



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

Software description

ADDIDRIVER

Interrupt

| | | |
|----------|---|----------|
| 1 | INTRODUCTION | 1 |
| 2 | INTERRUPT FUNCTIONS..... | 2 |
| | 1) b_ADDIDATA_SetFunctionalityIntRoutineWin32 (..)..... | 2 |
| | 2) b_ADDIDATA_TestInterrupt (..) | 7 |
| | 3) i_ADDIDATA_ResetFunctionalityIntRoutine (..)..... | 8 |

Tables

Table 2-1: Functionality/interrupt mask

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 INTERRUPT FUNCTIONS

1) **b_ADDIDATA_SetFunctionalityIntRoutineWin32 (..)**

Syntax:

<Return value> = b_ADDIDATA_SetFunctionalityIntRoutineWin32

```
(DWORD dw_DriverHandle,
DWORD dw_Functionality,
BYTE b_UserCallingMode,
BYTE b_SharedMemoryMode,
DWORD dw_UserSharedMemorySize,
VOID **ppv_UserSharedMemory,
VOID v_FunctionName (DWORD dw_DriverHandle,
DWORD dw_Functionality,
DWORD dw_InterruptMask,
BYTE * pb_ByteArray,
WORD * pw_WordArray,
DWORD * pdw_DwordArray,
BYTE b_UserCallingMode,
VOID * pv_UserSharedMemory))
```

Parameters:

- Input:

| | |
|--------------------------------------|--|
| DWORD <i>dw_DriverHandle</i> | Handle of the driver |
| BYTE <i>dw_Functionality</i> | Determines the functionality on which the user want to install an interrupt routine. See table 2-1 |
| BYTE <i>b_UserCallingMode</i> | ADDIDATA_ASYNCHRONOUS_MODE: User routine is called by the driver interrupt thread. ADDIDATA_SYNCHRONOUS_MODE: User routine is directly called by the driver interrupt routine. |
| BYTE <i>b_SharedMemoryMode</i> | Determines if a shared memory is used or not and which shared memory is to be allocated ADDIDATA_SHARED_MEMORY_NOT_USED: No shared memory is used. ADDIDATA_NEW_SHARED_MEMORY: A new shared memory address is used. ADDIDATA_ALREADY_USED_SHARED_MEMORY: The user shared memory address is already used. This address is passed to the variable <i>ppv_UserSharedMemory</i> |
| DWORD <i>dw_UserSharedMemorySize</i> | Determines the size of the user shared memory in bytes. Only used if you have selected the mode ADDIDATA_NEW_SHARED_MEMORY. |
| VOID <i>v_FunctionName</i> | Name of the user interrupt routine |

i

IMPORTANT!

The size of the User Shared Memory is limited to 63 MB. It could cause problems if more memory is required.

- Output:

VOID ** *ppv_UserSharedMemory* Returns:

- the new address of the user shared memory if you have selected the `ADDIDATA_NEW_SHARED_MEMORY` mode
- or the used shared memory address if you have selected the mode `ADDIDATA_ALREADY_SHARED_MEMORY`.

Task:**i****Windows 32-bit information:**

For Windows 98/NT/2000/XP, 4 rings (ring 0 to ring 3) are available.

- The user application operates in ring 3. This ring does not give access to hardware.
- VXD and SYS driver operate in ring 0 and give access to hardware.
- Ring 0 has no direct access to global variable from ring 3. It has to use a shared memory.
- Ring 0 and ring 3 have a pointer that points on this shared memory. The 2 pointers are not configured under the same address.

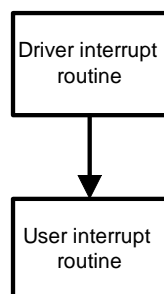
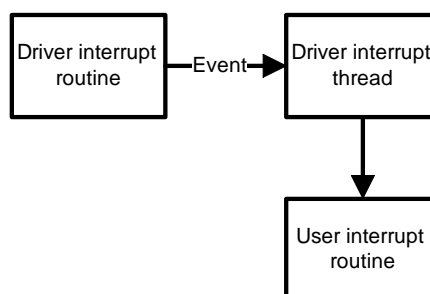
This function must be called up for each functionality for which an interrupt is to be enabled. It installs one user interrupt function for all functionalities of the same type.

Interrupt

The user interrupt routine is called up by the system when an interrupt is generated. The user interrupt routine can be called:

- directly by the driver interrupt routine (Synchronous mode). The code of the user interrupt routine directly operates in ring 0.
- by the driver interrupt thread (Asynchronous mode). An event is generated and the interrupt thread calls up the user interrupt routine. The code of the user interrupt routine operates in ring 3.

The driver interrupt thread have the highest priority (31) in the system.

Synchronous mode**Asynchronous mode**

| | SYNCHRONOUS MODE |
|---------------------|---|
| ADVANTAGE | The code of the user interrupt routine is directly called by the driver interrupt routine (ring 0). The time between the interrupt and the user interrupt routine is reduced. |
| RESTRICTIONS | The user cannot debug the user interrupt routine. |
| | The user routine cannot call Windows API functions. |
| | The user routine cannot call functions which have access to global variables. The user can still use a shared memory. |
| | This mode is not available for Visual Basic. |

| | ASYNCHRONOUS MODE |
|--------------------|---|
| ADVANTAGES | The user can debug the user interrupt routine provided he did not program in Visual Basic 5. |
| | The user routine can call Windows API functions. |
| | The user routine can call functions which give access to global variables. |
| RESTRICTION | The code of the user interrupt routine is called by the driver interrupt thread routine (ring 3). The time between the interrupt and the user interrupt routine is increased. |

Shared memory

If you have selected the `ADDIDATA_SYNCHRONOUS_MODE` you cannot have access to the Windows API variables. But you have the possibility to create a shared memory (`ppv_UserSharedMemory`). The user shared memory can have all predefined compiler types or user define types.

The variable `dw_UserSharedMemorySize` indicates the size in bytes of the selected user type. A pointer of the variable `ppv_UserSharedMemory` is given to the user interrupt routine with the variable `pv_UserSharedMemory`.

The `ADDIDATA_SYNCHRONOUS_MODE` is not possible for Visual Basic.

In `ADDIDATA_ASYNCHRONOUS_MODE` you can also use a shared memory. You do not have to pass through a global variable to access the shared variable.

If you selected the `ADDIDATA_ALREADY_USED_SHARED_MEMORY` mode (`b_SharedMemoryMode`) you have the possibility to use the same shared memory (`ppv_UserSharedMemory`) for 2 or more functionalities. If you have installed one interrupt functionality in the `ADDIDATA_SYNCHRONOUS_MODE` mode and another in the `ADDIDATA_ASYNCHRONOUS_MODE` mode you can use the same shared memory.

The user interrupt routine must have the following syntax:

```
VOID  v_FunctionName (DWORD  dw_DriverHandle,
                      DWORD  dw_Functionality,
                      DWORD  dw_InterruptMask,
                      BYTE *  pb_ByteArray,
                      WORD *  pw_WordArray,
                      DWORD * pdw_DwordArray,
                      BYTE    b_UserCallingMode,
                      VOID *  pv_UserSharedMemory)
```

| | |
|----------------------------|--|
| <i>v_FunctionName</i> | Name of the user interrupt routine |
| <i>dw_DriverHandle</i> | Handle of the driver which has generated the interrupt |
| <i>dw_Functionality</i> | Mask of the functionality which has generated the interrupt |
| <i>dw_InterruptMask</i> | Mask of the events which have generated the interrupt |
| <i>pb_ByteArray</i> | Pointer to a byte array. The value passed to this variable depends from the used functionality. See table 2-1 |
| <i>pw_WordArray</i> | Pointer to a word array. The value passed to this variable depends from the used functionality. See table 2-1 |
| <i>pdw_DwordArray</i> | Pointer to a double word array. The value passed to this variable depends from the used functionality. See table 2-1 |
| <i>b_UserCallingMode</i> | ADDIDATA_ASYNCHRONOUS_MODE: The user routine is called by the driver interrupt thread ADDIDATA_SYNCHRONOUS_MODE: The user routine is directly called by the driver interrupt routine. |
| <i>pv_UserSharedMemory</i> | Pointer of the user shared memory. |

The user can give another name for *v_FunctionName*, *dw_DriverHandle*, *dw_Functionality*, *dw_InterruptMask*, *pb_ByteArray*, *pw_WordArray*, *pdw_DwordArray*, *b_UserCallingMode*, *pv_UserSharedMemory*.

i

IMPORTANT!

If you use Visual Basic 4 the following parameters have no meaning. You must use the "i_ADDIDATA_TestInterrupt" function.

```
BYTE    b_UserCallingMode,
DWORD   dw_UserSharedMemorySize,
VOID ** ppv_UserSharedMemory,
VOID    v_FunctionName (DWORD  dw_DriverHandle,
                      DWORD  dw_Functionality,
                      DWORD  dw_InterruptMask,
                      BYTE *  pb_ByteArray,
                      WORD *  pw_WordArray,
                      DWORD * pdw_DwordArray,
                      BYTE    b_UserCallingMode,
                      VOID *  pv_UserSharedMemory)
```

Calling convention:ANSI C :

typedef struct

```
{
.
.
.
}str_UserStruct;
```

str_UserStruct * ps_UserSharedMemory;

```
void      v_FunctionName (DWORD      dw_DriverHandle,
                          DWORD      dw_Functionality,
                          DWORD      dw_InterruptMask,
                          BYTE *     pb_ByteArray,
                          WORD *     pw_WordArray,
                          DWORD *    pdw_DwordArray,
                          BYTE       b_UserCallingMode,
                          VOID *     pv_UserSharedMemory)
{
    str_UserStruct * ps_InterruptSharedMemory;

    ps_InterruptSharedMemory = (str_UserStruct *) pv_UserSharedMemory;
    .
    .
}
```

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

```
b_ReturnValue = b_ADDIDATA_SetFunctionalityIntRoutineWin32
                (dw_DriverHandle,
                 ADDIDATA_TIMER,
                 ADDIDATA_SYNCHRONOUS_MODE,
                 ADDIDATA_NEW_SHARED_MEMORY,
                 sizeof (str_UserStruct),
                 (void **) &ps_UserSharedMemory,
                 v_FunctionName);
```

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_TestInterrupt (..)**Syntax:**

```
<Return value> = b_ADDIDATA_TestInterrupt
                                (DWORD      dw_DriverHandle,
                                 PDWORD     pdw_Functionality,
                                 PDWORD     pdw_InterruptMask,
                                 PBYTE *    pb_ByteArray,
                                 PWORD *    pw_WordArray,
                                 PDWORD *   pdw_DwordArray,)
```

Parameters:**- Input:**

DWORD *dw_DriverHandle* Handle of the driver

- Output:

PDWORD *pdw_Functionality* Mask of the functionality which has generated the interrupt. See table 2-1

PDWORD *pdw_InterruptMask* Mask of the events which have generated the interrupt. See table 2-1

PBYTE *pb_ByteArray* Pointer to a byte array. The value passed to this variable depends from the used functionality. See table 2-1

PWORD *pw_WordArray* Pointer to a word array. The value passed to this variable depends from the used functionality. See table 2-1

PDWORD *pdw_DwordArray* Pointer to a double word array. The value passed to this variable depends from the used functionality. See table 2-1

Task:

Checks if a functionality has generated an interrupt. If yes, the function returns the functionality type and the interrupt source.

Calling convention:ANSI C:

```
BYTE      b_ReturnValue;
DWORD     dw_DriverHandle;
DWORD     dw_Functionality;
DWORD     dw_InterruptMask;
BYTE      b_ByteArray [120];
WORD      w_WordArray [120];
DWORD     dw_DwordArray [120];
b_ReturnValue = b_ADDIDATA_TestInterrupt (&dw_DriverHandle,
                                           &dw_Functionality,
                                           &dw_InterruptMask,
                                           b_ByteArray,
                                           w_WordArray,
                                           dw_DwordArray);
```

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_ResetFunctionalityIntRoutine (..)**

Syntax:

```
<Return value> = b_ADDIDATA_ResetFunctionalityIntRoutine
                                     (DWORD dw_DriverHandle,
                                     DWORD dw_Functionality)
```

Parameters:

- Input:

| | |
|-------------------------------|---|
| DWORD <i>dw_DriverHandle</i> | Driver handle |
| DWORD <i>dw_Functionality</i> | Determines the functionality on which the interrupt management is to be stopped. See table 2-1 |

- Output:

No output signal has occurred

Task:

Stops the interrupt management of the selected functionality (*dw_Functionality*).
Deinstalls the user interrupt routine for this functionality.

Calling convention:

ANSI C :

| | |
|-------|------------------|
| BYTE | b_ReturnValue; |
| DWORD | dw_DriverHandle; |

```
b_ReturnValue = b_ADDIDATA_ResetFunctionalityIntRoutine
                                     (dw_DriverHandle,
                                     ADDIDATA_TIMER);
```

Return value:

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

| Functionality mask (binary) | Functionality type (define) | Interrupt mask (binary) | Interrupt type | pb_ByteArray | pw_WordArray | pdw_DwordArray |
|-----------------------------|-----------------------------|---|---|--|---|---|
| 0000 0000 0000 0000 | ADDIDATA_DIGITAL_INPUT | Not used | See pw_WordArray | Not used Digital input status by the interrupt (only at the OR logic) | [0] type of interrupt logic 0 = Not used 1 = OR, 2 = AND | Mask of the digital inputs which generated the interrupt. 1 = Has generated, 0 = Has not generated. (Each bit stands for one input) |
| 0000 0000 0000 0001 | ADDIDATA_DIGITAL_OUTPUT | Not used | See pw_WordArray | Not used | [0] type of interrupt source D0→ 0= CC interrupt → 1= no CC interrupt D1→ 0 = VCC interrupt → 1= no VCC interrupt | Mask of the digital outputs which have generated the interrupt. 1 = Has generated 0 = Has not generated (Each bit stands for one output) |
| 0000 0000 0000 0100 | ADDIDATA_TIMER | 0000 0000 0000 0000 0000 0000 0000 0000 | Interrupt on Timer 0 | Not used | Not used | Not used |
| | | 0000 0000 0000 0000 0000 0000 0000 0001 | Interrupt on Timer 1 | | | |
| | | 0000 0000 0000 0000 0000 0000 0000 0010 | Interrupt on Timer 2 | | | |
| | | 0000 0000 0000 0000 0000 0000 0000 0011 | Interrupt on Timer 3 | | | |
| | | | ... | | | |
| 0000 0000 0000 0101 | ADDIDATA_COUNTER | 0000 0000 0000 0000 0000 0000 0000 0000 | Interrupt on Counter 0 | Not used | Not used | Not used |
| | | 0000 0000 0000 0000 0000 0000 0000 0001 | Interrupt on Counter 1 | | | |
| | | 0000 0000 0000 0000 0000 0000 0000 0010 | Interrupt on Counter 2 | | | |
| | | 0000 0000 0000 0000 0000 0000 0000 0011 | Interrupt on Counter 3 | | | |
| | | | ... | | | |
| 0000 0000 0000 0101 | ADDIDATA_WATCHDOG | 0000 0000 0000 0001 0000 0000 0000 0000 | Interrupt on Watchdog 0 (digital type) | Not used | Not used | Not used |
| | | 0000 0000 0000 0001 0000 0000 0000 0001 | Interrupt on Watchdog 1 (digital type) | | | |
| | | | | | | |
| | | 0000 0000 0000 0010 0000 0000 0000 0000 | Interrupt on Watchdog 0 (analog type) | | | |
| | | 0000 0000 0000 0010 0000 0000 0000 0001 | Interrupt on Watchdog 1 (analog type) | | | |
| | | | | | | |
| | | 0000 0000 0000 0011 0000 0000 0000 0000 | Interrupt on Watchdog 0 (digital/analog type) | | | |
| | | 0000 0000 0000 0011 0000 0000 0000 0001 | Interrupt on Watchdog 1 (digital/analog type) | | | |
| | | | | | | |
| | | 0000 0000 0000 0100 0000 0000 0000 0000 | Interrupt on Watchdog 0 (system type) | Not used | Not used | Not used |

| Functionality mask (binary) | Functionality type (define) | Interrupt mask (binary) | Interrupt type | pb_ByteArray | pw_WordArray | pdw_DwordArray |
|-----------------------------|-----------------------------|---|---|--------------|------------------------|--|
| 0000 0000 0000 0110 | ADDIDATA_TEMPERATURE | 0000 0000 0000 0100 0000 0000 0000 0001 | Interrupt on Watchdog 1 (system type) | Not used | [0] = Number of values | Returns the temperature digital value |
| | | | | | | |
| | | 0000 0000 0000 0001 0000 0000 0000 0000 | Interrupt generated after reading 1 temperature value on the channel 0 | Not used | [0] = Number of values | Returns the temperature digital values |
| | | 0000 0000 0000 0001 0000 0000 0000 0001 | Interrupt generated after reading 1 temperature value on the channel 1 | | | |
| | | 0000 0000 0000 0010 xxxx xxxx xxxx xxxx | Interrupt generated after reading several temperature values. (EOS) xxxx = Nbr of channels converted | Not used | [0] = Number of values | Returns the temperature digital values |
| | | 0000 0000 0000 0100 0000 0000 0000 0000 | Interrupt generated after reading several temperature values from a sequence | Not used | [0] = Number of values | Returns the temperature digital values [0] = Sequence Handle |
| | | 0000 0000 0000 1000 0000 0000 0000 0000 | Temperature warning interrupt from channel 0 | Not used | Not used | Returns the temperature warning digital value |
| | | 0000 0000 0000 1000 0000 0000 0000 0001 | Temperature warning interrupt from channel 1 | | | |
| | | 0000 0000 0001 0000 0000 0000 0000 0000 | Interrupt generated after the last conversion of the DMA. Temperature input module 0 | Not used | Not used | Returns the temperature digital values [0] = Number of buffer [1] = Number of digital values for the buffer 0 [2] = Buffer 0 ring 0 address [3] = Buffer 0 ring 3 address [4] = Number of digital values for the buffer 1 [5] = Buffer 1 ring 0 address [6] = Buffer 1 ring 3 address |
| | | 0000 0000 0001 0000 0000 0000 0000 0001 | Interrupt generated after the last conversion of the DMA. Temperature input module 1 | | | |
| | | | | | | |
| | | 0000 0000 0010 0000 0000 0000 0000 0000 | Interrupt generated at the end of a SCAN conversion | Not used | [0] = Number of values | Return the temperature digital values [0] = SCAN Handle |
| | | 0000 0000 0100 0000 0000 0000 0000 0000 | DMA Hardware FIFO overflow. Temperature input module 0 | Not used | Not used | Not used |
| | | 0000 0000 0100 0000 0000 0000 0000 0001 | DMA Hardware FIFO overflow. Temperature input module 1 | | | |
| | | .. | | | | |

| Functionality mask (binary) | Functionality type (define) | Interrupt mask (binary) | Interrupt type | pb_ByteArray | pw_WordArray | pdw_DwordArray |
|-----------------------------|-----------------------------|---|---|--------------|------------------------|---|
| 0000 0000 0000 0010 | ADDIDATA_ANALOG_INPUT | 0000 0000 0000 0001 0000 0000 0000 0000 | Interrupt generated after reading 1 analog input value on the channel 0 | Not used | [0] = Number of values | Returns the analog input digital value |
| 0000 0000 0000 0010 | ADDIDATA_ANALOG_INPUT | 0000 0000 0000 0001 0000 0000 0000 0001 | Interrupt generated after reading 1 analog input value on the channel 1 | | | |
| | | 0000 0000 0000 0010 xxxx xxxx xxxx xxxx | Interrupt generated after reading more analog input values. (EOS) xxxx = Nbr of channels converted | Not used | [0] = Number of values | Returns the analog input digital values |
| | | 0000 0000 0000 0100 0000 0000 0000 0000 | Interrupt generated after reading more analog input values from a sequence. | Not used | [0] = Number of values | Returns the analog input digital values [0] = Sequence Handle |
| | | 0000 0000 0000 1000 0000 0000 0000 0000 | Analog input warning interrupt from channel 0 | Not used | Not used | Returns the analog input warning digital value |
| | | 0000 0000 0000 1000 0000 0000 0000 0001 | Analog input warning interrupt from channel 1 | | | |
| | | 0000 0000 0001 0000 0000 0000 0000 0000 | Interrupt generated after the last conversion of the DMA. Analog input module 0 | Not used | Not used | Returns the analog input digital values [0] = Number of buffer [1] = Number of digital values for the buffer 0 [2] = Buffer 0 ring 0 address [3] = Buffer 0 ring 3 address [4] = Number of digital values for the buffer 1 [5] = Buffer 1 ring 0 address [6] = Buffer 1 ring 3 address |
| | | 0000 0000 0001 0000 0000 0000 0000 0001 | Interrupt generated after the last conversion of the DMA. Analog input module 1 | | | |
| | | 0000 0000 0010 0000 0000 0000 0000 0000 | Interupt generated at the end of a SCAN conversion | Not used | [0] = Number of values | Return the analog input digital values [0] = SCAN Handle |
| | | 0000 0000 0100 0000 0000 0000 0000 0000 | DMA hardware FIFO overflow. Analog input module 0 | Not used | Not used | Not used |
| | | 0000 0000 0100 0000 0000 0000 0000 0001 | DMA hardware FIFO overflow. Analog input module 1 | | | |
| | | | | | | |

| Functionality mask (binary) | Functionality type (define) | Interrupt mask (binary) | Interrupt type | pb_ByteArray | pw_WordArray | pdw_DwordArray |
|-----------------------------|-----------------------------|---|---|--------------|------------------------|---|
| 0000 0000 0000 1001 | ADDIDATA_RESISTANCE | 0000 0000 0000 0001 0000 0000 0000 0000 | Interrupt generated after reading 1 resistance value on the channel 0 | Not used | [0] = Number of values | Returns the resistance digital value |
| | | 0000 0000 0000 0001 0000 0000 0000 0001 | Interrupt generated after reading 1 resistance value on the channel 1 | | | |
| 0000 0000 0000 1001 | ADDIDATA_RESISTANCE | 0000 0000 0000 0010 xxxx xxxx xxxx xxxx | Interrupt generated after reading more resistance values. (EOS) xxxx = Nbr of channels converted | Not used | [0] = Number of values | Returns the resistance digital values |
| | | 0000 0000 0000 0100 0000 0000 0000 0000 | Interrupt generated after reading more resistance value from a sequence. | Not used | [0] = Number of values | Returns the resistance digital values [0] = Sequence Handle |
| | | 0000 0000 0000 1000 0000 0000 0000 0000 | Resistance warning interrupt from channel 0 | Not used | Not used | Returns the resistance warning digital value |
| | | 0000 0000 0000 1000 0000 0000 0000 0001 | Resistance warning interrupt from channel 1 | | | |
| | | 0000 0000 0001 0000 0000 0000 0000 0000 | Interrupt generated after the last conversion of the DMA. Resistance input module 0 | Not used | Not used | Returns the resistance digital values [0] = Number of buffer [1] = Number of digital values for the buffer 0 [2] = Buffer 0 ring 0 address [3] = Buffer 0 ring 3 address [4] = Number of digital values for the buffer 1 [5] = Buffer 1 ring 0 address [6] = Buffer 1 ring 3 address |
| | | 0000 0000 0001 0000 0000 0000 0000 0001 | Interrupt generated after the last conversion of the DMA. Resistance input module 1 | | | |
| | | | | | | |
| | | 0000 0000 0010 0000 0000 0000 0000 0000 | Interrupt generated at the end of a SCAN conversion | Not used | [0] = Number of values | Return the resistance digital value [0] = SCAN Handle |
| 0000 0000 0000 1011 | ADDIDATA_PRESSURE | 0000 0000 0000 0001 0000 0000 0000 0000 | Interrupt generated after reading 1 pressure value on the channel 0 | Not used | [0] = Number of values | Returns the pressure digital value |
| | | 0000 0000 0000 0001 0000 0000 0000 0001 | Interrupt generated after reading 1 pressure value on the channel 1 | | | |

| Functionality mask (binary) | Functionality type (define) | Interrupt mask (binary) | Interrupt type | pb_ByteArray | pw_WordArray | pdw_DwordArray |
|-----------------------------|-----------------------------|---|---|--------------|------------------------|---|
| | | 0000 0000 0000 00 10 xxxx xxxx xxxx xxxx | Interrupt generated after reading more pressure values. (EOS) xxxx = Nbr of channels converted | Not used | [0] = Number of values | Returns the pressure digital values |
| | | 0000 0000 0000 0 100 0000 0000 0000 0000 | Interrupt generated after reading more pressure values of a sequence. | Not used | [0] = Number of values | Returns the pressure digital values [0] = Sequence Handle |
| 0000 0000 0000 1011 | ADDIDATA_PRESSURE | 0000 0000 0000 1000 0000 0000 0000 0000 | Pressure warning interrupt, channel 0 | Not used | Not used | Returns the pressure warning digital value |
| | | 0000 0000 0000 1000 0000 0000 0000 000 1 | Pressure warning interrupt, channel 1 | | | |
| | | 0000 0000 000 1 0000 0000 0000 0000 0000 | Interrupt generated after the last conversion of the DMA. Pressure input module 0 | Not used | Not used | Returns the pressure digital values [0] = Number of buffer [1] = Number of digital values for the buffer 0 [2] = Buffer 0 ring 0 address [3] = Buffer 0 ring 3 address [4] = Number of digital values for the buffer 1 [5] = Buffer 1 ring 0 address [6] = Buffer 1 ring 3 address |
| | | 0000 0000 000 1 0000 0000 0000 0000 000 1 | Interrupt generated after the last conversion of the DMA. Pressure input module 1 | | | |
| | | | | | | |
| | | 0000 0000 00 10 0000 0000 0000 0000 0000 | Interrupt generated at the end of a SCAN conversion | Not used | [0] = Number of values | Return the pressure digital value [0] = SCAN Handle |
| | | 0000 0000 0 100 0000 0000 0000 0000 0000 | DMA Hardware FIFO overflow. Pressure input module 0 | Not used | Not used | Not used |
| | | 0000 0000 0 100 0000 0000 0000 0000 000 1 | DMA Hardware FIFO overflow. Pressure input module 1 | | | |
| 0000 0000 0000 1100 | ADDIDATA_TRANSUCER | 0000 0000 0000 000 1 0000 0000 0000 0000 | Interrupt generated after reading 1 transducer value on the channel 0 | Not used | [0] = Number of values | Returns the transducer digital value |
| | | 0000 0000 0000 000 1 0000 0000 0000 000 1 | Interrupt generated after reading 1 transducer value on the channel 1 | | | |

| Functionality mask (binary) | Functionality type (define) | Interrupt mask (binary) | Interrupt type | pb_ByteArray | pw_WordArray | pdw_DwordArray |
|-----------------------------|-----------------------------|---|---|--------------|------------------------|---|
| | | 0000 0000 0000 00 10 xxxx xxxx xxxx xxxx | Interrupt generated after reading more transducer values. (EOS) xxxx = Nbr of channels converted | Not used | [0] = Number of values | Returns the transducer digital values |
| | | 0000 0000 0000 0 100 0000 0000 0000 0000 | Interrupt generated after reading more transducer values from a sequence. | Not used | [0] = Number of values | Returns the transducer digital values [0] = Sequence Handle |
| | | 0000 0000 0000 1000 0000 0000 0000 0000 | Transducer warning interrupt from channel 0 | Not used | Not used | Returns the transducer warning digital value |
| 0000 0000 0000 1100 | ADDIDATA_TRANSDUCER | 0000 0000 0000 1000 0000 0000 0000 000 1 | Transducer warning interrupt from channel 1 | | | |
| | | 0000 0000 000 1 0000 0000 0000 0000 0000 | Interrupt generated after the last conversion of the DMA. Transducer input module 0 | Not used | Not used | Returns the transducer digital values [0] = Number of buffer [1] = Number of digital values for the buffer 0 [2] = Buffer 0 ring 0 address [3] = Buffer 0 ring 3 address [4] = Number of digital values for the buffer 1 [5] = Buffer 1 ring 0 address [6] = Buffer 1 ring 3 address |
| | | 0000 0000 000 1 0000 0000 0000 0000 000 1 | Interrupt generated after the last conversion of the DMA. Transducer input module 1 | | | |
| | | | | | | |
| | | 0000 0000 00 10 0000 0000 0000 0000 0000 | Interrupt generated at the end of a SCAN conversion | Not used | [0] = Number of values | Return the transducer digital value [0] = SCAN Handle |
| | | 0000 0000 0 100 0000 0000 0000 0000 0000 | DMA Hardware FIFO overflow. Transducer input module 0 | Not used | Not used | Not used |
| | | 0000 0000 0 100 0000 0000 0000 0000 000 1 | DMA Hardware FIFO overflow. Transducer input module 1 | | | |
| | | | | | | |
| | | 0000 0000 1000 0000 0000 0000 0000 0000 | Primary line short circuit interrupt on transducer module 0 | Not used | Not used | Not used |
| | | 0000 0000 1000 0000 0000 0000 0000 000 1 | Primary line short circuit interrupt on transducer module 1 | | | |

| Functionality mask (binary) | Functionality type (define) | Interrupt mask (binary) | Interrupt type | pb_ByteArray | pw_WordArray | pdw_DwordArray |
|-----------------------------|-----------------------------|-------------------------|----------------|--------------|--------------|----------------|
| | | ... | | | | |