

.NET-Klassenbibliothek zur Verwendung der

Cypress USB-Controller

AN2131 & AN2135 – Version 1.7

Copyright © 2003 - 2004 by Karsten Böhme

Karsten Böhme Hard- und Softwareentwicklung

An der Bismarckhöhe 8
01737 Tharandt

e-mail : karsten.boehme@arcor.de

Tel. : 035245 / 72530

Mobil : 0160 / 6763917

Fax : 069 / 13304446109

Inhalt :

1. Die Klassenbibliothek AN21XX_NET.dll
 - 1.1 Allgemeines
 - 1.2 Installation
 - 1.3 Anschlussbelegung des Controllers
 - 1.4 Die Klassen der Klassenbibliothek
2. Die Klasse clsAN21XX
 - 2.1 Die Eigenschaften der Klasse clsAN21XX
 - 2.2 Die Ereignisse, Exception und Delegaten der Klasse clsAN21XX
 - 2.3 Allgemeine Methoden der Klasse clsAN21XX (für alle Intefacemodi erforderlich)
 - 2.4 Methoden des Interface-Mode „CommonMode“
 - 2.5 Methoden des Parallelbus für die MC_Modi IMode_1 ... IMode_5
 - 2.6 Allgemeine Methoden des I2C-Bus
 - 2.7 Methoden zur Nutzung von I2C-EEPROM's
 - 2.8 Methoden des MC-Mode „SimpleMode“
 - 2.9 Nutzung der externen Interrupts EX0 und EX1

1. Die Klassenbibliothek (AN21XX_NET.DLL)

1.1 Allgemeines

Die Klassenbibliothek (geschrieben in C#) wurde zur Verwendung der Cypress-USB-Controller AN2131 u. AN2135 entwickelt. Das gewählte Konzept erlaubt die Verwendung der Klassenbibliothek mit allen .NET-Sprachen wie **C#** und **VB.NET**.

Daten des Controllers :

- Full-Speed USB-Maschine [12 MHz]
- Hot Plug in
- Automatische Enumeration
- 8051-kompatibler Microcontroller
- 8k Programm-RAM beim AN2131 und AN2135
- Firmware wird über den USB in den Programmspeicher geladen
- Integrierte I2C-Master-Kern

Funktionalität der Klassenbibliothek

- Kapseln der Treiberfunktionen
- Unterstützung von 32-Controllern am Bus
- Abfangen und Auswerten von auftretenden Fehlern
- Laden und Betreiben eigener Controllerprogramme über den USB
- Bereitstellen eines 8 Bit Parallelbus (wahlweise incl. 8Bit-Adressbus)
- Bereitstellen eines I2C-Bus
- Lesen und schreiben von Bytes auf den Parallelbus oder den I2C-Bus
- Lesen und schreiben von Blöcken auf den Parallelbus oder den I2C-Bus
- Lesen und schreiben von Bytes in die von der Firmware unterstützten EEPROM's
- Lesen und schreiben von Blöcken in die von der Firmware unterstützten EEPROM's
- Freie Verwendung eines oder aller Ports (abhängig vom Parameter „MC_Mode“)
- Nutzung der externen Interrupts 0 und 1 und benachrichtigen der Anwendung über Ereignisse
- Nach Einbinden der Klassenbibliothek sofortige Nutzung aller Funktionen

Welche Funktionalität der Controller zur Verfügung stellt ist vom übergebenen Wert des Parameters „**MC_Mode**“ abhängig. Danach richtet sich welche Firmware durch die DLL in den Controller geladen wird. Für jeden Wert dieses Parameters werden weiter unten alle möglichen Funktionen sowie die grundlegende Funktionalität des Controllers beschrieben.

Da der in mehreren Modi bereitgestellte Parallelbus über 8 Datenleitungen, 8 Adressleitungen sowie eine Lese- und eine Schreibleitung verfügt, können praktisch alle Peripheriebausteine für 8 Bit -Systeme betrieben werden. Es können damit umfangreiche Schaltungen und Geräte gesteuert werden. Genauso einfach können bereits bestehende Schaltungen für die RS232- oder den Druckerport nun für den USB modifiziert werden. Auch bei mancher Einsteckkarte lohnt sich die Überlegung das Projekt nun aus dem Rechner herauszuholen und als USB-Gerät auszuführen. Da einige Modi auf den Adressbus verzichten kann der dadurch frei werdende PortC für allgemeine Anwendung genutzt werden. Jedes Pin kann dabei wahlweise als Ein- oder Ausgang betrieben werden.

Bei Verwendung des **AN2135** sind Datenübertragungsraten von **1 MByte/s** über den Parallelbus möglich, dazu muss der Parameter „MC_Mode“ den Wert „IMode_3“ oder „IMode_4“ besitzen. Der AN2131 ermöglicht ca. 180 kByte/s

Mit den Funktionen, welche im allgemeinen Controllermodus (CommonMode) zur Verfügung stehen kann der Anwender eine selbst entwickelte Firmware über den USB in den Controller laden (wahlweise aus einer Binär- oder einer Intel-Hex-Datei). Die Kommunikation zwischen PC-Software und Controller erfolgt über das RAM des AN21XX, aus welchem auch die Firmware gestartet wird.

Durch Verwendung der von der Firma „Braintechology“ bereitgestellten Interfaceplatine können alle Eigenschaften dieses Paketes ausgetestet werden. Auf Grund des 2.54mm-Rastermaßes der Platine kann diese auch auf einem Steckbrett betrieben werden bzw. auf einer anderen Platine aufgesteckt werden.

Im folgenden wird die Installation der Software und des USB-Treibers beschrieben. Des weiteren werden die Übergabeparameter an die DLL, die möglichen Fehlermeldungen, sowie die Methoden (Funktionen und Prozeduren) der 6 möglichen Controller-Modi (MC_Mode) eingehend beschrieben.

1.2 Installation

Nach dem Entpacken der , AN21XX_NET.zip' existiert das Verzeichnis ,AN21XX_NET' mit folgenden Unterverzeichnissen :

\Doc	- enthält Dokumentation sowie Hinweise für Registrierung der Shareware
\Driver	- enthält inf-Datei sowie Gerätetreiber für USB-Controller
\Driver_XP_NT	- enthält modifizierte inf-Datei sowie Gerätetreiber für Windows-XP und Windows-NT
\NET_Dll	- enthält die Klassenbibliothek „AN21XX_NET.DLL“
\VB_NET_Sample	- enthält Testprogramm für Visual Basic .NET inklusive der AN21XX_NET.DLL
\CS_Sample	- enthält Testprogramm für C#

Achtung ! Die Klassenbibliothek enthält einen „**starken Namen**“ und kann somit im „Globalen Assembly Cache“ installiert werden. Dies muss beim Endkunden durch eine Installations-Routine erfolgen. Für den Entwickler genügt es, die DLL in das gewünschte Verzeichnis zu kopieren, diese anschließend bei gedrückter linker Maustaste auf den Ordner „assembly“ zu ziehen und dort die Maustaste loszulassen. Dadurch wird nicht die DLL kopiert sondern ein Verweis auf diese im „Global Assembly Cache“ erstellt. Nach diesem Vorgang ist der Verweis im Ordner Assembly sichtbar.

Zum Starten der Testprogramme und nutzen der Klassenbibliothek muss das **.NET-Framework** mindestens in **Version 1.1** auf dem Computer installiert sein ! Da die Testprogramme auch im Sourcecode vorliegen, können sie direkt in der IDE geladen und ausgeführt werden. Sollte das Testprogramm die DLL nicht finden, muss der Verweis erneuert werden.

Installieren des Gerätetreibers :

Wird der Controller erstmalig mit dem USB verbunden fragt der PC nach der zum Treiber gehörenden *.inf-Datei. Wurde der Pfad ausgewählt lädt der PC den Treiber.

Sollten Probleme auftreten, so kopieren Sie den Treiber ,EZUSB.sys' bitte von Hand in das entsprechende Verzeichnis (...System oder ...System32\Drivers). Nach erfolgreichem Abschluss der Installation wird der Treiber beim nächsten Verbinden des Controllers mit dem USB automatisch geladen.

1.3 Anschubelegung des Controllers

IMode_1 = 1 IMode_2 = 2 IMode_3 = 3 IMode_4 = 4 IMode_5 = 5 SimpleMode = 6

Port A

PA4	Pin39	#WR	#WR	#WR	#WR	#WR	frei verfügbar
PA5	Pin40	#RD	#RD	#RD	#RD	#RD	frei verfügbar

Port B

PB0	Pin24	D0	D0	D0	D0	AD0	frei verfügbar
PB1	Pin25	D1	D1	D1	D1	AD1	frei verfügbar
PB2	Pin26	D2	D2	D2	D2	AD2	frei verfügbar
PB3	Pin27	D3	D3	D3	D3	AD3	frei verfügbar
PB4	Pin28	D4	D4	D4	D4	AD4	frei verfügbar
PB5	Pin29	D5	D5	D5	D5	AD5	frei verfügbar
PB6	Pin30	D6	D6	D6	D6	AD6	frei verfügbar
PB7	Pin31	D7	D7	D7	D7	AD7	frei verfügbar

Port C

PC0	Pin14	A0	frei verfügbar	A0	frei verfügbar	frei verfügbar	frei verfügbar
PC1	Pin15	A1	frei verfügbar	A1	frei verfügbar	frei verfügbar	frei verfügbar
PC2	Pin16	A2	frei verfügbar	A2	frei verfügbar	frei verfügbar	frei verfügbar
PC3	Pin17	A3	frei verfügbar	A3	frei verfügbar	frei verfügbar	frei verfügbar
PC4	Pin18	A4	frei verfügbar	A4	frei verfügbar	frei verfügbar	frei verfügbar
PC5	Pin19	A5	frei verfügbar	A5	frei verfügbar	frei verfügbar	frei verfügbar
PC6	Pin20	A6	frei verfügbar	A6	frei verfügbar	#DACK	frei verfügbar
PC7	Pin21	A7	frei verfügbar	A7	frei verfügbar	ALE	frei verfügbar

ALE = Adress-Latch Enable

#DACK = Daten-Acknowledge (L-aktiv) ADx = gemultiplexte Adress- und Datenleitung

I2C-Bus :

SDA = Pin35
SCL = Pin36

! Achtung !

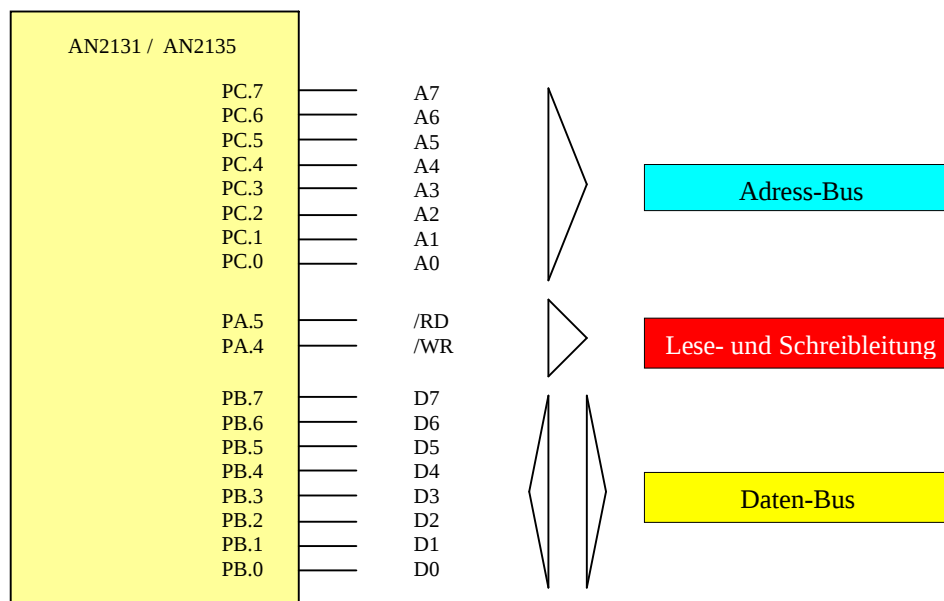
In Abhängigkeit des Wertes des Parameter „**MC_Mode**“ können der Port C bzw. alle Ports frei verfügbar sein. Lesen Sie dazu bitte die Beschreibung der einzelnen Werte dieses Übergabeparameter !

Der Parameter ‚MC_Mode‘ wird der Funktion ‚**SetDeviceEnabled**‘ übergeben und entscheidet darüber, in welchem Mode der Controller aktiviert wird, also ob, und wenn ja, welche Firmware geladen wird.

Beim AN2135 kann der PortB grundsätzlich nicht frei genutzt werden. Dieser ist für den schnellen Datentransfer (Fast-Transfer) vorgesehen und kann nur durch eigene Firmware angesprochen werden !

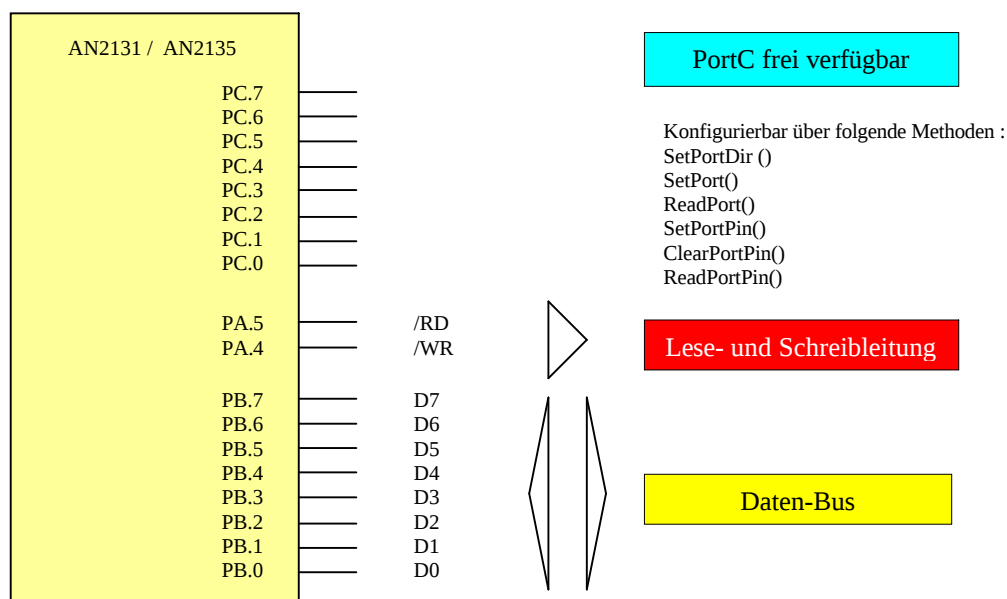
Betrieb der Controller in den Interface-Modi IMode_1 und IMode_3

IMode_1 = AN2131
Imode_3 = AN2135



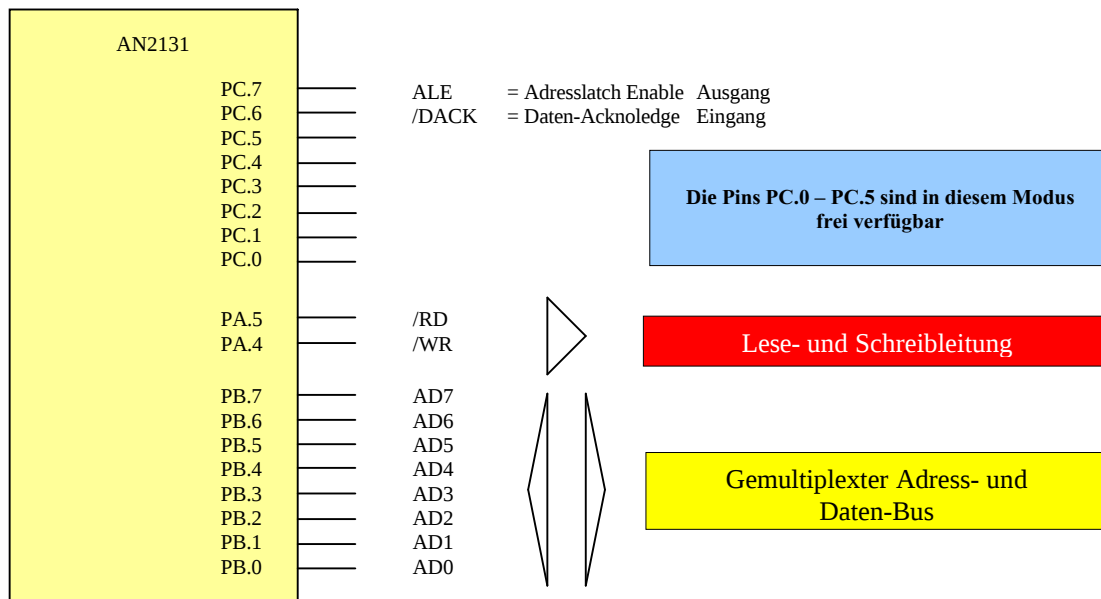
Betrieb der Controller in den Interface-Modi IMode_2 und IMode_4

IMode_2 = AN2131
Imode_4 = AN2135

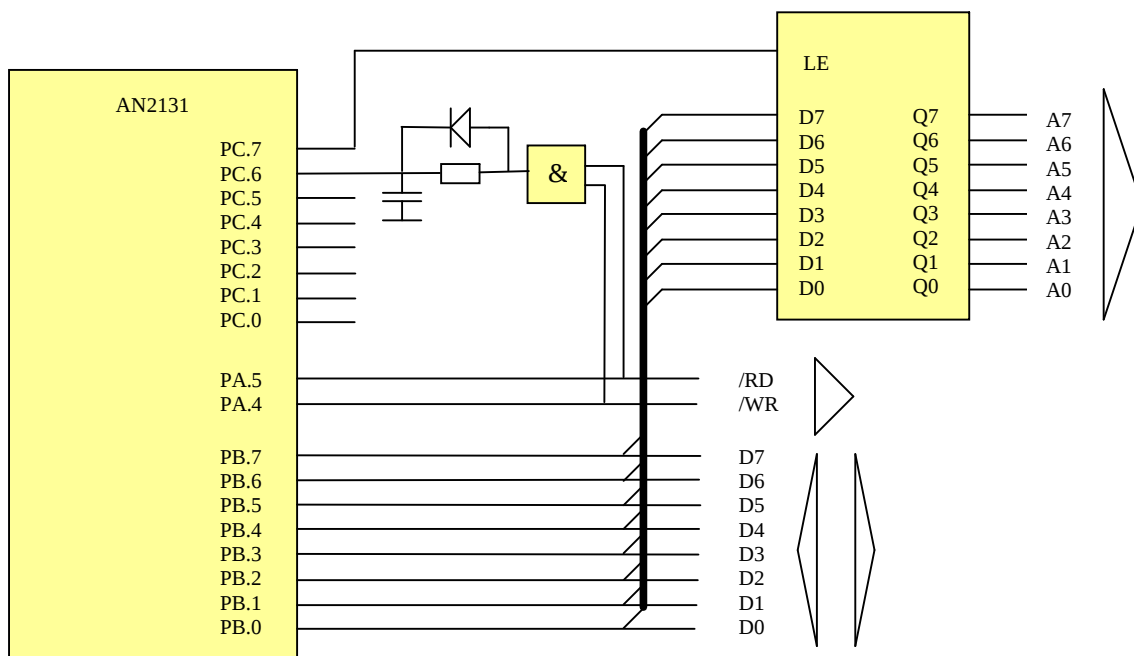


Betrieb des Controller im Interface-Mode IMode_5

Nur AN2131 !



Schaltungsbeispiel



Hinweis !

Wenn nicht benötigt, kann der Eingang /DACK auch über einen Widerstand (ca. 4k) auf GND gelegt werden. Bei Microcontroller-Systemen kann /DACK über einen Ausgang des Controllers nach erfolgter Datenübernahme auf Low gesetzt werden !
Wenn /RD oder /WR wieder H-Pegel annehmen muss /DACK sofort wieder auf H-Pegel gesetzt werden.

1.4 Die Klassen der Klassenbibliothek

Im folgenden werden die von der Klassenbibliothek AN21XX_NET.DLL bereitgestellten Klassen, Eigenschaften und Methoden sowie deren Anwendung beschrieben. Alle Klassen der Klassenbibliothek befinden sich im `namespace ,USB'`.

Nachdem dem Projekt ein Verweis auf die Assembly „AN21XX_NET.dll“ hinzugefügt wurde, muss der namespace ,USB' geöffnet werden und die Klasse `clsAN21XX` instanziiert werden :

```
VB.NET      imports USB
              ...
              Dim AN21XX As New clsAN21XX

—

C#         using System;
              using USB;
              ...
              clsAN21XX AN21XX = new clsAN21XX;
```

Da alle anderen Klassen nur Konstanten enthalten müssen von diesen auch keine Instanzen erstellt werden. Auf sie kann direkt über ihren Namen zugegriffen werden . (siehe weiter unten auf dieser Seite)

Klassen :

<code>clsAN21XX</code>	- Klasse enthält die Eigenschaft <code>I_Error</code> sowie alle Methoden zur Nutzung der Controller AN2131 und AN2135
<code>DeviceException</code>	- Ausnahmeklasse abgeleitet von <code>SystemException</code> . Besitzt die Eigenschaft „ <code>ThrowException</code> “ den Wert <code>,True'</code> , wird bei Auftreten eines Fehlers eine Ausnahme vom Typ <code>, DeviceException '</code> geworfen.
<code>MC_Modi</code>	- Klasse enthält Konstanten für die implementierten MC-Modi.
<code>Ports</code>	- Klasse enthält Konstanten zur Übergabe des betreffenden Ports (<code>PortA ... PortC</code>)
<code>Pins</code>	- Klasse enthält Konstanten zur Definition der einzelnen Pins eines Ports (<code>Pin0 ... Pin7</code>)
<code>EEPTyp</code>	- Klasse enthält Konstanten zur Definition der softwaremässig implementierten EEPROM-Typen
<code>IError</code>	- Klasse enthält Konstanten für alle zur Zeit spezifizierten Fehler der Klasse <code>clsAN21XX</code> . Diese werden durch die Eigenschaft <code>I_Error</code> der Klasse <code>clsAN21XX</code> zurück gegeben.
<code>Interrupt</code>	- Klasse enthält Konstanten für den Aufruf der Methode <code>, SetInterrupt '</code>

Hinweis : Bitte nutzen Sie bei der Verwendung der AN21XX_NET.DLL immer die zur Verfügung gestellten Konstanten. Dadurch werden Wartungsarbeiten bei Nutzung einer neuen Version der Klassenbibliothek minimiert.

Beispiel :	VB.NET	<pre>If AN21XX.I_Error = IError.Err_DeviceHandle Then ... End If</pre>
	C#	<pre>if (AN21XX.I_Error == IError.Err_DeviceHandle) { ... }</pre>

Konstanten der Klasse IError :

Err_No	= 0x0000	kein Fehler aufgetreten
Err_DeviceHandle	= 0x0001	kein gültiges Device-Handle
Err_DeviceNumber	= 0x0002	unzulässige Device-Nummer (zulässig 0 .. 31)
Err_DriverName	= 0x0004	kein gültiger Treiber-Name
Err_IsDisabled	= 0x0008	Control ist Disabled gesetzt
Err_DeviceIOCtrl	= 0x0010	Funktion DeviceIOControl hat 'False' zurückgeliefert
Err_AI2CNotAck	= 0x0020	I2C-Bus NotAcknowledge nach Slaveadresse
Err_DI2CNotAck	= 0x0040	I2C-Bus NotAcknowledge nach Datum
Err_I2CBus	= 0x0080	allgemeiner I2C-Busfehler
Err_Array	= 0x0100	ByteArray hat unerlaubte Dimension z.B. < 2 Elemente
Err_McMode	= 0x0200	Methode im aktuellen Controllermodus nicht verfügbar
Err_Pointer	= 0x0400	Ungültige RAM-/EEP-Adresse übergeben
Err_DownloadFile	= 0x0800	Datei existiert nicht, lässt sich nicht öffnen, hat falsches Format
Err_PinNumber	= 0x1000	Unzulässige Pinnummer
Err_EEPTyp	= 0x2000	unzulässiger EEPROM-Typ
Err_ClockDelay	= 0x4000	Parameter ausserhalb des zulässigen Bereiches (0 ... 7 erlaubt)
Err_InterruptNumber	= 0x8000	Externer Interrupt existiert nicht - zulässig 0 und 1

Hinweis : Wird bei den Block-Methoden für die EEPROM's eine Array übergeben welches grösser als die Speichertiefe des gewählten EEPROM ist, so wird der Fehler „Err_Array“ zurückgegeben.
Wird ein EEPROM-Pointer übergeben bei welchem bei der Grösse des Array's die Speicher-Tiefe des gewählten EEPROM überschritten wird, so wird der Fehler „Err_Pointer“ zurückgegeben.

Bei den Methoden des Parallelbus ist als obere Grenze der Bytezahl der Wert **16777215**. Dies ist aber mehr ein theoretischer Wert, da die Übertragung gerade beim AN2131 (180kB/s) andere Programme zu lange blockieren würde.

Bei den allgemeinen Methoden des I2C-Bus beträgt die maximale Byte-Zahl 65535.

2. Die Klasse clsAN21XX

2.1 Die Eigenschaften der Klasse clsAN21XX

ThrowException Die Eigenschaft **ThrowException** ist eine Eigenschaft der Klasse clsAN21XX und wird über eine Instanz dieser Klasse bereitgestellt. Der Datentyp ist **System.bool** . Besitzt diese Eigenschaft den Wert True so wird bei Auftreten eines Fehlers eine Exception vom Type „DeviceException“ geworfen. (Beispiel weiter unten in diesem Abschnitt)

I_Error Die Eigenschaft **I_Error** ist eine Eigenschaft der Klasse clsAN21XX und wird über eine Instanz dieser Klasse bereitgestellt. Der Datentyp ist **System.Int32** (signed 32-Bit) . Sie gibt alle während eines Methodenaufrufes aufgetretenen Fehler zurück. **I_Error** kann nur gelesen werden.

Die Fehlerkonstanten sind in der Klasse ‚IError‘ der Klassenbibliothek veröffentlicht.
(siehe weiter oben in diesem Abschnitt)

IntRefreshTime Die Eigenschaft **IntRefreshTime** ist eine Eigenschaft der Klasse clsAN21XX und wird über eine Instanz dieser Klasse bereitgestellt. Der Datentyp ist **System.Int32** (signed 32-Bit) . Über diese Eigenschaft wird festgelegt, aller wieviel Millisekunden die Flags der externen Interrupts EX0 und EX1 abgefragt werden. Dabei darf ein Wert von 50 ms nicht unterschritten und ein Wert von 10000 ms nicht überschritten werden. Ist dies der Fall, so wird ein Wert von 100 ms eingestellt. Dies ist auch der Default-Wert beim Instanzieren der Klasse !

Die genaue Verfahrensweise bei der Nutzung der externen Interrupts wird im Abschnitt 2.8 beschrieben !

2.2 Die Ereignisse, Exception und Delegaten der Klasse clsAN21XX

Interrupt **Delegat** Wird in C# mit den Ereignissen **InterruptEX0** und **InterruptEX1** verbunden

InterruptEX0 **Ereigniss** Am Pin des externen Interrupt 0 ist ein H/L-Flanke aufgetreten

InterruptEX1 **Ereigniss** Am Pin des externen Interrupt 1 ist ein H/L-Flanke aufgetreten

Die Klasse clsAN21XX besitzt 2 Ereignisse, welche bei Nutzung der externen Interrupts EX0 und EX1 des Controllers ausgelöst werden. Zuvor muss der jeweilige Interrupt jedoch mit der Methode „SetInterrupt“ freigegeben bzw. aktiviert worden sein. Die genaue Verfahrensweise zur Nutzung der externen Interrupts wird in Abschnitt 2.9 beschrieben.

2.3 Allgemeine Methoden (für alle Intefacemodi erforderlich)

SetDeviceEnabled (aktiviert einen Controller im angegeben Mode)

SetDeviceEnabled (DevNumber, DriverName, MC_Mode, ID, KeyWord)

Übergabeparameter :

DevNumber	(System.Byte)	-	Nummer des zu aktivierenden Controllers am Bus. Ist nur ein AN21xx am Bus, so muss immer 0 übergeben werden.
DriverName	(System.String)	-	Hier wird der Name des Treibers übergeben. Wird der Standarttreiber von Cypress verwendet ist hier „EZUSB“ einzugeben. Wurde der Treiber abgeändert (siehe Elektor 11/2002) so ist hier der Name des abgeänderten Treibers zu übergeben.
MC_Mode		-	CommonMode allgemeiner Mode, es kann eigene Software in den Controller geladen und ausgeführt werden. Siehe : Methoden MC_Mode „CommonMode“ IMode_1 für AN2131 – PortB als Datenbus, PortC als Adressbus IMode_2 für AN2131 – PortB als Datenbus, PortC frei verfügbar IMode_3 für AN2135 – PortB als Datenbus, PortC als Adressbus IMode_4 für AN2135 – PortB als Datenbus, PortC frei verfügbar Imode_5 für AN2131 – PortB als gemultiplexer Datn/Adressbus Pin7 von PortC stellt das ALE-Signal zur Verfügung mit welchem die Adresse gelatcht werden kann Pin6 ist der nDACK-Eingang für die Daten. Somit können auch langsame Systeme an den AN2131 angeschlossen werden. Pin0 bis Pin5 des PortC sind frei verfügbar. (über SetPort-Methoden ...) SimpleMode Alle Ports frei verfügbar (über SetPort-Methoden ...)

Hinweis : Die Methoden für den allgemeinen I2C-Bus (I2C_...) und die EEPROM's (EEP_...) sind in allen Modi ausser „CommonMode“ verfügbar !

ID - Hier wird bei der Mehrplatzlizens die persönliche ID eingetragen, bei der Einzelplatzlizens Wird hier „HD“ für ‚Hard-Disk‘ übergeben.

KeyWord - Hier wird die verschlüsselte ID übergeben. Diesen Schlüssel erhalten Sie nach der Lizenzierung .

Mögliche Fehler :
 IError.Err_DeviceNumber
 IError.Err_DriverName
 IError.Err_DeviceHandle
 IError.Err_DeviceIoControl

Beispiel	VB.NET	<pre> AN21XX.SetDeviceEnabled(0, „EZUSB“, MC_Modi.IMode_1, "", "") If AN21XX.I_Error <> IError.Err_No Then ... End If </pre>
	C#	<pre> AN21XX.SetDeviceEnabled(0, „EZUSB“, MC_Modi.IMode_1, "", ""); If (AN21XX.I_Error != IError.Err_No) { ... } </pre>

SetDeviceDisabled (deaktiviert einen Controller – Disabled – und schliesst dessen Treiber)

SetDeviceDisabled (DevNumber)

DevNumber (System.Byte) - Nummer des Controllers am Bus. Ist nur ein AN21xx am Bus, so muss immer 0 übergeben werden.

Mögliche Fehler : IError.DeviceNumber

Beispiel	VB.NET	AN21XX.SetDeviceDisabled(0) ` Deaktiviert Cotroller 0 u. schliesst Treiber
	C#	AN21XX.SetDeviceDisabled(0); // Deaktiviert Cotroller 0 u. schliesst Treiber

SetAllDisabled

(deaktiviert alle aktiven Controller und schliesst deren Treiber)

Beispiel	VB.NET	AN21XX.SetAllDisabled ()
	C#	AN21XX.SetAllDisabled ();

GetDeviceEnabled

(ermittelt ob ein Controller aktiviert ist – Enabled)

GetDeviceEnabled (DevNumber, MC_Mode)

Rückgabewert (System.Bool) - gibt bei aktiven Interface „True“ zurück, ansonsten „False“

Übergabeparameter :

DevNumber (System.Byte) - Nummer des Controllers am Bus. Ist nur ein AN21xx am Bus, so muss immer 0 übergeben werden.

MC_Mode (System.Byte) - liefert bei aktivem Controller den eingestellten Controllermodus zurück (z.B. IMode_1)

Mögliche Fehler :
IError.Err_DeviceNumber
IError.Err_DeviceHandle

Beispiel	VB.NET	<pre>Dim MC_Mode As Byte If Not GetDeviceEnabled (0, MC_Mode) AN21XX.SetDeviceEnabled(0, „EZUSB“, MC_Mode.IMode_1, „“, „“) ... End If</pre>
	C#	<pre>byte MC_Mode If (!GetDeviceEnabled (0, ref MC_Mode)) { AN21XX.SetDeviceEnabled(0, „EZUSB“, MC_Mode.IMode_1, „“, „“); ... }</pre>

GetDeviceID

(gibt die ID's eines aktiven Controllers zurück)

GetDeviceID (DevNumber, VID, PID, DID)

Übergabeparameter :

DevNumber (System.Byte) - Nummer des Controllers am Bus. Ist nur ein AN21xx am Bus, so muss immer 0 übergeben werden.

VID (System.Byte) - liefert die Vendor ID des Controllers

PID (System.Byte) - liefert die Produkt ID des Controllers

DID (System.Byte) - liefert die Device ID des Controllers

Mögliche Fehler :
IError.Err_DeviceNumber
IError.Err_IsDisabled
IError.Err_DeviceIoControl
IError.Err_MC_Mode

Beispiel	VB.NET	<pre>Dim VID As Byte Dim PID As Byte Dim DID As Byte AN21XX.GetDeviceID(0, VID, PID, DID) ...</pre>
	C#	<pre>byte VID = 0; byte PID = 0; byte DID = 0; AN21XX.GetDeviceID(0, ref VID, ref PID, ref DID) { ... }</pre>

SetClockDelay

Methode setzt in allen Modi ausser „CommonMode die Clock-Verzögerung für X-RAM-Zugriffe.

Das wirkt sich in den Modi „lMode_2“ und „lMode_4“ auch auf die Übertragungsgeschwindigkeit der Parallelbus-Blockübertragung aus

SetClockDelay (DevNum, Delay)

Übergabeparameter :

DevNumber (System.Byte) - Nummer des Controllers am Bus. Ist nur ein AN21xx am Bus, so muss immer 0 übergeben werden.

Delay (System.Byte) - Wert der Clockverzögerung. Erlaubte Werte 0 ... 7 (0 = höchste Geschwindigkeit)

Mögliche Fehler :
IError.Err_DeviceNumber
IError.Err_IsDisabled
IError.Err_DeviceIoControl
IError.Err_MC_Mode
IError.Err_ClockDelay

Beispiel	VB.NET	AN21XX.SetClockDelay (0, 5)	` Setzt die Verzögerung auf den Wert 5
	&		
	C#	AN21XX.SetClockDelay (0, 5);	// Setzt die Verzögerung auf den Wert 5

2.4 Methoden des Interface-Mode „CommonMode“

Dieser Interfacemodus dient der Nutzung eigener Firmware. Diese kann in den Controller geladen und dort ausgeführt werden. Die Nutzung dieses Modus erfordert genaue Kenntnisse des Controller-Innenen.

SetInterface (setzt ein Interface des USB-Gerätes und ein AltSetting – Konfiguration der Endpoint's)

Übergabeparameter :

SetInterface (DevNumber, InterfaceNumber, Setting)

DevNumber	(System.Byte)	- Nummer des Controllers am Bus. Ist nur ein AN21xx am Bus, so muss immer 0 übergeben werden.
InterfaceNumber	(System.Byte)	- zu setzendes Interface
Setting	(System.Byte)	- zu aktivierende Konfiguration des Interface

Mögliche Fehler :

- IErrors.Err_DeviceNumber
- IErrors.Err_IsDisabled
- IErrors.Err_DeviceIoControl
- IErrors.Err_MC_Mode

Beispiel	VB.NET	AN21XX.SetInterface(0, 0, 1) ...	` Interface = 0, Setting = 1
	C#	AN21XX.SetInterface(0, 0, 1);	// Interface = 0, Setting = 1

DownloadBinFile (lädt Firmware aus einem Binärfile in den Controller und startet diese)

DownloadHexFile (lädt Firmware aus einem Hexadezimal-File in den Controller und startet diese)

DownloadBinFile (DevNumber, FileName)

DownloadHexFile (DevNumber, FileName)

Übergabeparameter :

DevNumber	(System.Byte)	- Nummer des Controllers am Bus. Ist nur ein AN21xx am Bus, so muss immer 0 übergeben werden.
FileName	(System.String)	- Name und kompletter Pfad der Binär- bzw. Hexadezimaldatei

Mögliche Fehler :

- IErrors.Err_DeviceNumber
- IErrors.Err_IsDisabled
- IErrors.Err_DeviceIoControl
- IErrors.Err_MC_Mode
- IErrors.Err_DownloadFile

Beispiel	VB.NET	Dim FileName As String FileName = „C:\Probe.bin“ AN21XX.DownloadBinFile(0, FileName) Oder FileName = „C:\Probe.hex“ AN21XX.DownloadHexFile(0, FileName) ...
	C#	string FileName FileName = 'C:\Probe.bin'; AN21XX.DownloadBinFile(0, FileName); Oder FileName = 'C:\Probe.hex'; AN21XX.DownloadHexFile(0, FileName); ...

AbortPipe (bricht Transfer über Pipe ab)
ResetPipe (setzt Pipe zurück)

AbortPipe (DevNumber, PipeNumber)
ResetPipe (DevNumber, PipeNumber)

Übergabeparameter :

DevNumber (System.Byte) - Nummer des Controllers am Bus. Ist nur ein AN21xx am Bus, so muss immer 0 übergeben werden.

PipeNumber (System.Byte) - Nummer der gewählten Pipe (bei AN21XX : 0 .. 12)

Mögliche Fehler :
IError.Err_DeviceNumber
IError.Err_IsDisabled
IError.Err_MC_Mode

Beispiel	VB.NET	AN21XX.AbortPipe(0, 1) oder AN21XX.ResetPipe(0, 1)
	C#	AN21XX.AbortPipe(0, 4); oder AN21XX.ResetPipe(0, 4);

RAM_Write (schreibt ein Byte (einen Block) an eine bestimmte (ab einer bestimmten) Adresse des Code/Daten-RAM)

3 Überladungen

RAM_Write (DevNumber, RamAddr, DByte) // einzelnes Byte schreiben
RAM_Write (DevNumber, RamAddr, ByteArray) // Block schreiben

Übergabeparameter :

DevNumber (System.Byte) - Nummer des Controllers am Bus. Ist nur ein AN21xx am Bus, so muss immer 0 übergeben werden.

RamAddr (System.Int32) - RAM-Adresse in welche das Byte (der Block) geschrieben werden soll
Achtung ! Es kann nicht in alle Bereiche geschrieben werden (Controllertyp)

Data * (System.Byte) - Wert des zu schreibenden Bytes

Barray * (System.Byte-Array) - ByteArray enthält die zu schreibenden Daten
(Die zu schreibende Byte-Zahl entspricht der Grösse des Array's)

***) Ob ein Byte oder ein Block geschrieben wird richtet sich nach dem übergebenen Parameter (Byte oder Block)**

Mögliche Fehler :
IError.Err_DeviceNumber
IError.Err_IsDisabled
IError.Err_DeviceIoControl
IError.Err_MC_Mode
IError.Err_Pointer - zulässige Werte 0 .. 65535
IError.Err_Array - 2 .. 65535 Bytes zulässig

Beispiel	VB.NET	Dim Data As Byte Dim Barray(44) As Byte Dim i As Byte Data = 12 For i = 0 to 44 Barray(i) = i Next i AN21XX.RAM_Write (0, &H20, Data) ' Byte Data an RAM-Adresse 20h AN21XX.RAM_Write (0, &H20, Barray) ' 45 Byte ab RAM-Adresse 20h
	C#	byte Data = 12; byte i; byte[] Barray = new byte[45]; for (i = 0, i < 45, i++) Barray[i] = i; AN21XX.RAM_Write (0, 0x020, Data); // Byte Data an RAM-Adresse 20h AN21XX.RAM_Write (0, 0x020, Barray); // 45 Byte ab RAM-Adresse 20h

RAM_Read (liest ein Byte (einen Block) an eine bestimmte (ab einer bestimmten) Adresse des Code/Daten-RAM)

3 Überladungen

```
RAM_Read ( DevNumber, RamAddr)           // einzelnes Byte lesen
RAM_Read ( DevNumber, RamAddr, ByteArray) // Block lesen
```

Rückgabewert :

Data* (System.Byte) - **die Methode zum Lesen eines Blockes besitzt keinen Rückgabewert (void)**

Übergabeparameter :

DevNumber (System.Byte) - Nummer des Controllers am Bus. Ist nur ein AN21xx am Bus, so muss immer 0 übergeben werden.

RamAddr (System.Int32) - RAM-Adresse aus welcher das Byte bzw. ab welcher der Block gelesen werden soll

BArray * (System.Byte-Array) - ByteArray erhält die gelesenen Daten
(Die zu lesende Byte-Zahl entspricht der Grösse des Array's)

***) Ob ein Byte oder ein Block geschrieben wird richtet sich nach den Parametern der Methode**

Mögliche Fehler :

- IErrror.Err_DeviceNumber
- IErrror.Err_IsDisabled
- IErrror.Err_DeviceIoControl
- IErrror.Err_MC_Mode
- IErrror.Err_Pointer - zulässige Werte 0 .. 65535
- IErrror.Err_Array - 2 .. 65535 Bytes zulässig

Beispiel	VB.NET
	<pre>Dim Data As Byte Dim BArray(44) As Byte Data = AN21XX.RAM_Read (0, &H20) ' Byte aus RAM-Adresse 20h lesen AN21XX.RAM_Read (0, &H20, BArray) ' 45 Byte ab RAM-Adresse 20h lesen</pre>
	<pre>C# byte[] BArray = new byte[45]; byte Data = AN21XX.RAM_Read (0, 0x020); // Byte aus RAM-Adresse 20h lesen AN21XX.RAM_Read (0, 0x020, ref BArray); // 45 Byte ab RAM-Adresse 20h lesen</pre>

Bulk_Write (schreibt ein Byte (einen Block) im Bulk-Transfermodus)

1 Überladung

Bulk_Write (DevNumber, PipeNum, DByte) // einzelnes Byte schreiben
Bulk_Write (DevNumber, PipeNum, ByteArray) // Block schreiben

Übergabeparameter :

DevNumber	(System.Byte)	- Nummer des Controllers am Bus. Ist nur ein AN21xx am Bus, so muss immer 0 übergeben werden.
PipeNum	(System.Byte)	- Pipe-Nummer über welche der Transfer stattfinden soll - zulässig 0 .. 12
Data *	(System.Byte)	- Wert des zu schreibenden Bytes
Barray *	(System.Byte-Array)	- ByteArray enthält die zu schreibenden Daten (Die zu schreibende Byte-Zahl entspricht der Grösse des Array's)

***) Ob ein Byte oder ein Block geschrieben wird richtet sich nach dem übergebenen Parameter (Byte oder Block)**

Mögliche Fehler :

IErrror.Err_DeviceNumber	
IErrror.Err_IsDisabled	
IErrror.Err_DeviceIoControl	
IErrror.Err_MC_Mode	
IErrror.Err_Array	- 2 .. 65535 Bytes zulässig

Beispiel	VB.NET	<pre>Dim Data As Byte Dim BArray(44) As Byte Dim i As Byte Data = 12 For i = 0 to 44 BArray(i) = i Next i AN21XX.Bulk_Write (0, 2, Data) ' Byte Data im Bulk-Transfer schreiben AN21XX.Bulk_Write (0, 2, BArray) ' 45 Byte im Bulk-Transfer schreiben</pre>
	C#	<pre>byte Data = 12; byte i; byte[] BArray = new byte[45]; for (i = 0, i < 45, i++) BArray[i] = i; AN21XX.Bulk_Write (0, 2, Data); // Byte Data im Bulk-Transfer schreiben AN21XX.Bulk_Write (0, 2, BArray); // 45 Byte im Bulk-Transfer schreiben</pre>

Bulk_Read (liest ein Byte (einen Block) im Bulk-Transfermodus)

1 Überladung

Bulk_Read (DevNumber, PipNum) // einzelnes Byte lesen
Bulk_Read (DevNumber, PipeNum, ByteArray) // Block lesen

Rückgabewert :

Data* (System.Byte) - die Methode zum Lesen eines Blockes besitzt keinen Rückgabewert (**void**)

Übergabeparameter :

DevNumber (System.Byte) - Nummer des Controllers am Bus. Ist nur ein AN21xx am Bus, so muss immer 0 übergeben werden.

PipeNum (System.Byte) - Pipe, über welche der Transfer stattfinden soll – zulässig 0 .. 12

Barray * (System.Byte-Array) - Array erhält die gelesenen Daten
(Die zu lesende Byte-Zahl entspricht der Grösse des Array's)

***) Ob ein Byte oder ein Block gelesen wird richtet sich nach den Parametern der Methode**

Mögliche Fehler :
IError.Err_DeviceNumber
IError.Err_IsDisabled
IError.Err_DeviceIoControl
IError.Err_MC_Mode
IError.Err_Array - 2 .. 65535 Bytes zulässig

Beispiel	VB.NET
	<pre>Dim Data As Byte Dim Aarray(44) As Byte Data = AN21XX.Bulk_Read (0, 1) ` Byte Data im Bulk-Transfer lesen AN21XX.Bulk_Read ( 0, 1, BArray )         ` Block im Bulk-Transfer lesen</pre>
	<pre>C# byte[] BArray = new byte[45]; byte Data = AN21XX.Bulk_Read (0, 1); // Byte Data im Bulk-Transfer lesen AN21XX.Bulk_Read (0, 1, ref BArray); // Block im Bulk-Transfer lesen</pre>

2.5 Methoden des Parallelbus für die MC_Modi IMode_1 ... IMode_5

Um ein unnötiges aufblähen der Dokumentation zu verhindern werden die Methoden des Parallelbus komplett behandelt. Hier noch einmal die Unterschiede zwischen den einzelnen Modi :

IMode_1	- AN2131 (180 kByte/s im Blocktransfer) - mit Adressbus an Port C
IMode_2	- AN2131 (180 kByte/s im Blocktransfer) - ohne Adressbus an Port C (PortC frei konfigurierbar)
IMode_3	- AN2135 (1 MByte/s im Blocktransfer) - mit Adressbus an Port C
IMode_4	- AN2135 (1 MByte/s im Blocktransfer) - ohne Adressbus an Port C (PortC frei konfigurierbar)
IMode_5	- AN2131 (180 kByte/s im Blocktransfer) - mit gemultiplexten Adressbus an PortB (wird durch ALE-Signal / PortC Pin7 übernommen) - #RD und #WR müssen auf den #DACK-Eingang (PortC Pin6) zurückgeführt werden. Bei Schaltungen die schnell genug sind, kann der Eingang #DACK über einen Widerstand mit GND verbunden werden!

Auf den Parallelbus wird über die Methoden „**PB_Read**“ und „**PB_Write**“ zugegriffen. Diese Methoden besitzen jeweils 3 Überladungen, welche folgende Zugriffe abdecken :

PB_Read	- ein Byte vom Parallel-Bus lesen - ein Byte vom Parallel-Bus lesen - einen Block vom Parallel-Bus lesen - einen Block vom Parallel-Bus lesen	(incl. Adress-Bus) (ohne Adressbus – PortC frei verfügbar) (incl. Adress-Bus) (ohne Adressbus – PortC frei verfügbar)
PB_Write	- ein Byte zum Parallel-Bus schreiben - ein Byte zum Parallel-Bus schreiben - einen Block zum Parallel-Bus schreiben - einen Block zum Parallel-Bus schreiben	(incl. Adress-Bus) (ohne Adressbus – PortC frei verfügbar) (incl. Adress-Bus) (ohne Adressbus – PortC frei verfügbar)

Achtung ! Nur im Modus „IMode_1“

Ausschliesslich im Controller-Modus „IMode_1“ stehen die Methoden „**PB_ReadInc**“ und „**PB_WriteInc**“ zur Verfügung. Beide unterscheiden sich von den ansonsten identischen Blockübertragungs-Methoden „PB_Read“ und „PB_Write“ dadurch dass nach jedem erfolgten Lese- bzw, Schreibzugriff der Adress-Bus inkrementiert wird. Die Anfangsadresse ist dabei die an die Methode übergebene Adresse.

PB_WriteInc	- einen Block zum Parallel-Bus schreiben	(incl. inkrementierenden Adress-Bus)
PB_ReadInc	- einen Block vom Parallel-Bus lesen	(incl. Inkrementierenden Adress-Bus)

PB_Read (liest ein Byte (einen Block) vom Parallel-Bus)

3 Überladungen

PB_Read (DevNumber, Addr)	// einzelnes Byte lesen	(incl. Adressbus)
PB_Read (DevNumber)	// einzelnes Byte lesen	(ohne Adressbus)
PB_Read (DevNumber, Addr, ByteArray)	// Block lesen	(incl. Adressbus)
PB_Read (DevNumber, ByteArray)	// Block lesen	(ohne Adressbus)

Rückgabewert :

Data* (System.Byte) - die Methode zum Lesen eines Blockes besitzt keinen Rückgabewert (**void**)

Übergabeparameter :

DevNumber	(System.Byte)	- Nummer des Controllers am Bus. Ist nur ein AN21xx am Bus, so muss immer 0 übergeben werden.
Addr	(System.Byte)	- auszugebende Peripherie-Adresse (an PortC o. gemultiplext an PortB im IMode_5) Nicht in den Modi „IMode_2“ und „IMode_4“

Barray * (System.Byte-Array) - Array erhält die gelesenen Daten
(Die zu lesende Byte-Zahl entspricht der Grösse des Array's)

***) Ob ein Byte oder ein Block geschrieben wird richtet sich nach den Parametern der Methode**

Mögliche Fehler :	IError.Err_DeviceNumber	
	IError.Err_IsDisabled	
	IError.Err_DeviceIoControl	
	IError.Err_MC_Mode	
	IError.Err_Array	- 2 .. 16777215 Bytes zulässig

Beispiel	VB.NET	<pre>Dim Data As Byte Dim Addr As Byte Dim BArray(44) As Byte Addr = 144 Data = AN21XX.PB_Read (0, Addr) Data = AN21XX.PB_Read (0) AN21XX.PB_Read (0, Addr, BArray) AN21XX.PB_Read (0, BArray)</pre>	<pre>` Peripherie-Adresse = 144 ` Byte vom Parallel-Bus lesen ` Byte vom Parallel-Bus lesen ` 45 Byte vom Parallel-Bus lesen ` 45 Byte vom Parallel-Bus lesen</pre>
	C#	<pre>byte[] BArray = new byte[45]; byte Addr; Addr = 144; byte Data = AN21XX.PB_Read (0, Addr); byte Data = AN21XX.PB_Read (0); AN21XX.PB_Read (0, Addr, ref BArray); AN21XX.PB_Read (0, ref BArray);</pre>	<pre>// Peripherie-Adresse = 144 // Byte vom Parallel-Bus lesen // Byte vom Parallel-Bus lesen // 45 Byte vom Parallel-Bus lesen // 45 Byte vom Parallel-Bus lesen</pre>

PB_Write (schreibt ein Byte (einen Block) zum Parallel-Bus)

3 Überladungen

PB_Write (DevNumber, Addr, Data)	// einzelnes Byte schreiben	(incl. Adressbus)
PB_Write (DevNumber, Data)	// einzelnes Byte schreiben	(ohne Adressbus)
PB_Write (DevNumber, Addr, ByteArray)	// Block schreiben	(incl. Adressbus)
PB_Write (DevNumber, ByteArray)	// Block schreiben	(ohne Adressbus)

Übergabeparameter :

DevNumber	(System.Byte)	- Nummer des Controllers am Bus. Ist nur ein AN21xx am Bus, so muss immer 0 übergeben werden.
Addr	(System.Byte)	- auszugebende Peripherie-Adresse (an PortC o. gemultiplext an PortB im IMode_5) Nicht in den Modi „IMode_2“ und „IMode_4“
Data*	(System.Byte)	- zu schreibendes Byte
BArray *	(System.Byte-Array)	- Array erhält die zu schreibenden Daten (Die zu schreibende Byte-Zahl entspricht der Grösse des Array's)

*) **Ob ein Byte oder ein Block geschrieben wird richtet sich nach den Parametern der Methode**

Mögliche Fehler :

IError.Err_DeviceNumber	
IError.Err_IsDisabled	
IError.Err_DeviceIoControl	
IError.Err_MC_Mode	
IError.Err_Array	- 2 .. 16777215 Bytes zulässig

Beispiel	VB.NET	<pre>Dim Data As Byte Dim Addr As Byte Dim i As Byte Dim BArray(44) As Byte For i = 0 to 44 BArray(i) = i Next i Addr = 144 ` Peripherie-Adresse = 144 Data = 234 AN21XX.PB_Write ( 0, Addr, Data )             ` Byte zum Parallel-Bus schreiben AN21XX.PB_Write (0, Data) ` Byte zum Parallel-Bus schreiben AN21XX.PB_Write ( 0, Addr, BArray )           ` 45 Byte zum Parallel-Bus schreiben AN21XX.PB_Write (0, BArray) ` 45 Byte zum Parallel-Bus schreiben</pre>
	C#	<pre>byte[] BArray = new byte[45]; byte Addr; byte Data; byte i; for (i = 0, i < 45, i++) BArray[i] = i; Addr = 144; // Peripherie-Adresse = 144 Data = 234; // zu schreibendes Byte = 234 AN21XX.PB_Write (0, Addr, Data); // Byte schreiben AN21XX.PB_Write (0, Data); // Byte schreiben AN21XX.PB_Write (0, Addr, BArray); // 45 Byte schreiben AN21XX.PB_Write (0, BArray); // 45 Byte schreiben</pre>

2.6 Allgemeine Methoden des I2C-Bus

Hinweis : Diese Methoden sind in allen MC_Modi ausser dem Modus „CommonMode“ verfügbar !

I2C_Read (liest ein Byte (einen Block) vom I2C-Bus)

1 Überladung

```
I2C_Read ( DevNumber, SlaveAddr)           // einzelnes Byte lesen  
I2C_Read ( DevNumber, SlaveAddr, ByteArray) // Block lesen
```

Rückgabewert :

Data* (System.Byte) - **die Methode zum Lesen eines Blockes besitzt keinen Rückgabewert (void)**

Übergebeparameter :

DevNumber (System.Byte) - Nummer des Controllers am Bus. Ist nur ein AN21xx am Bus, so muss immer 0 übergeben werden.

SlaveAddr (System.Byte) - Slave-Adresse des Bausteines am I²C-Bus

Barray * (System.Byte-Array) - Array erhält die gelesenen Daten
(Die zu lesende Byte-Zahl entspricht der Grösse des Array's)

***) Ob ein Byte oder ein Block gelesen wird richtet sich nach den Parametern der Methode**

Mögliche Fehler :

IErrror.Err_DeviceNumber	
IErrror.Err_IsDisabled	
IErrror.Err_DeviceIoControl	
IErrror.Err_MC_Mode	
IErrror.Err_Array	- 2 .. 16777215 Bytes zulässig
IErrror.Err_AI2CNotAck	I2C-Bus NotAcknowledge nach Slaveadresse
IErrror.Err_DI2CNotAck	I2C-Bus NotAcknowledge nach Datum
IErrror.Err_I2CBus	allgemeiner I2C-Busfehler

Beispiel	VB.NET	<pre>Dim Data As Byte Dim SlaveAddr As Byte Dim BArray(44) As Byte SlaveAddr = 160 Data = AN21XX.I2C_Read (0, SlaveAddr) ' Adresse des Bausteines am I2C_Bus AN21XX.I2C_Read (0, SlaveAddr, BArray) ' Byte vom I2C-Bus lesen ' 45 Byte vom I2C-Bus lesen</pre>
	C#	<pre>byte[] BArray = new byte[45]; byte SlaveAddr; SlaveAddr = 160 // Slave-Adresse = 160 byte Data = AN21XX.I2C_Read (0, SlaveAddr); // Byte vom I2C-Bus lesen AN21XX.I2C_Read (0, SlaveAddr, ref BArray); // 45 Byte vom I2C-Bus lesen</pre>

I2C_Write (schreibt ein Byte (einen Block) zum I2C-Bus)

1 Überladung

I2C_Write (DevNumber, SlaveAddr, Data) // einzelnes Byte schreiben
I2C_Write (DevNumber, SlaveAddr, ByteArray) // Block schreiben

Übergabeparameter :

DevNumber	(System.Byte)	- Nummer des Controllers am Bus. Ist nur ein AN21xx am Bus, so muss immer 0 übergeben werden.
SlaveAddr	(System.Byte)	- Slave-Adresse des Bausteines am I2C-Bus
Data*	(System.Byte)	- auf den I2C-Bus zu schreibendes Byte
Barrray *	(System.Byte-Array)	- Array enthält die zu schreibenden Daten (Die zu schreibende Byte-Zahl entspricht der Grösse des Array's)

***) Ob ein Byte oder ein Block geschrieben wird richtet sich nach den übergebenen Parametern der Methode**

Mögliche Fehler :	IError.Err_DeviceNumber	
	IError.Err_IsDisabled	
	IError.Err_DeviceIoControl	
	IError.Err_MC_Mode	
	IError.Err_Array	- 2 .. 65535 Bytes zulässig
	IError.Err_AI2CNotAck	I2C-Bus NotAcknowledge nach Slaveadresse
	IError.Err_DI2CNotAck	I2C-Bus NotAcknowledge nach Datum
	IError.Err_I2CBus	allgemeiner I2C-Busfehler

Beispiel	VB.NET	<pre>Dim Data As Byte Dim SlaveAddr As Byte Dim i As Byte Dim BArray(44) As Byte For i = 0 to 44 BArray(i) = i Next i SlaveAddr = 160 Data = 234 AN21XX.PB_Write (0, SlaveAddr, Data) AN21XX.PB_Write (0, SlaveAddr, BArray)</pre>	<pre>` Adresse des IC am I2C_Bus ` zu schreibendes Datenbyte ` Byte zum I2C-Bus schreiben ` 45 Byte zum I2C-Bus schreiben</pre>
	C#	<pre>byte[] BArray = new byte[45]; Byte Addr; byte Data; byte i; for (i = 0, i < 45, i++) BArray[i] = i; SlaveAddr = 160; Data = 234; AN21XX.PB_Write (0, SlaveAddr, Data); AN21XX.PB_Write (0, SlaveAddr, BArray);</pre>	<pre>// Adresse des IC am I2C_Bus // zu schreibendes Byte = 234 // Byte zum I2C-Bus schreiben // 45 Byte zum I2C-Bus schreiben</pre>

2.7 Methoden zur Nutzung von I2C-EEPROM's

Hinweis : Diese Methoden sind in allen MC_Modi ausser dem Modus „CommonMode“ verfügbar !

Allgemeines : Unterstützt werden die EEPROM's 24LC00 bis 24LC256 oder funktionskompatible Typen. Unabhängig von der Page-Grösse des jeweiligen Types kann immer ein beliebiger Block gelesen oder geschrieben werden. Das Aufteilen des Blockes und das Schreiben der einzelnen Pages (Bytes) wird dabei von der jeweiligen Methode verwaltet. Gleiches gilt für Leseoperationen.

Durch Nutzung von **Acknowledge-Polling** wird immer die minimal mögliche Schreibzeit erreicht.

Die Klasse „**EEPTyp**“ enthält Konstanten für jeden unterstützten EEPROM-Typ. Nutzen Sie bei der Anwendung der Methoden bitte diese Konstanten zu Parameterübergabe. Die Verfahrensweise wird Bei den Beispielen zu den einzelnen Methoden sichtbar.

Zustandekommen der Fehler „IError.Err_Array“ und „IError.Err_Pointer“ :

IError.Err_Pointer : Wird als Pointer ein Wert kleiner 0 übergeben, oder wird ein Wert übergeben welcher die Speichertiefe des betreffenden EEPROM's überschreitet, so wird der Eigenschaft **I_Error** Fehler **IError.Err_Pointer** zugewiesen .

IError.Err_Array : Wird ein Array übergeben, welches weniger als zwei Elemente besitzt, oder bei dem übergebenen Pointer plus der Array-Grösse ein Überlauf des internen Adresszählers des EEPROM's auftreten, so wird der Eigenschaft „**I_Error**“ der Fehler „**IError.ErrArray**“ zugewiesen.

EEP_Read (liest ein Byte (einen Block) aus einem I2C-EEPROM)

1 Überladung

```
EEP_Read ( DevNumber, EEPTyp, SlaveAddr, EEP_Pointer)           // einzelnes Byte lesen  
EEP_Read ( DevNumber, EEPTyp, SlaveAddr, EEP_Pointer, ByteArray) // Block lesen
```

Rückgabewert :

Data* (System.Byte) - die Methode zum Lesen eines Blockes besitzt keinen Rückgabewert (**void**)

Übergabeparameter :

DevNumber (System.Byte) - Nummer des Controllers am Bus. Ist nur ein AN21xx am Bus, so muss immer 0 übergeben werden.

EEPTyp (System.Byte) - Konstanten für alle implementierten EEP-Typen befinden sich in der Klasse „EEPTyp“

SlaveAddr (System.Byte) - Slave-Adresse des Bausteines am I²C-Bus

EEP_Pointer (System.Int32) - Adresse der Speicherstelle im EEPROM aus welcher (ab welcher) gelesen werden soll.

Barrray * (System.Byte-Array) - Array erhält die gelesenen Daten
(Die zu lesende Byte-Zahl entspricht der Grösse des Array's)

***) Ob ein Byte oder ein Block gelesen wird richtet sich nach den Parametern der Methode**

Mögliche Fehler :

IErrror.Err_DeviceNumber	
IErrror.Err_IsDisabled	
IErrror.Err_DeviceIoControl	
IErrror.Err_MC_Mode	
IErrror.Err_Pointer	
IErrror.Err_Array	
IErrror.Err_AI2CNotAck	I2C-Bus NotAcknowledge nach Slaveadresse
IErrror.Err_DI2CNotAck	I2C-Bus NotAcknowledge nach Datum
IErrror.Err_I2CBus	allgemeiner I2C-Busfehler

```
Beispiel  VB.NET  Dim Data As Byte  
              Dim SlaveAddr As Byte  
              Dim BArray(44) As Byte  
  
              ' Adresse des EEPROM am I2C_Bus  
              SlaveAddr = 160  
              ' Byte an Adresse 10h des I2C-EEPROM 24C64 lesen  
              Data = AN21XX.EEP_Read ( 0, EEPTyp.C64, SlaveAddr, &H10)  
              ' 45 Byte ab Adresse 10h des I2C-EEPROM 24C64 lesen  
              AN21XX.EEP_Read ( 0, EEPTyp.C64, SlaveAddr, &H10, BArray )
```

```
C#  byte[] BArray = new byte[45];  
    byte SlaveAddr;  
  
    // Adresse des EEPROM am I2C_Bus  
    SlaveAddr = 160;  
    // Byte an Adresse 10h des I2C-EEPROM 24C64 lesen  
    byte Data = AN21XX.I2C_Read ( 0, EEPTyp.C64, SlaveAddr, 0x10);  
    // 45 Byte ab Adresse 10h des I2C-EEPROM 24C64 lesen  
    AN21XX.I2C_Read ( 0, EEPTyp.C64, SlaveAddr, 0x10, ref BArray )
```

EEP_Write (schreibt ein Byte (einen Block) in ein I2C-EEPROM)

1 Überladung

```
EEP_Write ( DevNumber, EEPTyp, SlaveAddr, EEP_Pointer, Data)           // einzelnes Byte schreiben
EEP_Write ( DevNumber, EEPTyp, SlaveAddr, EEP_Pointer, ByteArray)      // Block schreiben
```

Übergabeparameter :

DevNumber	(System.Byte)	- Nummer des Controllers am Bus. Ist nur ein AN21xx am Bus, so muss immer 0 übergeben werden.
EEPTyp	(System.Byte)	- Konstanten für alle implementierten EEP-Typen befinden sich in der Klasse „EEPTyp“
SlaveAddr	(System.Byte)	- Slave-Adresse des Bausteines am I ² C-Bus
EEP_Pointer	(System.Int32)	- Adresse der Speicherstelle im EEPROM aus welcher (ab welcher) gelesen werden soll.
Data*	(System.Byte)	- in das EEPROM zu schreibendes Datenbyte
BArray*	(System.Byte-Array)	- Array enthält die zu schreibende Daten (Die zu schreibendes Byte-Zahl entspricht der Grösse des Array's)

***) Ob ein Byte oder ein Block geschrieben wird richtet sich nach den übergebenen Parametern der Methode**

Mögliche Fehler :	IErrror.Err_DeviceNumber	
	IErrror.Err_IsDisabled	
	IErrror.Err_DeviceIoControl	
	IErrror.Err_MC_Mode	
	IErrror.Err_Pointer	
	IErrror.Err_Array	
	IErrror.Err_AI2CNotAck	I2C-Bus NotAcknowledge nach Slaveadresse
	IErrror.Err_DI2CNotAck	I2C-Bus NotAcknowledge nach Datum
	IErrror.Err_I2CBus	allgemeiner I2C-Busfehler

```
Beispiel    VB.NET    Dim Data As Byte
                Dim SlaveAddr As Byte
                Dim i As Byte
                Dim BArray(44) As Byte

                ' zu schreibendes Array initialisieren
                For i = 0 to 44
                    BArray(i) = i
                Next i
                ' Adresse des EEPROM am I2C_Bus
                SlaveAddr = 160
                ' zu schreibendes Daten-Byte
                Data = 183
                ' Byte an Adresse 10h des I2C-EEPROM 24C64 schreiben
                AN21XX.EEP_Write ( 0, EEPTyp.C64, SlaveAddr, &H10, Data)
                ' 45 Byte ab Adresse 10h des I2C-EEPROM 24C64 lesen
                AN21XX.EEP_Write ( 0, EEPTyp.C64, SlaveAddr, &H10, BArray )
```

```
C#    byte[] BArray = new byte[45];
        byte SlaveAddr; byte Data;

        // zu schreibendes Array initialisieren
        for ( i = 0, i < 45, i++ ) BArray[i] = i;
        // Adresse des EEPROM am I2C_Bus
        SlaveAddr = 160;
        // zu schreibendes Daten-Byte
        Data = 183
        // Data an Adresse 10h des I2C-EEPROM 24C64 lesen
        AN21XX.I2C_Read ( 0, EEPTyp.C64, SlaveAddr, 0x10, Data);
        // 45 Byte ab Adresse 10h des I2C-EEPROM 24C64 lesen
        AN21XX.I2C_Read ( 0, EEPTyp.C64, SlaveAddr, 0x10, BArray )
```

2.8 Methoden des MC-Mode „SimpleMode“

Hinweis : Die Methoden des PortC sind zusätzlich in den Modi „IMode_2“ und „IMode_4“ zugänglich !
In diesen Modi ist der PortC des Controllers frei verfügbar (kein Adressbus)
Im IMode_5 sind die Pins PC0 – PC5 frei verfügbar. Die Pins 6 und 7 werden als #DACK und ALE genutzt !

Die Klasse „**Ports**“ enthält Konstanten für die Ports A - C.

Die Klasse „**Pins**“ enthält Konstanten für die Pins Pin0 – Pin7.

Nutzen Sie bei der Anwendung der Methoden bitte diese Konstanten zu Parameterübergabe.
Die Verfahrensweise wird bei den Beispielen zu den einzelnen Methoden sichtbar.

SetPortDir

Diese Methode setzt die Übertragungsrichtung der einzelnen Pins eines Port's.
Ein gesetztes Bit entspricht dabei einem als Ausgang konfigurierten Pin.

SetPortDir (DevNumber, Port, Pins)

Übergabeparameter :

DevNumber	(System.Byte)	- Nummer des Controllers am Bus. Ist nur ein AN21xx am Bus, so muss immer 0 übergeben werden.
Port	(System.Byte)	- Port, dessen Datenrichtung geändert werden soll Siehe Konstanten der Klasse „Ports“ !
Pins	(System.Byte)	- ein gesetztes Bit in diesem Parameter schaltet das betreffende Pin als Ausgang Siehe Konstanten der Klasse „Pins“ !

Mögliche Fehler :

- IEError.Err_DeviceNumber
- IEError.Err_IsDisabled
- IEError.Err_DeviceIoControl
- IEError.Err_MC_Mode

```
Beispiel    VB.NET    Dim PortDir As Byte

                ' Variable für Richtungsregister initialisieren
                ' Pin0, Pin4 und Pin7 als Ausgang
                PortDir = 145
                ' oder so :
                PortDir = Pins.Pin7 Or Pins.Pin4 Or Pins.Pin0

                ' Richtungsregister von PortC setzen
                AN21XX.SetPortDir ( 0, Ports.PortC, PortDir )
```

```
C#            byte PortDir;

                // Variable für Richtungsregister initialisieren
                // Pin0, Pin4 und Pin7 als Ausgang
                PortDir = 145;
                // oder so :
                PortDir = Pins.Pin7 | Pins.Pin4 | Pins.Pin0;

                // Richtungsregister von PortC setzen
                AN21XX.SetPortDir ( 0, Ports.PortC, PortDir );
```

SetPort

Diese Methode setzt oder löscht Pins eines Port's.
Ein gesetztes Bit entspricht dabei einem auf High gestzten Pin .

SetPort (DevNumber, Port, Pins)

Übergabeparameter :

DevNumber	(System.Byte)	- Nummer des Controllers am Bus. Ist nur ein AN21xx am Bus, so muss immer 0 übergeben werden.
Port	(System.Byte)	- Port, dessen Pins gesetzt oder gelöscht werden soll Siehe Konstanten der Klasse „Ports“ !
Pins	(System.Byte)	- ein gesetztes Bit in diesem Parameter entspricht dabei einem auf High gestzten Pin. Siehe Konstanten der Klasse „Pins“ !
Mögliche Fehler :		IErrror.Err_DeviceNumber IErrror.Err_IsDisabled IErrror.Err_DeviceIoControl IErrror.Err_MC_Mode

Beispiel	VB.NET	
	<pre>Dim PortOut As Byte ' Variable für Port-Ausgangsregister initialisieren ' Pin0, Pin4 und Pin7 = H-Pegel, alle anderen Low-Pegel PortOut = 145 ' oder so : PortOut = Pins.Pin7 Or Pins.Pin4 Or PinsPin0 ' Ausgangsregister von PortC setzen AN21XX.SetPort (0, Ports.PortC, PortOut)</pre>	
	C#	<pre>byte PortOut; // Variable für Port-Ausgangsregister initialisieren // Pin0, Pin4 und Pin7 = H-Pegel, alle anderen Low-Pegel PortOut = 145; // oder so : PortOut = Pins.Pin7 Pins.Pin4 PinsPin0; // Ausgangsregister von PortC setzen AN21XX.SetPort (0, Ports.PortC, PortOut);</pre>

Die übergebenen Pegel werden an den Pins nur wirksam, wenn das entsprechende Pin mit der Methode „SetPortDir“
Als Ausgang konfiguriert wurde !

ReadPort

Diese Methode setzt setzt öder löscht Pins eines Port's.
Ein gesetztes Bit entspricht dabei einem auf High gestzten Pin .

ReadPort (DevNumber, Port)

Übergabeparameter :

DevNumber (System.Byte) - Nummer des Controllers am Bus. Ist nur ein AN21xx am Bus, so muss immer 0 übergeben werden.

Port (System.Byte) - Port, dessen Pins gelesen werden sollen
Siehe Konstanten der Klasse „Ports“ !

Mögliche Fehler :
IError.Err_DeviceNumber
IError.Err_IsDisabled
IError.Err_DeviceIoControl
IError.Err_MC_Mode

```
Beispiel    VB.NET    Dim PortIn As Byte

                ' Pin-Register von PortC lesen
                PortIn = AN21XX.SetPort ( 0, Ports.PortC, PortOut )
```

```
                C#      byte PortIn;

                // Pin-Register von PortC lesen
                PortIn = AN21XX.ReadPort ( 0, Ports.PortC );
```

Wurde ein Pin mit der Methode „**SetPortDir**“ als Ausgang konfiguriert, so wird der Wert des Ausgangsregisters des betreffenden Pins zurück gelesen .

SetPortPin

Diese Methode setzt ein Pin eines Port's.

SetPortPin (DevNumber, Port, Pin)

Übergabeparameter :

DevNumber (System.Byte) - Nummer des Controllers am Bus. Ist nur ein AN21xx am Bus, so muss immer 0 übergeben werden.

Port (System.Byte) - Port, dessen Pin gelöscht werden soll
Siehe Konstanten der Klasse „Ports“ !

Pin (System.Byte) - Pin welches gelöscht werden soll
Siehe Konstanten der Klasse „Pins“ !

Mögliche Fehler :
IError.Err_DeviceNumber
IError.Err_IsDisabled
IError.Err_DeviceIoControl
IError.Err_MC_Mode

```
Beispiel    VB.NET    ' Pin7 von PortC löschen
                AN21XX.ClearPortPin ( 0, Ports.PortC, Pins.Pin7 )
```

```
                C#      // Pin7 von PortC löschen
                AN21XX.ClearPortPin ( 0, Ports.PortC, Pins.Pin7 );
```

Ist ein Pin als Eingang konfiguriert so bleibt der Aufruf dieser Methode wirkungslos !

ClearPortPin

Diese Methode löscht ein Pin eines Port's.

ClearPortPin (DevNumber, Port, Pin)

Übergabeparameter :

DevNumber (System.Byte) - Nummer des Controllers am Bus. Ist nur ein AN21xx am Bus, so muss immer 0 übergeben werden.

Port (System.Byte) - Port, dessen Pin gelöscht werden soll
Siehe Konstanten der Klasse „Ports“ !

Pin (System.Byte) - Pin welches gelöscht werden soll
Siehe Konstanten der Klasse „Pins“ !

Mögliche Fehler :
IError.Err_DeviceNumber
IError.Err_IsDisabled
IError.Err_DeviceIoControl
IError.Err_MC_Mode

```
Beispiel    VB.NET    ` Pin7 von PortC löschen
                AN21XX.ClearPortPin ( 0, Ports.PortC, Pins.Pin7 )
```

```
                C#      // Pin7 von PortC löschen
                AN21XX.ClearPortPin ( 0, Ports.PortC, Pins.Pin7 );
```

Wurde ein Pin mit der Methode „SetPortDir“ als Eingang konfiguriert so bleibt der Aufruf dieser Methode wirkungslos !

ReadPortPin

Diese Methode gibt den Pegel eines Pins zurück. (High-Pegel = 1 / Low-Pegel = 0)

ReadPortPin (DevNumber, Port, Pin)

Rückgabewert :

Pegel (System.Byte)

Übergabeparameter :

DevNumber (System.Byte) - Nummer des Controllers am Bus. Ist nur ein AN21xx am Bus, so muss immer 0 übergeben werden.

Port (System.Byte) - Port, dessen Pin gelesen werden soll
Siehe Konstanten der Klasse „Ports“ !

Pin (System.Byte) - Pin welches gelesen werden soll
Siehe Konstanten der Klasse „Pins“ !

Mögliche Fehler :
IError.Err_DeviceNumber
IError.Err_IsDisabled
IError.Err_DeviceIoControl
IError.Err_MC_Mode

```
Beispiel    VB.NET    Dim PinLevel As Byte
                ` Pin3 von PortC lesen
                PinLevel = AN21XX.ReadPortPin ( 0, Ports.PortC, Pins.Pin3 )
```

```
                C#      // Pin3 von PortC lesen
                byte PinLevel = AN21XX. ReadPortPin ( 0, Ports.PortC, Pins.Pin3 );
```

Wurde ein Pin mit der Methode „SetPortDir“ als Ausgang konfiguriert, so wird der Wert des Ausgangsregisters des betreffenden Pins zurück gelesen !

2.9 Nutzung der externen Interrupts EX0 und EX1

Prinzipielles zur Nutzung der externen Interrupts des AN21XX :

Wie alle Controller mit 51'er Kern besitzt auch der AN2131 / 35 die externen Interrupts EX0 und EX1. Diese befinden sich am PortC des Controllers und können somit nur genutzt werden, wenn der PortC frei verfügbar ist. Das ist in den MC_Modi „IMode_2“, „IMode_4“, „IMode_5“ und „SimpleMode“ der Fall. Die Pins zum Auslösen eines Interrupts sind folgende :

PortC.2 (Pin2 des PortC) - EX0 zugehöriges Ereignis der Klasse [InterruptEX0](#)

PortC.3 (Pin3 des PortC) - EX1 zugehöriges Ereignis der Klasse [InterruptEX1](#)

Zum Nutzen der externen Interrupts stehen zwei Ereignisse eine Eigenschaft und zwei Methoden zur Verfügung :

- [InterruptEX0](#) - **Ereignis** wird ausgelöst, wenn an PortC.2 eine H/L-Flanke aufgetreten ist
[Siehe Abschnitt 2.2](#)
- [InterruptEX1](#) - **Ereignis** wird ausgelöst, wenn an PortC.3 eine H/L-Flanke aufgetreten ist
[Siehe Abschnitt 2.2](#)
- SetInterrupt - **Methode** aktiviert oder deaktiviert einen externen Interrupt
- IntRefreshTime - **Eigenschaft** legt fest, in welchem Intervall die Interrupt-Flags im Controller ausgelesen werden.
[Siehe Abschnitt 2.1](#)

Im folgenden wird das Prinzip des Interrupt-Handling sowie die softwaremässige Umsetzung beschrieben !

[Da Methoden, Ereignisse und das Testprogramm hier stark miteinander verwoben sind, wird von der bisherigen Gliederung der Dokumentation abgewichen und der gesamte Prozess im Zusammenhang erläutert !](#)

Der USB unterstützt Interrupts nicht im eigentlichen Sinn. Zwar gibt es einen Transfermodus der den Namen ‚Interrupt-Transfer‘ trägt, jedoch muss auch bei diesem der Host explizit den Transfer starten. Ob ein Interrupt aufgetreten ist muss also softwaremässig abgefragt werden. Die Abfrage geschieht in gleichmässigen Intervallen, deren Zeit über die Eigenschaft „**IntRefreshTime**“ festgelegt werden kann (siehe weiter unten). Nachdem ein Interrupt durch die Methode „**SetInterrupt**“ aktiviert (freigegeben) wurde, laufen bei Auftreten einer H/L-Flanke am entsprechenden Pin folgende Vorgänge ab :

EX0

- H/L-Flanke an Pin2 des PortC
- Flag wird im RAM des AN21XX gesetzt
- Entsprechend der eingestellten Refresh-Time wird das Flag durch die DLL ausgelesen.
- Bei gesetztem Flag wird dieses gelöscht und durch die Klasse das entsprechende Ereignis ausgelöst (in diesem Falle „[InterruptEX0](#)“)
- nach Ablauf der eingestellten ‚IntRefreshTime‘ wird eine erneute Abfrage gestartet.

Hinweis ! Dieses Beispiel zeigt, dass das Auftreten mehrerer Interrupts zwischen zwei Abfragen die gleiche Wirkung hat, wie das Auftreten eines einzigen. Es kann also nur gesagt werden, **dass** ein Interrupt zwischen zwei Abfragen aufgetreten ist und **nicht** wie viele !

Somit eignen sich die Interrupts auf jeden Fall zur Benachrichtigung eines Programme, dass ein bestimmter Zustand der Hardware erreicht wurde. Sie sind nicht geeignet um in kurzen Abständen auftretende Ereignisse an den Pin's zuverlässig an die Software zu melden.

SetInterrupt

Diese Methode aktiviert oder deaktiviert einen der beiden externen Interrupts

SetInterrupt (DevNumber, IntNum, State)

Übergabeparameter :

- | | | | |
|-----------|---------------|---|--|
| DevNumber | (System.Byte) | - Nummer des Controllers am Bus. Ist nur ein AN21xx am Bus, so muss immer 0 übergeben werden. | |
| IntNum | (System.Byte) | - Interrupt, der aktiviert oder deaktiviert werden soll | Siehe Konstanten der Klasse „Interrupt“ ! |
| State | (System.bool) | - Status des Interrupt - aktivieren = True | Siehe Konstanten der Klasse „Interrupt“ ! |

Mögliche Fehler :

- IErrors.Err_DeviceNumber
- IErrors.Err_IsDisabled
- IErrors.Err_DeviceIoControl
- IErrors.Err_MC_Mode
- IErrors.Err_InterruptNumber

```
Beispiel    VB.NET    ` Externen Interrupt 0 aktivieren
                AN21XX.SetInterrupt ( 0, Interrupt.EX0, Interrupt.IntEnabled )
```

```
                C#      // Externen Interrupt 0 aktivieren
                AN21XX.SetInterrupt ( 0, Interrupt.EX0, Interrupt.IntEnabled );
```

Einbindung der Interrupts in eigene Anwendungen :

Zuerst muss für beide Interrupts jeweils eine Methode bereitgestellt werden und diese mit den Ereignissen verbunden werden. Diese geschieht bei C# und VB.NET auf etwas unterschiedliche Weise . Während bei VB.NET das Ereignis einfach mit einer Methode verbunden wird, geschieht das bei C# über den öffentlichen Delegaten „Interrupt“ der Klasse clsAN21XX.

Dies sieht dann folgendermaßen aus :

Beispiel VB.NET

```
` Interruptroutine mit Ereignis verbinden - in FormLoad() eintragen
AddHandler AN21XX.InterruptEX0, AddressOf Interrupt_EX0
AddHandler AN21XX.InterruptEX1, AddressOf Interrupt_EX1

` Interruptroutine für externen Interrupt 0
Private Sub Interrupt_EX0(ByVal DevNum As Byte)
    ` Hier auszuführenden Code für EX0 eintragen
End Sub

` Interruptroutine für externen Interrupt 1
Private Sub Interrupt_EX1(ByVal DevNum As Byte)
    ` Hier auszuführenden Code für EX1 eintragen
End Sub

` Aktivieren beider Interrupts
AN21XX.SetInterrupt ( 0, Interrupt.EX1, Interrupt.IntEnabled )
AN21XX.SetInterrupt ( 0, Interrupt.EX0, Interrupt.IntEnabled )
```

C#

```
// Interruptroutine mit Ereignis verbinden
AN21XX.InterruptEX0 += new clsAN21XX.interrupt( Interrupt_EX0 );
AN21XX.InterruptEX1 += new clsAN21XX.interrupt( Interrupt_EX1 );

// Interruptroutine für externen Interrupt 0
private void Interrupt_EX0 ( byte DevNum ) {
    // Hier auszuführenden Code für EX0 eintragen
}

// Interruptroutine für externen Interrupt 0
private void Interrupt_EX1 ( byte DevNum ) {
    // Hier auszuführenden Code für EX1 eintragen
}

// Aktivieren beider Interrupts
AN21XX.SetInterrupt ( 0, Interrupt.EX1, Interrupt.IntEnabled );
AN21XX.SetInterrupt ( 0, Interrupt.EX0, Interrupt.IntEnabled );
```

Der Code, welcher beim Auftreten eines Interrupts ausgeführt werden soll wird in der Interruptroutine des entsprechenden Interrupts eingetragen.
(Interrupt_EX0 und Interrupt_EX1) **Nach dem Deaktivieren eines Interrupts ist das zugehörige Pin wieder frei verfügbar !**

Hinweise : Beachten Sie bitte, dass die Eigenschaft `IntRefreshTime`` nicht auf einen Controller beschränkt ist.
Wenn Sie diesen Wert ändern, wird der neue Wert für alle Controller wirksam .

Wurde ein Interrupt aktiviert, so haben Methoden zum Ändern des Pin-Status, wie `SetPortDir()`, `ClearPortPin()` und Ähnliche, auf den Interrupt-Pin keine Auswirkung !