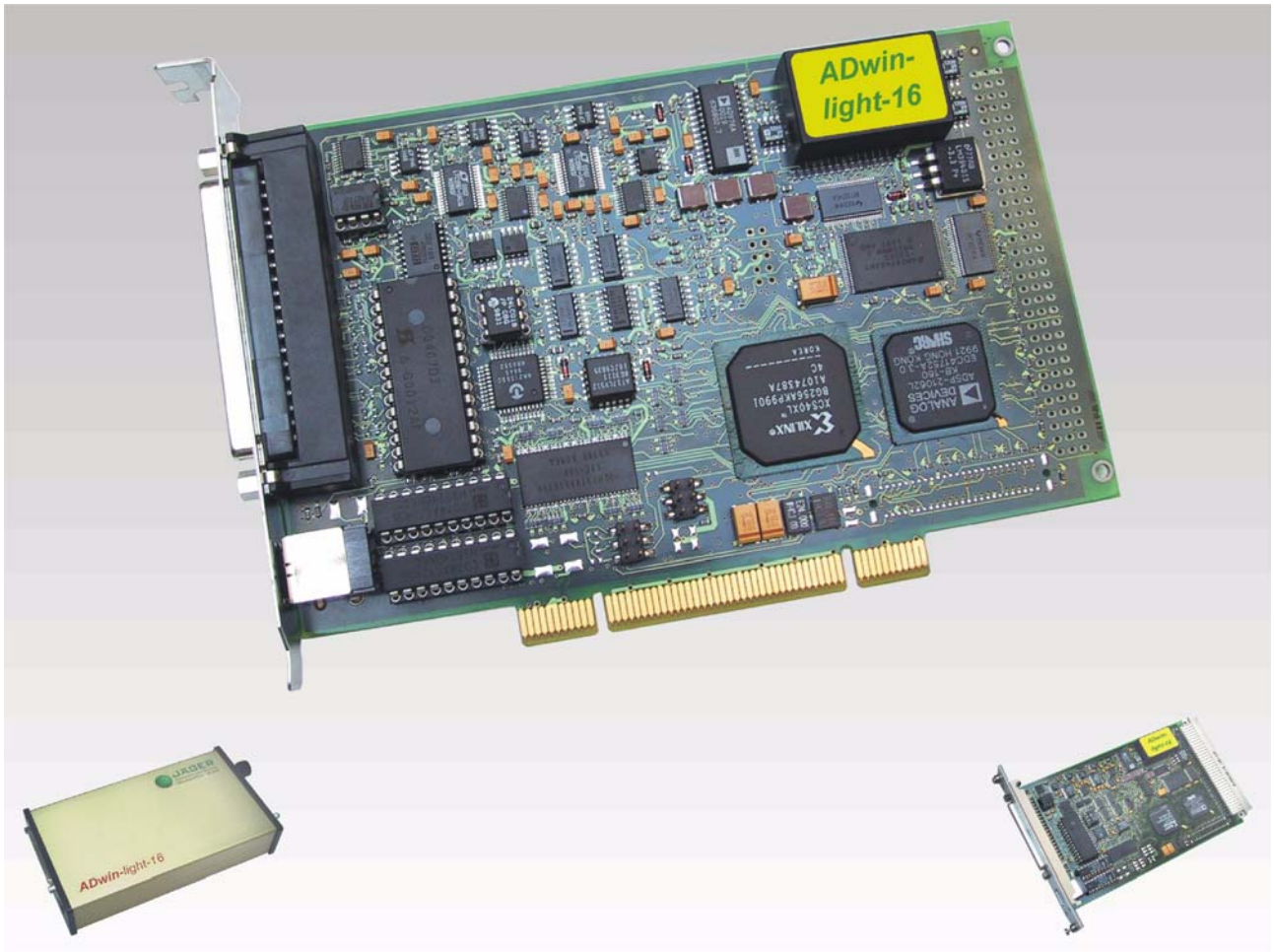
ADwin-light-16

Manual



ADwin-light-16 , manual version 3.5, November 2013

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.

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

<C:\ADwin\ …>

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

Program text

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

Var_1

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

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

1 Information about this manual

This manual contains comprehensive information about the operation of the *ADwin-light-16* system. Additional information is available in

– the manual "*ADwin* Driver Installation", which describes all interface installations for the *ADwin* systems.
Commence your installation with the help of this manual.

– the description of the configuration program *ADconfig*. With it, you initialize the communication of the corresponding interface with *ADwin-light-16*.

– the manual *ADbasic*, which contains all instructions for the compiler *ADbasic* and explains the principle of *ADwin* systems in particular.

The online help of *ADbasic* contains the same information.

– the description of the driver installation and command instructions for all well known development environments.

– the manual "*ADwin* HSM-24V", a module on the LS bus.

Please note:

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

Qualified personnel

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

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

Availability of the documents

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

Legal information

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

Subject to change.

Hotline address: see inner side of cover page.

2 System description

2.1 *ADwin* system concept

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

– very fast digital closed-loop control systems

– very fast open-loop control systems

– data acquisition with very fast online analysis of the measurement data

– monitoring of complex trigger conditions and many more

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

System features

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

Processor

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

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

Real-time operating system

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

Timing

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

ADbasic

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

Communication between *ADwin* system and PC

Interfaces

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

Instruction processing

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

Software interfaces

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

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

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

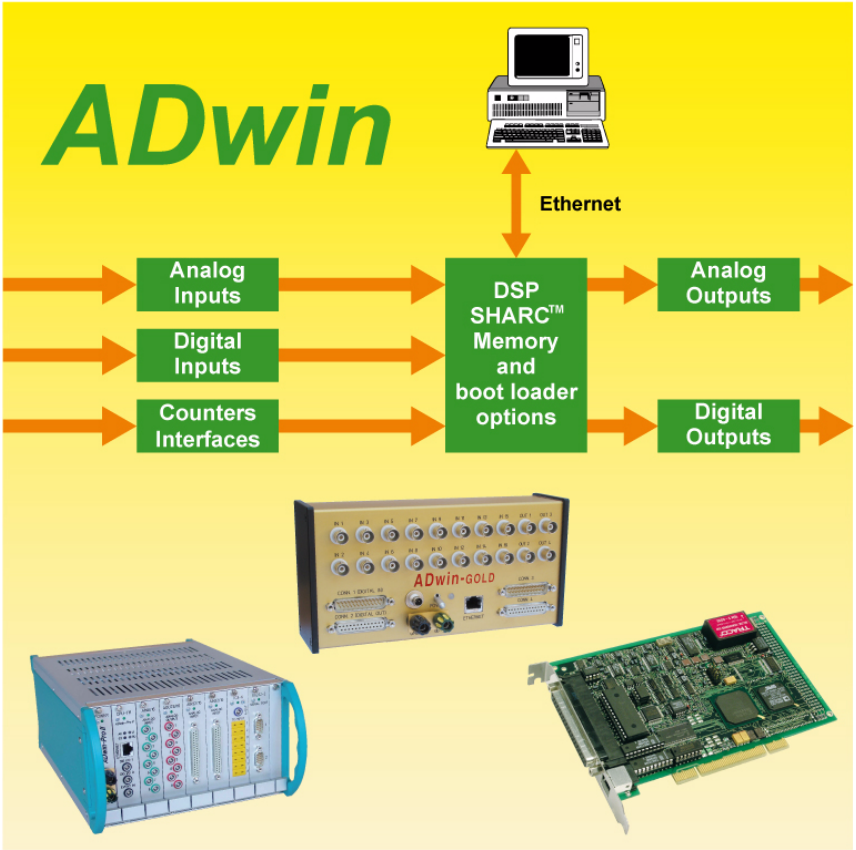


Fig. 1 – Concept of the ***ADwin*** systems

2.2 ADwin-light-16

Processor and memory

The *ADwin-light-16* system is equipped with the **32 bit signal processor** ADSP 21062 (SHARC) from Analog Devices with floating point and integer processing. It is responsible for the complete measurement data acquisition, online processing, and signal output and can instantaneously process - in combination with the A/D-converter - each measured value with sample rates of up to 100 kHz; from Rev. B an optional sample rate of 500 kHz is available.

The **on-chip memory with 256 KiB** has a very short access time of 25 ns and is large enough to hold the complete *ADwin* operating system, the *ADbasic* process and all variables.

In order to get maximum access speed, 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** (SDRAM) of **8 MiB** (Rev. B has 16 MiB).

Analog inputs

In a 37-pin D-SUB socket there are **8 analog inputs** available, which are connected to a multiplexer, whose output signal is converted with a 16 bit analog-to-digital converter (ADC, see figure below). Since revision B a sequential control for automatic conversion of several channels is available.

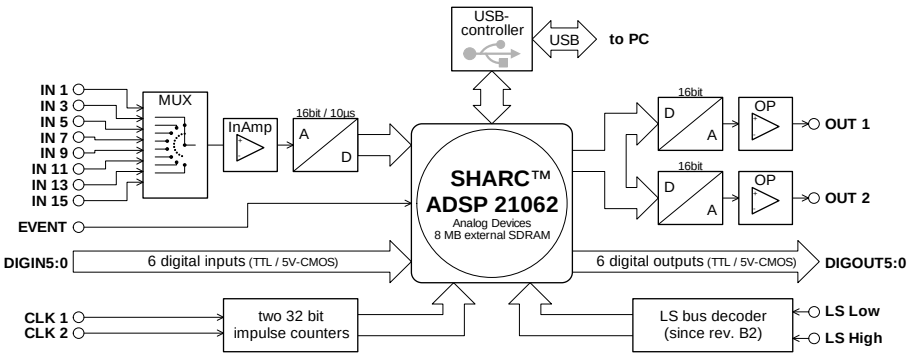


Fig. 2 – Functional diagram (with USB interface)

Analog outputs

ADwin-light-16 is equipped with **2 analog outputs** with 16 bit resolution and an output voltage range of -10 V ... +10 V. The output voltages of all DACs are synchronized and calibrated per software. In order to smooth the output signal, it passes through a low pass filter with a cut-off frequency of $f_c = 700$ kHz.

Digital inputs and outputs

6 digital inputs and 6 digital outputs are available on the 37-pin D-SUB socket. The inputs and outputs are TTL-compatible. Furthermore, there are inputs for 2 counters with 32 bit each.

Trigger input

ADwin-light-16 is equipped with a trigger input (EVENT, see also chapter 5.2 "Digital Inputs and Outputs"). Thus, processes can be triggered by a signal (trigger) and completely processed at once (see *ADbasic* manual, chapter "Processes in the ADwin Operating System").

A serial interface (LS bus from Rev. B2, see page 16) enables the connection of upt o 15 additiional modules.

Scope of delivery

The standard scope of delivery for *ADwin-light-16*:

- *ADwin-light-16* system
- USB or Ethernet connecting cable, length 1.8 m
- *ADwin*-CD-ROM
- Manual "Driver Installation"
- this hardware manual

Additional items supplied with the type with external enclosure (*L16-EXT*) are:

– Power adapter: a PC slot plate with power supply socket and PC-internal three-pole connecting cable

– power supply cable for connection between the slot plate and *L16-EXT*.

Variants

ADwin-light-16 is available as basic version with USB connection in several variants.

PC plug-in board (*L16-PCI*)

19" plug-in board (*L16-EURO*)

external enclosure (*L16-EXT*)

Fig. 3 – Variants

Types

The variants *L16-EURO* and *L16-EXT* can optionally be delivered with USB or 10/100 MBit Ethernet interface. The available types of the *ADwin-light-16* basic version are described in the following table.

Type	Interface	
	USB	Ethernet
PC plug-in board	*L16-PCI*	–
19" plug-in board (Euro)	*L16-EURO*	*L16-EURO-ENET*
External enclosure	*L16-EXT*	*L16-EXT-ENET*

Fig. 4 – Types of the *ADwin-light-16* basic version

Please take into account that the power supply for the different variants varies:

– +5 Volt for *L16-PCI* and *L16-EURO*

– +10 ... +18 Volt for *L16-EXT*, since Rev. B +10 ... +36 Volt

2.2.1 Ordering options (later upgrade not possible)

ADwin-light-16 can be equipped with the following options:

– *L16-CO1*: counter option (description see page 23) with a 32-bit up/down counter with four edge evaluation for incremental encoders.

– *L16-DIO1*: add-on module (description see page 25) with
  - 32 digital inputs/outputs (programmable in groups of 8)
  - one SSI decoder (since Rev. B)
  - CAN interface (high speed; low speed as an alternative)
  - two 32 bit up/down counters for impulse, period duration and duty cycle measurements as well as a four edge evaluation for the connection of incremental encoders.

– *L16-DIO2*: add-on module (description see page 41) with
  - 32 digital input/outputs (programmable in groups of 8).
  - one SSI decoder
  - two 32 Bit up/down counters for impulse, period duration and duty cycle measurements as well as a four edge evaluation for incremental encoders.

– *L16-DIO3*: add-on module description see page 41) contains 32 digital inputs/outputs (programmable in groups of 8).

– *L16-PWM1*: Software add-on (description see page 53) cointains 1 PWM output and 1 SPI interface (Master).

– *L16-Boot*: Flash-EPROM bootloader for stand alone operation without a PC (description see page 57). May only be ordered in combination with an Ethernet interface.

– *L16-Mount*: Kit for installation of an *L16-EXT* system on a DIN top-hat rail in an electrical control cabinet with isolated mountings.

Please take into account, that the counters of the add-on boards are not additionally available, but that they replace the counters of the basic version. Therefore you cannot use the counters of different add-ons at the same time.

2.2.2 Accessories

The following equipment for *ADwin-light-16* is available as accessory; description see page 58:

ADbasic

– *ADbasic*, real-time development tool for all *ADwin* systems

– cable connectors for an external power supply (for *L16-EXT* only)

– external power supply *ADwin-light-16*-pow (necessary for notebook operation)

3 Operating Environment

The board of the *ADwin-light-16* may only be operated in a closed casing (already given with *L16-EXT* variant).

According to type and accessories (see chapter 2.2.1f, delivery options / accessories) the system can be operated in 19" enclosures, control cabinets or as a mobile system (e.g. in vehicels).

The *ADwin-light-16* **system must be earth-protected**, in order to

– build a ground reference point for the electronic

– conduct interferencing energy to earth.

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

Galvanic connection

The types with USB interface have a galvanic connection to the PC via USB and possibly also via power supply.

The types with Ethernet interface have their data lines galvanically isolated, but the ground potentials are connected, because the shielding of the Ethernet connector (RJ-45) is connected with GND.

Excluding transient currents

Transient currents, which are conducted via the enclosure or the shielding, have influence on the measured signal.
If you want to prevent transient currents, please make sure that the shielding is fully operative. Take measures for bleeding off interference, such as earthing the shielding close to the entry into the control cabinet. The more frequently you earth the shielding on its way to the machine the better the shielding will operate.

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

Protective extra low voltage

The *ADwin-light-16* system is internally operated with a voltage of +5 V and ±15 V against GND and thus is not life-threatening. For operation with an external power supply, the instructions of the manufacturer apply.

Ambient temperature

ADwin-light-16 is designed for operation in dry rooms. The installation environment (PC or 19" rack) may have an ambient temperature within the range +5 ... +50 °C, and a relative humidity of 0 ... 80 % (none condensing, see also Annex).

Chassis temperature

The temperature of the casing (surface temperature) of the type *L16-EXT* must not exceed +55 °C, even under extreme operating conditions - e.g. in an electrical control cabinet or if the system is exposed to the sun for longer periods of time. Otherwise, you risk damage to the device or the output of undefined data (values) which can cause damages to your measurement equipment under unfavorable circumstances.

4 Start-up of the Hardware

Do not connect any signal cables to *ADwin-light-16* **on start-up** before taking the **following steps**:

1. Software installation / hardware installation in the PC or 19“ rack.

   Follow the instructions in the manual: "*ADwin* Driver Installation".
2. Set the operating environment as described in chapter 3.
3. Read chapter 5 "Inputs and Outputs" in this manual.
4. Only now connect the signal lines to the inputs and outputs.

Notes

Avoid direct contact with uninsulated parts in order to protect them against electrostatic discharges.

Reliable power supply

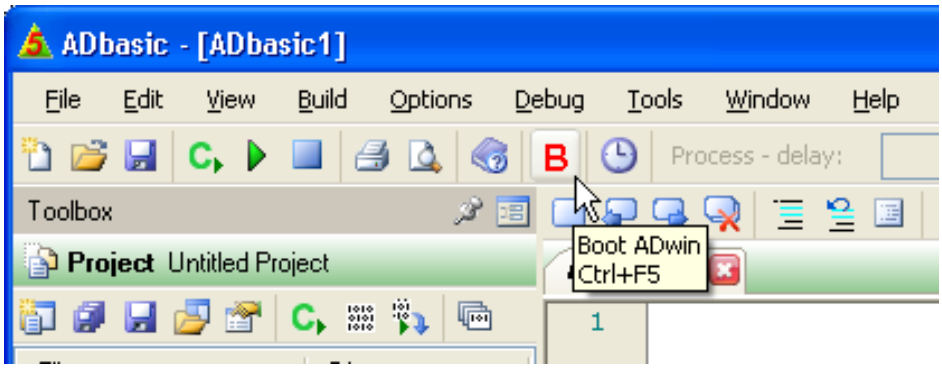
Please pay attention that a reliable power source is used.
For standard version, this concerns the PC, and otherwise the external power supply, or if operated in a vehicle, the battery voltage.

If using current-limiting power supplies, please pay attention the current demand at power-up which can be a multiple of the operating current. Detailed information is contained in the Technical Data (Annex).

In case of power failure all unsaved data are lost. Undefined data could cause damage to your equipment under unfavorable circumstances.

Check Data Communication

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



The display in the status line: "`ADwin is booted`" shows that the operating system has been loaded appropriately and that *ADbasic* can establish a connection to the *ADwin* system.

***ADbasic* programs**

Programming the *ADwin* systems is described in more detail in the *ADbasic* manual.
Start with the programming examples in the *ADbasic* Tutorial.

5 Inputs and Outputs

Connections

The *ADwin-light-16* system has the following connectors (pin assignment see next page):

– connector for USB or Ethernet

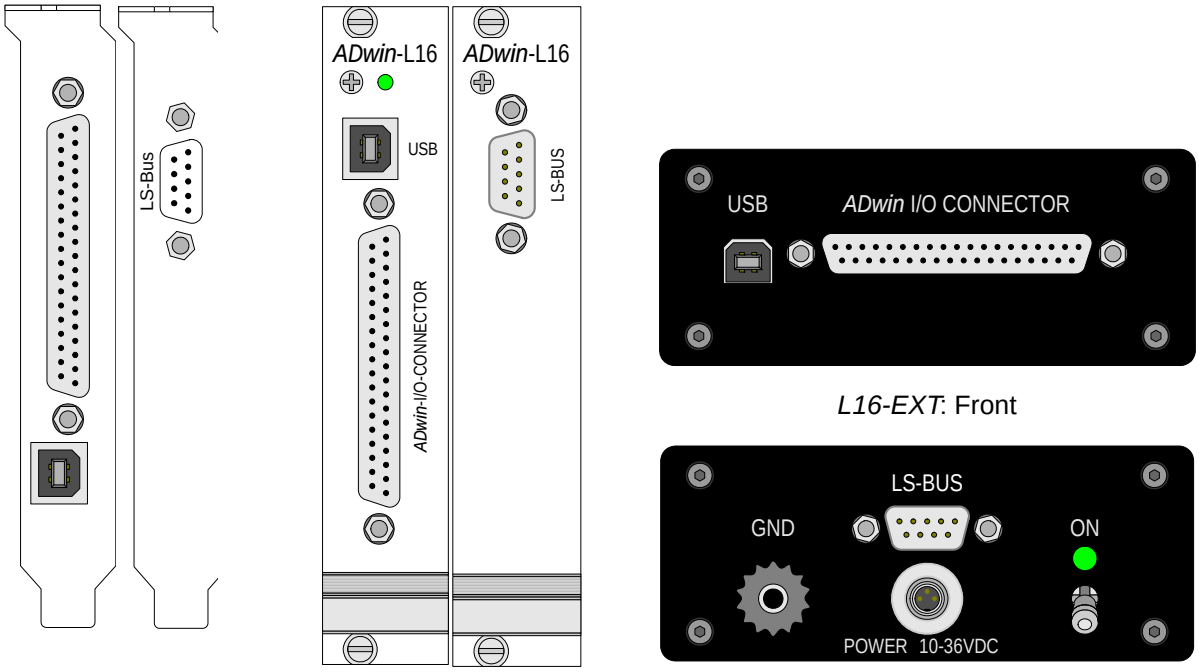
– 37-pin D-SUB socket `ADwin I/O CONNECTOR` for
  - 8 analog inputs
  - 2 analog outputs
  - 6 each, digital inputs and outputs
  - 1 digital trigger input
  - 2 impulse/event counters with 32 bit
  - Output for power supply +5V; *L16-PCI* also ±12V

– 9-pin D-SUB socket `LS-BUS` for the LS bus interface (since Rev. B2).

The variant *L16-EXT* has an additional GND socket (see earth protection, page 7), a power input socket and a manual on/off switch.

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 equipment to which you intend to connect the *ADwin-light-16* system with.

Open inputs can cause errors – above all in an environment which is not free of any interferences. For your own safety, connect unused inputs as close as possible to the D-SUB socket on a defined level (e.g. GND). Separate these inputs from open circuit lines.
Exception to this is the event input, which already has an internal pull-up resistor (10 kΩ).



L16-PCI *L16-EURO* *L16-EXT*: Front *L16-EXT*: Back

Fig. 5 – Connectors *ADwin-light-16*

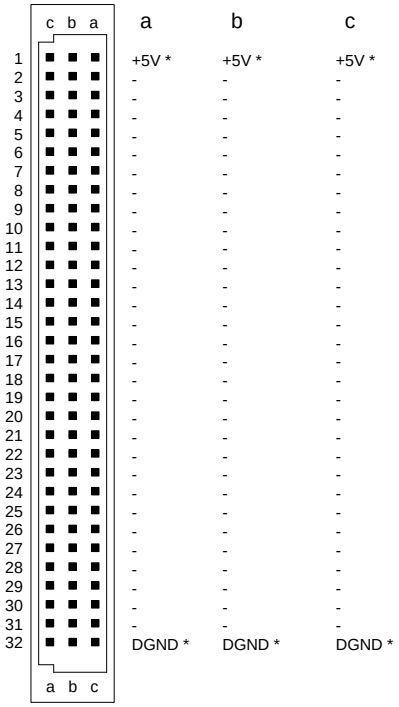


Fig. 6 – *L16-EURO* VG96 connector for power supply (female)

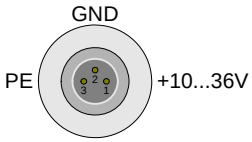


Fig. 7 – *L16-EXT* power connector (male)

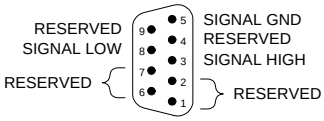
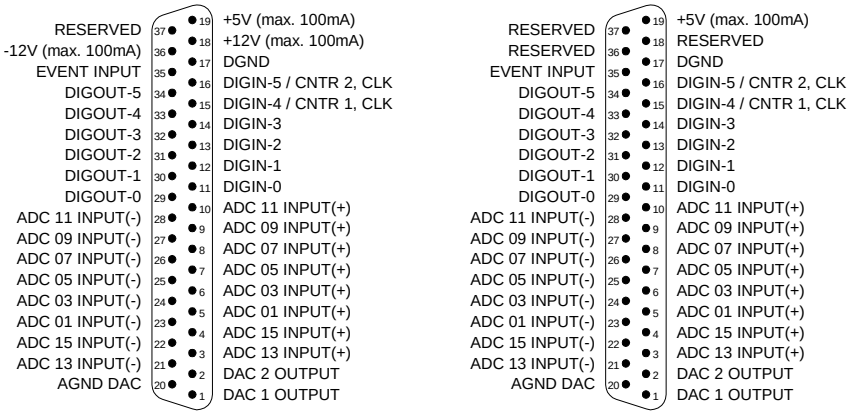


Fig. 8 – Pin assignment LS-BUS (female)



L16-PCI *L16-EURO* and *L16-EXT*

Fig. 9 – Pin assignment inputs/outputs (female)

Standard instructions

For fast and easy programming there are standard instructions available in the compiler *ADbasic*, which enable a user to easily measure or output data (see also *ADbasic* manual). Use other instructions (such as direct register access) only if extremely time-critical or special tasks require to do so. (see also chapter 13).

More detailed information about the analog as well as the digital inputs and outputs can be found in the following chapters.

5.1 Analog Inputs and Outputs

Earth protection

The variant *L16-EXT* has to be earth-protected, in order to perform measurements without interference. For this, connect the GND socket via a low impedance solid-type cable to the central earth connection point of your installation. When using the variants PCI or EURO, the earth protection is made via PC or the 19" rack.

For *L16-EXT*, the enclosure is connected to the protective earth conductor of the PC via the GND-line of the power supply cable as well as via the GND-line of the USB cable.

5.1.1 Analog Inputs

Multiplexer

The system has 8 analog measurement inputs, which are connected through a multiplexer to the 16-bit analog-to-digital converter (ADC). The multiplexer settling time is 6.5µs with a full scale range of 20V.

The inputs are all odd-numbered (ADC 01, ADC 03, ... ADC 15), which has to be considered during programming.

Differential

The analog inputs are differential. For each of the measurement channels there is a positive and a negative input, between which the voltage difference is measured (pay attention to the potential of the input lines).

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

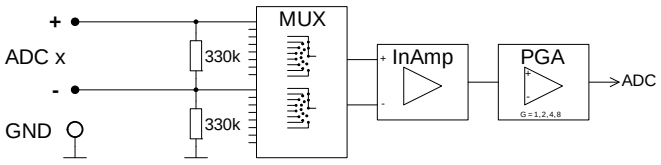


Fig. 10 – Input circuitry of an analog input

16-bit measurement

The signal at the multiplexer output is converted by a 16-bit analog-to-digital converter (ADC); see Fig. 2 – Functional diagram (with USB interface). The conversion time is 10µs (since Rev. B selectable per software: 2µs) at a resolution of 305 µV.

Complete measurement

The instruction **ADC**( ) executes a complete measurement with an ADC on one analog input. Thus, this instruction considers for instance the settling time of the multiplexer and assures perfect measurements (see also *ADbasic* manual).

Rev. B with sequential control

From Rev. B the signals of several selected analog inputs can be converted (sequentially) with a single instruction. The inputs are selected with **Seq_Init**, conversion is started with **Start_Conv** and results are read with **Seq_Read**.

5.1.2 Analog Outputs

DAC instruction

The standard instruction `DAC(number,value)` checks each of the values exceeding or falling below of the 16-bit value range. If the value is in the 16-bit value range, the indicated value is output on the output `number`. If it is not, the maximum or minimum value is output (see also *ADbasic* manual).

5.1.3 Calculation Basis

Voltage range

The voltage range of the *ADwin-light-16* system at the analog inputs and outputs is –10 V to +10 V (bipolar 10 V).

Allocation of digits and 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 65,536 digits, exactly 10 Volt, is just outside the measurement range, so that there is a maximum voltage value of 9.999695 Volt for the 16-bit conversion.

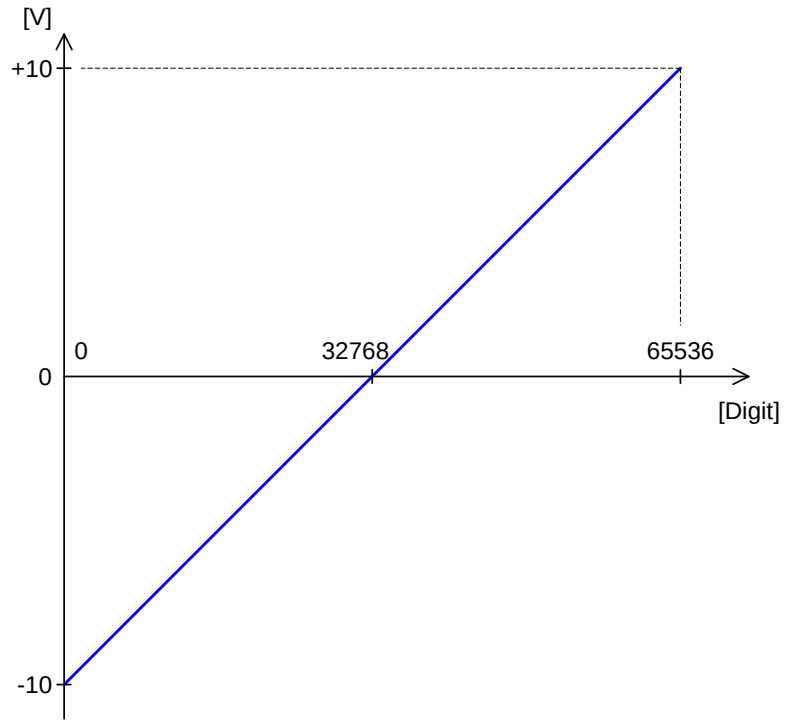


Fig. 11 – Zero offset in the standard setting of bipolar 10 Volt

Bipolar range leads to a zero offset which is shown below.

For the voltage range of –10 V ... +10 V applies U_{OFF} = –10 V

Least Significant Bit

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 U_{LSB} is equivalent to the formula: 20 V / 2^{16} = 305.175 µV.

The measured 16-bit value of the ADC is returned to the lower word of the binary cell. Here you must also find the DAC value to be output.

Bit	31...16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
32-bit memory	upper word	16-bit value of the ADC/DAC in the lower word															

Conversion Digits ⇆ Voltage

For a DAC:

DAC

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

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

For an ADC:

ADC

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

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

Tolerances

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

– The integral non-linearity (INL) defines the maximum deviation from the ideal straight line of the conversion characteristics curve, covering the whole input voltage range.

INL

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

DNL

5.2 Digital Inputs and Outputs

Digital inputs/outputs

6 digital inputs (DIGIN 00 … DIGIN 05) and 6 digital outputs (DIGOUT 00 … DIGOUT 05) are available on the 37-pin D-SUB socket. The pin assignment be found in figure 9 on page 10.

The inputs DIGIN 04 and DIGIN 05 are used as counter inputs at the same time and may be read as digital inputs while being used as counters. This is not true when using a DIO1-add-on (see page 25) or DIO2-add-on (see page 41).

The digital inputs and outputs are TTL-compatible and not protected against overvoltage.
Do not use connections marked as "RESERVED". They are reserved for upcoming changes or expansions and can cause damages to your system if you do not pay attention to this fact.

Trigger input (EVENT)

The *ADwin-light-16* system is equipped with an external trigger input (EVENT). An external signal (trigger) with rising edge may be used to start processes, which are processed immediately and completely (see *ADbasic* manual, chapter "Structure of an *ADbasic*-Program").

Programming

The functions of the digital channels are easily programmed with *ADbasic* instructions:

Function	Instruction
Read single digital input.	DIGIN
Read all digital inputs.	DIGIN_WORD
Set all digital outputs.	DIGOUT_WORD
Set one digital output to level High.	SET_DIGOUT
Set one digital output to level Low.	CLEAR_DIGOUT

The instructions are included in the file <ADWL16.INC> and are described in the *ADbasic* manual and in the online help.

5.3 Impulse/Event Counter

The *ADwin-light-16* system is equipped with 2 impulse/event counters, each with 32 bits, which may be configured or read out both together or individually.

With the options *L16-CO1*, *L16-DIO1 or L16-DIO2* the counters described here are replaced by other counters. You will find the corresponding description in chapter 7 "CO1 Counter Add-On", chapter 8 "DIO1 Add-On" or chapter 9 "DIO2 / DIO3 Add-On".

5.3.1 Hardware

The figure shows the design of a single counter.

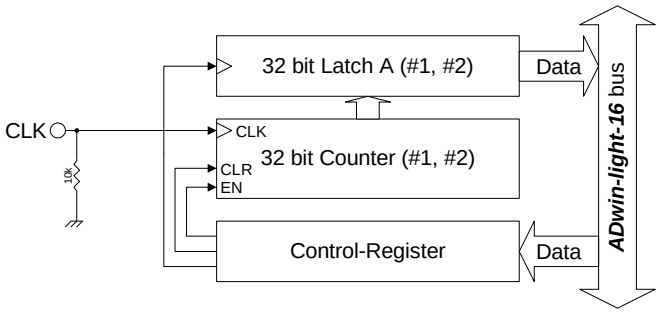


Fig. 12 – Block diagram of the impulse/event counter

The counters are externally clocked that means they increase their counter values by incrementing at each positive edge at the clock input (CLK). Both counters have a latch A, into which the counter value can be latched (under program control) for read out.
The counters are controlled with special *ADbasic* instructions via the control register. The instructions are described in Fig. 13 – Counter instructions - short reference (below) and in the *ADbasic* manual (or online help).

Setting inputs

The clock inputs are on pins 15 and 16 (see Fig. 9 – Pin assignment inputs/outputs (female), page 10); for the correct function TTL compatible signals are required. More details and limit values can be found in the Technical Data in the Annex. Both inputs can optionally be used as digital signal input (see also chapter 5.2).

The inputs of the impulse/event counters have pull-down resistors. Nevertheless open inputs can cause errors in an environment which is not free of interference. Therefore set unused inputs to a defined level (e.g. GND).

5.3.2 Software

Include file

The counters are easily programmed using *ADbasic* instructions. The instructions are part of an include file which must be included at the beginning of a program: `#INCLUDE ADWL16.INC`

The instructions for both counters are shortly illustrated in the following table and more detailed in the *ADbasic* manual or online help. You can configure each counter individually or both counters together.

Counter nº Bit	2 1	1 0	Comment
`Cnt_Clear( )`	0	0	no effect
	1	1	clear counter*
* these functions are reset after being executed. All other functions are reset by the opposing function..			

Fig. 13 – Counter instructions - short reference

Counter nº Bit	2 1	1 0	Comment
Cnt_Enable( )	0	0	disable counter
	1	1	enable counter (pay attention to running counters)
Cnt_Latch( )	0	0	no effect
	1	1	copy counter value into latch A *
Cnt_ReadLatch(#)			read latch A (# = counter-nº 1, 2)
Cnt_Read(#)			copy counter value into latch A and read it (# = counter nº1, 2)
* these functions are reset after being executed. All other functions are reset by the opposing function..			

Fig. 13 – Counter instructions - short reference

Please configure the counters in the following sequence:

Sequence of instructions

1. disable specified counter (**Cnt_Enable**)

   The instruction **Cnt_Enable** always accesses all counters. Even if the status (disabled/enabled) of only one counter shall be changed, also those counters must be configured whose status shall remain unchanged.

2. clear counter (**Cnt_Clear**)
3. enable counter (**Cnt_Enable**)

For further processing of the counter value in the *ADbasic* program, transfer the value into latch A and read it from there.

5.3.3 Evaluation of the counter contents

The binary counters generate 32 bit values. Distinguish clearly between the evaluation of these binary values (e.g. differences) and the screen representation as decimal numbers.

To correctly evaluate the counter contents you need its original 32 bit values, especially with calculating differences. This is guaranteed only by use of *ADbasic* variables of type **Long**.

unchanged bit pattern

The representation of 32 bit values in *ADbasic* often leads to confusion, because the signless counter value is shown as signed decimal number (see circle below). Consequently a transition between positive and negative range of numbers is shown on the screen, which has no influence upon the evaluation of the counter contents.

For the sake of completeness the following describes the interpretation of a binary value:
The most significant bit (MSB) is interpreted as sign.
With a positive sign, the following 31 bits are directly interpreted as numerical value; binary and decimal value are similar.
With a negative sign, the 31 bits are first inverted, one added, and then interpreted as numerical value (2's complement); thus, negative decimal numbers have an absolute value different to the corresponding binary value.

Calculate differences only with integer numbers (**LONG**).

"Lap overflow"

For programming please remember that an "lap overflow" between the read out of two counts - i.e. the current counter value "laps" the last counter value which has been read - is not registered.

Circle

Such a lap overflow occurs after some 3½ minutes with an input frequency of 20 MHz or after more than 14 minutes with 5 MHz.

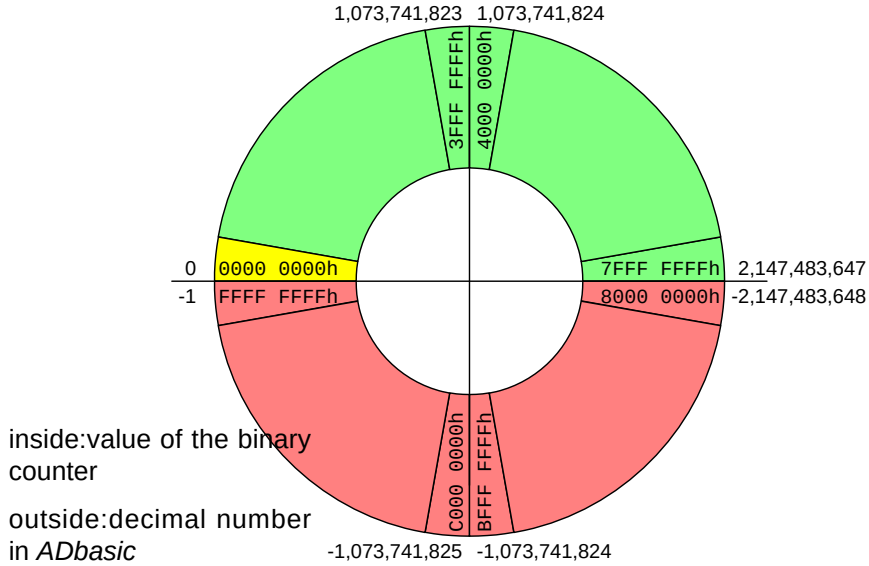


Fig. 14 – Circle model for interpretation of counter values

5.4 LS Bus

ADwin-light-16 provides an interface for LS bus on a 9-pin DSub connector (female); the pin assignemtn is shown on page 10.

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).

The variants L16-PCI and L16-Euro have the DSub connector on a separate cover plate. If the LS bus is not used, the connecting cable may be pulled from from the main board and the cover removed.
Please note for reconnecting to have the connector socket pick up all 10 pins of the on-board plug and to not have the cable twisted.

5.5 Time-critical tasks

Time-critical tasks

For extremely time-critical tasks instructions may be used which allow direct access to control and data registers of the hardware (see *ADbasic* manual or online help). These registers are to be found in the memory address area of the ADSP (memory mapped). These instructions also allow optimization of the program structure.

Contrary to the standard instructions **ADC()** and **DAC()** the instructions for direct access do not have any test routines. Before using them, good knowlegde is required about programming and time and function sequences in an analog-to-digital converter, because this kind programming is closely related to the hardware.

Analog inputs and outputs

ADC()

Execute the following *ADbasic* instructions instead of the standard instruction **ADC()** according to the following order:

Program structure

```
Set_Mux()
...                          'wait for settling time
Start_Conv()
Wait_EOC()                   'wait for end of conversion
ReadADC()
```

It is important to set a sufficient time-delay using additional programming instructions between the instructions **Start_Conv()** and **Set_Mux()**, in order to account for multiplexer settling time (see also *ADbasic* manual: "Instruction Reference").

Use the waiting times shown below for instance for computing operations and thus, save computing time:

- Settling time of the multiplexer: At a maximum voltage jump of 20 V it is 6.5 µs.
- Conversion time of the 16 bit ADC: 10 µs; since Rev. B: 2µs optional.

Hardware addresses of the control and data registers

Using the instructions **Peek** and **Poke** (see *ADbasic* manual or online help) you can directly access the control and data registers. This may accelerate the processing of the program, e.g.:

ADC

- a measurement can be executed very quickly.

DAC

- you can write very quickly into one or more DAC registers, and the output may be synchronously started.

Please ensure that the calculated analog outputs values are within the range limits.

The hardware addresses of the registers may be found in the following tables, grouped as analog inputs, analog outputs, digital inputs/outputs and counters.

Please take into account that some registers have an influence on several processes.

Address [HEX]	Function	Bit n° 31-16	15-10	9	8	7	6	5	4	3	2	1	0	Comment
20 40 00 00	set multiplexer to input channel (ADC 01... ADC 15)	-	-	-	-	-	-	0	0	0	n	n	n	"nnn" binary = 0...7 decimal; selected channel = nnn*2 + 1
20 40 00 10	start conversion: ADC#1	-	-	-	-	-	-	-	1	1	1	1	s	s = 0 : start conversion s = 1 : no effect
20 40 00 20	conversion status(EOC): ADC#1	-	-	-	-	-	-	-	-	-	-	-	e	e = 0 :end conversion e = 1 : conversion is running
20 40 00 30	read register: ADC#1	-	x	x	x	x	x	x	x	x	x	x	x	x : result of the conversion
20 40 01 00	read register and start conversion: ADC#1	-	x	x	x	x	x	x	x	x	x	x	x	

Fig. 15 – ADC hardware addresses of the control and data registers

Address [HEX]	Function	Bit n° 31-16	15-10	9	8	7	6	5	4	3	2	1	0	Comment
20 40 00 10	start conversion: all DAC synchronously	-	-	-	-	-	-	-	1	1	s	1	1	s = 0 : start conversion s = 1 : no effect
20 40 00 50	write only to the register: DAC #1	-	x	x	x	x	x	x	x	x	x	x	x	x : digital value to be converted
20 40 00 60	write only to the register: DAC #2	-	x	x	x	x	x	x	x	x	x	x	x	
20 40 02 00	write to the register and start conversion immediately: DAC #1	-	x	x	x	x	x	x	x	x	x	x	x	
20 40 02 10	write to the register and start conversion immediately: DAC #2	-	x	x	x	x	x	x	x	x	x	x	x	

Fig. 16 – DAC hardware addresses of the control and data registers

Address [HEX]	Function	Bit n° 31:16	15:6	5	4	3	2	1	0	Comment
20 40 00 B0	input register DIGIN-05...DIGIN-00	-	-	x	x	x	x	x	x	x : digital value read in
20 40 00 C0	output register DIGOUT-05 ... DIGOUT-00	-	-	x	x	x	x	x	x	x : digital value to be output
20 40 00 C4	DIGOUT Bit-SET-Register	-	-	0	0	0	0	0	0	no effect
		-	-	1	1	1	1	1	1	set bit
20 40 00 C8	DIGOUT Bit-CLEAR-Register	-	-	0	0	0	0	0	0	no effect
		-	-	1	1	1	1	1	1	clear bit

Fig. 17 – DIO hardware addresses of the control and data registers

Address [HEX]	Function	Bit n° 31:16	15:6	5	4	3	2	1	0	Comment
20 40 02 04	contents of latch A, counter #1	x	x	x	x	x	x	x	x	x : latched counter value
20 40 02 14	contents of latch A, counter #2	x	x	x	x	x	x	x	x	x : latched counter value
20 40 03 00	enable/disable counter (= **CNT_ENABLE**( ))	-	-	-	-	-	-	0	0	counter disabled
		-	-	-	-	-	-	1	1	counter enabled
20 40 03 10	clear counter(= **CNT_CLEAR**( ))	-	-	-	-	-	-	0	0	no effect
		-	-	-	-	-	-	1	1	clear counter *
20 40 03 20	latch counter (= **CNT_LATCH**( ))	-	-	-	-	-	-	0	0	no effect
		-	-	-	-	-	-	1	1	latch counter value into latch A *
* The bits are reset after the function has been executed. All other functions are reset by the opposing function.										

Fig. 18 – Counter hardware addresses of the control and data registers

6 Calibration

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

Calibration is made via software. The program <L16Calib.exe> is located in the Start folder at <Programs\ADwin\Calibration\L16Calib>.

You need the following tools for the calibration:

– a digital multimeter (DMM) with a resolution of 30 µV.

– a reference voltage source with a resolution of 30 µV. Optionally you connect DAC 1 to ADC 01(+), DAC 2 to ADC 03(+) and AGND DAC with ADC 01(-) and ADC 03(-), for instance in form of a test connector. You need these connections also for the calibration diagram.

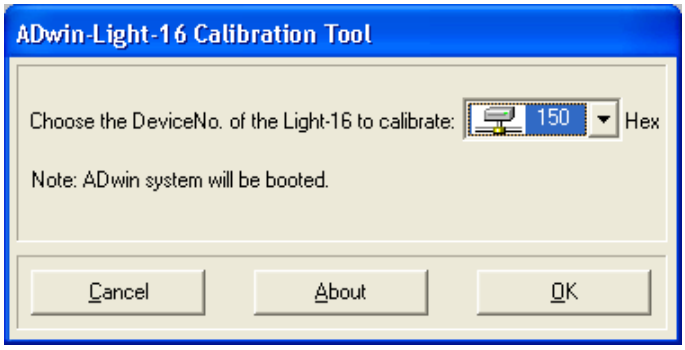
– connection cables from the inputs/outputs to the reference voltage source and to the measurement device.

Step 1

Connect your *ADwin-light-16* system with the PC and configure it with the program <ADconfig.exe>.

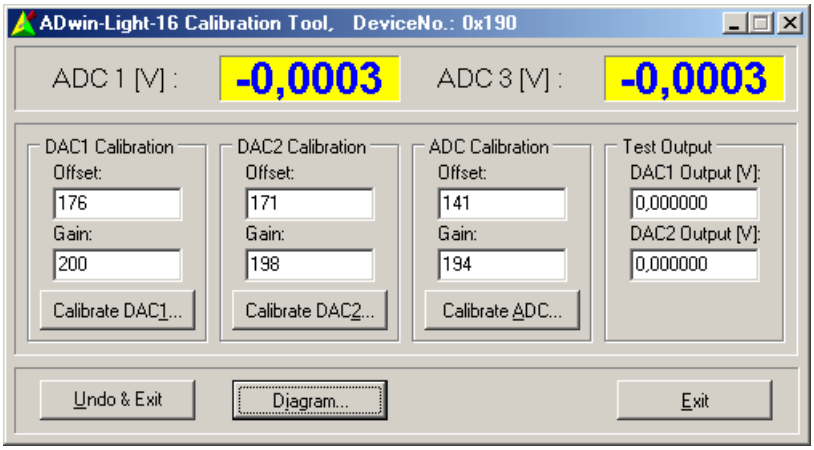
Step 2

Start the **calibration program** <L16Calib.exe>. The window "ADwin-light-16 Calibration Tool" appears.



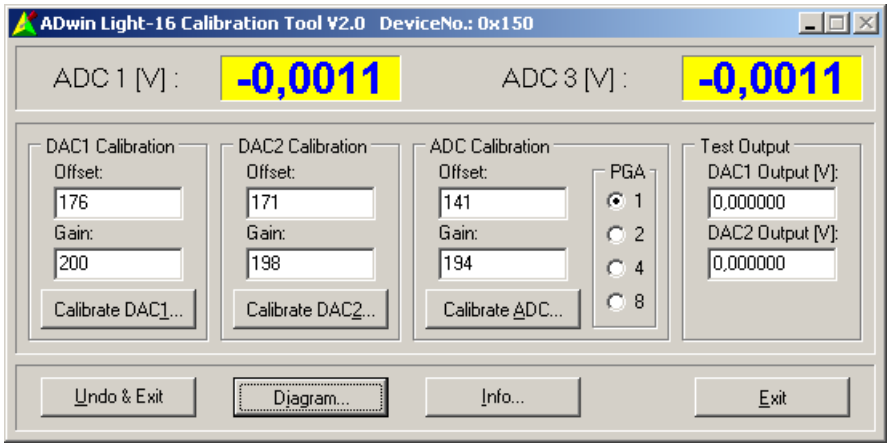
Select the device number of the system to be calibrated and confirm by pressing "OK".
You will get a warning, if you haven't selected an *ADwin-light-16* system or if you have selected one with an older firm ware version. You can ignore the warning with "YES" or return to the previous windows with "NO".

An **overview window** appears. In the header the selected device number is displayed.



The upper field shows the current measured values at the inputs ADC 01 and ADC 03. Below you will find the calibration settings for `Offset` and `Gain` of both DACs and the ADC; there you can directly enter values. You start calibration of the relevant converter with "`Calibrate ...`"
At the right side ("Test Output") you can enter voltage values in the fields DAC1 and DAC2, which will automatically be output on the corresponding outputs.

For Revision B the overview window has additional items: The PGA field so set the gain value and the `Info` button to show some version information of the device.



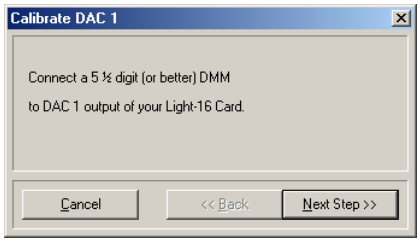
All settings you have made are automatically saved.
In the lower line you can undo all inputs with "`Undo & Exit`" and leave the calibration program. "`Diagram`" displays in a graph the accuracy of the current calibration setting. If you leave the program with "`Exit`" all settings remain.

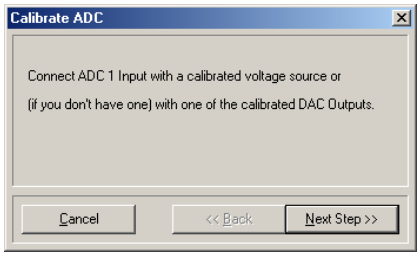
Calibrate the converters in the order you like (only with reference voltage source). The calibration of a converter is effected in 3 steps; you can switch between the windows of the steps by using the forward/backward buttons. Calibration is also possible without reference voltage source, but it will not be so precise. Calibrate first the DAC1 and DAC2, and then the ADC.

Step 3

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

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



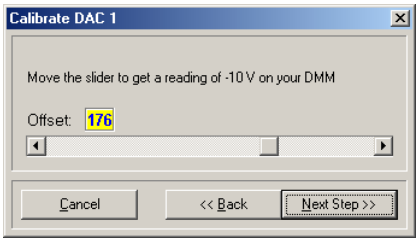


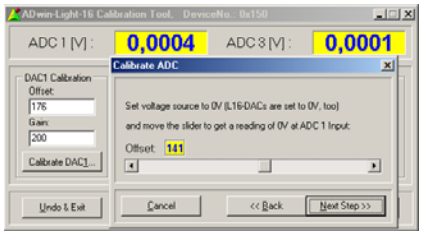
`Connect a DMM with the pins` AGND DAC and DAC1 or DAC2.

Connect the voltage source (or a DAC output) with the inputs ADC 01 or ADC 03.

Please note Fig. 9 – Pin assignment inputs/outputs (female).
Select "`Next Step` >>".

2. Set offset





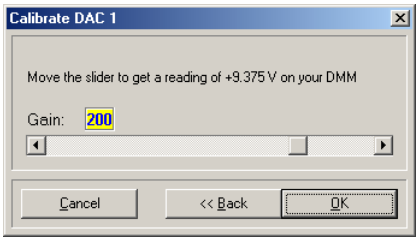
Adjust the offset value at the scroll-bar in such a manner that your digital multimeter displays -10 V.

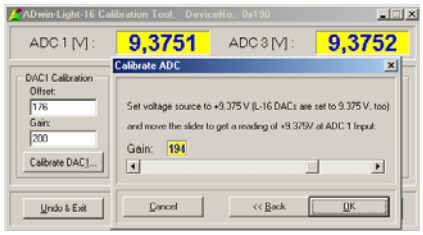
Set your voltage source to 0 V (set-point). The setting of the ADC to this value is made automatically.
Adjust the offset value at the scroll-bar in such a manner that the set-point at the ADC 01 is displayed in the overview window.

With Rev. B all settings of this step are done automatically.

Select "`Next Step` >>".

3. Set gain





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

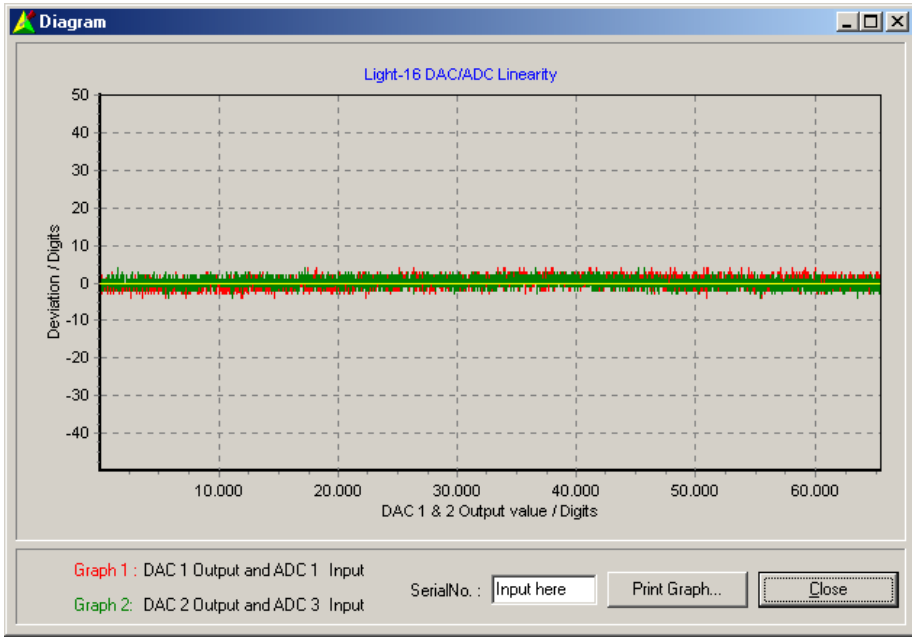
Set your voltage source to 9,375 V (setpoint). The setting of the ADC to this value is made automatically.
Adjust the offset value at the scroll-bar in such a manner that the set-point at the ADC 01 is displayed in the overview window.

With Rev. B all settings of this step are done automatically.

The calibration for this converter has finished. Select "`OK`". Repeat step 3 for the other converters if necessary.

Step 4

With a diagram (button `Diagram` in the overview window) you can check the **accuracy** of the calibration.
Connect first the output DAC 1 with the input ADC 01 as well as the output DAC 2 with the input ADC 03.



The program outputs the values 0...65,535 digits on both DACs, compares them to the measured input values and displays the deviation in graphs. Graph 1 (red) for DAC 1 / ADC 01 and graph 2 (green) for DAC 2 / ADC 03. The deviation should be smaller than 5 digits.

You can print the graph with `Print Graph` (a color printer is recommended). To do so, enter the serial number of your *ADwin* system, so that you can allocate the printout later. On the printout you will also find the calibration settings and the date of print.

With `Close` you return to the overview window.

Step 5

The calibration is finished.

CO1 Counter Add-On

7 CO1 Counter Add-On

The counter add-on CO1 (option *L16-CO1*) provides a 32-bit up/down counter with four edge evaluation and replaces the counter of the basic version (Fig. 19 – Block diagram of the L16-CO1 counter add-on shows the design of the counter).

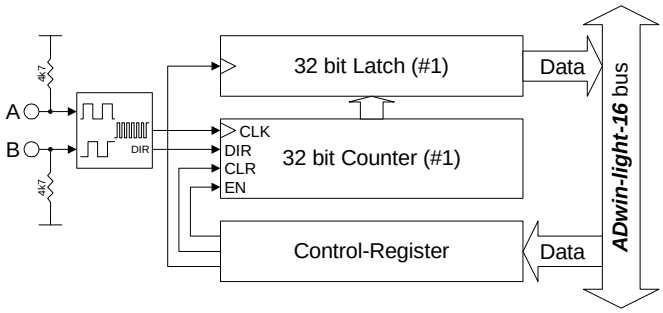


Fig. 19 – Block diagram of the *L16-CO1* counter add-on

7.1 Hardware

The counter is **externally clocked** and has a four edge evaluation for the connection of an encoder. The counter is read out via latch A, the control is made with ***ADbasic*** instructions via a control register (see "CO1 instructions, short reference" on page 24).

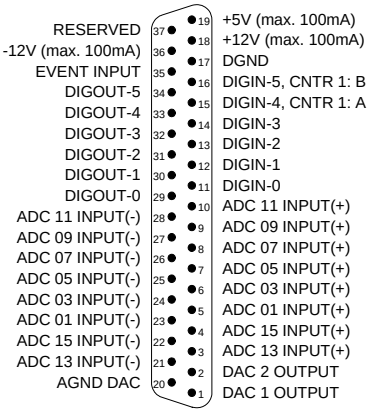
The four edge evaluation converts the digital input signals A and B (which are 90 degrees off-phase) into a clock (CLK) and direction signal (DIR) for the counter. Here a clock signal is generated from each edge of the A and B signals. The count direction (DIR) is determined by the order of the rising and falling edges of these signals.

Setting inputs

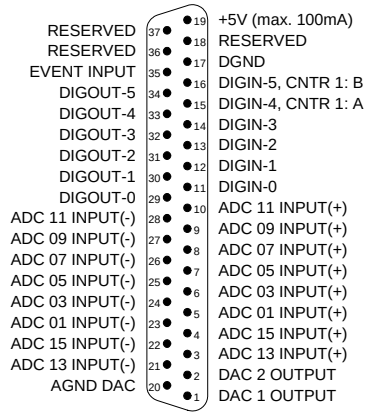
The clock inputs A and B are on the pins 15 and 16 of the D-SUB socket (see pin assignment below); TTL-compatible signals are necessary for the correct function. More details and limit values can be found in the Technical Data in the Annex.

Both inputs can be used optionally as digital signal input (see also chapter 5.2).

Although all inputs of the CO1 add-on have a pull-down resistor, open-ended inputs can cause errors in an environment which is not free of interferences. Therefore set unused inputs on a defined level (e.g. GND).



L16-PCI



L16-EURO and *L16-EXT*

Fig. 20 – Pin assignment of *L16-CO1*

Include file

7.2 Programming

The CO1 add-on is easily programmed by using ***ADbasic*** instructions. The instructions are part of an include file which must be included at the beginning of a program:

```
#Include ADWL16.INC
```

The instructions for the CO1 add-on are shortly illustrated in the following table and more detailed in the ***ADbasic*** manual or online help.

Counter nº Bit	1 0	Comment
Cnt_Clear( )	0	no effect
	1	clear counter*
Cnt_Enable( )	0	disable counter
	1	enable counter (pay attention to running counters)
Cnt_Latch( )	0	no effect
	1	copy counter value into latch A *
Cnt_ReadLatch(#)		read latch A (# = counter nº 1)
Cnt_Read(#)		copy counter value into latch A and read it (# = counter nº 1)
* The bits are reset after the function has been executed. All other functions are reset by the opposing function.		

Fig. 21 – CO1 instructions, short reference

Please evaluate the counter contents only with variables of type **Integer** or **Long**, above all when you want to evaluate differences or the count direction (see also page 15).

The count direction (up/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.

For extremely time-critical tasks you can use instructions with which you have direct access to control and data registers of the counter. In the table the corresponding hardware addresses are illustrated.

The hardware addresses of the CO1 counters are identical with or replace those of the basic counter version.

Address [HEX]	Function	Bit number 31:01	Bit number 00	Comment
20 40 02 04	contents of latch A, counter #1	x	x	x : latched counter value
20 40 03 00	enable counter **CNT_ENABLE()**	-	0	disable counter
		-	1	enable counter (pay attention to running counters)
20 40 03 10	clear counter **CNT_CLEAR()**	-	0	no effecft
		-	1	clear counter*
20 40 03 20	latch counter **CNT_LATCH()**	-	0	no effect
		-	1	latch counter*
* The bits are reset after the function has been executed. All other functions are reset by the opposing function.				

Fig. 22 – CO1 hardware addresses of the control and data registers

DIO1 Add-On

8 DIO1 Add-On

With the DIO1 add-on you are additionally provided with:

– 32 digital inputs/outputs (programmable in groups of 8), page 28

– 2 counters, page 29: 32 bit up/down counters for impulse, period duration and duty cycle measurements as well as a four edge evaluation for connection of incremental encoders.
Inputs can be set to single-ended or differential via DIP siwtches.

The counters of the basic version are replaced by the DIO1 counters.

– CAN interface (high-speed), page 36

– 1 SSI decoder (page 40), since Rev. B.

The SSI decoder enables the connection of an incremental encoder with SSI interface. The inputs are available on the COUNTER socket, the signals are differential and have RS422/485 levels (5V).

The block diagram shows the basic functions of an L16 system with the additional functions of the DIO1 add-on (as USB version).

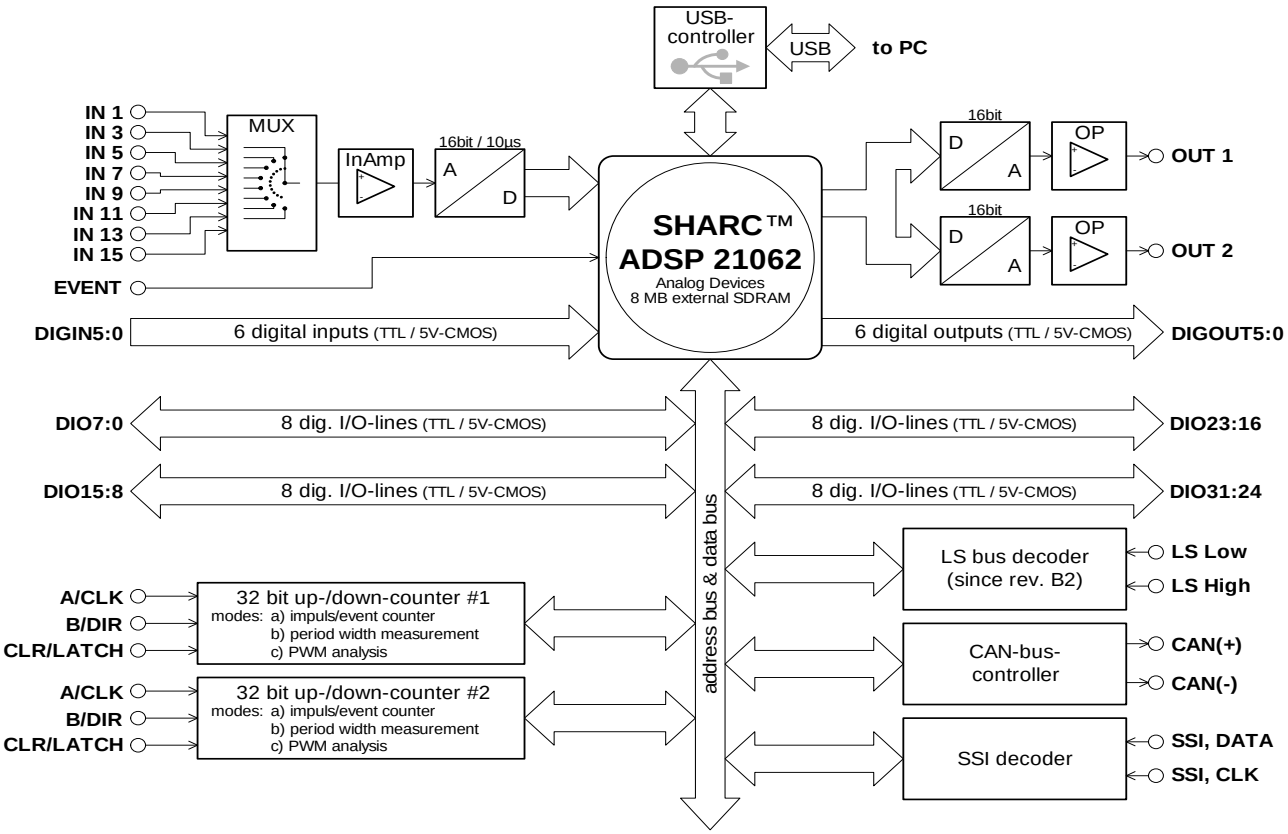


Fig. 23 – Block diagram of *L16-DIO1* (with USB interface)

The pin assignment at the connection "***ADwin*** I/O-CONNECTOR" is similar to the basic version, except one difference: The pins 15/16 - in the basic version each with double functions - are now solely used as DIGIN-04 and DIGIN-05.

The pin assignment of the LS bus interface is shown on page 10.

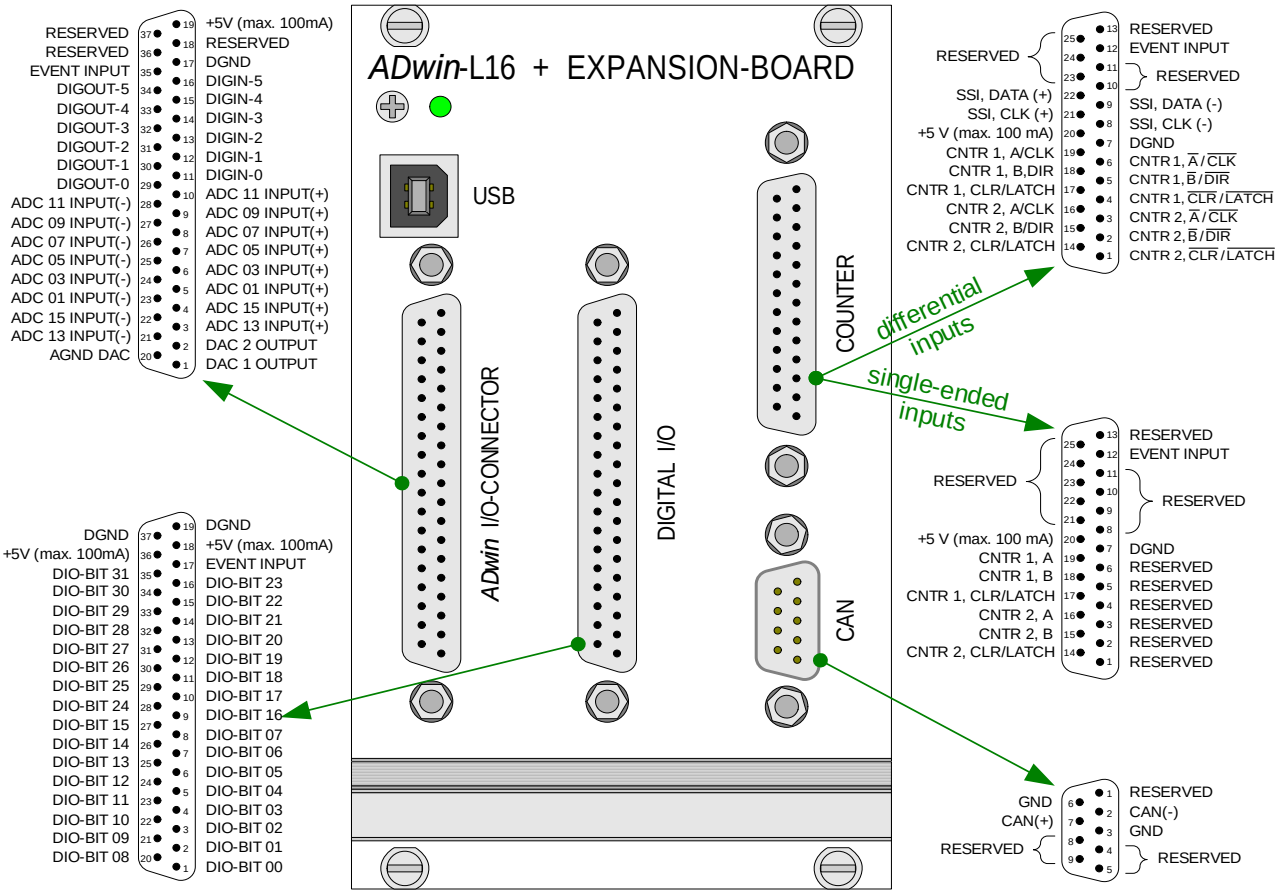


Fig. 24 – Overview of the *L16-EURO-DIO1* with pin assignments
For other L16 variants, the plugs are named identical.

Hardware configuration

There are several DIP switches on the DIO1 board, with which you can change the settings for the counters and the CAN interface.

The setting can only be made when the board is not installed or when the casing is opened. Please pay attention to the safety instructions at the beginning of this documentation.

variant	How to proceed
L16-EURO, L16-PCI	**Deinstalling the printed circuit board (PCB)** The deinstallation of the board is made in reverse order to the installation, which is described in the manual "***ADwin*** Driver Installation".
L16-EXT	**Opening the casing** You open the casing by unscrewing the upper Allen screws (2 mm) at both sides and loosening the lower screws. Slope the side plates and pull off the upper part of the casing. Note the orientation of the casing for the reinstallation. You are now looking directly on the DIO1 PCB.

Reassambling is made in reverse order.

Position of DIP switches

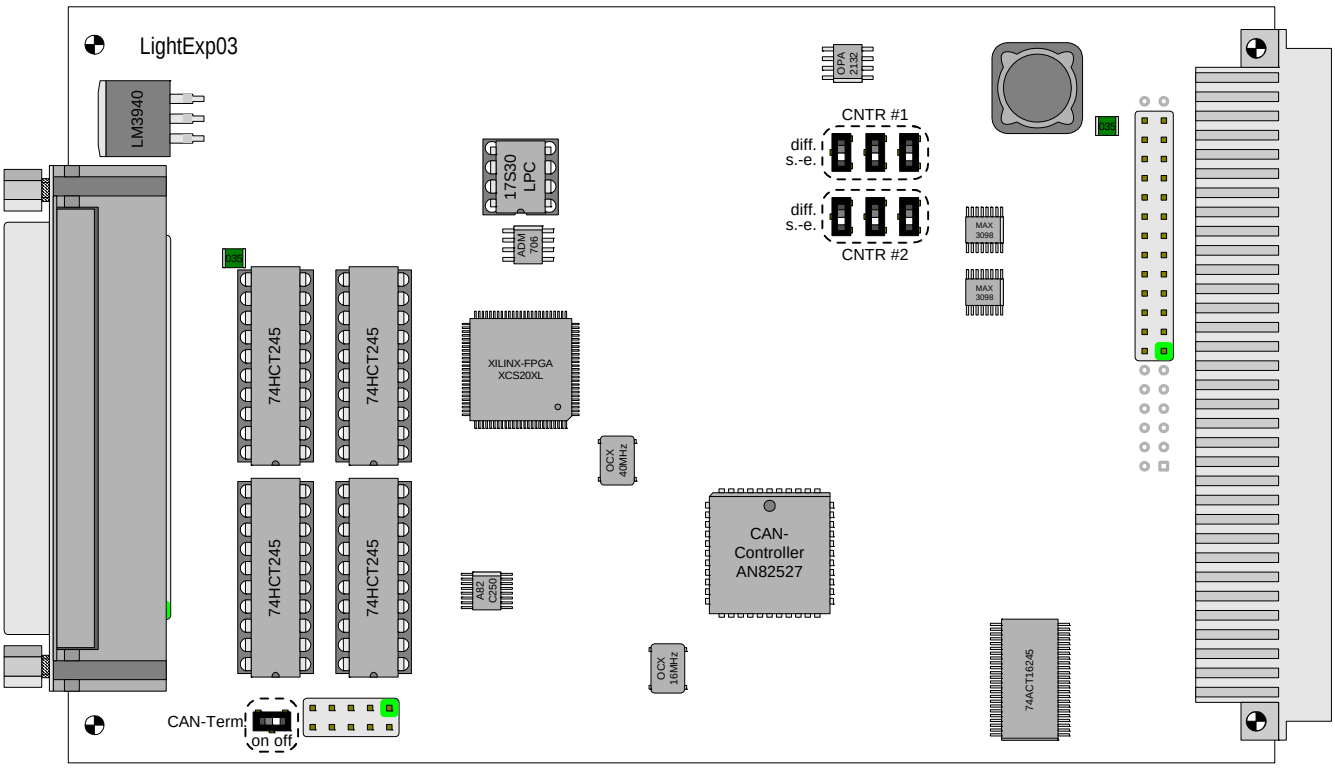
The exact position of the DIP switches is shown in the following figure (pay attention to the revision). The dotted frame shows the allocation of the DIP switches to the counters 1 and 2 (upper half, half at right) and to the CAN interface (lower left corner).

DIO1 Add-On

Counters 1 and 2: For differential mode set all 3 DIP switches of the corresponding counter into upward position (direction see also fig. 25); for single-ended operation set the switches into downward position.

CAN interface: If a termination is necessary, set the DIP switch to the top (L16 Rev. B) or toward the CAN connector (L16 Rev. A).

L16-DIO1 Rev. A



L16-DIO1 Rev. B

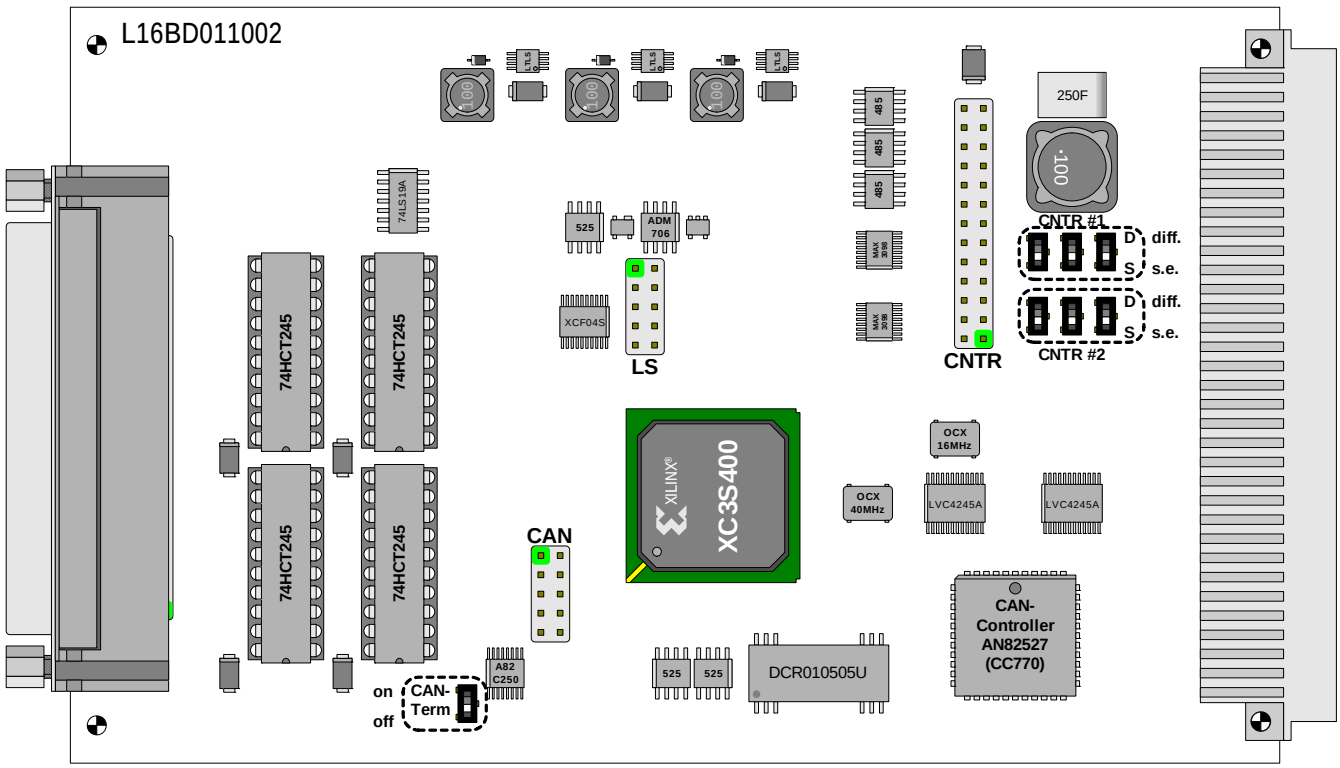


Fig. 25 – Position of the DIP switches on the DIO1 PCB

Find more information on setting of the DIP switches in chapter 8.2 "Counters" and chapter 8.3 "CAN-Bus".

The technical data of the DIO1 add-on is shown in the Annex.

8.1 Digital Inputs and Outputs

In addition to the digital inputs/outputs of the basic version (DIGIN, DIGOUT, EVENT), you have 32 digital inputs or outputs (abbrev. DIO) on the 37-pin D-SUB socket "Digital I/O". They are programmable in groups of 8 each as inputs or outputs.

Trigger input

There is an external trigger input (EVENT) on the following D-SUB sockets: `ADwin I/O Connector`, `Counter` and `Digital I/O`. The sockets "Counter" and "Digital I/O" are galvanically connected and have the same pull-up resistor of 4.7 kΩ, so that you can select one of the inputs if necessary. Use only one of the three Event inputs.

With an external signal (trigger) at the event input a process can be triggered, and can be processed immediately and completely (see ***ADbasic*** manual, chapter: "Structure of an ***ADbasic*** program").

The digital inputs are TTL compatible and are not protected against overvoltage.

Power-up configuration

After power-up all connections are configured as inputs; this corresponds to the instruction **`Conf_DIO_E(0)`**. With the instruction

`Conf_DIO_E(n)`

You program the 32 DIO lines in 4 groups with 8 lines each as input or output (see online help). The following table shows the 16 possible configurations you will get with this instruction. In order to use this instruction you have to include the file `<adwl16.inc>`.

`Conf_DIO_E``(n)`	DIO 31 ... DIO 24	DIO 23 ... DIO 16	DIO 15 ... DIO 08	DIO 07... DIO 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
10	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
ADbasic instructions to use:	**`Digin_Word2_E`** **`Digout_Word2_E`** **`Digout_Set2_E`** **`Digout_Reset2_E`**		**`Digin_Word1_E`** **`Digout_Word1_E`** **`Digout_Set1_E`** **`Digout_Reset1_E`**	

Fig. 26 – Configurations with **`Conf_DIO_E`**

More information about programming of time-critical tasks can be found in chapter 5.5 on page 17.

8.2 Counters

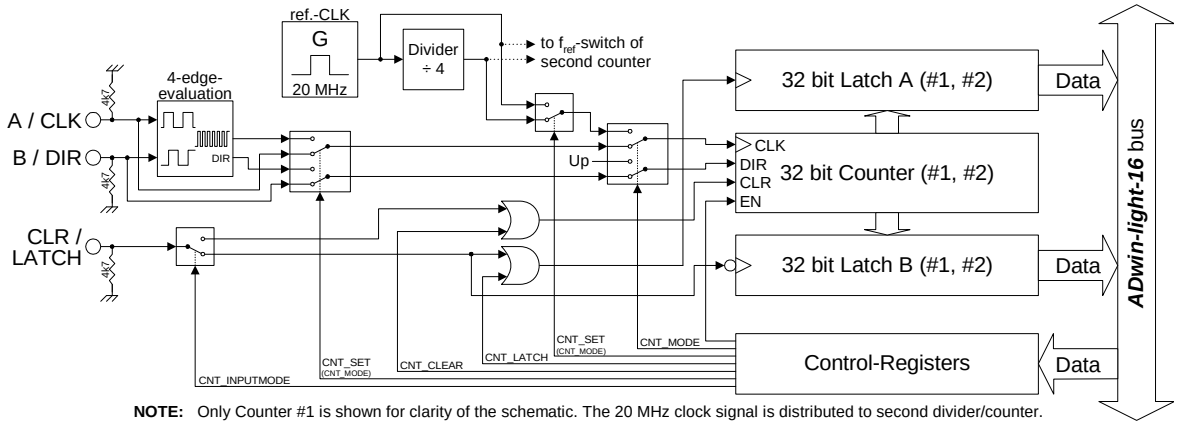
Counter

The add-on DIO1 provides **two 32-bit counters**, which you can configure and read out individually or all together. You can transfer single-ended or differential signals to the inputs; operating mode to be set via DIP switches, see below. The counters replace the incremental counters of the basic version.

Latch

The counters can be **internally or externally clocked** and are read out via accompanying latches. All counters have a latch A as well as a latch B (the figure shows the design of a single counter).

The counter values can be cleared or transferred into a latch by using programming instructions or (at special configurations) when there is an external signal at CLR/LATCH.



NOTE: Only Counter #1 is shown for clarity of the schematic. The 20 MHz clock signal is distributed to second divider/counter.

Fig. 27 – Block diagram of DIO1 counter

There are the following operating modes: event counting (external clock) and pulse width measurement (internal clock), see also chapter 8.2.2 / 8.2.3:

External clocking

4. **Event counting:** Incrementing/decrementing of the counter is caused by external square-wave signals at the inputs A/CLK and B/DIR. A signal at CLR/LATCH either sets the counter to zero (CLR) or has the counter value written into the latch (LATCH).

   There are the modes:
   - **Clock and direction**: Every positive edge at CLK increments or drecrements the counter value by one. The signal at DIR determines the counting direction (0 = down, 1 = up).
   - **Four edge evaluation**: Every edge of the signals (off-phase 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 incremental encoders.

Internal clock

5. **Pulse width measurement:** Incrementing/decrementing of the counter is caused by an internal reference clock with a signal frequency of 20 MHz (optionally 5 MHz after scaler). The square-wave signal at CLR/LATCH is evaluated: With every positive edge of the input signal the counter value is written to latch A, with a negative edge to latch B.

   You can calculate:
   - the period duration of the input signal at CLR/LATCH from the values in latch A and latch B.
   - the impulse width and pause time from the values in latch A and latch B.

Selecting the operating mode of the counter inputs

You can use the counter inputs in single-ended or differential mode. The setting is not fixed upon delivery. Therefore set the operating mode at the DIP switches of the DIO1 printed circuit board (position of DIP switches see fig. 25).

For differential mode set all 3 DIP switches of the corresponding counter into upward position (direction see also fig. 25); for single-ended operation set the switches into downward position.

8.2.1 Programming

The DIO1 counters are easily programmed by using ***ADbasic*** instructions. The instructions are part of an include file which must be included at the beginning of a program:

```
#Include ADWL16.INC
```

The instructions for the DIO1 counters are shortly illustrated in the following table and more detailed in the ***ADbasic*** manual or online help.

Counter n°	2	1	Comment
Bit	1	0	
Cnt_Clear( )	0	0	no effect
	1	1	clear counter*
Cnt_Enable( )	0	0	disable counter
	1	1	enable counter (pay attention to running counters)
Cnt_InputMode( )	0	0	set CLR/LATCH input to CLR mode
	1	1	set CLR/LATCH input to LATCH mode
Cnt_Latch( )	0	0	no effect
	1	1	copy counter value to latch register A*
Cnt_Mode( )	0	0	external clock input
	1	1	internal reference clock (20MHz / 5MHz)
Cnt_Set( )	0	0	**Cnt_Mode** bit = 0 : 4 edge evaluation (A & B)
			Cnt_Mode bit = 1 : internal reference clock of 20MHz
	1	1	**Cnt_Mode** bit = 0 : clock and direction inputs (CLK & DIR)
			Cnt_Mode bit = 1 : internal reference clock of 5MHz
Cnt_ClearEnable()	0	0	disable CLR input
	1	1	enable CLR input
Cnt_GetStatus(#) **			read status register (for the meaning of the bits see ADbasic manual or online help)
Cnt_Read(#) **			copy counter value to latch A and read it
Cnt_ReadLatch(#) **			read out latch A (triggered by positive edge)
Cnt_ReadFLatch(#) **			read out latch B (triggered by a negative edge)

* The bits are reset after the function has been executed. All other functions are reset by the opposing function.
** # = counter no. 1 or 2

Fig. 28 – DIO1 counter instructions - short reference

With these instructions of the table matrix you will be able to configure every counter individually or both counters together.

Please initialize the counters in the following order:

1. disable specified counter (**Cnt_Enable**)

   The instruction **Cnt_Enable** always accesses all counters. Even if you want to change the status (disabled/enabled) of only one counter, you

DIO1 Add-On

also have to configure the counters whose status shall remain unchanged.

2. set operating mode (**Cnt_Mode**, **Cnt_Set**, **Cnt_InputMode**)

   Please take into account that the instruction **Cnt_Set** is dependent on the instruction **Cnt_Mode**.

3. clear counter (**Cnt_Clear**)

4. enable counter (**Cnt_Enable**)

Evaluate the counter value only with **Integer** or **Long** variables. ***ADbasic*** then keeps internally the read bit patterns unmodified and considers automatically the transition from the positive to the negative numerical range (see also page 15). Then you get:

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 value.

Calculate the difference only with integer variables. (**Integer**, **Long**).

A task can be very quickly processed if you access the control and data registers directly (see chapter 5.5 as well as ***ADbasic*** manual). The hardware addresses of the DIO1 add-on are illustrated in the following table (see also Fig. 28 – DIO1 counter instructions - short reference).

Address [hex]	Function	Bit 31:16	15:10	9	8	7	6	5	4	3	2	1	0	Comment
20 40 02 04	contents of latch A, counter #1	x	x	x	x	x	x	x	x	x	x	x	x	x : contents of the latch
20 40 02 08	contents of latch B, counter #1	x	x	x	x	x	x	x	x	x	x	x	x	x : contents of the latch
20 40 02 14	contents of latch A, counter #2	x	x	x	x	x	x	x	x	x	x	x	x	x : contents of the latch
20 40 02 18	contents of latch B, counter #2	x	x	x	x	x	x	x	x	x	x	x	x	x : contents of the latch
20 40 03 00	enable/disable counter **Cnt_Enable()**	-	-	-	-	-	-	-	-	x	x	x	x	x = 0 : disable counter x = 1 : enable counter
20 40 03 10	clear counter **Cnt_Clear()**	-	-	-	-	-	-	-	-	x	x	x	x	x = 0 : no effect x = 1 : clear counter
20 40 03 20	latch counter **Cnt_Latch()**	-	-	-	-	-	-	-	-	x	x	x	x	x = 0 : no effect x = 1 : latch counter
20 40 03 30	input:: CLR or LATCH	-	-	-	-	-	-	-	-	x	x	x	x	x = 0 : CLR input x = 1 : LATCH input
20 40 03 40	impulse/event counter or pulse width/period duration measurement	-	-	-	-	-	-	-	-	x	x	x	x	x = 0 : external clock input x =1: internal reference clock (20MHz/5MHz)
20 40 03 50	4 edge evaluation/CLK + DIR or 20MHz/5MHz refeference clock	-	-	-	-	-	-	-	-	x	x	x	x	**Cnt_Mode**=0: x=0: 4-Fl.;x=1:CLK+DIR **Cnt_Mode**=1: x=0: 20MHz; x=1: 5MHz
20 40 04 54	DIO1 add-on bit 0...15	-	x	x	x	x	x	x	x	x	x	x	x	x: sets/clears output <bit n^{o}>
20 40 04 64	DIO1 add-on bit 16...31	-	x	x	x	x	x	x	x	x	x	x	x	x: sets/clears output <bit n^{o}+16>
20 40 04 74	DIO1 add-on set_bit 0...15	-	x	x	x	x	x	x	x	x	x	x	x	x = 0: no effect x = 1: set output <bit n^{o}> *
20 40 04 84	DIO1 add-on set_bit 16...31	-	x	x	x	x	x	x	x	x	x	x	x	x = 0: no effect x = 1: set output <bit n^{o}+16>*
20 40 04 94	DIO1 add-on reset_bit 0...15	-	x	x	x	x	x	x	x	x	x	x	x	x = 0: no effect x = 1: clear output <bit n^{o}> *
20 40 04 A4	DIO1 add-on reset_bit 16...31	-	x	x	x	x	x	x	x	x	x	x	x	x = 0: no effect x = 1: clear output <bit n^{o}+16> *
20 40 04 6C	configure inputs/outputs **Conf_DIO**() bit 0: DIO 00...07; bit 1: DIO 08...15 bit 2: DIO 16...23; bit 3: DIO 24...31	-	-	-	-	-	-	-	-	x	x	x	x	x = 0: configure channels as input x = 1: configure channels as output
* Function has no effect with inputs														

Fig. 29 – DIO1 hardware addresses of the control and data register

8.2.2 Operating mode impulse/event counting

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

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

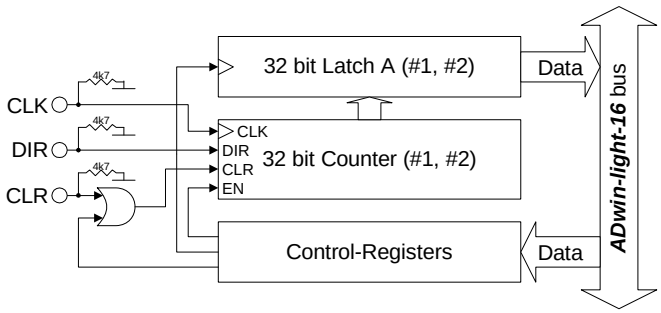
Clearing

– clear the counter (CLR)

Latching

– latch the counter value into latch A (LATCH).

Clock and direction



Every positive edge of a square-wave signal at the clock input (CLK) is counted (incremented or decremented) up to a maximum frequency of 20 MHz. The direction is derived from a high signal (increment) or low signal (decrement) at the direction input (DIR); this signal can be a fixed voltage or a dynamic signal (e.g. given by an external logic).

Programming example

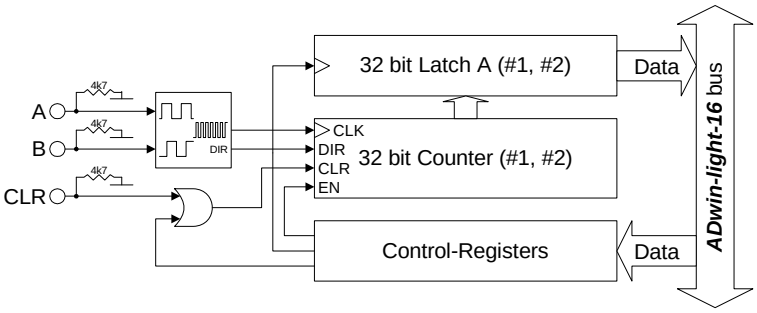
```
INIT:                          initialize
. . .
CNT_ENABLE(0)                  disable counter
CNT_CLEAR(1)                   clear counter
CNT_MODE(0)                    external clock input at the CLK input
CNT_SET(1)                     set mode clock and direction
CNT_INPUTMODE(0)               set input CLR/LATCH to CLR mode
CNT_ENABLE(1)                  enable counter
. . .

                               latch current counter value to latch A
EVENT:
CNT_LATCH(1)                   read latch A
CNT_READLATCH(1)               evaluation in the program
. . .
```

DIO1 Add-On

Four edge evaluation

This mode determines clock and direction of two signals, which are input at A and B off-phase by 90 degrees (ideally). The count direction is determined by the temporal sequence of the rising and falling edges of the two input signals.



Please note:

– The counter counts 4 edges in each cycle.

– The maximum count frequency is 20 MHz. Together with the 4 edges per cycle it will result in a maximum input frequency of 5 MHz (at A or B).

– The time lap 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 (to ≠90 degrees) will have an effect on the maximum input frequency because of the minimum time lap of the edges. If it differs from 90 degrees, the maximum input freqeuency of 5 MHz decreases for instance to 45 degrees at 2.5 MHz.

Programming example

```
INIT:                          initialize
  . . .
CNT_ENABLE(0)                  disable counter
CNT_CLEAR(1)                   clear counter
CNT_MODE(0)                    external clock input at CLK input
CNT_SET(0)                     enable mode four edge evaluation
CNT_INPUTMODE(0)               set input CLR/LATCH to CLR mode
CNT_ENABLE(1)                  enable counter
  . . .

                               latch current counter value to latch A
EVENT:                         read latch A
CNT_LATCH(1)                   evaluation in the program
CNT_READLATCH(1)
  . . .
```

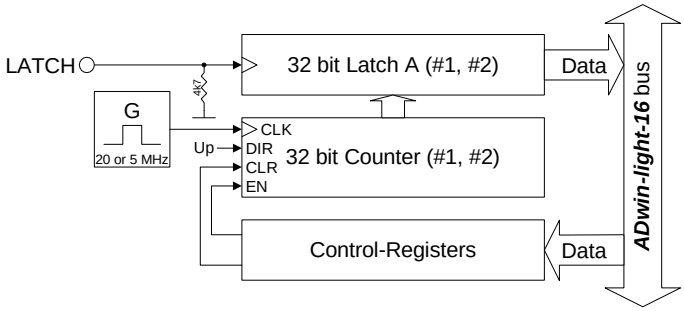
8.2.3 Operating mode pulse width and period duration measurement

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

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

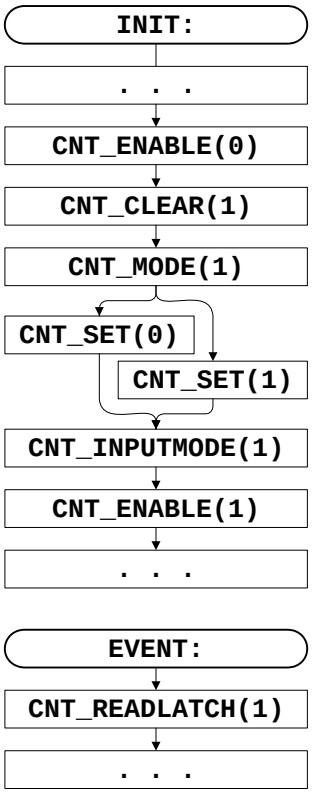
Period duration measurement

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



DIO1 Add-On

Programming example



initialize

disable counter

clear counter

mode internal reference clock at internal CLK input of the counter ...

... with 20 MHz or

... with 5 MHz

set input CLR/LATCH to LATCH mode

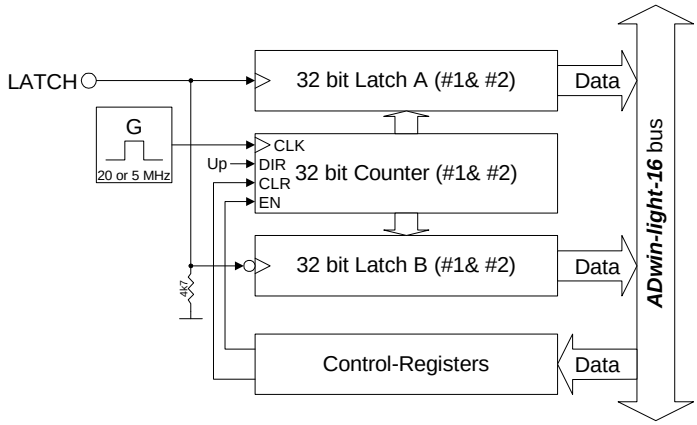
enable counter

read latch A

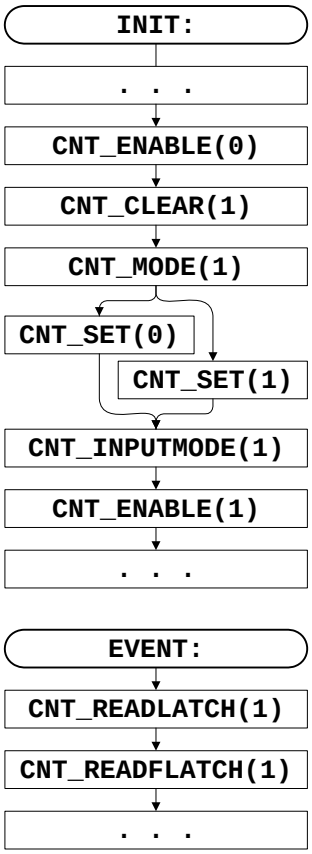
evaluation in the program

Pulse width and pause duration measurement

The counters have each a latch A for positive and a latch B for negative edges. Thus, pulse and pause duration can be evaluated separately by calculating the differences of the latches.



Programming example



8.3 CAN-Bus

The add-on board DIO1 has a CAN bus interface for the high-speed CAN protocol. The connections are available on a 9-pin D-SUB connector (male); the pin assignment can be found on page 26.

Bus termination

The CAN bus has to be terminated at its physical end (and only there) by a resistor, that means only at the first and the last CAN node. This is made by setting a corresponding DIP switch on the DIO1 printed circuit board. (see Fig. 25 – Position of the DIP switches on the DIO1 PCB).

If a termination is necessary, set the DIP switch to the top (L16 Rev. B) or toward the CAN connector (L16 Rev. A).

8.3.1 Functions of the CAN controller

The CAN bus interface is equipped with the Intel® CAN controller AN82527 which works according to the specification CAN 2.0 parts A and B as well as to ISO 11898. You program the interface with *ADbasic* instructions, which are directly accessing the controller's registers.

Message

Messages sent via CAN bus are data telegrams with up to 8 bytes, which are characterized by so-called identifiers. The CAN controller supports identifiers with a length of 11 bit and 29 bit. The communication, that means the management of bus messages, is effected by 15 message objects.

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

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

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

Message Management

Identifier

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

Message objects

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

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

Transferring messages

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

Sending messages

Sending a message is made as follows:

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

Receiving messages

Receiving a message is made as follows:

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

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

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

Assigning messages

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

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

Global mask

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

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

x: Message is admitted
0: Message is not admitted

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

Setting the bus frequency

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

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

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

Bus frequency for special cases

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

Enable Interrupt / Trigger Event

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

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

If message objects be enabled to trigger interrupts, the event inputs (see page 28) may not be wired at the same time.

Programming

The DIO1 CAN interface is easily programmed by using ***ADbasic*** instructions. The instructions are part of an include file which must be included at the beginning of a program:

```
#Include ADWL16.INC
```

DIO1 Add-On

The instructions for the DIO1 CAN interface are shortly illustrated in the following table and more detailed in the ***ADbasic*** manual or online help.

Function	Instruction
Initializing the CAN controller	**Init_CAN**
Setting and reading of registers	Set_CAN_Reg, **Get_CAN_Reg**
Initializing of message objects	En_Receive, **En_Transmit**
Transmitting and receiving data sets	**Transmit** Read_Msg, **Read_Msg_Con**
Enabling of interrupts	**En_Interrupt**
Setting the Baud rate	**Set_CAN_Baudrate**

Fig. 30 – DIO1 Overview of CAN instructions

8.4 SSI Decoder

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

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

The connections of the decoder are on the connectors COUNTER (25-pin, DSUB), on the pins 8, 9, 21 and 22 (see Fig. 24 – Overview of the L16-EURO-DIO1 with pin assignments For other L16 variants, the plugs are named identical.). The pins 7 and 20 provide DGND and +5V.

Setting properties

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

– Clock rates: With **SSI_Set_Clock** clock rates of approx. 40kHz up to 1MHz 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. 31 – Listing: Conversion of Gray code into binary code

Programming

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

Range	Instructions
Initialization	**SSI_INIT**
Receive data	**SSI_READ**
Set decoder resolution	**SSI_SET_BITS**
Set decoder clock rate	**SSI_SET_CLOCK**
Start reading encoder data	**SSI_START**
Return reading status	**SSI_STATUS**

The instructions are in the include file <ADWL16.INC>. More information can be found in the ***ADbasic*** manual and the online help.

DIO2 / DIO3 Add-On

9 DIO2 / DIO3 Add-On

With the DIO2 add-on you are additionally provided with:

– 32 digital inputs/outputs, programmable in groups of 8; page 43.

– 2 counters, page 44: 32 bit up/down counters for impulse, period duration and duty cycle measurements as well as a four edge evaluation for connection of incremental encoders.
Counter 1 has TTL inputs (single-ended), counter 2 has differential inputs.

The counters of the basic version are replaced by the DIO2 counters.

– 1 SSI decoder (page 51)

The SSI decoder is connected to an incremental encoder with SSI interface. The inputs are available on the "Counter" socket, the signals are differential and have RS422/485 levels.

The block diagram shows the basic functions of an L16 system with the additional functions of the DIO2 add-on (as USB version).

The DIO3 add-on contains 32 digital inputs/outputs only (see page 43). Properties and function of the inputs/outputs are identical to the I/Os of the DIO2 add-on.

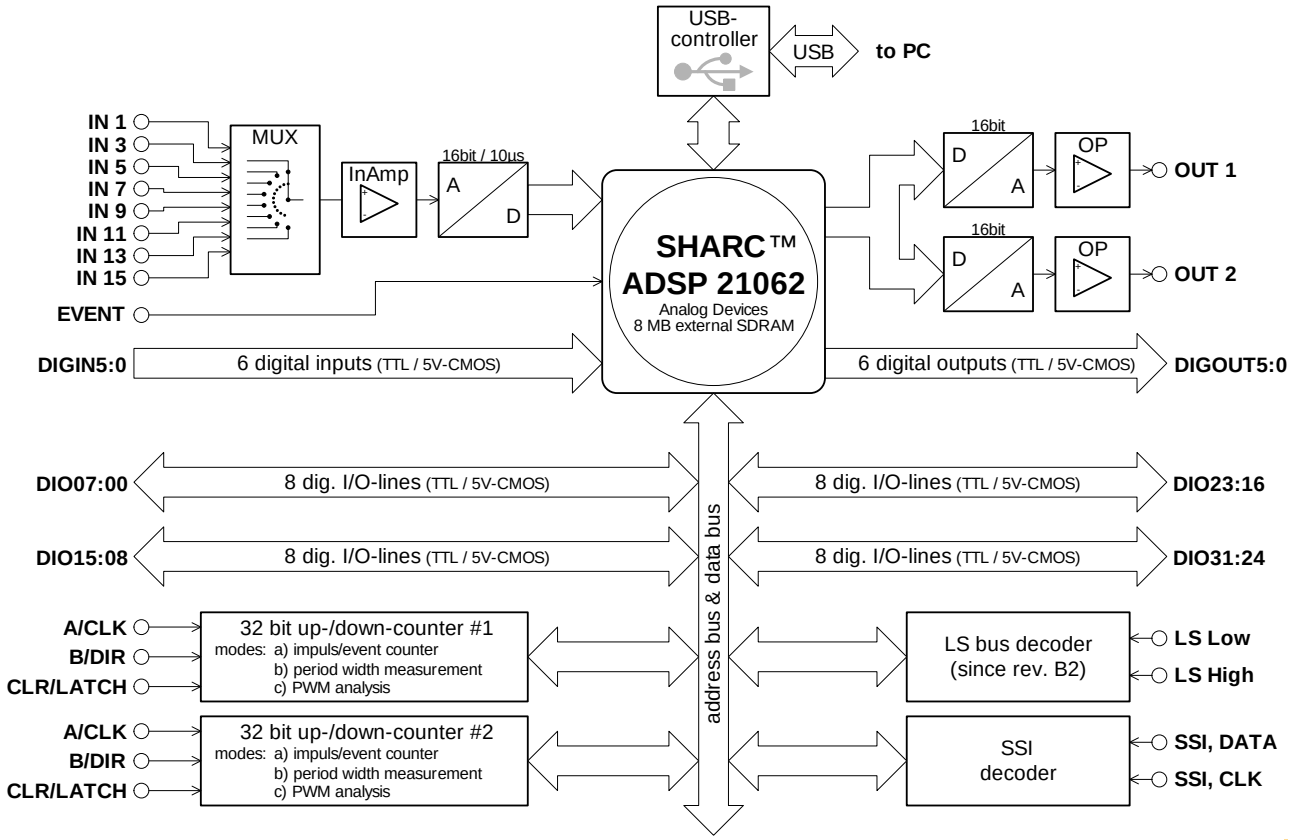


Fig. 32 – Block diagram of *L16-DIO2* (with USB interface)

The pin assignment at the connection "***ADwin*** I/O-CONNECTOR" is similar to the basic version, except one difference: Pin 13 - in the basic version with sin-

gle function - is now used alternatively as digital input Digin-03 or as counter input CLR for counter 1.

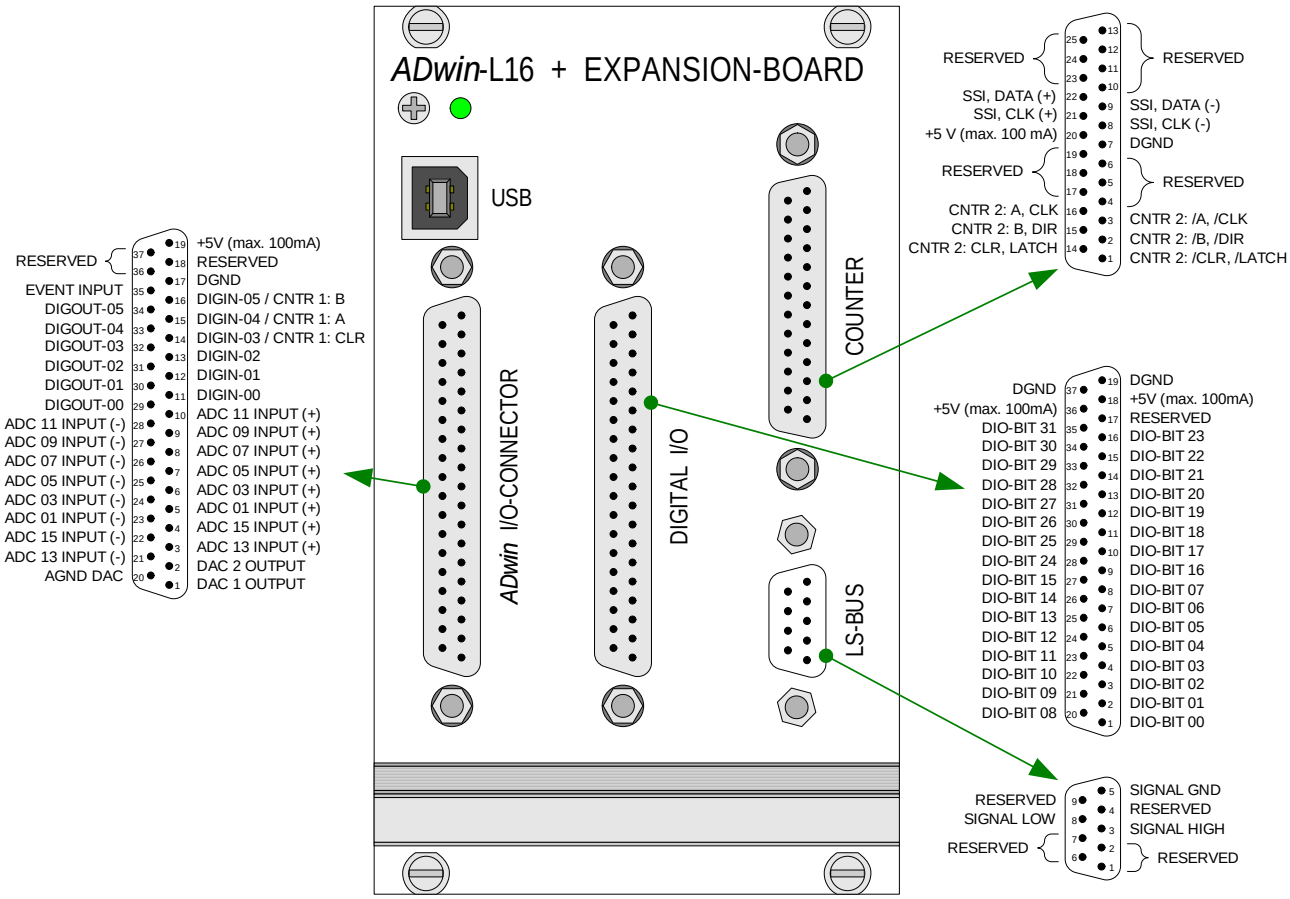


Fig. 33 – Overview of the *L16-EURO-DIO2* with pin assignments

The technical data of the DIO2 add-on are shown in the Annex.

DIO2 / DIO3 Add-On

9.1 Digital Inputs and Outputs

In addition to the digital inputs/outputs of the basic version (Digin, Digout, EVENT), you have 32 digital inputs or outputs (abbrev. DIO) on the 37-pin D-SUB socket "Digital I/O". They are programmable in groups of 8 each as inputs or outputs.

The digital inputs are TTL compatible and are not protected against overvoltage.

Power-up configuration

After power-up all connections are configured as inputs; this corresponds to the instruction **Conf_DIO_E(0)**.

Conf_DIO_E(n) is the instruction to program the 32 DIO lines in 4 groups with 8 lines each as input or output (see chapter 13 or ***ADbasic*** online help).

The following table shows the 16 possible configurations of the instruction. The last row shows the instructions availabel for the different DIO groups.

Conf_DIO_E(n)	DIO 31 ... DIO 24	DIO 23 ... DIO 16	DIO 15 ... DIO 08	DIO 07... DIO 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
10	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
ADbasic instructions to use:	**Digin_Word2_E** **Digout_Word2_E** **Digout_Set2_E** **Digout_Reset2_E**		**Digin_Word1_E** **Digout_Word1_E** **Digout_Set1_E** **Digout_Reset1_E**	
	Digin_Long_E, **Digout_Long_E**			

Fig. 34 – Configurations with **Conf_DIO_E**

More information about programming of time-critical tasks can be found in chapter 5.5 on page 17.

9.2 Counters

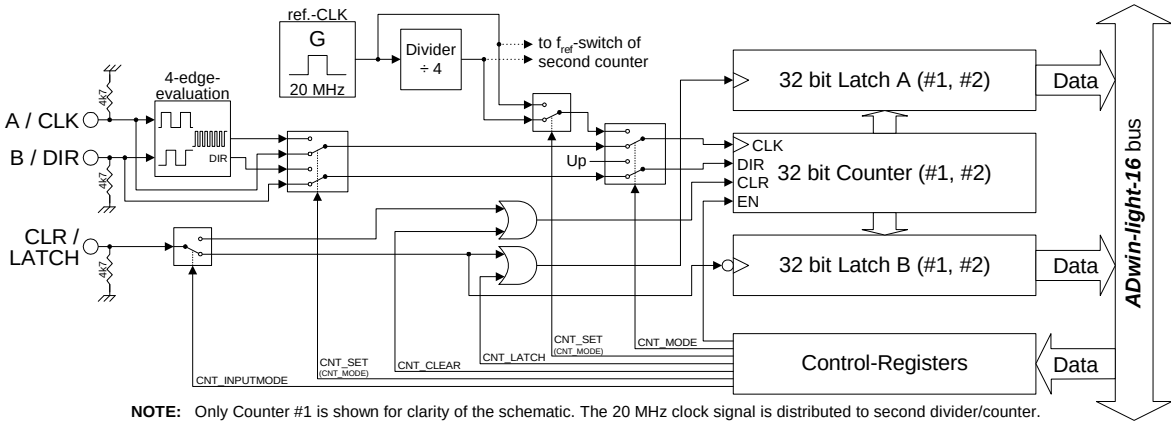
Counter

The add-on DIO2 provides **two 32-bit counters**, which you can configure and read out individually or all together. You can use TTL signals (single-ended) with counter 1 or differential signals with counter 2.
The counters replace the incremental counters of the basic version.

Latch

The counters can be **internally or externally clocked** and are read out via accompanying latches. All counters have a latch A as well as a latch B (the figure shows the design of a single counter).

The counter values can be cleared or transferred into a latch by using programming instructions or (at special configurations) when there is an external signal at CLR/LATCH.



NOTE: Only Counter #1 is shown for clarity of the schematic. The 20 MHz clock signal is distributed to second divider/counter.

Fig. 35 – Block diagram of DIO2 counter

External clocking

There are the following operating modes: event counting (external clock) and pulse width measurement (internal clock), see also chapter 9.2.2 / 9.2.3:

5. **Event counting:** Incrementing/decrementing of the counter is caused by external square-wave signals at the inputs A/CLK and B/DIR. A signal at CLR/LATCH either sets the counter to zero (CLR) or has the counter value written into the latch (LATCH).

   There are the modes:
   - **Clock and direction**: Every positive edge at CLK increments or drecrements the counter value by one. The signal at DIR determines the counting direction (0 = down, 1 = up).
   - **Four edge evaluation**: Every edge of the signals (off-phase 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 incremental encoders.

Internal clock

6. **Pulse width measurement:** Incrementing/decrementing of the counter is caused by an internal reference clock with a signal frequency of 20 MHz (optionally 5 MHz after scaler). The square-wave signal at CLR/LATCH is evaluated: With every positive edge of the input signal the counter value is written to latch A, with a negative edge to latch B.

   You can calculate:
   - the period duration of the input signal at CLR/LATCH from the values in latch A and latch B.
   - the impulse width and pause time from the values in latch A and latch B.

DIO2 / DIO3 Add-On

9.2.1 Programming

The DIO2 counters are easily programmed by using ***ADbasic*** instructions. The instructions are part of an include file which must be included at the beginning of a program:

```
#INCLUDE ADWL16.INC
```

The instructions for the DIO2 counters are shortly illustrated in the following table and more detailed in the ***ADbasic*** manual or online help.

Counter nº	2	1	Comment
Bit	1	0	
Cnt_Clear( )	0	0	no effect
	1	1	clear counter*
Cnt_Enable( )	0	0	disable counter
	1	1	enable counter (pay attention to running counters)
Cnt_InputMode( )	0	0	set CLR/LATCH input to CLR mode
	1	1	set CLR/LATCH input to LATCH mode
Cnt_Latch( )	0	0	no effect
	1	1	copy counter value to latch register A*
Cnt_Mode( )	0	0	external clock input
	1	1	internal reference clock (20MHz / 5MHz)
Cnt_Set( )	0	0	**Cnt_Mode** bit = 0 : 4 edge evaluation (A & B)
			Cnt_Mode bit = 1 : internal reference clock of 20MHz
	1	1	**Cnt_Mode** bit = 0 : clock and direction inputs (CLK & DIR)
			Cnt_Mode bit = 1 : internal reference clock of 5MHz
Cnt_ClearEnable()	0	0	disable CLR input
	1	1	enable CLR input
Cnt_GetStatus(#) **			read status register (for the meaning of the bits see ADbasic manual or online help)
Cnt_Read(#) **			copy counter value to latch A and read it
Cnt_ReadLatch(#) **			read out latch A (triggered by positive edge)
Cnt_ReadFLatch(#) **			read out latch B (triggered by a negative edge)

* The bits are reset after the function has been executed. All other functions are reset by the opposing function.
** # = counter no. 1 or 2

Fig. 36 – DIO2 counter instructions - short reference

With these instructions of the table matrix you will be able to configure every counter individually or both counters together.

Please initialize the counters in the following order:

1. disable specified counter (**Cnt_Enable**)

   The instruction **Cnt_Enable** always accesses all counters. Even if you want to change the status (disabled/enabled) of only one counter, you also have to configure the counters whose status shall remain unchanged.

2. set operating mode (**Cnt_Mode**, **Cnt_Set**, **Cnt_InputMode**)

   Please take into account that the instruction **Cnt_Set** is dependent on the instruction **Cnt_Mode**.

3. clear counter (**Cnt_Clear**)
4. enable counter (**Cnt_Enable**)

Evaluate the counter value only with LONG variables. ***ADbasic*** then keeps internally the read bit patterns unmodified and considers automatically the transition from the positive to the negative numerical range (see also page 15). Then you get:

The count direction (up or down) can reliably be derived from the

sign of the difference: [new counter value] – [old counter value]

and not from the *comparison* (< >) of the counter value.

Calculate the difference only with variables of type LONG.

A task can be very quickly processed if you access the control and data registers directly via **Peek** and **Poke** (see chapter 5.5 as well as ***ADbasic*** manual / online help). The hardware addresses of the DIO2 add-on are illustrated in the following table (see also Fig. 36 – DIO2 counter instructions - short reference).

Address [hex]	Function	Bit 31:16	15:10	9	8	7	6	5	4	3	2	1	0	Comment
20 40 02 04	contents of latch A, counter #1	x	x	x	x	x	x	x	x	x	x	x	x	x : contents of the latch
20 40 02 08	contents of latch B, counter #1	x	x	x	x	x	x	x	x	x	x	x	x	x : contents of the latch
20 40 02 14	contents of latch A, counter #2	x	x	x	x	x	x	x	x	x	x	x	x	x : contents of the latch
20 40 02 18	contents of latch B, counter #2	x	x	x	x	x	x	x	x	x	x	x	x	x : contents of the latch
20 40 03 00	enable/disable counter **Cnt_Enable()**	-	-	-	-	-	-	-	-	x	x	x	x	x = 0 : disable counter x = 1 : enable counter
20 40 03 10	clear counter **Cnt_Clear()**	-	-	-	-	-	-	-	-	x	x	x	x	x = 0 : no effect x = 1 : clear counter
20 40 03 20	latch counter **Cnt_Latch()**	-	-	-	-	-	-	-	-	x	x	x	x	x = 0 : no effect x = 1 : latch counter
20 40 03 30	input:: CLR or LATCH	-	-	-	-	-	-	-	-	x	x	x	x	x = 0 : CLR input x = 1 : LATCH input
20 40 03 40	impulse/event counter or pulse width/period duration measurement	-	-	-	-	-	-	-	-	x	x	x	x	x = 0 : external clock input x =1: internal reference clock (20MHz/5MHz)
20 40 03 50	4 edge evaluation/CLK + DIR or 20MHz/5MHz refeference clock	-	-	-	-	-	-	-	-	x	x	x	x	**Cnt_Mode**=0: x=0: 4-Fl.;x=1:CLK+DIR **Cnt_Mode**=1: x=0: 20MHz; x=1: 5MHz
20 40 04 54	DIO2 add-on bit 0...15	-	x	x	x	x	x	x	x	x	x	x	x	x: sets/clears output <bit n°>
20 40 04 64	DIO2 add-on bit 16...31	-	x	x	x	x	x	x	x	x	x	x	x	x: sets/clears output <bit n°+16>
20 40 04 74	DIO2 add-on set_bit 0...15	-	x	x	x	x	x	x	x	x	x	x	x	x = 0: no effect x = 1: set output <bit n°> *
20 40 04 84	DIO2 add-on set_bit 16...31	-	x	x	x	x	x	x	x	x	x	x	x	x = 0: no effect x = 1: set output <bit n°+16>*
20 40 04 94	DIO2 add-on reset_bit 0...15	-	x	x	x	x	x	x	x	x	x	x	x	x = 0: no effect x = 1: clear output <bit n°> *
20 40 04 A4	DIO2 add-on reset_bit 16...31	-	x	x	x	x	x	x	x	x	x	x	x	x = 0: no effect x = 1: clear output <bit n°+16> *
20 40 04 6C	configure inputs/outputs Conf_DIO() bit 0: DIO 00...07; bit 1: DIO 08...15 bit 2: DIO 16...23; bit 3: DIO 24...31	-	-	-	-	-	-	-	-	x	x	x	x	x = 0: configure channels as input x = 1: configure channels as output
* Function has no effect with inputs														

Fig. 37 – DIO2 hardware addresses of the control and data register

DIO2 / DIO3 Add-On

9.2.2 Operating mode impulse/event counting

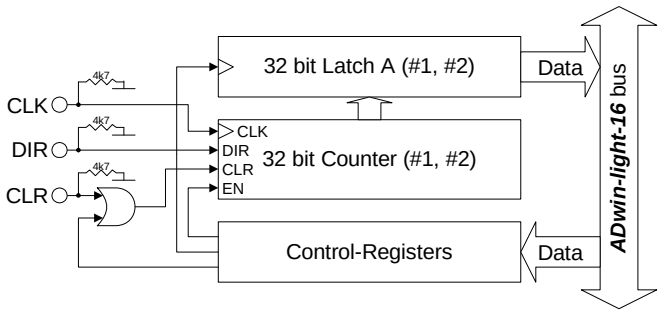
External square-wave signals at the inputs A/CLK and B/DIR clock the counters in this mode. With **Cnt_Set** you either activate the mode for determining the clock frequency and direction or the four edge evaluation.

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

– clear the counter (CLR) — **Clearing**

– latch the counter value into latch A (LATCH). — **Latching**

Clock and direction



Every positive edge of a square-wave signal at the clock input (CLK) is counted (incremented or decremented) up to a maximum frequency of 20 MHz. The direction is derived from a high signal (increment) or low signal (decrement) at the direction input (DIR); this signal can be a fixed voltage or a dynamic signal (e.g. given by an external logic).

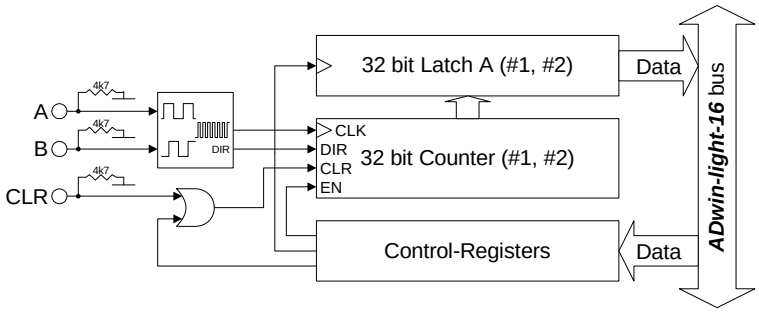
Programming example

```
INIT:                          initialize
. . .
CNT_ENABLE(0)                  disable counter
CNT_CLEAR(1)                   clear counter
CNT_MODE(0)                    external clock input at the CLK input
CNT_SET(1)                     set mode clock and direction
CNT_INPUTMODE(0)               set input CLR/LATCH to CLR mode
CNT_ENABLE(1)                  enable counter
. . .
```

```
EVENT:
CNT_LATCH(1)                   latch current counter value to latch A
CNT_READLATCH(1)               read latch A
. . .                          evaluation in the program
```

Four edge evaluation

This mode determines clock and direction of two signals, which are input at A and B off-phase by 90 degrees (ideally). The count direction is determined by the temporal sequence of the rising and falling edges of the two input signals.



Please note:

– The counter counts 4 edges in each cycle.

– The maximum count frequency is 20 MHz. Together with the 4 edges per cycle it will result in a maximum input frequency of 5 MHz (at A or B).

– The time lap 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 (to ≠90 degrees) will have an effect on the maximum input frequency because of the minimum time lap of the edges. If it differs from 90 degrees, the maximum input freqeuency of 5 MHz decreases for instance to 45 degrees at 2.5 MHz.

Programming example

`INIT:`	initialize
`. . .`	
`CNT_ENABLE(0)`	disable counter
`CNT_CLEAR(1)`	clear counter
`CNT_MODE(0)`	external clock input at CLK input
`CNT_SET(0)`	enable mode four edge evaluation
`CNT_INPUTMODE(0)`	set input CLR/LATCH to CLR mode
`CNT_ENABLE(1)`	enable counter
`. . .`	

`EVENT:`	
`CNT_LATCH(1)`	latch current counter value to latch A
`CNT_READLATCH(1)`	read latch A
`. . .`	evaluation in the program

DIO2 / DIO3 Add-On

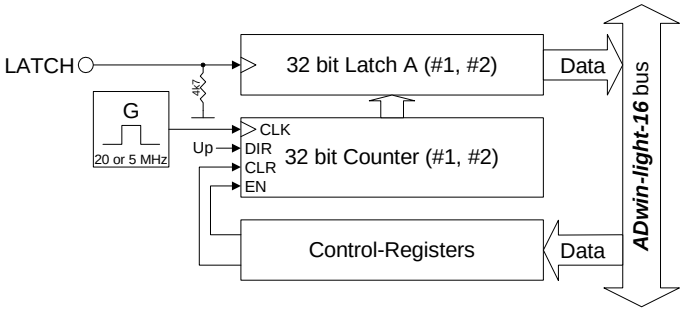
9.2.3 Operating mode pulse width and period duration measurement

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

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

Period duration measurement

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



Programming example

```
INIT:
. . .
CNT_ENABLE(0)
CNT_CLEAR(1)
CNT_MODE(1)
CNT_SET(0)
CNT_SET(1)
CNT_INPUTMODE(1)
CNT_ENABLE(1)
. . .
```

initialize

disable counter

clear counter

mode internal reference clock at internal CLK input

... with 20 MHz or

... with 5 MHz

set input CLR/LATCH to LATCH mode

enable counter

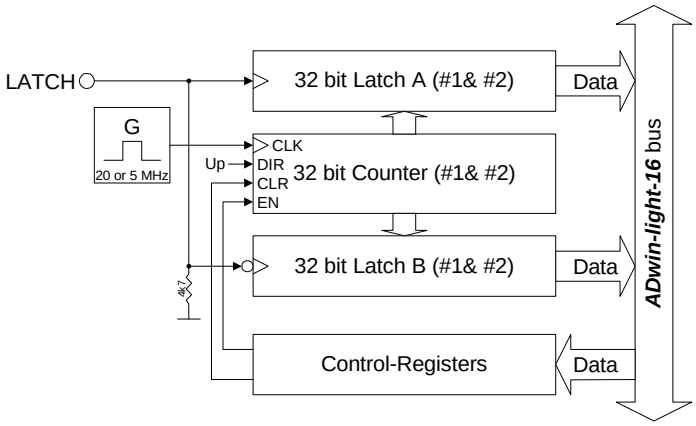
```
EVENT:
CNT_READLATCH(1)
. . .
```

read latch A

evaluation in the program

Pulse width and pause duration measurement

The counters have each a latch A for positive and a latch B for negative edges. Thus, pulse and pause duration can be evaluated separately by calculating the differences of the latches.



Programming example

```
INIT:
. . .
CNT_ENABLE(0)
CNT_CLEAR(1)
CNT_MODE(1)
CNT_SET(0)
CNT_SET(1)
CNT_INPUTMODE(1)
CNT_ENABLE(1)
. . .
```

initialize

disable counter

clear counter

mode internal reference clock at internal CLK input

... with 20 MHz or

... with 5 MHz

set input CLR/LATCH to LATCH mode

enable counter

```
EVENT:
CNT_READLATCH(1)
CNT_READFLATCH(1)
. . .
```

read latch A

read latch B

evaluation in the program

DIO2 / DIO3 Add-On

9.3 SSI Decoder

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

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

The connections of the decoder are on the connectors COUNTER (25-pin, DSUB), on the pins 8, 9, 21 and 22 (see Fig. 33 – Overview of the L16-EURO-DIO2 with pin assignments). The pins 7 and 20 provide DGND and +5V.

Setting properties

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

– Clock rates: With **SSI_Set_Clock** clock rates of approx. 40kHz up to 1MHz 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. 38 – Listing: Conversion of Gray code into binary code

Programming

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

Range	Instructions
Initialization	SSI_INIT
Receiving of data	SSI_READ
Set decoder resolution	SSI_Set_BITS
Set decoder clock rate	SSI_Set_CLOCK
Start reading encoder data	SSI_START
Return reading status	SSI_STATUS

The instructions are in the include file <ADWL16.INC>. More information can be found in the ***ADbasic*** manual and the online help.

10 PWM1 Add-On

With the PWM1 add-on you are additionally provided with:

– 1 PWM output, page 54.

  One digital output of the basic version is replaced by the PWM1 output.

– 1 SPI interface with SPI master functionality, page 55.

The PWM1 add-on can be combined with the Light-16 basic variant and any add-on (CO1, DIO1, DIO2, DIO3).

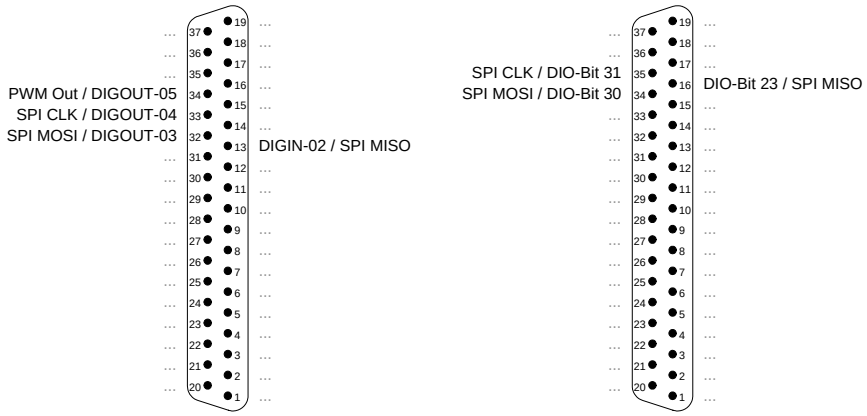
pin assignments

The PWM1 add-on is a retrofitable add-on which uses given pins. The function of the double-use pins is switched via software between original function and new function.

With the Light-16 basic variant (without add-ons) and the add-ons CO1 and DIO1 only the pins of the DSub connector `ADwin I/O-Connector` can be switched.

Alternatively, with the add-ons DIO2 and DIO3 pins of the DSub connector `Digital I/O` can be also switched.

The switchable pins in the pin assignments (below) are named: `PWM1`, `SPI CLK`, `SPI MOSI` and `SPI MISO`.



DSub `ADwin I/O-Connector` DSub connector `Digital I/O` (only with DIO2 and DIO3)

Fig. 39 – Pin assignments

10.1 PWM Output

The add-on PWM1 provides one PWM output. The PWM output enables to output a pulse-width-modulated signals with selectable duty cycle. The output is clocked with 40 MHz.

The PWM output is provided at pin 34 on the 37-pole DSub socket `ADwin I/O-Connector` (see above). Pin 34 has double-use and may be run as digital output `DIGOUT-05` or as PWM output `PWM Out`. You set the pin function via software using the instruction **`PWM_Activate`**.

After power-up pin 34 is configured as digital output `DIGOUT-05`.

Programming

The functionality of the PWM output is easily programmed with *ADbasic* instructions; description see chapter 13.7, starting from page 135:

Function	Instructions
Switch pin 34 to PWM output	**`PWM_Activate`**
Initialize PWM output	**`PWM_Init`**
Select operating mode	**`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 are provided in the include file `ADWL16.inc`. More information can be found in the online help.

PWM1 Add-On

10.2 SPI Interface

The add-on PWM1 provides an SPI interface with SPI master functionality. The SPI master runs with a bus frequency of up to 5 MHz.

The SPI interface uses 3 pins of the DSub connector ADwin I/O-Connector oder der Sub-D-Buchse Digital I/O for SPI signals (see table). These pins have double-use and may be run as digital channels or as SPI signals. You set the pin function via software using the instruction **SPI_Enable**. The pin assignment is shown on page 53.

SPI Signal	ADwin I/O-Connector		Digital I/O	
	pin	replaces	pin	replaces
SPI CLK: clock	33	DIGOUT-04	35	DIO-31
SPI MOSI: Master out, slave in	32	DIGOUT-03	34	DIO-30
SPI MISO: Master in, slave out	13	DIGIN-02	16	DIO-23
Slave Select (SS)	e.g. digital output			

Please note: If you switch pins of the DSub connector Digital I/O for SPI signals, the pins DIO-24…DIO-31 are automatically configured as outputs and the pins DIO16…DIO-23 as inputs. If later you reset the pins as digital channels the previous configuration is valid again.

In addition, you require a separate Slave Select line for each SPI slave to address them. If you use the remaining digital outputs, you set the selected TTL level with the appropriate instructions for digital outputs, e.g. **Digout_Clear** or **Digout_Set**.

SPI protocol

In theory, an unlimited number of members can be connected to the SPI bus while there has to be exactly one SPI master. The master creates the clock signal on the SPI CLK line and selects via a Slave Select (SS) line the slave he will communicate with. If the master pulls SS to the appropriate TTL level, the slave is activated, listens to SPI MOSI and sends its data to SPI MISO with the clock rate of SPI CLK. Thus, a number of bits are transferred from master to slave and the same number of bits from slave to master.

A protocol for data transfer has not been fixed, but practically four modes have been established. The modes are are selected using the parameters clock polarity (CPOL) and clock phase (CPHA):

Mode	CPOL	CPHA
0	0	0
1	0	1
2	1	0
3	1	1

– Clock polarity: At CPOL=0 the base value of the clock is low, at CPOL=1 the base value is high.

– Clock Phase: CPHA selects at which edge data is to be sampled. That is, CPHA=0 means sample on the leading (first) clock edge, while CPHA=1 means sample on the trailing (second) clock edge.

– Modes 0…3: Thus with CPOL=0 and CPHA=0 data is captured on the clock's rising edge. With CPHA=1 data is captured on the (second) falling

edge. With CPOL=1 all is reversed, so with CPHA=0 data is captured on the falling edge, with CPHA=1 on the rising edge

Please note that with CPHA=0 the slave sends its data to SPI MISO while SS is activated, so the master can capture it on the first edge. With CPHA=1 the slave sends its data to SPI MISO on the first edge, so the master can capture it on the second edge.

With each clock a bit is transferred. A common data byte requires 8 clock periods to be completely transferred. You can also transfer several bytes in a row where there is no definition whether a short SS deselect signal is required after each byte. The data transfer is finished if the SS deselect signal is permanent.

Programmierung

The functionality of the SPI interface is easily programmed by using *ADbasic* instructions:

Function	Instructions
Enable SPI slave for data transfer (slave select)	for example **Digout_Clear**, **Digout_Set**
Switch pins for SPI signals	**SPI_Enable**
Configure SPI interface	**SPI_Config**
Write and read data	SPI_Set_MOSI, **SPI_Get_MISO**
Start data transfer	**SPI_Start**
Wait for end of data transfer	**SPI_Wait**
Return status of data transfer	**SPI_Status**
Read TTL level of the data line	**SPI_Static_MISO**

The instructions are provided in the include file ADWL16.inc. More information can be found in chapter 13.8 on page 146.

ADwin-light-16-Boot

11 *ADwin-light-16*-Boot

This option is only available in conjunction with the Ethernet interface, that means with the *L16-EXT-ENET* or the *L16-EURO-ENET*.

After power-up ADwin-light-16-Boot automatically starts an application which has been programmed before. Thus after initializing the application an operation without PC is possible.

ADwin-light-16-Boot executes the following steps after power-up:

– Loading the operating system

– Loading the processes generated with the ***ADbasic*** compiler (max. 10)

– Automatic starting of the process n° 10. Here you have also to program the start of all other processes

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

– Boot the system after power-up and the programmed processes are disabled

– After switching off and powering up anew, the bootloader option is enabled again

By programming the Flash-EEPROMs without processes the system will only be booted after restart with the file <ADwin9.btl>. A process will not be executed.

With the installation of the ***ADwin***-Developer software from the ***ADwin-***CD-ROM, the program for the bootloader option (ADethflash) is automatically copied. The version of the CD-ROM should be 3.00.2735 or higher.

Use the program ADethflash for an ***ADwin*** system with an Ethernet interface.

At standard installation you will find the program in the directory <C:\ADwin\Tools\Ethernet Interface\...>.

Notes for the bootloader with Ethernet interface can be found in the documentation "***ADwin*** Driver Installation".

Together with with an Ethernet-Interface and bootloader you can save and read up to 2,000 Long or Float values with 32 bit via ***ADbasic*** process in the built-in Flash-EEPROM memory. You can find a more detailed description in the program <ADethflash.exe>, if you click on the button "Info about eeprom support".

12 Accessories

The following accessories are available for the ***ADwin-light-16*** system:

– ***ADbasic***: The real-time development environment for prgramming all of the ***ADwin*** systems.

  ADbasic is required to develop processes for ***ADwin*** systems, which are loaded and controlled from a development environment (e.g. C#, Visual Basic, Matlab and others).

– ***ADwin-light-16***-pow: external 12V power supply unit.

  The power supply unit provides 12 Volt on the secondary side at a maximum continuous load of 2 Ampere. The power supply is rated for the highest load and maximum expansions.

– Various lengths of power supply and USB or Ethernet cables.

  Please pay attention to a sufficient shielding of the USB and Ethernet cable, in order to avoid interferences in the data lines. Interferences have to be conducted before the chassis via ground (see also chapter 3).

– Cable connector for an external power supply

  The cable connector be used if an external power supply is used.

– installation kit for enclosures

13 Software

You are programming the ***ADwin-light-16*** - all expansions included - with simple ***ADbasic*** instructions.

All instructions to access inputs, outputs and interfaces are described on the following pages (or in the online help):

13.1 Example Program

13.1.1 CAN: Cyclic Read and Send of Messages

This programs describes the initialization of the CAN controller in the section **INIT:** and the cyclic read and send of messages in the section **EVENT:**

```
REM The program initializes the CAN controller,
REM configures one message object as sender
REM and one as receiver. The program exchanges all 10 ms data
REM between CAN controller and transputer.

#include adwl16.inc

DIM result AS LONG

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

 EN_RECEIVE(2,385,0) 'configure message object 2 for reading
                             'with 11 bit identifier 385
 EN_TRANSMIT(3,1,0) 'configure message object 3 for writing
                             'with 11 bit ident. 1

EVENT:
 REM read 1 data set and write 1 data set

 result = READ_MSG(2)'read data
 'If there are new data, they are written into the field
 'CAN_MSG.


 can_msg[1]=1                'data, which are to be sent,
 can_msg[2]=2                'are written into the field
 can_msg[3]=3                'CAN_MSG. You are getting the
 can_msg[4]=4                'data from this field to be
 can_msg[5]=5                'sent later
 can_msg[6]=6
 can_msg[7]=7
 can_msg[8]=8
 can_msg[9]=8                '8 data bytes

  TRANSMIT(3)                'send message in message object 3
```

13.1.2 CAN: Interrupt-Controlled Reading

The following program shows the initialization of the CAN controller and the interrupt-controlled reading of new messages:

```
REM The program initializes the CAN controller and
REM configures a message object as receiver.
REM The program reads interrupt-controlled messages
REM as soon as a new message arrives.

#INCLUDE adwl16.inc

DIM result,status,object AS LONG

INIT:
  INIT_CAN()                    'Initialize the CAN controller
                                'set Baud rate to 125 kBit/s
  SET_CAN_BAUDRATE(125000)

  EN_RECEIVE(1,385,0)           'message object 1 is configured
                                'for reading. Only messages with
                                '11 bit identifier 385 are saved.

  status = GET_CAN_REG(1)       'read status
  EN_INTERRUPT(1)               'When a message arrives in
                                'message object 1 an interrupt
                                'is triggered.

EVENT:
  object = GET_CAN_REG(5Fh)     'read interrupt register

  IF (objekt = 2) THEN          'Get the number of the message
    objekt = 15                 'object, where the new message
  ELSE                          'can be found
    object = object - 2
  ENDIF

  result = READ_MSG(object)     'read out new data

  REM The data are available in the field CAN_MSG.
```

13.2 Analog Inputs and Outputs

This section describes the following instructions:

DAC

DAC outputs a defined voltage on a specified analog output.

Syntax

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

Parameters

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

Notes

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

See also

ADC

Valid for

L16

Example

```
REM Digital proportional controller
Dim set_to, gain, diff, Out As Long 'Declaration

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

ADC

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

Syntax

```
ret_val = ADC(channel)
```

Parameters

channel	Number (1, 3, 5, …, 15) of analog input.	LONG
ret_val	Measurement value in digits (0…65,535).	LONG

Notes

ADC is a combination of consecutive functions:

- **Set_Mux**: Set the multiplexer to the specified input channel.
- Wait for settling of the multiplexer.
- **Start_Conv**: Start measurement: Convert analog signal to a digital value.
- **Wait_EOC**: Wait for end of conversion.
- **ReadADC**: 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.

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

In the following examples the instructions **Set_Mux**, **Start_Conv**, **Wait_EOC** and **ReadADC** should be used instead of **ADC** in the following cases:

- Very short cycle times: **Processdelay** < 240 (s.a.).
- High internal resistance (>3kΩ) of the voltage source of the measurement signal: This increases the settling time of multiplexer.
- You want to use inevitable waiting times for additional program tasks.

The measurement range depends on the gain factor:

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

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

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

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

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

See also

ReadADC, Set_Mux, Start_Conv, Wait_EOC, L16_Mode

Valid for

L16

Example

```
Dim iw As Long                  'Declaration

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

L16_Mode

L16_Mode sets the operating mode of *ADwin-light-16* Rev. B.

Syntax

```
#Include ADWL16.Inc
L16_Mode(mode)
```

Parameters

`mode` Bit pattern to set the operating mode. LONG

Bits in `mode`	Meaning
Bit 0:	Bit = 0: Standard operation (default). Bit = 1: Fast operation.
Bits 1…31:	Reserved

Notes

In standard mode the device runs fully compatible to revision A. After power-up the device is always set to standard mode.

In fast mode the A/D converter runs with maximum sampling rate of 500 kHz.

See also

ADC, ReadADC, Set_Mux, Start_Conv, Wait_EOC

Valid for

L16 Rev. B

Example

```
#Include ADWL16.Inc

Init:
  REM activate fast mode
  L16_Mode(1)
```

ReadADC

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

Syntax

```
ret_val = ReadADC(1)
```

Parameters

1	The number of the A/D converter to read.	LONG
ret_val	Measurement value in digits which corresponds to the voltage at the converter's input.	LONG

Notes

- / -

See also

ADC, Set_Mux, Start_Conv, Wait_EOC, L16_Mode

Valid for

L16

Example

```
Event:
  'Set multiplexer to channel 3
 Set_Mux(001b)
 Rem wait for MUX settling time
 Rem …
 Start_Conv(1)                'Start ADC conversion
 Wait_EOC(1)                  'Wait for end of conversion
 Par_1 = ReadADC(1)           'Read value of ADC1
```

Seq_Init

Seq_Init initializes the sequential control.

These settings are done: Operating mode, gain factor, channel selection and muliplexer settling time.

Syntax

```
#Include ADWL16.Inc
Seq_Init(mode, gain, channels, muxtime)
```

Parameters

`mode` — LONG

Operating mode of the sequential control:
0: Standard mode (default), single conversion.
1: Mode "single shot", single conversion cycle.
2: Mode "continuous", continuous conversion.
3: Mode "continuous max" using max. speed.

`gain` — LONG

Gain factor (Modes 1 … 3 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.

`channels` — LONG

Bit pattern to select the channels for conversion.
Bit = 0: No conversion.
Bit = 1: Do conversion.

Bit no.	31:15	14	13	12	…	3	2	1	0
Channel no.	–	15	–	11	…	–	3	–	1

`muxtime` — LONG

Number of time units, which sets the settling time of the sequential control:
0: Standard waiting time (200 ≙ 5µs).
200…2^31: Waiting time in units of 25ns.

Notes

After power-up mode 0 is active.

Modes 1 … 3 activate the sequential control, which converts several channels consecutively; according to the mode the conversion cycle is done once or cyclic. The sequential control is always related to those channels being selected by `channels`.

The modes differ in the following items:

Mode		Kind of conversion
0	Standard:	Single conversion of one channel, see **ADC**.
1	Single shot:	The sequential control is started by **Start_Conv**; it ends as soon as each of the selected channels is converted once. The end of the sequential control is queried with **Wait_EOC** and measurement values are read with **Seq_Read**.

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

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

The multiplexer settling time (parameter `muxtime`) sets the time between 2 conversions of the sequential control. We recommend, not to underrun the given range of values, because a shorter settling time leads to more imprecise or even wrong measurement values.

If the internal resistance of the voltage source of the measurement signal is too high, the predefined settling time of the multiplexer will not be sufficient for an exact measurement. You can then raise the multiplexer settling time with a higher value of the parameter `muxtime`.

See also

ADC, Seq_Read, Start_Conv, Wait_EOC

Valid for

L16 Rev. B

Example

```
#Include ADWL16.Inc

Dim Data_1[8] As Long At DM_Local
Dim i As Long

Init:
  REM Sequential control: Continuous Mode, gain 2
  REM channels 1, 3, ..., 15, standard settling time
  Seq_Init(3,1,5555h,0)
  Start_Conv(1)              'Start conversion cycle

Event:
  REM The conversion of all selected channels has just
  REM ended, so measurement values are read.
  For i = 1 To 8
    Data_1[i] = Seq_Read(i*2-1) 'read values
  Next i
  REM process values
```

Seq_Read

Seq_Read returns the last saved measurement value of the selected channel.

Syntax

```
#Include ADWL16.Inc
ret_val = Seq_Read(channel)
```

Parameters

channel	Channel no. (1, 3, …, 15).	LONG
ret_val	Measurement value (0…65535) of selected channel.	LONG

Notes

You can only reasonably use this instruction if the sequential control of the module has been activated before with **Seq_Init** and if the given channel has been selected, too.

In "single shot" mode the end of conversion must be queried with **Wait_EOC**, before reading the measurement values.

See also

Seq_Init, Start_Conv, Wait_EOC

Valid for

L16 Rev. B

Example

```
#Include ADWL16.Inc

Dim Data_1[400] As Long At DM_Local

Init:
  REM sequential control: Single shot, gain 1
  REM channels 5, 7, 13, 15, standard settling time
  Seq_Init(1,0,1010000010100000b,0)
  Start_Conv(1)              'start conversion cycle

Event:
  Wait_EOC(1)                'wait for end of conversion
  REM read channels 5, 7, 13, 15
  Data_1[1] = Seq_Read(5)
  Data_1[2] = Seq_Read(7)
  Data_1[3] = Seq_Read(13)
  Data_1[4] = Seq_Read(15)
  Start_Conv(1)              'start next conversion cycle
```

Set_Mux

Set_Mux sets the A/D input multiplexer to the selected channel.

Syntax

```
Set_Mux(pattern)
```

Parameters

pattern	Bit pattern for the allocation of measurement channels and gain.	LONG
MUX	The bits 0…2 determine the channel to which the multiplexer is set: **000**: Channel 1 **001**: Channel 3 **010**: Channel 5 **011**: Channel 7 **100**: Channel 9 **101**: Channel 11 **110**: Channel 13 **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, ReadADC, Start_Conv, Wait_EOC, L16_Mode

Valid for

L16

Example

```
Dim val As Long

Event:
  Set_Mux(0)                  'Set multiplexer to channel 1
  Rem Wait here for the settling time of the multiplexer
  Rem by inserting some instructions.
  Start_Conv(1)               'Start AD-conversion ADC1
  Wait_EOC(1)                 'Wait for end of conversion of
                              'ADC
  val = ReadADC(1)            'Read value of ADC1
```

Start_Conv

START_CONV can start the conversion of the A/D converter and of all D/A converters.

Syntax

```
Start_Conv(pattern)
```

Parameters

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

Bit no.	31…3	2	1	0
ADC1, 16-bit	–	–	–	x
all DACs	–	x	–	–

Notes

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

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

See also

ADC, ReadADC, Set_Mux, Wait_EOC, L16_Mode

Valid for

L16

Example

```
Dim val1 As Long

Event:
  Set_Mux(0)                  'Set multiplexer to channel 1
  Rem Bypass the settling time with command lines
  Start_Conv(1)               'Start ADC1 A/D-conversion
  Wait_EOC(1)                 'Wait for end of conversion
  val1 = ReadADC(1)           'Read out value
```

Multiplexer settling time is given on page 17.

Wait_EOC

WAIT_EOC waits for the end of the A/D conversion.

Syntax

```
Wait_EOC(1)
```

Parameters

Only the constant 1 is allowed as passed parameter. The parameter is interpreted as bit pattern to specify the converter.

CONST
LONG

Notes

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

See also

ADC, ReadADC, Set_Mux, Start_Conv, L16_Mode

Valid for

L16

Example

```
Dim val1 As Long

Event:
  Set_Mux(0)                    'Set MUX to channel 1
  Rem Bypass the settling time of the multiplexer with
  Rem command lines
  Start_Conv(1)                 'Start A/D-conversion ADC1
  Wait_EOC(1)                   'Wait for end of conversion
  val1 = ReadADC(1)             'Read out value
```

Multiplexer settling time is given on page 17.

13.3 Digital Inputs and Outputs

This section describes the following instructions:

Clear_Digout

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

Syntax

```
Clear_Digout(bit_no)
```

Parameters

bit_no — Bit number (0…5) which specifies the output (see table). CONST, LONG

bit_no	0	1	…	5
Output	0	1	…	5

Notes

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

See also

Digout_Word, Set_Digout

Valid for

L16, L16-CO1, L16-DIO1, L16-DIO2, L16-DIO3

Example

```
Dim val As Long                 'Declaration

Init:
  Set_Digout(0)                 'Set digital output 0 to level
                                'high

Event:
  val = ADC(1)                  'Measurement data acquisition
  If (val > 3000) Then
    Clear_Digout(0)             'Set dig. output 0 to level low
  EndIf
```

Digin

DIGIN returns the value of one of the digital inputs 0…5.

Syntax

```
ret_val = Digin(channel_no)
```

Parameters

`channel_no` Number which specifies the input to be queried: LONG, CONST

`ret_val` 1:TTL-level high. LONG
0: TTL-level low.

channel_no	0	…	5
Eingang Nr.	0	…	5

Notes

Digin accepts only constants as parameter.

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

See also

Digin_Word, Digout_Word

Valid for

L16

Example

```
Dim Data_1[10000] As Long As FIFO

Event:
  Rem Is digital input 0 set?
  If (Digin(0) = 1) Then
    Data_1 = ADC(1)             'Measurement data acquisition
  EndIf
```

Digin_Word

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

Syntax

```
ret_val = Digin_Word()
```

Parameters

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

Bit number in ret_val	31 …6	5	…	0
Input No.	–	5	…	0

Notes

- / -

See also

Digin, Digout_Word

Valid for

L16

Example

```
Dim Data_1[10000] As Long As FIFO

Event:
  Rem Query if inputs 0 and 1 are set
  If ((Digin_Word() And 11b) = 11b) Then
    Data_1 = ADC(1)            'Measurement data acquisition
  EndIf
```

Digout_Word

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

Syntax

```
Digout_Word(pattern)
```

Parameters

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 number in pattern	31 …6	5	…	0
Output No.	–	5	…	0

Notes

- / -

See also

Clear_Digout, Digin_Word, Set_Digout

Valid for

L16

Example

```
Dim value As Long

Event:
  value = ADC(1)                'Measurement data acquisition
  If (value > 3000) Then'Is the limit value exceeded?
    Digout_Word(101b)           'Set outputs 0 and 2,
                                'clear all other outputs
  EndIf
```

Set_Digout

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

Syntax

```
Set_Digout(bit_no)
```

Parameters

bit_no — Bit number (0…5) which specifies the output (see table). CONST, LONG

bit_no	0	1	…	5
Output	0	1	…	5

Notes

Set_Digout accepts only a constant as parameter bit_no.

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

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

See also

Clear_Digout, Digout_Word

Valid for

L16

Example

```
Dim val As Long

Event:
  val = ADC(1)                  'Measurement data acquisition
  If (val > 3000) Then
    Set_Digout(0)               'Set digital output 0 to level
                                'high
  EndIf
```

Conf_DIO_E

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

Syntax

```
#Include ADWL16.Inc
Conf_DIO_E(pattern)
```

Parameters

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

Bit no. in pattern	15...4	3	2	1	0
Channels	–	DIO31 … DIO24	DIO23 … DIO16	DIO15 … DIO08	DIO07 … DIO00

Notes

After power-up all digital I/O-lines are configured as inputs and cannot be accessed as outputs. Channels can be configured in groups of 8 as inputs or 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

Digin_Word1_E, Digin_Word2_E, Digout_Reset1_E, Digout_Reset2_E, Digout_Set1_E, Digout_Set2_E, Digout_Word1_E, Digout_Word2_E

Valid for

L16-DIO1, L16-DIO2, L16-DIO3

Example

```
#Include ADWL16.Inc

Init:
  Conf_DIO_E(1100b)          'Configures DIOs 15:00 as inputs
                             'and
                             'DIOs 31:16 as outputs.
```

Digin_Word1_E

DIGIN_WORD1_E returns the values of the digital inputs 0...15 at the same time.

Syntax

```
#Include ADWL16.Inc
ret_val = Digin_Word1_E()
```

Parameters

`ret_val` Bit pattern, that corresponds to the TTL-level at the digital inputs. LONG
1: TTL-level high.
0: TTL-level low.

Bit number in ret_val	31 … 16	15	14	…	1	0
Input No.	–	DIO15	DIO14	…	DIO01	DIO00

Notes

If you have configured the channels as outputs, the contents of the output register of these bits is returned.

See also

Conf_DIO_E, Digin_Word2_E, Digout_Reset1_E, Digout_Reset2_E, Digout_Set1_E, Digout_Set2_E, Digout_Word1_E, Digout_Word2_E

Valid for

L16-DIO1, L16-DIO2, L16-DIO3

Example

```
#Include ADWL16.Inc

Init:
  Conf_DIO_E(1100b)             'Configures DIOs 15:00 as inputs
                                'and
                                'DIOs 31:16 as outputs

Event:
  Par_1 = Digin_Word1_E()'Read low-word (bits 15:00)
```

Digin_Word2_E

DIGIN_WORD2_E returns the values of the digital inputs 16...31 at the same time.

Syntax

```
#Include ADWL16.Inc
ret_val = Digin_Word2_E()
```

Parameters

`ret_val` Bit pattern, that corresponds to the TTL-level at the digital inputs. LONG
1: TTL-level high.
0: TTL-level low.

Bit number in ret_val	31 … 16	15	14	…	1	0
Input No.	–	DIO31	DIO30	…	DIO17	DIO16

Notes

If you have configured the channels as outputs, the contents of the output register of these bits is returned.

See also

Conf_DIO_E, Digin_Word1_E, Digout_Reset1_E, Digout_Reset2_E, Digout_Set1_E, Digout_Set2_E, Digout_Word1_E, Digout_Word2_E

Valid for

L16-DIO1, L16-DIO2, L16-DIO3

Example

```
#Include ADWL16.Inc

Init:
  Conf_DIO_E(0)                 'Configure DIOs 31:00 as inputs
                                'like in power-up status

Event:
  Par_1 = Digin_Word1_E()'Read low-word (bits 15:00)
  Par_2 = Digin_Word2_E()'Read high-word (bits 31:16)
```

Digin_Long_E

Digin_Long_E returns the values of the digital inputs 0...31 at the same time.

Syntax

```
#Include ADWL16.Inc
ret_val = Digin_Long_E()
```

Parameters

`ret_val` Bit pattern, that corresponds to the TTL-level at LONG the digital inputs.
1: TTL-level high.
0: TTL-level low.

Bit number in ret_val	31	30	...	1	0
Input No.	DIO31	DIO30	...	DIO01	DIO00

Notes

If you have configured the channels as outputs, the contents of the output register of these bits is returned.

See also

Conf_DIO_E, Digin_Word1_E, Digin_Word2_E, Digout_Reset1_E, Digout_Reset2_E, Digout_Set1_E, Digout_Set2_E, Digout_Word1_E, Digout_Word2_E

Valid for

L16-DIO1, L16-DIO2, L16-DIO3

Example

```
#Include ADWL16.Inc

Init:
  Conf_DIO_E(0)                 'Configure DIOs 31:00 as inputs
                                'like in power-up status

Event:
  Par_1 = Digin_Word1_E()'Read low-word (bits 15:00)
  Par_2 = Digin_Long_E()'Read high-word (bits 31:16)
```

Digout_Reset1_E

DIGOUT_RESET1_E sets the selected digital outputs 0...15 to TTL-level low.

Syntax

```
#Include ADWL16.Inc
Digout_Reset1_E(clear)
```

Parameters

clear	Bit pattern for setting specified outputs: Bit = 1: Set to TTL-level low. Bit = 0: no influence.	LONG

Bit number in clear	31 … 16	15	14	…	1	0
Output No.	–	DIO15	DIO14	…	DIO01	DIO00

Notes

- / -

See also

Conf_DIO_E, Digin_Word1_E, Digin_Word2_E, Digout_Reset2_E, Digout_Set1_E, Digout_Set2_E, Digout_Word1_E, Digout_Word2_E

Valid for

L16-DIO1, L16-DIO2, L16-DIO3

Example

```
#Include ADWL16.Inc

Init:
  Conf_DIO_E(0011b)             'Configures DIOs 15:00 as
outputs
                                'and
                                'DIOs 31:16 as inputs

Init:
  Par_1 = 5555h                 'Delete all odd-numbered bits of
                                'the
                                'low-word upon output.
  Digout_Word1_E(0FFFFh)'Output DIO-bits 15:00

Event:
  Digout_Reset1_E(Par_1)'Delete DIO-bits equivalent to Par_1
  Par_1 = Par_1 XOr 0FFFFh'Invert output-word
  Digout_Word1_E(Par_1)'Output DIO-bits 15:00
```

Digout_Reset2_E

DIGOUT_RESET2_E sets the selected digital outputs 16…31 to TTL-level low.

Syntax

```
#Include ADWL16.Inc
Digout_Reset2_E(clear)
```

Parameters

clear	Bit pattern for setting specified outputs. Bit = 1: Set to TTL-level low. Bit = 0: no influence.	LONG

Bit number in clear	31 … 16	15	14	…	1	0
Output No.	–	DIO31	DIO30	…	DIO17	DIO16

Notes

- / -

See also

Conf_DIO_E, Digin_Word1_E, Digin_Word2_E, Digout_Reset1_E, Digout_Set1_E, Digout_Set2_E, Digout_Word1_E, Digout_Word2_E

Valid for

L16-DIO1, L16-DIO2, L16-DIO3

Example

```
#Include ADWL16.Inc

Init:
  Conf_DIO_E(1100b)             'Configure DIOs 15:00 as inputs
                                'and
                                'DIOs 31:16 as outputs

Init:
  Par_2 = 5555h                 'Clear all odd-numbered bits of
                                'the
                                'high-word during output.
  Digout_Word2_E(0FFFFh)'Output DIO bits 31:16

Event:
  Digout_Reset2_E(Par_2)'Clear DIO bits according to Par_2
  Par_2 = Par_2 XOr 0FFFFh'Invert output-word
  Digout_Word2_E(Par_2)         'Output DIO bits 31:16
```

Digout_Set1_E

DIGOUT_SET1_E sets the selected digital ouptuts 0...15 to TTL-level high.

Syntax

```
#Include ADWL16.Inc
Digout_Set1_E(set)
```

Parameters

set	Bit pattern to set specified outputs: Bit = 1: Set to TTL-level high. Bit = 0: No change.	LONG

Bit number in set	31 … 16	15	14	…	1	0
Output No.	–	DIO15	DIO14	…	DIO01	DIO00

Notes

- / -

See also

Conf_DIO_E, Digin_Word1_E, Digin_Word2_E, Digout_Reset1_E, Digout_Reset2_E, Digout_Set2_E, Digout_Word1_E, Digout_Word2_E

Valid for

L16-DIO1, L16-DIO2, L16-DIO3

Example

```
#Include ADWL16.Inc

Init:
  Conf_DIO_E(0011b)           'Configures DIOs 15:00 as
outputs
                              'and
                              'DIOs 31:16 as input
  Par_1 = 0AAAAh              'Set all even-numbered bits of
                              'the
                              'low-word during the output
  Digout_Word1_E(0)           'Output DIO bits 15:00

Event:
  Digout_Set1_E(Par_1)        'Set DIO bits according to Par_1
  Par_1 = Par_1 XOr 0FFFFh'Invert output-word
  Digout_Word1_E(Par_1)'Output DIO bits 15:00
```

Digout_Set2_E

DIGOUT_SET2_E sets the selected digital outputs 16...31 to TTL-level high.

Syntax

```
#Include ADWL16.Inc
Digout_Set2_E(set)
```

Parameters

set	Bit pattern to set specified outputs. Bit = 1: Set to TTL-level high. Bit = 0: No change.	LONG

Bit number in set	31 … 16	15	14	…	1	0
Output No.	–	DIO31	DIO30	…	DIO17	DIO16

Notes

- / -

See also

Conf_DIO_E, Digin_Word1_E, Digin_Word2_E, Digout_Reset1_E, Digout_Reset2_E, Digout_Set1_E, Digout_Word1_E, Digout_Word2_E

Valid for

L16-DIO1, L16-DIO2, L16-DIO3

Example

```
#Include ADWL16.Inc

Init:
  Conf_DIO_E(1100b)           'Configure DIOs 15:00 as inputs
                              'and
                              'DIOs 31:16 as outputs
  Par_1 = 0AAAAh              'Set even-numbered bits of the
                              'low-word during output
  Digout_Word2_E(0)           'Set DIO bits 31:16 to level low

Event:
  Digout_Set2_E(Par_2)        'Set DIO bits 31:16 according to
                              'Par_2
  Par_2 = Par_2 XOr 0FFFFh 'Invert output-word
  Digout_Word2_E(Par_2)       'Output DIO bits 31:16
```

Digout_Word1_E

DIGOUT_WORD1_E sets all digital outputs 0...15 to specified TTL-levels using a bit pattern.

Syntax

```
#Include ADWL16.Inc
Digout_Word1_E(pattern)
```

Parameters

`pattern` Bit pattern, corresponding to the TTL level desired at the digital outputs. LONG
Bit = 1: Set to TTL-level high.
Bit = 0: Set to TTL-level low.

Bit number in `pattern`	31 … 16	15	14	…	1	0
Output No.	–	DIO15	DIO14	…	DIO01	DIO00

Notes

- / -

See also

Conf_DIO_E, Digin_Word1_E, Digin_Word2_E, Digout_Reset1_E, Digout_Reset2_E, Digout_Set1_E, Digout_Set2_E, Digout_Word2_E

Valid for

L16-DIO1, L16-DIO2, L16-DIO3

Example

```
#Include ADWL16.Inc

Init:
  Conf_DIO_E(0011b)             'Conigures DIOs 15:00 as outputs
                                'and
                                'DIOs 31:16 as inputs
  Par_1 = 5555h                 'Set all odd-numbered bits of the
                                'low-word

Event:
  Digout_Word1_E(Par_1)         'Output DIO bits 15:00
```

Digout_Word2_E

DIGOUT_WORD2_E sets all the digital outputs 16…31 to specified TTL-levels using a bit pattern.

Syntax

```
#Include ADWL16.Inc
Digout_Word2_E(pattern)
```

Parameters

`pattern` Bit pattern, corresponding to the TTL level desired at the digital outputs. LONG
Bit = 1:Set to TTL-level high.
Bit = 0: Set to TTL-level low.

Bit number in `pattern`	31 … 16	15	14	…	1	0
Output No.	–	DIO31	DIO30	…	DIO17	DIO16

Notes

- / -

See also

Conf_DIO_E, Digin_Word1_E, Digin_Word2_E, Digout_Reset1_E, Digout_Reset2_E, Digout_Set1_E, Digout_Set2_E, Digout_Word1_E

Valid for

L16-DIO1, L16-DIO2, L16-DIO3

Example

```
#Include ADWL16.Inc

Init:
  Conf_DIO_E(12)              'Configures DIOs 15:00 as inputs
                              'and
                              'DIOs 31:16 as outputs
  Par_2 = 0AAAAh              'Set all even-numbered bits of
                              'the
                              'low-word.

Event:
  Digout_Word2_E(Par_2)       'Output DIO bits 31:16
```

Digout_Long_E

Digout_Long_E sets all the digital outputs 0…31 to specified TTL-levels using a bit pattern.

Syntax

```
#Include ADWL16.Inc
Digout_Long_E(pattern)
```

Parameters

pattern	Bit pattern, corresponding to the TTL level desired at the digital outputs. Bit = 1:Set to TTL-level high. Bit = 0: Set to TTL-level low.	LONG

Bit number in pattern	31	30	…	1	0
Output No.	DIO31	DIO30	…	DIO01	DIO00

Notes

- / -

See also

Conf_DIO_E, Digin_Word1_E, Digin_Word2_E, Digin_Long_E, Digout_Reset1_E, Digout_Reset2_E, Digout_Set1_E, Digout_Set2_E, Digout_Word1_E, Digout_Word2_E

Valid for

L16-DIO3, L16-DIO2, L16-DIO3

Example

```
#Include ADWL16.Inc

Init:
  Conf_DIO_E(12)              'Configures DIOs 15:00 as inputs
                              'and
                              'DIOs 31:16 as outputs
  Par_2 = 0AAAAh              'Set all even-numbered bits of
                              'the
                              'low-word.

Event:
  Digout_Long_E(Par_2)        'Output DIO bits 31:16
```

13.4 Counter

This section describes the following instructions:

Cnt_Clear

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

Syntax

```
#Include ADWL16.Inc
Cnt_Clear(pattern)
```

Parameters

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

Bit no.	31…2	1	0
Counter no.	–	2[a]	1

a. not available with add-on CO1

Notes

After **Cnt_Clear** has been executed the bit pattern is automatically reset to 0 (zero), so the counters start counting from 0.

See also

Cnt_ClearEnable, Cnt_Enable, Cnt_GetStatus, Cnt_InputMode, Cnt_Latch, Cnt_Mode, Cnt_Read, Cnt_ReadLatch, Cnt_ReadFLatch, Cnt_Set

Valid for

L16, L16-CO1, L16-DIO1, L16-DIO2

Example

```
#Include ADWL16.Inc

Dim old_1, new_1 As Long'Dimension
Dim old_2, new_2 As Long' variables

Init:
  old_1 = 0                     'Initialize
  old_2 = 0                     ' variables
  Cnt_Mode(0)                   'All counters on external clock
                                'input
  Cnt_Set(11b)                  'counters 1+2 with clock (CLK)
                                'and
                                'direction (DIR) input
  Cnt_InputMode(0)              'Determine functionality
                                'CLR/LATCH: All as CLR
  Cnt_ClearEnable(11b)          'Enables the CLR function of
                                'counters 1+2
  Cnt_Clear(11b)                'Reset counters 1+2 to 0
  Cnt_Enable(11b)               'Start counters 1+2

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

Cnt_ClearEnable

Cnt_ClearENABLE disables or enables the CLR input of one or more counters according to the bit pattern in pattern.

Syntax

```
#Include ADWL16.Inc        '
Cnt_ClearEnable(pattern)
```

Parameters

pattern	Bit pattern. Bit = 0: disable CLR input at the counter. Bit = 1: enable CLR input at the counter.	LONG

Bit no.	31…2	1	0
Counter no.	–	2	1

Notes

This instruction affects all counters at the same time. It only works if the CLR mode is set by **Cnt_InputMode**.

Use this instruction only if the counter is disabled.

See also

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

Valid for

L16-DIO1, L16-DIO2

Example

see Cnt_Clear

Cnt_Enable

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

Syntax

```
#Include ADWL16.Inc
Cnt_Enable(pattern)
```

Parameters

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

Bit no.	31…2	1	0
Counter no.	–	2[a]	1

a. not available with add-on CO1

See also

Cnt_Clear, Cnt_ClearEnable, Cnt_GetStatus, Cnt_InputMode, Cnt_Latch, Cnt_Mode, Cnt_Read, Cnt_ReadLatch, Cnt_ReadFLatch, Cnt_Set

Valid for

L16, L16-CO1, L16-DIO1, L16-DIO2

Example

see Cnt_Clear

Cnt_GetStatus

Cnt_GetStatus reads out and returns the counter status register.

Syntax

```
#Include ADWL16.Inc
ret_val = Cnt_GetStatus()
```

Parameters

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

Bit no.	31… 28	27	26	25	24	23… 20	19	18	17	16	15… 06	05	04	03… 02	01	00
Signal	-	L2	C2	L1	C1	-	B2	A2	B1	A1	-	N2	N1	-	R2	R1

- :don't care (signal status is not defined, mask out with **0F 0F 00 33h**)
Ax:Signal A (signal is not changing states)
Bx: Signal B (signal is not changing states)
Cx:Correlation error* (signals A and B are identical, they are not phase-shifted by approx. 90°)
Lx: Line error* (cable not connected or the line is broken)
Nx:CLR-/LATCH-input (signal is not changing states)
Rx:Reset-Enable (value which was set by **Cnt_ClearEnable**)
x:Counter number (1 or 2)

* Auto-Reset (is reset during reading out)

Notes

- / -

See also

Cnt_Clear, Cnt_ClearEnable, Cnt_Enable, Cnt_InputMode, Cnt_Latch, Cnt_Mode, Cnt_Read, Cnt_ReadLatch, Cnt_ReadFLatch, Cnt_Set

Valid for

L16-DIO1, L16-DIO2

Example

```
#Include ADWL16.Inc
Dim error As Long

Init:
  Cnt_Mode(0)                   'All counters at external clock
                                'input
  Cnt_Set(0)                    'All counters with A/B-input
                                '(for
                                'instance for incremental
                                'encoder)
  Cnt_InputMode(0)              'Determine functionality
                                'CLR/LATCH: At
                                'all counters as CLR-input
  Cnt_ClearEnable(11b)          'Enables the CLR-function of
                                'counters 1+2
  Cnt_Clear(11b)                'Reset counters 1+2 to 0
  Cnt_Enable(1)                 'Start counter 1
  error = 0                     'Reset error indicator

Event:
  Par_1 = Cnt_Read(1)           'Read out counter 1
  REM Read out counter status register
  Par_2 = Cnt_GetStatus() And 0F0F0033h
  If (Par_2 And 2000000h = 2000000h) Then'Line or cable error
                                'counter 1?
    Inc Par_3                   'Number of line or cable errors
                                'until
                                'now...
    error = 1                   'Set error indicator
  EndIf
  If (Par_2 And 1000000h = 1000000h) Then'Correlation error
                                'counter 1?
    Inc Par_4                   'Number of correlation errors
                                'until
                                'now...
    error = 1                   'Set error indicator
  EndIf
  Par_5 = Shift_Right(Par_2 And 10h,4)
                                'current status of CLR-input
  Par_6 = Shift_Right(Par_2 And 10000h,16)
                                'current status of input A.
  Par_7 = Shift_Right(Par_2 And 20000h,17)
                                'current status of input B.
```

Cnt_InputMode

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

Syntax

```
#Include ADWL16.Inc
Cnt_InputMode(pattern)
```

Parameters

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

Bit no.	31…2	1	0
Counter no.	–	2	1

Notes

Use this instruction only when the counter is not enabled.

See also

Cnt_Clear, Cnt_ClearEnable, Cnt_Enable, Cnt_GetStatus, Cnt_Latch, Cnt_Mode, Cnt_Read, Cnt_ReadLatch, Cnt_ReadFLatch, Cnt_Set

Valid for

L16-DIO1, L16-DIO2

Example

see Cnt_Clear

Cnt_Latch

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

Syntax

```
#Include ADWL16.Inc
Cnt_Latch(pattern)
```

Parameters

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

Bit no.	31…2	1	0
Counter no.	–	2[a]	1

a. not available with add-on CO1

Notes

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

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

Valid for

L16, L16-CO1, L16-DIO1, L16-DIO2

See also

Cnt_Clear, Cnt_ClearEnable, Cnt_Enable, Cnt_GetStatus, Cnt_InputMode, Cnt_Mode, Cnt_Read, Cnt_ReadLatch, Cnt_ReadFLatch, Cnt_Set

Example

see Cnt_Clear

Cnt_Mode

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

Syntax

```
#Include ADWL16.Inc
Cnt_Mode(pattern)
```

Parameters

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

Bit no.	31…2	1	0
Counter no.	–	2	1

Notes

Cnt_Set determines the mode of the selected clock input.

Please use **Cnt_Mode** only when the counter is disabled.

See also

Cnt_Clear, Cnt_ClearEnable, Cnt_Enable, Cnt_GetStatus, Cnt_InputMode, Cnt_Latch, Cnt_Read, Cnt_ReadLatch, Cnt_ReadFLatch, Cnt_Set

Valid for

L16-DIO1, L16-DIO2

Example

see Cnt_Clear

Cnt_Read

Cnt_Read copies the current counter value into Latch A and returns it as return value.

Syntax

```
#Include ADWL16.Inc
ret_val = Cnt_Read(CounterNo)
```

Parameters

CounterNo	Counter number: 1…2; L16-CO1: 1.	LONG
ret_val	Counter value.	LONG

Notes

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

See also

Cnt_Clear, Cnt_ClearEnable, Cnt_Enable, Cnt_GetStatus, Cnt_InputMode, Cnt_Latch, Cnt_Mode, Cnt_ReadLatch, Cnt_ReadFLatch, Cnt_Set

Valid for

L16, L16-CO1, L16-DIO1, L16-DIO2

Example

```
#Include ADWL16.Inc

Dim old, new As Long          'Dimension

Init:
  old = 0                     'Initialize
  Cnt_Mode(0)                 'All counters on external clock
                              'input
  Cnt_Set(1b)                 'counter 1 with clock (CLK) and
                              'direction (DIR) input
  Cnt_InputMode(0)            'Determine functionality
                              'CLR/LATCH: All as CLR
  Cnt_ClearEnable(1b)         'Enables the CLR function of
                              'counter 1
  Cnt_Clear(1b)               'Reset counter 1 to 0
  Cnt_Enable(1b)              'Start counter 1

Event:
  new = Cnt_Read(1)           'read out Latch A counter 1
  Par_1 = new - old           'Calculate the difference (f =
                              'impulses / time)
  old = new                   'Save new counter values as old
```

Cnt_ReadLatch

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

Syntax

```
#Include ADWL16.Inc
ret_val = Cnt_ReadLatch(CounterNo)
```

Parameters

CounterNo	Counter number: 1…2; L16-CO1: 1.	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_ClearEnable, Cnt_Enable, Cnt_GetStatus, Cnt_InputMode, Cnt_Latch, Cnt_Mode, Cnt_Read, Cnt_ReadFLatch, Cnt_Set

Valid for

L16, L16-CO1, L16-DIO1, L16-DIO2

Notes

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

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

Example

```
#Include ADWL16.Inc
Dim rise, rise_old, fall, fall_old As Long
#Define high Par_1
#Define low Par_2
#Define T Par_9
#Define f Par_10

Init:
  rise_old = 0                  'Initialize variables
  fall_old = 0
  Cnt_Mode(11b)                 'Counters 1+2 on internal clock
                                'input
  Cnt_Set(0)                    'All counters with 20 MHz
                                'internal
                                'reference clock (= 50 ns period
                                'duration)
  Cnt_InputMode(11b)            'Determine functionality
                                'CLR/LATCH: At
                                'counters 1+2 as LATCH input
  Cnt_ClearEnable(0)            'Disables the CLR-function of
all
                                'counters
  Cnt_Clear(11b)                'Reset counters 1+2 to 0
  Cnt_Enable(1)                 'Start couner 1

Event:
  rise = Cnt_ReadLatch(1) 'Read out Latch A counter 1
  fall = Cnt_ReadFLatch(1) 'Read out Latch B counter 1
  If (rise <> rise_old) Then 'Is a rising edge detected?
    T = (rise - rise_old) * 50 'Period duration in nanoseconds
    f = 1E9 / T                 'Frequency in Hertz
    If (fall <> fall_old) Then 'Is a falling edge detected?
      high = (fall - rise) * 50 'Impulse duration in nanoseconds
      low = (rise - fall_old) * 50 'Pause duration in nanoseconds
    Else                        'No falling edge is detected
      high = (fall - rise_old) * 50 'Impulse duration in
                                'nanoseconds
      low = (rise - fall) * 50 'Pause duration in nanoseconds
    EndIf
  EndIf
  rise_old = rise               'Save contents of the latch
  fall_old = fall               'Save contents of the latch
```

Cnt_ReadFLatch

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

Syntax

```
#Include ADWL16.Inc
ret_val = Cnt_ReadFLatch(CounterNo)
```

Parameters

CounterNo	Counter number: 1…2.	LONG
ret_val	Contents of Latch B.	LONG

Comment

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

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

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

See also

Cnt_Clear, Cnt_ClearEnable, Cnt_Enable, Cnt_GetStatus, Cnt_InputMode, Cnt_Latch, Cnt_Mode, Cnt_Read, Cnt_ReadLatch, Cnt_Set

Valid for

L16-DIO1, L16-DIO2

Example

see Cnt_ReadLatch

Cnt_Set

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

Syntax

```
#Include ADWL16.Inc
Cnt_Set(pattern)
```

Parameters

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

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

Bit no.	31…2	1	0
Counter no.	–	2	1

Comment

Please use this instruction only when the counter is disabled.

See also

Cnt_Clear, Cnt_ClearEnable, Cnt_Enable, Cnt_GetStatus, Cnt_InputMode, Cnt_Latch, Cnt_Mode, Cnt_Read, Cnt_ReadLatch, Cnt_ReadFLatch

Valid for

L16-DIO1, L16-DIO2

Example

```
#Include ADWL16.Inc

Init:
  Cnt_Mode(0)                 'All counters on external clock
                              'input
  Cnt_Set(00b)                'counters 1+2 with edge
                              'detection
  Cnt_Enable(11b)             'Start counters 1+2
```

13.5 CAN interface

This section describes the following instructions:

CAN_Msg

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

Syntax

```
#Include ADWL16.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

Init_CAN, Read_Msg, Read_Msg_Con, Transmit

Valid for

L16-DIO1

Example

```
#Include ADWL16.Inc
REM Sends a 32 Bit FLOAT-value (here: Pi) as sequence of
REM 4 bytes in a message object

#Define pi 3.14159265
Dim i As Long

Init:
  Init_CAN()                    'Initialize CAN controller

  REM Enable message object 6
  REM for sending with the identifier 40 (11 bit)
  En_Transmit(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(6)                   'Send the message object 6
```

Receiving of a float value see example at Read_Msg.

En_Interrupt

En_Interrupt configures a specified message object to generate an external event when a message arrives.

Syntax

```
#Include ADWL16.Inc
En_Interrupt(msg_no)
```

Parameters

msg_no	Number (1…15) of the message object in the CAN controller.	LONG

Notes

- / -

See also

CAN_Msg, En_Receive, Get_CAN_Reg, Set_CAN_Reg

Valid for

L16-DIO1

Example

```
#Include ADWL16.Inc

Init:
  Init_CAN()                    'Initialization of the CAN
                                'controller
  En_Receive(1,200,0)           'Initialize the message object 1
                                'to
                                'receive CAN messages with
                                'identifier 200
  En_Interrupt(1)               'Enables the triggering of
                                'interrupts
                                '(ext. EVENT) when receiving the
                                'message object 1
```

En_Receive

EN_RECEIVE enables a specified message object to receive messages.

Syntax

```
#Include ADWL16.Inc
En_Receive(msg_no, id, id_extend)
```

Parameters

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, Read_Msg, Read_Msg_Con

Valid for

L16-DIO1

Example

```
#Include ADWL16.Inc

Init:
  Init_CAN()                    'Initialization of the CAN
                                'controller
  En_Receive(1,200,0)           'Initialize the message object 1
                                'to
                                'receive CAN messages with the
                                'identifier 200
```

En_Transmit

EN_TRANSMIT enables a specified message object to send messages.

Syntax

```
#Include ADWL16.Inc
En_Transmit(msg_no, id, id_extend)
```

Parameters

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, Transmit

Valid for

L16-DIO1

Example

```
#Include ADWL16.Inc

Init:
  Init_CAN()                    'Initialization of the CAN
                                'controller
  En_Transmit(6,40,0)           'Initialize the message object 6
                                'to
                                'send CAN messages with
                                'identifier 40
```

Get_CAN_Reg

GET_CAN_REG reads the value of a specified register in the CAN controller.

Syntax

```
#Include ADWL16.Inc
ret_val = Get_CAN_Reg(regno)
```

Parameters

regno	Register number in the CAN controller (0…255).	LONG
ret_val	Contents of the register (transfer to the lower 8 bits).	LONG

Notes

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

See also

Init_CAN, Set_CAN_Baudrate, Set_CAN_Reg

Valid for

L16-DIO1

Example

```
#Include ADWL16.Inc
Init:
  Init_CAN()                    'Initialization of the CAN
                                'controller
  Par_1 = Get_CAN_Reg(0)'Read out the control register
```

Init_CAN

INIT_CAN initializes the CAN controller.

Syntax

```
#Include ADWL16.Inc
Init_CAN()
```

Parameters

- / -

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_Interrupt, En_Receive, En_Transmit, Get_CAN_Reg, Set_CAN_Baudrate, Set_CAN_Reg

Valid for

L16-DIO1

Example

```
#Include ADWL16.Inc

Init:
  Init_CAN()                    'Initialize the CAN controller
```

Read_Msg

READ_MSG checks if 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 ADWL16.Inc
ret_val = Read_Msg(msg_no)
```

Parameters

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, 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_Interrupt, En_Receive, En_Transmit, Read_Msg_Con

Valid for

L16-DIO1

Example

```
#Include ADWL16.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()                  'Initialize the CAN controller
 En_Receive(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)

 If (Par_9 = 40) Then
  REM New message for message object with the identifier 40
  REM has arrived
  Par_1 = CAN_Msg[1]         'Read out high-byte
  For n = 2 To 4             'Combine with remaining 3 bytes
                             'to
   Par_1 = Shift_Left(Par_1,8) + CAN_Msg[n]'a 32-bit value
  Next n
  REM Convert the bit pattern in Par_1 to data type FLOAT and
  REM assign to the variable FPar_1.
  FPar_1 = Cast_LongToFloat(Par_1)
 EndIf
```

Sending a float value see example at Transmit.

Read_Msg_Con

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

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

Syntax

```
#Include ADWL16.Inc
ret_val = Read_Msg_Con(msg_no)
```

Parameters

msg_no	Number (1…15) of the 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_Interrupt, En_Receive, En_Transmit, Read_Msg

Valid for

L16-DIO1

Example

```
#Include ADWL16.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.
REM Sending a float value see example of Transmit.
Dim n As Long

Init:
Par_1 = 0
 Init_CAN()                  'Initialize the CAN controller
 En_Receive(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)

 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
```

Set_CAN_ Baudrate

Set_CAN_Baudrate sets the Baud rate of the CAN controller.

Syntax

```
#Include ADWL16.Inc
ret_val = Set_CAN_Baudrate(rate)
```

Parameters

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-16). Please use the table's notation exactly, i.e. non-integer baud rates with 4 decimal places; values with different notation will be rejected as not allowed.

Set_CAN_Baudrate executes the following actions:

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

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

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

See also

Get_CAN_Reg, Init_CAN, Set_CAN_Reg

Valid for

L16-DIO1

Example

```
#Include ADWL16.Inc
Dim status As Long

Init:
  Init_CAN()                    'Initialize the CAN controller
  status = Set_CAN_Baudrate(125000)'Set the Baud rate to 125
                                'kBit/s
```

Set_CAN_Reg

SET_CAN_REG writes a value into a specified register of the CAN controller.

Syntax

```
#Include ADWL16.Inc
Set_CAN_Reg(regno, value)
```

Parameters

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

Notes

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

See also

Get_CAN_Reg, Init_CAN, Set_CAN_Baudrate

Valid for

L16-DIO1

Example

```
#Include ADWL16.Inc

Init:
  Init_CAN()                    'Initialization of the CAN
                                'controller
  Set_CAN_Reg(0,1)              'Set control register to the
                                'value 1
```

Transmit

TRANSMIT sends the message in **CAN_Msg** via the specified message object.

Syntax

```
#Include ADWL16.Inc
Transmit(msg_no)
```

Parameters

msg_no	Number (1…14) of the message object.	LONG

Notes

To send a message, follow these steps:

- Enable the message object for sending with **En_Transmit**.
- Enter the message into the array **CAN_Msg**: data bytes and the number of data bytes.
- Send the message with **Trasnmit**.

The CAN interface sends the message as soon as the message object has received access rights to the CAN bus.

See also

CAN_Msg, En_Transmit, Init_CAN, Set_CAN_Baudrate

Valid for

L16-DIO1

Example

```
#Include ADWL16.Inc
#Define pi 3.14159265
Dim i As Long

Init:
  Init_CAN()                  'Initialize CAN controller

  REM Enable message object 6
  REM for sending with the identifier 40 (11 bit)
  En_Transmit(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(6)                 'Send message object 6
```

Receiving of a float value see example at Read_Msg.

13.6 SSI interface

This section describes the following instructions:

SSI_Mode

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

Syntax

```
#Include ADWL16.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). Bit = 0: "Single shot" mode, the encoder is read once. Bit = 1: "Continuous" mode, the encoder is read continuously.	LONG

Bit no.	31:1	0
SSI decoder	–	1

Notes

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

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

See also

SSI_Read, SSI_Set_Bits, SSI_Set_Clock, SSI_Start, SSI_Status

Valid for

L16-DIO1, L16-DIO2

Example

```
#Include ADWL16.Inc

Init:
  SSI_Set_Clock(1,10)          'clock rate 1.0 MHz
  SSI_Mode(1b)                 'Set continuous-mode
  SSI_Set_Bits(1,10)           '10 encoder bits

Event:
  Par_1 = SSI_Read(1)          'Read out position value
```

SSI_Read

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

Syntax

```
#Include ADWL16.Inc
ret_val = SSI_Read(dcdr_no)
```

Parameters

dcdr_no	Number (1) 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

L16-DIO1, L16-DIO2

Example

```
#Include ADWL16.Inc
Dim m, n, y As Long

Init:
  SSI_Set_Clock(1,50)          'clock rate 200 kHz
  SSI_Mode(1)                  'Set continuous-mode)
  SSI_Set_Bits(1,23)           '23 encoder bits

Event:
  Par_1 = SSI_Read(1)          'Read out position value

  REM Change value from Gray-code into a binary value:
  m = 0                        'delete value of 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

  Rem The result of the Gray/binary conversion in Par_9
  Par_9 = y
```

SSI_Set_Bits

SSI_SET_BITS sets for an SSI counter the amount of bits which generate a complete encoder value.

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

Syntax

```
#Include ADWL16.Inc
SSI_Set_Bits(dcdr_no,bit_no)
```

Parameters

dcdr_no	Number (1) 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

L16-DIO1, L16-DIO2

Example

```
#Include ADWL16.Inc

Init:
  SSI_Set_Clock(1,10)          'clock rate 1.0 MHz
  SSI_Mode(1b)                 'Set continuous-mode
  SSI_Set_Bits(1,10)           '10 encoder bits

Event:
  Par_1 = SSI_Read(1)          'Read out position value
```

SSI_Set_Clock

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

Syntax

```
#Include ADWL16.Inc
SSI_Set_Clock(dcdr_no,prescale)
```

Parameters

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

Notes

Scale factors < 10 are automatically corrected to the value 10; from values > 255 only the least significant 8 bits are used as scale factor.

The possible clock frequency depends on the length of the cable, cable type, and the send and receive components of the encoder or decoder. Basically the following rule applies: The higher the clock frequency the shorter the cable length.

See also

SSI_Mode, SSI_Read, SSI_Set_Bits, SSI_Start, SSI_Status

Valid for

L16-DIO1, L16-DIO2

Example

```
#Include ADWL16.Inc

Init:
  SSI_Set_Clock(1,10)          'clock rate 1.0 MHz
  SSI_Mode(1b)                 'Set continuous-mode
  SSI_Set_Bits(1,10)           '10 encoder bits

Event:
  Par_1 = SSI_Read(1)          'Read out position value
                               '(encoder 2)
```

SSI_Start

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

Syntax

```
#Include ADWL16.Inc
SSI_Start(dcdr_no)
```

Parameters

dcdr_no	Number (1) of SSI decoder which is to be started.	LONG

Notes

In "continuous" mode **SSI_Start** has no function, because the encoder values are nevertheless read out continuously.

An encoder value will be saved only when the amount of bits is read which is set by **SSI_Set_Bits**.

A complete encoder value is always transferred, even if the operation mode is changing meanwhile.

See also

SSI_Mode, SSI_Read, SSI_Set_Bits, SSI_Set_Clock, SSI_Status

Valid for

L16-DIO1, L16-DIO2

Example

```
#Include ADWL16.Inc

Init:
  SSI_Set_Clock(1,250)          'clock rate 40 kHz
  SSI_Mode(0)                   'Set single shot-mode
  SSI_Set_Bits(1,23)            '23 encoder bits

Event:
  SSI_Start(1b)                 'Read position value
  Do
  Until (SSI_Status(1) = 0)
  Rem If position value is read completely, then …

  Rem read out and display position value
  Par_1 = SSI_Read(1)
```

SSI_Status

SSI_Status returns the current read-status for a specified decoder.

Syntax

```
#Include ADWL16.Inc
ret_val = SSI_Status(dcdr_no)
```

Parameters

dcdr_no	Number (1) 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 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

L16-DIO1, L16-DIO2

Example

```
#Include ADWL16.Inc

Init:
  SSI_Set_Clock(1,250)          'clock rate 40 kHz
  SSI_Mode(0)                   'Set single shot-mode
  SSI_Set_Bits(1,23)            '23 encoder bits

Event:
  SSI_Start(1b)                 'Read position value
  Do
  Until (SSI_Status(1) = 0)
  Rem If position value is read completely, then …

  Rem read out and display position value
  Par_1 = SSI_Read(1)
```

13.7 PWM Outputs

This section describes instructions to access PWM outputs of *ADwin-light-16* with PWM1 add-on:

PWM_Activate

PWM_Activate activates pin 34 (DIGOUT-05) as PWM output or as digital output.

Syntax

```
#Include ADWL16.inc
PWM_Activate(enable)
```

Parameters

enable	Status of pin 34 (DIGOUT-05): 0: activate pin as digital output. 1: activate pin as PWM output.	LONG

Notes

After power-up pin 34 is configured as digital output.

See also

PWM_Enable, PWM_Get_Status, PWM_Init, PWM_Latch, PWM_Reset, PWM_Standby_Value, PWM_Write_Latch

Valid for

L16-PWM1

Example

see PWM_Init (page 139)

PWM_Enable

PWM_Enable enables or disables the PWM output.

Syntax

```
#Include ADWL16.inc
PWM_Enable(enable)
```

Parameters

enable	Status of PWM output: 0: Disable PWM output. 1: Enable PWM output.	LONG

Notes

The time, when the PWM output is disabled–at once or after the next end of period–depends on the setting which was done with **PWM_Init** (parameter mode).

If the PWM output is disabled but activated, the output is set to the standby level (see **PWM_Standby_Value**).

See also

PWM_Activate, PWM_Get_Status, PWM_Init, PWM_Latch, PWM_Reset, PWM_Standby_Value, PWM_Write_Latch

Valid for

L16-PWM1

Example

see PWM_Init (page 139)

PWM_Get_Status

PWM_Get_Status returns the operation status of the PWM output.

Syntax

```
#Include ADWL16.inc
ret_val = PWM_Get_Status()
```

Parameters

ret_val	Status of PWM output. 0: PWM output has finished. 1: PWM output is running.	LONG

Notes

- / -

See also

PWM_Activate, PWM_Enable, PWM_Init, PWM_Latch, PWM_Reset, PWM_Standby_Value, PWM_Write_Latch

Valid for

L16-PWM1

Example

- / -

PWM_Init

PWM_Init sets the defaults for the PWM output.

Syntax

```
#Include ADWL16.inc
PWM_Init(pwm_output, startlevel, startvalue, mode,
     count)
```

Parameters

pwm_output	Number (1) of PWM output.	LONG
startdelay	Start delay in units of 25ns.	LONG
startlevel	Start level of PWM output: 0: TTL level low. 1: TTL level high.	LONG
mode	Operating mode of PWM output as bit pattern (bits 0…2 only). Bit 0: Moment to take over a new PW frequency: Bit = 0: Take over at end of period. Bit = 1: Take over immediately. Bit 1: Number of pulses: Bit = 0: infinite number of periods. Bit = 1: number of periods is count. Bit 2: Moment to stop after stop instruction: Bit = 0: Stop at end of period. Bit = 1: Stop immediately.	LONG
count	Number of periods (1…32/68), which are processed during an output cycle. Only relevant, if mode, bit 1 = 1.	LONG

Notes

- / -

See also

PWM_Activate, PWM_Enable, PWM_Get_Status, PWM_Latch, PWM_Reset, PWM_Standby_Value, PWM_Write_Latch

Valid for

L16-PWM1

Example

```
#Include ADWL16.inc
Rem You can set frequency and duty cycle online with the
Rem global variables FPar_1 and FPar_2:
#Define freq1 FPar_1       'frequency
#Define pw1 FPar_2         'duty cycle

Init:
  PWM_Activate(1)           'enable pin 34 as PWM output
  freq1 = 1000              '1000 Hz
  pw1 = 50                  '50 %
  PWM_Reset(01b)            'stop PWM channel
  PWM_Init(1,0,0,0,0)       'initialize PWM settings

  PWM_Write_Latch(1,pw1,freq1)'set frequency and duty cycle
  PWM_Latch(1)              'enable output of PWM signal
  PWM_Enable(1)             'start output

Event:
  PWM_Write_Latch(1,pw2,freq2)'set new frequency and duty cycle
  PWM_Latch(1)              'set frequency and duty cycle

Finish:
  PWM_Activate(0)           'reset pin 34 as digital output
```

PWM_Latch

PWM_Latch enables frequency and duty cycle of the PWM output to be output.

Syntax

```
#Include ADWL16.inc
PWM_Latch(enable)
```

Parameters

pattern	Output status of the PWM output: 0: No influence. 1: latch = enable for output.	LONG

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_Activate, PWM_Enable, PWM_Get_Status, PWM_Init, PWM_Reset, PWM_Standby_Value, PWM_Write_Latch

Valid for

L16-PWM1

Example

see PWM_Init (page 139)

PWM_Reset

PWM_Reset stops the output of the PWM output immediately.

Syntax

```
#Include ADWL16.inc
PWM_Reset(pattern)
```

Parameters

pattern	Status of the PWM output: 0: No influence 1: Stop PWM output immediately.	LONG

Notes

The output will be stopped immediately even if **PWM_Init** was set a different stop mode.

See also

PWM_Activate, PWM_Enable, PWM_Get_Status, PWM_Init, PWM_Latch, PWM_Standby_Value, PWM_Write_Latch

Valid for

L16-PWM1

Example

see PWM_Init (page 139)

PWM_Standby_Value

PWM_Standby_Value sets the standby TTL level for the PWM output.

Syntax

```
#Include ADWL16.inc
PWM_Standby_Value(level)
```

Parameters

level	Standby TTL level of the PWM output: 0: TTL-level low (default) 1: TTL-level high	LONG

Notes

If the PWM output is disabled with **PWM_Enable**, the output is set to the standby level. The standby level will also be set after the PWM output has stopped.
After power-up the output is set to TTL-level low.

See also

PWM_Activate, PWM_Enable, PWM_Get_Status, PWM_Init, PWM_Latch, PWM_Reset, PWM_Write_Latch

Valid for

L16-PWM1

Example

- / -

PWM_Write_Latch

PWM_Write_Latch writes duty cycle and frequency into the latch register.

Syntax

```
#Include ADWL16.inc
PWM_Write_Latch(pwm_output, dutycycle, frequency)
```

Parameters

pwm_output	Number (1) 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 …20 MHz.	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 about 400 kHz.

See also

PWM_Activate, PWM_Enable, PWM_Get_Status, PWM_Init, PWM_Latch, PWM_Reset, PWM_Standby_Value

Valid for

L16-PWM1

Example

see PWM_Init (page 139)

13.8 SPI Interface

This section describes instructions to access the SPI interface of *ADwin-light-16* with PWM1 add-on:

SPI_Config

SPI_Config configures the SPI master.

Syntax

```
#Include ADWL16.inc
SPI_Config(data_order, mode, bits, clock)
```

Parameters

data_order	Order how data is being sent: 0: send MSB first. 1: send LSB first.	LONG
mode	Operation mode of the SPI Master: 0: CPOL=0, CPHA=0 1: CPOL=0, CPHA=1 2: CPOL=1, CPHA=0 3: CPOL=1, CPHA=1	LONG
bits	Number (1…32) of bits transferred in a SPI message.	LONG
clock	Setting (1…7) of the clock rate: 1: 5000 kHz 2: 2500 kHz 3: 1250 kHz 4: 620 kHz 5: 312.5 kHz 6: 156.25 kHz 7: 78.125 kHz	LONG

Notes

We recommend to disable all slaves via Slave Select lines before configuring the SPI master. If you configure the SPI master spikes can occur which are misinterpreted by the connected slaves. This fault will confuse the data transfer.

You find more information about the SPI bus in chapter 10.2 "SPI Interface" on page 55.

See also

SPI_Enable, SPI_Get_MISO, SPI_Set_MOSI, SPI_Start, SPI_Static_MISO, SPI_Status, SPI_Wait

Valid for

L16-PWM1

Example

```
#Include ADwL16.inc

Rem SPI settings
#Define bits 8                'number of data bits
#Define prescale 7            'clock divider: 78.125 kHz
#Define data_order 0          '0 = MSB first
#Define mode 3                'CPOL = 1, CPHA = 1

Init:
  SPI_Enable(1)               'enable ADwin I/O Connector pins
  SPI_Config(data_order, mode, bits, prescale)
  Par_1 = 10                  'first data value

Event:
  SPI_Set_MOSI(Par_1)         'set data value to be sent

  Rem To select a slave you have to send the signal "Slave
  Rem select" to the SPI slave. Connect a free DIO output to
  Rem the slave SPI input and set the required TTL output level
  Rem with the appropriate standard DIO instruction.

  Rem Start slave select via TTL low on pin DIGOUT-1
  Clear_Digout(0)
  SPI_Start()                 'start data transfer
  SPI_Wait()                  'wait until end of data transfer
  Set_Digout(0)               'end slave select

  Par_2 = SPI_Get_MISO()      'read received data value
```

SPI_Enable

SPI_Enable switches pins as SPI signals CLK, MISO and MOSI or as digital outputs.

Syntax

```
#Include ADWL16.inc
SPI_Enable(enable)
```

Parameters

enable	Pin status on DSub sockets: 0: all pins as digital outputs (default). 1: pins 13, 32 and 33 at ADwin I/O-Connector socket as SPI signals. 3: pins 16, 34 and 35 at Digital I/O socket as SPI signals.	LONG

Notes

Pin assignments are found on page 55chapter 10.2 "SPI Interface".

Please note: If you switch pins on the DSub socket Digital I/O as SPI signals, the pins DIO-24…DIO-31 are automatically configured as outputs and the pins DIO16…DIO-23 as inputs. If you switch the pins as digital outputs afterwards, the previous configuration done with **Conf_DIO_E** is valid again.

In addition to the pins described above, you require a separate Slave Select line for each addressed SPI slave to enable or disable data transfer. If you use the remaining digital outputs, set the required TTL level with the appropriate instructions for digital outputs like **Digout_Clear** or **Digout_Set** (see chapter 13.3 on page 77).

See also

SPI_Config, SPI_Get_MISO, SPI_Set_MOSI, SPI_Start, SPI_Static_MISO, SPI_Status, SPI_Wait, Conf_DIO_E, Clear_Digout, Set_Digout

Valid for

L16-PWM1

Example

see SPI_Config (page 147)

SPI_Get_MISO

SPI_Get_MISO reads (an already received) SPI message from the input register.

Syntax

```
#Include ADWL16.inc
ret_val = SPI_Get_MISO()
```

Parameters

ret_val	Data value which was sent from the addressed SPI slave.	LONG

Notes

You set the number of bits of the SPI message with the parameter bits of the instruction **SPI_Config**.

You can only read the SPI message when the data transfer has finished; check the state of data transfer with **SPI_Wait** or **SPI_Status**.

See also

SPI_Config, SPI_Enable, SPI_Set_MOSI, SPI_Start, SPI_Static_MISO, SPI_Status, SPI_Wait

Valid for

L16-PWM1

Example

see SPI_Config (page 147)

SPI_Set_MOSI

SPI_Set_MOSI provides data to be sent to a selected SPI slave.

Syntax

```
#Include ADWL16.inc
SPI_Set_MOSI(value)
```

Parameters

value	Data value to be sent to a selected SPI slave.	LONG

Notes

Using **SPI_Set_MOSI**, the SPI message is only prepared to be sent. You start the data transfer with the instruction **SPI_Start**.

You set the number of bits of the SPI message with the parameter bits of the instruction **SPI_Config**.

See also

SPI_Config, SPI_Enable, SPI_Get_MISO, SPI_Start, SPI_Static_MISO, SPI_Status, SPI_Wait

Valid for

L16-PWM1

Example

see SPI_Config (page 147)

SPI_Start

SPI_Start starts the data transfer via the SPI bus.

Syntax

```
#Include ADWL16.inc
SPI_Start()
```

Parameters

- / -

Notes

The data transfer runs in both directions: **SPI_Start** sends the SPI message which has previously been provided with **SPI_Set_MOSI**. The SPI slave normally answers within the same data transfer; you read the answer with **SPI_Get_MOSI**.

You can query the end of data transfer with **SPI_Wait** or **SPI_Status**.

See also

SPI_Config, SPI_Enable, SPI_Get_MISO, SPI_Set_MOSI, SPI_Static_MISO, SPI_Status, SPI_Wait

Valid for

L16-PWM1

Example

see SPI_Config (page 147)

SPI_Static_MISO

SPI_Static_MISO reads the current TTL level of the data line of the SPI bus.

Syntax

```
#Include ADWL16.inc
ret_val = SPI_Static_MISO()
```

Parameters

ret_val	TTL level of the data line: 0: TTL level low. 1: TTL level high.	LONG

Notes

Some SPI slaves use the data line not only for data transfer but also to send a signal to the SPI master. In this case, you can read the TTL level of the data line with **SPI_Static_MISO** and react according to the SPI slave.

See also

SPI_Config, SPI_Enable, SPI_Get_MISO, SPI_Set_MOSI, SPI_Start, SPI_Status, SPI_Wait

Valid for

L16-PWM1

Example

```
#Include ADwL16.inc
Rem The program communicates with an SPI slave to make it
Rem convert an analog value and put the converted value on
Rem the SPI bus.

Rem SPI settings
#Define data_order 0          'MSB first
#Define mode 0                'CPOL = 0, CPHA = 0
#Define bits 8                'number of data bits
#Define clock 3               'clock divider: 1250 kHz

Dim state As Long
Dim value As Long

Init:
  Rem Disable slave select via TTL high on pin DIGOUT-1
  Set_Digout(0)
  SPI_Enable(1)               'enable ADwin I/O Connector pins
  SPI_Config(data_order, mode, bits, clock)
  state = 0                   'state: idle

Event:
  If (state = 0) then         'send command „start conversion“
    Rem Set output value 11 to make the SPI slave start a
    Rem conversion with the slave’s ADC.
    SPI_Set_MOSI(11)
    Clear_Digout(0)           'start Slave Select
    SPI_Start(1)              'send command
    state = 1                 'state: start conversion
  EndIf

  If (state = 1) Then         'check if command is transferred
    value = SPI_Status()
    If (value = 0) Then state = 2 'state: slave runs conversion
  EndIf

  If (state = 2) Then         'check if slave is ready
    Rem The slave sets the data line TTL high if the conversion
    Rem is completed.
    value = SPI_Static_MISO()
    If (value = 1) Then state = 3 'state: conversion completed
  EndIf

  If (state = 3) Then'send command „transfer ADC value“
    Rem Set output value 12 to make the SPI slave put the
    Rem converted value on the bus.
    SPI_Set_MOSI(12)
    SPI_Start(1)              'send command and receive value
    state = 4                 'state: transfer ADC value
  EndIf

  If (state = 4) Then         'check if command is transferred
    value = SPI_Status()
    If (value = 0) Then state = 5 'state: ADC value is ready
  EndIf

  If (state = 5) Then
    Set_Digout(0)             'end Slave Select
    Par_2 = SPI_Get_MISO()    'read ADC value
    End
  EndIf
```

SPI_Status

SPI_Status returns the status of the current SPI data transfer.

Syntax

```
#Include ADWL16.inc
ret_val = SPI_Status()
```

Parameters

ret_val	Status of SPI data transfer: 0: Data transfer is still running. 1: Data transfer is completed.	LONG

Notes

You can use **SPI_Status** as an alternative to **SPI_Wait** to detect the end of data transfer.

See also

SPI_Config, SPI_Enable, SPI_Get_MISO, SPI_Set_MOSI, SPI_Start, SPI_Static_MISO, SPI_Wait

Valid for

L16-PWM1

Example

- / -

SPI_Wait

SPI_Wait waits until the end of an SPI data transfer.

Syntax

```
#Include ADWL16.inc
SPI_Wait()
```

Parameters

- / -

Notes

- / -

See also

SPI_Config, SPI_Enable, SPI_Get_MISO, SPI_Set_MOSI, SPI_Start, SPI_Static_MISO, SPI_Status

Valid for

L16-PWM1

Example

see SPI_Config (page 147)

Annex

A.1 Technical Data

General Data / Limit Values						
	Symbol	Conditions	min.	typ.	max.	Unit
Supply Voltage						
voltage	U_b	*L16-PCI, -EURO*	4.75	5	5.25	V
		L16-EXT	10	12	36	
Supply Voltage with USB						
operating current						
L16, *L16-CO1*	I_{idle} at $U_{b, typ.}$	*L16-PCI, -EURO*	0.75	0.9	1.5	A
		L16-EXT	0.35	0.5	0.7	
L16-DIO1, -DIO2, -DIO3		*L16-PCI, -EURO*	0.85	1.0	1.6	
		L16-EXT	0.55	0.7	1.0	
inrush current						
L16, *L16-CO1*	$I_{power-on}$ bei $U_{b, typ.}$	*L16-PCI, -EURO*		6		A
		L16-EXT		3		
L16-DIO1, -DIO2, -DIO3		*L16-PCI, -EURO*		7		
		L16-EXT		3		
Supply Voltage with Ethernet						
operating current						
L16, *L16-CO1*	I_{idle} bei $U_{b, typ.}$	*L16-PCI, -EURO*	1.0	1.2	1.8	A
		L16-EXT	0.55	0.7	0.9	
L16-DIO1, -DIO2, -DIO3		*L16-PCI, -EURO*	1.15	1.3	1.9	
		L16-EXT	0.55	0.7	1.0	
inrush current						
L16, *L16-CO1*	$I_{power-on}$ bei $U_{b, typ.}$	*L16-PCI, -EURO*		6.4		A
		L16-EXT		3.3		
L16-DIO1, -DIO2, -DIO3		*L16-PCI, -EURO*		7.5		
		L16-EXT		3.3		
Operation						
temperature	$T_{environment}$	*L16-PCI,* -EURO	+5		+50	°C
	$T_{chassis}$	L16-EXT	+5		+55	
relative humidity	H_{rel}	no condensation	0		80	%
Storage						
temperature	T		-20		+70	°C

General Data / Limit Values						
	Symbol	Conditions	min.	typ.	max.	Unit
Dimensions						
width x height x depth	w x h x d	L16-PCI-USB		21.5 x 121.0 x 172.5		mm
		L16-EURO-USB [a]		5 TE (1") x 3 HE (5") x 187		
		L16-EURO-ENET [a]		10 TE (2") x 3 HE (5") x 187		
		L16-EXT-USB		226 x 109 x 44		
		L16-EXT-ENET		226 x 109 x 74		
		+ CO1 add-on		see basic version		
		+ DIO1/DIO2 add-on		*L16-EURO* [a]: width +10 TE *L16-EXT*: height +30		
		+ DIO3 add-on		*L16-EURO* [a]: width +5 TE *L16-EXT*: height +16		
Net Weight						
weight	m_{Netto}	L16-PCI-USB		135		g
		L16-EURO-USB		165		
		L16-EURO-ENET		275		
		L16-EXT-USB		1.100		
		L16-EXT-ENET		1.400		
		+ CO1 add-on		see basic version		
		+ DIO1/DIO2 add-on		+140		
		+ DIO3 add-on		+80		
Connectors						
DSub-connectors	Metric ISO thread; UNC thread available as ordering option only.					
Mounting						
standard	*L16-PCI*: installation in the PC *L16-EURO*: installation in the 19"-enclosure *L16-EXT*: desktop unit					
optional	DNI rail mounting and wall monting for *L16-EXT*					

[a] Conversion for L16-EURO: 5 TE = 25.4mm = 1 inch; 10 TE = 50.8mm = 2 inch; 3 HE = 133.35 mm

Digital Inputs/Outputs						
Parameters	Symbol	Conditions	min.	typ.	max.	Unit
I/O-lines						
line	DIGIN05:00 DIGOUT05:00	6 inputs and 6 outputs (TTL- / 5V-CMOS-level)				
	EVENT	1 ext. trigger input (positive TTL-logic)				
Circuitry see "Circuitry of digital Inputs/Outputs", TTL inputs / TTL outputs, page A-6						
Counters						
Number and function	2 incremental counters.					
Counter / latch res.				32		Bit
max. counter frequency	f_{CNT}			10		MHz
EVENT input						
Edge recognition, pos.	V_{T+} (Low)	V_{CC} = 5V	1.65	1.9	2.15	V
Edge recognition, neg.	V_{T-} (High)	V_{CC} = 5V	0.75	1.0	1.25	
Switching hysteresis	V_{T+} - V_{T-}		0.4	0.9		

Digital Inputs/Outputs						
Parameters	Symbol	Conditions	min.	typ.	max.	Unit
input current	I_{IH}	V_I = 2.7V			20	µA
	I_{IL}	V_I = 0.4V			-50	

Analog Inputs/Outputs						
Parameters	Symbol	Conditions	min.	typ.	max.	Unit
Inputs						
number	8, differential via multiplexer					
input resistance	R_i		323.4	330	336.6	kΩ
overvoltage	$U_{in\ max.}$	ON & OFF			±35	V
MUX settling time	t_{MUX}	2 LSB 16 Bit		6.5 [a]		µs
*at an internal resistance of thevoltage source of less than 10 Ω						
ADC 16-Bit						
conversion time	t_{conv}	Rev. A			10	µs
		Rev. B [b]			2	
measurement range	U_{in}		-10		+9.999695	V
diff. common mode voltage					±2.0	V
integral non-linearity	INL			±1	±3	LSB
differential non-linearity	DNL			±0.25	±0.5	
offset	drift [c]			±2		ppm/°C
	error	adjustable				
gain	drift [c]			±20		ppm/°C
	error	adjustable				
DAC 16-Bit						
number	2					
output voltage	U_{out}		-10		+9.999695	V
settling time	t_{settle}	2V step			3	µs
		FSR (20V) [A]			10	
maximum current					±5	mA
integral non-linearity	INL				±2	LSB
differential non-linearity	DNL				±1	
offset	error	adjustable				
gain	error	adjustable				

[a] FSR = Full Scale Range
[b] Software instructions use conversion time 10µs unless faster conversion ist enabled.
[c] Refers to total voltage range (FSR)

Processor						
Parameters	Symbol	Conditions	min.	typ.	max.	Unit
type	ADSP21062 (SHARC™)					
manufacturer	Analog Devices					
clock frequency	f_{CLK}			40		MHz
register width				32		Bit
internal memory	SRAM	for the program		128		kByte
		for data		128		
external memory	SDRAM	Rev. A		8		MByte
		Rev. B		16		

CO1 Add-on						
Parameters	Symbol	Conditions	min.	typ.	max.	Unit
Counters						
Number and function	1 up/down counter with four edge evaluation for connection of incremental encoders. The incremental counters of the basic version are replaced.					
Counter inputs	2 inputs (A, B), alternatively available as digital inputs					
Counter and latch resolution				32		Bit

DIO1 Add-On						
Parameters	Symbol	Conditions	min.	typ.	max.	Unit
Reference crystal oscillator						
reference frequency	f_{ref}			20		MHz
prescaler by 4	$f_{ref/4}$			5		
accuracy and drift					100	ppm
Counters						
Number and function	2 up/down counters for impulse, period duration and duty cycle measurements as well as a four edge evaluation for connection of incremental encoders. The incremental counters of the basic version are replaced.					
Counter inputs	3 differental inputs (A/CLK, B/DIR, CLR/LATCH) for each counter; counter programmable with differential or single ended inputs.					
Counter and latch resolution				32		Bit
count frequency	f_{CLK}	input CLK		20		MHz
		input A/B		5		
Digital Inputs/Outputs						
number	DIO31:00	32 (programmable in groups of 8 as inputs or outputs)				
	EVENT	ext. trigger input (pos. TTL-logic)				
Circuitry see "Circuitry of digital Inputs/Outputs", TTL inputs / TTL outputs, page A-6						
Interfaces						
CAN	CAN High speed, 1 interface					
SSI	SSI decoder (since Rev. B), 1 interface					

DIO2 Add-On						
Parameters	Symbol	Conditions	min.	typ.	max.	Unit
Reference crystal oscillator						
reference frequency	f_{ref}			20		MHz
prescaler by 4	$f_{ref/4}$			5		
accuracy and drift					100	ppm
Counters						
Number and function	2 up/down counters for impulse, period duration and duty cycle measurements as well as a four edge evaluation for connection of incremental encoders. The incremental counters of the basic version are replaced.					
Counter inputs	3 differental inputs (A/CLK, B/DIR, CLR/LATCH) for each counter; counter programmable with differential or single ended inputs.					
Counter and latch resolution				32		Bit
count frequency	f_{CLK}	input CLK		20		MHz
		input A/B		5		
Digital Inputs/Outputs						
number	DIO31:00	32 (programmable in groups of 8 as inputs or outputs)				
	EVENT	ext. trigger input (pos. TTL-logic)				
Circuitry see "Circuitry of digital Inputs/Outputs", TTL inputs / TTL outputs						
Interfaces						
SSI	SSI decoder (since Rev. B), 1 interface					

DIO3 Add-On						
Parameters	Symbol	Conditions	min.	typ.	max.	Unit
Digital Inputs/Outputs						
number	DIO31:00	32 (programmable in groups of 8 as inputs or outputs)				
	EVENT	ext. trigger input (pos. TTL-logic)				
Circuitry see "Circuitry of digital Inputs/Outputs", TTL inputs / TTL outputs						

Circuitry of digital Inputs/Outputs						
TTL Inputs*						
max. input voltage		TTL level	-0.5		+5.5	V
logic-input voltage	V_{IH} (High)	V_{CC} = 5V	2			
	V_{IL} (Low)	V_{CC} = 5V			0.8	
logic-input current	I_I	V_{CC} = 5V		±0.1	±1000	nA
differential inputs						
Differential input threshold voltage	V_{TH}	-10V ≤ V_{CM} ≤ 13,2V	-200		+200	mV
Input hysteresis	ΔV_{TH}	-10V ≤ V_{CM} ≤ 13,2V		40		mV
Common-mode range	V_{CM}		-10		+13.2	V
Differential slew rate			0.33			V/µs
Allowed differential input voltage		each input			±3.9	V
TTL 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	each DIO line			±35	mA
	I_{TOTAL}	each DIO group (8) via V_{CC} / GND			±70	
* see also data sheet SN74 HCT 245 from Texas Instruments						

A.2 Hardware Addresses - General Overview

Address [HEX]	Function	Bit No.								Comments	Register available in the module		
		31...16	15...6	5	4	3	2	1	0		L16	L16+ CO1	L16+ DIO1
20 40 00 00	set multiplexer to input channel (ADC 01... ADC 15)	-	-	-	-	-	n	n	n	"nnn" binary = 0...7 decimal, selected channel = nnn*2 + 1	x	x	x
20 40 00 10	start conversion: ADC #1	-	-	-	-	-	-	-	s	s = 0 : start conversion s = 1 : no effect	x	x	x
	start conversion: all DACs synchronously	-	-	-	-	-	s	-	-				
20 40 00 20	conversion status (EOC) ADC #1	-	-	-	-	-	-	-	e	e = 0 : end of conversion e = 1 : conversion is running	x	x	x
20 40 00 30	read out register: ADC #1	-	x	x	x	x	x	x	x	x : result of conversion	x	x	x
20 40 00 50	only write into register: DAC #1	-	x	x	x	x	x	x	x	x : digital value to be converted	x	x	x
20 40 00 60	only write into register: DAC #2	-	x	x	x	x	x	x	x				
20 40 00 B0	input register DIGIN-05:00	-	-	x	x	x	x	x	x	x : read digital value	x	x	x
20 40 00 C0	output register DIGOUT-05:00	-	-	x	x	x	x	x	x	x : digital value to be output	x	x	x
20 40 00 C4	set DIGOUT bits	-	-	x	x	x	x	x	x	x = 0 : no effect x = 1 : set bit	x	x	x
20 40 00 C8	clear DIGOUT bits	-	-	x	x	x	x	x	x	x = 0 : no effect x = 1 : clear bit	x	x	x
20 40 01 00	read out register and start conversion: ADC #1	-	x	x	x	x	x	x	x	x : digital value to be converted	x	x	x
20 40 02 00	write into register and start conversion immediately: DAC #1	-	x	x	x	x	x	x	x	x : digital value to be converted	x	x	x
20 40 02 04	contents of Latch A, counter #1	x	x	x	x	x	x	x	x	x : contents of latch register	x	x	x
20 40 02 08	contents of Latch B, counter #1 (DIO1 only)	x	x	x	x	x	x	x	x	x :contents of latch register	-	-	x
20 40 02 10	write into register and start conversion immediately: DAC #2	-	x	x	x	x	x	x	x	x : digital value to be converted	x	x	x
20 40 02 14	contents of Latch A, counter #2	x	x	x	x	x	x	x	x	x : contents of latch register	x	-	x
20 40 02 18	contents of Latch B, counter #2	x	x	x	x	x	x	x	x	x : contents of latch register	-	-	x
20 40 03 00	enable/disable counter: **CNT_ENABLE**()	-	-	-	-	-	-	x	x	x = 0 : disable counter x = 1 : enable counter	x	Bit 0 only	x
20 40 03 10	clear counter: CNT_CLEAR() *	-	-	-	-	-	-	x	x	x = 0 : no effect x = 1 : clear counter	x	Bit 0 only	x
20 40 03 20	latch counter: CNT_LATCH() *	-	-	-	-	-	-	x	x	x = 0 : no effect x = 1 : latch counter	x	Bit 0 only	x
20 40 03 30	counter inputs CLR or LATCH	-	-	-	-	-	-	x	x	x = 0 : CLR input x = 1 : LATCH input	-	-	x
20 40 03 40	impulse/event counter or pulse width/period duration measurement	-	-	-	-	-	-	x	x	x = 0 :external clock input x = 1 : internal reference clock (20MHz / 5MHz)	-	-	x
20 40 03 50	4 edge evaluation/ CLK+DIR or 20MHz / 5MHz reference clock	-	-	-	-	-	-	x	x	CNT_MODE=0: x=0: 4 edge evaluation; x=1: CLK+DIR CNT_MODE=1: x=0: 20MHz; x=1: 5MHz	-	-	x
20 40 04 54	bits DIO-15:00	-	x	x	x	x	x	x	x	x = 0: clear output x = 1: set output	-	-	x
20 40 04 64	bits DIO-31:16	-	x	x	x	x	x	x	x	x = 0: clear output x = 1: set output	-	-	x
20 40 04 74	set bits DIO-15:00 **	-	x	x	x	x	x	x	x	x = 0: no effect x = 1: set output	-	-	x
20 40 04 84	set bits DIO-31:16 **	-	x	x	x	x	x	x	x	x = 0: no effect x = 1: set output	-	-	x
20 40 04 94	clear bits DIO-15:00 **	-	x	x	x	x	x	x	x	x = 0: no effect x = 1: clear output	-	-	x
20 40 04 A4	clear bits DIO-31:16 **	-	x	x	x	x	x	x	x	x = 0: no effect x = 1: clear output	-	-	x
20 40 04 6C	configure inputs/outputs CONF_DIO() Bit 0: DIO 07:00; Bit 1: DIO 15:08 Bit 2: DIO 23:16; Bit 3: DIO 31:24	-	-	-	-	x	x	x	x	x = 0: group as input x = 1: group as output	-	-	x

* after execution the register is automatically reset
** function without any effect at inputs

A.3 Hardware-Revisions

The revision of a dvice is marked on the casing. The differences of the revision status' are shown below:

Revision	First release	Changes to previous revision status
A	1998	First release.
B1	Mar. 2006	New layout, but compatible to rev. A. External memory extended to 16 MiB. Faster A/D conversion available via instruction **L16_MODE**. Additional sequence control for conversion of analog inputs. Additional SSI interface for DIO1-add-on, new add-ons DIO2 and DIO3.
B2	Aug. 2006	New interface for LS bus

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.

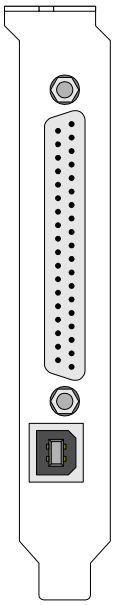
The following substances are involved:

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

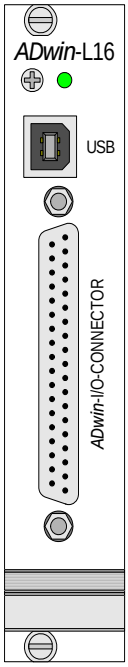
The product line ***ADwin-light-16*** complies with the requirements of the RoHS directive in all delivered variants since revision B1.

A.5 Overview Connectors / Enclosures

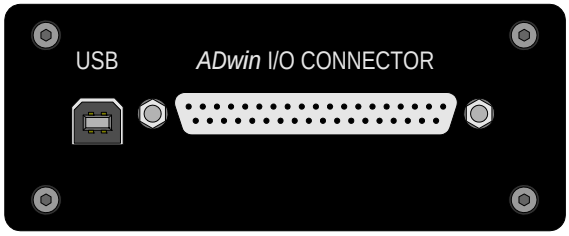
Basic version and add-on CO1 up to Rev. B1



L16-PCI



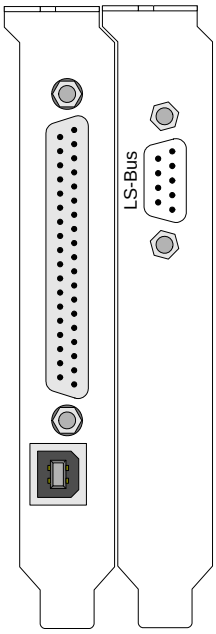
L16-EURO



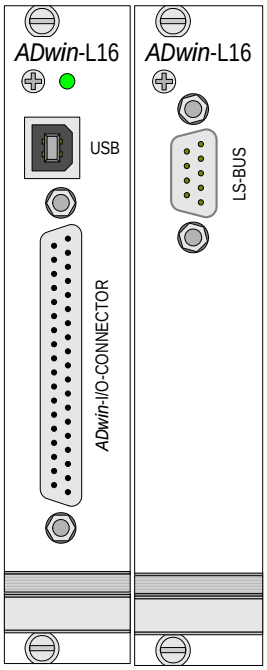
L16-EXT: Front side

L16-EXT: Back side

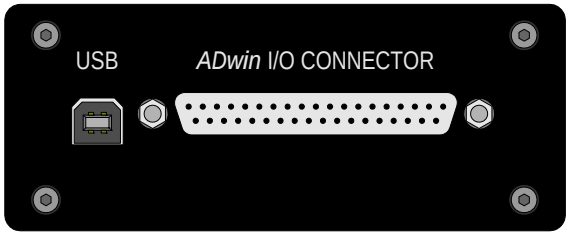
Basic version and add-on CO1 since Rev. B2



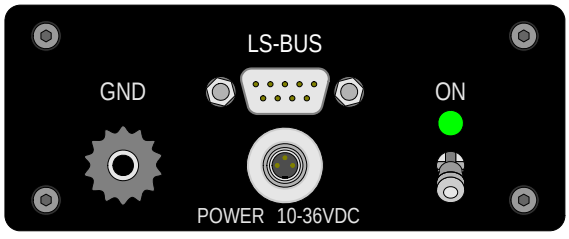
L16-PCI



L16-EURO

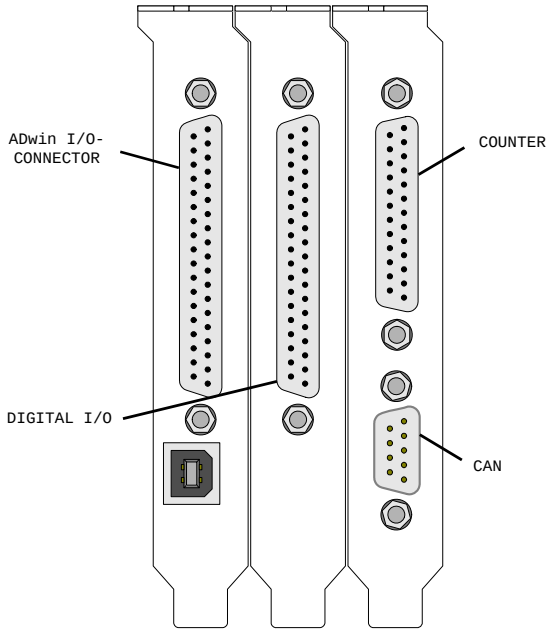


L16-EXT: Front side

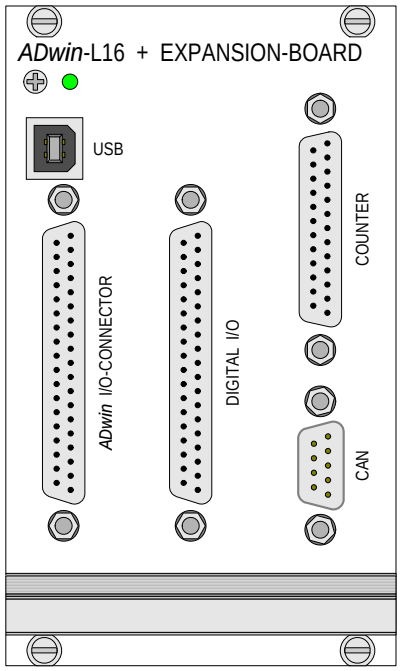


L16-EXT: Back side

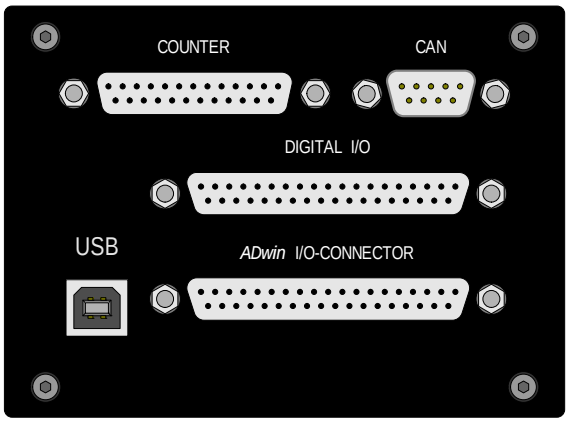
ADwin-light-16 with DIO1 add-on up to Rev. B1



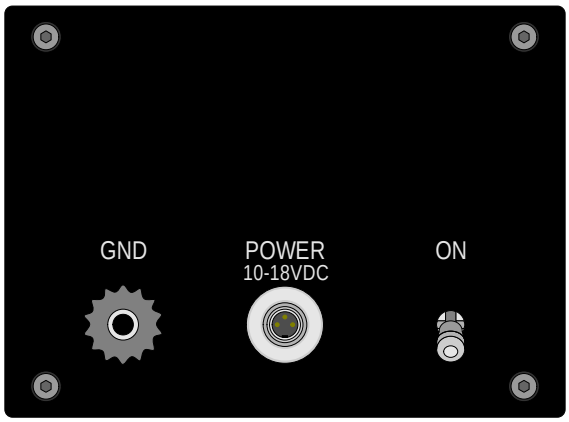
L16-DIO1-PCI



L16-DIO1-EURO

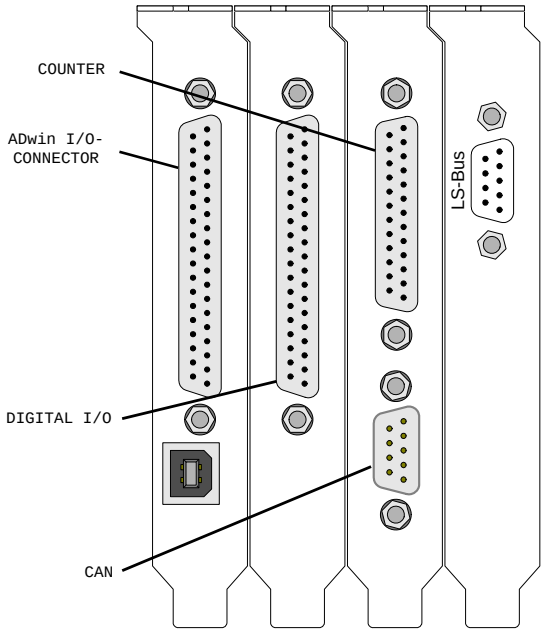


L16-DIO1-EXT: Front side

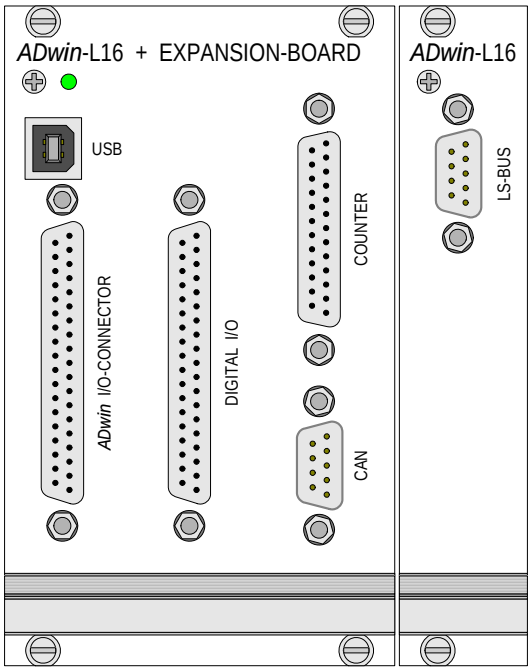


L16-DIO1-EXT: Back side

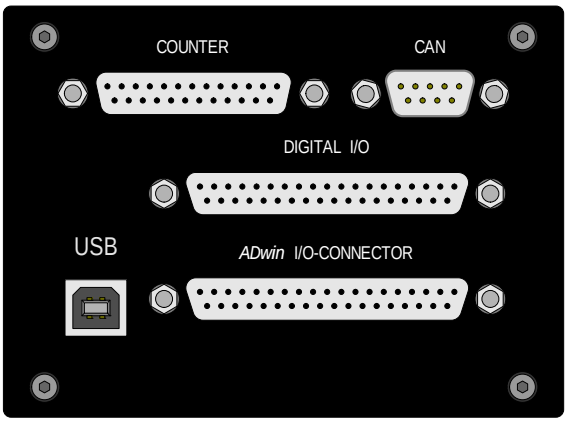
ADwin-light-16 with DIO1 add-on since Rev. B2



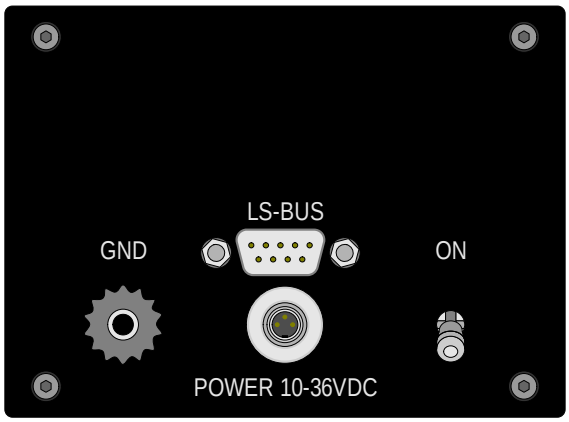
L16-DIO1-PCI



L16-DIO1-EURO

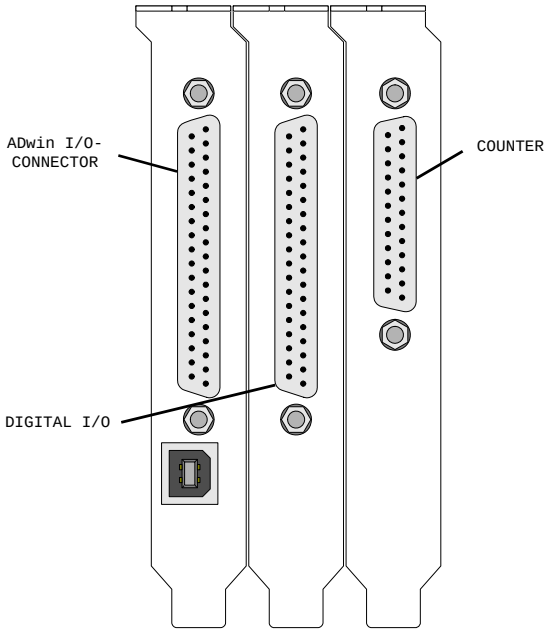


L16-DIO1-EXT: Front side

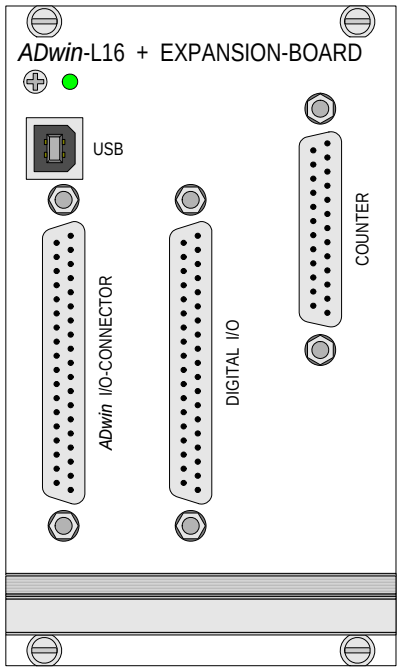


L16-DIO1-EXT: Back side

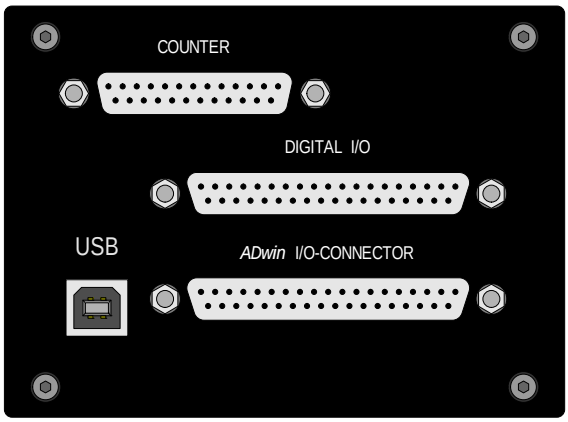
ADwin-light-16 with DIO2 add-on, Rev. B1



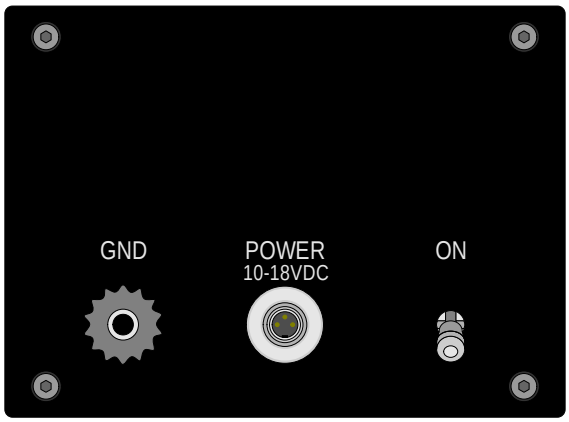
L16-DIO2-PCI



L16-DIO2-EURO

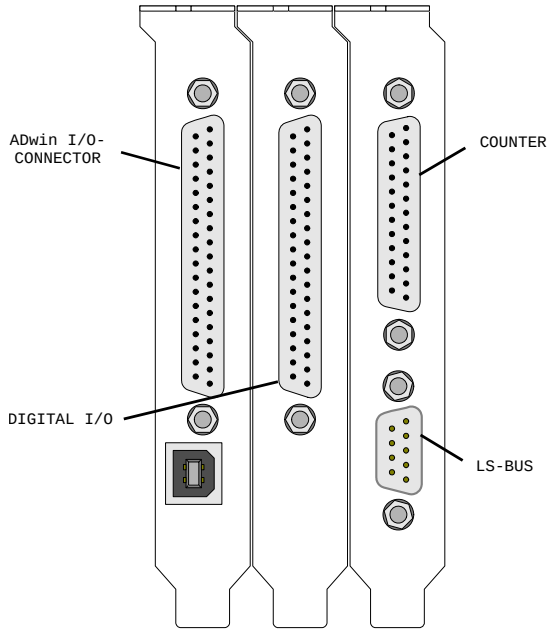


L16-DIO2-EXT: Front side

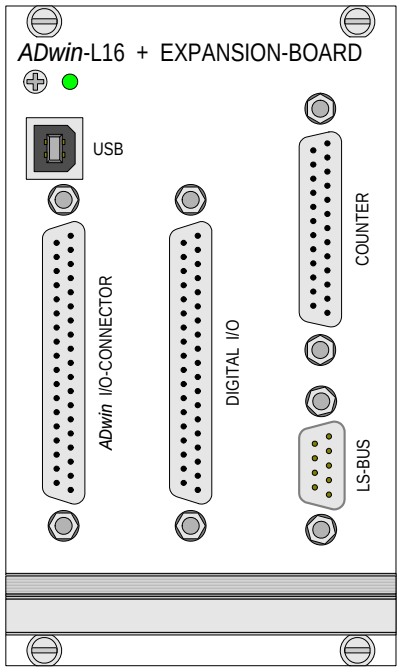


L16-DIO2-EXT: Back side

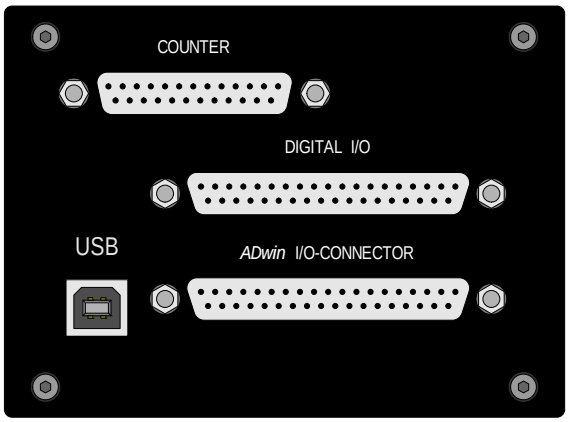
ADwin-light-16 with DIO2 add-on since Rev. B2



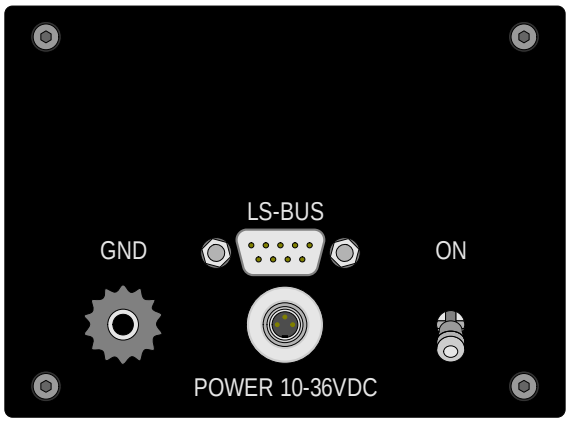
L16-DIO2-PCI



L16-DIO2-EURO

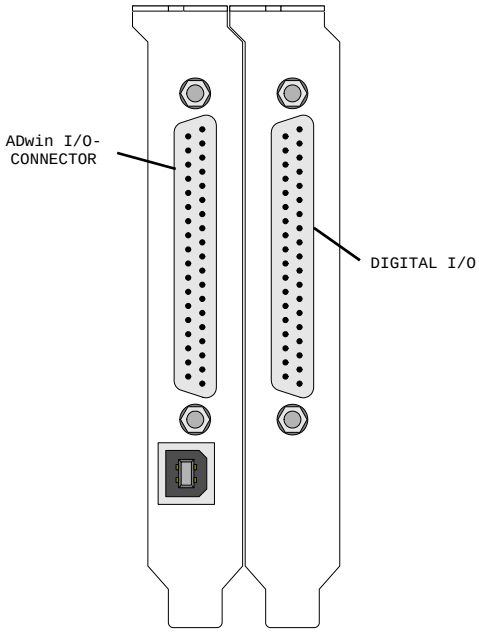


L16-DIO2-EXT: Front side

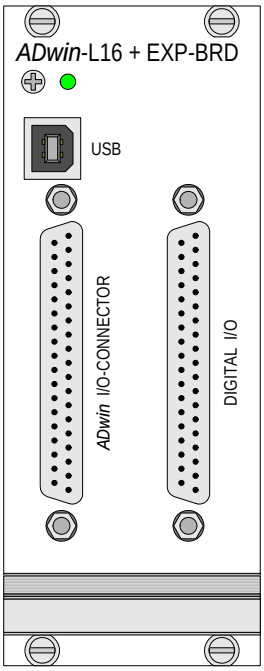


L16-DIO2-EXT: Back side

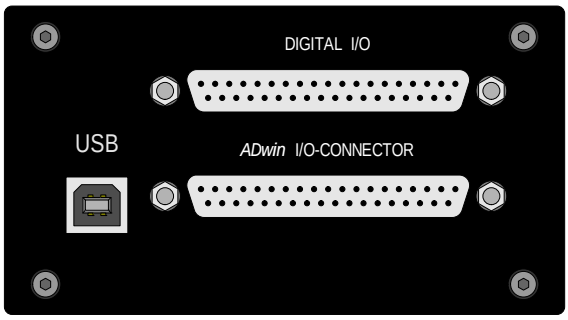
ADwin-light-16 with DIO3 add-on, Rev. B1



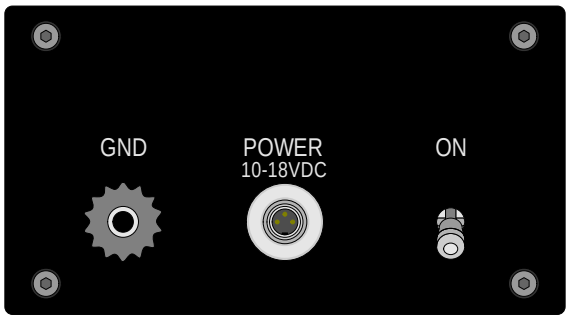
L16-DIO3-PCI



L16-DIO3-EURO

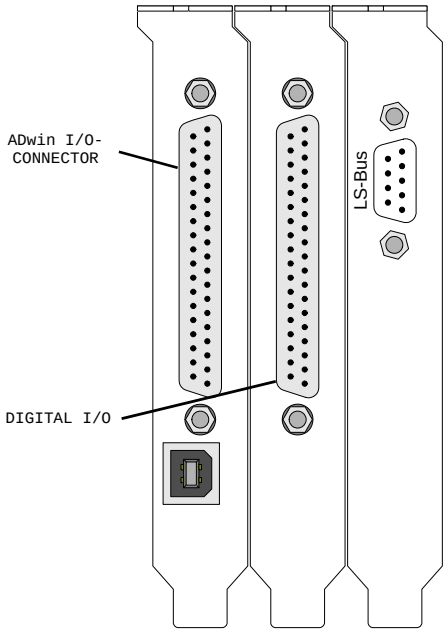


L16-DIO3-EXT: Front side

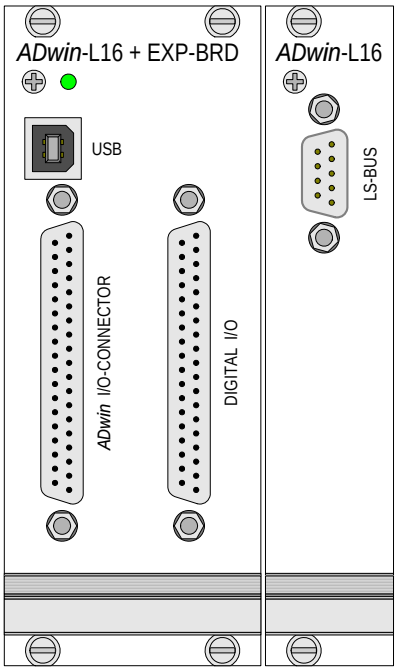


L16-DIO3-EXT: Back side

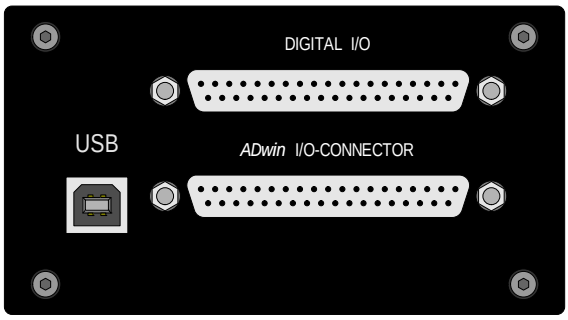
ADwin-light-16 with DIO3 add-on since Rev. B2



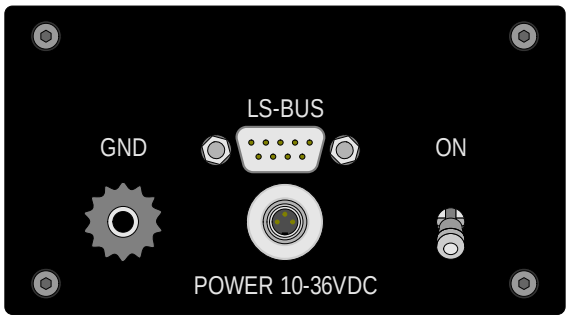
L16-DIO3-PCI



L16-DIO3-EURO



L16-DIO3-EXT: Front side



L16-DIO3-EXT: Back side

A.6 Baud rates for CAN bus

ADwin-light-16-DIO1 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.7 Table of figures

A.8 Index