



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

Software description

ADDIDRIVER

Timer

1	INTRODUCTION.....	1
2	SOFTWARE FUNCTIONS.....	2
1)	b_ADDIDATA_GetNumberOfTimers(...)	2
2)	b_ADDIDATA_GetTimerInformation (...)	3
3)	b_ADDIDATA_GetTimerInformationEx (...).....	6
4)	b_ADDIDATA_InitTimer (...).....	8
5)	b_ADDIDATA_EnableDisableTimerInterrupt (...).....	11
6)	b_ADDIDATA_StartTimer (...).....	12
7)	b_ADDIDATA_StartAllTimers (...)	13
8)	b_ADDIDATA_TriggerTimer (...)	14
9)	b_ADDIDATA_TriggerAllTimers (...)	15
10)	b_ADDIDATA_StopTimer (...).....	16
11)	b_ADDIDATA_StopAllTimers (...)	17
12)	b_ADDIDATA_ReleaseTimer (...).....	18
13)	b_ADDIDATA_ReadTimerValue	19
14)	b_ADDIDATA_ReadTimerStatus (...)	20
15)	b_ADDIDATA_EnableDisableTimerHardwareGate (...)	21
16)	b_ADDIDATA_GetTimerHardwareGateStatus (...)	22
17)	b_ADDIDATA_EnableDisableTimerHardwareTrigger (...)	23
18)	b_ADDIDATA_GetTimerHardwareTriggerStatus (...).....	24
19)	b_ADDIDATA_EnableDisableHardwareTimerOutput (...).....	25
20)	b_ADDIDATA_GetTimerHardwareOutputStatus (...).....	26
21)	b_ADDIDATA_TestTimerAsynchronousFIFOFull (...).....	27

Tables

Table 1-1: Type Declaration for Windows 98/NT/2000/XP	1
Table 2-1: Timer modes.....	4
Table 2-2: Resolution	4
Table 2-3: Time units	5
Table 2-4: Timer mode description.....	9

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

1) b_ADDIDATA_GetNumberOfTimers(...)

Syntax:

```
<Return value> = b_ADDIDATA_GetNumberOfTimers
                                   (DWORD dw_DriverHandle,
                                   PBYTE pb_TimerNumber)
```

Parameters:

- Input:

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

- Output:

PBYTE	<i>pb_TimerNumber</i>	Number of timers
-------	-----------------------	------------------

Task:

Returns the number of timers available on the board.

Calling convention:

ANSI C :

BYTE	b_ReturnValue;
DWORD	dw_DriverHandle;
BYTE	b_TimerNumber;

[illegible]

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_GetTimerInformation (...)

Syntax:

<Return value> = b_ADDIDATA_GetTimerInformation

(DWORD dw_DriverHandle,
 BYTE b_TimerNumber,
 PBYTE pb_TimerMode,
 PBYTE pb_TimerTimeUnit,
 PWORD pw_TimerTimeStep,
 PBYTE pb_Resolution,
 PDWORD pdw_HardwareGateAvailable,
 PDWORD pdw_HardwareTriggerAvailable,
 PDWORD pdw_OutputAvailable)

Parameters:

- Input:

DWORD *dw_DriverHandle* Handle of the ADDI-DATA driver
 BYTE *b_TimerNumber* Number of the timer.
 The first timer begins from 0.

- Output:

PBYTE *pb_TimerMode* Modes available for the timer
 See table 3-1.
 PBYTE *pb_TimerTimeUnit* Time units available. See table 3-3.
 PWORD *pw_TimerTimeStep* Possible time steps for the timer
 PBYTE *pb_Resolution* Selection of the resolution for the timer.
 See table 3-2.
 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

Task:

Returns the time units (*pb_TimerTimeUnit*), the time steps (*pw_TimerTimeStep*) and the resolution (*pb_Resolution*) which can be used for the selected timer (*b_TimerNumber*)

Table 2-1: Timer modes

Wert (Dec)	Wert (hex)	Selectable modes
1	1	Mode 0 available
2	2	Mode 1 available
3	3	Mode 0, 1 available
4	4	Mode 2 available
5	5	Mode 0, 2 available
6	6	Mode 1, 2 available
7	7	Mode 0,1,2 available
8	8	Mode 3 available
9	9	Mode 0, 3 available
10	A	Mode 1, 3 available
11	B	Mode 0,1, 3 available
12	C	Mode 2, 3 available
13	D	Mode 0, 2, 3 available
14	E	Mode 1, 2, 3 available
15	F	Mode 0, 1, 2, 3 available
16	10	Mode 4 available
17	11	Mode 0, 4 available
18	12	Mode 1, 4 available
19	13	Mode 0, 1, 4 available
20	14	Mode 2, 4 available
21	15	Mode 0, 2, 4 available
22	16	Mode 1, 2, 4 available
23	17	Mode 0, 1, 2, 4 available
24	18	Mode 3, 4 available
25	19	Mode 0, 3, 4 available
26	1A	Mode 1, 3, 4 available
27	1B	Mode 0, 1, 3, 4 available
28	1C	Mode 2, 3, 4 available
29	1D	Mode 0, 2, 3, 4 available
30	1E	Mode 1, 2, 3, 4 available
31	1F	Mode 0, 1, 2, 3, 4 available

Wert (Dec)	Wert (hex)	Selectable modes
32	20	Mode 5 available
33	21	Mode 0, 5 available
34	22	Mode 1, 5 available
35	23	Mode 0, 1, 5 available
36	24	Mode 2, 5 available
37	25	Mode 0, 2, 5 available
38	26	Mode 1, 2, 5 available
39	27	Mode 0,1,2,5 available
40	28	Mode 3, 5 available
41	29	Mode 0, 3, 5 available
42	2A	Mode 1, 3, 5 available
43	2B	Mode 0,1, 3, 5 available
44	2C	Mode 2, 3, 5 available
45	2D	Mode 0, 2, 3, 5 available
46	2E	Mode 1, 2, 3, 5 available
47	2F	Mode 0, 1, 2, 3, 5 available
48	30	Mode 4, 5 available
49	31	Mode 0, 4, 5 available
50	32	Mode 1, 4, 5 available
51	33	Mode 0, 1, 4, 5 available
52	34	Mode 2, 4, 5 available
53	35	Mode 0, 2, 4, 5 available
54	36	Mode 1, 2, 4, 5 available
55	37	Mode 0, 1, 2, 4, 5 available
56	38	Mode 3, 4, 5 available
57	39	Mode 0, 3, 4, 5 available
58	3A	Mode 1, 3, 4, 5 available
59	3B	Mode 0, 1, 3, 4, 5 available
60	3C	Mode 2, 3, 4, 5 available
61	3D	Mode 0, 2, 3, 4, 5 available
62	3E	Mode 1, 2, 3, 4, 5 available
63	3F	Mode 0, 1, 2, 3, 4, 5 available

Table 2-2: Resolution

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

Table 2-3: Time units

Value	Description
1	ns
2	µs
3	ns und µs
4	ms
5	ns und ms
6	µs und ms
7	ns und µs und ms
8	s
9	ns und s
10	µs und s
11	ns und ms und s
12	ms und s
13	ns und ms und s
14	µs und ms und s
15	ns und µs und ms und s

Calling convention:ANSI C:

```

BYTE      b_ReturnValue;
DWORD     dw_DriverHandle;
BYTE      b_TimerMode;
BYTE      b_TimerTimeUnit;
WORD      w_TimerTimeStep;
BYTE      b_Resolution;
DWORD     dw_HardwareGateAvailable;
DWORD     dw_HardwareTriggerAvailable;
DWORD     dw_OutputAvailable;

```

```

b_ReturnValue = b_ADDIDATA_GetTimerInformation
                (dw_DriverHandle,
                 0,
                 &b_TimerMode,
                 &b_TimerTimeUnit,
                 &w_TimerTimeStep,
                 &b_Resolution,
                 &dw_HardwareGateAvailable,
                 &dw_HardwareTriggerAvailable,
                 &dw_OutputAvailable);

```

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_GetTimerInformationEx (...)**Syntax:**

```
<Return value> = b_ADDIDATA_GetTimerInformationEx
                                (DWORD    dw_DriverHandle,
                                BYTE      b_TimerNumber,
                                pstr_GetTimerInformation ps_TimerInformation,
                                DWORD     dw_StructSize)
```

Parameters:**- Input:**

DWORD	<i>dw_DriverHandle</i>	Handle of the ADDI-DATA driver
BYTE	<i>b_TimerNumber</i>	Timer number. The first timer begins from 0.
DWORD	<i>dw_StructSize</i>	Size of the structure.

- Output:

pstr_GetTimerInformation ps_TimerInformation
Timer information.

structure returned :

```
typedef struct
{
```

BYTE	<i>b_TimerMode;</i>	Available timer mode
BYTE	<i>b_TimerTimeUnit;</i>	Available time unit
BYTE	<i>b_Resolution;</i>	Timer resolution. See table 2-3
BYTE	<i>b_InterruptAvailable;</i>	FALSE : Interrupt not available TRUE : Interrupt available
WORD	<i>w_TimerTimeStep;</i>	Time step of the timer
WORD	<i>w_Reserved1;</i>	
DWORD	<i>dw_HardwareGateAvailable;</i>	FALSE :Hardware gate not available TRUE : Hardware gate available
DWORD	<i>dw_HardwareGateLowAvailable;</i>	FALSE :Hardware gate low not available TRUE : Hardware gate low available
DWORD	<i>dw_HardwareGateHighAvailable;</i>	FALSE :Hardware gate high not available TRUE : Hardware gate high available
DWORD	<i>dw_HardwareTriggerAvailable;</i>	FALSE :Hardware trigger not available TRUE : Hardware trigger available
DWORD	<i>dw_HardwareTriggerLowAvailable;</i>	FALSE :Hardware trigger low not available TRUE : Hardware trigger low available
DWORD	<i>dw_HardwareTriggerHighAvailable;</i>	FALSE :Hardware trigger high not available TRUE : Hardware trigger high available
DWORD	<i>dw_HardwareOutputAvailable;</i>	FALSE :Hardware output not available TRUE : Hardware output available
DWORD	<i>dw_HardwareOutputLowAvailable;</i>	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_Reserved3;  
  
}str_GetTimerInformation,*pstr_GetTimerInformation;
```

Task:

Returns the time units (*pb_TimerTimeUnit*), the time steps (*pw_TimerTimeStep*) and the resolution (*pb_Resolution*) which can be used for the selected timer (*b_TimerNumber*)

Calling convention:

ANSI C:

```
BYTE      b_ReturnValue;  
DWORD     dw_DriverHandle;  
str_GetTimerInformation s_TimerInformation;
```

```
b_ReturnValue = b_ADDIDATA_GetTimerInformation  
                (dw_DriverHandle,  
                 0,  
                 &s_TimerInformation,  
                 sizeof (str_GetTimerInformation));
```

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_InitTimer (...)**Syntax:**

<Return value> = b_ADDIDATA_InitTimer

(DWORD dw_DriverHandle,
 BYTE b_TimerNumber,
 BYTE b_TimerMode,
 BYTE b_TimerTimeUnit,
 DWORD dw_ReloadValue)

Parameters:**- Input:**

DWORD	<i>dw_DriverHandle</i>	Handle of the ADDI-DATA driver
BYTE	<i>b_TimerNumber</i>	Number of the timer
		The first timer begins from 0.
BYTE	<i>b_TimerMode</i>	Selection of the timer mode. See table 3-4
BYTE	<i>b_TimerTimeUnit</i>	Selection of the time unit
		0: ns
		1: μ s
		2: ms
		3: s
		4: min
DWORD	<i>dw_ReloadValue</i>	Start value or time interval (depends from the used ADDI-DATA board) see function "b_ADDIDATA_GetTimerInformation"

- Output:

No output signal has occurred.

Task:

Initialises the timer.

Table 2-4: Timer mode description

Selected mode	Mode description	<i>dw_ReloadValue</i> description	Hardware gate input action
0	Mode 0 is typically used for event counting. After the initialisation, OUT is initially low, and remains low until the counter reaches 0. OUT goes high and remains high until a new count is written. See "b_ADDIDATA_WriteTimerValue" function.	Start counting value	Hardware gate
1	Mode 1 is similar to mode 0 except for the gate input action. The gate input is not used to enable or disable the timer (like in Mode 0), but to trigger it.	Start counting value	Hardware trigger
2	This mode functions like a divide-by- <i>ul_ReloadValue</i> counter. It is used to generate a real time clock interrupt. OUT is initially high after the initialisation. When the initial count has decremented to 1, OUT goes low for one CLK pulse. OUT then goes high again, the counter reloads the initial count (<i>ul_ReloadValue</i>) and the process is repeated. This action can generate an interrupt. See function "b_ADDIDATA_SetFunctionalityIntRoutine" and "b_ADDIDATA_EnableDisableTimerInterrupt"	Divider factor	Hardware gate
3	Mode 3 is typically used for baud rate generation. This mode is similar to mode 2 except for the duty cycle of OUT. OUT will initially be high after the initialisation. When half the initial count (<i>ul_ReloadValue</i>) has expired, OUT goes low for the remainder of the count. The mode is periodic; the sequence above is repeated indefinitely.	Divider factor	Hardware gate
4	OUT is initially high after the initialisation. When the initial count expires, OUT will go low for one CLK pulse and then go high again. The counting sequence is triggered by writing a new value. See "i_ADDIDATA_WriteTimerValue" function. If a new count is written during counting, it will be loaded on the next CLK pulse.	Start counting value	Hardware gate
5	Mode 5 is similar to mode 4 except for the gate input action. The gate input is not used to enable or disable the timer, but to trigger it.	Start counting value	Hardware trigger

Calling convention:ANSI C :

BYTE b_ReturnValue;

DWORD dw_DriverHandle;

b_ReturnValue = b_ADDIDATA_InitTimer

```
(dw_DriverHandle,  
0,  
0,  
0,  
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_EnableDisableTimerInterrupt (...)**Syntax:**

```
<Return value> = b_ADDIDATA_EnableDisableTimerInterrupt
                                     (DWORD dw_DriverHandle,
                                     BYTE    b_TimerNumber,
                                     BYTE    b_InterruptFlag)
```

Parameters:**- Input:**

DWORD	<i>dw_DriverHandle</i>	Handle of the ADDI-DATA driver
BYTE	<i>b_TimerNumber</i>	Number of the timer
		The first timer 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 timer process.

Calling convention:ANSI C :

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

```
b_ReturnValue = b_ADDIDATA_EnableDisableTimerInterrupt
               (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).

6) b_ADDIDATA_StartTimer (...)**Syntax:**

<Return value> = b_ADDIDATA_StartTimer (DWORD dw_DriverHandle,
BYTE b_TimerNumber)

Parameters:**- Input:**

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

- Output:

No output signal has occurred.

Task:

Starts the timer.

Calling convention:ANSI C:

BYTE	b_ReturnValue;
DWORD	dw_DriverHandle;

b_ReturnValue = b_ADDIDATA_StartTimer (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.

7) b_ADDIDATA_StartAllTimers (...)**Syntax:**

<Return value> = b_ADDIDATA_StartAllTimers (DWORD dw_DriverHandle)

Parameters:**- Input:**

DWORD *dw_DriverHandle* Handle of the ADDI-DATA driver

- Output:

No output signal has occurred.

Task:

Starts all timers of the board.

Calling convention:ANSI C :

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

```
b_ReturnValue = b_ADDIDATA_StartAllTimers (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.

8) b_ADDIDATA_TriggerTimer (...)**Syntax:**

<Return value> = b_ADDIDATA_TriggerTimer (DWORD dw_DriverHandle,
BYTE b_TimerNumber)

Parameters:**- Input:**

DWORD	<i>dw_DriverHandle</i>	Handle of the ADDI-DATA driver
BYTE	<i>b_TimerNumber</i>	Number of the timer

The first timer begins from 0.

- Output:

A trigger occurs on the timer.

Task:

Triggers the timer.

Calling convention:ANSI C:

BYTE	b_ReturnValue;
DWORD	dw_DriverHandle;

b_ReturnValue = b_ADDIDATA_TriggerTimer (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.

9) b_ADDIDATA_TriggerAllTimers (...)**Syntax:**

<Return value> = b_ADDIDATA_TriggerAllTimers (DWORD dw_DriverHandle)

Parameters:**- Input:**

DWORD *dw_DriverHandle* Handle of the ADDI-DATA driver

- Output:

A trigger occurs on the timer.

Task:

Triggers all timers of all boards.

Calling convention:ANSI C :

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

```
b_ReturnValue = b_ADDIDATA_TriggerAllTimers      (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.

10) b_ADDIDATA_StopTimer (...)**Syntax:**

<Return value> = b_ADDIDATA_StopTimer (DWORD dw_DriverHandle,
BYTE b_TimerNumber)

Parameters:**- Input:**

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

- Output:

No output signal has occurred.

Task:

Stops the timer.

Calling convention:ANSI C:

BYTE	b_ReturnValue;
DWORD	dw_DriverHandle;

b_ReturnValue = b_ADDIDATA_StopTimer (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_StopAllTimers (...)**Syntax:**

<Return value> = b_ADDIDATA_StopAllTimers (DWORD dw_DriverHandle)

Parameters:**- Input:**

DWORD *dw_DriverHandle* Handle of the ADDI-DATA driver

- Output:

No output signal has occurred.

Task:

Stops all timers of all boards.

Calling convention:ANSI C:

BYTE b_ReturnValue;
DWORD dw_DriverHandle;

b_ReturnValue = b_ADDIDATA_StopAllTimers (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_ReleaseTimer (...)**Syntax:**

<Return value> = b_ADDIDATA_ReleaseTimer (DWORD dw_DriverHandle,
BYTE b_TimerNumber)

Parameters:**- Input:**

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

- Output:

No output signal has occurred.

Task:

Releases the timer for a new initialisation.

Calling convention:ANSI C:

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

```
b_ReturnValue = b_ADDIDATA_ReleaseTimer (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_ReadTimerValue

<Return value> = b_ADDIDATA_ReadTimerValue (DWORD dw_DriverHandle,
 BYTE b_TimerNumber,
 PDWORD pdw_TimerValue)

Parameters:**- Input:**

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

- Output:

PDWORD	<i>pdw_TimerValue</i>	Timer value.
--------	-----------------------	--------------

Task:

Reads the timer value.

Calling convention:ANSI C:

```

BYTE      b_ReturnValue;
DWORD dw_DriverHandle;
DWORD dw_TimerValue;

```

```

b_ReturnValue = b_ADDIDATA_ReadTimerValue
               (dw_DriverHandle, 0, &dw_TimerValue);

```

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_ReadTimerStatus (...)**Syntax:**

<Return Value> = b_ADDIDATA_ReadTimerStatus
 (DWORD dw_DriverHandle,
 BYTE b_TimerNumber,
 PBYTE pb_TimerStatus,
 PBYTE pb_SoftwareTriggerStatus,
 PBYTE pb_HardwareTriggerStatus)

Parameters:**- Input:**

DWORD *dw_DriverHandle* Handle of the ADDI-DATA driver
 BYTE *b_TimerNumber* Number of the timer.
 The first timer begins from 0.

- Output:

PBYTE *pb_TimerStatus* 0: Timer ran down or did not start
 1: Timer 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

Task:

Returns the status of the timer, the software trigger and the hardware trigger.

Calling convention:ANSI C:

BYTE b_ReturnValue;
 DWORD dw_DriverHandle;
 BYTE b_TimerStatus;
 BYTE b_SoftwareTriggerStatus;
 BYTE b_HardwareTriggerStatus;

b_ReturnValue = b_ADDIDATA_ReadTimerStatus
 (dw_DriverHandle, 0, &b_TimerStatus,
 &b_SoftwareTriggerStatus, &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.

15) b_ADDIDATA_EnableDisableTimerHardwareGate (...)**Syntax:**

<Return Value> = b_ADDIDATA_EnableDisableTimerHardwareGate
 (DWORD dw_DriverHandle,
 BYTE b_TimerNumber,
 BYTE b_HardwareGateFlag,
 BYTE b_HardwareGateLevel)

Parameters:**- Input:**

DWORD	<i>dw_DriverHandle</i>	Handle of the ADDI-DATA driver
BYTE	<i>b_TimerNumber</i>	Number of the timer
		The first timer begins from 0.
BYTE	<i>b_ExternGateFlag</i>	ADDIDATA_ENABLE: enables the Hardware Gate. ADDIDATA_DISABLE: Hardware Gate disabled by starting the timer
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_EnableDisableTimerHardwareGate
 (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.

16) b_ADDIDATA_GetTimerHardwareGateStatus (...)

Syntax:

```
<Return Value> = b_ADDIDATA_GateTimerHardwareGateStatus  
                (DWORD dw_DriverHandle,  
                 BYTE   b_TimerNumber,  
                 PBYTE  pb_HardwareGateStatus)
```

Parameters:

- Input:

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

- Output:

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 timer hardware gate (Active or not).

Calling convention:

ANSI C :

```

BYTE      b_ReturnValue;
DWORD     dw_DriverHandle;
BYTE      b_HardwareGateStatus;

```

```
b_ReturnValue = b_ADDIDATA_GateTimerHardwareGateStatus
(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.

17) b_ADDIDATA_EnableDisableTimerHardwareTrigger (...)**Syntax:**

```
<Return Value> = b_ADDIDATA_EnableDisableTimerHardwareTrigger
                                (DWORD dw_DriverHandle,
                                BYTE   b_TimerNumber,
                                BYTE   b_HardwareTriggerFlag,
                                BYTE   b_HardwareTriggerLevel)
```

Parameters:**- Input:**

DWORD	<i>dw_DriverHandle</i>	Handle of the ADDI-DATA driver
BYTE	<i>b_TimerNumber</i>	Number of the timer The first timer begins from 0.
BYTE	<i>b_HardwareTriggerFlag</i>	ADDIDATA_ENABLE: Enables the hardware trigger. ADDIDATA_DISABLE: Hardware trigger disabled by triggering the timer
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" and 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_EnableDisableTimerHardwareTrigger
                (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_GetTimerHardwareTriggerStatus (...)**Syntax:**

<Return Value> = b_ADDIDATA_GateTimerHardwareTriggerStatus
 (DWORD dw_DriverHandle,
 BYTE b_TimerNumber,
 PBYTE pb_HardwareTriggerStatus)

Parameters:**- Input:**

DWORD	<i>dw_DriverHandle</i>	Handle of the ADDI-DATA driver
BYTE	<i>b_TimerNumber</i>	Number of the timer
		The first timer 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_GateTimerHardwareTriggerStatus
(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.

19) b_ADDIDATA_EnableDisableHardwareTimerOutput (...)**Syntax:**

```
<Return value> = b_ADDIDATA_EnableDisableTimerHardwareOutput
                                     (DWORD dw_DriverHandle,
                                     BYTE b_TimerNumber,
                                     BYTE b_OutputFlag,
                                     BYTE b_OutputLevel)
```

Parameters:**- Input:**

DWORD	<i>dw_DriverHandle</i>	Handle of the ADDI-DATA driver
BYTE	<i>b_TimerNumber</i>	Number of the timer
		The first timer begins from 0.
BYTE	<i>b_OutputFlag</i>	ADDIDATA_ENABLE or ADDIDATA_DISABLE
BYTE	<i>b_OutputInverted</i>	ADDIDATA_DISABLE: the timer output is not inverted. ADDIDATA_ENABLE: The timer output is inverted.

- Output:

No output signal has occurred.

Task:

Activates or deactivates the timer output.

Calling convention:ANSI C:

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

```
b_ReturnValue = b_ADDIDATA_EnableDisableTimerHardwareOutput
                (dw_DriverHandle,
                 0,
                 ADDIDATA_ENABLE,
                 ADDIDATA_DISABLE);
```

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_GetTimerHardwareOutputStatus (...)

Syntax:

```
<Return Value> = b_ADDIDATA_GateTimerHardwareOutputStatus  
                (DWORD   dw_DriverHandle,  
                 BYTE     b_TimerNumber,  
                 PBYTE    pb_HardwareOutputStatus)
```

Parameters:

- Input:

DWORD	<i>dw_DriverHandle</i>	Handle of the ADDI-DATA driver
BYTE	<i>b_TimerNumber</i>	Number of the timer The first timer 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 timer hardware output (Active or not).

Calling convention:

ANSI C :

```

BYTE    b_ReturnValue;
DWORD   dw_DriverHandle;
BYTE    b_HardwareOutputStatus;

```

```
b_ReturnValue = b_ADDIDATA_GateTimerHardwareOutputStatus
(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.

21) b_ADDIDATA_TestTimerAsynchronousFIFOFull (...)**Syntax:**

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

Parameters:**- Input:**

DWORD dw_DriverHandle Driver handle

- Output:

PBYTE pb_Full 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_TestTimerAsynchronousFIFOFull
                (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.