AnaGate API

Programmer's Manual

Analytica GmbH

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

AnaGate API: Programmer's Manual

Analytica GmbH
von A. Schmidt und S. Welisch

Dieses Dokument wurde mittels DocBook am 13.03.2012 09:35:11 erzeugt.

Hilfe-Datei (dtsch.): *AnaGate-API.chm*

Hilfe-Datei (engl.): *AnaGate-API-EN.chm*

PDF-Datei (dtsch.): *AnaGate-API-1.10.pdf*

PDF-Datei (engl.): *AnaGate-API-1.10-EN.pdf*

Veröffentlicht 09. September 2010

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 1.10 und dem AnaGate Kommunikationsprotokoll V1.3 (siehe [TCP-2010]).

Analytica GmbH
Vorholzstraße 36
76137 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 1.4	01.10.2010	ASce	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

Abbildungsverzeichnis

Tabellenverzeichnis

Beispiele

Einleitung

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

Im folgenden geht das Dokument auf folgendes Schnittstellen ein:

- Application Programming Interface (Teil I, „AnaGate API“)
- LUA Scripting Interface (Teil II, „Skriptspache LUA“)

Teil I. AnaGate API

Inhaltsverzeichnis

Kapitel 1. Die Programmierschnittstelle der *AnaGate*-Serie

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

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, ...), programm-gesteuert auf die Geräte der *AnaGate*-Serie zugreifen.

Speziell für Nutzer von Windows- und X86-Linux-Betriebssystemen, stellt die 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.

Tabelle 1.1. Programm-Bibliotheken der API unter Windows

Gerät	Windows-Bibliothek
AnaGate CAN	AnaGateCAN.dll
AnaGate CAN uno / duo / quattro	AnaGateCAN.dll
AnaGate CAN USB	AnaGateCAN.dll
AnaGate SPI	AnaGateSPI.dll
AnaGate I2C / I2C X7	AnaGateI2C.dll
AnaGate Universal Programmer	AnaGateSPI.dll, AnaGateI2C.dll

Anmerkung

Um eine umfassende Unterstützung für verschiedene Programmiersprachen wie z.B. C++, Visual Basic, Delphi oder auch die Programmiersprachen der .NET-Familie unter Windows-Betriebsstemen 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.

Tabelle 1.2. Programm-Bibliotheken der API unter Linux

Gerät	Linux-Bibliothek (X86)	ARM9
AnaGate CAN	libCANdll.a, libAnaCommon.a	-
AnaGate CAN uno / duo / quattro	libCANdll.a, libAnaCommon.a	verfügbar
AnaGate CAN USB	libCANdll.a, libAnaCommon.a	verfügbar
AnaGate SPI	libSPIdll.a, libAnaCommon.a	-

Gerät	Linux-Bibliothek (X86)	ARM9
AnaGate I2C / I2C X7	libI2Cdll.a, libAnaCommon.a	-
AnaGate Universal Programmer	libSPIdll.a, libI2Cdll.a, libAnaCommon.a	verfügbar

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.

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.

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.

Tabelle 2.1. AnaGate-Modelle und Ihre Portnummern

Gerät	Portnummer
AnaGate I2C, AnaGate Universal Programmer	5000
AnaGate CAN, AnaGate CAN uno	5001
AnaGate CAN duo	5001, 5101
AnaGate CAN quattro	5001, 5101, 5201, 5301
AnaGate SPI, AnaGate Universal Programmer	5002
AnaGate Renesas, AnaGate Universal Programmer	5008

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 Reihefolge beim Empfänger ankommen. Wird ein Datenpaket vom Empfänger nicht innerhalb einer Timout-Zeit quittiert, wird das Datenpaket erneut gesendet. Doppelten Pakete werden beim Empfänger erkannt und verworfen. Der

Datentransfer ansich 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.

Inbesondere gestaltet sich der Erkennung und Einschätzung von aktuellen Leitungsstörungen als schwierig, wenn nur sporadische Kommunikation auf der Verbindung stattfindet, 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-Mechnismus.

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 (Administratorrechte).

Inbesondere können die CAN-Ethernet-Gatways von der Problematik der Verbindungskontrolle betroffen sein, wenn kundenspezifische Systemanforderungen eine schnellere Erkennung eines Verbindungsabbruches benötigen als das 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 anwendungsspezifischen 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.

Kapitel 3. Allgemeine Funktionen

DLLInfo

DLLInfo — Ermittelt die aktuelle Version der AnaGate DLL.

Syntax

```
#include <AnaGateDLL.h>

int DLLVersion(char * pcMessage, int 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.

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

CANOpenDevice, CANOpenDeviceEx

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

Syntax

```
#include <AnaGateDllCan.h>

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

int CANOpenDeviceEx(int *pHandle, BOOL bSendDataConfirm, BOOL bSendDataInd, int nCANPort, const char * pcIPAddress, int nTimeout , int 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 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)
pcIPAddress	Netzwerkadresse des AnaGate Partners.
nTimeout	Standard-Timeout für AnaGate-Zugriffe in Millisekunden. Ein Timeout wird festgestellt, wenn der 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 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 Layer4 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 Schliesen der Verbindung ist notwendig, um die in der DLL angelegten Systemresourcen 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 intialen Zugriff auf eine Schnittstelle.

```
#include <AnaGateCANDll.h>
int main()
{
   int hHandle;
   int nRC = CANOpenDevice(&hHandle, TRUE, TRUE, 0, "192.168.0.254", 5000);
   if ( nRC == 0 )
   {
       // ... now do something
       CANCloseDevice(hHandle);
   }
   return 0;
}
```

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 `CANOpenDeviceEx` fehlerhaft beendet (Timeout).

Siehe auch

CANCloseDevice

CANRestart

CANCloseDevice

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

Syntax

```
#include <AnaGateDllCan.h>

int CANCloseDevice(int 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 Schliesen der Verbindung ist notwendig, um die in der DLL angelegten Systemresourcen 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

CANSetGlobals

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

Syntax

```
#include <AnaGateDllCAN.h>

int CANSetGlobals(int hHandle, int nBaudrate, unsigned char nOperatingMode, BOOL bTermination, BOOL bHighSpeedMode, BOOL 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

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

- 0 = für Standard-Modus.
- 1 = für LoopBack-Modus: In diesem Modus werden alle über das AnaGate CAN versendete Nachrichten zusätzlich an die verbundenen Partner als Data Indications versendet.
- 2 = für 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).

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 abgeschaltet.

bTimeStampOn Aktuelle Einstellung für den Zeitstempel-Modus (TRUE= ein, FALSE= aus). Diese Einstellung wird nur 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 angibt. 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.

Siehe auch

CANGetGlobals

CANGetGlobals

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

Syntax

#include <AnaGateDllCAN.h>

int **CANGetGlobals**(int *hHandle*, int * *pnBaudrate*, unsigned char * *pnOperatingMode*, BOOL * *pbTermination*, BOOL * *pbHighSpeedMode*, BOOL * *pbTimeStampOn*);

Parameter

hHandle Gültiges Zugriffs-Handle eines erfolgreichen Aufrufs von CANOpenDevice.

pnBaudrate Aktuell eingestellte Baudrate.

pnOperatingMode Aktuell eingestellter Betriebsmodus. Folgende Werte sind möglich:

- 0 = für Standard-Modus.
- 1 = für LoopBack-Modus: In diesem Modus werden alle über das AnaGate CAN versendete Nachrichten zusätzlich an die verbundenen Partner als Data Indications versendet.
- 2 = für 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).

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 bestätigt und die via CANSetFilter definierten Filter werden abgeschaltet.

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 angibt. 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.

Siehe auch

CANGetGlobals

CANSetFilter

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

Syntax

```
#include <AnaGateDllCAN.h>

int CANSetFilter(int hHandle, const int * 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 höchstmöglich 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 jeweilegen Verbindungspartner weitergeleitet.

Ein Maskenfilter besteht aus der Filtermaske, die angibt, welche Bits des CAN-Identifier geprüft werden sollen, und des entsprechenden Filterwertes. 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.

Filter sind grundsätzlich deaktiviert, falls der Parameter *bSendDataInd* beim Ausführen der Funktion `CANOpenDevice` gesetzt wurde. Wird über die Funktion `CANSetGlobals` der Highspeed-Modus aktiviert, werden alle Filter ausgeschaltet, um den Datensatzdurchsatz zu erhöhen.

Tabelle 4.1. Beispiele für Maskenfilter für CAN Identifier

CAN id	Filtermaske	Filterwert	Ergebnis
0x0F	0x0E	0x0C	unterdrückt
0x0C	0x0E	0x0C	ok

CAN id	Filtermaske	Filterwert	Ergebnis
0x5D	0x0E	0x0C	ok

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

```
#include <AnaGateCANDll.h>
int main()
{
{  int 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
   };
   int hHandle;
   int nRC = CANOpenDevice(&hHandle, TRUE, TRUE, 0, "192.168.0.254", 5000);
   if ( nRC == 0 )
   {
       nRC = CANSetFilter( hHandle, &anFilter );
       // ... now do something
       CANCloseDevice(hHandle);
   }
   return 0;
}
```

Siehe auch

CANGetFilter

CANGetFilter

CANGetFilter — Liefert die Filtereinstellungen für die aktuelle Verbindung zurück.

Syntax

```
#include <AnaGateDllCAN.h>

int CANGetFilter(int hHandle, int * 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. Bei unbenutzten Maskenfiltereinträgen sind beide Filterwerte mit 0 besetzt. Bei unbenutzten Bereichfilterwerten 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 jeweilegen Verbindungspartner weitergeleitet.

Siehe auch

CANSetFilter

CANSetTime

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

Syntax

#include <AnaGateDllCAN.h>

```
int CANSetTime(int hHandle, long nSeconds, long 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 zurückübermittelt, 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.

CANWrite, CANWriteEx

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

Syntax

#include <AnaGateDllCan.h>

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

int CANWriteEx(int hHandle, int nIdentifier, const char * pcBuffer, int nBufferLen, int nFlags, long * pnSeconds, long * 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 von 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 am Zeitstempel.

Rückgabewert

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

Beschreibung

Beide Funktionen senden ein Datentelegramm auf den CAN-Bus.

Mit `CANWriteEx` kann zusätzlich der Zeitpunkt auf dem Geräte 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 <AnaGateCANDll.h>
int main()
{
   char cMsg[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
   int hHandle = 0;
   int nRC = 0;
   int nFlags = 0x0; //  11bit address + standard (not remote frame)
   int nIdentifier = 0x25; // send with CAN ID 0x25;

   int nRC = CANOpenDevice(&hHandle, TRUE, TRUE, 0, "192.168.0.254", 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.

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)(int nIdentifier, const char *
pcBuffer, int nBufferLen, int nFlags, int hHandle);

int CANSetCallback(int hHandle, CAN_PF_CALLBACK pCallbackFunction);

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

int CANSetCallbackEx(int 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

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`, `alloc` oder `free`).

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

```
#include <AnaGateCANDll.h>

// Defintion of a callback, which writes incoming CAN data with timestamp to console
void WINAPI MyCallbackEx(int nIdentifier, const char * pcBuffer, int nBufferLen, int nFlags,
                         int hHandle, long nSeconds, long nMicroseconds)
{
   std::cout << "CAN-ID=" << nIdentifier << ", Data=";
   for ( int 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()
{
   int hHandle = 0;
   int nRC = 0;

   int nRC = CANOpenDevice(&hHandle, TRUE, TRUE, 0, "192.168.0.254", 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

CANReadDigital

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

Syntax

#include <AnaGateDllCan.h>

`int` **`CANReadDigital`**`(int` *`hHandle`*`, unsigned long * ` *`pnInputBits`*`, unsigned long * ` *`pnOutputBits`*`);`

Parameter

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

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

```
#include <AnaGateCANDll.h>
int main()
{
   int hHandle = 0;
   int nRC = 0;
   unsigned long nInputs;
   unsigned long nOutputs = 0x03;

   int nRC = CANOpenDevice(&hHandle, TRUE, TRUE, 0, "192.168.0.254", 5000);
   if ( nRC == 0 )
   {
       // 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

CANWriteDigital

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

Syntax

```
#include <AnaGateDllCAN.h>

int CANWriteDigital(int hHandle, unsigned long 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 und 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 `CANWriteDigital` verändert werden.

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

Siehe auch

CANReadDigital

CANRestart

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

Syntax

```
#include <AnaGateDllCan.h>

int CANRestart(const char * pcIPAddress, int 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 schliest 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

CANDeviceConnectState

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

Syntax

```
#include <AnaGateDllCAN.h>

int CANDeviceConnectState(int hHandle);
```

Parameter

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

Rückgabewert

Gibt den aktueller 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

CANStartAlive

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

Syntax

```
#include <AnaGateDllCan.h>

int CANStartAlive(int hHandle, int 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 Prozess, 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 “

CANErrorMessage

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

Syntax

#include <AnaGateDllCAN.h>

```
int CANErrorMessage(int nRetCode, char * pcMessage, int 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++.

```
int nRC;
char cBuffer[200];
int nRC;
//... call a API function here
CANErrorMessage(nRC, cBuffer, 200);
std::cout << "Fehler: " << cBuffer << std::endl;
```


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

SPIOpenDevice

SPIOpenDevice — Baut eine Netzwerkverbindung zu einem AnaGate SPI auf.

Syntax

#include <AnaGateDllSPI.h>

`int` **`SPIOpenDevice`**`(int * pHandle, const char * pcIPAddress, int 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 der 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 neuer Verbindungsversuch abgelehnt.

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

```
#include <AnaGateDllSPI.h>
int main()
{
   int hHandle;
   int nRC = SPIOpenDevice(&hHandle, "192.168.0.254", 5000);
   if ( nRC == 0 )
   {
       // ... now do something
```

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

Siehe auch

SPICloseDevice

SPICloseDevice

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

Syntax

```
#include <AnaGateDllSPI.h>

int SPICloseDevice(int 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 Schliesen der Verbindung ist notwendig, um die in der DLL angelegten Systemresourcen 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

SPISetGlobals

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

Syntax

```
#include <AnaGateDllSPI.h>

int SPISetGlobals(int hHandle, int nBaudrate, unsigned char nSigLevel,
unsigned char nAuxVoltage, unsigned char 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 500kBit
- 1.000.000 für 1MBit
- 5.000.000 für 5MBit

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 +5.0V für die Signale.
- 2 für +3.3V für die Signale.
- 3 für +2.5V für die Signale.

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

- 0 für Hilfsspannung +3.3V.
- 1 für Hilfsspannung +2.5V für die Signale.

nClockMode Gibt die Phase und die Polarität der Clock-Leitung für die Datenübertragung an. 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

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

SPIGetGlobals

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

Syntax

#include <AnaGateDllSPI.h>

`int` **`SPIGetGlobals`**`(int` *`hHandle`*`, int * ` *`pnBaudrate`*`, unsigned char * ` *`pnSigLevel`*`, unsigned char * ` *`pnAuxVoltage`*`, unsigned char * ` *`pnClockMode`*`);`

Parameter

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

pnBaudrate Aktuell auf dem SPI-Bus eingestellte Baudrate in kBit.

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

- 0 für Ausgänge im High Impedance Modus (Standard-Modus).
- 1 für +5.0V für die Signale.
- 2 für +3.3V für die Signale.
- 3 für +2.5V für die Signale.

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

- 0 für Hilfsspannung +3.3V.
- 1 für Hilfsspannung +2.5V für die Signale.

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.

Siehe auch

SPISetGlobals

SPIDataReq

SPIDataReq — Führt einen Datentransfer auf dem SPI Bus durch.

Syntax

```
#include <AnaGateDllSPI.h>

int SPIDataReq(int hHandle, const char * pcBufWrite, int nBufWriteLen,
char * pcBufRead, int 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 SPI Partner gesendet werden, aufnehmen soll.
nBufReadLen	Anzahl der Bytes, die gelesen werden sollen. Darf die Länge des Datenpuffers *pcBufReda* 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 vollduplex (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 den SPI Bus sendet und empfängt.

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

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

   int 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*).

SPIReadDigital

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

Syntax

#include <AnaGateDllSPI.h>

int **SPIReadDigital**(int *hHandle*, unsigned long * *pnInputBits*, unsigned long * *pnOutputBits*);

Parameter

hHandle	Gültiges Zugriffs-Handle eines erfolgreichen Aufrufs von SPIOpenDevice.
pnInputBits	Zeiger auf den aktueller Inhalt des Registers mit den digitalen Eingängen. Dabei entsprechen Bit 0 bis Bit 3 den vier digitalen Eingängen und Bit 4 bis Bit 31 sind auf 0 gesetzt.
pnOutputBits	Zeiger auf den aktueller Inhalt des Registers mit den digitalen Ausgängen. Dabei entsprechen Bit 0 bis Bit 3 den vier digitalen Ausgängen und 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 digitale IOs setzt und zurückliest.

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

   int nRC = SPIOpenDevice(&hHandle, "192.168.0.254", 5000);
   if ( nRC == 0 )
   {
       // set the digital output register (PIN 0 and PIN 1 to HIGH value)
       nRC = SPIWriteDigital( hHandle, nOutputs );
```



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

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

Siehe auch

SPIWriteDigital

SPIWriteDigital

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

Syntax

#include <AnaGateDllSPI.h>

```
int SPIWriteDigital(int hHandle, unsigned long 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 und 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 IO ist bei der Beschreibung von `SPIReadDigital` zu finden.

Siehe auch

SPIReadDigital

SPIErrorMessage

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

Syntax

#include <AnaGateDllCAN.h>

```
int SPIErrorMessage(int nRetCode, char * pcMessage, int 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++.

```
int nRC;
char cBuffer[200];
int nRC;
//... call a API function here
SPIErrorMessage(nRC, cBuffer, 200);
std::cout << "Fehler: " << cBuffer << std::endl;
```


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

I2COpenDevice

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

Syntax

#include <AnaGateDllI2C.h>

```
int I2COpenDevice(int * pHandle, unsigned int nBaudrate, const char *
pcIPAddress, int 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 100kBit (Standard Mode)
- 400000 für 400kBit (Fast Mode)

Anmerkung

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

pcIPAddress — Netzwerkadresse des AnaGate Partners.

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

Ein Timeout wird festgestellt, wenn der 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

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

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

```
#include <AnaGateDllI2C.h>
int main()
{
   int hHandle;
   int nRC = I2COpenDevice(&hHandle, 100000, "192.168.0.254", 5000);
   if ( nRC == 0 )
   {
       // ... now do something
       I2CCloseDevice(hHandle);
   }
   return 0;
}
```

Siehe auch

I2CCloseDevice

I2CCloseDevice

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

Syntax

```
#include <AnaGateDllI2C.h>

int I2CCloseDevice(int 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 Schliesen der Verbindung ist notwendig, um die in der DLL angelegten Systemresourcen 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

I2CReset

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

Syntax

```
#include <AnaGateDllI2C.h>

int I2CReset(int 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.

I2CRead

I2CRead — Liest Daten von einem I2C Partner.

Syntax

```
#include <AnaGateDllI2C.h>

int I2CRead(int hHandle, unsigned short nSlaveAddress, const char * pcBuffer, int nBufferLen);
```

Parameter

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

I2CWrite

I2CWrite

I2CWrite — Schreibt Daten zu einem I2C Partner.

Syntax

```
#include <AnaGateDllI2C.h>

int I2CWrite(int hHandle, unsigned short nSlaveAddress, const char * pcBuffer, int nBufferLen);
```

Parameter

hHandle	Gültiges Zugriffs-Handle eines erfolgreichen Aufrufs von `I2COpenDevice`.
nSlaveAddress	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*).
pcBuffer	Byte-Puffer mit den Daten, die an den I2C Partner gesendet werden sollen.
nBufferLen	Anzahl von Datenbytes, die gelesen werden sollen.

Rückgabewert

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

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

I2CRead

I2CSequence

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

Syntax

#include <AnaGateDllI2C.h>

int **I2CSequence**(int *hHandle*, const char * *pcWriteBuffer*, int *nNumberOfBytesToWrite*, char * *pcReadBuffer*, int *nNumberOfBytesToRead*, int * *pnNumberOfBytesRead*, int * *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.

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 1. Lesekommandos, direkt danach die Daten des 2. 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.

I2CReadDigital

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

Syntax

#include <AnaGateDllI2C.h>

`int` **`I2CReadDigital`**`(int` *`hHandle`*`, unsigned long *` *`pnInputBits`*`, unsigned long *` *`pnOutputBits`*`);`

Parameter

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

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

   int nRC = I2COpenDevice(&hHandle, 400000, "192.168.0.254", 5000);
   if ( nRC == 0 )
   {
       // set the digital output register (PIN 0 and PIN 1 to HIGH value)
       nRC = I2CWriteDigital( hHandle, nOutputs );
```



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

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

Siehe auch

I2CWriteDigital

I2CWriteDigital

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

Syntax

```
int I2CWriteDigital(int hHandle, unsigned long 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 und 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 IO ist bei der Beschreibung von `I2CReadDigital` zu finden.

Siehe auch

I2CReadDigital

I2CErrorMessage

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

Syntax

#include <AnaGateDllCAN.h>

```
int I2CErrorMessage(int nRetCode, char * pcMessage, int 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++.

```
int nRC;
char cBuffer[200];
int nRC;
//... call a API function here
I2CErrorMessage(nRC, cBuffer, 200);
std::cout << "Fehler: " << cBuffer << std::endl;
```


I2CReadEEPROM

I2CReadEEPROM — Liest Daten von einem EEPROM am I2C-Bus.

Syntax

```
#include <AnaGateDllI2C.h>

int I2CReadEEPROM(int hHandle, unsigned short nSubAddress, unsigned
int nOffset, const char * pcBuffer, int nBufferLen, unsigned int
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, wieviele Bits im Adressbyte (bzw. Addresswort) für die Adressierung verwendet werden.

Die Bits 8-10 geben an, wieviele 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 Speichergöße eines EEPROM 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 ingesamt 512 Bytes adressiert werden.

Rückgabewert

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

Beschreibung

Die `I2CReadEEPROM` liest Daten von einem seriellen I2C-EEPROM.

Der Zugriff auf eine Speicheradresse auf einem I2C-EEPROM wird prinzipell 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*

I2CWriteEEPROM

I2CWriteEEPROM — Beschreibt ein serielles EEPROM am I2C-Bus.

Syntax

#include <AnaGateDllI2C.h>

```
int I2CWriteEEPROM(int hHandle, unsigned short nSubAddress, unsigned int nOffset, const char * pcBuffer, int nBufferLen, unsigned int 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 soll.
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, wieviele Bits im Adressbyte (bzw. Addresswort) für die Adressierung verwendet werden. Die Bits 8-10 geben an, wieviele 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 Speichergöße eines EEPROM 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 ingesamt 512 Bytes adressiert werden.

Rückgabewert

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

Beschreibung

Die `I2CWriteEEPROM` schreibt Daten auf ein serielles I2C-EEPROM.

Der Zugriff auf eine Speicheradresse auf einem I2C-EEPROM wird prinzipell 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 EEPROM's 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;
   unsigned short nSubAddress = 0; 1
   unsigned int   nOffsetFormat = 0x10|0x0F; 2

   int nRC = I2COpenDevice(&hHandle, 400000, "192.168.0.254", 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.

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 Byte, im vorliegenden Fall werden alle Seiten komplett über eine for-Schleife programmiert.

Siehe auch

I2CReadEEPROM

Anhang C, *Programmierung von I2C EEPROM*

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 Betriebsystemen bzw. der Einsatz von unterschiedlichen Compilern entsprechend angepasst werden.

Windows-Betriebssysteme

Für den Zugriff auf die Funktionen der API stehen für C/C++-Programmierer prinzipell 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 VC7 sind im Lieferumfang enthalten und können direkt verwendet werden, so lautet die Importbiblitothek von `AnaGateCAN.dll` z.B. `AnaGateCANDll.lib`.

  The import libaries for MS VC7 are included and can be used directly. The libraries are named after their corresponding DLL's, so the name of the import library of `AnaGateCAN.dll` is for example `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-Funktionen 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(int nIdentifier, const char * pcBuffer, int nBufLen, int nFlags, int nHandle)
{
    std::cout << "CAN-ID=" << nIdentifier << ", Data=";
    for ( int i = 0; i < nBufLen; i++ )
    {
```

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

int main( )
{
   int hHandle = NULL;

   // opens AnaGate CAN duo device on port A, timeout after 1000 millseconds
   int nRC = CANOpenDevice(&hHandle,FALSE,TRUE,0,"192.168.2.1",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 Bibliothken mit den Zugriffsfunkionen für die AnaGate-Hardware auch in ein speziellen Version für die Programmierung von VB6-Anwendungen zru 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 exact 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:

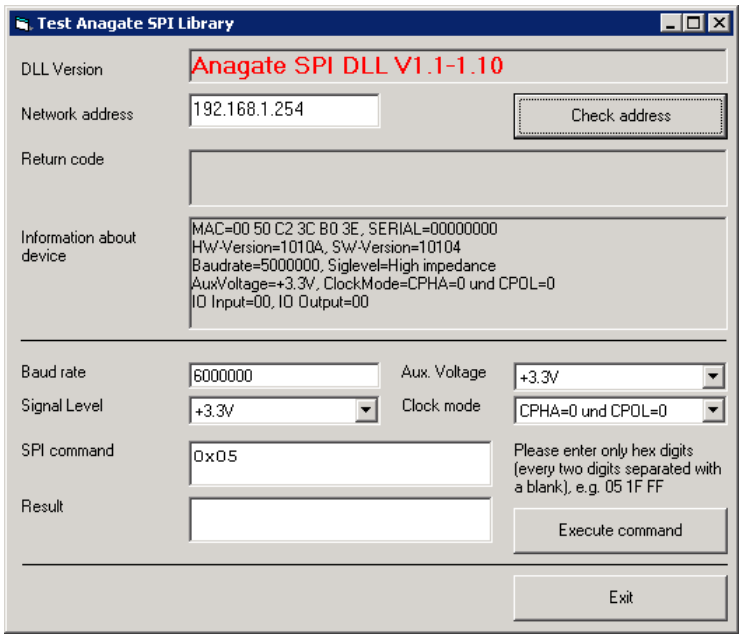
- Auslesen der globalen Geräteeinstellungen (z.B. Baudrate)
- Ausführen eines einzelnen Kommandos auf dem SPI-Bus

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 anschliesend 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`").
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 vollduplex 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. Gegebenfalls 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 Dateiauschnitt 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 es 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
   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
             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 Einstellung 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 Plausibiltät 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 ansich 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-Funktionen empfangene CAN-Telegramme verarbeitet werden können.

```
Declare Function CANOpenDevice Lib "AnaGateCAN" (ByRef Handle As Int32, _
                                                 ByVal ConfirmData As Int32, _
                                                 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 Integer = 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, "192.168.2.1", 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
         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 es Return-Codes ermittelt.

2. `SPISetGlobals` setzt die globalen Einstellungen auf dem Gerät (Baudrate, Signalspannung, Hilfspannung, Clock-Modus).
3. Mit `SPIDataReq` werden Daten zum SPI-Bus gesendet. Die zurückgelesenen Daten werden im Empfangsbuffer 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 einen ensprechenden 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, _
                                                        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
```


Teil II. Skriptspache LUA

Inhaltsverzeichnis

Kapitel 8. 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 und 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 Skriptspache *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. Anschliesend 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* ([LuaProg2006-EN]) als gedruckte Nachschlagewerke empfohlen. Das letztgenannte Paperback kann auch in einer deutschen Übersetzung *Programmieren in Lua* ([LuaProg2006-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

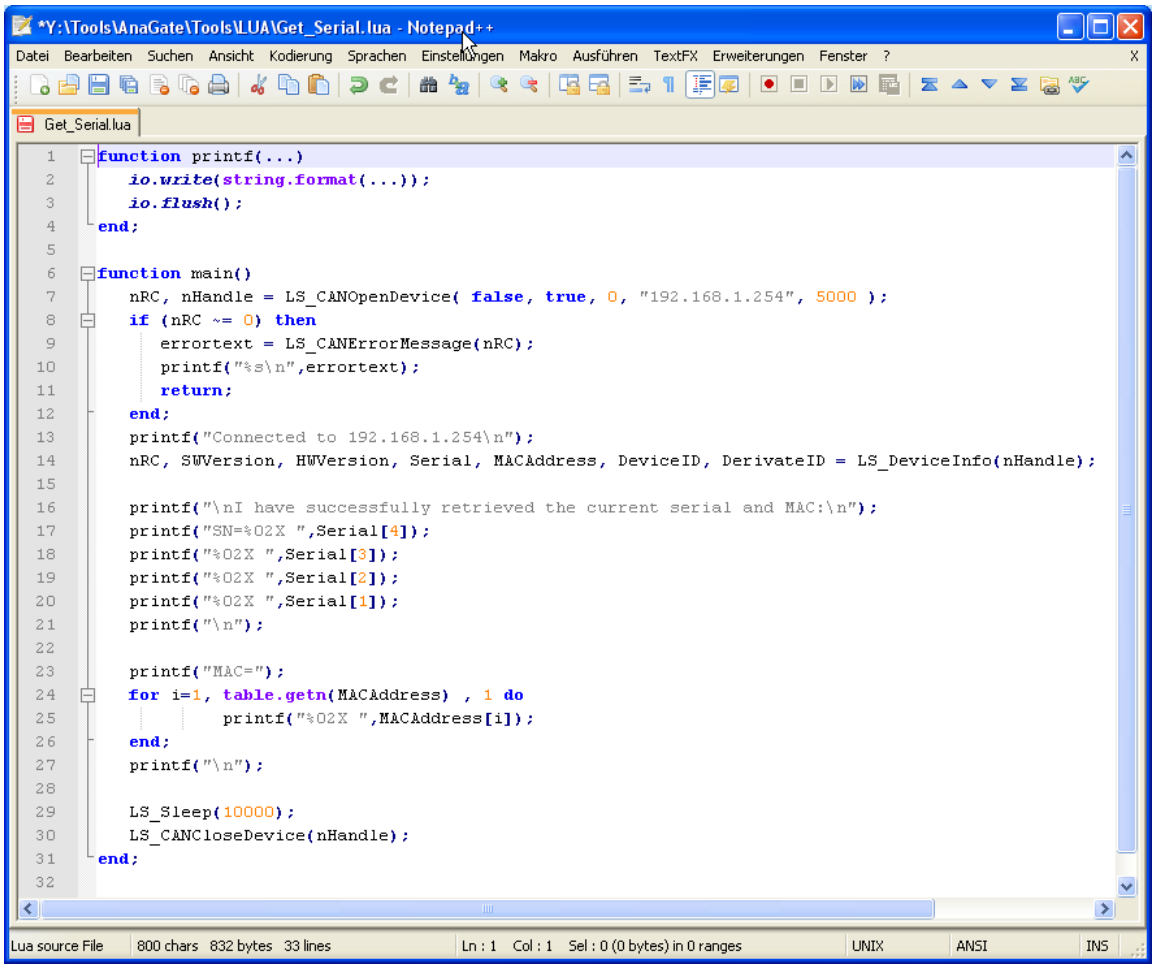
steht auch als Online-Version unter Lua.org [http://www.lua.org] zur Verfügung.

8.1. Skriptdateien erstellen

Die Erstellung und Bearbeitung von Skripten für die Skriptsprache *LUA* ist denkbar einfach und kann über einen herkömmlichen Texteditor erfolgen. Unter Windowsbetriebssystem 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 8.1. 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.

8.2. 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.

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 Kommondozeilen-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-Kommandozeilen-Eingabe 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>
```

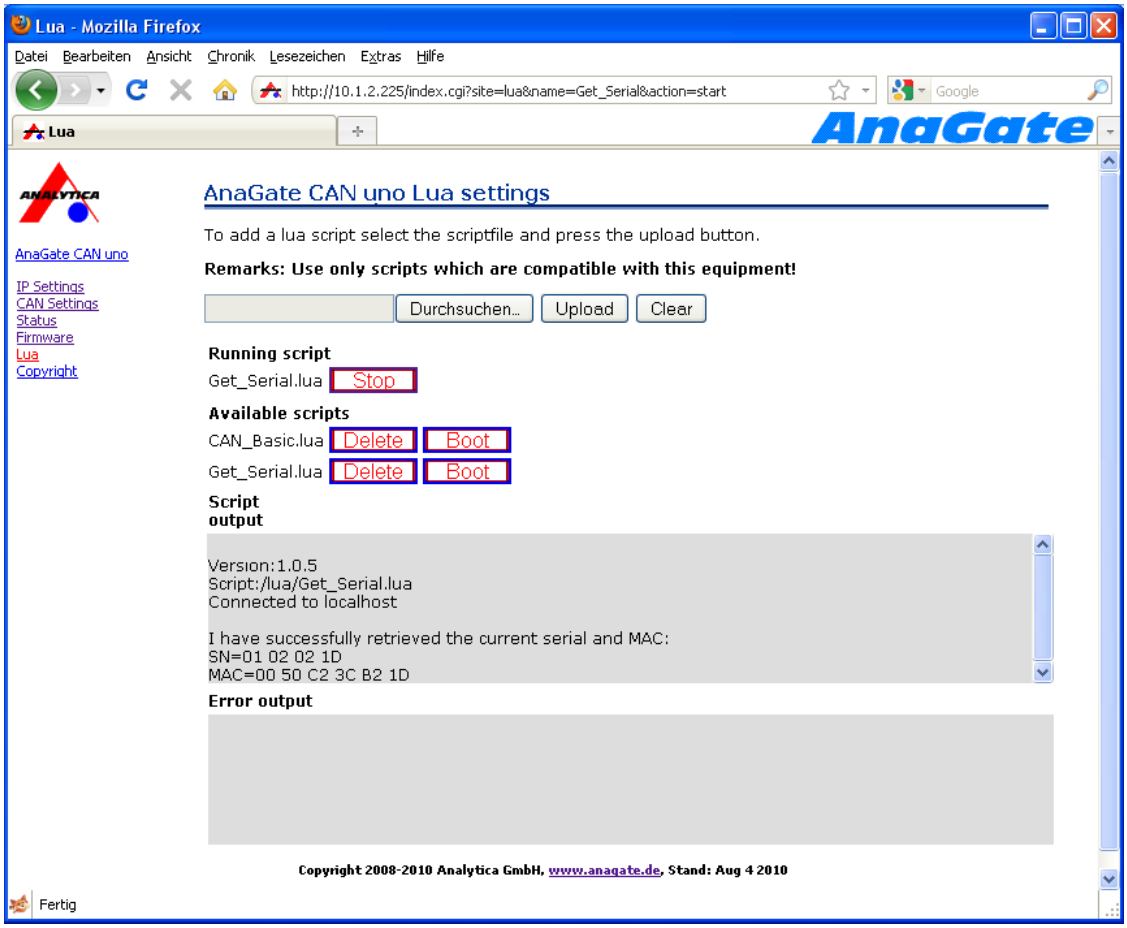
1. Der Name der auszuführenden Skriptdatei muß dem Interpreter als Parameter übergeben werden.
2. Die Seriennummer und MAC-Adresse eines Gerätes mit IP-Addresse `192.168.1.254` wird ermittelt und über die Standardausgabe ausgegeben.

8.3. 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 Beispiele eines *AnaGate CAN uno* demonstriert.

Abbildung 8.2. 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 Schaltfäche **Start**. Über die Schaltfläche **Delete** kann ein Skript vom Gerät gelöscht werden und über **Boot** wird das entsprechend Skript als Boot-Skript definiert.

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 beiden Textbereiche manuell aktualisiert werden.

Kapitel 9. Allgemeine Funktionen

LS_DeviceInfo

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

Syntax

```
int RC, int nSWVersion, int nHWVersion, table(4) tabSerial, table(6) tabMACAddress, int nDeviceID, int nSWDerivateID = LS_DeviceInfo(int hHandle);
```

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*).

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

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

tabSerial — Seriennummer der AnaGate-Hardware (4 Byte).

tabMACAddress — 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

nSWDerivateID — 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

LS_GetTime

LS_GetTime — Gibt die aktuelle Systemzeit zurück.

Syntax

```
int RC, table(2) tabTime = LS_GetTime(void);
```

Parameter

Die Funktion besitzt keine Übergabeparameter.

Rückgabewerte

RC	Die Funktion gibt im Erfolgsfall `0` zurück, andernfalls einen Fehlercode (Tabelle A.5, „Rückgabewerte für LUA Scripting“).
tabTime(1), tabTime(2)	*`tabTime(1)`* gibt die Anzahl der Sekunden, die seit dem 01.01.1970 vergangen sind an. In *`tabTime(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.

LS_Sleep

LS_Sleep — Wartet die angegebene Zeit in Millisekunden.

Syntax

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

Parameter

nMilliseconds Gibt die zu wartende Zeit in Millisekunden an.

Rückgabewert

RC Die Funktion gibt im Erfolgsfall `0` zurück, andernfalls einen Fehlercode (Tabelle A.5, „Rückgabewerte für LUA Scripting“).

Beschreibung

Wartet die angegebene Zeit in Millisekunden.

Kapitel 10. 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

Anmerkung

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

LS_CANOpenDevice

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

Syntax

```
int RC, int Handle = LS_CANOpenDevice(bool bSendDataConfirm, bool bSendDataInd, int nCANPort, string sIPAddress, int nTimeout );
```

Parameter

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 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)

sIPAddress — Netzwerkadresse des AnaGate Partners.

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

Ein Timeout wird festgestellt, wenn der 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.

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

Wichtig

Das explizite Schliesen der Verbindung ist notwendig, um die in der DLL angelegten Systemresourcen 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 intialen Zugriff auf eine Schnittstelle.

```
-- open: use no confirmations and receive incomping CAN data
nRC, hHandle = LS_CANOpenDevice( false, true, 0, "192.168.0.254", 5000);
if ( nRC == 0 ) then
   -- now do something
   LS_CANCloseDevice(hHandle);
end;
```

Siehe auch

LS_CANCloseDevice

LS_CANRestartDevice

LS_CANCloseDevice

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

Syntax

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

Parameter

hHandle 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 *`hHandle`* ist ein Rückgabewert aus einem vorangegangenen erfolgreichen Aufruf der Funktion `LS_CANOpenDevice`.

Wichtig

Das explizite Schliesen der Verbindung ist notwendig, um die in der DLL angelegten Systemresourcen 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

LS_CANRestartDevice

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

Syntax

```
int RC = LS_CANRestartDevice(string sIPAddress, int nTimeout );
```

Parameter

sIPAddress 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 schliest 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

LS_CANSetGlobals

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

Syntax

```
int RC = LS_CANSetGlobals(int hHandle, int nBaudrate, int nOperatingMode, bool bTermination, bool bHighSpeedMode, bool bTimeStampOn);
```

Parameter

hHandle — Gültiges Zugriffs-Handle eines erfolgreichen Aufrufs von `LS_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

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

- 0 = für Standard-Modus.
- 1 = für LoopBack-Modus: In diesem Modus werden alle über das AnaGate CAN versendete Nachrichten zusätzlich an die verbundenen Partner als Data Indications versendet.
- 2 = für 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).

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 `LS_CANSetFilter` definierten Filter werden abgeschaltet.

bTimeStampOn Aktuelle Einstellung für den Zeitstempel-Modus (`true`= ein, `false`= aus). Diese Einstellung wird nur 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 angibt. 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.

Siehe auch

LS_CANGetGlobals

LS_CANGetGlobals

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

Syntax

```
int RC, int nBaudrate, int nOperatingMode, bool bTermination, bool
bHighSpeedMode, bool bTimeStampOn = LS_CANGetGlobals(int hHandle);
```

Parameter

hHandle 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.

nOperatingMode — Aktuell eingestellter Betriebsmodus. Folgende Werte sind möglich:

- 0 = für Standard-Modus.
- 1 = für LoopBack-Modus: In diesem Modus werden alle über das AnaGate CAN versendete Nachrichten zusätzlich an die verbundenen Partner als Data Indications versendet.
- 2 = für 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).

bTermination — Aktuelle Einstellung für CAN-Bus-Terminierung (`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 `LS_CANSetFilter` definierten Filter werden abgeschaltet.

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.

Siehe auch

LS_CANSetGlobals

LS_CANWrite

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

Syntax

```
RC = CANWrite(int hHandle, int nCANId, int nDataLen, table (nDataLen) tabData, int nFlags);
```

Parameter

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

nCANId — 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.

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

tabData — Datenpuffer mit den Telegrammdaten.

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*).

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 Geräte 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

die Länge des Puffers entsprechend der zu erwartenden Datenlänge anzugeben.

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

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

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

nRC, hHandle = LS_CANOpenDevice(, true, true, 0, "192.168.0.254", 5000);
if ( nRC == 0 ) then
   // send 8 bytes with CAN id 37
   nRC = LS_CANWrite( hHandle, nCANId, 8, tabData, nFlags );

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

   LS_CANCloseDevice(hHandle);
end;
```

Bemerkung

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

Siehe auch

LS_CANWriteEx

LS_CANWriteEx

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

Syntax

```
RC, int nSeconds, int nMicroseconds = CANWriteEx(int hHandle, int nCANId,
int nDataLen, table (nDataLen) tabData, int nFlags);
```

Parameter

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

nCANId — 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.

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

tabData — Datenpuffer mit den Telegrammdaten.

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*).

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

nMicroseconds — Anteil der Microsekunden 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 Geräte 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 die Länge des Puffers entsprechend der zu erwartenden Datenlänge anzugeben.

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

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

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

nRC, hHandle = LS_CANOpenDevice(, true, true, 0, "192.168.0.254", 5000);
if ( nRC == 0 ) then
   // send 8 bytes with CAN id 37
   nRC, nSeconds, nMicroSeconds = LS_CANWriteEx( hHandle, nCANId, 8, tabData, nFlags );

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

   LS_CANCloseDevice(hHandle);
end;
```

Bemerkung

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

Siehe auch

LS_CANWrite

LS_CANSetCallback

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

Syntax

```
int RC = LS_CANSetCallback(int hHandle, string sCallbackFunction);

function MY_LS_CALLBACK(int nCANId, int nDataLen, table(nDataLen)
tabData, int nFlags, int hHandle, int nSeconds, int nMicroseconds);
```

Parameter

hHandle	Gültiges Zugriffs-Handle eines erfolgreichen Aufrufs von `LS_CANOpenDevice`.
sCallbackFunction	Name der selbstdefinierten Callback-Funktion. Zum Deaktivieren der Callback-Funktionalität, ist dieser Parameter auf "" zu setzen. Der Aufbau der Funktionsparameter ist unter *Callback-Parameter* dokumentiert.

Callback-Parameter

nCANId	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.
nDataLen	Länge des Datenpuffers (0-8 Bytes).
tabData	Datenpuffer mit den Telegrammdaten.
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.
hHandle	Gültiges Zugriffs-Handle eines erfolgreichen Aufrufs von `LS_CANOpenDevice`.
nSeconds	Zeitstempel der Quittung vom CAN-Controller in Sekunden seit dem 01.01.1970.
nMicroseconds	Anteil der Microsekunden am Zeitstempel.

Rückgabewert

RC Die Funktion gibt im Erfolgsfall `0` 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. Alternativ können die eingehenden CAN-Telegramme auch ohne Callback-Funktion mittels der Funktion `LS_CANGetMessage` abgerufen werden.

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

```
function MyCallback(nID, nLen, tabData, nFlags, nHandle, nSecond, nMSecond)
   io.write(string.format("%.3X::%.3X,%d", nHandle, nID, nLen));
   for i=1, nLen, 1 do
      io.write(string.format(%.2X ", tabData[i]));
   end;
   io.write("\n");
   io.flush();
end;

function main()
    nRC, nHandle = LS_CANOpenDevice( false, true, 0, "127.0.0.1", 5000 );
    if (nRC ~= 0) then
        errortext = LS_CANErrorMessage(nRC);
        printf("%s\n",errortext);
        return;
    end;
    -- set globals: 100kBit, standard mode, termination off, no highspeed, no timestamp
    nRC = LS_CANSetGlobals( nHandle, 100000, 0, true, false, false );

    nRC = LS_CANSetCallback( nHandle, "MyCallback");

    while true do   -- forever
        LS_Sleep(500);
    end;

    nRC = LS_CANSetCallback( hHandle, "" );

    LS_CANCloseDevice(nHandle);
end;
```

Siehe auch

LS_CANWrite, LS_CANWriteEx, LS_CANGetMessage

LS_CANGetMessage

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

Syntax

```
int nAvail, int nCANId, int nDataLen, table() tabData, int nFlags,
int nSeconds, int nMicroseconds = LS_CANGetMessage(int hHandle, int
nTimeout);
```

Parameter

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

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

Rückgabewerte

nAvail Anzahl der Nachrichten, die aktuell noch im Nachichtenpuffer zum Abruf vorhanden sind. Ist aktuell keine Nachricht verfügbar enthält der Rückgabewert die Konstante `-10` (ERR_NO_DATA).

nCANId 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.

nDataLen Länge des Datenpuffers.

tabData Datenpuffer mit den Telegrammdaten.

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.

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

nMicroseconds Anteil der Microsekunden am Zeitstempel.

Beschreibung

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

Über den Parameter *nTimeout* 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 *nAvail* den Rückgabewert -10 (ERR_NO_DATA) zurück.

Warnung

Bei Verwendung einer eigenen Callback-Funktion (siehe LS_CANSetCallback), die beim Empfang eines CAN-Telegramms aufgerufen wird, wird der interne Messagepuffer nicht befüllt. In diesem Fall kann auch keine Nachricht mittels LS_CANGetMessage abgerufen werden.

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

```
nRC, hHandle = LS_CANOpenDevice(true, true, 0, "192.168.0.254", 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);
    nCurMsg = 0

    repeat
        nAvail, ID, Len, Data, 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) );  -- 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 *nSeconds* und *nMicroseconds* werden nicht gesetzt.

Siehe auch

LS_CANSetCallback

LS_CANSetFilter

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

Syntax

```
int RC = LS_CANSetFilter(int hHandle, table(16)tabFilter);
```

Parameter

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

tabFilter 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 höchstmöglich 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 jeweilegen Verbindungspartner weitergeleitet.

Ein Maskenfilter besteht aus der Filtermaske, die angibt, welche Bits des CAN-Identifier geprüft werden sollen, und des entsprechenden Filterwertes. 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.

Filter sind grundsätzlich deaktiviert, falls der Parameter *bSendDataInd* beim Ausführen der Funktion `LS_CANOpenDevice` gesetzt wurde.

Siehe auch

LS_CANGetFilter

LS_CANGetFilter

LS_CANGetFilter — Liefert die Filtereinstellungen für die aktuelle Verbindung zurück.

Syntax

```
int RC, table(16) tabFilter = LS_CANGetFilter(int hHandle);
```

Parameter

hHandle 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*).

tabFilter Zeiger auf ein Feld mit 8 Fitereinträgen (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 Bereichfilterwerten 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 jeweilegen Verbindungspartner weitergeleitet.

Siehe auch

LS_CANSetFilter

LS_CANSetTime

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

Syntax

```
int RC = LS_CANSetTime(int hHandle, long nSeconds, long 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

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 zurückübermittelt, 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.

LS_CANErrorMessage

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

Syntax

```
string sErrorMsg LS_CANErrorMessage(int nRetCode);
```

Parameter

nRetCode Fehlercode, dessen Fehlerbeschreibung ermittelt werden soll.

Rückgabewert

sErrorMsg 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.

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

//... call a API function here

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


LS_CANReadDigital

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

Syntax

`int RC, int nInputBits, int nOutputBits = **LS_CANReadDigital**(int *hHandle*);`

Parameter

hHandle 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*).

nInputBits Zeiger auf den aktueller Inhalt des Registers mit den digitalen Eingängen. Dabei entsprechen Bit 0 bis Bit 3 den vier digitalen Eingängen und Bit 4 bis Bit 31 sind auf 0 gesetzt.

nOutputBits Zeiger auf den aktueller Inhalt des Registers mit den digitalen Ausgängen. Dabei entsprechen Bit 0 bis Bit 3 den vier digitalen Ausgängen und 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.

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

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

```
nOutputs = 0x03;

nRC, hHandle = LS_CANOpenDevice( 400000, "192.168.0.254", 5000 );
if ( nRC == 0 ) then
   // set the digital output register (PIN 0 and PIN 1 to HIGH value)
   nRC = LS_CANWriteDigital( hHandle, nOutputs );

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

   LS_CANCloseDevice(hHandle);
end;
```

Siehe auch

LS_CANWriteDigital

LS_CANWriteDigital

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

Syntax

```
int RC = LS_CANWriteDigital(int hHandle, int nOutputBits);
```

Parameter

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

nOutputBits — Neuer Wert des Registers mit den digitalen Ausgängen. Dabei entsprechen Bit 0 bis Bit 3 den vier digitalen Ausgängen und 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.

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

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

Siehe auch

LS_SPIReadDigital

Kapitel 11. 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.

LS_SPIOpenDevice

LS_SPIOpenDevice — Baut eine Netzwerkverbindung zu einem AnaGate SPI auf.

Syntax

```
int RC, int Handle = LS_SPIOpenDevice(string sIPAddress, int nTimeout);
```

Parameter

sIPAddress — Netzwerkadresse des AnaGate Partners.

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

Ein Timeout wird festgestellt, wenn der 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 intialen Zugriff auf das Gerät.

```
nRC, nHandle  = LS_SPIOpenDevice( "192.168.0.254", 5000 );
if (nRC ~= 0) then
   print(LS_SPIErrorMessage(nRC));
   exit();
end;

-- now do something

LS_SPICloseDevice(nHandle);
```


Siehe auch

LS_SPICloseDevice

LS_SPICloseDevice

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

Syntax

```
int RC = LS_SPICloseDevice(int hHandle);
```

Parameter

hHandle 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 *`hHandle`* ist ein Rückgabewert aus einem vorangegangenen erfolgreichen Aufruf der Funktion `LS_SPIOpenDevice`.

Wichtig

Das explizite Schliesen der Verbindung ist notwendig, um die in der DLL angelegten Systemresourcen 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

LS_SPISetGlobals

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

Syntax

```
int LS_SPISetGlobals(int hHandle, int nBaudrate, unsigned char nSigLevel, unsigned char nAuxVoltage, unsigned char nClockMode);
```

Parameter

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

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

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

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 +5.0V für die Signale.
- 2 für +3.3V für die Signale.
- 3 für +2.5V für die Signale.

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

- 0 für Hilfsspannung +3.3V.
- 1 für Hilfsspannung +2.5V für die Signale.

nClockMode — Gibt die Phase und die Polarität der Clock-Leitung für die Datenübertragung an. 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

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

LS_SPIGetGlobals

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

Syntax

```
int RC, int nBaudrate, int nSigLevel, int nAuxVoltage, int nClockMode
= LS_SPIGetGlobals(int hHandle);
```

Parameter

hHandle 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*).

nBaudrate Aktuell auf dem SPI-Bus eingestellte Baudrate in kBit.

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

- 0 für Ausgänge im High Impedance Modus (Standard-Modus).
- 1 für +5.0V für die Signale.
- 2 für +3.3V für die Signale.
- 3 für +2.5V für die Signale.

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

- 0 für Hilfsspannung +3.3V.
- 1 für Hilfsspannung +2.5V für die Signale.

nClockMode 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.

Siehe auch

LS_SPISetGlobals

LS_SPIDataReq

LS_SPIDataReq — Führt einen Datentransfer auf dem SPI Bus durch.

Syntax

```
int RC, table(nReadLen) tabRead = LS_SPIDataReq(int hHandle, int nWriteLen, int nReadLen, table(nWriteLen) tabWrite);
```

Parameter

hHandle	Gültiges Zugriffs-Handle eines erfolgreichen Aufrufs von `LS_SPIOpenDevice`.
nWriteLen	Länge des Datenpuffers *pcBufWrite* (Anzahl Bytes).
nReadLen	Anzahl der Bytes, die gelesen werden sollen. Darf die Länge des Datenpuffers *pcBufReda* nicht überschreiten.
tabWrite	Puffer mit den Daten, die an den SPI Partner gesendet werden sollen.

Rückgabewert

RC	Die Funktion gibt im Erfolgsfall `0` zurück, andernfalls einen Fehlercode (Anhang A, *Rückgabewerte aus den API Funktionen*).
tabRead	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 `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 den SPI Bus sendet und empfängt.

```
tabWrite = {};
for i=1, 10 , 1 do
```



```
   table.insert(tabWrite, i );
end;

nRC, hHandle = SPIOpenDevice("192.168.1.254", 5000);
if ( nRC == 0 ) then
   // send 1 byte and receive 1 byte
   nRC, tabRead = LS_SPIDataReq( hHandle, 1, 1, tabWrite );
   // send 1 byte and receive 5 byte
   nRC, tabRead = LS_SPIDataReq( hHandle, 1, 5, tabWrite );
   // send 2 byte and receive 1 byte
   nRC, tabRead = LS_SPIDataReq( hHandle, 2, 1, tabWrite );

   LS_SPICloseDevice(hHandle);
end;
```


LS_SPIErrorMessage

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

Syntax

```
string sErrorMsg LS_SPIErrorMessage(int nRetCode);
```

Parameter

nRetCode Fehlercode, dessen Fehlerbeschreibung ermittelt werden soll.

Rückgabewert

sErrorMsg 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.

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

//... call a API function here

sErrorText = LS_SPIErrorMessage(nRC);
print(sErrorText);
```


LS_SPIReadDigital

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

Syntax

`int RC, int nInputBits, int nOutputBits = **LS_SPIReadDigital**(int *hHandle*);`

Parameter

hHandle 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*).

nInputBits Zeiger auf den aktueller Inhalt des Registers mit den digitalen Eingängen. Dabei entsprechen Bit 0 bis Bit 3 den vier digitalen Eingängen und Bit 4 bis Bit 31 sind auf 0 gesetzt.

nOutputBits Zeiger auf den aktueller Inhalt des Registers mit den digitalen Ausgängen. Dabei entsprechen Bit 0 bis Bit 3 den vier digitalen Ausgängen und 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.

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

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

```
nOutputs = 0x03;

nRC, hHandle = LS_SPIOpenDevice( 400000, "192.168.0.254", 5000 );
if ( nRC == 0 ) then
   // set the digital output register (PIN 0 and PIN 1 to HIGH value)
   nRC = LS_SPIWriteDigital( hHandle, nOutputs );

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

   LS_SPICloseDevice(hHandle);
end;
```

Siehe auch

LS_SPIWriteDigital

LS_SPIWriteDigital

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

Syntax

```
int RC = LS_SPIWriteDigital(int hHandle, int nOutputBits);
```

Parameter

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

nOutputBits — Neuer Wert des Registers mit den digitalen Ausgängen. Dabei entsprechen Bit 0 bis Bit 3 den vier digitalen Ausgängen und 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.

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

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

Siehe auch

LS_SPIReadDigital

Kapitel 12. 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

LS_I2COpenDevice

LS_I2COpenDevice — Baut eine Netzwerkverbindung zu einem AnaGate I2C (bzw. AnaGate Universal Programmer) auf.

Syntax

```
int RC, int Handle = LS_I2COpenDevice(unsigned int nBaudrate, string sIPAddress, int nTimeout);
```

Parameter

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

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

Anmerkung

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

sIPAddress — Netzwerkadresse des AnaGate Partners.

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

Ein Timeout wird festgestellt, wenn der 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

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

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

```
nRC, nHandle  = LS_I2COpenDevice( 1000000, "192.168.0.254", 5000 );
if (nRC ~= 0) then
   print(LS_I2CErrorMessage(nRC));
   exit();
end;

-- now do something

LS_I2CCloseDevice(nHandle);
```

Siehe auch

LS_I2CCloseDevice

LS_I2CCloseDevice

LS_I2CCloseDevice — Schließt eine geöffnete Netzwerk-Verbindung zu einem AnaGate I2C Device.

Syntax

```
int RC = LS_I2CCloseDevice(int hHandle);
```

Parameter

hHandle 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 *`hHandle`* ist ein Rückgabewert aus einem vorangegangenen erfolgreichen Aufruf der Funktion `LS_I2COpenDevice`.

Wichtig

Das explizite Schliesen der Verbindung ist notwendig, um die in der DLL angelegten Systemresourcen 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

LS_I2CReset

LS_I2CReset — Setzt den I2C-Controller auf dem AnaGate I2C Device zurück.

Syntax

```
int RC = LS_I2CReset(int hHandle);
```

Parameter

hHandle 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.

LS_I2CRead

LS_I2CRead — Liest Daten von einem I2C Partner.

Syntax

```
int RC, table(nBufferLen) tabBuffer = LS_I2CRead(int hHandle, unsigned short nSlaveAddress, int nBufferLen);
```

Parameter

hHandle	Gültiges Zugriffs-Handle eines erfolgreichen Aufrufs von `LS_I2COpenDevice`.
nSlaveAddress	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*).
nBufferLen	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*).
tabBuffer	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

LS_I2CWrite

LS_I2CWrite — Schreibt Daten zu einem I2C Partner.

Syntax

```
int RC, int ErrorByte = LS_I2CWrite(int hHandle, unsigned short nSlaveAddress, table(nBufferLen) tabBuffer, int nBufferLen);
```

Parameter

hHandle	Gültiges Zugriffs-Handle eines erfolgreichen Aufrufs von `I2COpenDevice`.
nSlaveAddress	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*).
nBufferLen	Anzahl von Datenbytes, die gelesen werden sollen.
tabBuffer	Byte-Puffer mit den Daten, die an den I2C Partner gesendet 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

LS_I2CReadDigital

LS_I2CReadDigital — Liest die aktuellen Werte der digitale Ein-/Ausgabe-Register der AnaGate Hardware zurück.

Syntax

`int RC, int nInputBits, int nOutputBits = **LS_I2CReadDigital**(int *hHandle*);`

Parameter

hHandle 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*).

nInputBits Zeiger auf den aktueller Inhalt des Registers mit den digitalen Eingängen. Dabei entsprechen Bit 0 bis Bit 3 den vier digitalen Eingängen und Bit 4 bis Bit 31 sind auf 0 gesetzt.

nOutputBits Zeiger auf den aktueller Inhalt des Registers mit den digitalen Ausgängen. Dabei entsprechen Bit 0 bis Bit 3 den vier digitalen Ausgängen und 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.

Die aktuellen Zustände der digitalen Eingänge und Ausgänge können mit der Funktion `LS_I2CReadDigital` ermittelt werden.

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

```
nOutputs = 0x03;

nRC, hHandle = LS_I2COpenDevice( 400000, "192.168.0.254", 5000 );
if ( nRC == 0 ) then
   // set the digital output register (PIN 0 and PIN 1 to HIGH value)
   nRC = LS_I2CWriteDigital( hHandle, nOutputs );

   // read all input and output registers
   nRC, nInputs, nOutputs = LS_I2CReadDigital( hHandle );

   LS_I2CCloseDevice(hHandle);
end;
```

Siehe auch

LS_I2CWriteDigital

LS_I2CWriteDigital

LS_I2CWriteDigital — Setzt das digitale Ausgabe-Register der AnaGate-Hardware auf einen neuen Wert.

Syntax

```
int RC = LS_I2CWriteDigital(int hHandle, int nOutputBits);
```

Parameter

hHandle — Gültiges Zugriffs-Handle eines erfolgreichen Aufrufs von `LS_I2COpenDevice`.

nOutputBits — Neuer Wert des Registers mit den digitalen Ausgängen. Dabei entsprechen Bit 0 bis Bit 3 den vier digitalen Ausgängen und 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.

Die digitalen Ausgänge können mit der Funktion `LS_I2CWriteDigital` verändert werden.

Ein Programmier-Beispiel zum Lesen/Schreiben der IO ist bei der Beschreibung von `LS_I2CReadDigital` zu finden.

Siehe auch

LS_I2CReadDigital

LS_I2CErrorMessage

LS_I2CErrorMessage — Gibt eine textuelle Beschreibung eines Rückgabewertes aus einer API-Funktion zurück.

Syntax

```
string sErrorMsg LS_I2CErrorMessage(int nRetCode);
```

Parameter

nRetCode Fehlercode, dessen Fehlerbeschreibung ermittelt werden soll.

Rückgabewert

sErrorMsg 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.

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

//... call a API function here

sErrorText = LS_I2CErrorMessage(nRC);
print(sErrorText);
```


LS_I2CReadEEPROM

LS_I2CReadEEPROM — Liest Daten von einem EEPROM am I2C-Bus.

Syntax

```
int RC, tabData(nDatLen) = LS_I2CReadEEPROM(int hHandle, unsigned short nSubAddress, unsigned int nOffset, unsigned int nOffsetFormat, int nDataLen);
```

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.

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, wieviele Bits im Adressbyte (bzw. Addresswort) für die Adressierung verwendet werden.

Die Bits 8-10 geben an, wieviele 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 Speichergöße eines EEPROM 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 ingesamt 512 Bytes adressiert werden.

nDataLenLen Länge des Datenpuffers.

Rückgabewerte

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

tabData Enthält die vom EEPROM gelesenen Daten.

Beschreibung

Die `LS_I2CReadEEPROM` liest Daten von einem seriellen I2C-EEPROM.

Der Zugriff auf eine Speicheradresse auf einem I2C-EEPROM wird prinzipell 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*

LS_I2CWriteEEPROM

LS_I2CWriteEEPROM — Beschreibt ein serielles EEPROM am I2C-Bus.

Syntax

```
int RC = LS_I2CWriteEEPROM(int hHandle, unsigned short nSubAddress,
unsigned int nOffset, unsigned int nOffsetFormat, int nDataLen,
table(nDataLen) tabData);
```

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 soll.

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, wieviele Bits im Adressbyte (bzw. Addresswort) für die Adressierung verwendet werden.

Die Bits 8-10 geben an, wieviele 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 Speichergöße eines EEPROM 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 ingesamt 512 Bytes adressiert werden.

nDataLen	Länge des Datenpuffers.
tabData	Zeichenfolgepuffer mit den Daten, die geschrieben werden sollen.

Rückgabewert

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

Beschreibung

Die `LS_I2CWriteEEPROM` schreibt Daten auf ein serielles I2C-EEPROM.

Der Zugriff auf eine Speicheradresse auf einem I2C-EEPROM wird prinzipell 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 EEPROM's 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.

```
tabData = {};
for i=1, 256 , 1 do
   table.insert(tabData, 0x0 );
end;

nSubAddress = 0; 1
nOffsetFormat = 0x10+0x0F; 2

RC, hHandle = LS_I2COpenDevice(400000, "192.168.0.254", 5000);
if ( RC == 0 ) then
   for page=0, 512-1, 1 do
      RC = LS_I2CWriteEEPROM( hHandle, nSubAddress, i*256, nOffSetFormat, 256, tabData ); 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 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 Byte, im vorliegenden Fall werden alle Seiten komplett über eine for-Schleife programmiert.

Siehe auch

LS_I2CReadEEPROM

Anhang C, *Programmierung von I2C EEPROM*

LS_I2CSequence

LS_I2CSequence — Schreibt eine Sequenz zu einem I2C Partner über das AnaGate I2C Device.

Synopsis

```
int RC, int NumberOfBytesRead, int ByteNumberLastError,
table(NumberOfBytesToRead) ReadData = LS_I2CSequence( int Handle
, int NumberOfBytesToWrite , int NumberOfBytesToRead ,
table(NumberOfBytesToWrite) WriteData );
```

Beschreibung

Parameter

int Handle	Gültiges Zugriffs-Handle eines erfolgreichen Aufrufs von I2COpenDevice.
int NumberOfBytesToWrite	Gesamtanzahl der Bytes die in dieser Sequenz geschrieben werden sollen.
int NumberOfBytesToRead	Gesamtanzahl der Bytes die in dieser Sequenz gelesen werden sollen.
table(NumberOfBytesToWrite) WriteData	Zeichenfolgepuffer mit den Daten der Sequence.

Rückgabewerte

int RC	Treten keine Fehler auf, wird Null zurückgegeben. Falls die Funktion fehlschlägt, ist der Returncode ungleich Null.
int NumberOfBytesRead	Anzahl der tatsächlich gelesenen Bytes.
int ByteNumberLastError	Nummer des Bytes an dem der letzte Fehler auftrat.
table (NumberOfBytesRead) ReadData	Zeichenfolgepuffer mit den gelesenen Daten des I2C Partners.

Kapitel 13. CANOpen Funktionen

CANopen® ist ein auf CAN basierendes Kommunikationsprotokoll, das zur Vernetzung und Steuerung von intelligenten Geräten, insbesondere in der Automatisierung eingesetzt wird. Es wird von der Organisation CAN in Automation (**CiA**) gepflegt und ist als Europäische Norm `EN 50325-4` standardisiert (siehe [CiA-DS301]).

Nutzer der *AnaGate API* können auf eine einfache Schnittstelle zurückgreifen, die als CANopen-Master auf die im nachfolgenden beschriebenen CANopen-Dienste zur Verfügung stellt.

LS_CANopenSetConfig

LS_CANopenSetConfig — Konfiguriert die verbindungsspezifischen Geräteeinstellungen für den CANOpen-Betrieb. Muß am Anfang einmal Aufgerufen werden um die CANOpen Funktionalität einzuschalten.

Synopsis

```
int RC = LS_CANopenSetConfig( int Handle , int CANOpenConfig );
```

Beschreibung

Parameter

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

int CANOpenConfig — Bit 0:

0: Die CANOpen Funktionalität wird nicht unterstützt (Standard). Ein Aufruf einer CANOpen Funktion wird negativ (FFh) quittiert. Empfangene CAN-Daten werden immer als Standard DataIndication (OP_ANAGATE_CAN_DATA_IND) gesendet.

1: Die CANOpen Funktionalität ist eingeschaltet und es können die nachfolgend beschriebenen Funktionen verwendet werden.

Rückgabewerte

int RC — Treten keine Fehler auf, wird Null zurückgegeben. Falls die Funktion fehlschlägt, ist der Returncode ungleich Null.

LS_CANopenGetConfig

LS_CANopenGetConfig — Liest die verbindungsspezifischen Geräteeinstellungen des CANOpen-Betriebs aus.

Synopsis

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

Beschreibung

Parameter

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

Rückgabewerte

int RC — Treten keine Fehler auf, wird Null zurückgegeben. Falls die Funktion fehlschlägt, ist der Returncode ungleich Null.

int CANOpenConfig — Bit 0:

0: Die CANOpen Funktionalität wird nicht unterstützt (Standard). Ein Aufruf einer CANOpen Funktion wird negativ (FFh) quittiert. Empfangene CAN-Daten werden immer als Standard DataIndication (OP_ANAGATE_CAN_DATA_IND) gesendet.

1: Die CANOpen Funktionalität ist eingeschaltet und es können die nachfolgend beschriebenen Funktionen verwendet werden.

LS_CANopenSetSYNCMode

LS_CANopenSetSYNCMode — Dies Funktion wird benötigt um den periodischen SYNC-Modus. In dem periodischen SYNC-Modus werden vom AnaGate in dem festgelegten Intervall SYNC-Nachrichten erzeugt und auf dem CAN-Bus versendet. Um den periodischen SYNC-Modus abzuschalten muss die Funktion mit PeriodTime=0 aufgerufen werden.

Synopsis

```
int RC = LS_CANopenSetSYNCMode( int Handle , int PeriodTime );
```

Beschreibung

Parameter

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

int PeriodTime
- = 0: periodischer SYNC-Modus wird abgeschaltet.
- > 0: SYNC-Modus wird mit angegebenem Intervall in ms gestartet.

Rückgabewerte

int RC — Treten keine Fehler auf, wird Null zurückgegeben. Falls die Funktion fehlschlägt, ist der Returncode ungleich Null.

LS_CANopenSetCallbacks

LS_CANopenSetCallbacks — Setzt die verschiedenen Callback-Funktion, die beim Eingang von CAN-Telegrammen aufgerufen werden.

Synopsis

```
int RC = LS_CANopenSetCallbacks( int Handle , string CallbackFunctionPDO , string CallbackFunctionSYNC , string CallbackFunctionEMCY , string CallbackFunctionGUARD , string CallbackFunctionUndefined );
```

Beschreibung

Parameter

int Handle	Gültiges Zugriffs-Handle eines erfolgreichen Aufrufs von LS_CANOpenDevice.
string CallbackFunctionPDO	Name einer selbstdefinierten Lua-Callback-Funktion die beim Empfang von PDO Nachrichten aufgerufen wird. Zum deaktivieren der Callback-Funktionalität, ist dieser Parameter auf "" zu setzen. Der Aufbau der Funktionsparameter ist der Callback Definition zu entnehmen.
string CallbackFunctionSYNC	Name einer selbstdefinierten Lua-Callback-Funktion die beim Empfang von SYNC Nachrichten aufgerufen wird. Zum deaktivieren der Callback-Funktionalität, ist dieser Parameter auf "" zu setzen. Der Aufbau der Funktionsparameter ist der Callback Definition zu entnehmen.
string CallbackFunctionEMCY	Name einer selbstdefinierten Lua-Callback-Funktion die beim Empfang von EMCY Nachrichten aufgerufen wird. Zum deaktivieren der Callback-Funktionalität, ist dieser Parameter auf "" zu setzen. Der Aufbau der Funktionsparameter ist der Callback Definition zu entnehmen.
string CallbackFunctionGUARD	Name einer selbstdefinierten Lua-Callback-Funktion die beim Empfang von GUARD Nachrichten aufgerufen wird. Zum deaktivieren der Callback-Funktionalität, ist dieser Parameter auf "" zu setzen. Der Aufbau der Funktionsparameter ist der Callback Definition zu entnehmen.
string CallbackFunctionUndefined	Name einer selbstdefinierten Lua-Callback-Funktion die beim Empfang von nicht zuordenbaren Nachrichten (PDO, SYNC, EMCY bzw. GUARD) aufgerufen wird. Wenn auf dem CAN-Bus z.B. auch noch Standard CAN Nachrichten versendet werden kann es Nachrichten geben die nicht zuordenbar sind. Diese werden dann an diese Callback-

Funktion gesendet. Zum deaktivieren der Callback-Funktionalität, ist dieser Parameter auf "" zu setzen. Der Aufbau der Funktionsparameter ist dem CANWrite zu entnehmen.

Rückgabewerte

int RC Treten keine Fehler auf, wird Null zurückgegeben. Falls die Funktion fehlschlägt, ist der Returncode ungleich Null.

LS_CANopenGetPDO

LS_CANopenGetPDO — Liest eine CANOpen PDO aus dem internen Empfangspuffer aus. Solange die Callbackfunktion für PDO Nachrichten nicht gesetzt wurde, werden die eingehenden PDO Nachrichten in einem internen Puffer zwischengespeichert und müssen mit dieser Funktion einzeln abgepollt werden. Sollte in dem internen Puffer keine Nachricht vorhanden sein wird höchstens die Zeit "Timeout" auf eine eingehende PDO gewartet.

Synopsis

```
int AvailMessages, int NodeID, int PDOTyp, table(1-8) Data, int Seconds,
int Microseconds = LS_CANopenGetPDO( int Handle , int Timeout );
```

Beschreibung

Parameter

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

int Timeout — Zeit in ms wie lange auf eine eingehende PDO gewartet werden soll wenn der interne Puffer für PDO Nachrichten leer ist.

Rückgabewerte

int AvailMessages — Anzahl der Nachrichten die sich noch im Puffer befinden.

- = -2: Es wurden seit dem Start noch keine PDO's empfangen. Die restlichen Rückgabeparameter sind ungültig
- = -1: Es waren keine Nachrichten im Puffer. Die restlichen Rückgabeparameter sind ungültig
- >= 0: Anzahl der noch verfügbaren Nachrichten im Puffer.

int NodeID — CANOpen NodeID des Versenders der PDO Nachricht.. Nur gültig wenn AvailMessages >=0.

int PDOTyp — PDO-Typ (Transmit/Receive ; 1...4) der PDO Nachricht.. Nur gültig wenn AvailMessages >=0.

table(1-8) Data — Daten der PDO Nachricht

int Seconds — Zeit des Empfangs vom CAN-Bus in Sekunden seit dem 01.01.1976

int Microseconds — Zeit des Empfangs vom CAN-Bus: Mikrosekundenanteil

LS_CANopenGetSYNC

LS_CANopenGetSYNC — Liest eine CANOpen SYNC Nachricht aus dem internen Empfangspuffer aus. Solange die Callbackfunktion für SYNC Nachrichten nicht gesetzt wurde, werden die eingehenden SYNC Nachrichten in einem internen Puffer zwischengespeichert und müssen mit dieser Funktion einzeln abgepollt werden. Sollte in dem internen Puffer keine Nachricht vorhanden sein wird höchstens die Zeit "Timeout" auf eine eingehende SYNC Nachricht gewartet.

Synopsis

```
int AvailMessages, int ReturnCode, int Seconds, int Microseconds =
LS_CANopenGetSYNC( int Handle , int Timeout );
```

Beschreibung

Parameter

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

int Timeout — Zeit in ms wie lange auf eine eingehende SYNC Nachricht gewartet werden soll wenn der interne Puffer für SYNC Nachrichten leer ist.

Rückgabewerte

int AvailMessages — Anzahl der Nachrichten die sich noch im Puffer befinden.

- = -2: Es wurden seit dem Start noch keine SYNC Nachrichten empfangen. Die restlichen Rückgabeparameter sind ungültig
- = -1: Es waren keine Nachrichten im Puffer. Die restlichen Rückgabeparameter sind ungültig
- >= 0: Anzahl der noch verfügbaren Nachrichten im Puffer.

int ReturnCode — Rückgabewert beim automatischen Versenden von SYNC Nachrichten. Nur gültig wenn AvailMessages >=0.

int Seconds — Zeit des Empfangs vom CAN-Bus in Sekunden seit dem 01.01.1976. Nur gültig wenn AvailMessages >=0.

int Microseconds — Zeit des Empfangs vom CAN-Bus: Mikrosekundenanteil. Nur gültig wenn AvailMessages >=0.

LS_CANopenGetEMCY

LS_CANopenGetEMCY — Liest eine CANOpen EMCY Nachricht aus dem internen Empfangspuffer aus. Solange die Callbackfunktion für EMCY Nachrichten nicht gesetzt wurde, werden die eingehenden EMCY Nachrichten in einem internen Puffer zwischengespeichert und müssen mit dieser Funktion einzeln abgepollt werden. Sollte in dem internen Puffer keine Nachricht vorhanden sein wird höchstens die Zeit "Timeout" auf eine eingehende EMCY Nachricht gewartet.

Synopsis

```
int AvailMessages, int NodeID, int ErrorCode, int ErrorRegister,
table(5) ErrorDescription, int Seconds, int Microseconds =
LS_CANopenGetEMCY( int Handle , int Timeout );
```

Beschreibung

Parameter

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

int Timeout — Zeit in ms wie lange auf eine eingehende EMCY Nachricht gewartet werden soll wenn der interne Puffer für EMCY Nachrichten leer ist.

Rückgabewerte

int AvailMessages — Anzahl der Nachrichten die sich noch im Puffer befinden.

- = -2: Es wurden seit dem Start noch keine EMCY Nachrichten empfangen. Die restlichen Rückgabeparameter sind ungültig
- = -1: Es waren keine Nachrichten im Puffer. Die restlichen Rückgabeparameter sind ungültig
- >= 0: Anzahl der noch verfügbaren Nachrichten im Puffer.

int NodeID — CANOpen NodeID des Versenders der PDO Nachricht. Nur gültig wenn AvailMessages >=0.

int ErrorCode — ErrorCode der aktuellen Message. Nur gültig wenn AvailMessages >=0.

int ErrorRegister — ErrorRegister der aktuellen Message. Nur gültig wenn AvailMessages >=0.

table(5) ErrorDescritpion — Table mit der Fehlerbeschreibung. Nur gültig wenn AvailMessages >=0.

int Seconds — Zeit des Empfangs vom CAN-Bus in Sekunden seit dem 01.01.1976. Nur gültig wenn AvailMessages >=0.

int Microseconds	Zeit des Empfangs vom CAN-Bus: Mikrosekundenanteil. Nur gültig wenn AvailMessages >=0.

LS_CANopenGetGUARD

LS_CANopenGetGUARD — Liest eine CANOpen GUARD Nachricht aus dem internen Empfangspuffer aus. Solange die Callbackfunktion für GUARD Nachrichten nicht gesetzt wurde, werden die eingehenden GUARD Nachrichten in einem internen Puffer zwischengespeichert und müssen mit dieser Funktion einzeln abgepollt werden. Sollte in dem internen Puffer keine Nachricht vorhanden sein wird höchstens die Zeit "Timeout" auf eine eingehende GUARD Nachricht gewartet.

Synopsis

```
int AvailMessages, int NodeID, int Status, int Seconds, int Microseconds
= LS_CANopenGetGUARD( int Handle , int Timeout );
```

Beschreibung

Parameter

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

int Timeout — Zeit in ms wie lange auf eine eingehende GUARD Nachrichten gewartet werden soll wenn der interne Puffer für GUARD Nachrichten leer ist.

Rückgabewerte

int AvailMessages — Anzahl der Nachrichten die sich noch im Puffer befinden.

- = -2: Es wurden seit dem Start noch keine GUARD Nachrichten empfangen. Die restlichen Rückgabeparameter sind ungültig
- = -1: Es waren keine Nachrichten im Puffer. Die restlichen Rückgabeparameter sind ungültig
- >= 0: Anzahl der noch verfügbaren Nachrichten im Puffer.

int NodeID — CANOpen NodeID des Versenders der PDO Nachricht. Nur gültig wenn AvailMessages >=0.

int Status — Status der aktuellen Message. Nur gültig wenn AvailMessages >=0.

int Seconds — Zeit des Empfangs vom CAN-Bus in Sekunden seit dem 01.01.1976. Nur gültig wenn AvailMessages >=0.

int Microseconds — Zeit des Empfangs vom CAN-Bus: Mikrosekundenanteil. Nur gültig wenn AvailMessages >=0.

LS_CANopenGetUndefined

LS_CANopenGetUndefined — Liest eine Nachricht aus dem internen Empfangspuffer für undefinierte Nachrichten [150] aus. Solange die Callbackfunktion für undefinierte Nachrichten [150] nicht gesetzt wurde, werden die eingehenden undefinierte Nachrichten [150] in einem internen Puffer zwischengespeichert und müssen mit dieser Funktion einzeln abgepollt werden. Sollte in dem internen Puffer keine Nachricht vorhanden sein wird höchstens die Zeit "Timeout" auf eine eingehende undefinierte Nachricht [150] gewartet.

Synopsis

```
int AvailMessages, int CANID, int NodeID, int FunctionCode, table(1-8)
Data, int Seconds, int Microseconds = LS_CANopenGetUndefined( int Handle
, int Timeout );
```

Beschreibung

Parameter

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

int Timeout — Zeit in ms wie lange auf eine eingehende undefinierte Nachricht gewartet werden soll wenn der interne Puffer für undefinierte Nachrichten leer ist.

Rückgabewerte

int AvailMessages — Anzahl der Nachrichten die sich noch im Puffer befinden.

- = -2: Es wurden seit dem Start noch keine undefinierten Nachrichten empfangen. Die restlichen Rückgabeparameter sind ungültig
- = -1: Es waren keine Nachrichten im Puffer. Die restlichen Rückgabeparameter sind ungültig
- >= 0: Anzahl der noch verfügbaren Nachrichten im Puffer.

int CANID — CAN-ID der aktuellen Message. Nur gültig wenn AvailMessages >=0.

int NodeID — CANOpen NodeID der aktuellen Message. Nur gültig wenn AvailMessages >=0.

int FunctionCode — CANOpen Funktionscode der aktuellen Message. Nur gültig wenn AvailMessages >=0.

table(1-8) Data — Table mit den Daten. Nur gültig wenn AvailMessages >=0.

int Seconds — Zeit des Empfangs vom CAN-Bus in Sekunden seit dem 01.01.1976. Nur gültig wenn AvailMessages >=0.

int Microseconds	Zeit des Empfangs vom CAN-Bus: Mikrosekundenanteil. Nur gültig wenn AvailMessages >=0.

Als undefinierte Nachrichten werden die eingehenden Nachrichten bezeichnet die anhand des Funktionscodes keiner CANOpen Funktion zugeordnet werden können. Sollte nur in Netzen vorkommen in denen gleichzeitig zu CANOpen noch Standard CAN oder andere Protokolle betrieben werden.

LS_CANopenSendNMT

LS_CANopenSendNMT — Sendet eine CANOpen NMT Nachricht an die angegebene NodeID.

Synopsis

```
int RC = LS_CANopenSendNMT( int Handle , int NodeID , int NMTTyp );
```

Beschreibung

Parameter

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

int NodeID — CANOpen NodeID des Empfängers der NMT Nachricht.

int NMTTyp — Typ der NMT Nachricht.

Rückgabewerte

int RC — Treten keine Fehler auf, wird Null zurückgegeben. Falls die Funktion fehlschlägt, ist der Returncode ungleich Null.

LS_CANopenSendSYNC

LS_CANopenSendSYNC — Sendet eine SYNC Nachricht.

Synopsis

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

Beschreibung

Parameter

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

Rückgabewerte

int RC Treten keine Fehler auf, wird Null zurückgegeben. Falls die Funktion fehlschlägt, ist der Returncode ungleich Null.

LS_CANopenSendTIME

LS_CANopenSendTIME — Sendet eine TIME STAMP Nachricht.

Synopsis

```
int RC = LS_CANopenSendTIME( int Handle , int Day , int Milliseconds );
```

Beschreibung

Parameter

int Handle	Gültiges Zugriffs-Handle eines erfolgreichen Aufrufs von LS_CANOpenDevice.
int Day	gangene Tage seit dem 01.01.1984.
int Milliseconds	Vergangene Millisekunden seit Mitternacht.

Rückgabewerte

int RC Treten keine Fehler auf, wird Null zurückgegeben. Falls die Funktion fehlschlägt, ist der Returncode ungleich Null.

LS_CANopenSendPDO

LS_CANopenSendPDO — Sendet eine PDO Nachrict an die angegebene NodeID.

Synopsis

```
int RC = LS_CANopenSendPDO( int Handle , int NodeID , int PDOTyp , int
DataLength , table(1-8) SendData );
```

Beschreibung

Parameter

int Handle	Gültiges Zugriffs-Handle eines erfolgreichen Aufrufs von LS_CANOpenDevice.
int NodeID	CANOpen NodeID des Empfängers der PDO Nachricht.
int PDOTyp	PDO Typ der Nachricht.
int DataLength	Anzahl der zu versendenen Datenbytes.
table(1-8) SendData	Table mit den zu versendenden Daten.

Rückgabewerte

int RC Treten keine Fehler auf, wird Null zurückgegeben. Falls die Funktion fehlschlägt, ist der Returncode ungleich Null.

LS_CANopenSendSDORead

LS_CANopenSendSDORead — Sendet ein SDO Anfrage an einen CANOpen Slave um ein Objektverzeichniseintrag zu lesen. Bei erfolgreichem Lesen werden die gelesenen Daten zurückgegeben.

Synopsis

```
int RC, int SDOReadType, int SDOReadData = LS_CANopenSendSDORead( int Handle , int NodeID , int Index , int Subindex , int Timeout );
```

Beschreibung

Parameter

int Handle	Gültiges Zugriffs-Handle eines erfolgreichen Aufrufs von LS_CANOpenDevice.
int NodeID	CANOpen NodeID des Empfängers der SDO Nachricht.
int Index	Index des Objektverzeichniseintrages der gelesen werden soll.
int Subindex	Subindex des Objektverzeichniseintrages der gelesen werden soll.
int TimeOut	Timeout wie lange auf die Antwort des CANOpen Slaves gewartet werden soll.

Rückgabewerte

int RC	Treten keine Fehler auf, wird Null zurückgegeben. Falls die Funktion fehlschlägt, ist der Returncode ungleich Null.
int SDOReadType	Anzahl der gelesenen Bytes und Ergebnis der Leseanforderung.
int SDOReadData	Gelesene Daten. Nur gültig wenn Leseanforderung OK war (SDOReadType != 0x80).

LS_CANopenSendSDOWrite

LS_CANopenSendSDOWrite — Setzt das digitale Ausgabe-Register auf neue Werte.

Synopsis

```
int RC, int SDOReadType, int SDOReadData = LS_CANopenSendSDOWrite( int Handle , int NodeID , int SDOWriteTyp , int Index , int Subindex , int Timeout , int SDOWriteData );
```

Beschreibung

Parameter

int Handle	Gültiges Zugriffs-Handle eines erfolgreichen Aufrufs von LS_CANOpenDevice.
int NodeID	CANOpen NodeID des Empfängers der SDO Nachricht.
int SDOWriteTyp	Typ der Schreibanforderung.
int Index	Index des Objektverzeichniseintrages der geschrieben werden soll.
int Subindex	Subindex des Objektverzeichniseintrages der geschrieben werden soll.
int Timeout	Timeout wie lange auf die Antwort des CANOpen Slaves gewartet werden soll.
int SDOWriteData	Daten die geschrieben werden sollen.

Rückgabewerte

int RC	Treten keine Fehler auf, wird Null zurückgegeben. Falls die Funktion fehlschlägt, ist der Returncode ungleich Null.
int SDOReadTyp	.
int SDOReadData	.

LS_CANopenSendSDOReadBlock

LS_CANopenSendSDOReadBlock — Setzt das digitale Ausgabe-Register auf neue Werte.

Synopsis

```
int RC, int SDOReadType, int ReadLen, table ReadData = LS_CANopenSendSDOReadBlock( int Handle , int NodeID , int Index , int Subindex , int Timeout , int ReadLen );
```

Beschreibung

Parameter

int Handle	Gültiges Zugriffs-Handle eines erfolgreichen Aufrufs von LS_CANOpenDevice.
int NodeID	.
int Index	.
int Subindex	.
int Timeout	.
int ReadLen	.

Rückgabewerte

int RC	Treten keine Fehler auf, wird Null zurückgegeben. Falls die Funktion fehlschlägt, ist der Returncode ungleich Null.
int SDOReadType	.
int ReadLen	.
table ReadData	.

LS_CANopenSendSDOWriteBlock

LS_CANopenSendSDOWriteBlock — Setzt das digitale Ausgabe-Register auf neue Werte.

Synopsis

```
int RC, int SDOReadTyp, int SDOReadData = LS_CANopenSendSDOWriteBlock( int Handle , int SDOReadTyp , int SDOReadData );
```

Beschreibung

Parameter

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

int NodeID .

int Index .

int Subindex .

int Timeout .

int WriteLen .

Rückgabewerte

int RC — Treten keine Fehler auf, wird Null zurückgegeben. Falls die Funktion fehlschlägt, ist der Returncode ungleich Null.

int SDOReadType .

int SDOReadData .

Programmier-Beispiel

Programmier-Beispiel

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 Datenpackete auf 1. AnaGate CAN Device versenden.
- Die Datenpackete aus dem internen Puffer des 2. AnaGate CAN Device lesen.
- Überprüfen ob Endlosschleife verlassen werden soll.

Beispiel 13.1. CANOpen - LUA Scriptbeispiel

```
-- Filter: alle CAN-Identifier akzeptieren
aFilter = { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
            0x00, 0x1FFFFFFF, 0x00, 0x1FFFFFFF,
            0x00, 0x1FFFFFFF, 0x00, 0x1FFFFFFF};

aSendData = { 0xF1, 0xF2, 0xF3, 0xF4, 0xF5, 0xF6, 0xF7, 0xF8 };
--************************************************
function printf(...)
   io.write(string.format(...))
   io.flush();
end
--************************************************
function main()
   -- Open
   nRC, nHandle = LS_CANOpenDevice( false, true, 5001, "10.1.2.160", 5000 );
   if (nRC ~= 0) then
      print(LS_CANErrorMessage(nRC));
      exit();
   end;

   nRC, nHandle2 = LS_CANOpenDevice( false, true, 5001, "10.1.2.161", 5000 );
   if (nRC ~= 0) then
      print(LS_CANErrorMessage(nRC));
      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
      -- 1 Datenpackete auf dem 1. AnaGate CAN Device versenden
      nRC = LS_CANWrite(nHandle, 1, 8, aSendData );

      LS_Sleep(20); -- 20Millisekunden warten

      -- Datenpacket auf 2. AnaGate CAN Device empfangen
      nAvail, ID, Len, Data, Sec, Microsec = LS_CANGetMessage(nHandle2, 10);
      while nAvail>=0 do
         nAvail, ID, Len, Data, Sec, Microsec = LS_CANGetMessage(nHandle2, 10);
      end;
   until (false);

   -- Verbindungen beenden
   LS_CANCloseDevice(nHandle);
   LS_CANCloseDevice(nHandle2);
end;
```


Kapitel 14. LUA Programmier-Beispiele

14.1. 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 Datenpackete auf 1. AnaGate CAN Device versenden.
- Die Datenpackete aus dem internen Puffer des 2. AnaGate CAN Device lesen.
- Überprüfen ob Endlosschleife verlassen werden soll.

Beispiel 14.1. CAN- LUA Scriptbeispiel

```
-- Filter: alle CAN-Identifier akzeptieren
aFilter = { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
            0x00, 0x1FFFFFFF, 0x00, 0x1FFFFFFF,
            0x00, 0x1FFFFFFF, 0x00, 0x1FFFFFFF};

aSendData = { 0xF1, 0xF2, 0xF3, 0xF4, 0xF5, 0xF6, 0xF7, 0xF8 };
--************************************************
function printf(...)
   io.write(string.format(...))
   io.flush();
end
--************************************************
function main()
   -- Open
   nRC, nHandle = LS_CANOpenDevice( false, true, 5001, "10.1.2.160", 5000 );
   if (nRC ~= 0) then
      print(LS_CANErrorMessage(nRC));
      exit();
   end;

   nRC, nHandle2 = LS_CANOpenDevice( false, true, 5001, "10.1.2.161", 5000 );
   if (nRC ~= 0) then
      print(LS_CANErrorMessage(nRC));
      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
      -- 1 Datenpackete auf dem 1. AnaGate CAN Device versenden
      nRC = LS_CANWrite(nHandle, 1, 8, aSendData );

      LS_Sleep(20); -- 20Millisekunden warten

      -- Datenpacket auf 2. AnaGate CAN Device empfangen
      nAvail, ID, Len, Data, Sec, Microsec = LS_CANGetMessage(nHandle2, 10);
      while nAvail>=0 do
         nAvail, ID, Len, Data, Sec, Microsec = LS_CANGetMessage(nHandle2, 10);
      end;
   until (false);

   -- Verbindungen beenden
   LS_CANCloseDevice(nHandle);
   LS_CANCloseDevice(nHandle2);
end;
```


14.2. 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 DataRequest an den SPI-Partner gesendet.

- Setzen des WriteEnable Flags des SPI-Partners
- Abfragen des Status Registers
- Lesen von 20 Byte ab Adresse 0x00

Beispiel 14.2. SPI - LUA Scriptbeispiel

```
--************************************************
function printf(...)
   io.write(string.format(...))
   io.flush();
end;
--************************************************
function main()
   -- Verbindung zu AnaGate SPI-Device herstellen
   nRC, nHandle    = LS_SPIOpenDevice( "10.1.2.162", 5000 );
   if (nRC ~= 0) then
      errortext = LS_SPIErrorMessage(nRC);
      print(errortext);
      exit();
   end

   -- Setzen der globalen Einstellungen
   nRC = LS_SPISetGlobals( nHandle, 100, 2, 0, 0 );

   -- OP-Codes des SPI-Partners mit Daten
   OPWriteEnab = {0x06};
   OPStatusReg = {0x05, 0x00};
   OPRead      = {0x03, 0x00, 0x00, 0x00};

   -- WriteEnable-Flag des SPI-Partners setzen
   nRC, Value = LS_SPIDataReq(nHandle, 1, 1, OPWriteEnab);

   -- Statusregister des SPI-Partners abfragen
   nRC, Value = LS_SPIDataReq(nHandle, 2, 2, OPStatusReg);
   for i=1, table.getn(Value), 1 do
      printf("Data Status: %02X\n", Value[i]);
   end;

   -- Lesen von 20Bytes ab Adresse 0x00
   nRC, Value = LS_SPIDataReq(nHandle, 4, 20, OPRead);
   for i=1, table.getn(Value), 1 do
      printf("Data: %02X\n", Value[i]);
   end;

   -- Alle digitalen Ausgaenge zuruecksetzen
   LS_SPIWriteDigital(nHandle, 0);

   -- Verbindung zu AnaGate SPI-Device beenden
   LS_SPICloseDevice(nHandle);
end;
```


14.3. 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.

Beispiel 14.3. I2C - LUA Scriptbeispiel EEPROM-Funktionen

1. Beispiel

- 64 * 1024Bytes aus dem EEPROM lesen
- 10 * 128Bytes auf das EEPROM schreiben

```
--************************************************
function printf(...)
   io.write(string.format(...))
   io.flush();
end;
--************************************************
function getn(t)
   if type(t.n) == "number" then return t.n end;
   local max = 0
   for i, _ in t do
      if type(i) == "number" and i>max then max=i end;
   end;
   return max;
end;
--************************************************
function main()
   aSendData = {};
   for i=1, 128 , 1 do
      table.insert(aSendData, i-1 );
   end;

   nRC, nHandle    = LS_I2COpenDevice( 1000000, "10.1.2.162", 5000 );
   if (nRC ~= 0) then
      print(LS_I2CErrorMessage(nRC));
      exit();
   end;

   --Read EEPROM
   CountBytes = 1024;
   for Address = 0, CountBytes*64, CountBytes do
      nRC, Value = LS_I2CReadEEProm(nHandle, 1, Address, CountBytes, 16);
      for j=1, table.getn(Value), 1 do
         printf("%02X ", Value[j]);
         if (j%16 == 0) then
            printf("\n");
         end;
      end;
   end;

   --Write EEPROM
   CountBytes = table.getn(aSendData);
   for Address = 0, CountBytes * 10, CountBytes do
      nRC = LS_I2CWriteEEProm(nHandle, 1, Address, 16, CountBytes, aSendData);
   end;

   LS_I2CWriteDigital(nHandle, 0);
   LS_I2CCloseDevice(nHandle);
end;
```


Beispiel 14.4. I2C - LUA Scriptbeispiel I2C-Direkt

2. Beispiel

- 2Bytes an einen I2C-Partner zu schreiben senden
- 1024Bytes aus einem I2C-Partner lesen

```
--************************************************
function printf(...)
   io.write(string.format(...))
   io.flush();
end;
--************************************************
function getn(t)
   if type(t.n) == "number" then return t.n end;
   local max = 0
   for i, _ in t do
      if type(i) == "number" and i>max then max=i end;
   end;
   return max;
end;
--************************************************
function main()
   aSendData = {};
   for i=1, 128 , 1 do
      table.insert(aSendData, i-1 );
   end;

   nRC, nHandle    = LS_I2COpenDevice( 1000000, "10.1.2.162", 5000 );
   if (nRC ~= 0) then
      print(LS_I2CErrorMessage(nRC));
      exit();
   end;

   --Write
   aData = {0x00, 0x05}; -- ab Adresse 5 lesen
   nRC, Value = LS_I2CWrite(nHandle, 0xa2, 2, aData);

   --Read
   nRC, Value = LS_I2CRead(nHandle, 0xa2, 1024);
   for i=1 , table.getn(Value), 1 do
      printf("%02X ", Value[i]);
   end;
   printf("\n");

   LS_I2CWriteDigital(nHandle, 0);
   LS_I2CCloseDevice(nHandle);
end;
```


Beispiel 14.5. I2C - LUA Scriptbeispiel Sequence

3. Beispiel

- Sequenz an Anagate I2C Device senden

```
--************************************************
function printf(...)
   io.write(string.format(...))
   io.flush();
end;
--************************************************
function getn(t)
   if type(t.n) == "number" then return t.n end;
   local max = 0
   for i, _ in t do
      if type(i) == "number" and i>max then max=i end;
   end;
   return max;
end;
--************************************************
function main()
   aSendData = {};
   for i=1, 128 , 1 do
      table.insert(aSendData, i-1 );
   end;

   nRC, nHandle    = LS_I2COpenDevice( 1000000, "10.1.2.162", 5000 );
   if (nRC ~= 0) then
      print(LS_I2CErrorMessage(nRC));
      exit();
   end;

   --Sequence
   aData = {0xa2, 0x00,  --SLA
            0x02, 0x00,  --Laenge Schreibkommando
            0x00, 0x00,  --Daten Schreibkommando
            0xa3, 0x00,  --SLA 2. Lesekommando
            0x30, 0x00,  --Laenge 1. Lesekommando
            0xa3, 0x00,  --SLA 2. Lesekommando
            0x20, 0x00}; --Laenge 2. Lesekommando

   nRC,CountRead,LastError,Value = LS_I2CSequence(nHandle, 0x0E, 0x0050, aData);
   printf("CountRead:%02X  LastError:%02X\n", CountRead, LastError);
   for i=1, CountRead, 1 do
      printf("%02X ", Value[i]);
   end;
   printf("\n");

   LS_I2CWriteDigital(nHandle, 0);
   LS_I2CCloseDevice(nHandle);
end;
```


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.

Tabelle A.1. Allgemeine Rückgabewerte für alle Geräte der AnaGate Serie

Wert	Name	Beschreibung
0	ERR_NONE	Kein Fehler aufgetreten.
0x000001	ERR_OPEN_MAX_CONN	Open fehlgeschlagen, Maximale Anzahl 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-Parter 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 nich 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-Telegramm wurde vom Partner nicht korrekt quittiert.
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.2. Rückgabewerte für AnaGate I2C

Wert	Name	Beschreibung
0x000120	ERR_I2C_NACK	I2C-NACK
0x000121	ERR_I2C_TIMEOUT	I2C Timeout

Eine textuelle Beschreibung des Rückgabewertes kann mit der Funktion `I2CErrorMessage()` ermittelt werden. Die Beschreibung ist in englischer Sprache.

Tabelle A.3. Rückgabewerte für AnaGate CAN

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

Eine textuelle Beschreibung des Rückgabewertes kann mit der Funktion `CANErrorMessage()` ermittelt werden. Die Beschreibung ist in englischer Sprache.

Tabelle A.4. Rückgabewerte für AnaGate Renesas

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

Eine textuelle Beschreibung des Rückgabewertes kann mit der Funktion `RenesasErrorMessage()` ermittelt werden. Die Beschreibung ist in englischer Sprache.

Tabelle A.5. Rückgabewerte für LUA Scripting

Wert	Name	Beschreibung
-1	ERR_SYNTAX	Syntax Error
-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.

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 untersten 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 SPI* 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 E0, E1 und E2 bezeichnet):

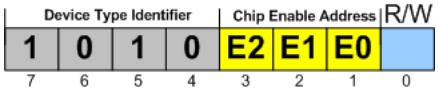


Tabelle B.1. Adressierungs-Beispiele von I2C-EEPROM's

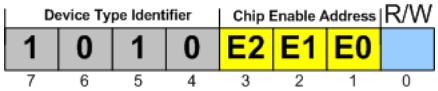
	Device Type Identifier				Chip Enable [1] [2]			R/W	EEPROM-Speicher
	b7	b6	b5	b4	b3	b2	b1	b0	
M24C01	1	0	1	0	E2	E1	E0	R/W	128 byte
M24C02	1	0	1	0	E2	E1	E0	R/W	256 byte
M24C04	1	0	1	0	E2	E1	A8	R/W	512 byte
M24C08	1	0	1	0	E2	A9	A8	R/W	1024 byte
M24C16	1	0	1	0	A10	A9	A8	R/W	2048 byte
M24C64	1	0	1	0	E2	E1	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 höchstwertigen Bits der Speicheradresse interpretiert.

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 addressiert (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.



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 Byte (16-Bit, MSB First) 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.

Tabelle C.1. Verwendung der Chip-Enable Bits bei I2C EEPROMs

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 Addressierung verwendet. Es entspricht dabei dem Adressbit A8 bzw. A16.
0x2	E2-A1-A0	Die Bits E0 und E1 werden für die Addressierung verwendet. E0 entspricht dabei dem Adressbit A8 bzw. A16 und E1 dem Adressbit A9 und A17.
0x3	A2-A1-A0	E0, E1 und E2 werden für die Addressierung verwendet. E0 entspricht dabei dem Adressbit A8 bzw. A16, E1 dem Adressbit A9 bzw. A17 und E2 dem Adressbit A10 bzw. A18.

Modus [1]	**Verwendung**	**Beschreibung**
0x5	A0-E1-E0	Das E2-Bit wird für die Addressierung verwendet. Es entspricht dabei dem Adressbit A8 bzw. A16.
0x6	A1-A0-E0	Die Bits E2 und E1 werden für die Addressierung 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.

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, anderfalls 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 leuchet, 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**

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 sitzt.

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-Addresse 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 IP4-Netzwerken wird das **Address Resolution Protocol (ARP)** verwendet, um die MAC-Adressen zu gegebenen IP-Adressen zu ermitteln. Diese 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, nachdem 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 dem Kommando **arp -d** kann der gesamte *ARP Cache* gelöscht werden.

Anmerkung

Eventuell muß auch nach dem Ändern der IP-Adresse eines Gerätes der *ARP Cache* des PCs gelöscht werden.

F: Verwendung einer Firewall

A: Bei Verwendung einer Firewall muß der entsprechende Port für die Kommunikation mit dem AnaGate freigeschaltet sein:

Tabelle D.1. Nutzung AnaGate-Hardware mit Firewall

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 SPI	5002
AnaGate Renesas	5008
AnaGate Universal Programmer	5000, 5002, 5008

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 die ANALYTICA auch auch ein CAN-Gateway ohne galvanisch getrennte CAN-Schnittstelle an?

A: Ein Gerät, dass aktiv am CAN-Bus betrieben wird, sollte grundsätzlich eine galvanische Trennung besitzten. 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!

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 zum AnaGate I2C

F: Welche Reihenfolge ist bei der Kontaktierung von GND / SCL und SDA bei Verwendung eines externen Power Supplys zu beachten?

A: Um potentielle Schäden am *AnaGate I2C* zu vermeiden, muß immer zuerst der GND Pin mit dem Application Board verbunden werden und erst danach dürfen die Pins SCL/SDA mit dem Application Board kontaktiert werden.

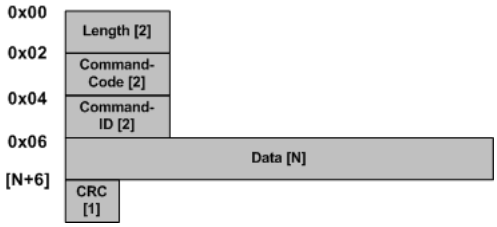
Anhang E. FAQ zu den Programmier-Schnittstellen

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 Prüfksumme (CRC) für die AnaGate-Telegramme ist unklar!

A: Die folgende Abbildung zeigt den grundsätzlichen Aufbau eines *AnaGate*-Befehlstelegrammes:



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 Befehlstelegrams:

```
unsigned char CalcCRC( char * pBuffer, int nBufferLength )
{
   int i;
   unsigned char nCRC = pBuffer[2]; // skip the length bytes

   // XOR over 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)
```


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.

EMail

Für technische Unterstützung Über Internet, senden Sie bitte eine EMail 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

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*.

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-2010] Analytica GmbH. Copyright © 2010 Analytica GmbH. *AnaGate API* . Programmer's Manual . Version 1.4. 01.10.2010.

[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*.