



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

**Software description**

**ADDIDRIVER**

**Counter**

5<sup>th</sup> edition 07/2004



<b>1</b>	<b>INTRODUCTION .....</b>	<b>1</b>
<b>2</b>	<b>COUNTER FUNCTIONS .....</b>	<b>2</b>
1)	b_ADDIDATA_GetNumberOfCounters(...) .....	2
2)	b_ADDIDATA_GetCounterInformation (...) .....	3
3)	b_ADDIDATA_GetCounterInformationEx (...).....	5
4)	b_ADDIDATA_InitCounter (...) .....	7
5)	b_ADDIDATA_SetCounterDirection (...).....	8
6)	b_ADDIDATA_EnableDisableCounterInterrupt (...).....	9
7)	b_ADDIDATA_StartCounter (...) .....	10
8)	b_ADDIDATA_StartAllCounters (...).....	11
9)	b_ADDIDATA_ClearCounter (...) .....	12
10)	b_ADDIDATA_TriggerCounter (...) .....	13
11)	b_ADDIDATA_TriggerAllCounters (...) .....	14
12)	b_ADDIDATA_StopCounter (...).....	15
13)	b_ADDIDATA_StopAllCounters (...).....	16
14)	b_ADDIDATA_ReleaseCounter (...).....	17
15)	b_ADDIDATA_ReadCounterValue .....	18
16)	b_ADDIDATA_ReadCounterStatus (...) .....	19
17)	b_ADDIDATA_EnableDisableCounterHardwareGate (...) .....	21
18)	b_ADDIDATA_GetCounterHardwareGateStatus (...) .....	22
19)	b_ADDIDATA_EnableDisableCounterHardwareTrigger (...).....	23
20)	b_ADDIDATA_GetCounterHardwareTriggerStatus (...).....	24
21)	b_ADDIDATA_EnableDisableCounterHardwareOutput (...).....	25
22)	b_ADDIDATA_GetCounterHardwareOutputStatus (...).....	26
23)	b_ADDIDATA_TestCounterAsynchronousFIFOFull (...).....	27

## Tables

Table 1-1: Type Declaration for Windows 98/NT/2000/XP .....	1
Table 2-1: Resolution .....	4



# 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
<b>VOID</b>	void	void	pointer	any
<b>BYTE</b>	unsigned char	unsigned char	byte	integer
<b>INT</b>	int	int	integer	integer
<b>WORD</b>	unsigned short int	unsigned short int	long	long
<b>DWORD</b>	long	long	longint	long
<b>PBYTE</b>	unsigned char *	unsigned char *	var byte	integer
<b>PINT</b>	int *	int *	var integer	integer
<b>PWORD</b>	unsigned short int *	unsigned short int *	var long	long
<b>PCHAR</b>	char *	char *	var string	string
<b>PDWORD</b>	long *	long *	var longint	long
<b>DOUBLE</b>	double	double	double	double

## 2 COUNTER FUNCTIONS

## 1) b\_ADDIDATA\_GetNumberOfCounters(...)

### Syntax:

```
<Return value> = b_ADDIDATA_GetNumberOfCounters
                                (DWORD   dw_DriverHandle,
                                PBYTE    pb_CounterNumber)
```

### Parameters:

**- Input:**

DWORD	<i>dw_DriverHandle</i>	Handle of the ADDI-DATA driver
-------	------------------------	--------------------------------

**- Output:**

PBYTE	<i>pb_CounterNumber</i>	Number of counters
-------	-------------------------	--------------------

### Task:

Returns the number of counters available on the board.

### Calling convention:

ANSI C :

```

BYTE      b_ReturnValue;
DWORD     dw_DriverHandle;
BYTE      b_CounterNumber;

```

```
b_ReturnValue = b_ADDIDATA_GetNumberOfCounters
(dw_DriverHandle, &b_CounterNumber);
```

### 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\_GetCounterInformation (...)

### Syntax:

<Return value> = b\_ADDIDATA\_GetCounterInformation

(DWORD dw\_DriverHandle,  
 BYTE b\_CounterNumber,  
 PBYTE pb\_Resolution,  
 PBYTE pb\_InterruptAvailable,  
 PDWORD pdw\_InputLevelSelection,  
 PDWORD pdw\_HardwareGateAvailable,  
 PDWORD pdw\_HardwareTriggerAvailable,  
 PDWORD pdw\_OutputAvailable,  
 PDWORD pdw\_UpDownSelection)

### Parameters:

#### - Input:

DWORD *dw\_DriverHandle* Handle of the ADDI-DATA driver  
 BYTE *b\_CounterNumber* Number of the counter.  
 The first counter begins from 0.

#### - Output:

PBYTE *pb\_Resolution* Selection of the resolution for the counter.  
 See table 2-1.

PBYTE *pb\_InterruptAvailable* 0 : Counter cannot generate an interrupt  
 1 : Counter can generate an interrupt

PDWORD *pdw\_InputLevelSelection*  
 1 : Counter only counts low pulses  
 2 : Counter only counts high pulses  
 3 : Counter can count low and high pulses

PDWORD *pdw\_HardwareGateAvailable*  
 0: Hardware gate not available.  
 1: Hardware gate available.

PDWORD *pdw\_HardwareTriggerAvailable*  
 0: Hardware trigger not available.  
 1: Hardware trigger available.

PDWORD *pdw\_OutputAvailable* 0: Hardware output not available  
 1: Hardware output available

PDWORD *pdw\_UpDownSelection* 0: Up/Down selection not available and counter counts down  
 1: Up/Down selection not available and counter counts upwards  
 2: Up/Down selection available

**Task:**

Returns the hardware gate (*pdw\_HardwareGateAvailable*), the hardware trigger (*pdw\_HardwareTriggerAvailable*), the up/down selection (*pdw\_UpDownSelection*) and the resolution (*pb\_Resolution*) which can be used for the selected counter (*b\_CounterNumber*).

**Table 2-1: Resolution**

Value	Resolution
8	8-bit
12	12-bit
16	16-bit
24	24-bit
32	32-bit

**Calling convention:**ANSI C:

```

BYTE    b_ReturnValue;
DWORD   dw_DriverHandle;
BYTE    b_Resolution;
BYTE    b_InterruptAvailable;
DWORD   dw_InputLevelSelection;
DWORD   dw_HardwareGateAvailable;
DWORD   dw_HardwareTriggerAvailable;
DWORD   dw_OutputAvailable;
DWORD   dw_UpDownSelection;

```

```

b_ReturnValue = b_ADDIDATA_GetCounterInformation
                (dw_DriverHandle,
                 0,
                 &b_Resolution,
                 &b_InterruptAvailable,
                 &dw_InputLevelSelection,
                 &dw_HardwareGateAvailable,
                 &dw_HardwareTriggerAvailable,
                 &dw_OutputAvailable,
                 &dw_UpDownSelection);

```

**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\_GetCounterInformationEx (...)****Syntax:**

<Return value> = b\_ADDIDATA\_GetCounterInformationEx  
 (DWORD dw\_DriverHandle,  
 BYTE b\_CounterNumber,  
 pstr\_GetCounterInformation ps\_CounterInformation,  
 DWORD dw\_StructSize)

**Parameters:****- Input:**

DWORD *dw\_DriverHandle* Handle of the ADDI-DATA driver  
 BYTE *b\_CounterNumber* Counter number.  
 The first counter begins from 0.  
 DWORD *dw\_StructSize* Size of the structure.

**- Output:**

pstr\_GetCounterInformation ps\_CounterInformation Counter information.

**structure returned :**

```
typedef struct
{
```

BYTE *b\_Resolution*; Counter resolution. See table 2-3  
 BYTE *b\_InterruptAvailable*;

FALSE : Interrupt not available  
 TRUE : Interrupt available

WORD *w\_Reserved1*;  
 DWORD *dw\_InputLevelSelection*; Level selection of the counter

DWORD *dw\_HardwareGateAvailable*;  
 FALSE :Hardware gate not available  
 TRUE : Hardware gate available

DWORD *dw\_HardwareGateLowAvailable*;  
 FALSE :Hardware gate low not available  
 TRUE : Hardware gate low available

DWORD *dw\_HardwareGateHighAvailable*;  
 FALSE :Hardware gate high not available  
 TRUE : Hardware gate high available

DWORD *dw\_HardwareTriggerAvailable*;  
 FALSE :Hardware trigger not available  
 TRUE : Hardware trigger available

DWORD *dw\_HardwareTriggerLowAvailable*;  
 FALSE :Hardware trigger low not available  
 TRUE : Hardware trigger low available

DWORD *dw\_HardwareTriggerHighAvailable*;  
 FALSE :Hardware trigger high not available  
 TRUE : Hardware trigger high available

DWORD *dw\_HardwareOutputAvailable*;  
 FALSE :Hardware output not available  
 TRUE : Hardware output available

DWORD *dw\_HardwareOutputLowAvailable*;  
 FALSE :Hardware output low not available  
 TRUE : Hardware output low available

```

DWORD  dw_HardwareOutputHighAvailable;
                                FALSE :Hardware output high not available
                                TRUE  : Hardware output high available
DWORD  dw_UpDownSelection; Up/down counting selection of the counter

```

```

}str_GetCounterInformation,*pstr_GetCounterInformation;

```

**Task:**

Returns the time units (*pb\_CounterTimeUnit*), the time steps (*pw\_CounterTimeStep*) and the resolution (*pb\_Resolution*) which can be used for the selected counter (*b\_CounterNumber*)

**Calling convention:**

ANSI C:

```

BYTE      b_ReturnValue;
DWORD     dw_DriverHandle;
str_GetCounterInformation s_CounterInformation;

b_ReturnValue = b_ADDIDATA_GetCounterInformation
                                (dw_DriverHandle,
                                0,
                                &s_CounterInformation,
                                sizeof (str_GetCounterInformation));

```

**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\_InitCounter (...)

### Syntax:

<Return value> = b\_ADDIDATA\_InitCounter

```
(DWORD   dw_DriverHandle,
BYTE     b_CounterNumber,
BYTE     b_LevelSelection
DWORD    dw_ReloadValue)
```

### Parameters:

**- Input:**

DWORD *dw\_DriverHandle*

## Handle of the ADDI-DATA driver

BYTE      *b\_CounterNumber*

Number of the counter

The first counter begins from 0.

BYTE      *b\_LevelSelection*

ADDIDATA\_LOW : Counter counts low levels

ADDIDATA\_HIGH : Counter counts high levels

ADDIDATA\_LOW\_HIGH: Counter counts high and low levels

This parameter has no influence if the selection of the counter direction is not possible. See the function “b\_ADDIDATA\_GetCounterInformation”

DWORD *dw\_ReloadValue*

Reload or overflow value (depends from the used ADDI-DATA board) see

```
function "b_ADDIDATA_SetCounterDirection"
```

**- Output:**

No output signal has occurred.

### Task:

Initialises the counter.

### Calling convention:

## ANSI C :

```

BYTE    b_ReturnValue;

```

```
DWORD dw_DriverHandle;
```

b ReturnValue = b\_ADDIDATA InitCounter

```
(dw_DriverHandle,  
0,  
ADDIDATA_HIGH,  
1000);
```

### 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\_SetCounterDirection (...)

**i**

### IMPORTANT!

The function b\_ADDIDATA\_SetCounterDirection must be called up **before** calling up the function b\_ADDIDATA\_StartCounter.

The function b\_ADDIDATA\_SetCounterDirection generates a software trigger which loads the initialisation values necessary for the function b\_ADDIDATA\_StartCounter.

### Syntax:

```
<Return value> = b_ADDIDATA_SetCounterDirection
                                     (DWORD dw_DriverHandle,
                                     BYTE   b_CounterNumber,
                                     BYTE   b_DirectionSelection)
```

### Parameters:

#### - Input:

DWORD	<i>dw_DriverHandle</i>	Handle of the ADDI-DATA driver
BYTE	<i>b_CounterNumber</i>	Number of the counter The first counter begins from 0.
BYTE	<i>b_DirectionSelection</i>	ADDIDATA_UP: Selects the counter for up-counting ADDI_DATA_DOWN : Selects the counter for down-counting

### Output:

No output signal has occurred.

### Task:

Sets the counting direction (upward or downward)

### Calling convention:

#### ANSI C:

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

```
b_ReturnValue = b_ADDIDATA_SetCounterDirection
               (dw_DriverHandle,
               0,
               ADDIDATA_DOWN);
```

### 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\_EnableDisableCounterInterrupt (...)****Syntax:**

```
<Return value> = b_ADDIDATA_EnableDisableCounterInterrupt
                                     (DWORD dw_DriverHandle,
                                     BYTE    b_CounterNumber,
                                     BYTE    b_InterruptFlag)
```

**Parameters:****- Input:**

DWORD	<i>dw_DriverHandle</i>	Handle of the ADDI-DATA driver
BYTE	<i>b_CounterNumber</i>	Number of the counter
		The first counter begins from 0.
BYTE	<i>b_InterruptFlag</i>	ADDIDATA_ENABLE or ADDIDATA_DISABLE

**- Output:**

No output signal has occurred.

**Task:**

Enables or disables the IRQ for the counter process.

**Calling convention:**ANSI C:

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

```
b_ReturnValue = b_ADDIDATA_EnableDisableCounterInterrupt
                (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.

**i****IMPORTANT!**

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

## 7) b\_ADDIDATA\_StartCounter (...)

**i**

### IMPORTANT!

The function b\_ADDIDATA\_SetCounterDirection must be called up **before** calling up the function b\_ADDIDATA\_StartCounter.

The function b\_ADDIDATA\_SetCounterDirection generates a software trigger which loads the initialisation values necessary for the function b\_ADDIDATA\_StartCounter.

### Syntax:

```
<Return value> = b_ADDIDATA_StartCounter (DWORD dw_DriverHandle,
                                           BYTE      b_CounterNumber)
```

### Parameters:

#### - Input:

DWORD	<i>dw_DriverHandle</i>	Handle of the ADDI-DATA driver
BYTE	<i>b_CounterNumber</i>	Number of the counter
		The first counter begins from 0.

#### - Output:

No output signal has occurred.

### Task:

Starts the counter.

### Calling convention:

#### ANSI C:

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

```
b_ReturnValue = b_ADDIDATA_StartCounter (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\_StartAllCounters (...)****Syntax:**

<Return value> = b\_ADDIDATA\_StartAllCounters (DWORD dw\_DriverHandle)

**Parameters:****- Input:**

DWORD *dw\_DriverHandle*                      Handle of the ADDI-DATA driver

**- Output:**

No output signal has occurred.

**Task:**

Starts all counters of the board.

**Calling convention:**ANSI C :

BYTE                      b\_ReturnValue;

DWORD                    dw\_DriverHandle;

b\_ReturnValue = b\_ADDIDATA\_StartAllCounters                      (dw\_DriverHandle);

**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\_ClearCounter (...)

### Syntax:

<Return value> = b\_ADDIDATA\_ClearCounter (DWORD dw\_DriverHandle,  
BYTE b\_CounterNumber)

### Parameters:

#### - Input:

DWORD	<i>dw_DriverHandle</i>	Handle of the ADDI-DATA driver
BYTE	<i>b_CounterNumber</i>	Number of the counter
		The first counter begins from 0.

#### - Output:

A clear action occurs on the counter.

### Task:

Clear the counter.

### Calling convention:

#### ANSI C:

BYTE	b_ReturnValue;
DWORD	dw_DriverHandle;

b\_ReturnValue = b\_ADDIDATA\_ClearCounter (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.



**10) b\_ADDIDATA\_TriggerCounter (...)****Syntax:**

<Return value> = b\_ADDIDATA\_TriggerCounter (DWORD dw\_DriverHandle,  
 BYTE b\_CounterNumber)

**Parameters:****- Input:**

DWORD	<i>dw_DriverHandle</i>	Handle of the ADDI-DATA driver
BYTE	<i>b_CounterNumber</i>	Number of the counter
		First counter begins from 0.

**- Output:**

A trigger action occurs on the counter.

**Task:**

Trigger the counter.

If you have selected ADDIDATA\_DOWN for the function

b\_ADDIDATA\_SetCounterDirection, the counter reloads the start value.

If you have selected ADDIDATA\_UP for the function

b\_ADDIDATA\_SetCounterDirection, the counter is cleared.

**Calling convention:**ANSI C:

BYTE	b_ReturnValue;
DWORD	dw_DriverHandle;

b\_ReturnValue = b\_ADDIDATA\_TriggerCounter (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.

**11) b\_ADDIDATA\_TriggerAllCounters (...)****Syntax:**

<Return value> = b\_ADDIDATA\_TriggerAllCounters (DWORD dw\_DriverHandle)

**Parameters:****- Input:**

DWORD *dw\_DriverHandle*            Handle of the ADDI-DATA driver

**- Output:**

A trigger action occurs on the counter.

**Task:**

Trigger all counters of the board.

**Calling convention:**ANSI C :

BYTE            b\_ReturnValue;

DWORD          dw\_DriverHandle;

b\_ReturnValue = b\_ADDIDATA\_TriggerAllCounters (dw\_DriverHandle);

**Return value:**

1: No error

0: Error by calling up the function. Use the function "i\_ADDIDATA\_GetLastError",  
to find the error source.

## 12) b\_ADDIDATA\_StopCounter (...)

### Syntax:

<Return value> = b\_ADDIDATA\_StopCounter (DWORD dw\_DriverHandle,  
BYTE b\_CounterNumber)

### Parameters:

#### - Input:

DWORD	<i>dw_DriverHandle</i>	Handle of the ADDI-DATA driver
BYTE	<i>b_CounterNumber</i>	Number of the counter

The first counter begins from 0.

#### - Output:

No output signal has occurred.

### Task:

Stops the counter.

### Calling convention:

#### ANSI C:

BYTE	b_ReturnValue;
DWORD	dw_DriverHandle;

b\_ReturnValue = b\_ADDIDATA\_StopCounter (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.

### 13) b\_ADDIDATA\_StopAllCounters (...)

**Syntax:**

<Return value> = b\_ADDIDATA\_StopAllCounters (DWORD dw\_DriverHandle)

**Parameters:****- Input:**

DWORD *dw\_DriverHandle*      Handle of the ADDI-DATA driver

**- Output:**

No output signal has occurred.

**Task:**

Stops all counters of the board.

**Calling convention:**ANSI C:

BYTE      b\_ReturnValue;

DWORD      dw\_DriverHandle;

b\_ReturnValue = b\_ADDIDATA\_StopAllCounters(dw\_DriverHandle);

**Return value:**

1: No error

0: Error by calling up the function. Use the function "i\_ADDIDATA\_GetLastError",  
to find the error source.

#### 14) b\_ADDIDATA\_ReleaseCounter (...)

**Syntax:**

<Return value> = b\_ADDIDATA\_ReleaseCounter (DWORD dw\_DriverHandle,  
BYTE b\_CounterNumber)

**Parameters:****- Input:**

DWORD	<i>dw_DriverHandle</i>	Handle of the ADDI-DATA driver
BYTE	<i>b_CounterNumber</i>	Number of the counter
		First counter begins from 0.

**- Output:**

No output signal has occurred.

**Task:**

Releases the counter for a new initialisation.

**Calling convention:**ANSI C:

BYTE	b_ReturnValue;
DWORD	dw_DriverHandle;

b\_ReturnValue = b\_ADDIDATA\_ReleaseCounter (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.

### 15) b\_ADDIDATA\_ReadCounterValue

### Syntax:

<Return value> = b\_ADDIDATA\_ReadCounterValue

```
(DWORD    dw_DriverHandle,
BYTE      b_CounterNumber,
PDWORD    pdw_CounterValue)
```

### Parameters:

**- Input:**

DWORD	<i>dw_DriverHandle</i>	Handle of the ADDI-DATA driver
-------	------------------------	--------------------------------

BYTE	<i>b_CounterNumber</i>	Number of the counter
------	------------------------	-----------------------

The first counter begins from 0.

**- Output:**

PDWORD <i>pdw_CounterValue</i>	Counter value.
--------------------------------	----------------

### Task:

Reads the counter value.

### Calling convention:

## ANSI C :

BYTE	b_ReturnValue;
DWORD	dw_DriverHandle;
DWORD	dw_CounterValue;

```
b_ReturnValue = b_ADDIDATA_ReadCounterValue
(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.

**16) b\_ADDIDATA\_ReadCounterStatus (...)****Syntax:**

<Return Value> = b\_ADDIDATA\_ReadCounterStatus

(DWORD dw\_DriverHandle,  
 BYTE b\_CounterNumber,  
 PBYTE pb\_CounterStatus,  
 PBYTE pb\_SoftwareTriggerStatus,  
 PBYTE pb\_HardwareTriggerStatus,  
 PBYTE pb\_SoftwareClearStatus)

**Parameters:****- Input:**

DWORD *dw\_DriverHandle* Handle of the ADDI-DATA driver  
 BYTE *b\_CounterNumber* Number of the counter.  
 The first counter begins from 0.

**- Output:**

PBYTE *pb\_CounterStatus* 0: Counter ran down or did not start  
 1: Counter is running  
 PBYTE *pb\_SoftwareTriggerStatus* 0: Software trigger did not occur  
 1: Software trigger occurred  
 When the status of the software trigger is read, it is automatically reset. By the next calling of the parameter, 0 is returned if no trigger occurred during this period.  
 PBYTE *pb\_HardwareTriggerStatus* 0: Hardware trigger did not occur  
 1: Hardware trigger occurred  
 PBYTE *pb\_SoftwareClearStatus* 0: Software clear did not occur  
 1: Software clear occurred

**Task:**

Returns the status of the counter, the software trigger, the hardware trigger and the software clear.

**Calling convention:**ANSI C:

BYTE b\_ReturnValue;  
 DWORD dw\_DriverHandle;  
 BYTE b\_CounterStatus;  
 BYTE b\_SoftwareTriggerStatus;  
 BYTE b\_HardwareTriggerStatus;  
 BYTE b\_SoftwareClearStatus;

b\_ReturnValue = b\_ADDIDATA\_ReadCounterStatus  
 (dw\_DriverHandle,  
 0,  
 &b\_CounterStatus,  
 &b\_SoftwareTriggerStatus,  
 &b\_HardwareTriggerStatus,  
 &b\_SoftwareClearStatus);

**Return value:**

1: No error

0: Error by calling up the function. Use the function "i\_ADDIDATA\_GetLastError",  
to find the error source.



**17) b\_ADDIDATA\_EnableDisableCounterHardwareGate (...)****Syntax:**

<Return Value> = b\_ADDIDATA\_EnableDisableCounterHardwareGate  
 (DWORD dw\_DriverHandle,  
 BYTE b\_CounterNumber,  
 BYTE b\_HardwareGateFlag,  
 BYTE b\_HardwareGateLevel)

**Parameters:****- Input:**

DWORD	<i>dw_DriverHandle</i>	Handle of the ADDI-DATA driver
BYTE	<i>b_CounterNumber</i>	Number of the counter
		The first counter begins from 0.
BYTE	<i>b_ExternGateFlag</i>	ADDIDATA_ENABLE: enables the hardware gate. ADDIDATA_DISABLE: Hardware gate disabled by starting the counter
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 hardware gate.

**Calling convention:**ANSI C:

BYTE b\_ReturnValue;  
 DWORD dw\_DriverHandle;

b\_ReturnValue = b\_ADDIDATA\_EnableDisableCounterHardwareGate  
 (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.

## 18) b\_ADDIDATA\_GetCounterHardwareGateStatus (...)

### Syntax:

```
<Return Value> = b_ADDIDATA_GetCounterHardwareGateStatus  
                (DWORD dw_DriverHandle,  
                 BYTE   b_CounterNumber,  
                 PBYTE  pb_HardwareGateStatus)
```

### Parameters:

**- Input:**

DWORD	<i>dw_DriverHandle</i>	Handle of the ADDI-DATA driver
BYTE	<i>b_CounterNumber</i>	Number of the counter
		The first counter begins from 0.

**- Output:**

PBYTE    *pb\_HardwareGateStatus0*: Hardware gate is not active (low status)  
               1: Hardware gate is active (high status)

### Task:

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

### Calling convention:

## ANSI C :

```

BYTE      b_ReturnValue;
DWORD     dw_DriverHandle;
BYTE      b_HardwareGateStatus;

```

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

**19) b\_ADDIDATA\_EnableDisableCounterHardwareTrigger (...)****Syntax:**

<Return Value> = b\_ADDIDATA\_EnableDisableCounterHardwareTrigger  
 (DWORD dw\_DriverHandle,  
 BYTE b\_CounterNumber,  
 BYTE b\_HardwareTriggerFlag,  
 BYTE b\_HardwareTriggerLevel)

**Parameters:****- Input:**

DWORD	<i>dw_DriverHandle</i>	Handle of the ADDI-DATA driver
BYTE	<i>b_CounterNumber</i>	Number of the counter The first counter begins from 0.
BYTE	<i>b_HardwareTriggerFlag</i>	ADDIDATA_ENABLE: Enables the hardware trigger. ADDIDATA_DISABLE: Hardware trigger disabled by triggering the counter
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_HIGH_LOW: If the hardware trigger is used, it triggers from "0" to "1" or from "1" to "0"

**- Output:**

No output signal has occurred.

**Task:**

Releases or blocks the action of the hardware trigger.

**Calling convention:**ANSI C :

BYTE b\_ReturnValue;  
 DWORD dw\_DriverHandle;

```
b_ReturnValue = b_ADDIDATA_EnableDisableCounterHardwareTrigger
(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.

**20) b\_ADDIDATA\_GetCounterHardwareTriggerStatus (...)****Syntax:**

```
<Return Value> = b_ADDIDATA_GetCounterHardwareTriggerStatus
                                     (DWORD dw_DriverHandle,
                                     BYTE    b_CounterNumber,
                                     PBYTE   pb_HardwareTriggerStatus)
```

**Parameters:****- Input:**

DWORD	<i>dw_DriverHandle</i>	Handle of the ADDI-DATA driver
BYTE	<i>b_CounterNumber</i>	Number of the counter
		First counter begins from 0.

**- Output:**

PBYTE	<i>pb_HardwareTriggerStatus</i>	0: Hardware trigger is not active (low status)
		1: Hardware trigger is active (high status)

**Task:**

Returns the status of the hardware trigger (active or not).

**Calling convention:**ANSI C:

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

```
b_ReturnValue = b_ADDIDATA_GetCounterHardwareTriggerStatus
                                     (dw_DriverHandle,
                                     0,
                                     &b_HardwareTriggerStatus);
```

**Return value:**

1: No error  
 0: Error by calling up the function. Use the function "i\_ADDIDATA\_GetLastError",  
 to find the error source.

## 21) b\_ADDIDATA\_EnableDisableCounterHardwareOutput (...)

### Syntax:

<Return value> = b\_ADDIDATA\_EnableDisableCounterHardwareOutput  
(DWORD dw\_DriverHandle,  
BYTE b\_CounterNumber,  
BYTE b\_OutputFlag,  
BYTE b\_OutputLevel)

### Parameters:

**- Input:**

DWORD	<i>dw_DriverHandle</i>	Handle of the ADDI-DATA driver
BYTE	<i>b_CounterNumber</i>	Number of the counter The first counter begins from 0.
BYTE	<i>b_OutputFlag</i>	ADDIDATA_DISABLE: the counter output is not used. ADDIDATA_ENABLE: The counter output is used.
BYTE	<i>b_OutputLevel</i>	ADDIDATA_HIGH: If the counter overflows, output is set to high ADDIDATA_LOW: If the counter overflows the output is set to low

**- Output:**

No output signal has occurred.

### Task:

Activates or deactivates the counter output.

### Calling convention:

ANSI C :

BYTE	b_ReturnValue;
DWORD	dw_DriverHandle;

```
b_ReturnValue = b_ADDIDATA_EnableDisableCounterHardwareOutput
(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.

## 22) b\_ADDIDATA\_GetCounterHardwareOutputStatus (...)

### Syntax:

[illegible]

### Parameters:

**- Input:**

DWORD	<i>dw_DriverHandle</i>	Handle of the ADDI-DATA driver
BYTE	<i>b_CounterNumber</i>	Number of the counter
		The first counter begins from 0.

**- Output:**

PBYTE		<i>pb_HardwareOutputStatus</i> 0: Hardware output is not active (Low status)
		1: Hardware output is active (High status)

### Task:

Returns the status of the counter hardware output (active or not).

### Calling convention:

ANSI C :

BYTE	b_ReturnValue;
DWORD	dw_DriverHandle;
BYTE	b_HardwareOutputStatus;

```
b_ReturnValue = b_ADDIDATA_GateCounterHardwareOutputStatus
(dw_DriverHandle,
0,
&b_HardwareOutputStatus);
```

### Return value:

1: No error  
0: Error by calling up the function. Use the function "i\_ADDIDATA\_GetLastError",  
to find the error source.

### 23) b\_ADDIDATA\_TestCounterAsynchronousFIFOFull (...)

### Syntax:

<Return value> = b\_ADDIDATA\_TestCounterAsynchronousFIFOFull  
(DWORD dw\_DriverHandle,  
PBYTE pb\_Full)

### Parameters:

**- Input:**

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

**- Output:**

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

### Task:

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

### Calling convention:

ANSI C :

```

BYTE      b_ReturnValue;
DWORD     dw_DriverHandle;
BYTE      b_FIFOFull;

```

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

### Return value:

1: No error  
0: Error by calling up the function. Use the function "i\_ADDIDATA\_GetLastError",  
to find the error source.