ADDI-DATA®

Technichal support:
+49 7229 1847-0

CE

Technical description

PA 311

Multifunction board

Edition: 12.05 – 05/2008

Guarantee and responsibility

Basically are effective our "general terms of delivery and payment". The manager receives them at the latest with the invoice. Claims for guarantee and responsibility in case of injuries and material damages are excluded, if they are due to one or some of the following causes:

- if the board has not been used for the intended purpose
- improper installation, operation and maintenance of the board
- if the board has been operated with defective safety devices or with not appropriate or nonfunctioning safety equipment
- nonobservance of the instructions concerning: transport, storage, inserting the board, use, limit values, maintenance, device drivers
- altering the board at the user's own initiative
- altering the source files at the user's own initiative
- not checking properly the parts which are subject to wear
- disasters caused by the intrusion of foreign bodies and by influence beyond the user's control.

Licence for ADDI-DATA software products

Read carefully this licence before using the standard software. The right for using this software is given to the customer, if he/she agrees to the conditions of this licence.

- this software can only be used for configuring ADDI-DATA boards.
- copying the software is forbidden (except for archiving/ saving data and for replacing defective data carriers)
- deassembling, decompiling, decoding and reverse engineering of the software are forbidden.
- this licence and the software can be transfered to a third party, so far as this party has purchased a board, declares to agree to all the clauses of this licence contract and the preceding owner has not kept copies of the software.

Trademarks

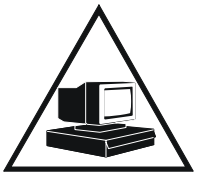
Borland C and Turbo Pascal are registered trademarks of Borland International, INC.
Burr-Brown is a registered trademark of Burr-Brown Corporation
Intel is a registered trademark of Intel Corporation
AT, IBM, ISA and XT are registered trademarks of International Business Machines Corporation
Microsoft, MS-DOS, Visual Basic and Windows are registered trademarks of Microsoft Corporation

The original version of this manual is in German. You can obtain it on request.

WARNING

In case of wrong uses and if the board is not used for the purpose it is intended for:

people may be injured



the board, PC and peripheral may be destroyed

the environment may be polluted

Protect yourself, the others and the environment !

- **Do read the safety leaflet!**
  If this leaflet is not with the documentation, please contact us and ask for it.

- **Observe the instructions of the manual!**
  Make sure that you do not forget or skip any step. We are not liable for damages resulting from a wrong use of the board.

- **Symbols used**

WARNING!
It designates a possibly dangerous situation.
If the instructions are ignored **the board, PC and/or peripheral may be damaged**.

IMPORTANT!
designates hints and other useful information.

- **Any question?**
  Our technical support is at your disposal

1 DEFINITION OF APPLICATION 1
1.1 Intended use 1
1.2 Usage restrictions 1
1.3 General description of the board 1
2 USER 2
2.1 Qualification 2
2.2 Country-specific regulations 2
3 HANDLING THE BOARD 3
4 TECHNICAL DATA 4
4.1 Electromagnetic compatibility (EMC) 4
4.2 Physical set-up of the board 4
4.3 Versions 5
4.4 Options 5
4.5 Limit values 5
5 SETTINGS 9
5.1 Component scheme 9
5.2 Jumper settings 11
5.2.1 Jumper location on the PA 311 and settings at delivery 11
5.2.2 Jumper settings for the different board functions 12
6 INSTALLATION 15
6.1 Base address 16
6.2 Inserting the board 17
6.2.1 Opening the PC 17
6.2.2 Selecting a free slot 17
6.2.3 Inserting the board 18
6.2.4 Closing the PC 18
6.3 Installing the software 19
6.3.1 Software installation under MS-DOS and Windows 3.11 19
6.3.2 Software installation under Windows NT / 95 19
6.4 Board configuration with ADDIREG 20
6.4.1 Program description 20
6.4.2 Registrating a new board 23
6.4.3 Changing the registration of a board 24
6.4.4 Removing the ADDIREG program 24

6.5 Error analysis per Internet 25

7 CONNECTION TO THE PERIPHERAL 26

7.1 Connection principle 26

7.2 Connector pin assignment 26

7.3 Connection examples 27

8 FUNCTIONS OF THE BOARD 28

8.1 Block diagram 28

8.2 I/O mapping 29

8.2.1 16-bit I/O mapping 29

8.2.1 Description of the I/O map 30

WR-MUX register 30

1) UNI/BIP 30

2) GATE0-2 30

3) PR0-3 30

4) GAIN0-3: gain selection (PGA) 31

5) MUX0-3: input selection 32

WR-ADS register (start of conversion) 32

RD-STAT register 33

RD-ADS register 33

8.2.2 8-bit I/O mapping 34

8.3 Analog data acquisition 35

8.3.1 Operating modes 37

Configuring the operating mode 37

1) Signal and ground connection in the single ended mode without INA 37

2) Signal and ground connection in single ended mode with INA 38

3) Signal and ground connection in differential mode 38

8.3.2 Configuring the gain 39

Gain with INA 111 39

Gain with OPA627 40

8.3.3 Conversion 41

Software conversion 41

1) Programming and functioning 42

2) Delayed A/D conversion triggered by software 42

3) Directly triggered software A/D conversion 44

Conversion driven by timer - Automatic conversion 46

1) Timer 46

2) Converting several inputs during a programmable period 47

3) Analog acquisition sequences 48

8.3.4 Options 50

Option SF, DF 50

Option SC, DC 50

Inputs with 4-20mA or 0-20mA current loop signals (option) 51

1) Position of the resistors for current inputs in single-ended mode (option SC) ... 51

2) Position of the resistors for current inputs in differential mode (option DC) 52

3) Mixed mode 52

8.4 Analog output channels 52
8.4.1 Structure of output 0 52
8.4.2 Remarks 53
8.4.3 Analog output channels: programming and functioning 53
14-bit output value 54
1) Calculating the binary code for output signal 54
2) Conversion of the 14-bit value into a 16-bit output value 54
3) Write on output 0 55
16-Bit output value 55
1) Calculating the binary code for output signal 55
2) Write on output 1 55
Relation table for the analog output of a 12-bit value 56

8.5 Parallel input and output channels (TTL) 57
8.5.1 24 inputs and output channels, TTL 57
8.5.2 Interrupt 57
8.5.3 Programming 57
8.5.4 Programming examples 58

8.6 Timer 58
8.6.1 Function 58
8.6.2 Interrupt 59
8.6.3 Programming 59
8.6.4 Programming examples 59
Example in Pascal 60

8.7 DMA 61

8.8 Interrupt 61
8.8.1 End of Conversion Interrupt (EOC) 61
8.8.2 Timer interrupt 61
8.8.3 TTL I/O interrupt through port line PC3 61
8.8.4 Terminal Count Interrupt 61

9 SOFTWARE EXAMPLES 62

9.1 Initialisation 62
a) Flow chart for DOS and Windows 3.11 62
b) Example in C 63
c) Flow chart for Windows NT/95 64
d) Example in C for Windows NT/95 65

9.2 Interrupt 66
a) Flow chart 66
b) Example in C for DOS and Windows 3.11 67
c) Example in C for Windows NT/95 (asynchronous mode) 68
d) Example in C for Windows NT/95 (synchronous mode) 69

9.3 Direct conversion of analog input channels 70
9.3.1 Testing one analog input channel 70
a) Flow chart 70
b) Example in C 71
9.3.2 Testing several analog input channels 72
9.3.2 Testing several analog input channels 72

a) Flow chart....72
b) Example in C....73

9.4 Cyclic conversion of analog input channels....74
9.4.1 Cyclic conversion without DMA and delay....74
a) Flow chart....74
b) Example in C for DOS....75
c) Example in C for Windows 3.11....76
d) Example in C for Windows NT/95 (asynchronous mode)....77
e) Example in C for Windows NT/95 (synchronous mode)....78
9.4.2 Cyclic conversion with DMA without delay....79
a) Flow chart....79
b) Example in C for DOS....80
c) Example in C for Windows 3.11....81
d) Example in C for Windows NT/95 (asynchronous mode)....82
e) Example in C for Windows NT/95 (synchronous mode)....83

9.5 Analog output channels....84
9.5.1 Testing one analog output channel....84
a) Flow chart....84
b) example in C....85
9.5.2 Testing several output channel....86
a) Flow chart....86
b) example in C....87

9.6 Timer....88
a) Flow chart....88
b) Example in C for DOS....89
c) Example in C for Wiydow 3.1x....90
d) Example in C for Window NT/95 (asynchronous mode)....91
e) Example in C for Window NT/95 (synchronous mode)....92

9.7 PIO....93
9.7.1 PIO input channels....93
a) Flow chart....93
b) Example in C....94
9.7.2 PIO output channels....95
a) Flow chart....95
b) Example in C....96

INDEX....A

Figures

Fig. 3-1: Wrong handling 3
Fig. 3-2: Correct handling 3
Fig. 5-1: Component scheme (Left side) 9
Fig. 5-2: Component scheme (right side) 10
Fig. 5-3: Jumper location and settings at delivery 11
Fig. 5-4: Terminal count and DMA line: J6, J8 and J9 12
Fig. 5-5: Interrupt lines and sources: J7 13
Fig. 5-6: Jumper J8: Delay 13
Fig. 5-7: Selection of the conversion start: J23 14
Fig. 5-8: Signal setting of the timer: J28 14
Fig. 6-1: DIP switches 16
Fig. 6-2: Slot types 17
Fig. 6-4: Inserting the board 18
Fig. 6-5: Securing the board at the back cover 18
Fig. 6-6: ADDIREG registration program 20
Fig. 6-7: Configuring a new board 22
Fig. 7-1: Connection principle 26
Fig. 7-2: 37-pin SUB-D male connector 26
Fig. 7-3: Pin header X22 connected to ribbon cable FB 311: 26-pin to 37-pin connector 27
Fig. 7-4: Connection to screw terminal panels PX 901-ZG and PX 901-A 27
Fig. 8-1: Block diagram of the PA 311 28
Fig. 8-2: Analog data acquisition 36
Fig. 8-3: Single ended without INA 37
Fig. 8-4: Single ended with INA 38
Fig. 8-5: Differential with INA (PGA) 39
Fig. 8-6: Programmable amplifier 40
Fig. 8-7: Delayed A/D conversion 43
Fig. 8-8: Time diagram of conversion 47
Fig. 8-9: Time diagram of conversion 48
Fig. 8-10: Board PA 311 SF (single ended with input filter) 50
Fig. 8-11: Board PA 311 DF (differential with input filter) 50
Fig. 8-12: Board PA 311 SC (single ended with current converter) 50
Fig. 8-13: Board PA 311 DC (differential with current converter) 51
Fig. 8-14: Structure of output 0 52
Fig. 8-15: Connection of output 0 53
Fig. 8-16: Timer setting for the generation of interrupts 58

Tables

Table 5-1: Analog data acquisition: J2, J3, J4 and J5 12
Table 5-2: Output range 13
Table 5-3: Setting the delay 13
Table 6-1: Decoding table 16
Table 8-1: I/O map for 16-bit accesses 29
Table 8-2: I/O map for 8-bit accesses 34
Table 8-3: Input voltage ranges for different gain factors 40

1 DEFINITION OF APPLICATION

1.1 Intended use

The **PA 311** board must be inserted in a PC with ISA slots which is used as electrical equipment for measurement, control and laboratory pursuant to the norm EN 61010-1 (IEC 61010-1). The used personal computer (PC) must fulfil the requirements of IEC 60950-1 or EN 60950-1 and 55022 or IEC/CISPR 22 and EN 55024 or IEC/CISPR 24.

The use of the board **PA 311** in combination with external screw terminal panels requires correct installation according to IEC 60439-1 or EN 60439-1 (switch cabinet / switch box).

1.2 Usage restrictions

The **PA 311** board must not to be used as safety related part (SRP).

The board must not be used for safety related functions, for example for emergency stop functions.

The **PA 311** board must not be used in potentially explosive atmospheres.

The **PA 311** board must not be used as electrical equipment according to the Low Voltage Directive 2006/95/EC.

1.3 General description of the board

Data exchange between the board **PA 311** and the peripheral is to occur through a shielded cable, which has to be connected to the 37-pin SUB-D male connector of the board **PA 311**.

The board has up to 8 differential or 16 Single-Ended input channels intended for processing analog signals. The screw terminal panel **PX 901** allows to connect the analog signals through a shielded cable. The use of the board in combination with external screw terminal panels is to occur in a closed switch cabinet; the installation is to be effected competently.

The connection with our standard cable **ST010** complies with the specifications:
- metallized plastic hoods
- shielded cable
- cable shield folded back and firmly screwed to the connector housing

Uses beyond these specifications are not allowed. The manufacturer is not liable for any damages which would result from the non-observance of this clause.

The use of the board according to its intended purpose includes observing all advices given in this manual and in the safety leaflet.

Our boards are not to be used for securing emergency stop functions.

The emergency stop functions are to be secured separately. This securing must not be influenced by the board or the PC.

Make sure that the board remains in its protective blister pack **until it is used**.

Do not remove or alter the identification numbers of the board. If you do, the guarantee expires.

2 USER

2.1 Qualification

Only persons trained in electronics are entitled to perform the following:

- installation
- use
- maintenance

2.2 Country-specific regulations

Consider the country-specific regulations about

- the prevention of accidents
- electrical and mechanical installations
- radio interference suppression

3 HANDLING THE BOARD

Fig. 3-1: Wrong handling

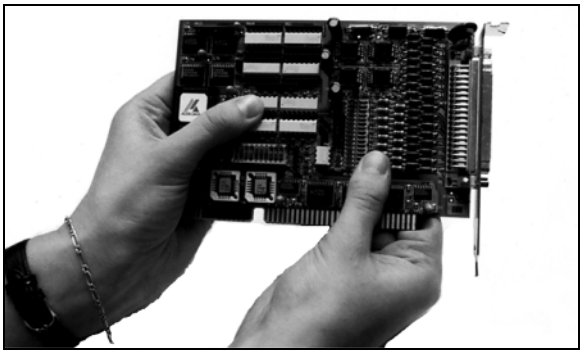
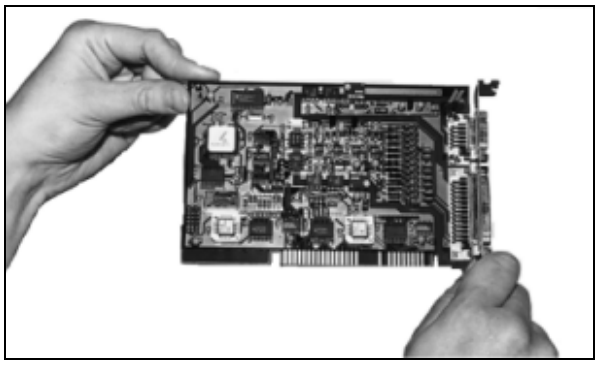


Fig. 3-2: Correct handling



4 TECHNICAL DATA

4.1 Electromagnetic compatibility (EMC)

The board **PA 311** complies with the European EMC directive. The tests were carried out by a certified EMC laboratory in accordance with the norm from the EN 61326 series (IEC 61326). The limit values as set out by the European EMC directive for an industrial environment are complied with.

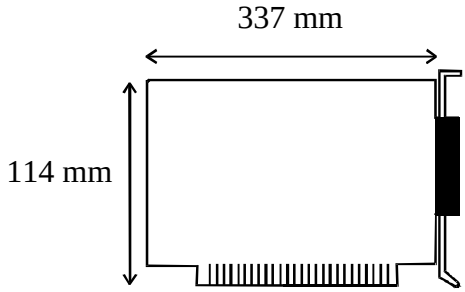
The respective EMC test report is available on request.

WARNING!
The EMC tests have been carried out in a specific appliance configuration. We guarantee these limit values **only** in this configuration[1].

Consider the following aspects:
- your test program must be able to detect operation errors.
- Your system mut be set up so that you can find out what caused errors.

4.2 Physical set-up of the board

The board is assembled on a 4-layer printed circuit card.



Weight: 330 g

Installation in: AT slot

Connection to the peripheral..... 37-pin SUB-D male connector

Cables: ..... **ST010**, **ST011**, **FB311**: ribbon cable

Screw terminal panels:..... **PX 901-A(G)** **PX 901-ZG** for dig. I/O

See Fig. 7-3: Connection principle

WARNING!
The supply lines must be installed safely against mechanical loads.

[1] We transmit our appliance configuration on request

4.3 Versions

The board is available in 3 versions:

Version	Inputs	outputs
PA311-8-2	8 SE / 4 Diff.	2
PA311-8-4	8 SE / 4 Diff.	4
PA311-8-8	16 SE / 8 Diff.	8
PA311-16-2	16 SE / 8 Diff.	2
PA311-16-4	16 SE / 8 Diff.	4
PA311-16-8	16 SE / 8 Diff.	8

4.4 Options

Option SF: Filter for 16 SE input channels
Option DF: Precision filter for 8 differential input channels
Option SC: Current input channels 0-20 mA or 4-20 mA for 16 SE input channels.
Option DC: Current input channels 0-20 mA or 4-20 mA for 8 differential input channels.

IMPORTANT!
By 4-20mA the precision is altered.

4.5 Limit values

Operating temperature: ..... 0 to 60°C
Storage temperature: ..... -25 to 70°C
Relative humidity at indoor installation
50% at +40 °C
80% at +31 °C

Energy requirements
- operating voltage of the PC: ..... 5V ± 5%
- current consumption in mA(without load)
  typically: ..... see table below ± 10%

	PA 311-8-2	PA 311-8-4	PA 311-8-8	PA 311-16-2	PA 311-16-4	PA 311-16-8
+ 5 V from the PC	750 mA	1014 mA	1540	840 mA	1104 mA	1630 mA

Analog input channels:

Number of analog input channels: ..... 16 SE / 8 diff. for **PA 311-16x**
Analog resolution: ..... 16-bit, 1 among 65535
Data transfer rate (1 input): ..... 100 kHz
Data transfer: ..... Data to the PC (16-bit only)
1) through I/O commands
2) Interrupt at EOC[1]
3) DMA transfer at EOC
Start of conversion: ..... 1) per software trigger
2) TIMER 0 of 82C54
3) Sequences through TIMER 0 & 1
Monotony: 14-bit
Offset error: ..... after calibration:
- Bipolar: ± 2 LSB
- Unipolar: ± 2 LSB
Drift (0°C to 70°C):
- Bipolar: ± 2 ppm / °C
- Unipolar: ± 2 ppm / °C
Gain error: ..... after calibration:
- Bipolar: ± 1 ½ LSB
- Unipolar: ± 1 ½ LSB
Drift (0°C to 70°C):
- Bipolar: ± 7 ppm / °C
- Unipolar: ± 7 ppm / °C
Analog input ranges: ..... Voltage
- Unipolar: 0-10 V
- Bipolar: ±10 V
Software programmable
Current
- Unipolar: 0-20 mA
Unipolar mode and gain x2 must be selected
Overvoltage protection: ..... ± 20 V at Power On
Common mode rejection: ..... DC up to 60 Hz, 90 dB minimum
Bandwidth (-3dB): ..... Limited to 159 kHz (-3 dB) with low-pass filter 1st order; but the minimum SINAD is still 83 dB at 45 kHz (fin)
Bias currents for the input channels: ..... ± 2 nA max.
Input impedance: ..... $10^{12}\,\Omega$ // 10 nF Single-Ended
$10^{12}\,\Omega$ // 20 nF diff. against GND
Integral non-linearity (INL): ..... ± 1/2 LSB
Differential non-linearity (DNL): ..... ± 1/2 LSB

[1] EOC: End of Conversion

Selectable gain: ..... free through resistors with INA 111

via PGA gain 1, 2, 10 (software programmable for each channel)

System noise: ..... Bipolar: Gain x1: ± 1 LSB
Gain x2: ± 2 LSB
Gain x10: ± 3 LSB
Unipolar: Gain x1: ± 1 LSB
Gain x2: ± 2 LSB
Gain x10: ± 3 LSB

Digital coding: ..... 2's complement

Analog output channels

Overvoltage protection: ..... ±12V

Number of outputs: ..... 2, 4 or 8

Type of DAC: ..... Monolithic, multiplying DAC PORT

Data transfer: ..... The board is located in the I/O address space of the PC. The values are written on the board with 16-bit accesses.
The outputs are updated with a dummy reading on an address (simultaneous updating of the outputs).

Settling time at 0.0008 % FS,
(FS = Full scale)
with 2KΩ & 1000pF load: ..... 8 µs typ. for a 20V voltage jump at 25°C
10 µs typ. for a 20V voltage jump above the temperature range
6 µs typ. for a 10V voltage jump at 25°C
8 µs typ. for a 10V voltage jump above the temperature range
2.5 µs typ. for a 10 V voltage jump above the temperature range

Output voltage ranges: ..... Unipolar: 0-10 V
Bipolar: ± 10 V

Digital coding: ..... Unipolar: Straight binary coding
Bipolar: Offset binary coding

Output current: ..... ± 5 mA maximum

Capacitive load: ..... 1000 pF maximum

Short circuit current: ..... must be limited externally to ± 25 mA

Integral non-linearity (INL): ..... ± 1/2 LSB maximum at 25°C
± 1 LSB maximum above the temperature range

Differential non-linearity (DNL): ..... ± 1/2 LSB maximum at 25°C
± 1 LSB maximum above the temperature range

Monotony: ..... 14-bit minimum

Offset error:	after calibration ± 20 µV (max.) Drift (0°C to 70°C): ±2 ppm / °C Unipolar (max.) ±5 ppm / °C Bipolar (max.)
Gain error:	After calibration ± 1 LSB (max.) Drift (0°C to 70°C): ±18 ppm / °C (max.)

Digital I/O

Compatibility:	TTL-compatible
Configuration:	Three 8-bit ports (use 82C55PPI)
Input at logic "0":	0.8 V maximum
Input at logic "1":	2 V minimum
Input load current $0 \leq V_{in} \leq 5$ V:	±10 µA maximum
Output at logic "0"	
when output current = 1.7 mA:	0.45 V maximun
Output at logic "1"	
when output current = -200 µA:	2.4 V minimum
Darlington drive current (ports B and C only)	
R_{EXT} = 750Ω; V_{EXT} = 1.5 V:	1.0 mA minimum 4.0mA maximum

Timer I/O

Configuration:	Three 16-bit timers (82C54)
Compatibility:	TTL-compatible inputs and outputs; Counter gate is provided with a 100 kΩ pull up resistor.
Input at logic "0":	0.8 V maximum
Input at logic "1":	2.2 V minimum
Output at logic "0"	
for output current = 1.6 mA:	0.45 V maximum
Output at logic "1"	
for output current = -150 µA:	2.4 V minimum
Input capacity at 1 MHz:	10 pF maximum
Available input frequencies:	892.857 kHz 27.901 kHz

5 SETTINGS

5.1 Component scheme

Fig. 5-1: Component scheme (Left side)

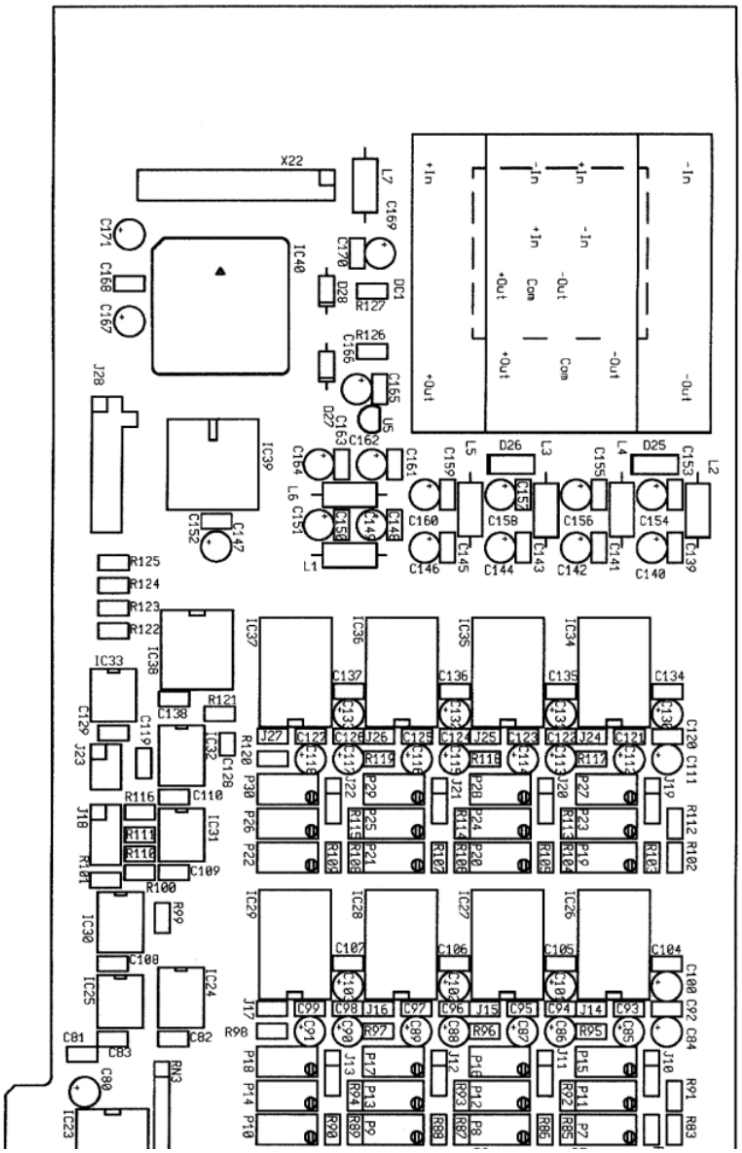
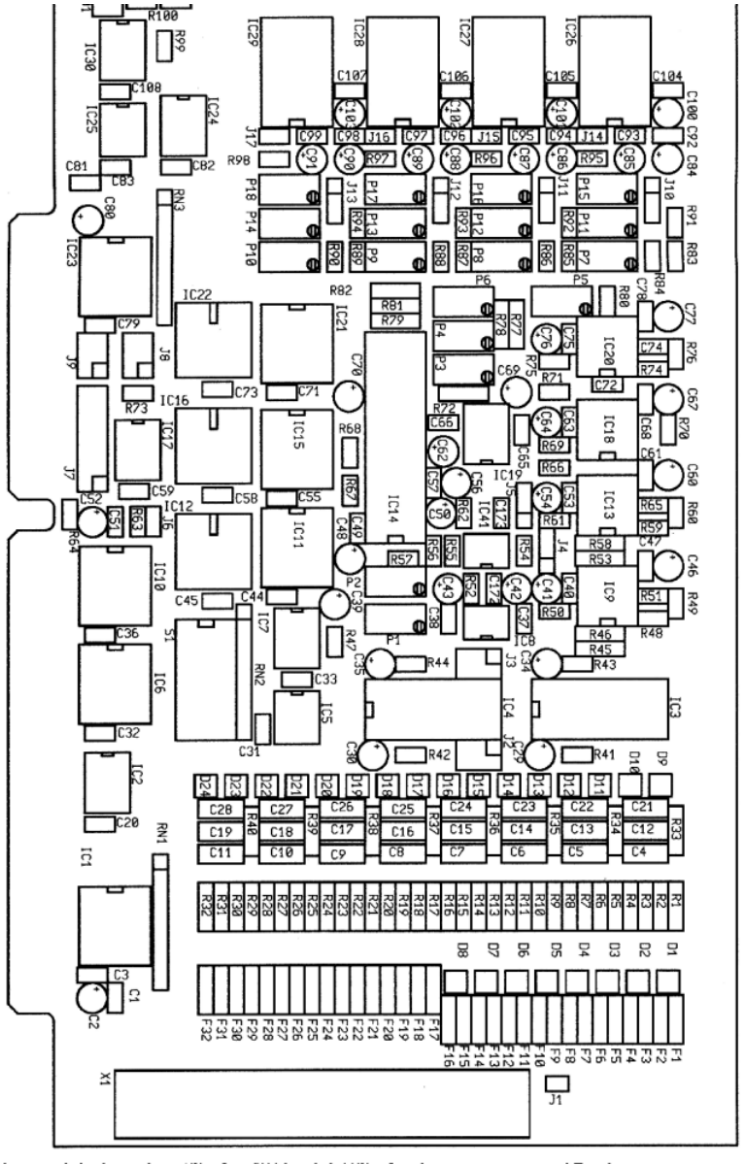


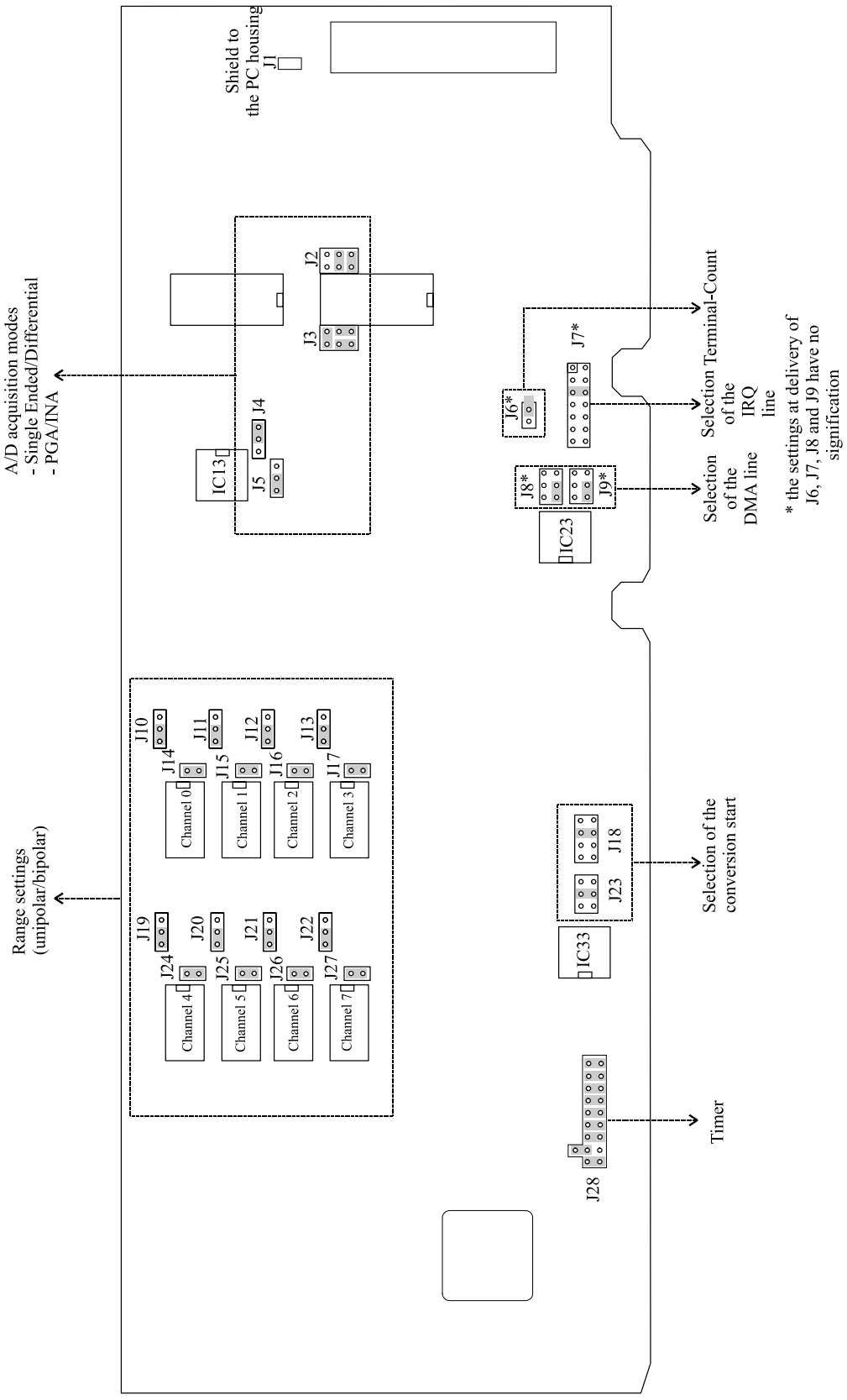
Fig. 5-2: Component scheme (right side)



5.2 Jumper settings

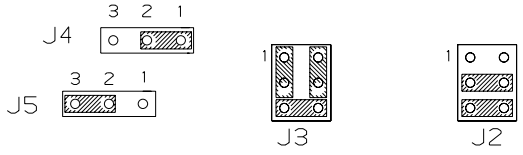
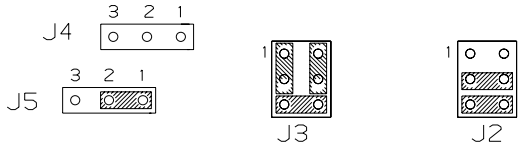
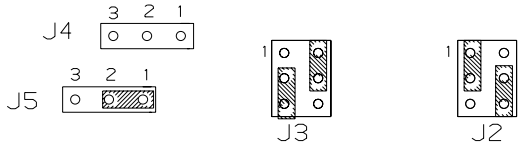
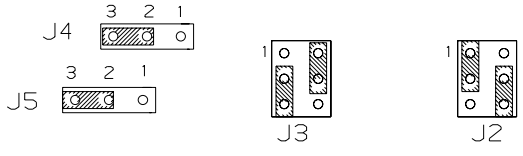
5.2.1 Jumper location on the PA 311 and settings at delivery

Fig. 5-3: Jumper location and settings at delivery



5.2.2 Jumper settings for the different board functions

Table 5-1: Analog data acquisition: J2, J3, J4 and J5

SINGLE ENDED WITHOUT INA WITH PGA	
SINGLE ENDED WITH INA WITHOUT PGA	
DIFFERENTIAL WITH INA WITHOUT PGA	
DIFFERENTIAL WITH INA + PGA	

Setting at delivery: Single-Ended output signal without INA with PGA

Fig. 5-4: Terminal count and DMA line: J6, J8 and J9

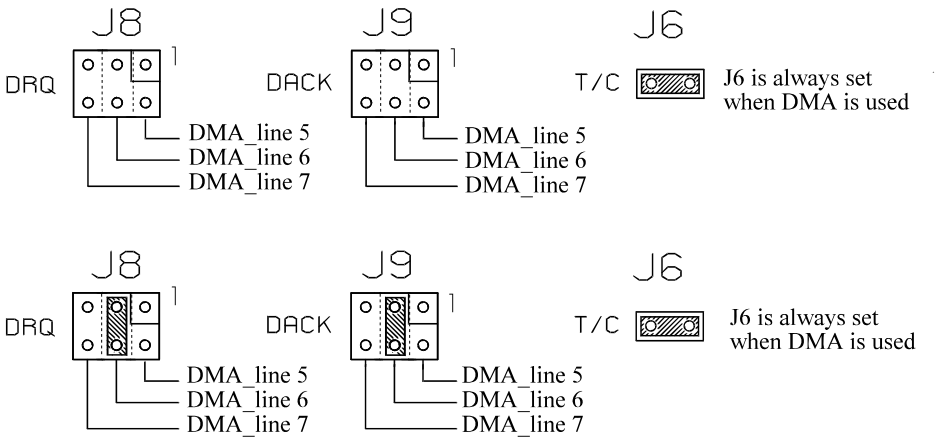


Fig. 5-5: Interrupt lines and sources: J7

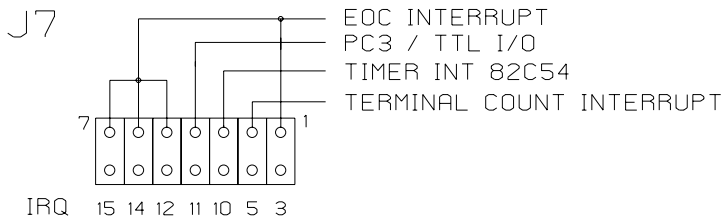


Table 5-2: Output range

The output channels can be individually configured. Proceed in the same way for all output channels as in the example below. (output 0)

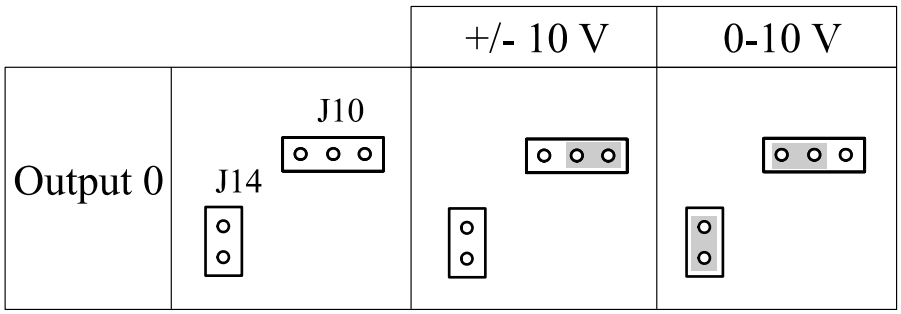


Fig. 5-6: Jumper J8: Delay

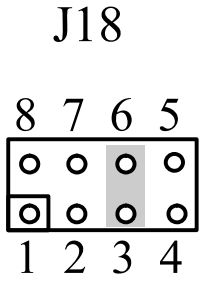


Table 5-3: Setting the delay

Position	Delay	Rq
J18-1	15 µs	< 100R
J18-2	100 µs	< 1K
J18-3	1 ms	< 10K
J18-4	10 ms	< 1M

Rq = internal resistance of the source.
Several jumpers can be set simultaneously. In this case the times are added up.

For example, if jumpers are set to J18-1 and J18-2, a delay of 115 µs is reached. Conversion time lasts 10 µs. It results a global time of 125 µs and thus a throughput of 8000 Hz.

Fig. 5-7: Selection of the conversion start: J23

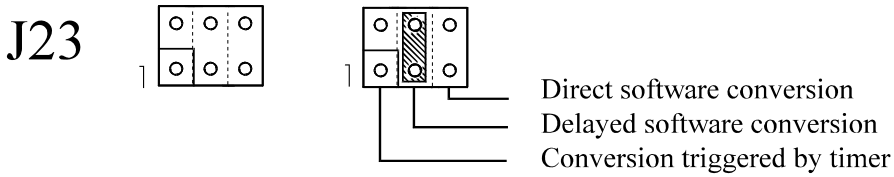
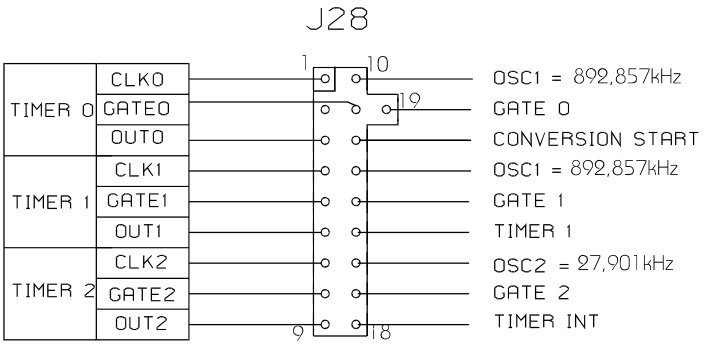


Fig. 5-8: Signal setting of the timer: J28



6 INSTALLATION

IMPORTANT!
If you want to install simultaneously **several** ADDI-DATA boards, consider the following procedure.

- **Install and configure** the boards one after the other.
  You will thus avoid configuration errors.

1. Switch off the PC
2. Install the **first** board
3. Start the PC
4. Install the software (once is enough)
5. Configure the board

6. Switch off the PC
7. Install the **second** board
8. Start the PC
9. Configure the board

etc

You will find additional information to these different steps in the sections 6.1 to 6.5.

IMPORTANT!
You have installed already **one or more** ADDI-DATA boards in your PC, and you wish to install **an additional** board?

Proceed as if you wished to install one single board.

6.1 Base address

WARNING!
If the base address set is wrong, the board and/or the PC may be destroyed.

Before installing the board

The base address is set at delivery on the address 0300H.

- **Check**, that
  - the base address is free
  - the address range required by the board is not already used by the PC or by boards already installed in the PC.

The board occupuies at the most 28 addresses within the I/O address space of the PC.

PA 311-8-2	PA 311-16-2	16 I/O addresses occupied
PA 311-8-4	PA 311-16-4	20 I/O addresses occupied
PA 311-8-8	PA 311-16-8	28 I/O addresses occupied

If the base address and/or the address range **are wrong,**
• **select** another base address with the 8-pin block of DIP switches S1

The address 0300H is decoded in the following figure.

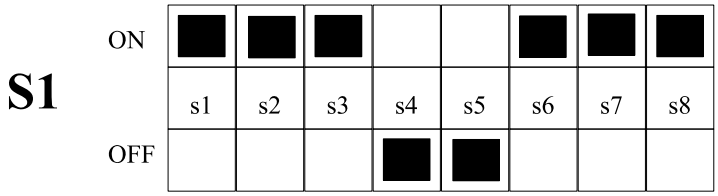
Table 6-1: Decoding table

	MSB															LSB
Decoded address bus	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
Hex base address to be set	0				3				0				0			
Binary base address to be set	0	0	0	0	0	0	1	1	1	0	0	1	0	0	0	0
DIP switches S1 Logic "0"= ON Logic "1" = OFF	*	*	*	s8 ON	s7 ON	s6 ON	s5 OFF	s4 OFF	s3 ON	s2 ON	s1 ON	X	X	X	X	X

X: Decoded address range of the board
* : permanently decoded on logic "0"

Fig. 6-1: DIP switches

IMPORTANT!
You will find the switch **s1 on the left** of the DIP switches!



6.2 Inserting the board

IMPORTANT!
Please follow absolutely the *safety instructions.*

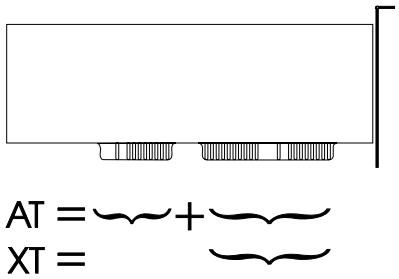
6.2.1 Opening the PC

- Switch off your PC and all the units connected to the PC.
- Pull the PC mains plug from the socket.
- Open your PC as described in the manual of the PC manufacturer.

6.2.2 Selecting a free slot

1. Select a free AT slot

Fig. 6-2: Slot types



The board can also be used in EISA slots.

2. Remove the back cover of the selected slot
according to the instructions of the PC manufacturer.
Keep the back cover. You will need it if you remove the board.

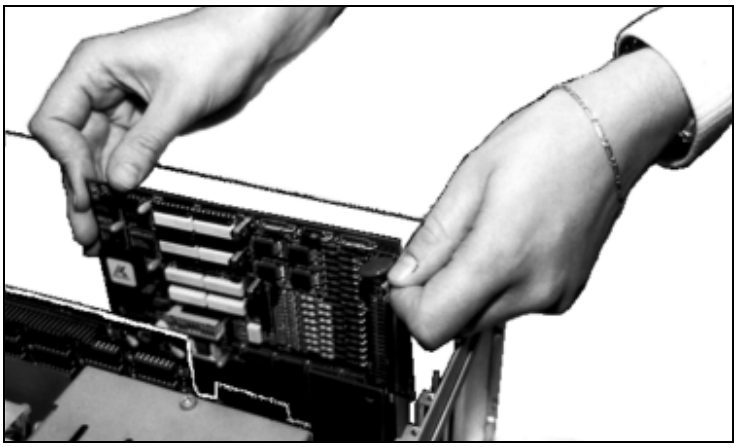
3. Discharge yourself from electrostatic charges.

4. Take the board from its protective pack.

6.2.3 Inserting the board

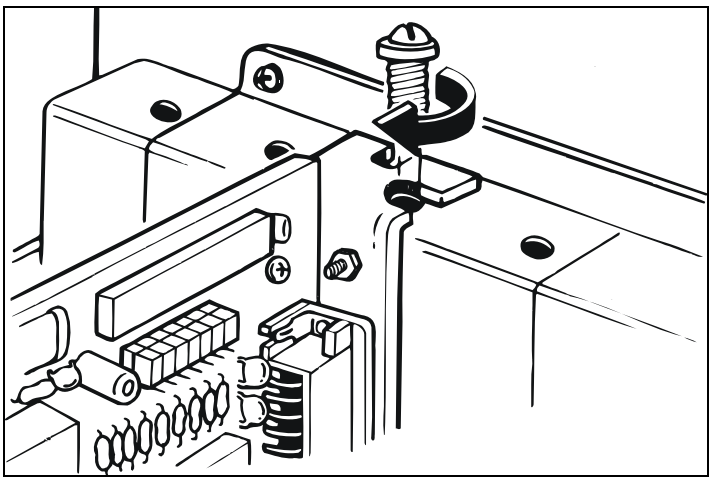
- **Discharge yourself from electrostatic charges.**
- Insert the board **vertically into the chosen slot.**

Fig. 6-4: Inserting the board



- **Secure the board** to the rear of the PC housing with the screw which was fixed on the back cover.

Fig. 6-5: Securing the board at the back cover



- **Tighten all the loosed screws.**

6.2.4 Closing the PC

- **Close** your PC as described in the manual of the PC manufacturer.

6.3 Installing the software

The board is supplied with a CD-ROM.

The CD contains:
- ADDIREG for Windows NT 4.0 and Windows 95,
  You can also download the ADDIREG program from Internet.
- Standard software for the ADDI-DATA boards:
  - 16-bit for MS-DOS and Windows 3.11
  - 32-bit for Windows NT/95.

6.3.1 Software installation under MS-DOS and Windows 3.11

- Copy the contents of PA311\16bit on a diskette.
  If several diskettes are to be used, the directory content is stored in several sub-directories (Disk1, Disk2, Disk3...).
- Insert the (first) diskette into a driver and change to this drive.
- Enter <INSTALL>.

The installation program gives you further instructions.

6.3.2 Software installation under Windows NT / 95

- Select the directory PA311\32bit\Disk1.
- Start the setup program "setup.exe" (double click)
- Select one of the 3 parameters
  1- typical
  2- compact
  3- custom

Proceed as indicated on the screen and read attentively the "Software License" and "Readme".
In "custom", you can select your operating system.

The installation program gives you further instructions.

6.4 Board configuration with ADDIREG

The ADDIREG registration program is a 32-bit program for Windows NT 4.0/ 95. The user can registrate all hardware informations necessary to operate the ADDI-DATA PC boards.

IMPORTANT!
If you use one or several resources of the board, you cannot start the ADDIREG program.

6.4.1 Program description

IMPORTANT!
Insert the ADDI-DATA boards to be registrated before starting the ADDIREG program.

If the board is not inserted, the user cannot test the registration.
Once the program is called up, the following dialog box appears.

Fig. 6-6: ADDIREG registration program

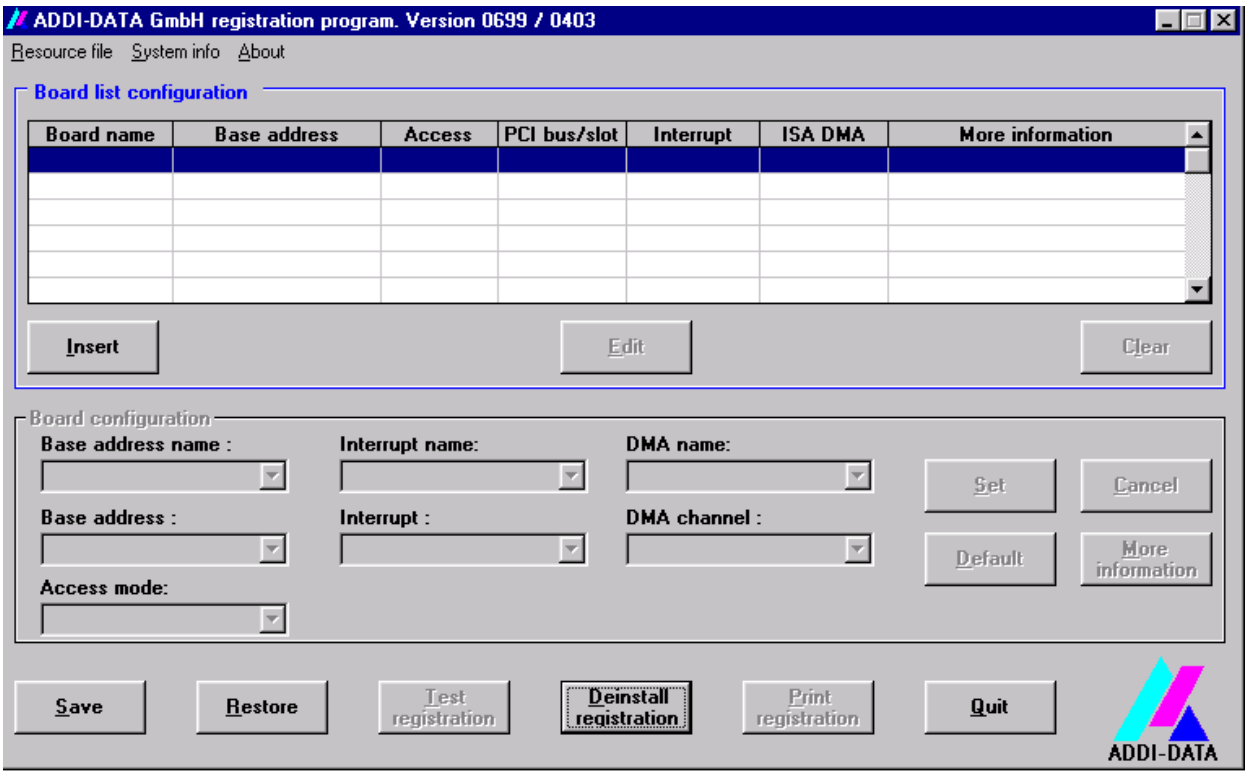


Table:

The table in the middle lists the registrated boards and their respective parameters.

Board name:
Names of the different registrated boards (eg.: APCI-3120).
When you start the program for the first time, no board is registrated in this table.

Base address:
Selected base address of the board.

IMPORTANT!
The base address selected with the ADDIREG program must correspond to the one set through DIP-switches.

Access:
Selection of the access mode for the ADDI-DATA digital boards.
Access in 8-bit or 16-bit.

PCI bus / slot:
Used PCI slot. If the board is no PCI board, the message "NO" is displayed.

Interrupt:
Used interrupt of the board. If the board uses no interrupt, the message "Not available" is displayed.

IMPORTANT!
The interrupt selected with the ADDIREG program must correspond to the one set through DIP-switches.

More information:
Additional information like the identifier string (eg.: PCI1500-50) or the installed COM interfaces.

Text boxes:

Under the table you will find 6 text boxes in which you can change the parameters of the board.

Base address name:
When the board operates with several base addresses (One for port 1, one for port 2, etc.) you can select which base address is to be changed.

Base address:
In this box you can select the base addresses of your PC board. The free base addresses are listed. The used base addresses do not appear in this box.

Interrupt name:
When the board must support different interrupt lines (common or single interrupts), you can select them in this box.

Interrupt:
Selection of the interrupt number which the board has to use.

DMA name:
When the board supports 2 DMA channels, you can select which DMA channel is to be changed.

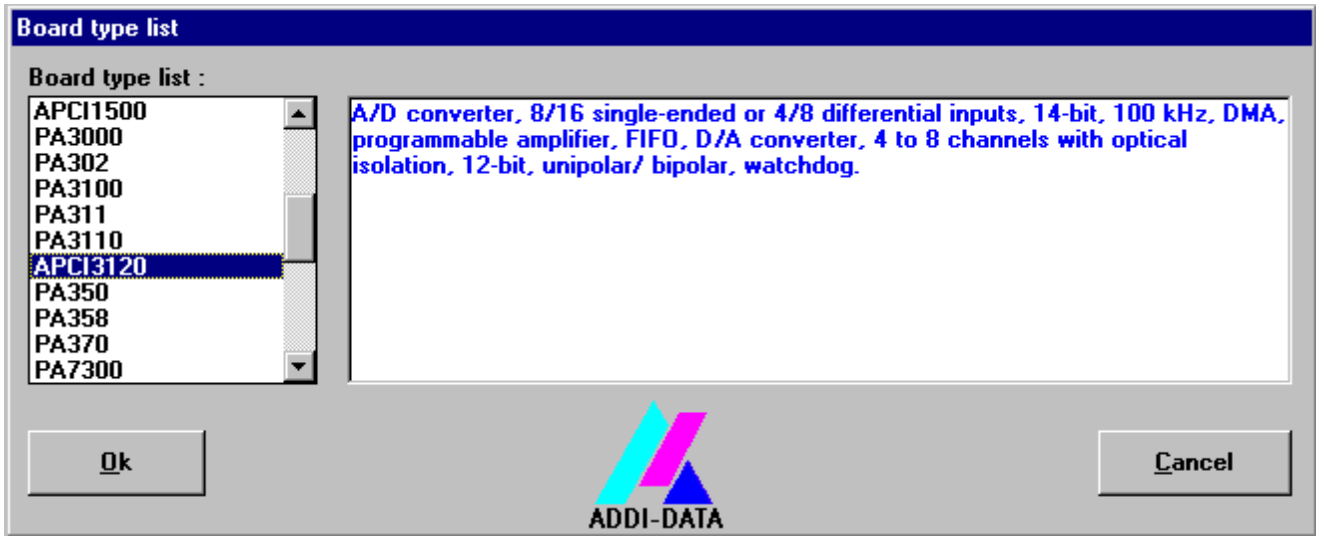
DMA channel:
Selection of the used DMA channel.

Buttons:

Edit [1]**:**
Selection of the highlighted board with the different parameters set in the text boxes. Click on "Edit" to activate the data or click twice on the selected board.

Insert:
When you want to insert a new board, click on "Insert". The following dialog window appears:

Fig. 6-7: Configuring a new board



All boards you can registrate are listed on the left. Select the wished board. (The corresponding line is highlighted).
On the right you can read technical informations about the board(s).
Activate with "OK"; You come back to the former screen.

Clear:
You can delete the registration of a board. Select the board to be deleted and click on "Clear".

Set:
Sets the parametered board configuration. The configuration should be set before you save it.

Cancel:
Reactivates the former parameters of the saved configuration.

Default:
Sets the standard parameters of the board.
More information:
You can change the board specific parameters like the identfier string, the COM number, the operating mode of a communication board, etc...
If your board does not support these informations, you cannot activate this button.

1 "x": Keyboard shortcuts; e.g. "Alt + e" for Edit

Save:
Saves the parameters and registrates the board.

Restore:
Reactivates the last saved parameters and registration.

Test registration:
Controls if there is a conflict between the board and other devices.
A message indicates the parameter which has generated the conflict. If there is no conflict, "OK" is displayed.

Deinstall registration:
Deinstalls the registrations of all board listed in the table.

Print registration:
Prints the registration parameter on your standard printer.

Quit:
Quits the ADDIREG program.

6.4.2 Registrating a new board

IMPORTANT!
To registrate a new board, you must have administrator rights.
Only an administrator is allowed to registrate a new board or change a registration.

- Call up the ADDIREG program. The figure 6-6 is displayed on the screen. Click on „Insert“. Select the wished board.

- Click on „OK“. The default address, interrupt, and the other parameters are automatically set and listed in the lower fields.
  If the parameters are not automatically set by the BIOS, you can change them.
  Click on the wished scroll function(s) and select a new value.
  Activate your selection with a click.

- Once the wished configuration is set, click on „Set“.

- Save the configuration with „Save“.

- You can test if the registration is „OK“.
  This test controls if the registration is right and if the board is present.
  If the test has been successfully completed you can quit the ADDIREG program.
  The board is initialised with the set parameters and can now be operated.

  In case the registration data is to be modified, it is necessary to boot your PC again. A message asks you to do so. When it is not necessary you can quit the ADDIREG program and directly begin with your application.

6.4.3 Changing the registration of a board

IMPORTANT!
To registrate a new board, you must have administrator rights.
Only an administrator is allowed to registrate a new board or change a registration.

- Call up the ADDIREG program. Select the board to be changed.
  The board parameters (Base address, DMA channel, ..) are listed in the lower fields.
- Click on the parameter(s) you want to set and open the scroll function(s).
- Select a new value. Activate it with a click.
  Repeat the operation for each parameter to be modified.
- Once the wished configuration is set, click on „Set“.
- Save the configuration with „Save“.
- You can test if the registration is „OK“.
  This test controls if the registration is right and if the board is present.
  If the test has been successfully completed you can quit the ADDIREG program.
  The board is initialised with the set parameters and can now be operated.
  In case the registration data is to be modified, it is necessary to boot your PC again. A message asks you to do so. When it is not necessary you can quit the ADDIREG program and directly begin with your application.

6.4.4 Removing the ADDIREG program

The ADDI_UNINSTAL program is delivered on the CD-ROM.

- Install the ADDI_UNINSTAL program on your computer.
- Start the ADDIREG program and click on "Deinstall registration"
- Quit ADDIREG
- Start the ADDI_UNINSTAL program
- Proceed as indicated until the complete removing of ADDIREG.

You can also download the programm from Internet.

6.5 Error analysis per Internet

Do not hesitate to visit us or e-mail your questions.
Our Internet page is accessed:
- per e-mail: info@addi-data.de
- per Internet : http://www.addi-data.com

Free driver download

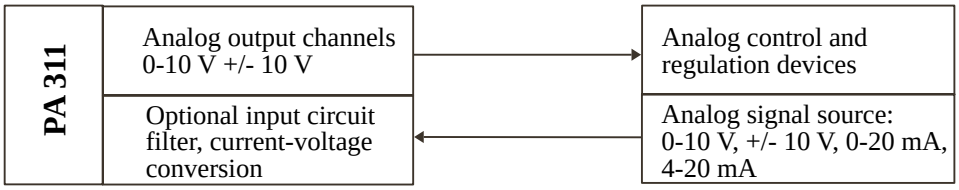
You can download the latest version of the standard sotware for the board **PA 311**.

IMPORTANT!
Before using the board or in case of malfunction during operation, check if there is an update of the product (technical description, driver). The current version can be found on the internet or contact us directly.

7 CONNECTION TO THE PERIPHERAL

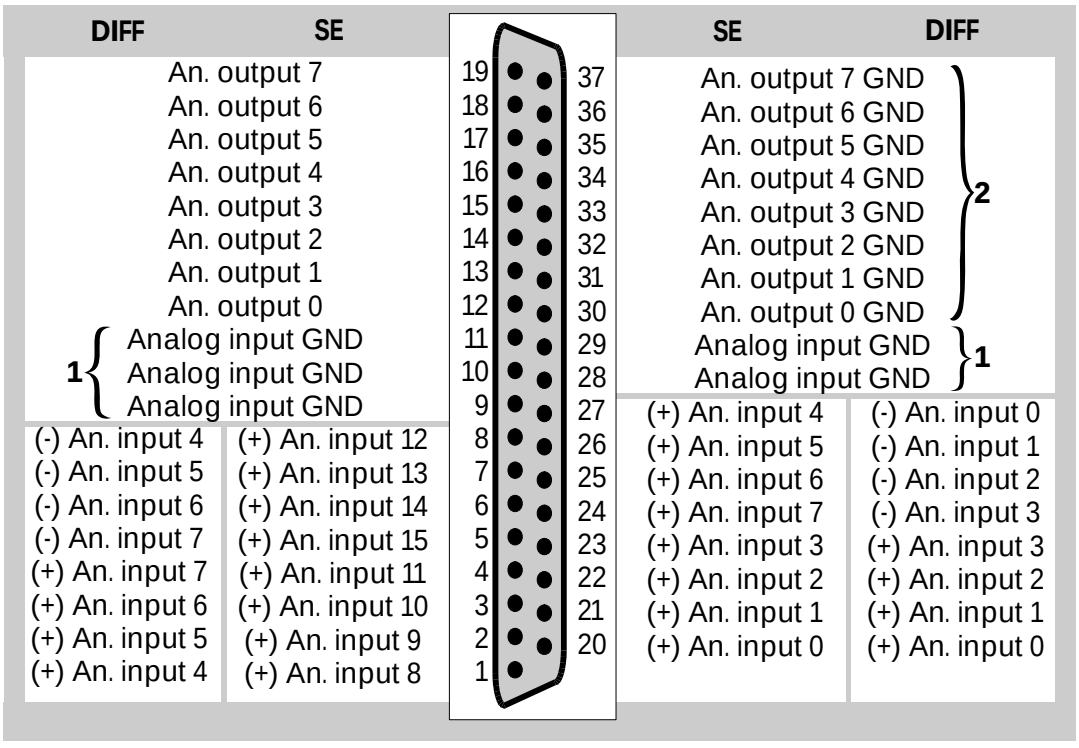
7.1 Connection principle

Fig. 7-1: Connection principle



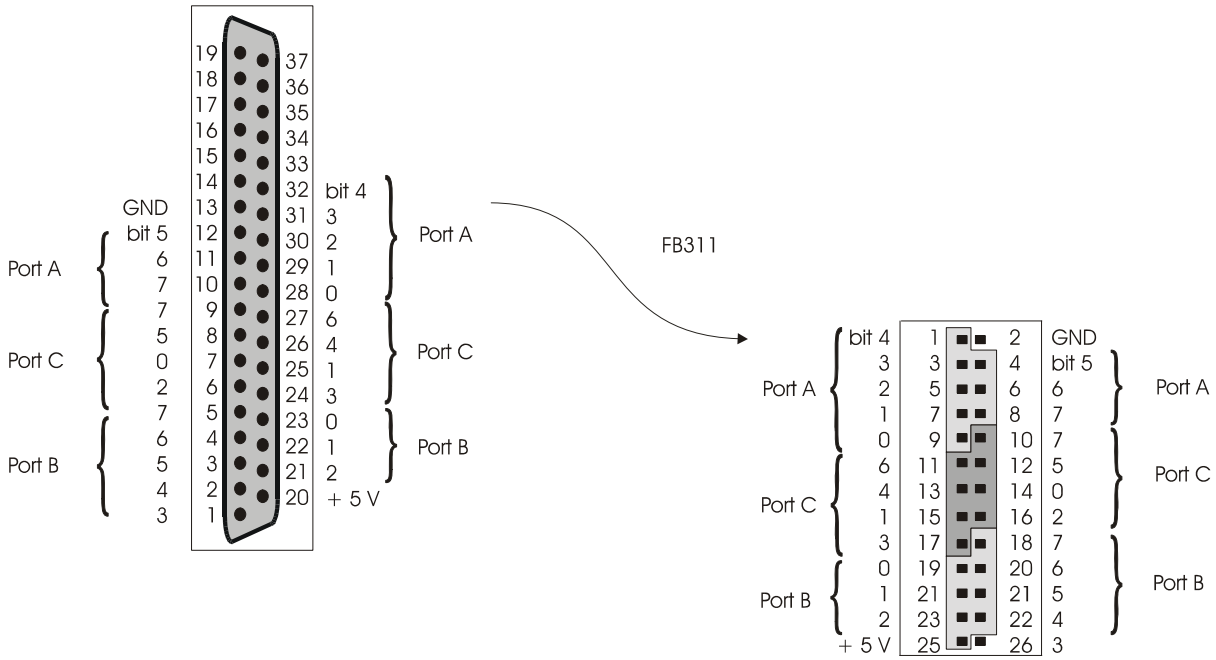
7.2 Connector pin assignment

Fig. 7-2: 37-pin SUB-D male connector



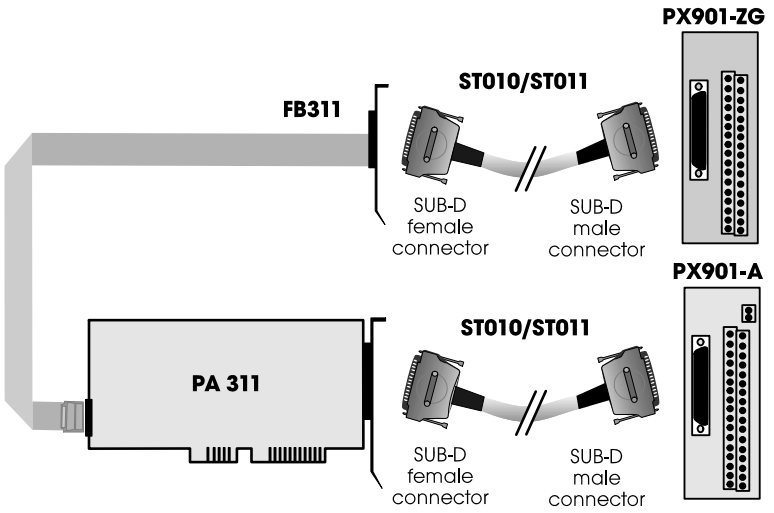
1: The analog inputs have a common ground line
2: The analog outputs have separate ground lines

Fig. 7-3: Pin header X22 connected to ribbon cable FB 311: 26-pin to 37-pin connector



7.3 Connection examples

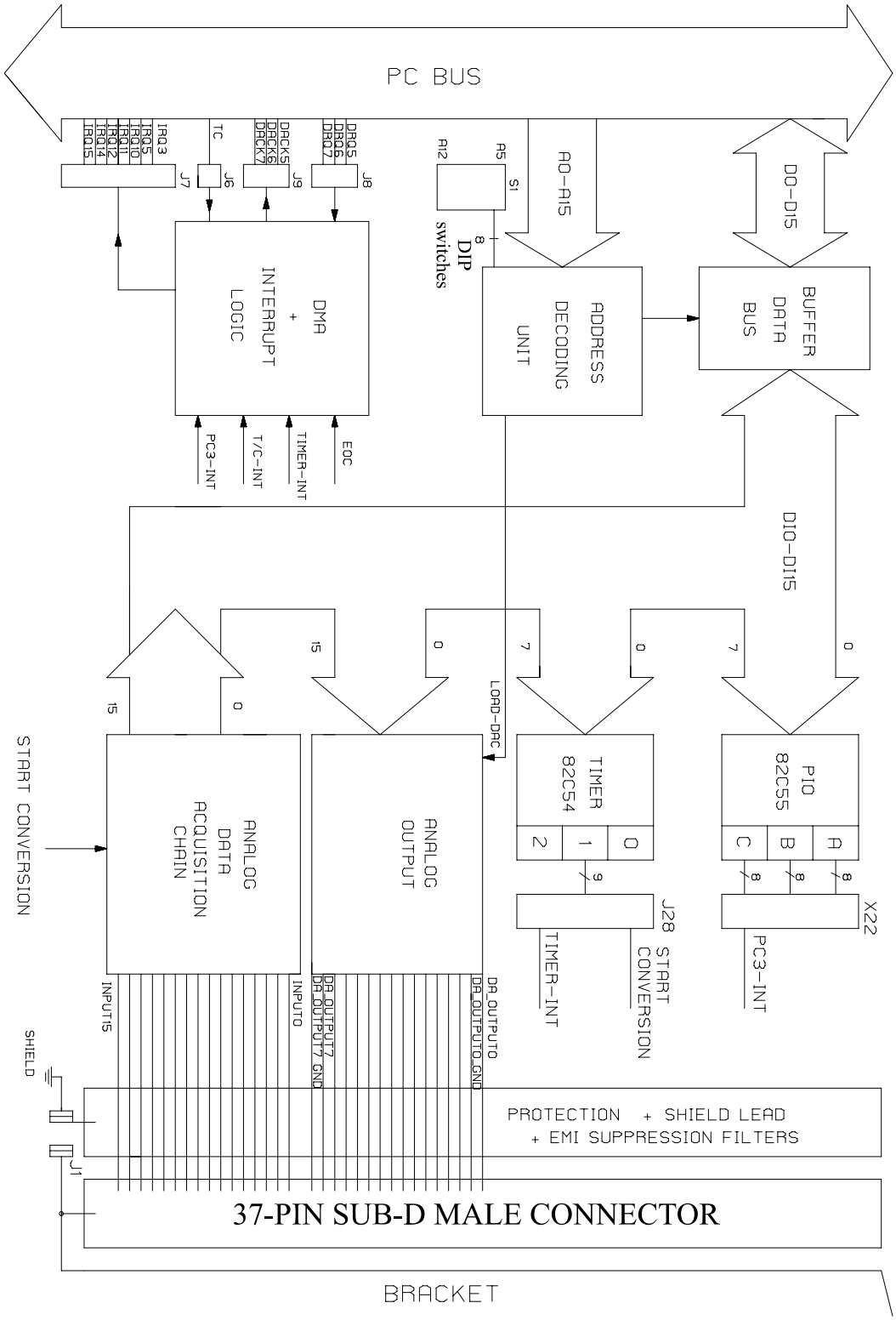
Fig. 7-4: Connection to screw terminal panels PX 901-ZG and PX 901-A



8 FUNCTIONS OF THE BOARD

8.1 Block diagram

Fig. 8-1: Block diagram of the PA 311



8.2 I/O mapping

The board **PA 311** requires a free address range of 28 I/O addresses within the I/O address space of the PC.

The functionalities of the board are responded with an I/O write or read command on this address range.

The decoding logic is related to the whole 64 KB I/O address space of the PC.

- **Set the base address** (beginning of the address range) with the block of DIP switches S1 in steps of 28 I/O addresses.
- **Analog data acquisition:** the word addresses 0 and 2 are used. The accesses occur **always** in 16 bits.
- **Analog output:** the word addresses 12 to 26 are used. The accesses occur **always** in 16-bits.
- The timer (82C54) is programmed through the addresses 4 to 7.
  The PIO (82C55) is programmed through the addresses 8 to 11.
  The accesses to the timer and the PIO occur always in 8-bits.

8.2.1 16-bit I/O mapping

Table 8-1: I/O map for 16-bit accesses

	I/O Read				I/O Write		
	D15	D14		D0	D15		D0
Base +0	MSB	RD - ADS		LSB	WR - MUX		
Base +2	EOC	TC	RD - STAT		WR - ADS		
Base 4+11	**See below *I/O map for 8-bit accesses***						
Base +12	LATCH - DAC				WR - DA OUTPUT 0		
Base +14	Not decoded				WR - DA OUTPUT 1		
Base +16	Not decoded				WR - DA OUTPUT 2		
Base +18	Not decoded				WR - DA OUTPUT 3		
Base +20	Not decoded				WR - DA OUTPUT 4		
Base +22	Not decoded				WR - DA OUTPUT 5		
Base +24	Not decoded				WR - DA OUTPUT 6		
Base +26	Not decoded				WR - DA OUTPUT 7		

8.2.1 Description of the I/O map

Four addresses are used in the I/O map for A/D conversion. Readings and writings occur only in 16-bits on this part of the I/O map (word accesses).

WR-MUX register

The **WR-MUX register** is located on **Base +0**.
The following parameters are configured through a writing on this register:

- **UNI/BIP** Input range: unipolar or bipolar
- **GATE0-3** Enables the GATE inputs of the timers
- **PR0-3** Number of inputs to be converted (for sequences)
- **GAIN0-3** Gain (PGA if the operating mode allows it)
- **MUX0-3** Input number: 0 to 15 (configures the input to be converted)

WR-MUX (Base +0)

D15															D0
UNI BIP	GATE 2	GATE 1	GATE 0	PR 3	PR 2	PR 1	PR 0	GAIN 3	GAIN 2	GAIN 1	GAIN 0	MUX 3	MUX 2	MUX 1	MUX 0

This register cannot be read. We advise to store the last written value in a shadow register reflecting the content of the **WR-MUX register**. After a PC reset this register is set to "0".

1) UNI/BIP

This bit determines the voltage range of the input to be converted.

"0": bipolar ±10V with gain = 1
"1": unipolar 0-10V with gain = 1

2) GATE0-2

These three bits are used to control the 3 gate inputs of timer 82C54 by software. These 3 digital signals are routed to the timer, if the corresponding jumpers are set in jumper field J28.

After a reset these three bits are at "0" and the timer is disabled.
GATEx = "0" TIMERx = disabled
GATEx = "1" TIMERx = enabled

3) PR0-3

The bits **PR0-3** determine the number of the inputs to be converted during a conversion sequence. If no sequence is stored (see *Conversion driven by timer - Automatic conversion*), these bits are always set to logic "1".

IOWR on base + 0				Number of channels to be converted			
				PA 311-8-x		PA 311-16-x	
PR3	PR2	PR1	PR0	SE	DIFF	SE	DIFF
0	0	0	0	0	0	0	0
0	0	0	1	1	1	1	1
0	0	1	0	2	2	2	2
0	0	1	1	3	3	3	3
0	1	0	0	4	-	4	4
0	1	0	1	5	-	5	5
0	1	1	0	6	-	6	6
0	1	1	1	7	-	7	7
1	0	0	0	-	-	8	-
1	0	0	1	-	-	9	-
1	0	1	0	-	-	10	-
1	0	1	1	-	-	11	-
1	1	0	0	-	-	12	-
1	1	0	1	-	-	13	-
1	1	1	0	-	-	14	-
1	1	1	1	-	-	15	-

4) GAIN0-3: gain selection (PGA)

IOWR on Base +0				Gain
GAIN3	GAIN2	GAIN1	GAIN0	selection
0	0	0	1	1
0	0	1	0	2
0	1	0	0	10
1	0	0	0	user selectable

5) MUX0-3: input selection

IOWR on base + 0				Selected analog input			
				PA 311-8-x		PA 311-16-x	
MUX3	MUX2	MUX1	MUX0	SE	DIFF	SE	DIFF
0	0	0	0	0	0	0	0
0	0	0	1	1	1	1	1
0	0	1	0	2	2	2	2
0	0	1	1	3	3	3	3
0	1	0	0	4	-	4	4
0	1	0	1	5	-	5	5
0	1	1	0	6	-	6	6
0	1	1	1	7	-	7	7
1	0	0	0	-	-	8	-
1	0	0	1	-	-	9	-
1	0	1	0	-	-	10	-
1	0	1	1	-	-	11	-
1	1	0	0	-	-	12	-
1	1	0	1	-	-	13	-
1	1	1	0	-	-	14	-
1	1	1	1	-	-	15	-

WR-ADS register (start of conversion)

This register is located on **Base +2**. Conversion begins by writing a dummy word on this register.

RD-STAT register

This register is located on **Base +2**. You can detect by reading a word from this register whether a conversion (bit **EOC**) or a DMA process (bit **TC**) is completed.

RD-STAT (Base +2)

D15	D1	.....	D0
EOC	**TC**		

EOC

With this bit you detect if a started conversion is completed.
EOC = "0" Conversion is completed
EOC = "1" Conversion is running

This bit is set to "1" when a new conversion is started.

TC

This bit indicates if a DMA transfer of a block is completed.
The logic state of bit **TC** is undefined after a reset.
TC = "1" DMA transfer completed (block)
TC = "0" DMA transfer is running

This bit is only used in DMA mode.

RD-ADS register

This register is located on **Base +0**. It contains after conversion the digital value of the converted analog signal. This value is a two's complement representation in 16-bit format.

RD-ADS (Base +0)

D15	.....	D0
MSB		LSB

The bits D4 to D0 are not used.

Transfer table

The A/D converter operates in bipolar mode. The conversion of the analog value into a digital code occurs in two's complement.

Description	±10V	0-10V	Two's complement	
			Binary code	Hex code
+ Full Scale	9,999695V	9,999847V	0 111 1111 1111 1111	07FFF
Midscale	0V	5V	0 000 0000 0000 0000	0000
Midscale - without LBS	-305µV	4,999847V	1 111 1111 1111 1111	0FFFF
- Full Scale	-10V	0V	1 000 0000 0000 0000	08000
1 LSB	305 µV	153 µV		0001

8.2.2 8-bit I/O mapping

Table 8-2: I/O map for 8-bit accesses

	I/O Read								I/O Write							
	D7	D6	D5	D4	D3	D2	D1	D0	D7	D6	D5	D4	D3	D2	D1	D0
Base +4	TIMER 0															
Base +5	TIMER 1															
Base +6	TIMER 2															
Base +7	STATUS TIMER								CONTROL TIMER							
Base +8	PORT A															
Base +9	PORT B															
Base +10	PORT C															
Base +11	STATUS - PIO								CONTROL - PIO							

8.3 Analog data acquisition

Up to 16 inputs can be acquired with 14-bit ot 16-bit (16-bit CMOS ADC) resolution.

The input ranges of the signals to be measured are:
0-10V (unipolar) or ±10V (bipolar).
Wider input ranges like 0-30V or ±30V can be obtained on request by building voltage dividers in the filtering area of the inputs.

Options SC and DC
Even currents of 0-20mA can be measured.
Smaller signals can also be acquired. For reaching the same resolution, the signal can be amplified.

Gain
Gain can be determined by the instrumentation amplifier INA through resistor or can be programmed by software over PGA.
Are available the gains 1, 2, 10 and user-configurable.

Input ranges and gain
Since they can be configured by software, signals with various ranges can be led into the system such as 0-20mA, 0-10V, ± 10, 0-5V, ±5V.

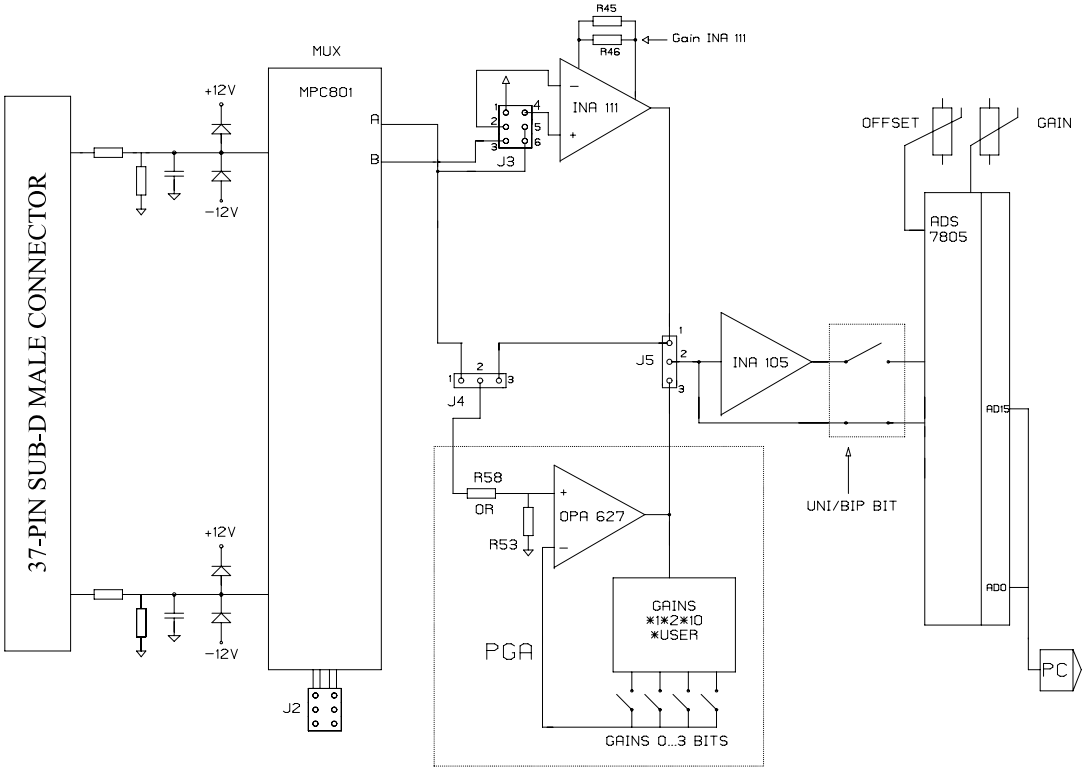
Before each conversion, gain and range are to be configured correspondingly by software.

Two measuring procedures are posssible:
- measuring connected to ground (single ended)
- measuring not connected to ground (differential amplifier)

The conversion time of the 12-bit A/D converter is 10 µs. A 12-bit value is then available.

You drive analog data acquisition through the addresses **Base +0** and **Base +2** of board **PA 311**.

Fig. 8-2: Analog data acquisition



You determine gain, input range and the input to be converted by a writing in the **WR-MUX register**.

- ***Conversion is started***
  via software by writing a dummy value in the **WR-ADS register**.

- ***Detecting if conversion is completed***:
  You analyse bit **EOC** (End of Conversion) in the **RD-ADS register**.

- You read the 12-bit analog value in the **RD-ADS register**.

You can also drive conversion with the timer 82C54.
It is also possible to generate an interrupt or a single DMA transfer after the end of conversion. In both cases interrupt is driven by bit **EOC**.

8.3.1 Operating modes

Fig. 8-2 *Analog data acquisition* shows that a signal can be routed in different ways to the A/D converter via the jumper fields J2, J3, J4, J5.

The operating modes are:
- Single ended without INA (gain with OPA627)
- Single ended with INA (gain with INA 111)
- Differential measuring with INA (gain with INA 111)
- Differential measuring with INA and PGA (gain with OPA627)

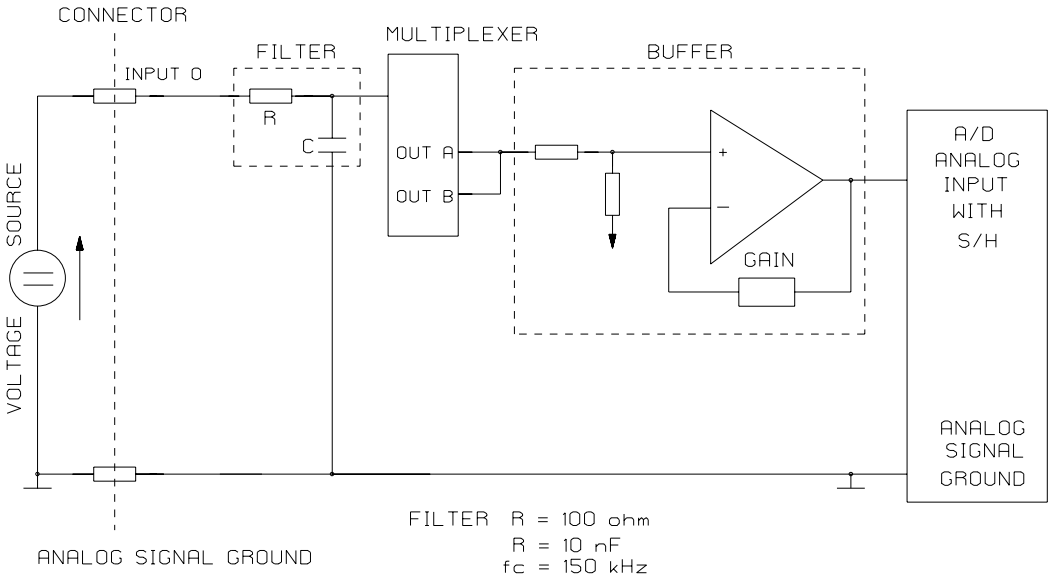
Configuring the operating mode

1) Signal and ground connection in the single ended mode without INA

If the **PA 311** operates in single ended mode, the return line of all input signals is common and is connected with the analog ground of the A/D converter. The multiplexer outputs are connected with the positive input of the "Sample and Hold" amplifier. See configuration without INA.

The signal ground of the A/D converter is connected to the analog ground pins of the 37-pin front connector.

Fig. 8-3: Single ended without INA

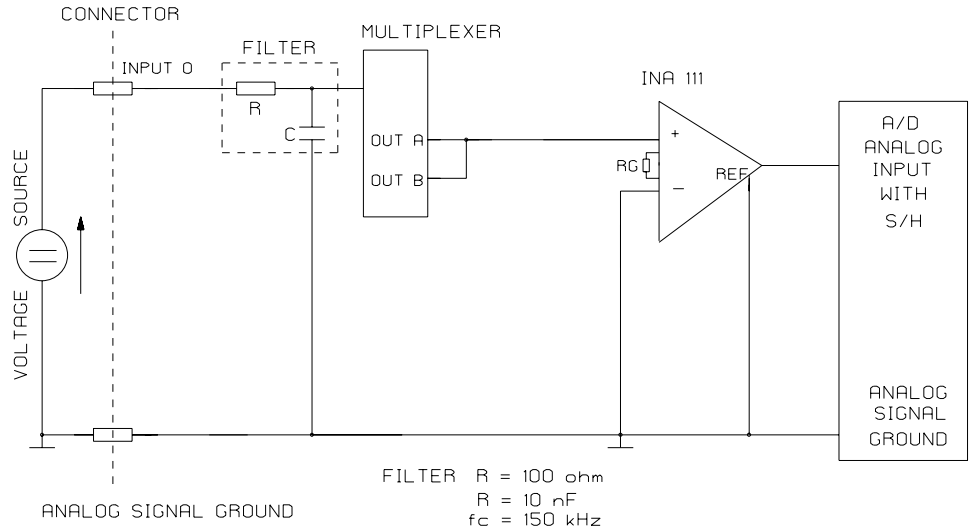


2) Signal and ground connection in single ended mode with INA

If the board is used with the configurable amplifier (INA), the multiplexer outputs are connected with the positive input of INA.

The negative input of INA is connected with the analog ground. The output of INA is connected with the input of "Sample and Hold" amplifier.

Fig. 8-4: Single ended with INA



3) Signal and ground connection in differential mode

In differential mode, the board is generally operated with INA.

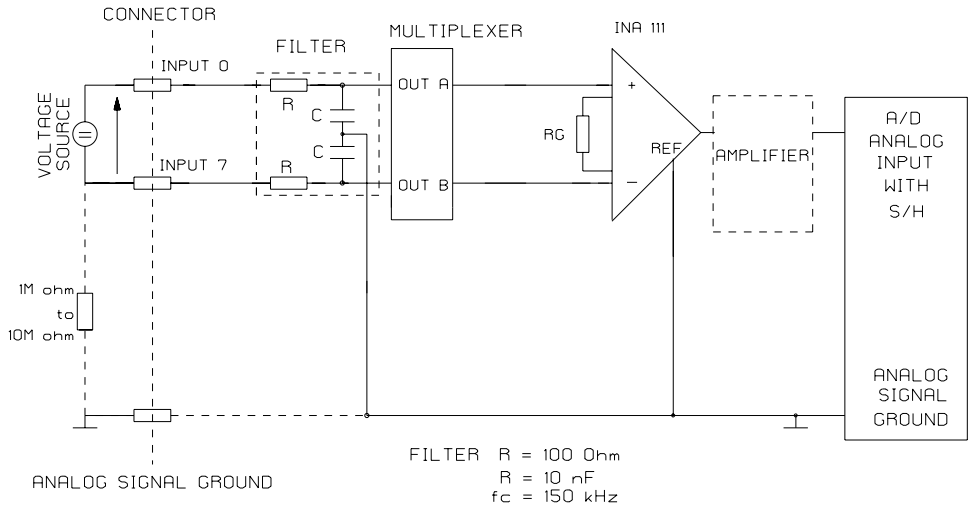
The output of the multiplexer - with channels 0-7 - is connected with the positive input of INA.
The output of the multiplexer - with channels 8-15 - is connected with the negative input of INA.

The INA output is connected with the input of the "Sample and Hold" amplifier.

The analog input signal has in this mode no ground connection.

If the voltage is outside the common-mode range of INA, the differential voltage may not be constituted at the INA input (±15 V voltage supply).
In this case a virtual ground is to be created with a 1 MΩ to 10 MΩ resistor.

Fig. 8-5: Differential with INA (PGA)



8.3.2 Configuring the gain

According to the operating mode you have selected, you configure the gain factor of analog acquisition:
- either with the INA 111 component through resistors
- or by software through component OPA627 (see section 5.1.2).

Gain with INA 111

The instrumentation amplifier INA 111 allows changing the gain factor by wiring a resistor. If you wish to select gain 1, make sure that no resistor is soldered (state at delivery).

If a gain factor superior to 1 is needed for all the inputs, you obtain it by inserting a resistor Rg whose value is calculated as follows:

$$G = 1 + 50k\Omega / Rg \rightarrow Rg = 50k\Omega / (G - 1)$$

G = wished gain factor

Make sure to insert only precision resistors with Tk < 25ppm/°C, reference grid 10mm.

You may possibly not find the wished value Rg among the standard resistors. Rg can also be obtained by installing two standard resistors in parallel on positions R46 and R45. See "Component scheme" (chapter 5).

Before installation, make sure that through the combination of these two resistors you obtain actually the value Rg. Thus the following formula:

$$Rg = R45 * R46 / (R45 + R46)$$

IMPORTANT!
When using the INA amplifier make sure that the time for starting conversion is adapted to the gain factor. See *delayed A/D conversion triggered by software*

Table 8-3: Input voltage ranges for different gain factors

Input voltage ranges		
Gain	**Unipolar input**	**Bipolar input**
	0 to 10 V	**-10 V to 10 V**
1	0-10V	-10 to +10V
2	0-5 V	-5 to +5 V
10	0-1 V	-1 to +1 V
20	0-500 mV	-500 to +500 mV
50	0-200 mV	-200 to +200 mV

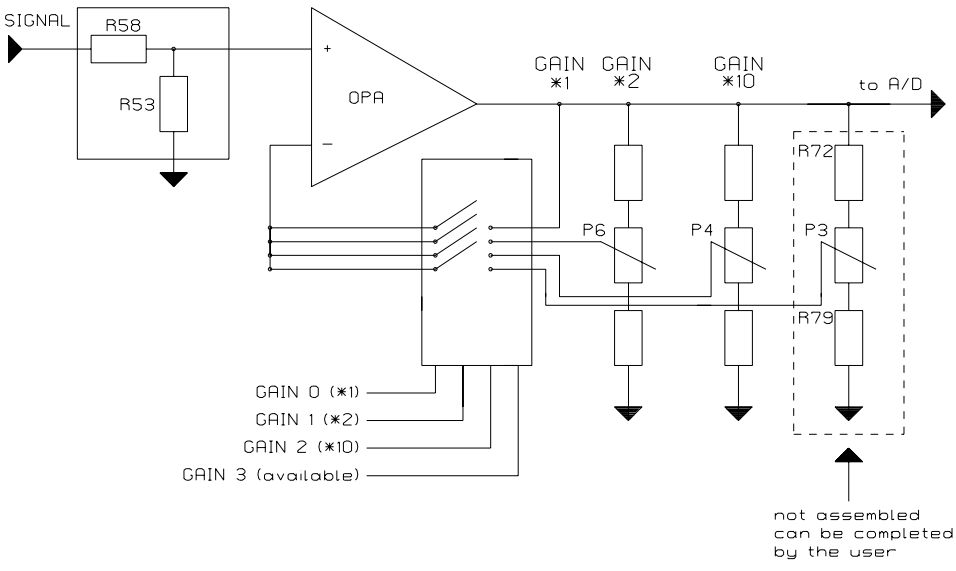
The table above is not exhaustive. Intermediate values can be set by selecting the appropriate resistor value Rg. The maximum configurable gain factor is 100.

The noise generated by the A/D converter is increased in the same proportion as gain is increased.

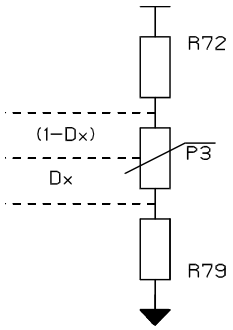
Gain with OPA627

The device OPA627 is built like a precision amplifier with a digitally programmable gain.

Fig. 8-6: Programmable amplifier



Gain is configured with the bits **GAIN0-3** of the **WR-MUX register** (**Base+0**).
The standard gain factors available are 1, 2, 10.
You configure one of the programmable gains with the resistors R72, R79 and P3 and then select it through the bit **GAIN3** of the **WR-MUX register**.



This gain is calculated as follows:

G = 1 + R72 + P3 * (1-Dx) / (R79 + P3 * (Dx))

R72, R79:	resistors 0,1 % Tk < 25 ppm reference grid 10mm R72 + R79 > 10K
P3 trimpotentiometer:	Tk ≤ 100ppm P3 ≤ 500R Typ 3296 of **BOURNS**

IMPORTANT!
The higher the gain is, the higher the settling time of the system is. **Noise is proportionally amplified according to the increase of the gain factor.**

8.3.3 Conversion

- **Configure** the wished operating mode for analog data acquisition.

You can now start the conversions.

Conversions can be started either:
- by software (see *Software conversion*)
- or by timer (see C*onversion driven by timer*)

IMPORTANT!
If the conversion is driven by timer, conversion cannot be triggered by software.

- **Select** how conversion is started through jumper field J23.

Software conversion

During the user program sequence, conversion is started at a definite line of the program. There are two different software conversions:
- delayed software conversion
- direct software conversion

1) Programming and functioning

Single conversion

To convert one single analog value, 3 or 4 of the following program operations are necessary.

1. The input number, gain and mode are written on the board base address through the **WR-MUX register** .

2. A/D conversion is started by a dummy writing on **Base +2** of the board.

3. Bit **EOC** of the **RD-STAT register** is analyzed. It is detected if conversion is completed or not.

4. The value is read through the **RD-ADS register** .

Programming example in C

An analog signal with a voltage between 0-10V is led to input 0 and must be converted.
The mode selected is single ended without INA, gain = 1.
The base address of the board **PA 311** is 0300H.
Conversion can occur as follows:

```
main ( )
    {
    int i_Value, i_EOC_Bit;
    outport (0x300, 0x8F10);          /* Select 0-10V, input 0,Gain = 1     */
    outport (0x302, 0);               /* Conversion start triggered         */
    do
    {
    i_Value = inport (0x302);
    i_EOC_Bit = i_Value & 0x8000;
    }
    while (i_EOC_Bit ! = 0);          /* Test EOC bit                       */
    i_Value = inport (0x300):         /* Reading converted value            */
    printf ("Value = %d", i_Value);   /* Display value                      */
    }
```

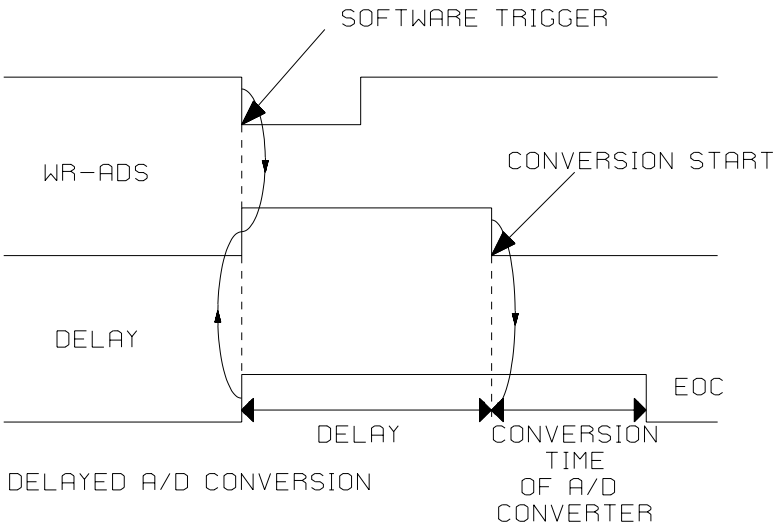
2) Delayed A/D conversion triggered by software

The board **PA 311** can be operated with delayed A/D conversion. Delay is triggered by software (writing on **Base +2**).

Delay depends on three factors:

- Gain of INA 111 (when used) or of PGA (OPA627)
- Internal resistance of the source (Rq)
- Limit frequency of the input filters (options SF and DF)

Fig. 8-7: Delayed A/D conversion



Influence of the internal resistance of the signal source

Delay time allows the internal capacities to recharge to the new voltage value.

The conversion time lasts typically 10 µs in the delivered version. This time relates to a signal source with an internal resistance < 100R.

With the falling edge of delay, the analog value is held in the "Sample and Hold" amplifier and conversion is started.

In this case is guaranteed that the multiplexer capacities are recharged quickly enough to the new voltage value. If the signal sources used are of larger internal resistance (>100R), the internal capacities will need more time to recharge to the new voltage value. The delay time must then be prolonged.

Influence of the gain selected at INA 111 on the delay time

When a signal is amplified by the INA, you must take into account the delay time which corresponds to the settling time (t_R) of the amplifier (see the following table).

The settling time of INA 111 at 0,01 % and with a voltage jump of 20V equals*:

Gain =1	t_R = 2 µs
Gain =10	t_R = 2 µs
Gain =100	t_R = 4 µs

* extracted from the data sheet of **BURR BROWN**

The delay time is adjusted with jumper field DELAY (J18).

3) Directly triggered software A/D conversion

In this mode no delay time is added between the software triggering and the real start of A/D conversion. To recharge the internal capacities, only the conversion time of the A/D converter is available, i.e. maximum 10µs. It results a maximum throughput of 100kHz.

To select this mode, the following conditions are required:
- Rq (internal resistance of source) must be < 100R
- the gain used must be comprised between 1 and 10.
- the corresponding mode is configured via jumper J23.

This mode allows optimising the times of conversion duration: for example when one single input or when several inputs are converted one after the other.

Example

Input0 should be converted 100 times successively. Input0 is in bipolar mode, gain=1

The following points are carried out:
1. Selection of input, gain and mode by writing on the **WR-MUX register** (**Base +0)**.
2. The first conversion is started by writing on the **WR-ADS register** (**Base +2)**.
3. Wait for the end of conversion and analyse bit **EOC** of the **RD-STAT register** (**Base +2**).
4. The conversion value is read on **Base +0**.
5. When conversion is completed the next conversion is started.
6. Point 3 is executed until the 100 values have been read.

Example in Pascal

```
CONST       BASE      = $300;                          (* Base address of board PA311   *)
VAR         b_Dummy : BYTE;
            w_Value   : ARRAY [1..100] of WORD;
            b_Index   : BYTE;
BEGIN
    .
    .
    .
      PORTW [BASE + 0]      := $0F10;                  (* Input, gain and range         *)
                                                       (* selection                     *)
      PORTW [BASE + 2]      := b_Dummy;                (* Trigger conversion            *)
      FOR     b_Index       := 1 to 100 DO
      BEGIN
         REPEAT                                        (* Test EOC bit                  *)
         UNTIL (PORTW[BASE + 2] AND $8000 <> $8000);
         PORTW [BASE + 2] := b_Dummy;                  (* Trigger conversion            *)
         w_Value [b_Index]:= PORTW [BASE + 0];         (* Reading in the converted      *)
                                                       (* value                         *)
      END;

      .
      .
      .
  END.
```

Several inputs should be converted one after the other. For example inputs 0 to 8. The inputs can have different or identical modes and gains.

The following points are carried out:

1. The first input and its gain and mode are selected by a writing on the **WR-MUX register** (**Base +0)**.
2. The first conversion is started by a dummy writing on the **WR-ADS register** (**Base +2)**.
3. The next input and its gain and mode are selected by writing on the **WR-MUX register** (**Base +0)**. The next input will use the conversion time of the first input (≅ 10µs) for settling.
4. You can verify if the first input is converted by analysing the bit **EOC** of the **RD-STAT register** (**Base +2**).
5. The converted value is read on **Base +0**.
6. When conversion is ended, the conversion of the next input is started with a dummy writing on the **WR-ADS register**.
7. Point 3 is executed until the values of all the inputs have been read.

Example in Pascal

```
CONST BASE          = $300;          (* Base address of board PA
311     *)
      UNIPOLAR      = $8000;         (* Unipolar 0-10V          *)
      PRESET        = $0F00;         (* Without sequence        *)
      GAIN1         = $0010;         (* Gain = 1, PGA           *)
      SET           = UNIPOLAR OR PRESET OR GAIN1;

VAR   b_Dummy       : BYTE;
      w_Value       : ARRAY [1..8, 1..100] of WORD;
      I_Index, J_Index : BYTE;

BEGIN
   FOR J_Index := 1 to 100 DO
   BEGIN
      PORTW [BASE + 0] := SET;
      PORTW [BASE + 2] := b_Dummy;
      FOR     J_Index := 1 to 8 DO
         BEGIN
            PORTW [BASE + 0] := SET OR I_Index;
            PORTW [BASE + 2] := b_Dummy;
            REPEAT
            UNTIL (PORTW [BASE + 2] AND $8000 < > $8000);
            w_Value [I_Index ; J_Index] : = PORTW [BASE +0]
         END;
      END;
   END;
.
.
.
END.
```

Conversion driven by timer - Automatic conversion

1) Timer

The board **PA 311** is equipped with a timer device of type 82C54.
Its consists of timer 0 and timer 1 which can be used for automatic conversion.
The function of this component is programmed by software (8-bit access) and is configured through jumper field J28.

IMPORTANT!
If the conversion is driven by timer, it cannot be triggered by software.

For analog data acquisition the board **PA 311** offers the mode of automatic conversion.

The user can convert :
- several inputs successively in sequences. The time between two conversions is programmable.
- several inputs successively in sequences. The time between the conversion of each individual input is programmable. The recurrence period of this sequence can also be programmed.

Timer 82C54

Timer 0 and timer 1 of the component 82C54 are intended for these uses.
Timer 0 drives A/D conversion and selects the input to convert.

The time between two A/D conversions is programmed according to the input
circuitry of the analog data acquisition (gain factor, filter).

Timer 0 can also change from one input to the other. Input 0 to the last defined input are converted cyclically depending on the reference value written in the **WR-MUX register**.

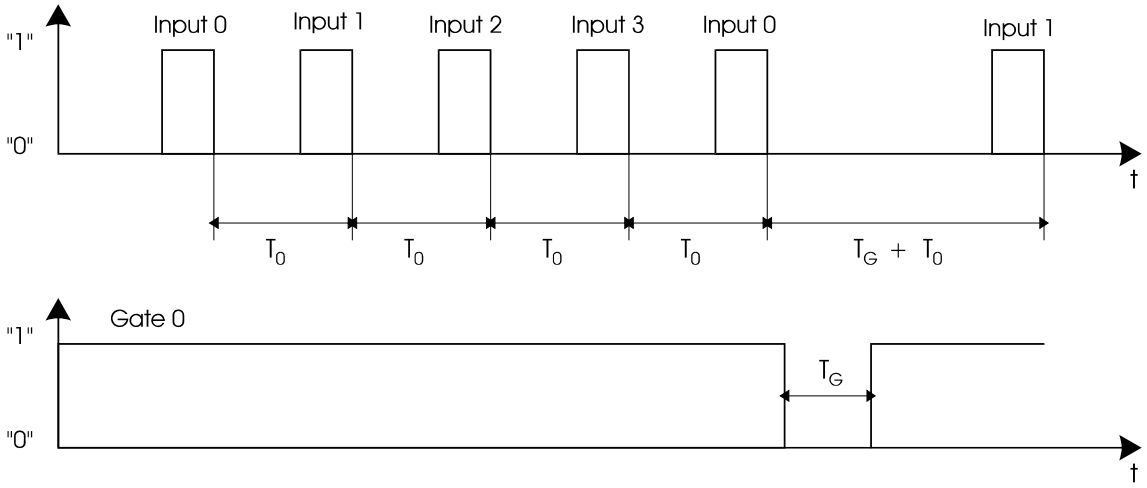
Timer 1 is used for the time between two sequences.

The automatic conversion mode is recommended in relation with interrupt or DMA. You configure this mode through jumper field J23.

2) Converting several inputs during a programmable period

In this mode several inputs are converted successively during a programmable period.

Fig. 8-8: Time diagram of conversion



The number of inputs to convert is determined through the bits **PR0-3** of the **WR-MUX register** (**Base +0**).

Example 1

If **PR0-3** = 0, only input 0 is converted.
If **PR0-3** = 7, inputs 0 to 7 are converted.
If **PR0-3** = 15, inputs 0 to 15 are converted.

The time between 2 conversions is determined through TIMER0.
The time T_G can be extended via bit **GATE0** of the **WR-MUX register**.
This bit can also be used for synchronising conversion with software or an external signal.

Timer0 is programmed in mode 2. The divider factor determines time T_0 between 2 conversions.

The minimum time must be superior to the input delay set + the conversion time of the A/D converter i. e. at best ≥ 10µs.

Example 2

An input has to be converted every 500 µs = 2000 Hz.
If fin (Timer 0) = 892.857 kHz
Then the following divider factor is to be written in timer 0:

Divider factor = 892857 / 2000 ≠ 446 = 01BEH

This mode is interesting when conversion is used with interrupt or DMA. Conversions can be started and stopped by software through bit **GATE0** of the **WR-MUX register.**

If **GATE0** = "0", timer or A/D conversion is stopped.
If **GATE0** = "1", timer is running anew.

See also mode 2 of timer 82C54.[1]

Software handling

Inputs 0 to 7; every 500µs; GAIN 1; voltage ±10V

1. Write value **0710H** in the **WR-MUX register** (**Base +0**)

2. Program timer 0 in mode 2
   Write **034H** on **Base +7**
   Write low byte **0BEH** on **Base +4**
   Write high byte **01H** on **Base +4**

3. Enable Timer 0 by setting to "1" bit **GATE0** of the **WR-MUX register**

4. Conversion can now be driven or synchronised by software through bit **GATE0**.

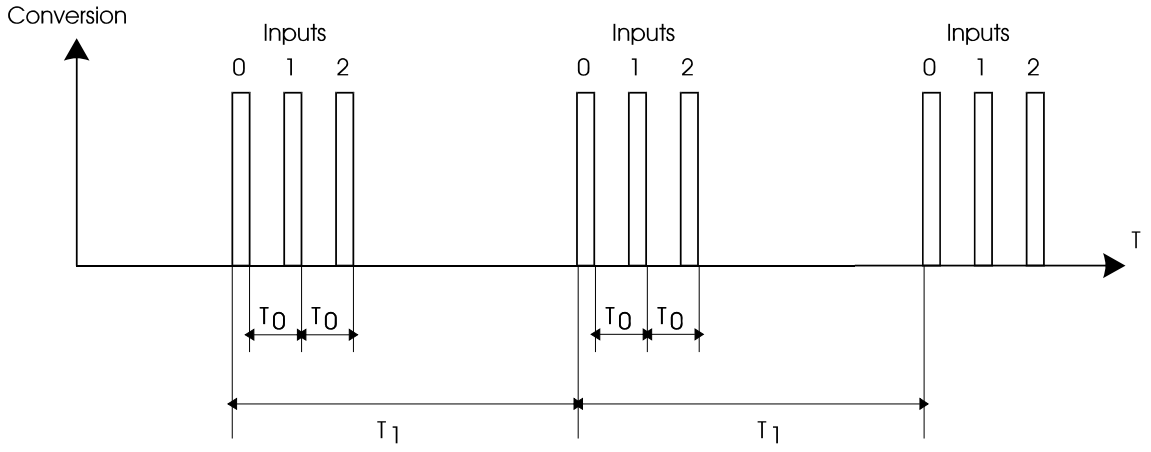
IMPORTANT!
With this operating mode, the settling time of an input can be optimised.

3) Analog acquisition sequences

In this mode two times are defined:

- the time between the conversion of 2 inputs through timer 0
- the time between 2 cycles through timer 1

Fig. 8-9: Time diagram of conversion



Time T_0 is determined through TIMER0. It sets the time between 2 conversions.
Time T_1 is determined through TIMER1. It sets the time between two sequences.

Example 1

[1] you receive this data sheet on request

The inputs 0-7 should be converted within 100 µs. This cycle is to be repeated every 10 ms. Gain = 1, voltage = ± 10V

Input CLK0 is at 892857 Hz. To reach a time of 100 µs = 10 kHz, the following divider factor must be written in TIMER0.

Divider factor TIMER0 = 892857 / 10000 = 059H

Input CLK1 is at 892857 Hz. To reach a time of 10 ms = 100 Hz the following divider factor is written in TIMER1.

Divider factor TIMER1 = 892857 / 100 = 022E1H

Software Handling

1. Write value **0710H** in the **WR-MUX register** (**Base +0**)
   Bipolar mode, gain = 1, inputs 0 to 7 are converted.
2. Program timer 1 in mode 2
   Write **074H** on **Base+7**
   Write low byte **0E1H** on **Base +5**
   Write high byte **022H** on **Base +5**
3. Program timer 0 in mode 2
   Write **034H** on **Base +7**
   Write low byte **059H** on **Base +4**
   Write high byte **00H** on **Base +4**
4. Write value **02710H** in the **WR-MUX register**
   GATE1 = "1"
5. Sequence is running.

IMPORTANT!
The recurrence period of the cycle **must be >**
the programmed *number of inputs to be converted* **x** T_0, (time between conversions).

Time of Timer 1 ≥ (time of Timer 0 x number of programmed inputs)

Check the defined times of example 1:

Time of timer 1 = 10ms ≥ time of timer 0 = 100 µs * 8
10ms ≥ 100 µs x 8
10ms ≥800 µs is real

The sequence can run.

8.3.4 Options

Option SF, DF

With option **x**F, 2 wirings with filter are possible. The used filters are low-pass
filters of 1st order ,-20dB / decade.

Fig. 8-10: Board PA 311 SF (single ended with input filter)

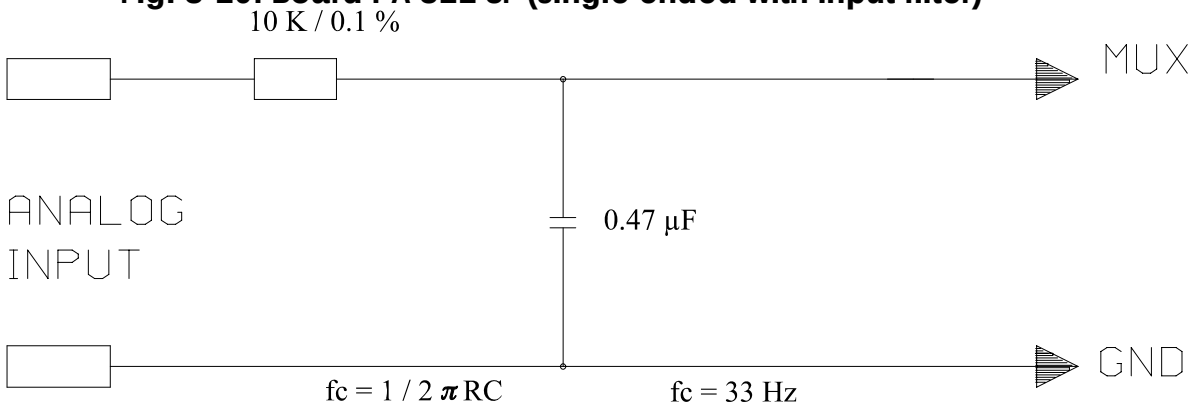
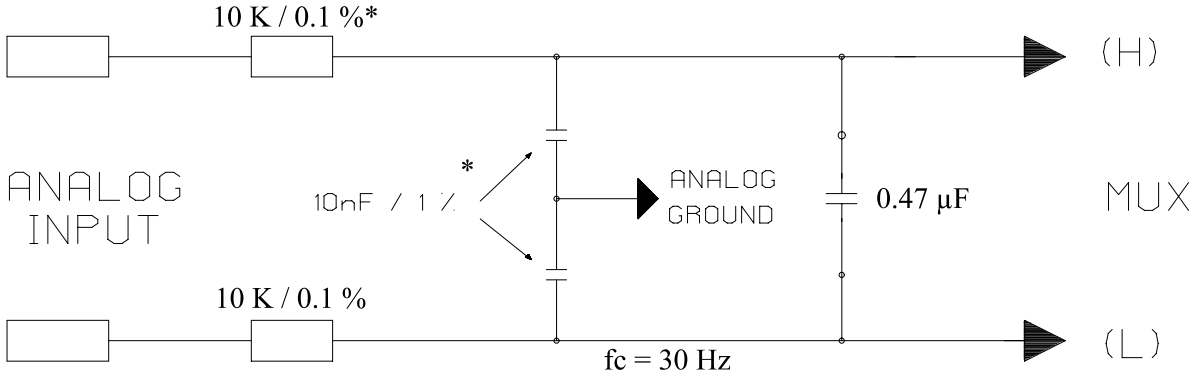


Fig. 8-11: Board PA 311 DF (differential with input filter)



* selected components

Option SC, DC

With option **x**C, the inputs are wired as follows:

Fig. 8-12: Board PA 311 SC (single ended with current converter)

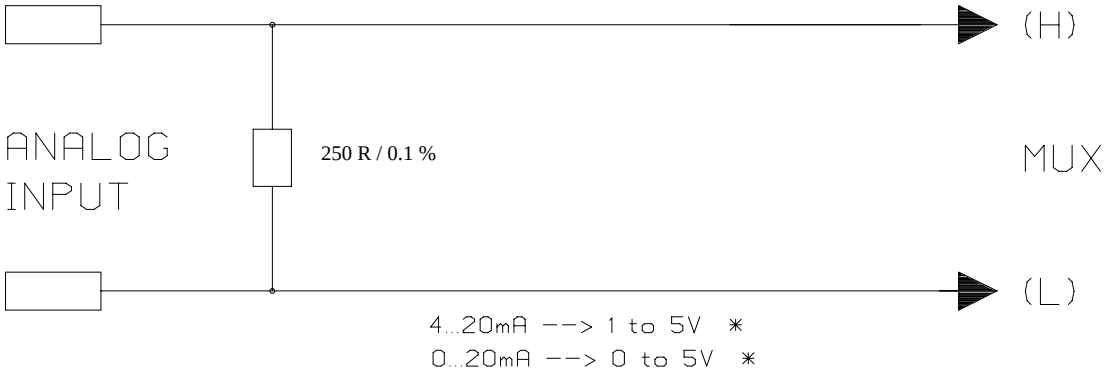
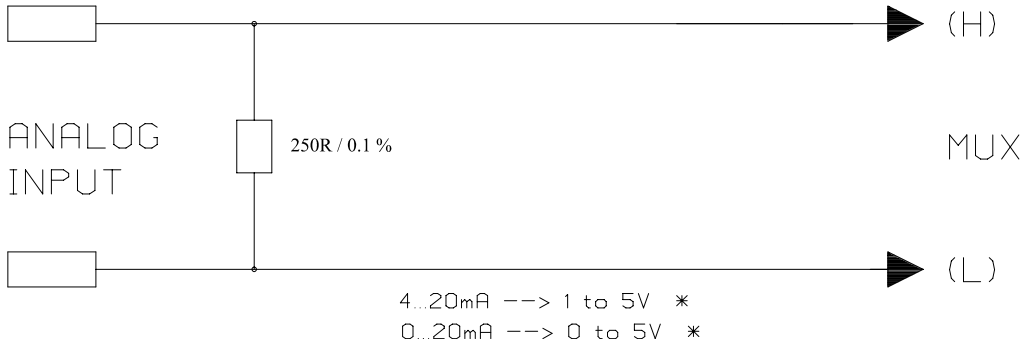


Fig. 8-13: Board PA 311 DC (differential with current converter)



Inputs with 4-20mA or 0-20mA current loop signals (option)

With option xC the board **PA 311** is equipped so that current is converted to voltage in the input range. This mode is often used in industrial and process control.

In single-ended mode, all 16 inputs are equipped with a current-voltage converter (option SC). In differential mode 8 inputs are equipped so (option DC).

A highly precise measuring resistor of 250R 1/4W is used for converting current to voltage.

In single ended mode, each resistor is installed between the analog input of the front connector and the analog ground.

In differential mode, the resistor is installed in parallel to the input signal.

***** A current input signal of 0(4)-20mA produces at the measuring resistor a voltage drop of 0(1) to 5V.
When using the option C, the input voltage of the board should be set at the range of 0-10V (gain = 2).

You can install the option C yourself by inserting measured resistors of a value of 250 0,1 % Tk ≤10ppm on the positions mentioned in sections A and B.
The resistors have to be selected at 250R.
See *component scheme*. This installation occurs at your own risk (no guarantee).

1) Position of the resistors for current inputs in single-ended mode (option SC)

The order in which the following resistors appear corresponds to inputs 0 -15:
R3, R7, R11, R15, R1, R5, R9, R13, R19, R23, R27, R31, R17, R21, R25, R29

2) Position of the resistors for current inputs in differential mode (option DC)

The order in which the following resistors appear corresponds to inputs 0-8:
R2, R6, R10, R14, R18, R22, R26, R30

3) Mixed mode

Since gain and voltage ranges can be determined by software, it is possible to operate the board with current and voltage inputs simultaneously.

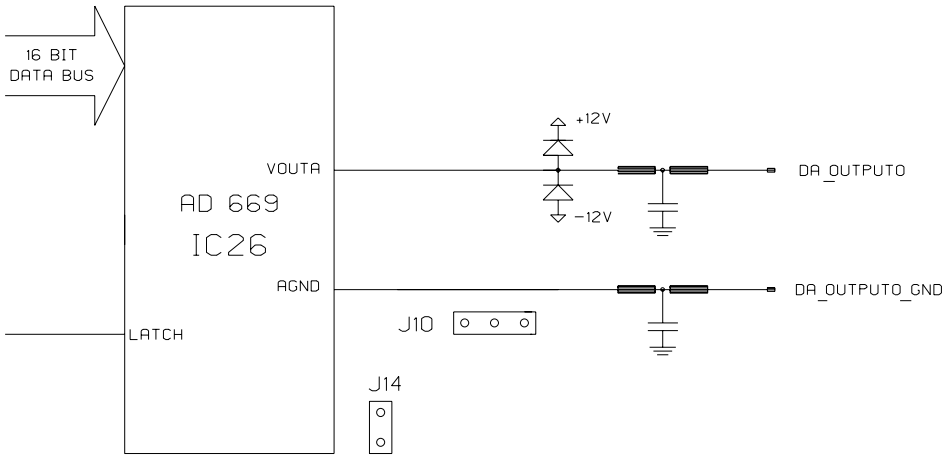
8.4 Analog output channels

The board **PA 311** is equipped with 2, 4 or 8 analog outputs.
The analog outputs are provided with a multiplying DACPORT of ANALOG DEVICES. These components are built so that they can easily generate voltages of 0-10V (unipolar) or ±10V (bipolar) without external components.

The AD 669 has a double-buffered interface. Digital data are first loaded for each output in the input register. Then the output of each channel is updated. This occurs with a dummy I/O read on **Base +12**.

8.4.1 Structure of output 0

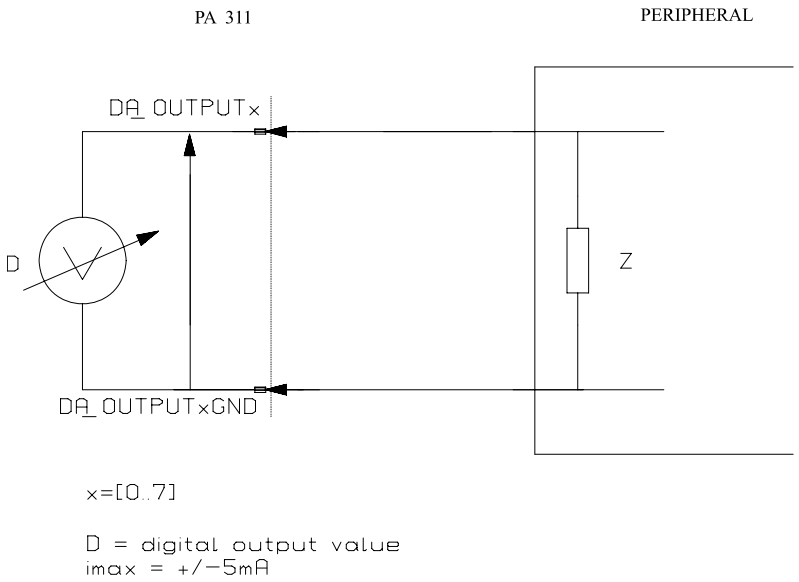
Fig. 8-14: Structure of output 0



The outputs 0 to 7 are built like shown in Fig. 8-14.

The following connection is possible:

Fig. 8-15: Connection of output 0



8.4.2 Remarks

- The resistive part of the input impedance (z) must be ≥ to 2 kΩ and the parallel global capacity must be < to 1000 pF.
- There is no optical isolation between the PC and the analog outputs.
- After PC reset the voltage level of the outputs is undefined between the minimum and the maximum value of the A/D converter.

8.4.3 Analog output channels: programming and functioning

The board occupies 16 to 28 bytes in the I/O address space.

PA 311-162	16 bytes
PA 311-164	20 bytes
PA 311-168	28 bytes

- **Make sure** that the base address occupies **always** a block of 32 bytes.

Each analog output occupies 2 addresses. The first address for the low byte and the second address for the high byte.

The values cannot be read.
We advise to store the last written value in a shadow register.
Writings on the board occur only in 16-bit mode.

If the analog output value is to be changed, the new digital value is to be written in the addresses.
Afterwards the value is written in the **DAC output register** by a dummy I/O Read command on **Base +12** .

14-bit output value

The range of 0-10V is selected for input 0. It has to be used as a 12-bit output.
The user would like to output a voltage of +9.3774 V.
The value Dx repesents the binary output value. The following points have to be executed:

1. Calculating the binary value Dx which corresponds to +9.3774 V in the range of 0-10V.
2. Converting Dx into a 16-bit value by shifting from four places to the left.
3. Write value Dx on output 0.

1) Calculating the binary code for output signal

$$Dx = \frac{\text{(analog output value)}}{\text{range}} * (2^{\text{resolution}} - 1)$$

Example: with analog output value = +9.3774 V

$$Dx = \frac{+9.3774V}{10V} * (2^{14} - 1) = 15363$$

14-bit binary representation of Dx = 15363

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	1

MSB LSB

2) Conversion of the 14-bit value into a 16-bit output value

In order to write the 12-bit value Dx on the DAC port with its real value, it has to be converted into a 16-bit value. Therefore you can either:
- multiply value Dx by 4
- or shift value Dx from four places to the left.

16-bit binary representation

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
1	1	1	1	0	0	0	0	0	0	0	0	1	1	0	0

MSB LSB

Bits D0 and D1 are not used and should be set to "0".

3) Write on output 0

```
PROGRAM SET-DAC-OUTPUT-TO-FIVE-VOLTS;

CONST   BASE = $300;                                  (* Base address of   *)
                                                      (* board PA 311      *)
VAR     b_Dummy              : BYTE;
        w_Value              : WORD;
BEGIN
        w_Value              := 15363;                (* 14-bit value      *)
        w_Value              := w_Value SHL 24;       (* 16-bit value      *)
        PORTW [BASE + 12]    := w_Value;              (* Write on output 0 *)
        b_Dummy              := PORT [BASE + 12];     (* Update value      *)
END:
```

16-Bit output value

The range of 0-10V is selected for input 1. It is to be used as a 16-bit output.
The user would like to output a voltage of + 8 V.
The value Dx repesents the binary output value. Proceed as follows:

1. Calculate the binary value Dx which corresponds to + 8 V in the range of 0-10V.
2. Write value Dx on output 1.

1) Calculating the binary code for output signal

$$Dx = \frac{\text{(analog output value)}}{\text{range}} * (2^{\text{resolution}} - 1)$$

Example: with analog output value = + 8 V

$$Dx = \frac{+ 8\ V}{10V} * (2^{16} - 1) = 58981.5$$

2) Write on output 1

```
PROGRAM SET-DAC-OUTPUT-TO-EIGHT-VOLTS;

CONST   BASE = $300;                                  (* Base address of   *)
                                                      (* board PA 311      *)
VAR     b_Dummy              : BYTE;
        w_Value              : WORD;
BEGIN
        w_Value              := 59981;                (* 16-bit value      *)
        PORTW [BASE + 14]    := w_Value;              (* Write on output 1 *)
        b_Dummy              := PORT [BASE + 12];     (* Update value      *)
END:
```

Relation table for the analog output of a 12-bit value

Digital value	Unipolar (USB*)	Bipolar (BOB*)
0FFFFH	+ Full Scale	+ Full Scale
08000H	+ 1/2 Full Scale	Zero
07FFFH	+ 1/2 Full Scale - 1LSB	Zero - 1LSB
0000H	Zero	- Full Scale

*USB = Unipolar Straight Binary
*BOB = Bipolar Offset Binary

For example:

Output 0 set on range:	written with value:	Output voltage
0-10V	0000H	0V
0-10V	0FFFFH	10V
0-10V	08000H	5V
±10V	0000H	-10V
±10V	0FFFFH	+10V
±10V	08000H	0V

8.5 Parallel input and output channels (TTL)

8.5.1 24 inputs and output channels, TTL

The **PA 311** allows processing simultaneously 24 TTL input and output signals. There is no optical isolation. The signals can be programmed as inputs or output channels. The board is connected to the peripheral through the 26-pin header X22 (See Connection to ribbon cable FB 311)

Component 82C55A
The interface is realised by the parallel port 82C55A.
Since this device is programmable, it offers a wide range of applications.
The parallel port is divided into three 8-bit groups. (port A, B and C).
They can be configured independently from the others as input or output.
Port A can also be configured bidirectional. In different modes port C functions like a control port.
The various functionalities are explained in the data sheet of component 82C55A.

This component needs software initialisation to select the wished function.
It is ready for programming immediately after applying the working voltage and successful power on / reset. All the port signals are available on connector X22.

8.5.2 Interrupt

With this function, it is possible to generate interrupts.
Thus port line PC3 must be led through a buffer to pin 4 of jumper field J7.
In several modes of port 82C55A the port line PC3 acts as an interrupt output. This signal is not latched outside the port circuit. In the same way, an external TTL input signal connected to port line PC3 (Pin 17) of pin header X22 can be used for generating an interrupt.

8.5.3 Programming

The port functions of board **PA 311** are selected via the following addresses. Programming occurs in bytes in the 8-bit mode:

Base +08	Port A
Base +09	Port B
Base +0AH	Port C
Base +0BH	Port Status

"Base" designates the base address configured via DIP switches. It is set to 0300H.

The above designated port status is selected in this case over address 030BH.
For programming the wished mode see data sheet 82C55A.

8.5.4 Programming examples

Task: program all the port groups as TTL outputs

Solution 1: ASM86

```
.
.
.
MOV     DX, 030BH   ; Port status address
MOV     AL, 80H     ; All three ports as TTL outputs (mode 0)
OUT     DX, AL      ; Programmation
MOV     DX, 0308H   ; Load address port A
MOV     AL, DATEA   ; Output data for port A
OUT     DX, AL      ; Set the outputs of port A
INC     DX          ; Set the address of port B
MOV     AL, DATEB   ; Output data for port B
OUT     DX, AL      ; Set the outputs of port B
INC     DX          ; Set the address of port C
MOV     AL, DATEC   ; Output data for port C
OUT     DX, AL      ; Set the outputs of port C
.
.
.
```

Solution 2: BASIC

```
OUT&H30B,&H80    ' Program mode
OUT&H308,&Hxx    ' xx = output data for port A

OUT&H309,&Hxx    ' xx = output data for port B
OUT&H30A,&Hxx    ' xx = output data for port C
...
```

8.6 Timer

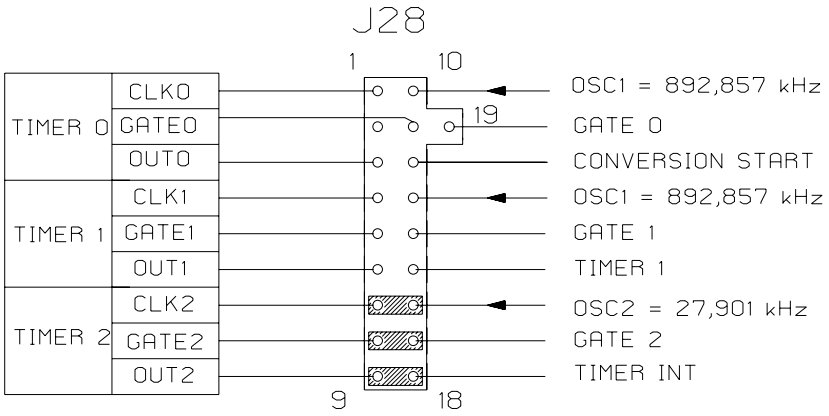
8.6.1 Function

The function timer is performed by timer 82C54. This device consists of three free programmable 16-bit timers. 2 timers can be used onboard to generate the automatic conversion mode.

Each of the three timers can be programmed independently from the others. The various modes are described in the data sheet of device 82C54.

The gate inputs **GATE0-2** are provided with 100K pull up resistors.
They can be commanded by software over the write register **WR-MUX** (**base +0**), if the jumpers for the gate inputs are set in jumper field J28.
See *WR-MUX register*.
Component 82C54 needs software initialisation for selecting the wished function. The device is ready to program immediately after applying the working voltage.

Configuration is shown in the next figure.
Fig. 8-16: Timer setting for the generation of interrupts



8.6.2 Interrupt

Interrupts can be generated with the timer.

- **Set the jumpers** in jumper field J28 as shown in fig. 8-16 .
  The buffered output signal of timer 2 is available on pin 3 of jumper field J7.
  It can be cabled to a free interrupt line. Timer 2 is to be programmed to generate an interrupt signal and enabled through bit **GATE2** of the **WR-MUX register**.

8.6.3 Programming

Programming occurs in bytes in the 8-bit mode. The individual timer functions are selected via the following addresses:

Base +04H	Timer 0, data address
Base +05H	Timer 1, data address
Base +06H	Timer 2, data address
Base +07H	Timer, Status & Control

The timer modes are programmed by the status address **Base +7**.
The divider factors of the individual timers are programmed by the data addresses.
These are divided into 2 bytes: low byte and high byte.
The low byte is first written in the data address.

The timer is started or synchronised by software through bits **GATE0-2** of the **WR-MUX register** (**Base +0**). Therefore configure the corresponding jumpers in jumper field J28.
After reset, these bits are set to logic "0"..

8.6.4 Programming examples

Task: An interrupt signal of 100Hz is generated. The timer is configured as shown in fig. 8-17. The IRQ line has been selected through jumper field J7.

Calculating the divider factor:

T = fin / fout ; T = 27901Hz / 100Hz = 27.9 → 28

Example in Pascal

```
Procedure Timer-Start (divider : word);
    CONST       Timer-Mode = $B4;       (* Timer 2 in Mode 2       *)
    VAR   Lowbyte , Highbyte : Byte;
    Begin
        Lowbyte : = divider Mod 256;
        Highbyte := divider Div 256;
        Port [ base + 7 ] :0 Timer_Mode;   (* TIMER 2 Control Byte *)
        Delay(1);
        Port [ base + 6 ] := Lowbyte;      (* TIMER 2 initial value low byte *)
        Delay(1);
        Port [ base + 6 ] := Highbyte;     (* TIMER 2 initial value high byte *)
    End;
```

MAIN

```
    BEGIN
        clrscr;
        value  := 0;
        Install_New_Irq_3;                 (* Install the NEW IRQ    *)
        Timer_Start ( 28 );                (* Timer conversion every 10 mS *)
        Portw [base]:= shadow_Reg OR $4000;
                                           (* GATE2 = "1", Start of TIMER2 *)
        (*  User application   *)
        Desinstall_Irq_3;
    END.
```

8.7 DMA

The board **PA 311** can send a DMA request to the PC as soon as an A/D conversion is completed. The DMA lines 5, 6 and 7 are available (16-bit DMA transfer).

Make sure that the DMA line selected is free. Selection occurs through the jumper fields J8, J9 and J6.

When a DMA transfer is used, the end of transfer (64K block) can request a terminal count interrupt to the PC (see section 5.6.4).
The end of transfer can also be indicated by Bit **TC** of the **RD-STAT register** (**Base +2**).

8.8 Interrupt

The board **PA 311** is equipped with four sources which can produce an interrupt request. The lines IRQ3, 5, 10, 12, 14 and 15 are available. Interrupts are selected through J7.

8.8.1 End of Conversion Interrupt (EOC)

When a conversion is completed, an interrupt request is sent to the PC.
The interrupt request is reset by a reading of the converted data on **Base +0** or by a new conversion start on **Base +2**.

8.8.2 Timer interrupt

Timer 2 can interrupt the PC cyclically with signal TIMER-INT. The cycle time is defined over Timer2. This timer **has to be** programmed in mode 2.

8.8.3 TTL I/O interrupt through port line PC3

With this function, it is possible to generate interrupts.
Thus port line PC3 must be led through buffer to pin 4 of jumper field J7. In several modes of port 82C55A the port line PC3 acts as an interrupt output. This signal is not latched outside the port circuit. In the same way, an external TTL input signal connected to port line PC3 (Pin 17) of pin header X22 can be used for generating an interrupt.

8.8.4 Terminal Count Interrupt

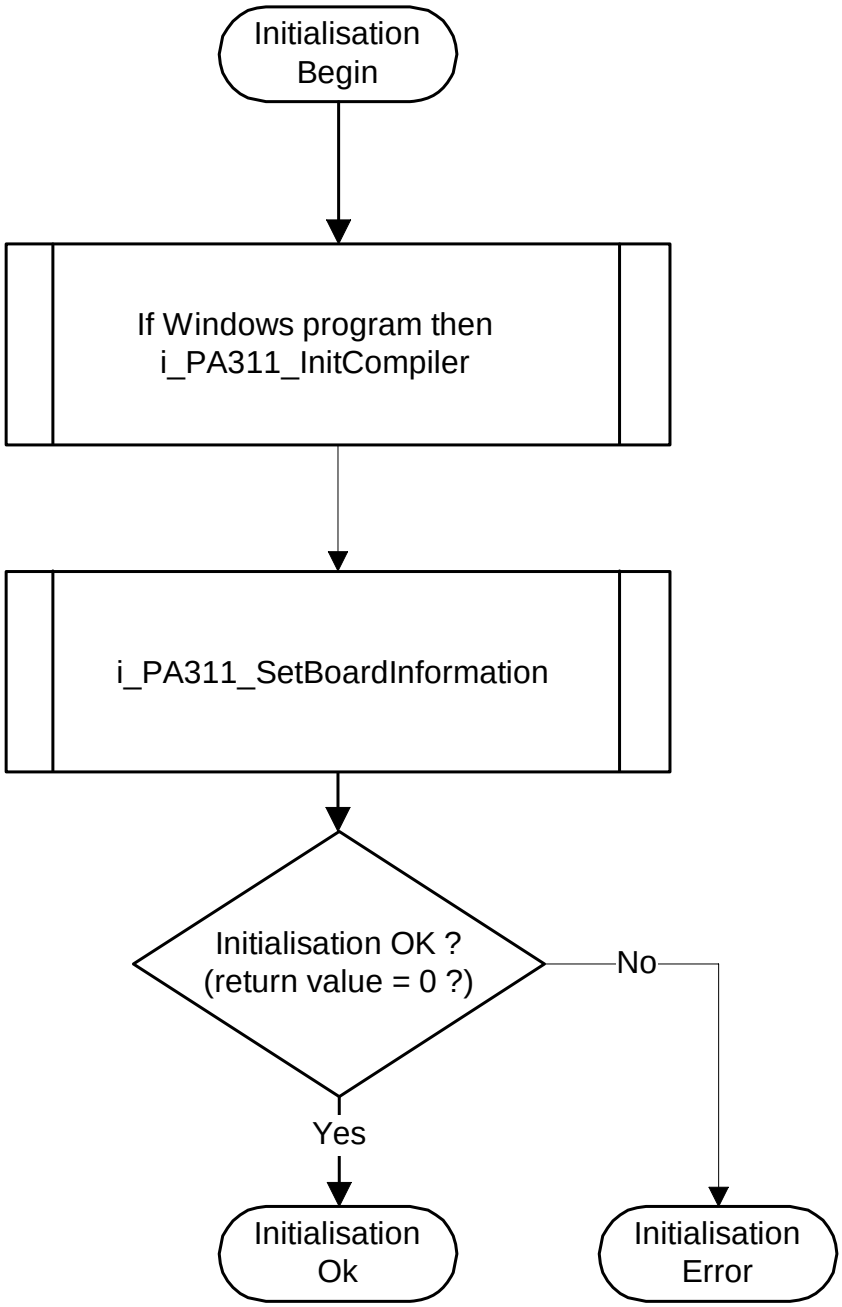
This interrupt is used with DMA. It is generated by the TC signal of the DMA controller and means that the last DMA transfer has occured. This interrupt is reset by a dummy reading on **Base +2**.

9 SOFTWARE EXAMPLES

IMPORTANT!
These **examples** are not the functions of a real-time application. They only represent the **possible functions** which can be procccesed with the board PA 311.

9.1 Initialisation

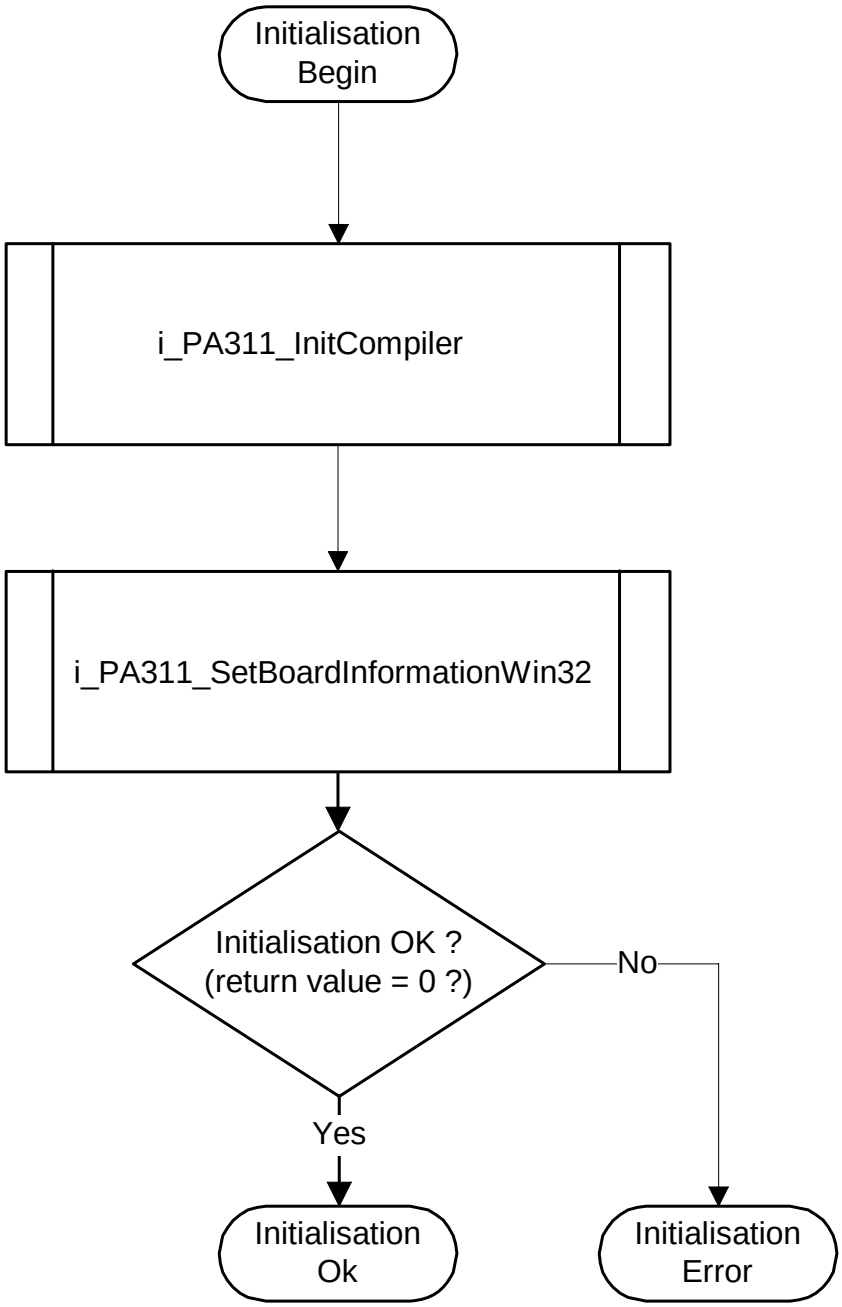
a) Flow chart for DOS and Windows 3.11



b) Example in C

```
int Initialisation(unsigned char *pb_BoardHandle)
    {
    #ifdef _Windows
       i_PA311_InitCompiler (DLL_COMPILER_C);
    #endif

    if(i_PA311_SetBoardInformation  (0x390,
                                     3,   // IRQ3
                                     5,   // IRQ5
                                     10,  // IRQ10
                                     11,  // IRQ11
                                     5,   // DMA5
                                     14,  // 14 Bit input resolution
                                     16,  // 16 analog inputs
                                     14,  // 14 Bit output resolution
                                     8,   // 8 analog outputs
                                     pb_BoardHandle) == 0)
       {
       return (0);         /* OK */
       }
    else
       {
       return (-1);        /* ERROR */
       }
    }
```

c) Flow chart for Windows NT/95

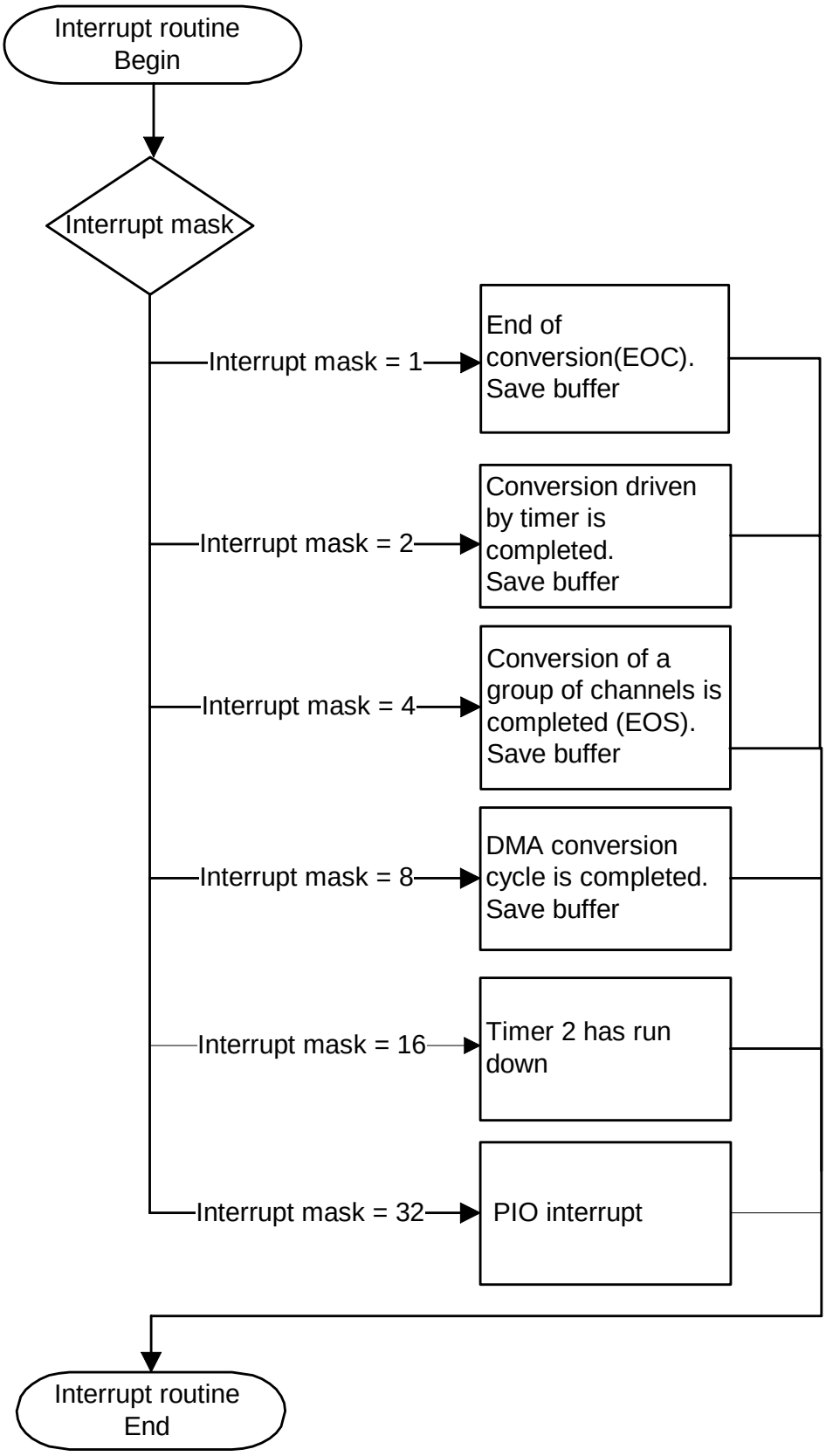


d) Example in C for Windows NT/95

```
int Initialisation(unsigned char *pb_BoardHandle)
    {
    if (i_PA311_InitCompiler (DLL_COMPILER_C) == 0)
       {
       if(i_PA311_SetBoardInformationWin32    (“PA311-00“,
                                               14,  // 14 Bit input resolution
                                               16,  // 16 analog inputs
                                               14,  // 14 Bit output resolution
                                               8,   // 8 analog outputs
                                               pb_BoardHandle) == 0)
          {
          return (0);      /* OK */
          }
       else
          {
          return (-1);     /* ERROR */
          }
       }
    else
       {
       return (-1);        /* ERROR */
       }
    }
```

9.2 Interrupt

a) Flow chart



b) Example in C for DOS and Windows 3.11

```
unsigned int  ui_SaveArray [16];        /* Global buffer                */
unsigned int ui_TimerIntCpt       = 0;  /* Timer interrupt counter*/
unsigned char b_ReceiveInterrupt  = 0;  /* Interrupt flag               */

_VOID_ v_InterruptRoutine (BYTE_ b_BoardHandle, BYTE_ b_InterruptMask, PUINT_
pui_ValueArray)
     {
     unsigned int ui_Cpt;

     switch(b_InterruptMask)
        {
        case 1:
                /* EOC interrupt */
               ui_SaveArray[0] = pui_ValueArray[1];
               break;
         case 2:
               /* Timer conversion */
                for (ui_Cpt=0; ui_Cpt < pui_ValueArray [0]; ui_Cpt++)
                  ui_SaveArray [ui_Cpt] = pui_ValueArray [ui_Cpt + 1];
               break;
         case 4:
               /* EOS interrupt */
                for (ui_Cpt=0; ui_Cpt < pui_ValueArray [0]; ui_Cpt++)
                  ui_SaveArray [ui_Cpt] = pui_ValueArray [ui_Cpt + 1];
               break;
         case 8:
               /* DMA completed */
               for (ui_Cpt=0;ui_Cpt<16;ui_Cpt++)
                  ui_SaveArray [ui_Cpt] = pui_ValueArray [ui_Cpt];
               break;
          case 16:
               /* Timer 2 has run down */
               ui_TimerIntCpt = ui_TimerIntCpt + 1;
               break;
         default :
               break;
        }
   b_ReceiveInterrupt = 1;
   }
```

c) Example in C for Windows NT/95 (asynchronous mode)

```
unsigned int ui_SaveArray [16];         /* Global Buffer */
unsigned int ui_TimerIntCpt = 0;        /* Timer interrupt counter */
unsigned char b_ReceiveInterrupt = 0;   /* Interrupt flag */

_VOID_ v_InterruptRoutine ( BYTE_ b_BoardHandle, BYTE_ b_InterruptMask,PUINT_
pui_ValueArray,
                            BYTE_ b_UserCallingMode,VOID *pv_UserSharedMemory)
   {
   unsigned long ul_Cpt;
   unsigned short int *pusi_Index;

   pusi_Index = (unsigned short int *) pui_ValueArray;

   switch(b_InterruptMask)
        {
        case 1:
                /* EOC interrupt */
               ui_SaveArray[0] = pui_ValueArray[1];
               break;
         case 2:
               /* Timer conversion */
               for (ui_Cpt=0; ui_Cpt < pui_ValueArray [0]; ui_Cpt++)
                  ui_SaveArray [ui_Cpt] = pui_ValueArray [ui_Cpt + 1];
               break;
         case 4:
               /* EOS interrupt */
               for (ui_Cpt=0; ui_Cpt < pui_ValueArray [0]; ui_Cpt++)
                  ui_SaveArray [ui_Cpt] = pui_ValueArray [ui_Cpt + 1];
               break;
         case 8:
              /* DMA completed */
              for (ui_Cpt=0;ui_Cpt<ul_NbrAcquisitionDMA;ui_Cpt++)
                 ui_SaveArray [ui_Cpt] = pusi_Index[ui_Cpt];
              break;
          case 16:
               /* Timer 2 has run down */
               ui_TimerIntCpt = ui_TimerIntCpt + 1;
               break;
         default :
               break;
        }
   b_ReceiveInterrupt = 1;
   }
```

d) Example in C for Windows NT/95 (synchronous mode)

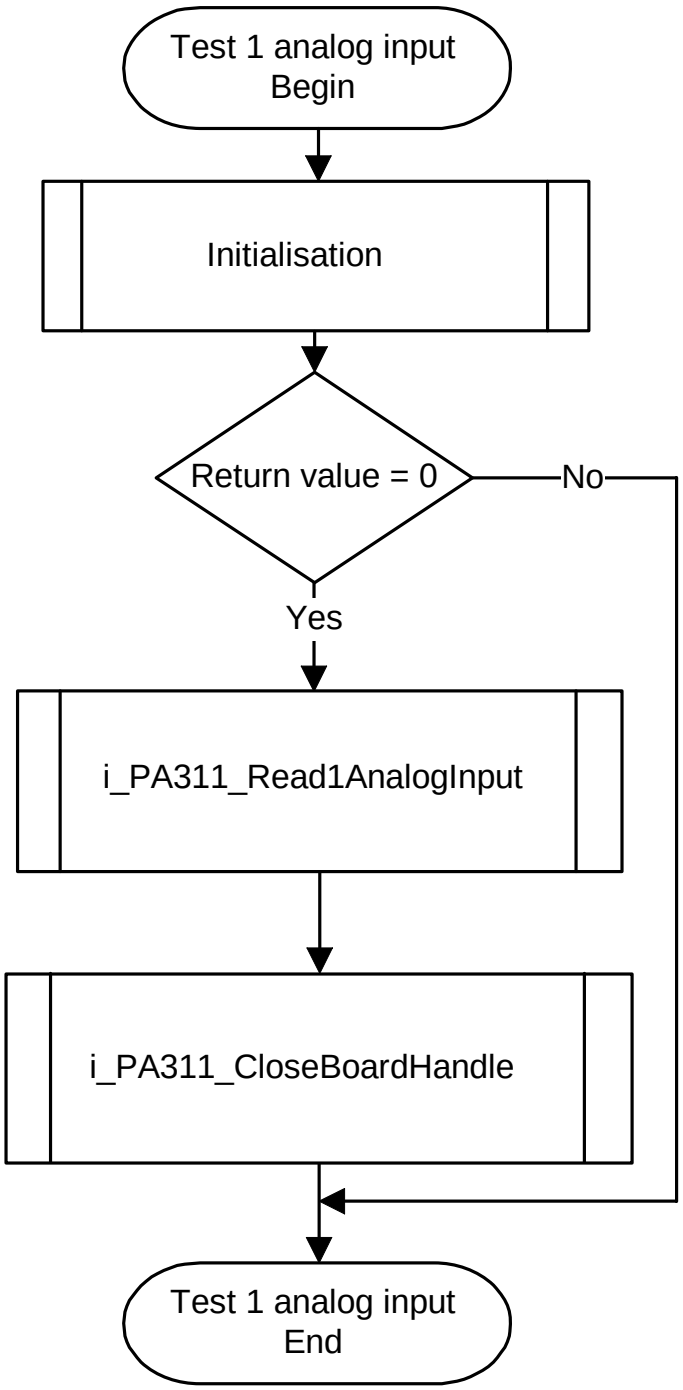
```
typedef struct
{
   unsigned int ui_SaveArray [16];      /* Global Buffer */
   unsigned int ui_TimerIntCpt ;        /* Timer interrupt counter */
   unsigned char b_ReceiveInterrupt ;   /* Interrupt flag */
}str_UserStruct;
str_UserStruct *ps_GlobalUserStruct;

_VOID_ v_InterruptRoutine ( BYTE_ b_BoardHandle,BYTE_ b_InterruptMask,
                            PUINT_ pui_ValueArray,
                           BYTE_ b_UserCallingMode,VOID *pv_UserSharedMemory)
   {
   unsigned int ui_Cpt;
   unsigned short int *pusi_Index;

   str_UserStruct *ps_UserStruct = (str_UserStruct *) pv_UserSharedMemory;
   pusi_Index = (unsigned short int *) pui_ValueArray;

   if ((b_InterruptMask&1) == 1) /* EOC interrupt */
      {
       ps_UserStruct->ui_SaveArray[0] = pui_ValueArray[1];
      }
   if ((b_InterruptMask & 2) == 2) /* EOS interrupt Acquisition */
      {
       for (ui_Cpt=0; ui_Cpt < pui_ValueArray [0]; ui_Cpt++)
           ps_UserStruct->ui_SaveArray [ui_Cpt] = pui_ValueArray [ui_Cpt + 1];
       }
   if ((b_InterruptMask & 4) == 4) /* Timer interrupt Acquisition */
      {
       for (ui_Cpt=0; ui_Cpt < pui_ValueArray [0]; ui_Cpt++)
           ps_UserStruct->ui_SaveArray [ui_Cpt] = pui_ValueArray [ui_Cpt + 1];
       }
   if ((b_InterruptMask & 8) == 8) /* DMA completed */
      {
       for (ui_Cpt=0;ui_Cpt<16;ui_Cpt++)
           ps_UserStruct->ui_SaveArray [ui_Cpt] = pusi_Index[ui_Cpt];
      }
   if ((b_InterruptMask & 16) == 16) /* Timer 2 has run down */
      {
       ps_UserStruct->ui_TimerIntCpt = ps_UserStruct->ui_TimerIntCpt + 1;
      }
   ps_UserStruct->b_ReceiveInterrupt =ps_UserStruct->b_ReceiveInterrupt + 1;
   }
```

9.3 Direct conversion of analog input channels

9.3.1 Testing one analog input channel

The input channels are in single mode. (See jumper settings).
For input channel 1: Pin 28 is set to 0 V
Pin 21 is set to a voltage between 0 and 10 V.

a) Flow chart



b) Example in C

```
void main (void)
     {
     unsigned char b_BoardHandle;
     unsigned int  ui_ReadValue;

     if (Initialisation (&b_BoardHandle) == 0)
        {
        if (i_PA311_Read1AnalogInput    (b_BoardHandle,
                                         PA311_CHANNEL_1,
                                         PA311_1_GAIN,
                                         PA311_UNIPOLAR,
                                         PA311_DISABLE,
                                         &ui_ReadValue)  == 0)
           {
           printf ("ui_ReadValue = %u", ui_ReadValue);
           }
        else
           {
           printf ("Read value error");
           }
        i_PA311_CloseBoardHandle (b_BoardHandle);
        }
     else
        {
        printf ("Initialisation error");
        }
     }
```

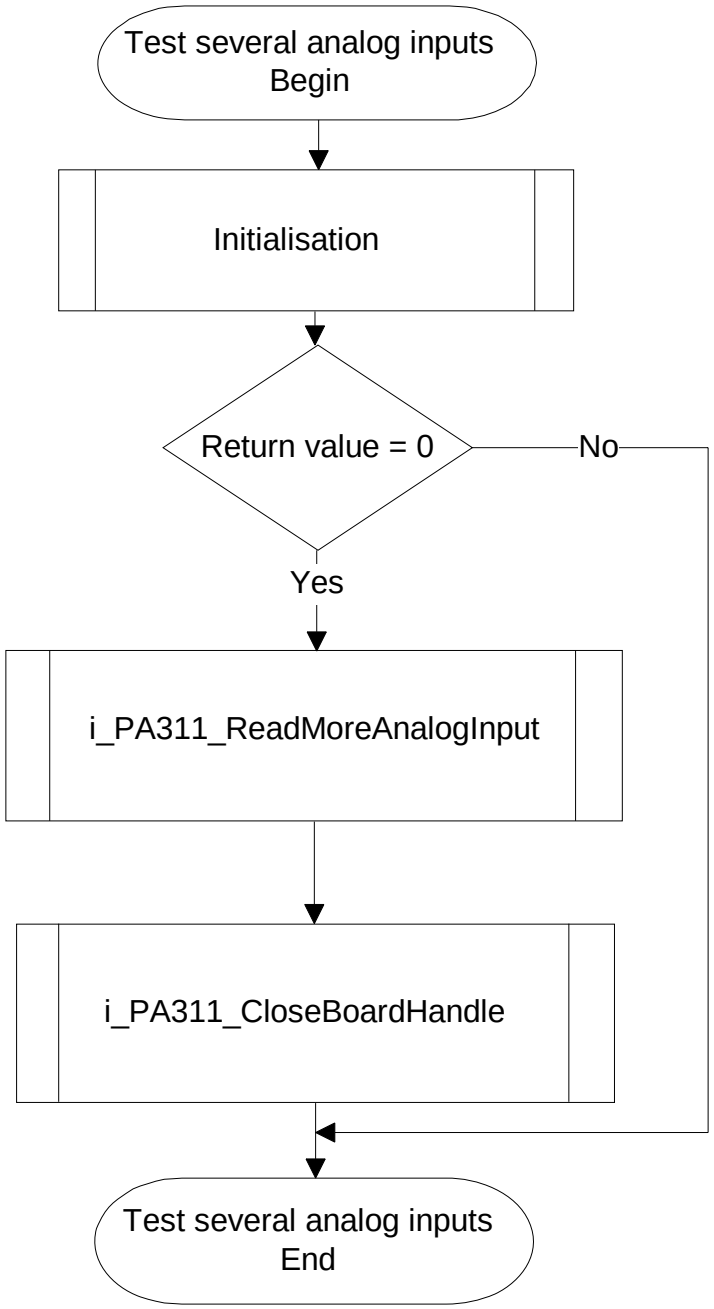
9.3.2 Testing several analog input channels

The input channels are in single mode.

Pin 28: GND: Set to 0 V	Pin 20: Set the an. input 0 to 10 V.
Pin 21: Set the an. input 1 to 10 V.	Pin 22: Set the an. input 2 to 10 V.
Pin 23: Set the an. input 3 to 10 V	Pin 27: Set the an. input 4 to 10 V.
Pin 26: Set the an. input 5 to 10 V	Pin 25: Set the an. input 6 to 10 V.
Pin 24: Set the an. input 7 to 10 V	

The value to be read is 13383 for each input.

a) Flow chart



b) Example in C

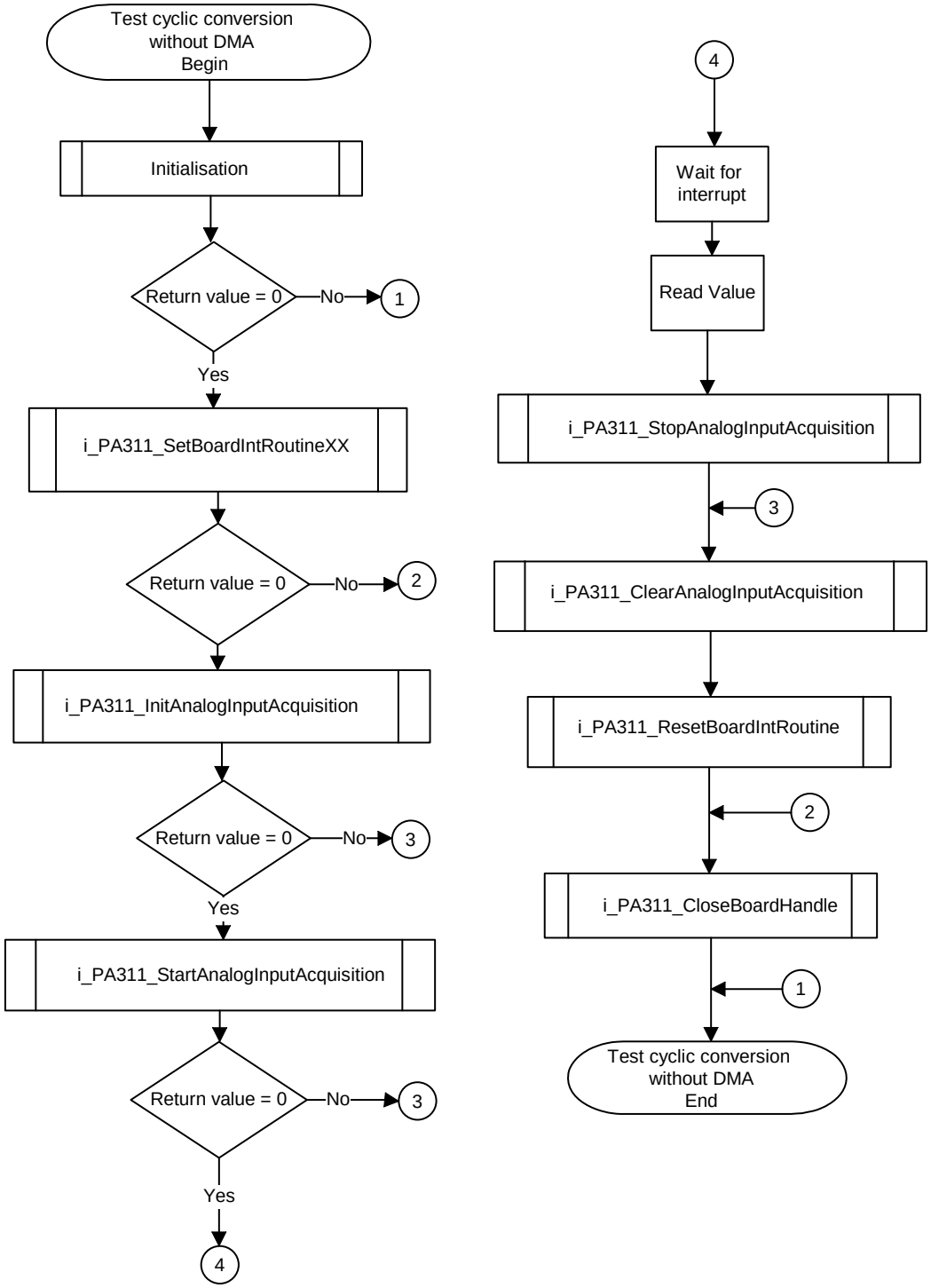
```
void main (void)
    {
    unsigned char b_BoardHandle;
    unsigned char b_Gain            [8];
    unsigned char b_Polar           [8];
    unsigned char b_Channel         [8];
    unsigned int  ui_ReadValueArray [8];

    b_Channel[0] = PA311_CHANNEL_0; b_Gain[0] = PA311_1_GAIN; b_Polar[0] = PA311_UNIPOLAR;
    b_Channel[1] = PA311_CHANNEL_1; b_Gain[1] = PA311_1_GAIN; b_Polar[1] = PA311_UNIPOLAR;
    b_Channel[2] = PA311_CHANNEL_2; b_Gain[2] = PA311_1_GAIN; b_Polar[2] = PA311_UNIPOLAR;
    b_Channel[3] = PA311_CHANNEL_3; b_Gain[3] = PA311_1_GAIN; b_Polar[3] = PA311_UNIPOLAR;
    b_Channel[4] = PA311_CHANNEL_4; b_Gain[4] = PA311_1_GAIN; b_Polar[4] = PA311_UNIPOLAR;
    b_Channel[5] = PA311_CHANNEL_5; b_Gain[5] = PA311_1_GAIN; b_Polar[5] = PA311_UNIPOLAR;
    b_Channel[6] = PA311_CHANNEL_6; b_Gain[6] = PA311_1_GAIN; b_Polar[6] = PA311_UNIPOLAR;
    b_Channel[7] = PA311_CHANNEL_7; b_Gain[7] = PA311_1_GAIN; b_Polar[7] = PA311_UNIPOLAR;

    if (Initialisation (&b_BoardHandle) == 0)
       {
       if (i_PA311_ReadMoreAnalogInput       (b_BoardHandle, 8, b_Channel,  b_Gain,
                                              b_Polar, PA311_DISABLE,
                                              ui_ReadValueArray)  == 0)
          {
          printf ("ui_ReadValue = %u %u %u %u %u %u %u %u",
                  ui_ReadValue [0], ui_ReadValue [1], ui_ReadValue [2], ui_ReadValue [3],
                  ui_ReadValue [4], ui_ReadValue [5], ui_ReadValue [6], ui_ReadValue [7]);
          }
       else
          {
          printf ("Read value error");
          }
       i_PA311_CloseBoardHandle (b_BoardHandle);
       }
    else
       {
       printf ("Initialisation error");
       }
    }
```

9.4 Cyclic conversion of analog input channels

9.4.1 Cyclic conversion without DMA and delay

The analog input channels are in single mode.
Pin 28: GND: Set to 0 V. Pin 20: Set input 0 to 10 V
Pin 21: Set input 1 to 10 V Pin 22: Set input 2 to 10 V
Pin 23: Set input 3 to 10 V
The value to be read is 16383 for each input channel.

a) Flow chart



b) Example in C for DOS

```
void main(void)
   {
   int i_Cpt;
   unsigned char b_BoardHandle;

   if (Initialisation(&b_BoardHandle) == 0)
      {
      if (i_PA311_SetBoardIntRoutineDos(b_BoardHandle, v_InterruptRoutine) == 0)
        {
        if (i_PA311_InitAnalogInputAcquisition (b_BoardHandle,4, PA311_1_GAIN,
                                                PA311_UNIPOLAR,
                                                16, PA311_SIMPLE_MODUS,
                                                PA311_SINGLE,
                                                PA311_HIGH_FREQUENCY,1000,
                                                PA311_HIGH_FREQUENCY, 0
                                                PA311_DMA_NOT_USED)==0)
           {
           b_ReceiveInterrupt = 0;
           if (i_PA311_StartAnalogInputAcquisition (b_BoardHandle) == 0)
              {
              for (i_Cpt=0;i_Cpt<4;i_Cpt++)
                {
                while (b_ReceiveInterrupt == 0);
                b_ReceiveInterrupt = 0;
                printf("\n Acquisition 1 %u %u %u %u",ui_SaveArray[0],
                            ui_SaveArray[1], ui_SaveArray[2], ui_SaveArray[3]);
                }
              i_PA311_StopAnalogInputAcquisition(b_BoardHandle);
              }
           else
              {
              printf("\n Start acquisition error");
              }
           i_PA311_ClearAnalogInputAcquisition(b_BoardHandle);
           }
        else
           {
           printf("\n Acquisition initialisation error");
           }
        i_PA311_ResetBoardIntRoutine(b_BoardHandle);
        }
      else
        printf("\n Interrupt routine initialisation error");
      i_PA311_CloseBoardHandle(b_BoardHandle);
      }
   else
      printf("\n Initialisation error");
   }
```

c) Example in C for Windows 3.11

```
void main(void)
   {
   int i_Cpt;
   unsigned char b_BoardHandle;

   if (Initialisation(&b_BoardHandle) == 0)
      {
      if (i_PA311_SetBoardIntRoutineWin16(b_BoardHandle, v_InterruptRoutine) == 0)
        {
        if (i_PA311_InitAnalogInputAcquisition (b_BoardHandle,4, PA311_1_GAIN,
                                                PA311_UNIPOLAR, 16, PA311_SIMPLE_MODUS,
                                                PA311_SINGLE, PA311_HIGH_FREQUENCY,1000,
                                                PA311_HIGH_FREQUENCY, 0
                                                PA311_DMA_NOT_USED)==0)
          {
          b_ReceiveInterrupt = 0;
          if (i_PA311_StartAnalogInputAcquisition (b_BoardHandle) == 0)
             {
             for (i_Cpt=0;i_Cpt<4;i_Cpt++)
               {
               while (b_ReceiveInterrupt == 0);
               b_ReceiveInterrupt = 0;
               printf("\n Acquisition 1 %u %u %u %u",ui_SaveArray[0], ui_SaveArray[1],
                                                    ui_SaveArray[2], ui_SaveArray[3]);
               }
             i_PA311_StopAnalogInputAcquisition(b_BoardHandle);
             }
          else
             printf("\n Start acquisition error");
          i_PA311_ClearAnalogInputAcquisition(b_BoardHandle);
          }
        else
          printf("\n Acquisition initialisation error");
        i_PA311_ResetBoardIntRoutine(b_BoardHandle);
        }
      else
        printf("\n Interrupt routine initialisation error");
      i_PA311_CloseBoardHandle(b_BoardHandle);
      }
   else
      printf("\n Initialisation error");
   }
```

d) Example in C for Windows NT/95 (asynchronous mode)

```
void main(void)
   {
   int i_Cpt;
   unsigned char b_BoardHandle;

   if (Initialisation(&b_BoardHandle) == 0)
      {
      if (i_PA311_SetBoardIntRoutineWin32(b_BoardHandle,PA311_ASYNHRONOUS_MODE,0,NULL,
                                          v_InterruptRoutine) == 0)
        {
        if (i_PA311_InitAnalogInputAcquisition (b_BoardHandle,4, PA311_1_GAIN,
                                                PA311_UNIPOLAR,16, PA311_SIMPLE_MODUS,
                                                PA311_SINGLE, PA311_HIGH_FREQUENCY,1000,
                                                PA311_HIGH_FREQUENCY, 0
                                                PA311_DMA_NOT_USED)==0)
           {
           b_ReceiveInterrupt = 0;
           if (i_PA311_StartAnalogInputAcquisition (b_BoardHandle) == 0)
              {
              for (i_Cpt=0;i_Cpt<4;i_Cpt++)
                {
                while (b_ReceiveInterrupt == 0);
                b_ReceiveInterrupt = 0;
                printf("\n Acquisition 1 %u %u %u %u",ui_SaveArray[0], ui_SaveArray[1],
                                                     ui_SaveArray[2], ui_SaveArray[3]);
                }
              i_PA311_StopAnalogInputAcquisition(b_BoardHandle);
              }
           else
              printf("\n Start acquisition error");
           i_PA311_ClearAnalogInputAcquisition(b_BoardHandle);
           }
        else
           printf("\n Acquisition initialisation error");
        i_PA311_ResetBoardIntRoutine(b_BoardHandle);
        }
      else
        printf("\n Interrupt routine initialisation error");
      i_PA311_CloseBoardHandle(b_BoardHandle);
      }
   else
      printf("\n Initialisation error");
   }
```

e) Example in C for Windows NT/95 (synchronous mode)

```
void main(void)
   {
   int i_Cpt;
   unsigned char b_BoardHandle;

   if (Initialisation(&b_BoardHandle) == 0)
      {
      if (i_PA311_SetBoardIntRoutineWin32(b_BoardHandle,PA311_SYNHRONOUS_MODE,
                                          sizeof(str_UserStruct),
                                          (void **) &ps_GlobalUserStruct,
                                          v_InterruptRoutine) == 0)
        {
        if (i_PA311_InitAnalogInputAcquisition (b_BoardHandle,4, PA311_1_GAIN,
                                                PA311_UNIPOLAR,16, PA311_SIMPLE_MODUS,
                                                PA311_SINGLE, PA311_HIGH_FREQUENCY,
                                                1000, PA311_HIGH_FREQUENCY, 0
                                                PA311_DMA_NOT_USED)==0)
           {
           ps_GlobalUserStruct -> b_ReceiveInterrupt = 0;
           if (i_PA311_StartAnalogInputAcquisition (b_BoardHandle) == 0)
              {
              for (i_Cpt=0;i_Cpt<4;i_Cpt++)
                {
                while (ps_GlobalUserStruct -> b_ReceiveInterrupt == 0);
                ps_GlobalUserStruct -> b_ReceiveInterrupt = 0;
                printf("\n Acquisition 1 %u %u %u %u", ps_GlobalUserStruct ->
                       ui_SaveArray[0], ps_GlobalUserStruct -> ui_SaveArray[1],
                       ps_GlobalUserStruct -> ui_SaveArray[2],
                       ps_GlobalUserStruct -> ui_SaveArray[3]);
                }
              i_PA311_StopAnalogInputAcquisition(b_BoardHandle);
              }
           else
              printf("\n Start acquisition error");
           i_PA311_ClearAnalogInputAcquisition(b_BoardHandle);
           }
        else
           printf("\n Acquisition initialisation error");
        i_PA311_ResetBoardIntRoutine(b_BoardHandle);
        }
      else
        printf("\n Interrupt routine initialisation error");
      i_PA311_CloseBoardHandle(b_BoardHandle);
      }
   else
      printf("\n Initialisation error");
   }
```

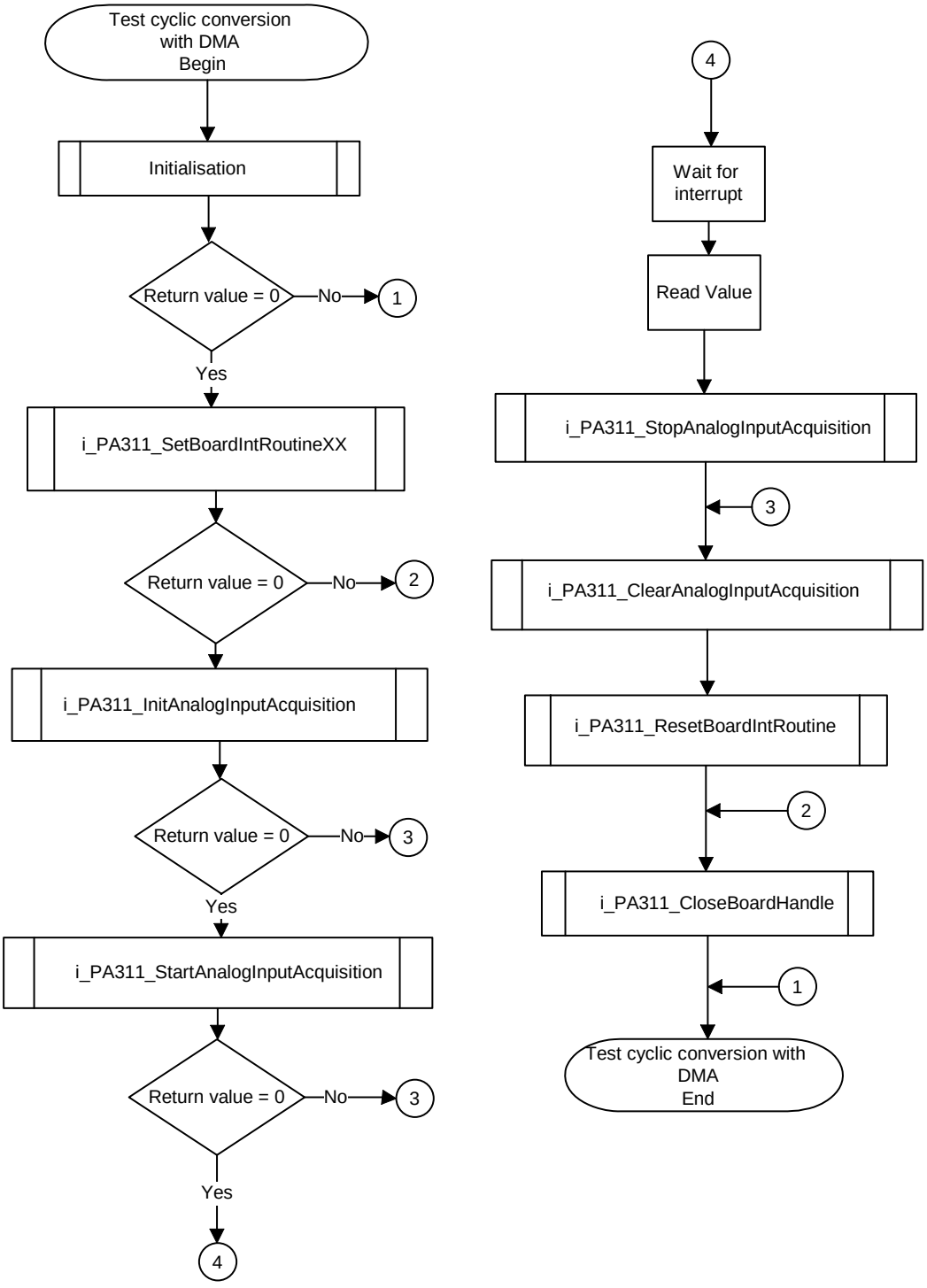
9.4.2 Cyclic conversion with DMA without delay

The analog input channels are in single mode.

Pin 28: GND: Set to 0 V. Pin 20: Set input 0 to 10 V
Pin 21: Set input 1 to 10 V Pin 22: Set input 2 to 10 V
Pin 23: Set input 3 to 10 V

The value to be read is 16383 for each input channel.

a) Flow chart



b) Example in C for DOS

```
void main(void)
   {
   int i_Cpt;
   unsigned char b_BoardHandle;

   if (Initialisation(&b_BoardHandle) == 0)
      {
      if (i_PA311_SetBoardIntRoutineDos(b_BoardHandle, v_InterruptRoutine) == 0)
        {
        if (i_PA311_InitAnalogInputAcquisition (b_BoardHandle,4, PA311_1_GAIN,
                                                PA311_UNIPOLAR,16,PA311_SIMPLE_MODUS,
                                                PA311_SINGLE, PA311_HIGH_FREQUENCY,10,
                                                PA311_HIGH_FREQUENCY, 0
                                                PA311_DMA_USED)==0)
           {
           b_ReceiveInterrupt = 0;
           if (i_PA311_StartAnalogInputAcquisition (b_BoardHandle) == 0)
              {
              while (b_ReceiveInterrupt == 0);
              b_ReceiveInterrupt = 0;
              printf("\n Acquisition 1 %u %u %u %u %u %u %u %u \n %u %u %u %u %u %u %u
%u",
                      ui_SaveArray[0],ui_SaveArray[1],ui_SaveArray[2],ui_SaveArray[3],
                      ui_SaveArray[4],ui_SaveArray[5],ui_SaveArray[6],ui_SaveArray[7],

ui_SaveArray[8],ui_SaveArray[9],ui_SaveArray[10],ui_SaveArray[11],

ui_SaveArray[12],ui_SaveArray[13],ui_SaveArray[14],ui_SaveArray[15]);
              i_PA311_StopAnalogInputAcquisition(b_BoardHandle);
              i_PA311_ClearAnalogInputAcquisition(b_BoardHandle);
              }
           else
              printf("\n Start acquisition error");
           }
        else
           printf("\n Acquisition initialisation error");
        i_PA311_ResetBoardIntRoutine(b_BoardHandle);
        }
      else
        printf("\n Interrupt routine initialisation error");
      i_PA311_CloseBoardHandle(b_BoardHandle);
      }
   else
      printf("\n Initialisation error");
   }
```

c) Example in C for Windows 3.11

```
void main(void)
   {
   int i_Cpt;
   unsigned char b_BoardHandle;

   if (Initialisation(&b_BoardHandle) == 0)
      {
      if (i_PA311_SetBoardIntRoutineWin16(b_BoardHandle, v_InterruptRoutine) == 0)
        {
        if (i_PA311_InitAnalogInputAcquisition (b_BoardHandle,4, PA311_1_GAIN,
PA311_UNIPOLAR,
                                                16, PA311_SIMPLE_MODUS, PA311_SINGLE,
                                                PA311_HIGH_FREQUENCY,10,
PA311_HIGH_FREQUENCY, 0
                                                PA311_DMA_USED)==0)
          {
          b_ReceiveInterrupt = 0;
          if (i_PA311_StartAnalogInputAcquisition (b_BoardHandle) == 0)
             {
             while (b_ReceiveInterrupt == 0);
             b_ReceiveInterrupt = 0;
             printf("\n Acquisition 1 %u %u %u %u %u %u %u %u \n %u %u %u %u %u %u %u %u",
                      ui_SaveArray[0],ui_SaveArray[1],ui_SaveArray[2],ui_SaveArray[3],
                      ui_SaveArray[4],ui_SaveArray[5],ui_SaveArray[6],ui_SaveArray[7],
                      ui_SaveArray[8],ui_SaveArray[9],ui_SaveArray[10],ui_SaveArray[11],
                     ui_SaveArray[12],ui_SaveArray[13],ui_SaveArray[14],ui_SaveArray[15]);
             i_PA311_StopAnalogInputAcquisition(b_BoardHandle);
             i_PA311_ClearAnalogInputAcquisition(b_BoardHandle);
             }
          else
             printf("\n Start acquisition error");
          }
        else
           printf("\n Acquisition initialisation error");
        i_PA311_ResetBoardIntRoutine(b_BoardHandle);
        }
      else
        printf("\n Interrupt routine initialisation error");
      i_PA311_CloseBoardHandle(b_BoardHandle);
      }
   else
      printf("\n Initialisation error");
   }
```

d) Example in C for Windows NT/95 (asynchronous mode)

```
void main(void)
   {
   int i_Cpt;
   unsigned char b_BoardHandle;

   if (Initialisation(&b_BoardHandle) == 0)
      {
      if (i_PA311_SetBoardIntRoutineWin32(b_BoardHandle, PA311_ASYNHRONOUS_MODE,
                                          0,NULL,v_InterruptRoutine) == 0)
        {
        if (i_PA311_InitAnalogInputAcquisition (b_BoardHandle,4, PA311_1_GAIN,
                                                PA311_UNIPOLAR,
                                                16, PA311_SIMPLE_MODUS, PA311_SINGLE,
                                                PA311_HIGH_FREQUENCY,10,
PA311_HIGH_FREQUENCY, 0
                                                PA311_DMA_USED)==0)
           {
           b_ReceiveInterrupt = 0;
           if (i_PA311_StartAnalogInputAcquisition (b_BoardHandle) == 0)
              {
              while (b_ReceiveInterrupt == 0);
              b_ReceiveInterrupt = 0;
              printf("\n Acquisition 1 %u %u %u %u %u %u %u %u \n %u %u %u %u %u %u %u %u",
                      ui_SaveArray[0],ui_SaveArray[1],ui_SaveArray[2],ui_SaveArray[3],
                      ui_SaveArray[4],ui_SaveArray[5],ui_SaveArray[6],ui_SaveArray[7],
                      ui_SaveArray[8],ui_SaveArray[9],ui_SaveArray[10],ui_SaveArray[11],
                      i_SaveArray[12],ui_SaveArray[13],ui_SaveArray[14],ui_SaveArray[15]);
              i_PA311_StopAnalogInputAcquisition(b_BoardHandle);
              i_PA311_ClearAnalogInputAcquisition(b_BoardHandle);
              }
           else
              printf("\n Start acquisition error");
           }
        else
           printf("\n Acquisition initialisation error");
        i_PA311_ResetBoardIntRoutine(b_BoardHandle);
        }
      else
        printf("\n Interrupt routine initialisation error");
      i_PA311_CloseBoardHandle(b_BoardHandle);
      }
   else
      printf("\n Initialisation error");
   }
```

e) Example in C for Windows NT/95 (synchronous mode)

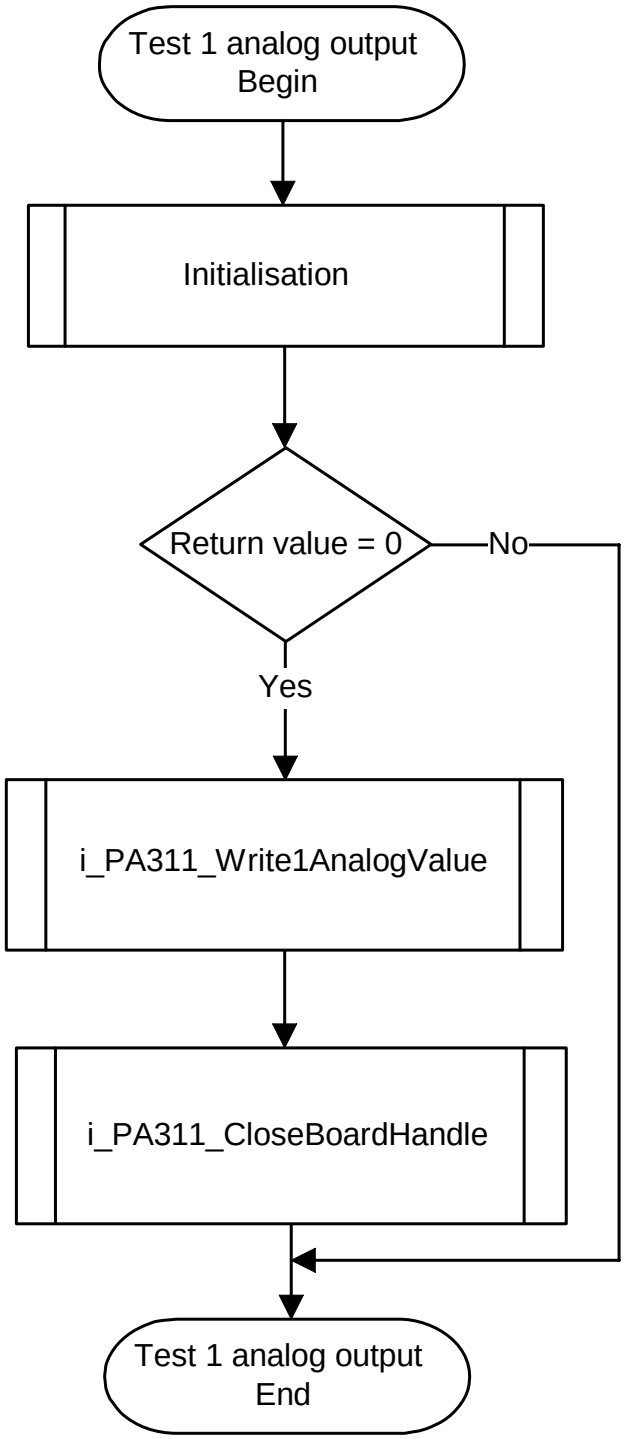
```
void main(void)
   {
   int i_Cpt;unsigned char b_Gain [4];unsigned char b_Polar [4];unsigned char b_Channel
[4];
   unsigned char b_BoardHandle;
   if (Initialisation(&b_BoardHandle) == 0)
      {
      if (i_PA311_SetBoardIntRoutineWin32(b_BoardHandle, PA311_SYNHRONOUS_MODE,
                                          sizeof(str_UserStruct),
                                          void **) &GlobalUserStruct, v_InterruptRoutine)
                                          == 0)
       {
       if (i_PA311_InitAnalogInputAcquisition (b_BoardHandle,4, PA311_1_GAIN,
                                               PA311_UNIPOLAR,
                                               16, PA311_SIMPLE_MODUS, PA311_SINGLE,
                                               PA311_HIGH_FREQUENCY,10,
                                               PA311_HIGH_FREQUENCY, 0
                                               PA311_DMA_USED)==0)
          {
          ps_GlobalUserStruct -> b_ReceiveInterrupt = 0;
          if (i_PA311_StartAnalogInputAcquisition (b_BoardHandle) == 0)
             {
             while (ps_GlobalUserStruct -> b_ReceiveInterrupt == 0);
      ps_GlobalUserStruct -> b_ReceiveInterrupt = 0;
      printf("\n Acquisition 1 %u %u %u %u %u %u %u %u \n %u %u %u %u %u %u %u %u",
         ps_GlobalUserStruct -> ui_SaveArray[0],
         ps_GlobalUserStruct -> ui_SaveArray[1], ps_GlobalUserStruct -> ui_SaveArray[2],
         ps_GlobalUserStruct -> ui_SaveArray[3],ps_GlobalUserStruct -> ui_SaveArray[4],
         ps_GlobalUserStruct -> ui_SaveArray[5], ps_GlobalUserStruct -> ui_SaveArray[6],
         ps_GlobalUserStruct -> ui_SaveArray[7], ps_GlobalUserStruct -> ui_SaveArray[8],
         ps_GlobalUserStruct -> ui_SaveArray[9], ps_GlobalUserStruct -> ui_SaveArray[10],
        ps_GlobalUserStruct -> ui_SaveArray[11], ps_GlobalUserStruct -> ui_SaveArray[12],
        ps_GlobalUserStruct -> ui_SaveArray[13], ps_GlobalUserStruct -> ui_SaveArray[14],
        ps_GlobalUserStruct -> ui_SaveArray[15]);
             i_PA311_StopAnalogInputAcquisition(b_BoardHandle);
             i_PA311_ClearAnalogInputAcquisition(b_BoardHandle);
             }
          else
             printf("\n Start acquisition error");
          }
       else
          printf("\n Acquisition initialisation error");
       i_PA311_ResetBoardIntRoutine(b_BoardHandle);
       }
      else
       printf("\n Interrupt routine initialisation error");
      i_PA311_CloseBoardHandle(b_BoardHandle);
      }
   else
      printf("\n Initialisation error");
   }
```

9.5 Analog output channels

9.5.1 Testing one analog output channel

Write 1 analog output.
Connect a voltmeter: minus to Pin 30 (analog output 0 GND) plus to Pin 12 (analog output 0).
The voltage to mesure is 10V.

a) Flow chart

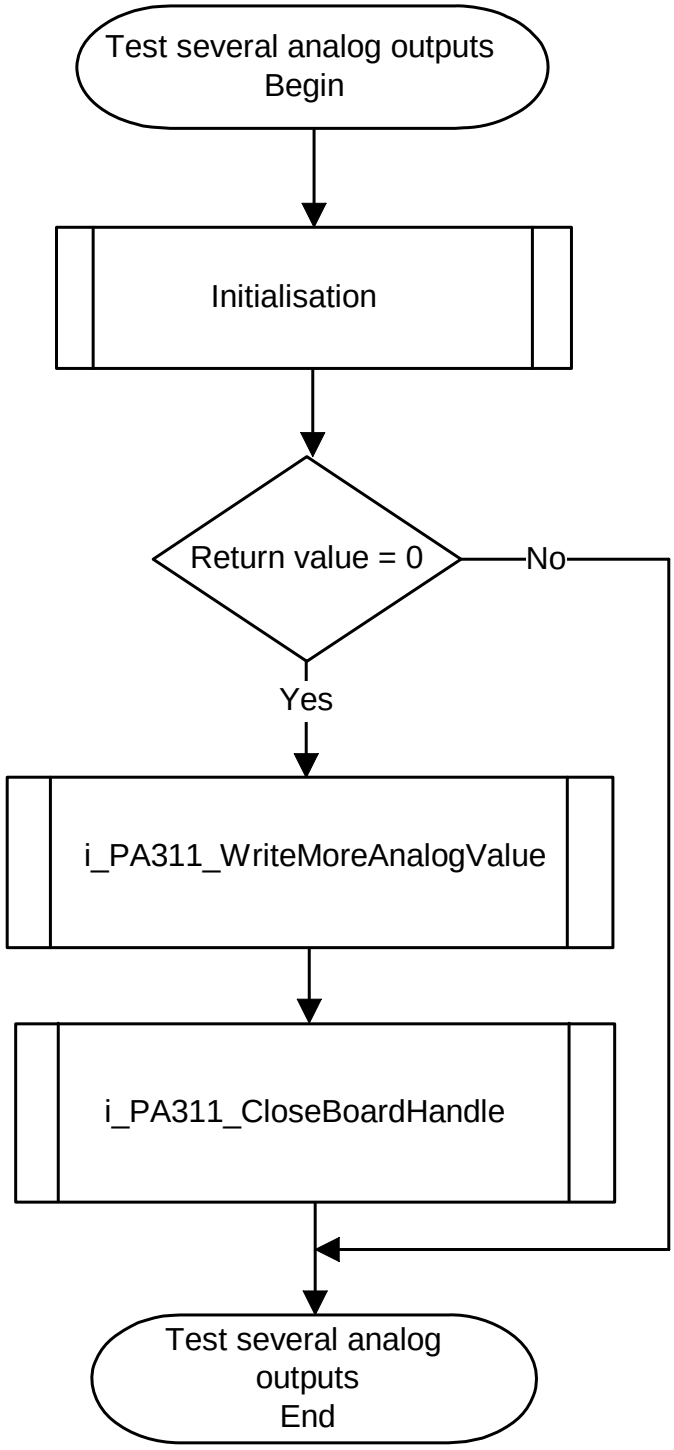


b) example in C

```
void main (void)
     {
     unsigned char b_BoardHandle;

     if (Initialisation (&b_BoardHandle) == 0)
        {
        if (i_PA311_WriteAnalogValue    (b_BoardHandle,
                                         0,
                                         16383)  == 0)
           {
           printf (“Write test OK“);
           }
        else
           {
           printf (“Write value error“);
           }
        i_PA311_CloseBoardHandle (b_BoardHandle);
        }
     else
        {
        printf (“Initialisation error“);
        }
     }
```

9.5.2 Testing several output channel

a) Flow chart



Write more analog output:
Output 0: Connect a voltmeter: minus to Pin 30 (analog output 0 GND)
plus to Pin 12 (analog output 0).
The voltage to mesure is 0V.
Output 1: Connect a voltmeter: minus to Pin 31 (analog output 1 GND)
plus to Pin 13 (analog output 1).
The voltage to mesure is 5V.

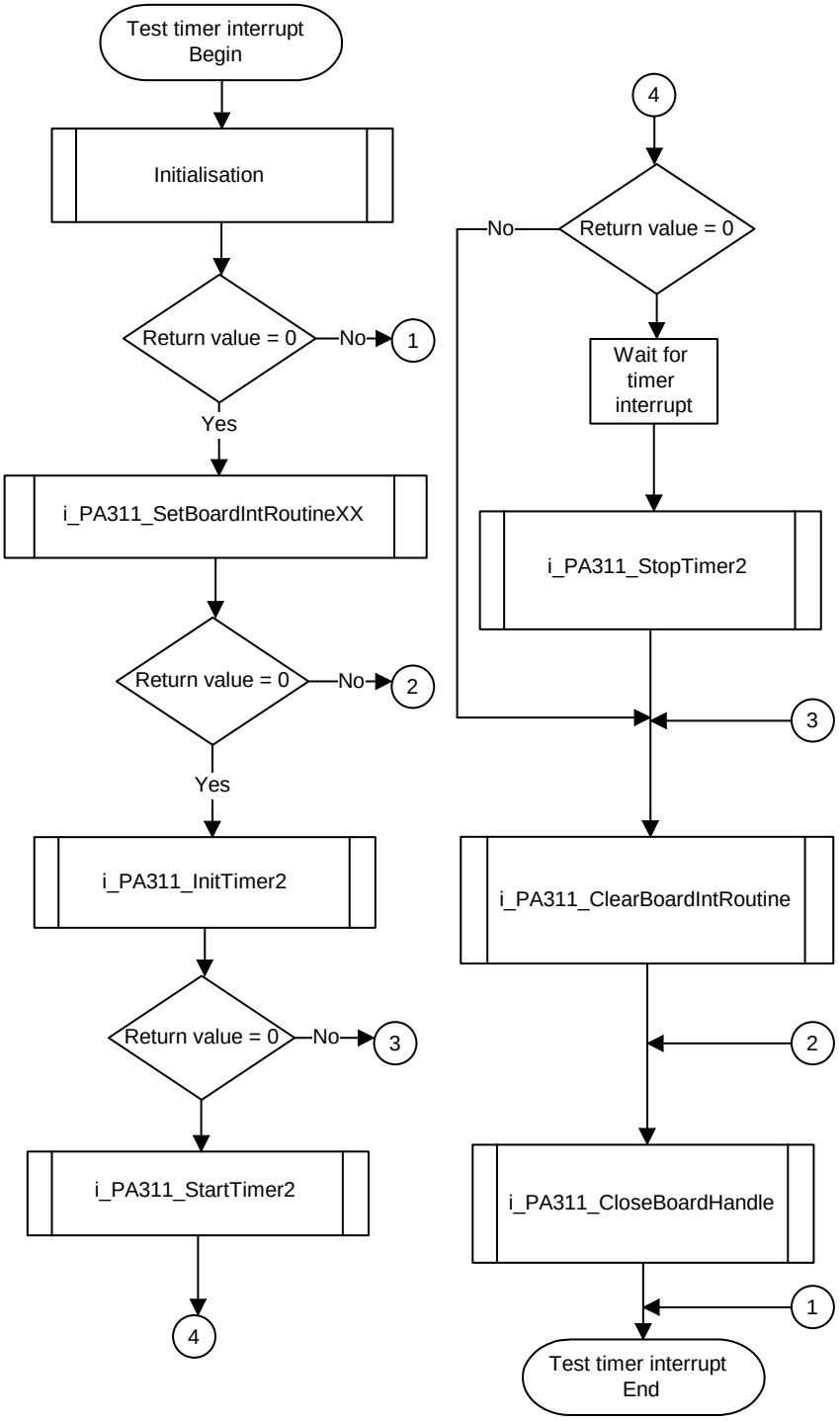
Output 2: Connect a voltmeter: minus to Pin 32 (analog output 2 GND)
plus to Pin 14 (analog output 2).
The voltage to mesure is 10V.
Output 3: Connect a voltmeter: minus to Pin 33 (analog output 3 GND)
plus to Pin 15 (analog output 3).
The voltage to mesure is 0V.
Output 4: Connect a voltmeter: minus to Pin 34 (analog output 4 GND)
plus to Pin 16 (analog output 4).
The voltage to mesure is 5V.
Output 5: Connect a voltmeter: minus to Pin 35 (analog output 5 GND)
plus to Pin 17 (analog output 5).
The voltage to mesure is 10V.
Output 6: Connect a voltmeter: minus to Pin 36 (analog output 6 GND)
plus to Pin 18 (analog output 6).
The voltage to mesure is 0V.
Output 7: Connect a voltmeter: minus to Pin 37 (analog output 7 GND)
plus to Pin 19 (analog output 7).
The voltage to mesure is 5V.

b) example in C

```
void main (void)
     {
     unsigned char b_BoardHandle;
     unsigned char b_Channel            [8];
     unsigned int  ui_WriteValueArray   [8];

     b_Channel[0] = 0; ui_WriteValueArray [0] = 0;
     b_Channel[1] = 1; ui_WriteValueArray [1] = 8192;
     b_Channel[2] = 2; ui_WriteValueArray [2] = 16383;
     b_Channel[3] = 3; ui_WriteValueArray [3] = 0;
     b_Channel[4] = 4; ui_WriteValueArray [4] = 8192;
     b_Channel[5] = 5; ui_WriteValueArray [5] = 16383;
     b_Channel[6] = 6; ui_WriteValueArray [6] = 0;
     b_Channel[7] = 7; ui_WriteValueArray [7] = 16383;

     if (Initialisation (&b_BoardHandle) == 0)
        {
        if (i_PA311_WriteMoreAnalogValue      (b_BoardHandle, 0, 8,
                                               ui_WriteValueArray)  == 0)
           {
           printf ("Write test OK");
           }
        else
           {
           printf ("Write value error");
           }

        i_PA311_CloseBoardHandle (b_BoardHandle);
        }
     else
        {
        printf ("Initialisation error");
        }
     }
```

9.6 Timer

a) Flow chart



b) Example in C for DOS

```
void main (void)
     {
     unsigned char b_BoardHandle;

     if (Initialisation (&b_BoardHandle) == 0)
        {
        if (i_PA311_SetBoardIntRoutineDOS (b_BoardHandle, v_InterruptRoutine) == 0)
           {
           ui_TimerIntCpt = 0;
           if (i_PA311_InitTimer2      (b_BoardHandle, PA311_LOW_FREQUENCY,
                                        1000, PA311_ENABLE) == 0)
              {
              if (i_PA311_StartTimer2 (b_BoardHandle) == 0)
                 {
                 while (ui_TimerIntCpt == 0);
                 printf ("Receive timer interrupt");
                 i_PA311_StopTimer2 (b_BoardHandle);
                 }
              else
                 printf ("Start timer error");
              }
           else
              printf ("Init timer error");
           i_PA311_ResetBoardIntRoutine (b_BoardHandle);
           }
        else
           printf ("Interrupt routine initialisation error");
        i_PA311_CloseBoardHandle (b_BoardHandle);
        }
     else
        printf ("Initialisation error");
     }
```

c) Example in C for Wiydow 3.1x

```
void main (void)
     {
     unsigned char b_BoardHandle;

     if (Initialisation (&b_BoardHandle) == 0)
        {
        if (i_PA311_SetBoardIntRoutineWin16 (b_BoardHandle, v_InterruptRoutine) == 0)
           {
           ui_TimerIntCpt = 0;
           if (i_PA311_InitTimer2       (b_BoardHandle, PA311_LOW_FREQUENCY,
                                         1000, PA311_ENABLE) == 0)
              {
              if (i_PA311_StartTimer2 (b_BoardHandle) == 0)
                 {
                 while (ui_TimerIntCpt == 0);
                 printf ("Receive timer interrupt");
                 i_PA311_StopTimer2 (b_BoardHandle);
                 }
              else
                 {
                 printf ("Start timer error");
                 }
              }
           else
              {
              printf ("Init timer error");
              }
           i_PA311_ResetBoardIntRoutine (b_BoardHandle);
           }
        else
           {
           printf ("Interrupt routine initialisation error");
           }
        i_PA311_CloseBoardHandle (b_BoardHandle);
        }
     else
        {
        printf ("Initialisation error");
        }
     }
```

d) Example in C for Window NT/95 (asynchronous mode)

```
void main (void)
     {
     unsigned char b_BoardHandle;

     if (Initialisation (&b_BoardHandle) == 0)
        {
        if (i_PA311_SetBoardIntRoutineWin32 (b_BoardHandle,PA311_ASYNCHRONOUS_MODE,
                                             0,NULL,v_InterruptRoutine) == 0)
           {
           ui_TimerIntCpt = 0;
           if (i_PA311_InitTimer2       (b_BoardHandle, PA311_LOW_FREQUENCY,
                                         1000, PA311_ENABLE) == 0)
              {
              if (i_PA311_StartTimer2 (b_BoardHandle) == 0)
                 {
                 while (ui_TimerIntCpt == 0);
                 printf ("Receive timer interrupt");
                 i_PA311_StopTimer2 (b_BoardHandle);
                 }
              else
                 printf ("Start timer error");
              }
           else
              printf ("Init timer error");
           i_PA311_ResetBoardIntRoutine (b_BoardHandle);
           }
        else
           printf ("Interrupt routine initialisation error");
        i_PA311_CloseBoardHandle (b_BoardHandle);
        }
     else
         printf ("Initialisation error");
     }
```

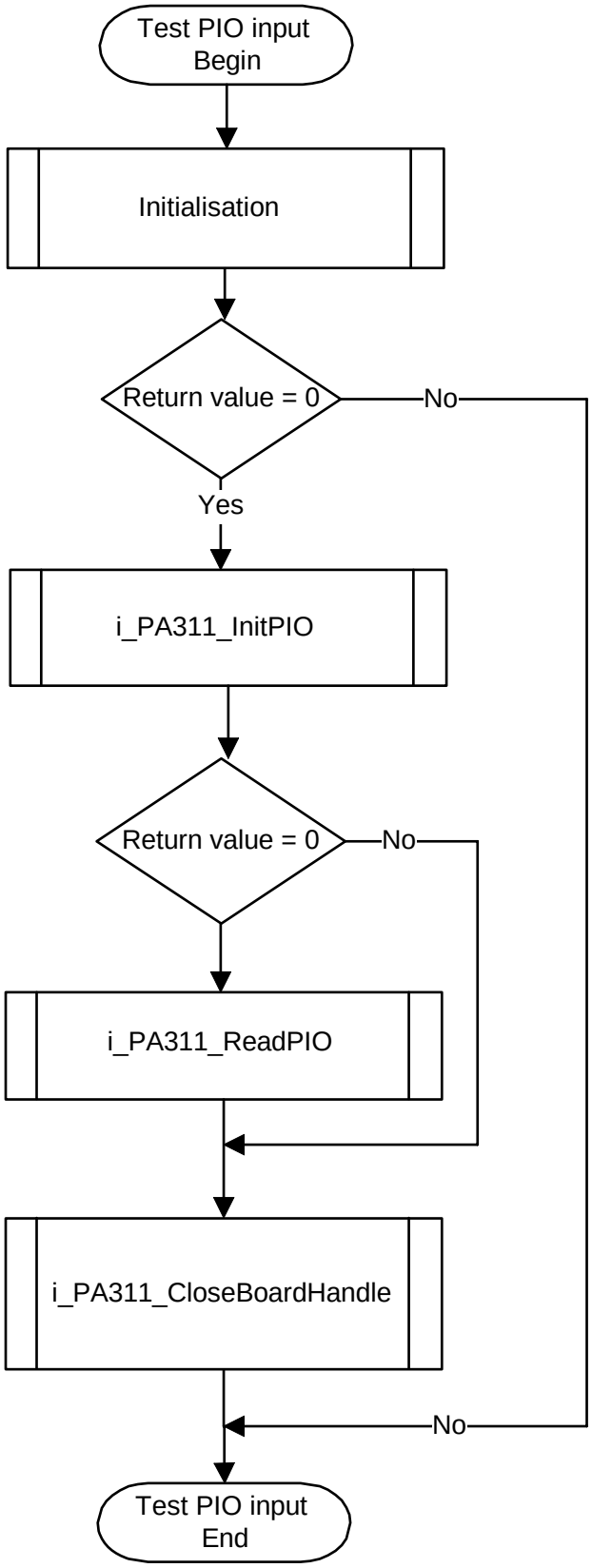
e) Example in C for Window NT/95 (synchronous mode)

```
void main (void)
     {
     unsigned char b_BoardHandle;

     if (Initialisation (&b_BoardHandle) == 0)
        {
        if (i_PA311_SetBoardIntRoutineWin32 (b_BoardHandle,PA311_SYNCHRONOUS_MODE,
           sizeof(str_UserStruct),(void **) &ps_GlobalUserStruct,v_InterruptRoutine) == 0)
           {
           ps_GlobalUserStruct->ui_TimerIntCpt = 0;
           if (i_PA311_InitTimer2       (b_BoardHandle, PA311_LOW_FREQUENCY,
                                         1000, PA311_ENABLE) == 0)
              {
              if (i_PA311_StartTimer2 (b_BoardHandle) == 0)
                 {
                 while (ps_GlobalUserStruct->ui_TimerIntCpt == 0);
                 printf ("Receive timer interrupt");
                 i_PA311_StopTimer2 (b_BoardHandle);
                 }
              else
                 {
                 printf ("Start timer error");
                 }
              }
           else
              {
              printf ("Init timer error");
              }
           i_PA311_ResetBoardIntRoutine (b_BoardHandle);
           }
        else
           {
           printf ("Interrupt routine initialisation error");
           }
        i_PA311_CloseBoardHandle (b_BoardHandle);
        }
     else
        {
        printf ("Initialisation error");
        }
     }
```

9.7 PIO

9.7.1 PIO input channels

a) Flow chart

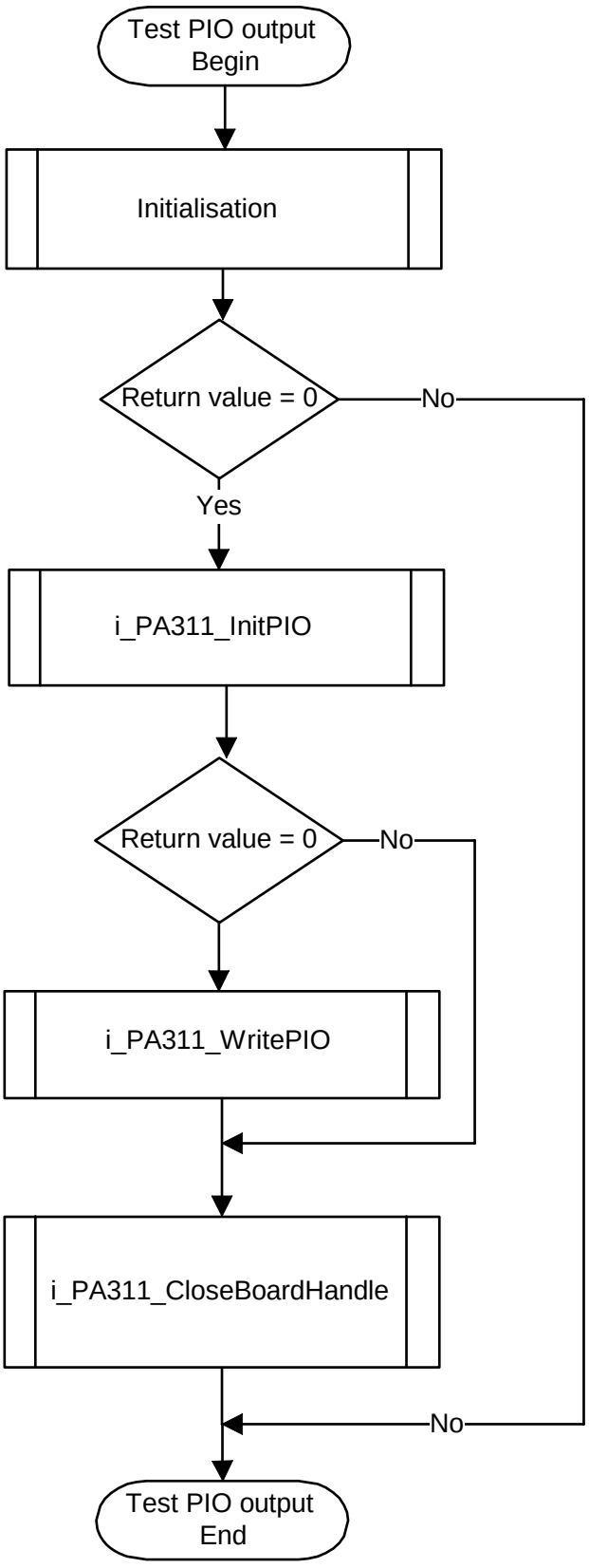


b) Example in C

```
void main (void)
     {
     unsigned char b_BoardHandle;
     unsigned char b_ReadValue;

     if (Initialisation (&b_BoardHandle) == 0)
        {
        if (i_PA311_InitPIO      (b_BoardHandle, 1, 1, 1, 1) == 0)
            {
            if (i_PA311_ReadPio (b_BoardHandle, 0, &b_ReadValue) == 0)
               {
               printf ("Port A = %X Hex", b_ReadValue);
               }
            else
               {
               printf ("Read PIO error");
               }
            }
        else
            {
            printf ("Init PIO error");
            }

        i_PA311_CloseBoardHandle (b_BoardHandle);
        }
     else
        printf ("Initialisation error");
     }
```

9.7.2 PIO output channels

a) Flow chart



b) Example in C

```
void main (void)
     {
     unsigned char b_BoardHandle;

     if (Initialisation (&b_BoardHandle) == 0)
        {
        if (i_PA311_InitPIO      (b_BoardHandle, 0, 0, 0, 0) == 0)
            {
            if (i_PA311_WritePio (b_BoardHandle, 0, 0x55) == 0)
               {
               printf ("Write port A = 55 Hex");
               }
            else
               {
               printf ("Read PIO error");
               }
            }
        else
            {
            printf ("Init PIO error");
            }

        i_PA311_CloseBoardHandle (b_BoardHandle);
        }
     else
        printf ("Initialisation error");
     }
```

INDEX

ADDIREG 20–24
  changing the configuration 24

analog data acquisition
  operating modes 36
  options 49–51

base address 16

board
  functions 27–60
  handling 3
  insert 18
  options 5
  physical set-up 4
  slot 4
  versions 5

component scheme 9

connecction
  pin assignment 25

connection
  examples 26
  peripheral 25

connection
  output channels 52

conversion 40–48
  start 35

conversion driven by timer 45–48

differential mode 37

DIP switches 16

DMA 60

end of conversion
  interrupt 60

functions 27–60

gain factor
  examples 39

installation 12–99
  Windows NT/95 19
  DOS and Windows 3.11 19

intended purpose 1

Internet 24

interrupt 60

jumpers
  location 11
  settings at delivery 11

limit values 5–8

operating modes
  analog data acquisition 36

output channels 51–55
  connection 52
  structure 51

parallel inputs and outputs
  component 82C55A 56
  interrupt 56
  programming 56
  programming examples 57

PC
  close 18
  open 17

peripheral
  connection 25

PIO see parallel input and output

single-ended
  with INA 37
  without INA 36

slot
  select 17
  types, figure 17

software
  delivered 19

software conversion 40–44

delayed 41–42
direct 43–44
single conversion 41

technical data 4–10

terminal count interrupt 60

timer 57–59
interrupt 58
programming 58
programming examples 58

timer interrupt 60

use
limits 1