



Technical support:
+49 (0)7223 / 9493-0

Software description

ADDIDRIVER

Analog inputs

Edition: 07.03- 07/2006

1	INTRODUCTION	4
2	ANALOG INPUTS	5
2.1	General information	5
1)	b_ADDIDATA_GetNumberOfAnalogInputs (..)	5
2)	b_ADDIDATA_GetNumberOfAnalogInputModules (..)	6
3)	b_ADDIDATA_GetNumberOfAnalogInputsForTheModule (..)	7
4)	b_ADDIDATA_GetAnalogInputModuleNumber (..)	8
5)	b_ADDIDATA_GetAnalogInputInformation (..)	9
6)	b_ADDIDATA_GetAnalogInputModuleGeneralInformation (..)	18
7)	b_ADDIDATA_GetAnalogInputModuleSingleAcquisitionInformation (..)	21
8)	b_ADDIDATA_GetAnalogInputModuleAutoRefreshInformation (..)	23
9)	b_ADDIDATA_GetAnalogInputModuleSCANInformation (..)	25
10)	b_ADDIDATA_GetAnalogInputModuleSequenceInformation (..)	28
2.2	Initialisation	32
1)	b_ADDIDATA_InitAnalogInput (..)	32
	Table 2-1: Gain/Polarity for a max. voltage of 5 V	33
2)	b_ADDIDATA_ReleaseAnalogInput (..)	34
2.3	Single acquisition	35
1)	b_ADDIDATA_Read1AnalogInput (..)	35
2)	b_ADDIDATA_ReadMoreAnalogInputs (..)	37
3)	b_ADDIDATA_ConvertDigitalToRealAnalogValue (..)	39
4)	b_ADDIDATA_ConvertMoreDigitalToRealAnalogValues (..)	40
2.4	Autorefresh acquisition	41
1)	b_ADDIDATA_GetAnalogInputAutoRefreshChannelPointer (..)	41
2)	b_ADDIDATA_GetAnalogInputAutoRefreshModulePointer (..)	42
3)	b_ADDIDATA_GetAnalogInputAutoRefreshModuleCounterPointer (..)	43
4)	b_ADDIDATA_StartAnalogInputAutoRefresh (..)	44
5)	b_ADDIDATA_StopAnalogInputAutoRefresh(..)	45
2.5	SCAN acquisition	46
1)	b_ADDIDATA_InitAnalogInputSCAN (...)	46
2)	b_ADDIDATA_InitAnalogInputSCANAcquisition (..)	49
3)	b_ADDIDATA_StartAnalogInputSCAN (..)	54
4)	b_ADDIDATA_GetAnalogInputSCANStatus (..)	55
5)	b_ADDIDATA_ConvertDigitalToRealAnalogValueSCAN (..)	56
6)	b_ADDIDATA_StopAnalogInputSCAN (..)	57
7)	b_ADDIDATA_CloseAnalogInputSCAN (..)	58
2.6	Sequence acquisition	59
1)	b_ADDIDATA_InitAnalogInputSequenceAcquisition (..)	59
2)	b_ADDIDATA_StartAnalogInputSequenceAcquisition (..)	64
3)	b_ADDIDATA_ConvertDigitalToRealAnalogValueSequence(..)	66
4)	b_ADDIDATA_GetAnalogInputSequenceAcquisitionHandleStatus (..)	67
5)	b_ADDIDATA_StopAnalogInputSequenceAcquisition (..)	69
6)	b_ADDIDATA_ReleaseAnalogInputSequenceAcquisition (..)	70
2.7	Hardware trigger	71
1)	b_ADDIDATA_GetAnalogInputHardwareTriggerInformation (..)	71
2)	b_ADDIDATA_EnableDisableAnalogInputHardwareTrigger (...)	73

3) b_ADDIDATA_GetAnalogInputHardwareTriggerStatus (...)	75
2.8 Software trigger functions	77
1) b_ADDIDATA_GetAnalogInputSoftwareTriggerInformation (...)	77
2) b_ADDIDATA_EnableDisableAnalogInputSoftwareTrigger (...)	79
3) b_ADDIDATA_GetAnalogInputSoftwareTriggerStatus (...)	81
4) b_ADDIDATA_AnalogInputSoftwareTrigger (...)	82
2.9 Other functions	83
1) b_ADDIDATA_TestAnalogInputAsynchronousFIFOFull.....	83

Tables

Table 1-1: Type Declaration for Windows 98/NT/2000/XP	4
Table 2-1: Gain/Polarity for a max. voltage of 5 V	33
Table 2-2: Selection of the conversion time	36
Table 2-3: Time selection of a SCAN	47
Table 2-4: SCAN time mode	47
Table 2-5: SCAN mode	48
Table 2-6: Trigger mode	48

1 INTRODUCTION

i

IMPORTANT!

Note the following conventions in the text:

Function: "b_ADDIDATA_GetNumberOfAnalogInputs"
Variable *dw_DriverHandle*

Table 1-1: Type Declaration for Windows 98/NT/2000/XP

	Borland C	Microsoft C	Borland Pascal	Microsoft Visual Basic Windows
VOID	void	void	pointer	any
BYTE	unsigned char	unsigned char	byte	integer
INT	int	int	integer	integer
WORD	unsigned short int	unsigned short int	long	long
DWORD	long	long	longint	long
PBYTE	unsigned char *	unsigned char *	var byte	integer
PINT	int *	int *	var integer	integer
PWORD	unsigned short int *	unsigned short int *	var long	long
PCHAR	char *	char *	var string	string
PDWORD	long *	long *	var longint	long
DOUBLE	double	double	double	double

2 ANALOG INPUTS

2.1 General information

1) b_ADDIDATA_GetNumberOfAnalogInputs (..)

Syntax:

```
<Return value> = b_ADDIDATA_GetNumberOfAnalogInputs
                    (DWORD   dw_DriverHandle,
                     PWORD   pw_ChannelNbr)
```

Parameters:

- Input:

DWORD *dw_DriverHandle* Driver handle

- Output:

PWORD *pw_ChannelNbr* Returns the number of analog inputs

Task:

Returns the number of analog inputs.

Calling convention:

ANSI C:

```
BYTE      b_ReturnValue;
DWORD     dw_DriverHandle;
WORD      w_ChannelNbr;
```

```
b_ReturnValue = b_ADDIDATA_GetNumberOfAnalogInputs (dw_DriverHandle,
                                                    &w_ChannelNbr);
```

Return value:

1: No error

0: Error by calling up the function. Use the function "i_ADDIDATA_GetLastError",
to find the error source.

2) b_ADDIDATA_GetNumberOfAnalogInputModules (..)**Syntax:**

<Return value> = b_ADDIDATA_GetNumberOfAnalogInputModules
(DWORD dw_DriverHandle,
PWORD pw_ModuleNbr)

Parameter:**- Input:**

DWORD *dw_DriverHandle* Driver handle

- Output:

PWORD *pw_ModuleNbr* Returns the number of analog input modules.

Task:

Returns the number of analog input modules.

Calling convention:ANSI C :

BYTE b_ReturnValue;
DWORD dw_DriverHandle;
WORD w_ModuleNbr;

b_ReturnValue = b_ADDIDATA_GetNumberOfAnalogInputModules
(dw_DriverHandle,
&w_ModuleNbr);

Return value:

1: No error

0: Error by calling up the function. Use the function "i_ADDIDATA_GetLastError",
to find the error source.

3) **b_ADDIDATA_GetNumberOfAnalogInputsForTheModule (..)**

Syntax:

```
<Return value> = b_ADDIDATA_GetNumberOfAnalogInputsForTheModule
                (DWORD    dw_DriverHandle,
                 WORD      w_Module,
                 PWORD     pw_ChannelNbr)
```

Parameters:

- Input:

DWORD	<i>dw_DriverHandle</i>	Driver handle
WORD	<i>w_Module</i>	Analog input module

- Output:

PWORD	<i>pw_ChannelNbr</i>	Returns the number of analog input channels for the selected module.
-------	----------------------	--

Task:

Returns the number of analog input channels for the module selected with *w_Module*.

Calling convention:

ANSI C :

BYTE	b_ReturnValue;
DWORD	dw_DriverHandle;
WORD	w_ChannelNbr;

[illegible]

Return value:

1: No error

0: Error by calling up the function. Use the function "i_ADDIDATA_GetLastError",
to find the error source.

4) b_ADDIDATA_GetAnalogInputModuleNumber (..)

Syntax:

```
<Return value> = b_ADDIDATA_GetAnalogInputModuleNumber
                    (DWORD    dw_DriverHandle,
                     WORD     w_Channel,
                     PWORD    pw_Module)
```

Input:

DWORD	dw_DriverHandle	Driver handle
WORD	w_Channel	Analog input channel

Output:

WORD pw_Module	Returns the analog input module number from selected channel.
-----------------------	---

Task:

Return the analog input module index (pw_Module) from selected channel (w_Channel)

Calling convention:

ANSI C :

```

BYTE      b_ReturnValue;
DWORD     dw_DriverHandle;
WORD      w_Module;

```

```
b_ReturnValue = b_ADDIDATA_b_ADDIDATA_GetAnalogInputModuleNumber
                (dw_DriverHandle,
                 0,
                 &w_Module);
```

Return Value:

1: No error

0: Error by calling up the function. Use the function "i_ADDIDATA_GetLastError", to find the error source.

5) **b_ADDIDATA_GetAnalogInputInformation (..)****i****IMPORTANT!**

This function is only available for old user applications (up to version 2132-0304 of ADDIPACK).

If you develop new applications with ADDIDRIVER, please use the following functions:

b_ADDIDATA_GetAnalogInputModuleGeneralInformation
 b_ADDIDATA_GetAnalogInputModuleSingleAcquisitionInformation,
 b_ADDIDATA_GetAnalogInputModuleAutoRefreshInformation,
 b_ADDIDATA_GetAnalogInputModuleSCANInformation and
 b_ADDIDATA_GetAnalogInputModuleSequenceInformation.

Syntax:

```
<Return value> = b_ADDIDATA_GetAnalogInputInformation
                    (DWORD    dw_DriverHandle,
                     WORD     w_Channel,
                     pstr_GetAnalogMeasureInformation ps_ChannelInformation,
                     DWORD    dw_StructSize)
```

Parameters:**- Input:**

DWORD	<i>dw_DriverHandle</i>	Driver handle
WORD	<i>w_Channel</i>	Number of the analog input
DWORD	<i>dw_StructSize</i>	Size of the structure entered in the pointer.

- Output:

pstr_GetAnalogMeasureInformation	ps_ChannelInformation	Channel information.
----------------------------------	-----------------------	----------------------

structure returned :

```
typedef struct
```

```
{
  BYTE    b_InputsResolution;    Returns the input resolution
                                         8: 8-bit resolution
                                         16: 16-bit resolution, ...
  BYTE    b_CanUsedInterrupt;    0: No interrupt can be generated
                                         1: Interrupt can be generated
  BYTE    b_UnipolarBipolarConfigurable
                                         0: Unipolar/bipolar hardware configurable
                                         1: Unipolar/bipolar configurable
  BYTE    b_UnipolarAvailable;    0: Cannot configure in unipolar
                                         1: Can configure in unipolar
  BYTE    b_BipolarAvailable;    0: Cannot configure in bipolar
                                         1: Can configure in bipolar
  BYTE    b_SingleDifferenceSelected;
                                         0: Single mode selected
                                         1: Differential mode selected
  BYTE    b_DCCouplingAvailable;
                                         0: Cannot configure the direct coupling (DC)
                                         1: Can configure direct coupling (DC)
```

BYTE	<i>b_ACCouplingAvailable</i> ;	0: Cannot configure alternative coupling (AC) 1: Can configure alternative coupling (AC)
BYTE	<i>b_BufferAvailable</i> ;	0: No hardware buffer available 1: Hardware buffer available
BYTE	<i>b_CanGereneratedWarning</i> ;	0: Alarm not available 1: Alarm available
BYTE	<i>b_CurrentSourceSetBySoft</i> ;	0: The current source must not be activated by software 1: The current source must be activated by software.
BYTE	<i>b_NbrOfGain</i> ;	Returns the number of gain values available
DWORD	<i>dw_ModuleNumber</i>	Module number
DOUBLE	<i>d_GainAvailable</i> [255];	Defines the available gain value
BYTE	<i>b_OffsetRangeAvailable</i> ;	0: Offset not allowed 1: Offset allowed
BYTE	<i>b_Reserved2</i> ;	
WORD	<i>w_OffsetRangeResolution</i> ;	Offset resolution 8: 8-bit resolution 16: 16-bit resolution, ...
BYTE	<i>b_OffsetRangeDenominator</i> ;	Offset denominator step value
BYTE	<i>b_OffsetRangeNumerator</i> ;	Offset numerator step value
BYTE	<i>b_OpenInputDetection</i> ;	0: Open input detection not available 1: Open input detection available
BYTE	<i>b_ShortCircuitDetection</i> ;	0: Short-circuit detection not available 1: Short-circuit detection available
DOUBLE	<i>d_Umax</i> ;	Returns the maximum input voltage value in V or A
DOUBLE	<i>d_URef</i> ;	Returns the reference input voltage value in V or A
BYTE	<i>b_InputType</i> ;	Selected user input type ADDIDATA_RTD ADDIDATA_THERMOCOUPLE ADDIDATA_TEMPERATURE_ON_BOARD ADDIDATA_OHM ADDIDATA_ANALOG_INPUT
BYTE	<i>b_TypePrecision</i> ;	Precision of the input ADDIDATA_THERMOCOUPLE_TYPE_B ADDIDATA_THERMOCOUPLE_TYPE_E ADDIDATA_THERMOCOUPLE_TYPE_J ADDIDATA_THERMOCOUPLE_TYPE_K ADDIDATA_THERMOCOUPLE_TYPE_N ADDIDATA_THERMOCOUPLE_TYPE_R ADDIDATA_THERMOCOUPLE_TYPE_S ADDIDATA_THERMOCOUPLE_TYPE_T or ADDIDATA_RTD_TYPE_PT ADDIDATA_RTD_TYPE_Ni
WORD	<i>w_InputTypeValue</i> ;	If a Pt sensor is connected type of the sensor 100 → Pt100

Module Information

BYTE	<i>b_AutoCalibration;</i>	0: Auto-calibration not available 1: Auto-calibration available
BYTE	<i>b_CJCAvailable;</i>	0: Without CJC (cold junction compensation) 1: With CJC
BYTE	<i>b_ConversionMustSetting;</i>	0: The conversion time for a single acquisition is determined by hardware (jumper) 1: The conversion time for a single acquisition is determined by software
BYTE	<i>b_ConversionCalcType;</i>	0: Binary type (XX000,XX001,XX010,XX011,XX100) 1: Multiple type (60, 120, 240, 480, 960, ...)
BYTE	<i>b_ConversionUnitType;</i>	0: Time unit (ns, μ s, ms, s, ...) 1: Frequency unit (MHz, kHz, Hz, mHz, ...)
BYTE	<i>b_AvailableConversionUnit;</i>	For time unit: D0: 0: ns not available 1: ns available D1: 0: μ s not available 1: μ s available D2: 0: ms not available 1: ms available D3: 0: s not available 1: s available For frequency unit: D0: 0: MHz not available 1: MHz available D1: 0: kHz not available 1: kHz available D2: 0: Hz not available 1: Hz available D3: 0: mHz not available 1: mHz available
WORD	<i>w_ConversionResolution;</i>	8 : 8-bit resolution 16: 16-bit resolution, ...
WORD	<i>w_MinConversionTime;</i>	Minimum conversion time. 7000: 7000(ns) 10000: 10000(ns), ...
WORD	<i>w_ConversionStep;</i>	Conversion time steps 20: 20 steps 50: 50 steps, ...
BYTE	<i>b_SEQArrayAvailable;</i>	0: Sequence array not available 1: Sequence array available
BYTE	<i>b_SEQConfigurable;</i>	0: Sequence fixed 1: Sequence configurable
BYTE	<i>b_SEQHardwareTriggerAvailable;</i>	0: Hardware trigger not available 1: Hardware trigger available
BYTE	<i>b_SEQHardwareTriggerHighAvailable;</i>	0: Hardware high trigger level not available 1: Hardware high trigger level available

BYTE	<i>b_SEQHardwareTriggerLowAvailable;</i>	0: Low level not available for hardware trigger 1: Low level available for hardware trigger
BYTE	<i>b_SEQHardwareTriggerAvailableMode;</i>	001: External trigger start a 1 DMA cycle 010: Each trigger starts a sequence 100: Each trigger starts a DMA cycle
BYTE	<i>b_SEQHardwareGateAvailable;</i>	0: Hardware gate not available 1: Hardware gate available
BYTE	<i>b_SEQHardwareGateHighAvailable;</i>	0: High level not available for hardware gate 1: High level available for hardware gate
BYTE	<i>b_SEQHardwareGateLowAvailable;</i>	0: Low level not available for hardware gate 1: Low level available for hardware gate
BYTE	<i>b_SEQClrIndexAvailable;</i>	0: It is not possible to restart the acquisition with the next selected channel in the sequence 1: It's possible to restart the acquisition with the next selected channel in the sequence
WORD	<i>w_SEQAcquisitionMode;</i>	00: The acquisition can operate in single mode 01: The acquisition can operate in continuous mode 11: The acquisition can operate in single and continuous mode
BYTE	<i>b_DMAAvailable;</i>	0: Board cannot use the DMA 1: Board can use the DMA
BYTE	<i>b_DualDMAChannel;</i>	0: Only 1 DMA channel can be used 1: Board can use 2 DMA channels
BYTE	<i>b_SEQCounterAvailable;</i>	0: DMA conversion counter not available 1: DMA conversion counter available
BYTE	<i>b_SEQCounterMode;</i>	01: Counter can count the number of sequences 10: Counter can count the number of DMA cycles 11: Counter can count the number of DMA cycles or the number of sequences
BYTE	<i>b_SEQCommonGain;</i>	Defines if the gain can differ for each input 0: Gain is the same for each input 1: Gain can differ for each input.
BYTE	<i>b_SEQCommonPolarity;</i>	Defines if the polarity can differ for each input 0: Polarity is the same for each input 1: Polarity can differ for each input.
BYTE	<i>b_SEQCommonOffsetRange;</i>	Defines if the offset can differ for each input 0: Offset is the same for each input 1: Offset can differ for each input.
BYTE	<i>b_SEQCommonCoupling;</i>	
WORD	<i>w_SEQCounterResolution;</i>	Counter resolution 8 : 8-bit resolution 16: 16-bit resolution, ...
BYTE	<i>b_SEQDelayTimeConfigurable;</i>	0: Delay after each sequence not available 1: Delay after each sequence available

BYTE	<i>b_SEQDelayMode</i> ;	
BYTE	<i>b_SEQDelayCalcType</i> ;	0: Binary type (XX000,XX001,XX010,XX011,XX100) 1: Multiple type (60,120,240,480,960, ...)
BYTE	<i>b_SEQDelayTimeUnitType</i> ;	0: Time unit (ns, μ s, ms, s, ...) 1: Frequency unit (MHz, kHz, Hz, mHz, ...)
BYTE	<i>b_SEQDelayValueType</i> ;	0: Value to write in the register is in s or in Hz 1: The step multiplier is to be written in the register
BYTE	<i>b_SEQDelayTimeUnit</i> ;	For time unit: D0: 0: ns not available 1: ns available D1: 0: μ s not available 1: μ s available D2: 0: ms not available 1: ms available D3: 0: s not available 1: s available For frequency unit: D0: 0: MHz not available 1: MHz available D1: 0: kHz not available 1: kHz available D2: 0: Hz not available 1: Hz available D3: 0: mHz not available 1: mHz available
WORD	<i>w_SEQDelayTimeResolution</i> ;	Delay time resolution 8: 8-bit resolution 16: 16-bit resolution, ...
WORD	<i>w_SEQMinDelayTime</i> ;	Minimum delay time. 7000: 7000(ns) 10000: 10000(ns), ...
WORD	<i>w_SEQDelayTimeStep</i> ;	Conversion delay time steps 20: 20 steps 50: 50 steps, ...
BYTE	<i>b_SEQUnipolarBipolarConfigurable</i> ;	0: Unipolar/Bipolar hardware configurable 1: Unipolar/Bipolar configurable
BYTE	<i>b_SEQUnipolarAvailable</i> ;	0: Cannot configure in unipolar 1: Can configure in unipolar
BYTE	<i>b_SEQBipolarAvailable</i> ;	0: Cannot configure in bipolar 1: Can configure in bipolar
BYTE	<i>b_SEQDCCouplingAvailable</i> ;	0: Cannot configure the direct coupling (DC) 1: Can configure direct coupling (DC)
BYTE	<i>b_SEQACCouplingAvailable</i> ;	0: Cannot configure alternative coupling (AC) 1: Can configure alternative coupling (AC)
BYTE	<i>b_SEQBufferAvailable</i> ;	0: No hardware buffer available 1: Hardware buffer available

BYTE	<i>b_SEQNbrOfGain;</i>	Returns the number of gain values available
BYTE	<i>b_Reserved3;</i>	
WORD	<i>w_Reserved4;</i>	
DWORD	<i>dw_Reserved5;</i>	
DOUBLE	<i>d_SEQGainAvailable[255];</i>	Defines the available gain value
BYTE	<i>b_SEQOffsetRangeAvailable;</i>	0: Offset not allowed 1: Offset allowed
BYTE	<i>b_Reserved6;</i>	
WORD	<i>w_SEQOffsetRangeResolution;</i>	Offset resolution 8 : 8-bit resolution 16: 16-bit resolution, ...
BYTE	<i>b_SEQOffsetRangeDenominator;</i>	Offset denominator step value
BYTE	<i>b_SEQOffsetRangeNumerator;</i>	Offset numerator step value
BYTE	<i>b_SCANAvailable;</i>	0: SCAN not available 1: SCAN available
BYTE	<i>b_SCANConfigurable;</i>	0: SCAN fixed 1: SCAN configurable (The first and the last channel can be given)
BYTE	<i>b_SCANHardwareTriggerAvailable;</i>	0: Hardware trigger not available 1: Hardware trigger available
BYTE	<i>b_SCANHardwareTriggerHighAvailable;</i>	0: Hardware high trigger level not available 1: Hardware high trigger level available
BYTE	<i>b_SCANHardwareTriggerLowAvailable;</i>	0: Hardware low trigger level not available 1: Hardware low trigger level available
BYTE	<i>b_SCANHardwareTriggerAvailableMode;</i>	0001: External trigger starts each acquisition from a SCAN 0010: Each trigger starts each SCAN 1000: the first Trigger starts the SCAN cycle
BYTE	<i>b_SCANHardwareGateAvailable;</i>	0: Hardware gate not available 1: Hardware gate available
BYTE	<i>b_SCANHardwareGateHighAvailable;</i>	0: Hardware high gate level not available 1: Hardware high gate level available
BYTE	<i>b_SCANHardwareGateLowAvailable;</i>	0: Low level not available for hardware gate 1: Low level available for hardware gate
BYTE	<i>b_SCANClrIndexAvailable;</i>	0: It is not possible to restart the acquisition with the next selected channel of SCAN 1: It is possible to restart the acquisition with the next selected channel of SCAN
WORD	<i>w_SCANAcquisitionMode;</i>	00: The acquisition can operate in single mode 01: The acquisition can operate in continuous mode 11: The acquisition can operate in single and continuous mode

BYTE	<i>b_SCANCounterAvailable</i> ; 0: SCAN conversion counter not available 1: SCAN conversion counter available
BYTE	<i>b_SCANCounterMode</i> ; 01: Counter can count the number of SCANS
BYTE	<i>b_SCANCommonGain</i> ; Defines if the gain can differ for each input 0: Gain is the same for each input 1: Gain can differ for each input.
BYTE	<i>b_SCANCommonPolarity</i> ; Defines if the polarity can differ for each input 0: Polarity is the same for each input 1: Polarity can be differ for each input.
BYTE	<i>b_SCANCommonOffsetRange</i> ; Defines if the offset can differ for each input 0: Offset is the same for each input 1: Offset can differ for each input.
BYTE	<i>b_SCANCommonCoupling</i> ;
WORD	<i>w_SCANCounterResolution</i> ; Counter resolution 8: 8-bit resolution 16: 16-bit resolution, ...
BYTE	<i>b_SCANDelayTimeConfigurable</i> ; 0: Delay after each sequence not available 1: Delay after each sequence available
BYTE	<i>b_SCANDelayMode</i> ;
BYTE	<i>b_SCANDelayCalcType</i> ; 0: Binary type (XX000,XX001,XX010,XX011,XX100) 1: Multiple type (60,120,240,480,960, ...)
BYTE	<i>b_SCANDelayTimeUnitType</i> ; 0: Time unit (ns, μ s, ms, s, ...) 1: Frequency unit (MHz, kHz, Hz, mHz, ...)
BYTE	<i>b_SCANDelayValueType</i> ; 0: Value to write in the register is the value in s or in Hz 1: The step multiplier is to be written in the register
BYTE	<i>b_SCANDelayTimeUnit</i> ; For time unit: D0: 0: ns not available 1: ns available D1: 0: μ s not available 1: μ s available D2: 0: ms not available 1: ms available D3: 0: s not available 1: s available For frequency unit: D0: 0: MHz not available 1: MHz available D1: 0: kHz not available 1: kHz available D2: 0: Hz not available 1: Hz available D3: 0: mHz not available 1: mHz available
WORD	<i>w_SCANDelayTimeResolution</i> ; Delay time resolution 8: 8-bit resolution 16: 16-bit resolution, ...

WORD	<i>w_SCANMinDelayTime</i> ; Minimum delay time. 7000: 7000(ns) 10000: 10000(ns), ...
WORD	<i>w_SCANDelayTimeStep</i> ; Conversion delay time steps 20: 20 steps 50: 50 steps, ...
BYTE	<i>b_SCANUnipolarBipolarConfigurable</i> ; 0: Unipolar/bipolar hardware configurable 1: Unipolar/bipolar configurable
BYTE	<i>b_SCANUnipolarAvailable</i> ; 0: Cannot configure in unipolar 1: Can configure in unipolar
BYTE	<i>b_SCANBipolarAvailable</i> ; 0: Cannot configure in bipolar 1: Can configure in bipolar
BYTE	<i>b_SCANDCCouplingAvailable</i> ; 0: Cannot configure the direct coupling (DC) 1: Can configure direct coupling (DC)
BYTE	<i>b_SCANACCouplingAvailable</i> ; 0: Cannot configure alternative coupling (AC) 1: Can configure alternative coupling (AC)
BYTE	<i>b_SCANBufferAvailable</i> ; 0: No hardware buffer available 1: Hardware buffer available
BYTE	<i>b_SCANNbrOfGain</i> ; Returns the number of gain values available
BYTE	<i>b_Reserved7</i> ;
WORD	<i>w_Reserved8</i> ;
DOUBLE	<i>d_SCANGainAvailable</i> [255]; Defines the available gain values
BYTE	<i>b_SCANOffsetRangeAvailable</i> ; 0: Offset not allowed 1: Offset allowed
BYTE	<i>b_Reserved9</i> ;
WORD	<i>w_SCANOffsetRangeResolution</i> ; Offset resolution 8 : 8-bit resolution 16: 16-bit resolution, ...
BYTE	<i>b_SCANOffsetRangeDenominator</i> ; Offset denominator step value
BYTE	<i>b_SCANOffsetRangeNumerator</i> ; Offset numerator step value
WORD	<i>w_Reserved10</i> ;
}str_GetAnalogMesureInformation,*pstr_GetAnalogMesureInformation;	

Task:

Returns information about the selected analog input channels.

Calling convention:ANSI C :

```

BYTE          b_ReturnValue;
DWORD         dw_DriverHandle;
WORD          w_ChannelNbr;
str_GetAnalogMesureInformation s_ChannelInformation;

b_ReturnValue = b_ADDIDATA_GetAnalogInputInformation
(dw_DriverHandle,
0,
&s_ChannelInformation,
sizeof (str_GetAnalogMesureInformation));

```


Return value:

1: No error

0: Error by calling up the function. Use the function "i_ADDIDATA_GetLastError",
to find the error source.

6) b_ADDIDATA_GetAnalogInputModuleGeneralInformation (..)

Syntax:

```
<Return value> = b_ADDIDATA_GetAnalogInputModuleGeneralInformation
(DWORD dw_DriverHandle,
WORD w_Module,
pstr_AnalogInputModuleInformation
ps_ModuleInformation,
DWORD dw_StructSize)
```

Input:

DWORD	dw_DriverHandle	Driver handle
DWORD	dw_StructSize	Size of the structure entered in the pointer.

Output:

pstr_AnalogInputModuleInformation	ps_ModuleInformation	Module information.
-----------------------------------	----------------------	---------------------

structure returned :

```
typedef struct
```

```
{
    BYTE    b_InputsResolution;           Returns the input resolution
                                             8: 8-bit resolution
                                             16: 16-bit resolution, ...
    BYTE    b_UnipolarBipolarConfigurable; 0: Unipolar/Bipolar hardware
                                             configurable
                                             1: Unipolar/Bipolar software
                                             configurable
    BYTE    b_UnipolarAvailable;           0: Unipolar mode not
                                             available
                                             1: Unipolar mode available
    BYTE    b_BipolarAvailable;            0: Bipolar mode not available
                                             1: Bipolar mode available
    BYTE    b_SingleDifferenceSelected;     0: Single mode selected
                                             1: Difference mode selected
    BYTE    b_ACAvailable;                 0: AC available
                                             1: AC not available
    BYTE    b_DCAvailable;                 0: DC available
                                             1: DC not available
    BYTE    b_AutoCalibration;             0: Auto calibration not
                                             available
                                             1: Auto calibration available
    DOUBLE  d_UMax;                        Return the maximal input
                                             voltage
                                             value in V or A
    BYTE    b_ConversionCalcType;          0: Binary type
```

```
(XX000,XX001,XX010,XX011)
```

```
1: Multiple type
```

```
(60,120,240,480, ...)
```

BYTE	b_ConversionUnitType;	0: Time unit (ns, μ s, ms, s, ...) 1: Frequency unit (MHz, KHz, Hz, etc.)
BYTE	b_AvailableConversionUnit;	For time unit: D0: 0: ns not available 1: ns available D1: 0: μ s not available 1: μ s available D2: 0: ms not available 1: ms available D3: 0: s not available 1: s available For frequency unit: D0: 0: MHz not available 1: MHz available D1: 0: KHz not available 1: KHz available D2: 0: Hz not available 1: Hz available D3: 0: mHz not available 1: mHz available
BYTE	b_Reserved2[5];	
WORD	w_ConversionResolution;	Convert time resolution 8: 8-bit resolution 16: 16-bit resolution, ...
WORD	w_MinConversionTime;	Minimum conversion time. 7000 : 7000(ns) 10000 : 10000(ns), ...
WORD	w_ConversionStep;	Conversion time steps 20: 20 steps 50: 50 steps, ...
BYTE	b_Reserved3[2];	
BYTE	b_SingleAcquisition;	0: Single acquisition not available 1: Single acquisition available
BYTE	b_AutoRefreshAcquisition;	0: Auto refresh acquisition not available 1: Auto refresh acquisition available
BYTE	b_ScanAcquisition;	0: Scan acquisition not available 1: Scan acquisition available
BYTE	b_SequenceAcquisition;	0: Sequence acquisition not available 1: Sequence acquisition available
BYTE	b_Reserved4[4];	

WORD	w_FirstChannelNumber;	Return the number from first channel number
WORD	w_LastChannelNumber;	Return the number from last channel number
BYTE	b_Reserved5[4];	

}str_AnalogInputModuleInformation, *pstr_AnalogInputModuleInformation;

Task:

Returns information about the analog input module (input resolution, convert time, available acquisition modes, ...)

Calling convention:

ANSI C:

```

BYTE                b_ReturnValue;
DWORD               dw_DriverHandle;
str_AnalogInputModuleInformation  s_ModuleInformation;

```

```

b_ReturnValue = b_ADDIDATA_GetAnalogInputModuleGeneralInformation
                (dw_DriverHandle,
                0,

```

```

&s_ModuleInformation,

```

```

                sizeof

```

```

(s_ModuleInformation));

```

Return Value:

1: No error

0: Error by calling up the function. Use the function "i_ADDIDATA_GetLastError", to find the error source.

7) **b_ADDIDATA_GetAnalogInputModuleSingleAcquisitionInformation (..)**

Syntax:

<Return value> =

```
b_ADDIDATA_GetAnalogInputModuleSingleAcquisitionInformation
        (DWORD dw_DriverHandle,
         WORD w_Module,
         pstr_AnalogInputSingleAcquisitionInformation
         ps_ModuleInformation,
         DWORD dw_StructSize)
```

Input:

DWORD	dw_DriverHandle	Driver handle
DWORD	dw_StructSize	Size of the structure entered in the pointer.

Output:

pstr_AnalogInputSingleAcquisitionInformation	ps_ModuleInformation
Module	

information.

structure returned :

```
typedef struct
{
    BYTE    b_Interrupt;           0: No interrupt can by generated
                                   1: Interrupt can by generated
    BYTE    b_SoftwareTrigger;     0: Software trigger not available
                                   1: Software trigger available
    BYTE    b_HardwareTrigger;     0: Hardware trigger not available
                                   1: Hardware trigger available
    BYTE    b_HardwareGate;        0: Hardware gate not available
                                   1: Hardware gate available
    BYTE    b_NbrOfGain;           Returns the number of gain values
                                   available
    BYTE    b_Reserved1[3];
    DOUBLE  d_GainAvailable[255];  Defines the available gain value
}str_AnalogInputSingleAcquisitionInformation,
*psr_AnalogInputSingleAcquisitionInformation;
```

Task:

Returns information about the analog input module for the single acquisition mode. (b_ADDIDATA_Read1AnalogInput and b_ADDIDATA_ReadMoreAnalogInputs)

Calling convention:ANSI C :

```
BYTE                                b_ReturnValue;
DWORD                              dw_DriverHandle;
str_AnalogInputSingleAcquisitionInformation  s_ModuleInformation;

b_ReturnValue =
b_ADDIDATA_GetAnalogInputModuleSingleAcquisitionInformation
(dw_DriverHandle,
0,
&s_ModuleInformation,
sizeof
(s_ModuleInformation));
```

Return Value:

1: No error

0: Error by calling up the function. Use the function "i_ADDIDATA_GetLastError", to find the error source.

8) **b_ADDIDATA_GetAnalogInputModuleAutoRefreshInformation (..)**

Syntax:

```
<Return value> = b_ADDIDATA_GetAnalogInputModuleAutoRefreshInformation
(DWORD dw_DriverHandle,
WORD w_Module,
pstr_AnalogInputModuleAutoRefreshInformation
ps_ModuleInformation,
DWORD dw_StructSize)
```

Input:

DWORD	dw_DriverHandle	Driver handle
DWORD	dw_StructSize	Size of the structure entered in the pointer.

Output:

pstr_AnalogInputModuleAutoRefreshInformation	ps_ModuleInformation	Module information.
--	----------------------	---------------------

structure returned :

```
typedef struct
```

```
{
    BYTE    b_Interrupt;           0: No interrupt can be generated
                                     1: Interrupt can be generated
    BYTE    b_SoftwareTrigger;     0: Software trigger not available
                                     1: Software trigger available
    BYTE    b_HardwareTrigger;     0: Hardware trigger not available
                                     1: Hardware trigger available
    BYTE    b_HardwareGate;        0: Hardware gate not available
                                     1: Hardware gate available
    BYTE    b_AccessMode;          Auto refresh buffer access mode
                                     8: 8-bit access mode
                                     16: 16-bit access mode
                                     32: 32-bit access mode
    BYTE    b_CommonGain;          Define if the gain can be different
                                     for each input
                                     0: Gain is common on each input
                                     1: Gain can be different for each
                                     input.
    BYTE    b_CommonPolarity;      Define if the polarity can be
                                     different for each input
                                     0: Polarity is common on each input
                                     1: Polarity can be different for each
                                     input.
    BYTE    b_Reserved1 [1];
    BYTE    b_NbrOfGain;           Returns the number of gain values
                                     available
    BYTE    b_Reserved2 [7];
    DOUBLE  d_GainAvailable[255];  Defines the available gain value
}
```

```
str_AnalogInputModuleAutoRefreshInformation,*pstr_AnalogInputModuleAutoRefreshInformation
```

Task:

Returns information about the analog input module for the auto refresh acquisition mode

(b_ADDIDATA_GetAnalogInputAutoRefreshChannelPointer and b_ADDIDATA_StartAnalogInputAutoRefresh).

The auto refresh mode acquire continually the analog inputs and give the values via a memory space.

Calling convention:

ANSI C:

```

BYTE                                b_ReturnValue;
DWORD                              dw_DriverHandle;
str_AnalogInputAutoRefreshInformation  s_ModuleInformation;
```

```

b_ReturnValue = b_ADDIDATA_GetAnalogInputModuleAutoRefreshInformation
                                                         (dw_DriverHandle,
                                                         0,
```

```

&s_ModuleInformation,
```

```

                                                         sizeof
```

```

(s_ModuleInformation));
```

Note:

All channels of a module must be initialised to run the auto refresh acquisition.

Return Value:

1: No error

0: Error by calling up the function. Use the function "i_ADDIDATA_GetLastError", to find the error source.

9) **b_ADDIDATA_GetAnalogInputModuleSCANInformation (..)**

Syntax:

```
<Return value> = b_ADDIDATA_GetAnalogInputModuleSCANInformation
(DWORD dw_DriverHandle,
WORD w_Module,
pstr_AnalogInputModuleSCANInformation
ps_ModuleInformation,
DWORD dw_StructSize)
```

Input:

DWORD	dw_DriverHandle	Driver handle
DWORD	dw_StructSize	Size of the structure entered in the pointer.

Output:

pstr_AnalogInputModuleSCANInformation	ps_ModuleInformation	Module information.
---------------------------------------	----------------------	---------------------

structure returned :

```
typedef struct
```

```
{
    BYTE    b_SCANConfigurable;    0: SCAN fixed.
                                         User must pass the first and last
                                         channel from the selected module
                                         1: SCAN configurable
                                         User is free to define the first and
                                         last channel
    BYTE    b_SoftwareTrigger;    0: Software trigger not available
                                         1: Software trigger available
    BYTE    b_HardwareTrigger;    0: Hardware trigger not available
                                         1: Hardware trigger available
    BYTE    b_HardwareGate;    0: Hardware gate not available
                                         1: Hardware gate available
    BYTE    b_CommonGain;    Define if the gain can be different
                                         for each input
                                         0: Gain is common on each input
                                         1: Gain can be different for each
                                         input.
    BYTE    b_CommonPolarity;    Define if the polarity can be
                                         different for each input
                                         0: Polarity is common on each input
                                         1: Polarity can be different for each
                                         input.
    BYTE    b_Reserved1;
    BYTE    b_NbrOfGain;    Returns the number of gain values
                                         available
    DOUBLE  d_GainAvailable[255];    Defines the available gain value
    BYTE    b_DelayTimeConfigurable;    0: Delay after each SCAN not
                                         available
                                         1: Delay after each SCAN available
    BYTE    b_DelayAvailableMode;    D0: 0: Mode 1 not available
```

			1: Mode 1 available D1: 0: Mode 2 not available 1: Mode 2 available
BYTE	b_DelayCalcType;		0: Binary type
(XX000,XX001,XX010)			1: Multiple type
(60,120,240,480,960, ...)			
BYTE	b_DelayTimeUnitType;		0: Time unit (ns,µs,ms,s, ...)
(MHz,kHz,Hz,mHz, ...)			1: Frequency unit
BYTE	b_DelayTimeUnit;		For time unit:
		D0:	0: ns not available 1: ns available
		D1:	0: µs not available 1: µs available
		D2:	0: ms not available 1: ms available
		D3:	0: s not available 1: s available
			For frequency unit:
		D0:	0: MHz not available 1: MHz available
		D1:	0: KHz not available 1: KHz available
		D2:	0: Hz not available 1: Hz available
		D3:	0: mHz not available 1: mHz available
BYTE	b_Reserved2 [3];		
WORD	w_DelayTimeResolution;		Delay time resolution
			8: 8-bit resolution
			16: 16-bit resolution, ...
WORD	w_MinDelayTime;		Minimum delay time.
			7000 : 7000(ns)
			10000: 10000(ns), ...
WORD	w_DelayTimeStep;		Conversion delay time steps
			20: 20 steps
			50: 50 steps, ...
WORD	w_AcquisitionMode;		XX1: The acquisition can work in Single mode
			X11: The acquisition can work in continuous mode
			1XX: The acquisition can work in counting mode
DWORD	dw_MaxNumberOfAcquisition;		Return the max number of acquisition for the counting acquisition SCAN mode
BYTE	b_Reserved4 [4];		
	} str_AnalogInputSCANInformation,*pstr_AnalogInputSCANInformation		

Task:

Returns information about the analog input module for the SCAN acquisition mode (b_ADDIDATA_InitAnalogInputSCANAcquisition and

b_ADDIDATA_StartAnalogInputSCAN).

Into the SCAN mode the user define the first and last channel to acquire. After each SCAN a interrupt is generated and the user receive the analog input values.

Calling convention:

ANSI C:

```
BYTE                b_ReturnValue;
DWORD               dw_DriverHandle;
pstr_AnalogInputSCANInformation  s_ModuleInformation;
```

```
b_ReturnValue = b_ADDIDATA_GetAnalogInputModuleSCANInformation
                                   (dw_DriverHandle,
                                   0,
```

```
&s_ModuleInformation,
```

```
sizeof
```

```
(s_ModuleInformation));
```

Return Value:

1: No error

0: Error by calling up the function. Use the function "i_ADDIDATA_GetLastError", to find the error source.

10) b_ADDIDATA_GetAnalogInputModuleSequenceInformation (..)

Syntax:

```
<Return value> = b_ADDIDATA_GetAnalogInputModuleSequenceInformation
(DWORD dw_DriverHandle,
WORD w_Module,
pstr_AnalogInputModuleSequenceInformation
ps_ModuleInformation,
DWORD dw_StructSize)
```

Input:

DWORD	dw_DriverHandle	Driver handle
DWORD	dw_StructSize	Size of the structure entered in the pointer.

Output:

pstr_AnalogInputModuleSequenceInformation	ps_ModuleInformation
Module	
information.	

structure returned :

typedef struct

{		
BYTE	b_SequenceConfigurable;	0: Sequence fixed. User muss pass the first and last channel from the selected module 1: Sequence configurable User is free to define the first and last channel
BYTE	b_SoftwareTrigger;	0: Software trigger not available 1: Software trigger available
BYTE	b_HardwareTrigger;	0: Hardware trigger not available 1: Hardware trigger available
BYTE	b_HardwareGate;	0: Hardware gate not available 1: Hardware gate available
BYTE	b_CommonGain;	Define if the gain can be different for each input 0: Gain is common on each input 1: Gain can be different for each input.
BYTE	b_CommonPolarity;	Define if the polarity can be different for each input 0: Polarity is common on each input 1: Polarity can be different for each input.
BYTE	b_Reserved1;	

BYTE	b_NbrOfGain;	Returns the number of gain values available
DOUBLE	d_GainAvailable[255];	Defines the available gain value
BYTE	b_DelayTimeConfigurable;	0: Delay after each sequence not available 1: Delay after each sequence available
BYTE	b_DelayAvailableMode;	D0: 0: Mode 1 not available 1: Mode 1 available D1: 0: Mode 2 not available 1: Mode 2 available
BYTE	b_DelayCalcType;	0: Binary type 1: Multiple type
(XX000,XX001,XX010)		
(60,120,240,480,960, ...)		
BYTE	b_DelayTimeUnitType;	0: Time unit (ns,µs,ms,s, ...) 1: Frequency unit
(MHz,kHz,Hz,mHz, ...)		
BYTE	b_DelayTimeUnit;	For time unit: D0: 0: ns not available 1: ns available D1: 0: µs not available 1: µs available D2: 0: ms not available 1: ms available D3: 0: s not available 1: s available For frequency unit: D0: 0: MHz not available 1: MHz available D1: 0: KHz not available 1: KHz available D2: 0: Hz not available 1: Hz available D3: 0: mHz not available 1: mHz available
BYTE	b_Reserved2 [3];	
WORD	w_DelayTimeResolution;	Delay time resolution 8: 8-bit resolution 16: 16-bit resolution, ...
WORD	w_MinDelayTime;	Minimum delay time. 7000: 7000 (ns) 10000: 10000 (ns), ...
WORD	w_DelayTimeStep;	Conversion delay time steps 20: 20 steps 50: 50 steps, ...
WORD	w_AcquisitionMode;	XX1: The acquisition can work in Single mode X11: The acquisition can work in continuous mode 1XX: The acquisition can work in counting mode

```
        DWORD    dw_MaxNumberOfAcquisition; Return the max number of
                                                acquisition for the counting
                                                acquisition SCAN mode
    BYTE         b_Reserved4 [4];
    }
    str_AnalogInputSequenceInformation,*pstr_AnalogInputSequenceInformation
```

Task:

Returns information about the analog input module for the SCAN acquisition mode (b_ADDIDATA_InitAnalogInputSequenceAcquisition and b_ADDIDATA_StartAnalogInputSequenceAcquisition).

The sequence mode use the DMA. This give the possibility to acquire in background. Into the sequence mode the user define a channel sequence to acquire. After each X sequence a interrupt is generated and the user receive the analog input values via a buffer.

Calling convention:

ANSI C :

```

BYTE                b_ReturnValue;
DWORD               dw_DriverHandle;
str_AnalogInputSequenceInformation s_ModuleInformation;
```

```

b_ReturnValue = b_ADDIDATA_GetAnalogInputModuleSequenceInformation
                (dw_DriverHandle,
                0,
```

```

&s_ModuleInformation,
```

```
                sizeof
```

```

(s_ModuleInformation));
```

Return Value:

1: No error

0: Error by calling up the function. Use the function "i_ADDIDATA_GetLastError", to find the error source.

2.2 Initialisation

1) **b_ADDIDATA_InitAnalogInput (..)**

Syntax:

```
<Return value> = b_ADDIDATA_InitAnalogInput (DWORD dw_DriverHandle,
                                              WORD   w_Channel,
                                              pstr_InitAnalogInput ps_InitParameters,
                                              DWORD
dw_StructSize)
```

Input:

DWORD	dw_DriverHandle	Driver handle
WORD	w_Channel	Number of the analog input channel to be initialised
	pstr_InitAnalogInputChannel ps_InitParameters	
DWORD	dw_StructSize	Size of the structure entered in the pointer.

The structure must contain the following information:

typedef struct

```
{
DOUBLE   d_Gain           Gain of the channel (Refer to table 2-1)
BYTE     b_Polarity       Polarity of the selected channel
                                ADDIDATA_UNIPOLAR for unipolar and
                                ADDIDATA_BIPOLAR for bipolar
                                (see table 2-1)

BYTE     b_Reserved1
WORD     w_OffsetRange    Offset range of the channel.
                                Not used for the moment. Set it to 0.

BYTE     b_Coupling        Select the coupling:
                                ADDIDATA_AC_COUPLING :
                                Alternative coupling
                                ADDIDATA_DC_COUPLING :
                                Direct coupling

BYTE     b_Reserved2
WORD     w_Reserved3
}str_InitAnalogInput,*pstr_InitAnalogInput
```

Output:

No output signal has occurred.

Task:

Initialises the selected analog input channel.

Table 2-1: Gain/Polarity for a max. voltage of 5 V

Gain selection	Polarity selection	Min. voltage	Max. voltage
1	ADDIDATA_UNIPOLAR	0 V	5V
	ADDIDATA_BIPOLAR	-5V	5V
2	ADDIDATA_UNIPOLAR	0 V	2.5V
	ADDIDATA_BIPOLAR	- 2.5V	2.5V
4	ADDIDATA_UNIPOLAR	0 V	1.25 V
	ADDIDATA_BIPOLAR	- 1.25 V	1.25 V
5	ADDIDATA_UNIPOLAR	0 V	1 V
	ADDIDATA_BIPOLAR	- 1 V	1 V
8	ADDIDATA_UNIPOLAR	0 V	0.625 V
	ADDIDATA_BIPOLAR	- 0.625 V	0.625 V
10	ADDIDATA_UNIPOLAR	0 V	0.5 V
	ADDIDATA_BIPOLAR	- 0.5 V	0.5 V
16	ADDIDATA_UNIPOLAR	0 V	0.3125 V
	ADDIDATA_BIPOLAR	- 0.3125 V	0.3125 V
20	ADDIDATA_UNIPOLAR	0 V	0.250 V
	ADDIDATA_BIPOLAR	- 0.250 V	0.250 V
32	ADDIDATA_UNIPOLAR	0 V	156.25 mV
	ADDIDATA_BIPOLAR	- 156.25 mV	156.25 mV
50	ADDIDATA_UNIPOLAR	0 V	100 mV
	ADDIDATA_BIPOLAR	- 100 mV	100 mV
64	ADDIDATA_UNIPOLAR	0 V	78.125 mV
	ADDIDATA_BIPOLAR	- 78.125 mV	78.125 mV
100	ADDIDATA_UNIPOLAR	0 V	50 mV
	ADDIDATA_BIPOLAR	- 50 mV	50 mV
128	ADDIDATA_UNIPOLAR	0 V	39,0625 mV
	ADDIDATA_BIPOLAR	- 39,0625 mV	39,0625 mV
...			

Calling convention:ANSI C :

```

BYTE      b_ReturnValue;
DWORD     dw_DriverHandle;
str_InitAnalogInputChannel s_InitParameters;

b_ReturnValue = b_ADDIDATA_InitAnalogInput
(dw_DriverHandle,
0,
&s_InitParameters,
sizeof (str_InitAnalogInputChannel));

```

Return Value:

1: No error

0: Error by calling up the function. Use the function "i_ADDIDATA_GetLastError", to find the error source.

2) **b_ADDIDATA_ReleaseAnalogInput (..)**

Syntax:

<Return value> = b_ADDIDATA_ReleaseAnalogInput
(DWORD dw_DriverHandle,
WORD w_Channel)

Parameters:

- Input:

DWORD	<i>dw_DriverHandle</i>	Driver handle
WORD	<i>w_Channel</i>	Number of the analog input channel to be released before a new initialisation.

- Output:

No output signal has occurred.

Task:

Releases the analog input logic for the selected channel (*w_Channel*) to allow a new initialisation.

Calling convention:

ANSI C :

BYTE	b_ReturnValue;
DWORD	dw_DriverHandle;

```
b_ReturnValue = b_ADDIDATA_ReleaseAnalogInput
                                   (dw_DriverHandle,
                                   0);
```

Return value:

1: No error
0: Error by calling up the function. Use the function "i_ADDIDATA_GetLastError",
to find the error source.

2.3 Single acquisition

1) **b_ADDIDATA_Read1AnalogInput (..)**

Syntax:

```
<Return value> = b_ADDIDATA_Read1AnalogInput
                                (DWORD    dw_DriverHandle,
                                 WORD      w_Channel,
                                 DWORD     dw_ConvertingTime,
                                 BYTE      b_ConvertingTimeUnit,
                                 BYTE      b_InterruptFlag,
                                 PDWORD    pdw_ChannelValue)
```

Input:

DWORD	dw_DriverHandle	Driver handle
DWORD	w_Channel	Number of the analog input channel to be read
DWORD	dw_ConvertingTime	Conversion time. Refer to table 2-2
BYTE	b_ConvertingTimeUnit	Determines the conversion time unit. Refer to table 2-2
BYTE	b_InterruptFlag	ADDIDATA_DISABLE: No interrupt is generated after the conversion and the variable <i>pdw_ChannelValue</i> returns the digital value. ADDIDATA_ENABLE: An interrupt is generated after the conversion. The digital value of the selected channel is returned through the interrupt function. Refer to the chapter "Interrupt"

Output:

PDWORD	pdw_ChannelValue	Returns the digital value of the selected channel *pdw_ChannelValue[0] = digital value of the channel *pdw_ChannelValue[1] = digital value of the calibration offset (if available) *pdw_ChannelValue[2] = digital value of the calibration gain (if available)
--------	------------------	--

Task:

Returns the digital value (*pdw_ChannelValue*) of the selected channel (*w_Channel*). To obtain the real analog input value, you must call up the "**b_ADDIDATA_ConvertDigitalToRealAnalogValue (...)**" function. *dw_ConvertingTime* and *b_ConvertingTimeUnit* determine the conversion time.

i

IMPORTANT!

The **interrupt mask** for the function is detailed in the "**Interrupt**" function description. (Tables 2-1).

Table 2-2: Selection of the conversion time

<i>b_ConvertingTimeUnit</i>	Unit selection	<i>dw_ConvertingTime</i>	Conversion time
0	ns / MHz	10	10 ns / MHz
		20	20 ns / MHz
		300	300 ns / MHz
		1000	1000 ns / MHz
		22222	22222 ns / MHz
1	μ s / KHz	10	10 μ s / KHz
		20	20 μ s / KHz
		300	300 μ s / KHz
		1000	1000 μ s / KHz
		22222	22222 μ s / KHz
2	ms / Hz	10	10 ms / Hz
		20	20 ms / Hz
		300	300 ms / Hz
		1000	1000 ms / Hz
		22222	22222 ms / Hz
3	s / mHz	10	10 s / mHz
		20	20 s / mHz
		300	300 s / mHz
		1000	1000 s / mHz
		22222	22222 s / mHz

Calling convention:ANSI C:

```

BYTE      b_ReturnValue;
DWORD     dw_DriverHandle;
DWORD     dw_ChannelValue[4];

```

```

b_ReturnValue = b_ADDIDATA_Read1AnalogInput (dw_DriverHandle,
                                              0,
                                              10,
                                              1,
                                              ADDIDATA_DISABLE,
                                              dw_ChannelValue);

```

Return Value:

1: No error

0: Error by calling up the function. Use the function "i_ADDIDATA_GetLastError", to find the error source.

2) **b_ADDIDATA_ReadMoreAnalogInputs (..)**

Syntax:

```
<Return value> = b_ADDIDATA_ReadMoreAnalogInputs
                                (DWORD    dw_DriverHandle,
                                 WORD      w_FirstChannel,
                                 WORD      w_LastChannel,
                                 DWORD     dw_ConvertingTime,
                                 BYTE      b_ConvertingTimeUnit,
                                 BYTE      b_InterruptFlag,
                                 PDWORD    pdw_ChannelArrayValue)
```

Input:

DWORD	dw_DriverHandle	Driver handle
WORD	w_FirstChannel	Selection of the first channel to be read
WORD	w_LastChannel	Selection of the last channel to be read
DWORD	dw_ConvertingTime	Conversion time. Refer to table 2-2
BYTE	b_ConvertingTimeUnit	Determines the unit of the conversion time. Refer to table 2-2
BYTE	b_InterruptFlag	ADDIDATA_DISABLE: No interrupt is generated after the last conversion and the variable <i>pdw_ChannelArrayValue</i> returns the digital value. ADDIDATA_ENABLE: An interrupt is generated after the last conversion. The digital value of all selected channels is returned through the interrupt function. Refer to the chapter "Interrupt"

Output:

PDWORD	pdw_ChannelArrayValue	Returns the digital value of all selected channels
--------	-----------------------	--

*pdw_ChannelValue[0] = digital value of the first channel
 *pdw_ChannelValue[1] = digital value of the calibration offset (if available)
 *pdw_ChannelValue[2] = digital value of the calibration gain (if available)
 *pdw_ChannelValue[3] = digital value of the next channel
 *pdw_ChannelValue[4] = digital value of the calibration offset (if available)
 *pdw_ChannelValue[5] = digital value of the calibration gain (if available)
 ...

Task:

Returns the digital value (*pdw_ChannelValue*) of all selected channels (*w_FirstChannel*, *w_LastChannel*). To obtain the real analog input value, you must call up the "**b_ADDIDATA_ConvertDigitalToRealAnalogValue (...)**" function.

dw_ConvertingTime and *b_ConvertingTimeUnit* determine the conversion time.

Calling convention:ANSI C:

```
BYTE      b_ReturnValue;  
DWORD     dw_DriverHandle;  
DWORD     dw_ChannelArrayValue [48];
```

```
b_ReturnValue = b_ADDIDATA_ReadMoreAnalogInputs (dw_DriverHandle,  
                                                  0,  
                                                  11,  
                                                  10,  
                                                  1,
```

```
ADDIDATA_DISABLE,
```

```
dw_ChannelArrayValue);
```

Return Value:

1: No error

0: Error by calling up the function. Use the function "i_ADDIDATA_GetLastError", to find the error source.

3) **b_ADDIDATA_ConvertDigitalToRealAnalogValue (..)**

Syntax:

```
<Return value> = b_ADDIDATA_ConvertDigitalToRealAnalogValue
                                (DWORD      dw_DriverHandle,
                                 WORD        w_Channel,
                                 PDWORD     pdw_DigitalValue,
                                 PDOUBLE    pd_RealValue)
```

Input:

DWORD	dw_DriverHandle	Driver handle
WORD	w_Channel	Number of the channel to be converted
PDWORD	pdw_DigitalValue	Digital analog input value (composed of the digital value of the channel, the calibration offset and of the calibration gain if available)

Output:

PDOUBLE pd_RealValueArray Returns the real analog input value

Task:

Converts the digital analog input value (*pdw_DigitalValue*) into a real analog input value (*pd_RealValue*) for the selected channel (*w_Channel*).

Calling convention:

ANSI C:

```
BYTE      b_ReturnValue;
DWORD     dw_DriverHandle;
DWORD     dw_DigitalValue[4];
DOUBLE    d_RealValue;
```

```
b_ReturnValue = b_ADDIDATA_ConvertDigitalToRealAnalogValue
                                (dw_DriverHandle,
                                 0,
                                 dw_DigitalValue,
                                 &d_RealValue);
```

Return Value:

1: No error

0: Error by calling up the function. Use the function "i_ADDIDATA_GetLastError", to find the error source.

4) **b_ADDIDATA_ConvertMoreDigitalToRealAnalogValues (..)**

Syntax:

```
<Return value> = b_ADDIDATA_ConvertMoreDigitalToRealAnalogValues
                                (DWORD      dw_DriverHandle,
                                WORD        w_FirstChannel,
                                WORD        w_LastChannel,
                                PDWORD      pdw_DigitalValue,
                                PDOUBLE     pd_RealValue)
```

Input:

DWORD	dw_DriverHandle	Driver handle
WORD	w_FirstChannel	Selection of the first channel
WORD	w_LastChannel	Selection of the last channel
PDWORD	pdw_DigitalValue	Digital value of the analog input (composed of the digital value of the channel, of the calibration offset and of the calibration gain if available)

Output:

PDOUBLE pd_RealValueArray Returns the real analog value of the input channel

Task:

Converts the digital value (*pdw_DigitalValue*) into a real analog value (*pd_RealValue*) for the selected channels.

Calling convention:

ANSI C:

```
BYTE      b_ReturnValue;
DWORD     dw_DriverHandle;
DWORD     dw_DigitalValue[4];
DOUBLE    d_RealValue;
```

```
b_ReturnValue = b_ADDIDATA_ConvertMoreDigitalToRealAnalogValues
                                (dw_DriverHandle,
                                0,
                                3,
                                dw_DigitalValue,
                                &d_RealValue);
```

Return Value:

1: No error

0: Error by calling up the function. Use the function "i_ADDIDATA_GetLastError", to find the error source.

2.4 Autorefresh acquisition

1) b_ADDIDATA_GetAnalogInputAutoRefreshChannelPointer (..)

Syntax:

```
<Return value> = b_ADDIDATA_GetAnalogInputAutoRefreshChannelPointer
                    (DWORD    dw_DriverHandle,
                     DWORD    dw_Channel,
                     VOID
                     **ppv_ChannelValueApplicationLevelPointer,
                     VOID      **ppv_ChannelValueKernelLevelPointer)
```

Input:

```
DWORD    dw_DriverHandle    Driver handle
DWORD    dw_Channel         Channel Index
```

Output:

```
VOID **ppv_ChannelValueApplicationLevelPointer
                                     Return the pointer of the value
                                     from the selected channel in the
                                     application level.

VOID **ppv_ChannelValueKernelLevelPointer
                                     Return the pointer of the value
                                     from the selected channel in the kernel
                                     level.
```

Task:

Return the pointer of the value from the selected channel.

Calling convention:

ANSI C:

```
BYTE      b_ReturnValue;
DWORD     dw_DriverHandle;
PDWORD    pdw_ChannelValueRing0;
PDWORD    pdw_ChannelValueRing3;

b_ReturnValue =
b_ADDIDATA_GetAnalogInputAutoRefreshChannelPointer(dw_DriverHandle,0,
                                                    (VOID **)&pdw_ChannelValueRing3,
                                                    (VOID **)&pdw_ChannelValueRing0);
```

Return Value:

1: No error
0: Error by calling up the function. Use the function "i_ADDIDATA_GetLastError",
to find the error source.

2) b_ADDIDATA_GetAnalogInputAutoRefreshModulePointer (..)

Syntax:

```

<Return value> = b_ADDIDATA_GetAnalogInputAutoRefreshModulePointer
                    (DWORD    dw_DriverHandle,
                     WORD     w_Module,
                     VOID
                     **ppv_ModuleValueApplicationLevelPointer,
                     VOID     **ppv_ModuleValueKernelLevelPointer)

```

Input:

DWORD	dw_DriverHandle	Driver handle
WORD	w_Module	Module index

Output:

VOID **ppv_ModuleValueApplicationLevelPointer
Returns the pointer of the values from the selected module in the application level.

VOID **ppv_ModuleValueKernelLevelPointer
Returns the pointer of the values from the selected module in the kernel level.

Task:

Return the pointer of the value from the selected module.

Calling convention:ANSI C:

```

BYTE    b_ReturnValue;
DWORD   dw_DriverHandle;
PDWORD  pdw_ModuleValueRing0;
PDWORD  pdw_ModuleValueRing3;

```

```

b_ReturnValue =
b_ADDIDATA_GetAnalogInputAutoRefreshModulePointer(dw_DriverHandle,0,
                                                    (VOID **)&pdw_ChannelValueRing3,
                                                    (VOID **)&pdw_ChannelValueRing0);

```

Return Value:

1: No error

0: Error by calling up the function. Use the function "i_ADDIDATA_GetLastError", to find the error source.

3) **b_ADDIDATA_GetAnalogInputAutoRefreshModuleCounterPointer (..)**

Syntax:

<Return value> =

```
b_ADDIDATA_GetAnalogInputAutoRefreshModuleCounterPointer
(DWORD    dw_DriverHandle,
DWORD    dw_Channel,
VOID     **ppv_ApplicationLevelPointer,
VOID     **ppv_KernelLevelPointer)
```

Input:

DWORD	dw_DriverHandle	Driver handle
DWORD	dw_Channel	Channel Index

Output:

VOID	**ppv_ApplicationLevelPointer	Return the pointer of the counter from the selected module in the application level.
VOID	**ppv_KernelLevelPointer	Return the pointer of the counter from the selected module in the kernel level.

Task:

Return the pointer of the auto refresh counter from the selected module.
After each auto refresh cycle, the counter is incremented from 1.

Calling convention:

ANSI C :

```
BYTE      b_ReturnValue;
DWORD    dw_DriverHandle;
PDWORD   pdw_CounterValueRing0;
PDWORD   pdw_CounterValueRing3;
```

```
b_ReturnValue =
b_ADDIDATA_GetAnalogInputAutoRefreshModuleCounterPointer (dw_DriverHandle,0,
                                                             (VOID **)&pdw_CounterValueRing3,
                                                             (VOID **)&pdw_CounterValueRing0);
```

Return Value:

1: No error
0: Error by calling up the function. Use the function "i_ADDIDATA_GetLastError",
to find the error source.

4) b_ADDIDATA_StartAnalogInputAutoRefresh (..)

Syntax:

<Return value> = b_ADDIDATA_StartAnalogInputAutoRefresh
(DWORD dw_DriverHandle,
WORD w_Module,
DWORD dw_ConvertingTime,
BYTE b_ConvertingTimeUnit)

Parameters:

- Input:

DWORD	dw_DriverHandle	Driver handle
WORD	w_Module	Index of the module for which the auto refresh acquisition must be started
DWORD	dw_ConvertingTime	Conversion time. See table 2-2
BYTE	b_ConvertingTimeUnit	Determines the conversion time unit. See table 2-2

- Output:

No output signal has occurred.

Task:

Starts the auto refresh acquisition on the selected module.

dw_ConvertingTime and *b_ConvertingTimeUnit* determines the conversion time.

i

IMPORTANT!

Do initialise **all** analog input channels of the module to run an acquisition in the auto refresh mode.

Calling convention:

ANSI C :

BYTE	b_ReturnValue;
DWORD	dw_DriverHandle;

```
b_ReturnValue = b_ADDIDATA_StartAnalogInputAutoRefresh
(dw_DriverHandle,
0,
10,
1);
```

Return Value:

1: No error

0: Error by calling up the function. Use the function "i_ADDIDATA_GetLastError", to find the error source.

5) b_ADDIDATA_StopAnalogInputAutoRefresh(..)**Syntax:**

<Return value> = b_ADDIDATA_StopAnalogInputAutoRefresh
(DWORD dw_DriverHandle,
WORD w_Module)

Parameters:**- Input:**

DWORD	dw_DriverHandle	Driver handle
WORD	w_Module	Index of the module for which the auto refresh acquisition must be stopped

- Output:

No output signal has occurred.

Task:

Stops an auto refresh acquisition on the selected module.

Calling convention:ANSI C :

BYTE	b_ReturnValue;
DWORD	dw_DriverHandle;

b_ReturnValue = b_ADDIDATA_StopAnalogInputAutoRefresh(dw_DriverHandle,0);

Return Value:

1: No error

0: Error by calling up the function. Use the function "i_ADDIDATA_GetLastError", to find the error source.

2.5 SCAN acquisition

i

IMPORTANT!

The SCAN acquisition functions can only be operated on one module (group of channels). Please check that all selected channels are on the same module.

1) b_ADDIDATA_InitAnalogInputSCAN (...)

i

IMPORTANT!

This function is only available for old user applications (up to version 2132-0304 of ADDIPACK).

If you develop new applications with ADDIDRIVER, please use the following functions: b_ADDIDATA_InitAnalogInputSCANAcquisition

Syntax:

```
<Return value> = b_ADDIDATA_InitAnalogInputSCAN
                    (DWORD   dw_DriverHandle,
                     pstr_InitAnalogInputSCAN ps_InitParameters,
                     DWORD   dw_StructSize,
                     PDWORD  pdw_SCANHandle)
```

Parameters:

- Input:

DWORD	<i>dw_DriverHandle</i>	Driver handle
	<i>pstr_InitAnalogInput s_InitParameters</i>	pointer on a structure which contains the initialisation parameters
DWORD	<i>dw_StructSize</i>	Size of the structure

The structure must contain the following information:

WORD	<i>w_FirstChannel</i>	First channel to be converted.
WORD	<i>w_LastChannel</i>	Last channel to be converted
DWORD	<i>dw_ConvertingTime</i>	Conversion time. Refer to Table 2-2
BYTE	<i>b_ConvertingTimeUnit</i>	Conversion time unit. Refer to Table 2-2
BYTE	<i>b_SCANTimeMode</i>	Determines the mode of the delay. Refer to Table 2-4
DWORD	<i>dw_SCANTime</i>	Total time of the SCAN. Refer to Table 2-3 and Table 2-4
BYTE	<i>b_SCANTimeUnit</i>	Unit used for the total time. Refer to Table 2-3 and Table 2-4
BYTE	<i>b_SCANMode</i>	SCAN mode. Refer to Table 2-5
BYTE	<i>b_ExternTriggerMode</i>	External trigger action. Refer to Table 2-6

BYTE *b_ExternGateMode* External gate action.
 ADDIDATA_DISABLE: External gate not used
 ADDIDATA_LOW_LEVEL: The SCAN is activated when the external gate is low
 ADDIDATA_HIGH_LEVEL: The SCAN is activated when the external gate is high
 DWORD *dw_SCANCounter* Number of SCAN. Refer to Table 2-5
- Output:
 PDWORD *pdw_SCANHandle* Handle of the initialised SCAN.
 This handle is used for all SCAN functions.

Task:

Initialises the SCAN function for the acquisition of analogue inputs.

Table 2-3: Time selection of a SCAN

<i>b_SCANTimeUnit</i>	Unit selection	<i>dw_SCANTime</i>	Total time of the SCAN
0	ns or MHz	10	10 ns / MHz
		20	20 ns / MHz
		300	300 ns / MHz
		1000	1000 ns / MHz
		22222	22222 ns / MHz
1	μ s or KHz	10	10 μ s / KHz
		20	20 μ s / KHz
		300	300 μ s / KHz
		1000	1000 μ s / KHz
		22222	22222 μ s / KHz
2	ms or Hz	10	10 ms / Hz
		20	20 ms / Hz
		300	300 ms / Hz
		1000	1000 ms / Hz
		22222	22222 ms / Hz
3	s or mHz	10	10 s / mHz
		20	20 s / mHz
		300	300 s / mHz
		1000	1000 s / mHz
		22222	22222 s / mHz

Table 2-4: SCAN time mode

<i>b_SequenceTimeMode</i>	Mode description
ADDIDATA_DELAY_NOT_USED	The delay between two SCAN is not used
ADDIDATA_DELAY_MODE1_USED	The delay between two SCAN is used. The conversion is started at once, and the delay is used after the first SCAN.
ADDIDATA_DELAY_MODE2_USED	The delay between two SCANS is used. The conversion is started after a delay.

Table 2-5: SCAN mode

<i>b_SCANMode</i>	<i>dw_SCANCounter</i>	SCAN description
ADDIDATA_SINGLE_SCAN	Not used	Only one SCAN is started after calling up the function "b_ADDIDATA_StartAnalogInputSCAN"
ADDIDATA_DEFINED_SCAN_NUMBER	Determines the number of SCAN	A predefined number of SCANS is started after calling up the function "b_ADDIDATA_StartAnalogInputSCAN"
ADDIDATA_CONTINUOUS_SCAN	Not used	An undefined number of SCANS is started after calling up the function "b_ADDIDATA_StartAnalogInputSCAN"

Table 2-6: Trigger mode

ADDIDATA_DISABLE	External trigger not used
ADDIDATA_FIRST_LOW_EDGE_START_ALL_SCAN	The first low edge starts all SCANS
ADDIDATA_FIRST_HIGH_EDGE_START_ALL_SCAN	The first high edge starts all SCANS
ADDIDATA_FIRST_EDGE_START_ALL_SCAN	The first low/high edge starts all SCANS
ADDIDATA_EACH_LOW_EDGE_START_A_SCAN	The first low edge starts a single SCAN
ADDIDATA_EACH_HIGH_EDGE_START_A_SCAN	The first high edge starts a single SCAN
ADDIDATA_EACH_EDGE_START_A_SCAN	Each edge starts a single SCAN
ADDIDATA_EACH_LOW_EDGE_START_A_SINGLE_ACQUISITION	Each low edge starts a single acquisition of the SCAN
ADDIDATA_EACH_HIGH_EDGE_START_A_SINGLE_ACQUISITION	Each high edge starts a single acquisition of the SCAN
ADDIDATA_EACH_EDGE_START_A_SINGLE_ACQUISITION	Each edge starts a single acquisition of the SCAN

Calling convention:ANSI C :

```

BYTE          b_ReturnValue;
DWORD         dw_DriverHandle;
DWORD         dw_SCANHandle;
str_InitAnalogInputSCAN s_InitParameters;
.
.
.
b_ReturnValue = b_ADDIDATA_InitAnalogInputSCAN
                (dw_DriverHandle,
                 &s_InitParameters,
                 sizeof (str_InitAnalogInputSCAN),
                 &dw_SCANHandle);

```

Return value:

1: No error

0: Error by calling up the function. Use the function "i_ADDIDATA_GetLastError", to find the error source.

2) b_ADDIDATA_InitAnalogInputSCANAcquisition (..)

Syntax:

```
<Return value> = b_ADDIDATA_InitAnalogInputSCANAcquisition
(DWORD dw_DriverHandle,
pstr_InitAnalogInputSCANAcquisition ps_InitParameters,
DWORD dw_StructSize,
PDWORD pdw_SCANHandle)
```

Input:

DWORD	dw_DriverHandle	Driver handle
pstr_InitAnalogInputSCANAcquisition	ps_InitParameters	
DWORD	dw_StructSize	Size of the structure

The structure must contain the following information:

```
typedef struct
```

```
{
    WORD w_FirstChannel;           First SCAN channel selection
    WORD w_LastChannel;           Last SCAN channel selection
    DWORD dw_ConversionTime;       Selected conversion time
    BYTE b_ConversionTimeUnit;     Selected conversion time unit
                                   For time unit:
                                   0: ns selected
                                   1: µs selected
                                   2: ms selected
                                   3: s selected
                                   For frequency unit:
                                   0: MHz selected
                                   1: KHz selected
                                   2: Hz selected
                                   3: mHz selected
    BYTE b_DelayTimeMode;         Delay mode selection
                                   Refer to table 2-1 and sample1 and
                                   sample 2
    BYTE b_DelayTimeUnit;         Delay time unit selection
                                   Refer to sample 1 and sample 2
                                   For time unit:
                                   0: ns selected
                                   1: µs selected
                                   2: ms selected
                                   3: s selected
                                   For frequency unit:
                                   0: MHz selected
                                   1: KHz selected
                                   2: Hz selected
                                   3: mHz selected
    BYTE b_Reserved1[5];
    DWORD dw_DelayTime;           Delay time
    DWORD dw_SCANCounter;         Define the number of SCAN cycles
                                   if the
                                   SCAN mode (b_SCANMode) is
                                   setting to
```

ADDIDATA_DEFINED_SCAN_NUMBER

BYTE b_SCANMode; Refer to table 5, sample1 and sample2

BYTE b_Reserved2[7];

}

str_InitAnalogInputSCANAcquisition,*pstr_InitAnalogInputSCANAcquisition

Output:

PDWORD pdw_SCANHandle Handle of the initialised SCAN.
This handle is used for all SCAN functions.

Task:

Initialises the analogue input acquisition SCAN. This can be combined with the hardware or software trigger. See the functions

“b_ADDIDATA_EnableDisableAnalogInputHardwareTrigger“ and

“b_ADDIDATA_EnableDisableAnalogInputSoftwareTrigger“

Table 2-1: SCAN delay time mode

<i>b_DelayTimeMode</i>	Mode description
ADDIDATA_DELAY_NOT_USED	The delay between two SCANS is not used
ADDIDATA_DELAY_MODE1_USED	The delay between two SCAN cycles is used. The delay is started after the first acquisition. After this delay a new SCAN cycle is started
ADDIDATA_DELAY_MODE2_USED	The delay between two SCAN cycles is used. The delay is started after the last SCAN channel acquisition. After this delay a new SCAN cycle is started

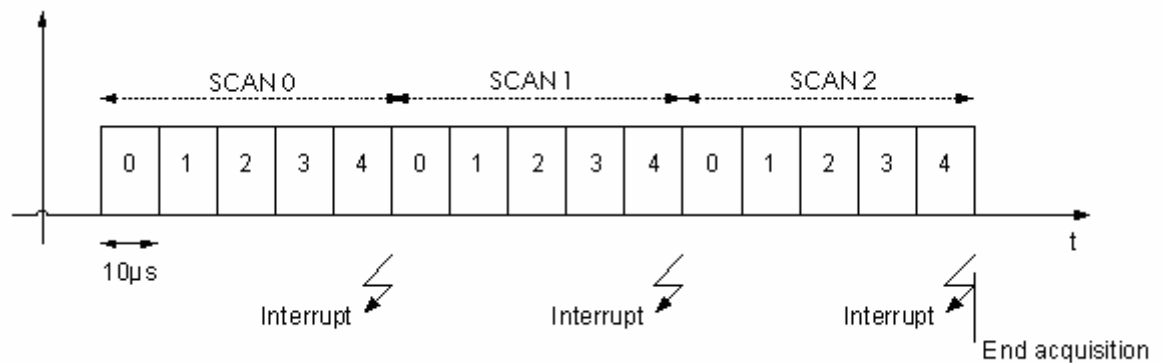
Table 2-2: SCAN mode

<i>b_SCANMode</i>	<i>dw_SCANCounter</i>	SCAN description
ADDIDATA_SINGLE_SCAN	Not used	Only one SCAN is started after calling up the function "b_ADDIDATA_StartAnalogInputSCAN"
ADDIDATA_DEFINED_SCAN_NUMBER	Determines the number of SCAN	A predefined number of SCANS is started after calling up the function "b_ADDIDATA_StartAnalogInputSCAN"
ADDIDATA_CONTINUOUS_SCAN	Not used	An undefined number of SCANS is started after calling up the function "b_ADDIDATA_StartAnalogInputSCAN"

Samples

Sample 1 :

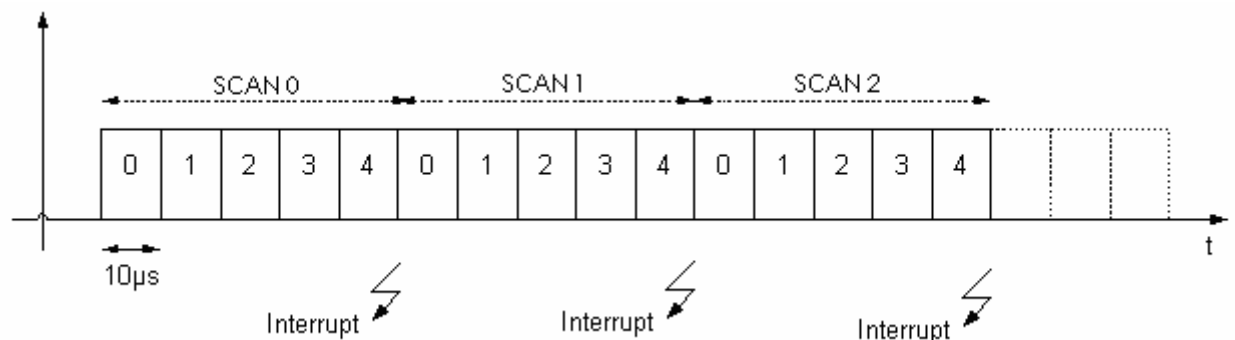
w_FirstChannel = 0
 w_LastChannel = 5
 b_ConvertingTimeUnit = 1 (μ s)
 dw_ConvertingTime = 10
 b_DelayTimeMode = ADDIDATA_DELAY_NOT_USED
 b_SCANMode = ADDIDATA_DEFINED_SCAN_NUMBER
 dw_SCANCounter = 3



An interrupt occur after each end of SCAN (5 acquisitions) and the acquisition is stopped after 3 SCAN cycles

Sample 2 :

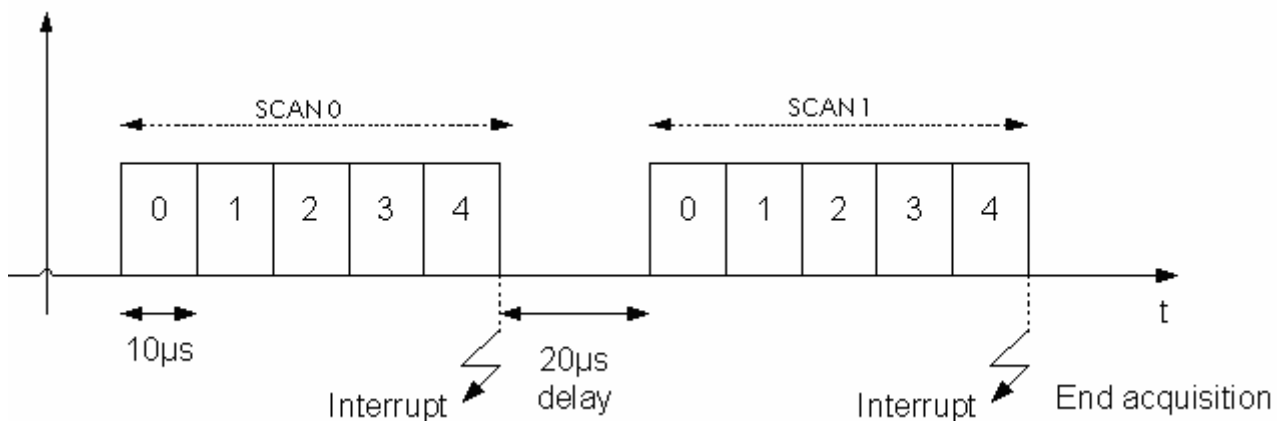
w_FirstChannel = 0
 w_LastChannel = 4
 b_ConvertingTimeUnit = 1 (μ s)
 dw_ConvertingTime = 10
 b_DelayTimeMode = ADDIDATA_DELAY_NOT_USED
 b_SCANMode = ADDIDATA_CONTINUOUS_SCAN



An interrupt occurs after each SCAN (5 acquisitions) and the acquisition is stopped via the function b_ADDIDATA_StopAnalogInputSCAN

Sample 3 :

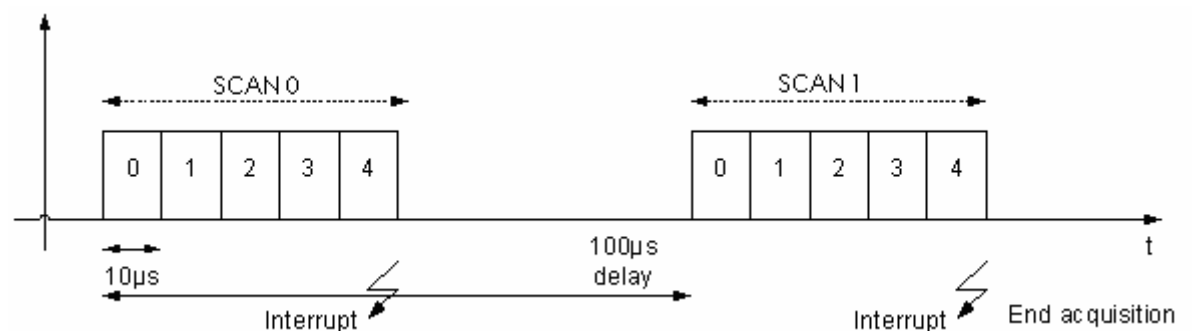
w_FirstChannel = 0
 w_LastChannel = 4
 b_ConvertingTimeUnit = 1 (μ s)
 dw_ConvertingTime = 10
 b_DelayTimeUnit; = 1 (μ s)
 dw_DelayTime = 20
 b_DelayTimeMode = ADDIDATA_DELAY_MODE2_USED
 b_SCANMode = ADDIDATA_DEFINED_SCAN_NUMBER
 dw_SCANCounter = 2



An interrupt occurs after each SCAN (5 acquisitions) and the acquisition is stopped after 2 SCAN cycles. A Delay from 20 μ s is respected between 2 SCAN cycles. The SCAN total time is 70 μ s.

Sample 4 :

w_FirstChannel = 0
 w_LastChannel = 4
 b_ConvertingTimeUnit = 1 (μ s)
 dw_ConvertingTime = 10
 b_DelayTimeUnit; = 1 (μ s)
 dw_DelayTime = 20
 b_DelayTimeMode = ADDIDATA_DELAY_MODE1_USED
 b_SCANMode = ADDIDATA_DEFINED_SCAN_NUMBER
 dw_SCANCounter = 2



An interrupt occurs after each SCAN (5 acquisitions) and the acquisition is stopped after 2 SCAN cycles. The SCAN total time is 100 μ s.

Calling convention:ANSI C :

```
BYTE                                     b_ReturnValue;
DWORD                                  dw_DriverHandle;
DWORD                                  dw_SCANHandle;
str_InitAnalogInputChannelSCANAcquisition s_InitParameters;

b_ReturnValue = b_ADDIDATA_InitAnalogInputSCANAcquisition
(dw_DriverHandle, &s_InitParameters,
sizeof (str_InitAnalogInputChannelSCANAcquisition),
&dw_SCANHandle);
```

Return Value:

1: No error

0: Error by calling up the function. Use the function "i_ADDIDATA_GetLastError", to find the error source.

3) **b_ADDIDATA_StartAnalogInputSCAN (..)**

Syntax:

```
<Return value> = b_ADDIDATA_StartAnalogInputSCAN
                                (DWORD    dw_DriverHandle,
                                DWORD    dw_SCANHandle)
```

Parameters:
- Input:

DWORD	<i>dw_DriverHandle</i>	Driver handle
DWORD	<i>dw_SCANHandle</i>	SCAN handle. This handle is returned by function "b_ADDIDATA_InitAnalogInputSCAN"

- Output:

No output signal has occurred.

Task:

Starts the selected SCAN (*dw_SCANHandle*).

Calling convention:
ANSI C:

```
BYTE    b_ReturnValue;
DWORD   dw_DriverHandle;
DWORD   dw_SCANHandle;
```

```
b_ReturnValue = b_ADDIDATA_StartAnalogInputSCAN
                                (dw_DriverHandle,
                                dw_SCANHandle);
```

Return value:

1: No error

0: Error by calling up the function. Use the function "i_ADDIDATA_GetLastError", to find the error source.

4) b_ADDIDATA_GetAnalogInputSCANStatus (..)

Syntax:

```
<Return value> = b_ADDDATA_GetAnalogInputSCANStatus
                                     (DWORD    dw_DriverHandle,
                                     DWORD    dw_SCANHandle,
                                     PBYTE    pb_SCANStatus)
```

Parameters:

- Input:

DWORD	<i>dw_DriverHandle</i>	Driver handle
DWORD	<i>dw_SCANHandle</i>	SCAN handle. This handle is returned by the "b_ADDIDATA_InitAnalogInputSCAN" function

- Output:

PBYTE	<i>pb_SCANStatus</i>	Status of the SCAN. 0: SCAN not started 1: SCAN started 2: SCAN completed 3: SCAN completed and new SCAN started
-------	----------------------	--

Task:

Returns the status (*pb_SCANStatus*) of the selected SCAN (*dw_SCANHandle*).

Calling convention:

ANSI C :

```

BYTE      b_ReturnValue;
DWORD     dw_DriverHandle;
DWORD     dw_SCANHandle;
BYTE      b_SCANStatus;

```

```
b_ReturnValue = b_ADDDATA_GetAnalogInputSCANStatus(dw_DriverHandle, dw_SCANHandle, &b_SCANStatus);
```

Return value:

1: No error
0: Error by calling up the function. Use the function "i_ADDIDATA_GetLastError",
to find the error source.

5) **b_ADDIDATA_ConvertDigitalToRealAnalogValueSCAN (.)**

Syntax:

```
<Return value> = b_ADDIDATA_ConvertDigitalToRealAnalogValueSCAN
                                (DWORD      dw_DriverHandle,
                                DWORD      dw_SCANHandle,
                                PDWORD     pdw_DigitalValueArray,
                                PDOUBLE    pd_RealValueArray)
```

Parameters:

- Input:

DWORD *dw_DriverHandle* Driver handle
 DWORD *dw_SCANHandle* SCAN handle. This handle is returned by the
 "b_ADDIDATA_InitAnalogInputSCAN" function
 PDWORD *pdw_DigitalValueArray* Digital value array of the analog input

- Output:

PDOUBLE *pd_RealValueArray* Real analog value array

Task:

Converts the digital value array of the analog input into a real analog value array.

Calling convention:

ANSI C:

```
BYTE      b_ReturnValue;
DWORD     dw_DriverHandle;
DWORD     dw_SCANHandle;
DWORD     dw_DigitalValueArray [600];
DOUBLE    d_RealValueArray [600];
```

```
b_ReturnValue = b_ADDIDATA_ConvertDigitalToRealAnalogValueSCAN
                                (dw_DriverHandle,
                                dw_SCANHandle,
                                dw_DigitalValueArray,
                                d_RealValueArray);
```

Return value:

1: No error
 0: Error by calling up the function. Use the function "i_ADDIDATA_GetLastError",
 to find the error source.

6) **b_ADDIDATA_StopAnalogInputSCAN (..)**

Syntax:

<Return value> = b_ADDIDATA_StopAnalogInputSCAN
(DWORD dw_DriverHandle,
DWORD dw_SCANHandle)

Parameters:

- Input:

DWORD	<i>dw_DriverHandle</i>	Driver handle
DWORD	<i>dw_SCANHandle</i>	SCAN handle. This handle is returned by the "b_ADDIDATA_InitAnalogInputSCAN" function

- Output:

No output signal has occurred.

Task:

Stops the selected SCAN (*dw_SCANHandle*).

Calling convention:

ANSI C:

```

BYTE      b_ReturnValue;
DWORD     dw_DriverHandle;
DWORD     dw_SCANHandle;

```

```
b_ReturnValue = b_ADDIDATA_StopAnalogInputSCAN(dw_DriverHandle, dw_SCANHandle);
```

Return value:

1: No error

0: Error by calling up the function. Use the function "i_ADDIDATA_GetLastError", to find the error source.

7) **b_ADDIDATA_CloseAnalogInputSCAN (..)**

Syntax:

<Return value> = b_ADDIDATA_CloseAnalogInputSCAN
(DWORD dw_DriverHandle,
DWORD dw_SCANHandle)

Parameters

- Input:

DWORD	<i>dw_DriverHandle</i>	Driver handle
DWORD	<i>dw_SCANHandle</i>	SCAN handle. This handle is returned by the "b_ADDIDATA_InitAnalogInputSCAN" function

- Output:

No output signal has occurred.

Task:

Closes the selected SCAN (*dw SCANHandle*) and releases the SCAN handle.

Calling convention:

ANSI C :

```

BYTE      b_ReturnValue;
DWORD     dw_DriverHandle;
DWORD     dw_SCANHandle;

```

```
b_ReturnValue = b_ADDIDATA_CloseAnalogInputSCAN(dw_DriverHandle, dw_SCANHandle);
```

Return value:

1: No error
0: Error by calling up the function. Use the function "i_ADDIDATA_GetLastError",
to find the error source.

2.6 Sequence acquisition



IMPORTANT!

This functionality can only be operated on one module.

1) **b_ADDIDATA_InitAnalogInputSequenceAcquisition (..)**

Syntax:

```
<Return value> = b_ADDIDATA_InitAnalogInputSequenceAcquisition
(DWORD    dw_DriverHandle,
DWORD    dw_NbrOfChannel,
PWORD    pw_SequenceChannelArray,
str_InitAnalogMeasureSequenceAcquisition
ps_InitParam,
DWORD    dw_StructSize,
PDWORD   pd_SEQHandle)
```

Input:

DWORD	dw_DriverHandle	Driver handle
DWORD	dw_NbrOfChannel	Number of channel in the sequence.
PDWORD	pdw_SequenceChannelArray	Array of index of the channel to be converted.

str_InitAnalogMeasureSequenceAcquisition ps_InitParam

Typedef struct

```
{
    BYTE    b_ConvertingTimeUnit;
    BYTE    b_DelayTimeMode;
    BYTE    b_DelayTimeUnit;
}
```

Selected conversion time unit
For time unit:
0: ns selected
1: μ s selected
2: ms selected
3: s selected
For frequency unit:
0: MHz selected
1: KHz selected
2: Hz selected
3: mHz selected
Delay mode selection
Refer to table 2-3, sample 7 and sample 8
Delay time unit selection
Refer to sample 7 and sample 8
For time unit:
0: ns selected
1: μ s selected
2: ms selected
3: s selected

For frequency unit:

- 0: MHz selected
- 1: KHz selected
- 2: Hz selected
- 3: mHz selected

```

BYTE    b_Reserved [5];
DWORD   dw_DelayTime;

DWORD   dw_ConvertingTime;
DWORD   dw_SequenceCounter;

DWORD   dw_InterruptSequenceCounter;

}str_InitAnalogMeasureSequenceAcquisition,
*ptr_InitAnalogMeasureSequenceAcquisition

```

Delay time

Selected conversion time

Number of sequences.

Refer to table 2-4

Number of sequences before an interrupt is generated.

DWORD dw_StructSize Size of the structure.

Output:

PDWORD pdw_SEQHandle Return the Sequence Handle.

Task:

Initialises the analogue input acquisition sequence.

dw_SequenceCounter defines the number of sequences. A sequence is composed of the channels that are defined in the pdw_SequenceChannelArray.

dw_InterruptSequenceCounter defines the number of sequences before an interrupt is generated.



IMPORTANT!

In the sequence mode only an even number of acquisitions can be initialised and executed (see examples below).

Examples for possible acquisitions in the sequence mode (even numbers)	Examples for acquisitions that are not possible in the sequence mode (odd numbers)
dw_NbrOfChannel = 3, dw_InterruptSequenceCounter = 10 => Number of acquisitions = 3 * 10 = 30	Dw_NbrOfChannel = 3, dw_InterruptSequenceCounter = 11 => Number of acquisitions = 3 * 11 = 33
dw_NbrOfChannel = 1, dw_InterruptSequenceCounter = 8 => Number of acquisitions = 1 * 8 = 8	dw_NbrOfChannel = 1, dw_InterruptSequenceCounter = 7 => Number of acquisitions = 1 * 7 = 7
dw_NbrOfChannel = 2, dw_InterruptSequenceCounter = 5 => Number of acquisitions = 2 * 5 = 10	

Table 2-3: Delay time mode

<i>b_DelayTimeMode</i>	Mode description
ADDIDATA_DELAY_NOT_USED	The delay between two SCAN is not used
ADDIDATA_DELAY_MODE1_USED	The delay between two SCAN cycles is used. The delay is started after the first acquisition. After this delay a new SCAN cycle is started
ADDIDATA_DELAY_MODE2_USED	The delay between two SCAN cycles is used. The delay is started after the last SCAN channel acquisition. After this delay a new SCAN cycle is started

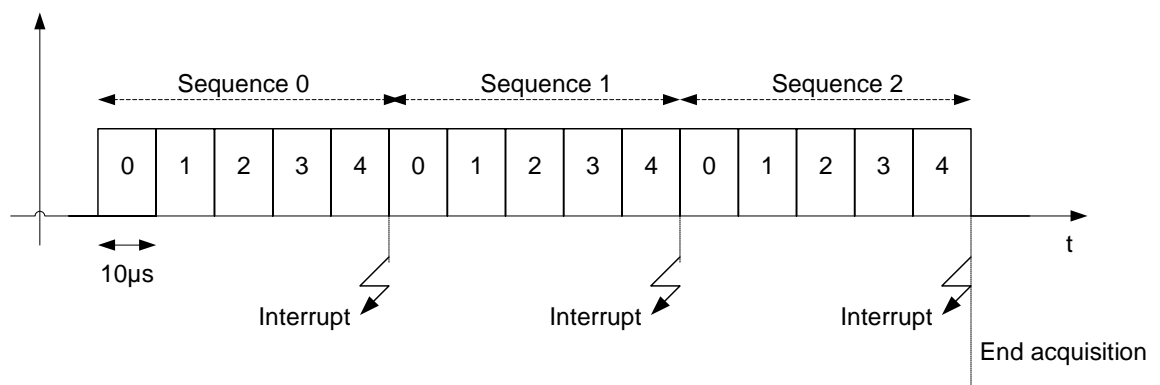
Table 2-4: Sequence mode

<i>dw_SequenceCounter</i>	Sequence description
> 0	A predefined number of sequences is started after calling up the function "b_ADDIDATA_StartAnalogInputSequenceAcquisition"
0	An undefined number of sequences is started after calling up the function "b_ADDIDATA_StartAnalogInputSequenceAcquisition"

Samples

Sample 1 :

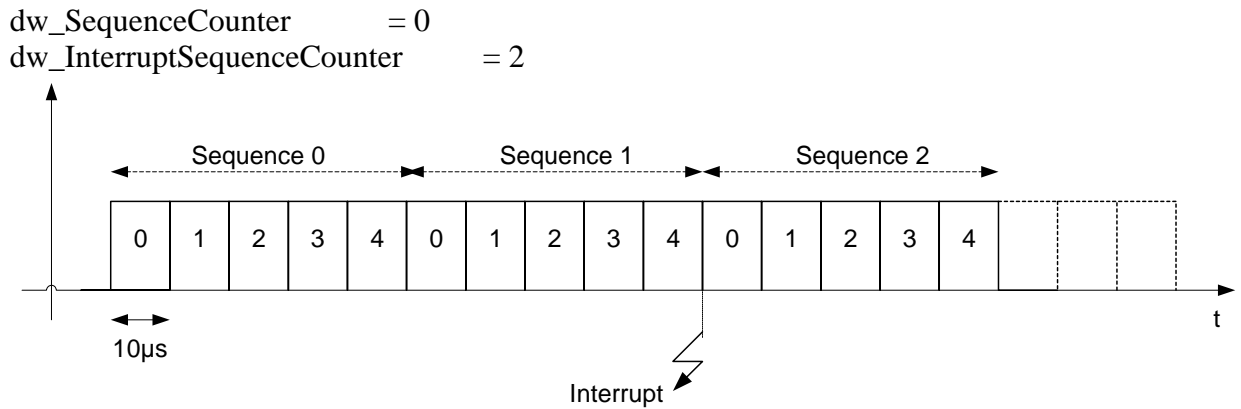
dw_NbrOfChannel = 5
 dw_SequenceChannelArray = 0,1,2,3,4
 b_ConvertingTimeUnit = 1 (μs)
 b_DelayTimeMode = ADDIDATA_DELAY_NOT_USED
 dw_ConvertingTime = 10
 dw_SequenceCounter = 3
 dw_InterruptSequenceCounter = 1



An interrupt occurs after each end of sequence (5 acquisitions) and the acquisition is stopped after 3 sequences.

Sample 2 :

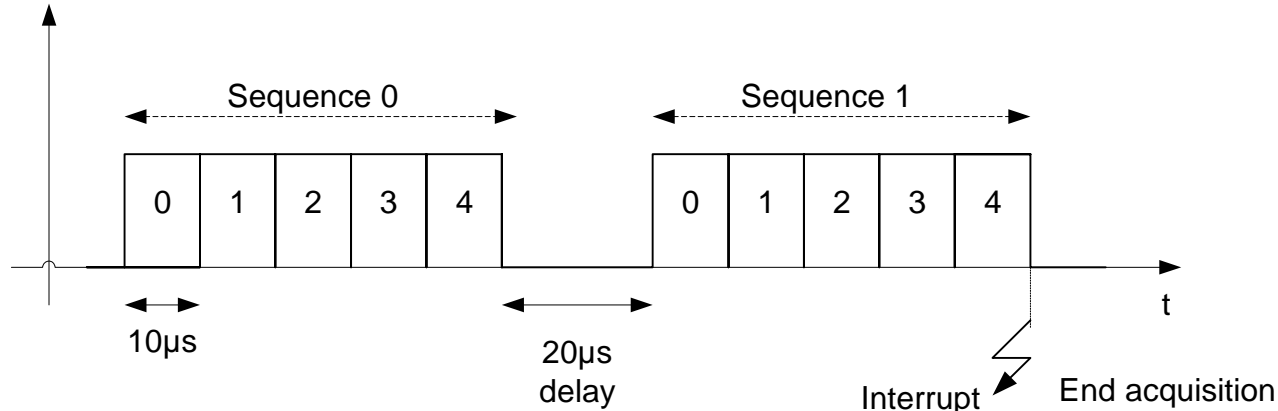
dw_NbrOfChannel = 5
 dw_SequenceChannelArray = 0,1,2,3,4
 b_ConvertingTimeUnit = 1 (μs)
 b_DelayTimeMode = ADDIDATA_DELAY_NOT_USED
 dw_ConvertingTime = 10



An interrupt occurs after 2 end of sequence (10 acquisitions) and the acquisition is stopped via the function `b_ADDIDATA_StopAnalogInputSequenceAcquisition`

Sample 3 :

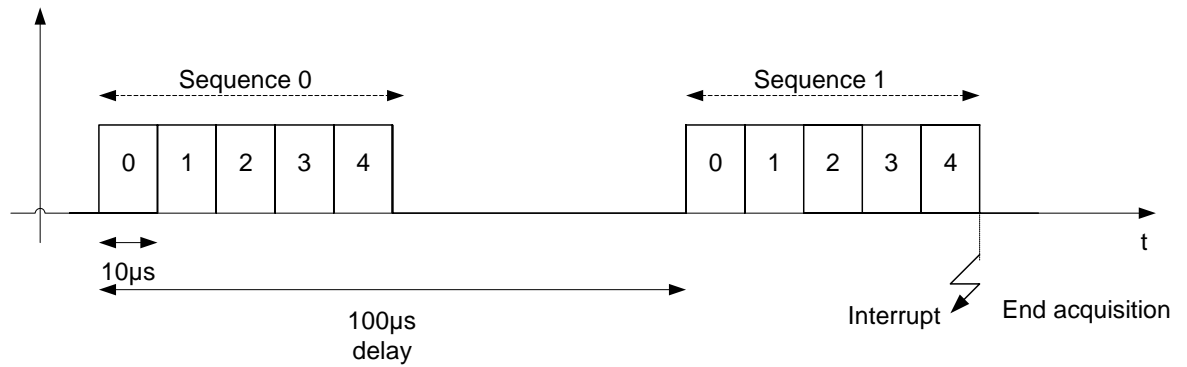
$dw_NbrOfChannel = 5$
 $dw_SequenceChannelArray = 0,1,2,3,4$
 $b_ConvertingTimeUnit = 1 (\mu s)$
 $b_DelayTimeMode = ADDIDATA_DELAY_MODE2_USED$
 $b_DelayTimeUnit; = 1 (\mu s)$
 $dw_DelayTime = 20$
 $dw_ConvertingTime = 10$
 $dw_SequenceCounter = 2$
 $dw_InterruptSequenceCounter = 2$



An interrupt occurs after 2 end of sequence (10 acquisitions) and the acquisition is stopped after 2 sequences. A delay from 20 µs is respected between 2 sequences.

Sample 4 :

$dw_NbrOfChannel = 5$
 $dw_SequenceChannelArray = 0,1,2,3,4$
 $b_ConvertingTimeUnit = 1 (\mu s)$
 $b_DelayTimeMode = ADDIDATA_DELAY_MODE1_USED$
 $b_DelayTimeUnit; = 1 (\mu s)$
 $dw_DelayTime = 100$
 $dw_ConvertingTime = 10$
 $dw_SequenceCounter = 2$
 $dw_InterruptSequenceCounter = 2$



An interrupt occurs after 2 end of sequence (10 acquisitions) and the acquisition is stopped after 2 sequences. The sequence total time is 100 µs.

Calling convention:

ANSI C :

```
BYTE      b_ReturnValue;
DWORD     dw_DriverHandle;
DWORD     dw_Channel[10];
DWORD     dw_SEQHandle;
str_InitAnalogMeasureSequenceAcquisition s_InitParameters;
```

```
b_ReturnValue = b_ADDIDATA_InitAnalogInputSequenceAcquisition
(dw_DriverHandle,
5,
dw_Channel,
s_InitParameters,
sizeof (str_InitAnalogMeasureSequenceAcquisition),
& dw_SEQHandle);
```

Return Value:

1: No error

0: Error by calling up the function. Use the function "i_ADDIDATA_GetLastError", to find the error source.

2) **b_ADDIDATA_StartAnalogInputSequenceAcquisition (..)**

Syntax:

<Return value> = b_ADDIDATA_StartAnalogInputSequenceAcquisition
(DWORD dw_DriverHandle,
DWORD dw_SEQHandle)

Parameters:

- Input:

DWORD	dw_DriverHandle	Driver handle
DWORD	dw_SEQHandle	Sequence handle. This handle is returned by the "b_ADDIDATA_InitAnalogInputSequenceAcquisition" function

- Output:

No output signal has occurred.

Task:

Starts the selected sequence (*dw_SEQHandle*).

Calling convention:

ANSI C :

```

BYTE      b_ReturnValue;
DWORD     dw_DriverHandle;
DWORD     dw_SEQHandle;

```

```
b_ReturnValue = b_ADDIDATA_StartAnalogInputSequenceAcquisition
                                     (dw_DriverHandle,
                                     dw_SEQHandle);
```


Return Value:

1: No error

0: Error by calling up the function. Use the function "i_ADDIDATA_GetLastError",
to find the error source.

3) **b_ADDIDATA_ConvertDigitalToRealAnalogValueSequence(..)**

Syntax:

```
<Return value> = b_ADDIDATA_ConvertDigitalToRealAnalogValueSequence
                    (DWORD      dw_DriverHandle,
                     DWORD      dw_SEQHandle,
                     PDWORD     pdw_DigitalValueArray,
                     PDOUBLE    pd_RealValueArray)
```

Parameters:
- Input:

DWORD	dw_DriverHandle	Driver handle
DWORD	dw_SEQHandle	Sequence handle. This handle is returned by the "b_ADDIDATA_InitAnalogInputSequenceAcquisition" function

PDWORD	pdw_DigitalValueArray	Digital value array for the analog input
--------	-----------------------	--

- Output:

PDOUBLE	pd_RealValueArray	Real analog input value array
---------	-------------------	-------------------------------

Task:

Converts the digital analog input value array into a real analog input value array.

Calling convention:
ANSI C:

```
BYTE      b_ReturnValue;
DWORD     dw_DriverHandle;
DWORD     dw_SEQHandle;
DWORD     dw_DigitalValueArray [600];
DOUBLE    d_RealValueArray [600];
```

```
b_ReturnValue = b_ADDIDATA_ConvertDigitalToRealAnalogValueSequenceAcquisition
                (dw_DriverHandle,
                 dw_SEQHandle,
                 dw_DigitalValueArray,
                 d_RealValueArray);
```

Return Value:

1: No error

0: Error by calling up the function. Use the function "i_ADDIDATA_GetLastError", to find the error source.

4) b_ADDIDATA_GetAnalogInputSequenceAcquisitionHandleStatus (..)

Syntax:

<Return value> = b_ADDIDATA_GetAnalogInputSequenceAcquisitionHandleStatus

(DWORD	dw_DriverHandle,
WORD	w_Module,
PBYTE	pb_InitialisationStatus,
PDWORD	pdw_LastInitialisedSEQHandle,
PBYTE	pb_CurrentSEQStatus
PDWORD	pdw_CurrentSEQHandle)

Parameters:

- Input:

DWORD	dw_DriverHandle	Driver handle
WORD	w_Module	AnalogInput module index
DWORD	dw_SEQHandle	Sequence handle. This handle is returned by the "b_ADDIDATA_InitAnalogInputSequenceAcquisition" function

-Output:

PBYTE	pb_InitialisationStatus	: 0:	No sequence initialised
		>0:	Number of initialised sequence
PDWORD	pdw_LastInitialisedSEQ	Handle:	Last initialised sequence handle
PBYTE	pb_CurrentSEQStatus	:	Current sequence status
		0:	No sequence started
		1:	Sequence started
		2:	Sequence pause
		3:	Sequence ended
PDWORD	pdw_CurrentSEQHandle:		Current sequence handle
PDWORD	pdw_SequenceCounter		Returns the number of acquired sequences

Task:

Gets the sequence handle status.

Returns the last initialised sequence status, the selected acquisition sequence handle and the status.

Returns the current number of acquired sequences.

Calling convention:

ANSI C :

BYTE	b_ReturnValue;
WORD	w_Module;
BYTE	b_InitialisationStatus;
DWORD	dw_LastInitialisedSEQHandle;
BYTE	b_CurrentSEQStatus;
DWORD	dw_CurrentSEQHandle;

```
b_ReturnValue = b_ADDIDATA_GetAnalogInputSequenceAcquisitionHandleStatus
(dw_DriverHandle,
w_Module,
```

```
&b_InitialisationStatus,  
&dw_LastInitialisedSEQHandle,  
&b_CurrentSEQStatus  
&dw_CurrentSEQHandle);
```

Return Value:

1: No error

0: Error by calling up the function. Call the function "i_ADDIDATA_GetLastError",
to find the error number.

5) **b_ADDIDATA_StopAnalogInputSequenceAcquisition (..)**

Syntax:

<Return value> = b_ADDIDATA_StopAnalogInputSequenceAcquisition
(DWORD dw_DriverHandle,
DWORD dw_SEQHandle)

Parameters:

- Input:

DWORD	dw_DriverHandle	Driver handle
DWORD	dw_SEQHandle	Sequence handle. This handle is returned by the "b_ADDIDATA_InitAnalogInputSequenceAcquisition" function

- Output:

No output signal has occurred.

Task:

Stops the selected sequence (*dw_SEQHandle*).

Calling convention:

ANSI C :

```

BYTE      b_ReturnValue;
DWORD     dw_DriverHandle;
DWORD     dw_SEQHandle;

```

```
b_ReturnValue = b_ADDIDATA_StopAnalogInputSequenceAcquisition
                                     (dw_DriverHandle,
                                     dw_SEQHandle);
```

Return Value:

1: No error

0: Error by calling up the function. Use the function "i_ADDIDATA_GetLastError", to find the error source.

6) **b_ADDIDATA_ReleaseAnalogInputSequenceAcquisition (..)**

Syntax:

<Return value> = b_ADDIDATA_ReleaseAnalogInputSequenceAcquisition
(DWORD dw_DriverHandle,
DWORD dw_SEQHandle)

Parameters:

- Input:

DWORD	dw_DriverHandle	Driver handle
DWORD	dw_SEQHandle	Sequence handle. This handle is returned by the "b_ADDIDATA_InitAnalogInputSequenceAcquisition" function

- Output:

No output signal has occurred.

Task:

Releases the selected sequence (*dw_SEQHandle*) and lets the resources free for another process.

Calling convention:

ANSI C:

BYTE	b_ReturnValue;
DWORD	dw_DriverHandle;
DWORD	dw_SEOHandle;

```
b_ReturnValue = b_ADDIDATA_ReleaseAnalogInputSequenceAcquisition(
    dw_DriverHandle,
    dw_SEQHandle);
```

Return Value:

1: No error
0: Error by calling up the function. Use the function "i_ADDIDATA_GetLastError",
to find the error source.

2.7 Hardware trigger

1) b_ADDIDATA_GetAnalogInputHardwareTriggerInformation (...)

Syntax:

```
<Return Value> = b_ADDIDATA_GetAnalogInputHardwareTriggerInformation
    (DWORD   dw_DriverHandle,
     WORD    w_Module,
     pstr_AnalogInputHardwareTriggerInformation
     ps_TriggerInformation,
     DWORD dw_StructSize)
```

Parameters:

- Input:

DWORD dw_DriverHandle	Handle of the ADDI-DATA driver
WORD w_Module	Index of the analog input module
DWORD dw_StructSize	Size of the structure.

- Output:

pstr_AnalogInputHardwareTriggerInformation

ps_TriggerInformation Hardware trigger information.

Returned structure:

```
typedef struct
{
    BYTE b_LowLevelTrigger;                      0: Hardware low trigger
                                                    level not available
                                                    1: Hardware low trigger
                                                    level available
    BYTE b_HighLevelTrigger;                    0: Hardware high trigger
                                                    level not available
                                                    1: Hardware high trigger
                                                    level available
    BYTE b_HardwareTriggerCount;                0: Hardware trigger counter
                                                    not available
                                                    1: Hardware trigger counter
                                                    available
    BYTE b_HardwareTriggerAutoRefreshAvailableMode;      D3 to D0
                                                    XXX1 : One shot trigger
                                                    available
                                                    XX1X : Single auto refresh
                                                    trigger available
                                                    X1XX : X auto refresh
                                                    trigger available
    BYTE b_HardwareTriggerSCANAvailableMode;      XXX1 : One shot trigger
                                                    available
```

	XX1X : Single scan trigger available
	X1XX : X scan trigger available
BYTE b_HardwareTriggerSequenceAvailableMode;	XXX1 : One shot trigger available
	XX1X : Single sequence trigger available
	X1XX : X sequence trigger available
BYTE b_Reserverd1 [2];	
DWORD dw_MaxTriggerCountValue;	Max number of trigger pulses before the hardware trigger action occurs
BYTE b_Reserverd2 [4];	
} str_AnalogInputHardwareTriggerInformation,	
*ptr_AnalogInputHardwareTriggerInformation;	

Task:

Return the hardware trigger available trigger actions and the available configuration mode

Calling convention:ANSI C:

BYTE	b_ReturnValue;
DWORD	dw_DriverHandle;
str_AnalogInputHardwareTriggerInformation	s_TriggerInformation;
b_ReturnValue = b_ADDIDATA_GetAnalogInputHardwareTriggerInformation	
	(dw_DriverHandle,
	0,
	& s_TriggerInformation,
	sizeof(s_TriggerInformation));

Return value:

1: No error

0: Error by calling up the function. Use the function "i_ADDIDATA_GetLastError", to find the error source.

2) **b_ADDIDATA_EnableDisableAnalogInputHardwareTrigger (...)****Syntax:**

<Return Value> = b_ADDIDATA_EnableDisableAnalogInputHardwareTrigger

(DWORD dw_DriverHandle,
 WORD w_Module,
 BYTE b_HardwareTriggerFlag,
 BYTE b_HardwareTriggerLevel,
 BYTE b_HardwareTriggerAction,
 DWORD

dw_HardwareTriggerCycleCount

DWORD dw_HardwareTriggerCount,
 DWORD dw_TimeOut)

Parameters:**- Input:**

DWORD	dw_DriverHandle	Handle of the ADDI-DATA driver
WORD	w_Module	Index of the module
BYTE	b_HardwareTriggerFlag	ADDIDATA_ENABLE: Enables the hardware trigger. ADDIDATA_DISABLE: Hardware trigger disabled by triggering the analog input module
BYTE	b_HardwareTriggerLevel	ADDIDATA_LOW: If the hardware trigger is used, it triggers from "1" to "0" ADDIDATA_HIGH: If the hardware trigger is used, it triggers from "0" to "1" ADDIDATA_HIGH_LOW: If the hardware trigger is used, it triggers from "0" to "1" or from "1" to "0"
BYTE	b_HardwareTriggerAction	Trigger action selection Refer to table 2-5
DWORD	dw_HardwareTriggerCycleCount	Define the number of sequences, auto refresh cycles or SCAN cycles to trigger Refer to table 2-5
DWORD	dw_HardwareTriggerCount	Hardware trigger counter. Define the number of trigger events before the action occur (> 0)
DWORD	dw_TimeOut	Define the time out for the ADDIDATA_TRIGGER_START_A_SINGLE_CONVERSION mode. Unit is ms. 0: no time out used

- Output:

No output signal has occurred.

Task:

Releases or blocks the action of the hardware trigger.

Table 2-5: Hardware trigger action

<i>b_HardwareTriggerAction</i>	Mode description
ADDIDATA_TRIGGER_START_A_SINGLE_CONVERSION	After each <i>dw_HardwareTriggerCount</i> trigger a single conversion is started
ADDIDATA_ONE_SHOT_TRIGGER	After the first <i>dw_HardwareTriggerCount</i> trigger the conversion are started. All next trigger have not effect. The trigger are rearmed after the next call from the function "b_ADDIDATA_StartAnalogInputSequenceAcquisition " or "b_ADDIDATA_StartAnalogInputSCAN " "b_ADDIDATA_StartAnalogInputAutoRefreshAcquisition "
ADDIDATA_TRIGGER_START_A_SINGLE_AUTO_REFRES H	After <i>dw_HardwareTriggerCount</i> trigger a single auto refresh cycle is started
ADDIDATA_TRIGGER_START_A_AUTO_REFRESH_SERIES	After each <i>dw_HardwareTriggerCount</i> trigger a series of <i>dw_HardwareTriggerCycleCount</i> auto refresh cycles is started.
ADDIDATA_TRIGGER_START_A_SINGLE_SCAN	After <i>dw_HardwareTriggerCount</i> trigger a single SCAN is started
ADDIDATA_TRIGGER_START_A_SCAN_SERIES	After each <i>dw_HardwareTriggerCount</i> trigger a series of <i>dw_HardwareTriggerCycleCount</i> SCAN is started.
ADDIDATA_TRIGGER_START_A_SINGLE_SEQUENCE	After <i>dw_HardwareTriggerCount</i> trigger a single sequence is started
ADDIDATA_TRIGGER_START_A_SEQUENCE_SERIES	After each <i>dw_HardwareTriggerCount</i> trigger a series of <i>dw_HardwareTriggerCycleCount</i> sequences is started.

Calling convention:ANSI C:

BYTE b_ReturnValue;
 DWORD dw_DriverHandle;

b_ReturnValue = b_ADDIDATA_EnableDisableAnalogInputHardwareTrigger
 (dw_DriverHandle,
 0,
 ADDIDATA_ENABLE,
 ADDIDATA_HIGH,

ADDIDATA_TRIGGER_START_A_SINGLE_CONVERSION,
 0,
 1,
 0);

Return value:

1: No error

0:Error by calling up the function. Use the function "i_ADDIDATA_GetLastError", to find the error source.

3) **b_ADDIDATA_GetAnalogInputHardwareTriggerStatus (...)**

Syntax:

```
<Return Value> = b_ADDIDATA_GetAnalogInputHardwareTriggerStatus
                                (DWORD    dw_DriverHandle,
                                 WORD      w_Module,
                                 PBYTE     pb_HardwareTriggerFlag,
                                 PBYTE     pb_HardwareTriggerStatus,
                                 PDWORD    pdw_HardwareTriggerCount,
                                 PBYTE     pb_HardwareTriggerState)
```

Parameters:
- Input:

DWORD	dw_DriverHandle	Handle of the ADDI-DATA driver
WORD	w_Module	Index of the analog input module

- Output:

PBYTE	pb_HardwareTriggerFlag	ADDIDATA_ENABLE: The hardware trigger is enabled. ADDIDATA_DISABLE: The hardware trigger is disabled
PBYTE	pb_HardwareTriggerStatus	0: Hardware trigger did not occur 1: Hardware trigger occurred
PDWORD	pdw_HardwareTriggerCount	Number of pulses that fail before the next trigger occurs
PBYTE	pb_HardwareTriggerState	0: Hardware trigger is not active (Low state) 1: Hardware trigger is active (High state)

Task:

Returns the status (occur or not), the state from input (active or not) and the number of that fail before the next trigger occur.

Calling convention:
ANSI C:

```
BYTE    b_ReturnValue;
DWORD   dw_DriverHandle;
BYTE    b_HardwareTriggerFlag;
BYTE    b_HardwareTriggerStatus;
BYTE    b_HardwareTriggerState;
DWORD   dw_HardwareTriggerCount;
```

```
b_ReturnValue = b_ADDIDATA_GetAnalogInputHardwareTriggerStatus
                (dw_DriverHandle,
                 0,
                 &b_HardwareTriggerFlag,
                 &b_HardwareTriggerStatus,
                 &dw_HardwareTriggerCount,
                 &b_HardwareTriggerState);
```

Return value:

- 1: No error
- 0: Error by calling up the function. Use the function "i_ADDIDATA_GetLastError", to find the error source.

2.8 Software trigger functions

1) **b_ADDIDATA_GetAnalogInputSoftwareTriggerInformation (...)**

Syntax:

```
<Return Value> = b_ADDIDATA_GetAnalogInputSoftwareTriggerInformation
(DWORD
 dw_DriverHandle,
 WORD
 pstr_AnalogInputSoftwareTriggerInformation
 ps_TriggerInformation,
 DWORD
 dw_StructSize)
```

Parameters:

- Input:

DWORD	dw_DriverHandle	Handle of the ADDI-DATA driver
WORD	w_Module	Index of the analog input module
DWORD	dw_StructSize	Size of the structure.

- Output:

pstr_AnalogInputSoftwareTriggerInformation ps_TriggerInformation
Software

trigger
information.

Returned structure:

```
typedef struct
{
    BYTE  b_SoftwareTriggerAutoRefreshAvailableMode;  D3 to D0
                                                       XXX1 : One shot
                                                       trigger available
                                                       XX1X : Single auto
                                                       refresh trigger
                                                       available
                                                       X1XX :
                                                       X autorefresh
                                                       trigger available
    BYTE  b_SoftwareTriggerSCANAvailableMode;         XXX1 : One shot
                                                       trigger available
                                                       XX1X : Single scan
                                                       trigger available
                                                       X1XX : X scan
                                                       trigger available
    BYTE  b_SoftwareTriggerSequenceAvailableMode;    XXX1 : One shot
                                                       trigger available
                                                       XX1X : Single
                                                       sequence trigger
                                                       available
                                                       X1XX : X sequence
                                                       trigger available
    BYTE  b_Reserverd1 [5];
```

```
} str_AnalogInputSoftwareTriggerInformation,  
 *pstr_AnalogInputSoftwareTriggerInformation;
```

Task:

Return the software trigger available actions.

Calling convention:ANSI C:

```
BYTE b_ReturnValue;  
DWORD dw_DriverHandle;  
str_AnalogInputSoftwareTriggerInformation s_TriggerInformation;  
  
b_ReturnValue = b_ADDIDATA_GetAnalogInputSoftwareTriggerInformation  
                (dw_DriverHandle,  
                0,  
                & s_TriggerInformation,  
                sizeof(s_TriggerInformation));
```

Return value:

1: No error

0: Error by calling up the function. Use the function "i_ADDIDATA_GetLastError",
to find the error source.

2) **b_ADDIDATA_EnableDisableAnalogInputSoftwareTrigger (...)**

Syntax:

<Return Value> = b_ADDIDATA_EnableDisableAnalogInputSoftwareTrigger
 (DWORD dw_DriverHandle,
 WORD w_Module,
 BYTE b_SoftwareTriggerFlag,
 BYTE b_SoftwareTriggerAction)

Parameters:

- Input:

DWORD dw_DriverHandle Handle of the ADDI-DATA driver
 WORD w_Module Index of the module
 BYTE b_SoftwareTriggerFlag ADDIDATA_ENABLE: Enables the Software trigger.
 ADDIDATA_DISABLE: Software trigger disabled by triggering the analog input module
 BYTE b_SoftwareTriggerAction Trigger action selection
 Refer to table 2-6

- Output:

No output signal has occurred.

Task:

Releases or blocks the action of the Software trigger.

The software trigger are making via the function

“b_ADDIDATA_AnalogInputSoftwareTrigger”

Table 2-6: Software trigger action

<i>b_HardwareTriggerAction</i>	Mode description
ADDIDATA_TRIGGER_START_A_SINGLE_CONVERSION	After each software trigger a single conversion is started
ADDIDATA_ONE_SHOT_TRIGGER	After the first software trigger the conversion is started. All next triggers have not effect. The trigger are read after the next call from the function "b_ADDIDATA_StartAnalogInputSequenceAcquisition" or "b_ADDIDATA_StartAnalogInputSCAN" or "b_ADDIDATA_StartAnalogInputAutoRefreshAcquisition"
ADDIDATA_TRIGGER_START_A_SINGLE_AUTO_REFRES H	After software trigger a single auto refresh cycle is started
ADDIDATA_TRIGGER_START_A_AUTO_REFRESH_SERIES	After each software trigger a series of <i>dw_HardwareTriggerCycleCount</i> auto refresh cycles are started.
ADDIDATA_TRIGGER_START_A_SINGLE_SCAN	After software trigger a single SCAN is started
ADDIDATA_TRIGGER_START_A_SCAN_SERIES	After each software trigger a series of <i>dw_HardwareTriggerCycleCount</i> SCAN are started.
ADDIDATA_TRIGGER_START_A_SINGLE_SEQUENCE	After software trigger a single sequence is started
ADDIDATA_TRIGGER_START_A_SEQUENCE_SERIES	After each software trigger a series of <i>dw_HardwareTriggerCycleCount</i> sequence are started.

Calling convention:

ANSI C:

```
BYTE    b_ReturnValue;  
DWORD dw_DriverHandle;
```

```
b_ReturnValue = b_ADDIDATA_EnableDisableAnalogInputSoftwareTrigger  
                (dw_DriverHandle,  
                 0,  
                 ADDIDATA_ENABLE,
```

```
                ADDIDATA_TRIGGER_START_A_SINGLE_CONVERSION);
```

Return value:

1: No error

0: Error by calling up the function. Use the function "i_ADDIDATA_GetLastError",
to find the error source.

3) b_ADDIDATA_GetAnalogInputSoftwareTriggerStatus (...)

Syntax:

```
<Return Value> = b_ADDIDATA_GetAnalogInputSoftwareTriggerStatus  
                (DWORD    dw_DriverHandle,  
                 WORD     w_Module,  
                 PBYTE    pb_SoftwareTriggerFlag,  
                 PBYTE    pb_SoftwareTriggerStatus)
```

Parameters:

- Input:

DWORD	dw_DriverHandle	Handle of the ADDI-DATA driver
WORD	w_Module	Index of the analog input module

- Output:

PBYTE	pb_SoftwareTriggerFlag	ADDIDATA_ENABLE: The software trigger is enabled. ADDIDATA_DISABLE: The software trigger is disabled
PBYTE	pb_SoftwareTriggerStatus	0: Software trigger did not occur 1: Software trigger occurred

Task:

Returns the status (occur or not) of the selected analog input software trigger.

Calling convention:

ANSI C :

```

BYTE      b_ReturnValue;
DWORD     dw_DriverHandle;
BYTE      b_SoftwareTriggerFlag;
BYTE      b_SoftwareTriggerStatus;

```

```
b_ReturnValue = b_ADDIDATA_GetAnalogInputSoftwareTriggerStatus
(dw_DriverHandle,
0,
&b_SoftwareTriggerFlag,
&b_SoftwareTriggerStatus);
```

Return value:

1: No error
0: Error by calling up the function. Use the function "i_ADDIDATA_GetLastError",
to find the error source.

4) b_ADDIDATA_AnalogInputSoftwareTrigger (...)

Syntax:

<Return Value> = b_ADDIDATA_AnalogInputSoftwareTrigger
(DWORD dw_DriverHandle,
WORD w_Module)

Parameters:

- Input:

DWORD	dw_DriverHandle	Handle of the ADDI-DATA driver
WORD	w_Module	Index of the analog input module

- Output:

No output signal has occurred.

Task:

Make a software trigger. The action depends from the initialisation. Refer to the function “b_ADDIDATA_EnableDisableAnalogInputSoftwareTrigger”

Calling convention:

ANSI C :

```

BYTE    b_ReturnValue;
DWORD   dw_DriverHandle;

```

```
b_ReturnValue = b_ADDIDATA_AnalogInputSoftwareTrigger
(dw_DriverHandle,
0,);
```

Return value:

1: No error

0: Error by calling up the function. Use the function "i_ADDIDATA_GetLastError",
to find the error source.

2.9 Other functions

1) **b_ADDIDATA_TestAnalogInputAsynchronousFIFOFull**

Syntax:

<Return value> = b_ADDIDATA_TestAnalogInputAsynchronousFIFOFull
(DWORD dw_DriverHandle,
PBYTE pb_Full)

Parameters:

- Input:

DWORD	<i>dw_DriverHandle</i>	Driver handle
-------	------------------------	---------------

- Output:

PBYTE	<i>pb_Full</i>	0: Asynchronous interrupt FIFO memory not full 1: Asynchronous interrupt FIFO memory is full
-------	----------------	---

Task:

Tests if the asynchronous interrupt FIFO memory is full or not.
The asynchronous interrupt FIFO memory is the FIFO memory in which the asynchronous events generated by the asynchronous interrupt are stored.

Calling convention:

ANSI C :

```

BYTE      b_ReturnValue;
DWORD     dw_DriverHandle;
BYTE      b_FIFOFull

```

```
b_ReturnValue = b_ADDIDATA_TestAnalogInputAsynchronousFIFOFull
(dw_DriverHandle,
&b_FIFOFull);
```

Return value:

1: No error
0: Error by calling up the function. Use the function "i_ADDIDATA_GetLastError",
to find the error source.