



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

Software description

ADDIDRIVER

Transducer

Edition: 03.02 – 07/2006

1	INTRODUCTION	1
2	SOFTWARE FUNCTIONS.....	2
2.1	General information	2
1)	b_ADDIDATA_GetNumberOfTransducerChannels (..)	2
2)	b_ADDIDATA_GetNumberOfTransducerModules (..)	3
3)	b_ADDIDATA_GetNumberOfTransducerChannelsForTheModule (..)	4
4)	b_ADDIDATA_GetTransducerChannelModuleNumber (..).....	5
5)	b_ADDIDATA_GetTransducerModuleGeneralInformation (..)	6
6)	b_ADDIDATA_GetTransducerModuleSingleAcquisitionInformation (..)	9
7)	b_ADDIDATA_GetTransducerModuleAutoRefreshInformation (..).....	10
8)	b_ADDIDATA_GetTransducerModuleSequenceInformation (..).....	12
9)	b_ADDIDATA_GetTransducerModuleConvertTimeDivisionFactorInformation (...).....	14
10)	b_ADDIDATA_InitTransducerModuleConvertTimeDivisionFactor (...)	15
11)	b_ADDIDATA_ReleaseTransducerModuleConvertTimeDivisionFactor (...)	16
2.2	Transducer initialisation.....	17
1)	b_ADDIDATA_InitTransducerChannel (..)	17
2)	b_ADDIDATA_ReleaseTransducerChannel (..).....	18
3)	b_ADDIDATA_GetTransducerModuleConvertTimeDivisionFactorInformation (...).....	19
4)	b_ADDIDATA_ReleaseTransducerModuleConvertTimeDivisionFactor (...)	21
2.3	Single acquisition.....	22
1)	b_ADDIDATA_Read1TransducerChannel (..).....	22
2)	b_ADDIDATA_ConvertDigitalToRealMetricValue (..)	23
3)	b_ADDIDATA_ReadMoreTransducerChannels (..).....	24
4)	b_ADDIDATA_ConvertMoreDigitalToRealMetricValues (..)	25
2.4	Autorefresh mode.....	26
1)	b_ADDIDATA_GetTransducerAutoRefreshChannelPointer (..)	26
2)	b_ADDIDATA_GetTransducerAutoRefreshModulePointer (..)	27
3)	b_ADDIDATA_GetTransducerAutoRefreshModuleCounterPointer (..)	28
4)	b_ADDIDATA_StartTransducerAutoRefresh (..)	30
5)	b_ADDIDATA_StopTransducerAutoRefresh (..).....	31
2.5	Auto Refresh Acquisition.....	32
1)	b_ADDIDATA_GetTransducerAutoRefreshChannelPointer (..)	32
2)	b_ADDIDATA_GetTransducerAutoRefreshModulePointer (..)	33
3)	b_ADDIDATA_GetTransducerAutoRefreshModuleCounterPointer (..)	35
4)	b_ADDIDATA_StartTransducerAutoRefresh (..)	36
5)	b_ADDIDATA_StopTransducerAutoRefresh (..).....	37
6)	b_ADDIDATA_Read1TransducerAutorefreshValue(..)	38
7)	b_ADDIDATA_ReadTransducerAutoRefreshCounterValue	39
8)	b_ADDIDATA_SetTransducerAutorefreshAverageMode(..)	40
2.6	Sequence acquisition	41
1)	b_ADDIDATA_InitTransducerSequenceAcquisition (..).....	41
2)	b_ADDIDATA_StartTransducerSequenceAcquisition (..).....	46
3)	b_ADDIDATA_PauseTransducerSequenceAcquisition (..)	47
4)	b_ADDIDATA_ConvertTransducerSequenceDigitalToRealMetricValue (..)	48
5)	b_ADDIDATA_StopTransducerSequenceAcquisition (..)	49
6)	b_ADDIDATA_GetTransducerSequenceAcquisitionHandleStatus (..)	50
7)	b_ADDIDATA_ReleaseTransducerSequenceAcquisition (..).....	51
2.7	Trigger/gate.....	52

1) b_ADDIDATA_GetTransducerHardwareTriggerInformation (...)	52
2) b_ADDIDATA_EnableDisableTransducerHardwareTrigger (...)	54
3) b_ADDIDATA_GetTransducerHardwareTriggerStatus (...)	56
4) b_ADDIDATA_GetTransducerHardwareTriggerInformationEx(..)	57
5) b_ADDIDATA_EnableDisableTransducerHardwareTriggerEx	59
6) b_ADDIDATA_EnableDisableTransducerSoftwareTrigger (...)	61
7) b_ADDIDATA_TransducerSoftwareTrigger (...)	62
8) b_ADDIDATA_GetTransducerSoftwareTriggerStatus (...)	63
9) b_ADDIDATA_GetTransducerHardwareGateInformation (...)	64
10) b_ADDIDATA_EnableDisableTransducerHardwareGate (...)	65
11) b_ADDIDATA_GetTransducerHardwareGateStatus (...)	66
2.8 Diagnostic	67
1) b_ADDIDATA_TestTransducerChannelSecondaryConnection (..)	67
2) b_ADDIDATA_EnableDisableTransducerModulePrimaryConnectionTest (..)	68
3) b_ADDIDATA_TestTransducerModulePrimaryConnection (..).....	69
4) b_ADDIDATA_EnableDisableTransducerModulePrimaryShortCircuitInterrupt (..)	70
5) b_ADDIDATA_RearmTransducerModulePrimaryShortCircuitConnectionTest (..)	71

Tables

Table 1-1: Type Declaration for Windows 98/NT/2000/XP	1
Table 2-1: Delay mode	42
Table 2-2: Selection of the converting time	42
Table 2-3: Sequence mode	42
Table 2-4: Action of the hardware/software trigger.....	55
Table 2-5: Action of the hardware/software trigger.....	60

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 SOFTWARE FUNCTIONS

2.1 General information

1) b_ADDIDATA_GetNumberOfTransducerChannels (..)

Syntax:

<Return value> = b_ADDIDATA_GetNumberOfTransducerChannels
(DWORD dw_DriverHandle,
WORD pw_ChannelNbr)

Parameters:

- Input:

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

- Output:

WORD <i>pw_ChannelNbr</i>	Returns the number of transducer channels
---------------------------	---

Task:

Returns the number of transducers.

Calling convention:

ANSI C :

BYTE	b_ReturnValue;
DWORD	dw_DriverHandle;
WORD	w_ChannelNbr;

```
b_ReturnValue = b_ADDIDATA_GetNumberOfTransducerChannels
                (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_GetNumberOfTransducerModules (..)

Syntax:

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

Parameters:

- Input:

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

- Output:

PCWORD <i>pw_ModuleNbr</i>	Returns the number of transducer modules
-----------------------------------	--

Task:

Returns the number of transducer modules.

Calling convention:

ANSI C:

BYTE	b_ReturnValue;
DWORD	dw_DriverHandle;
WORD	w_ModuleNbr;

```
b_ReturnValue = b_ADDIDATA_GetNumberOfTransducerModules
                                     (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_GetNumberOfTransducerChannelsForTheModule (..)

Syntax:

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

Parameters:
- Input:

DWORD	<i>dw_DriverHandle</i>	Driver handle
WORD	<i>w_Module</i>	Transducer module

- Output:

PWORD	<i>pw_ChannelNbr</i>	Returns the number of transducer channels for the given module.
-------	----------------------	---

Task:

Returns the number of transducer channels for the module *w_Module*.

Calling convention:
ANSI C :

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

```
b_ReturnValue = b_ADDIDATA_GetNumberOfTransducerChannelsForTheModule
                                (dw_DriverHandle,
                                0,
                                &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.

4) **b_ADDIDATA_GetTransducerChannelModuleNumber** (..)

Syntax:

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

Parameters:
- Input:

DWORD	<i>dw_DriverHandle</i>	Driver handle
WORD	<i>w_Channel</i>	Number of the transducer channel

- Output:

PWORD	<i>pw_Module</i>	Module number of the selected channel.
-------	------------------	--

Task:

Returns the number of the module (*pw_Module*) for the selected transducer channel (*w_Channel*).

Calling convention:
ANSI C :

```
BYTE    b_ReturnValue;
DWORD   dw_DriverHandle;
WORD    w_Module;
```

```
b_ReturnValue = b_ADDIDATA_GetTransducerChannelModuleNumber
                                (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_GetTransducerModuleGeneralInformation (..)****Syntax:**

```
<Return value> = b_ADDIDATA_GetTransducerModuleGeneralInformation
                    (DWORD    dw_DriverHandle,
                     WORD     w_Module,
                     pstr_TransducerModuleInformation ps_ModuleInformation,
                     DWORD    dw_StructSize)
```

Parameters:**- Input:**

DWORD	<i>dw_DriverHandle</i>	Driver handle
WORD	<i>w_Module</i>	Transducer module
DWORD	<i>dw_StructSize</i>	Size of the structure entered in the pointer

- Output:

pstr_TransduceModuleInformation	<i>ps_ModuleInformation</i>	Module information
---------------------------------	-----------------------------	--------------------

structure returned :

```
typedef struct
```

```
{
  BYTE    b_InputsResolution;      Returns the input resolution
                                         8: 8-bit resolution
                                         16: 16-bit resolution, ...
  BYTE    b_SingleAcquisition;      0 : Single acquisition not available
                                         1 : Single acquisition available
  BYTE    b_AutoRefreshAcquisition; 0:Autorefresh acquisition not available
                                         1 :Autorefresh 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_PrimaryOpenInputDetection;
                                         0:Open input detection not available on the
                                         primary circuit
                                         1:Open input detection available on the
                                         primary circuit
  BYTE    b_PrimaryShortCircuitDetection;
                                         0:Short-circuit detection not available on the
                                         primary circuit
                                         1:Short-circuit detection available on the
                                         primary circuit
  BYTE    b_SecondaryOpenInputDetection;
                                         0:Open input detection not available on the
                                         secondary circuit
                                         1:Open input detection available on the
                                         secondary circuit
  BYTE    b_SecondaryShortCircuitDetection;
                                         0: Short-circuit detection not available on the
                                         secondary circuit
                                         1: Short-circuit detection available on the
                                         secondary circuit
}
```

BYTE	<i>b_PrimaryOpenInputDetectionIRQ;</i>	0: No interrupt available for the open input detection on the primary circuit 1: Interrupt available for the open input detection on the primary circuit
BYTE	<i>b_PrimaryShortCircuitDetectionIRQ;</i>	0: No interrupt available for the short-circuit detection on the primary circuit 1: Interrupt available for the short-circuit detection on the primary circuit
BYTE	<i>b_SecondaryOpenInputDetectionIRQ;</i>	0: No interrupt available for the open input detection on the secondary circuit 1: Interrupt available for the open input detection on the secondary circuit
BYTE	<i>b_SecondaryShortCircuitDetectionIRQ;</i>	0: No interrupt available for the short-circuit detection on the secondary circuit 1: Interrupt available for the short-circuit detection on the secondary circuit
BYTE	<i>b_Reserved1 [3];</i>	
WORD	<i>w_FirstChannelNumber</i>	Returns the number from first channel number
WORD	<i>w_LastChannelNumber</i>	Returns the number from last channel number
BYTE	<i>b_InitialisationType</i>	Initialisation type for the channels of the module: 0: Each channel can have a different initialisation 1: All channels of the same group must have the same initialisation. The group is defined by the variable <i>b_GroupChannelNumber</i>
BYTE	<i>b_ChannelNumber</i>	Defines the number of channels per group
BYTE	<i>b_Reserved 2 [2]</i>	

Available Transducer information

BYTE	<i>b_NumberOfAvailableTransducersType;</i>	Returns the number of transducers types available.
------	--	--

Struct

```

{
  CHAR    c_TransducerType [104]; Transducer type (name)
  DOUBLE  d_Range;                Transducer range (mm)
  DOUBLE  d_Sensitivity;           Transducer sensitivity (mv/V/mm)
  DOUBLE  d_NominalFrequency;     Nominal frequency (Hz)
  DOUBLE  d_MinFrequency;         Minimal frequency (Hz)
  DOUBLE  d_MaxFrequency;         Maximal frequency (Hz)
  DOUBLE  d_PrimaryNominalVoltage; Primary nominal voltage (Veff)
  WORD    w_SelectionIndex;       Transducer index selection for the
                                   initialisation function;

  WORD    w_Reserved [3];
} s_TransducerInformation [50];
} str_TransducerModuleInformation;
```

Task:

Returns information about the transducer module and all transducer types available.

Calling convention:ANSI C :

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

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

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_GetTransducerModuleSingleAcquisitionInformation (..)

Syntax:

```
<Return value> = b_ADDIDATA_GetTransducerModuleSingleAcquisitionInformation
                                (DWORD    dw_DriverHandle,
                                 WORD      w_Module,
                                pstr_TransducerSingleAcquisitionInformation ps_ModuleInformation,
                                 DWORD     dw_StructSize)
```

Parameters:

- Input:

DWORD <i>dw_DriverHandle</i>	Driver handle
WORD <i>w_Module</i>	Transducer module number
DWORD <i>dw_StructSize</i>	Size of the structure entered in the pointer

- Output:

pstr_TransducerSingleAcquisitionInformation ps_ModuleInformation
Module information.

structure returned :

```
typedef struct
{
    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_Interrupt;          0: Interrupt cannot be generated
                                   1: Interrupt can be generated (EOC)
    BYTE    b_Reserved [4];
}str_TransducerSingleAcquisitionInformation;
```

Task:

Returns information about the transducer module for the single acquisition mode.

Calling convention:

ANSI C :

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

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

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_GetTransducerModuleAutoRefreshInformation (..)

Syntax:

```
<Return value> = b_ADDIDATA_GetTransducerModuleAutoRefreshInformation
                    (DWORD   dw_DriverHandle,
                     WORD     w_Module,
pstr_TransducerAutoRefreshInformation ps_ModuleInformation,
                     DWORD   dw_StructSize)
```

Parameters:

- Input:

DWORD <i>dw_DriverHandle</i>	Driver handle
WORD <i>w_Module</i>	Transducer module number.
DWORD <i>dw_StructSize</i>	Size of the structure entered in the pointer.

- Output:

pstr_TransducerAutoRefreshInformation	ps_ModuleInformation
	Module information.

structure returned :

```
typedef struct
```

```
{
    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_Interrupt;          0: Interrupt cannot be generated
                                1: Interrupt can be generated (EOC)
    BYTE   b_AccessMode          8: 8-bit access mode
                                16: 16-bit access mode
                                32: 32-bit access mode
```

```
    BYTE b_AutoRefreshAverageModePerChannel;
```

```
                                0: Auto Refresh Average Mode Per Channel not
                                available
                                1: Auto Refresh Average Mode Per Channel
                                available
```

```
    BYTE  b_Reserved [2];
```

```
    }str_TransducerAutoRefreshInformation, *pstr_TransducerAutoRefreshInformation;
```

```
    }str_TransducerAutoRefreshInformation;
```

Task:

Returns information about the transducer module for the autorefresh acquisition.

Calling convention:ANSI C :

```
BYTE    b_ReturnValue;  
DWORD   dw_DriverHandle;  
WORD    w_ChannelNbr;  
str_TransducerAutoRefreshInformation    s_ModuleInformation;  
  
b_ReturnValue = b_ADDIDATA_GetTransducerModuleAutoRefreshInformation  
                (dw_DriverHandle,  
                0,  
                &s_ModuleInformation,  
                sizeof (str_TransducerAutoRefreshInformation));
```

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_GetTransducerModuleSequenceInformation (..)****Syntax:**

```
<Return value> = b_ADDIDATA_GetTransducerModuleSequenceInformation
                    (DWORD      dw_DriverHandle,
                     WORD       w_Module,
                     pstr_TransducerSequenceInformation ps_ModuleInformation,
                     DWORD      dw_StructSize)
```

Parameters:**- Input:**

DWORD *dw_DriverHandle* Driver handle
 WORD *w_Module* Transducer module number.
 DWORD *dw_StructSize* Size of the structure entered in the pointer.

- Output:

pstr_TransducerSequenceInformation ps_ModuleInformation
 Module information.

structure returned:

```
typedef struct
```

```
{
  BYTE  b_SequenceConfigurable; 0: Sequence fixed. User muss pass the first and
                                last channel
                                1: Sequence configurable
  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_DelayTimeConfigurable; 0: No delay available after a sequence
                                1: Delay after each sequence available
  BYTE  b_DelayCalcType;          0: Binary type
                                (XX000,XX001,XX010,XX011)
                                1: Multiple type (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
 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, ...
 DWORD *dw_MaxNumberOfAcquisition*;
 Returns the maximum number of acquisitions for
 the single acquisition sequence mode
 }str_TransducerSequenceInformation;

Task:

Returns information about the transducer module for the autorefresh acquisition mode.

Calling convention:

ANSI C:

```

BYTE    b_ReturnValue;
DWORD   dw_DriverHandle;
WORD     w_ChannelNbr;
str_TransducerSequenceInformation s_ModuleInformation;

```

```

b_ReturnValue = b_ADDIDATA_GetTransducerModuleSequenceInformation
(dw_DriverHandle,
0,
&s_ModuleInformation,
sizeof(str_TransducerSequenceInformation));

```

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_GetTransducerModuleConvertTimeDivisionFactorInformation (...)****Syntax:**

```
<Return value> = b_ADDIDATA_GetTransducerModuleConvertTimeDivisionFactorInformation
                                (DWORD      dw_DriverHandle,
                                WORD        w_Module,
                                pstr_TransducerConvertTimeDivisionFactorInformation ps_ModuleInformation,
                                DWORD      dw_StructSize)
```

Parameters:**- Input:**

DWORD dw_DriverHandle	Driver handle
WORD w_Module	Transducer module selection
DWORD dw_StructSize	ps_ModuleInformation structure size

- Output:

pstr_TransducerConvertTimeDivisionFactorInformation ps_ModuleInformation:
Convert time division factor information structure

Structure declaration :

The structure contains :

BYTE b_Configurable	0: Converting time division factor is fixed 1: Converting time division factor is configurable. See function “b_ADDIDATA_InitTransducerModuleConvertTimeDivisionFactor”
BYTE b_Steps	Return the converting time division factor steps.
BYTE b_Initialised	0 : Converting time division factor not initialised 1 : Converting time division factor initialised
DWORD dw_InitialisationValue	If initialised, this parameter returns the initialisation value
DWORD dw_MinDivisionFactor	Returns the minimal division factor value
DWORD dw_MaxDivisionFactor	Returns the maximal division factor value

Task:

Gets the converting time division factor information. The converting time division factor sets the switching time from channel to channel.

The base time is the transducer frequency.

For example if the transducer connected to channel 0 uses the 14 kHz frequency and the division factor is set to 5 (dw_InitialisationValue = 5) then the switching time from channel 0 to the next is: $1 / (14/5) = 0.357 \text{ ms}$

Calling convention:

ANSI C :

```
BYTE      b_ReturnValue;
DWORD     dw_DriverHandle,
str_TransducerConvertTimeDivisionFactorInformation s_ModuleInformation;
b_ReturnValue =
b_ADDIDATA_GetTransducerModuleConvertTimeDivisionFactorInformation
(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_InitTransducerModuleConvertTimeDivisionFactor (...)**Syntax:**

```

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

```

Parameters:**- Input:**

DWORD dw_DriverHandle Driver handle

WORD w_Module Selection of the transducer module

DWORD dw_ConvertTimeDivisionFactor Selection of the converting time division factor

- Output:

No output has occurred.

Task:

Sets the converting time division.

The converting time division factor sets the switching time from one channel to channel.

The base time is the transducer frequency.

For example if the transducer connected to channel 0 uses the 14 kHz frequency and the division factor is set to 5 (dw_ConvertTimeDivisionFactor = 5) then the switching time from channel 0 to the next is: $1 / (14/5) = 0.357\text{ms}$

Calling convention:ANSI C:

```

BYTE      b_ReturnValue;
DWORD     dw_DriverHandle,

```

```

b_ReturnValue = b_ADDIDATA_InitTransducerModuleConvertTimeDivisionFactor
                                     (dw_DriverHandle,
                                     0,
                                     5);

```

Return value:

1: No error

0: Error by calling up the function. Use the function

"i_ADDIDATA_GetLastError", to find the error source.

11) b_ADDIDATA_ReleaseTransducerModuleConvertTimeDivisionFactor (...)**Syntax:**

<Return value> =

```

b_ADDIDATA_ReleaseTransducerModuleConvertTimeDivisionFactor
                                (DWORD    dw_DriverHandle,
                                WORD      w_Module)

```

Parameters:**- Input:**

DWORD dw_DriverHandle	Driver handle
WORD w_Module	Transducer module selection

- Output:

No output has occurred.

Task:

Releases the converting time division factor. Then any other process can set a new converting time division factor for this transducer module.

Calling convention:ANSI C :

```

BYTE    b_ReturnValue;
DWORD   dw_DriverHandle,

```

```

b_ReturnValue = b_ADDIDATA_ReleaseTransducerModuleConvertTimeDivisionFactor
                (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.

2.2 Transducer initialisation

1) b_ADDIDATA_InitTransducerChannel (..)

Syntax:

```
<Return value> = b_ADDIDATA_InitTransducerChannel
                    (DWORD dw_DriverHandle,
                     WORD  w_ChannelNbr,
                     WORD  w_TransducerIndex,
                     DOUBLE d_Frequency)
```

Parameters:

- Input:

DWORD <i>dw_DriverHandle</i>	Driver handle
WORD <i>w_ChannelNbr</i>	Number of the transducer channel selected
WORD <i>w_TransducerIndex</i>	Selection of the transducer type
DOUBLE <i>d_Frequency</i>	Frequency selection for the primary circuit (Hz)

- Output:

No output signal has occurred.

Task:

Initialises the selected transducer channel (*w_ChannelNbr*).

w_TransducerIndex determines the transducer type connected to the input. Refer to the function "b_ADDIDATA_GetTransducerModuleGenerationInformation" for more details.

Calling convention:

ANSI C:

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

```
b_ReturnValue = b_ADDIDATA_InitTransducerChannel (dw_DriverHandle,
                                                    0,
                                                    0,
                                                    2500);
```

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_ReleaseTransducerChannel (..)

Syntax:

```
<Return value> = b_ADDIDATA_ReleaseTransducerChannel
                    (DWORD   dw_DriverHandle,
                     WORD     w_ChannelNb)
```

Parameters:

- Input:

DWORD <i>dw_DriverHandle</i>	Driver handle
WORD <i>w_ChannelNbr</i>	Number of the transducer channel selected

- Output:

No output signal has occurred.

Task:

Releases the selected transducer channel (*w_ChannelNbr*).

Calling convention:

ANSI C :

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

```
b_ReturnValue = b_ADDIDATA_ReleaseTransducerChannel (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.

3) b_ADDIDATA_GetTransducerModuleConvertTimeDivisionFactorInformation (...)

Syntax:

```

<Return value> =
b_ADDIDATA_GetTransducerModuleConvertTimeDivisionFactorInformation
    (DWORD
        dw_DriverHandle,
        WORD
        w_Module,
        pstr_TransducerConvertTimeDivisionFactorInformation ps_ModuleInformation,
        DWORD
        dw_StructSize)

```

Parameters:

- Input:

```

        DWORD    dw_DriverHandle    :
Driver handle
WORD    w_Module    : Transducer module selection
        DWORD    dw_StructSize:
ps_ModuleInformation structure size

```

- Output:

```

pstr_TransducerConvertTimeDivisionFactorInformation ps_ModuleInformation:
Convert time division factor
information structure
Structure declaration :

```

The structure contains :

```

        BYTE    b_Configurable    0 : Convert time division factor fix
        1 : Convert time division factor configurable. See
        function
        "b_ADDIDATA_InitTransducerModuleConvertTimeDivisionFactor"

        BYTE    b_Steps    Return the convert time division factor
        steps.
        BYTE    b_Initialised    0 : Convert time division factor not
        initialised
        1 : Convert time division factor initialised

        DWORD    dw_InitialisationValue    If initialised, this parameter return the
        initialisation value
        DWORD    dw_MinDivisionFactor    Return the minimal division factor value
        DWORD    dw_MaxDivisionFactor    Return the maximal division factor value

```

Task:

Get the convert time division factor information's .

The convert time division factor set the switch time from channel to channel.

The base time is the transducer frequency.

For sample if your transducer connected to channel 0 use the 14KHz frequency and the division factor is setting to 5 (dw_InitialisationValue = 5) then the switch time from channel 0 to the next is : $1 (14 / 5) = 0.357\text{ms}$

Calling convention:**ANSI C :**

```
BYTE          b_ReturnValue;
DWORD dw_DriverHandle,
str_TransducerConvertTimeDivisionFactorInformation s_ModuleInformation;

b_ReturnValue =
b_ADDIDATA_GetTransducerModuleConvertTimeDivisionFactorInformation
(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.

4) b_ADDIDATA_ReleaseTransducerModuleConvertTimeDivisionFactor (...)

Syntax:

```
<Return value> =
b_ADDIDATA_ReleaseTransducerModuleConv
ertTimeDivisionFactor
        (DWORD
        dw_DriverHandle,
WORD     w_Module)
```

Parameters:

- Input:

```
DWORD     dw_DriverHandle
: Driver handle

WORD     w_Module
: Transducer module selection
```

- Output:

-

Task:

Release the convert time division factor.

After this any other process can set a new convert time division factor for this transducer module

Calling convention:

ANSI C:

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

```
        b_ReturnValue =
b_ADDIDATA_ReleaseTransducerModuleConvertTimeDivisionFactor
        (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_Read1TransducerChannel (..)

Syntax:

```
<Return value> = b_ADDIDATA_Read1TransducerChannel
                                     (DWORD    dw_DriverHandle,
                                     WORD      w_ChannelNbr,
                                     BYTE      b_InterruptFlag,
                                     PDWORD    pdw_ChannelValue)
```

Parameters

- Input:

DWORD <i>dw_DriverHandle</i>	Driver handle
DWORD <i>w_ChannelNbr</i>	Number of the transducer channel to be read
BYTE <i>b_InterruptFlag</i>	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 software description "Interrupt"

- Output:

PDWORD <i>pdw_ChannelValue</i>	Returns the digital value of the selected channel
--------------------------------	---

Task:

Returns the digital value (*pdw_ChannelValue*) of the selected channel (*w_Channel*). To obtain the real metric value, you must call up the "b_ADDIDATA_ConvertDigitalToRealMetricValue (...)" function.

Calling convention:

ANSI C :

```
BYTE      b_ReturnValue;
DWORD     dw_DriverHandle;
DWORD     dw_ChannelValue [3];
```

```
b_ReturnValue = b_ADDIDATA_Read1TransducerChannel
                                     (dw_DriverHandle,
                                     0,
                                     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_ConvertDigitalToRealMetricValue (..)

Syntax:

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

Parameters:

-Input:

DWORD <i>dw_DriverHandle</i>	Driver handle
DWORD <i>w_Channel</i>	Number of the transducer to be converted
PDWORD <i>pdw_DigitalValue</i>	Digital value

-Output:

PDOUBLE <i>pd_MetricValue</i>	Returns the real metric value
-------------------------------	-------------------------------

Task:

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

Calling convention:

ANSI C :

```
BYTE    b_ReturnValue;
DWORD   dw_DriverHandle;
DWORD   dw_DigitalValue;
DOUBLE  d_MetricValue;
```

```
b_ReturnValue = b_ADDIDATA_ConvertDigitalToRealMetricValue
                                     (dw_DriverHandle,
                                     0,
                                     dw_DigitalValue,
                                     &d_MetricValue);
```

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_ReadMoreTransducerChannels (..)

Syntax:

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

Parameters:

- Input:

DWORD <i>dw_DriverHandle</i>	Driver handle
WORD <i>w_FirstChannel</i>	Selection of the first channel to be read
WORD <i>w_LastChannel</i>	Selection of the last channel to be read
BYTE <i>b_InterruptFlag</i>	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 software description "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 next channel ...

Task:

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

Calling convention:

ANSI C :

```
BYTE    b_ReturnValue;
DWORD   dw_DriverHandle;
DWORD   dw_ChannelArrayValue [16];
```

```
b_ReturnValue = b_ADDIDATA_ReadMoreTransducerChannels
                                (dw_DriverHandle,
                                 0,
                                 15,
                                 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.

4) b_ADDIDATA_ConvertMoreDigitalToRealMetricValues (..)**Syntax:**

```

<Return value> = b_ADDIDATA_ConvertMoreDigitalToRealMetricValues
                                (DWORD    dw_DriverHandle,
                                WORD      w_FirstChannel,
                                WORD      w_LastChannel,
                                PDWORD    pdw_DigitalValue,
                                PDOUBLE   pd_MetricValue)

```

Parameters:**- Input:**

DWORD <i>dw_DriverHandle</i>	Driver handle
WORD <i>w_FirstChannel</i>	Selection of the first channel to be read
WORD <i>w_LastChannel</i>	Selection of the last channel to be read
PDWORD <i>pdw_DigitalValue</i>	Digital value array

- Output:

PDOUBLE <i>pd_MetricValue</i>	Returns the real metric value array
-------------------------------	-------------------------------------

Task:

Converts the digital input value (*pdw_DigitalValue*) into a real metric value (*pd_MetricValue*) for the selected channels (*w_FirstChannel* to *w_LastChannel*).

Calling convention:ANSI C :

```

BYTE      b_ReturnValue;
DWORD     dw_DriverHandle;
DWORD     dw_DigitalValue [16];
DOUBLE    d_MetricValue  [16];

```

```

b_ReturnValue = b_ADDIDATA_ConvertMoreDigitalToRealMetricValues
                                (dw_DriverHandle,
                                0,
                                15,
                                dw_DigitalValue,
                                d_MetricValue);

```

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 mode

1) b_ADDIDATA_GetTransducerAutoRefreshChannelPointer (..)

Syntax:

```
<Return value> = b_ADDIDATA_GetTransducerAutoRefreshChannelPointer
                                (DWORD      dw_DriverHandle,
                                 WORD       w_ChannelNbr,
                                 VOID       **ppv_ApplicationLevelPointer,
                                 VOID       **ppv_KernelLevelPointer)
```

Parameters:

- Input:

DWORD dw_DriverHandle	Driver handle
DWORD w_ChannelNbr	Number of the transducer input channel for which the autorefresh pointer is requested

- Output:

VOID **ppv_ApplicationLevelPointer	Returns the pointer of the value for the selected channel in the application level (digital value)
VOID **ppv_KernelLevelPointer	Returns the pointer of the value for the selected channel in the kernel level (digital value).

Task:

Returns the pointers (*ppv_ApplicationLevelPointer* and *ppv_KernelLevelPointer*) of the value for the selected channel (*w_ChannelNbr*). To obtain the real metric value, you must call up the "b_ADDIDATA_ConvertDigitalToRealMetricValue (..)" function.

Calling convention:

ANSI C :

```
BYTE      b_ReturnValue;
DWORD     dw_DriverHandle;
DWORD     ** ppdw_ApplicationLevelPointer;
DWORD     ** ppdw_KernelLevelPointer;
```

```
b_ReturnValue = b_ADDIDATA_GetTransducerAutoRefreshChannelPointer
                (dw_DriverHandle, 0,
                 (void **) &ppdw_ApplicationLevelPointer,
                 (void **) &ppdw_KernelLevelPointer);
```

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_GetTransducerAutoRefreshModulePointer (..)

Syntax:

```
<Return value> = b_ADDIDATA_GetTransducerAutoRefreshModulePointer
                                (DWORD      dw_DriverHandle,
                                 WORD       w_Module,
                                 VOID       **ppv_ApplicationLevelPointer,
                                 VOID       **ppv_KernelLevelPointer)
```

Parameters:

- Input:

DWORD dw_DriverHandle	Driver handle
DWORD w_Module	Number of the transducer module for which the autorefresh pointer is requested

- Output:

VOID	**ppv_ApplicationLevelPointer	Returns the pointer of the value for the selected module in the application level (digital value). ppv_ApplicationLevelPointer [0] = digital value of the first channel ppv_ApplicationLevelPointer [1] = digital value of the next channel
VOID	**ppv_KernelLevelPointer	Returns the pointer of the value for the selected module in the kernel level (digital value). ppv_KernelLevelPointer [0] = digital value of the first channel ppv_KernelLevelPointer [1] = digital value of the next channel

Task:

Returns the pointers (*ppv_ApplicationLevelPointer* and *ppv_KernelLevelPointer*) of the value for the selected module (*w_Module*). To obtain the real metric value, you must call up the "b_ADDIDATA_ConvertMoreDigitalToRealMetricValues (..)" function.

Calling convention:ANSI C :

```

BYTE      b_ReturnValue;
DWORD     dw_DriverHandle;
DWORD     ** ppdw_ApplicationLevelPointer;
DWORD     ** ppdw_KernelLevelPointer;

```

```

b_ReturnValue = b_ADDIDATA_GetTransducerAutoRefreshModulePointer
                (dw_DriverHandle, 0,
                (void **) &ppdw_ApplicationLevelPointer,
                (void **) &ppdw_KernelLevelPointer);

```

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_GetTransducerAutoRefreshModuleCounterPointer (..)**Syntax:**

```

<Return value> = b_ADDIDATA_GetTransducerAutoRefreshModuleCounterPointer
                (DWORD     dw_DriverHandle,
                WORD      w_Module,
                VOID      **ppv_ApplicationLevelPointer,
                VOID      **ppv_KernelLevelPointer)

```

Paramters:**- Input:**

DWORD dw_DriverHandle	Driver handle
DWORD w_Module	Number of the transducer module for which the autorefresh counter pointer is requested.

- Output:

VOID **ppv_ApplicationLevelPointer	Returns the pointer of the value for the selected module counter
------------------------------------	--

VOID **ppv_KernelLevelPointer	Returns the pointer of the value for the selected module counter in the kernel level
-------------------------------	--

Task:

Returns the pointers (*ppv_ApplicationLevelPointer* and *ppv_KernelLevelPointer*) of the counter for the selected module (*w_Module*).

After each sequence acquisition the counter increments which enables to test the end of a sequence.

Calling convention:ANSI C :

```
BYTE      b_ReturnValue;  
DWORD     dw_DriverHandle;  
DWORD     ** ppdw_ApplicationLevelPointer;  
DWORD     ** ppdw_KernelLevelPointer;
```

```
b_ReturnValue = b_ADDIDATA_GetTransducerAutoRefreshModuleCounterPointer  
                (dw_DriverHandle, 0,  
                 (void **) &ppdw_ApplicationLevelPointer,  
                 (void **) &ppdw_KernelLevelPointer);
```

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_StartTransducerAutoRefresh (..)

Syntax:

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

Parameters:
- Input:

DWORD dw_DriverHandle	Driver handle
DWORD w_Module	Number of the transducer module for which the autorefresh acquisition is started

- Output:

No output signal has occurred.

Task:

Starts the selected (*w_Module*) auto-acquisition transducer module.


IMPORTANT!

Do initialise all channels of the module before running an acquisition in the auto refresh mode.

Calling convention:
ANSI C :

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

```
b_ReturnValue = b_ADDIDATA_StartTransducerAutoRefresh
                (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.

5) b_ADDIDATA_StopTransducerAutoRefresh (..)

Syntax:

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

Parameters:**- Input:**

DWORD dw_DriverHandle Driver handle
DWORD w_Module Number of the transducer module for which
the autorefresh acquisition is stopped

- Output:

No output signal has occurred.

Task:

Stops the selected (*w_Module*) auto-acquisition transducer module.

Calling convention:ANSI C:

BYTE b_ReturnValue;
DWORD dw_DriverHandle;

b_ReturnValue = b_ADDIDATA_StartTransducerAutoRefresh
(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 Auto Refresh Acquisition

1) b_ADDIDATA_GetTransducerAutoRefreshChannelPointer (..)

Syntax:

```
<Return value> = b_ADDIDATA_GetTransducerAutoRefreshChannelPointer
                (DWORD      dw_DriverHandle,
                 WORD       w_ChannelNbr,
                 VOID      **ppv_ApplicationLevelPointer,
                 VOID      **ppv_KernelLevelPointer)
```

Input:

DWORD	dw_DriverHandle	Driver handle
DWORD	w_ChannelNbr	Number of the transducer input channel to get the auto refresh pointer

Output:

VOID	**ppv_ApplicationLevelPointer	Return the pointer of the value from the selected channel in the application level (Digital value)
VOID	**ppv_KernelLevelPointer	Return the pointer of the value from the selected channel in the kernel level (Digital value).

Task:

Return the pointers (*ppv_ApplicationLevelPointer* and *ppv_KernelLevelPointer*) of the value from the selected channel (*w_ChannelNbr*).

To obtain the real metric value, you must call up the

"**b_ADDIDATA_ConvertDigitalToRealMetricValue (...)**" function.

Calling convention:

ANSI C :

```
BYTE      b_ReturnValue;
DWORD     dw_DriverHandle;
DWORD     ** ppdw_ApplicationLevelPointer;
DWORD     ** ppdw_KernelLevelPointer;
```

```
b_ReturnValue = b_ADDIDATA_GetTransducerAutoRefreshChannelPointer
                (dw_DriverHandle, 0,
                 (void **) &ppdw_ApplicationLevelPointer,
                 (void **) &ppdw_KernelLevelPointer);
```

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_GetTransducerAutoRefreshModulePointer (..)

Syntax:

```
<Return value> = b_ADDIDATA_GetTransducerAutoRefreshModulePointer
                (DWORD      dw_DriverHandle,
                 WORD       w_Module,
                 VOID       **ppv_ApplicationLevelPointer,
                 VOID       **ppv_KernelLevelPointer)
```

Input:

DWORD	dw_DriverHandle	Driver handle
DWORD	w_Module	Number of the transducer module to get the auto refresh pointer

Output:

VOID	**ppv_ApplicationLevelPointer	Return the pointer of the value from the selected module in the application level (Digital value). ppv_ApplicationLevelPointer [0] = digital value of the first channel ppv_ApplicationLevelPointer [1] = digital value of the next channel
VOID	**ppv_KernelLevelPointer	Return the pointer of the value from the selected module in the kernel level (Digital value). ppv_KernelLevelPointer [0] = digital value of the first channel ppv_KernelLevelPointer [1] = digital value of the next channel

Task:

Return the pointers (*ppv_ApplicationLevelPointer* and *ppv_KernelLevelPointer*) of the value from the selected module (*w_Module*).

To obtain the real metric value, you must call up the

"**b_ADDIDATA_ConvertMoreDigitalToRealMetricValues (...)**" function.

Calling convention:

ANSI C :

```
BYTE          b_ReturnValue;
DWORD         dw_DriverHandle;
DWORD         ** ppdw_ApplicationLevelPointer;
DWORD         ** ppdw_KernelLevelPointer;
```

```
b_ReturnValue = b_ADDIDATA_GetTransducerAutoRefreshModulePointer
(dw_DriverHandle, 0,
(void **) &ppdw_ApplicationLevelPointer,
(void **) &ppdw_KernelLevelPointer);
```

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_GetTransducerAutoRefreshModuleCounterPointer (..)

Syntax:

<Return value> =

```
b_ADDIDATA_GetTransducerAutoRefreshModuleCounterPointer
        (DWORD      dw_DriverHandle,
         WORD       w_Module,
         VOID       **ppv_ApplicationLevelPointer,
         VOID       **ppv_KernelLevelPointer)
```

Input:

DWORD	dw_DriverHandle	Driver handle
DWORD	w_Module	Number of the transducer module to get the auto refresh counter pointer

Output:

VOID	**ppv_ApplicationLevelPointer	Return the pointer of the value from the selected module counter
------	-------------------------------	---

VOID	**ppv_KernelLevelPointer	Return the pointer of the value from the selected module counter in the kernel level
------	--------------------------	--

Task:

Return the pointers (*ppv_ApplicationLevelPointer* and *ppv_KernelLevelPointer*) of the counter from the selected module (*w_Module*).

After each sequence acquisition the counter is increment. This give you the possibility to test the end of a sequence.

Calling convention:

ANSI C :

```
BYTE      b_ReturnValue;
DWORD     dw_DriverHandle;
DWORD     ** ppdw_ApplicationLevelPointer;
DWORD     ** ppdw_KernelLevelPointer;
```

b_ReturnValue =

```
b_ADDIDATA_GetTransducerAutoRefreshModuleCounterPointer
        (dw_DriverHandle, 0,
         (void **) &ppdw_ApplicationLevelPointer,
         (void **) &ppdw_KernelLevelPointer);
```

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_StartTransducerAutoRefresh (..)

Syntax:

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

Input:

DWORD	dw_DriverHandle	Driver handle
DWORD	w_Module	Number of the transducer module to start the auto refresh acquisition

Output:

-

Task:

Start the selected (*w_Module*) transducer auto acquisition module .

Calling convention:

ANSI C :

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

```
b_ReturnValue = b_ADDIDATA_StartTransducerAutoRefresh
                (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.

5) **b_ADDIDATA_StopTransducerAutoRefresh (..)****Syntax:**

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

Input:

DWORD	dw_DriverHandle	Driver handle
DWORD	w_Module	Number of the transducer module to stop the auto refresh acquisition

Output:

-

Task:

Stop the selected (*w_Module*) transducer auto acquisition module .

Calling convention:

ANSI C :

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

```
b_ReturnValue = b_ADDIDATA_StartTransducerAutoRefresh
                (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.

6) b_ADDIDATA_Read1TransducerAutorefreshValue(..)

Syntax:

```
<Return value> = b_ADDIDATA_Read1TransducerAutoRefreshValue
                (DWORD      dw_DriverHandle,
                 WORD       w_Channel
                 PDWORD     pdw_ChannelValue)
```

Input:

DWORD	dw_DriverHandle	Driver handle
WORD	w_Channel	Number of the transducer to be read

Output:

PDWORD	pdw_ChannelValue	Returns the value of the selected channel
--------	------------------	---

Task:

Returns the value (*pdw_ChannelValue*) of the selected channel (*w_Channel*). To obtain the real transducer autorefresh value, you must call up the "**b_ADDIDATA_ConvertDigitalToRealMetricValue**" function.

Calling convention:

ANSI C:

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

```
b_ReturnValue = b_ADDIDATA_Read1TransducerAutoRefreshValue
                (dw_DriverHandle,
                0, & 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.

7) b_ADDIDATA_ReadTransducerAutoRefreshCounterValue**Syntax:**

```
<Return value> = b_ADDIDATA_ReadTransducerAutoRefreshCounterValue
                (DWORD      dw_DriverHandle,
                 WORD       w_Module,
                 PDWORD     pdw_CounterValue)
```

Input:

DWORD	dw_DriverHandle	Driver handle
WORD	w_Module	Transducer module

Output:

PDWORD	pdw_CounterValue	Returns the counter value of the selected module
--------	------------------	--

Task:

Returns the value (*pdw_CounterValue*) of the selected module (*w_module*).

Calling convention:

ANSI C :

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

```
b_ReturnValue = b_ADDIDATA_ReadTransducerAutoRefreshCounterValue
(dw_DriverHandle,
                                0,
                                dw_CounterValue);
```

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_SetTransducerAutorefreshAverageMode(..)

Syntax:

```
<Return value> = b_ADDIDATA_SetupTransducerAutoRefreshAverageMode
                                (DWORD      dw_DriverHandle,
                                WORD        w_Module
                                BYTE        b_AverageMode);
```

Parameters:

- Input:

DWORD	dw_DriverHandle	Driver handle
DWORD	w_Module	Number of the transducer module for which the average mode is defined
BYTE	b_AverageMode	<p>AverageMode = 0 The standard average mode (per sequence): The sequence is converted x times to do the average. The user receives the values at the end of all conversion.</p> <p>AverageMode = 1 Average mode per channel: This channel will be converted x times to do the average. This new mode permits to receive the value of each channel early.</p>

- Output:

No output signal has occurred.

Task:

Defines the average mode. If this function is not called, then the standard mode is used (b_AverageMode = 0)

Calling convention:

ANSI C :

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

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

Return Value:

1: No error

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

2.6 Sequence acquisition

1) b_ADDIDATA_InitTransducerSequenceAcquisition (..)



IMPORTANT!

This functionality can only be operated on one module.

Syntax:

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

Parameters:

- Input:

DWORD *dw_DriverHandle* Driver handle
 DWORD *dw_NbrOfChannel* Number of channel in the sequence.
 PWORD *pw_SequenceChannelArray* Index array of the channel to be converted.

str_InitAnalogMeasureSequenceAcquisition ps_InitParam

Typedef struct

```
{
  BYTE    b_ConvertingTimeUnit;    Not used
  BYTE    b_DelayTimeMode;         Delay mode. Refer to Table 2-1.
  BYTE    b_DelayTimeUnit;         Delay time unit. Refer to Table 2-1 and Table 2-2
  DWORD   dw_DelayTime;            Delay time value. Refer to Table 2-1 and
                                   Table 2-2

  BYTE    b_Reserved[5];
  DWORD   dw_ConvertingTime;       Not used
  DWORD   dw_SequenceCounter;      Number of sequences. Refer to Table 2-3
  DWORD   dw_InterruptSequenceCounter;
                                   Number of sequences before an interrupt is
                                   generated.
}
```

} str_InitAnalogMeasureSequenceAcquisition;

DWORD *dw_StructSize* Size of the structure.

- Output:

PDWORD *pdw_SEQHandle* Sequence handle.

Task:

Initialises the transducer acquisition sequence.

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

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

Table 2-1: Delay mode

b_DealyTimeMode	Mode description
ADDIDATA_DELAY_NOT_USED	The delay between two SCANs is not used
ADDIDATA_DELAY_MODE1_USED	The delay between two SCANs 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 2-2: Selection of the converting time

<i>b_DelayTimeUnit</i>	Unit selection	<i>dw_DelayTime</i>	Delay 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

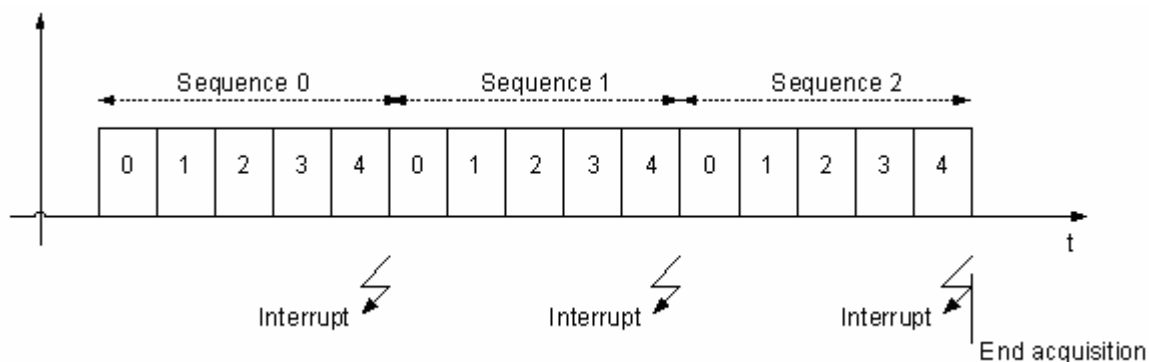
Table 2-3: Sequence mode

<i>dw_SequenceCounter</i>	Sequence description
> 0	A predefined number of sequences are started after calling up "b_ADDIDATA_StartTransducerSequenceAcquisition"
0	The number of sequences started after calling up the function "b_ADDIDATA_StartTransducerSequenceAcquisition" is not defined.

Samples

Sample 1 :

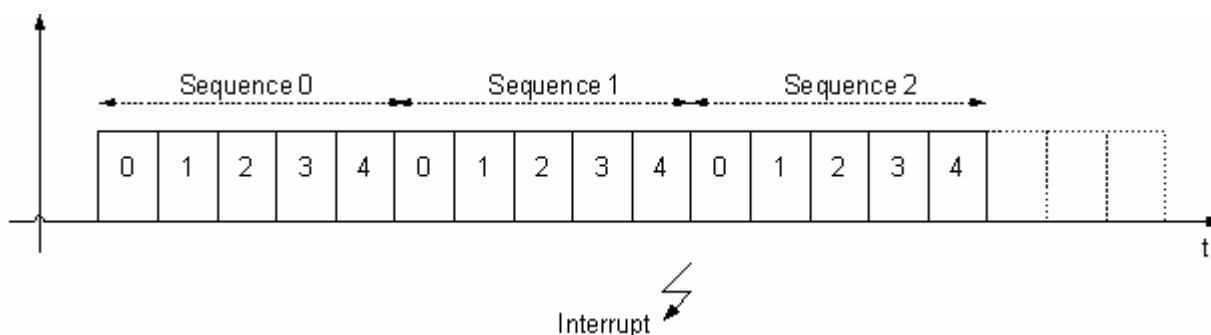
dw_NbrOfChannel = 5
 dw_SequenceChannelArray = 0,1,2,3,4
 b_DelayTimeMode = ADDIDATA_DELAY_NOT_USED
 dw_SequenceCounter = 3
 dw_InterruptSequenceCounter = 1



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

Sample 2 :

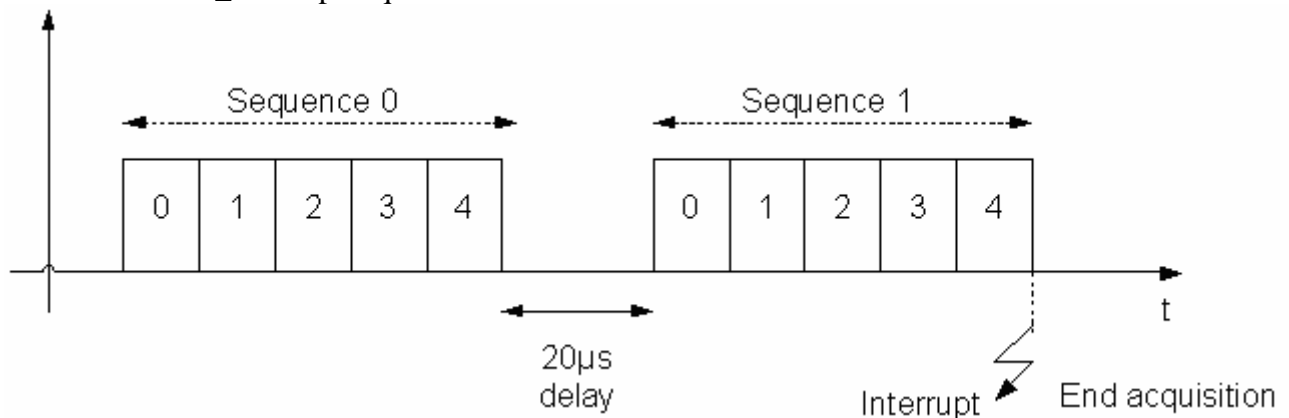
dw_NbrOfChannel = 5
 dw_SequenceChannelArray = 0,1,2,3,4
 b_DelayTimeMode = ADDIDATA_DELAY_NOT_USED
 dw_SequenceCounter = 0
 dw_InterruptSequenceCounter = 2



Interrupt occurs after 2 ends of sequence (10 acquisitions) and the acquisition is stopped by the function b_ADDIDATA_StopTransducerSequenceAcquisition.

Sample 3 :

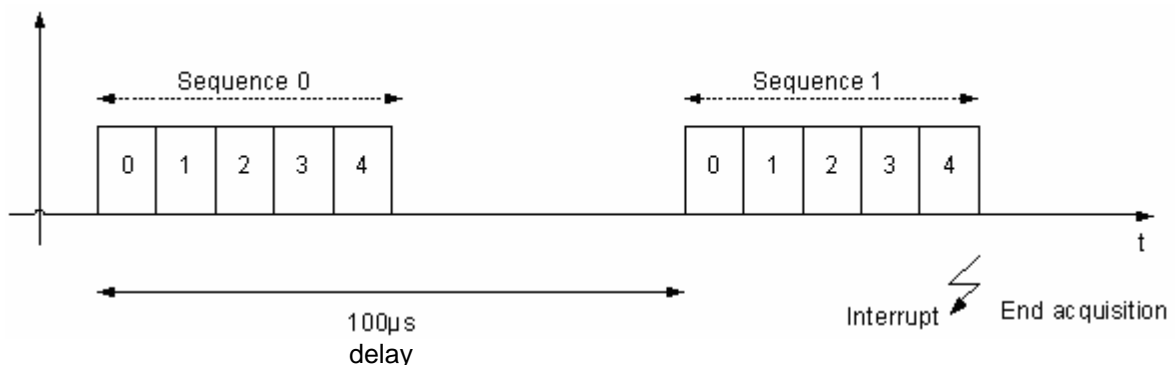
dw_NbrOfChannel = 5
 dw_SequenceChannelArray = 0,1,2,3,4
 b_DelayTimeMode = ADDIDATA_DELAY_MODE1_USED
 b_DelayTimeUnit; = 1 (μ s)
 dw_DelayTime = 20
 dw_SequenceCounter = 2
 dw_InterruptSequenceCounter = 2



Interrupt occurs after 2 ends of sequences (10 acquisitions) and the acquisition is stopped after 2 sequences. A delay of 20 μ s is respected between 2 sequences.

Sample 4 :

dw_NbrOfChannel = 5
 dw_SequenceChannelArray = 0,1,2,3,4
 b_DelayTimeMode = ADDIDATA_DELAY_MODE2_USED
 b_DelayTimeUnit; = 1 (μ s)
 dw_DelayTime = 100
 dw_SequenceCounter = 2
 dw_InterruptSequenceCounter = 2



Interrupt occurs after 2 ends of sequences (10 acquisitions) and the acquisition is stopped after 2 sequences. The sequence total time is 100 μ s.

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_InitTransducerSequenceAcquisition  
                (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_StartTransducerSequenceAcquisition (..)

Syntax:

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

Parameters:

- Input:

DWORD <i>dw_DriverHandle</i>	Driver handle
DWORD <i>dw_SEQHandle</i>	Sequence handle

- Output:

No output signal has occurred.

Task:

Starts the acquisition in sequence mode.

Calling convention:

ANSI C :

```

BYTE      b_ReturnValue;
DWORD     dw_DriverHandle;
DWORD     dw_SEQHandle;

```

```
b_ReturnValue = b_ADDIDATA_StartTransducerSequenceAcquisition
                                     (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_PauseTransducerSequenceAcquisition (..)

Syntax:

```
<Return value> = b_ADDIDATA_PauseTransducerSequenceAcquisition
                    (DWORD      dw_DriverHandle,
                     DWORD      dw_SEQHandle)
```

Parameters:

- Input:

DWORD <i>dw_DriverHandle</i>	Driver handle
DWORD <i>dw_SEQHandle</i>	Sequence acquisition handle

- Output:

No output signal has occurred.

Task:

Pauses the sequence acquisition. The sequence can be restarted by the function "b_ADDIDATA_StartTransducerSequenceAcquisition"

Calling convention:

ANSI C :

BYTE	b_ReturnValue;
DWORD	dw_DriverHandle;
DWORD	dw_SEQHandle;

```
b_ReturnValue = b_ADDIDATA_PauseTransducerSequenceAcquisition(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.

4) b_ADDIDATA_ConvertTransducerSequenceDigitalToRealMetricValue (..)

Syntax:

```
<Return value> = b_ADDIDATA_ConvertTransducerSequenceDigitalToRealMetricValue
                (DWORD      dw_DriverHandle,
                 DWORD      dw_SEQHandle,
                 PDWORD     pdw_DigitalValue,
                 PDOUBLE    pd_MetricValue)
```

Parameters:

- Input:

DWORD <i>dw_DriverHandle</i>	Driver handle
DWORD <i>dw_SEQHandle</i>	Sequence acquisition handle
PDWORD <i>pdw_DigitalValue</i>	Digital value array

- Output:

PDOUBLE <i>pd_MetricValue</i>	Returns the real metric value array
-------------------------------	-------------------------------------

Task:

Converts the sequence digital input value array (*pdw_DigitalValue*) into a real metric value (*pd_MetricValue*) for the selected sequence (*dw_SEQHandle*).

Calling convention:

ANSI C :

```

BYTE      b_ReturnValue;
DWORD     dw_DriverHandle;
DWORD     dw_DigitalValue [16];
DOUBLE    d_MetricValue   [16];
DWORD     dw_SEQHandle;

```

```
b_ReturnValue = b_ADDIDATA_ConvertTransducerSequenceDigitalToRealMetricValue
(dw_DriverHandle,
dw_SEQHandle,
dw_DigitalValue,
d_MetricValue);
```

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_StopTransducerSequenceAcquisition (..)

Syntax:

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

Parameters:**- Input:**

DWORD *dw_DriverHandle* Driver handle
DWORD *dw_SEQHandle* Sequence handle

- Output:

No output signal has occurred.

Task:

Stops the transducer acquisition sequence.

Calling convention:ANSI C :

BYTE b_ReturnValue;
DWORD dw_DriverHandle;
DWORD dw_SEQHandle;

b_ReturnValue = b_ADDIDATA_StopTransducerSequenceAcquisition
(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_GetTransducerSequenceAcquisitionHandleStatus (..)

Syntax:

<Return value> = b_ADDIDATA_GetTransducerSequenceAcquisitionHandleStatus
 (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
 DWORD *w_Module* Number of the transducer module for which the sequence handle status is returned

- Output:

PBYTE *pb_InitialisationStatus* 0: No sequence initialised
 >0: Number of initialised sequence
 PDWORD *pdw_LastInitialisedSEQHandle* Handle of the last initialised sequence
 PBYTE *pb_CurrentSEQStatus* Current sequence status
 0: No sequence started
 1: Sequence started
 2: Sequence halted
 3: Sequence ended
 PDWORD *pdw_CurrentSEQHandle* Current sequence handle

Task:

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

Calling convention:

ANSI C :

BYTE b_ReturnValue;
 BYTE b_InitialisationStatus;
 BYTE b_CurrentSEQStatus;
 DWORD dw_DriverHandle;
 DWORD dw_LastInitialisedSEQHandle;
 DWORD dw_CurrentSEQHandle;

b_ReturnValue = b_ADDIDATA_GetTransducerSequenceAcquisitionHandleStatus
 (dw_DriverHandle,
 0,
 &dw_SEQHandle;
 &b_InitialisationStatus,
 &dw_LastInitialisedSEQHandle,
 &b_CurrentSEQStatus,
 &dw_CurrentSEQHandle);

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_ReleaseTransducerSequenceAcquisition (..)**Syntax:**

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

Parameters:**- Input:**

DWORD *dw_DriverHandle* Driver handle
DWORD *dw_SEQHandle* Sequence handle

- Output:

No output signal has occurred.

Task:

Releases the transducer sequence acquisition.

Calling convention:ANSI C :

BYTE b_ReturnValue;
DWORD dw_DriverHandle;
DWORD dw_SEQHandle;

b_ReturnValue = b_ADDIDATA_ReleaseTransducerSequenceAcquisition
(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 Trigger/gate

1) b_ADDIDATA_GetTransducerHardwareTriggerInformation (...)

Syntax:

```
<Return value> = b_ADDIDATA_GetTransducerHardwareTriggerInformation
                                (DWORD    dw_DriverHandle,
                                 WORD      w_Module,
                                 PBYTE     pb_LowLevelTrigger,
                                 PBYTE     pb_HighLevelTrigger,
                                 PBYTE     pb_HardwareTriggerCount,
                                 PDWORD    pdw_MaxTriggerCountValue)
```

Parameters:

- Input:

DWORD	<i>dw_DriverHandle</i>	Driver handle
WORD	<i>w_Module</i>	Transducer module index

- Output:

PBYTE	<i>pb_LowLevelTrigger</i>	0: Low level hardware trigger not available 1: Low level hardware trigger available If available, the hardware trigger can trigger from "1" to "0"
PBYTE	<i>pb_HighLevelTrigger</i>	0: High level hardware trigger not available 1: High level hardware trigger available If available, the hardware trigger can trigger from "0" to "1"
PBYTE	<i>pb_HardwareTriggerCount</i>	0: Hardware trigger with counter not available 1: Hardware trigger with counter available The user defines the number of trigger events before the trigger action occurs.
PDWORD	<i>pdw_MaxTriggerCountValue</i>	If hardware counter available (<i>pb_HardwareTriggerCount</i> = 1) this variable returns the max. value for the trigger counter.

Task:

Returns the levels available for the hardware trigger (*pb_LowLevelTrigger* and *pb_HighLevelTrigger*), the available trigger counter (*pb_HardwareTriggerCount*) and the max counter value (*pdw_MaxTriggerCountValue*).

Calling convention:

ANSI C:

BYTE	b_ReturnValue;
DWORD	dw_DriverHandle;
BYTE	b_LowLevelTrigger;
BYTE	b_HighLevelTrigger;
BYTE	b_HardwareTriggerCount;
DWORD	dw_MaxTriggerCountValue;


```
b_ReturnValue = b_ADDIDATA_GetTransducerHardwareTriggerInformation  
                (dw_DriverHandle,  
                 0,  
                 &b_LowLevelTrigger,  
                 &b_HighLevelTrigger,  
                 &b_HardwareTriggerCount,  
                 &dw_MaxTriggerCountValue);
```

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_EnableDisableTransducerHardwareTrigger (...)

Syntax:

```
<Return value> = b_ADDIDATA_EnableDisableTransducerHardwareTrigger
(DWORD dw_DriverHandle,
WORD    w_Module,
BYTE    b_HardwareTriggerFlag,
BYTE    b_HardwareTriggerLevel,
BYTE    b_HardwareTriggerAction,
WORD    dw_HardwareTriggerCount,
DWORD dw_TimeOut)
```

Parameters:

- Input:

DWORD	<i>dw_DriverHandle</i>	Driver handle
WORD	<i>w_Module</i>	Index of the module
BYTE	<i>b_HardwareTriggerFlag</i>	ADDIDATA_ENABLE: Enables the hardware trigger. ADDIDATA_DISABLE: Hardware trigger disabled by triggering the transducer module
BYTE	<i>b_HardwareTriggerLevel</i>	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_LOW_HIGH: If the hardware trigger is used, it triggers from "0" to "1" and from "1" to "0"
BYTE	<i>b_HardwareTriggerAction</i>	Selection of the trigger action. See Table 2-4
DWORD	<i>dw_HardwareTriggerCount</i>	Hardware trigger counter. Defines the number of trigger events before the action occurs (> 0)
DWORD	<i>dw_TimeOut</i>	Specifies the hardware trigger timeout interval, in milliseconds. The function returns if the interval elapses or the hardware trigger occur. If <i>dw_TimeOut</i> is ADDIDATA_INFINITE, the function's timeout interval never elapses. This parameter is only used for the single acquisition functions "b_ADDIDATA_Read1TransducerChannel" and "b_ADDIDATA_ReadMoreTransducerChannels"

- Output:

No output signal has occurred.

Task:

Releases or blocks the action of the hardware trigger

Table 2-4: Action of the hardware/software trigger

<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_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_InterruptSequenceCounter</i> sequences is started. Refer to function "b_ADDIDATA_InitTransducerSequenceAcquisition"
ADDIDATA_ONE_SHOT_TRIGGER	After the first <i>dw_HardwareTriggerCount</i> trigger the conversion is started. The next triggers have not effect. The triggers are reactivated after the next callup from the function "b_ADDIDATA_StartTransducerSequenceAcquisition"

Calling convention:ANSI C:

BYTE b_ReturnValue;
 DWORD dw_DriverHandle;

```
b_ReturnValue =    b_ADDIDATA_EnableDisableTransducerHardwareTrigger
                   (dw_DriverHandle,
                    0,
                    ADDIDATA_ENABLE,
                    ADDIDATA_HIGH,
                    ADDIDATA_TRIGGER_START_A_SINGLE_
                                         CONVERSION,
                    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_GetTransducerHardwareTriggerStatus (...)

Syntax:

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

Parameters:
- Input

DWORD *dw_DriverHandle* Driver handle
WORD *w_Module* Index of the transducer 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 pulse missing until 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 input status (active or not) and the number of pulse missing till the next trigger.

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_GetTransducerHardwareTriggerStatus
(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.

4) b_ADDIDATA_GetTransducerHardwareTriggerInformationEx(..)

Syntax:

```
<Return value> = b_GetTransducerHardwareTriggerInformationEx
                                (DWORD      dw_DriverHandle,
                                WORD        w_Module
                                pstr_TransducerHardwareTriggerInformation
                                ps_HardwareTriggerInformation
                                DWORD      dw_StructSize)
```

Parameters:
- Input:

DWORD	dw_DriverHandle	Driver handle
DWORD	w_Module	Transducer module
DWORD	dw_StructSize	Size of the structure entered in the pointer

- Output:

pstr_TransducerHardware Trigger Information	Trigger module information
ps_HardwareTriggerInformation	

Task:

Returns the levels available for the hardware trigger (*pb_LowLevelTrigger* and *pb_HighLevelTrigger*), the available trigger counter (*pb_HardwareTriggerCount*) and the max counter value (*pdw_MaxTriggerCountValue*).

Structure returned:

```
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;
                                      XXX1: One shot trigger available
                                      XX1X: Single autorefresh trigger available
                                      X1XX: X autorefresh 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;

    BYTE  b_Reserverd2 [4];
```

```
}str_TransducerHardwareTriggerInformation,*pstr_TransducerHardwareTriggerInf  
ormation;
```

Calling convention:ANSI C:

BYTE *b_ReturnValue*;

DWORD dw_DriverHandle;

str_TransducerHardwareTriggerInformation s_Information;

```
b_ReturnValue = b_ADDIDATA_ GetTransducerHardwareTriggerInformationEx  
                    (dw_DriverHandle,
```

```
0,
```

```
&s_Information,
```

```
sizeof (str_TransducerHardwareTriggerInformation));
```

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_EnableDisableTransducerHardwareTriggerEx

Syntax:

```
<Return value> = b_ADDIDATA_EnableDisableTransducerHardwareTriggerEx
(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	<i>dw_DriverHandle</i>	Driver handle
WORD	<i>w_Module</i>	Index of the module
BYTE	<i>b_HardwareTriggerFlag</i>	ADDIDATA_ENABLE: Enables the hardware trigger. ADDIDATA_DISABLE: Hardware trigger disabled by triggering the transducer module
BYTE	<i>b_HardwareTriggerLevel</i>	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_LOW_HIGH: If the hardware trigger is used, it triggers from "0" to "1" and from "1" to "0"
BYTE	<i>b_HardwareTriggerAction</i>	Selection of the trigger action. See table below
DWORD	<i>b_HardwareTriggerCycleCount</i>	Defines the number of sequences, auto refresh cycles or SCAN cycles to trigger (Refer to table below)
DWORD	<i>dw_HardwareTriggerCount</i>	Hardware trigger counter. Defines the number of trigger events before the action occurs (> 0)
DWORD	<i>dw_TimeOut</i>	Specifies the hardware trigger timeout interval,

Table 2-5: Action of the hardware/software trigger

<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_StartTransducerSequenceAcquisition" or "b_ADDIDATA_StartTransducerAutoRefresh"
ADDIDATA_TRIGGER_START_A_SINGLE_AUTO_REFRESH	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_EnableDisableTransducerHardwareTriggerEx
                    (dw_DriverHandle,
                     0,
                     ADDIDATA_ENABLE,
                     ADDIDATA_HIGH,
                     ADDIDATA_TRIGGER_START_A_SINGLE_
                                     CONVERSION,
                     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.

6) b_ADDIDATA_EnableDisableTransducerSoftwareTrigger (...)

Syntax:

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

Parameters:

- Input:

DWORD	<i>dw_DriverHandle</i>	Driver handle
WORD	<i>w_Module</i>	Index of the module
BYTE	<i>b_SoftwareTriggerFlag</i>	ADDIDATA_ENABLE: Enables the software trigger. ADDIDATA_DISABLE: Software trigger disabled by triggering the transducer module
BYTE	<i>b_SoftwareTriggerAction</i>	Trigger action selection. Refer to table 12

- Output

No output signal has occurred.

Task:

Releases or blocks the action of the software trigger.

Calling convention:

ANSI C :

```

BYTE    b_ReturnValue;
DWORD   dw_DriverHandle;

```

```
b_ReturnValue = b_ADDIDATA_EnableDisableTransducerSoftwareTrigger
(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.

7) b_ADDIDATA_TransducerSoftwareTrigger (...)

Syntax:

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

Parameters:**- Input:**

DWORD *dw_DriverHandle* Driver handle
WORD *w_Module* Index of the module

- Output

No output signal has occurred.

Task:

Make a software trigger

Calling convention:ANSI C:

BYTE b_ReturnValue;
DWORD dw_DriverHandle;

b_ReturnValue = b_ADDIDATA_TransducerSoftwareTrigger
(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.

8) b_ADDIDATA_GetTransducerSoftwareTriggerStatus (...)

Syntax:

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

Parameters:

- Input:

DWORD *dw_DriverHandle* Driver handle
WORD *w_Module* Index of the transducer module

- Output:

PBYTE *pb_SoftwareTriggerFlag* ADDIDATA_ENABLE: Software trigger is enabled.
 ADDIDATA_DISABLE: Software trigger is disabled

PBYTE *pb_SoftwareTriggerStatus* 0: Software trigger did not occur
 1: Software trigger occurred

Task:

Returns the status (occur or not) from selected transducer software trigger.

Calling convention:

ANSI C:

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

```
b_ReturnValue = b_ADDIDATA_GetTransducerSoftwareTriggerStatus
                (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.

9) b_ADDIDATA_GetTransducerHardwareGateInformation (...)

Syntax:

```
<Return value> = b_ADDIDATA_GetTransducerHardwareGateInformation
                                     (DWORD    dw_DriverHandle,
                                     WORD      w_Module,
                                     PBYTE     pb_LowLevelGate,
                                     PBYTE     pb_HighLevelGate)
```

Parameters:

- Input:

DWORD	<i>dw_DriverHandle</i>	Driver handle
WORD	<i>w_Module</i>	Transducer module index

- Output:

PBYTE	<i>pb_LowLevelGate</i>	0: Low level hardware gate not available 1: Low level hardware gate available
PBYTE	<i>pb_HighLevelGate</i>	0: High level hardware gate not available 1: High level hardware gate available

Task:

Returns the levels available for the hardware trigger (*pb_LowLevelGate* and *pb_HighLevelGate*).

Calling convention:

ANSI C :

```
BYTE    b_ReturnValue;
DWORD   dw_DriverHandle;
BYTE    b_LowLevelGate;
BYTE    b_HighLevelGate;
```

```
b_ReturnValue = b_ADDIDATA_GetTransducerHardwareGateInformation
                (dw_DriverHandle,
                0,
                &b_LowLevelGate,
                &b_HighLevelGate);
```

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_EnableDisableTransducerHardwareGate (...)**

Syntax:

[illegible]

Parameters:

- Input:

DWORD	<i>dw_DriverHandle</i>	Driver handle
WORD	<i>w_Module</i>	Index of the transducer module
BYTE	<i>b_ExternGateFlag</i>	ADDIDATA_ENABLE: enables the hardware gate. ADDIDATA_DISABLE: Disables the hardware gate.
BYTE	<i>b_HardwareGateLevel</i>	ADDIDATA_LOW: If the hardware gate is enabled, it is active at "0" ADDIDATA_HIGH: If the hardware gate is enabled, it is active at "1"

- Output:

No output signal has occurred

Task:

Releases or blocks the action of the transducer hardware gate.

Calling convention:

ANSI C :

```

BYTE    b_ReturnValue;
DWORD   dw_DriverHandle;

```

```
b_ReturnValue = b_ADDIDATA_EnableDisableTransducerHardwareGate(
    dw_DriverHandle,
    0,
    ADDIDATA_ENABLE,
    ADDIDATA_HIGH);
```

Return Value:

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

11) **b_ADDIDATA_GetTransducerHardwareGateStatus (...)**

Syntax:

```
<Return Value> = b_ADDIDATA_GateTransducerHardwareGateStatus  
                (DWORD   dw_DriverHandle,  
                 WORD     w_Module,  
                 PBYTE    pb_HardwareGateFlag,  
                 PBYTE    pb_HardwareGateStatus)
```

Parameters:

- Input:

DWORD	<i>dw_DriverHandle</i>	Driver handle
WORD	<i>w_Module</i>	Index of the transducer module

- Output:

PBYTE	<i>pb_HardwareGateFlag</i>	ADDIDATA_ENABLE: The hardware gate is enabled. ADDIDATA_DISABLE: The hardware gate is disabled
PBYTE	<i>pb_HardwareGateStatus</i>	0: Hardware gate is not active (Low status) 1: Hardware gate is active (High status)

Task:

Returns the status of the transducer hardware gate (Active or not).

Calling convention:

ANSI C :

```

BYTE      b_ReturnValue;
DWORD     dw_DriverHandle;
BYTE      b_HardwareGateFlag;
BYTE      b_HardwareGateStatus;

```

```
b_ReturnValue = b_ADDIDATA_GateTransducerHardwareGateStatus  
                (dw_DriverHandle,  
                0,  
                &b_HardwareGateFlag,  
                &b_HardwareGateStatus);
```

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 Diagnostic

1) b_ADDIDATA_TestTransducerChannelSecondaryConnection (..)

Syntax:

```
<Return value> = b_ADDIDATA_TestTransducerChannelSecondaryConnection
                                (DWORD dw_DriverHandle,
                                WORD   w_Channel,
                                PBYTE  pb_ShortCircuit,
                                PBYTE  pb_Connection)
```

Parameters:

- Input:

DWORD <i>dw_DriverHandle</i>	Driver handle
WORD <i>w_Channel</i>	Number of the transducer channel to be tested

- Output:

PBYTE <i>pb_ShortCircuit</i>	Returns if a short circuit is present on the secondary circuit 0: No short-circuit 1: Short-circuit 2: Test not available
PBYTE <i>pb_Connection</i>	Returns if there is a connection problem on the channel 0: Channel connection OK 1: Channel connection error 2: Test not available

Task:

Test if the selected channel has a short circuit (secondary circuit) and if a line is not correctly connected.

Calling convention:

ANSI C :

```
BYTE    b_ReturnValue;
DWORD   dw_DriverHandle;
BYTE    b_ShortCircuit;
BYTE    b_Connection;
```

```
b_ReturnValue = b_ADDIDATA_TestTransducerChannelSecondaryConnection
                (dw_DriverHandle,
                0,
                &b_ShortCircuit,
                &b_Connection);
```

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_EnableDisableTransducerModulePrimaryConnectionTest (..)

Syntax:

```
<Return value> = b_ADDIDATA_EnableDisableTransducerModulePrimaryConnectionTest
                    (DWORD dw_DriverHandle,
                     WORD   w_Module,
                     BYTE    b_TestFlag)
```

Parameters:

- Input:

DWORD <i>dw_DriverHandle</i>	Driver handle
WORD <i>w_Module</i>	Number of the transducer module to be enable or disable the test
BYTE <i>b_TestFlag</i>	ADDIDATA_ENABLE: Enables a connection test on the primary circuit ADDIDATA_DISABLE: Disables the connection test on the primary circuit.

- Output:

No output signal has occurred.

Task:

Releases or blocks the primary short-circuit connection and primary connection test.

Calling convention:

ANSI C :

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

```
b_ReturnValue = b_ADDIDATA_EnableDisableTransducerModulePrimaryConnectionTest
                (dw_DriverHandle,
                 0,
                 ADDIDATA_ENABLE);
```

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_TestTransducerModulePrimaryConnection (..)

Syntax:

```
<Return value> = b_ADDIDATA_TestTransducerModulePrimaryConnection
                    (DWORD dw_DriverHandle,
                     WORD   w_Module,
                     PBYTE  pb_TestFlag,
                     PBYTE  pb_ShortCircuit,
                     PBYTE  pb_Connection)
```

Parameters:
- Input:

DWORD <i>dw_DriverHandle</i>	Driver handle
WORD <i>w_Module</i>	Number of the transducer module to be tested

- Output:

PBYTE <i>pb_TestFlag</i>	ADDIDATA_ENABLE: Primary connection test enabled. ADDIDATA_DISABLE: Primary connection test disabled
PBYTE <i>pb_ShortCircuit</i>	Returns if a short-circuit is present on the primary connection 0: No short-circuit 1: Short-circuit. One or more transducers connected to this module has caused a short-circuit
PBYTE <i>pb_Connection</i>	Returns if there is a connection problem on the channel: 0: Channel connection OK. 1: Channel connection error. One or more transducers are disconnected

Task:

Test if the one or more transducers connected to the selected module make a short circuit (primary connection) and if one or more transducers are disconnected

Calling convention:
ANSI C:

```
BYTE    b_ReturnValue;
DWORD   dw_DriverHandle;
BYTE    b_ShortCircuit;
BYTE    b_Connection;
BYTE    b_TestFlag;
```

```
b_ReturnValue = b_ADDIDATA_TestTransducerModulePrimaryConnection
                (dw_DriverHandle,
                 0,
                 &b_TestFlag,
                 &b_ShortCircuit,
                 &b_Connection);
```

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_EnableDisableTransducerModulePrimaryShortCircuitInterrupt (..)

Syntax:

<Return value> =

```
b_ADDIDATA_EnableDisableTransducerModulePrimaryShortCircuitInterrupt
                                (DWORD dw_DriverHandle,
                                WORD    w_Module,
                                BYTE    b_InterruptFlag)
```

Parameters:
- Input:

DWORD	<i>dw_DriverHandle</i>	Driver handle
WORD	<i>w_Module</i>	Number of the transducer module to be enabled the primary short circuit interrupt
BYTE	<i>b_InterruptFlag</i>	ADDIDATA_ENABLE: Enable the primary short circuit interrupt. ADDIDATA_DISABLE: Disable the primary short circuit interrupt.

- Output:

No output signal has occurred.

Task:

Releases or blocks the primary short-circuit interrupt

Calling convention:

ANSI C:

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

b_ReturnValue =

```
b_ADDIDATA_EnableDisableTransducerModulePrimaryShortCircuitInterrupt
                                (dw_DriverHandle,
                                0,
                                ADDIDATA_ENABLE);
```

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_RearmTransducerModulePrimaryShortCircuitConnectionTest (..)

Syntax:

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

Parameters:

- Input:

DWORD <i>dw_DriverHandle</i>	Driver handle
WORD <i>w_Module</i>	Number of the transducer module on which the short-circuit interrupt for the primary side is enabled.

- Output:

No output signal has occurred.

Task:

Rearms the primary short-circuit connection test.

Calling convention:

ANSI C:

```
BYTE    b_ReturnValue;
DWORD   dw_DriverHandle;
BYTE    b_ShortCircuit;
BYTE    b_Connection;
```

```
b_ReturnValue =
b_ADDIDATA_RearmTransducerModulePrimaryShortCircuitConnectionTest
(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.