AnaGate API 2

Programmer's Manual

Analytica GmbH

A. Schmidt, Analytica GmbH
S. Welisch, Analytica GmbH
J. Gossens, Analytica GmbH

AnaGate API 2: Programmer's Manual

Analytica GmbH
von A. Schmidt, S. Welisch und J. Gossens

Dieses Dokument wurde mittels DocBook am 07.04.2016 11:00:03 erzeugt.

Hilfe-Datei (dtsch.): *Manual-AnaGateAPI.chm*

Hilfe-Datei (engl.): *Manual-AnaGateAPI-EN.chm*

PDF-Datei (dtsch.): *Manual-AnaGateAPI-2.2.pdf*

PDF-Datei (engl.): *Manual-AnaGateAPI-2.2-EN.pdf*

Veröffentlicht 2014-12-10
Copyright © 2007-2015 Analytica GmbH

Zusammenfassung

Das AnaGate Programmer's Manual umfasst die Beschreibung der Programmierschnittstellen zu den Hardware-Komponenten der *AnaGate*-Serie.

Basis der Beschreibung ist das *AnaGate* Application Programming Interface (API) in der aktuellen Version 2.2 und dem AnaGate Kommunikationsprotokoll V1.3 (siehe [TCP-2010]).

Alle Rechte vorbehalten. Sämtliche Angaben zum Handbuch wurden sorgfältig erarbeitet, erfolgen jedoch ohne Gewähr.

Kein Teil des Handbuchs, der Programm-Beispiele oder Programms darf in irgendeiner Form (Druck, Fotokopie, Mikrofilm oder in einem anderen Verfahren) ohne unsere vorherige schriftliche Genehmigung reproduziert oder unter Verwendung elektronischer Systeme verarbeitet, vervielfältigt oder verbreitet werden.

Wir weisen darauf hin, dass die in der Dokumentation verwendeten Bezeichnungen und Markennamen der jeweiligen Firmen im Allgemeinen warenzeichen-, marken- oder patentrechtlichem Schutz unterliegen.

Analytica GmbH
Bannwaldallee 60
76185 Karlsruhe
Germany
Fon +49 (0) 721-43035-0
Fax +49 (0) 721-43035-20
<support@analytica-gmbh.de>

www.analytica-gmbh.de [http://www.analytica-gmbh.de]
www.anagate.de [http://www.anagate.de]

Versionsgeschichte			
Version 2.2	25.06.2015	ASc	Beschreibung für SocketCAN-Support hinzugefügt.
Version 2.1	25.04.2014	SWe	Beschreibung und Beispiel der Benutzerspezifischen CAN-Baudrate hinzugefügt
Version 2.0	15.05.2013	JGo	Komplette Überarbeitung für AnaGate API 2
Version 1.7	25.02.2013	SWe	LS_I2COpenDeviceEx hinzugefügt
Version 1.6	09.11.2012	JGo	CANSetMaxSizePerQueue und CANGetMessage hinzugefügt
Version 1.5	31.08.2012	JGo	I2CWriteEEPromPollAck hinzugefügt
Version 1.4	01.10.2010	ASc	Komplette Überarbeitung aller Kapitel
Version 1.3	12.07.2010	SWe	CAN UDP-Funktionalität hinzugefügt (nur Lua)
Version 1.2	04.06.2010	SWe	I2C RAW-Funktionen hinzugefügt (temporär)
Version 1.1	01.04.2010	ASc	englische Version
Version 1.0	08.06.2009	ASc	Handbuch auf **DocBook**-Format umgestellt

Inhaltsverzeichnis

Einleitung ix
I. AnaGate API 1
1. Die Programmierschnittstelle der *AnaGate*-Serie 4
2. Anmerkungen zum Kommunikationsprotokoll TCP 7
2.1. Besondere Protokolleigenschaften 7
3. Allgemeine Funktionen 9
DLLInfo 10
4. CAN API Funktionen 11
CANOpenDevice, CANOpenDeviceEx 12
CANCloseDevice 15
CANSetGlobals 16
CANGetGlobals 19
CANSetFilter 21
CANGetFilter 23
CANSetTime 24
CANGetTime 25
CANWrite, CANWriteEx 26
CANSetCallback, CANSetCallbackEx 28
CANSetMaxSizePerQueue 30
CANGetMessage 32
CANReadDigital 34
CANWriteDigital 36
CANReadAnalog 37
CANWriteAnalog 39
CANRestart 41
CANDeviceConnectState 42
CANStartAlive 43
CANErrorMessage 44
5. SPI API Funktionen 45
SPIOpenDevice 46
SPICloseDevice 48
SPISetGlobals 49
SPIGetGlobals 51
SPIDataReq 53
SPIReadDigital 55
SPIWriteDigital 57
SPIErrorMessage 58
6. I2C API Funktionen 59
I2COpenDevice 60
I2CCloseDevice 62
I2CReset 63
I2CRead 64
I2CWrite 65
I2CSequence 66
I2CReadDigital 68
I2CWriteDigital 70
I2CErrorMessage 71
I2CReadEEProm 72
I2CWriteEEProm 74
I2CWriteEEPromPollAck 77
7. Programmier-Beispiele 80

© 2007-2015 Analytica GmbH

7.1. Programmiersprache C/C++ 80
7.2. Programmiersprache Visual Basic 6 81
7.3. Programmiersprache VB.NET 85
II. SocketCAN-Schnittstelle 89
8. Die *SocketCAN*-Schnittstelle der *AnaGate*-Serie 91
9. Beschreibung des SocketCAN-Gateway 92
10. Verwendung des SocketCAN-Gateway 93
10.1. Virtuelles SocketCAN-Netzwerkgerät 93
10.2. SocketCANGateway 94
10.3. SocketCAN-Beispielanwendung 96
III. Skriptsprache Lua 98
11. Die Lua-Skripting-Schnittstelle der *AnaGate*-Serie 100
11.1. Skriptdateien erstellen 101
11.2. Skriptdateien auf einem PC ausführen 101
11.3. Skriptdateien auf der *AnaGate*-Hardware ausführen 102
12. Allgemeine Funktionen 105
LS_DeviceInfo 106
LS_GetTime 107
LS_Sleep 108
13. CAN Funktionen 109
LS_CANOpenDevice 110
LS_CANCloseDevice 112
LS_CANRestartDevice 113
LS_CANSetGlobals 114
LS_CANGetGlobals 116
LS_CANWrite 118
LS_CANWriteEx 120
LS_CANGetMessage 122
LS_CANSetFilter 124
LS_CANGetFilter 125
LS_CANSetTime 126
LS_CANErrorMessage 127
LS_CANReadDigital 128
LS_CANWriteDigital 130
LS_CANReadAnalog 131
LS_CANWriteAnalog 132
14. SPI Funktionen 133
LS_SPIOpenDevice 134
LS_SPICloseDevice 136
LS_SPISetGlobals 137
LS_SPIGetGlobals 139
LS_SPIDataReq 141
LS_SPIErrorMessage 143
LS_SPIReadDigital 144
LS_SPIWriteDigital 146
15. I2C-Funktionen 147
LS_I2COpenDevice 148
LS_I2COpenDeviceEx 150
LS_I2CCloseDevice 152
LS_I2CReset 153
LS_I2CRead 154
LS_I2CWrite 155
LS_I2CSequence 156
LS_I2CReadDigital 158

 © 2007-2015 Analytica GmbH

LS_I2CWriteDigital 160
LS_I2CErrorMessage 161
LS_I2CReadEEProm 162
LS_I2CWriteEEProm 164
16. Lua-Programmier-Beispiele 167
16.1. Beispiele für Geräte mit CAN-Schnittstelle 167
16.2. Beispiele für Geräte mit SPI-Schnittstelle 168
16.3. Beispiele für Geräte mit I2C-Schnittstelle 169
A. Rückgabewerte aus den API-Funktionen 172
B. Adressierung auf dem I2C-Bus 175
C. Programmierung von I2C-EEPROM 177
D. FAQ - Häufig gestellte Fragen 179
E. FAQ zu den Programmier-Schnittstellen 184
F. Technischer Support 185
Literaturverzeichnis 186

© 2007-2015 Analytica GmbH

Abbildungsverzeichnis

7.1. Eingabe-Formular SPI-Beispiel (VB6) 82
11.1. Bearbeiten von Lua-Skript im Texteditor 101
11.2. HTTP-Interface, Lua-Einstellungen 103
B.1. Definition einer I2C-Slaveadresse im 7-Bit-Format 175
B.2. Definition einer I2C-Slaveadresse im 10-Bit-Format 175

 © 2007-2015 Analytica GmbH

Tabellenverzeichnis

1.1. Programm-Bibliotheken der API unter Windows 4
1.2. Programm-Bibliotheken der API unter 32-Bit-Linux 4
1.3. Programm-Bibliotheken der API unter 64-Bit-Linux 5
2.1. AnaGate-Modelle und Ihre Portnummern 7
3.1. Datentypen mit festen Größen 9
4.1. Beispielkonfiguration CAN-Baudrate 16
4.2. Beispiele für Maskenfilter für CAN Identifier 21
A.1. Allgemeine Rückgabewerte für alle Geräte der AnaGate-Serie 172
A.2. Rückgabewerte für AnaGate I2C 172
A.3. Rückgabewerte für AnaGate CAN 173
A.4. Rückgabewerte für AnaGate SPI 173
A.5. Rückgabewerte für AnaGate Renesas 173
A.6. Rückgabewerte für Lua Scripting 174
B.1. Adressierungs-Beispiele von I2C-EEPROMs 176
C.1. Verwendung der Chip-Enable-Bits bei I2C-EEPROMs 177
D.1. Nutzung AnaGate-Hardware mit Firewall 181

 © 2007-2015 Analytica GmbH

Liste der Beispiele

10.1. Programmaufruf 96
16.1. CAN-Lua-Scriptbeispiel 167
16.2. SPI-Lua-Scriptbeispiel 168
16.3. I2C-Lua Scriptbeispiel EEPROM-Funktionen 169
16.4. I2C - Lua Scriptbeispiel I2C-Direkt 170
16.5. I2C-Lua-Scriptbeispiel Sequence 171

 © 2007-2015 Analytica GmbH

Einleitung

Das AnaGate Programmer's Manual umfasst die Beschreibung der Programmierschnittstellen zu den Hardware-Komponenten der AnaGate Serie.

Im Folgenden geht das Dokument auf folgende Schnittstellen ein:

- Application Programming Interface (Teil I, „AnaGate API“)
- Linux SocketCAN Interface(Teil II, „ SocketCAN-Schnittstelle “)
- Lua Scripting Interface (Teil III, „ Skriptsprache Lua “)

© 2007-2015 Analytica GmbH

Teil I. AnaGate API

Inhaltsverzeichnis

1. Die Programmierschnittstelle der *AnaGate*-Serie 4
2. Anmerkungen zum Kommunikationsprotokoll TCP 7
   2.1. Besondere Protokolleigenschaften 7
3. Allgemeine Funktionen 9
   DLLInfo 10
4. CAN API Funktionen 11
   CANOpenDevice, CANOpenDeviceEx 12
   CANCloseDevice 15
   CANSetGlobals 16
   CANGetGlobals 19
   CANSetFilter 21
   CANGetFilter 23
   CANSetTime 24
   CANGetTime 25
   CANWrite, CANWriteEx 26
   CANSetCallback, CANSetCallbackEx 28
   CANSetMaxSizePerQueue 30
   CANGetMessage 32
   CANReadDigital 34
   CANWriteDigital 36
   CANReadAnalog 37
   CANWriteAnalog 39
   CANRestart 41
   CANDeviceConnectState 42
   CANStartAlive 43
   CANErrorMessage 44
5. SPI API Funktionen 45
   SPIOpenDevice 46
   SPICloseDevice 48
   SPISetGlobals 49
   SPIGetGlobals 51
   SPIDataReq 53
   SPIReadDigital 55
   SPIWriteDigital 57
   SPIErrorMessage 58
6. I2C API Funktionen 59
   I2COpenDevice 60
   I2CCloseDevice 62
   I2CReset 63
   I2CRead 64
   I2CWrite 65
   I2CSequence 66
   I2CReadDigital 68
   I2CWriteDigital 70
   I2CErrorMessage 71
   I2CReadEEProm 72
   I2CWriteEEProm 74
   I2CWriteEEPromPollAck 77
7. Programmier-Beispiele 80
   7.1. Programmiersprache C/C++ 80
   7.2. Programmiersprache Visual Basic 6 81

 © 2007-2015 Analytica GmbH

7.3. Programmiersprache VB.NET 85

© 2007-2015 Analytica GmbH

Kapitel 1. Die Programmierschnittstelle der *AnaGate*-Serie

Die *AnaGate*-Serie besteht aus verschiedenen Hardware-Modellen, die über ein herkömmmliches Netzwerkprotokoll Zugriff auf unterschiedliche Bussysteme (I2C, SPI, CAN) bzw. Prozessoren (Renesas) bieten.

Die Kommunikation zu den einzelnen Geräten erfolgt prinzipiell über ein proprietäres offengelegtes Netzwerkprotokoll. Somit können alle Steuergeräte, die ein sog. Socket-Interface besitzen (wie z.B. PC, embedded PC, SPS, ...), programmgesteuert auf die Geräte der *AnaGate*-Serie zugreifen.

Speziell für Nutzer von Windows- und X86-Linux-Betriebssystemen stellt Analytica seinen Kunden eine Programmierschnittstelle zur Verfügung, die das spezifische Kommunikations-Protokoll auf einfache Funktionsaufrufe umsetzt. Diese Software API (Application Programming Interface) ist als Schnittstellen-Bibliothek in der Programmiersprache C implementiert und steht für die Betriebssysteme Windows und Linux (X86) kostenlos zur Verfügung.

Gerät	Windows-Bibliothek 32-Bit	64-Bit
AnaGate CAN	AnaGateCan.dll	AnaGateCan64.dll
AnaGate CAN uno / duo / quattro	AnaGateCan.dll	AnaGateCan64.dll
AnaGate CAN USB	AnaGateCan.dll	AnaGateCan64.dll
AnaGate CAN X2 / X4 / X8	AnaGateCan.dll	AnaGateCan64.dll
AnaGate SPI	AnaGateSPI.dll	AnaGateSPI64.dll
AnaGate I2C / I2C X7	AnaGateI2C.dll	AnaGateI2C64.dll
AnaGate Universal Programmer	AnaGateSPI.dll, AnaGateI2C.dll	AnaGateSPI64.dll, AnaGateI2C64.dll

Tabelle 1.1. Programm-Bibliotheken der API unter Windows

Anmerkung

Um eine umfassende Unterstützung für verschiedene Programmiersprachen wie z.B. C++, Visual Basic, Delphi oder auch die Programmiersprachen der .NET-Familie unter Windows-Betriebssystemen zu ermöglichen, wurde die sog. cdecl-Aufrufkonvention bei der Funktionsdefinition verwendet. Bei dieser Aufrufkonvention werden die Funktionsparameter in umgekehrter Reihenfolge auf dem Stack abgelegt (von rechts nach links) und der Aufrufer ist für die Wiederherstellung des Stacks verantwortlich.

Gerät	Linux-Bibliothek (X86)	ARM9
AnaGate CAN	libCANDLLStaticRelease.a, libAnaGateExtStaticRelease.a, libAnaGateStaticRelease.a	-

© 2007-2015 Analytica GmbH

Gerät	Linux-Bibliothek (X86)	ARM9
AnaGate CAN uno / duo / quattro	libCANDLLStaticRelease.a, libAnaGateExtStaticRelease.a, libAnaGateStaticRelease.a	verfügbar
AnaGate CAN USB	libCANDLLStaticRelease.a, libAnaGateExtStaticRelease.a, libAnaGateStaticRelease.a	verfügbar
AnaGate SPI	libSPIDLLStaticRelease.a, libAnaGateExtStaticRelease.a, libAnaGateStaticRelease.a	-
AnaGate I2C / I2C X7	libI2CDLLStaticRelease.a, libAnaGateExtStaticRelease.a, libAnaGateStaticRelease.a	-
AnaGate Universal Programmer	libSPIDLLStaticRelease.a, libI2CDLLStaticRelease.a, libAnaGateExtStaticRelease.a, libAnaGateStaticRelease.a	verfügbar

Tabelle 1.2. Programm-Bibliotheken der API unter 32-Bit-Linux

Gerät	Linux-Bibliothek (X86-64)
AnaGate CAN	libCANDLLStaticRelease64.a, libAnaGateExtStaticRelease.a, libAnaGateStaticRelease.a
AnaGate CAN uno / duo / quattro	libCANDLLStaticRelease64.a, libAnaGateExtStaticRelease.a, libAnaGateStaticRelease.a
AnaGate CAN USB	libCANDLLStaticRelease64.a, libAnaGateExtStaticRelease.a, libAnaGateStaticRelease.a
AnaGate SPI	libSPIDLLStaticRelease64.a, libAnaGateExtStaticRelease.a, libAnaGateStaticRelease.a
AnaGate I2C / I2C X7	libI2CDLLStaticRelease64.a, libAnaGateExtStaticRelease.a, libAnaGateStaticRelease.a
AnaGate Universal Programmer	libSPIDLLStaticRelease64.a, libI2CDLLStaticRelease.a, libAnaGateExtStaticRelease.a, libAnaGateStaticRelease.a

Tabelle 1.3. Programm-Bibliotheken der API unter 64-Bit-Linux

Die Bibliotheken enthalten sowohl allgemeine Funktionen als auch die spezifischen Funktionen, die zum Zugriff auf die einzelnen Geräte der AnaGate-Serie notwendig sind. Im folgenden sind alle Programmfunktionen der Software API für die *AnaGate*-Serie im Detail dokumentiert.

 © 2007-2015 Analytica GmbH

Tipp

Für die neueren Gerätemodelle mit embedded Linux (Kernel 2.6) und ARM9-Prozessor können auch individuelle Erweiterungen für das Gerät selbst erstellt werden. Die vollständige Software-API in einer cross-kompilierten Version vereinfacht die Erstellung der Geräte-Erweiterung, da damit die identische Schnittstelle sowohl auf dem PC als auch auf dem Gerät selbst genutzt werden kann.

Eine vorkonfigurierte virtuelle Maschine (Virtual-Box-Image) mit Ubuntu-Linux „READY-to-USE" mit installierter Entwicklungsumgebung (**Kdevelop**, **Eclipse**) und allen notwendigen Linux-Paketen (**GCC**, Cross-Compiler, Libraries, **Lua**, ...) ist optional verfügbar.

© 2007-2015 Analytica GmbH

Kapitel 2. Anmerkungen zum Kommunikationsprotokoll TCP

Der Zugriff von einem PC auf die unterschiedlichen Modelle der *AnaGate*-Serie erfolgt über das sehr verbreitete Netzwerkprotokoll TCP (Transmission Control Protocol).

TCP ist ein verbindungsorientiertes, paketvermittelndes Transportprotokoll, das in Schicht 4 des OSI-Referenzmodells angesiedelt ist. Dabei ist TCP im Prinzip eine Ende-zu-Ende-Verbindung, welche die Übertragung der Informationen in beide Richtungen zur selben Zeit zulässt. Ein Endpunkt stellt ein geordnetes Paar dar, bestehend aus IP-Adresse und Port. Ein solches Paar bildet eine bidirektionale Software-Schnittstelle und wird auch als Socket bezeichnet.

Das *AnaGate* bietet seine Funktionalität als sog. TCP-Server an. Es erzeugt einen Endpunkt (Socket) mit seiner IP-Adresse und einer Geräte-spezifischen Portnummer. Bei den Modellen mit CAN-Schnittstelle(n) wird für jede vorhandene CAN-Schnittstelle ein eigener Socket mit unterschiedlicher Portnummer erzeugt. Auf diesen Modellen werden außerdem auf jedem Socket bis zu 5 Client-Verbindungen angenommen. Auf den SPI-, I2C- und Renesas-Schnittstellen ist jeweils nur eine einzige Verbindung zulässig.

Gerät	Portnummer
AnaGate I2C, AnaGate Universal Programmer	5000
AnaGate CAN, AnaGate CAN uno	5001
AnaGate CAN duo, AnaGate CAN X2	5001, 5101
AnaGate CAN quattro, AnaGate CAN X4	5001, 5101, 5201, 5301
AnaGate CAN X8	5001, 5101, 5201, 5301, 5401, 5501, 5601, 5701
AnaGate SPI, AnaGate Universal Programmer	5002
AnaGate Renesas, AnaGate Universal Programmer	5008

Tabelle 2.1. AnaGate-Modelle und Ihre Portnummern

Wichtig

Damit grundsätzlich eine Verbindung zum *AnaGate* möglich ist, muss sichergestellt sein, dass vom PC aus auch alle jeweils verwendeten Ports freigeschaltet sind. Eventuell vorhandene Firewalls oder ähnliches sind entsprechend zu konfigurieren.

2.1. Besondere Protokolleigenschaften

TCP setzt in den meisten Fällen auf das Internet-Protokoll (IP) auf. IP ist paketorientiert, wobei Datenpakete verlorengehen können, in verkehrter Reihenfolge ankommen dürfen und sogar doppelt empfangen werden können.

TCP beseitigt dieses Verhalten und stellt sicher, dass die Datenpakete in der korrekten Reihenfolge beim Empfänger ankommen. Wird ein Datenpaket vom Empfänger

© 2007-2015 Analytica GmbH

nicht innerhalb einer Timeout-Zeit quittiert, wird das Datenpaket erneut gesendet. Doppelte Pakete werden beim Empfänger erkannt und verworfen. Der Datentransfer an sich kann aber jederzeit nach dem Verbindungsaufbau gestört, verzögert oder ganz unterbrochen werden. Ein erfolgreicher Verbindungsaufbau stellt also keinerlei Gewähr für eine nachfolgende, dauerhaft gesicherte Übertragung dar.

Insbesondere gestaltet sich der Erkennung und Einschätzung von aktuellen Leitungsstörungen als schwierig, wenn nur sporadische Kommunikation auf der Verbindung stattfindet, bei der Datenpakete nur in eine Richtung gesendet werden. Wie kann man erkennen, ob die Leitung gestört ist oder aktuell keine Daten vom Partner gesendet werden?

Um diese Verbindungsproblematik zu entschärfen, bietet TCP einen sog. Keepalive-Mechanismus. Keepalives sind besondere Datenpakete, die in regelmäßigen Abständen durch einen bestehenden Kommunikationskanal zwischen den Partnern ausgetauscht werden. Vom Empfänger eines solchen Paketes wird innerhalb einer gewissen Zeitschranke eine Antwort erwartet. Bleibt das Keepalive-Paket oder die Reaktion darauf (ggf. mehrfach) aus, geht der entsprechende Kommunikationspartner von einer Unterbrechung der Verbindung oder einer Nichtfunktion des Kommunikationspartners aus.

Der TCP-Keepalive-Mechanismus ist standardmäßig deaktiviert und muss über die Funktion `setsockopt` explizit für jede Verbindung eingeschaltet werden. Die API-Funktionen zur Verbindung mit einem *AnaGate* - wie z.B. `CANOpenDevice()` - aktivieren grundsätzlich den TCP-KeepAlive-Mechanismus.

Anmerkung

Unter Windows können verschiedene Parameter hinsichtlich KeepAlive eingestellt werden. Diese sind aber für alle Netzwerkverbindungen des Rechners gültig und nicht nur für eine bestimmte Verbindung.

In der Windows-Registry unter dem Schlüssel `\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Tcpip\Parameters` können die Einstellungen **KeepAliveTime** und **KeepAliveInterval** entsprechend angepasst werden (erfordert Administratorrechte).

Insbesondere können die CAN-Ethernet-Gateways von der Problematik der Verbindungskontrolle betroffen sein, wenn kundenspezifische Systemanforderungen eine schnellere Erkennung eines Verbindungsabbruches benötigen, als dies über den Standard-Mechanismus möglich ist. Speziell für diese Geräte wurde ein anwendungsspezifischer Algorithmus in die Firmware integriert, über den eine individuelle Verbindungskontrolle durchgeführt werden kann. Über einen vorgegebenes Zeitintervall werden zwischen der Anagate-Hardware und der steuernden Einheit sog. Alive-Pakete (ALIVE_REQ, siehe [TCP-2010]) ausgetauscht, die von der Gegenseite entsprechend quittiert werden müssen. Dieser integrierte Alive-Mechanismus kann individuell für jede Verbindung mit unterschiedlichem Timeout-Intervall eingeschaltet werden.

Anmerkung

Für Nutzer der AnaGate-API muss der anwendungsspezifische Alive-Mechanismus nicht mehr implementiert werden. Über den Aufruf der Funktion `CANStartAlive` wird ein nebenläufiger Prozess gestartet, der die Kontrolle von der PC-Seite aktiviert und zeitgesteuert überwacht.

 © 2007-2015 Analytica GmbH

Kapitel 3. Allgemeine Funktionen

Die Bibliotheks-Schnittstelle der AnaGate API ist in der Programmiersprache C implementiert. Die Standard-Datentypen dieser Sprache sind nicht plattformübergreifend auf feste Wertebereiche festgelegt, was den Zugriff auf C-Funktionen aus anderen Programmiersprachen heraus erschweren kann. Datentypen mit fest definierten Wertebereichen wurden erst in neueren C-Versionen ab C99 plattformübergreifend zur Verfügung gestellt, welche jedoch nicht von allen, vor allem älteren Compilerversionen unterstützt werden.

Um auch zu älteren Compiler-Versionen kompatibel zu sein, die diese Version des Standards nicht oder nicht vollständig unterstützen, werden daher in der Anagate API eigene Aliase für Datentypen mit festen Wertebereichen definiert und verwendet. Diese Datentypen erleichtern den Zugriff auf API-Funktionen aus anderen Programmiersprachen heraus, indem sie die Speichergrößen der einzelnen Parameter explizit festlegen.

Bitgröße	Typ mit Vorzeichen	Typ ohne Vorzeichen
8	AnaInt8	AnaUInt8
16	AnaInt16	AnaUInt16
32	AnaInt32	AnaUInt32
64	AnaInt64	AnaUInt64

Tabelle 3.1. Datentypen mit festen Größen

© 2007-2015 Analytica GmbH

DLLInfo

DLLInfo — Ermittelt die aktuelle Version der AnaGate DLL.

Syntax

```
#include <AnaGateDLL.h>

AnaInt32 DLLVersion(char * pcMessage, AnaInt32 nMessageLen);
```

Parameter

pcMessage Datenpuffer, der die Versionskennung der AnaGate DLL aufnehmen soll.

nMessageLen Größe des übergebenen Datenpuffers in Byte.

Rückgabewert

Tatsächliche Größe der zurückgegebenen Versionskennung.

Bemerkung

Passt die ermittelte Versionskennung nicht in den übergebenen Datenpuffer, so wird die Kennung auf die angegebene Anzahl von Zeichen (*nMessageLen*) gekürzt.

© 2007-2015 Analytica GmbH

Kapitel 4. CAN API Funktionen

Mit den Funktionen der CAN API können alle CAN-Gateways der AnaGate-Serie angesprochen werden. Die Programmierschnittstelle ist für alle Geräte identisch und erfolgt generell über das Netzwerk-Protokoll TCP bzw. UDP.

Aktuell können folgende Geräte über die CAN API genutzt werden:

- AnaGate CAN
- AnaGate CAN uno
- AnaGate CAN duo
- AnaGate CAN quattro
- AnaGate CAN USB
- AnaGate CAN X2
- AnaGate CAN X4
- AnaGate CAN X8

© 2007-2015 Analytica GmbH

CANOpenDevice, CANOpenDeviceEx

CANOpenDevice, CANOpenDeviceEx — Baut eine Netzwerkverbindung (TCP bzw. UDP) zu einem AnaGate CAN auf.

Syntax

```
#include <AnaGateDllCan.h>

AnaInt32 CANOpenDevice(AnaInt32 *pHandle, AnaInt32 bSendDataConfirm,
AnaInt32 bSendDataInd, AnaInt32 nCANPort, const char * pcIPAddress,
AnaInt32 nTimeout);

AnaInt32 CANOpenDeviceEx(AnaInt32 *pHandle, AnaInt32 bSendDataConfirm,
AnaInt32 bSendDataInd, AnaInt32 nCANPort, const char * pcIPAddress,
AnaInt32 nTimeout, AnaInt32 nSocketType);
```

Parameter

pHandle	Zeiger auf eine Variable, in die das Zugriffs-Handle gespeichert wird, falls die Verbindung zum Gerät erfolgreich hergestellt wurde.
bSendDataConfirm	Sollen die gesendeten bzw. empfangenen Telegramme von der Gegenseite bestätigt werden? Ohne Bestätigung ist eine höhere Übertragungsperformance zu erreichen.
bSendDataInd	Gibt an, ob das AnaGate CAN empfangende Telegramme weiterleiten soll. Alle eingehenden Telegramme werden verworfen, falls dieser Parameter auf FALSE gesetzt wird.
nCANPort	Gibt die CAN Schnittstelle an, die verwendet werden soll. Erlaubte Werte sind: `0` für Port 1/A (alle AnaGate CAN Modelle) `1` für Port 2/B (AnaGate CAN duo, AnaGate CAN quattro, AnaGate CAN X2/X4/X8) `2` für Port 3/C (AnaGate CAN quattro,AnaGate CAN X4/X8) `3` für Port 4/D (AnaGate CAN quattro, AnaGate CAN X4/X8) `4` für Port 5/E (AnaGate CAN X8) `5` für Port 6/F (AnaGate CAN X8) `6` für Port 7/G (AnaGate CAN X8) `7` für Port 8/H (AnaGate CAN X8)
pcIPAddress	Netzwerkadresse des AnaGate-Partners.
nTimeout	Standard-Timeout für AnaGate-Zugriffe in Millisekunden.

 © 2007-2015 Analytica GmbH

Ein Timeout wird festgestellt, wenn die AnaGate-Hardware nicht innerhalb der vereinbarten Timeout-Zeit antwortet. Diese Timeout-Zeit gilt auf der aktiven Netzwerkverbindung für alle Kommandos bzw. Funktionen, für die kein spezifischer Timeout-Wert definiert werden kann.

nSocketType — Gibt den Sockettyp (Ethernet-Layer 4) an, der für die Verbindung verwendet werden soll. Es werden zwei verschiedene Typen unterstützt: TCP und UDP. Die Funktion `CANOpenDevice` fordert immer einen TCP-Socket an. Folgende Parameterwerte sind zu verwenden:

1 (SOCK_STREAM)	TCP (Transmission Control Protocol)
2 (SOCK_DGRAM)	UDP (User Datagram Protocol)

Rückgabewert

Die Funktion gibt im Erfolgsfall `Null` zurück, andernfalls einen Fehlercode (Anhang A, *Rückgabewerte aus den API-Funktionen*).

Beschreibung

Baut eine Netzwerkverbindung (TCP) zu einer CAN-Schnittstelle eines Gerätes der AnaGate CAN Serie auf. Mit `CANOpenDeviceEx` kann zusätzlich das Layer-4-Protokoll (TCP oder UDP) mit angegeben werden. Erst nach dem erfolgreichen Verbinden zur CAN-Schnittstelle ist ein Zugriff auf den CAN-Bus möglich.

Durch einen Aufruf der Funktion `CANCloseDevice` wird die bestehende Verbindung wieder geschlossen.

Wichtig

Das explizite Schließen der Verbindung ist notwendig, um die in der DLL angelegten Systemressourcen wieder freizugeben und dem verbundenen Gerät mitzuteilen, dass die Verbindung nicht mehr genutzt wird und wieder für neue Verbindungsanfragen zur Verfügung gestellt werden soll.

Im folgenden ein Programmier-Beispiel für den initialen Zugriff auf eine Schnittstelle.

```
#include <AnaGateDllCan.h>
int main()
{
   AnaInt32 hHandle;
   AnaInt32 nRC = CANOpenDevice(&hHandle, TRUE, TRUE, 0, "192.168.1.254", 5000);
   if ( nRC == 0 )
   {
      // ... now do something
      CANCloseDevice(hHandle);
   }
   return 0;
}
```


© 2007-2015 Analytica GmbH

Bemerkung

Die Funktion `CANOpenDeviceEx` ist erst ab Version 1.5-1.10 der Laufzeitbibliothek vorhanden und wird erst ab der Geräte-Firmwareversion 1.3.7 unterstützt.

Die Gerätemodelle vom Typ AnaGate CAN (Hardware-Version 1.1.A) können keine Netzwerkverbindungen über UDP annehmen. Beim Versuch, ein solches Gerät über UDP anzusprechen, wird die Funktion `CANOpenDeviceEx` fehlerhaft beendet (Timeout).

Siehe auch

CANCloseDevice

CANRestart

© 2007-2015 Analytica GmbH

CANCloseDevice

CANCloseDevice — Schließt eine geöffnete Netzwerk-Verbindung zu einem AnaGate CAN Device.

Syntax

```
#include <AnaGateDllCan.h>

AnaInt32 CANCloseDevice(AnaInt32 hHandle);
```

Parameter

hHandle Gültiges Zugriffs-Handle eines erfolgreichen Aufrufs von `CANOpenDevice`.

Rückgabewert

Die Funktion gibt im Erfolgsfall `Null` zurück, andernfalls einen Fehlercode (Anhang A, *Rückgabewerte aus den API-Funktionen*).

Beschreibung

Schließt eine geöffnete Netzwerk-Verbindung zu einem AnaGate CAN Device. Das übergebene Handle *`hHandle`* ist ein Rückgabewert aus einem vorangegangenen erfolgreichen Aufruf der Funktion `CANOpenDevice`.

Wichtig

Das explizite Schließen der Verbindung ist notwendig, um die in der DLL angelegten Systemressourcen wieder freizugeben und dem verbundenen Gerät mitzuteilen, dass die Verbindung nicht mehr genutzt wird und wieder für neue Verbindungsanfragen zur Verfügung gestellt werden soll.

Siehe auch

CANOpenDevice, CANOpenDeviceEx

© 2007-2015 Analytica GmbH

CANSetGlobals

CANSetGlobals — Setzt die globalen Einstellungen, mit denen auf dem CAN-Bus gearbeitet werden soll.

Syntax

#include <AnaGateDllCan.h>

```
AnaInt32 CANSetGlobals(AnaInt32 hHandle, AnaUInt32 nBaudrate, AnaUInt8 nOperatingMode, AnaInt32 bTermination, AnaInt32 bHighSpeedMode, AnaInt32 bTimeStampOn);
```

Parameter

hHandle Gültiges Zugriffs-Handle eines erfolgreichen Aufrufs von `CANOpenDevice`.

nBaudrate Baudrate, mit der gearbeitet werden soll. Folgende Werte werden unterstützt:

- 10.000 für 10kBit
- 20.000 für 20kBit
- 50.000 für 50kBit
- 62.500 für 62,5kBit
- 100.000 für 100kBit
- 125.000 für 125kBit
- 250.000 für 250kBit
- 500.000 für 500kBit
- 800.000 für 800kBit (nicht AnaGate CAN)
- 1.000.000 für 1MBit

Zum Setzen individueller Baudraten an einer CAN-Schnittstelle, müssen die Kontrollregister CNF1, CNF2 und CNF3 des von uns eingesetzten CAN-Controllers MCP2515 direkt gesetzt werden. Durch Setzen des Bit 24 im Baudrate-Parameter wird der Geräte-Firmware mitgeteilt, dass die 3 unteren Bytes des Parameterwertes direkt in die Kontrollregister geschrieben werden sollen. Der Parameter ist in diesem Fall folgendermaßen definiert:

	MCP2515 CNF1	**MCP2515 CNF2**	**MCP2515 CNF3**	**Baudrate Parameter**
	0xAA	0xBB	0xCC	0x01AABBCC
Zum Bsp: 1MHz	0x41	0x89	0x00	0x01418900

 © 2007-2015 Analytica GmbH

	MCP2515 CNF1	MCP2515 CNF2	MCP2515 CNF3	Baudrate Parameter
Zum Bsp: 250KHz	0x43	0xA1	0x03	0x0143A103

Tabelle 4.1. Beispielkonfiguration CAN-Baudrate

Anmerkung

Bitte beachten Sie die Beschreibung des MCP2515 hinsichtlich der Kontrollregister CNF1, CNF2 und CNF3. Die Ozillator-Frequenz des AnaGate CAN beträgt FOSC=20MHz, alle anderen CAN-Modelle verwenden die Frequenz FOSC= 24MHz.

nOperatingMode Betriebsmodus, in dem gearbeitet werden soll. Folgende Werte werden unterstützt:

- 0 = Standard-Modus.
- 1 = LoopBack-Modus: In diesem Modus werden keine Telegramme auf dem CAN-Bus versendet. Stattdessen werden an das AnaGate CAN versendete Nachrichten so wieder empfangen, als ob sie von einem anderen Bus-Teilnehmer über den CAN-Bus übertragen worden wären.
- 2 = Listen-Modus: In diesem Modus arbeitet das AnaGate CAN als passiver Partner am Bus, d.h. es werden grundsätzlich keine Telegramme über das Gerät versendet (dies gilt auch bei ACKs für eingehende Telegramme).
- 3 = Offline-Modus: In diesem Modus werden keine Telegramme auf dem CAN-Bus versendet oder empfangen. Dadurch werden auch keine Error-Frames auf dem Bus erzeugt, wenn andere Busteilnehmer Telegramme mit einer anderen als der konfigurierten Baudrate senden.

bTermination Geräte-integrierte CAN-Bus-Terminierung ein- bzw. ausschalten (TRUE=ein, FALSE=aus). Diese Einstellung wird nicht bei allen AnaGate CAN Modellvarianten unterstützt.

bHighSpeedMode Aktuelle Einstellung für den High-Speed-Modus (TRUE=ein, FALSE=aus). Diese Einstellung wird nicht bei allen AnaGate CAN Modellvarianten unterstützt.

Der Highspeed-Modus wurde eingeführt, um auch bei großen Baudraten mit kontinuierlich hoher Buslast keine Pakete zu verlieren. In diesem Modus werden die gesendeten bzw. empfangenen Telegramme auf Protokollebene nicht mehr bestätigt und die via `CANSetFilter` definierten Filter werden ignoriert.

bTimeStampOn Aktuelle Einstellung für den Zeitstempel-Modus (TRUE=ein, FALSE=aus). Diese Einstellung wird nicht bei allen AnaGate CAN Modellvarianten unterstützt.

 © 2007-2015 Analytica GmbH

Im Zeitstempel-Modus wird mit dem CAN-Telegramm ein Zeitstempel übertragen. Beim Senden gibt der Zeitstempel den Zeitpunkt an, zu dem die Nachricht vom CAN-Controller versendet wurde. Analog ist bei empfangenen Nachrichten der Zeitstempel der Zeitpunkt, zu dem die Nachricht vom CAN-Controller empfangen wurde.

Rückgabewert

Die Funktion gibt im Erfolgsfall `Null` zurück, andernfalls einen Fehlercode (Anhang A, *Rückgabewerte aus den API-Funktionen*).

Beschreibung

Setzt die globalen Geräte-Einstellungen der verwendeten CAN-Schnittstelle. Diese Einstellungen sind für alle gleichzeitigen Verbindungen auf der entsprechenden CAN-Schnittstelle gültig. Die Einstellungen bleiben nicht permanent im Gerät gespeichert, sie sind nach einem Geräte-Neustart undefiniert.

Bemerkung

Die Einstellungen für die CAN-Bus-Terminierung, den High-Speed-Modus und den Zeitstempel können beim AnaGate CAN (Hardware-Version 1.1.A) nicht vorgenommen werden. Die Angaben werden vom Gerät ignoriert.

Der Offline-Modus wird erst ab der Geräte-Firmwareversion 1.3.12 unterstützt. Das AnaGate CAN unterstützt diesen Modus generell nicht.

Siehe auch

CANGetGlobals

 © 2007-2015 Analytica GmbH

CANGetGlobals

CANGetGlobals — Ermittelt die globalen Einstellungen, mit denen auf dem CAN Bus gearbeitet wird.

Syntax

#include <AnaGateDllCan.h>

```
AnaInt32 CANGetGlobals(AnaInt32 hHandle, AnaUInt32 * pnBaudrate, AnaUInt8 * pnOperatingMode, AnaInt32 * pbTermination, AnaInt32 * pbHighSpeedMode, AnaInt32 * pbTimeStampOn);
```

Parameter

hHandle Gültiges Zugriffs-Handle eines erfolgreichen Aufrufs von `CANOpenDevice`.

pnBaudrate Aktuell eingestellte Baudrate.

pnOperatingMode Aktuell eingestellter Betriebsmodus. Folgende Werte sind möglich:

- 0 = Standard-Modus.
- 1 = LoopBack-Modus: In diesem Modus werden keine Telegramme auf dem CAN-Bus versendet. Stattdessen werden an das AnaGate CAN versendete Nachrichten so wieder empfangen, als ob sie von einem anderen Bus-Teilnehmer über den CAN-Bus übertragen worden wären.
- 2 = Listen-Modus: In diesem Modus arbeitet das AnaGate CAN als passiver Partner am Bus, d.h. es werden grundsätzlich keine Telegramme über das Gerät versendet (dies gilt auch bei ACKs für eingehende Telegramme).
- 3 = Offline-Modus: In diesem Modus werden keine Telegramme auf dem CAN-Bus versendet oder empfangen. Dadurch werden auch keine Error-Frames auf dem Bus erzeugt, wenn andere Busteilnehmer Telegramme mit einer anderen als der konfigurierten Baudrate senden.

pbTermination Aktuelle Einstellung für CAN-Bus-Terminierung (TRUE=ein, FALSE=aus). Diese Einstellung wird nicht bei allen AnaGate CAN Modellvarianten unterstützt.

pbHighSpeedMode Aktuelle Einstellung für den High-Speed Modus (TRUE=ein, FALSE=aus). Diese Einstellung wird nicht bei allen AnaGate CAN Modellvarianten unterstützt.

Der Highspeed-Modus wurde eingeführt, um auch bei großen Baudraten mit kontinuierlich hoher Buslast keine Pakete zu verlieren. In diesem Modus werden die gesendeten bzw. empfangenen Telegramme auf Protokollebene nicht mehr

 © 2007-2015 Analytica GmbH

bestätigt und die via `CANSetFilter` definierten Filter werden ignoriert.

pbTimeStampOn Aktuelle Einstellung, ob der Zeitstempel-Modus aktiviert ist. Diese Einstellung wird nicht bei allen AnaGate CAN Modellvarianten unterstützt.

Im Zeitstempel-Modus wird mit dem CAN-Telegramm ein Zeitstempel übertragen. Beim Senden gibt der Zeitstempel den Zeitpunkt an, zu dem die Nachricht vom CAN-Controller versendet wurde. Analog ist bei empfangenen Nachrichten der Zeitstempel der Zeitpunkt, zu dem die Nachricht vom CAN-Controller empfangen wurde.

Rückgabewert

Die Funktion gibt im Erfolgsfall `Null` zurück, andernfalls einen Fehlercode (Anhang A, *Rückgabewerte aus den API-Funktionen*).

Beschreibung

Ermittelt die globalen Geräte-Einstellungen der verwendeten CAN-Schnittstelle. Diese Einstellungen sind für alle gleichzeitigen Verbindungen auf der entsprechenden CAN-Schnittstelle gültig.

Bemerkung

Die Einstellungen für die CAN-Bus-Terminierung, den High-Speed-Modus und den Zeitstempel können beim AnaGate CAN (Hardware-Version 1.1.A) nicht vorgenommen werden. Die Angaben werden vom Gerät ignoriert.

Der Offline-Modus wird erst ab der Geräte-Firmwareversion 1.3.12 unterstützt. Das AnaGate CAN unterstützt diesen Modus generell nicht.

Siehe auch

CANGetGlobals

 © 2007-2015 Analytica GmbH

CANSetFilter

CANSetFilter — Setzt die Software-Filter für die aktuelle Verbindung.

Syntax

#include <AnaGateDllCan.h>

```
AnaInt32 CANSetFilter(AnaInt32 hHandle, const AnaUInt32 * pnFilter);
```

Parameter

hHandle Gültiges Zugriffs-Handle eines erfolgreichen Aufrufs von `CANOpenDevice`.

pnFilter Zeiger auf ein Feld mit 8 Fitereinträgen (jeweils 4 Maskenfilter und 4 Bereichsfilter). Ein Filtereintrag besteht grundsätzlich aus zwei 32-Bit-Werten. Es müssen immer alle Filter gleichzeitig gesetzt werden. Sollen Filtereinträge unbenutzt bleiben, sind beim Maskenfilter beide Filterwerte mit 0 und beim Bereichsfilter der Startwert mit 0 und der Endewert mit der höchstmöglichen Zahl (0x1FFFFFFF) zu besetzen.

Rückgabewert

Die Funktion gibt im Erfolgsfall `Null` zurück, andernfalls einen Fehlercode (Anhang A, *Rückgabewerte aus den API-Funktionen*).

Beschreibung

Setzt die Software-Filter für die aktuelle Verbindung. Durch Filter werden nur Nachrichten mit definierten CAN-IDs von der AnaGate CAN Hardware an den jeweiligen Verbindungspartner weitergeleitet.

Ein Maskenfilter besteht aus der Filtermaske, die angibt, welche Bits des CAN-Identifiers geprüft werden sollen, und dem entsprechenden Filterwert. Alle eingehenden Telegramme, die in der Filtermaske nicht mit dem Filterwert übereinstimmen, werden nicht an den Partner weitergeleitet.

Ein Bereichsfilter definiert einen Bereich durch eine Start- und eine Ende-Adresse. Liegt der CAN-Identifier eines Telegramms innerhalb dieses Bereichs, wird die Nachricht an den Partner weitergeleitet.

Standardmäßig sind keine Filter gesetzt, so dass alle CAN-IDs weiter geleitet werden. Wird über die Funktion `CANSetGlobals` der Highspeed-Modus aktiviert, werden alle Filter ignoriert, um den Datensatzdurchsatz zu erhöhen.

CAN ID	Filtermaske	Filterwert	Ergebnis
0x0F	0x0E	0x0C	unterdrückt
0x0C	0x0E	0x0C	ok
0x5D	0x0E	0x0C	ok

Tabelle 4.2. Beispiele für Maskenfilter für CAN Identifier

 © 2007-2015 Analytica GmbH

Im folgenden ein Programmier-Beispiel für das Festlegen von Software-Filtern.

```
#include <AnaGateDllCan.h>
int main()
{
   AnaUInt32 anFilter[16] = {
      0xFF, 0x0F,    // mask  filter 1: mask = 0xFF, value = 0x0F: route only 0x*0F values
      0, 0,          // mask  filter 2: unused
      0, 0,          // mask  filter 3: unused
      0, 0,          // mask  filter 4: unused
      0, 0x00000FFF, // range filter 1: all IDs greater than 0xFFF are discarded
      0, 0x1FFFFFFF, // range filter 2: unused
      0, 0x1FFFFFFF, // range filter 3: unused
      0, 0x1FFFFFFF, // range filter 4: unused
   };
   AnaInt32 hHandle;
   AnaInt32 nRC = CANOpenDevice(&hHandle, TRUE, TRUE, 0, "192.168.1.254", 5000);
   if ( nRC == 0 )
   {
      nRC = CANSetFilter( hHandle, &anFilter );
      // ... now do something
      CANCloseDevice(hHandle);
   }
   return 0;
}
```

Siehe auch

CANGetFilter

© 2007-2015 Analytica GmbH

CANGetFilter

CANGetFilter — Liefert die Filtereinstellungen für die aktuelle Verbindung zurück.

Syntax

#include <AnaGateDllCan.h>

```
AnaInt32 CANGetFilter(AnaInt32 hHandle, AnaUInt32 * pnFilter);
```

Parameter

hHandle Gültiges Zugriffs-Handle eines erfolgreichen Aufrufs von `CANOpenDevice`.

pnFilter Zeiger auf ein Feld mit 8 Filtereinträgen (jeweils 4 Maskenfilter und 4 Bereichsfilter). Ein Filtereintrag besteht grundsätzlich aus zwei 32-Bit-Werten. Bei unbenutzten Maskenfiltereinträgen sind beide Filterwerte mit 0 besetzt. Bei unbenutzten Bereichsfilterwerten ist ein Eintrag mit dem Wertepaar (0,0x1FFFFFFF) besetzt.

Rückgabewert

Die Funktion gibt im Erfolgsfall `Null` zurück, andernfalls einen Fehlercode (Anhang A, *Rückgabewerte aus den API-Funktionen*).

Beschreibung

Liest die aktuelle Einstellung der Software-Filter für die aktuelle Verbindung zurück. Durch Filter werden nur Nachrichten mit definierten CAN-IDs von der AnaGate CAN Hardware an den jeweiligen Verbindungspartner weitergeleitet.

Siehe auch

CANSetFilter

 © 2007-2015 Analytica GmbH

CANSetTime

CANSetTime — Setzt die System-Uhrzeit auf dem AnaGate CAN Gerät für den Zeitstempel-Modus.

Syntax

#include <AnaGateDllCan.h>

`AnaInt32` **`CANSetTime`**`(AnaInt32` *`hHandle`*`, AnaUInt32` *`nSeconds`*`, AnaUInt32` *`nMicroseconds`*`);`

Parameter

hHandle Gültiges Zugriffs-Handle eines erfolgreichen Aufrufs von `CANOpenDevice`.

nSeconds Aktuelle Uhrzeit in Sekunden seit dem 01.01.1970.

nMicroseconds Zusätzlicher Anteil der Mikrosekunden für die aktuelle Uhrzeit.

Rückgabewert

Die Funktion gibt im Erfolgsfall `Null` zurück, andernfalls einen Fehlercode (Anhang A, *Rückgabewerte aus den API-Funktionen*).

Beschreibung

Mit der Funktion `CANSetTime` kann die System-Uhrzeit auf dem AnaGate-Gerät voreingestellt werden. Das Anpassen der System-Uhrzeit ist vor allem dann sinnvoll, falls der Zeitstempel-Modus auf der Verbindung aktiviert ist.

Falls über die Funktion `CANSetGlobals` der Zeitstempel-Modus eingeschaltet worden ist, enthalten alle empfangenen CAN-Nachrichten einen zusätzlichen Zeitstempel, der angibt, zu welchem Zeitpunkt die Nachricht empfangen wurde. Beim Senden von CAN-Nachrichten wird ein entsprechender Zeitstempel über die Quittung des Schreibkommandos an den Sender zurück übermittelt, der angibt, zu welchem Zeitpunkt die Nachricht vom CAN-Controller quittiert wurde (dies nur falls die sog. Confirmations aus Performance-Gesichtspunkten eingeschaltet sind).

Bemerkung

Die Funktion `CANSetTime` ist erst ab Version 1.4-1.8 der Laufzeitbibliothek vorhanden.

Die Uhrzeit-Einstellung für den Zeitstempel-Modus können beim AnaGate CAN (Hardware-Version 1.1.A) nicht vorgenommen werden. Die Angaben werden vom Gerät ignoriert.

Siehe auch

CANGetTime

 © 2007-2015 Analytica GmbH

CANGetTime

CANGetTime — Liest die System-Uhrzeit des AnaGate CAN Geräts für den Zeitstempel-Modus.

Syntax

#include <AnaGateDllCan.h>

AnaInt32 **CANGetTime**(AnaInt32 *hHandle*, AnaInt32 * *pbTimeWasSet*, AnaUInt32 * *pnSeconds*, AnaUInt32 * *pnMicroseconds*);

Parameter

hHandle Gültiges Zugriffs-Handle eines erfolgreichen Aufrufs von `CANOpenDevice`.

pbTimeWasSet Wahr, wenn die Systemzeit zuvor über `CANSetTime` gesetzt wurde.

pnSeconds Aktuelle Uhrzeit in Sekunden seit dem 01.01.1970.

pnMicroseconds Zusätzlicher Anteil der Mikrosekunden für die aktuelle Uhrzeit.

Rückgabewert

Die Funktion gibt im Erfolgsfall `Null` zurück, andernfalls einen Fehlercode (Anhang A, *Rückgabewerte aus den API-Funktionen*).

Beschreibung

Liest die System-Uhrzeit des AnaGate CAN Geräts für den Zeitstempel-Modus.

Falls über die Funktion `CANSetGlobals` der Zeitstempel-Modus eingeschaltet worden ist, enthalten alle empfangenen CAN-Nachrichten einen zusätzlichen Zeitstempel, der angibt, zu welchem Zeitpunkt die Nachricht empfangen wurde. Beim Senden von CAN-Nachrichten wird ein entsprechender Zeitstempel über die Quittung des Schreibkommandos an den Sender zurück übermittelt, der angibt, zu welchem Zeitpunkt die Nachricht vom CAN-Controller quittiert wurde (dies nur falls die sog. Confirmations aus Performance-Gesichtspunkten eingeschaltet sind).

Bemerkung

Die Funktion `CANGetTime` ist erst ab Version 1.7-1.13 der Laufzeitbibliothek vorhanden und wird erst ab der Geräte-Firmwareversion 1.3.16 unterstützt.

Die Uhrzeit-Einstellung für den Zeitstempel-Modus kann beim AnaGate CAN (Hardware-Version 1.1.A) nicht ausgelesen werden. Das Kommando wird vom Gerät ignoriert.

Siehe auch

CANSetTime

 © 2007-2015 Analytica GmbH

CANWrite, CANWriteEx

CANWrite, CANWriteEx — Sendet ein Datentelegramm über die AnaGate-Hardware auf den CAN-Bus.

Syntax

#include <AnaGateDllCan.h>

```
AnaInt32 CANWrite(AnaInt32 hHandle, AnaInt32 nIdentifier, const char *
pcBuffer, AnaInt32 nBufferLen, AnaInt32 nFlags);
```

```
AnaInt32 CANWriteEx(AnaInt32 hHandle, AnaInt32 nIdentifier, const char
* pcBuffer, AnaInt32 nBufferLen, AnaInt32 nFlags, AnaInt32 * pnSeconds,
AnaInt32 * pnMicroSeconds);
```

Parameter

hHandle	Gültiges Zugriffs-Handle eines erfolgreichen Aufrufs von `CANOpenDevice`.
nIdentifier	CAN-Identifier des Absenders. Mittels *`nFlags`* kann definiert werden, ob die Adresse im sog. Extended Format (29-Bit-Adresse) oder Standard-Format (11-Bit-Adresse) vorliegt.
pcBuffer	Zeiger auf einen Datenpuffer mit den Telegrammdaten.
nBufferLen	Länge des Datenpuffers. Größere Werte als 8 werden ignoriert.
nFlags	Mit den Format-Flags kann das Sendeverhalten beeinflusst werden: • Bit 0: Falls gesetzt 29-bit CAN Identifier (Extended Format), sonst 11-bit (Standard format). • Bit 1: Falls gesetzt, wird das Telegramm als Remote-Telegramm versendet. • Bit 2: Falls gesetzt, enthält das Telegramm Timestamp-Informationen. Dieses Bit ist im Regelfall nur bei eingehenden Nachrichten gesetzt. Bei Verwendung der Funktionen `CANWrite` und `CANWriteEx` muss das Bit nicht gesetzt werden.
pnSeconds	Zeitstempel der Quittung vom CAN-Controller in Sekunden seit dem 01.01.1970.
pnMicroSeconds	Anteil der Microsekunden des Zeitstempels.

Rückgabewert

Die Funktion gibt im Erfolgsfall `Null` zurück, andernfalls einen Fehlercode (Anhang A, *Rückgabewerte aus den API-Funktionen*).

 © 2007-2015 Analytica GmbH

Beschreibung

Beide Funktionen senden ein Datentelegramm auf den CAN-Bus.

Mit `CANWriteEx` kann zusätzlich der Zeitpunkt auf dem Gerät ermittelt werden, zu dem das Telegramm tatsächlich versendet wurde.

Anmerkung

Mit Hilfe eines Remoteframes kann ein Teilnehmer einen anderen auffordern, seine Daten zu senden. Die Datenlänge muss entsprechend der zu erwartenden Datenlänge gesetzt werden, auf dem CAN-Bus selbst werden dabei keine Daten versendet.

Bei Verwendung der Funktionen `CANWrite` bzw. `CANWriteEx` ist beim Versenden von Remoteframes sowohl ein Datenpuffer als auch die Länge des Puffers entsprechend der zu erwartenden Datenlänge anzugeben.

Im folgenden ein Programmier-Beispiel, das ein Datentelegramm auf den CAN-Bus sendet.

```
#include <AnaGateDllCan.h>
int main()
{
   char cMsg[] = { 1, 2, 3, 4, 5, 6, 7, 8 };
   AnaInt32 hHandle = 0;
   AnaInt32 nRC = 0;
   AnaInt32 nFlags = 0x0; //  11bit address + standard (not remote frame)
   AnaInt32 nIdentifier = 0x25; // send with CAN ID 0x25;

   AnaInt32 nRC = CANOpenDevice(&hHandle, TRUE, TRUE, 0, "192.168.1.254", 5000);
   if ( nRC == 0 )
   {
      // send 8 bytes with CAN ID 37
      nRC = CANWrite( hHandle, nIdentifier, cMsg, 8, nFlags );

      // send a remote frame to CAN ID 37 (request 4 data bytes)
      nRC = CANWrite( hHandle, nIdentifier, cMsg, 4, 0x02 );

      CANCloseDevice(hHandle);
   }
   return 0;
}
```

Bemerkung

Die Funktion `CANWriteEx` ist erst ab Version 1.4-1.8 der Laufzeitbibliothek vorhanden.

Für Geräte vom Typ AnaGate CAN (Hardware-Version 1.1.A) ist die Funktion `CANWriteEx` mit `CANWrite` identisch. Die Rückgabewerte *`pnSeconds`* und *`pnMicroSeconds`* werden nicht gesetzt.

© 2007-2015 Analytica GmbH

CANSetCallback, CANSetCallbackEx

CANSetCallback, CANSetCallbackEx — Definiert eine asynchrone Callback-Funktion, die beim Empfang eines CAN-Telegramms aus der API aufgerufen werden soll.

Syntax

#include <AnaGateDllCan.h>

```
typedef void (WINAPI * CAN_PF_CALLBACK)(AnaInt32 nIdentifier, const char * pcBuffer, AnaInt32 nBufferLen, AnaInt32 nFlags, AnaInt32 hHandle);
```

```
AnaInt32 CANSetCallback(AnaInt32 hHandle, CAN_PF_CALLBACK pCallbackFunction);
```

```
typedef void (WINAPI * CAN_PF_CALLBACK_EX)(AnaInt32 nIdentifier, const char * pcBuffer, AnaInt32 nBufferLen, AnaInt32 nFlags, AnaInt32 hHandle, AnaInt32 nSeconds, AnaInt32 nMicroseconds);
```

```
AnaInt32 CANSetCallbackEx(AnaInt32 hHandle, CAN_PF_CALLBACK_Ex pCallbackFunctionEx);
```

Parameter

hHandle	Gültiges Zugriffs-Handle eines erfolgreichen Aufrufs von `CANOpenDevice`.
pCallbackFunction	Funktionszeiger auf die selbstdefinierte Callback-Funktion. Zum Deaktivieren der Callback-Funktionalität ist dieser Parameter auf `NULL` zu setzen. Der Aufbau der Funktionsparameter ist der Beschreibung der Funktion `CANWrite` zu entnehmen.
pCallbackFunctionEx	Funktionszeiger auf die selbstdefinierte Callback-Funktion. Zum Deaktivieren der Callback-Funktionalität ist dieser Parameter auf `NULL` zu setzen. Der Aufbau der Funktionsparameter ist der Beschreibung der Funktion `CANWriteEx` zu entnehmen.

Rückgabewert

Die Funktion gibt im Erfolgsfall `Null` zurück, andernfalls einen Fehlercode (Anhang A, *Rückgabewerte aus den API-Funktionen*).

Beschreibung

Eingehende CAN-Telegramme können über eine individuelle Callback-Funktion, die von der API beim Empfang von Daten nebenläufig aufgerufen wird, weiterverarbeitet werden.

Achtung

Die Callback-Funktion wird aus einem Thread aufgerufen, der aus der API gestartet wird, um Daten vom Socket zu lesen. Damit ist beim

© 2007-2015 Analytica GmbH

Abarbeiten des Callback-Codes der Heap-Speicher der API-DLL und nicht der Heap-Speicher des Anwendungsprogrammes aktiv. Aufgrund dieser Arbeitsweise ist Programmcode innerhalb der Callback zu vermeiden, der Heap-Speicher freigibt oder anfordert (z.B. `new`, `delete`, `malloc` oder `free`).

Im folgenden ein Programmier-Beispiel, das eine Empfangs-Callback verwendet.

```
#include <AnaGateDllCan.h>

// Definition of a callback which writes incoming CAN data with timestamp to console
void WINAPI MyCallbackEx(AnaInt32 nIdentifier, const char * pcBuffer, AnaInt32 nBufferLen,
                         AnaInt32 nFlags, AnaInt32 hHandle,
                         AnaInt32 nSeconds, AnaInt32 nMicroseconds)
{
   std::cout << "CAN-ID=" << nIdentifier << ", Data=";
   for ( AnaInt32 i = 0; i < nBufferLen; ++i )
   {
      std::cout << " 0x" << std::hex << int((unsigned char)(pcBuffer[i]));
   }
   time_t tTime = nSeconds;
   struct tm * psLocalTime = localtime(&tTime );
   std::cout << " " << std::setw(19) << asctime( psLocalTime ) << " ms(" << std::dec
             << std::setw(3) << nMicroseconds/1000 << "." << nMicroseconds%1000 << ")" << std::endl;
}

int main()
{
   AnaInt32 hHandle = 0;
   AnaInt32 nRC = 0;

   AnaInt32 nRC = CANOpenDevice(&hHandle, TRUE, TRUE, 0, "192.168.1.254", 5000);
   if ( nRC == 0 )
   {
      // deactivate callback
      nRC = CANSetCallbackEx( hHandle, MyCallbackEx );

      getch();  // wait for keyboard input

      // deactivate callback
      nRC = CANSetCallbackEx( hHandle, 0 );

      CANCloseDevice(hHandle);
   }
   return 0;
}
```

Bemerkung

Die beiden unterschiedlichen Callback-Funktionen können je nachdem, wie die aktuelle globalen Zeitstempel-Einstellung (`CANSetGlobals`) auf dem AnaGate-Gerät ist, verwendet werden. Es ist zu beachten, dass nur eine der beiden Callback-Funktionen aktiviert werden kann.

Siehe auch

CANWrite, CANWriteEx

© 2007-2015 Analytica GmbH

CANSetMaxSizePerQueue

CANSetMaxSizePerQueue — Setzt die maximale Größe des Empfangspuffers, der empfangene CAN-Telegramme zwischenspeichert.

Syntax

#include <AnaGateDllCan.h>

`AnaInt32` **`CANSetMaxSizePerQueue`**`(AnaInt32` *`hHandle`*`, AnaUInt32` *`nMaxSize`*`);`

Parameter

hHandle Gültiges Zugriffs-Handle eines erfolgreichen Aufrufs von `CANOpenDevice`.

nMaxSize Maximale Größe des Empfangspuffers.

Rückgabewert

Die Funktion gibt im Erfolgsfall `Null` zurück, andernfalls einen Fehlercode (Anhang A, *Rückgabewerte aus den API-Funktionen*).

Beschreibung

Setzt die maximale Größe des Empfangspuffers, der empfangene CAN-Telegramme zwischenspeichert. Empfangene Telegramme werden nicht gepuffert, bevor diese Funktion aufgerufen wurde. Nachdem empfangene Telegramme im Puffer gespeichert wurden, können diese mit `CANGetMessage` ausgelesen werden. Falls der Puffer voll ist, während ein neues Telegramm eintrifft, wird dieses verworfen.

Falls die Größe des Empfangspuffers auf 0 gesetzt wird, werden alle zuvor empfangenen Telegramme aus dem Puffer gelöscht. Wird die Größe jedoch auf einen Wert ungleich 0 reduziert, werden überschüssige Telegramme nicht verworfen. Stattdessen werden so lange keine neu empfangenen Telegramme gespeichert, bis genügend viele Einträge mittels `CANGetMessage` entfernt wurden, um Platz für neue Nachrichten zu schaffen, oder bis die Maximalgröße wieder erhöht wird.

Im folgenden ein Programmier-Beispiel, das die Größe des Empfangspuffers setzt.

```
#include <AnaGateDllCan.h>
int main()
{
   AnaInt32 hHandle;
   AnaInt32 nRC = CANOpenDevice(&hHandle, TRUE, TRUE, 0, "192.168.1.254", 5000);
   if ( nRC == 0 )
   {
      nRC = CANSetMaxSizePerQueue( hHandle, 1000 );
      // ... now do something
      CANCloseDevice(hHandle);
   }
   return 0;
}
```

 © 2007-2015 Analytica GmbH

Bemerkung

Empfangene Telegramme werden nur gepuffert, wenn keine Callback-Funktion mittels `CANSetCallback` oder `CANSetCallbackEx` eingerichtet wurde. Sobald eine Callback-Funktion aktiviert ist, können zuvor gepufferte Telegramme weiterhin mit `CANGetMessage` ausgelesen werden. Danach eintreffende Telegramme werden jedoch nicht mehr gepuffert.

Siehe auch

CANSetCallback, CANSetCallbackEx

CANGetMessage

© 2007-2015 Analytica GmbH

CANGetMessage

CANGetMessage — Liefert ein empfangenes CAN-Telegram aus dem Empfangspuffer.

Syntax

#include <AnaGateDllCan.h>

```
AnaInt32 CANGetMessage(AnaInt32 hHandle, AnaUInt32 * pnAvailMsgs,
AnaInt32 * pnID, char * pcData, AnaUInt8 * pnDataLen, AnaInt32 * pnFlags,
AnaInt32 * pnSeconds, AnaInt32 * pnMicroseconds);
```

Parameter

hHandle	Gültiges Zugriffs-Handle eines erfolgreichen Aufrufs von `CANOpenDevice`.
pnAvailMsgs	Zeiger auf eine Variable, in der die Anzahl der verfügbaren Telegramme gespeichert wird, die nach diesem Funktionsaufruf noch gespeichert sind. Falls beim Aufruf dieser Funktion kein Telegramm verfügbar ist, wird diese Variable auf -1 gesetzt. In diesem Fall sind alle folgenden Parameter ungültig.
pnID	Zeiger auf eine Variable, in der die CAN-ID des Telegrams gespeichert wird, es sei denn, der Zeiger ist NULL.
pcData	Zeiger auf einen Puffer, in dem die Daten-Bytes des Telegrams gespeichert werden, es sei denn, der Zeiger ist NULL. Der Puffer muss mindestens 8 Bytes groß sein.
pnDataLen	Zeiger auf eine Variable, in der die Anzahl der Datenbytes des Telegrams gespeichert wird, es sei denn, der Zeiger ist NULL.
pnFlags	Zeiger auf eine Variable, in der die Flags des Telegrams gespeichert werden, es sei denn, der Zeiger ist NULL.
pnSeconds	Zeiger auf eine Variable, in der die Empfangszeit des Telegrams (in Sekunden seit dem 1.1.1970) gespeichert wird, es sei denn, der Zeiger ist NULL.
pnMicroseconds	Zeiger auf eine Variable, in der der Mikrosekunden-Anteil der Empfangszeit des Telegrams gespeichert wird, es sei denn, der Zeiger ist NULL.

Rückgabewert

Die Funktion gibt im Erfolgsfall `Null` zurück, andernfalls einen Fehlercode (Anhang A, *Rückgabewerte aus den API-Funktionen*).

Beschreibung

Liefert ein empfangenes CAN-Telegram aus dem Empfangspuffer. Der Aufrufer stellt zu diesem Zweck Speicherplatz für die Telegramm-Parameter bereit, an denen er

 © 2007-2015 Analytica GmbH

interessiert ist. Für Werte, an denen kein Interesse besteht, kann der jeweilige Parameter auf NULL zeigen.

Über den Parameter pnAvailMsgs liefert die Funktion die Anzahl der nach dem Aufruf noch in der Queue verfügbaren Telegramme. Falls die Queue kein Telegramm enthält, das zurück geliefert werden könnte, wird dieser Wert auf -1 gesetzt. In diesem Fall sind alle Telegramm-Parameter ungültig.

Im folgenden ein Programmier-Beispiel, das ein Telegramm aus dem Empfangspuffer ausliest.

```
#include <AnaGateDllCan.h>
int main()
{
   AnaInt32 hHandle;
   AnaInt32 nRC = CANOpenDevice(&hHandle, TRUE, TRUE, 0, "192.168.1.254", 5000);
   if ( nRC == 0 )
   {
      nRC = CANSetMaxSizePerQueue( hHandle, 1000 );
      // wait so a received telegram can be queued
      Sleep( 5 );
      AnaUInt32 nAvailMsgs;
      AnaInt32 nID;
      char acData[8];
      AnaUInt8 nDataLen;
      AnaInt32 nFlags;
      AnaInt32 nSeconds, nMicroseconds;
      nRC = CANGetMessage( hHandle, &nAvailMsgs, &nID, acData, &nDataLen,
                           &nFlags, &nSeconds, &nMicroseconds );
      if( nAvailMsgs != (AnaUInt32)(-1) && nAvailMsgs > 0 )
      {
         // ... now do something
      }
      CANCloseDevice(hHandle);
   }
   return 0;
}
```

Bemerkung

Empfangene Telegramme werden nur gepuffert, wenn keine Callback-Funktion mittels `CANSetCallback` oder `CANSetCallbackEx` eingerichtet wurde. Sobald eine Callback-Funktion aktiviert ist, können zuvor gepufferte Telegramme weiterhin mit `CANGetMessage` ausgelesen werden. Danach eintreffende Telegramme werden jedoch nicht mehr gepuffert.

Siehe auch

CANSetCallback, CANSetCallbackEx

CANSetMaxSizePerQueue

 © 2007-2015 Analytica GmbH

CANReadDigital

CANReadDigital — Liest die aktuellen Werte der digitalen Ein-/Ausgabe-Register der AnaGate Hardware zurück.

Syntax

#include <AnaGateDllCan.h>

`AnaInt32` **`CANReadDigital`**`(AnaInt32` *`hHandle`*`, AnaUInt32 *` *`pnInputBits`*`, AnaUInt32 *` *`pnOutputBits`*`);`

Parameter

hHandle	Gültiges Zugriffs-Handle eines erfolgreichen Aufrufs von `CANOpenDevice`.
pnInputBits	Zeiger auf den aktuellen Inhalt des Registers mit den digitalen Eingängen. Dabei entsprechen Bit 0 bis Bit 3 den vier digitalen Eingängen; Bit 4 bis Bit 31 sind auf 0 gesetzt.
pnOutputBits	Zeiger auf den aktuellen Inhalt des Registers mit den digitalen Ausgängen. Dabei entsprechen Bit 0 bis Bit 3 den vier digitalen Ausgängen; Bit 4 bis Bit 31 sind auf 0 gesetzt.

Rückgabewert

Die Funktion gibt im Erfolgsfall `Null` zurück, andernfalls einen Fehlercode (Anhang A, *Rückgabewerte aus den API-Funktionen*).

Beschreibung

Alle Geräte der AnaGate-Serie (außer der Gerätevariante AnaGate CAN uno im Hutschienengehäuse) besitzen auf der Gehäuserückseite Anschlüsse für jeweils 4 digitale Eingänge und 4 digitale Ausgänge. Das AnaGate CAN uno im Hutschienengehäuse besitzt dagegen an der Oberseite des Gehäuses Anschlüsse für jeweils 2 digitale Ein- und Ausgänge.

Die aktuellen Zustände der digitalen Eingänge und Ausgänge können mit der Funktion `CANReadDigital` ermittelt werden.

Im folgenden ein Programmier-Beispiel, das die digitalen IOs setzt und zurück liest.

```
#include <AnaGateDllCan.h>
int main()
{
   AnaInt32 hHandle = 0;
   AnaInt32 nRC = 0;
   AnaUInt32 nInputs;
   AnaUInt32 nOutputs = 0x03;

   nRC = CANOpenDevice(&hHandle, TRUE, TRUE, 0, "192.168.1.254", 5000);
   if ( nRC == 0 )
   {
```

 © 2007-2015 Analytica GmbH

```
      // set the digital output register (PIN 0 and PIN 1 to HIGH value)
      nRC = CANWriteDigital( hHandle, nOutputs );

      // read all input and output registers
      nRC = CANReadDigital( hHandle, &nInputs, &nOutputs );

      CANCloseDevice(hHandle);
   }
   return 0;
}
```

Siehe auch

CANWriteDigital

© 2007-2015 Analytica GmbH

CANWriteDigital

CANWriteDigital — Setzt das digitale Ausgabe-Register der AnaGate-Hardware auf einen neuen Wert.

Syntax

#include <AnaGateDllCan.h>

```
AnaInt32 CANWriteDigital(AnaInt32 hHandle, AnaUInt32 nOutputBits);
```

Parameter

hHandle	Gültiges Zugriffs-Handle eines erfolgreichen Aufrufs von `CANOpenDevice`.
nOutputBits	Neuer Wert des Registers mit den digitalen Ausgängen. Dabei entsprechen Bit 0 bis Bit 3 den vier digitalen Ausgängen; Bit 4 bis Bit 31 sind reserviert und auf 0 zu setzen.

Rückgabewert

Die Funktion gibt im Erfolgsfall `Null` zurück, andernfalls einen Fehlercode (Anhang A, *Rückgabewerte aus den API-Funktionen*).

Beschreibung

Alle Geräte der AnaGate-Serie (außer der Gerätevariante AnaGate CAN uno im Hutschienengehäuse) besitzen auf der Gehäuserückseite Anschlüsse für jeweils 4 digitale Eingänge und 4 digitale Ausgänge. Das AnaGate CAN uno im Hutschienengehäuse besitzt dagegen an der Oberseite des Gehäuses Anschlüsse für jeweils 2 digitale Ein- und Ausgänge.

Die digitalen Ausgänge können mit der Funktion `CANWriteDigital` verändert werden.

Ein Programmier-Beispiel zum Lesen/Schreiben der IO ist bei der Beschreibung von `CANReadDigital` zu finden.

Siehe auch

CANReadDigital

 © 2007-2015 Analytica GmbH

CANReadAnalog

CANReadAnalog — Liest die aktuellen Werte der analogen Eingänge der AnaGate-Hardware zurück.

Syntax

#include <AnaGateDllCan.h>

AnaInt32 **CANReadAnalog**(AnaInt32 *hHandle*, AnaUInt32 * *pnPowerSupply*, AnaUInt32 *anAnalogInputs*[], AnaUInt16 * *pnInputCount*);

Parameter

hHandle	Gültiges Zugriffs-Handle eines erfolgreichen Aufrufs von `CANOpenDevice`.
pnPowerSupply	Zeiger auf eine Variable, in der die aktuelle Versorgungsspannung in Millivolt gespeichert wird, es sei denn, der Zeiger ist NULL.
anAnalogInputs	Variablen-Array, in dem die aktuellen Werte der analogen Eingänge in Millivolt gespeichert werden.
pnInputCount	Zeiger auf eine Variable, die beim Aufruf die Anzahl der Elemente von anAnalogInputs enthält. Nach dem Funktionsaufruf enthält die Variable die Anzahl der tatsächlich empfangenen Werte.

Rückgabewert

Die Funktion gibt im Erfolgsfall `Null` zurück, andernfalls einen Fehlercode (Anhang A, *Rückgabewerte aus den API-Funktionen*).

Beschreibung

Die Geräte der AnaGate CAN X-Serie besitzen an der Oberseite des Gehäuses Anschlüsse für jeweils 4 analoge Ein- und Ausgänge.

Die aktuellen Zustände der analogen Eingänge sowie die aktuelle Versorgungsspannung können mit der Funktion `CANReadAnalog` ermittelt werden.

Im folgenden ein Programmier-Beispiel, das die analogen Eingänge liest.

```
#include <AnaGateDllCan.h>
#include <algorithm>
#include <stdio.h>
int main()
{
   AnaInt32 hHandle = 0;
   AnaInt32 nRC = 0;

   nRC = CANOpenDevice(&hHandle, TRUE, TRUE, 0, "192.168.1.254", 5000);
   if ( nRC == 0 )
   {
      const AnaUInt16 INPUT_SIZE = 4;
```

 © 2007-2015 Analytica GmbH

```
      AnaUInt32 nPowerSupply;
      AnaUInt32 anAnalogInputs[INPUT_SIZE];
      AnaUInt16 nInputCount = INPUT_SIZE;
      nRC = CANReadAnalog(nHandle, &nPowerSupply, anAnalogInputs, &nInputCount);
      if ( nRC == 0 )
      {
         printf("Power supply: %d mV\n", nPowerSupply);
         for ( AnaUInt16 i = 0; i < std::min(INPUT_SIZE, nInputCount); ++i )
         {
            printf("Analog input %d: %d mV\n", i, anAnalogInputs[i]);
         }
      }
      CANCloseDevice(hHandle);
   }
   return 0;
}
```

Siehe auch

CANWriteAnalog

© 2007-2015 Analytica GmbH

CANWriteAnalog

CANWriteAnalog — Setzt die analogen Ausgänge der AnaGate-Hardware auf neue Werte.

Syntax

#include <AnaGateDllCan.h>

`AnaInt32` **`CANWriteAnalog`**`(AnaInt32` *`hHandle`*`, AnaUInt32` *`anAnalogOutputs`*`[], AnaUInt16` *`nOutputCount`*`);`

Parameter

hHandle	Gültiges Zugriffs-Handle eines erfolgreichen Aufrufs von `CANOpenDevice`.
anAnalogOutputs	Array mit neuen Werten der analogen Ausgänge in Millivolt.
nOutputCount	Anzahl der Werte in anAnalogOutputs.

Rückgabewert

Die Funktion gibt im Erfolgsfall `Null` zurück, andernfalls einen Fehlercode (Anhang A, *Rückgabewerte aus den API-Funktionen*).

Beschreibung

Die Geräte der AnaGate CAN X-Serie besitzen an der Oberseite des Gehäuses Anschlüsse für jeweils 4 analoge Ein- und Ausgänge.

Die analogen Ausgänge können mit der Funktion `CANWriteAnalog` verändert werden. Die tatsächlich an den analogen Ausgängen anliegende Spannung ist nach oben durch die Versorgungsspannung des AnaGate-Geräts begrenzt. Die aktuelle Versorgungsspannung kann mit der Funktion `CANReadAnalog` ausgelesen werden.

Im folgenden ein Programmier-Beispiel, das die analogen Ausgänge setzt.

```
#include <AnaGateDllCan.h>
int main()
{
   AnaInt32 hHandle = 0;
   AnaInt32 nRC = 0;

   nRC = CANOpenDevice(&hHandle, TRUE, TRUE, 0, "192.168.1.254", 5000);
   if ( nRC == 0 )
   {
      const AnaUInt16 OUTPUT_SIZE = 4;
      AnaUInt32 anAnalogOutputs[OUTPUT_SIZE] = { 0, 9000, 24000, 0 };
      nRC = CANWriteAnalog(nHandle, anAnalogOutputs, OUTPUT_SIZE);

      CANCloseDevice(hHandle);
   }
   return 0;
}
```

 © 2007-2015 Analytica GmbH

Siehe auch

CANReadAnalog

© 2007-2015 Analytica GmbH

CANRestart

CANRestart — Führt einen Geräte-Restart der AnaGate CAN Hardware durch.

Syntax

```
#include <AnaGateDllCan.h>

AnaInt32 CANRestart(const char * pcIPAddress, AnaInt32 nTimeout );
```

Parameter

pcIPAddress	Netzwerkadresse des AnaGate Partners.
nTimeout	Standard-Timeout für AnaGate-Zugriffe in Millisekunden. Ein Timeout wird festgestellt, wenn der AnaGate-Partner nicht innerhalb der vereinbarten Timeout-Zeit antwortet.

Rückgabewert

Die Funktion gibt im Erfolgsfall `Null` zurück, andernfalls einen Fehlercode (Anhang A, *Rückgabewerte aus den API-Funktionen*).

Beschreibung

Führt einen Wiederhochlauf der AnaGate CAN Hardware unter der angegebenen Netzwerkadresse durch und schließt damit alle noch offenen Netzwerkverbindungen. Das Restart-Kommando ist auch dann noch möglich, wenn die maximale Anzahl von gleichzeitigen Verbindungen bereits erreicht ist.

Wichtig

Dieses Kommando sollte nur verwendet werden, falls eine Verbindung zum Gerät notwendig, aber aktuell nicht möglich ist, da die maximale Anzahl gleichzeitiger Verbindungen bereits erreicht ist.

Siehe auch

CANOpenDevice

© 2007-2015 Analytica GmbH

CANDeviceConnectState

CANDeviceConnectState — Ermittelt den Verbindungsstatus der aktuellen Netzwerk-Verbindung zur AnaGate-Hardware.

Syntax

#include <AnaGateDllCan.h>

```
AnaInt32 CANDeviceConnectState(AnaInt32 hHandle);
```

Parameter

hHandle Gültiges Zugriffs-Handle eines erfolgreichen Aufrufs von `CANOpenDevice`.

Rückgabewert

Gibt den aktuellen Status der Netzwerkverbindung zurück. Folgende Werte sind möglich:

- `1 = DISCONNECTED`: Die Verbindung ist nicht hergestellt.
- `2 = CONNECTING`: Die Verbindung wird aufgebaut.
- `3 = CONNECTED` : Die Verbindung ist hergestellt.
- `4 = DISCONNECTING`: Die Verbindung wird abgebaut.
- `5 = NOT_INITIALIZED`: Das Netzwerkprotokoll ist nicht vollständig initialisiert.

Beschreibung

Die Funktion ermittelt den aktuellen Verbindungsstatus der Netzwerk-Verbindung zur AnaGate-Hardware. Über periodische Aufrufe dieser Funktion kann vom Anwendungsprogramm ermittelt werden, ob ein Verbindungsabbruch stattgefunden hat.

Wie schnell ein Abbruch der Verbindung erkannt wird, hängt stark davon ab, ob der AnaGate-ALIVE-Mechnismus aktiv oder nur der Keepalive-Mechanismus auf Netzwerk-Protokollebene verwendet wird. Der anwendungspezifische ALIVE-Mechanismus [TCP-2010] wird über die DLL-Funktion `CANStartAlive` eingeschaltet und überprüft den Verbindungsstatus periodisch innerhalb eines vorgegebenen Zeitintervalls.

Bemerkung

Die Funktion `CANDeviceConnectState` ist erst ab Version 1.4-1.10 der Laufzeitbibliothek vorhanden.

Siehe auch

CANStartAlive

 © 2007-2015 Analytica GmbH

CANStartAlive

CANStartAlive — Startet den ALIVE Mechanismus, der periodisch den Status der aktiven Netzwerk-Verbindung zur AnaGate-Hardware überprüft.

Syntax

#include <AnaGateDllCan.h>

```
AnaInt32 CANStartAlive(AnaInt32 hHandle, AnaUInt32 nAliveTime );
```

Parameter

hHandle	Gültiges Zugriffs-Handle eines erfolgreichen Aufrufs von `CANOpenDevice`.
nAliveTime	Timeout-Intervall in Sekunden für den ALIVE-Mechanismus.

Rückgabewert

Die Funktion gibt im Erfolgsfall `Null` zurück, andernfalls einen Fehlercode (Anhang A, *Rückgabewerte aus den API-Funktionen*).

Beschreibung

Das AnaGate-Kommunikationsprotokoll (siehe [TCP-2010]) unterstützt eine anwendungsspezifische Verbindungsüberwachung, über die ungeplante Verbindungsabbrüche schneller erkannt werden können.

Die `CANStartAlive`-Funktion startet innerhalb der API einen nebenläufigen Thread, der periodisch definierte Alive-Telegramme (ALIVE_REQ) über die aktuelle Netzwerkverbindung mit der AnaGate-Hardware austauscht. Wird das Alive-Telegramm nicht innerhalb der vorgegebenen Intervallzeit vom Partner quittiert, wird die Verbindung als `disconnected` markiert und explizit abgebaut.

Mit der Funktion `CANDeviceConnectState` kann der Netzwerk-Verbindungsstatus überprüft werden.

Bemerkung

Die Funktion `CANStartAlive` ist erst ab Version 1.4-1.10 der Laufzeitbibliothek vorhanden.

Auf der Hardware-Seite wird mindestens die Firmware-Version 1.3.8 oder höher vorausgesetzt. Geräte vom Typ AnaGate CAN (Hardware-Version 1.1.A) unterstützen den anwendungsspezifischen Alive-Mechanismum nicht.

Siehe auch

CANDeviceConnectState

Abschnitt 2.1, „ Besondere Protokolleigenschaften “

 © 2007-2015 Analytica GmbH

CANErrorMessage

CANErrorMessage — Gibt eine textuelle Beschreibung eines Rückgabewertes aus einer API-Funktion zurück.

Syntax

#include <AnaGateDllCan.h>

AnaInt32 **CANErrorMessage**(AnaInt32 *nRetCode*, char * *pcMessage*, AnaInt32 *nMessageLen*);

Parameter

nRetCode Error-Code, dessen Fehlerbeschreibung ermittelt werden soll.

pcMessage Datenpuffer, der die Fehlerbeschreibung aufnehmen soll.

nMessageLen Größe des übergebenen Datenpuffers in Byte.

Rückgabewert

Tatsächliche Größe der zurückgegebenen Beschreibung in Byte.

Beschreibung

Gibt eine textuelle Beschreibung des übergebenen Rückgabewertes zurück (siehe auch Anhang A, *Rückgabewerte aus den API-Funktionen*). Ist der Ziel-Datenpuffer nicht groß genug, um den Fehlertext aufzunehmen, wird der Text auf die angegebene Puffergröße gekürzt. Alle Fehlertexte sind in englischer Sprache.

Im folgenden ein Programmier-Beispiel in C/C++.

```
#include <AnaGateDllCan.h>
// ...
AnaInt32 nRC;
char cBuffer[200];
//... call an API function here
CANErrorMessage(nRC, cBuffer, sizeof cBuffer);
std::cout << "Fehler: " << cBuffer << std::endl;
```


© 2007-2015 Analytica GmbH

Kapitel 5. SPI API Funktionen

Der Serial Peripheral Interface (kurz SPI) ist ein von Motorola entwickeltes Bus-System für einen synchronen seriellen Datenbus, mit dem digitale Schaltungen nach dem Master-Slave-Prinzip miteinander verbunden werden können. Die SPI-Gateways aus der AnaGate-Serie bieten einen Zugriff auf den SPI Bus über einen herkömmlichen Netzwerkanschluss.

Mit den Funktionen der SPI API können diese SPI-Gateways und damit der SPI Bus auf einfache Art und Weise angesprochen werden. Die Programmierschnittstelle ist für alle Geräte identisch und erfolgt generell über das Netzwerk-Protokoll TCP.

Aktuell können folgende Geräte über die SPI API genutzt werden:

- AnaGate SPI
- AnaGate Universal Programmer

© 2007-2015 Analytica GmbH

SPIOpenDevice

SPIOpenDevice — Baut eine Netzwerkverbindung zu einem AnaGate SPI auf.

Syntax

#include <AnaGateDllSPI.h>

```
AnaInt32 SPIOpenDevice(AnaInt32 * pHandle, const char * pcIPAddress,
AnaInt32 nTimeout);
```

Parameter

pHandle	Zeiger auf eine Variable, in die das Zugriffs-Handle gespeichert wird, falls die Verbindung zum Gerät erfolgreich hergestellt wurde.
pcIPAddress	Netzwerkadresse des AnaGate Partners.
nTimeout	Standard-Timeout für AnaGate-Zugriffe in Millisekunden. Ein Timeout wird festgestellt, wenn die AnaGate-Hardware nicht innerhalb der vereinbarten Timeout-Zeit antwortet. Diese Timeout-Zeit gilt auf der aktiven Netzwerkverbindung für alle Kommandos bzw. Funktionen, für die kein spezifischer Timeout-Wert definiert werden kann.

Rückgabewert

Die Funktion gibt im Erfolgsfall `Null` zurück, andernfalls einen Fehlercode (Anhang A, *Rückgabewerte aus den API-Funktionen*).

Beschreibung

Baut eine Netzwerkverbindung über TCP/IP zu einem AnaGate SPI (bzw. AnaGate Universal Programmer) auf. Erst nach dem erfolgreichen Verbinden mit dem Gerät ist ein Zugriff auf den SPI-Bus möglich.

Anmerkung

Das AnaGate SPI (bzw. die SPI-Schnittstelle eines AnaGate Universal Programmers) erlaubt nur eine einzige Netzwerkverbindung. Solange eine bestehende Verbindung aufrechterhalten wird, wird jeder neue Verbindungsversuch abgelehnt.

Im folgenden ein Programmier-Beispiel für den initialen Zugriff auf das Gerät.

```
#include <AnaGateDllSPI.h>
int main()
{
   AnaInt32 hHandle;
   AnaInt32 nRC = SPIOpenDevice(&hHandle, "192.168.1.254", 5000);
   if ( nRC == 0 )
   {
```

 © 2007-2015 Analytica GmbH

```
      // ... now do something
      SPICloseDevice(hHandle);
   }
   return 0;
}
```

Siehe auch

SPICloseDevice

© 2007-2015 Analytica GmbH

SPICloseDevice

SPICloseDevice — Schließt eine geöffnete Netzwerk-Verbindung zu einem AnaGate SPI Device.

Syntax

#include <AnaGateDllSPI.h>

```
AnaInt32 SPICloseDevice(AnaInt32 hHandle);
```

Parameter

hHandle Gültiges Zugriffs-Handle eines erfolgreichen Aufrufs von `SPIOpenDevice`.

Rückgabewert

Die Funktion gibt im Erfolgsfall `Null` zurück, andernfalls einen Fehlercode (Anhang A, *Rückgabewerte aus den API-Funktionen*).

Beschreibung

Schließt eine geöffnete Netzwerk-Verbindung zu einem AnaGate SPI Device. Das übergebene Handle *`hHandle`* ist ein Rückgabewert aus einem vorangegangenen erfolgreichen Aufruf der Funktion `SPIOpenDevice`.

Wichtig

Das explizite Schließen der Verbindung ist notwendig, um die in der DLL angelegten Systemressourcen wieder freizugeben und dem verbundenen Gerät mitzuteilen, dass die Verbindung nicht mehr genutzt wird und wieder für neue Verbindungsanfragen zur Verfügung gestellt werden soll.

Siehe auch

SPIOpenDevice

© 2007-2015 Analytica GmbH

SPISetGlobals

SPISetGlobals — Setzt die globalen Einstellungen, mit denen auf dem AnaGate SPI gearbeitet werden soll.

Syntax

#include <AnaGateDllSPI.h>

```
AnaInt32 SPISetGlobals(AnaInt32 hHandle, AnaInt32 nBaudrate, AnaUInt8 nSigLevel, AnaUInt8 nAuxVoltage, AnaUInt8 nClockMode);
```

Parameter

hHandle Gültiges Zugriffs-Handle eines erfolgreichen Aufrufs von `SPIOpenDevice`.

nBaudrate Baudrate, mit der gearbeitet werden soll. Werte können individuell eingestellt werden, z.B.

- 500.000 für 500 kBit
- 1.000.000 für 1 MBit
- 5.000.000 für 5 MBit

Anmerkung

Die gewünschte Baudrate kann u.U. von dem tatsächlich verwendeten Wert abweichen, da die Taktfrequenz auf dem Gerät nur in Abhängigkeit der Hardware (Schwingungsdauer des Quarz) eingestellt werden kann. Ist die exakte Einstellung einer angegebenen Baudrate nicht möglich, wird der nächstkleinere mögliche Wert eingestellt.

nSigLevel Gibt den Pegelwert für SPI-Signale an. Folgende Werte werden unterstützt:

- 0 für Ausgänge im High Impedance Modus (Standard-Modus).
- 1 für +5,0 V für die Signale.
- 2 für +3,3 V für die Signale.
- 3 für +2,5 V für die Signale.

nAuxVoltage Gibt die Ausgangsspannung für die Hilfsspannungsversorgung an. Folgende Werte sind möglich:

- 0 für Hilfsspannung +3,3 V.
- 1 für Hilfsspannung +2,5 V für die Signale.

nClockMode Gibt die Phase und die Polarität der Clock-Leitung für die Datenübertragung an. Folgende Werte sind möglich:

 © 2007-2015 Analytica GmbH

- 0 für CPHA=0 und CPOL=0.
- 1 für CPHA=0 und CPOL=1.
- 2 für CPHA=1 und CPOL=0.
- 3 für CPHA=1 und CPOL=1.

Rückgabewert

Die Funktion gibt im Erfolgsfall `Null` zurück, andernfalls einen Fehlercode (Anhang A, *Rückgabewerte aus den API-Funktionen*).

Beschreibung

Setzt die globalen Geräte-Einstellungen mit denen auf der SPI-Schnittstelle des AnaGate SPI bzw. AnaGate Universal Programmers gearbeitet werden soll. Die Einstellungen bleiben nicht permanent im Gerät gespeichert, sie sind nach einem Geräte-Neustart undefiniert.

Siehe auch

SPIGetGlobals

 © 2007-2015 Analytica GmbH

SPIGetGlobals

SPIGetGlobals — Ermittelt die globalen Einstellungen, mit denen auf dem AnaGate SPI gearbeitet wird.

Syntax

#include <AnaGateDllSPI.h>

```
AnaInt32 SPIGetGlobals(AnaInt32 hHandle, AnaInt32 * pnBaudrate, AnaUInt8
* pnSigLevel, AnaUInt8 * pnAuxVoltage, AnaUInt8 * pnClockMode);
```

Parameter

hHandle Gültiges Zugriffs-Handle eines erfolgreichen Aufrufs von `SPIOpenDevice`.

pnBaudrate Aktuell auf dem SPI-Bus eingestellte Baudrate in Bit pro Sekunde.

pnSigLevel Aktuell eingestellter Pegelwert für die SPI Signale. Folgende Werte sind möglich:

- 0 für Ausgänge im High Impedance Modus (Standard-Modus).
- 1 für +5,0 V für die Signale.
- 2 für +3,3 V für die Signale.
- 3 für +2,5 V für die Signale.

pnAuxVoltage Aktuell eingestellte Ausgangsspannung für die Hilfsspannungsversorgung. Folgende Werte sind möglich:

- 0 für Hilfsspannung +3,3 V.
- 1 für Hilfsspannung +2,5 V.

pnClockMode Aktuell eingestellter Clock-Modus (Phase und Polarität der Clock-Leitung für die Datenübertragung). Folgende Werte sind möglich:

- 0 für CPHA=0 und CPOL=0.
- 1 für CPHA=0 und CPOL=1.
- 2 für CPHA=1 und CPOL=0.
- 3 für CPHA=1 und CPOL=1.

Rückgabewert

Die Funktion gibt im Erfolgsfall `Null` zurück, andernfalls einen Fehlercode (Anhang A, *Rückgabewerte aus den API-Funktionen*).

Beschreibung

Ermittelt die globalen Geräte-Einstellungen, mit denen auf der SPI-Schnittstelle des AnaGate SPI bzw. AnaGate Universal Programmers gearbeitet wird.

 © 2007-2015 Analytica GmbH

Siehe auch

SPISetGlobals

© 2007-2015 Analytica GmbH

SPIDataReq

SPIDataReq — Führt einen Datentransfer auf dem SPI-Bus durch.

Syntax

#include <AnaGateDllSPI.h>

```
AnaInt32 SPIDataReq(AnaInt32 hHandle, const char * pcBufWrite, AnaInt32 nBufWriteLen, char * pcBufRead, AnaInt32 nBufReadLen);
```

Parameter

hHandle	Gültiges Zugriffs-Handle eines erfolgreichen Aufrufs von `SPIOpenDevice`.
pcBufWrite	Puffer mit den Daten, die an den SPI-Partner gesendet werden sollen.
nBufWriteLen	Länge des Datenpuffers *`pcBufWrite`* (Anzahl Bytes).
pcBufRead	Byte-Puffer, der die empfangenen Daten, die vom SPI-Partner gesendet werden, aufnehmen soll.
nBufReadLen	Anzahl der Bytes, die gelesen werden sollen. Darf die Länge des Datenpuffers *`pcBufRead`* nicht überschreiten.

Beschreibung

Sendet Daten auf den SPI-Bus und empfängt Daten vom SPI-Bus.

Daten werden auf dem SPI-Bus immer auf zwei Leitungen voll-duplex (SDO und SDI) übertragen. Die Funktion `SPIDatReq` arbeitet bedingt durch die räumliche Trennung zum SPI-Bus jedoch zwangsläufig in zwei Schritten. Zuerst wird der Schreibdatenpuffer in einem Netzwerkdatenpaket an das AnaGate SPI gesendet, das dann den eigentlichen Datentransfer auf dem SPI-Bus durchführt. Nach erfolgreicher Kommunikation auf dem SPI-Bus sendet das Anagate SPI eine Quittung mit den gelesenen Daten zurück, die dann im Lesedatenpuffer abgelegt werden.

Wichtig

Es ist hardwaretechnisch nicht möglich zu erkennen, ob tatsächlich ein Baustein am SPI-Bus angeschlossen ist. Auch wenn kein Baustein angeschlossen ist, wird vom AnaGate SPI die angeforderte Anzahl von Datenbytes zurückgeliefert - der Lese-Datenbuffer wird in diesem Fall mit Null-Werten aufgefüllt.

Im folgenden ein Programmier-Beispiel, das Daten auf dem SPI-Bus sendet und empfängt.

```
#include <AnaGateDllSPI.h>
int main()
{
```

 © 2007-2015 Analytica GmbH

```
   char cBufWrite[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
   char cBufReceive[100];
   AnaInt32 hHandle = 0;
   AnaInt32 nRC = 0;

   AnaInt32 nRC = SPIOpenDevice(&hHandle, "192.168.1.254", 5000);
   if ( nRC == 0 )
   {
      // send 1 byte and receive 1 byte
      nRC = SPIDataReq( hHandle, cBufWrite, 1, cBufReceive, 1 );
      // send 1 byte and receive 5 byte
      nRC = SPIDataReq( hHandle, cBufWrite, 1, cBufReceive, 5 );
      // send 2 byte and receive 1 byte
      nRC = SPIDataReq( hHandle, cBufW2ite, 2, cBufReceive, 1 );

      SPICloseDevice(hHandle);
   }
   return 0;
}
```

Rückgabewert

Die Funktion gibt im Erfolgsfall `Null` zurück, andernfalls einen Fehlercode (Anhang A, *Rückgabewerte aus den API-Funktionen*).

© 2007-2015 Analytica GmbH

SPIReadDigital

SPIReadDigital — Liest die aktuellen Werte der digitale Ein-/Ausgabe-Register der AnaGate-Hardware zurück.

Syntax

#include <AnaGateDllSPI.h>

`AnaInt32 **SPIReadDigital**(AnaInt32 *hHandle*, AnaUInt32 * *pnInputBits*, AnaUInt32 * *pnOutputBits*);`

Parameter

hHandle	Gültiges Zugriffs-Handle eines erfolgreichen Aufrufs von `SPIOpenDevice`.
pnInputBits	Zeiger auf den aktuellen Inhalt des Registers mit den digitalen Eingängen. Dabei entsprechen Bit 0 bis Bit 3 den vier digitalen Eingängen; Bit 4 bis Bit 31 sind auf 0 gesetzt.
pnOutputBits	Zeiger auf den aktuellen Inhalt des Registers mit den digitalen Ausgängen. Dabei entsprechen Bit 0 bis Bit 3 den vier digitalen Ausgängen; Bit 4 bis Bit 31 sind auf 0 gesetzt.

Rückgabewert

Die Funktion gibt im Erfolgsfall `Null` zurück, andernfalls einen Fehlercode (Anhang A, *Rückgabewerte aus den API-Funktionen*).

Beschreibung

Alle Geräte der AnaGate-Serie (außer der Gerätevariante AnaGate CAN uno im Hutschienengehäuse) besitzen auf der Gehäuserückseite Anschlüsse für jeweils 4 digitale Eingänge und 4 digitale Ausgänge.

Die aktuellen Zustände der digitalen Eingänge und Ausgänge können mit der Funktion `SPIReadDigital` ermittelt werden.

Im folgenden ein Programmier-Beispiel, das die digitalen IOs setzt und zurückliest.

```
#include <AnaGateDllSPI.h>
int main()
{
   AnaInt32 hHandle = 0;
   AnaUInt32 nInputs;
   AnaUInt32 nOutputs = 0x03;

   AnaInt32 nRC = SPIOpenDevice(&hHandle, "192.168.1.254", 5000);
   if ( nRC == 0 )
   {
      // set the digital output register (PIN 0 and PIN 1 to HIGH value)
      nRC = SPIWriteDigital( hHandle, nOutputs );
```

 © 2007-2015 Analytica GmbH

```
      // read all input and output registers
      nRC = SPIReadDigital( hHandle, &nInputs, &nOutputs );

      SPICloseDevice(hHandle);
   }
   return 0;
}
```

Siehe auch

SPIWriteDigital

© 2007-2015 Analytica GmbH

SPIWriteDigital

SPIWriteDigital — Setzt das digitale Ausgabe-Register der AnaGate-Hardware auf einen neuen Wert.

Syntax

#include <AnaGateDllSPI.h>

```
AnaInt32 SPIWriteDigital(AnaInt32 hHandle, AnaUInt32 nOutputBits);
```

Parameter

hHandle	Gültiges Zugriffs-Handle eines erfolgreichen Aufrufs von `CANOpenDevice`.
nOutputBits	Neuer Wert des Registers mit den digitalen Ausgängen. Dabei entsprechen Bit 0 bis Bit 3 den vier digitalen Ausgängen; Bit 4 bis Bit 31 sind reserviert und auf 0 zu setzen.

Rückgabewert

Die Funktion gibt im Erfolgsfall `Null` zurück, andernfalls einen Fehlercode (Anhang A, *Rückgabewerte aus den API-Funktionen*).

Beschreibung

Alle Geräte der AnaGate-Serie (außer der Gerätevariante AnaGate CAN uno im Hutschienengehäuse) besitzen auf der Gehäuserückseite Anschlüsse für jeweils 4 digitale Eingänge und 4 digitale Ausgänge.

Die digitalen Ausgänge können mit der Funktion `SPIWriteDigital` verändert werden.

Ein Programmier-Beispiel zum Lesen/Schreiben der IOs ist bei der Beschreibung von `SPIReadDigital` zu finden.

Siehe auch

SPIReadDigital

© 2007-2015 Analytica GmbH

SPIErrorMessage

SPIErrorMessage — Gibt eine textuelle Beschreibung eines Rückgabewertes aus einer API-Funktion zurück.

Syntax

#include <AnaGateDllSPI.h>

AnaInt32 **SPIErrorMessage**(AnaInt32 *nRetCode*, char * *pcMessage*, AnaInt32 *nMessageLen*);

Parameter

nRetCode Error-Code, dessen Fehlerbeschreibung ermittelt werden soll.

pcMessage Datenpuffer, der die Fehlerbeschreibung aufnehmen soll.

nMessageLen Größe des übergebenen Datenpuffers in Byte.

Rückgabewert

Tatsächliche Größe der zurückgegebenen Beschreibung in Bytes.

Beschreibung

Gibt eine textuelle Beschreibung des übergebenen Rückgabewertes zurück (siehe auch Anhang A, *Rückgabewerte aus den API-Funktionen*). Ist der Ziel-Datenpuffer nicht groß genug, um den Fehlertext aufzunehmen, wird der Text auf die angegebene Puffergröße gekürzt. Alle Fehlertexte sind in englischer Sprache.

Im folgenden ein Programmier-Beispiel in C/C++.

```
#include <AnaGateDllSPI.h>
// ...
AnaInt32 nRC;
char cBuffer[200];
//... call a API function here
SPIErrorMessage(nRC, cBuffer, sizeof cBuffer);
std::cout << "Fehler: " << cBuffer << std::endl;
```


© 2007-2015 Analytica GmbH

Kapitel 6. I2C API Funktionen

I2C, für engl. Inter-Integrated Circuit, im Deutschen gesprochen als I-Quadrat-C, ist ein von Philips Semiconductors (heute NXP Semiconductors) entwickelter serieller Datenbus. Er wird hauptsächlich geräteintern für die Kommunikation zwischen verschiedenen Schaltungsteilen mit geringer Übertragungsgeschwindigkeit benutzt, z. B. zwischen einem Controller und Peripherie-ICs. Technisch benötigt I2C lediglich zwei Signalleitungen: Takt (engl. serial clock line, SCL) und Datenleitung (engl. serial data line, SDA). Der Datentransfer kann bidirektional 8-bit orientiert erfolgen, und zwar mit bis zu 100 kbit/s im Standard-Modus, bis zu 400 kbit/s in Fast-mode, bis zu 1 Mbit/s in Fast-mode Plus (Fm+) oder bis zu 3.4 Mbit/s im High-speed Modus.[NXP-I2C]

Einige Hersteller verwenden die Bezeichnung TWI (Two-Wire Interface), obwohl I2C kein eingetragenes Markenzeichen von NXP Semiconductors ist. Technisch sind TWI und I2C identisch.

Die I2C-Gateways aus der AnaGate-Serie bieten einen Zugriff auf den I2C-Bus über einen herkömmlichen Netzwerkanschluss. Mit den Funktionen der I2C API können diese I2C-Gateways und damit der I2C-Bus auf einfache Art und Weise angesprochen werden. Die Programmierschnittstelle ist für alle Geräte identisch und erfolgt generell über das Netzwerk-Protokoll TCP.

Aktuell können folgende Geräte über die I2C API genutzt werden:

- AnaGate I2C
- AnaGate Universal Programmer

© 2007-2015 Analytica GmbH

I2COpenDevice

I2COpenDevice — Baut eine Netzwerkverbindung zu einem AnaGate I2C (bzw. AnaGate Universal Programmer) auf.

Syntax

#include <AnaGateDllI2C.h>

```
AnaInt32 I2COpenDevice(AnaInt32 * pHandle, AnaUInt32 nBaudrate, const char * pcIPAddress, AnaInt32 nTimeout);
```

Parameter

pHandle — Zeiger auf eine Variable, in die das Zugriffs-Handle gespeichert wird, falls die Verbindung zum Device erfolgreich hergestellt wurde.

nBaudrate — Baudrate, mit der der I2C-Bus betrieben werden soll. Die Werte können individuell eingestellt werden, z.B.

- 100000 für 100 kBit (Standard Mode)
- 400000 für 400 kBit (Fast Mode)

Anmerkung

Größere Werte als 400 kBit werden auf dem AnaGate SPI ignoriert.

pcIPAddress — Netzwerkadresse des AnaGate Partners.

nTimeout — Standard-Timeout für AnaGate-Zugriffe in Millisekunden.

Ein Timeout wird festgestellt, wenn die AnaGate-Hardware nicht innerhalb der vereinbarten Timeout-Zeit antwortet. Diese Timeout-Zeit gilt auf der aktiven Netzwerkverbindung für alle Kommandos bzw. Funktionen, für die kein spezifischer Timeout-Wert definiert werden kann.

Rückgabewert

Die Funktion gibt im Erfolgsfall `Null` zurück, andernfalls einen Fehlercode (Anhang A, *Rückgabewerte aus den API-Funktionen*).

Beschreibung

Baut eine Netzwerkverbindung über TCP/IP zu einem AnaGate I2C (bzw. AnaGate Universal Programmer) auf. Erst nach dem erfolgreichen Verbinden mit dem Gerät ist ein Zugriff auf den I2C-Bus möglich.

Anmerkung

Das AnaGate I2C (bzw. die I2C-Schnittstelle eines AnaGate Universal Programmers) erlaubt nur eine einzige Netzwerkverbindung. Solange

© 2007-2015 Analytica GmbH

eine bestehende Verbindung aufrechterhalten wird, wird jeder neue Verbindungsversuch abgelehnt.

Im folgenden ein Programmier-Beispiel für den initialen Zugriff auf das Gerät.

```
#include <AnaGateDllI2C.h>
int main()
{
   AnaInt32 hHandle;
   AnaInt32 nRC = I2COpenDevice(&hHandle, 100000, "192.168.1.254", 5000);
   if ( nRC == 0 )
   {
      // ... now do something
      I2CCloseDevice(hHandle);
   }
   return 0;
}
```

Siehe auch

I2CCloseDevice

© 2007-2015 Analytica GmbH

I2CCloseDevice

I2CCloseDevice — Schließt eine geöffnete Netzwerk-Verbindung zu einem AnaGate I2C Device.

Syntax

#include <AnaGateDllI2C.h>

```
AnaInt32 I2CCloseDevice(AnaInt32 hHandle);
```

Parameter

hHandle Gültiges Zugriffs-Handle eines erfolgreichen Aufrufs von `I2COpenDevice`.

Rückgabewert

Die Funktion gibt im Erfolgsfall `Null` zurück, andernfalls einen Fehlercode (Anhang A, *Rückgabewerte aus den API-Funktionen*).

Beschreibung

Schließt eine geöffnete Netzwerk-Verbindung zu einem AnaGate I2C Device. Das übergebene Handle *`hHandle`* ist ein Rückgabewert aus einem vorangegangenen erfolgreichen Aufruf der Funktion `I2COpenDevice`.

Wichtig

Das explizite Schließen der Verbindung ist notwendig, um die in der DLL angelegten Systemressourcen wieder freizugeben und dem verbundenen Gerät mitzuteilen, dass die Verbindung nicht mehr genutzt wird und wieder für neue Verbindungsanfragen zur Verfügung gestellt werden soll.

Siehe auch

I2COpenDevice

© 2007-2015 Analytica GmbH

I2CReset

I2CReset — Setzt den I2C-Controller auf dem AnaGate I2C Device zurück.

Syntax

```
#include <AnaGateDllI2C.h>

AnaInt32 I2CReset(AnaInt32 hHandle);
```

Parameter

hHandle Gültiges Zugriffs-Handle eines erfolgreichen Aufrufs von `I2COpenDevice`.

Rückgabewert

Die Funktion gibt im Erfolgsfall `Null` zurück, andernfalls einen Fehlercode (Anhang A, *Rückgabewerte aus den API-Funktionen*).

Beschreibung

Setzt den I2C-Controller auf dem AnaGate I2C Device zurück.

© 2007-2015 Analytica GmbH

I2CRead

I2CRead — Liest Daten von einem I2C-Partner.

Syntax

#include <AnaGateDllI2C.h>

```
AnaInt32 I2CRead(AnaInt32 hHandle, AnaUInt16 nSlaveAddress, const char
* pcBuffer, AnaInt32 nBufferLen);
```

Parameter

hHandle	Gültiges Zugriffs-Handle eines erfolgreichen Aufrufs von `I2COpenDevice`.
nSlaveAddress	Slave-Adresse des I2C-Partners. Die Slave-Adresse kann eine sog. 7 oder 10-Bit Adresse darstellen (siehe Anhang B, *Adressierung auf dem I2C-Bus*).
pcBuffer	Byte-Puffer, der die vom I2C-Partner empfangenen Daten aufnehmen soll.
nBufferLen	Anzahl der Bytes, die gelesen werden sollen.

Rückgabewert

Die Funktion gibt im Erfolgsfall `Null` zurück, andernfalls einen Fehlercode (Anhang A, *Rückgabewerte aus den API-Funktionen*).

Beschreibung

Liest Daten von einem I2C-Baustein (Slave), die AnaGate-Hardware fungiert dabei als I2C-Master am Bus.

Die AnaGate-Hardware started die Übertragung mit dem **Start**-Signal, gefolgt von der angegebenen Slave-Adresse (das R/W-Bit der Slave-Adresse muss vom Anwender nicht explizit auf `1` gesetzt werden, es wird bei Verwendung der Funktion `I2CRead` automatisch gesetzt). Die Slave-Adresse wird durch das **ACK**-Bit vom entsprechenden Slave bestätigt. Im Anschluß sendet der Slave byteweise Daten, die vom Master (AnaGate) jeweils durch ein **ACK**-Signal bestätigt werden. Nachdem die angeforderte Anzahl von Datenbytes empfangen wurden, wird das letzte Datenbyte mit einem **NAK**-Signal quittiert, um das Ende der Übertragung anzuzeigen. Die Übertragung wird anschliesend vom AnaGate durch das **Stop**-Signal beendet.

Siehe auch

I2CWrite

© 2007-2015 Analytica GmbH

I2CWrite

I2CWrite — Schreibt Daten zu einem I2C-Partner.

Syntax

#include <AnaGateDllI2C.h>

```
AnaInt32 I2CWrite(AnaInt32 hHandle, AnaUInt16 nSlaveAddress, const char
* pcBuffer, AnaInt32 nBufferLen, AnaInt32 * pnErrorByte);
```

Parameter

hHandle	Gültiges Zugriffs-Handle eines erfolgreichen Aufrufs von `I2COpenDevice`.
nSlaveAddress	Slave-Adresse des I2C-Partners. Die Slave-Adresse kann eine sog. 7 oder 10-Bit Adresse darstellen (siehe Anhang B, *Adressierung auf dem I2C-Bus*).
pcBuffer	Byte-Puffer mit den Daten, die an den I2C-Partner gesendet werden sollen.
nBufferLen	Anzahl von Datenbytes, die gelesen werden sollen.
pnErrorByte	Zeiger auf eine Variable, in der die Byte-Position eines aufgetretenen Fehlers innerhalb des Datenpuffers gespeichert wird.

Rückgabewert

Die Funktion gibt im Erfolgsfall `Null` zurück, andernfalls einen Fehlercode (Anhang A, *Rückgabewerte aus den API-Funktionen*).

Beschreibung

Schreibt Daten zu einem I2C-Slave, die AnaGate-Hardware fungiert dabei als I2C-Master am Bus.

Der Anwender muss einen korrekten Aufbau des Datenpuffers und der Adresse des I2C-Slave sicherstellen.

Die AnaGate-Hardware started die Übertragung mit dem **Start**-Signal, gefolgt von der angegebenen Slave-Adresse (das R/W-Bit der Slave-Adresse muss vom Anwender nicht explizit auf `0` werden, es wird bei Verwendung der Funktion `I2CWrite` automatisch auf `0` gesetzt). Die Slave-Adresse wird durch das **ACK**-Bit vom addressierten Slave bestätigt. Das AnaGate sendet im Anschluß die Daten byteweise an den Slave, der Slave bestätigt jeweils die einzelnen Datenbytes durch ein **ACK**-Signal. Sind alle Datenbytes abgearbeitet, wird die Übertragung durch das AnaGate durch das **Stop**-Signal beendet.

Siehe auch

I2CRead

 © 2007-2015 Analytica GmbH

I2CSequence

I2CSequence — Schreibt eine beliebige Folge von I2C-Schreib- und Lese-Kommandos als Sequenz zu einem I2C-Partner.

Syntax

#include <AnaGateDllI2C.h>

```
AnaInt32 I2CSequence(AnaInt32 hHandle, const char * pcWriteBuffer, AnaInt32 nNumberOfBytesToWrite, char * pcReadBuffer, AnaInt32 nNumberOfBytesToRead, AnaInt32 * pnNumberOfBytesRead, AnaInt32 * pnByteNumberLastError);
```

Parameter

hHandle — Gültiges Zugriffs-Handle eines erfolgreichen Aufrufs von `I2COpenDevice`.

pcWriteBuffer — Zeichenfolgepuffer, der die Kommandos enthält, die zum AnaGate I2C gesendet werden. Die einzelnen Kommandos werden hintereinander in diesen Puffer abgelegt.

Der Aufbau eines Lesekommandos ist dabei wie folgt definiert:

Aufbau eines Lesekommandos für I2CSequence

Lesekommando	Beschreibung
2 Bytes (LSB,MSB)	Slave Adresse im 7- bzw. 10-bit Format, wobei das R/W Bit explizit auf 1 gesetzt werden muss.
2 Bytes (LSB,MSB)	Bit 0-14: Anzahl der Datenbytes, die vom I2C-Partner gelesen werden sollen. Die erfolgreich gelesenen Daten werden im Zeichenfolgepuffer *pcReadBuffer* zurückgeliefert. Bit 15: Ist dieses Bit gesetzt, wird am Ende des Lesekommandos kein Stop-Bit an den Slave gesendet.

Der Aufbau eines Schreibkommandos ist dabei wie folgt definiert:

Aufbau eines Schreibkommandos für I2CSequence

Schreibkommando	Beschreibung
2 Bytes (LSB,MSB)	Slave Adresse im 7- bzw. 10-bit Format, wobei das R/W Bit explizit auf 0 gesetzt werden muss.

 © 2007-2015 Analytica GmbH

Schreibkommando	Beschreibung
2 Bytes (LSB,MSB)	Bit 0-14: Anzahl der folgenden Datenbytes, die zum I2C-Partner geschrieben werden sollen (N). Bit 15: Ist dieses Bit gesetzt, wird am Ende des Schreibkommandos kein Stop-Bit an den Slave gesendet.
N Bytes	Datenbytes.

nNumberOfBytesToWrite — Länge des pcWriteBuffers

pcReadBuffer — Zeichenfolgepuffer, der die empfangenen Daten, die vom I2C-Partner gesendet werden, aufnehmen soll. Hierbei werden die vom I2C-Partner empfangenen Daten hintereinander abgelegt (zuerst die Daten des ersten Lesekommandos, direkt danach die Daten des zweiten Lesekommandos, etc.)

nNumberOfBytesRead — Länge des Zeichenfolgepuffers, der die empfangenen Daten aufnehmen soll.

pnNumberOfBytesRead — Anzahl tatsächlich gelesener Bytes, die vom I2C-Partner empfangen wurden.

pnByteNumberLastError — Der Byte-Offset im Zeichenfolgepuffer *`pcWriteBuffer`*, der einen Fehler erzeugt hat.

Rückgabewert

Die Funktion gibt im Erfolgsfall `Null` zurück, andernfalls einen Fehlercode (Anhang A, *Rückgabewerte aus den API-Funktionen*).

Beschreibung

Der Anwender muss einen korrekten Aufbau des Datenpuffers und der Adresse des I2C Partners sicherstellen.

© 2007-2015 Analytica GmbH

I2CReadDigital

I2CReadDigital — Liest die aktuellen Werte der digitalen Ein-/Ausgabe-Register der AnaGate Hardware zurück.

Syntax

#include <AnaGateDllI2C.h>

`AnaInt32 **I2CReadDigital**(AnaInt32 *hHandle*, AnaUInt32 * *pnInputBits*, AnaUInt32 * *pnOutputBits*);`

Parameter

hHandle	Gültiges Zugriffs-Handle eines erfolgreichen Aufrufs von `I2COpenDevice`.
pnInputBits	Zeiger auf den aktuellen Inhalt des Registers mit den digitalen Eingängen. Dabei entsprechen Bit 0 bis Bit 3 den vier digitalen Eingängen; Bit 4 bis Bit 31 sind auf 0 gesetzt.
pnOutputBits	Zeiger auf den aktuellen Inhalt des Registers mit den digitalen Ausgängen. Dabei entsprechen Bit 0 bis Bit 3 den vier digitalen Ausgängen; Bit 4 bis Bit 31 sind auf 0 gesetzt.

Rückgabewert

Die Funktion gibt im Erfolgsfall `Null` zurück, andernfalls einen Fehlercode (Anhang A, *Rückgabewerte aus den API-Funktionen*).

Beschreibung

Alle Geräte der AnaGate-Serie (außer der Gerätevariante AnaGate CAN uno im Hutschienengehäuse) besitzen auf der Gehäuserückseite Anschlüsse für jeweils 4 digitale Eingänge und 4 digitale Ausgänge.

Die aktuellen Zustände der digitalen Eingänge und Ausgänge können mit der Funktion `I2CReadDigital` ermittelt werden.

Im folgenden ein Programmier-Beispiel, das die digitalen IOs setzt und zurückliest.

```
#include <AnaGateDllI2C.h>
int main()
{
   AnaInt32 hHandle = 0;
   AnaUInt32 nInputs;
   AnaUInt32 nOutputs = 0x03;

   AnaInt32 nRC = I2COpenDevice(&hHandle, 400000, "192.168.1.254", 5000);
   if ( nRC == 0 )
   {
      // set the digital output register (PIN 0 and PIN 1 to HIGH value)
      nRC = I2CWriteDigital( hHandle, nOutputs );
```

 © 2007-2015 Analytica GmbH

```
      // read all input and output registers
      nRC = I2CReadDigital( hHandle, &nInputs, &nOutputs );

      CANCloseDevice(hHandle);
   }
   return 0;
}
```

Siehe auch

I2CWriteDigital

© 2007-2015 Analytica GmbH

I2CWriteDigital

I2CWriteDigital — Setzt das digitale Ausgabe-Register der AnaGate-Hardware auf einen neuen Wert.

Syntax

```
AnaInt32 I2CWriteDigital(AnaInt32 hHandle, AnaUInt32 nOutputBits);
```

Parameter

hHandle	Gültiges Zugriffs-Handle eines erfolgreichen Aufrufs von `I2COpenDevice`.
nOutputBits	Neuer Wert des Registers mit den digitalen Ausgängen. Dabei entsprechen Bit 0 bis Bit 3 den vier digitalen Ausgängen; Bit 4 bis Bit 31 sind reserviert und auf 0 zu setzen.

Rückgabewert

Die Funktion gibt im Erfolgsfall `Null` zurück, andernfalls einen Fehlercode (Anhang A, *Rückgabewerte aus den API-Funktionen*).

Beschreibung

Alle Geräte der AnaGate-Serie (außer der Gerätevariante AnaGate CAN uno im Hutschienengehäuse) besitzen auf der Gehäuserückseite Anschlüsse für jeweils 4 digitale Eingänge und 4 digitale Ausgänge.

Die digitalen Ausgänge können mit der Funktion `I2CWriteDigital` verändert werden.

Ein Programmier-Beispiel zum Lesen/Schreiben der IOs ist bei der Beschreibung von `I2CReadDigital` zu finden.

Siehe auch

I2CReadDigital

© 2007-2015 Analytica GmbH

I2CErrorMessage

I2CErrorMessage — Gibt eine textuelle Beschreibung eines Rückgabewertes aus einer API-Funktion zurück.

Syntax

#include <AnaGateDllI2C.h>

AnaInt32 **I2CErrorMessage**(AnaInt32 *nRetCode*, char * *pcMessage*, AnaInt32 *nMessageLen*);

Parameter

nRetCode Error-Code, dessen Fehlerbeschreibung ermittelt werden soll.

pcMessage Datenpuffer, der die Fehlerbeschreibung aufnehmen soll.

nMessageLen Größe des übergebenen Datenpuffers in Byte.

Rückgabewert

Tatsächliche Größe der zurückgegebenen Beschreibung in Byte.

Beschreibung

Gibt eine textuelle Beschreibung des übergebenen Rückgabewertes zurück (siehe auch Anhang A, *Rückgabewerte aus den API-Funktionen*). Ist der Ziel-Datenpuffer nicht groß genug, um den Fehlertext aufzunehmen, wird der Text auf die angegebene Puffergröße gekürzt. Alle Fehlertexte sind in englischer Sprache.

Im folgenden ein Programmier-Beispiel in C/C++.

```
#include <AnaGateDllI2C.h>
// ...
AnaInt32 nRC;
char cBuffer[200];
//... call a API function here
I2CErrorMessage(nRC, cBuffer, sizeof cBuffer);
std::cout << "Fehler: " << cBuffer << std::endl;
```


© 2007-2015 Analytica GmbH

I2CReadEEProm

I2CReadEEProm — Liest Daten von einem EEPROM am I2C-Bus.

Syntax

#include <AnaGateDllI2C.h>

```
AnaInt32 I2CReadEEProm(AnaInt32 hHandle, AnaUInt16 nSubAddress, AnaUInt32 nOffset, char * pcBuffer, AnaInt32 nBufferLen, AnaUInt32 nOffsetFormat);
```

Parameter

hHandle — Gültiges Zugriffs-Handle eines erfolgreichen Aufrufs von `I2COpenDevice`.

nSubAddress — Subadresse des EEPROMs, mit dem kommuniziert werden soll. Die gültigen Werte für die *nSubAddress* werden von der Einstellung des Parameters *nOffsetFormat* (Bit 8-10) beeinflusst. Werden von dem EEPROM-Typ Bits der *Chip Enable Address* zur Adressierung des internen Speichers verwendet, so können nur noch die frei verfügbaren Bits als Steuerpins für die Ansteuerung des Bausteins auf der Platine genutzt werden.

- kein Bit für die Adressierung verwendet: 0 bis 7
- 1 Bit für die Adressierung verwendet: 0 bis 3
- 2 Bits für die Adressierung verwendet: 0 bis 1
- 3 Bits für die Adressierung verwendet: 0

nOffset — Daten-Offset auf dem EEPROM, ab dem Daten gelesen werden sollen.

pcBuffer — Zeichenfolgepuffer, der die vom EEPROM gelesenen Daten aufnehmen soll.

nBufferLen — Länge des Datenpuffers.

nOffsetFormat — Dieser Parameter ist als Bitfeld definiert und gibt an, wie eine Speicheradresse auf dem EEPROM bei Schreib- und Lesezugriffen abgebildet wird.

Die Bits 0-7 geben an, wie viele Bits im Adressbyte (bzw. Addresswort) für die Adressierung verwendet werden.

Die Bits 8-10 geben an, wie viele und welche der *Chip Enable Bits* zusätzlich zur Adressierung des EEPROM-Speichers verwendet werden (Tabelle C.1, „ Verwendung der Chip-Enable-Bits bei I2C-EEPROMs “).

 © 2007-2015 Analytica GmbH

Anmerkung

Die maximal adressierbare Speichergröße eines EEPROMs kann aus der Summe aller Adressbits berechnet werden. So benötigt z.B. ein M24C08 acht Bits des Adressbytes und 1 zusätzliches Bit. Damit können über die 9 Bits insgesamt 512 Bytes adressiert werden.

Rückgabewert

Die Funktion gibt im Erfolgsfall `Null` zurück, andernfalls einen Fehlercode (Anhang A, *Rückgabewerte aus den API-Funktionen*).

Beschreibung

Die Funktion `I2CReadEEProm` liest Daten von einem seriellen I2C-EEPROM.

Der Zugriff auf eine Speicheradresse auf einem I2C-EEPROM wird prinzipiell über eine normale Schreib bzw. Leseanforderung auf dem I2C-Bus realisiert. Somit müssen bei einem Lesezugriff nur die passende Slave-Adresse, die Offset-Adresse auf dem Chip und die eigentlichen Daten gesendet werden.

Die Funktion`I2CReadEEProm` setzt die übergebene Speicheradresse auf dem Chip anhand der Angabe von der Subadresse und der Adressierungsvariante des vorliegenden EEPROM-Typs korrekt um. Eine Angabe der Slave-Adresse entfällt, da diese bereits durch die vorhandenen Parameter festgelegt ist.

Ein Programmier-Beispiel, das ein EEPROM vom Typ **ST24C1024** komplett löscht, ist bei der Beschreibung von I2WriteEEPROM zu finden.

Siehe auch

I2CWriteEEProm

Anhang C, *Programmierung von I2C-EEPROM*

© 2007-2015 Analytica GmbH

I2CWriteEEProm

I2CWriteEEProm — Beschreibt ein serielles EEPROM am I2C-Bus.

Syntax

#include <AnaGateDllI2C.h>

```
AnaInt32 I2CWriteEEProm(AnaInt32 hHandle, AnaUInt16 nSubAddress,
AnaUInt32 nOffset, const char * pcBuffer, AnaInt32 nBufferLen, AnaUInt32
nOffsetFormat);
```

Parameter

hHandle — Gültiges Zugriffs-Handle eines erfolgreichen Aufrufs von `I2COpenDevice`.

nSubAddress — Subadresse des EEPROMs, mit dem kommuniziert werden soll. Die gültigen Werte für die *nSubAddress* werden von der Einstellung des Parameters *nOffsetFormat* (Bit 8-10) beeinflusst. Werden von dem EEPROM-Typ Bits der *Chip Enable Address* zur Adressierung des internen Speichers verwendet, so können nur noch die frei verfügbaren Bits als Steuerpins für die Ansteuerung des Bausteins auf der Platine genutzt werden.

- kein Bit für die Adressierung verwendet: 0 bis 7
- 1 Bit für die Adressierung verwendet: 0 bis 3
- 2 Bits für die Adressierung verwendet: 0 bis 1
- 3 Bits für die Adressierung verwendet: 0

nOffset — Daten-Offset auf dem EEPROM, ab dem die übergebenen Daten geschrieben werden sollen.

pcBuffer — Zeichenfolgepuffer mit den Daten, die geschrieben werden sollen.

nBufferLen — Länge des Datenpuffers.

nOffsetFormat — Dieser Parameter ist als Bitfeld definiert und gibt an, wie eine Speicheradresse auf dem EEPROM bei Schreib- und Lesezugriffen abgebildet wird.

Die Bits 0-7 geben an, wie viele Bits im Adressbyte (bzw. Addresswort) für die Adressierung verwendet werden.

Die Bits 8-10 geben an, wie viele und welche der *Chip Enable Bits* zusätzlich zur Adressierung des EEPROM-Speichers verwendet werden (Tabelle C.1, „ Verwendung der Chip-Enable-Bits bei I2C-EEPROMs “).

Anmerkung

Die maximal adressierbare Speichergröße eines EEPROMs kann aus der Summe aller Adressbits

© 2007-2015 Analytica GmbH

berechnet werden. So benötigt z.B. ein M24C08 acht Bits des Adressbytes und 1 zusätzliches Bit. Damit können über die 9 Bits insgesamt 512 Bytes adressiert werden.

Rückgabewert

Die Funktion gibt im Erfolgsfall `Null` zurück, andernfalls einen Fehlercode (Anhang A, *Rückgabewerte aus den API-Funktionen*).

Beschreibung

Die Funktion `I2CWriteEEProm` schreibt Daten auf ein serielles I2C-EEPROM.

Der Zugriff auf eine Speicheradresse auf einem I2C-EEPROM wird prinzipiell über eine normale Schreib- bzw. Leseanforderung auf dem I2C-Bus realisiert. Somit müssen bei einem Schreibzugriff nur die passende Slave-Adresse, die Offset-Adresse auf dem Chip und die eigentlichen Daten gesendet werden.

Die Funktion `I2CWriteEEProm` setzt die übergebene Speicheradresse auf dem Chip anhand der Angabe von der Subadresse und der Adressierungsvariante des vorliegenden EEPROM-Typs korrekt um. Eine Angabe der Slave-Adresse entfällt, da diese bereits durch die vorhandenen Parameter festgelegt ist.

Tipp

Beim Programmieren von EEPROMs ist zu beachten, dass der EEPROM-Speicher in sogenannte Pages unterteilt ist, und dass mit einem einzelnen Schreibbefehl nur innerhalb einer Speicherseite programmiert werden kann. Benutzer von `I2CWriteEEProm` müssen selbst sicherstellen, dass sie nicht über Seitengrenzen hinaus schreiben. Die jeweilige Größe einer Speicherseite ist abhängig vom EEPROM-Typ.

Im folgenden ein Programmier-Beispiel, das ein EEPROM vom Typ **ST24C1024** komplett löscht.

```
#include <AnaGateDllSPI.h>
int main()
{
   char      cBufferPage[256];
   int       hHandle = 0;
   int       nRC = 0;
   AnaUInt16 nSubAddress = 0;1
   AnaUInt32 nOffsetFormat = 0x10|0x0F;2

   int nRC = I2COpenDevice(&hHandle, 400000, "192.168.1.254", 5000);
   if ( nRC == 0 )
   {
      memset(cBufferPage,0,256);  // clear page buffer
      for (int i=0; i<512;i++)
      {
         I2CWriteEEProm( hHandle, nSubAddress, i*256, cBufferPage, 256, nOffsetFormat );3
      }
      I2CCloseDevice(hHandle);
   }
   return 0;
}
```

1 Es können bis zu 4 ST24C1024 am I2C-Bus verdrahtet werden. Über die Angabe der Subadresse `0` liegen die Steuerpins E2 und E1 auf LOW.

 © 2007-2015 Analytica GmbH

2. Der ST24C1024 benötigt 17 Adressbits zur Adressierung der 128kB. 16 Bits werden über die Adressangabe des Schreibbefehls festgelegt: `16=0x0F`. Das Adressbit A16 wird über das E0 der *Chip Enable Address* festgelegt, deshalb ist der Mode `1` (E2-E1-A0) zu verwenden: `0x10`.
3. Die Seitengröße eines ST24C1024 beträgt 256 Bytes, im vorliegenden Fall werden alle Seiten komplett über eine for-Schleife programmiert.

Siehe auch

I2CReadEEProm, I2CWriteEEPromPollAck

Anhang C, *Programmierung von I2C-EEPROM*

 © 2007-2015 Analytica GmbH

I2CWriteEEPromPollAck

I2CWriteEEPromPollAck — Beschreibt ein serielles EEPROM am I2C-Bus und prüft, ob der Schreibvorgang innerhalb einer gegebenen Zeitspanne abgeschlossen wird.

Syntax

```
#include <AnaGateDllI2C.h>

AnaInt32 I2CWriteEEPromPollAck(AnaInt32 hHandle, AnaUInt16 nSubAddress,
AnaUInt32 nOffset, const char * pcBuffer, AnaInt32 nBufferLen, AnaUInt32
nOffsetFormat, AnaUInt16 nTimeout);
```

Parameter

hHandle — Gültiges Zugriffs-Handle eines erfolgreichen Aufrufs von `I2COpenDevice`.

nSubAddress — Subadresse des EEPROMs, mit dem kommuniziert werden soll. Die gültigen Werte für die *nSubAddress* werden von der Einstellung des Parameters *nOffsetFormat* (Bit 8-10) beeinflusst. Werden von dem EEPROM-Typ Bits der *Chip Enable Address* zur Adressierung des internen Speichers verwendet, so können nur noch die frei verfügbaren Bits als Steuerpins für die Ansteuerung des Bausteins auf der Platine genutzt werden.

- kein Bit für die Adressierung verwendet: 0 bis 7
- 1 Bit für die Adressierung verwendet: 0 bis 3
- 2 Bits für die Adressierung verwendet: 0 bis 1
- 3 Bits für die Adressierung verwendet: 0

nOffset — Daten-Offset auf dem EEPROM, ab dem die übergebenen Daten geschrieben werden sollen.

pcBuffer — Zeichenfolgepuffer mit den Daten, die geschrieben werden sollen.

nBufferLen — Länge des Datenpuffers.

nOffsetFormat — Dieser Parameter ist als Bitfeld definiert und gibt an, wie eine Speicheradresse auf dem EEPROM bei Schreib- und Lesezugriffen abgebildet wird.

Die Bits 0-7 geben an, wie viele Bits im Adressbyte (bzw. Addresswort) für die Adressierung verwendet werden.

Die Bits 8-10 geben an, wie viele und welche der *Chip Enable Bits* zusätzlich zur Adressierung des EEPROM-Speichers verwendet werden (Tabelle C.1, „ Verwendung der Chip-Enable-Bits bei I2C-EEPROMs “).

 © 2007-2015 Analytica GmbH

Anmerkung

Die maximal adressierbare Speichergröße eines EEPROMs kann aus der Summe aller Adressbits berechnet werden. So benötigt z.B. ein M24C08 acht Bits des Adressbytes und 1 zusätzliches Bit. Damit können über die 9 Bits insgesamt 512 Bytes adressiert werden.

nTimeout Timeout in Millisekunden für den Schreibvorgang.

Rückgabewert

Die Funktion gibt im Erfolgsfall `Null` zurück, andernfalls einen Fehlercode (Anhang A, *Rückgabewerte aus den API-Funktionen*).

Beschreibung

Die Funktion `I2CWriteEEPromPollAck` schreibt Daten auf ein serielles I2C-EEPROM und prüft, ob der Schreibvorgang innerhalb einer gegebenen Zeitspanne abgeschlossen wird.

Der Zugriff auf eine Speicheradresse auf einem I2C-EEPROM wird prinzipiell über eine normale Schreib- bzw. Leseanforderung auf dem I2C-Bus realisiert. Somit müssen bei einem Schreibzugriff nur die passende Slave-Adresse, die Offset-Adresse auf dem Chip und die eigentlichen Daten gesendet werden.

Die Funktion `I2CWriteEEPromPollAck` setzt die übergebene Speicheradresse auf dem Chip anhand der Angabe von der Subadresse und der Adressierungsvariante des vorliegenden EEPROM-Typs korrekt um. Eine Angabe der Slave-Adresse entfällt, da diese bereits durch die vorhandenen Parameter festgelegt ist.

Tipp

Beim Programmieren von EEPROMs ist zu beachten, dass der EEPROM-Speicher in sogenannte Pages unterteilt ist, und dass mit einem einzelnen Schreibbefehl nur innerhalb einer Speicherseite programmiert werden kann. Benutzer von `I2CWriteEEPromPollAck` müssen selbst sicherstellen, dass sie nicht über Seitengrenzen hinaus schreiben. Die jeweilige Größe einer Speicherseite ist abhängig vom EEPROM-Typ.

Im folgenden ein Programmier-Beispiel, das ein EEPROM vom Typ **ST24C1024** komplett löscht.

```
#include <AnaGateDllSPI.h>
int main()
{
   char      cBufferPage[256];
   AnaInt32  hHandle = 0;
   AnaUInt16 nSubAddress = 0;1
   AnaUInt32 nOffsetFormat = 0x10|0x0F;2
   AnaUInt16 nTimeout = 100;

   int nRC = I2COpenDevice(&hHandle, 400000, "192.168.1.254", 5000);
```

 © 2007-2015 Analytica GmbH

```
   if ( nRC == 0 )
   {
      memset(cBufferPage,0,256);  // clear page buffer
      for (int i=0; i<512;i++)
      {
         I2CWriteEEPromPollAck( hHandle, nSubAddress, i*256,
                                cBufferPage, 256, nOffsetFormat, nTimeout );3
      }
      I2CCloseDevice(hHandle);
   }
   return 0;
}
```

1. Es können bis zu 4 ST24C1024 am I2C-Bus verdrahtet werden. Über die Angabe der Subadresse `0` liegen die Steuerpins E2 und E1 auf LOW.
2. Der ST24C1024 benötigt 17 Adressbits zur Adressierung der 128kB. 16 Bits werden über die Adressangabe des Schreibbefehls festgelegt: `16=0x0F`. Das Adressbit A16 wird über das E0 der *Chip Enable Address* festgelegt, deshalb ist der Mode `1` (E2-E1-A0) zu verwenden: `0x10`.
3. Die Seitengröße eines ST24C1024 beträgt 256 Bytes, im vorliegenden Fall werden alle Seiten komplett über eine for-Schleife programmiert.

Siehe auch

I2CWriteEEProm

Anhang C, *Programmierung von I2C-EEPROM*

© 2007-2015 Analytica GmbH

Kapitel 7. Programmier-Beispiele

7.1. Programmiersprache C/C++

Die AnaGate Programmier-API kann sowohl unter Windows-Systemen als auch unter X86-Linux-Systemen verwendet werden. Alle über die API zur Verfügung gestellten Funktionen sind Betriebssystem-unabhängig implementiert, so dass einmal erstellter Source-Code auf beiden Betriebssystemen eingesetzt werden kann. Lediglich die Einbindung der Bibliotheken muss auf unterschiedlich Betriebssystemen bzw. der Einsatz von unterschiedlichen Compilern entsprechend angepasst werden.

Windows-Betriebssysteme

Für den Zugriff auf die Funktionen der API stehen für C/C++-Programmierer prinzipiell zwei Möglichkeiten zur Verfügung:

- Beim direkten Zugriff auf die Bibliothek müssen die Funktionen durch vorangehenden Aufrufe der Win32-Methoden `LoadLibrary` und `GetProcAddress` bekanntgemacht werden.

- Der Zugriff kann auch alternativ über eine sog. Import-Bibliothek erfolgen. Hierbei wird das Laden der DLL und der DLL-Methoden über die Import-Bibliothek automatisch durchgeführt.

  Die Import-Bibliotheken für MS VC8 sind im Lieferumfang enthalten und können direkt verwendet werden, so lautet die Importbibliothek von `AnaGateCAN.dll` z.B. `AnaGateCANDll.lib`.

In den folgenden Beispielen wird davon ausgegangen, dass die zweite Möglichkeit verwendet und die jeweilige Import-Bibliothek zum Beispiel-Programm gebunden wird.

7.1.1. CAN Console-Anwendung C/C++ (MSVC)

Das folgende Programmierbeispiel für C++ zeigt, wie eine Verbindung mit dem AnaGate CAN aufgebaut wird und wie über eine Callback-Funktion empfangene CAN-Telegramme verarbeitet werden können.

Anmerkung

Der vollständige Quellcode des vorliegenden Beispiels ist auf der jedem Gerät beiliegenden CD-ROM im Verzeichnispfad `Samples/CAN-VC6` bzw. `Samples/CAN-VC7` zu finden.

```
#include <AnaGateDllCan.h>

WINAPI void MyCallback(AnaInt32 nIdentifier, const char * pcBuffer,
                       AnaInt32 nBufLen, AnaInt32 nFlags, AnaInt32 nHandle)
{
   std::cout << "CAN-ID=" << nIdentifier << ", Data=";
   for ( AnaInt32 i = 0; i < nBufLen; i++ )
```


© 2007-2015 Analytica GmbH

```
   {
      std::cout " 0x" << std::hex << pcBuffer[i];
   }
   std::cout << std::endl;
}

int main( )
{
   AnaInt32 hHandle = NULL;

   // opens AnaGate CAN duo device on port A, timeout after 1000 milliseconds
   AnaInt32 nRC = CANOpenDevice(&hHandle, FALSE, TRUE, 0, "192.168.1.254", 1000);

   if ( nRC == 0 )
   {
      nRC = CANSetCallback(hHandle, MyCallback);
      getch();  // wait for keyborad input
   }
   if ( nRC == 0 )
   {
      nRC = CANCloseDevice(hHandle); // close device
   }
}
```

7.2. Programmiersprache Visual Basic 6

Wie bereits in den einleitenden Kapiteln beschrieben, verwenden die Bibliotheken der *AnaGate*-API die sog. cdecl-Aufrufkonvention bei der Übergabe der Funktionsparameter auf den Programm-Stack. Leider unterstützt die Programmiersprache *Visual Basic 6* diese Aufrufkonvention generell nicht.

Um diese Einschränkung von VB6 zu umgehen, werden die Bibliotheken mit den Zugriffsfunkionen für die AnaGate-Hardware auch in einer speziellen Version für die Programmierung von VB6-Anwendungen zur Verfügung gestellt. In diesen Version wird die für VB6 notwendige stdcall-Aufrufkonvention verwendet. Abgesehen von der Art und Weise, wie die Parameter auf den Programm-Stack gebracht werden, sind die VB6-Versionen der einzelnen Bibliotheken exakt identisch mit den Standardversionen.

Anmerkung

Anstatt der Bibliothek *`AnaGateCAN.dll`* muss die *`AnaGateCANVB6.dll`* verwendet werden.

Anstatt der Bibliothek *`AnaGateSPI.dll`* muss die *`AnaGateSPIVB6.dll`* verwendet werden.

7.2.1. SPI Beispiel mit Benutzeroberfläche für VB6

Dieses einfache Programmierbeispiel für Visual Basic 6 zeigt, wie eine Verbindung mit dem AnaGate SPI aufgebaut wird und wie ein Befehl auf dem SPI-Bus abgesetzt wird. Im einzelnen werden folgende Benutzeraktionen zur Verfügung gestellt:

- Auslesen der globalen Geräteeinstellungen (z.B. Baudrate)
- Ausführen eines einzelnen Kommandos auf dem SPI-Bus

 © 2007-2015 Analytica GmbH

Anmerkung

Der vollständige Quellcode des vorliegenden Beispiels ist auf der jedem Gerät beiliegenden CD-ROM im Verzeichnispfad `Samples/SPI-VB6` zu finden.

7.2.1.1. Benutzeroberfläche

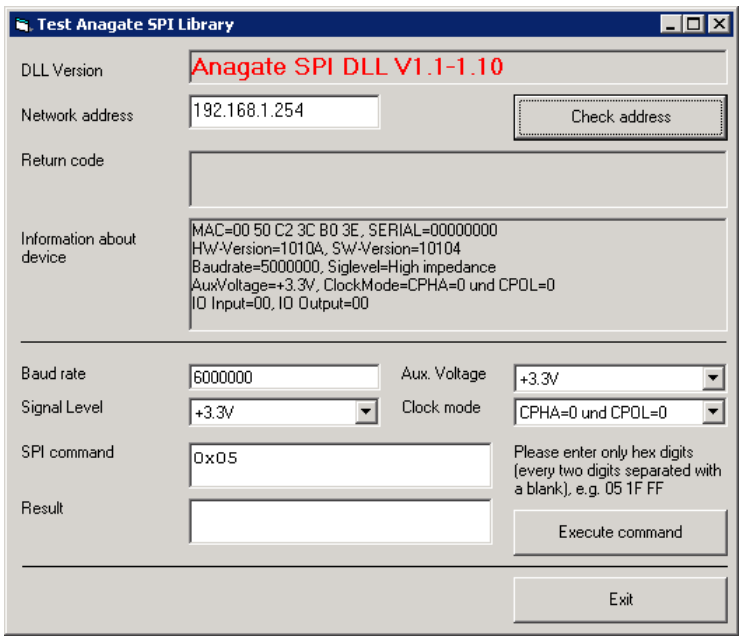


Abbildung 7.1. Eingabe-Formular SPI-Beispiel (VB6)

Dialogfelder

Network address	Netzwerk-Adresse, unter der das Anagate SPI zu erreichen ist.
Check address	Stellt eine Verbindung zu dem AnaGate SPI unter der angegebenen Netzwerkadresse her und liest anschließend verschiedene Geräteinformationen bzw. Geräteeinstellungen zurück.
Baud rate	Baudrate, mit der gearbeitet werden soll. Werte können individuell eingestellt werden.
Signal Level	Gibt den Pegelwert für SPI Signale an.
Aux. Voltage	Gibt die Ausgangsspannung für die Hilfsspannungsversorgung an.
Clock mode	Gibt die Phase und die Polarität der Clock-Leitung für die Datenübertragung an.
SPI command	SPI-Befehlsfolge, die an den angeschlossenen SPI-Baustein gesendet werden soll. Die Eingabe muss hexadezimal erfolgen, jedes einzelne Byte durch ein Leerzeichen getrennt (z.B. "`05 1F 3A`").

© 2007-2015 Analytica GmbH

Execute command

Führt ein SPI-Kommando aus und schreibt das Ergebnis in das Dialogfeld *Result*. Es ist zu beachten, dass der SPI Bus als solcher voll-duplex betrieben wird, d.h. dass beim Schreiben gleichzeitig gelesen wird. Aus diesem Grund muss die Anzahl der geschriebenen Bytes mindestens so groß sein, wie die Anzahl der Bytes, die gelesen werden sollen. Gegebenenfalls ist der Schreibpuffer mit entsprechend mit Dummy-Daten aufzufüllen.

Beispiel: Auf einem **M25P80** ist das Kommando **Read Status Register** durch die Bytefolge `0x05` festgelegt. Als Ergebnis wird das Status-Register als einzelnes Byte (8 Bit) zurückgeliefert.

7.2.1.2. Ermittlung der globalen Geräte-Einstellungen

Alle SPI-Funktionen der AnaGate-API sind in der Datei `AnaGateSPI.bas` bereits deklariert und können sofort verwendet werden. Der nachfolgende Dateiausschnitt beinhaltet die Deklarationen von den verwendeten Funktionen aus der API.

```
Public Declare Function SPIOpenDevice Lib "AnaGateSPIVB6" _
                Alias "_SPIOpenDevice@12" (ByRef Handle As Long, _
                                           ByVal TCPAddress As String, _
                                           ByVal Timeout As Long) As Long

Public Declare Function SPICloseDevice Lib "AnaGateSPIVB6" _
                Alias "_SPICloseDevice@4" (ByVal Handle As Long) As Long

Public Declare Function SPIGetGlobals Lib "AnaGateSPIVB6" _
               Alias "_SPIGetGlobals@20" (ByVal hHandle As Long, _
                                          ByRef nBaudrate As Long, _
                                          ByRef SigLevel As Byte, _
                                          ByRef nAuxVoltage As Byte, _
                                          ByRef nClockMode As Byte) As Long
```

Die Eventprozedur `btnCheckAddress_Click` wird beim Klicken des Buttons mit der Aufschrift ***Check Address*** aufgerufen.

```
Private Sub btnCheckAddress_Click()
    Dim nRC As Long, Data As String, sText As String, I As Long

    nRC = SPIOpenDevice(hHandle, Me.IPAddresse.Text, 2000) 1
    If nRC <> 0 Then
        Me.lblErrorMsg.Caption = "Fehler bei SPIOpenDevice: " & GetErrorMsg(nRC)
    Else
        Me.lblErrorMsg.Caption = GetAnagateInfo(hHandle)
    End If
    nRC = SPICloseDevice(hHandle) 2
End Sub
```

1. Über einen Aufruf von `SPIOpenDevice` wird eine Netzwerkverbindung zum Gerät aufgebaut. Schlägt der Verbindungsaufbau fehl, so wird über die Funktion `GetErrorMsg` eine Fehlerbeschreibung anhand des Return-Codes ermittelt.
2. Die Verbindung zum Gerät wird mit `SPICloseDevice` geschlossen.

Das Auslesen der Geräteinformationen und die textuelle Aufbereitung dieser Daten erfolgt über die Funktion `GetAnagateInfo`.

```
Private Function GetAnagateInfo(hHandle As Long) As String
    Dim nRC As Long, sText As String
```

 © 2007-2015 Analytica GmbH

```
    Dim nBaudrate As Long, nSigLevel As Byte, nAuxVoltage As Byte, nClockMode As Byte
    Dim nDigitalOutput As Long, nDigitalInput As Long

    nRC = SPIGetGlobals(hHandle, nBaudrate, nSigLevel, nAuxVoltage, nClockMode)
    If (nRC = 0) Then
        sText = sText & "Baudrate=" & CStr(nBaudrate) & ", Siglevel="
        Select Case nSigLevel
            Case 1: sText = sText & "+5.0V"
            Case 2: sText = sText & "+3.3V"
            Case 3: sText = sText & "+2.5V"
            Case Else: sText = sText & "High impedance"
        End Select
        sText = sText & vbCrLf & "AuxVoltage="
        Select Case nAuxVoltage
            Case 1: sText = sText & "+2.5V"
            Case Else: sText = sText & "+3.3V"
        End Select
        sText = sText & ", ClockMode="
        Select Case nClockMode
            Case 1: sText = sText & "CPHA=0 und CPOL=1"
            Case 2: sText = sText & "CPHA=1 und CPOL=0"
            Case 3: sText = sText & "CPHA=1 und CPOL=1"
            Case Else: sText = sText & "CPHA=0 und CPOL=0"
        End Select
    Else
       sText = sText & "Fehler bei SPIGetGlobals: " & GetErrorMsg(nRC) & vbCrLf
    End If
    GetAnagateInfo = sText
End Function
```

7.2.1.3. Ausführen eines Kommandos auf dem SPI-Bus

Das AnaGate SPI ist in der Lage, beliebige Kommandofolgen auf den angeschlossenen SPI-Bus zu schreiben. Zum Schreiben und Lesen von Daten auf PC-Seite wird lediglich die Funktion `SPIDataReq` benötigt.

```
Public Declare Function SPIDataReq Lib "AnaGateSPIVB6"
                Alias "_SPIDataReq@20" (ByVal hHandle As Long, _
                                        ByVal lpBufferWrite As Any, _
                                        ByVal nBufferWriteLen As Long, _
                                        ByVal lpBufferRead As Any, _
                                        ByVal nBufferReadlLen As Long) As Long
```

Die Eventprozedur `btnStart_Click` wird beim Klicken des Buttons mit der Aufschrift ***Execute command*** aufgerufen.

```
Private Sub btnStart_Click()
    Dim nRC As Long, sText As String, I As Integer, sByteText As String
    Dim nBaudrate As Long, nSigLevel As Byte, nAuxVoltage As Byte, nClockMode As Byte
    Dim nBufferWriteLen As Long, nBufferReadLen As Long
    Dim arrWrite(1 To 255) As Byte, arrRead(1 To 255) As Byte

    nRC = SPIOpenDevice(hHandle, Me.IPAddresse.Text, 2000)
    If nRC <> 0 Then
        sText = "Fehler bei SPIOpenDevice: " & GetErrorMsg(nRC)
    Else
        nBaudrate = CLng(Me.txtBaudrate)
        nSigLevel = CLng(Me.cmbSigLevel.ListIndex)
        nAuxVoltage = CLng(Me.cmbAuxVoltage.ListIndex)
        nClockMode = CLng(Me.cmbClockMode.ListIndex)
        nRC = SPISetGlobals(hHandle, nBaudrate, nSigLevel, nAuxVoltage, nClockMode) 1

        If nRC <> 0 Then
```

 © 2007-2015 Analytica GmbH

```
            sText = sText & "Fehler bei SPISetGlobals: " & GetErrorMsg(nRC) & vbCrLf
        End If
        Me.lblDeviceInfo.Caption = GetAnagateInfo(hHandle)

        nBufferWriteLen = GetCommand(arrWrite)  2
        nBufferReadLen = nBufferWriteLen

        nRC = SPIDataReq(hHandle, VarPtr(arrWrite(1)), nBufferWriteLen, _
                                  VarPtr(arrRead(1)), nBufferReadLen)      3
        If nRC = 0 Then
            For I = 1 To nBufferReadLen
               sByteText = sByteText & "0x" & ToHex(arrRead(I)) & " "
            Next I
            Me.txtBufferRead = sByteText
            sText = sText & "SPIDataReq OK: " & vbCrLf
        Else
            sText = sText & "Fehler bei SPIDataReq: " & GetErrorMsg(nRC) & vbCrLf
        End If

        nRC = SPICloseDevice(hHandle)
    End If

End Sub
```

1. Über einen Aufruf von `SPISetGlobals` werden die globalen Einstellungen am Gerät vorgenommen. Die Parameterwerte werden den entsprechenden Eingabefeldern des Dialogs entnommen.
2. Die Funktion `GetCommand` wandelt das SPI-Kommando, das über die Oberfläche als Text eingegeben wurde, in ein Bytearray um.
3. Über die Funktion `SPIDataReq` wird ein Befehl auf dem SPI-Bus abgesetzt. Das Bytearray *`arrWrite`* enthält das SPI-Kommando und im Feld *`arrRead`* sind im Erfolgsfall die zurückgelieferten Daten abgelegt. Für beide Parameter muss die Feldgröße angegeben werden. Es sei hier bemerkt, dass die Funktion keine Plausibilität der Befehlsfolge durchführt. Der Anwender muss in jedem Fall auch genügend Speicherplatz für die zurückgelieferten Daten bereitstellen.

   Um die Daten im Schreib- bzw. Empfangspuffer bequem verarbeiten zu können, wurde jeweils ein Bytearray als VB6-Datentyp gewählt. Damit dies funktioniert, muss an die DLL-Funktion die tatsächliche Speicheradresse der Felddaten übergeben werden. Dies wird durch die Anwendung der Funktion `VarPtr` auf das erste Feldelement erreicht.

7.3. Programmiersprache VB.NET

Selbstverständlich können die Funktionen der AnaGate-API auch mit .NET-Programmiersprachen verwendet werden. In diesem Fall müssen die verwendeten Funktionen lediglich korrekt deklariert werden. Die Deklaration an sich kann prinzipiell in einer beliebigen .NET-Sprache erfolgen. Das Laden und Entladen der DLL erfolgt automatisch durch das .NET-Framework.

7.3.1. CAN Console-Anwendung VB.NET

Das folgende Programmierbeispiel für VB.NET zeigt, wie eine Verbindung mit dem AnaGate CAN aufgebaut wird und wie über eine Callback-Funktion empfangene CAN-Telegramme verarbeitet werden können.

```
Declare Function CANOpenDevice Lib "AnaGateCAN" (ByRef Handle As Int32, _
                                                 ByVal ConfirmData As Int32, _
```

 © 2007-2015 Analytica GmbH

```
                                                ByVal MonitorOn As Int32, _
                                                ByVal PortNumber As Int32, _
                                                ByVal TCPAddress As String, _
                                                ByVal Timeout As Int32) As Int32
Declare Function CANCloseDevice Lib "AnaGateCAN" (ByVal Handle As Int32) As Int32
Public Delegate Sub CAN_CALLBACK(ByVal ID As Int32, ByVal Buffer As IntPtr, _
                                 ByVal Bufferlen As Int32, ByVal Flags as Int32, _
                                 ByVal Handle as Int32)
Declare Function CANSetCallback Lib "AnaGateCAN" (ByVal Handle As Int32, _
                                                   ByVal MyCB As CAN_CALLBACK) As Int32

Sub CANCallback( ByVal ID As Int32, ByVal Buffer As IntPtr, _
                 ByVal Bufferlen As Int32, ByVal Flags as Int32, _
                 ByVal Handle as Int32)
  Dim Bytes as Byte(8)

  System.Runtime.InteropServices.Marshal.Copy(Buffer, Bytes, 0, Bufferlen )
  Console.Out.Write( "CAN-ID=" )
  Console.Out.Write( ID )
  Console.Out.Write( ",Data=" )
  For I As Int32 = 0 To BufferLen - 1
    Console.Out.Write( Bytes(I) )
  Next

End Sub

Function Main(ByVal CmdArgs() As String) As Integer

  'Opens the single CAN port of a AnaGate CAN
  Dim RC as Int32 = CANOpenDevice(Handle, 0, 1, 400, 0, "192.168.1.254", 1000)
  If RC = 0 Then
    CANSetCallback( Handle, AddressOf CANCallback )
  End If
  If RC = 0 Then
    CANCloseDevice( Handle )
  End If
End Function
```

7.3.2. SPI Console-Anwendung VB.NET

Das folgende Programmierbeispiel für VB.NET zeigt, wie eine Verbindung mit dem AnaGate SPI aufgebaut wird und Daten auf den SPI-Bus gesendet bzw. empfangen werden.

Anmerkung

Der vollständige Quellcode des vorliegenden Beispiels ist auf der jedem Gerät beiliegenden CD-ROM im Verzeichnispfad `Samples/SPI-VB.NET` zu finden.

```
Sub Main()

  Dim hHandle As Int32, nIndex As Integer
  Dim BufferWrite(100) As Byte, BufferRead(100) As Byte
  Dim nBaudrate As Int32 = 5000000 ' 500kBit
  Dim nSigLevel As Byte = 2   ' +3.3V for the signals.
  Dim nAuxVoltage As Byte = 0 ' support voltage is +3.3V.
  Dim nClockMode As Byte = 3  ' CPHA=1 and CPOL=1.

  Dim nRC = SPIOpenDevice(hHandle, "192.168.1.254", 5000) 1
  If nRC <> 0 Then
```


© 2007-2015 Analytica GmbH

```
    Console.WriteLine("Error SPIOpenDevice: " & GetErrorMsg(nRC) & vbCrLf)
  Else
    nRC = SPISetGlobals(hHandle, nBaudrate, nSigLevel, nAuxVoltage, nClockMode) 2
    nRC = SPIGetGlobals(hHandle, nBaudrate, nSigLevel, nAuxVoltage, nClockMode)

    For nIndex = 0 To 100  ' init buffers with some data
      BufferWrite(nIndex) = 69
      BufferRead(nIndex) = 96
    Next nIndex

    BufferWrite(0) = 5 * 16 ' 0x50 = READ STATUS (M25P80)
    BufferWrite(1) = 5 * 16 ' 0x50 = READ STATUS (M25P80)

    nRC = SPIDataReq(hHandle, BufferWrite, 2, BufferRead, 2) 3
    If nRC <> 0 Then
      Console.WriteLine("Error SPIDatReg: " & GetErrorMsg(nRC) & vbCrLf)
    Else
      Console.Write("Result: DATAREQ")
      For nIndex = 0 To 1  ' init buffers with some data
        Console.Write(BufferRead(nIndex) & " ")
      Next
      Console.WriteLine()
    End If

    SPICloseDevice(hHandle) 4
  End If
End Sub
```

1. Über einen Aufruf von `SPIOpenDevice` wird eine Netzwerkverbindung zum Gerät aufgebaut. Schlägt der Verbindungsaufbau fehl, so wird über die Funktion `GetErrorMsg` eine Fehlerbeschreibung anhand des Return-Codes ermittelt.
2. `SPISetGlobals` setzt die globalen Einstellungen auf dem Gerät (Baudrate, Signalspannung, Hilfsspannung, Clock-Modus).
3. Mit `SPIDataReq` werden Daten zum SPI-Bus gesendet. Die zurück gelesenen Daten werden im Empfangspuffer zur Verfügung gestellt, wenn die Funktion erfolgreich abgearbeitet wurde.
4. Die Verbindung zum Gerät wird mit `SPICloseDevice` geschlossen.

Die einzelnen Funktionen der Programmier-API sind über ein entsprechendes Wrapper-Modul definiert. Folgend ein Auszug des Wrapper-Moduls, das Deklarationen für alle vorhandenen Funktionen der API beinhaltet und auf der dem Gerät beiliegenden CD-Rom im Sample-Bereich zur Verfügung gestellt wird.

```
Imports System.Runtime.InteropServices

Namespace Analytica.AnaGate
  Public Module AnaGateAPI

    Declare Function SPIOpenDevice Lib "AnaGateSPI" (ByRef Handle As Int32, _
                                                     ByVal TCPAddress As String, _
                                                     ByVal Timeout As Int32) As Int32

    Declare Function SPICloseDevice Lib "AnaGateSPI" (ByVal Handle As Int32) As Int32

    Declare Function SPISetGlobals Lib "AnaGateSPI" (ByVal Handle As Int32, _
                                                     ByVal Baudrate As Int32, _
                                                     ByVal SigLevel As Byte, _
                                                     ByVal AuxVoltage As Byte, _
                                                     ByVal ClockMode As Byte) As Int32

    Declare Function SPIGetGlobals Lib "AnaGateSPI" (ByVal Handle As Int32, _
                                                     ByRef Baudrate As Int32, _
                                                     ByRef SigLevel As Byte, _
```


© 2007-2015 Analytica GmbH

```
                                                    ByRef AuxVoltage As Byte, _
                                                    ByRef ClockMode As Byte) As Int32

    Declare Function SPIDataReq Lib "AnaGateSPI" (ByVal Handle As Int32, _
                                                  <MarshalAs(UnmanagedType.LPArray)> _
                                                    ByVal BufferWrite() As Byte, _
                                                  ByVal BufferWriteLen As Int32, _
                                                  <MarshalAs(UnmanagedType.LPArray)> _
                                                    ByVal BufferRead() As Byte, _
                                                  ByVal BufferReadLen As Int32) As Int32

    Declare Function SPIErrorMessage Lib "AnaGateSPI" (ByVal RC As Int32, _
                                                       ByVal Buffer As IntPtr, _
                                                       ByVal Bufferlen As Int32) As Int32
  End Module
End Namespace
```


© 2007-2015 Analytica GmbH

Teil II. SocketCAN-Schnittstelle

Inhaltsverzeichnis

8. Die *SocketCAN*-Schnittstelle der *AnaGate*-Serie 91
9. Beschreibung des SocketCAN-Gateway 92
10. Verwendung des SocketCAN-Gateway 93
    10.1. Virtuelles SocketCAN-Netzwerkgerät 93
    10.2. SocketCANGateway 94
    10.3. SocketCAN-Beispielanwendung 96

 © 2007-2015 Analytica GmbH

Kapitel 8. Die *SocketCAN*-Schnittstelle der *AnaGate*-Serie

> *SocketCAN* ist eine Sammlung von CAN-Treibern und einer Netzwerkschicht, beigestellt von Volkswagen Research zum Linux Kernel als Open Source. Sie ist auch bekannt als Low Level CAN Framework (LLCF).
>
> —Wikipedia, *socketCAN*

Alle Geräte der *AnaGate CAN*-Serie werden über eine propriertäte Netzwerkschicht angesteuert. Um trotzdem ein Zugriff über die SocketCAN-Schnittstelle des Linux-Kernel zu ermöglichen, erfolgt die Kommunikation zur AnaGate-Hardware über ein sog. virtuelles SocketCAN-Netzwerkgerät. Die Datenverbindung zwischen der Hardware und der virtuellen Netzwerkschnittstelle wird über eine Gateway-Software, dem *SocketCANGetway*, realisiert.

© 2007-2015 Analytica GmbH

Kapitel 9. Beschreibung des SocketCAN-Gateway

Das *SocketCANGateway* der Analytica GmbH dient als Verbindung zwischen einer CAN-Schnittstelle eines Linux-Systems und einem CAN-Port eines *AnaGate CAN*-Gateways. Das *SocketCANGateway* setzt dabei die offene SocketCAN-Schnittstelle des Linux-Kernels auf das proprietäre TCP/IP-Protokoll des *AnaGate CAN* um.

Das SocketCAN-Konzept bildet einen CAN-Bus auf ein Netzwerkgerät ab. Über dieses Netzwerkgerät können beliebig viele Programme gleichzeitig auf den Bus zugreifen. Ein Netzwerkgerät kann mit einem lokalen CAN-Port des Linux-Systems verbunden sein. Alternativ besteht die Möglichkeit, virtuelle CAN-Netzwerkgeräte zu erzeugen, über die Anwendungen untereinander CAN-Telegramme austauschen können, ohne dass das Linux-System selbst eine physische CAN-Schnittstelle benötigt.

SocketCANGateway ist eine Anwendung, die sich sowohl zu einem SocketCAN-Netzwerkgerät als auch zu einem CAN-Port eines *AnaGate CAN*-Gateways verbindet. Sämtliche Telegramme, die auf einer Seite empfangen werden, werden an die jeweils andere Seite weitergeleitet. Dadurch können Anwendungen ein lokales virtuelles SocketCAN-Netzwerkgerät genauso benutzen, als wäre es direkt mit der CAN-Hardware verbunden.

© 2007-2015 Analytica GmbH

Kapitel 10. Verwendung des SocketCAN-Gateway

Um die *SocketCANGateway*-Anwendung zu starten, wird ein SocketCAN-Netzwerkgerät benötigt. In der Regel wird ein virtuelles Netzwerkgerät verwendet, über welches Linux-Anwendungen der Zugriff auf einen CAN-Port eines *AnaGate CAN*-Gateways ermöglicht wird. Prinzipiell kann *SocketCANGateway* aber auch verwendet werden, um eine lokale physische CAN-Schnittstelle des Linux-Systems mit einem *AnaGate CAN* zu verbinden.

10.1. Virtuelles SocketCAN-Netzwerkgerät

Zunächst muss das benötigte Treibermodul für virtuelle SocketCAN-Netzwerkgeräte geladen werden:

```
$ sudo modprobe vcan
```

Als Nächstes kann mit den folgenden Befehlen ein virtuelles SocketCAN-Netzwerkgerät angelegt und aktiviert werden:

```
$ sudo ip link add dev vcan0 type vcan
$ sudo ip link set up vcan0
```

Mit dem Befehl **ip link show vcan0** kann das Netzwerkgerät angezeigt werden:

```
$ ip link show vcan0
3: vcan0: <NOARP,UP,LOWER_UP> mtu 16 qdisc noqueue state UNKNOWN
    link/can
```

Mittels **ip link show** kann auch eine Liste aller aktuell vorhandenen Netzwerkgeräte ausgegeben werden:

```
$ ip link show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue state UNKNOWN
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
    link/ether 08:00:27:4d:39:d9 brd ff:ff:ff:ff:ff:ff
3: vcan0: <NOARP,UP,LOWER_UP> mtu 16 qdisc noqueue state UNKNOWN
    link/can
```

Falls auf mehrere Busse an unterschiedlichen CAN-Ports eines oder mehrerer *AnaGate CAN*-Gateways zugegriffen werden soll, sollte für jeden Bus ein separates Netzwerkgerät angelegt werden:

```
$ sudo ip link add dev vcan1 type vcan
$ sudo ip link set up vcan1
$ ip link show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue state UNKNOWN
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
    link/ether 08:00:27:4d:39:d9 brd ff:ff:ff:ff:ff:ff
3: vcan0: <NOARP,UP,LOWER_UP> mtu 16 qdisc noqueue state UNKNOWN
    link/can
4: vcan1: <NOARP,UP,LOWER_UP> mtu 16 qdisc noqueue state UNKNOWN
```


© 2007-2015 Analytica GmbH

```
    link/can
```

10.2. SocketCANGateway

© 2007-2015 Analytica GmbH

SocketCANGateway

SocketCANGateway — Eine Linux-Konsolenanwendung, die ein SocketCAN-Netzwerkgerät mit einem CAN-Port eines *AnaGate CAN*-Gateways verbindet. Die Anwendung ist als 32-Bit- und als 64-Bit-Version für x86-CPUs verfügbar.

Syntax

```
SocketCANGateway {interface}
[ -i | --ipaddress= ipaddress]
[ -b | --baudrate= baudrate]
[ -p | --canport= canport]
[ -t | --termination= { 0 | 1 }]
[ -s | --highspeed= { 0 | 1 }]
[ -d | --timestamp= { 0 | 1 }]
[ -h | --help | --version ]
```

Parameter

interface	Name des SocketCAN-Netzwerkgeräts.
-i, --ipaddress	IP-Adresse des *AnaGate CAN* (Standard 192.168.1.254).
-b, --baudrate	CAN-Bus-Baudrate in bps (Standard 1000000=1 MBit/s). Aktuell sind folgende Baudraten erlaubt: 10000 für 10 kBit/s, 20000 für 20 kBit/s, 50000 für 50 kBit/s, 62500 für 62,5 kBit/s, 100000 für 100 kBit/s, 125000 für 125 kBit/s, 250000 für 250 kBit/s, 500000 für 500 kBit/s und 1000000 für 1 MBit/s. Die Geräte AnaGate CAN uno/duo/quattro/USB/X2/X4/X8 unterstützen zusätzlich 800000 für 800 kBit/s.
-p, --canport	CAN-Port des *AnaGate CAN* (Standard 0=A).
-t, --termination	Bus-Terminierung (Standard 0=Aus).
-s, --highspeed	High-Speed-Modus (Standard 0=Aus).
-d, --timestamp	Zeitstempel in Nachrichten (Standard 0=Aus).
-h, --help	Gibt einen Hilfetext aus und beendet sich wieder.
--version	Gibt Versionsinformationen aus und beendet sich wieder.

Beschreibung

SocketCANGateway verbindet sich sowohl zu einem SocketCAN-Netzwerkgerät als auch zu einem CAN-Port eines *AnaGate CAN*-Gateways. Sämtliche Telegramme, die auf einer Seite empfangen werden, werden an die jeweils andere Seite weitergeleitet.

Beim Verbinden zum *AnaGate CAN*-Gateway wird der CAN-Port mit den gegebenen Parameter-Werten eingestellt. Werden keine Einstellungen als Parameter übergeben, werden die Standard-Werte gesetzt.

 © 2007-2015 Analytica GmbH

Im Folgenden ein Beispiel, das die virtuelle SocketCAN-Netzwerkschnittstelle vcan0 mit Port A eines *AnaGate CAN*-Gateways mit der IP-Adresse 192.168.1.254 verbindet und dessen Einstellungen explizit angibt.

```
$ sudo modprobe vcan
$ sudo ip link add dev vcan0 type vcan
$ sudo ip link set up vcan0
$ SocketCANGateway vcan0 --ipaddress=192.168.1.254 --baudrate=1000000 \
> --canport=0 --termination=1 --highspeed=1 --timestamp=0
****IP:192.168.1.254****
****Baudrate:1000000 1000000****
****Port:0****
****Termination:1****
****HighSpeed:1****
****Date/Time:0****
SocketCANGateway (Connection between SocketCAN and AnaGate
CAN), version 0.1.0 of Jul  9 2014
Copyright (C) 2014 Analytica GmbH

Hardware=vcan0...OK!

Press ^C to abort.
```

Beispiel 10.1. Programmaufruf

10.3. SocketCAN-Beispielanwendung

Der folgende C-Code ist ein Beispiel für die Verwendung der SocketCAN-API. Er versendet ein CAN-Telegram über die vcan0-Schnittstelle.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>

#include <net/if.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/ioctl.h>

#include <linux/can.h>
#include <linux/can/raw.h>

int
main(void)
{
 int s;
 int nbytes;
 struct sockaddr_can addr;
 struct can_frame frame;
 struct ifreq ifr;

 char *ifname = "vcan0";

 if((s = socket(PF_CAN, SOCK_RAW, CAN_RAW)) < 0) {
  perror("Error while opening socket");
  return -1;
 }
```

 © 2007-2015 Analytica GmbH

```
 strcpy(ifr.ifr_name, ifname);
 ioctl(s, SIOCGIFINDEX, &ifr);

 addr.can_family  = AF_CAN;
 addr.can_ifindex = ifr.ifr_ifindex;

 printf("%s at index %d\n", ifname, ifr.ifr_ifindex);

 if(bind(s, (struct sockaddr *)&addr, sizeof(addr)) < 0) {
  perror("Error in socket bind");
  return -2;
 }

 frame.can_id  = 0x123;
 frame.can_dlc = 2;
 frame.data[0] = 0x11;
 frame.data[1] = 0x22;

 nbytes = write(s, &frame, sizeof(struct can_frame));

 printf("Wrote %d bytes\n", nbytes);

 return 0;
}
```


© 2007-2015 Analytica GmbH

Teil III. Skriptsprache Lua

Inhaltsverzeichnis

11. Die Lua-Skripting-Schnittstelle der *AnaGate*-Serie 100
    11.1. Skriptdateien erstellen 101
    11.2. Skriptdateien auf einem PC ausführen 101
    11.3. Skriptdateien auf der *AnaGate*-Hardware ausführen 102
12. Allgemeine Funktionen 105
    LS_DeviceInfo 106
    LS_GetTime 107
    LS_Sleep 108
13. CAN Funktionen 109
    LS_CANOpenDevice 110
    LS_CANCloseDevice 112
    LS_CANRestartDevice 113
    LS_CANSetGlobals 114
    LS_CANGetGlobals 116
    LS_CANWrite 118
    LS_CANWriteEx 120
    LS_CANGetMessage 122
    LS_CANSetFilter 124
    LS_CANGetFilter 125
    LS_CANSetTime 126
    LS_CANErrorMessage 127
    LS_CANReadDigital 128
    LS_CANWriteDigital 130
    LS_CANReadAnalog 131
    LS_CANWriteAnalog 132
14. SPI Funktionen 133
    LS_SPIOpenDevice 134
    LS_SPICloseDevice 136
    LS_SPISetGlobals 137
    LS_SPIGetGlobals 139
    LS_SPIDataReq 141
    LS_SPIErrorMessage 143
    LS_SPIReadDigital 144
    LS_SPIWriteDigital 146
15. I2C-Funktionen 147
    LS_I2COpenDevice 148
    LS_I2COpenDeviceEx 150
    LS_I2CCloseDevice 152
    LS_I2CReset 153
    LS_I2CRead 154
    LS_I2CWrite 155
    LS_I2CSequence 156
    LS_I2CReadDigital 158
    LS_I2CWriteDigital 160
    LS_I2CErrorMessage 161
    LS_I2CReadEEProm 162
    LS_I2CWriteEEProm 164
16. Lua-Programmier-Beispiele 167
    16.1. Beispiele für Geräte mit CAN-Schnittstelle 167
    16.2. Beispiele für Geräte mit SPI-Schnittstelle 168
    16.3. Beispiele für Geräte mit I2C-Schnittstelle 169

© 2007-2015 Analytica GmbH

Kapitel 11. Die Lua-Skripting-Schnittstelle der *AnaGate*-Serie

> **Lua** [http://www.lua.org] (portugiesisch für Mond) ist eine imperative und erweiterbare Skriptsprache zum Einbinden in Programme, um diese leichter weiterentwickeln und warten zu können. Eine der besonderen Eigenschaften von *Lua* ist die geringe Größe des kompilierten Skript-Interpreters. *Lua* wurde 1993 von der *Computer Graphics Technology Group* der Päpstlichen Katholischen Universität von Rio de Janeiro in Brasilien entwickelt. Lua ist freie Software ...
>
> ... Insbesondere die geringe Größe von 120 kB, die Erweiterbarkeit und die hohe Geschwindigkeit verglichen mit anderen Skriptsprachen überzeugen viele Entwickler davon, *Lua* einzusetzen.
>
> —Wikipedia, *Lua*

Um mit der Skriptsprache *Lua* einfache Programmieraufgaben mit den Modellen der *AnaGate*-Serie lösen zu können, wurde der *Lua*-Interpreter um zusätzliche Funktionen zur Ansteuerung der verschiedenen Geräte erweitert. Diese Zusatzfunktionen sind im Detail nachfolgend dokumentiert und lehnen sich stark an die Bibliotheksfunktionen der *AnaGate API* an.

Quelltextdateien für *Lua* (kurz Skripte) werden auf dem PC unter Windows oder Linux in einem herkömmlichen Text-Editor erstellt bzw. bearbeitet. Anschließend wird das fertige Skript über die Eingabeaufforderung (bzw. Command-Shell) mit einem kostenlosen *Lua*-Interpreter ausgeführt. Dabei steht die gesamte Funktionalität der Skriptsprache *Lua* mit Funktions-Erweiterungen zur Programmierung der unterschiedlichen *AnaGate*-Hardware zur Verfügung.

Die Skriptsprache *Lua* eignet sich durch ihre geringe Laufzeitgröße und der guten Ausführungsgeschwindigkeit auch hervorragend für den Einsatz auf *eingebetteten Systemen*. Der *Lua*-Interpreter wurde deshalb in die Geräte-Firmware der *AnaGate*-Hardware[1] integriert, so dass erstellte Skripte nicht nur auf dem PC, sondern auch direkt auf dem Gerät ausgeführt werden können.

Anmerkung

Interessierten seien die beiden englischen Standardwerke *Lua Reference Manual* ([LuaRef2006-EN]) und *Programming in Lua* ([LuaProg2013-EN]) als gedruckte Nachschlagewerke empfohlen. Das letztgenannte Paperback kann auch in einer deutschen Übersetzung *Programmieren in Lua* ([LuaProg2013-DE]) bezogen werden. Das *Reference Manual*

[1] nur bei den Modellen AnaGate CAN uno, AnaGate CAN duo, AnaGate CAN quattro, AnaGate CAN USB und AnaGate Universal Programmer

© 2007-2015 Analytica GmbH

steht auch als Online-Version unter Lua.org [http://www.lua.org] zur Verfügung.

11.1. Skriptdateien erstellen

Die Erstellung und Bearbeitung von Skripten für die Skriptsprache *Lua* ist denkbar einfach und kann über einen herkömmlichen Texteditor erfolgen. Unter Windows-Betriebssystemen sind z.B. Notepad oder Wordpad geeignet, unter Linux etwa vi oder ähnliche.

Mittlerweile sind auch verschiedene teilweise kostenlose Editoren mit integriertem Syntax-Highlighting für *Lua* verfügbar, die eine Erleichterung beim Programmieren verschaffen.

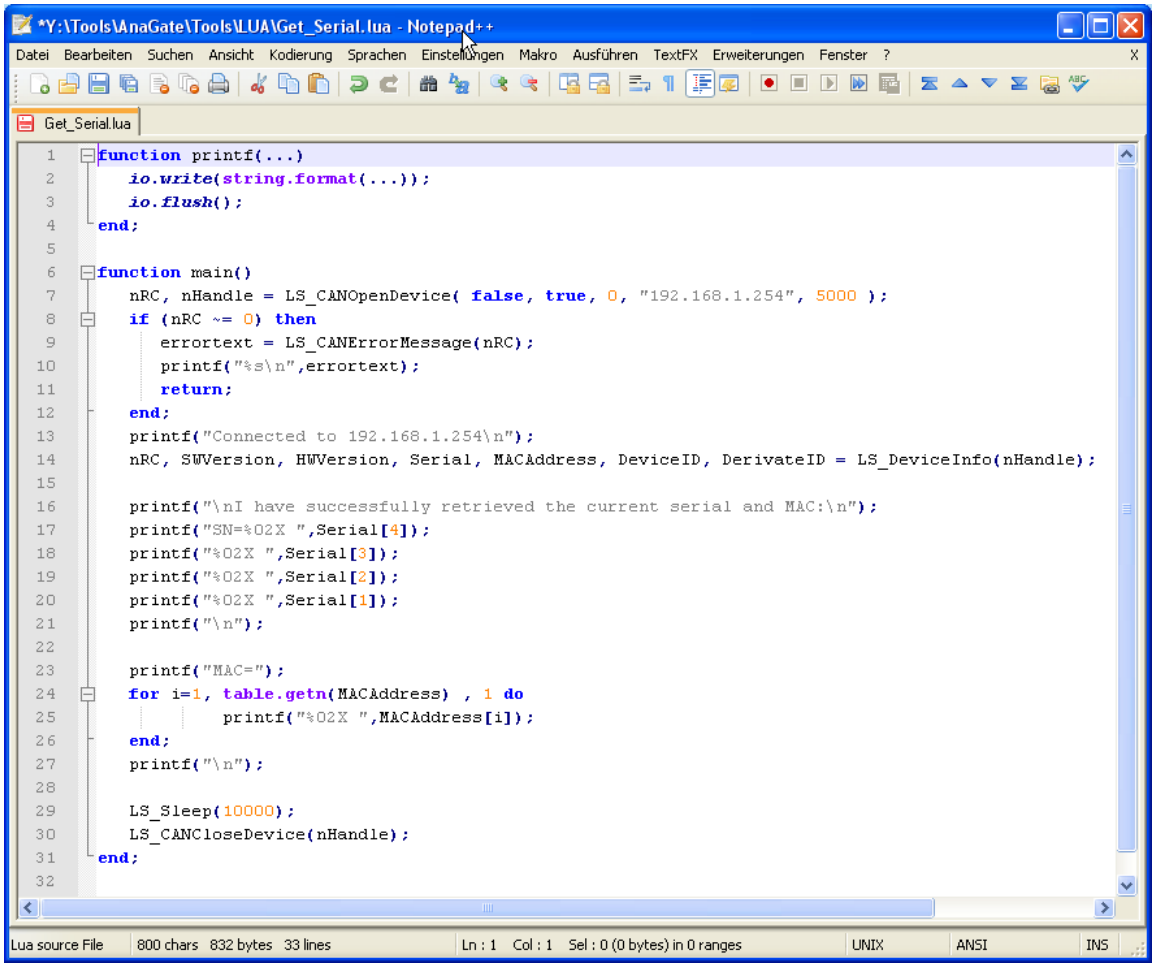


Abbildung 11.1. Bearbeiten von Lua-Skript im Texteditor

Sobald ein Skript fertiggestellt ist, kann es direkt wie im folgenden beschrieben auf dem PC ausgeführt und getestet werden.

11.2. Skriptdateien auf einem PC ausführen

Damit *Lua*-Skripte auf dem PC ausgeführt werden können, muss eine aktuelle Version des *Lua*-Interpreters auf dem PC verfügbar sein.

 © 2007-2015 Analytica GmbH

Auf der dem Gerät beiliegenden CD-ROM wird ein modifizierter *Lua*-Interpreter zur Verfügung gestellt, der auch die spezifischen Funktionserweiterungen für die *AnaGate*-Produkte beinhaltet. Dieser Interpreter besteht aus der einzelnen ausführbaren Datei `LUA.exe` und ist auf der CD-Rom im Verzeichnis `Lua` zu finden.

Tipp

Eine aktuelle Version der `LUA.exe` kann jederzeit über den Supportbereich der Produkthomepage [http://www.anagate.de/support/download.htm] kostenlos abgerufen werden.

Außer der Programmdatei `LUA.exe` werden auf dem PC keine zusätzlichen Programmdateien benötigt, so dass nur eine einzelne Datei auf die Computer-Festplatte (Fileserver, USB-Stick,...) kopiert werden muss.

Ausgeführt wird eine zuvor erstellte Skriptdatei über die Kommandozeilen-Eingabe. Dabei muss der Name der Skriptdatei als Programmparameter angegeben werden.

Im folgenden zeigt ein Beispiel, wie die Skriptdatei `get_serial.lua` in der Windows-Eingabeaufforderung gestartet wird.

```
T:\Tools\LUA>LUA.exe Get_Serial.lua 1
Connected to 192.168.1.254  2
I have successfully retrieved the current serial and MAC:
SN=01 02 02 1D
MAC=00 50 C2 3C B2 1D
T:\Tools\LUA>
```

1. Der Name der auszuführenden Skriptdatei muss dem Interpreter als Parameter übergeben werden.
2. Die Seriennummer und MAC-Adresse eines Gerätes mit IP-Adresse `192.168.1.254` werden ermittelt und über die Standardausgabe ausgegeben.

11.3. Skriptdateien auf der *AnaGate*-Hardware ausführen

Wie bereits bei der Einleitung zum Thema *Programmierung mit Lua* erwähnt, besteht die Möglichkeit, eigene Programm-Skripte über einen vorhandenen Lua-Skript-Interpreter auf der *AnaGate*-Hardware lokal auszuführen.

Das Laden bzw. Ausführen der Skriptdateien erfolgt über das HTTP-Interface des jeweiligen Gerätes und ist in der Anwendung leicht verständlich. Einen Überblick über die Schnittstelle wird im folgenden am Beispiel eines *AnaGate CAN uno* demonstriert.

 © 2007-2015 Analytica GmbH

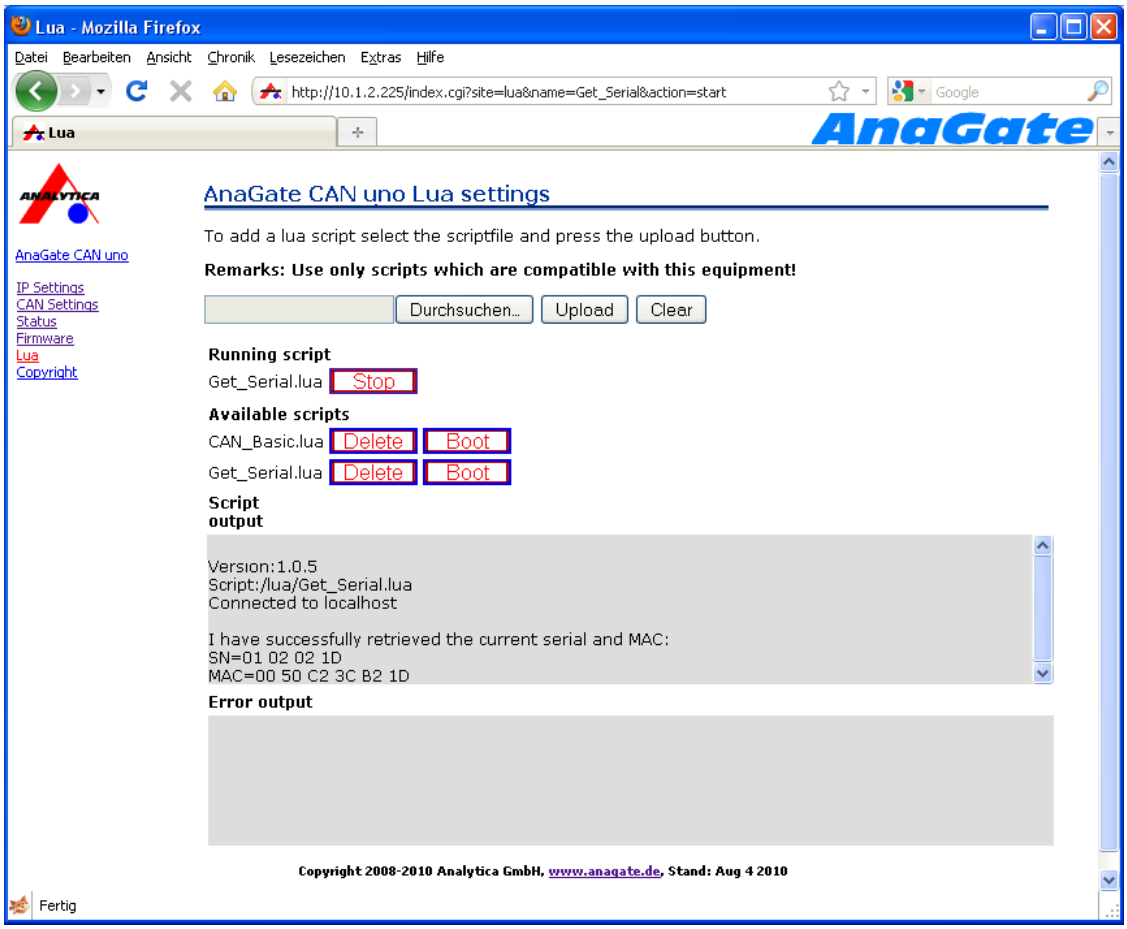


Abbildung 11.2. HTTP-Interface, Lua-Einstellungen

Durchsuchen.. Öffnet einen Dateiauswahl-Dialog für die Auswahl der Lua-Skriptdatei, die auf das Gerät geladen werden soll.

Upload Lädt die ausgewählte Skriptdatei auf das Gerät.

Clear Setzt die aktuelle Dateiauswahl zurück.

Boot script Skriptdatei, die beim Geräte-Hochlauf gestartet wird. Über die Schaltfläche **Delete** kann das Boot-Skript deaktiviert werden. Es kann jeweils nur ein Boot-Skript definiert werden.

Running script Zeigt das aktuell ausgeführte Skript an. Über die Schaltfläche **Stop** kann die Ausführung abgebrochen werden.

Available scripts Zeigt alle aktuell auf dem Gerät verfügbaren Skripte an.

Der Start eines Skripts erfolgt über die Schaltfläche **Start**. Über die Schaltfläche **Delete** kann ein Skript vom Gerät gelöscht werden und über **Boot** wird das entsprechende Skript als Boot-Skript definiert.

 © 2007-2015 Analytica GmbH

script output area	In diesem Ausgabebereich wird die Standardausgabe (stdout) des aktuell ausgeführten Skripts angezeigt. Über die Schaltfläche **Clear** kann der Textbereich gelöscht werden.
error output area	In diesem Ausgabebereich wird die Standardfehlerausgabe (stderr) des aktuell ausgeführten Skripts angezeigt. Über die Schaltfläche **Clear** kann der Textbereich gelöscht werden.

Tipp

Bei laufendem Skript werden die Textbereiche für die Standardausgabe und die Standardfehlerausgabe nicht automatisch aktualisiert. Durch ein erneutes Laden der Webseite können beide Textbereiche manuell aktualisiert werden.

 © 2007-2015 Analytica GmbH

Kapitel 12. Allgemeine Funktionen

© 2007-2015 Analytica GmbH

LS_DeviceInfo

LS_DeviceInfo — Ermittelt globale Informationen über eine Gerät der AnaGate-Serie.

Syntax

```
int RC, int SWVersion, int HWVersion, table(4) Serial, table(6)
MACAddress, int DeviceID, int SWDerivateID = LS_DeviceInfo(int Handle);
```

Parameter

hHandle Gültiges Zugriffs-Handle eines erfolgreichen Aufrufs von einer der Funktionen `LS_CANOpenDevice`, `LS_I2COpenDevice` oder `LS_SPIOpenDevice`.

Rückgabewerte

RC Die Funktion gibt im Erfolgsfall `0` zurück, andernfalls einen Fehlercode (Anhang A, *Rückgabewerte aus den API-Funktionen*).

SWVersion Firmware-Version. Die Versionsnummer besteht jeweils aus drei Zahlen (Major.Minor.Revision), die in einem 4-Byte-Ganzzahlwert abgelegt sind.

HWVersion Hardware-Version. Die Versionsnummer besteht jeweils aus drei Zahlen (Major.Minor.Revision), die in einem 4-Byte-Ganzzahlwert abgelegt ist.

Serial Seriennummer der AnaGate-Hardware (4 Byte).

MACAddress MAC-Adresse der AnaGate-Hardware (6 Byte).

nDeviceID Geräte-spezifische Kennung. Gibt an, um welchen Gerätetyp es sich handelt.

- 1 = AnaGate I2C
- 2 = AnaGate CAN
- 3 = AnaGate SPI
- 8 = AnaGate Universal Programmer
- 9 = AnaGate Renesas

SWDerivateID Gibt an, dass eine kundenspezifische Firmware auf dem Gerät installiert ist, wenn ungleich `0x00`.

Beschreibung

Gibt gerätespezifische Informationen über ein Gerät der AnaGate-Serie zurück.

Siehe auch

LS_CANOpenDevice, LS_SPIOpenDevice, LS_I2COpenDevice

 © 2007-2015 Analytica GmbH

LS_GetTime

LS_GetTime — Gibt die aktuelle Systemzeit zurück.

Syntax

```
int RC, table(2) Time = LS_GetTime();
```

Parameter

Die Funktion besitzt keine Übergabeparameter.

Rückgabewerte

RC	Die Funktion gibt im Erfolgsfall `0` zurück, andernfalls einen Fehlercode (Tabelle A.6, „Rückgabewerte für Lua Scripting").
Time[1], Time[2]	*`Time[1]`* gibt die Anzahl der Sekunden, die seit dem 01.01.1970 vergangen sind, an. In *`Time[2]`* sind zusätzlich die Sekunden-Bruchteile in Millisekunden angegeben.

Beschreibung

Gibt aktuelle Systemzeit als Anzahl der seit Mitternacht am 01.01.1970 verstrichenen Sekunden und Millisekunden zurück. Die Auflösung der Systemzeit ist auf Millisekunden genau.

© 2007-2015 Analytica GmbH

LS_Sleep

LS_Sleep — Wartet die angegebene Zeit in Millisekunden.

Syntax

```
int RC = LS_Sleep(unsigned int Milliseconds);
```

Parameter

nMilliseconds Gibt die zu wartende Zeit in Millisekunden an.

Rückgabewert

RC Die Funktion gibt im Erfolgsfall `0` zurück, andernfalls einen Fehlercode (Tabelle A.6, „Rückgabewerte für Lua Scripting“).

Beschreibung

Wartet die angegebene Zeit in Millisekunden.

 © 2007-2015 Analytica GmbH

Kapitel 13. CAN Funktionen

Mit den Funktionen der CAN API können alle CAN-Gateways der AnaGate-Serie angesprochen werden. Die Programmierschnittstelle ist für alle Geräte identisch und erfolgt generell über das Netzwerk-Protokoll TCP bzw. UDP.

Aktuell können folgende Geräte über die CAN API genutzt werden:

- AnaGate CAN
- AnaGate CAN uno
- AnaGate CAN duo
- AnaGate CAN quattro
- AnaGate CAN USB
- AnaGate CAN X2
- AnaGate CAN X4
- AnaGate CAN X8

Anmerkung

Die gesamte CAN-Funktionalität der AnaGate C-API steht auch den Lua-Anwendern über die nachfolgend dokumentierten Lua-Erweiterungen zur Verfügung.

© 2007-2015 Analytica GmbH

LS_CANOpenDevice

LS_CANOpenDevice — Baut eine Netzwerkverbindung (TCP) zu einem AnaGate CAN auf.

Syntax

```
int RC, int Handle = LS_CANOpenDevice(bool SendDataConfirm, bool SendDataInd, uint8 CANPort, string IPAddress, int Timeout);
```

Parameter

SendDataConfirm — Sollen die gesendeten bzw. empfangenen Telegramme von der Gegenseite bestätigt werden? Ohne Bestätigung ist eine höhereÜbertragungsperformance zu erreichen.

SendDataInd — Gibt an, ob das AnaGate CAN empfangende Telegramme weiterleiten soll. Alle eingehenden Telegramme werden verworfen, falls dieser Parameter auf false gesetzt wird.

CANPort — Gibt die CAN-Schnittstelle an, die verwendet werden soll. Erlaubte Werte sind:

- `0` für Port A (Modelle AnaGate CAN uno, AnaGate CAN duo, AnaGate CAN quattro, AnaGate CAN USB und AnaGate CAN)
- `1` für Port B (AnaGate CAN duo, AnaGate CAN quattro)
- `2` für Port C (AnaGate CAN quattro)
- `3` für Port D (AnaGate CAN quattro)

IPAddress — Netzwerkadresse des AnaGate-Partners.

Timeout — Standard-Timeout für AnaGate-Zugriffe in Millisekunden.

Ein Timeout wird festgestellt, wenn die AnaGate-Hardware nicht innerhalb der vereinbarten Timeout-Zeit antwortet. Diese Timeout-Zeit gilt auf der aktiven Netzwerkverbindung für alle Kommandos bzw. Funktionen, für die kein spezifischer Timeout-Wert definiert werden kann.

Rückgabewert

RC — Die Funktion gibt im Erfolgsfall `0` zurück, andernfalls einen Fehlercode (Anhang A, *Rückgabewerte aus den API-Funktionen*).

Handle — Zugriffs-Handle, falls die Verbindung zum Device erfolgreich hergestellt wurde.

Beschreibung

Baut eine Netzwerkverbindung (TCP) zu einer CAN-Schnittstelle eines Gerätes der AnaGate CAN Serie auf. Erst nach dem erfolgreichen Verbinden zur CAN-Schnittstelle ist ein Zugriff auf den CAN-Bus möglich.

 © 2007-2015 Analytica GmbH

Durch einen Aufruf der Funktion `CANCloseDevice` wird die bestehende Verbindung wieder geschlossen.

Wichtig

Das explizite Schließen der Verbindung ist notwendig, um die in der DLL angelegten Systemressourcen wieder freizugeben und dem verbundenen Gerät mitzuteilen, dass die Verbindung nicht mehr genutzt wird und wieder für neue Verbindungsanfragen zur Verfügung gestellt werden soll.

Im folgenden ein Programmier-Beispiel für den initialen Zugriff auf eine Schnittstelle.

```
-- open: use no confirmations and receive incoming CAN data
local nRC, Handle = LS_CANOpenDevice(false, true, 0, "192.168.1.254", 5000)
if nRC == 0 then
   -- now do something
   LS_CANCloseDevice(Handle);
end
```

Siehe auch

LS_CANCloseDevice

LS_CANRestartDevice

© 2007-2015 Analytica GmbH

LS_CANCloseDevice

LS_CANCloseDevice — Schließt eine geöffnete Netzwerk-Verbindung zu einem AnaGate CAN Device.

Syntax

```
int RC = LS_CANCloseDevice(int Handle);
```

Parameter

Handle Gültiges Zugriffs-Handle eines erfolgreichen Aufrufs von `LS_CANOpenDevice`.

Rückgabewert

RC Die Funktion gibt im Erfolgsfall `0` zurück, andernfalls einen Fehlercode (Anhang A, *Rückgabewerte aus den API-Funktionen*).

Beschreibung

Schließt eine geöffnete Netzwerk-Verbindung zu einem AnaGate CAN Device. Das übergebene Handle *`Handle`* ist ein Rückgabewert aus einem vorangegangenen erfolgreichen Aufruf der Funktion `LS_CANOpenDevice`.

Wichtig

Das explizite Schließen der Verbindung ist notwendig, um die in der DLL angelegten Systemressourcen wieder freizugeben und dem verbundenen Gerät mitzuteilen, dass die Verbindung nicht mehr genutzt wird und wieder für neue Verbindungsanfragen zur Verfügung gestellt werden soll.

Siehe auch

LS_CANOpenDevice

 © 2007-2015 Analytica GmbH

LS_CANRestartDevice

LS_CANRestartDevice — Führt einen Geräte-Restart der AnaGate CAN Hardware durch.

Syntax

```
int RC = LS_CANRestartDevice(string IPAddress, int Timeout);
```

Parameter

IPAddress Netzwerkadresse des AnaGate-Partners.

nTimeout Standard-Timeout für AnaGate-Zugriffe in Millisekunden. Ein Timeout wird festgestellt, wenn der AnaGate-Partner nicht innerhalb der vereinbarten Timeout-Zeit antwortet.

Rückgabewert

RC Die Funktion gibt im Erfolgsfall `0` zurück, andernfalls einen Fehlercode (Anhang A, *Rückgabewerte aus den API-Funktionen*).

Beschreibung

Führt einen Wiederhochlauf der AnaGate CAN Hardware unter der angegebenen Netzwerkadresse durch und schließt damit alle noch offenen Netzwerkverbindungen. Das Restart-Kommando ist auch dann noch möglich, wenn die maximale Anzahl von gleichzeitigen Verbindungen bereits erreicht ist.

Wichtig

Dieses Kommando sollte nur verwendet werden, falls eine Verbindung zum Gerät notwendig, aber aktuell nicht möglich ist, da die maximale Anzahl gleichzeitiger Verbindungen bereits erreicht ist.

Siehe auch

LS_CANOpenDevice

 © 2007-2015 Analytica GmbH

LS_CANSetGlobals

LS_CANSetGlobals — Setzt die globalen Einstellungen, mit denen auf dem CAN-Bus gearbeitet werden soll.

Syntax

```
int RC = LS_CANSetGlobals(int Handle, uint32 Baudrate, uint8 OperatingMode, bool Termination, bool HighSpeedMode, bool TimeStampOn);
```

Parameter

Handle — Gültiges Zugriffs-Handle eines erfolgreichen Aufrufs von `LS_CANOpenDevice`.

Baudrate — Baudrate, mit der gearbeitet werden soll. Folgende Werte werden unterstützt:

- 10.000 für 10kBit
- 20.000 für 20kBit
- 50.000 für 50kBit
- 62.500 für 62,5kBit
- 100.000 für 100kBit
- 125.000 für 125kBit
- 250.000 für 250kBit
- 500.000 für 500kBit
- 800.000 für 800kBit (nicht AnaGate CAN)
- 1.000.000 für 1MBit

OperatingMode — Betriebsmodus, in dem gearbeitet werden soll. Folgende Werte werden unterstützt:

- 0 = Standard-Modus.
- 1 = LoopBack-Modus: In diesem Modus werden keine Telegramme auf dem CAN-Bus versendet. Stattdessen werden an das AnaGate CAN versendete Nachrichten so wieder empfangen, als ob sie von einem anderen Bus-Teilnehmer über den CAN-Bus übertragen worden wären.
- 2 = Listen-Modus: In diesem Modus arbeitet das AnaGate CAN als passiver Partner am Bus, d.h. es werden grundsätzlich keine Telegramme über das Gerät versendet (dies gilt auch bei ACKs für eingehende Telegramme).
- 3 = Offline-Modus: In diesem Modus werden keine Telegramme auf dem CAN-Bus versendet oder empfangen. Dadurch

 © 2007-2015 Analytica GmbH

werden auch keine Error-Frames auf dem Bus erzeugt, wenn andere Busteilnehmer Telegramme mit einer anderen als der konfigurierten Baudrate senden.

Termination — Geräte-integrierte CAN-Bus-Terminierung ein- bzw. ausschalten (`true`=ein, `false`=aus). Diese Einstellung wird nicht bei allen AnaGate CAN Modellvarianten unterstützt.

HighSpeedMode — Aktuelle Einstellung für den High-Speed Modus (`true`=ein, `false`=aus). Diese Einstellung wird nicht bei allen AnaGate CAN Modellvarianten unterstützt.

Der Highspeed-Modus wurde eingeführt, um auch bei großen Baudraten mit kontinuierlich hoher Buslast keine Pakete zu verlieren. In diesem Modus werden die gesendeten bzw. empfangenen Telegramme auf Protokollebene nicht mehr bestätigt und die via `LS_CANSetFilter` definierten Filter werden ignoriert.

TimeStampOn — Aktuelle Einstellung für den Zeitstempel-Modus (`true`=ein, `false`=aus). Diese Einstellung wird nicht bei allen AnaGate CAN Modellvarianten unterstützt.

Im Zeitstempel-Modus wird mit dem CAN-Telegramm ein Zeitstempel übertragen. Beim Senden gibt der Zeitstempel den Zeitpunkt an, zu dem die Nachricht vom CAN-Controller versendet wurde. Analog ist bei empfangenen Nachrichten der Zeitstempel der Zeitpunkt, zu dem die Nachricht vom CAN-Controller empfangen wurde.

Rückgabewert

RC Die Funktion gibt im Erfolgsfall `0` zurück, andernfalls einen Fehlercode (Anhang A, *Rückgabewerte aus den API-Funktionen*).

Beschreibung

Setzt die globalen Geräte-Einstellungen der verwendeten CAN-Schnittstelle. Diese Einstellungen sind für alle gleichzeitigen Verbindungen auf der entsprechenden CAN-Schnittstelle gültig. Die Einstellungen bleiben nicht permanent im Gerät gespeichert, sie sind nach einem Geräte-Neustart undefiniert.

Bemerkung

Die Einstellungen für die CAN-Bus-Terminierung, den High-Speed-Modus und den Zeitstempel können beim AnaGate CAN (Hardware-Version 1.1.A) nicht vorgenommen werden. Die Angaben werden vom Gerät ignoriert.

Der Offline-Modus wird erst ab der Geräte-Firmwareversion 1.3.12 unterstützt. Das AnaGate CAN unterstützt diesen Modus generell nicht.

Siehe auch

LS_CANGetGlobals

 © 2007-2015 Analytica GmbH

LS_CANGetGlobals

LS_CANGetGlobals — Ermittelt die globalen Einstellungen, mit denen auf dem CAN-Bus gearbeitet wird.

Syntax

```
int RC, int Baudrate, uint8 OperatingMode, bool Termination, bool
HighSpeedMode, bool TimeStampOn = LS_CANGetGlobals(int Handle);
```

Parameter

Handle Gültiges Zugriffs-Handle eines erfolgreichen Aufrufs von `LS_CANOpenDevice`.

Rückgabewerte

RC Die Funktion gibt im Erfolgsfall `0` zurück, andernfalls einen Fehlercode (Anhang A, *Rückgabewerte aus den API-Funktionen*).

nBaudrate Aktuell eingestellte Baudrate.

OperatingMode Aktuell eingestellter Betriebsmodus. Folgende Werte sind möglich:

- 0 = Standard-Modus.
- 1 = LoopBack-Modus: In diesem Modus werden keine Telegramme auf dem CAN-Bus versendet. Stattdessen werden an das AnaGate CAN versendete Nachrichten so wieder empfangen, als ob sie von einem anderen Bus-Teilnehmer über den CAN-Bus übertragen worden wären.
- 2 = Listen-Modus: In diesem Modus arbeitet das AnaGate CAN als passiver Partner am Bus, d.h. es werden grundsätzlich keine Telegramme über das Gerät versendet (dies gilt auch bei ACKs für eingehende Telegramme).
- 3 = Offline-Modus: In diesem Modus werden keine Telegramme auf dem CAN-Bus versendet oder empfangen. Dadurch werden auch keine Error-Frames auf dem Bus erzeugt, wenn andere Busteilnehmer Telegramme mit einer anderen als der konfigurierten Baudrate senden.

Termination Aktuelle Einstellung für CAN-Bus-Terminierung (`true`=ein, `false`=aus). Diese Einstellung wird nicht bei allen AnaGate CAN Modellvarianten unterstützt.

HighSpeedMode Aktuelle Einstellung für den High-Speed Modus (`true`=ein, `false`=aus). Diese Einstellung wird nicht bei allen AnaGate CAN Modellvarianten unterstützt.

Der Highspeed-Modus wurde eingeführt, um auch bei großen Baudraten mit kontinuierlich hoher Buslast keine Pakete zu

 © 2007-2015 Analytica GmbH

verlieren. In diesem Modus werden die gesendeten bzw. empfangenen Telegramme auf Protokollebene nicht mehr bestätigt und die via `LS_CANSetFilter` definierten Filter werden ignoriert.

Beschreibung

Ermittelt die globalen Geräte-Einstellungen der verwendeten CAN-Schnittstelle. Diese Einstellungen sind für alle gleichzeitigen Verbindungen auf der entsprechenden CAN-Schnittstelle gültig.

Bemerkung

Die Einstellungen für die CAN-Bus-Terminierung, den High-Speed-Modus und den Zeitstempel können beim AnaGate CAN (Hardware-Version 1.1.A) nicht vorgenommen werden. Die Angaben werden vom Gerät ignoriert.

Der Offline-Modus wird erst ab der Geräte-Firmwareversion 1.3.12 unterstützt. Das AnaGate CAN unterstützt diesen Modus generell nicht.

Siehe auch

LS_CANSetGlobals

© 2007-2015 Analytica GmbH

LS_CANWrite

LS_CANWrite — Sendet ein Datentelegramm über die AnaGate-Hardware auf den CAN-Bus.

Syntax

```
int RC = LS_CANWrite(int Handle, int32 CANID, table(uint8[DataLen])
Data, uint8 DataLen, uint8 Flags);
```

Parameter

Handle — Gültiges Zugriffs-Handle eines erfolgreichen Aufrufs von `LS_CANOpenDevice`.

CANID — CAN-Identifier des Absenders. Mittels *`Flags`* kann definiert werden, ob die Adresse im sog. Extended Format (29-Bit-Adresse) oder Standard-Format (11-Bit-Adresse) vorliegt.

Data — Datenpuffer mit den Telegrammdaten.

DataLen — Länge des Datenpuffers. Größere Werte als 8 werden ignoriert.

Flags — Mit den Format-Flags kann das Sendeverhalten beeinflusst werden:

- Bit 0: Falls gesetzt 29-bit CAN Identifier (Extended Format), sonst 11-bit (Standard format).
- Bit 1: Falls gesetzt, wird das Telegramm als Remote-Telegramm versendet.

Rückgabewert

RC Die Funktion gibt im Erfolgsfall `0` zurück, andernfalls einen Fehlercode (Anhang A, *Rückgabewerte aus den API-Funktionen*).

Beschreibung

Die Funktion sendet ein Datentelegramm auf den CAN-Bus analog zu der Function `LS_CANWriteEx`.

Mit `LS_CANWriteEx` kann zusätzlich der Zeitpunkt auf dem Gerät ermittelt werden, zu dem das Telegramm tatsächlich versendet wurde.

Anmerkung

Mit Hilfe eines Remoteframes kann ein Teilnehmer einen anderen auffordern, seine Daten zu senden. Die Datenlänge muss entsprechend der zu erwartenden Datenlänge gesetzt werden, auf dem CAN-Bus selbst werden dabei keine Daten versendet.

Bei Verwendung der Funktionen `LS_CANWrite` bzw. `LS_CANWriteEx` ist beim Versenden von Remoteframes sowohl ein Datenpuffer als auch

 © 2007-2015 Analytica GmbH

die Länge des Puffers entsprechend der zu erwartenden Datenlänge anzugeben.

Im folgenden ein Programmier-Beispiel, das ein Datentelegramm auf den CAN-Bus sendet.

```
local tabData = {}
for i = 1, 8, 1 do
   table.insert(tabData, i)
end

local nFlags = 0x0 //  11bit address + standard (not remote frame)
local nCANId = 0x25 // send with CAN ID 0x25;

local nRC, hHandle = LS_CANOpenDevice(true, true, 0, "192.168.1.254", 5000)
if nRC == 0 then
   // send 8 bytes with CAN id 37
   nRC = LS_CANWrite(hHandle, nCANId, tabData, #tabData, nFlags)

   // send a remote frame to CAN id 37 (request 4 data bytes)
   nRC = LS_CANWrite( hHandle, nCANId, tabData, 4, 0x02 )

   LS_CANCloseDevice(hHandle)
end
```

Bemerkung

Für Geräte vom Typ AnaGate CAN (Hardware-Version 1.1.A) ist die Funktion `LS_CANWriteEx` mit `LS_CANWrite` identisch. Die Rückgabewerte *`nSeconds`* und *`nMicroseconds`* werden nicht gesetzt.

Siehe auch

LS_CANWriteEx

© 2007-2015 Analytica GmbH

LS_CANWriteEx

LS_CANWriteEx — Sendet ein Datentelegramm über die AnaGate-Hardware auf den CAN-Bus.

Syntax

```
int RC, int32 Seconds, int32 Microseconds = LS_CANWriteEx(int Handle,
int32 CANID, table(uint8[DataLen]) Data, uint8 DataLen, uint8 Flags);
```

Parameter

Handle — Gültiges Zugriffs-Handle eines erfolgreichen Aufrufs von `LS_CANOpenDevice`.

CANID — CAN-Identifier des Absenders. Mittels *`Flags`* kann definiert werden, ob die Adresse im sog. Extended Format (29-Bit-Adresse) oder Standard-Format (11-Bit-Adresse) vorliegt.

Data — Datenpuffer mit den Telegrammdaten.

DataLen — Länge des Datenpuffers. Größere Werte als 8 werden ignoriert.

nFlags — Mit den Format-Flags kann das Sendeverhalten beeinflusst werden:

- Bit 0: Falls gesetzt 29-bit CAN Identifier (Extended Format), sonst 11-bit (Standard format).
- Bit 1: Falls gesetzt, wird das Telegramm als Remote-Telegramm versendet.

Rückgabewert

RC — Die Funktion gibt im Erfolgsfall `0` zurück, andernfalls einen Fehlercode (Anhang A, *Rückgabewerte aus den API-Funktionen*).

Seconds — Zeitstempel der Quittung vom CAN-Controller in Sekunden seit dem 01.01.1970.

Microseconds — Anteil der Mikrosekunden am Zeitstempel.

Beschreibung

Die Funktion sendet ein Datentelegramm auf den CAN-Bus analog zu der Function `LS_CANWrite`.

Mit `LS_CANWriteEx` kann zusätzlich der Zeitpunkt auf dem Gerät ermittelt werden, zu dem das Telegramm tatsächlich versendet wurde.

Anmerkung

Mit Hilfe eines Remoteframes kann ein Teilnehmer einen anderen auffordern, seine Daten zu senden. Die Datenlänge muss entsprechend

© 2007-2015 Analytica GmbH

der zu erwartenden Datenlänge gesetzt werden, auf dem CAN-Bus selbst werden dabei keine Daten versendet.

Bei Verwendung der Funktionen `LS_CANWrite` bzw. `LS_CANWriteEx` ist beim Versenden von Remoteframes sowohl ein Datenpuffer als auch die Länge des Puffers entsprechend der zu erwartenden Datenlänge anzugeben.

Im folgenden ein Programmier-Beispiel, das ein Datentelegramm auf den CAN-Bus sendet.

```
local tabData = {}
for i = 1, 8, 1 do
   table.insert(tabData, i)
end

local nFlags = 0x0 //  11bit address + standard (not remote frame)
local nCANId = 0x25 // send with CAN ID 0x25;

local nRC, hHandle = LS_CANOpenDevice(true, true, 0, "192.168.1.254", 5000);
if nRC == 0 then
  local nSeconds, nMicroSeconds
   // send 8 bytes with CAN id 37
   nRC, nSeconds, nMicroSeconds = LS_CANWriteEx(hHandle, nCANId, tabData, #tabData, nFlags)

   // send a remote frame to CAN id 37 (request 4 data bytes)
   nRC, nSeconds, nMicroSeconds = LS_CANWriteEx(hHandle, nCANId, tabData, 4, 0x02)

   LS_CANCloseDevice(hHandle)
end
```

Bemerkung

Für Geräte vom Typ AnaGate CAN (Hardware-Version 1.1.A) ist die Funktion `LS_CANWriteEx` mit `LS_CANWrite` identisch. Die Rückgabewerte *`nSeconds`* und *`nMicroseconds`* werden nicht gesetzt.

Siehe auch

LS_CANWrite

© 2007-2015 Analytica GmbH

LS_CANGetMessage

LS_CANGetMessage — Liest eine CAN-Message aus dem internen Empfangspuffer.

Syntax

```
int Available, int32 CANID, uint8 DataLen, table(uint8[Length]) Data,
uint8 Flags, int32 Seconds, int32 Microseconds = LS_CANGetMessage(int
Handle, int Timeout);
```

Parameter

Handle — Gültiges Zugriffs-Handle eines erfolgreichen Aufrufs von `LS_CANOpenDevice`.

Timeout — Maximale Zeitspanne in Millisekunden, die auf ein neues Datentelegramm gewartet werden soll.

Rückgabewerte

Available — Anzahl der Nachrichten, die nach dem Aufruf von `LS_CANGetMessage` noch im Nachrichtenpuffer zum Abruf vorhanden sind. Ist aktuell keine Nachricht verfügbar, enthält der Rückgabewert die Konstante `-10` (ERR_NO_DATA).

CANID — CAN-Identifier des Absenders. Mittels *`Flags`* kann definiert werden, ob die Adresse im sog. Extended Format (29-Bit-Adresse) oder Standard-Format (11-Bit-Adresse) vorliegt.

DataLen — Länge des Datenpuffers.

Data — Datenpuffer mit den Telegrammdaten.

Flags — Mit den Format-Flags kann das Sendeverhalten beeinflusst werden:

- Bit 0: Falls gesetzt 29-bit CAN Identifier (Extended Format), sonst 11-bit (Standard format).
- Bit 1: Falls gesetzt, wird das Telegramm als Remote-Telegramm versendet.
- Bit 2: Falls gesetzt, enthält das Telegramm Timestamp-Informationen.

Seconds — Zeitstempel der Quittung vom CAN-Controller in Sekunden seit dem 01.01.1970.

Microseconds — Anteil der Mikrosekunden am Zeitstempel.

Beschreibung

Die Funktion liest ein CAN-Datentelegramm aus einem internen Message-Puffer, der automatisch nebenläufig mit den empfangenen Datenpaketen befüllt wird.

 © 2007-2015 Analytica GmbH

Über den Parameter *Timeout* kann gesteuert werden, wie lange die Funktion auf ein neues Datenpaket warten soll, wenn aktuell keine Telegramme im internen Puffer vorhanden sind. Ist nach Ablauf der angegebenen Wartezeit kein Paket empfangen worden, gibt die Funktion in *Available* den Rückgabewert `-10` (ERR_NO_DATA) zurück.

Im folgenden ein Programmier-Beispiel, das eingehende Datentelegramme verarbeitet.

```
local nRC, hHandle = LS_CANOpenDevice(true, true, 0, "192.168.1.254", 5000)
if nRC == 0 then
   -- set globals: 500Kbit, standard mode, termination on, no high speed, no timestamp
   nRC = LS_CANSetGlobals(hHandle, 500000, 0, true, false, false)
   local nCurMsg = 0

   repeat
      local nAvail, ID, Len, Data, Flags, Sec, Microsec = LS_CANGetMessage(hHandle, 100)
      if nAvail >= 0 then
         nCurMsg = nCurMsg + 1

         -- now do something with the incomming message data
         io.write(string.format("ID: %.8x", ID))  -- for example, write out CAN id
      else
         LS_Sleep(25)  -- wait 25 ms if no message available
      end
   until nCurMsg >= 100  -- read only 100 messages, then stop

   LS_CANCloseDevice(hHandle)
end
```

Bemerkung

Für Geräte vom Typ AnaGate CAN (Hardware-Version 1.1.A) werden die Rückgabewerte *Seconds* und *Microseconds* nicht gesetzt.

Siehe auch

LS_CANWrite

LS_CANWriteEx

 © 2007-2015 Analytica GmbH

LS_CANSetFilter

LS_CANSetFilter — Setzt die Software-Filter für die aktuelle Verbindung.

Syntax

```
int RC = LS_CANSetFilter(int Handle, table(uint32[16]) Filter);
```

Parameter

Handle Gültiges Zugriffs-Handle eines erfolgreichen Aufrufs von `LS_CANOpenDevice`.

Filter Tabelle mit 8 Filtereinträgen (jeweils 4 Maskenfilter und 4 Bereichsfilter). Ein Filtereintrag besteht grundsätzlich aus zwei 32-Bit-Werten. Es müssen immer alle Filter gleichzeitig gesetzt werden. Sollen Filtereinträge unbenutzt bleiben, sind beim Maskenfilter beide Filterwerte mit 0 und beim Bereichsfilter der Startwert mit 0 und der Endewert mit der höchstmöglichen Zahl (0x1FFFFFFF) zu besetzen.

Rückgabewert

RC Die Funktion gibt im Erfolgsfall `0` zurück, andernfalls einen Fehlercode (Anhang A, *Rückgabewerte aus den API-Funktionen*).

Beschreibung

Setzt die Software-Filter für die aktuelle Verbindung. Durch Filter werden nur Nachrichten mit definierten CAN-IDs von der AnaGate CAN Hardware an den jeweiligen Verbindungspartner weitergeleitet.

Ein Maskenfilter besteht aus der Filtermaske, die angibt, welche Bits des CAN-Identifier geprüft werden sollen, und dem entsprechenden Filterwert. Alle eingehenden Telegramme, die in der Filtermaske nicht mit dem Filterwert übereinstimmen, werden nicht an den Partner weitergeleitet.

Ein Bereichsfilter definiert einen Bereich durch eine Start- und Ende-Adresse. Liegt der CAN-Identifier eines Telegramms innerhalb dieses Bereichs, wird die Nachricht an den Partner weitergeleitet.

Standardmäßig sind keine Filter gesetzt, so dass alle CAN-IDs weiter geleitet werden. Wird über die Funktion `LS_CANSetGlobals` der Highspeed-Modus aktiviert, werden alle Filter ignoriert, um den Datensatzdurchsatz zu erhöhen.

Siehe auch

LS_CANGetFilter

 © 2007-2015 Analytica GmbH

LS_CANGetFilter

LS_CANGetFilter — Liefert die Filtereinstellungen für die aktuelle Verbindung zurück.

Syntax

```
int RC, table(uint32[16]) Filter = LS_CANGetFilter(int hHandle);
```

Parameter

Handle Gültiges Zugriffs-Handle eines erfolgreichen Aufrufs von `LS_CANOpenDevice`.

Rückgabewert

RC Die Funktion gibt im Erfolgsfall `0` zurück, andernfalls einen Fehlercode (Anhang A, *Rückgabewerte aus den API-Funktionen*).

Filter Tabelle mit 8 Filtereinträgen (jeweils 4 Maskenfilter und 4 Bereichsfilter). Ein Filtereintrag besteht grundsätzlich aus zwei 32-Bit-Werten. Bei unbenutzten Maskenfiltereinträgen sind beide Filterwerte mit 0 besetzt. Bei unbenutzten Bereichsfilterwerten ist ein Eintrag mit dem Wertepaar (0,0x1FFFFFFF) besetzt.

Beschreibung

Liest die aktuelle Einstellung der Software-Filter für die aktuelle Verbindung zurück. Durch Filter werden nur Nachrichten mit definierten CAN-IDs von der AnaGate CAN Hardware an den jeweiligen Verbindungspartner weitergeleitet.

Siehe auch

LS_CANSetFilter

 © 2007-2015 Analytica GmbH

LS_CANSetTime

LS_CANSetTime — Setzt die System-Uhrzeit auf dem AnaGate CAN Gerät für den Zeitstempel-Modus.

Syntax

```
int RC = LS_CANSetTime(int Handle, int32 Seconds, int32 Microseconds);
```

Parameter

Handle Gültiges Zugriffs-Handle eines erfolgreichen Aufrufs von `CANOpenDevice`.

Seconds Aktuelle Uhrzeit in Sekunden seit dem 01.01.1970.

Microseconds Zusätzlicher Anteil der Mikrosekunden für die aktuelle Uhrzeit.

Rückgabewert

RC Die Funktion gibt im Erfolgsfall `0` zurück, andernfalls einen Fehlercode (Anhang A, *Rückgabewerte aus den API-Funktionen*).

Beschreibung

Mit der Funktion `LS_CANSetTime` kann die System-Uhrzeit auf dem AnaGate-Gerät voreingestellt werden. Das Anpassen der System-Uhrzeit ist vor allem dann sinnvoll, falls der Zeitstempel-Modus auf der Verbindung aktiviert ist.

Falls über die Funktion `LS_CANSetGlobals` der Zeitstempel-Modus eingeschaltet worden ist, enthalten alle empfangenen CAN-Nachrichten einen zusätzlichen Zeitstempel, der angibt, zu welchem Zeitpunkt die Nachricht empfangen wurde. Beim Senden von CAN-Nachrichten wird ein entsprechender Zeitstempel über die Quittung des Schreibkommandos an den Sender zurück übermittelt, der angibt, zu welchem Zeitpunkt die Nachricht vom CAN-Controller quittiert wurde (dies nur falls die sog. Confirmations aus Performance-Gesichtspunkten eingeschaltet sind).

Bemerkung

Die Uhrzeit-Einstellung für den Zeitstempel-Modus können beim AnaGate CAN (Hardware-Version 1.1.A) nicht vorgenommen werden. Die Angaben werden vom Gerät ignoriert.

 © 2007-2015 Analytica GmbH

LS_CANErrorMessage

LS_CANErrorMessage — Gibt eine textuelle Beschreibung eines Rückgabewertes aus einer API-Funktion zurück.

Syntax

```
string ErrorMsg = LS_CANErrorMessage(int RetCode);
```

Parameter

RetCode Fehlercode, dessen Fehlerbeschreibung ermittelt werden soll.

Rückgabewert

ErrorMsg Textuelle Beschreibung des Fehlercodes (in englischer Sprache).

Beschreibung

Gibt eine textuelle Beschreibung des übergebenen Rückgabewertes zurück (siehe auch Anhang A, *Rückgabewerte aus den API-Funktionen*). Alle Fehlertexte sind in englischer Sprache.

Im folgenden ein Programmier-Beispiel in Lua.

```
local nRC = 0
local sErrorText = 'No Error'

//... call an API function here

sErrorText = LS_CANErrorMessage(nRC)
print(sErrorText)
```


© 2007-2015 Analytica GmbH

LS_CANReadDigital

LS_CANReadDigital — Liest die aktuellen Werte der digitalen Ein-/Ausgabe-Register der AnaGate-Hardware zurück.

Syntax

`int RC, uint32 InputBits, uint32 OutputBits = **LS_CANReadDigital**(int *Handle*);`

Parameter

Handle Gültiges Zugriffs-Handle eines erfolgreichen Aufrufs von `LS_CANOpenDevice`.

Rückgabewert

RC Die Funktion gibt im Erfolgsfall `0` zurück, andernfalls einen Fehlercode (Anhang A, *Rückgabewerte aus den API-Funktionen*).

InputBits Aktueller Inhalt des Registers mit den digitalen Eingängen. Dabei entsprechen Bit 0 bis Bit 3 den vier digitalen Eingängen; Bit 4 bis Bit 31 sind auf 0 gesetzt.

OutputBits Aktueller Inhalt des Registers mit den digitalen Ausgängen. Dabei entsprechen Bit 0 bis Bit 3 den vier digitalen Ausgängen; Bit 4 bis Bit 31 sind auf 0 gesetzt.

Beschreibung

Alle Geräte der AnaGate-Serie (außer der Gerätevariante AnaGate CAN uno im Hutschienengehäuse) besitzen auf der Gehäuserückseite Anschlüsse für jeweils 4 digitale Eingänge und 4 digitale Ausgänge. Das AnaGate CAN uno im Hutschienengehäuse besitzt dagegen an der Oberseite des Gehäuses Anschlüsse für jeweils 2 digitale Ein- und Ausgänge.

Die aktuellen Zustände der digitalen Eingänge und Ausgänge können mit der Funktion `LS_CANReadDigital` ermittelt werden.

Im folgenden ein Programmier-Beispiel, das die digitalen IOs setzt und zurück liest.

```
local nOutputs = 0x03

local nRC, hHandle = LS_CANOpenDevice(false, false, 0, "192.168.1.254", 5000)
if nRC == 0 then
   -- set the digital output register (PIN 0 and PIN 1 to HIGH value)
   nRC = LS_CANWriteDigital(hHandle, nOutputs)
   local nInputs
   -- read all input and output registers
   nRC, nInputs, nOutputs = LS_CANReadDigital(hHandle)

   LS_CANCloseDevice(hHandle)
end
```

 © 2007-2015 Analytica GmbH

Siehe auch

LS_CANWriteDigital

© 2007-2015 Analytica GmbH

LS_CANWriteDigital

LS_CANWriteDigital — Setzt das digitale Ausgabe-Register der AnaGate-Hardware auf einen neuen Wert.

Syntax

```
int RC = LS_CANWriteDigital(int Handle, uint32 OutputBits);
```

Parameter

Handle	Gültiges Zugriffs-Handle eines erfolgreichen Aufrufs von `LS_CANOpenDevice`.
OutputBits	Neuer Wert des Registers mit den digitalen Ausgängen. Dabei entsprechen Bit 0 bis Bit 3 den vier digitalen Ausgängen; Bit 4 bis Bit 31 sind reserviert und auf 0 zu setzen.

Rückgabewert

RC Die Funktion gibt im Erfolgsfall `0` zurück, andernfalls einen Fehlercode (Anhang A, *Rückgabewerte aus den API-Funktionen*).

Beschreibung

Alle Geräte der AnaGate-Serie (außer der Gerätevariante AnaGate CAN uno im Hutschienengehäuse) besitzen auf der Gehäuserückseite Anschlüsse für jeweils 4 digitale Eingänge und 4 digitale Ausgänge. Das AnaGate CAN uno im Hutschienengehäuse besitzt dagegen an der Oberseite des Gehäuses Anschlüsse für jeweils 2 digitale Ein- und Ausgänge.

Die digitalen Ausgänge können mit der Funktion `LS_CANWriteDigital` verändert werden.

Ein Programmier-Beispiel zum Lesen/Schreiben der IOs ist bei der Beschreibung von `LS_CANReadDigital` zu finden.

Siehe auch

LS_CANReadDigital

 © 2007-2015 Analytica GmbH

LS_CANReadAnalog

LS_CANReadAnalog — Liest die aktuellen Werte der analogen Eingänge der AnaGate-Hardware zurück.

Syntax

`int RC, uint32 PowerSupply, table(uint8[min(ReadCount,InputCount)]) AnalogInputs, uint16 ReadCount = **LS_CANReadAnalog**(int *Handle*, uint16 *InputCount*);`

Parameter

Handle	Gültiges Zugriffs-Handle eines erfolgreichen Aufrufs von `LS_CANOpenDevice`.
InputCount	Anzahl der analogen Eingänge der AnaGate-Hardware.

Rückgabewert

RC	Die Funktion gibt im Erfolgsfall `0` zurück, andernfalls einen Fehlercode (Anhang A, *Rückgabewerte aus den API-Funktionen*).
PowerSupply	Aktueller Versorgungsspannung in Millivolt.
AnalogInputs	Tabelle mit den aktuellen Werten der analogen Eingänge in Millivolt.
ReadCount	Anzahl der gelesenen analogen Eingänge.

Beschreibung

Die Geräte der AnaGate CAN X-Serie besitzen an der Oberseite des Gehäuses Anschlüsse für jeweils 4 analoge Ein- und Ausgänge.

Die aktuellen Zustände der analogen Eingänge sowie die aktuelle Versorgungsspannung können mit der Funktion `LS_CANReadAnalog` ermittelt werden.

Im folgenden ein Programmier-Beispiel, das die analogen Eingänge liest.

```
local nInputs = 4

local nRC, hHandle = LS_CANOpenDevice(false, false, 0, "192.168.1.254", 5000)
if nRC == 0 then
   local nPowerSupply, tAnalogInputs, nReadCount
   -- read input values
   nRC, nPowerSupply, tAnalogInputs, nReadCount = LS_CANReadAnalog(hHandle, nInputs)

   LS_CANCloseDevice(hHandle)
end
```

Siehe auch

LS_CANWriteAnalog

 © 2007-2015 Analytica GmbH

LS_CANWriteAnalog

LS_CANWriteAnalog — Setzt die analogen Ausgänge der AnaGate-Hardware auf neue Werte.

Syntax

`int RC = **LS_CANWriteAnalog**(int *Handle*, table(uint32[OutputCount]) *AnalogOutputs*, uint16 *OutputCount*);`

Parameter

Handle	Gültiges Zugriffs-Handle eines erfolgreichen Aufrufs von `LS_CANOpenDevice`.
AnalogOutputs	Array mit neuen Werten der analogen Ausgänge in Millivolt.
OutputCount	Anzahl der Werte in AnalogOutputs.

Rückgabewert

RC Die Funktion gibt im Erfolgsfall `0` zurück, andernfalls einen Fehlercode (Anhang A, *Rückgabewerte aus den API-Funktionen*).

Beschreibung

Die Geräte der AnaGate CAN X-Serie besitzen an der Oberseite des Gehäuses Anschlüsse für jeweils 4 analoge Ein- und Ausgänge.

Die analogen Ausgänge können mit der Funktion `LS_CANWriteAnalog` verändert werden. Die tatsächlich an den analogen Ausgängen anliegende Spannung ist nach oben durch die Versorgungsspannung des AnaGate-Geräts begrenzt. Die aktuelle Versorgungsspannung kann mit der Funktion `LS_CANReadAnalog` ausgelesen werden.

Im folgenden ein Programmier-Beispiel, das die analogen Ausgänge setzt.

```
local aOutputs = {0, 12000, 24000, 0}

local nRC, hHandle = LS_CANOpenDevice(false, false, 0, "192.168.1.254", 5000)
if nRC == 0 then
   -- write output values
   nRC = LS_CANWriteAnalog(hHandle, aOutputs, #aOutputs)

   LS_CANCloseDevice(hHandle)
end
```

Siehe auch

LS_CANReadAnalog

 © 2007-2015 Analytica GmbH

Kapitel 14. SPI Funktionen

Der Serial Peripheral Interface (kurz SPI) ist ein von Motorola entwickeltes Bus-System für einen synchronen seriellen Datenbus, mit dem digitale Schaltungen nach dem Master-Slave-Prinzip miteinander verbunden werden können. Die SPI-Gateways aus der AnaGate-Serie bieten einen Zugriff auf den SPI Bus über einen herkömmlichen Netzwerkanschluss.

Mit den Funktionen der SPI API können diese SPI-Gateways und damit der SPI Bus auf einfache Art und Weise angesprochen werden. Die Programmierschnittstelle ist für alle Geräte identisch und erfolgt generell über das Netzwerk-Protokoll TCP.

Aktuell können folgende Geräte über die SPI API genutzt werden:

- AnaGate SPI
- AnaGate Universal Programmer

Anmerkung

Die gesamte SPI-Funktionalität der AnaGate C-API steht auch den Lua-Anwendern über die nachfolgend dokumentierten Lua-Erweiterungen zur Verfügung.

© 2007-2015 Analytica GmbH

LS_SPIOpenDevice

LS_SPIOpenDevice — Baut eine Netzwerkverbindung zu einem AnaGate SPI auf.

Syntax

```
int RC, int Handle = LS_SPIOpenDevice(string IPAddress, int Timeout);
```

Parameter

IPAddress — Netzwerkadresse des AnaGate-Partners.

Timeout — Standard-Timeout für AnaGate-Zugriffe in Millisekunden.

Ein Timeout wird festgestellt, wenn die AnaGate-Hardware nicht innerhalb der vereinbarten Timeout-Zeit antwortet. Diese Timeout-Zeit gilt auf der aktiven Netzwerkverbindung für alle Kommandos bzw. Funktionen, für die kein spezifischer Timeout-Wert definiert werden kann.

Rückgabewerte

RC — Die Funktion gibt im Erfolgsfall `0` zurück, andernfalls einen Fehlercode (Anhang A, *Rückgabewerte aus den API-Funktionen*).

Handle — Zugriffs-Handle, falls die Verbindung zum Device erfolgreich hergestellt wurde.

Beschreibung

Baut eine Netzwerkverbindung über TCP/IP zu einem AnaGate SPI (bzw. AnaGate Universal Programmer) auf. Erst nach dem erfolgreichen Verbinden mit dem Gerät ist ein Zugriff auf den SPI-Bus möglich.

Anmerkung

Das AnaGate SPI (bzw. die SPI-Schnittstelle eines AnaGate Universal Programmers) erlaubt nur eine einzige Netzwerkverbindung. Solange eine bestehende Verbindung aufrechterhalten wird, wird jeder neuer Verbindungsversuch abgelehnt.

Im folgenden ein Programmier-Beispiel für den initialen Zugriff auf das Gerät.

```
local nRC, nHandle = LS_SPIOpenDevice("192.168.1.254", 5000)
if nRC ~= 0 then
   print(LS_SPIErrorMessage(nRC))
   os.exit()
end

-- now do something

LS_SPICloseDevice(nHandle)
```

 © 2007-2015 Analytica GmbH

Siehe auch

LS_SPICloseDevice

© 2007-2015 Analytica GmbH

LS_SPICloseDevice

LS_SPICloseDevice — Schließt eine geöffnete Netzwerk-Verbindung zu einem AnaGate SPI Device.

Syntax

```
int RC = LS_SPICloseDevice(int Handle);
```

Parameter

Handle Gültiges Zugriffs-Handle eines erfolgreichen Aufrufs von `LS_SPIOpenDevice`.

Rückgabewert

RC Die Funktion gibt im Erfolgsfall `0` zurück, andernfalls einen Fehlercode (Anhang A, *Rückgabewerte aus den API-Funktionen*).

Beschreibung

Schließt eine geöffnete Netzwerk-Verbindung zu einem AnaGate SPI Device. Das übergebene Handle *`Handle`* ist ein Rückgabewert aus einem vorangegangenen erfolgreichen Aufruf der Funktion `LS_SPIOpenDevice`.

Wichtig

Das explizite Schließen der Verbindung ist notwendig, um die in der DLL angelegten Systemressourcen wieder freizugeben und dem verbundenen Gerät mitzuteilen, dass die Verbindung nicht mehr genutzt wird und wieder für neue Verbindungsanfragen zur Verfügung gestellt werden soll.

Siehe auch

LS_SPIOpenDevice

 © 2007-2015 Analytica GmbH

LS_SPISetGlobals

LS_SPISetGlobals — Setzt die globalen Einstellungen, mit denen auf dem AnaGate SPI gearbeitet werden soll.

Syntax

```
int RC = LS_SPISetGlobals(int Handle, uint32 Baudrate, uint8 SigLevel,
uint8 AuxVoltage, uint8 ClockMode);
```

Parameter

Handle — Gültiges Zugriffs-Handle eines erfolgreichen Aufrufs von `LS_SPIOpenDevice`.

Baudrate — Baudrate, mit der gearbeitet werden soll. Werte können individuell eingestellt werden, z.B.

- 500.000 für 500 kBit
- 1.000.000 für 1 MBit
- 5.000.000 für 5 MBit

Anmerkung

Die gewünschte Baudrate kann u.U. von dem tatsächlich verwendeten Wert abweichen, da die Taktfrequenz auf dem Gerät nur in Abhängigkeit der Hardware (Schwingungsdauer des Quarz) eingestellt werden kann. Ist die exakte Einstellung einer angegebenen Baudrate nicht möglich, wird der nächstkleinere mögliche Wert eingestellt.

SigLevel — Gibt den Pegelwert für SPI-Signale an. Folgende Werte werden unterstützt:

- 0 für Ausgänge im High Impedance Modus (Standard-Modus).
- 1 für +5,0 V für die Signale.
- 2 für +3,3 V für die Signale.
- 3 für +2,5 V für die Signale.

AuxVoltage — Gibt die Ausgangsspannung für die Hilfsspannungsversorgung an. Folgende Werte sind möglich:

- 0 für Hilfsspannung +3,3 V.
- 1 für Hilfsspannung +2,5 V für die Signale.

ClockMode — Gibt die Phase und die Polarität der Clock-Leitung für die Datenübertragung an. Folgende Werte sind möglich:

- 0 für CPHA=0 und CPOL=0.

 © 2007-2015 Analytica GmbH

- 1 für CPHA=0 und CPOL=1.
- 2 für CPHA=1 und CPOL=0.
- 3 für CPHA=1 und CPOL=1.

Rückgabewert

RC Die Funktion gibt im Erfolgsfall `0` zurück, andernfalls einen Fehlercode (Anhang A, *Rückgabewerte aus den API-Funktionen*).

Beschreibung

Setzt die globalen Geräte-Einstellungen, mit denen auf der SPI-Schnittstelle des AnaGate SPI bzw. AnaGate Universal Programmers gearbeitet werden soll. Die Einstellungen bleiben nicht permanent im Gerät gespeichert, sie sind nach einem Geräte-Neustart undefiniert.

Siehe auch

LS_SPIGetGlobals

 © 2007-2015 Analytica GmbH

LS_SPIGetGlobals

LS_SPIGetGlobals — Ermittelt die globalen Einstellungen, mit denen auf dem AnaGate SPI gearbeitet wird.

Syntax

```
int RC, uint32 Baudrate, uint8 SigLevel, uint8 AuxVoltage, uint8
ClockMode = LS_SPIGetGlobals(int Handle);
```

Parameter

Handle Gültiges Zugriffs-Handle eines erfolgreichen Aufrufs von `LS_SPIOpenDevice`.

Rückgabewerte

RC Die Funktion gibt im Erfolgsfall `0` zurück, andernfalls einen Fehlercode (Anhang A, *Rückgabewerte aus den API-Funktionen*).

Baudrate Aktuell auf dem SPI-Bus eingestellte Baudrate in Bit pro Sekunde.

SigLevel Aktuell eingestellter Pegelwert für die SPI-Signale. Folgende Werte sind möglich:

- 0 für Ausgänge im High Impedance Modus (Standard-Modus).
- 1 für +5,0 V für die Signale.
- 2 für +3,3 V für die Signale.
- 3 für +2,5 V für die Signale.

AuxVoltage Aktuell eingestellte Ausgangsspannung für die Hilfsspannungsversorgung. Folgende Werte sind möglich:

- 0 für Hilfsspannung +3,3 V.
- 1 für Hilfsspannung +2,5 V für die Signale.

ClockMode Aktuell eingestellter Clock-Modus (Phase und Polarität der Clock-Leitung für die Datenübertragung). Folgende Werte sind möglich:

- 0 für CPHA=0 und CPOL=0.
- 1 für CPHA=0 und CPOL=1.
- 2 für CPHA=1 und CPOL=0.
- 3 für CPHA=1 und CPOL=1.

Beschreibung

Ermittelt die globalen Geräte-Einstellungen, mit denen auf der SPI-Schnittstelle des AnaGate SPI bzw. AnaGate Universal Programmers gearbeitet wird.

 © 2007-2015 Analytica GmbH

Siehe auch

LS_SPISetGlobals

© 2007-2015 Analytica GmbH

LS_SPIDataReq

LS_SPIDataReq — Führt einen Datentransfer auf dem SPI-Bus durch.

Syntax

```
int RC, table(uint8[ReadLen]) ReadData = LS_SPIDataReq(int Handle,
table(uint8[WriteLen]) WriteData, uint16 WriteLen, uint16 ReadLen);
```

Parameter

Handle	Gültiges Zugriffs-Handle eines erfolgreichen Aufrufs von `LS_SPIOpenDevice`.
WriteData	Puffer mit den Daten, die an den SPI-Partner gesendet werden sollen.
WriteLen	Länge des Datenpuffers *`WriteData`* (Anzahl Bytes).
ReadLen	Anzahl der Bytes, die gelesen werden sollen.

Rückgabewert

RC	Die Funktion gibt im Erfolgsfall `0` zurück, andernfalls einen Fehlercode (Anhang A, *Rückgabewerte aus den API-Funktionen*).
ReadData	Enthält die von der Gegenstelle empfangenen Daten.

Beschreibung

Sendet Daten auf den SPI-Bus und empfängt Daten vom SPI-Bus.

Daten werden auf dem SPI-Bus immer auf zwei Leitungen vollduplex (SDO und SDI) übertragen. Die Funktion `LS_SPIDataReq` arbeitet bedingt durch die räumliche Trennung zum SPI-Bus jedoch zwangsläufig in zwei Schritten. Zuerst wird der Schreibdatenpuffer in einem Netzwerkdatenpaket an das AnaGate SPI gesendet, das dann den eigentlichen Datentransfer auf dem SPI-Bus durchführt. Nach erfolgreicher Kommunikation auf dem SPI-Bus sendet das Anagate SPI eine Quittung mit den gelesenen Daten zurück, die dann im Lesedatenpuffer abgelegt werden.

Wichtig

Es ist hardwaretechnisch nicht möglich zu erkennen, ob tatsächlich ein Baustein am SPI-Bus angeschlossen ist. Auch wenn kein Baustein angeschlossen ist, wird vom AnaGate SPI die angeforderte Anzahl von Datenbytes zurückgeliefert - der Lese-Datenbuffer wird in diesem Fall mit Null-Werten aufgefüllt.

Im folgenden ein Programmier-Beispiel, das Daten auf den SPI-Bus sendet und empfängt.

```
local tabWrite = {}
for i = 1, 10, 1 do
   table.insert(tabWrite, i)
```

 © 2007-2015 Analytica GmbH

```
end

local nRC, hHandle = SPIOpenDevice("192.168.1.254", 5000)
if nRC == 0 then
   local tabRead
   // send 1 byte and receive 1 byte
   nRC, tabRead = LS_SPIDataReq(hHandle, tabWrite, 1, 1)
   // send 1 byte and receive 5 byte
   nRC, tabRead = LS_SPIDataReq(hHandle, tabWrite, 1, 5)
   // send 2 byte and receive 1 byte
   nRC, tabRead = LS_SPIDataReq(hHandle, tabWrite, 2, 1)

   LS_SPICloseDevice(hHandle)
end
```


© 2007-2015 Analytica GmbH

LS_SPIErrorMessage

LS_SPIErrorMessage — Gibt eine textuelle Beschreibung eines Rückgabewertes aus einer API-Funktion zurück.

Syntax

```
string ErrorMsg = LS_SPIErrorMessage(int ErrorCode);
```

Parameter

ErrorCode Fehlercode, dessen Fehlerbeschreibung ermittelt werden soll.

Rückgabewert

ErrorMsg Textuelle Beschreibung des Fehlercodes (in englischer Sprache).

Beschreibung

Gibt eine textuelle Beschreibung des übergebenen Rückgabewertes zurück (siehe auch Anhang A, *Rückgabewerte aus den API-Funktionen*). Alle Fehlertexte sind in englischer Sprache.

Im folgenden ein Programmier-Beispiel in Lua.

```
local nRC = 0
local sErrorText = 'No Error'

//... call an API function here

sErrorText = LS_SPIErrorMessage(nRC)
print(sErrorText)
```


© 2007-2015 Analytica GmbH

LS_SPIReadDigital

LS_SPIReadDigital — Liest die aktuellen Werte der digitalen Ein-/Ausgabe-Register der AnaGate-Hardware zurück.

Syntax

`int RC, uint32 InputBits, uint32 OutputBits = **LS_SPIReadDigital**(int *Handle*);`

Parameter

Handle Gültiges Zugriffs-Handle eines erfolgreichen Aufrufs von `LS_SPIOpenDevice`.

Rückgabewert

RC Die Funktion gibt im Erfolgsfall `0` zurück, andernfalls einen Fehlercode (Anhang A, *Rückgabewerte aus den API-Funktionen*).

InputBits Aktueller Inhalt des Registers mit den digitalen Eingängen. Dabei entsprechen Bit 0 bis Bit 3 den vier digitalen Eingängen; Bit 4 bis Bit 31 sind auf 0 gesetzt.

OutputBits Aktueller Inhalt des Registers mit den digitalen Ausgängen. Dabei entsprechen Bit 0 bis Bit 3 den vier digitalen Ausgängen; Bit 4 bis Bit 31 sind auf 0 gesetzt.

Beschreibung

Alle Geräte der AnaGate-Serie (außer der Gerätevariante AnaGate CAN uno im Hutschienengehäuse) besitzen auf der Gehäuserückseite Anschlüsse für jeweils 4 digitale Eingänge und 4 digitale Ausgänge. Das AnaGate CAN uno im Hutschienengehäuse besitzt dagegen an der Oberseite des Gehäuses Anschlüsse für jeweils 2 digitale Ein- und Ausgänge.

Die aktuellen Zustände der digitalen Eingänge und Ausgänge können mit der Funktion `LS_SPIReadDigital` ermittelt werden.

Im folgenden ein Programmier-Beispiel, das die digitalen IOs setzt und zurück liest.

```
local nOutputs = 0x03

local nRC, hHandle = LS_SPIOpenDevice(400000, "192.168.1.254", 5000)
if nRC == 0 then
   // set the digital output register (PIN 0 and PIN 1 to HIGH value)
   nRC = LS_SPIWriteDigital(hHandle, nOutputs)
   local nInputs
   // read all input and output registers
   nRC, nInputs, nOutputs = LS_SPIReadDigital(hHandle)

   LS_SPICloseDevice(hHandle)
end
```

 © 2007-2015 Analytica GmbH

Siehe auch

LS_SPIWriteDigital

© 2007-2015 Analytica GmbH

LS_SPIWriteDigital

LS_SPIWriteDigital — Setzt das digitale Ausgabe-Register der AnaGate-Hardware auf einen neuen Wert.

Syntax

```
int RC = LS_SPIWriteDigital(int Handle, uint32 OutputBits);
```

Parameter

Handle	Gültiges Zugriffs-Handle eines erfolgreichen Aufrufs von `LS_SPIOpenDevice`.
OutputBits	Neuer Wert des Registers mit den digitalen Ausgängen. Dabei entsprechen Bit 0 bis Bit 3 den vier digitalen Ausgängen; Bit 4 bis Bit 31 sind reserviert und auf 0 zu setzen.

Rückgabewert

RC Die Funktion gibt im Erfolgsfall `0` zurück, andernfalls einen Fehlercode (Anhang A, *Rückgabewerte aus den API-Funktionen*).

Beschreibung

Alle Geräte der AnaGate-Serie (außer der Gerätevariante AnaGate CAN uno im Hutschienengehäuse) besitzen auf der Gehäuserückseite Anschlüsse für jeweils 4 digitale Eingänge und 4 digitale Ausgänge. Das AnaGate CAN uno im Hutschienengehäuse besitzt dagegen an der Oberseite des Gehäuses Anschlüsse für jeweils 2 digitale Ein- und Ausgänge.

Die digitalen Ausgänge können mit der Funktion `LS_SPIWriteDigital` verändert werden.

Ein Programmier-Beispiel zum Lesen/Schreiben der IOs ist bei der Beschreibung von `LS_SPIReadDigital` zu finden.

Siehe auch

LS_SPIReadDigital

 © 2007-2015 Analytica GmbH

Kapitel 15. I2C-Funktionen

I2C, für engl. Inter-Integrated Circuit, im Deutschen gesprochen als I-Quadrat-C, ist ein von Philips Semiconductors (heute NXP Semiconductors) entwickelter serieller Datenbus. Er wird hauptsächlich geräteintern für die Kommunikation zwischen verschiedenen Schaltungsteilen mit geringer Übertragungsgeschwindigkeit benutzt, z. B. zwischen einem Controller und Peripherie-ICs. Technisch benötigt I2C lediglich zwei Signalleitungen: Takt (engl. serial clock line, SCL) und Datenleitung (engl. serial data line, SDA). Der Datentransfer kann bidirektional 8-bit orientiert erfolgen, und zwar mit bis zu 100 kbit/s im Standard-Modus, bis zu 400 kbit/s in Fast-mode, bis zu 1 Mbit/s in Fast-mode Plus (Fm+) oder bis zu 3.4 Mbit/s im High-speed Modus.[NXP-I2C]

Einige Hersteller verwenden die Bezeichnung TWI (Two-Wire Interface), obwohl I2C kein eingetragenes Markenzeichen von NXP Semiconductors ist. Technisch sind TWI und I2C identisch.

Die I2C-Gateways aus der AnaGate-Serie bieten einen Zugriff auf den I2C-Bus über einen herkömmlichen Netzwerkanschluss. Mit den Funktionen der I2C API können diese I2C-Gateways und damit der I2C-Bus auf einfache Art und Weise angesprochen werden. Die Programmierschnittstelle ist für alle Geräte identisch und erfolgt generell über das Netzwerk-Protokoll TCP.

Aktuell können folgende Geräte über die I2C API genutzt werden:

- AnaGate I2C
- AnaGate Universal Programmer

© 2007-2015 Analytica GmbH

LS_I2COpenDevice

LS_I2COpenDevice — Baut eine Netzwerkverbindung zu einem AnaGate I2C (bzw. AnaGate Universal Programmer) auf.

Syntax

```
int RC, int Handle = LS_I2COpenDevice(uint32 Baudrate, string IPAddress,
int Timeout);
```

Parameter

Baudrate — Baudrate, mit der der I2C-Bus betrieben werden soll. Die Werte können individuell eingestellt werden, z.B.

- 100000 für 100 kBit (Standard Mode)
- 400000 für 400 kBit (Fast Mode)

Anmerkung

Größere Werte als 400 kBit werden auf dem AnaGate SPI ignoriert.

IPAddress — Netzwerkadresse des AnaGate-Partners.

Timeout — Standard-Timeout für AnaGate-Zugriffe in Millisekunden.

Ein Timeout wird festgestellt, wenn die AnaGate-Hardware nicht innerhalb der vereinbarten Timeout-Zeit antwortet. Diese Timeout-Zeit gilt auf der aktiven Netzwerkverbindung für alle Kommandos bzw. Funktionen, für die kein spezifischer Timeout-Wert definiert werden kann.

Rückgabewerte

RC — Die Funktion gibt im Erfolgsfall `0` zurück, andernfalls einen Fehlercode (Anhang A, *Rückgabewerte aus den API-Funktionen*).

Handle — Zugriffs-Handle, falls die Verbindung zum Device erfolgreich hergestellt wurde.

Beschreibung

Baut eine Netzwerkverbindung über TCP/IP zu einem AnaGate I2C (bzw. AnaGate Universal Programmer) auf. Erst nach dem erfolgreichen Verbinden mit dem Gerät ist ein Zugriff auf den I2C Bus möglich.

Anmerkung

Das AnaGate I2C (bzw. die I2C-Schnittstelle eines AnaGate Universal Programmers) erlaubt nur eine einzige Netzwerkverbindung. Solange

 © 2007-2015 Analytica GmbH

eine bestehende Verbindung aufrechterhalten wird, wird jeder neuer Verbindungsversuch abgelehnt.

Im folgenden ein Programmier-Beispiel für den initialen Zugriff auf das Gerät.

```
local nRC, nHandle = LS_I2COpenDevice(1000000, "192.168.1.254", 5000)
if nRC ~= 0 then
   print(LS_I2CErrorMessage(nRC))
   os.exit()
end

-- now do something

LS_I2CCloseDevice(nHandle)
```

Siehe auch

LS_I2CCloseDevice

© 2007-2015 Analytica GmbH

LS_I2COpenDeviceEx

LS_I2COpenDeviceEx — Baut eine Netzwerkverbindung zu einem AnaGate I2C (bzw. AnaGate Universal Programmer) auf. Dabei muss zusätzlich zur LS_I2COpenDevice-Funktion noch der Port mit angegeben werden.

Syntax

```
int RC, int Handle = LS_I2COpenDeviceEx(uint32 Baudrate, uint8 DevicePort, string IPAddress, int Timeout);
```

Parameter

Baudrate — Baudrate, mit der der I2C-Bus betrieben werden soll. Die Werte können individuell eingestellt werden, z.B.

- 100000 für 100 kBit (Standard Mode)
- 400000 für 400 kBit (Fast Mode)

Anmerkung

Größere Werte als 400 kBit werden auf dem AnaGate SPI ignoriert.

DevicePort — Nummer des I2C-Bus (beginnt mit 0)

IPAddress — Netzwerkadresse des AnaGate-Partners.

Timeout — Standard-Timeout für AnaGate-Zugriffe in Millisekunden.

Ein Timeout wird festgestellt, wenn die AnaGate-Hardware nicht innerhalb der vereinbarten Timeout-Zeit antwortet. Diese Timeout-Zeit gilt auf der aktiven Netzwerkverbindung für alle Kommandos bzw. Funktionen, für die kein spezifischer Timeout-Wert definiert werden kann.

Rückgabewerte

RC — Die Funktion gibt im Erfolgsfall `0` zurück, andernfalls einen Fehlercode (Anhang A, *Rückgabewerte aus den API-Funktionen*).

Handle — Zugriffs-Handle, falls die Verbindung zum Device erfolgreich hergestellt wurde.

Beschreibung

Baut eine Netzwerkverbindung über TCP/IP zu einem AnaGate I2C (bzw. AnaGate Universal Programmer) auf. Erst nach dem erfolgreichen Verbinden mit dem Gerät ist ein Zugriff auf den I2C Bus möglich.

 © 2007-2015 Analytica GmbH

Anmerkung

Das AnaGate I2C (bzw. die I2C-Schnittstelle eines AnaGate Universal Programmers) erlaubt nur eine einzige Netzwerkverbindung. Solange eine bestehende Verbindung aufrechterhalten wird, wird jeder neuer Verbindungsversuch abgelehnt.

Im folgenden ein Programmier-Beispiel für den initialen Zugriff auf das Gerät.

```
local nRC, nHandle = LS_I2COpenDeviceEx(1000000, 0, "192.168.1.254", 5000)
if nRC ~= 0 then
   print(LS_I2CErrorMessage(nRC))
   os.exit()
end

-- now do something

LS_I2CCloseDevice(nHandle)
```

Siehe auch

LS_I2CCloseDevice

© 2007-2015 Analytica GmbH

LS_I2CCloseDevice

LS_I2CCloseDevice — Schließt eine geöffnete Netzwerk-Verbindung zu einem AnaGate I2C Device.

Syntax

```
int RC = LS_I2CCloseDevice(int Handle);
```

Parameter

Handle Gültiges Zugriffs-Handle eines erfolgreichen Aufrufs von `LS_I2COpenDevice`.

Rückgabewert

RC Die Funktion gibt im Erfolgsfall `0` zurück, andernfalls einen Fehlercode (Anhang A, *Rückgabewerte aus den API-Funktionen*).

Beschreibung

Schließt eine geöffnete Netzwerk-Verbindung zu einem AnaGate I2C Device. Das übergebene Handle *`Handle`* ist ein Rückgabewert aus einem vorangegangenen erfolgreichen Aufruf der Funktion `LS_I2COpenDevice`.

Wichtig

Das explizite Schließen der Verbindung ist notwendig, um die in der DLL angelegten Systemressourcen wieder freizugeben und dem verbundenen Gerät mitzuteilen, dass die Verbindung nicht mehr genutzt wird und wieder für neue Verbindungsanfragen zur Verfügung gestellt werden soll.

Siehe auch

LS_I2COpenDevice

 © 2007-2015 Analytica GmbH

LS_I2CReset

LS_I2CReset — Setzt den I2C-Controller auf dem AnaGate I2C Device zurück.

Syntax

```
int RC = LS_I2CReset(int Handle);
```

Parameter

Handle Gültiges Zugriffs-Handle eines erfolgreichen Aufrufs von `LS_I2COpenDevice`.

Rückgabewert

RC Die Funktion gibt im Erfolgsfall `0` zurück, andernfalls einen Fehlercode (Anhang A, *Rückgabewerte aus den API-Funktionen*).

Beschreibung

Setzt den I2C-Controller auf dem AnaGate I2C Device zurück.

© 2007-2015 Analytica GmbH

LS_I2CRead

LS_I2CRead — Liest Daten von einem I2C-Partner.

Syntax

```
int RC, table(uint8[BufferLen]) Data = LS_I2CRead(int Handle, uint16 SlaveAddress, uint16 BufferLen);
```

Parameter

Handle	Gültiges Zugriffs-Handle eines erfolgreichen Aufrufs von `LS_I2COpenDevice`.
SlaveAddress	Slave-Adresse des I2C-Partners. Die Slave-Address kann eine sog. 7 oder 10-Bit Adresse darstellen (siehe Anhang B, *Adressierung auf dem I2C-Bus*).
BufferLen	Anzahl der Bytes, die gelesen werden sollen.

Rückgabewerte

RC	Die Funktion gibt im Erfolgsfall `0` zurück, andernfalls einen Fehlercode (Anhang A, *Rückgabewerte aus den API-Funktionen*).
Data	Byte-Puffer, der die vom I2C Partner empfangenen Daten enthält.

Beschreibung

Liest Daten von einem I2C-Partner. Der Anwender muss einen korrekten Aufbau der Adresse des I2C Partners sicherstellen.

Das R/W-Bit der Slave-Adresse muss vom Anwender nicht explizit gesetzt werden.

Siehe auch

LS_I2CWrite

© 2007-2015 Analytica GmbH

LS_I2CWrite

LS_I2CWrite — Schreibt Daten zu einem I2C-Partner.

Syntax

```
int RC, uint16 ErrorByte = LS_I2CWrite(int Handle, uint16 nSlaveAddress,
table(uint8[BufferLen]) Buffer, uint16 BufferLen);
```

Parameter

Handle	Gültiges Zugriffs-Handle eines erfolgreichen Aufrufs von `I2COpenDevice`.
SlaveAddress	Slave-Adresse des I2C Partners. Die Slave-Address kann eine sog. 7 oder 10-Bit Adresse darstellen (siehe Anhang B, *Adressierung auf dem I2C-Bus*).
Buffer	Byte-Puffer mit den Daten, die an den I2C-Partner gesendet werden sollen.
BufferLen	Anzahl von Datenbytes, die gelesen werden sollen.

Rückgabewert

RC	Die Funktion gibt im Erfolgsfall `0` zurück, andernfalls einen Fehlercode (Anhang A, *Rückgabewerte aus den API-Funktionen*).
ErrorByte	Index des Datenbytes im Sendepuffer, bei dem ein Schreib-spezifischer Fehler aufgetreten ist.

Beschreibung

Schreibt Daten zu einem I2C-Partner. Der Anwender muss einen korrekten Aufbau des Datenpuffers und der Adresse des I2C Partners sicherstellen.

Das R/W-Bit der Slave-Adresse muss vom Anwender nicht explizit gesetzt werden.

Siehe auch

LS_I2CRead

© 2007-2015 Analytica GmbH

LS_I2CSequence

LS_I2CSequence — Schreibt eine beliebige Folge von I2C-Schreib- und Lese-Kommandos als Sequenz zu einem I2C-Partner.

Syntax

`int RC, uint16 NumberOfBytesRead, uint16 ByteNumberLastError, table(uint8[NumberOfBytesToRead]) ReadBuffer = **LS_I2CSequence**(int *Handle*, table(uint8[NumberOfBytesToWrite]) *WriteBuffer*, uint16 *NumberOfBytesToWrite*, uint16 *NumberOfBytesToRead*);`

Parameter

Handle — Gültiges Zugriffs-Handle eines erfolgreichen Aufrufs von `LS_I2COpenDevice`.

WriteBuffer — Bytepuffer, der die Kommandos enthält, die zum AnaGate I2C gesendet werden. Die einzelnen Kommandos werden hintereinander in diesen Puffer abgelegt.

Der Aufbau eines Lesekommandos ist dabei wie folgt definiert:

Aufbau eines Lesekommandos für LS_I2CSequence

Lesekommando	**Beschreibung**
2 Bytes (LSB,MSB)	Slave Adresse im 7- bzw. 10-Bit-Format, wobei das R/W Bit explizit auf 1 gesetzt werden muss.
2 Bytes (LSB,MSB)	Bit 0-14: Anzahl der Datenbytes, die vom I2C-Partner gelesen werden sollen. Die erfolgreich gelesenen Daten werden im Bytepuffer *`ReadBuffer`* zurückgeliefert. Bit 15: Ist dieses Bit gesetzt, wird am Ende des Lesekommandos kein Stop-Bit an den Slave gesendet.

Der Aufbau eines Schreibkommandos ist dabei wie folgt definiert:

Aufbau eines Schreibkommandos für LS_I2CSequence

Schreibkommando	**Beschreibung**
2 Bytes (LSB,MSB)	Slave Adresse im 7- bzw. 10-Bit-Format, wobei das R/W Bit explizit auf 0 gesetzt werden muss.
2 Bytes (LSB,MSB)	Bit 0-14: Anzahl der folgenden Datenbytes, die zum I2C-Partner geschrieben werden sollen (N).

 © 2007-2015 Analytica GmbH

Schreibkommando	Beschreibung
	Bit 15: Ist dieses Bit gesetzt, wird am Ende des Schreibkommandos kein Stop-Bit an den Slave gesendet.
N Bytes	Datenbytes.

NumberOfBytesToWrite Länge des pcWriteBuffers

NumberOfBytesToRead Erwartete Gesamtlänge der empfangenen Daten.

Rückgabewerte

RC Die Funktion gibt im Erfolgsfall `0` zurück, andernfalls einen Fehlercode (Anhang A, *Rückgabewerte aus den API-Funktionen*).

NumberOfBytesRead Anzahl tatsächlich gelesener Bytes, die vom I2C-Partner empfangen wurden.

ByteNumberLastError Position des Bytes im *`WriteBuffer`*, bei dem ein Fehler aufgetreten ist.

ReadBuffer Bytepuffer, der die empfangenen Daten, die vom I2C-Partner gesendet werden, enthält. Hierbei werden die vom I2C-Partner empfangenen Daten hintereinander abgelegt (zuerst die Daten des ersten Lesekommandos, direkt danach die Daten des zweiten Lesekommandos, etc.)

Beschreibung

Der Anwender muss einen korrekten Aufbau des Datenpuffers und der Adresse des I2C Partners sicherstellen.

 © 2007-2015 Analytica GmbH

LS_I2CReadDigital

LS_I2CReadDigital — Liest die aktuellen Werte der digitalen Ein-/Ausgabe-Register der AnaGate-Hardware zurück.

Syntax

`int RC, uint32 InputBits, uint32 OutputBits = **LS_I2CReadDigital**(int *Handle*);`

Parameter

Handle Gültiges Zugriffs-Handle eines erfolgreichen Aufrufs von `LS_I2COpenDevice`.

Rückgabewert

RC Die Funktion gibt im Erfolgsfall `0` zurück, andernfalls einen Fehlercode (Anhang A, *Rückgabewerte aus den API-Funktionen*).

InputBits Aktueller Inhalt des Registers mit den digitalen Eingängen. Dabei entsprechen Bit 0 bis Bit 3 den vier digitalen Eingängen; Bit 4 bis Bit 31 sind auf 0 gesetzt.

OutputBits Aktueller Inhalt des Registers mit den digitalen Ausgängen. Dabei entsprechen Bit 0 bis Bit 3 den vier digitalen Ausgängen; Bit 4 bis Bit 31 sind auf 0 gesetzt.

Beschreibung

Alle Geräte der AnaGate-Serie (außer der Gerätevariante AnaGate CAN uno im Hutschienengehäuse) besitzen auf der Gehäuserückseite Anschlüsse für jeweils 4 digitale Eingänge und 4 digitale Ausgänge. Das AnaGate CAN uno im Hutschienengehäuse besitzt dagegen an der Oberseite des Gehäuses Anschlüsse für jeweils 2 digitale Ein- und Ausgänge.

Die aktuellen Zustände der digitalen Eingänge und Ausgänge können mit der Funktion `LS_I2CReadDigital` ermittelt werden.

Im folgenden ein Programmier-Beispiel, das die digitalen IOs setzt und zurück liest.

```
local nOutputs = 0x03

local nRC, hHandle = LS_I2COpenDevice(400000, "192.168.1.254", 5000)
if nRC == 0 then
   // set the digital output register (PIN 0 and PIN 1 to HIGH value)
   nRC = LS_I2CWriteDigital(hHandle, nOutputs)
   local nInputs
   // read all input and output registers
   nRC, nInputs, nOutputs = LS_I2CReadDigital(hHandle)

   LS_I2CCloseDevice(hHandle)
end
```

 © 2007-2015 Analytica GmbH

Siehe auch

LS_I2CWriteDigital

© 2007-2015 Analytica GmbH

LS_I2CWriteDigital

LS_I2CWriteDigital — Setzt das digitale Ausgabe-Register der AnaGate-Hardware auf einen neuen Wert.

Syntax

```
int RC = LS_I2CWriteDigital(int Handle, uint32 OutputBits);
```

Parameter

Handle	Gültiges Zugriffs-Handle eines erfolgreichen Aufrufs von `LS_I2COpenDevice`.
OutputBits	Neuer Wert des Registers mit den digitalen Ausgängen. Dabei entsprechen Bit 0 bis Bit 3 den vier digitalen Ausgängen; Bit 4 bis Bit 31 sind reserviert und auf 0 zu setzen.

Rückgabewert

RC Die Funktion gibt im Erfolgsfall `0` zurück, andernfalls einen Fehlercode (Anhang A, *Rückgabewerte aus den API-Funktionen*).

Beschreibung

Alle Geräte der AnaGate-Serie (außer der Gerätevariante AnaGate CAN uno im Hutschienengehäuse) besitzen auf der Gehäuserückseite Anschlüsse für jeweils 4 digitale Eingänge und 4 digitale Ausgänge. Das AnaGate CAN uno im Hutschienengehäuse besitzt dagegen an der Oberseite des Gehäuses Anschlüsse für jeweils 2 digitale Ein- und Ausgänge.

Die digitalen Ausgänge können mit der Funktion `LS_I2CWriteDigital` verändert werden.

Ein Programmier-Beispiel zum Lesen/Schreiben der IOs ist bei der Beschreibung von `LS_I2CReadDigital` zu finden.

Siehe auch

LS_I2CReadDigital

 © 2007-2015 Analytica GmbH

LS_I2CErrorMessage

LS_I2CErrorMessage — Gibt eine textuelle Beschreibung eines Rückgabewertes aus einer API-Funktion zurück.

Syntax

```
string ErrorMsg = LS_I2CErrorMessage(int ErrorCode);
```

Parameter

ErrorCode Fehlercode, dessen Fehlerbeschreibung ermittelt werden soll.

Rückgabewert

ErrorMsg Textuelle Beschreibung des Fehlercodes (in englischer Sprache).

Beschreibung

Gibt eine textuelle Beschreibung des übergebenen Rückgabewertes zurück (siehe auch Anhang A, *Rückgabewerte aus den API-Funktionen*). Alle Fehlertexte sind in englischer Sprache.

Im folgenden ein Programmier-Beispiel in Lua.

```
local nRC = 0
local sErrorText = 'No Error'

//... call a API function here

sErrorText = LS_I2CErrorMessage(nRC)
print(sErrorText)
```

 © 2007-2015 Analytica GmbH

LS_I2CReadEEProm

LS_I2CReadEEProm — Liest Daten von einem EEPROM am I2C-Bus.

Syntax

```
int RC, table(uint8[DataLen]) Data = LS_I2CReadEEProm(int Handle, uint16
SubAddress, uint32 Offset, uint16 OffsetFormat, uint16 DataLen);
```

Parameter

Handle — Gültiges Zugriffs-Handle eines erfolgreichen Aufrufs von `I2COpenDevice`.

SubAddress — Subadresse des EEPROMs, mit dem kommuniziert werden soll. Die gültigen Werte für die *`SubAddress`* werden von der Einstellung des Parameters *`OffsetFormat`* (Bit 8-10) beeinflusst. Werden von dem EEPROM-Typ Bits der *Chip Enable Address* zur Adressierung des internen Speichers verwendet, so können nur noch die frei verfügbaren Bits als Steuerpins für die Ansteuerung des Bausteins auf der Platine genutzt werden.

- kein Bit für die Adressierung verwendet: 0 bis 7
- 1 Bit für die Adressierung verwendet: 0 bis 3
- 2 Bits für die Adressierung verwendet: 0 bis 1
- 3 Bits für die Adressierung verwendet: 0

Offset — Daten-Offset auf dem EEPROM, ab dem Daten gelesen werden sollen.

OffsetFormat — Dieser Parameter ist als Bitfeld definiert und gibt an, wie eine Speicheradresse auf dem EEPROM bei Schreib- und Lesezugriffen abgebildet wird.

Die Bits 0-7 geben an, wie viele Bits im Adressbyte (bzw. Adresswort) für die Adressierung verwendet werden.

Die Bits 8-10 geben an, wie viele und welche der *Chip Enable Bits* zusätzlich zur Adressierung des EEPROM-Speichers verwendet werden (Tabelle C.1, „ Verwendung der Chip-Enable-Bits bei I2C-EEPROMs “).

Anmerkung

Die maximal adressierbare Speichergröße eines EEPROMs kann aus der Summe aller Adressbits berechnet werden. So benötigt z.B. ein M24C08 acht Bits des Adressbytes und 1 zusätzliches Bit. Damit können über die 9 Bits insgesamt 512 Bytes adressiert werden.

DataLen — Länge des Datenpuffers.

 © 2007-2015 Analytica GmbH

Rückgabewerte

RC Die Funktion gibt im Erfolgsfall `0` zurück, andernfalls einen Fehlercode (Anhang A, *Rückgabewerte aus den API-Funktionen*).

Data Enthält die vom EEPROM gelesenen Daten.

Beschreibung

Die Funktion `LS_I2CReadEEProm` liest Daten von einem seriellen I2C-EEPROM.

Der Zugriff auf eine Speicheradresse auf einem I2C-EEPROM wird prinzipiell über eine normale Schreib- bzw. Leseanforderung auf dem I2C-Bus realisiert. Somit müssen bei einem Lesezugriff nur die passende Slave-Adresse, die Offset-Adresse auf dem Chip und die eigentlichen Daten gesendet werden.

Die Funktion `LS_I2CReadEEProm` setzt die übergebene Speicheradresse auf dem Chip anhand der Angabe von der Subadresse und der Adressierungsvariante des vorliegenden EEPROM-Typs korrekt um. Eine Angabe der Slave-Adresse entfällt, da diese bereits durch die vorhandenen Parameter festgelegt ist.

Ein Programmier-Beispiel, das ein EEPROM vom Typ **ST24C1024** komplett löscht, ist bei der Beschreibung von LS_I2WriteEEPROM zu finden.

Siehe auch

LS_I2CWriteEEProm

Anhang C, *Programmierung von I2C-EEPROM*

 © 2007-2015 Analytica GmbH

LS_I2CWriteEEProm

LS_I2CWriteEEProm — Beschreibt ein serielles EEPROM am I2C-Bus.

Syntax

```
int RC = LS_I2CWriteEEProm(int Handle, uint16 SubAddress, uint32 Offset,
table(uint8[DataLen]) Data, uint16 DataLen, uint16 nOffsetFormat);
```

Parameter

Handle — Gültiges Zugriffs-Handle eines erfolgreichen Aufrufs von `I2COpenDevice`.

SubAddress — Subadresse des EEPROMs, mit dem kommuniziert werden soll. Die gültigen Werte für die *SubAddress* werden von der Einstellung des Parameters *OffsetFormat* (Bit 8-10) beeinflusst. Werden von dem EEPROM-Typ Bits der *Chip Enable Address* zur Adressierung des internen Speichers verwendet, so können nur noch die frei verfügbaren Bits als Steuerpins für die Ansteuerung des Bausteins auf der Platine genutzt werden.

- kein Bit für die Adressierung verwendet: 0 bis 7
- 1 Bit für die Adressierung verwendet: 0 bis 3
- 2 Bits für die Adressierung verwendet: 0 bis 1
- 3 Bits für die Adressierung verwendet: 0

Offset — Daten-Offset auf dem EEPROM, ab dem die übergebenen Daten geschrieben werden soll.

Data — Byte-Tabelle mit den Daten, die geschrieben werden sollen.

DataLen — Länge des Datenpuffers.

OffsetFormat — Dieser Parameter ist als Bitfeld definiert und gibt an, wie eine Speicheradresse auf dem EEPROM bei Schreib- und Lesezugriffen abgebildet wird.

Die Bits 0-7 geben an, wie viele Bits im Adressbyte (bzw. Adresswort) für die Adressierung verwendet werden.

Die Bits 8-10 geben an, wie viele und welche der *Chip Enable Bits* zusätzlich zur Adressierung des EEPROM-Speichers verwendet werden (Tabelle C.1, „ Verwendung der Chip-Enable-Bits bei I2C-EEPROMs “).

Anmerkung

Die maximal adressierbare Speichergröße eines EEPROMs kann aus der Summe aller Adressbits berechnet werden. So benötigt z.B. ein M24C08 acht Bits

 © 2007-2015 Analytica GmbH

des Adressbytes und 1 zusätzliches Bit. Damit können über die 9 Bits insgesamt 512 Bytes adressiert werden.

Rückgabewert

RC Die Funktion gibt im Erfolgsfall `0` zurück, andernfalls einen Fehlercode (Anhang A, *Rückgabewerte aus den API-Funktionen*).

Beschreibung

Die Funktion `LS_I2CWriteEEProm` schreibt Daten auf ein serielles I2C-EEPROM.

Der Zugriff auf eine Speicheradresse auf einem I2C-EEPROM wird prinzipiell über eine normale Schreib- bzw. Leseanforderung auf dem I2C-Bus realisiert. Somit müssen bei einem Schreibzugriff nur die passende Slave-Adresse, die Offset-Adresse auf dem Chip und die eigentlichen Daten gesendet werden.

Die Funktion `LS_I2CWriteEEProm` setzt die übergebene Speicheradresse auf dem Chip anhand der Angabe von der Subadresse und der Adressierungsvariante des vorliegenden EEPROM-Typs korrekt um. Eine Angabe der Slave-Adresse entfällt, da diese bereits durch die vorhandenen Parameter festgelegt ist.

Tipp

Beim Programmieren von EEPROMs ist zu beachten, dass der EEPROM-Speicher in sogenannte Pages unterteilt ist und dass mit einem einzelnen Schreibbefehl nur innerhalb einer Speicherseite programmiert werden kann. Benutzer von `LS_I2CWriteEEProm` müssen selbst sicherstellen, dass sie nicht über Seitengrenzen hinaus schreiben. Die jeweilige Größe einer Speicherseite ist abhängig vom EEPROM-Typ.

Im folgenden ein Programmier-Beispiel, das ein EEPROM vom Typ **ST24C1024** komplett löscht.

```
local tabData = {}
for i = 1, 256, 1 do
   table.insert(tabData, 0x0)
end

local nSubAddress = 0 1
local nOffsetFormat = 0x10+0x0F 2

local RC, hHandle = LS_I2COpenDevice(400000, "192.168.1.254", 5000)
if RC == 0 then
   for page = 0, 512-1, 1 do
      RC = LS_I2CWriteEEProm(hHandle, nSubAddress, i*256, tabData, #tabData, nOffSetFormat) 3
   end
   LS_I2CCloseDevice(hHandle)
end
```

1. Es können bis zu 4 ST24C1024 am I2C-Bus verdrahtet werden. Über die Angabe der Subadresse `0` liegen die Steuerpins E2 und E1 auf LOW.
2. Der ST24C1024 benötigt 17 Adressbits zur Adressierung der 128 kB. 16 Bits werden über die Adressangabe des Schreibbefehls festgelegt: `16=0x0F`. Das Adressbit A16 wird über das E0 der *Chip Enable Address* festgelegt, deshalb ist der Mode `1` (E2-E1-A0) zu verwenden: `0x10`.
3. Die Seitengröße eines ST24C1024 beträgt 256 Byte, im vorliegenden Fall werden alle Seiten komplett über eine for-Schleife programmiert.

 © 2007-2015 Analytica GmbH

Siehe auch

LS_I2CReadEEProm

Anhang C, *Programmierung von I2C-EEPROM*

© 2007-2015 Analytica GmbH

Kapitel 16. Lua-Programmier-Beispiele

16.1. Beispiele für Geräte mit CAN-Schnittstelle

In diesem Beispiel wird eine Verbindung zu zwei AnaGate CAN-Devices aufgebaut, die über den CAN-Bus miteinander verbunden sind. Sollte die Verbindung fehlschlagen, wird eine Fehlermeldung ausgegeben und das Script beendet. Bei erfolgreicher Verbindung werden die globalen Einstellungen des AnaGate CAN-Device gesetzt.

- Verbindung zu den zwei AnaGate CAN Devices aufbauen.
- Filtereinstellungen setzen.
- aktuelle Uhrzeit setzen.
- Globals setzen.
- 3 Datenpakete auf 1. AnaGate CAN Device versenden.
- Die Datenpakete aus dem internen Puffer des 2. AnaGate CAN Device lesen.
- Überprüfen, ob Endlosschleife verlassen werden soll.

```
-- Filter: alle CAN-Identifier akzeptieren
local aFilter = { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
                  0x00, 0x1FFFFFFF, 0x00, 0x1FFFFFFF,
                  0x00, 0x1FFFFFFF, 0x00, 0x1FFFFFFF }

local aSendData = { 0xF1, 0xF2, 0xF3, 0xF4, 0xF5, 0xF6, 0xF7, 0xF8 }
--**************************************************
function main()
  local nRC, nHandle, nHandle2, oTime
  -- Open
  nRC, nHandle = LS_CANOpenDevice(false, true, 0, "10.1.2.160", 5000)
  if nRC ~= 0 then
    print(LS_CANErrorMessage(nRC))
    os.exit()
  end

  nRC, nHandle2 = LS_CANOpenDevice(false, true, 0, "10.1.2.161", 5000)
  if nRC ~= 0 then
    print(LS_CANErrorMessage(nRC))
    os.exit()
  end

  -- Filter setzen
  nRC = LS_CANSetFilter(nHandle,  aFilter)
  nRC = LS_CANSetFilter(nHandle2, aFilter)

  -- aktuelle Zeit auf den AnaGate Device setzen
  nRC, oTime = LS_GetTime()
  nRC = LS_CANSetTime(nHandle,  oTime[1], oTime[2])
  nRC = LS_CANSetTime(nHandle2, oTime[1], oTime[2])

  -- Globals setzen
  nRC = LS_CANSetGlobals(nHandle,  500000, 0, true, false, false)
  nRC = LS_CANSetGlobals(nHandle2, 500000, 0, true, false, false)

  -- Endlosschleife
  repeat
```


© 2007-2015 Analytica GmbH

```
    -- 1 Datenpakete auf dem 1. AnaGate CAN Device versenden
    nRC = LS_CANWrite(nHandle, 1, aSendData, #aSendData, 0)

    LS_Sleep(20) -- 20Millisekunden warten

    -- Datenpaket auf 2. AnaGate CAN Device empfangen
    local nAvail, ID, Len, Data, Flags, Sec, Microsec = LS_CANGetMessage(nHandle2, 10)
    while nAvail>=0 do
      nAvail, ID, Len, Data, Flags, Sec, Microsec = LS_CANGetMessage(nHandle2, 10)
    end
  until false

  -- Verbindungen beenden
  LS_CANCloseDevice(nHandle)
  LS_CANCloseDevice(nHandle2)
end
```

Beispiel 16.1. CAN-Lua-Scriptbeispiel

16.2. Beispiele für Geräte mit SPI-Schnittstelle

In diesem Beispiel wird eine Verbindung zu einem AnaGate SPI-Device aufgebaut. Sollte die Verbindung fehlschlagen, wird eine Fehlermeldung ausgegeben und das Script beendet. Bei erfolgreicher Verbindung werden die globalen Einstellungen des AnaGate SPI-Device gesetzt. Danach werden 4 DataRequests an den SPI-Partner gesendet.

- Setzen des WriteEnable-Flags des SPI-Partners.
- Abfragen des Status-Registers.
- Lesen von 20 Bytes ab Adresse 0x00.

```
--*****************************************************
local function printf(...)
  io.write(string.format(...))
  io.flush()
end
--*****************************************************
function main()
  -- Verbindung zu AnaGate SPI-Device herstellen
  local nRC, nHandle = LS_SPIOpenDevice("10.1.2.162", 5000)
  if nRC ~= 0 then
    local errortext = LS_SPIErrorMessage(nRC)
    print(errortext)
    os.exit()
  end

  -- Setzen der globalen Einstellungen
  nRC = LS_SPISetGlobals(nHandle, 100000, 2, 0, 0)

  -- OP-Codes des SPI-Partners mit Daten
  local OPWriteEnab = {0x06}
  local OPStatusReg = {0x05, 0x00}
  local OPRead      = {0x03, 0x00, 0x00, 0x00}
  local Value
  -- WriteEnable-Flag des SPI-Partners setzen
  nRC, Value = LS_SPIDataReq(nHandle, OPWriteEnab, #OPWriteEnab, 1)
```

 © 2007-2015 Analytica GmbH

```
  -- Statusregister des SPI-Partners abfragen
  nRC, Value = LS_SPIDataReq(nHandle, OPStatusReg, #OPStatusReg, 2)
  for i, v in ipairs(Value) do
    printf("Data Status: %02X\n", v)
  end

  -- Lesen von 20 Bytes ab Adresse 0x00
  nRC, Value = LS_SPIDataReq(nHandle, OPRead, #OPRead, 20)
  for i, v in ipairs(Value) do
    printf("Data Status: %02X\n", v)
  end

  -- Alle digitalen Ausgaenge zuruecksetzen
  LS_SPIWriteDigital(nHandle, 0)

  -- Verbindung zu AnaGate SPI-Device beenden
  LS_SPICloseDevice(nHandle)
end
```

Beispiel 16.2. SPI-Lua-Scriptbeispiel

16.3. Beispiele für Geräte mit I2C-Schnittstelle

In diesem Beispiel wird eine Verbindung zu einem AnaGate I2C-Device aufgebaut. Sollte die Verbindung fehlschlagen, wird eine Fehlermeldung ausgegeben und das Script beendet. Bei erfolgreicher Verbindung werden die folgenden Schritte ausgeführt.

1. Beispiel

- 64 * 1024 Bytes aus dem EEPROM lesen
- 10 * 128 Bytes auf das EEPROM schreiben

```
--**************************************************
local function printf(...)
  io.write(string.format(...))
  io.flush()
end
--**************************************************
function main()
  local aSendData = {}
  for i = 1, 128, 1 do
    table.insert(aSendData, i-1)
  end

  local nRC, nHandle = LS_I2COpenDevice(1000000, "10.1.2.162", 5000)
  if nRC ~= 0 then
    print(LS_I2CErrorMessage(nRC))
    os.exit()
  end

  --Read EEPROM
  local CountBytes = 1024
  for Address = 0, CountBytes*64, CountBytes do
    local Value
    nRC, Value = LS_I2CReadEEProm(nHandle, 1, Address, 16, CountBytes)
```

 © 2007-2015 Analytica GmbH

```
    for i, v in ipairs(Value) do
      printf("%02X ", v)
      if i % 16 == 0 then
        printf("\n")
      end
    end
  end

  --Write EEPROM
  CountBytes = #aSendData
  for Address = 0, CountBytes * 10, CountBytes do
    nRC = LS_I2CWriteEEProm(nHandle, 1, Address, aSendData, CountBytes, 16)
  end

  LS_I2CWriteDigital(nHandle, 0)
  LS_I2CCloseDevice(nHandle)
end
```

Beispiel 16.3. I2C-Lua Scriptbeispiel EEPROM-Funktionen

2. Beispiel

- 2 Bytes an einen I2C-Partner zum schreiben senden
- 1024 Bytes aus einem I2C-Partner lesen

```
--**************************************************
local function printf(...)
  io.write(string.format(...))
  io.flush()
end
--**************************************************
function main()
  local aSendData = {}
  for i = 1, 128, 1 do
    table.insert(aSendData, i-1)
  end

  local nRC, nHandle = LS_I2COpenDevice(1000000, "10.1.2.162", 5000)
  if nRC ~= 0 then
    print(LS_I2CErrorMessage(nRC))
    os.exit()
  end

  --Write
  local aData = {0x00, 0x05} -- ab Adresse 5 lesen
  local Value
  nRC, Value = LS_I2CWrite(nHandle, 0xa2, aData, #aData)

  --Read
  nRC, Value = LS_I2CRead(nHandle, 0xa2, 1024)
  for i, v in ipairs(Value) do
    printf("%02X ", v)
    if i % 16 == 0 then
      printf("\n")
    end
  end
  printf("\n")
```

 © 2007-2015 Analytica GmbH

```
  LS_I2CWriteDigital(nHandle, 0)
  LS_I2CCloseDevice(nHandle)
  end
```

Beispiel 16.4. I2C - Lua Scriptbeispiel I2C-Direkt

3. Beispiel

- Sequenz an Anagate I2C Device senden

```
--**************************************************
local function printf(...)
  io.write(string.format(...))
  io.flush()
end
--**************************************************
function main()
  local aSendData = {}
  for i = 1, 128, 1 do
    table.insert(aSendData, i-1)
  end

  local nRC, nHandle = LS_I2COpenDevice(1000000, "10.1.2.162", 5000)
  if nRC ~= 0 then
    print(LS_I2CErrorMessage(nRC))
    os.exit()
  end

  --Sequence
  local aData = {0xa2, 0x00,  --SLA
                 0x02, 0x00,  --Laenge Schreibkommando
                 0x00, 0x00,  --Daten Schreibkommando
                 0xa3, 0x00,  --SLA 1. Lesekommando
                 0x30, 0x00,  --Laenge 1. Lesekommando
                 0xa3, 0x00,  --SLA 2. Lesekommando
                 0x20, 0x00}  --Laenge 2. Lesekommando
  local CountRead, LastError, Value
  nRC, CountRead, LastError, Value = LS_I2CSequence(nHandle, aData, #aData, 0x0050)
  printf("CountRead:%02X  LastError:%02X\n", CountRead, LastError)
  for i, v in ipairs(Value) do
    printf("%02X ", v)
  end
  printf("\n")

  LS_I2CWriteDigital(nHandle, 0)
  LS_I2CCloseDevice(nHandle)
end
```

Beispiel 16.5. I2C-Lua-Scriptbeispiel Sequence

 © 2007-2015 Analytica GmbH

Anhang A. Rückgabewerte aus den API-Funktionen

Im folgenden eine Liste mit den Rückgabewerten der API-Funktionen. Die Werte sind in der Header-Datei `AnaGateErrors.h` definiert.

Wert	Name	Beschreibung
0	ERR_NONE	Kein Fehler aufgetreten.
0x000001	ERR_OPEN_MAX_CONN	Open fehlgeschlagen, maximale Anzahl von Verbindungen erreicht.
0x0000FF	ERR_OP_CMD_FAILED	Kommando mit unbekanntem Fehler beendet.
0x020000	ERR_TCPIP_SOCKET	Socket-Fehler auf TCP/IP-Ebene aufgetreten.
0x030000	ERR_TCPIP_NOTCONNECTED	Verbindung zum TCP/IP-Partner konnte nicht aufgebaut werden bzw. ist unterbrochen.
0x040000	ERR_TCPIP_TIMEOUT	Der TCP/IP-Partner antwortete nicht innerhalb des definierten Timeouts.
0x050000	ERR_TCPIP_CALLNOTALLOWED	Das Kommando ist zu diesem Zeitpunkt nicht erlaubt.
0x060000	ERR_TCPIP_NOT_INITIALIZED	TCP/IP-Stack konnte nicht initialisiert werden.
0x0A0000	ERR_INVALID_CRC	AnaGate TCP/IP-Telegramm hat fehlerhafte Checksumme (CRC).
0x0B0000	ERR_INVALID_CONF	AnaGate TCP/IP-Telegramm wurde vom Partner nicht quittiert.
0x0C0000	ERR_INVALID_CONF_DATA	AnaGate TCP/IP-Telegrammbestätigung ist ungültig.
0x900000	ERR_INVALID_DEVICE_HANDLE	Ungültiges Zugriffs-Handle.
0x910000	ERR_INVALID_DEVICE_TYPE	Funktion kann über das Zugriffs-Handle nicht ausgeführt werden, da sie einem anderen Gerätetyp der AnaGate-Serie zugeordnet ist.

Tabelle A.1. Allgemeine Rückgabewerte für alle Geräte der AnaGate-Serie

Wert	Name	Beschreibung
0x000120	ERR_I2C_NACK	I2C-NACK
0x000121	ERR_I2C_TIMEOUT	I2C-Timeout

© 2007-2015 Analytica GmbH

Wert	Name	Beschreibung
0x000125	ERR_I2C_POLL_TIMEOUT	Acknowledge-Polling-Timeout nach Schreibzugriff

Tabelle A.2. Rückgabewerte für AnaGate I2C

Eine textuelle Beschreibung des Rückgabewertes kann mit der Funktion `I2CErrorMessage()` ermittelt werden. Die Beschreibung ist in englischer Sprache.

Wert	Name	Beschreibung
0x000220	ERR_CAN_NACK	CAN-NACK
0x000221	ERR_CAN_TX_ERROR	CAN Transmit Error
0x000222	ERR_CAN_TX_BUF_OVERLOW	CAN buffer overflow
0x000223	ERR_CAN_TX_MLOA	CAN Lost Arbitration
0x000224	ERR_CAN_NO_VALID_BAUDRATE	CAN Setting no valid Baudrate

Tabelle A.3. Rückgabewerte für AnaGate CAN

Eine textuelle Beschreibung des Rückgabewertes kann mit der Funktion `CANErrorMessage()` ermittelt werden. Die Beschreibung ist in englischer Sprache.

Wert	Name	Beschreibung
0x000320	ERR_SPI_INCONSISTENT_TELEGRAM	SPI inkonsistente Telegrammdaten
0x000321	ERR_SPI_SEND_ERROR	SPI Fehler beim Senden
0x000330	ERR_SPI_SEQ_UNKNOWN_COMMAND	SPI Sequence unbekannter Befehlscode
0x000331	ERR_SPI_SEQ_TIMEOUT	SPI Sequence Timeout

Tabelle A.4. Rückgabewerte für AnaGate SPI

Eine textuelle Beschreibung des Rückgabewertes kann mit der Funktion `SPIErrorMessage()` ermittelt werden. Die Beschreibung ist in englischer Sprache.

Wert	Name	Beschreibung
0x000920	ERR_RENESAS_TIMEOUT	Renesas timeout
0x000921	ERR_RENESAS_INVALID_ID	Renesas Invalid ID
0x000922	ERR_RENESAS_FLASH_ERASE_FAILED	Renesas failed erase the flash
0x000923	ERR_RENESAS_PAGE_PROG_FAILED	Renesas failed prog the page

Tabelle A.5. Rückgabewerte für AnaGate Renesas

Eine textuelle Beschreibung des Rückgabewertes kann mit der Funktion `RenesasErrorMessage()` ermittelt werden. Die Beschreibung ist in englischer Sprache.

 © 2007-2015 Analytica GmbH

Wert	Name	Beschreibung
-1	ERR_SYNTAX	Syntax-Fehler
-2	ERR_RANGE	Wert außerhalb des gültigen Bereichs.
-3	ERR_NOT_A_NUMBER	Parameter ist nicht vom Typ *number*.
-4	ERR_NOT_A_STRING	Parameter ist nicht vom Typ *string*.
-5	ERR_NOT_A_BOOL	Parameter ist nicht vom Typ *boolean*.
-6	ERR_NOT_A_TABLE	Parameter ist nicht vom Typ *table*.
-10	ERR_NO_DATA	Keine Daten vorhanden.

Tabelle A.6. Rückgabewerte für Lua Scripting

 © 2007-2015 Analytica GmbH

Anhang B. Adressierung auf dem I2C-Bus

Eine Standard-I2C-Adresse ist das erste vom Master gesendete Byte, wobei die ersten sieben Bit die eigentliche Adresse darstellen und das achte Bit (R/W-Bit) die Lese- oder Schreibrichtung festlegt. I2C nutzt daher einen Adressraum von 7 Bit, was bis zu 112 Knoten auf einem Bus erlaubt (16 der 128 möglichen Adressen sind für Sonderzwecke reserviert).

Abbildung B.1. Definition einer I2C-Slaveadresse im 7-Bit-Format

Jeder I2C-fähige IC hat eine festgelegte Adresse. Die oberen 4 Bits der Busadresse werden als *Device Type Identifier* bezeichnet und legen im Regelfall den Chip-Typ fest. Die unteren drei Bits (Subadresse oder *Chip Enable Address* genannt) sind normalerweise über drei Steuerpins festgelegt. Es können also bis zu acht gleichartige ICs an einem I2C-Bus betrieben werden.

Wegen Adressknappheit wurde später eine 10-Bit-Adressierung eingeführt. Sie ist abwärtskompatibel zum 7-Bit-Standard durch Nutzung von 4 der 16 reservierten Adressen. Beide Adressierungsarten sind gleichzeitig verwendbar, was bis zu 1136 Knoten auf einem Bus erlaubt.

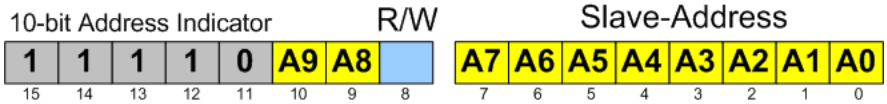


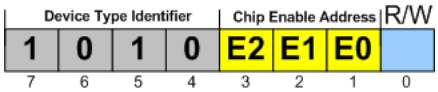
Abbildung B.2. Definition einer I2C-Slaveadresse im 10-Bit-Format

Anmerkung

Die Geräte vom Typ *AnaGate I2C* und *AnaGate Universal Programmer* unterstützen grundsätzlich beide Adressierungsarten. Über die API-Funktionen `I2CRead` und `I2CWrite` erfolgt die Adressierung jeweils über die Angabe einer 2-Byte Slave-Adresse.

Adressierung von seriellen EEPROM

Beispielsweise werden serielle EEPROM durch den Chip-Typ `0xA` festgelegt. Damit ergibt sich der folgende schematische Adressaufbau (die Chip Enable Bits werden meist mit E0, E1 und E2 bezeichnet):



© 2007-2015 Analytica GmbH

	Device Type Identifier				Chip Enable [1] [2]			R/W	EEPROM-Speicher
	b7	**b6**	**b5**	**b4**	**b3**	**b2**	**b1**	**b0**	
M24C01	1	0	1	0	E2	E1	E0	R/W	128 byte
M24C02	1	0	1	0	E2	E1	E0	R/W	256 byte
M24C04	1	0	1	0	E2	E1	A8	R/W	512 byte
M24C08	1	0	1	0	E2	A9	A8	R/W	1024 byte
M24C16	1	0	1	0	A10	A9	A8	R/W	2048 byte
M24C64	1	0	1	0	E2	E1	E0	R/W	8192 byte

[1] E0, E1 und E2 dienen der Ansteuerung des Speichermoduls über dessen externe Pins.
[2] A10, A9 und A8 werden als höchstwertige Bits der Speicheradresse interpretiert.

Tabelle B.1. Adressierungs-Beispiele von I2C-EEPROMs

© 2007-2015 Analytica GmbH

Anhang C. Programmierung von I2C-EEPROM

Das *AnaGate I2C* und der *AnaGate Universal Programmer* eignen sich u.a. auch für die Programmierung von seriellen I2C-EEPROM. Zur Unterstützung dieser speziellen Anforderung werden die beiden API-Funktionen `I2CReadEEProm` und `I2CWriteEEProm` zur Verfügung gestellt.

Wie alle I2C-fähigen Bausteine werden auch EEPROM über die Angabe einer Slave-Adresse auf dem I2C-Bus adressiert (siehe hierzu auch Anhang B, *Adressierung auf dem I2C-Bus*). Bei diesen Bausteintypen ist der sogenannte *Device Type Identifier* auf `0xA` festgelegt. Grundsätzlich können damit 8 gleichartige Bausteine angeschlossen werden und über die Angabe der *Chip Enable Bits* E0, E1 und E0 adressiert werden.

Device Type Identifier				Chip Enable Address			R/W
1	0	1	0	E2	E1	E0	
7	6	5	4	3	2	1	0

Der Beginn einer Übertragung wird mit dem **Start**-Signal vom Master angezeigt, dann folgt die Slave-Adresse. Diese wird durch das ACK-Bit vom entsprechenden Slave bestätigt. Abhängig vom R/W-Bit werden nun Daten Byte-weise geschrieben (Daten an Slave) oder gelesen (Daten vom Slave). Das **ACK** beim Schreiben wird vom Slave gesendet und beim Lesen vom Master. Das letzte Byte eines Lesezugriffs wird vom Master mit einem **NAK** quittiert, um das Ende der Übertragung anzuzeigen. Eine Übertragung wird durch das **Stop**-Signal beendet.

Nach der Übertragung der Slave-Adresse wird die Speicheradresse gesendet, ab der Daten auf dem Chip gelesen oder geschrieben werden sollen. Die Speicheradresse wird je nach EEPROM-Typ als einzelnes Byte (8 Bit) oder zwei Bytes (16 Bit, MSB zuerst) übertragen.

Um den Adressraum von 8 bzw. 16 Bit zu erweitern, werden bei diversen Bausteinen teilweise die *Chip Enable Bits* E0, E1, E2 zusätzlich zur Adressierung verwendet. Welche der Bits im Einzelfall als Adress-Bits verwendet werden, definiert der Chip-Hersteller. Im folgenden sind alle möglichen Variationen aufgelistet.

Modus [1]	**Verwendung**	**Beschreibung**
0x0	E2-E1-E0	Nur zur Auswahl des Chips, enthält keine Adressbits.
0x1	E2-E1-A0	Das E0-Bit wird für die Adressierung verwendet. Es entspricht dabei dem Adressbit A8 bzw. A16.
0x2	E2-A1-A0	Die Bits E0 und E1 werden für die Adressierung verwendet. E0 entspricht dabei dem Adressbit A8 bzw. A16 und E1 dem Adressbit A9 bzw. A17.
0x3	A2-A1-A0	E0, E1 und E2 werden für die Adressierung verwendet. E0 entspricht dabei dem Adressbit A8 bzw. A16, E1 dem Adressbit A9 bzw. A17 und E2 dem Adressbit A10 bzw. A18.
0x5	A0-E1-E0	Das E2-Bit wird für die Adressierung verwendet. Es entspricht dabei dem Adressbit A8 bzw. A16.

© 2007-2015 Analytica GmbH

Modus [1]	**Verwendung**	**Beschreibung**
0x6	A1-A0-E0	Die Bits E2 und E1 werden für die Adressierung verwendet. E1 entspricht dabei dem Adressbit A8 bzw. A16 und E2 dem Adressbit A9 bzw. A17.

[1] ist bei den API-Funktionen `I2CReadEEProm` und `I2CWriteEEProm` im Parameter *`nOffsetFormat`* (Bit 8-10) entsprechend zu verwenden.

Tabelle C.1. Verwendung der Chip-Enable-Bits bei I2C-EEPROMs

 © 2007-2015 Analytica GmbH

Anhang D. FAQ - Häufig gestellte Fragen

Im folgenden eine Aufstellung von häufig gestellten Fragen hinsichtlich der Inbetriebnahme und Verwendung der unterschiedlichen *AnaGate*-Hardware.

D.1. Allgemeine Fragen

F: Keine Netzwerk-Verbindung (1)

A: Überprüfen Sie bitte zuerst, ob eine physische Netzwerkverbindung zum Gerät vorhanden ist. Grundsätzlich muss das *AnaGate* direkt mit einem PC oder einer aktiven Netzwerkkomponente (Hub, Switch) über ein LAN-Kabel verbunden sein. Bei der Verbindung zu einem PC muss ein gekreuztes LAN-Kabel benutzt werden, andernfalls kann das mitgelieferte LAN-Kabel verwendet werden.

Die physische Verbindung ist okay, falls die gelbe Link-LED bei der RJ45-Buchse leuchtet, wenn das LAN-Kabel verbunden wird. Die Link-LED leuchtet konstant, solange die Verbindung besteht. Bei einigen Geräten-Modellen blinkt die LED im Betrieb auch synchron zur grünen Activity-LED an der RJ45-Buchse bei Datenverkehr.

Falls die Link-LED grundsätzlich nicht leuchtet, ist die physische Verbindung gestört. Überprüfen Sie in diesem Fall die Netzwerkverkabelung.

F: Keine Netzwerk-Verbindung (2)

A: Wenn die Link-LED eine Ethernet-Verbindung anzeigt (siehe vorherige FAQ), das *AnaGate* aber trotzdem nicht erreichbar ist, gehen Sie bitte wie folgt vor:

1. Prüfen Sie, ob das *AnaGate* per Ping erreichbar ist. Dazu geben Sie unter Windows in einer Eingabeaufforderung den Befehl **ping a.b.c.d** ein, wobei `a.b.c.d` durch die IP-Adresse des Geräts zu ersetzen ist.

2. Sollte das *AnaGate* mit Ping nicht erreichbar sein, setzen Sie das Gerät in den Auslieferungszustand zurück. Stellen Sie die IP-Adresse Ihres PCs auf `192.168.1.253` und die Subnetzmaske auf `255.255.255.0`. Prüfen Sie, ob das *AnaGate* mittels **ping 192.168.1.254** erreichbar ist.

3. Wenn das Gerät mittels Ping erreichbar ist, prüfen Sie als Nächstes, ob Sie eine TCP-Verbindung auf dem Port 5001 herstellen können. Dazu geben Sie unter Windows in einer Eingabeaufforderung den Befehl **telnet a.b.c.d**

© 2007-2015 Analytica GmbH

5001 ein, wobei `a.b.c.d` durch die IP-Adresse des Geräts zu ersetzen ist. Sollte dieser Befehl eine Fehlermeldung erzeugen, prüfen Sie, ob auf Ihrem PC eine Firewall aktiviert ist oder im Netzwerk zwischen Ihrem PC und dem *AnaGate* ein Paketfilter aktiv ist.

F: Keine Netzwerk-Verbindung nach Änderung der Netzwerkadresse

A: Nach der Änderung der Netzwerkadresse über das Web-Interface des Gerätes kann das Gerät nicht mehr erreicht werden. Der verwendete Internet-Browser gibt bei Eingabe der IP-Adresse eine leere Seite zurück. Eine Fehlermeldung, dass die Zieladresse nicht erreichbar ist, wird nicht angezeigt.

Überprüfen Sie, ob Ihr Antiviren-Programm den Zugriff auf die IP-Adresse blockiert. Beim Ändern der Netzwerk-Adresse wird ein Redirect auf die neue Adresse des Gerätes durchgeführt. Solche Umleitungen stufen Antivirenprogramme gelegentlich als verdächtig ein, was zur Folge hat, dass die neue Geräte-Adresse blockiert wird. Die Blockierung von Netzwerkadressen erfolgt teilweise automatisch ohne Benachrichtigung des Benutzers.

F: Verbindungsprobleme bei Verwendung mehrerer Geräte

A: Werden in einem lokalen Netzwerk mehrere Geräte gleichzeitig betrieben, kann es zu Verbindungsproblemen kommen, wenn zwei Geräte mit identischer IP-Adresse verwendet werden. Es ist deshalb sicherzustellen, dass auf allen gleichzeitig eingesetzten Geräten unterschiedliche IP-Adressen eingestellt sind.

Diese Problematik tritt ebenfalls auf, wenn Geräte mit gleicher IP-Adresse zwar nicht gleichzeitig im Netzwerk vorhanden sind, jedoch im kurzen zeitlichen Abstand nacheinander angeschlossen werden. Dies kann zum Beispiel bei der initialen Konfiguration von mehreren Neugeräten der Fall sein, die in der Grundeinstellung (IP-Adresse `192.168.1.254`) ausgeliefert werden.

Bei IPv4-Netzwerken wird das **Address Resolution Protocol (ARP)** verwendet, um die MAC-Adressen zu gegebenen IP-Adressen zu ermitteln. Die dafür notwendigen Informationen werden im *ARP-Cache* zwischengespeichert. Wenn falsche bzw. nicht mehr aktuelle Einträge vorhanden sind, kann mit dem betreffenden Host nicht kommuniziert werden.

Das Zeitintervall, nach dem ein Eintrag aus dem ARP-Cache gelöscht wird, ist implementierungsabhängig. So verwerfen aktuelle Linux-Distributionen Einträge nach ca. 5 Minuten. Sobald ein Eintrag in der Tabelle genutzt wird, wird dessen Ablaufzeit verlängert. Unter Unix und Windows kann der ARP-Cache mit dem Kommando **arp** angezeigt und manipuliert werden.

```
C:\>arp -a

Schnittstelle: 10.1.2.50 --- 0x2
  Internetadresse       Physikal. Adresse     Typ
  192.168.1.254         00-50-c2-3c-b0-df     dynamisch
```

Mittels des Kommandos **arp -d** kann der gesamte *ARP-Cache* geleert werden.

Anmerkung

Eventuell muss auch nach dem Ändern der IP-Adresse eines Gerätes der *ARP-Cache* des PCs gelöscht werden.

 © 2007-2015 Analytica GmbH

F: Verwendung einer Firewall

A: Bei Verwendung einer Firewall muss der entsprechende Port für die Kommunikation mit dem AnaGate freigeschaltet sein:

Gerät	Portnummer
AnaGate I2C	5000
AnaGate I2C X7	5100, 5200, 5300, 5400, 5500, 5600, 5700
AnaGate CAN	5001
AnaGate CAN USB	5001
AnaGate CAN uno	5001
AnaGate CAN duo	5001, 5101
AnaGate CAN quattro	5001, 5101, 5201, 5301
AnaGate CAN X1	5001
AnaGate CAN X2, AnaGate CAN-FD X2	5001, 5101
AnaGate CAN X4, AnaGate CAN-FD X4	5001, 5101, 5201, 5301
AnaGate CAN X8	5001, 5101, 5201, 5301, 5401, 5501, 5601, 5701
AnaGate SPI	5002
AnaGate Renesas	5008
AnaGate Universal Programmer UP/UPP	5000, 5002, 3333, 4444, 20, 21
AnaGate Universal Programmer UPR	5000, 5002, 5008, 3333, 4444, 20, 21
AnaGate Universal Programmer UP 2.0	5000, 5002, 3333, 4444, 20, 21

Tabelle D.1. Nutzung AnaGate-Hardware mit Firewall

D.2. Fragen zum AnaGate CAN

F: Wie hoch ist der Widerstand der Terminierung, wenn die integrierte Terminierung über die Einstellungen aktiviert wird?

A: Der Terminierungs-Widerstand eines *AnaGate* wird durch einen FET-Transistor getrieben. Der Widerstand selbst beträgt 110 Ohm, und der interne Widerstand des FET bei Aktivierung beträgt 10 Ohm, so dass der Gesamtwiderstand 120 Ohm beträgt, wie auf dem CAN Bus notwendig.

F: Bietet Analytica auch auch ein CAN-Gateway ohne galvanisch getrennte CAN-Schnittstelle an?

A: Ein Gerät, das aktiv am CAN-Bus betrieben wird, sollte grundsätzlich eine galvanische Trennung besitzen. Insbesondere bei USB-Geräten (wie z.B. dem *AnaGate USB*), deren Spannungsversorgung über den PC erfolgt, ist die galvanische Trennung zum CAN-Bus unabdingbar.

F: Was ist beim direkten Verbinden von 2 CAN-Ports zu beachten?

 © 2007-2015 Analytica GmbH

A: Wenn zwei CAN-Ports auf einem *AnaGate CAN* bzw. zwei Ports auf unterschiedlichen *AnaGate CAN* mit einem CAN-Kabel direkt verbunden werden sollen, muss auf beiden Seiten die interne Terminierung eingeschaltet werden. Ein CAN-Netzwerk muss auf beiden Seiten eine Terminierung aufweisen.

Anmerkung

Es ist auf jeden Fall zu empfehlen, die Terminierung einzuschalten, auch wenn möglicherweise keine Probleme bei geringen Busgeschwindigkeiten auftreten.

F: Beim Versand von CAN-Nachrichten wird ein NAK gesendet!

A: Wenn kein CAN-Partner am *AnaGate CAN* (resp. CAN-Bus) angeschlossen ist, sendet der CAN-Controller ein sog. NAK, d.h. das dass Paket nicht versendet werden konnte.

Warnung

Falls Sie keine Paket-Konfirmierung für Data-Requests aktiviert haben, erhalten Sie diese Fehler jedoch nicht, da Fehler über Bestätigungstelegramme versendet werden. Die Option *Bestätigungstelegramme für Data-Requests* kann über die Funktion `CANSetGlobals` gesetzt werden. Im Highspeed-Mode sind Bestätigungstelegramme grundsätzlich abgeschaltet.

D.3. Fragen zur SPI-Schnittstelle

F: Keine SPI-Kommunikation

A: Sollten Sie keine SPI-Kommunikation mit Ihrem SPI-Device herstellen können, gehen Sie wie folgt vor:

1. Prüfen Sie, dass das SPI-Device als auch die SPI-Schnittstelle des *AnaGate UP 2.0* mit Spannung versorgt wird.
2. Prüfen Sie, dass kein weiteres Gerät/µC auf dem SPI-Bus aktiv ist.
3. Prüfen Sie, dass keine weiteren elektronischen Bauelemente die Kommunikation auf dem SPI zwischen dem AnaGate SPI und dem SPI-Device stören können.

D.4. Fragen zur I2C-Schnittstelle

F: Keine I2C-Kommunikation

A: Sollten Sie keine I2C Kommunikation mit Ihrem I2C-Device herstellen können, gehen Sie wie folgt vor:

1. Prüfen Sie, dass das I2C Gerät und die I2C-Schnittstelle des *AnaGate UP 2.0* mit Spannung versorgt werden.
2. Prüfen Sie, dass kein weiteres Gerät/µC auf dem I2C-Bus aktiv ist.
3. Prüfen Sie, dass die SDA und SCL Leitungen mit einem adäquaten Pull-up-Widerstand (z.B. 4,7 kOhm) auf Versorgungsspannung gezogen werden.

 © 2007-2015 Analytica GmbH

4. Prüfen Sie, dass keine weiteren elektronischen Bauelemente die Kommunikation auf dem I2C zwischen Gerät und dem I2C-Device stören können.

5. Prüfen Sie, dass die Chip-Enable-Adresse vom I2C-Device und der Software identisch sind.

F: Welche Reihenfolge ist bei der Kontaktierung von GND, SCL und SDA bei Verwendung eines externen Stromversorgung zu beachten?

A: Um potentielle Schäden am *AnaGate UP 2.0* zu vermeiden, muss immer zuerst der GND-Pin mit dem Application Board verbunden werden; erst danach dürfen die Pins SCL und SDA mit dem Application Board kontaktiert werden.

D.5. Fragen zur JTAG-Schnittstelle

F: Keine JTAG-Kommunikation

A: Sollten Sie keine Kommunikation mit dem JTAG-Device herstellen können, gehen Sie wie folgt vor:

1. Prüfen Sie, dass das JTAG-Device als auch die JTAG-Schnittstelle des *AnaGate UP 2.0* mit Spannung versorgt wird.

2. Prüfen Sie, dass vom letzten JTAG-Device der TDO-Pin auf den TDO-Pin des *AnaGate UP 2.0* geführt wird.

D.6. Fragen zur Renesas-Schnittstelle

F: Keine Renesas-Kommunikation

A: Sollten Sie keine Kommunikation mit dem Renesas-Device herstellen können, gehen Sie wie folgt vor:

1. Prüfen Sie, dass das Renesas-Device als auch die Renesas-Schnittstelle des *AnaGate UP 2.0* mit Spannung versorgt wird.

2. Prüfen Sie, dass der Reset- und Mode-Pin nicht über zu kleine Pull-up- bzw. Pull-down-Widerstände auf Vcc bzw. GND gezogen wurde.

 © 2007-2015 Analytica GmbH

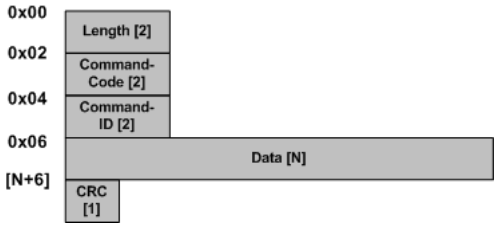
Anhang E. FAQ zu den Programmier-Schnittstellen

Im folgenden eine Aufstellung von häufig gestellten Fragen in Bezug zur Programmier-API bzw. der Programmierung als solches.

E.1. Fragen zum Kommunikationsprotokoll

F: Die Berechnung der Prüfsumme (CRC) für die AnaGate-Telegramme ist unklar.

A: Die folgende Abbildung zeigt den grundsätzlichen Aufbau eines *AnaGate*-Befehlstelegramms:



Als Prüfsumme wird ein Byte benutzt, das sich durch ein XOR von sämtlichen Bytes eines Befehlstelegramms ohne die Längen-Bytes und CRC selbst errechnet.

Die folgende C-Funktion berechnet den CRC eines bereits erzeugten Befehlstelegramms:

```
unsigned char CalcCRC( char * pBuffer, int nBufferLength )
{
   int i;
   unsigned char nCRC = pBuffer[2]; // skip the length bytes

   // XOR all bytes in the message except the length information and the last byte
   for( i = 3; i < nBufferLength - 1; ++i )
   {
      nCRC ^= pBuffer[i];
   }
   return nCRC;
}
```

Bei Verwendung der Funktion `CalcCRC` muss der Übergabeparameter *`pBuffer`* auf einen Datenpuffer zeigen, der das vollständig erzeugte Datentelegramm enthält. Die Längenangabe *`nBufferLength`* ist abhängig vom erzeugten Befehlstyp und kann über folgenden Pseudocode berechnet werden:

```
buffer length = sizeof( command length ) + sizeof( command code )
                + sizeof( command id ) + sizeof( CRC ) + sizeof(data)
              = 7 + sizeof(data)
```


© 2007-2015 Analytica GmbH

Anhang F. Technischer Support

Die Hardware-Serie AnaGate, die vorhandenen Programmierschnittstellen sowie verschiedene Tools werden von der Analytica GmbH entwickelt und unterstützt. Technische Unterstützung kann wie folgt angefordert werden:

Internet

Die AnaGate-Website [http://www.anagate.de] der Analytica GmbH enthält Informationen und Software-Downloads für Benutzer der AnaGate Library:

- Kostenlose Produkt-Updates, die Fehlerbehebungen oder neue Features beinhalten.

E-Mail

Für technische Unterstützung über Internet senden Sie bitte eine E-Mail an

`<support@anagate.de>`

Helfen Sie uns bei der optimalen Unterstützung und halten Sie stets folgende Informationen bereit, wenn Sie mit dem Support in Verbindung treten.

- Versionsnummer der jeweiligen Systemkomponente bzw. des Programm-Tools
- AnaGate-Modell und Firmware-Version
- Name und Version des verwendeten Betriebssystems

© 2007-2015 Analytica GmbH

Literaturverzeichnis

Bücher

[LuaRef2006-EN] Roberto Ierusalimschy, Luiz Henrique Figueiredo und Waldemar Celes. Copyright © 2006 R. Ierusalimschy, L. H. de Figueiredo, W. Celes. ISBN 85-903798-3-3. Lua.org. *Lua 5.1 Reference Manual*.

[LuaProg2006-EN] Roberto Ierusalimschy. Copyright © 2006 Roberto Ierusalimschy, Rio de Janeiro. ISBN 85-903798-2-5. Lua.org. *Programming in Lua (second edition)*.

[LuaProg2006-DE] Roberto Ierusalimschy. Copyright © 2006 Roberto Ierusalimschy, Rio de Janeiro. ISBN 3-937514-22-8. Open Source Press, München. *Programmieren mit Lua*.

[LuaProg2013-EN] Roberto Ierusalimschy. Copyright © 2013 Roberto Ierusalimschy, Rio de Janeiro. ISBN 85-903798-5-X. Lua.org. *Programming in Lua, Third Edition*.

[LuaProg2013-DE] Roberto Ierusalimschy. Copyright © 2013 Roberto Ierusalimschy, Rio de Janeiro. ISBN 978-3-95539-020-4. Open Source Press, München. *Programmieren mit Lua*.

Publikationen

[NXP-I2C] NXP Semiconductors. Copyright © 2007 NXP Semiconductors. *UM10204*. I2C-bus specification and user manual. Rev. 03. 19.06.2007.

[TCP-2010] Analytica GmbH. Copyright © 2010 Analytica GmbH. *Handbuch TCP-IP Kommunikation* . Version 1.3.1. 04.06.2010.

[Prog-2013] Analytica GmbH. Copyright © 2013 Analytica GmbH. *AnaGate API 2* . Programmer's Manual . Version 2.0. 15.05.2013.

[CiA-DS301] Copyright © 2002 CAN in Automation (CiA) e. V.. CAN in Automation (CiA) e.V.. Version 4.0.2. 13.02.2002. *Cia 301, CANopen Application Layer and Communication Profile*.