Programmierung des ROBO TX Controllers

Teil 3: C-Compiler für Robo-Programme

Package und Dokumentation
„C_Compiler_RoboTXC“

V1.2 vom 25.04.2012

Referenzen:

Bezeichnung	Version	Datum
ROBO TX Controller Firmware	1.30	19.03.2012
GNU ARM C-Compiler (GCC)	4.4.1	22.07.2009

MSC Vertriebs GmbH
Design Center Aachen
Pascalstr. 21
52076 Aachen

Inhalt

1 Einführung

Dieses Dokument beschreibt die direkte Programmierung des fischertechnik ROBO TX Controllers in Form eines lokal ausgeführten Programms, dass in C geschrieben wird. Dies stellt eine Alternative zu Programmen dar, die in ROBOPro entwickelt wurden und die als übersetzer Maschinencode von ROBOPro im Download-Modus ausgeführt werden. Die Programmierung in C bietet dabei eine Möglichkeit auch ohne ROBOPro zu arbeiten.

Der ROBO TX Controller hat einen 32-Bit ARM9 Microprozessor vom Hersteller Atmel mit der Bezeichnung AT91SAM9260, der mit rund 200 MHz Taktfrequenz betrieben wird. Eine passende Entwicklungsumgebung für diese CPU ist die YAGARTO Toolchain (siehe auch www.yagarto.de). Diese beinhaltet den GNU ARM C-Compiler GCC, der ausführbaren Code für diese CPU erzeugen kann, sowie das Make Tool. Diese sind alle kostenfrei verfügbar. Die in diesem Paket mitgelieferte Version des C-Compilers und der YAGARTO Toolchain repräsentieren nicht unbedingt den allerneuesten Stand, wohl aber eine verifizierte und von uns freigegebene Version.

Das Laden von eigenen Programmen erfolgt über die vorhandenen Standard-PC-Schnittstellen des ROBO TX Controllers, USB und Bluetooth, genauso, wie dies auch aus ROBOPro heraus der Fall wäre.

Es werden zudem eine Anzahl Demo-Programme im C-Source-Code mitgeliefert, die schon einem Programmierer mit C-Grundkenntnissen einen Einstieg in die direkte Programmierung des ROBO TX Controllers gibt. Experten hingegen können sowohl von der Seite der Tools, als auch von der Komplexität der C-Programme her das System weiter ausbauen und verfeinern.

1.1 Systemvoraussetzungen

Die C-Programmierung des ROBO TX Controllers, soweit in diesem Dokument beschrieben, setzt ein Microsoft Windows Betriebssystem (Windows 2000 oder neuer) voraus.

Weitere Systemvoraussetzungen sind ein USB-Fullspeed-Host-Port auf dem PC, sowie eine Bluetooth-Schnittstelle (im PC eingebaut oder über externen Adapter, z.B. ein Bluetooth-USB-Stick).

1.2 Begriffsdefinitionen

ROBO TX Controller heißt der fischertechnik Robotik-Controller selbst. Er wird auch abkürzend mit **TX-C** bezeichnet.

Firmware ist das Software-Basissystem, das auf dem ROBO TX Controller läuft, also das Betriebssystem, Gerätetreiber, Protokolle und der Bootloader.

Robo-Programm bezeichnet den Teil der Software, der als ladbare, ausführbare Robo-Applikation auf dem ROBO TX Controller laufen kann. Voraussetzung ist auf dem TX-C natürlich eine vorhandene und lauffähige Firmware.

WICHTIGER HINWEIS: das C-Compiler-Paket ist immer für ein bestimmte Firmware-Version des ROBO TX Controllers bestimmt. Die mitgelieferten Beispiele und Source-Dateien mögen sowohl zu älteren als auch zu neuere Versionen der Firmware nicht kompatibel sein. Bitte achten Sie daher darauf, dass Sie immer die Firmware-Version auf dem ROBO TX Controller verwenden, die auf dem Deckblatt als Referenz genannt ist.

2 Software-Struktur auf dem ROBO TX Controller

Nachfolgend ist beschrieben, was die Firmware auf dem ROBO TX Controller genau macht, wie sie aufgebaut ist und was ansonsten zur Programmierung eigener Robo-Programme in C noch wissenswert ist.

2.1 Das Betriebssystem 4NetOS®

Der Firmware, die auf dem ROBO TX Controller läuft, liegt das Multi-Tasking-Echtzeit-Betriebssystem 4NetOS® zugrunde. Es werden verschiedene System-Tasks parallel zu folgenden (i.d.R. zyklisch ausgeführten) Aufgaben ausgeführt:

- Lesen der Inputs und Setzen der Outputs
- Synchronisieren der Transfer Area Daten mit dem PC bzw. mit den Slave Boards
- Bedienen der seriellen Schnittstellen (inkl. USB und Bluetooth)
- Verwalten des Dateisystems
- Auslesen der Taster und Ausgabe auf das Displays
- Ausführen eines lokalen Robo-Programms, soweit vorhanden und gestartet, im Zeitmultiplex-Betrieb mit den anderen Tasks.
- Andere interne Aufgaben

Diese verschiedenen Tasks sind ständig aktiv und verhalten sich wie Zustandsautomaten, d.h. sie reagieren auf Ereignisse (events). Dies können z.B. eingehende Daten über eine der Kommunikationsschnittstellen oder ablaufende Timer-Ereignisse sein.

Der Grundbetrieb des Betriebssystems ist bezüglich der I/O-Kommunikation wie folgt: der Start erfolgt im „Local"-Modus. Insofern die Konfigurationsoption „Load-after-power-on" auf „YES" steht, wird geprüft, ob eine ausführbares Robo-Programm konfiguriert und auf der Flashdisk vorhanden ist. Wenn dies der Fall ist, und zudem die Konfigurationsoption „Start-after-power-on" auf „YES" steht, so wird dieses auch sofort ausgeführt. Ist dies nicht der Fall, so erfolgt zunächst keine I/O-Kommunikation. Werden I/O-Befehle von einer der Kommunikationsschnittstellen (USB oder Bluetooth) empfangen, so wird dynamisch in den Online-Modus gewechselt und die I/O-Befehle direkt interpretiert und ausgeführt. Wenn I/O-Befehle während der Ausführung eines lokalen Robo-Programms empfangen werden, so wird das lokale Programm unterbrochen. Dieses muss dann per Befehl wieder gestartet werden.

Dateisystem und Programmspeicher

Der ROBO TX Controller besitzt ein Dateisystem mit zwei „Laufwerken", der Ramdisk und der Flashdisk. Auch wenn dies physikalisch nur Speicherbausteine sind und keine Festplatten, so sind dennoch Dateistrukturen auf diesen im Einsatz, die den Begriff „Disk" rechtfertigen. Die Flashdisk stellt dabei ein nichtflüchtiges „Laufwerk" dar, das Dateien auch über Ein-/Ausschaltzyklen hinweg speichern kann, während die Ramdisk ein flüchtiges „Laufwerk" ist, das seinen Inhalt nach jedem Ausschalten wieder verliert.

Sowohl auf die Flashdisk als auch auf die Ramdisk kann vom PC aus fern zugegriffen werden, so dass auf einfache Weise Dateien vom PC zum ROBO TX Controller übertragen werden können. Auch Firmware-Updates erfolgen auf diese Weise.

Robo-Programme, egal ob von RoboPro oder über den C-Compiler erstellt, werden zunächst immer als Datei abgelegt. Durch das Vorhandensein eines Dateisystems können so auf dem Interface ein große Anzahl von Programmen gespeichert werden. Dies erfolgt i.d.R. auf der Flashdisk. Nur dann, wenn Programme dynamisch geladen werden und nicht permanent gespeichert werden sollen, erfolgt eine Zwischenspeicherung auf der Ramdisk als Datei.

Zum Ausführen von Programmen sind diese zunächst in den sogenannten Programmspeicher zu laden. Darunter versteht man einen reservierten Bereich im RAM in welchen der Inhalt von Programm-Dateien kopiert wird, um dort ausgeführt zu werden. Das nachfolgende Bild veranschaulicht den Speicheraufbau des Interfaces:

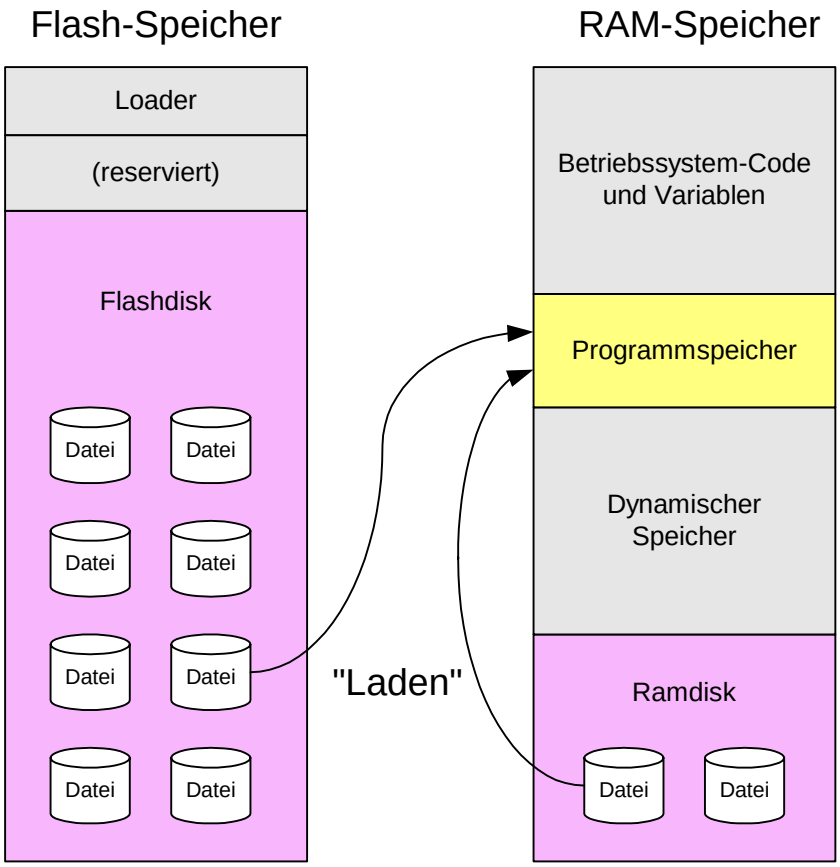


Abb. 1: Speicherstruktur auf dem ROBO TX Controller

Zum Programmspeicher gelten folgende Regeln:

- Es kann sich zu jedem Zeitpunkt immer nur ein Programm im Programmspeicher befinden.
- Nur das Programm im Programmspeicher kann auch gestartet werden. Es kann dort aber auch vorhanden sein ohne ausgeführt zu werden (Zustand „Stopped“).
- Mit den Konfigurationsoptionen „Load-after-power-on“ und „Start-after-power-on“ kann eingestellt warden, dass ein Programm nach dem Einschalten des Interfaces automatisch aus einer Programm-Datei in den Programmspeicher geladen und dort auf Wunsch auch automatisch gestartet wird. Alternativ kann es auch nur geladen werden und erst per Taster gestartet werden.

2.2 Die Transfer Area

Die Transfer Area ist ein Speicherbereich, der die Eingangs- und Ausgangswerte als Prozessabbild zwischenspeichert. Dieser Bereich wird nach dem Starten des Programms durch die ROBO TX Controller Firmware alle 10ms mit der Hardware abgeglichen. Zusätzlich sind auch Konfigurationsdaten in der Transfer Area abgelegt. Diese sind dazu bestimmt Optionen und Parameter einzustellen, etwa welcher Art ein Eingang sein soll: digital/analog, Spannungs-/Widerstandsmessung oder bidirektional für den Ultraschall-sensor, oder noch der Geschwindigkeitswert für die Motoren usw.

Programmierer die schon mit dem Paket „PC_Programmierung_RoboTXC“ und der ftMscLib.dll auf dem PC gearbeitet haben, kennen bereits das Prinzip der Transfer Area. Die PC-Library hat allerdings eine Vielzahl von Kapselungsfunktionen, während ein C-Programm auf dem Controller i.d.R. direkt auf die Transfer Area zugreift. Es sei an dieser Stelle auch auf die Dokumentation des „PC_Programmierung_RoboTXC“-Pakets verwiesen, in der die Felder und Variablen der Transfer Area näher beschrieben sind.

Einen weiteren Unterschied hat die Transfer Area auf dem Controller auch noch: es gibt, im Gegensatz zur PC-Version, eine Tabelle mit Funktionszeigern, die sogenannte „Hook Table“, hinter denen sich Firmware-Funktionsaufrufe verbergen, die von Robo-Programmen verwendet werden können. Näheres dazu in Kapitel 7 – Liste der Funktionsaufrufe (API).

2.3 Ausführung lokaler Programme

Wie weiter oben bereits angedeutet ist ein lokales Robo-Programm Teil eines Multitasking-Systems das auf dem ROBO TX Controller als Firmware läuft. Insofern erfolgt die Ausführung des Robo-Programms in Zeitscheiben à 1 ms, wovon rund die Hälfte (0,5 ms) dem Programm in jedem Zeit-Slot zur Verfügung steht. So (und nur so) ist sichergestellt, dass auch allen anderen System-Tasks noch ausreichend Rechenzeit zur Verfügung steht und dennoch das Robo-Programm präzise und in Echtzeit abläuft.

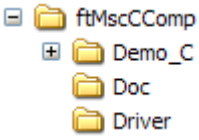
3 Installation der Entwicklungsumgebung

Für den Zweck einer möglichst einfachen Erläuterung, wie C-Programme für den ROBO TX Controller erstellt und geladen werden, haben wir eine einfache kommandozeilenbasierte Entwicklungsumgebung vorbereitet, die mit Hilfe von Make-Files C-Programme übersetzt (compiliert) und in ein ladbares Format umsetzt. Es gibt eine Vielzahl komfortabler, grafisch orientierter Entwicklungsoberflächen, die ebenso eingesetzt werden können, nur würde deren Beschreibung den Rahmen dieses Dokuments sprengen.

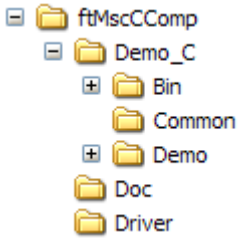
3.1 Auspacken der mitgelieferten ZIP-Files (Compiler + Tools)

Die mitgelieferte Entwicklungsumgebung kann in ein beliebiges Verzeichnis auf dem PC installiert werden. In dieser Beschreibung wird dafür der Verzeichnispfad C:\ftMscCComp als Ausgangspfad verwendet. Dieses Verzeichnis kann jederzeit mit allen Unterverzeichnissen an eine andere Stelle verschoben werden ohne dass die Verwendbarkeit beeinträchtig wäre, weil nur relative Pfadangaben in den Batch- und Make-Files verwendet werden. Beachten Sie jedoch, dass letztere von den vorgegebenen Unterverzeichnisnamen ausgehen.

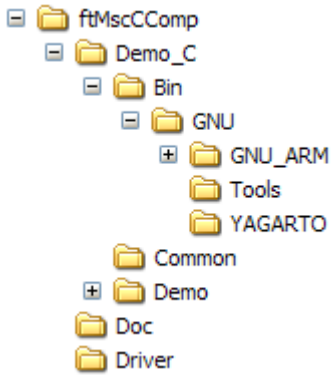
Als ersten Schritt packen Sie den Inhalt des ZIP-Files ftMscC**Demo**.zip in den Ordner C:\ftMscCComp aus. Dies sollte dann wie folgt aussehen:



Hier befindet sich im Verzeichnis „Doc“ wieder diese Beschreibung und im Verzeichnis „Demo_C“ finden Sie die folgende Verzeichnistruktur vor:



Im Moment hat das Verzeichnis “Bin” noch keine weiteren Unterverzeichnisse. Hierhin packen Sie bitte den Inhalt von ftMscC**Comp**.zip aus. Es erscheint dann unterhalb von „Bin“ das Unterverzeichnis „Gnu“, sowie weitere Unterverzeichnisse:



Im Unterverzeichnis “YAGARTO” sind auch die Setup-Files vorhanden, um die komplette YAGARTO-Toolchain zu installieren. Da für die mitgelieferten Demo-Programme nicht alle Komponenten dieser Toolchain verwendet werden, haben wir die benötigten Software-Teile in die Unterverzeichnisse „GNU_ARM“ (GNU ARM GCC compiler) und „Tools“ extrahiert (make und rm Tools). Bei den entsprechenden Beispielen, werden die Tools aus diesen beiden Verzeichnissen direkt aufgerufen und es ist daher nicht erforderlich die ganze YAGARTO-Toolchain auf dem PC zu installieren.

3.2 Installation des USB-Treibers

Dieses Kapitel können Sie überspringen, wenn eine USB-Verbindung zum ROBO TX Controller bereits möglich ist.

Der laufende ROBO TX Controller wird per USB-Kabel im Betrieb an den PC angeschlossen (ggf. nochmals aus- und wieder einstecken). Windows wird anschließend das Vorhandensein eines neuen USB-Devices melden und nach einem Treiber suchen. Wählen Sie im automatisch startenden Installationsassistenten auf die Frage nach „Windows-Update“ die Auswahl „Nein, diesmal nicht“ und im nächsten Fenster „Software von einer bestimmten Liste oder Quelle installieren“ und verweisen Sie auf die mitgelieferte Datei **`\Driver\ROBO_TX_Controller.inf`**. Anschließend sollte im Geräte-Manager ein zusätzliches COM-Port-Device mit der Bezeichnung “fischertechnik USB ROBO TX Controller” auftauchen. Notieren Sie am besten die COM-Port-Nummer, die von Windows automatisch vergeben wurde (siehe Geräte-Manager).

3.3 Bluetooth-Treiber-Installation

Dieses Kapitel können Sie überspringen, wenn eine Bluetooth-Verbindung zum ROBO TX Controller bereits möglich ist oder Sie ausschließlich mit USB arbeiten wollen.

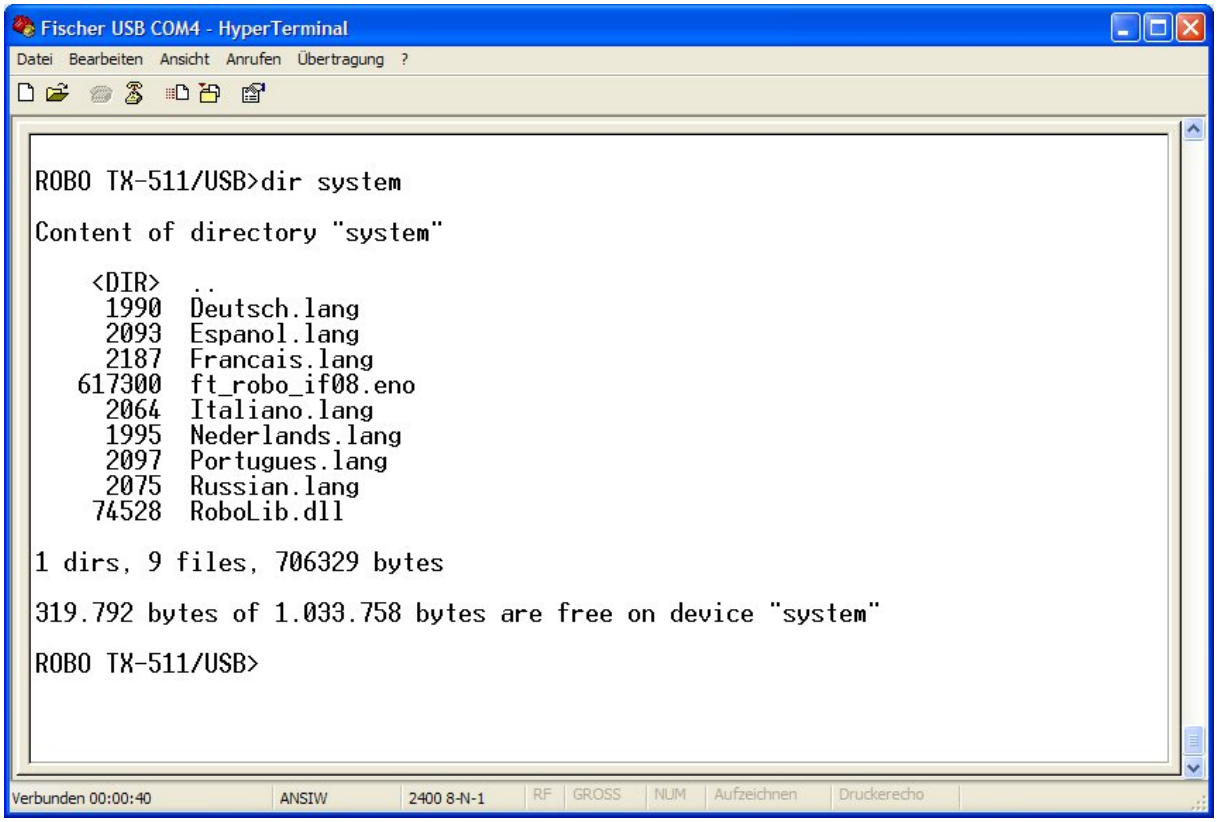
Der laufende ROBO TX Controller kann jederzeit vom PC-Bluetooth-Dienst erkannt werden, sofern diese Eigenschaft („Bluetooth Gerät sichtbar“) in den entsprechenden Einstellungen nicht ausgeschaltet ist. Es erscheint im Suchfenster auf dem PC mit seinem Host-Namen, der auch auf dem Display angezeigt wird (z.B. „ROBO TX-511“). Eine Kopplung mit dem PC erfolgt unter Verwendung des Hauptschlüssels (PIN) „1234“. Von dem Moment an ist dem ROBO TX Controller ein fester COM-Port zugeteilt und die Bluetooth-Verbindung wird vom PC automatisch aufgebaut, sobald auf den entsprechenden COM-Port zugegriffen wird.

Je nachdem, welcher Bluetooth-Protokoll-Stack und welche Bluetooth-Hardware (eingebaut oder Bluetooth USB-Stick) auf dem PC verwendet werden, kann sich die Installation und die Kopplung des PCs mit dem TX-C im Ablauf unterscheiden. Hierbei sind in jedem Fall die Installationsanweisungen des jeweiligen Herstellers zu befolgen.

3.4 Erster Test von USB und Bluetooth

Ein erster Test der USB-Kommunikation bzw. der Bluetooth-Kommunikation mit dem ROBO TX Controller Board kann mit einem einfachen Terminal-Programm (z.B. HyperTerminal, ZOC oder TeraTerm) erfolgen. Wählen Sie die richtige COM-Port-Nummer des USB- bzw. Bluetooth-Treibers aus und konfigurieren Sie eine beliebige Baurate (spielt bei virtuellen COM-Ports keine Rolle).

Wenn die Verbindung aufgebaut werden konnte, erhalten Sie im Terminal-Programm die Monitor-Konsole des Betriebssystems 4NetOS, das auf dem ROBO TX Controller läuft. Mit jedem Drücken von <ENTER> sollte nun ein Kommandozeilen-Prompt erscheinen. Durch Eingabe von „dir“ können Sie sich weiterhin den Inhalt der virtuellen Laufwerke „System“, „Flash“ oder „Ramdisk“ anzeigen lassen:



Weitere Befehle erhalten Sie durch Eingabe von „?“ oder „help“. Diese sind auch in der gesonderten Dokumentation zur PC-Programmierung (PC-Programming-Paket) näher beschrieben.

3.5 COM-Port-Einstellung für das Lade-Tool 4load_ft.exe

Es gibt einige Batch-Files (Endung .BAT) im Verzeichnis „Bin“. Diejenigen mit dem führenden Unterstrich '_' im Namen sind für den internen Gebrauch der von uns zusammengestellten Umgebung. Diese sollten von Ihnen daher nicht verändert werden. Es gibt nur eine Datei „set_port.bat“ die ggf. an Ihre Anforderungen angepasst werden müsste. Diese Datei legt den COM Port fest, auf welchen der ROBO TX Controller zum Laden von Programmen mit unserem Tool „4load_ft.exe“ angesprochen werden kann. Es gibt nur eine Zeile in diesem Batch-File:

```
set COM_PORT=
```

Wenn Sie den ROBO TX Controller vom PC aus über USB ansprechen möchten, dann können Sie diese Einstellung leer lassen. Das Lade-Tool „4load_ft.exe“ kann den richtigen COM-Port für das USB-Gerät selbstständig ermitteln, indem es in der Liste der COM-Ports (Windows Geräte-Manager) nach dem Namen „fischertechnik USB ROBO TX Controller“ sucht.

Wenn Sie jedoch den ROBO TX Controller über Bluetooth ansprechen wollen, um Programme über Funk zu laden, dann müssen Sie dem Lade-Tool „4load_ft“ die richtige COM-Portnummer im Batch-File „set_port.bat“ angeben, weil diese sich nicht automatisch ermitteln lässt, auf Grund der Vielzahl der unterschiedlichen Bluetooth-Protokoll-Stacks und Bluetooth-Schnittstellen alleine für Windows PCs. Erreichen Sie den ROBO TX Controller z.B. über den Port COM6 dann tragen Sie folgendes in „set_port.bat“ ein:

```
set COM_PORT=COM6
```

4 Mitgelieferte Demo-Programme in C

Im Verzeichnis C:\ftMscCComp\Demo_C\Demo\ befinden sich Source-Code-Beispiele für in der Sprache C geschriebene Robo-Programme.

Die Beispiele zeigen einfache Anwendungen zur Steuerung der ROBO TX Controllers:

LightRun – Digitalausgänge (Lampen)

Beispiel zur Verwendung der Einzelausgänge O1 bis O8, etwa zur Ansteuerung von Lampen mit Zeitsteuerung (Blinken).

Das Programm beendet sich nach kurzem Ablauf selbstständig.

MotorRun – Motor

Einfache Motoransteuerung auf Ausgang M1 mit verschiedenen Geschwindigkeiten in beiden Drehrichtungen.

Das Programm beendet sich nach kurzem Ablauf selbstständig.

StopGo – Encoder-Motor, Digitaleingang (Taster) und LC-Display-Ausgabe

Beispielprogramm zur Ansteuerung eines Encoder-Motors. Dabei werden Zählimpulse gelesen und bei einem Zählerstand von 1000 wird das Programm beendet. Der Motor wird über den Digitaleingang I8 gestartet (=1) oder gestoppt (=0). Enthält auch exemplarische Verwendung der Nachrichten-Ausgabe auf dem LC-Display und eines Digitaleingangs (Taster).

WarningLight – Ultraschallsensor und Digitalausgang (Lampe)

Beispielprogramm zur Ansteuerung einer Lampe auf O8 mit Distanzmessung über einen Ultraschallsensor auf I1. Die Lampe blinkt dabei in unterschiedlichen schnellen Intervallen, je kleiner der gemessene Abstand über den Ultraschallsensor ist.

Das Programm wird mit der Taste „Stop“ auf dem Controller beendet.

MotorEx_2M_Master – 2 Encoder-Motoren synchronisieren

Beispielprogramm zur Ansteuerung zweier Encoder-Motoren auf M1 und M2 (Encoder-Ausgänge auf C1 und C2 respektive) und zum synchronen Betrieb der beiden Motoren. Beide Motoren fahren zyklisch je 200 Schritte hin und her. Bremst man einen Motor während des Laufens (z.B. durch einen mechanischen Widerstand) ab, so passt der jeweils andere Motor seine Geschwindigkeit entsprechend an.

Das Programm wird mit der Taste „Stop“ auf dem Controller beendet.

MotorEx_Ext1 – Encoder-Motor auf Extension-Controller betreiben

Beispielprogramm zur Ansteuerung eines Encoder-Motors auf M1 (Encoder-Ausgang auf C1 respektive) auf einem Extension-Module (Slave Extension 1). Der Motor fährt zyklisch je 200 Schritte hin und her. Das Master-Modul ist das jenige, was am PC angeschlossen ist.

Das Programm wird mit der Taste „Stop“ auf dem Master-Controller beendet.

StopGoBtButtonPart und **StopGoBtMotorPart** – Bluetooth-Messaging

Siehe detaillierte Beschreibung weiter unten in Kapitel 5.1.

I2cTemp – I²C-Temperatursensor DS1631 und LC-Display-Ausgabe

Beispielprogramm zur Ansteuerung eines externen Temperatursensors DS1631 mit I²C-Schnittstelle (Conrad Electronic Best.-Nr. 19 82 98).

Nach einer kurzen Initialisierungssequenz wird der aktuelle Temperaturwert jede Sekunde ausgelesen und auf dem LC-Display ausgegeben.

Das Programm wird mit der Taste „Stop“ auf dem Controller beendet.

Hinweis: andere Conrad-Sensoren aus der C-Control-Reihe können ebenfalls verwendet und direkt an den ROBO TX Controller angeschlossen werden. Der I²C-Anschluss ist kompatibel.

I2cTpa81 – I²C-Infrarot-Temperatursensor-Zeile TPA81 und LC-Display-Ausgabe

Beispielprogramm zur Ansteuerung einer externen Temperatursensor-Zeile TPA81 mit I²C-Schnittstelle (erhältlich z.B. über www.roboter-teile.de). Dieser erlaubt eine berührungslose Temperaturmessung und über seine Zeilenstruktur (8 Pixel) auch ein einfache Ortsbestimmung von Objekten die Wärme abstrahlen.

Nach einer kurzen Initialisierungssequenz wird der Wert der Umgebungstemperatur sowie aller 8 Messstellen (Pixel) auf dem LC-Display ausgegeben. Bewegt man eine Wärmequelle vor der Linse so kann ein Anstieg des Wertes, dessen Pixel die Wärmequelle im Fokus hat, beobachtet werden. Die Anzeige wird jede Sekunde aktualisiert.

Das Programm wird mit der Taste „Stop“ auf dem Controller beendet.

Die erstellten ausführbaren Robo-Programme (BIN-Files) müssen sich im Dateisystem des ROBO TX Controllers befinden, um über die Taster gestartet zu werden.

5 Kompilieren und Laden der Demo-Programme

Alle Beispiele in diesem Paket sind im Unterverzeichnis „Demo“ des Verzeichnisses „Demo_C“ abgelegt. Jedes Programm liegt dort wiederum in einem Unterverzeichnis, was nach dem Programm, das es enthält, benannt ist. Auf derselben Ebene, wie das Verzeichnis „Demo“ liegt auch das Verzeichnis „Common“, in dem von allen Programmen gleichermaßen nutzbare Header-Files (.H) und Code-Fragmente (.C) abgelegt sind.

Wechseln Sie z.B. in das Verzeichnis C:\ftMscCComp\Demo_C\Demo\StopGo\ und sehen Sie sich die dort enthaltenen Batch-Files an (praktisch für alle Demos gleich):

clean.bat – ein Batch File, um alle erzeugten (compilierten) Dateien zu löschen. Es löscht auch alle temporären Dateien, die beim Compilieren entstanden sind und setzt das jeweilige Verzeichnis wieder in den Auslieferzustand zurück.

load_flash.bat – ein Batch File, um ein compiliertes ROBO-Programm auf die Flash-Disk des ROBO TX Controllers zu laden (zur permanenten Speicherung).

load_ramdisk.bat – ein Batch File, um ein compiliertes ROBO-Programm auf die Ramdisk des ROBO TX Controllers zu laden (zur vorübergehenden Speicherung).

run.bat – ein Batch File, um ein in den Programmspeicher des ROBO TX Controllers geladenes Robo-Programm vom PC aus zu starten. Hierzu wird dem Befehls-Tool 4cmd_ft.exe der Befehl „run“ übergeben. Ebenso gut kann der Programmstart auch mit der entsprechenden Taste auf dem Controller erfolgen.

stop.bat – ein Batch File, um ein auf dem ROBO TX Controllers bereits in Ausführung befindliches Robo-Programm vom PC aus wieder zu stoppen. Hierzu wird dem Befehls-Tool 4cmd_ft.exe der Befehl „stop“ übergeben. Ebenso gut kann das Programmende auch mit der entsprechenden Taste auf dem Controller herbeigeführt werden.

make.bat – ein Batch File, um den C-Source-Code des Demo-Programms zu compilieren. Dieses Batch-File ruft das Make-Tool der YAGARTO-Toolchain auf, was auch den Linker aufruft, der seinerseits eine ELF-File und anschließend ein ladbares BIN-File erzeugt. Solche BIN-Files können direkt auf den ROBO TX Controller geladen und dort zur Ausführung gebracht werden.

StopGo.c – C-Source-Datei des Demo-Programms “StopGo”.

param.mk – Parameter für das Make-Tool. Hierin sind Angaben über Namen und ggf. Anzahl der C-Source-Files and der Name der zu erzeugenden (Ausgangs-)Files (i.e. Dateiname des Robo-Programms) gemacht.

Führen Sie nun im Verzeichnis C:\ftMscCComp\Demo_C\Demo\StopGo\ das Batch File "make.bat" aus. Dann sollten Sie (in etwa) folgendes auf dem Bildschirm sehen:

```
arm-elf-gcc -S -mcpu=arm9e -DENDIAN_LITTLE -O3 -gdwarf-2 -fno-dwarf2-cfi-asm -ms
oft-float -fno-builtin -x c -Wall -c -I. -I../../Common -C -E ../../Common/prg_d
isp.c > ../../Common/prg_disp.p
arm-elf-gcc -S -mcpu=arm9e -DENDIAN_LITTLE -O3 -gdwarf-2 -fno-dwarf2-cfi-asm -ms
oft-float -fno-builtin -x c -Wall -c -I. -I../../Common -o ../../Common/prg_disp
.asm ../../Common/prg_disp.c
arm-elf-as -EL -mfloat-abi=soft -almns=../../Common/prg_disp.lst -o ../../Common
/prg_disp.o ../../Common/prg_disp.asm
arm-elf-gcc -S -mcpu=arm9e -DENDIAN_LITTLE -O3 -gdwarf-2 -fno-dwarf2-cfi-asm -ms
oft-float -fno-builtin -x c -Wall -c -I. -I../../Common -C -E StopGo.c > StopGo.
p
arm-elf-gcc -S -mcpu=arm9e -DENDIAN_LITTLE -O3 -gdwarf-2 -fno-dwarf2-cfi-asm -ms
oft-float -fno-builtin -x c -Wall -c -I. -I../../Common -o StopGo.asm StopGo.c
arm-elf-as -EL -mfloat-abi=soft -almns=StopGo.lst -o StopGo.o StopGo.asm
arm-elf-ld --cref --oformat elf32-littlearm --trace --nmagic --architecture=armv
5tej \
        --library-path=../../Bin/GNU/GNU_ARM/lib/gcc/arm-elf \
        --script=../../Common/ld.lcf \
        ../../Common/prg_disp.o StopGo.o \
        --library=gcc \
        -Map StopGo.map \
        -o StopGo.elf
arm-elf-ld: mode armelf
../../Common/prg_disp.o
StopGo.o
arm-elf-objcopy --output-target=binary StopGo.elf StopGo.bin
```

Wenn es irgendwelche Fehler oder Warnungen während des Compilier-Vorgangs gibt, dann würden Sie ebenfalls in demselben Fenster ausgegeben. Dies kann bei selbst erstellten Programmen vorkommen. Die mitgelieferten Demo-Programme hingegen compilieren natürlich ohne Fehlermeldungen.

Während des Compilier-Vorgangs wurden diverse neue Dateien erzeugt. Unter anderen wurde „StopGo.bin“ erstellt, was das ausführbare Robo-Programm ist. Wir können dieses nun in das Dateisystem des ROBO TX Controllers übertragen und dort zur Ausführung bringen:

1. Schalten Sie den ROBO TX Controller ein. Eine USB- oder Bluetooth-Verbindung vom PC zum ROBO TX Controller sollte möglich sein (siehe oben).
2. Führen Sie "load_ramdisk.bat" oder "load_flash.bat" aus je nachdem, ob Sie das Robo-Programm auf die Ramdisk (flüchtig) oder auf die Flashdisk (nicht-flüchtig) übertragen wollen. Das über die referenzierten Batch-Files aus dem Bin-Verzeichnis aufgerufene Lade-Tool „4load_ft.exe“ findet dabei den COM-Port automatisch (USB) oder über die Einstellung in „set_port.bat“ (Bluetooth, siehe oben). Sie können diese Voreinstellungen auch durch die explizite Angabe einer COM-Portnummer überschreiben, wenn Sie z.B. für COM6 folgendes eingeben:

   ```
   load_ramdisk COM6
   ```

   Der Ladevorgang selbst (Übertragen des Robo-Programms in das Dateisystem des ROBO TX Controllers) stellt sich am Bildschirm wie folgt dar:

   ```
   Loading program file ...
   Connecting to target on port COM3...OK.
   Read file "StopGo.bin" (376 bytes).
   Loading.....OK.
   ```

   Sie können nun den Namen des Robo-Programms, das gerade geladen worden ist, in der zweiten Zeile der Statusanzeige auf dem Display des ROBO TX Controllers

sehen. Sie können so eine Vielzahl von Programmen, die sich nur im Dateinamen eindeutig unterscheiden müssen, auf den ROBO TX Controller laden, nur begrenzt durch die Größe des Dateisystems. Eine Auswahl des zu startenden Programms erfolgt dann über das Datei-Menü und die Taster des ROBO TX Controllers.

3. Starten Sie nun das geladene Programm indem Sie den linken Taster „Start“ drücken. Einmal gestartet, können Sie es ebenfalls über den linken Taster „Stop“ wieder beenden, aber nur wenn neben dem Taster auch der Schriftzug „Stop“ zu sehen ist. Dies ist nicht der Fall, wenn das Robo-Programm selbst eine Ausgabe auf das Display macht, was den Status-Schirm überdeckt. In so einem Fall kann man das Beenden trotzdem erzwingen, wenn man beide Taster gleichzeitig drückt. Alternativ können Sie geladene Robo-Programme auch vom PC aus starten und stoppen, soweit noch eine Verbindung zu diesem besteht. Verwenden Sie hierzu die Batch-Files RUN.BAT und STOP.BAT (s.o.).

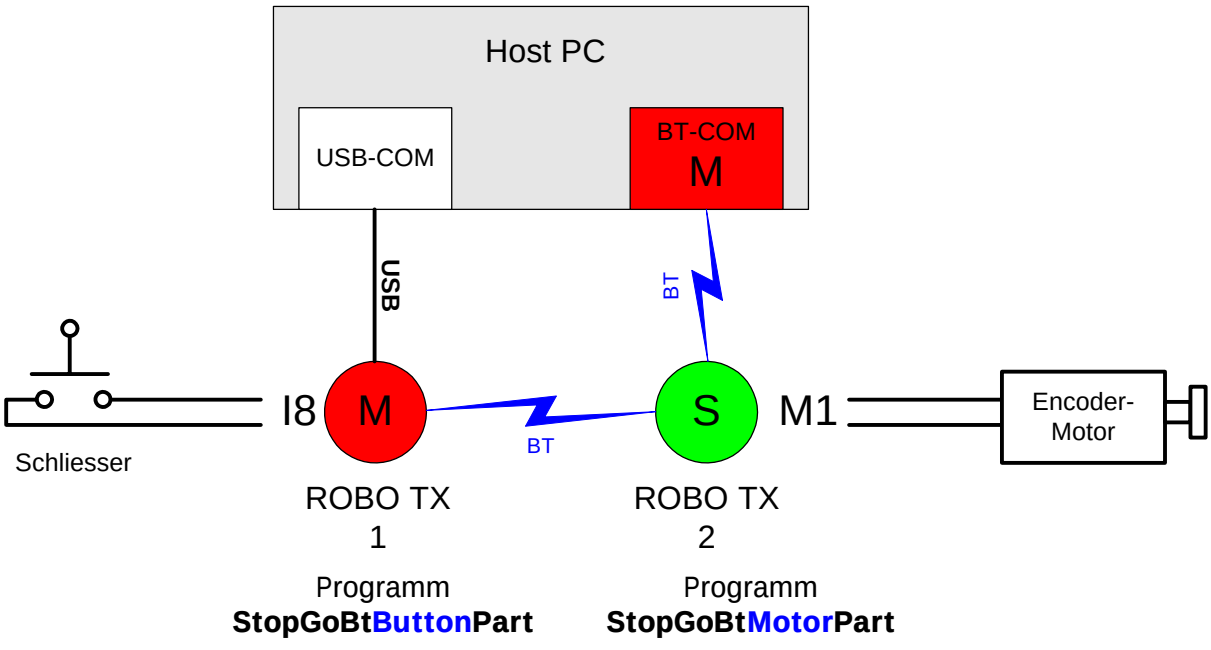
5.1 Beispiel mit 2 Controllern und Bluetooth Messaging

Im nachfolgenden Beispiel wird das Austauschen von Bluetooth-Nachrichten zwischen zwei Controllern mit C-Programmen demonstriert.

WICHTIGER HINWEIS: zum Thema „Bluetooth Messaging API“ wird die Lektüre des einführenden Kapitels 6 aus der Dokumentation des Pakets „PC_Programmierung_RoboTXC“ empfohlen. Dort ist detailliert und mit einer Vielzahl von Bildern erläutert, was bei der Bluetooth-Kommunikation zu beachten ist und welche Bedeutung die Begriffe MASTER (Aktiver Verbindungsaufbau, Connect) und SLAVE (Passiver Verbindungsaufbau, Listen) in diesem Zusammenhang haben.

Die im nachfolgenden Bild dargestellten Verbindungen zum Host-PC dienen hier nur dazu die Programm-Dateien der Robo-Programme auf die Controller zu laden. Dies kann über USB oder Bluetooth erfolgen, auch parallel wie im Bild gezeigt, aber – im Gegensatz zur Darstellung auf dem Bild – auch über nur eine einzige Verbindung, wenn z.B. beide Controller nacheinander über USB an den PC angeschlossen würden und die jeweils für sie bestimmte Programmdatei geladen bekommen.

Zu Ausführen der mitgelieferten Demo-Programme ist z.B. folgender Aufbau erforderlich:



Der MASTER-Controller (ROBO TX 1, auf dem Bild links) ist per USB an den PC angebunden, der SLAVE-Controller (ROBO TX 2, auf dem Bild rechts) hingegen per Bluetooth.

Das Robo-Programm StopGoBtButtonPart wir auf den MASTER-Controller geladen. Dazu verwendet man z.B. das Batch-File LOAD_RAMDISK.BAT, wie bei den anderen Demo-Programmen auch. Der SLAVE-Controller muß hingegen auf dem PC so eingerichtet werden, dass er einer COM-Port-Nummer auf dem PC zugeordnet wird. Diese COM-Port-Nummer wird in das Batch-File LOAD_RAMDISK_BT.BAT im Verzeichnis C:\ftMscCComp\Demo_C\Demo\StopGoBtMotorPart\ eingetragen (z.B. COM111). Damit kann das Robo-Programm StopGoBtMotorPart über Bluetooth auf den SLAVE-Controller geladen werden.

Vor dem Start der Programme sind die Bluetooth-Adressen der Controller in die Datei BT_addr.c im Verzeichnis `Common\C\` wie folgt einzutragen (hier beispielhaft):

```
UCHAR8 bt_address_table[BT_CNT_MAX][BT_ADDR_LEN] =
{
   {0x00, 0x13, 0x7B, 0x53, 0x10, 0xE7}, // Bluetooth-Adresse von ROBO TX 1 (MASTER)
   {0x00, 0x13, 0x7B, 0x52, 0xB2, 0x11}, // Bluetooth-Adresse von ROBO TX 2 (SLAVE)
};
```

Anschließend sind die Programme neu zu erzeugen, damit die richtigen Bluetooth-Adressen in diese übernommen werden.

Gestartet werden die Programme dann wie gehabt per Taster auf dem jeweiligen Controller:

Zuerst das Programm StopGoBtMotorPart auf dem SLAVE-Controller gestartet und anschließend das Programm StopGoBtMotorPart auf dem MASTER-Controller.

Betätigt man nun den Taster I8 auf ROBO TX 1 (MASTER), dann schickt das Programm StopGoBtButtonPart eine Bluetooth-Nachricht zu ROBO TX 2 (SLAVE) und der Motor auf diesem beginnt sich zu drehen. Lässt man den Taster wieder los, so stoppt der Motor. Das Programm StopGoBtMotorPart auf ROBO TX 2 schickt wiederum die Position des Encoder-Motors per Bluetooth-Nachricht an ROBO TX 1 zurück. Ist eine Position von 1000 erreicht so stoppt das Programm StopGoBtButtonPart und baut die Bluetooth-Verbindung zwischen ROBO TX 1 und ROBO TX 2 wieder ab.

Diverse LCD-Schirm-Ausgaben geben zusätzlich sichtbar Auskunft über den Connect-Status bzw. zeigen das Erreichen der Endposition (1000) des Motors an.

6 Eigene Robo-Programme schreiben

Der einfachste Weg eigene Robo-Programme zu schreiben ist eine Kopie eines der Demo-Programm-Verzeichnisse zu machen. Das Unterverzeichnis und das erzeugte Programm kann einen anderen Namen tragen (muss aber nicht zwingend).

6.1 Modifikation eines der Demo-Programme

Nehmen wir einmal an Sie wollen ein Programm mit dem Namen „MyProg“ erstellen, was einfach nur eine Lampe auf dem Ausgang O1 des ROBO TX Controllers einschalten soll. Wir kopieren hierfür das gesamte Verzeichnis „StopGo“ in dem Pfad "Demo_C\Demo" und benennen es nun „MyProg“. Anschließend benennen wir innerhalb des Verzeichnisses „MyProg“ die Datei „StopGo.c“ in „MyProg.c“ um. Schließlich verändern wir den Inhalt der Datei „param.mk“ wie folgt:

```
PROJ = MyProg
OBJS = MyProg.o
```

Hier ist unter **PROJ** (steht für „project“) der Output-Name (also der Dateiname des ausführbaren Robo-Programms) anzugeben. Die Endung „.bin“ wird automatisch durch den Make-Prozess angehängt. Unter OBJS (steht für „objects“) ist die Liste der Object-Files anzugeben, die in diesem Projekt zusammengefasst (gelinkt) werden sollen. Ist dies mehr als eins, so sind die Object-Files durch Leerzeichen zu trennen. Die Object-Files entstehen durch den C-Compiler bei der Übersetzung der C-Source-Files. Hinter OBJS führen Sie also einfach alle Namen der beteiligten C-Source-Files auf, die für die Robo-Programm-Erstellung erforderlich sind, nur dass Sie die Endungen ".c" durch ".o" ersetzen. IN unserem Beispiel hier ist unser Programm sehr klein und besteht nur aus einem C-Source-File „MyProg.c“. Die Liste hinter OBJS besteht also nur aus einem Eintrag: „MyProg.o“.

Jetzt wollen wir den Source Code von „MyProg.c“ auch noch verändern und werfen mal einen Blick auf die Struktur des Source-Codes.

Zunächst sei hier die Include-Datei „ROBO_TX_PRG.h“ genannt, die von jedem Robo-Programm verwendet werden sollte. Diese enthält wichtige Referenzen und Funktions-prototypen, aber vor allem auch eine weitere Include-Datei „ROBO_TX_FW.h“. Diese wiederum enthält alle Definitionen der sogenannten Transfer Area. Beide Include-Dateien befinden sich im Verzeichnis „Demo_C\Common“, da Sie von allen Demo-Programmen verwendet werden. Dort liegen auch weitere Dateien von allgemeiner Gültigkeit für alle Demo-Programme:

Makefile – Datei mit Einstellungen für das Make Tool.

prg_disp.c – Source Code für den „Program Dispatcher“, der den Hauptteil der Funktion “PrgDisp” enthält, welche der Einsprungpunkt für jedes Robo-Programm ist. „PrgDisp“ wird bei der lokalen Programmausführung (Download-Modus) periodisch in einem Zeitraster von 1 Millisekunde von der Firmware aufgerufen. Bei jedem Aufruf von „PrgDisp“ wird entschieden, ob die Funktion „PrgInit“ ihrerseits aufgerufen wird (nämlich nur beim ersten Aufruf von „PrgDisp“). Ansonsten wird immer die „PrgTic“ Funktion aufgerufen und der Rückgabewert (return code) von „PrgTic“ an die Firmware zurückgeliefert. Beide Funktionen „PrgInit“ und „PrgTic“ müssen in jedem Robo-Programm vorhanden sein.

ROBO_TX_FW.h – C Include File mit den Definitionen der Transfer Area Strukturen.

ROBO_TX_PRG.h – Haupt C Include File von jedem Robo-Programm-Projekt. Es muss immer vom Hauptteil eines Robo-Programms und auch allen weiteren C-Source-Files (soweit vorhanden) includiert sein, soweit diese ebenfalls auf die Transfer Area zugreifen wollen.

ld.lcf – Skript Datei für den Linker.

prg_bt.c – Hilfs-Funktionen für die Bluetooth Messaging Funktionen. Diese werden auch von den Demo-Programmen StopGoBtButtonPart und StopGoBtMotorPart verwendet.

prg_bt_addr.c – Liste der für das Bluetooth Messaging verwendeten Bluetooth-Adressen der beteiligten Controller. Diese werden auch von den Demo-Programmen StopGoBtButtonPart und StopGoBtMotorPart verwendet.

Zurück zur Struktur der Datei “MyProg.c”, können wir erkennen, dass die Funktionen “PrgInit” und PrgTic” immer enthalten sind. „PrgInit“ ist dabei eine Initialisierungsfunktion. Sie wird genau einmal beim Programmstart aufgerufen, etwa um die Eingänge und Ausgänge zu konfigurieren.

Nachfolgend ändern wir beispielsweise die Funktion „PrgInit“ wie folgt:

```
void PrgInit
(
    TA * p_ta_array,    // pointer to the array of transfer areas
    int ta_count        // number of transfer areas in array (equal to TA_COUNT)
)
{
    TA * p_ta = &p_ta_array[TA_LOCAL];

    // Configure M1 to be used as separate O1 and O2 outputs
    p_ta->config.motor[0] = FALSE;

    // Inform firmware that configuration was changed
    p_ta->state.config_id += 1;

    // Switch off the lamp O1
    p_ta->output.duty[0] = 0;
}
```

Tatsächlich hätten wir die Funktion “PrgInit” in diesem Fall auch komplett leer lassen können, da alle Initialisierungen, die wir hier vornehmen, auch dem Grundzustand der entsprechenden Variablen beim Programm-Start entsprechen. Der Grundzustand wird von der Firmware dadurch gesetzt, dass vor dem Programmstart alle Konfigurations- und Ausgangsfelder auf Null gesetzt werden.

„PrgTic“ ist die Hauptfunktion eines jeden Robo-Programms. Es wird vom „Program Dispatcher“ zyklisch jede Millisekunde aufgerufen. Die einzige Beschränkung die dem Code in der Funktion „PrgTic“ obliegt, ist die, dass diese nicht zuviel Zeit in Anspruch nehmen sollte. Es wird eine Ausführungsdauer von bis zu einer halben Millisekunde (500 Mikrosekunden) empfohlen. Andernfalls steht nicht mehr genug CPU-Zeit für die restliche Firmware zur Verfügung und ein geregeltes Echtzeit-Multitasking ist nicht mehr möglich.

Anders gesprochen: es sollte lang andauernde Vorgänge (worst case: unendliche Schleife) im Robo-Programm und ggf. in allen weiteren lokalen Funktionen, die dieses aufruft, unter allen Umständen vermieden werden. Sind längere Vorgänge unvermeidlich, so sind diese in

geeigneter Weise in einzelne Teilschritte zu zerlegen, so dass sie über mehrere Aufrufzyklen verteilt abgearbeitet werden können. Es gibt zur Ermittlung der Ausführungszeit die Firmware-Funktion „IsRunAllowed“ in der „Hook Table“ (siehe auch Kapitel 7 – Liste der Funktionsaufrufe) mit der man feststellen kann, ob noch Rechenzeit zur Verfügung steht. Liefert diese den Wert TRUE (1) zurück, so kann weiterer Code ausgeführt werden. Ansonsten ist die Funktion „PrgTic“ unter Zurückgabe eines Return-Codes zu verlassen.

Um das Ziel unseres Programms weiter zu verfolgen (d.h. eine Lampe auf Ausgang O1 einzuschalten) ändern wir beipielsweise die Funktion „PrgTic“ nun wie folgt:

```
int PrgTic
(
    TA * p_ta_array,    // pointer to the array of transfer areas
    int ta_count        // number of transfer areas in array (equal to TA_COUNT)
)
{
    int rc = 0x7FFF; // return code:
                     // 0x7FFF - program should be further called by the firmware;
                     // 0 - program should be normally stopped by the firmware;
                     // any other value is considered by the firmware as
                     // an error code and the program is stopped.
    TA * p_ta = &p_ta_array[TA_LOCAL];

    // Switch on the lamp O1 with the maximum brightness
    p_ta->output.duty[0] = DUTY_MAX;

    // Demonstration of usage of the IsRunAllowed function
    while (p_ta->hook_table.IsRunAllowed());

    return rc;
}
```

Von hier an ist für das Compilieren und Laden des Programms “MyProg” in bereits bekannter Weise zu verfahren (siehe Kapitel 5 – Kompilieren und Laden der Demo-Programme). Nach dem Laden und dem Start des Programms (über Taster) sollte die Lampe am Ausgang O1 an gehen.

6.2 Rückgabewerte (return codes)

Mit jedem zyklischen Aufruf muss die Hauptfunktion “PrgTic” einen Rückgabewert (return code) zurückliefern. Standardmäßig wird 0x7FFF zurückgeliefert, was sinngemäß heißt, dass das Robo-Programm wünscht weiterhin aufgerufen zu werden. Läuft das Robo-Programm nicht unendlich lang oder passiert ein (erkennbarer) Fehler, so sollte „PrgTic“ den Wert 0 zurückliefern, was die aufrufende Firmware dazu veranlasst das Programm zu beenden. Das hat dieselbe Wirkung wie das Drücken der „Stop“-Taste, nur das hierbei die Entscheidung für das Programmende aus dem Programm heraus gefällt wird.

Für Fehlermeldungen, die das Programm selbst melden will, wären auch alle anderen (von 0x7FFF verschiedenen) Rückgabewerte geeignet, die ebenfalls das Programm stoppen. Hierüber könnte man verschiedenartige Fehler unterscheiden. Die Return-Codes werden auf dem LCD-Schirm des ROBO TX Controllers nach Beendigung des Robo-Programms ausgegeben.

6.3 Ansprechen von Slave-Controllern über die RS485-Extension

Bisher sind wir immer davon ausgegangen, dass nur ein ROBO TX Controller betrieben wird. Dies kommt dadurch zum Ausdruck, dass die Transfer Area-Variable im Transfer Area Array der Index TA_LOCAL (Wert 0) verwendet wird:

```
TA * p_ta = &p_ta_array[TA_LOCAL];
```

Hat man hingegen eine Konstellation aus zwei (oder mehr) ROBO TX Controllern, die über ein RS485-Extension-Kabel verbunden sind, wobei ein Controller (mit dem auch der PC kommuniziert) der „Master“ ist und der andere der „Slave“, dann wird der Master weiterhin mit TA_LOCAL angesprochen und der Slave z.B. mit TA_EXT_1, wenn dieser auf die Slave-ID 1 konfiguriert wurde (siehe auch Handbuch ROBO TX Controller von fischertechnik). Dies würde man z.B. wie folgt programmieren:

```
TA * p_ta = &p_ta_array[TA_LOCAL];
TA * p_ta_ext = &p_ta_array[TA_EXT_1];
```

So ist es möglich von einem ROBO-Programm aus (das auf dem Master-Board läuft) mehrere ROBO TX Controller gleichzeitig zu steuern. Die Slaves werden dabei vom Master-Programm aus sozusagen ferngesteuert (wie Online-Modus vom PC aus). Es ist hierzu nicht erforderlich (und auch nicht möglich), dass gleichzeitig Robo-Programme auf den Slave-Boards laufen.

6.4 Warnhinweis

Mit dem C-Compiler ist es möglich Programme zu erzeugen, die den Prozessor im ROBO TX Controller beschädigen können! Im Gegensatz zu Programmen der fischertechnik RoboPro-Software kann mittels des eigenen C-Programms direkt auf die Hardwareports im Prozessor zugegriffen werden. Da bei diesem modernen Prozessor ein Großteil der internen Hardware durch Softwarebefehle konfiguriert wird, ist es denkbar, dass bei falschen Einstellungen der Prozessor mit der übrigen ROBO TX Controller Hardware nicht mehr richtig zusammen-arbeitet und im Extremfall sogar irreparable Schäden dadurch entstehen können. Insbesondere von Speicherzugriffen über Zeiger geht eine große Gefahr aus, da hierüber auch die Prozessorregister erreicht werden können.

Der Hersteller des ROBO TX Controllers muss daher Garantieansprüche bei beschädigten Prozessoren durch fehlerhafte C-Programme ablehnen.

7 Liste der Funktionsaufrufe (API)

Nachfolgend sind alle Aufrufe externer Betriebssystemfunktionen aufgelistet, auf die ein Robo-Programm über die „Hook Table“ zugriff hat.

7.1 IsRunAllowed

*BOOL32 (*IsRunAllowed) (void)*

Diese Funktion gibt dem Robo-Programm darüber Auskunft, ob im aktuellen Ausführungs-Zeit-Slot noch Rechenzeit zur Verfügung steht.

Return:	TRUE (1)	Es steht noch Rechenzeit zur Verfügung. Das Programm kann mit seiner Ausführung fortfahren.
	FALSE (0)	Es steht keine Rechenzeit mehr zur Verfügung. Das Programm muss den Funktionsaufruf „PrgTic“ mit einem Rückgabewert sofort beenden.

7.2 GetSystemTime

*UINT32 (*GetSystemTime) (enum TimerUnit unit)*

Diese Funktion liefert die Systemzeit seit dem Start des ROBO TX Controllers in Sekunden, in Millisekunden oder in Mikrosekunden zurück, je nach Wert des Parameters „unit“. Sie kann als absolute Zeitreferenz dienen.

Aufruf:	enum TimerUnit unit –	TIMER_UNIT_SECONDS (2) TIMER_UNIT_MILLISECONDS (3) TIMER_UNIT_MICROSECONDS (4)
Return:	UINT32 time	Ein 32-Bit-Integer-Wert mit der Anzahl akkumulierter Zeiteinheiten (abhängig vom Parameter „unit“) seit Systemstart.

7.3 DisplayMsg

*void (*DisplayMsg) (struct ta_s * p_ta, char * p_msg)*

Diese Funktion erlaubt es einem Robo-Programm eine Textnachricht (max. 32 Zeichen) auf dem Display des ROBO TX Controllers anzuzeigen.

Aufruf:	struct ta_s * p_ta	Zeiger auf gültige Transfer Area
	Char * msg	Zeiger auf die anzuzeigende Textnachricht (C String mit 0 terminiert). Wird hier NULL gesetzt, wird einer vorher angezeigte Nachricht wieder gelöscht und auf dem Display ist wieder der Status-Schirm zu sehen.

7.4 IsDisplayBeingRefreshed

*BOOL32 (*IsDisplayBeingRefreshed) (struct ta_s * p_ta)*

Diese Funktion gibt dem Robo-Programm darüber Auskunft, ob weitere Ausgaben auf das Display gemacht werden können oder noch nicht, etwa weil die vorherige Ausgabe noch in den Display-Puffer geschrieben wird.

Aufruf:	struct ta_s * p_ta	Zeiger auf gültige Transfer Area
Return:	TRUE (1)	Es kann auf das Display geschrieben werden.
	FALSE (0)	Es kann noch nicht wieder auf das Display geschrieben werden.

7.5 Standard C-Library Funktionen (Liste mit 21 Funktionen)

Wie in allen C-Programmierumgebungen gibt es eine Reihe von Standard C-Library Funktionen, die universell verwendbar sind. Die verfügbaren Funktionen sind nachfolgend nur aufgelistet (siehe auch Header File ROBO_TX_FW.H) und nicht detailliert beschrieben. Ein solche Beschreibung findet man bei Bedarf in jedem einschlägigen C-Programmierhandbuch oder auch im Internet, z.B. unter dem nachfolgende genannten Link (ohne Gewähr für die die Aktualität des Links oder die Richtigkeit des Inhaltes):

http://www.acm.uiuc.edu/webmonkeys/book/c_guide/

```
INT32     (*sprintf)     (char * s, const char * format, ...);
INT32     (*memcmp)      (const void * s1, const void * s2, UINT32 n);
void     *(*memcpy)      (void * s1, const void * s2, UINT32 n);
void     *(*memmove)     (void * s1, const void * s2, UINT32 n);
void     *(*memset)      (void * s, INT32 c, UINT32 n);
char     *(*strcat)      (char * s1, const char * s2);
char     *(*strncat)     (char * s1, const char * s2, UINT32 n);
char     *(*strchr)      (const char * s, INT32 c);
char     *(*strrchr)     (const char * s, INT32 c);
INT32     (*strcmp)      (const char * s1, const char * s2);
INT32     (*strncmp)     (const char * s1, const char * s2, UINT32 n);
INT32     (*stricmp)     (const char * s1, const char * s2);
INT32     (*strnicmp)    (const char * s1, const char * s2, UINT32 n);
char     *(*strcpy)      (char * s1, const char * s2);
char     *(*strncpy)     (char * s1, const char * s2, UINT32 n);
UINT32    (*strlen)      (const char * s);
char     *(*strstr)      (const char * s1, const char * s2);
char     *(*strtok)      (char * s1, const char * s2);
char     *(*strupr)      (char * s);
char     *(*strlwr)      (char * s);
INT32     (*atoi)        (const char * nptr);
```

7.6 Bluetooth Messaging API

Die nachfolgenden beschriebenen Funktionen gehören zur Bluetooth Messaging API. Zur Einführung in die Thematik wird auch die Lektüre des Kapitels 6 aus der Dokumentation des Pakets „PC_Programmierung_RoboTXC“ sowie ebenfalls des Kapitels 6 aus Dokumentation zur Windows Library (ebenfalls Teil des Pakets „PC_Programmierung_RoboTXC“) empfohlen, etwa zur Erläuterung der Begriffe Kanalnummer (Kanalindex) oder Rufbereitschaft und Empfangsbereitschaft.

7.6.1 BtConnect

*void (*BtConnect) (UINT32 channel,*
*UCHAR8 * btaddr,*
P_CB_FUNC p_cb_func);

Aktiver Aufbau einer Bluetooth-Verbindung zu einer durch die Bluetooth-Ziel-Adresse eindeutig beschriebenen Bluetooth-Gegenstelle. Über die Callback-Funktion wird das Resultat des Verbindungsversuches asynchron gemeldet. Nach erfolgreichem Verbindungsaufbau wird diese Verbindung in der Firmware mit der angegebenen Kanalnummer (channel) für weitere Zugriffe verwaltet. Um mehrere Bluetooth-Verbindungen gleichzeitig aktiv aufzubauen, kann diese Funktion mehrfach aufgerufen werden (jeweils mit eigener Kanalnummer).

Einschränkungen:

Die Funktion BtConnect meldet per Callback-Aufruf einen Fehler zurück (**BT_CHANNEL_BUSY**), wenn auf derselben Kanalnummer bereits eine Rufbereitschaft über die Funktion BtStartListen() registriert wurde. Dies soll verhindern, dass Doppelverbindungen zwischen zwei Controllern entstehen können, dadurch dass ein Programm von derselben Gegenstelle sowohl Rufe annimmt, als auch dorthin aufbaut.

Aufruf:

channel	-	Kanalnummer (Index), unter der die Verbindung in allen nachfolgenden Aufrufen verwaltet wird. Wird von der aufrufenden Applikation festgelegt (1 – 8).
*btaddr	-	Pointer auf die dazugehörige Bluetooth-Adresse (6 Byte)
p_cb_func	-	Callback-Funktion, die das Ergebnis des Verbindungsaufbaus meldet (siehe auch Kapitel 7.6.9)

Parameter der Callback-Funktion:

*p_data	-	Pointer auf Datenstruktur BT_CB

Datenstruktur:

```
typedef struct bt_cb_s {      //  4 bytes
      UINT16      chanIdx;    //  Kanalindex
      UINT16      status;     //  Ergebnis des Verbindungsaufbaus
} BT_CB;
```

7.6.2 BtStartListen

*void (*BtStartListen) (UINT32 channel,*
*UCHAR8 * btaddr,*
P_CB_FUNC p_cb_func);

Passiver Aufbau einer Bluetooth-Verbindung von einer durch die angegebene Bluetooth-Quell-Adresse eindeutig beschriebenen Bluetooth-Gegenstelle. Hiermit signalisiert der ROBO TX Controller seine Bereitschaft genau eine Bluetooth-Verbindung von der genannten Gegenstelle anzunehmen (aktivierte Rufbereitschaft). Über die Callback-Funktion wird die eingehende Verbindung asynchron gemeldet, sowie dieser Zustand eintritt, ebenso wie eventuell auftretende Fehler. Nach erfolgtem Verbindungsaufbau wird die Verbindung in der Firmware mit der angegebenen Kanalnummer (channel) für weitere Zugriffe verwaltet. Um mehrere Bluetooth-Verbindungen gleichzeitig annehmen zu können, kann diese Funktion mehrfach aufgerufen werden (jeweils mit eigener Kanalnummer).

Einschränkungen:

Die Funktion BtStartListen meldet per Callback-Aufruf einen Fehler zurück (**BT_CHANNEL_BUSY**), wenn auf demselben Kanalindex bereits eine aktive Verbindung besteht, die durch BtConnect() zustande kam. Dies soll verhindern, dass Doppelverbindungen zwischen zwei Controllern entstehen können, dadurch dass ein Programm von derselben Gegenstelle sowohl Rufe annimmt, als auch dorthin aufbaut.

Aufruf:

channel	-	Kanalnummer (Index), unter der die Verbindung in allen nachfolgenden Aufrufen verwaltet wird. Wird von der aufrufenden Applikation festgelegt (1 – 8).
*btaddr	-	Pointer auf die dazugehörige Bluetooth-Adresse (6 Byte)
p_cb_func	-	Callback-Funktion, die das Ergebnis des Verbindungsaufbaus meldet (siehe auch Kapitel 7.6.9)

Parameter der Callback-Funktion:

*p_data	-	Pointer auf Datenstruktur BT_CB

Datenstruktur:

```
typedef struct bt_cb_s {      //  4 bytes
      UINT16      chanIdx;    //  Kanalindex
      UINT16      status;     //  Ergebnis des Verbindungsaufbaus
} BT_CB;
```

7.6.3 BtStopListen

*void (*BtStopListen) (UINT32 channel,*
P_CB_FUNC p_cb_func);

Diese Funktion deaktiviert eine vorher durch die Funktion BtStartListen aktivierte Rufbereitschaft auf der angegebenen Kanalnummer. Das bedeutet, dass ab sofort keine Bluetooth-Verbindungen mehr angenommen werden. Allerdings wird eine bereits bestehende Bluetooth-Verbindung auf dieser Kanalnummer durch diesen Funktionsaufruf nicht unterbrochen, sondern bleibt bestehen. Die Beendigung der Rufbereitschaft wirkt sich in diesem Fall nur auf den Zeitraum nach Beendigung der bestehenden Verbindung aus (es wird keine Verbindung mehr erneut angenommen bis nicht wieder BtStartListen() aufgerufen wurde).

Aufruf:

channel - Kanalnummer (Index 1 – 8).
p_cb_func - Callback-Funktion, die das Ergebnis des Verbindungsabbaus meldet (siehe auch Kapitel 7.6.9)

Parameter der Callback-Funktion:

*p_data - Pointer auf Datenstruktur BT_CB

Datenstruktur:

```
typedef struct bt_cb_s {      //  4 bytes
      UINT16      chanIdx;    //  Kanalindex
      UINT16      status;     //  Ergebnis des Verbindungsabbaus
} BT_CB;
```

7.6.4 BtDisconnect

*void (*BtDisconnect) (UINT32 channel,*
P_CB_FUNC p_cb_func);

Abbau einer durch die angegebene Kanalnummer referenzierten aktiven Bluetooth-Verbindung. Hierbei ist es unerheblich, ob die Verbindung durch einen aktiven oder durch einen passiven Verbindungsaufbau zustande gekommen ist. Über die Callback-Funktion wird das Resultat des Verbindungsabbauversuches asynchron gemeldet.

Aufruf:

channel - Kanalnummer (Index 1 – 8).
p_cb_func - Callback-Funktion, die das Ergebnis des Verbindungsabbaus meldet (siehe auch Kapitel 7.6.9)

Parameter der Callback-Funktion:

*p_data - Pointer auf Datenstruktur BT_CB

Datenstruktur:

```
typedef struct bt_cb_s {      //  4 bytes
      UINT16      chanIdx;    //  Kanalindex
      UINT16      status;     //  Ergebnis des Verbindungsabbaus
} BT_CB;
```

7.6.5 BtSend

*void (*BtSend) (UINT32 channel,*
UINT32 len,
*UCHAR8 *p_msg,*
P_CB_FUNC p_cb_func);

Schreiben von Daten auf eine durch die Kanalnummer referenzierte aktive Bluetooth-Verbindung. Mit dem Aufruf wird ein Zeiger auf einen Schreibdatenpuffer (p_msg) und eine Länge (len) übergeben. Die Funktion wird aus dem Puffer die angegebene Anzahl von Bytes auslesen. Nach dem Funktionsaufruf kann der Schreibdatenpuffer wieder freigegeben werden. Über die Callback-Funktion wird das Resultat des Sendeversuches asynchron gemeldet.

Aufruf:

channel	- Kanalnummer (Index 1 – 8).
len	- Länge der Sendedaten im Sendebuffer (max. 255 Zeichen)
*p_msg	- Pointer auf Sendebuffer mit den Sendedaten (Nachricht)
p_cb_func	- Callback-Funktion, die das Ergebnis meldet (siehe auch Kapitel 7.6.9)

Parameter der Callback-Funktion:

*p_data	- Pointer auf Datenstruktur BT_CB

Datenstruktur:

```
typedef struct bt_cb_s {      //  4 bytes
      UINT16      chanIdx;    //  Kanalindex
      UINT16      status;     //  Ergebnis der Operation
} BT_CB;
```

7.6.6 BtStartReceive

*void (*BtStartReceive) (UINT32 channel,*
P_RECV_CB_FUNC p_cb_func);

Mit dieser Funktion wird die Empfangsbereitschaft von Daten (Nachrichten) auf eine durch die Kanalnummer referenzierte aktive Bluetooth-Verbindung angezeigt. Beim Empfang einer Nachricht wird die Callback-Funktion aufgerufen, die einen Zeiger auf die empfangen Daten und die Länge der Daten enthält. Zur Empfangsbereitschaft muss diese Funktion nur 1 mal (pro Kanalnummer) aufgerufen werden. Eingehende Daten rufen entsprechend mehrfach die Callback-Funktion auf.

Aufruf:

channel	-	Kanalnummer (Index 1 – 8).
p_cb_func	-	Callback-Funktion, die das Ergebnis (Nachricht) meldet (siehe auch Kapitel 7.6.9)

Parameter der Callback-Funktion:

*p_data	-	Pointer auf Datenstruktur BT_RECV_CB

Datenstruktur:

```
typedef struct bt_receive_cb_s
{
    UINT16          chan_idx;         //  Kanalindex
    UINT16          status;           //  Ergebnis der Operation
    UINT16          msg_len;          //  Länge der empfangenen Daten
    UCHAR8          msg[BT_MSG_LEN];  //  Puffer für empfangenen Daten
} BT_RECV_CB;
```

7.6.7 BtStopReceive

*void (*BtStopReceive) (UINT32 channel,*
P_RECV_CB_FUNC p_cb_func);

Deaktivierung der durch die Funktion *BtStartReceive()* aktivierten Empfangsbereitschaft. Durch Aufruf dieser Funktion wird der Datenempfang auf der Verbindung mit der angegebenen Kanalnumer unterbunden. Zwischenzeitlich dennoch eingehende Daten auf der ggf. noch bestehenden Bluetooth-Verbindung, werden vom ROBO TX Controller verworfen.

Aufruf:

channel - Kanalnummer (Index 1 – 8).
p_cb_func - Callback-Funktion, die das Ergebnis meldet (siehe auch Kapitel 7.6.9)

Parameter der Callback-Funktion:

*p_data - Pointer auf Datenstruktur BT_RECV_CB

Datenstruktur:

```
typedef struct bt_receive_cb_s
{
    UINT16          chan_idx;         //  Kanalindex
    UINT16          status;           //  Ergebnis der Operation
    UINT16          msg_len;          //  (hier ohne Bedeutung)
    UCHAR8          msg[BT_MSG_LEN];  //  (hier ohne Bedeutung)
} BT_RECV_CB;
```

7.6.8 BtAddrToStr

*char *(*BtAddrToStr) (UCHAR8 *btaddr,*
*char *str);*

Wandelt einen Bluetooth-Adresse in 6-Byte-Darstellung (btaddr) in einen lesbaren C-String (str) im Format „xx:xx:xx:xx:xx:xx“ um, etwa zur geeigneten Ausgabe auf dem LCD-Schirm.

Aufruf:

*btaddr - Pointer auf die dazugehörige Bluetooth-Adresse (6 Byte)
*str - Pointer auf Puffer für Ergebnis-String

Return-Wert:

*str - Pointer auf Ergebnis-String

7.6.9 Zusammenfassung der Statuswerte in den Callback-Funktionen

Auflistung der möglichen Statuswerte: (`enum CB_BtStatus`), siehe auch Header-Datei ROBO_TX_FW.H

Status	Bedeutung
0	= **BT_SUCCESS** , Aktion erfolgreich
1	= **BT_CON_EXIST** , Bereits verbunden
2	= **BT_CON_SETUP** , Verbindungsaufbau zu dieser BT-Adresse wird bereits aktiv durchgeführt
3	= **BT_SWITCHED_OFF** , Verbindung fehlgeschlagen: Bluetooth ist lokal per Konfiguration abgeschaltet
4	= **BT_ALL_CHAN_BUSY** , Verbindung fehlgeschlagen: kein Bluetooth-Kanal lokal mehr frei
5	= **BT_NOT_ROBOTX** , Verbindung fehlgeschlagen: nicht verbindbares fremdes BT-Gerät (kein ROBO TX Controller)
6	= **BT_CON_TIMEOUT** , fehlgeschlagen: Timeout, kein Gerät unter dieser Adresse erreichbar (Timeout)
7	= **BT_CON_INVALID**, es existiert keine aktive Verbindung mit der angegebenen Kanalnummer (Index).
8	= **BT_CON_RELEASE**, Verbindungsabbau zu dieser BT-Adresse ist bereits aktiviert und wird durchgeführt
9	**= BT_LISTEN_ACTIVE**, die Listen-Funktion ist für den angegebenen Kanalindex bereits aktiviert worden.
10	**= BT_RECEIVE_ACTIVE**, die Receive-Funktion ist bereits aktiviert worden.
11	**= BT_CON_INDICATION**, Signalisierung eines Verbindungsaufbaus auf der passiven Seite. Eine Bluetooth-Verbindung von der vorgegebenen Bluetooth-Adresse ist eingegangen.
12	**= BT_DISCON_INDICATION**, Signalisierung eines passiven Verbindungsabbaus (z.B. von der Gegenseite ausgelöst). Die Bluetooth-Verbindung besteht nicht mehr.
13	**= BT_MSG_INDICATION**, Signalisiert den Empfang einer Bluetooth-Nachricht (Message) von der Gegenstelle
14	**= BT_CHANNEL_BUSY**, der angegebene Kanalindex ist bereits registriert (Rufbereitschaft) bzw. in Verwendung (aktive Verbindung).
15	**= BT_BTADDR_BUSY**, zu dieser Bluetooth Adresse existiert bereits eine Rufbereitschaft oder eine aktive Verbindung über eine andere Kanalnummer (Index).
16	**= BT_NO_LISTEN_ACTIVE**, auf der Gegenstelle ist keine Listen-Funktion aktiviert worden, kein Verbindungsaufbau möglich.

7.7 I²C API

Die nachfolgenden beschriebenen Funktionen gehören zur I²C API. Zur Einführung in die Thematik wird auch die Lektüre des Kapitels 7 aus der Dokumentation des Pakets „PC_Programmierung_RoboTXC“ empfohlen

Wichtige Hinweise:

Die I²C-Geräteadresse (Device Address) ist laut I²C-Spezifikation mit nur 7 Bit anzugeben (Wertebereich: 0 – 127).

Weiterhin sind die I²C-Geräteadressen 80 und 84 (0x50 und 0x54) reserviert für ein internes EEPROM des ROBO TX-Controllers. Der Zugriff auf diese Adressen wird von der API nicht erlaubt. Die I²C-Geräteadressen 81-83 und 85-87 (0x51-0x53 und 0x55-0x57) sind ebenfalls von Speicherbereichen desselben EEPROMs belegt, werden aber von der Firmware nicht verwendet. Hier kann es (muss aber nicht) bei externen I2C-Geräten, die eine dieser Adressen verwenden, zu Buszugriffskonflikten kommen.

7.7.1 I2cRead

void I2cRead (UCHAR8 devaddr,
UINT32 offset,
UCHAR8 flags,
P_I2C_CB_FUNC p_cb_func);

Es wird auf dem I²C-Bus an der Geräteadresse „devaddr“ und ggf. innerhalb des Gerätes an der Unteradresse „offset“ ein Byte (8-Bit) oder zwei Bytes (16-Bit) gelesen. Mit dem Parameter „flags“ werden die Adressierung, die Datenbusbreite, die Byte-Reihenfolge (nur bei 16-Bit-Werten), das Verhalten bei Busfehlern und die Zugriffsgeschwindigkeit festgelegt. Über die Callback-Funktion wird das Ergebnis des Lesezugriffs sowie das gelesene Datum asynchron zurückgegeben, sowie dieser Zustand eintritt.

Aufruf: UCHAR8 devaddr - I2C Geräteadresse (Device Address)

UINT32 offset - wenn innerhalb des Gerätes eine Adressierung notwendig ist, dann wird diese interne Adresse hier übergeben. Der Wert von „flags“ spezifiziert die Länge der internen Adresse in Bit 0..1.

UCHAR8 flags - Verwendete Zugriffsflaggen:

Bit 0..1	Adressierung	00: keine („Offset“ ungültig) 01: 8-Bit Adressierung 10: 16-Bit Adressierung MSB zuerst 11: 16-Bit Adressierung LSB zuerst
Bit 2..3	Datenbreite	00: - *nicht erlaubt* - 01: 8-Bit Daten (1 Byte) 10: 16-Bit Daten (2 Bytes) MSB zuerst 11: 16-Bit Daten (2 Bytes) LSB zuerst
Bit 4	KeepOpen	0: normaler Zugriff 1: schneller Zugriff ohne STOP/START
Bit 5..6	Error Mask = Verhalten bei Busfehler	00: Abbruch 01: Bis zu 10x wiederholen 10: Wiederholen bis Erfolg 11: - *nicht erlaubt* -
Bit 7	Taktfrequenz	0: standard (100 kHz) 1: fast (400 kHz)

p_cb_func - Callback-Funktion, die das Ergebnis der Operation asynchron zurückmeldet.

Return: (kein)

Return-Werte der Callback-Funktion: Pointer auf Datenstruktur I2C_CB

Datenstruktur:

```
typedef struct {            // 4 bytes
      UINT16      value;    // Gelesenes Datum (bei 8-Bit nur LSByte gültig)
      UINT16      status;   // Ergebnis der I2C-Bus-Operation (siehe 7.7.3)
} I2C_CB;
```

7.7.2 I2cWrite

void I2cWrite (UCHAR8 devaddr,
UINT32 offset,
UINT16 data,
UCHAR8 flags,
P_I2C_CB_FUNC p_cb_func);

Es wird auf dem I²C-Bus an der Geräteadresse „devaddr“ und ggf. innerhalb des Gerätes an der Unteradresse „offset“ ein Byte (8-Bit) oder zwei Bytes (16-Bit) geschrieben. Mit dem Parameter „flags“ werden die Adressierung, die Datenbreite, die Byte-Reihenfolge (nur bei 16-Bit-Werten), das Verhalten bei Busfehlern und die Zugriffsgeschwindigkeit festgelegt. Über die Callback-Funktion wird das Ergebnis des Schreibzugriffs asynchron zurückgegeben, sowie dieser Zustand eintritt.

Aufruf:

UCHAR8 devaddr - I2C Geräteadresse (Device Address)

UINT32 offset - wenn innerhalb des Gerätes eine Adressierung notwendig ist, dann wird diese interne Adresse hier übergeben. Der Wert von „flags“ spezifiziert die Länge der internen Adresse in Bit 0..1.

UINT16 data - Datum (8-Bit oder 16-Bit), das geschrieben werden soll

UCHAR8 flags - Verwendete Zugriffsflaggen:

Bit 0..1	Adressierung	00: keine („Offset“ ungültig) 01: 8-Bit Adressierung 10: 16-Bit Adressierung MSB zuerst 11: 16-Bit Adressierung LSB zuerst
Bit 2..3	Datenbreite	00: - *nicht erlaubt* - 01: 8-Bit Daten (1 Byte) 10: 16-Bit Daten (2 Bytes) MSB zuerst 11: 16-Bit Daten (2 Bytes) LSB zuerst
Bit 4	KeepOpen	0: normaler Zugriff 1: schneller Zugriff ohne STOP/START
Bit 5..6	Error Mask = Verhalten bei Busfehler	00: Abbruch 01: Bis zu 10x wiederholen 10: Wiederholen bis Erfolg 11: - *nicht erlaubt* -
Bit 7	Taktfrequenz	0: standard (100 kHz) 1: fast (400 kHz)

p_cb_func - Callback-Funktion, die das Ergebnis der Operation asynchron zurückmeldet.

Return: (kein)

Return-Werte der Callback-Funktion: Pointer auf Datenstruktur I2C_CB

Datenstruktur:

```
typedef struct {            // 4 bytes
      UINT16      value;    // Geschriebenes Datum wiederholt
      UINT16      status;   // Ergebnis der I2C-Bus-Operation (siehe 7.7.3)
} I2C_CB;
```

7.7.3 Callback Status

Return-Werte beim Callback (status)

Wert von status	Bedeutung	
0	I2C-Operation erfolgreich ausgeführt	I2C_SUCCESS
1	I2C-Lesefehler	I2C_READ_ERROR
2	I2C-Schreibfehler	I2C_WRITE_ERROR

8 Update der ROBO TX Controller Firmware

Die Firmware des ROBO TX Controllers kann auf mehrere Arten ein Update erfahren:

- Über ROBOPro (wird i.d.R. automatisch beim ersten Verbindungsaufbau zum TX-C angeboten, soweit ein Update erforderlich ist)
- Mit dem Repair-Tool (siehe separates Paket mit entsprechender Bezeichnung)

9 Versionshistorie dieses Dokument

Version	Datum	Autor	Änderungen Bemerkungen
1.0	22.06.2010	Alexey Kucherenko Peter Duchemin	- Erste Fassung - Passend zur Firmware V1.18
1.1	24.01.2012	Peter Duchemin	- Neue Funktionen, u.a. Standard C-Library und Bluetooth Messaging API - Erweiterung der Demo-Programme - Passend zur Firmware V1.24
1.2	25.04.2012	Peter Duchemin	- Neue Funktionen I^2C API - Erweiterung der Demo-Programme - Passend zur Firmware V1.30