>10000) Then index = 1
  EndIf
```

Digin_FIFO_Read_Timer

T11 TiCo

Digin_FIFO_Read_Timer returns the current status of the 100 MHz timer.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
ret_val = Digin_FIFO_Read_Timer()
```

Parameters

ret_val — Current value (-2^{31}-1 … 2^{31}) of the 100 MHz timer. LONG

Notes

The module timer is used to provide time stamps for the edge detection unit, see **Digin_FIFO_Enable**.

The timer value is increased every 10 ns by 1, so the timer will reach the original timer value after about 43 seconds (= 10 ns × 2^{32}). For comparison of time this "overflow" must be considered, so the timer value must be queried regularly in the program before a overflow has happened.

See also

Digin_FIFO_Enable, Digin_FIFO_Read

Valid for

Gold II

Example

```
Rem Please select the appropriate include for ADbasic / TiCoBasic
#Include ADwinGoldII.inc / GoldIITiCo.inc
Dim start, diff, count As Long

Init:
  count = 0
  start = Digin_FIFO_Read_Timer()

Event:
  Rem count number of counter overflows
  If (Digin_FIFO_Read_Timer() < start) Then
    Inc(count)
    start = Digin_FIFO_Read_Timer()
  EndIf
```

Digin_Long

T11 TiCo

Digin_Long returns the value of the digital inputs DIO00…DIO31.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
ret_val = Digin_Long()
```

Parameterss

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 no. in ret_val	31	30	…	1	0
Channel	DIO31	DIO30	…	DIO01	DIO00

Notes

For any digital channel configured as output **Digin_Long** will return an undefined value.

Digital channels are configured as inputs after start-up. (And thus cannot be accessed as outputs yet). Channels can be configured as inputs or outputs in groups of 8.

Conf_DIO configures digital channels as inputs or outputs in groups of 8. A standard configuration is **Conf_DIO**(1100b) which specifies DIO00…DIO15 as inputs and DIO16…DIO31 as outputs.

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

Conf_DIO, Digin, Digin_Word1, Digin_Word2, Digout_Long

Valid for

Gold II

Example

```
Rem Please select the appropriate include for ADbasic / TiCoBasic
#Include ADwinGoldII.inc / GoldIITiCo.inc
Dim Data_1[10000] As Long As FIFO

Init:
  Rem Configure all channels As inputs
  Conf_DIO(0000b)

Event:
  Rem Is digital input DIO17 set?
  If ((Shift_Right(Digin_Long(),17) And 1) = 1) Then
    Data_1 = ADC(1)          'get measurement value
  EndIf
```

Digin_Word1

T11 TiCo

Digin_Word1 returns the values of the digital inputs DIO00…DIO15.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
ret_val = Digin_Word1()
```

Parameterss

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 no. in ret_val	31 ... 16	15	14	...	1	0
Channel	–	DIO15	DIO14	...	DIO01	DIO00

Notes

For any digital channel configured as output **Digin_Word1** will return an undefined value.

Digital channels are configured as inputs after start-up. (And thus cannot be accessed as outputs yet). Channels can be configured as inputs or outputs in groups of 8.

Conf_DIO configures digital channels as inputs or outputs in groups of 8. A standard configuration is **Conf_DIO**(1100b) which specifies DIO00…DIO15 as inputs and DIO16…DIO31 as outputs.

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

Conf_DIO, Digin, Digin_Long, Digin_Word2, Digout_Word1

Valid for

Gold II

Example

```
Rem Please select the appropriate include for ADbasic / TiCoBasic
#Include ADwinGoldII.inc / GoldIITiCo.inc
Dim Data_1[10000] As Long As FIFO
Init:
  REM Configure inputs and outputs
  Conf_DIO(1100b)

Event:
  REM query if inputs DIO01 and DIO02 are level High
  If ((Digin_Word1() And 110b) = 110b) Then
    Data_1 = ADC(1)          'get measurement value
  EndIf
```

Digin_Word2

T11 TiCo

Digin_Word2 returns the values of the digital inputs DIO16…DIO31.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
ret_val = Digin_Word2()
```

Parameterss

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 no. in ret_val	31 ... 16	15	14	...	1	0
Channel	–	DIO31	DIO30	...	DIO17	DIO16

Notes

For any digital channel configured as output **Digin_Word2** will return an undefined value.

Digital channels are configured as inputs after start-up. (And thus cannot be accessed as outputs yet). Channels can be configured as inputs or outputs in groups of 8.

Conf_DIO configures digital channels as inputs or outputs in groups of 8. A standard configuration is **Conf_DIO**(1100b) which specifies DIO00…DIO15 as inputs and DIO16…DIO31 as outputs.

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

Conf_DIO, Digin, Digin_Long, Digin_Word1, Digout_Word2

Valid for

Gold II

Example

```
Rem Please select the appropriate include for ADbasic / TiCoBasic
#Include ADwinGoldII.inc / GoldIITiCo.inc
Dim Data_1[10000] As Long As FIFO
Init:
  REM Configure inputs and outputs
  Conf_DIO(0011b)

Event:
  REM query if inputs DIO16 and DIO17 are level High
  If ((Digin_Word2() And 11b) = 11b) Then
    Data_1 = ADC(1)          'get measurement value
  EndIf
```

Digout

T11 TiCo

Digout sets a single channel to the specified TTL level.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
Digout(channel,value)
```

Parameterss

channel	Number (0…31) of digital output.	LONG
value	TTL level to which the output is set: 1: Set to TTL level high. 0: Set to TTL level low.	LONG

Notes

For any digital channel configured as input **Digout** has no function.

Conf_DIO configures digital channels as inputs or outputs in groups of 8.

See also

Conf_DIO, Digin, Digout_Bits, Digout_Long, Digout_Word1, Digout_Word2

Valid for

Gold II

Example

```
Rem Please select the appropriate include for ADbasic / TiCoBasic
#Include ADwinGoldII.inc / GoldIITiCo.inc
Dim value As Long

Init:
  REM Configure inputs and outputs
  Conf_DIO(1100b)

Event:
  value = ADC(1)            'get measurement value
  If (value > 1600) Then'limit exceeded?
    Digout(19,1)            'set output DIO19 to level high
    Digout(23,0)            'set output DIO23 to level low
  EndIf
```

Digout_Bits

T11 TiCo

Digout_Bits sets the specified channels to a defined TTL level.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
Digout_Bits(set,clear)
```

Parameterss

set	Bit pattern to specify outputs, which are set to TTL level High (see table). 1: set to TTL level high. 0: do not change TTL level.	LONG
clear	Bit pattern to specify outputs, which are set to TTL level Low (see table). 1: set to TTL level low. 0: do not change TTL level.	LONG

Bit no.	31	30	…	1	0
Channel	DIO31	DIO30	…	DIO01	DIO00

Notes

For any digital channel configured as input **Digout_Bits** has no function.

Conf_DIO configures digital channels as inputs or outputs in groups of 8.

If a bit is set to 1 in set as well as in clear, the appropriate channel is set to TTL level low.

See also

Conf_DIO, Digout, Digout_Long, Get_Digout_Long

Valid for

Gold II

Example

```
Rem Please select the appropriate include for ADbasic / TiCoBasic
#Include ADwinGoldII.inc / GoldIITiCo.inc
Dim value As Long

Init:
  REM Configure inputs and outputs
  Conf_DIO(0011b)

Event:
  value = ADC(1)            'get measurement value
  If (value > 3000) Then'limit exceeded?
    Rem Set outputs DIO00 and DIO02 to level high,
    Rem set outputs DIO001, DIO003 and DIO004 to level low.
    Digout_Bits(00101b, 11010b)
  EndIf
```

Digout_Long

T11 TiCo

Digout_Long sets all channels to the defined TTL levels.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
Digout_Long(pattern)
```

Parameterss

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

Bit no. in pattern	31	30	…	1	0
Channel	DIO31	DIO30	…	DIO01	DIO00

Notes

For any digital channel configured as input **Digout_Long** has no function.

Conf_DIO configures digital channels as inputs or outputs in groups of 8.

To set selected channels without changing the other channels, use the instruction **Digout_Bits**.

See also

Conf_DIO, Digin_Long, Digout, Digout_Bits, Digout_Word1, Digout_Word2, Get_Digout_Long

Valid for

Gold II

Example

```
Rem Please select the appropriate include for ADbasic / TiCoBasic
#Include ADwinGoldII.inc / GoldIITiCo.inc
Dim value As Long

Init:
  REM Configure all channels as outputs
  Conf_DIO(1111b)

Event:
  value = ADC(1)            'get measurement value
  If (value > 1500) Then 'limit exceeded?
    Rem Set outputs DIO00, DIO02, DIO06 to level high,
    Rem all other channels to level low.
    Digout_Long(1000101b)
  EndIf
```

Digout_Reset

T11 TiCo

Digout_Reset sets selected digital outputs to TTL level Low.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
Digout_Reset(clear)
```

Parameters

clear	Bit pattern to select the outputs, which are set to TTL level Low (see table). 1: set to TTL level Low. 0: do not change level.	LONG

Bit no.	31	30	…	1	0
Channel	DIO31	DIO30	…	DIO01	DIO00

Notes

For a channel configured as input, **Digout_Reset** has no function.

Conf_DIO configures digital channels as inputs or outputs in groups of 8.

See also

Conf_DIO, Digout_Word2, Digout_Bits, Digout_Long, Get_Digout_Long, Digout_Set

Valid for

Gold II

Example

```
Rem Please select the appropriate include for ADbasic /
 TiCoBasic
#Include ADwinGoldII.inc / GoldIITiCo.inc
DIM value As Long

Init:
 REM configure inputs/outputs
 CONF_DIO(0011b)
 Processdelay = 10000

Event:
 value = ADC(1)              'get measurement value
 IF (value > 3000) Then'limit exceeded?
  REM set outputs DIO01, DIO03, and DIO04 to level Low
  Digout_Reset(11010b)
 EndIf
```

Digout_Set

T11 TiCo

Digout_Set sets selected outputs to TTL level High.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
Digout_Set(set)
```

Parameters

set	Bit pattern to select the outputs, which are set to TTL level High (see table). LONG 1: set to TTL level High. 0: do not change level.

Bit no.	31	30	…	1	0
Channel	DIO31	DIO30	…	DIO01	DIO00

Notes

For a channel configured as input, **Digout_Set** has no function.

Conf_DIO configures digital channels as inputs or outputs in groups of 8.

See also

Conf_DIO, Digout_Word2, Digout_Bits, Digout_Long, Get_Digout_Long, Digout_Reset

Valid for

Gold II

Example

```
Rem Please select the appropriate include for ADbasic /
 TiCoBasic
#Include ADwinGoldII.inc / GoldIITiCo.inc
DIM value As Long

Init:
  REM configure inputs/outputs
  CONF_DIO(0011b)
  Processdelay = 10000

Event:
  value = ADC(1)                'get measurement value
  IF (value > 3000) Then'limit exceeded?
    REM set outputs DIO00 and DIO02 to level High
    Digout_Set(00101b)
  EndIf
```

Digout_Word1

T11 TiCo

Digout_Word1 sets the digital outputs DIO00…DIO15 to defined TTL-levels.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
Digout_Word1(pattern)
```

Parameterss

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

Bit no. in pattern	31 ... 16	15	14	...	1	0
Channel	–	DIO15	DIO14	...	DIO01	DIO00

Notes

For any digital channel configured as input **Digout_Word1** has no function.

Conf_DIO configures digital channels as inputs or outputs in groups of 8.

To set selected channels without changing the other channels, use the instruction **Digout_Bits**.

See also

Conf_DIO, Digin_Word1, Digout, Digout_Bits, Digout_Long, Digout_Word2, Get_Digout_Word1

Valid for

Gold II

Example

```
Rem Please select the appropriate include for ADbasic / TiCoBasic
#Include ADwinGoldII.inc / GoldIITiCo.inc
Dim value As Long

Init:
  REM Configure inputs and outputs
  Conf_DIO(0011b)

Event:
  value = ADC(1)            'get measurement value
  If (value > 3000) Then 'limit exceeded?
    Rem Set outputs DIO00, DIO02 to level high,
    Rem all other channels to level low.
    Digout_Word1(101b)
  EndIf
```

Digout_Word2

T11 TiCo

Digout_Word2 sets the digital outputs DIO16…DIO31 to defined TTL-levels.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
Digout_Word2(pattern)
```

Parameterss

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

Bit no. in pattern	31 ... 16	15	14	...	1	0
Channel	–	DIO31	DIO30	...	DIO17	DIO16

Notes

For any digital channel configured as input **Digout_Word2** has no function.

Conf_DIO configures digital channels as inputs or outputs in groups of 8.

To set selected channels without changing the other channels, use the instruction **Digout_Bits**.

See also

Conf_DIO, Digin_Word2, Digout, Digout_Bits, Digout_Long, Digout_Word1, Get_Digout_Word2

Valid for

Gold II

Example

```
Rem Please select the appropriate include for ADbasic / TiCoBasic
#Include ADwinGoldII.inc / GoldIITiCo.inc
Dim value As Long

Init:
  REM Configure inputs and outputs
  Conf_DIO(1100b)

Event:
  value = ADC(1)            'get measurement value
  If (value > 2500) Then 'limit exceeded?
    Rem Set outputs DIO17, DIO20 to level high,
    Rem all other channels to level low.
    Digout_Word2(10010b)
  EndIf
```

Get_Digout_Long

T11 TiCo

Get_Digout_Long returns the register contents of the digital outputs DIO00…DIO31.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
ret_val = Get_Digout_Long()
```

Parameterss

`ret_val` Contents (bit pattern) of the output register, bit allocation to outputs see table. LONG
1: TTL-level high.
0: TTL-level low.

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

Notes

The return value represents the status of the output register only. A read back of physical output status is technically impossible.

For any digital channel configured as input **Get_Digout_Long** will return an undefined value. **Conf_DIO** configures digital channels as inputs or outputs in groups of 8.

See also

Conf_DIO, Digin_Long, Digout_Bits, Digout_Long, Get_Digout_Word1, Get_Digout_Word2

Valid for

Gold II

Example

```
Rem Please select the appropriate include for ADbasic / TiCoBasic
#Include ADwinGoldII.inc / GoldIITiCo.inc

Init:
  REM Configure all channels as outputs
  Conf_DIO(1111b)

Event:
  Par_1 = Get_Digout_Long() 'read back bits 31:00 from register
```

Get_Digout_Word1

T11 TiCo

Get_Digout_Word1 returns the register contents of the digital outputs DIO00…DIO15.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
ret_val = Get_Digout_Word1()
```

Parameterss

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

Bit no. in ret_val	31 ... 16	15	14	...	1	0
Channel	–	DIO15	DIO14	...	DIO01	DIO00

Notes

The return value represents the status of the output register only. A read back of physical output status is technically impossible.

For any digital channel configured as input **Get_Digout_Long** will return an undefined value. **Conf_DIO** configures digital channels as inputs or outputs in groups of 8.

See also

Conf_DIO, Digin_Word1, Digout_Bits, Digout_Word1, Get_Digout_Long, Get_Digout_Word1, Get_Digout_Word2

Valid for

Gold II

Example

```
Rem Please select the appropriate include for ADbasic / TiCoBasic
#Include ADwinGoldII.inc / GoldIITiCo.inc

Init:
  REM Configure inputs and outputs
  Conf_DIO(0011b)

Event:
  Par_1 = Get_Digout_Word1() 'read back bits 15:00 from register
```

Get_Digout_Word2

T11 TiCo

Get_Digout_Word2 returns the register contents of the digital outputs DIO16…DIO31.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
ret_val = Get_Digout_Word2()
```

Parameterss

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

Bit no. in ret_val	31 ... 16	15	14	...	1	0
Channel	–	DIO31	DIO30	...	DIO17	DIO16

Notes

The return value represents the status of the output register only. A read back of physical output status is technically impossible.

For any digital channel configured as input **Get_Digout_Long** will return an undefined value. **Conf_DIO** configures digital channels as inputs or outputs in groups of 8.

See also

Conf_DIO, Digin_Word1, Digout_Bits, Digout_Word2, Get_Digout_Long, Get_Digout_Word1, Get_Digout_Word2

Valid for

Gold II

Example

```
Rem Please select the appropriate include for ADbasic / TiCoBasic
#Include ADwinGoldII.inc / GoldIITiCo.inc

Init:
  REM Configure inputs and outputs
  Conf_DIO(1100b)

Event:
  Par_1 = Get_Digout_Word2() 'read back bits 31:16 from register
```

15.4 Counters

This section describes instructions to access counter inputs of *ADwin-Gold II*:

Cnt_Clear

T11 TiCo

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

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.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.

Please pay attention to set **Cnt_Mode** parameter pattern to bit 1=0 for the appropriate counters. Else, with bit 1=1, the counter inputs A, B have also to be set to TTL level high, in order to clear the counter.

See also

Cnt_Enable, Cnt_Get_Status, Cnt_Get_PW, Cnt_Get_PW_HL, Cnt_Latch, Cnt_Mode, Cnt_PW_Latch, Cnt_Read, Cnt_Read_Latch, Cnt_SE_Diff

Valid for

Gold II-CNT

Example

```
Rem Please select the appropriate include for ADbasic /
 TiCoBasic
#Include ADwinGoldII.inc 'for ADbasic
Rem #Include GoldIITiCo.inc     'for TiCoBasic
Dim old_1, new_1 As Long
Dim old_2, new_2 As Long

Init:
  old_1 = 0                     'initialize
  old_2 = 0
  Cnt_SE_Diff(0011b)            'counters 1+2 diff. (3+4 single
                                'ended)
  Rem Counter 1: Mode clock-direction, enable CLR input
  Cnt_Mode(1,10000b)
  Cnt_Mode(2,0)                 'Set all counters to external
                                'clock
  Cnt_Clear(11b)                'reset counters 1+2 to 0
  Rem start counters 1+2, stop counters 3+4 and PWM1-4
  Cnt_Enable(11b)

Event:
  Cnt_Latch(11b)                'latch both counters 1+2
  new_1 = Cnt_Read_Latch(1)'read latch A of counter 1 and
  new_2 = Cnt_Read_Latch(2)' latch A of counter 2.
  Par_1 = new_1 - old_1 'caluclate 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

T11 TiCo

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

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
Cnt_Enable(pattern)
```

Parameters

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

Bit no.	31…9	11	10	9	8	7…4	3	2	1	0
Counter no.	–	PW4	PW3	PW2	PW1	–	VR4	VR3	VR2	VR1

Notes

Standard counters and PWM counters run separately.

See also

Cnt_Clear, Cnt_Get_Status, Cnt_Get_PW, Cnt_Get_PW_HL, Cnt_Latch, Cnt_Mode, Cnt_PW_Latch, Cnt_Read, Cnt_Read_Latch, Cnt_SE_Diff

Valid for

Gold II-CNT

Example

```
Rem Please select the appropriate include for ADbasic /
 TiCoBasic
#Include ADwinGoldII.inc 'for ADbasic
Rem #Include GoldIITiCo.inc     'for TiCoBasic
Dim old_1, new_1 As Long
Dim old_2, new_2 As Long

Init:
  old_1 = 0                   'initialize
  old_2 = 0
  Cnt_SE_Diff(0011b)          'counters 1+2 diff. (3+4 single
                              'ended)
  Rem Counter 1: Mode clock-direction, enable CLR input
  Cnt_Mode(1,10000b)
  Cnt_Mode(2,0)               'Set all counters to external
                              'clock
  Cnt_Clear(11b)              'reset counters 1+2 to 0
  Rem start counters 1+2, stop counters 3+4 and PWM1-4
  Cnt_Enable(11b)

Event:
  Cnt_Latch(11b)              'latch both counters 1+2
  new_1 = Cnt_Read_Latch(1)'read latch A of counter 1 and
  new_2 = Cnt_Read_Latch(2)' latch A of counter 2.
  Par_1 = new_1 - old_1 'caluclate the difference (f =
                              'impulses/time)
  Par_2 = new_2 - old_2 ' -"-
  old_1 = new_1               'Save new counter values
  old_2 = new_2               ' -"-
```

Cnt_Get_Status

T11 TiCo

Cnt_Get_Status returns the counter status register of one counter.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
ret_val = Cnt_Get_Status(counter_no)
```

Parameters

counter_no	Counter number: 1…4.	LONG
ret_val	Contents of status register: Hints for potential error sources. Meaning of bits 0…4 see table.	LONG

Bit No.	31…5	4	3	2	1	0
Signal	–	C	L	N	B	A

- :don't care (signal status is not defined, mask out with **01Fh**)
A: Signal A (static)
B: Signal B (static)
N: CLR-/LATCH input (static)
L: Line error (cable not connected or the line is broken)
C: Correlation error (signals A and B are identical, i.e. they are not phase-shifted by approx. 90°)

Notes

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

The status register is automatically reset by reading.

See also

Cnt_Clear, Cnt_Enable, Cnt_Get_PW, Cnt_Get_PW_HL, Cnt_Latch, Cnt_Mode, Cnt_PW_Latch, Cnt_Read, Cnt_Read_Latch, Cnt_SE_Diff

Valid for

Gold II-CNT

Example

- / -

Cnt_Get_PW

T11

Cnt_Get_PW returns frequency and duty cycle of a PWM counter.

Syntax

```
#Include ADwinGoldII.inc
Cnt_Get_PW(pwm_output,frequency,dutycycle)
```

Parameters

pwm_output	Number (1…4) of PWM counter.	LONG
frequency	Frequency in Hertz: 0,025 Hz …100 MHz.	FLOAT CONST
dutycycle	Duty cycle in percent: 0.0…100.0.	FLOAT CONST

Notes

The return values are given in the parameters frequency and dutycycle.

See also

Cnt_Enable, Cnt_Get_PW_HL, Cnt_Mode, Cnt_PW_Latch, Cnt_SE_Diff

Valid for

Gold II-CNT

Example

```
#Include ADwinGoldII.Inc

Init:
  Cnt_SE_Diff(0)              'all counters single ended (TTL)
  Cnt_Mode(1,0)               'Counter 1: PWM at input A
  Cnt_Mode(2,0)               'Counter 2: PWM at input A
  Cnt_Enable(1100000000b)     'start PWM counters 1+2

Event:
  Cnt_PW_Latch(11b)           'latch both counters 1+2
  Cnt_Get_PW_HL(1,Par_1,Par_2) 'read high/low time
  Cnt_Get_PW(1,FPar_1,FPar_2) 'read frequency./duty cycle
```

Cnt_Get_PW_HL

T11 TiCo

Cnt_Get_PW_HL returns a stored high and low time of a PWM counter.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
Cnt_Get_PW_HL(pwm_no,hightime,lowtime)
```

Parameters

pwm_no	Number (1…4) of PWM counter.	LONG
hightime	Pulse duration in units of 10ns: PWM high level time.	LONG CONST
lowtime	Pulse period in units of 10ns: PWM low level time.	LONG CONST

Notes

The return values are given in the parameters hightime and lowtime.

See also

Cnt_Enable, Cnt_Get_PW, Cnt_Mode, Cnt_PW_Latch, Cnt_SE_Diff

Valid for

Gold II-CNT

Example

```
Rem Please select the appropriate include for ADbasic /
 TiCoBasic
#Include ADwinGoldII.inc 'for ADbasic
Rem #Include GoldIITiCo.inc     'for TiCoBasic
Dim old_1, new_1 As Long
Dim old_2, new_2 As Long

Init:
  old_1 = 0                    'initialize
  old_2 = 0
  Cnt_SE_Diff(0)               'all counters single ended (TTL)
  Cnt_Mode(1,0)                'Counter 1: PWM at input A
  Cnt_Mode(2,0)                'Counter 2: PWM at input A
  Cnt_Enable(1100000000b)      'start PWM counters 1+2

Event:
  Cnt_PW_Latch(11b)            'latch both counters 1+2
  Cnt_Get_PW_HL(1,Par_1,Par_2) 'read high/low time
  Cnt_Get_PW(1,FPar_1,FPar_2) 'read frequency./duty cycle
```

Cnt_Latch

T11 TiCo

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

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
Cnt_Latch(pattern)
```

Parameters

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

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

Notes

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

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

Valid for

Gold II-CNT

See also

Cnt_Clear, Cnt_Enable, Cnt_Get_Status, Cnt_Get_PW, Cnt_Get_PW_HL, Cnt_Mode, Cnt_PW_Latch, Cnt_Read, Cnt_Read_Latch, Cnt_SE_Diff

Example

```
Rem Please select the appropriate include for ADbasic /
 TiCoBasic
#Include ADwinGoldII.inc 'for ADbasic
Rem #Include GoldIITiCo.inc     'for TiCoBasic
Dim old_1, new_1 As Long
Dim old_2, new_2 As Long

Init:
  old_1 = 0                     'initialize
  old_2 = 0
  Cnt_SE_Diff(0011b)            'counters 1+2 diff. (3+4 single
                                'ended)
  Rem Counter 1: Mode clock-direction, enable CLR input
  Cnt_Mode(1,10000b)
  Cnt_Mode(2,0)                 'Set all counters to external
                                'clock
  Cnt_Clear(11b)                'reset counters 1+2 to 0
  Rem start counters 1+2, stop counters 3+4 and PWM1-4
  Cnt_Enable(11b)

Event:
  Cnt_Latch(11b)                'latch both counters 1+2
  new_1 = Cnt_Read_Latch(1)'read latch A of counter 1 and
  new_2 = Cnt_Read_Latch(2)' latch A of counter 2.
  Par_1 = new_1 - old_1 'caluclate the difference (f =
                                'impulses/time)
  Par_2 = new_2 - old_2 ' -"-
  old_1 = new_1                 'Save new counter values
  old_2 = new_2                 ' -"-
```

Cnt_Mode defines the operating mode of one counter.

Cnt_Mode

T11 TiCo

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
Cnt_Mode(counter_no,pattern)
```

Parameters

counter_no	Counter number: 1…4.	LONG
pattern	Bit pattern to set the operating mode of a counter.	LONG

Bit no.	Meaning
Bit 0	Counter mode: Bit = 0: mode clock/direction. Bit = 1: mode A-B.
Bit 1	Clear mode. Signal condition which clears the counter: Bit = 0: TTL level high at input CLR. Bit = 1: TTL level high at all inputs A, B, CLR. Available in mode A-B only.
Bit 2	Invert input A / CLK in mode clock/direction: Bit = 0: Input is not inverted. Bit = 1: Input is inverted.
Bit 3	Invert input B / DIR in mode clock/direction: Bit = 0: Input is not inverted. Bit = 1: Input is inverted.
Bit 4	Set use of input CLR / LATCH. Bit = 0: CLR input: clear counter. Bit = 1: LATCH input: latch counter.
Bit 5	Enable input CLR / LATCH. Bit = 0: Input CLR / LATCH is disabled. Bit = 1: Input CLR / LATCH is enabled.
Bit 6	Select edge for PWM analysis. Bit = 0: rising edge. Bit = 1: falling edge.
Bit 7,8	Select input for PWM analysis. 00b: Input A / CLK 01b: Input B / DIR 10b: Input CLR / LATCH
Bits 9…31	reserved

Notes

Please use **Cnt_Mode** only when the counter is disabled, see **Cnt_Enable**.

With standard clear mode (bit 1=0), the counter value is reset to zero as long as TTL level high is given at the input. In order to clear the counter, the input CLR must be enabled with bit 5=1.

If you want to clear a counter with **Cnt_Clear** set pattern bit 1=0. Else, with bit 1=1, the counter inputs A, B have also to be set to TTL level high, in order to clear the counter.

See also

Cnt_Clear, Cnt_Enable, Cnt_Get_Status, Cnt_Get_PW, Cnt_Get_PW_HL, Cnt_Latch, Cnt_PW_Latch, Cnt_Read, Cnt_Read_Latch, Cnt_SE_Diff

Valid for

Gold II-CNT

Example

```
Rem Please select the appropriate include for ADbasic /
 TiCoBasic
#Include ADwinGoldII.inc 'for ADbasic
Rem #Include GoldIITiCo.inc     'for TiCoBasic
Dim old_1, new_1 As Long
Dim old_2, new_2 As Long

Init:
  old_1 = 0                   'initialize
  old_2 = 0
  Cnt_SE_Diff(0011b)          'counters 1+2 diff. (3+4 single
                              'ended)
  Rem Counter 1: Mode clock-direction, enable CLR input
  Cnt_Mode(1,10000b)
  Cnt_Mode(2,0)               'Set all counters to external
                              'clock
  Cnt_Clear(11b)              'reset counters 1+2 to 0
  Rem start counters 1+2, stop counters 3+4 and PWM1-4
  Cnt_Enable(11b)

Event:
  Cnt_Latch(11b)              'latch both counters 1+2
  new_1 = Cnt_Read_Latch(1)'read latch A of counter 1 and
  new_2 = Cnt_Read_Latch(2)' latch A of counter 2.
  Par_1 = new_1 - old_1 'caluclate the difference (f =
                              'impulses/time)
  Par_2 = new_2 - old_2 ' -"-
  old_1 = new_1               'Save new counter values
  old_2 = new_2               ' -"-
```

Cnt_PW_Latch

T11 TiCo

Cnt_PW_Latch copies the value of one or more PWM counters into a buffer.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
Cnt_PW_Latch(pattern)
```

Parameters

pattern	Bit pattern. Bit = 0: no function. Bit = 1: transfer PWM counter value into a buffer.	LONG

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

Notes

The buffer is to be read with **Cnt_PW_F_DC** or **Cnt_PW_HL**.

See also

Cnt_Enable, Cnt_Get_PW, Cnt_Get_PW_HL, Cnt_Mode, Cnt_SE_Diff

Valid for

Gold II-CNT

Example

```
Rem Please select the appropriate include for ADbasic /
 TiCoBasic
#Include ADwinGoldII.inc 'for ADbasic
Rem #Include GoldIITiCo.inc     'for TiCoBasic
Dim old_1, new_1 As Long
Dim old_2, new_2 As Long

Init:
  old_1 = 0                    'initialize
  old_2 = 0
  Cnt_SE_Diff(0)               'all counters single ended (TTL)
  Cnt_Mode(1,0)                'Counter 1: PWM at input A
  Cnt_Mode(2,0)                'Counter 2: PWM at input A
  Cnt_Enable(1100000000b)      'start PWM counters 1+2

Event:
  Cnt_PW_Latch(11b)            'latch both counters 1+2
  Cnt_Get_PW_HL(1,Par_1,Par_2) 'read high/low time
  Cnt_Get_PW(1,FPar_1,FPar_2)  'read frequency./duty cycle
```

Cnt_Read

T11 TiCo

Cnt_Read transfers a current counter value into Latch A and returns the value.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
ret_val = Cnt_Read(counter_no)
```

Parameters

counter_no	Counter number: 1…4.	LONG
ret_val	Counter values.	LONG

Notes

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

See also

Cnt_Clear, Cnt_Enable, Cnt_Get_Status, Cnt_Get_PW, Cnt_Get_PW_HL, Cnt_Latch, Cnt_Mode, Cnt_PW_Latch, Cnt_Read_Latch, Cnt_SE_Diff

Valid for

Gold II-CNT

Example

```
Rem Please select the appropriate include for ADbasic /
 TiCoBasic
#Include ADwinGoldII.inc 'for ADbasic
Rem #Include GoldIITiCo.inc     'for TiCoBasic
Dim old_1, new_1 As Long
Dim old_2, new_2 As Long

Init:
  old_1 = 0                    'initialize
  old_2 = 0
  Cnt_SE_Diff(0011b)           'counters 1+2 diff. (3+4 single
                               'ended)
  Rem Counter 1: Mode clock-direction, enable CLR input
  Cnt_Mode(1,10000b)
  Cnt_Mode(2,0)                'Set all counters to external
                               'clock
  Cnt_Clear(11b)               'reset counters 1+2 to 0
  Rem start counters 1+2, stop counters 3+4 and PWM1-4
  Cnt_Enable(11b)

Event:
  Cnt_Latch(11b)               'latch both counters 1+2
  new_1 = Cnt_Read_Latch(1)'read latch A of counter 1 and
  new_2 = Cnt_Read_Latch(2)' latch A of counter 2.
  Par_1 = new_1 - old_1 'caluclate the difference (f =
                               'impulses/time)
  Par_2 = new_2 - old_2 ' -"-
  old_1 = new_1                'Save new counter values
  old_2 = new_2                ' -"-
```

Cnt_Read_Int_Register

T11 TiCo

Cnt_Read_Int_Register returns the content of a counter register.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
ret_val = Cnt_Read_Int_Register(counter_no,reg_no)
```

Parameters

counter_no	Counter number: 1…4.	LONG
reg_no	Key number (0…15) for a counter register, see below.	LONG
ret_val	Content of the counter register.	LONG

reg_no	Register
0	Latch 1 for positive edges.
1	Latch 2 for positive edges.
2	Latch 3 for positive edges.
3	Latch 1 for negative edges.
4	Latch 2 for negative edges.
5	Latch 3 for negative edges.
6	Software latch for VR counter.
7	Software latch for PWM counter.
8	Shadow register for Latch 1, positive edges.
9	Shadow register for Latch 2, positive edges.
10	Shadow register for Latch 3, positive edges.
11	Shadow register for Latch 1, negative edges.
12	Shadow register for Latch 2, negative edges.
13	Shadow register for Latch 3, negative edges.
14	Shadow register for software latch, VR counter.
15	Counter status.

Notes

- / -

See also

Cnt_Sync_Latch

Valid for

Gold II-CNT

Example

see Cnt_Sync_Latch

Cnt_Read_Latch

T11 TiCo

Cnt_Read_Latch returns the value of a counter's Latch A.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
ret_val = Cnt_Read_Latch(counter_no)
```

Parameters

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

Notes

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

See also

Cnt_Clear, Cnt_Enable, Cnt_Get_Status, Cnt_Latch, Cnt_Mode, Cnt_Read, Cnt_SE_Diff

Valid for

Gold II-CNT

Example

```
Rem Please select the appropriate include for ADbasic /
 TiCoBasic
#Include ADwinGoldII.inc 'for ADbasic
Rem #Include GoldIITiCo.inc     'for TiCoBasic
Dim old_1, new_1 As Long
Dim old_2, new_2 As Long

Init:
  old_1 = 0                    'initialize
  old_2 = 0
  Cnt_SE_Diff(0011b)           'counters 1+2 diff. (3+4 single
                               'ended)
  Rem Counter 1: Mode clock-direction, enable CLR input
  Cnt_Mode(1,10000b)
  Cnt_Mode(2,0)                'Set all counters to external
                               'clock
  Cnt_Clear(11b)               'reset counters 1+2 to 0
  Rem start counters 1+2, stop counters 3+4 and PWM1-4
  Cnt_Enable(11b)

Event:
  Cnt_Latch(11b)               'latch both counters 1+2
  new_1 = Cnt_Read_Latch(1)'read latch A of counter 1 and
  new_2 = Cnt_Read_Latch(2)' latch A of counter 2.
  Par_1 = new_1 - old_1 'caluclate the difference (f =
                               'impulses/time)
  Par_2 = new_2 - old_2 ' -"-
  old_1 = new_1                'Save new counter values
  old_2 = new_2                ' -"-
```

Cnt_SE_Diff

T11 TiCo

Cnt_SE_Diff sets all counter inputs to input mode single-ended or differential.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
Cnt_SE_Diff(pattern)
```

Parameter

pattern	Bit pattern to set the counter input modes: Bit = 0: Run input single-ended. Bit = 1: Run input differential.	LONG

Bit no. in pattern	31 ... 4	3	2	1	0
Inputs of counter no.	–	4	3	2	1

Notes

After start-up, counter inputs are set to input mode single-ended.

See also

Cnt_Clear, Cnt_Enable, Cnt_Get_Status, Cnt_Latch, Cnt_Mode, Cnt_Read, Cnt_Read_Latch

Valid for

Gold II-CNT

Example

```
Rem Please select the appropriate include for ADbasic /
 TiCoBasic
#Include ADwinGoldII.inc 'for ADbasic
Rem #Include GoldIITiCo.inc     'for TiCoBasic
Dim error As Long
Dim old_1, new_1 As Long
Dim old_2, new_2 As Long

Init:
  Cnt_Enable(0)                 'stop all counters
  Cnt_SE_Diff(0001b)            'set counter 1 diff. (2-4 single
                                'ended)
  Cnt_Mode(1,10000b)            'counter 1: Mode clock-direction
  Cnt_Clear(0001b)              'set counter 1 to 0
  Cnt_Enable(0001b)             'start counter 1
  old_1 = 0                     'initialize
  old_2 = 0
  error = 0                     'reset error flag

Event:
  Par_1 = Cnt_Read(1)           'read counter 1
  Rem read status register and mask out
  Par_2 = Cnt_Get_Status(1) And 01Fh
  If (Par_2 And 01000b = 01000b) Then 'line error counter 1?
    Inc Par_3                   'number of line errors up to now
    error = 1                   'set error flag
  EndIf
  If (Par_2 And 10000b = 10000b) Then 'correlation error
    Inc Par_4                   'number correlation error up to
                                'now
    error = 1                   'set error flag
  EndIf
  Par_5 = Shift_Right(Par_2 And 100b,2) 'status CLR input
  Par_6 = Par_2 And 1b          'status input A
  Par_7 = Shift_Right(Par_2 And 10b,1) 'status input B
```

Cnt_Sync_Latch

T11 TiCo

Cnt_Sync_Latch copies the contents of the selected counters and PWN counters into a buffer.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
Cnt_Sync_Latch(pattern)
```

Parameters

pattern	Bit pattern Bit = 0: No function. Bit = 1: Copy counter content into a buffer.	LONG

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

Notes

Each bit is assigned to both a VR counter and a PWM counter. For each set bit, both counter contents copied simultaneously. The instruction therefore has the same function as **Cnt_Latch** and **Cnt_PW_Latch** together.

The buffers can be read e.g. with **Cnt_Read_Latch** or **Cnt_Get_PW**.

See also

Cnt_Get_PW, Cnt_Latch, Cnt_Mode, Cnt_Latch, Cnt_Read_Int_Register, Cnt_PW_Latch

Valid for

Gold II-CNT

Example

```
Rem Please select the appropriate include for ADbasic /
 TiCoBasic
#Include ADwinGoldII.inc 'for ADbasic
Rem #Include GoldIITiCo.inc     'for TiCoBasic
#Define frequency FPar_1
Dim time As Long
Dim edges As Long
Dim oldpw As Long
Dim oldcnt As Long
Dim newpw As Long
Dim newcnt As Long
Dim pw_cnt As Long

Init:
  Processdelay = 3000000'100Hz
  Cnt_Enable(0001b)
  Cnt_Mode(1,0b)              'mode: clock/dir
  Cnt_Clear(0Fh)
  Cnt_Enable(0F0Fh)           'enable standard and PWM
                              'counters
  Cnt_PW_Latch(0Fh)           'copy PWM counter values
  oldpw = 0
  oldcnt = 0
  frequency = 0

Event:
  REM latch values of counter 1 (both standard and PWM)
  Cnt_Sync_Latch(0001b)
  newcnt = Cnt_Read_Latch(1) 'value of clock/dir counter
  edges = (newcnt-oldcnt) 'number of edges between events
  If (edges <> 0) Then
    pw_cnt = Cnt_Read_Int_Register(1,8) 'positive edges latch 1
    time = pw_cnt - oldpw     'calculate timebase
    frequency = edges*100000000/time  'frequency
                              '(100000000=timer frequency)
    oldcnt=newcnt             'store VR-counter value
    oldpw =newpw              'store PW-counter value
  EndIf
```

15.5 SSI interface

This section describes instructions to access SSI decoders of *ADwin-Gold II*:

SSI_Mode

T11 TiCo

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

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.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 the mode "continuous", reading the encoder is started immediately. **SSI_Start** is not necessary for this.

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 II-CNT

Example

```
Rem Please select the appropriate include for ADbasic /
 TiCoBasic
#Include ADwinGoldII.inc / GoldIITiCo.inc
Rem Decoder 1 runs with 2.5 MHz, Decoder 2 with 1.0 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

T11 TiCo

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

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.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 II-CNT

Example

```
Rem Please select the appropriate include for ADbasic /
 TiCoBasic
#Include ADwinGoldII.inc / GoldIITiCo.inc
Rem Decoder 1 runs with 500 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

T11 TiCo

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 ADwinGoldII.inc / GoldIITiCo.inc
SSI_Set_Bits(dcdr_no,bit_no)
```

Parameters

dcdr_no	Number (1…4) of the SSI decoder whose resolution is to be set.	LONG
bit_no	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 II-CNT

Example

```
Rem Please select the appropriate include for ADbasic /
 TiCoBasic
#Include ADwinGoldII.inc / GoldIITiCo.inc
Rem Decoder 1 runs with 2.5 MHz, Decoder 2 with 1.0 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 (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

T11 TiCo

SSI_Set_Clock sets the clock rate (approx. 100 kHz to 2.5 MHz), with which the encoder is clocked.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
SSI_Set_Clock(dcdr_no,prescale)
```

Parameters

dcdr_no	Number (1…4) of the SSI decoder, the clock rate of which is to be set.	LONG
prescale	Scale factor (10…255) for setting the clock rate according to the equation: Clock rate = 25 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 II-CNT

Example

```
Rem Please select the appropriate include for ADbasic /
 TiCoBasic
#Include ADwinGoldII.inc / GoldIITiCo.inc
Rem Decoder 1 runs with 2.5 MHz, Decoder 2 with 1.0 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 encoder
                               '1+2
  SSI_Set_Bits(1,10)           '10 encoder bits for encoder 1
  SSI_Set_Bits(2,22)           '22 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

T11 TiCo

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

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
SSI_Start(pattern)
```

Parameters

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

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

Notes

In the continuous mode this instruction has no function, because the encoder values are nevertheless read out continuously.

An encoder value will be saved only when the amount of bits is read which is set by **SSI_Set_Bits**.

A complete encoder value is always transferred, even if the operation mode is changing meanwhile.

See also

SSI_Mode, SSI_Read, SSI_Set_Bits, SSI_Set_Clock, SSI_Status

Valid for

Gold II-CNT

Example

```
Rem Please select the appropriate include for ADbasic /
 TiCoBasic
#Include ADwinGoldII.inc / GoldIITiCo.inc
Rem Both decoders run with 100 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, then …
  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, then …
  Par_2 = SSI_Read(2)           'read out and display position
                                'value
```

SSI_Status

T11 TiCo

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

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.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 II-CNT

Example

```
Rem Please select the appropriate include for ADbasic /
 TiCoBasic
#Include ADwinGoldII.inc / GoldIITiCo.inc
Rem Both decoders run with 100 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 completely
                                'read, then …
  Par_1 = SSI_Read(1)           'Read out and display position
                                'value
  Do                            'For encoder 2:
  Until (SSI_Status(2) = 0)     'If position value is completely
                                'read, then …
  Par_3 = SSI_Read(2)           'Read out and display position
                                'value
```

15.6 PWM Outputs

This section describes instructions to access PWM outputs of *ADwin-Gold II*:

PWM_Enable

T11 TiCo

PWM_Enable enables or disables one or more PWM outputs.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
PWM_Enable(pattern)
```

Parameters

pattern	Bit pattern for selection of PWM outputs. Bit = 0: Disable PWM output. Bit = 1: Enable PWM output.	LONG

Bit no.	31…6	5	4	3	2	1	0
PWM output	–	6	5	4	3	2	1

Notes

The time, when the PWM outputs are disabled–at once or after the next end of period–depends on the setting which was done with **PWM_Init** (parameter mode).

See also

PWM_Get_Status, PWM_Init, PWM_Latch, PWM_Reset, PWM_Standby_Value, PWM_Write_Latch

Valid for

Gold II-CNT

Example

see PWM_Init (page 139)

PWM_Get_Status

T11 TiCo

PWM_Get_Status returns the operation status of all PWM outputs.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
ret_val = PWM_Get_Status()
```

Parameters

ret_val — Bit pattern with status bits of all PWM ouputs. LONG
Bit = 0: PWM output has finished.
Bit = 1: PWM output is running.

Bit no.	31…6	5	4	3	2	1	0
PWM output	–	6	5	4	3	2	1

Notes

- / -

See also

PWM_Enable, PWM_Init, PWM_Latch, PWM_Reset, PWM_Standby_Value, PWM_Write_Latch

Valid for

Gold II-CNT

Example

- / -

PWM_Init

T11 TiCo

PWM_Init sets the defaults for one PWM output.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
PWM_Init(pwm_
    output,startdelay,startvalue,mode,count)
```

Parameters

pwm_output	Number (1…6) of PWM output.	LONG
startdelay	Start delay in units of 20ns.	LONG
startvalue	Start level of PWM output: 0: TTL-level low. 1: TTL-level high.	LONG
mode	Operating mode of PWM outputs as bit pattern (bits 0…2 only). Bit 0: Moment to take over a new PW frequency: Bit = 0: Take over at end of period. Bit = 1: Take over immediately. Bit 1: Number of pulses: Bit = 0: infinite number of pulses. Bit = 1: number of pulses is count. Bit 2: Moment to stop after stop instruction: Bit = 0: Stop at end of period. Bit = 1: Stop immediately.	LONG
count	Number of periods (1…32768), which are processed during an output cycle.	LONG

Notes

The defaults get active as soon as the PWM outputs are enabled with **PWM_Enable**.

A change of defaults during PWM output is not possible. Instead, the PWM outputs must be stopped with **PWM_Reset** or disabled with **PWM_Enable** in order to change defaults. Afterwards the PWM outputs are enabled again.

See also

PWM_Enable, PWM_Get_Status, PWM_Latch, PWM_Reset, PWM_Standby_Value, PWM_Write_Latch

Valid for

Gold II-CNT

Example

```
Rem Please select the appropriate include for ADbasic /
 TiCoBasic
#Include ADwinGoldII.inc 'for ADbasic
Rem #Include GoldIITiCo.inc     'for TiCoBasic
#Define freq1 FPar_1
#Define freq2 FPar_2
#Define pw1 FPar_3
#Define pw2 FPar_4
Dim channel As Long

Init:
  freq1 = 1000                  '1000 Hz
  freq2 = 2000                  '2000 Hz
  pw1 = 50                      '50 %
  pw2 = 70                      '70 %
  PWM_Reset(011b)               'stop channels 1 und 2

  For channel = 1 To 2
    PWM_Init(channel,0,0,0,0)
  Next

  PWM_Write_Latch(1,pw1,freq1)
  PWM_Write_Latch(2,pw2,freq2)
  PWM_Latch(11b)
  PWM_Enable(011b)              'start output

Event:
  PWM_Write_Latch(1,pw1,freq1)
  PWM_Write_Latch(2,pw2,freq2)

  PWM_Latch(11b)
```

PWM_Latch

T11 TiCo

PWM_Latch enables frequency and duty cycle of one or more PWM outputs to be output.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
PWM_Latch(pattern)
```

Parameters

pattern	Bit pattern to select PWM outputs: Bit = 0: No influence. Bit = 1: latch = enable for output.	LONG

Bit no.	31…6	5	4	3	2	1	0
PWM output	–	6	5	4	3	2	1

Notes

PWM_Write_Latch writres frequency and duty cycle into the latch register. Only when **PWM_Latch** is processed the latch values are started to be output.

The time, when the output of the new values starts–at once or after the next end of period–depends on the setting which was done with **PWM_Init** (parameter mode).

See also

PWM_Enable, PWM_Get_Status, PWM_Init, PWM_Reset, PWM_Standby_Value, PWM_Write_Latch

Valid for

Gold II-CNT

Example

see PWM_Init (page 139)

PWM_Reset

T11 TiCo

PWM_Reset stops the output of one or more PWM outputs immediately.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
PWM_Reset(pattern)
```

Parameters

pattern	Bit pattern to select the PWM outputs: Bit = 0: No influence Bit = 1: Stop output immediately.	LONG

Bit no.	31…6	5	4	3	2	1	0
PWM output	–	6	5	4	3	2	1

Notes

The output will be stopped immediately even when **PWM_Init** has set a different stop mode.

See also

PWM_Enable, PWM_Get_Status, PWM_Init, PWM_Latch, PWM_Standby_Value, PWM_Write_Latch

Valid for

Gold II-CNT

Example

see PWM_Init (page 139)

PWM_Standby_Value

T11 TiCo

PWM_Standby_Value sets the default TTL levels for all PWM outputs.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
PWM_Standby_Value(pattern)
```

Parameters

`pattern` Bit pattern to select the default TTL level of the PWM outputs: LONG
Bit = 0: TTL-level low
Bit = 1: TTL-level high

Bit no.	31…6	5	4	3	2	1	0
PWM output	–	6	5	4	3	2	1

Notes

Using **PWM_Standby_Value**, PWM outputs may be used as simple TTL outputs.

If a PWM output is disabled with **PWM_Enable**, the output is set to default level from `pattern`. The default level will also be set after the PWM output has stopped.
After power-up the outputs are set to TTL-level low.

See also

PWM_Enable, PWM_Get_Status, PWM_Init, PWM_Latch, PWM_Reset, PWM_Write_Latch

Valid for

Gold II-CNT

Example

- / -

PWM_Write_Latch

T11 TiCo

PWM_Write_Latch writes frequency and duty cycle into the latch register.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
PWM_Write_Latch(pwm_output, dutycycle, frequency)
```

Parameters

pwm_output	Number (1…6) of PWM output.	LONG
dutycycle	Duty cycle / inverse duty cycle in percent between 0.0 and 100.0 (do not use 0.0 or 100.0).	FLOAT
frequency	Frequency in Hertz: 0.05 Hz …25000 kHz.	FLOAT

Notes

PWM_Write_Latch writes frequency and duty cycle into the latch register only. The values are enabled for PWM output with **PWM_Latch** only.

The value of dutycycle depends on the setting of the parameter startvalue from the instruction **PWM_Init:**

- startvalue = 1: Set dutycycle to the value of the duty cycle.
- startvalue = 0: Set dutycycle to the "inverse duty cycle": dutycycle = 100% - duty cycle

The highest output frequency where the duty cycle can be still defined in 1%-steps, is 500 kHz.

See also

PWM_Enable, PWM_Get_Status, PWM_Init, PWM_Latch, PWM_Reset, PWM_Standby_Value

Valid for

Gold II-CNT

Example

see PWM_Init (page 139)

15.7 CAN interface

This section describes instructions to access CAN interfaces of *ADwin-Gold II*:

CAN_Msg

T11 TiCo

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

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.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. in **CAN_Msg**	1…8	9
Contents	Message object(s) = databyte(s)	Number (0…8) of allocated databytes

Enter the data bytes to be transferred and their number into the field **CAN_Msg**[], *before* transferring them with **Transmit**.

See also

En_CAN_Interrupt, En_Receive, En_Transmit, Get_CAN_Reg, Init_CAN, Read_Msg, Set_CAN_Baudrate, Set_CAN_Reg, Transmit

Valid for

Gold II-CAN

Example

```
Rem Please select the appropriate include for ADbasic /
 TiCoBasic
#Include ADwinGoldII.inc / GoldIITiCo.inc
REM Send 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
  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
```

Receive a float value see example at Read_Msg

En_CAN_Interrupt

T11 TiCo

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

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.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, En_Transmit

Valid for

Gold II-CAN

Example

```
Rem Please select the appropriate include for ADbasic /
 TiCoBasic
#Include ADwinGoldII.inc / GoldIITiCo.inc
Init:
  Init_CAN(1)                  'Initialize CAN controller 1
  En_Receive(1,1,200,0)        'Initialize message object 1 of
                               'controller 1
                               'to receive CAN messages with
                               'identifier 200
  En_CAN_Interrupt(1,1)        'Enable triggering of interrupts
                               '(ext. EVENT) when receiving the
                               'message object 1
```

En_Receive

T11 TiCo

En_Receive enables a specified message object to receive messages.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.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 the 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

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.

See also

CAN_Msg, En_Transmit, Get_CAN_Reg

Valid for

Gold II-CAN

Example

```
Rem Please select the appropriate include for ADbasic /
 TiCoBasic
#Include ADwinGoldII.inc / GoldIITiCo.inc

Init:
  Init_CAN(1)                  'Initialization CAN controller 1
  En_Receive(1,1,200,0)        'Initialize message object 1 to
                               'receive CAN messages with
                               'identifier 200
```

En_Transmit

T11 TiCo

En_Transmit enables a specified message object to send messages.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.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 the 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, Get_CAN_Reg

Valid for

Gold II-CAN

Example

```
Rem Please select the appropriate include for ADbasic /
 TiCoBasic
#Include ADwinGoldII.inc / GoldIITiCo.inc

Init:
  Init_CAN(1)                  'Initialize CAN-Controller 1
  Rem Initialize message objects of interface 1:
  Rem Object 2 for receive with identifier 200,
  Rem Object 6 for sending with identifier 40
  En_Receive(1,1,200,0)
  En_Transmit(1,6,40,0)
```

Get_CAN_Reg

T11 TiCo

Get_CAN_Reg reads the value of a specified register in the CAN controller of a CAN interface.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.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 in lower 8 bits).	LONG

Notes

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

See also

Set_CAN_Baudrate, Set_CAN_Reg

Valid for

Gold II-CAN

Example

```
Rem Please select the appropriate include for ADbasic /
 TiCoBasic
#Include ADwinGoldII.inc / GoldIITiCo.inc
Init:
  Init_CAN(1)                  'Initialize CAN controller 1
  Par_1 = Get_CAN_Reg(1,0)'Read out control register
```

Init_CAN

T11 TiCo

Init_CAN initializes the CAN controller.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.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_Receive, En_Transmit, Get_CAN_Reg

Valid for

Gold II-CAN

Example

```
Rem Please select the appropriate include for ADbasic /
 TiCoBasic
#Include ADwinGoldII.inc / GoldIITiCo.inc

Init:
  Init_CAN(1)                  'Initialize CAN controller 1 on
                               'CAN module 1
```

Read_Msg

T11 TiCo

Read_Msg checks if new messages have 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 ADwinGoldII.inc / GoldIITiCo.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 the message object.	LONG
ret_val	-1: No new message. >0:New message; 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_Receive, En_Transmit, Get_CAN_Reg, Read_Msg_Con

Valid for

Gold II-CAN

Example

```
Rem Please select the appropriate include for ADbasic /
 TiCoBasic
#Include ADwinGoldII.inc / GoldIITiCo.inc
REM If a new message with the correct identifier is received
REM the data is read out. The first 4 bytes of the message are
REM combined to a float value of length 32 bit. (Sending a
REM float value see example of Transmit).
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(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

T11 TiCo

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 ADwinGoldII.inc / GoldIITiCo.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 II-CAN

Example

```
Rem Please select the appropriate include for ADbasic /
 TiCoBasic
#Include ADwinGoldII.inc / GoldIITiCo.inc
REM If a new message with the correct identifier is received
REM the data is read out. The first 4 bytes of the message are
REM combined to a float value of length 32 bit. (Sending a
REM float value see example of Transmit).
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

T11 TiCo

Set_CAN_Baudrate sets the Baud rate of the CAN controller.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.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 "Baud rates for CAN bus" (Annex, page A-7). Please use the table's notation exactly, i.e. non-integer baud rates with 4 decimal places; values with different notation will be rejected as not allowed.

The instruction executes the following actions:

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

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

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

See also

Get_CAN_Reg, Set_CAN_Reg

Valid for

Gold II-CAN

Example

```
Rem Please select the appropriate include for ADbasic /
 TiCoBasic
#Include ADwinGoldII.inc / GoldIITiCo.inc

Init:
  Init_CAN(1)                  'Initialize CAN controller 1
  Par_1 = Set_CAN_Baudrate(1,125000)'Set Baud rate to 125
                               'kBit/s
```

Set_CAN_Reg

T11 TiCo

Set_CAN_Reg writes a value into a specified register of the CAN controller.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.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

Set_CAN_Baudrate, Get_CAN_Reg

Valid for

Gold II-CAN

Example

```
Rem Please select the appropriate include for ADbasic /
 TiCoBasic
#Include ADwinGoldII.inc / GoldIITiCo.inc

Init:
  Init_CAN(1)                  'Initialize CAN controller 1
  Set_CAN_Reg(1,0,1)           'Set control register to the
                               'value 1
```

Transmit

T11 TiCo

TRANSMIT sends the message in **CAN_Msg** via the specified message object.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
Transmit(can_no,msg_no)
```

Parameters

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

Notes

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

- Once: Enable the message object with **En_Transmit** for sending.
- As often as needed: 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

Init_CAN, Read_Msg, En_Transmit

Valid for

Gold II-CAN

Example

```
Rem Please select the appropriate include for ADbasic /
 TiCoBasic
#Include ADwinGoldII.inc / GoldIITiCo.inc
REM Send 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(2)                  'Initialize CAN controller

  REM Enable message object 6
  REM for sending with the identifier 40 (11 bit)
  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)                'Send message object 6
```

Receiving a float value see example at Read_Msg.

15.8 RSxxx interface

This section describes instructions to access RSxxx interfaces of *ADwin-Gold II*:

Check_Shift_Reg

T11 TiCo

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

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
ret_val = Check_Shift_Reg(interface)
```

Parameters

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

Notes

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

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

See also

Get_RS, RS_Init, RS_Reset, Write_FIFO

Valid for

Gold II-CAN

Example

```
Rem Please select the appropriate include for ADbasic /
 TiCoBasic
#Include ADwinGoldII.inc / GoldIITiCo.inc

Event:
  Rem …
  Rem check if RSxxx interface 1 still has data to send
  Par_1 = Check_Shift_Reg(1)
  Rem …
```

Get_RS

T11 TiCo

Get_RS reads out a specified controller register.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.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 II-CAN

Example

- / -

Read_FIFO

T11 TiCo

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

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.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 II-CAN

Example

```
Rem Please select the appropriate include for ADbasic /
 TiCoBasic
#Include ADwinGoldII.inc / GoldIITiCo.inc

Init:
  RS_Reset()
  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.

Event:
  Par_1 = Read_FIFO(1)         'Get a value from the FIFO. If
                               'the FIFO is empty, -1 is
                               'returned.
```

RS485_Send

T11 TiCo

RS485_Send determines the transfer direction for a specified RSxxx interface.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.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 II-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)

T11 TiCo

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.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: RS232: 35 … 115,200 RS485: 35 … 2,304,000	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 **RS_Reset**.

If transfer protocol RS485 is set, the transfer direction must be set, too (with **RS485_Send**).

You find a list of standard baud rates on page 35 (fig. 23).

See also

Check_Shift_Reg, Get_RS, RS485_Send, RS_Reset, Set_RS

Valid for

Gold II-CAN

Example

```
Rem Please select the appropriate include for ADbasic /
 TiCoBasic
#Include ADwinGoldII.inc / GoldIITiCo.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

T11 TiCo

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

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.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 II-CAN

Example

```
Rem Please select the appropriate include for ADbasic /
 TiCoBasic
#Include ADwinGoldII.inc / GoldIITiCo.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

T11 TiCo

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

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.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 II-CAN

Example

- / -

Write_FIFO

T11 TiCo

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

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.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 II-CAN

Example

```
Rem Please select the appropriate include for ADbasic /
 TiCoBasic
#Include ADwinGoldII.inc / GoldIITiCo.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).
```

Write_Fifo_Full

T11 TiCo

Write_Fifo_Full returns if there is at least one free element in the send-Fifo of a specified channel on the specified module.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
ret_val = Write_Fifo_Full(interface)
```

Parameters

interface	Channel number to whose send-Fifo data are transferred (1, 2 or 1…4).	LONG
ret_val	Status message: 0: Data are transferred successfully. 1: Data were not transferred, send-Fifo is full.	LONG

Notes

The return value is the same as with **Write_Fifo**.

See also

Check_Shift_Reg, Get_RS, Read_FIFO, RS_Init, RS_Reset, RS485_Send, Set_RS, Write_FIFO

Valid for

Gold II-CAN

Example

```
Rem sending data to and receiving data from the PC while using
Rem a Fifo in ADwin-Gold II
Rem Please select the appropriate include for ADbasic / TiCoBasic
#Include ADwinGoldII.inc / GoldIITiCo.inc

#Define outfifo Data_1
#Define infifo Data_2
#Define rs_adr 5
#Define rs_channel 1

Dim outfifo[1000] As Long As Fifo
Dim infifo[1000] As Long As Fifo
Dim value, dummy, check As Long

Rem use LED as signal: red = sending, green = receiving,
Rem orange (red+green) = sending + receiving
Dim red_led, green_led As Long
Dim green_led_time As Long
Dim led_time As Long

Init:
  Rem reset and initialize interface
  RS_Reset(rs_adr)
  RS_Init(rs_adr, 1, 9600, 0, 8, 0, 0)
  Fifo_Clear(1)
  Fifo_Clear(2)
  green_led = 0
  red_led = 0

Event:
  Rem sending
  If (Fifo_Full(1) > 0) Then 'any data present?
    If (Write_Fifo_Full(rs_adr, rs_channel) = 0) Then
      Rem send Fifo empty?
      value = outFifo         'read value from Fifo
      dummy = Write_Fifo(rs_adr, rs_channel, value)
      Rem dummy is not to be checked, since Write_Fifo_Full has
      Rem proved that Fifo has empty elements.

      'do LED settings
      If (red_led = 0) Then
        red_led = 1
        led_time = Read_Timer()
      EndIf

    EndIf
  EndIf

  Rem receiving
  If (Fifo_Empty(2) > 0) Then 'are there empty elements?
    check = Read_Fifo(rs_adr, rs_channel)

    If (check <> -1) Then   'is a value in the receiving buffer?
      inFifo = check        'get value into inFifo

      'do LED settings
      If (green_led = 0) Then
        green_led = 1
        led_time = Read_Timer()
      EndIf
    EndIf
  EndIf

  'output LED settings
```

```
RS_Set_LED(rs_adr, rs_channel, (red_led And 1) Or
(Shift_Left((green_led And 1), 1)))
If ((red_led > 0) Or (green_led > 0)) Then
 If ((Read_Timer() - led_time) > 20000000) Then
   If (red_led > 0) Then Inc red_led
   If (green_led > 0) Then Inc green_led
   led_time = Read_Timer()
  EndIf
 EndIf

If ((red_led = 3) Or (green_led = 3)) Then
 red_led = 0
 green_led = 0
EndIf
```

15.9 Profibus interface

This section describes instructions to access a Profibus interface of *ADwin-Gold I*:

Init_Profibus

T11

Init_Profibus initializes the Profibus Slave.

Syntax

```
#Include ADwinGoldII.Inc
ret_val = Init_Profibus(dev_adr, in_mod_cnt,
    in_mod_type, out_mod_cnt, out_mod_type,
    work_arr[], info[])
```

Parameters

dev_adr	Slave node address / station address (1…125) on the Profibus.	LONG
in_mod_cnt	Number (0…76) of input areas in the Profibus Slave. The max. number depends on in_mod_type.	LONG
in_mod_type	Key number (1…3, 16) for the length of input areas: 1: 1 Byte; max. value for in_mod_cnt: 76. 2: 2 Byte; max. value for in_mod_cnt: 38. 3: 4 Byte; max. value for in_mod_cnt: 19. 16:8 Byte; max. value for in_mod_cnt: 9.	LONG
out_mod_cnt	Number (0…76) of output areas in the Profibus Slave. The max. number depends on out_mod_type.	LONG
out_mod_type	Key number (1…3, 16) for the length of output areas: 1: 1 Byte; max. value for out_mod_type: 76. 2: 2 Byte; max. value for out_mod_type: 38. 3: 4 Byte; max. value for out_mod_type: 19. 16:8 Byte; max. value for out_mod_type: 9.	LONG
work_arr[]	Array to store data for operation of the Profibus Slave. The array must have at least 200 elements.	ARRAY LONG
info[]	Array holding data about the Profibus Slave. The array must have at least 10 elements. The elements info[1] and info[2] contain the production type of the Profibus Slave: info[1]=1, info[2]=4	ARRAY LONG
ret_val	State of initialization: 0: no error. ≠0: Error; please contact the support of Jäger Messtechnik.	LONG

Notes

This instruction must be processed before working with Profibus Slave.

Init_Profibus should be processed in a program section with low priority, because of the long processing time (about 2-3 seconds). Using the instruction in a (non-interruptable) high priority process, the communication between PC and *ADwin* system would be interrupted too long and thus produce an error message (timeout).

Station address, number and length of modules must equal the project settings of the profibus. For projecting, the module length is also given in words: 1 word = 2 byte.

Valid for

Gold II-Profibus

See also

Run_Profibus

Example

```
#Include ADwinGoldII.INC
#Define node 2              'slave node address
#Define info Data_1         'info array
#Define out_arr Data_2
#Define in_arr Data_3

Dim out_arr[76] As Long At DM_Local
Dim in_arr[76] As Long At DM_Local
Dim conf_arr[200] As Long At DM_Local
Dim info[10] As Long At DM_Local
Dim i As Long
Dim error As Long

LowInit:
  Processdelay = 3000000    'set to 100 Hz
  For i = 1 To 10           'initialize info array
    info[i] = 0
  Next i
  Rem initialize profibus interface: 38 input data areas of 2 byte
  Rem and 76 output data bytes of 1 Byte
  error = Init_Profibus(node,38,2,76,1,conf_arr,info)
  If (error <> 0) Then      'initialization error
    Par_1 = error
    Exit
  EndIf

Event:
  Rem set data in out_arr[] to be transferred
  For i = 1 To 76
    out_arr[i] = (out_arr[i] + i) And 0FFh
  Next i

  Rem send and read data (output bytes: 76; input bytes: 76)
  error = Run_Profibus(out_arr,76,in_arr,76,conf_arr)And 7h

  Rem here the received data in in_arr[] can be processed
```

Run_Profibus

T11

Run_Profibus exchanges data with the Profibus Slave.

Syntax

```
#Include ADwinGoldII.Inc
ret_val = Run_Profibus(out_pd_arr[], out_pd_arr_len,
    in_pd_arr[], in_pd_arr_len, work_arr[])
```

Parameters

out_pd_arr[]	Array from which the Profibus Slave reads data and writes them to the Profibus.	ARRAY LONG
out_pd_arr_len	Number of output bytes (1…76), the data of which are read from the array out_pd_arr[]. The number may not be greater than given in out_mod_cnt with **Init_Profibus**.	LONG
in_pd_arr[]	Array into which the Profibus-Slave writes data, which are read by the Profibus.	ARRAY LONG
in_pd_arr_len	Number of input bytes (1…76), the data of which are returned in the array in_pd_arr[]. The number may not be greater than given in in_mod_cnt with **Init_Profibus**.	LONG
work_arr[]	Array holding data for operation of the Profibus Slave, see **Init_Profibus**.	ARRAY LONG
ret_val	Bit pattern holding the state of operation of the Profibus Slave. Only bits Bits 0…2 are important: **100b**: Slave is active and runs correcty. **010b**: Profibus inactive, Slave is waiting. **110b**, **111b**: Error.	LONG

Notes

Run_Profibus should be processed in a program section with low priority, because of the long processing time. Using the instruction in a (non-interruptable) high priority process, the communication between PC and *ADwin* system would be interrupted too long and thus produce an error message (timeout).

Each array element in in_pd_arr[] and out_pd_arr[] contains a single data byte only (bits 0…7). Data areas of more than one byte length are saved in the appropriate number of consecutive array elements. Example: 5 data areas of 4 byte length are stored in 5×4=20 array elements.

Valid for

Gold II-Profibus

See also

Init_Profibus

Example

see Init_Profibus

15.10DeviceNet interface

This section describes instructions to access a DeviceNet interface of *ADwin-Gold II*:

Init_DeviceNet

T11

Init_DeviceNet initializes the DeviceNet Slave.

Syntax

```
#Include ADwinGoldII.Inc
ret_val = Init_DeviceNet(dev_adr, baudrate,
    in_mod_cnt, in_mod_type, out_mod_cnt,
    out_mod_type, work_arr[], info[])
```

Parameters

dev_adr	Slave node address / station address (1…125) on the DeviceNet.	LONG
baudrate	Key number for the baudrate, the DeviceNet bus is run with: 0: 125 kBit 1: 250 kBit 2: 500 kBit 3: automatic adaption to the current baudrate of the DeviceNet bus (autobaud).	LONG
in_mod_cnt	Number (1…255) of input areas in the DeviceNet Slave. The max. number depends on in_mod_type.	LONG
in_mod_type	Key number (1…3, 16) for the length of input areas: 1: 1 Byte; max. value for in_mod_cnt: 255. 2: 2 Byte; max. value for in_mod_cnt: 127. 3: 4 Byte; max. value for in_mod_cnt: 63. 16:8 Byte; max. value for in_mod_cnt: 31.	LONG
out_mod_cnt	Number (0…76) of output areas in the DeviceNet Slave. The max. number depends on out_mod_type.	LONG
out_mod_type	Key number (1…3, 16) for the length of output areas: 1: 1 Byte; max. value for out_mod_type: 255. 2: 2 Byte; max. value for out_mod_type: 127. 3: 4 Byte; max. value for out_mod_type: 63. 16:8 Byte; max. value for out_mod_type: 31.	LONG
work_arr[]	Array to store data for operation of the DeviceNet Slave. The array must have at least 200 elements.	ARRAY LONG
info[]	Array holding data about the DeviceNet Slave. The array must have at least 10 elements. The elements info[1] and info[2] contain the production type of the DeviceNet Slave: info[1]=1, info[2]=4	ARRAY LONG
ret_val	State of initialization: 0: no error. ≠0: Error; please contact the support of Jäger Messtechnik.	LONG

Notes

This instruction must be processed before working with DeviceNet Slave.

Init_DeviceNet should be processed in a program section with low priority, because of the long processing time (about 2-3 seconds). Using

the instruction in a (non-interruptable) high priority process, the communication between PC and *ADwin* system would be interrupted too long and thus produce an error message (timeout).

Station address, number and length of modules must equal the project settings of the DeviceNet. For projecting, the module length is also given in words: 1 word = 2 byte.

Valid for

Gold II-DeviceNet

See also

Run_DeviceNet

Example

```
#Include ADwinGoldII.INC
#Define node 2              'slave node address
#Define info DATA_1         'info array
#Define out_arr DATA_2
#Define in_arr DATA_3

Dim out_arr[255] As Long At DM_Local
Dim in_arr[254] As Long At DM_Local
Dim conf_arr[200] As Long At DM_Local
Dim info[10] As Long At DM_Local
Dim i As Long
Dim error As Long

LowInit:
  Processdelay = 3000000    'set to 100 Hz
  For i = 1 To 10           'initialize info array
    info[i] = 0
  Next i
  Rem initialize DeviceNet interface: autobaud, 127 input data
  Rem areas of 2 byte and 255 output data bytes of 1 Byte
  error = Init_DeviceNet(node,3,127,2,255,1,conf_arr,info)
  If (error <> 0) Then      'initialization error
    PAR_1 = error
    Exit
  EndIf

Event:
  Rem set data in out_arr[] to be transferred
  For i = 1 To 255
    out_arr[i] = (out_arr[i] + i) And 0FFh
  Next i

  Rem send and read data (output bytes: 255; input bytes: 254)
  error = Run_DeviceNet(out_arr,255,in_arr,254,conf_arr) And 7
  PAR_2 = error

  Rem here the received data in in_arr[] can be processed
```

Run_DeviceNet

T11

Run_DeviceNet exchanges data with the DeviceNet Slave.

Syntax

```
#Include ADwinGoldII.Inc
ret_val = Run_DeviceNet(out_pd_arr[],
    out_pd_arr_len, in_pd_arr[], in_pd_arr_len,
    work_arr[])
```

Parameters

out_pd_arr[]	Array from which the DeviceNet Slave reads data and writes them to the DeviceNet.	ARRAY LONG
out_pd_arr_len	Number of output bytes (1…255), the data of which are read from the array out_pd_arr[]. The number may not be greater than given in out_mod_cnt with **Init_DeviceNet**.	LONG
in_pd_arr[]	Array into which the DeviceNet-Slave writes data, which are read by the DeviceNet.	ARRAY LONG
in_pd_arr_len	Number of input bytes (1…255), the data of which are returned in the array in_pd_arr[]. The number may not be greater than given in in_mod_cnt with **Init_DeviceNet**.	LONG
work_arr[]	Array holding data for operation of the DeviceNet Slave, see **Init_DeviceNet**.	ARRAY LONG
ret_val	Key number for the state of operation of the DeviceNet Slave: **0**: Slave is initialized. **2**: DeviceNet inactive, Slave is waiting. **4**: DeviceNet active. other values: Error.	LONG

Notes

Run_DeviceNet should be processed in a program section with low priority, because of the long processing time. Using the instruction in a (non-interruptable) high priority process, the communication between PC and *ADwin* system would be interrupted too long and thus produce an error message (timeout).

Each array element in in_pd_arr[] and out_pd_arr[] contains a single data byte only (bits 0…7). Data areas of more than one byte length are saved in the appropriate number of consecutive array elements. Example: 5 data areas of 4 byte length are stored in 5×4=20 array elements.

Valid for

Gold II-DeviceNet

See also

Init_DeviceNet

Example

see Init_DeviceNet

15.11EtherCAT interface

This section describes instructions to access an EtherCAT interface of *ADwin-Gold II*:

ECAT_Init

T11

ECAT_Init initializes the EtherCAT interface.

Syntax

```
#Include ADwinGoldII.inc
ret_val = ECAT_Init(in_mod_cnt, in_mod_type,
    out_mod_cnt, out_mod_type, work_arr[], info[])
```

Parameters

in_mod_cnt	Number (1…254) of input areas in the EtherCAT interface. The max. number depends on in_mod_type.	LONG
in_mod_type	Key number (1…3, 16) for the length of input areas: 1: 1 Byte; max. value for in_mod_cnt: 254. 2: 2 Byte; max. value for in_mod_cnt: 127. 3: 4 Byte; max. value for in_mod_cnt: 63. 16:8 Byte; max. value for in_mod_cnt: 31.	LONG
out_mod_cnt	Number (1…254) of output areas in the EtherCAT interface. The max. number depends on out_mod_type.	LONG
out_mod_type	Key number (1…3, 16) for the length of output areas: 1: 1 Byte; max. value for out_mod_type: 254. 2: 2 Byte; max. value for out_mod_type: 127. 3: 4 Byte; max. value for out_mod_type: 63. 16:8 Byte; max. value for out_mod_type: 31.	LONG
work_arr[]	Array to store data for operation of the EtherCAT interface. The array must have at least 200 elements.	ARRAY LONG
info[]	Array holding data about the EtherCAT interface. The array must have at least 10 elements. The elements info[1] and info[2] contain the production type of the EtherCAT interface: info[1]=1, info[2]=4	ARRAY LONG
ret_val	State of initialization: 0: no error. ≠0: Error; please contact the support of Jäger Messtechnik.	LONG

Notes

This instruction must be processed before working with the EtherCAT interface.

ECAT_Init should be processed in a program section with low priority, because of the long processing time (about 2-3 seconds). Using the instruction in a (non-interruptable) high priority process, the communication between PC and *ADwin* system would be interrupted too long and thus produce an error message (timeout).

If you configure the interface externally (e.g. using the program TwinCAT System Manager), you still have to configure the interface in *ADbasic* and use the very same settings.

Valid for

Gold II-EtherCAT

See also

ECAT_Run

Example

```
#Include ADwinGoldII.INC
#Define node 2              'slave node address
#Define info Data_1         'info array
#Define out_arr Data_2
#Define in_arr Data_3

Dim out_arr[76] As Long At DM_Local
Dim rx_arr[76] As Long At DM_Local
Dim conf_arr[200] As Long At DM_Local
Dim info[10] As Long At DM_Local
Dim i As Long
Dim error As Long

Init:
  Processdelay = 3000000    'set to 100 Hz
  For i = 1 To 10           'initialize info array
    info[i] = 0
  Next i
  Rem initialize EtherCAT interface: 38 input data areas of 2 byte
  Rem and 76 output data bytes of 1 Byte
  error = ECAT_Init(node,38,2,76,1,conf_arr,info)
  If (error <> 0) Then      'initialization error
    Par_1 = error
    Exit
  EndIf

Event:
  Rem set data in out_arr[] to be transferred
  For i = 1 To 76
    out_arr[i] = (out_arr[i] + i) And 0FFh
  Next i

  Rem send and read data (output bytes: 76; input bytes: 76)
  error = ECAT_Run(out_arr,76,in_arr,76,conf_arr)And 7h
  Par_2 = error

  Rem Here the received data in in_arr[] can be processed
```

ECAT_Run

T11

ECAT_Run exchanges data with the EtherCAT interface.

Syntax

```
#Include ADwinGoldII.inc
ret_val = ECAT_Run(in_pd_arr[], in_pd_arr_len,
    out_pd_arr[], out_pd_arr_len, work_arr[])
```

Parameters

in_pd_arr[]	Array from which the EtherCAT interface reads data and writes them to the EtherCAT bus.	ARRAY LONG
in_pd_arr_len	Number of input bytes (1…254), the data of which are returned in the array in_pd_arr[]. The number may not be greater than given in in_mod_cnt with **ECAT_Init**.	LONG
out_pd_arr[]	Array into which the EtherCAT slave writes data, which are read by the EtherCAT bus.	ARRAY LONG
out_pd_arr_len	Number of output bytes (1…254), the data of which are read from the array out_pd_arr[]. The number may not be greater than given in out_mod_cnt with **ECAT_Init**.	LONG
work_arr[]	Array holding data for operation of the EtherCAT slave, see **ECAT_Init**.	ARRAY LONG
ret_val	Bit pattern holding the state of operation of the EtherCAT slave. Only bits Bits 0…2 are important: **000b**: Interface is being initiallized. **010b**: Interface inactive and waiting. **100b**: Interface is active and runs correcty. **110b**, **111b**: Error, please check configuration.	LONG

Notes

ECAT_Run should be processed in a program section with low priority, because of the long processing time. Using the instruction in a (non-interruptable) high priority process, the communication between PC and *ADwin* system would be interrupted too long and thus produce an error message (timeout).

Each array element in in_pd_arr[] and out_pd_arr[] contains a single data byte only (bits 0…7). Data areas of more than one byte length are saved in the appropriate number of consecutive array elements. Example: 5 data areas of 4 byte length are stored in 5×4=20 array elements.

Valid for

Gold II-EtherCAT

See also

ECAT_Init

Example

see ECAT_Init

15.12Real-time clock

This section describes instructions to access real-time clock interfaces of *ADwin-Gold II*:

RTC_Get

T11 TiCo

RTC_Get returns date and time from the real-time clock.

Syntax

```
#Include ADwinGoldII.inc / GoldIITiCo.inc
RTC_Get(year,month,day,hour,minute,second)
```

Parameter

year	Year (0…99), corresponds to 2000…2099.	LONG
month	Month (1…12).	LONG
day	Day (1…31); valid value ranges according to month and leap year.	LONG
hour	Hour (0…23).	LONG
minute	Minute (0…59).	LONG
second	Second (0…59).	LONG

Notes

All parameters are return values; thus, you have to use variables as parameters.

See also

RTC_Set

Valid for

Gold II-Storage

Example

```
#Include ADwinGoldII.inc
 Dim year,mon,day,h,m,s As Long

Init:
 Rem read real-time clock
 RTC_Get(year,mon,day,h,m,s)
```

RTC_Set

T11

RTC_SET sets date and time on the real-time clock of the specified module. Invalid values are not accepted.

Syntax

```
#Include ADwinGoldII.inc
ret_val = RTC_Set(year,month,day,hour,minute,second)
```

Parameters

year	Year (0…99), corresponds to 2000…2099.	LONG
month	Month (1…12).	LONG
day	Day (1…31); valid value ranges according to month and leap year.	LONG
hour	Hour (0…23).	LONG
minute	Minute (0…59).	LONG
second	Second (0…59).	LONG
ret_val	Status for setting the real-time clock: 0: Date and time are set. -1: Error, real-time clock may only be set in a low-priority process. -2: Error, no real-time clock present.	LONG

Note

The instruction **RTC_Set** may only be used in a process section with low priority.

See also

RTC_Get

Valid for

Gold II-Storage

Example

```
#Include ADwinGoldII.inc

LowInit:
  REM Set real-time clock to July 4th, 2003 9:17:30
  Par_1 = RTC_Set(3,7,4,9,17,30)
```

15.13Storage media (ADbasic)

This section describes instructions to access the memory card of *ADwin-Gold II*:

Media_Init

T11

Media_Init initializes the memory card of *ADwin-Gold II*.

Syntax

```
#INCLUDE ADwinGoldII.Inc
ret_val = Media_Init(media_datatable[])
```

Parameters

media_datatable[]	Array, which contains data for the operation of the memory card. The array must have at least 100 elements.	ARRAY LONG
ret_val	Status of initialization: > 0: Initialization successful. ret_val contains the number of usable blocks on the memory card. < 0: An error has occurred. Set Bits in ret_val refer to the error cause, see table.	LONG

Notes

Media_Init may only be used in low priority process section:

- anywhere in a low priority process
- in the sections **LowInit** or **Finish** of a high priority process

If used in a (non-interruptable) high priority process the communication between PC and *ADwin* system would be stopped too long, and would therefore lead to an error message (timeout).

The instruction **Media_Init** is only available in *ADbasic*, but not in *TiCoBasic*.

See also

Media_Erase, Media_Write, Media_Read

Valid for

Gold II-Storage-16

Example

```
#Include ADwinGoldII.inc
Dim err, num_blocks As Long
Dim media_datatable[100] As Long

LowInit:
  Rem initialize media card
  num_blocks = Media_Init(media_datatable)
  If (num_blocks < 0) Then Exit
  err = Media_Erase(media_datatable)
```

Media_Erase

Media_Erase erases all data from the memory card of *ADwin-Gold II*.

T11

Syntax

```
#Include ADwinGoldII.inc
ret_val = Media_Erase(media_datatable[])
```

Parameters

media_datatable[]	Array containig data for the operation of the memory card, see **Media_Init**.	ARRAY LONG
ret_val	Status of erasing: = 0: The memory card has been completely erased. > 0: An error has occurred. Set Bits in ret_val refer to the error cause, see table.	LONG

Bit no.	Meaning
00	Bit = 1: Memory card does not reply.
01	Bit = 1: Timeout.
02:32	Bits may be set but cannot be used here.

Notes

Before using **Media_Erase**, the array media_datatable[]must be initialized with **Media_Init**. The instruction **Media_Init** is only available in *ADbasic*, but not in *TiCoBasic*.

Media_Erase should only be used in low priority process section:

- anywhere in a low priority process
- in the sections **LowInit** or **Finish** of a high priority process

Erasing the memory card may increase the data transfer rate for read and write.

See also

Media_Init, Media_Write, Media_Read

Valid for

Gold II-Storage-16

Example

```
#Include ADwinGoldII.inc
Dim err, num_blocks As Long
Dim media_datatable[100] As Long

LowInit:
  Rem initialize media card
  num_blocks = Media_Init(media_datatable)
  If (num_blocks < 0) Then Exit
  err = Media_Erase(media_datatable)
```

Media_Read

T11

Media_Read copies blocks of values from the memory card of *ADwin-Gold II* into an array.

Syntax

```
#Include ADwinGoldII.inc
ret_val = Media_Read(media_datatable[], start_block,
    count_blocks128, dest_array[],
    array_start_index)
```

Parameters

media_datatable[]	Array containig data for the operation of the memory card, see **Media_Init**.	ARRAY LONG
start_block	Index (0…m-1) of the first data block on the memory card which is to be read. m is the return value of **Media_Init**.	LONG
count_blocks128	Number (1…m-1) of data blocks to be read. A data blocks contains 128 values of 32 bit. m is the return value of **Media_Init**.	LONG
dest_array[]	Array where data is written to. Both data types Long or Float are allowed.	ARRAY LONG FLOAT
array_start_index	Index (1…n) of the first array element to be transferred.	LONG
ret_val	Status of data transfer: = 0: The data have been read successfully. > 0: An error has occurred. Set Bits in ret_val refer to the error cause, see table.	LONG

Bit no.	Meaning
00	Bit = 1: Memory card does not reply.
01	Bit = 1: Timeout.
02:32	Bits may be set but cannot be used here.

Notes

Before using **Media_Read**, the array media_datatable[]must be initialized with **Media_Init**. The instruction **Media_Init** is only available in *ADbasic*, but not in *TiCoBasic*.

Media_Read should only be used in low priority process section:

- anywhere in a low priority process
- in the sections **LowInit** or **Finish** of a high priority process

If used in a high priority process only a small number of data blocks may be transferred. Otherwise the communication between PC and *ADwin* system would be stopped too long, and would therefore lead to an error message (timeout).

The data transfer speed per data block increases with the number of transferred data blocks.

The array dest_array[] must be dimensioned with at least count_blocks128 × 128 elements.

The data type of `dest_array[]` must equal the data type of the prviously stored values on the memory card. If not, the values will be wrongly interpreted.

Values stored on the memory card always have 32 bit length, even if the data type is float.

See also

Media_Init, Media_Erase, Media_Write

Valid for

Gold II-Storage-16

Example

- / -

Media_Write

T11

Media_Write copies blocks of values from an array to the memory card of *ADwin-Gold II*.

Syntax

```
#Include ADwinGoldII.inc
ret_val = Media_Write(media_datatable[],
    start_block, count_blocks128, source_array[],
    array_start_index)
```

Parameters

media_datatable[]	Array containig data for the operation of the memory card, see **Media_Init**.	ARRAY LONG
start_block	Index (0…m-1) of the first data block on the memory card which is to be written. m is the return value of **Media_Init**.	LONG
count_blocks128	Number (1…m-1) of data blocks to be written. A data blocks contains 128 values of 32 bit. m is the return value of **Media_Init**.	LONG
source_array[]	Array from which data is read. Both data types Long or Float are allowed.	ARRAY LONG FLOAT
array_start_index	Index (1…n) of the first array element to be transferred.	LONG
ret_val	Status of data transfer: = 0: The data have been written successfully. > 0: An error has occurred. Set Bits in ret_val refer to the error cause, see table.	LONG

Bit no.	Meaning
00	Bit = 1: Memory card does not reply.
01	Bit = 1: Timeout.
02:32	Bits may be set but cannot be used here.

Notes

Before using **Media_Write**, the array media_datatable[]must be initialized with **Media_Init**. The instruction **Media_Init** is only available in *ADbasic*, but not in *TiCoBasic*.

Media_Write should only be used in low priority process section:

- anywhere in a low priority process
- in the sections **LowInit** or **Finish** of a high priority process

If used in a high priority process only a small number of data blocks may be transferred. Otherwise the communication between PC and *ADwin* system would be stopped too long, and would therefore lead to an error message (timeout).

The data transfer speed per data block increases with the number of transferred data blocks.

The array source_array[] must be dimensioned with at least count_blocks128 × 128 elements.

Each data block on the memory card holds 128 values of 32 bit length, not regarding the data type (Long or Float). If the array source_array[] contains values of data type Float–with a length of 40 bit–the float values are transformed into 32 bit format during data transfer.

See also

Media_Init, Media_Erase, Media_Read

Valid for

Gold II-Storage-16

Example

```
Rem store a sine table of 3600 values
#Include ADwinGoldII.inc

#Define blocks Par_52      'number of medium data blocks
#Define val_per_block 128  'values per medium data block
#Define media_info Data_197'array with media information
#Define sine Data_1        'array for sine values
#Define nds 3600           'number of sine values
#Define pi2 6.2831853      '2*pi

Dim media_info[100] As Long 'dimension arrays
Dim LUT[nds] As Long
Dim err, idx, blk_num As Long

LowInit:
  Rem initialize media card
  blocks = Media_Init(media_info)
  If (blocks < 0) Then Exit
  Par_53 = Media_Erase(media_info)
  If (Par_53 > 0) Then Exit

  For idx = 1 To nds        'calculate sine values
    LUT[idx] = 32767.5 * Sin((idx-1) * pi2 / nds)
  Next idx

  Rem write values
  blk_num = nds/val_per_block + 1
  If (blk_num > blocks) Then Exit
  err = Media_Write(media_info,1,blk_num, LUT,1)
  If (err > 0) Then Exit
```

15.14Storage media (TiCoBasic)

This section describes instructions to access the memory card of *ADwin-Gold II* with the *TiCo* processor:

Media_Read

TiCo

Media_Read copies blocks of values from the memory card of *ADwin-Gold II* into an array.

Syntax

```
#Include GoldIITiCo.inc
ret_val = Media_Read(start_block, count_blocks128,
    dest_array[], array_start_index)
```

Parameters

start_block	Index (1…m) of the first data block on the memory card which is to be read.	LONG
count_blocks128	Number (1…m) of data blocks to be read. A data blocks contains 128 values of 32 bit.	LONG
dest_array[]	Array where data is written to.	ARRAY LONG
array_start_index	Index (1…n) of the first array element to be transferred.	LONG
ret_val	Status of data transfer: = 0: The memory card has been completely erased. > 0: An error has occurred. Set Bits in ret_val refer to the error cause, see table.	LONG

Bit no.	Meaning
00	Bit = 1: Memory card does not reply.
01	Bit = 1: Timeout.
02:32	Bits may be set but cannot be used here.

Notes

Before using **Media_Read**, the array media_datatable[]must be initialized with **Media_Init**. The instruction **Media_Init** is only available in *ADbasic*, but not in *TiCoBasic*.

The execution time of **Media_Read** is quite long; therefore we recommend to use it in a process with low priority.

The data transfer speed per data block increases with the number of transferred data blocks.

The array dest_array[] must be dimensioned with at least count_blocks128 × 128 elements.

See also

Media_Write

Valid for

Gold II-Storage-16

Example

- / -

Media_Write

TiCo

Media_Write copies blocks of values from an array to the memory card of *ADwin-Gold II*.

Syntax

```
#Include GoldIITiCo.inc
ret_val = Media_Write(start_block, count_blocks128,
    source_array[], array_start_index)
```

Parameters

start_block	Index (1…m) of the first data block on the memory card which is to be written.	LONG
count_blocks128	Number (1…m) of data blocks to be written. A data blocks contains 128 values of 32 bit.	LONG
source_array[]	Array from which data is read.	ARRAY LONG
array_start_index	Index (1…n) of the first array element to be transferred.	LONG
ret_val	Status of data transfer: = 0: The memory card has been completely erased. > 0: An error has occurred. Set Bits in ret_val refer to the error cause, see table.	LONG

Bit no.	Meaning
00	Bit = 1: Memory card does not reply.
01	Bit = 1: Timeout.
02:32	Bits may be set but cannot be used here.

Notes

Before using **Media_Write**, the array media_datatable[]must be initialized with **Media_Init**. The instruction **Media_Init** is only available in *ADbasic*, but not in *TiCoBasic*.

The execution time of **Media_Write** is quite long; therefore we recommend to use it in a process with low priority.

The data transfer speed per data block increases with the number of transferred data blocks.

The array source_array[] must be dimensioned with at least count_blocks128 × 128 elements.

See also

Media_Read

Valid for

Gold II-Storage-16

Example

Part 1: ADbasic

```
Rem -----------------------------
Rem Part 1: ADbasic program
Rem initialize media card
#Include ADwinGoldII.inc
#Define blocks PAR_52      'number of medium data blocks
#Define media_info DATA_197'array with media information

Dim media_info[100] As Long

LowInit:
 blocks = Media_Init(media_info)
 If (blocks < 0) Then Exit
 PAR_53 = Media_Erase(media_info)
 If (PAR_53 > 0) Then Exit
 Rem Set flag on TiCo processor: card is initialized
 Set_Par(1, 12, blocks)
```

Part 2: TiCoBasic

```
Rem -----------------------------
Rem Part 2: TiCoBasic program
Rem store measurement values
#Include GoldIITiCo.inc

#Define val_per_blk 128    'values per medium data block
#Define blk_group 40       'number of blocks to write
#Define blk_total PAR_12   'number of medium data blocks
Dim values[5120] As Long   '5120=blk_group*val_per_blk
Dim idx, blk_no As Long

Init:
 Do                        'wait for flag: media card initialized
 Until (blk_total > 0)
 idx = 0                   'index of measurement values
 blk_no = 1                'first block to write

Event:
 Inc idx                   'get next value
 values[idx] = ADC(1)
 If (idx = val_per_blk*blk_group) Then
  Rem write 40 x 128 values
  PAR_1 = Media_Write(blk_no, blk_group, values, 1)
  If (PAR_1 > 0) Then End
  Rem adjust counters
  idx = 0
  blk_no = blk_no + blk_group
  If (blk_no+blk_group > blk_total) Then End
 EndIf
```

Annex

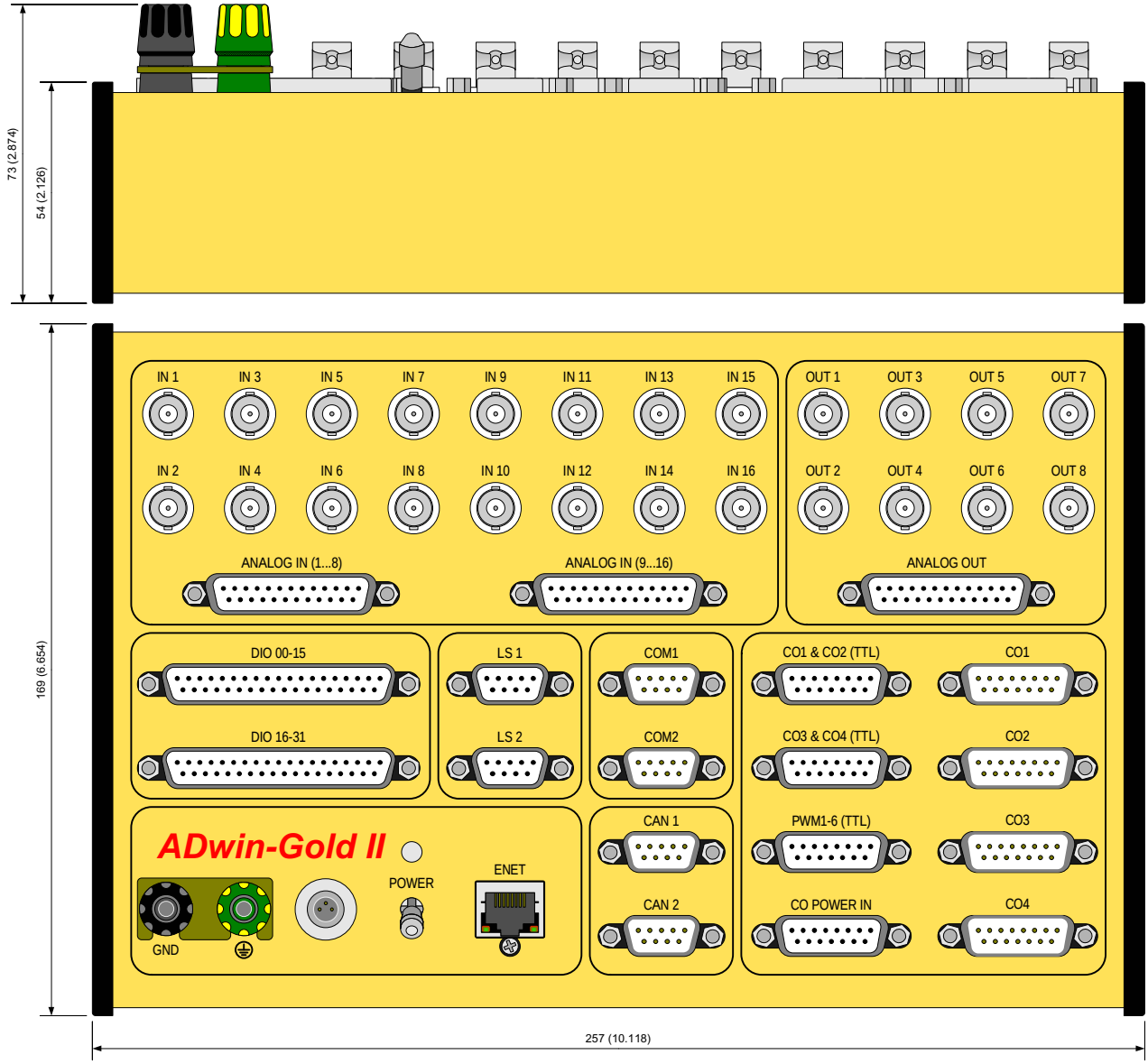
A.1 Technical Data

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

General Data/Limit Values						
	Symbol	Conditions	min.	typ.	max.	Unit
Supply Voltage/Supply Current						
Voltage	U_b		9	12	28	V
Idle current	I_{idle}	U_b=10V		1.3		A
		U_b=12V		0.9		
		U_b=24V		0.5		
Power-up current	$I_{power-on}$	U_b=12V	3			
Valid operation ranges						
Temperature	$T_{chassis}$		+5		+60	°C
Relative humidity	F_{rel}	no condensation	0		80	%
Storage						
Temperature	T		-20		+70	°C
Humidity	R_H	no condensation or aggressive atmosphere				
Connectors						
DSUB connectors	Metric ISO threads; UNC threads available as ordering option					
BNC connectors	Standard 50Ω version					
Dimensions						
Width x height x depth	W x H x D	Gold II	257 × 73 × 169			mm
Net weight						
Weight	m_{Net}	Gold II	2050 (4.5 lb)			g
		Clips[a]	32			

[a] Accessories for DIN rail mounting: *Gold-Mount*

(all dimensions are in millimeters [mm], values in parentheses are in inches, 25.4mm = 1")



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_{IH}	V_I = 5V			2.4	mA
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			1.1	mA
	I_{IL}	V_I = 0.4V			1.1	

Analog Inputs/Outputs						
Parameters	Symbol	Conditions	min.	typ.	max.	Unit
Inputs						
Number	2 ¥ 8 via multiplexer, differential					
Input resistance	Ri		323.4	330	336.6	kW
Overvoltage	U_{in} max.	ON & OFF			±35	V
Multiplexer settling time	t_{MUX}	1 LSB 16-bit		2.0		µs
ADC 16-bit						
Conversion time	t_{conv}				2	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[a]			±2		ppm/K
	Error	Adjustable ±1%				
Gain	Drift[a]			±5		ppm/K
	Error	Adjustable ±1%				
Outputs: DAC 16-bit						
Number	2 (with DA add-on: 4 or 8)					
Output voltage	U_{out}		-10		+9.999695	V
Settling time	t_{settle}	2V jump		2		µs
		FSR[b] (20V)		4		
Permissible current					10	mA
Integral non-linearity	INL				±2	LSB
Differential non-linearity	DNL				±1	
Offset	Drift[a]			±1		ppm/K
	Error	Adjustable				
Gain	Drift[a]			±3		ppm/K
	Error	Adjustable				

[a] refers to the total voltage range (FSR)

[b] Full Scale Range

Processor						
Parameters	Symbol	Conditions	min.	typ.	max.	Unit
Type	ADSP TS610S (TIGER-SHARC™)					
Manufacturer	Analog Devices					
Clock frequency	f_{CLK}			300		MHz
Register width				32		Bit
Internal memory	SRAM	for programs		256		kByte
		for data		256		
External memory	SDRAM			256		MByte

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	fref			100		MHz
Accuracy and Drift					±20	ppm
Counter inputs differential[a]						
Differential input threshold voltage	V_{TH}	$-10V \le V_{CM} \le 13.2V$	-200		+200	mV
Input hysteresis	ΔV_{TH}	$-10V \le V_{CM} \le 13.2V$		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		±5		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			0.5	mA
	I_L	V_I = 0.4V			0.04	

[a] see also data sheet MAX3098 from MAXIM

[b] see also data sheet 74LS19 from Texas Instruments

A.2 Hardware Addresses

For *ADbasic*, there are no hardware addresses documented, since the use would give no advantage compared to the use of the provided *ADbasic* instructions.

The following hardware addresses are valid for the *TiCo* processor. The addresses of the first column enable to trigger an externally controlled process. The addresses of the second column are required in the externally controlled process. Find examples under `C:\ADwin\TiCoBasic\samples_ADwin_GoldII`.

Address *TiCoBasic* [HEX] trigger process	use inside process	Function	Bit No. 31:24	23:17	16	15:9	8	7	6	5	4	3	2	1	0	Comment
Digital Input and Outputs																
000000D0	–	read digital inputs DIO00:DIO31	x	x	x	x	x	x	x	x	x	x	x	x	x	x: read digital value
000000E4	000000E0	Edge control unit: rising edge[a]	x	x	x	x	x	x	x	x	x	x	x	x	x	x=1: edge has occurred at this input x=0: no edge
000000E6	000000E2	Edge control unit: falling edge[a]	x	x	x	x	x	x	x	x	x	x	x	x	x	
000000E8	–	DIO: edge control FIFO	-	-	y	-	x									x: number (0…511) of values in edge control FIFO y=1: FIFO full
000000FE	000000FC	Event register[a]	-	-	-	-	-	-	-	-	-	c	b	a	x	Bit=1: Event signal has occurred. x: external Event input b: Event from Anybus c: Event from CAN2 d: Event from CAN1

[a] After being read, the first address keeps the register values unchanged, while the second address resets the register to zero.

A.3 Hardware revisions

The revision of a Gold II 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
A1	Nov. 2007	First release.
A2	Jun. 2008	*TiCo* processor released.

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 II* complies with the requirements of the RoHS directive in all delivered variants.

A.5 Baud rates for CAN bus

ADwin-Gold II-CAN provides interfaces for the CAN bus. The following baud rates 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

W