

DLL zur Verwendung des Cypress USB-Controllers

AN2131 & AN2135 – Version 3.9

Copyright © 2004 by Karsten Böhme

Karsten Böhme Hard- und Softwareentwicklung

An der Bismarckhöhe 8

01737 Tharand

Tel : 035203 / 32758

Mobil : 0160 / 6763917

Fax : 069 / 1330 444 6109

Mail : karsten.boehme@arcor.de

Inhalt :

1. Allgemeines
 - 1.1 Die DLL
 - 1.2 Installation
 - 1.3 Anschlussbelegung des Controllers in den einzelnen Interface-Modi
2. Methoden der DLL
 - 2.1 Allgemeine Methoden der DLL
 - 2.2 Methoden des Interface-Mode „CommonMode“
 - 2.3 Methoden des Parallelbus für die MC_Modi IMode_1 ... IMode_5
 - 2.4 Allgemeine Methoden des I2C-Bus
 - 2.5 Methoden zur Nutzung von I2C-EEPROM's
 - 2.6 Methoden des MC-Mode „SimpleMode“
 - 2.7 Nutzung der externen Interrupts EX0 und EX1

1 Allgemeines

1.1 Die DLL (AN21XX_v39.DLL)

Die DLL (geschrieben in Delphi 5) wurde zur Verwendung der Cypress-USB-Controller AN2131 u. AN2135 entwickelt. Das gewählte Konzept erlaubt die Verwendung der DLL in Verbindung mit **Delphi**, **Visual Basic** und **C++**. Für alle drei Programmiersprachen existieren Testprogramme. Für C++ wurde eine Wrapper-Klasse geschrieben, welche auch dem C++-Programmierer einen einfachen Zugriff auf die umfangreiche Funktionsvielfalt der DLL ermöglicht.

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

Daten der DLL

- 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 DLL sofortige Nutzung aller Funktionen

Welche Funktionalität der Controller zur Verfügung stellt ist vom, an die Methode „SetDeviceEnabled“ 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 Port C für allgemeine Anwendung genutzt werden. Jedes Pin kann dabei wahlweise als Ein- oder Ausgang betrieben werden. Die Pins 2 und 3 können als Interruptpins genutzt werden. Genaue Erläuterungen dazu finden Sie im letzten Abschnitt dieser Dokumentation.

Bei Verwendung des **AN2135** sind Datenübertragungsraten von **1 MByte/s**¹ 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

¹ bei Nutzung der Blockübertragungsmethoden „PB_ReadBlock“ und „PB_WriteBlock“

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 „Braintechonlogy“ 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_DLL.zip' existiert das Verzeichnis ,AN21XX_DLL' mit folgenden Unterverzeichnissen :

\DLL	- enthält die DLL ,AN21XX_v39.DLL' -> ins Systemverzeichnis kopieren - enthält die DLL ,BRLNDMM.DLL' -> ins Windowsverzeichnis kopieren
\Doc	- enthält Dokumentation sowie Hinweis 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
\Delphi_Sample	- enthält Testprogramm für Delphi
\VB_Sample	- enthält Testprogramm für Visual Basic
\VC_Sample	- enthält Wrapper-Klasse für DLL sowie C++-Testprogramm
\BorlandCpp	- enthält Header-Datei sowie PDF-Datei für Borland C++ Builder

Zum Starten der Testprogramme müssen Laufzeitbibliotheken etc. der einzelnen Entwicklungsumgebungen auf dem Computer vorhanden sein ! Da die Testprogramme auch im Sourcecode vorliegen können sie direkt in der IDE geladen und ausgeführt 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

		I Mode_1 = 1	I Mode_2 = 2	I Mode_3 = 3	I Mode_4 = 4	I mode_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	A D0	frei verfügbar
PB1	Pin25	D1	D1	D1	D1	A D1	frei verfügbar
PB2	Pin26	D2	D2	D2	D2	A D2	frei verfügbar
PB3	Pin27	D3	D3	D3	D3	A D3	frei verfügbar
PB4	Pin28	D4	D4	D4	D4	A D4	frei verfügbar
PB5	Pin29	D5	D5	D5	D5	A D5	frei verfügbar
PB6	Pin30	D6	D6	D6	D6	A D6	frei verfügbar
PB7	Pin31	D7	D7	D7	D7	A D7	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 :

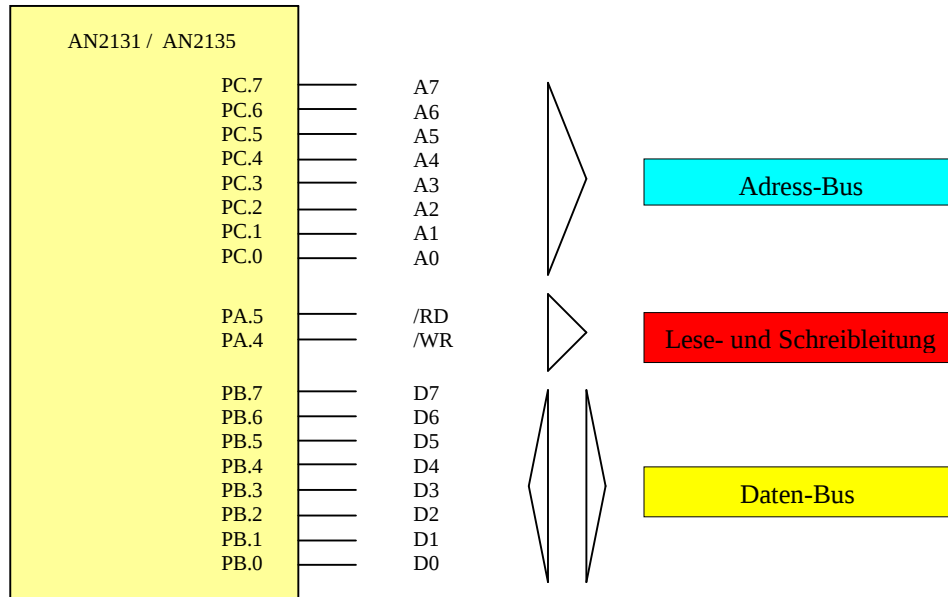
SDA = Pin35

SCL = Pin36

Hinweis ! Bei frei verfügbaren PortC können die Pins 2 und 3 dieses Ports als Interrupt-Pins genutzt werden. Diese Funktionalität ist in Abschnitt 2.7 der Dokumentation ausführlich beschrieben .

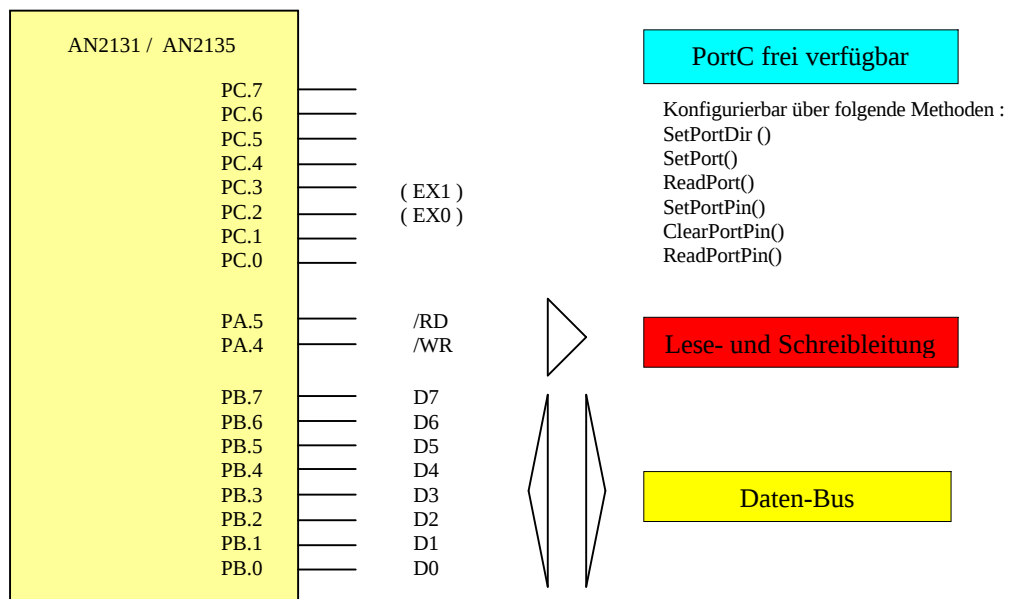
Betrieb der Controller in den Interface-Modi IMode_1 und IMode_3

IMode_1 = AN2131
 Imode_3 = AN2135



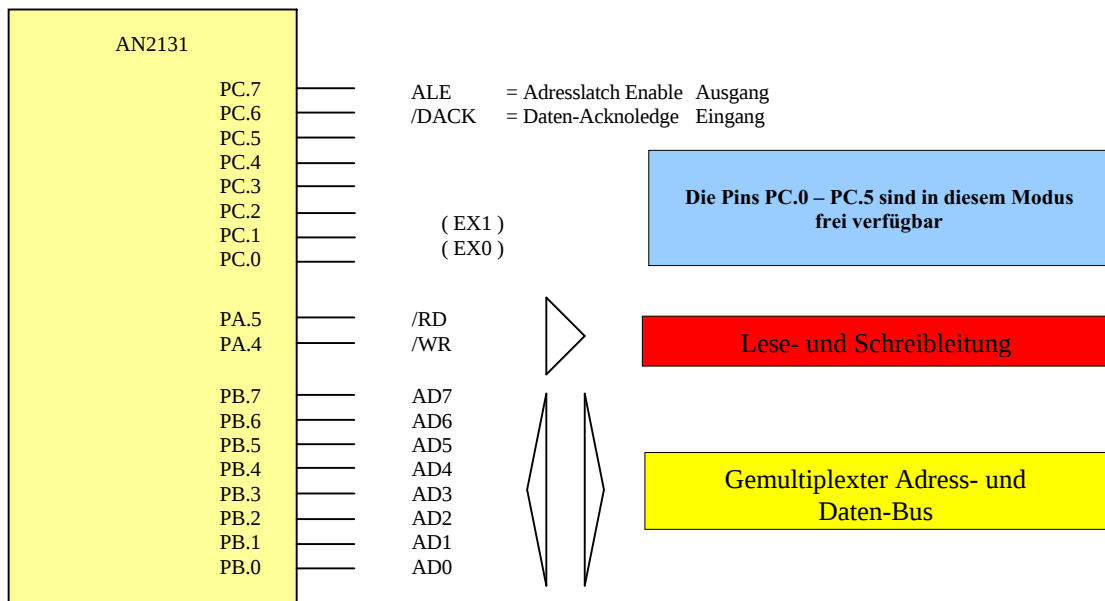
Betrieb der Controller in den Interface-Modi IMode_2 und IMode_4

IMode_2 = AN2131
 Imode 4 = AN2135

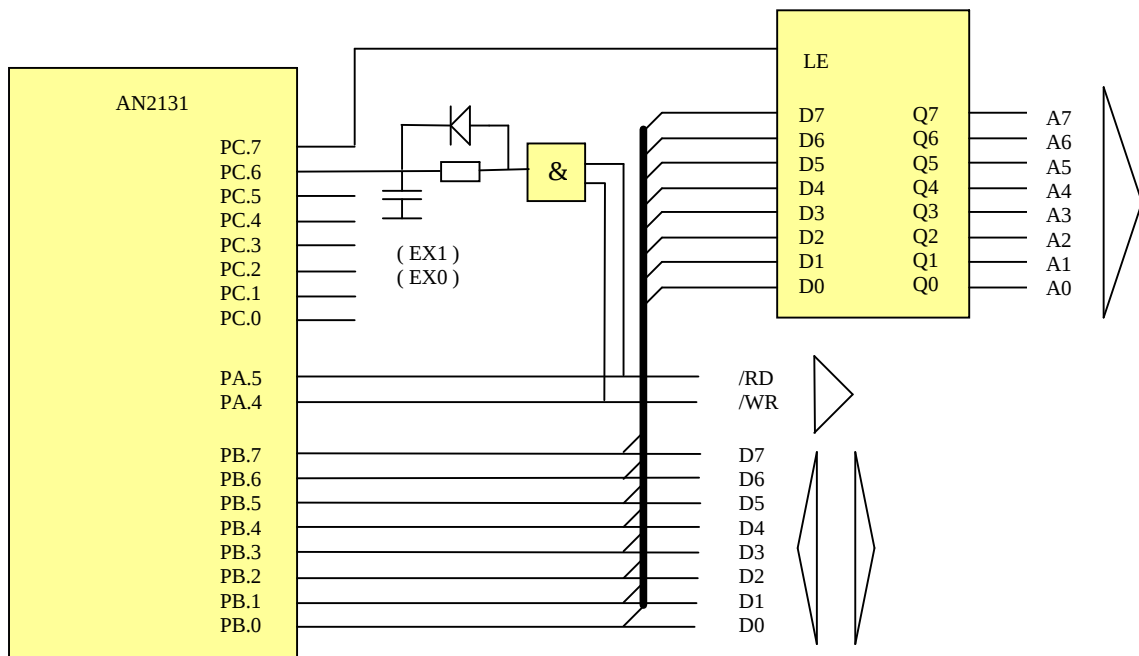


Betrieb des Controller im Interface-Mode IMode_5

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 sollte /DACK sofort wieder auf H-Pegel gesetzt werden.

! 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 !

2 Die Methoden der DLL

Im folgenden werden die von der AN21XX.DLL bereitgestellten Methoden sowie deren Anwendung beschrieben. Eine genaue Beschreibung der Datentypen in Abhängigkeit von der verwendeten Programmiersprache (Entwicklungsumgebung) erfolgt dabei nicht. Diese sind der entsprechenden Header-Datei zu entnehmen.

Visual Basic	-	AN21XX_dcl_v39.bas	
Visual C++	-	CUSBInterface.h	(AN21XX_dcl_v39.hpp für Borland C++ Builder)
Delphi	-	AN21XX_dcl_v39.pas	

Grundsätzliches zur Übergabe von Byteblöcken bei den Blockübertragungsroutinen

Ein Array wird an die DLL übergeben indem man die Adresse (einen Pointer) des ersten Elementes sowie die Anzahl der zu lesenden oder zu schreibenden Bytes übergibt. Es ist darauf zu achten, dass das übergebene Array mindestens die Grösse der zu übertragenden Bytes hat.

Hinweis ! Wird ein anderes, als das erste Element übergeben, so beginnt die Lese- oder Schreibaktion ab diesem Element. Die DLL sieht dann dieses Element als Beginn des Arrays.

Parmeter, der an fast alle Methoden der DLL übergeben werden muss

DevNumber - ein Wert zwischen 0 und 31. Gibt die Nummer des Controllers an, welcher angesprochen werden soll. Befindet sich nur ein Controller dieses Types am Bus so muss immer 0 übergeben werden.

Es sind momentan folgende Fehlerkonstanten festgelegt :

Err_No	= 0	kein Fehler aufgetreten
Err_DeviceHandle	= &H1	kein gültiges Device-Handle
Err_DeviceNumber	= &H2	unzulässige Device-Nummer (zulässig 0 .. 31)
Err_DriverName	= &H4	Es wurde bei einem vorherigen Device anderer Name gewählt
Err_IsDisabled	= &H8	Control ist Disabled gesetzt
Err_DeviceIOCtrl	= &H10	Funktion DeviceIOControl hat 'False' zurückgeliefert
Err_AI2CNotAck	= &H20	I2C-Bus NotAcknowledge nach Slaveadresse
Err_DI2CNotAck	= &H40	I2C-Bus NotAcknowledge nach Datum
Err_I2CBus	= &H80	allgemeiner I2C-Busfehler
Err_Array	= &H100	ByteArray hat unerlaubte Dimension
Err_McMode	= &H200	Methode im aktuellen Controllermodus nicht verfügbar
Err_Pointer	= &H400	Ungültige RAM-/EEP-Adresse übergeben
Err_DownloadFile	= &H800	Datei existiert nicht oder kann nicht gelesen werden
Err_PinNumber	= &H1000	unzulässige Pin-Nummer
Err_EEPType	= &H2000	nicht implementierter EEP-Type
Err_ClockDelay	= &H4000	unzulässige Taktverzögerung (IMode_3 + IMode_4)
Err_InterruptNumber	= &H8000	Interrupt-Nummer existiert nicht
Err_InterruptHandling	= &H10000	Fehler beim Abfragen der Interrupt-Flags des Controllers

Hinweis : Wird bei den Block-Methoden für die EEPROM's eine Bytezahl übergeben welche grösser als die Speichertiefe des gewählten EEPROM ist, so wird der Fehler „Err_Array“ durch die Methode GetDeviceError() zurückgegeben. Wird ein EEPROM-Pointer übergeben bei welchem, bei der angegebenen Byte-Zahl, die Speichertiefe 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.

2.1 Allgemeine Methoden der DLL (für alle Inteface-Modi erforderlich)

GetDeviceError (gibt den zuletzt aufgetretenen Fehler eines Controllers zurück)

GetDeviceError (DevNumber)

Rückgabewert : - zuletzt aufgetretener Fehler des Controllers „DevNumber“ . Mögliche Fehler siehe vorhergehende Seite

Übergabeparameter :

DevNumber - Nummer des anzusprechenden Controllers am Bus. Ist nur ein AN21xx am Bus, so muss immer 0 übergeben werden.

SetDeviceEnabled (aktiviert einen Controller im angegeben Mode)

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

Übergabeparameter :

DevNumber - Nummer des zu aktivierenden Controllers am Bus. Ist nur ein AN21xx am Bus, so muss immer 0 übergeben werden.

DriverName - 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 Daten/Adressbus Pin7 von PortC stellt das ALE-Signal zur Verfügung mit welchem die Adresse gelatcht werden kann Pin6 ist der nACK-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 .

GetDeviseEnabled (ermittelt ob ein Controller aktiviert ist – Enabled)

GetDeviceEnabled (DevNumber, MC_Mode)

Rückgabewert : - gibt bei aktiven Interface „True“ zurück, ansonsten „False“

Übergabeparameter :

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

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

GetDeviceDescriptor (gibt den Devicedescriptor eines aktiven Controllers zurück)

GetDeviceDescriptor (DevNumber, usbDD)

Übergabeparameter :

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

UsbDD - USB-Devicedescriptor
(Strukturvariable – wird im Deklarationsmodul der entsprechenden Programmiersprache veröffentlicht und ist **per Referenz** zu übergeben)

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

SetDeviceDisabled (DevNumber)

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

SetAllDisabled (deaktiviert alle aktiven Controller und schliesst deren Treiber)

Übergabeparameter : - keine

SetClockDelay

Methode setzt in allen Modi ausser ‚CommonMode‘ die Clock-Verzögerung für XRAM-Zugriffe des Controller. Dies wirkt sich auf die Ablaufgeschwindigkeit der Firmware des Controllers aus und beeinflusst die Transferraten bei allen Datenübertragungs-Methoden

SetClockDelay(DevNumber, Delay)

Übergabeparameter :

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

Delay - Wert der Clockverzögerung. Erlaubte Werte 0 ... 7

2.2 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-Inneren.

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

Übergabeparameter :

SetInterface (DevNumber, InterfaceNumber, Setting)

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

InterfaceNumber - zu setzendes Interface

Setting - zu aktivierende Konfiguration des Interface

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

DownloadBinFile (DevNumber, FileName)

Übergabeparameter :

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

FileName - Name und kompletter Pfad des Binärfiles

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

DownloadBinFile (DevNumber, FileName)

Übergabeparameter :

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

FileName - Name und kompletter Pfad des Hex-File

AbortPipe (beendet Transfer über Pipe)

AbortPipe (DevNumber, PipeNumber)

Übergabeparameter :

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

PipeNumber - Nummer der gewählten Pipe

ResetPipe (setzt Pipe zurück)

ResetPipe (DevNumber, PipeNumber)

Übergabeparameter :

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

PipeNumber - Nummer der gewählten Pipe

RAM_WriteByte (schreibt ein Byte an eine bestimmte Adresse des Code/Daten-RAM)

RAM_WriteByte (DevNumber, Adr, WrValue)

Übergabeparameter :

- | | |
|-----------|---|
| DevNumber | - Nummer des Controllers am Bus. Ist nur ein AN21xx am Bus, so muss immer 0 übergeben werden. |
| Adr | - RAM-Adresse in welches das Byte geschrieben werden soll
Achtung ! Es kann nicht in alle Bereiche geschrieben werden (Controllertype) |
| WrValue | - Wert des zu schreibenden Bytes |

RAM_ReadByte (gibt den Wert eine Speicherzelle des Code/Daten-RAM zurück)

RAM_ReadByte(DevNumber ,Adr)

Übergabeparameter :

- | | |
|-----------|---|
| DevNumber | - Nummer des Controllers am Bus. Ist nur ein AN21xx am Bus, so muss immer 0 übergeben werden. |
| Adr | - RAM-Adresse aus welcher der Wert gelesen werden soll |

RAM_WriteBlock (schreibt einen Block ab einer bestimmten Adresse in das Code/Daten-RAM)

RAM_WriteBlock (DevNumber, Adr, Barray, ByteCnt)

Übergabeparameter :

- | | |
|-----------|---|
| DevNumber | - Nummer des Controllers am Bus. Ist nur ein AN21xx am Bus, so muss immer 0 übergeben werden. |
| Adr | - RAM-Adresse ab welcher geschrieben werden soll.
Achtung ! Es kann nicht in alle Bereiche geschrieben werden (Controllertype) |
| BArray | - Pointer auf ein Element des zu schreibenden Bytearray's
Achtung ! Wird erstes Element übergeben, so wird ab diesem Element geschrieben, das ist aber nicht Bedingung.
Siehe Testprogramm der verwendeten Sprache |
| ByteCnt | - Anzahl der zu schreibenden Bytes (max. 8192 Bytes) |

RAM_ReadBlock (liest einen Block ab einer bestimmten Adresse aus dem Code/Daten-RAM)

RAM_ReadBlock (DevNumber, Adr, Barray, ByteCnt)

Übergabeparameter :

- | | |
|-----------|--|
| DevNumber | - Nummer des Controllers am Bus. Ist nur ein AN21xx am Bus, so muss immer 0 übergeben werden. |
| Adr | - RAM-Adresse ab welcher geschrieben werden soll.
Achtung ! Es kann nicht in alle Bereiche geschrieben werden (Controllertype) |
| BArray | - Pointer auf ein Element des Bytearray's
Achtung ! Wird erstes Element übergeben, so wird ab diesem Element eingelesen, das ist aber nicht Bedingung.
Siehe Testprogramm der verwendeten Sprache |
| ByteCnt | - Anzahl der zu lesenden Bytes (max. 1024 Bytes) |

WriteBulkData (schreibt Daten im Bulk-Transfermodus zum Controller)

WriteBulkData (DevNumber, PipeNumber, Barray, ByteCnt)

Übergabeparameter :

- DevNumber - Nummer des Controllers am Bus. Ist nur ein AN21xx am Bus, so muss immer 0 übergeben werden.
- PipeNumber - Nummer der zu verwendenden Pipe
- Barray - Pointer auf ein Element des Bytearray's oder auf ein einzelnes.
Achtung ! Wird erstes Element übergeben, so wird ab diesem Element geschrieben, das ist aber nicht Bedingung.
Siehe Testprogramm der verwendeten Sprache
- ByteCnt - Anzahl der zu schreibenden Bytes (max. 65535 Bytes)

ReadBulkData (liest Daten im Bulk-Transfermodus aus dem Controller)

ReadBulkData (DevNumber, PipeNumber, Barray, ByteCnt)

Übergabeparameter :

- DevNumber - Nummer des Controllers am Bus. Ist nur ein AN21xx am Bus, so muss immer 0 übergeben werden.
- PipeNumber - Nummer der zu verwendenden Pipe
- BArray - Pointer auf ein Element des. Bytearray's oder auf ein einzelnes.
Achtung ! Wird erstes Element übergeben, so wird ab diesem Element gelesen, das ist aber nicht Bedingung.
Siehe Testprogramm der verwendeten Sprache
- ByteCnt - Anzahl der zu lesenden Bytes (max. 65535 Bytes)

2.3 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 nACK-Eingang (PortC Pin6) zurückgeführt werden. Bei Schaltungen die schnell genug sind, kann der Eingang nACK über einen !Widerstand! an Masse gelegt werden!

Achtung ! In den Modi ohne Adressbus (IMode_2 und IMode_4) muss den Methoden trotzdem eine Adresse übergeben werden. Diese wird innerhalb der Methode dann nicht weiter verwendet.

PB_ReadByte

Funktion liest ein Byte am Parallelbus des AN21xx (PortB)

PB_ReadByte (DevNumber, Addr)

Übergabeparameter :

DevNumber	- Nummer des Controllers am Bus. Ist nur ein AN21xx am Bus, so muss immer 0 übergeben werden.
Addr	- auszugebende Peripherieadresse (an PortC o. gemultiplext an PortB im IMode_5) – Nicht in den Modi 2 und 4

PB_WriteByte

Methode schreibt ein Byte auf den Parallelbus des AN21xx (PortB)

PB_WriteByte (DevNumber, Addr, DByte)

Übergabeparameter :

DevNumber	- Nummer des Controllers am Bus. Ist nur ein AN21xx am Bus, so muss immer 0 übergeben werden.
Addr	- auszugebende Peripherieadresse (an PortC o. gemultiplext an PortB im IMode_5) – Nicht in den Modi 2 und 4
DByte	- an PortB auszugebendes Datenbyte

PB_ReadBlock

Methode liest ein Block vom Parallelbus des AN21xx (PortB)

PB_ReadBlock (DevNumber, Addr, Barray, ByteCnt)

Übergabeparameter :

- | | |
|-----------|--|
| DevNumber | - Nummer des Controllers am Bus. Ist nur ein AN21xx am Bus, so muss immer 0 übergeben werden. |
| BArray | - Pointer auf ein Element des. Bytearray's oder auf ein einzelnes.
Achtung ! Wird erstes Element übergeben, so wird ab diesem Element gelesen, das ist aber nicht Bedingung.
Siehe Testprogramm der verwendeten Sprache |
| ByteCnt | - Anzahl der zu lesenden Bytes (max. 16777215 Bytes) |
| Addr | - auszugebende Peripherieadresse (an PortC o. gemultiplext an PortB im IMode_5)
– Nicht in den Modi 2 und 4 |

PB_ReadBlockInc (nur Interface-Mode IMode_1)

Methode liest ein Block vom Parallelbus des AN2131 (PortB) und inkrementiert nach jedem Lesezugriff den Adressbus. Die der Methode übergebene Adresse stellt dabei die Anfangsadress da (ausgegeben vor erstem Lesezugriff)

PB_ReadBlockInc (DevNumber, Addr, Barray, ByteCnt)

Übergabeparameter :

- | | |
|-----------|--|
| DevNumber | - Nummer des Controllers am Bus. Ist nur ein AN21xx am Bus, so muss immer 0 übergeben werden. |
| BArray | - Pointer auf ein Element des. Bytearray's oder auf ein einzelnes.
Achtung ! Wird erstes Element übergeben, so wird ab diesem Element gelesen, das ist aber nicht Bedingung.
Siehe Testprogramm der verwendeten Sprache |
| ByteCnt | - Anzahl der zu lesenden Bytes (max. 16777215 Bytes) |
| Addr | - auszugebende Peripherieadresse (an PortC) wird nach jedem Lesezugriff inkrementiert |

PB_WriteBlock

Methode liest ein Block vom Parallelbus des AN21xx (PortB)

PB_WriteBlock (DevNumber, Addr, Barray, ByteCnt)

Übergabeparameter :

- | | |
|-----------|--|
| DevNumber | - Nummer des Controllers am Bus. Ist nur ein AN21xx am Bus, so muss immer 0 übergeben werden. |
| BArray | - Pointer auf ein Element des. Bytearray's oder auf ein einzelnes.
Achtung ! Wird erstes Element übergeben, so wird ab diesem Element geschrieben, das ist aber nicht Bedingung.
Siehe Testprogramm der verwendeten Sprache |
| ByteCnt | - Anzahl der zu schreibenden Bytes (max. 16777215 Bytes) |
| Addr | - auszugebende Peripherieadresse (an PortC o. gemultiplext an PortB im IMode_5)
– Nicht in den Modi 2 und 4 |

PB_WriteBlockInc (nur Interface-Mode IMode_1)

Methode schreibt einen Block zum Parallelbus des AN2131 (PortB) und inkrementiert nach jedem Schreibzugriff den Adressbus. Die der Methode übergebene Adresse stellt dabei die Anfangsadresse dar (ausgegeben vor erstem Schreibzugriff)

PB_WriteBlockInc (DevNumber, Addr, Barray, ByteCnt)

Übergabeparameter :

- | | |
|-----------|---|
| DevNumber | - Nummer des Controllers am Bus. Ist nur ein AN21xx am Bus, so muss immer 0 übergeben werden. |
| BArray | - Pointer auf ein Element des. Bytearray's oder auf ein einzelnes.
Achtung ! Wird erstes Element übergeben, so wird ab diesem Element geschrieben, das ist aber nicht Bedingung. |
| ByteCnt | - Anzahl der zu lesenden Bytes (max. 16777215 Bytes) |
| Addr | - auszugebende Peripherieadresse (an PortC) wird nach jedem Schreibeibzugriff inkrementiert |

2.4 Allgemeine Methoden des I2C-Bus

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

I2C_ReadByte Funktion gibt ein am I2C-Bus gelesenes Byte zurück

I2C_ReadByte (DevNumber, Addr)

Übergabeparameter :

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

Addr - Slave-Adresse des Bausteins am I2C-Bus

I2C_WriteByte Methode schreibt ein Byte zu einem bestimmten Slave am I2C-Bus

I2C_WriteByte (DevNumber, Addr, DByte)

Übergabeparameter :

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

Addr - Slave-Adresse des Bausteins am I2C-Bus

DByte - am I2C-Bus auszugebendes DatenByte

I2C_ReadBlock Methode liest einen Block von einem bestimmten Slave am I2C-Bus

I2C_ReadBlock (DevNumber, Addr, Barray, ByteCnt)

Übergabeparameter :

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

Addr - Slave-Adresse des Bausteins am I2C-Bus

BArray - Pointer auf ein Element des. Bytearray's oder auf ein einzelnes Byte.
Achtung ! Wird erstes Element übergeben, so wird ab diesem Element gelesen,
das ist aber nicht Bedingung.

Siehe Testprogramm der verwendeten Sprache

ByteCnt - Anzahl der zu lesenden Bytes (max. 65535 Bytes)

I2C_WriteBlock Methode schreibt einen Block zu einem bestimmten Slave am I2C-Bus

I2C_WriteBlock (DevNumber, Addr, Barray, ByteCnt)

Übergabeparameter :

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

Addr - Slave-Adresse des Bausteins am I2C-Bus

BArray - Pointer auf ein Element des. Bytearray's oder auf ein einzelnes Byte.
Achtung ! Wird erstes Element übergeben, so wird ab diesem Element geschrieben,
das ist aber nicht Bedingung.

Siehe Testprogramm der verwendeten Sprache

ByteCnt - Anzahl der zu schreibenden Bytes (max. 65535 Bytes)

2.5 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.

Konstanten der unterstützten EEPROM-Typen :

C00	=	0
C01	=	1
C02	=	2
C04	=	3
C08	=	4
C16	=	5
C32	=	6
C64	=	7
C128	=	8
C256	=	9

EEP_ReadByte Funktion liest ein Byte an einer definierten Adresse eines EEPROM's

EEP_ReadByte (DevNumber, EEPTyp, Addr, EEPPointer)

Übergabeparameter :

DevNumber	- Nummer des Controllers am Bus. Ist nur ein AN21xx am Bus, so muss immer 0 übergeben werden.
EEPTyp	- auszulesender EEP-Type - Siehe Konstanten am Beginn dieses Abschnittes
Addr	- Slave-Adresse des EEPROM's am I2C-Bus
EEPPointer	- Speicheradresse an der gelesen werden soll

EEP_WriteByte - Methode schreibt ein Byte an eine definierte Adresse eines EEPROM's

EEP_WriteByte (DevNumber, EEPTyp, Addr, EEPPointer, DByte)

Übergabeparameter :

DevNumber	- Nummer des Controllers am Bus. Ist nur ein AN21xx am Bus, so muss immer 0 übergeben werden.
EEPTyp	- zu beschreibender EEP-Type - Siehe Konstanten am Beginn dieses Abschnittes
Addr	- Slave-Adresse des EEPROM's am I2C-Bus
EEPPointer	- Speicheradresse an die geschrieben werden soll
DByte	- zu schreibendes Daten-Byte

EEP_ReadBlock - Methode liest einen Block ab einer definierten Adresse eines EEPROM's

EEP_ReadBlock (DevNumber, EEPTyp, Addr, EEPPointer, Barray, ByteCnt)

Übergabeparameter :

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

EEPTyp - auszulesender EEP-Type - Siehe Konstanten am Beginn dieses Abschnittes

Addr - Slave-Adresse des EEPROM's am I2C-Bus

EEPPointer - Speicheradresse ab der gelesen werden soll

Barray - Pointer auf ein Element des. Bytearray's.
Achtung ! Wird erstes Element übergeben, so wird ab diesem Element gelesen, das ist aber nicht Bedingung.

Siehe Testprogramm der verwendeten Sprache

ByteCnt - Anzahl der zu lesenden Bytes (max. Speichertiefe des EEPROM)

EEP_WriteBlock - Methode schreibt einen Block ab einer definierten Adresse eines EEPROM's

EEP_WriteBlock (DevNumber, EEPTyp, Addr, EEPPointer, Barray, ByteCnt)

Übergabeparameter :

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

EEPTyp - zu beschreibender EEP-Type - Siehe Konstanten am Beginn dieses Abschnittes

Addr - Slave-Adresse des EEPROM's am I2C-Bus

EEPPointer - Speicheradresse ab der geschrieben werden soll

Barray - Pointer auf ein Element des. Bytearray's.
Achtung ! Wird erstes Element übergeben, so wird ab diesem Element geschrieben, das ist aber nicht Bedingung.

Siehe Testprogramm der verwendeten Sprache

ByteCnt - Anzahl der zu schreibenden Bytes (max. Speichertiefe des EEPROM)

2.6 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 !

Konstanten für den Übergabeparameter Port:

PortA = 0
PortB = 1
PortC = 2

**In Visual Basic wird für den Parameter „Port“ ein Enumerations-Typ (ePort) verwendet.
Siehe Testprogramm !**

Konstanten für Übergabeparameter „Pins“ in SetPortDir und „Pin“ in SetPortPin, ClearPortPin und ReadPortPin :

Pin0 = 1 - 2^0
Pin1 = 2 - 2^1
Pin2 = 4 - 2^2
Pin3 = 8 - 2^3
Pin4 = 16 - 2^4
Pin5 = 32 - 2^5
Pin6 = 64 - 2^6
Pin7 = 128 - 2^7

In Visual Basic wird für den Parameter „Pin“ der Methoden SetPortPin, ClearPortPin und ReadPortPin ein Enumerations-Typ (ePin) verwendet. Siehe Testprogramm !

SetPortDir

Diese Methode setzen die Übertragungsrichtung der einzelnen Pins eines Port's

SetPortDir (DevNumber, Port, Pins)

Übergabeparameter :

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

Port - Port, dessen Datenrichtung geändert werden soll (PortA = 0, PortB = 1, PortC = 2)

Pins - ein gesetztes Bit in diesem Parameter schaltet das betreffende Pin als Ausgang
Siehe Konstanten am Beginn dieses Absatzes !

SetPort

Diese Methode setzt das Ausgangsregister eines Ports. Ist ein Pin als Eingang geschaltet, so wird der für dieses Pin übergebene Pegel nicht wirksam !

SetPort (DevNumber, Port, PortValue)

Übergabeparameter :

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

Port - Port, dessen Datenrichtung geändert werden soll (PortA = 0, PortB = 1, PortC = 2)

PortValue - Wert, auf welchen das Ausgangsregister dieses Ports gesetzt werden soll

SetPortPin

Diese Methode setzt das Pin eines Ports. Ist das Pin als Eingang geschaltet, so wird die Methode für dieses Pin nicht wirksam !

SetPortPin (DevNumber, Port, Pin)

Übergabeparameter :

- | | |
|-----------|---|
| DevNumber | - Nummer des Controllers am Bus. Ist nur ein AN21xx am Bus, so muss immer 0 übergeben werden. |
| Port | - Port, dessen Datenrichtung geändert werden soll (PortA = 0, PortB = 1, PortC = 2) |
| Pin | - zu setzendes Pin - Siehe Konstanten am Beginn dieses Absatzes ! |

ClearPortPin

Diese Methode löscht das Pin eines Ports. Ist das Pin als Eingang geschaltet, so wird die Methode für dieses Pin nicht wirksam !

ClearPortPin (DevNumber, Port, Pin)

Übergabeparameter :

- | | |
|-----------|---|
| DevNumber | - Nummer des Controllers am Bus. Ist nur ein AN21xx am Bus, so muss immer 0 übergeben werden. |
| Port | - Port, dessen Datenrichtung geändert werden soll (PortA = 0, PortB = 1, PortC = 2) |
| Pin | - zu löschendes Pin - Siehe Konstanten am Beginn dieses Absatzes ! |

ReadPort

Diese Funktion gibt den Wert des Pins-Registers eines Ports zurück. Sind Pins dieses Ports als Ausgang geschaltet, so wird für die betreffenden Pins der Wert des Ausgangsregisters zurückgegeben.

ReadPort (DevNumber, Port)

Übergabeparameter :

- | | |
|-----------|---|
| DevNumber | - Nummer des Controllers am Bus. Ist nur ein AN21xx am Bus, so muss immer 0 übergeben werden. |
| Port | - Port, dessen Datenrichtung geändert werden soll (PortA = 0, PortB = 1, PortC = 2) |

ReadPortPin

Funktion gibt den Wert eines Pins eines des Ports zurück. Mögliche Rückgabewerte : 0 und 1

ReadPortPin (DevNumber, Port, Pin)

Übergabeparameter :

- | | |
|-----------|---|
| DevNumber | - Nummer des Controllers am Bus. Ist nur ein AN21xx am Bus, so muss immer 0 übergeben werden. |
| Port | - Port, dessen Datenrichtung geändert werden soll (PortA = 0, PortB = 1, PortC = 2) |
| Pin | - zu lesendes Pin - Siehe Konstanten am Beginn dieses Absatzes ! |

2.7 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

PortC.3 (Pin3 des PortC) - EX1

Zum Nutzen der externen Interrupts können zwei Callback-Routinen eingesetzt werden. Dazu muss über die Methode „**SetCallbackHandle**“ die Adresse der jeweiligen Routine an die DLL übergeben werden. Dies sollte beim Start der Anwendung geschehen. Anschliessend kann zu jedem beliebigen Zeitpunkt ein Interrupt wirksam bzw. unwirksam geschaltet werden. Das geschieht mit der Methode „**SetInterrupt**“.

SetRefreshTime - Methode legt fest, in welchem Intervall die Interrupt-Flags im Controller ausgelesen werden.

SetInterrupt - Methode aktiviert oder deaktiviert einen externen Interrupt

SetCallbackHandle - Methode übergibt Adresse der Callback-Routine des jeweiligen Interrupts

Konstanten zur Nutzung der Interrupts EX0 und EX1

EX0	: Byte	=	0
EX1	: Byte	=	1
IntEnabled	: Boolean	=	True
IntDisabled	: Boolean	=	False

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

Da die Methoden der DLL 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 Sinne. 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 Methode „**SetRefreshTime**“ festgelegt werden kann (siehe weiter unten). Damit man eine Anwendung über einen aufgetretenen Interrupt informieren kann, muss die Adresse einer Rückruf-Funktion (Methode) von der Anwendung an die DLL übergeben werden. Diese Methode muss exakt dem in der DLL festgelegten Typ entsprechen (Übergabeparameter, Rückgabetyt). Im vorliegenden Fall handelt es sich um eine Methode, welche nur einen Parameter besitzt, nämlich die „DeviceNumber“ des Controllers, an welchem der Interrupt auftrat. Um beide Interrupts nutzen zu können, müssen also zwei Methoden im Anwenderprogramm existieren, eine für EX0 und eine weitere für EX1. Die Adressen dieser werden mit der DLL-Routine „**SetCallbackHandle**“ übergeben. Nachdem ein Interrupt durch die Methode „**SetInterrupt**“ aktiviert (freigegeben) wurde, laufen bei Auftreten einer H/L-Flanke am entsprechenden Pin folgende Vorgänge ab :

Beispiel 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 DLL die entsprechende Callback-Routine im Anwenderprogramm aufgerufen. Der in dieser Routine stehende Code wird ausgeführt.
- 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.

Nachfolgend werden die für das Interrupt-Handling benötigten Methoden beschrieben und deren Anwendung an jeweils einem Visual-Basic und einem Delphi-Beispiel demonstriert :

SetCallbackHandle

Diese Methode übergibt die Adresse einer Rückruf-Funktion, welche bei Auftreten eines Interrupts angesprungen werden soll.

SetCallbackHandle(IntNumber, pInterrupt)

Übergabeparameter :

- IntNum - Interrupt, der aktiviert oder deaktiviert werden soll **Siehe Konstanten zu Beginn dieses Abschnittes !**
- PInterrupt - Adresse der Funktion (Funktions-Zeiger) **In VB kann der AddressOf-Operator nur innerhalb eines Moduls angewandt werden**

SetInterrupt

Diese Methode aktiviert oder deaktiviert einen der beiden externen Interrupts

SetInterrupt (DevNumber, IntNum, State)

Übergabeparameter :

- DevNumber - Nummer des Controllers am Bus. Ist nur ein AN21xx am Bus, so muss immer 0 übergeben werden.
- IntNum - Interrupt, der aktiviert oder deaktiviert werden soll **Siehe Konstanten zu Beginn dieses Abschnittes !**
- State - Status des Interrupt - aktivieren = True **Siehe Konstanten zu Beginn dieses Abschnittes !**

SetRefreshTime

SetRefreshTime (RefreshTime)

Methode setzt eine Zeitintervall, in welchem der Controller über aufgetretene externe Interrupts ‚befragt‘ wird.

Übergabeparameter :

- RerfreshTime - Zulässige Werte : 100 ms ... 10 s , Intervall wird in Millisekunden übergeben.

Wird eine unzulässige Intervall-Zeit übergeben, so wird der interne Timer auf **100ms** eingestellt !

Einbindung der Interrupts in eigene Anwendungen :

Zuerst muss für jeden der benötigten Interrupts jeweils eine Methode bereitgestellt werden und deren Adresse mit der DLL-Routine „SetCallBackHandle“ an die DLL übergeben werden !

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 Methode `SetRefreshTime` 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 ! Auf eine Fehlerrückmeldung wurde aus Performance-Gründen verzichtet .

Anwendungsbeispiel für Delphi

Zur Nutzung der DLL in einer eigenen Anwendung fügen Sie bitte die Datei „AN21XX_dcl.pas“ in Ihr Projekt ein. Diese enthält alle Deklarationen der DLL-Routinen sowie eine grosse Anzahl von Konstanten.

```
// Type der CallBack-Routine festlegen - durch DLL vorgegeben( in AN21XX_dcl_v39.pas )
type TInterrupt = procedure ( DevNum : Byte ); stdcall;

// Deklaration der DLL-Routine zur Übergabe der Prozedur-Adresse an die Anwendung
// (in AN21XX_dcl_v39.pas)

procedure SetCallBackHandle(IntNumber : Byte;
                           pInterrupt : TInterrupt); stdcall; external 'AN21XX_v39.DLL';

// Code im Anwendungsprogramm bei Nutzung beider Interrupts

// CallBack-Routinen für beide Interrupts
procedure Interrupt_EX0( DevNum : Byte); stdcall;

begin
    // Code bei Auftreten eines externen Interrupts 0
end;

procedure Interrupt_EX1( DevNum : Byte); stdcall;

begin
    // Code bei Auftreten eines externen Interrupts 1
end;

. . .

// Funktionspointer übergeben
SetCallBackHandle( EX0, Interrupt_EX0);           // Adresse von Interrupt_EX0 wird übergeben
SetCallBackHandle( EX1, Interrupt_EX1);           // Adresse von Interrupt_EX1 wird übergeben

// Interrupts aktivieren
SetInterrupt(0, EX0, IntEnabled);                 // externen Interrupt 0 im AN21XX aktivieren
SetInterrupt(0, EX1, IntEnabled);                 // externen Interrupt 0 im AN21XX aktivieren

. . .

// Zum Deaktivieren eines oder beider Interrupts wird ebenfalls
// wieder die DLL-Routine „SetInterrupt“ verwandt

SetInterrupt(0, EX0, IntDisabled);                 // externen Interrupt 0 im AN21XX deaktivieren

// EX0, EX1, IntEnabled sowie IntDisabled sind Konstanten aus der Datei „AN21XX_dcl_v39.pas“
```

Anwendungsbeispiel für Visual Basic

Zur Nutzung der DLL in einer eigenen Anwendung fügen Sie bitte die Datei „AN21XX_dcl.bas“ in Ihr Projekt ein. Diese enthält alle Deklarationen der DLL-Routinen sowie eine grosse Anzahl von Konstanten.

Achtung !

Sowohl die CallBack-Routinen als auch die Routinen zur Übergabe der Prozedur-Adresse müssen sich in einem Modul befinden. Das hat seine Ursache darin, dass der AddressOf-Operator nur auf Modulebene zulässig ist. Wenn Sie die Datei "AN21XX_dcl_v39.bas" in Ihr Projekt einfügen sind bereits alle Vorbereitungen getroffen. Durch Aufrufen der Prozedur **SetCallBackHandles** werden beide Prozedur-Adressen an die DLL übergeben. In den CallBack-Routinen muss nur noch der gewünschte Code eingetragen werden und die Interrupts durch die DLL-Routine **"SetInterrupt"** aktiviert wurden .

```
' Deklaration der DLL-Routine zur Übergabe der Prozedur-Adresse an die Anwendung
' in AN21XX_dcl_v39.bas)
Declare Sub SetCallBackHandle Lib "AN21xx_v39.dll" (ByVal IntNumber As Byte,
                                                    ByVal pInterrupt As Long)

' CallBack-Routinen für beide Interrupts (in AN21XX_dcl_v39.bas)
Public Sub Interrupt_EX0(ByVal DevNum As Byte)
' Code bei Auftreten eines externen Interruts 0
End Sub

Public Sub Interrupt_EX1(ByVal DevNum As Byte)
' Code bei Auftreten eines externen Interruts 1
End Sub

. . .

' Funktionspointer übergeben (in AN21XX_dcl_v39.bas)
Public Sub SetCallBackHandles()
' Funktions-Pointer an DLL übergeben
Call SetCallBackHandle(EX0, AddressOf Interrupt_EX0)
Call SetCallBackHandle(EX1, AddressOf Interrupt_EX1)
End Sub

' Interrupts aktivieren
Call SetInterrupt(0, EX0, IntEnabled);           ' externen Interrupt 0 im AN21XX aktivieren
Call SetInterrupt(0, EX1, IntEnabled);           ' externen Interrupt 1 im AN21XX aktivieren

. . .

' Zum Deaktivieren eines oder beider Interrupts wird ebenfalls
' wieder die DLL-Routine "SetInterrupt" verwandt

Call SetInterrupt(0, EX0, IntDisabled);           ' externen Interrupt 0 im AN21XX aktivieren

' EX0, EX1, IntEnabled sowie IntDisabled sind Konstanten aus der Datei „AN21XX_dcl_v39.bas“
```