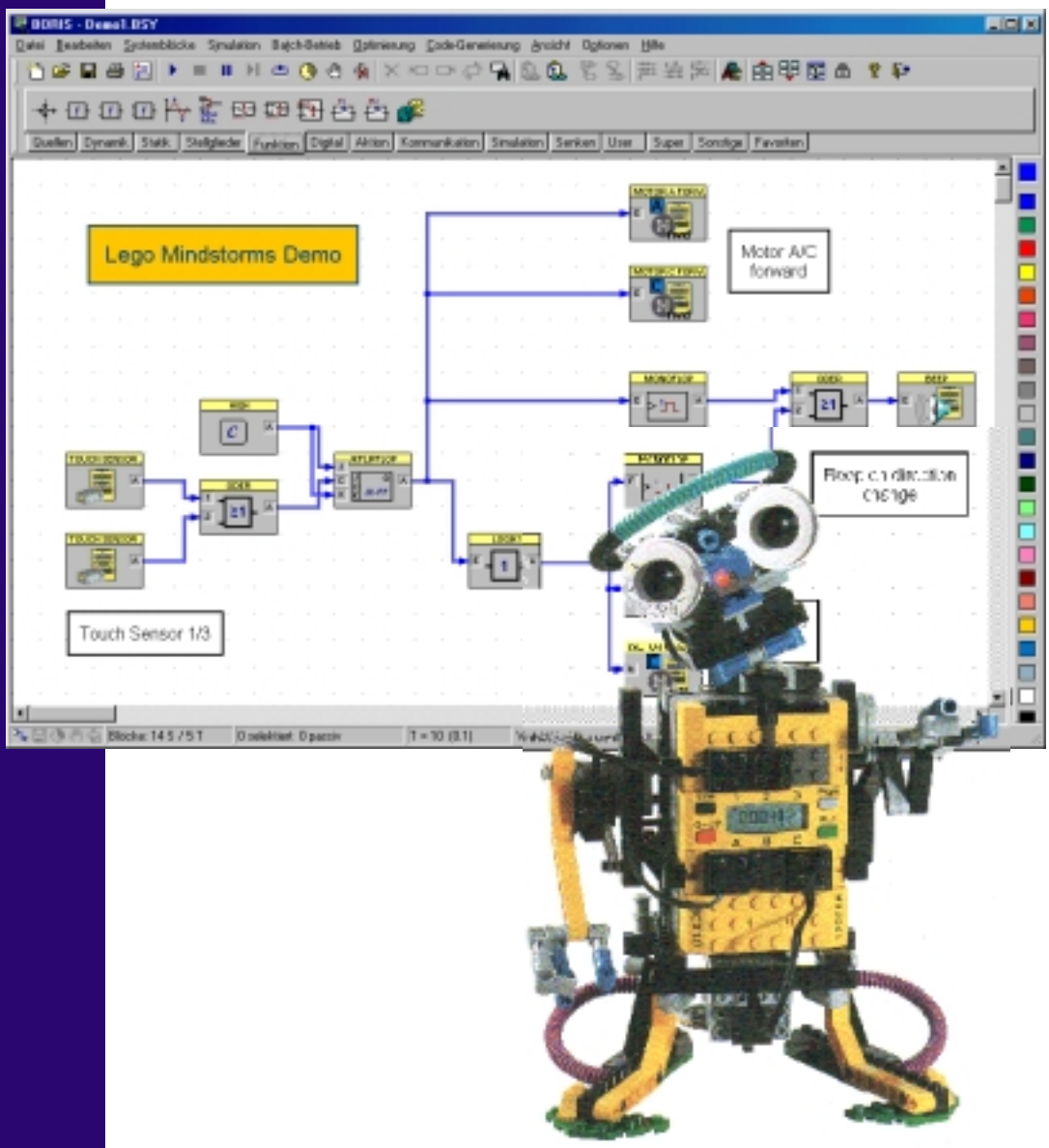
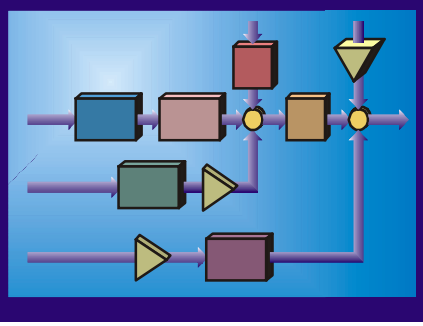
WinFACT



WinFACT

LEGO Mindstorms Toolbox



Inhalt

1. EINFÜHRUNG UND INSTALLATION 3

LEISTUNGSUMFANG DER TOOLBOX 3
INSTALLATION ALLER ERFORDERLICHEN KOMPONENTEN 4

2. INSTALLATION DES LEGOS-BETRIEBSSYSTEMS 5

3. PROGRAMMIERUNG DES RCX AUS BORIS 7

SCHRITT 1: GRUNDLEGENDE EINSTELLUNGEN DES AUTOCODE-GENERATORS 7
SCHRITT 2: ERSTELLEN DER SYSTEMSTRUKTUR 9
SCHRITT 3: WAHL DER SIMULATIONSPARAMETER 12
SCHRITT 4: GENERIERUNG DES C-CODES, COMPILIEREN UND ÜBERTRAGEN AUF DEN RCX 13
SCHRITT 5: STARTEN DES PROGRAMMS 14
ZUSAMMENFASSUNG DER VORGEHENSWEISE 14

4. DIE I/O-BESCHREIBUNGSDATEI `WFMINDSTORMS.C` 16

MAIN - ROUTINE 16
LCD-AUSGABE 16
SOUNDAUSGABE 18
MOTOR-STEUERUNG 18
LICHT-AUSGÄNGE 25
SENSOREN 26
RCX-BEDIENTASTEN 30
SONSTIGES 31

1. Einführung und Installation

Leistungsumfang der Toolbox

Vergleichbar mit der elektrischen Eisenbahn repräsentiert auch das LEGO-System ein "Spielzeug", welches sowohl technikbegeisterte Kinder und Jugendliche als auch Erwachsene bis ins hohe Alter immer wieder fasziniert. Ein wesentlicher Grund dafür liegt in der Flexibilität des Systems, mit der sich aus den gleichen Grundkomponenten immer wieder neue Modelle aufbauen lassen.

Mit dem *Mindstorms* System bietet LEGO seit einigen Jahren in Fortsetzung der LEGO-Technik-Reihe eine Sammlung von Systembaukästen an, aus denen sich einfache Roboter bis hin zu komplexen Maschinen bauen und mit Hilfe der mitgelieferten Software auch programmieren lassen. Mit der Lego Mindstorms Toolbox lässt sich das LEGO Mindstorms System nunmehr unmittelbar und praktisch ohne jegliche Programmierkenntnisse aus der komfortablen Bedienoberfläche des WinFACT-Moduls BORIS® heraus konfigurieren.

Den Kern des LEGO Mindstorms Systems bildet der sogenannte *RCX* Baustein in der Form eines "überdimensionalen" Legosteins. Dieser enthält einen 8-Bit-Microcontroller der Firma Hitachi (H8/3292) mit einer Taktfrequenz von 16 MHz. Damit eignet sich das System in Verbindung mit BORIS in hervorragender Weise dazu, den Entwicklungs- und Implementierungsprozess von Embedded Systems in der Steuerungs- und Regelungstechnik widerzuspiegeln oder z. B. im Rahmen der Ausbildung Grundlagen der grafischen Programmierung von Robotern zu erlernen. Auch als Demonstrationsobjekt auf Messen oder sonstigen Präsentationen ist der LEGO-Roboter immer ein "gerngesehener Gast".

Neben der mit den LEGO Mindstorms Baukästen gelieferten grafischen Programmieroberfläche, die sich insbesondere an computerunerfahrene Anwender (speziell Kinder und Jugendliche) wendet, steht mittlerweile eine Reihe frei verfügbarer Entwicklungssysteme auf Open-Source-Basis zur Verfügung, die eine wesentlich flexiblere Programmierung des RCX und damit auch die Realisierung komplexerer Anwendungen erlauben. Dies gilt vor allem für das Betriebssystem legOS, das eine vollständige C-Programmierumgebung mit Compiler und Echtzeit-Betriebssystem zur Verfügung stellt und kostenlos aus dem Internet geladen werden kann.

Die *LEGO Mindstorms Toolbox für WinFACT* stellt nunmehr das Bindeglied zwischen dem RCX und legOS auf der einen Seite und dem WinFACT-Modul BORIS mit seinem AutoCode-Generator auf der anderen Seite dar, indem sie den BORIS-AutoCode-Generator um die Funktionalität des LEGO RCX erweitert. Mit ihrer Hilfe lässt sich das gewünschte Verhalten des Roboters somit grafisch (d. h. ohne Schreiben einer einzigen Zeile Code) unter der komfortablen Bedienoberfläche von BORIS "programmieren" und dann auf Knopfdruck in C-Code überführen, compilieren und über die Infrarot-Schnittstelle auf den RCX laden - ein Vorgang, der selbst bei komplexen Strukturen nur wenige Sekunden dauert!

In der vorliegenden Version der *LEGO Mindstorms Toolbox* werden folgende Komponenten unterstützt:

- Antriebsmotoren A/B/C (Ein/Aus, Drehrichtung und -geschwindigkeit)
- Lichtquellen A/B/C
- Sensoren 1/2/3 (Rohdaten, Berührungssensor, Lichtsensor, Rotationssensor, Eigenbauten)

- RCX-Bedienschalter (Program und View)
- Ausgabe von Signalwerten (Integer, Hex, Float) auf dem integrierten LCD
- Soundausgabe
- Batterie-Ladezustand

Jeder Blocktyp besitzt innerhalb der BORIS-Systemstruktur ein individuelles Icon, sodass seine Funktion dem Strukturbild unmittelbar entnommen werden kann. Da alle Blöcke im Quelltext vorliegen, können sie vom Anwender auf einfache Weise modifiziert oder erweitert werden.

Installation aller erforderlichen Komponenten

Zum Betrieb der *LEGO Mindstorms Toolbox für WinFACT* sind folgende Software-Komponenten erforderlich:

- Das **Blockorientierte Simulationssystem BORIS** als Bestandteil des Programmsystems WinFACT 98 oder WinFACT 6 (wir empfehlen WinFACT 6, da diese Version die C-Code-spezifischen Block-Bitmaps unterstützt; siehe später!)
- Der zugehörige **AutoCode-Generator** für BORIS
- Das **RCX-Betriebssystem *legOS*** incl. diverser Zusatzkomponenten wie Compiler etc. Dieses Betriebssystem ist über das Internet frei verfügbar (siehe Kapitel 2)

Die mit den LEGO Mindstorms Baukästen gelieferte Software zur Programmierung der LEGO-Roboter ist für die Nutzung der *LEGO Mindstorms Toolbox für WinFACT* also selbst **nicht** erforderlich!

Zur Installation aller Komponenten gehen Sie wie folgt vor:

1. Installieren Sie - sofern noch nicht geschehen - das Programmsystem WinFACT 98 bzw. WinFACT 6. Machen Sie sich anschließend anhand der Dokumentation und Beispieldateien mit der Benutzung des blockorientierten Simulationssystems BORIS vertraut.
2. Installieren Sie - sofern nicht bereits geschehen - den WinFACT-AutoCode-Generator und machen Sie sich mit seiner Anwendung vertraut.
3. Beschaffen Sie sich das *legOS*-Betriebssystem und installieren Sie es wie im Kapitel *Installation des legOS-Betriebssystems* beschrieben.

2. Installation des legOS-Betriebssystems

Hinweis: Die nachfolgenden Anweisungen beziehen sich auf die Version *legOS* 0.2.4. Weiterführende Hinweise zur Installation - auch nachfolgender Versionen des Betriebssystems - finden Sie im Internet unter *http://legOS.sourceforge.net*.

Da *legOS* ursprünglich ein unter Unix entwickeltes Betriebssystem darstellt, muss für die Nutzung unter Windows zunächst eine Unix-Umgebung eingerichtet werden, aus der heraus *legOs* dann eingerichtet und benutzt werden kann. Dies geschieht wie nachfolgend beschrieben am besten in Form der frei verfügbaren *Cygwin* Umgebung.

Gehen Sie wie folgt vor[1]:

1. Downloaden Sie die folgenden Dateien aus dem Internet:
   - LegOS-0.2.4.tar.gz von *http://legOS.sourceforge.net/files/common/legOS-0.2.4.tar.gz*. Speichern Sie die Datei in Ihrem Root-Verzeichnis C:\ zur späteren Verwendung
   - Cygwin (Version b20) von *http://legOS.sourceforge.net/files/windows/cygwin/cygwin.exe*.
   - Den H8-Cross Compiler für Cygwin b20 von *http://legOS.sourceforge.net/files/windows/cygwin/win-h8-egcs-1.1.2.zip*.
2. Installieren Sie Cygwin durch einen Doppelklick auf cygwin.exe und übernehmen Sie nachfolgend alle Voreinstellungen (siehe Fußnote!).
3. Entpacken Sie die Datei win-h8-egcs-1.1.2.zip in das Verzeichnis c:\cygnus\cygwin-b20\h-i586-cygwin32. Sofern Sie eine Warnung bezüglich des Überschreibens der Darei cygwin1.dll erhalten, so ignorieren Sie diese einfach.
4. Legen Sie ein temporäres Verzeichnis C:\tmp an. Dieses wird für die Cygwin-Umgebung benötigt.
5. Starten Sie nun über das Windows-Startmenü die Cygwin-Umgebung über Start -> Cygnus Solutions -> Cygwin B20. Es öffnet sich ein Cygwin Fenster. Geben Sie dort *cd /* ein (**Achtung**: Bei der Cygwin-Konsole handelt es sich um eine Unix-Umgebung. Statt des von DOS/Windows bekannten Backslash muss dort immer der "normale" Schrägstrich eingegeben werden!!). Danach geben Sie die Zeile *tar xvzf legOS-0.2.4.tar.gz* ein. Daraufhin wird ein Unterverzeichnis c:\legOS angelegt, in das automatisch alle benötigten Dateien kopiert werden. Verlassen Sie das Fenster anschließend durch Eingabe von *exit*.
6. Es müssen nun noch einige Suchpfade für das Betriebssystem angelegt werden. Wählen Sie dazu im Windows-Startmenü die Option *Ausführen* und geben Sie dann *sysedit* ein, um den Systemeditor zu starten (unter Windows NT/2000 müssen Sie dazu als Systemadministrator angemeldet sein!). Im Fenster mit der Datei Autoexec.bat suchen Sie die Zeile für die Pfadangabe (beginnt mit *set path* =). Sofern diese Zeile existiert, hängen Sie folgenden Term an diese Zeile an (Semikolon am Anfang nicht vergessen!):

   ;c:\cygnus\cygwin-b20\h-i586-cygwin32\;c:\cygnus\cygwin-b20\h-i586-cygwin32\bin;c:\legos\util;c:\legos\util\firmdl

[1] Prinzipiell lassen sich alle Tools in *beliebige* Verzeichnisse installieren. Sie sollten aber wenn irgend möglich die hier vorgeschlagenen Verzeichnisse übernehmen, da ansonsten u. U. eine Vielzahl von Pfadangaben in unterschiedlichen Dateien geändert werden muss.

Sollte noch kein Suchpfad existieren, tragen Sie den neuen Suchpfad am Ende der Datei ein in der Form

set path = c:\cygnus\cygwin-b20\h-i586-cygwin32\;c:\cygnus\cygwin-b20\h-i586-cygwin32\bin;c:\legos\util;c:\legos\util\firmdl;%path%

7. Jetzt kann der Betriebssystem-Kernel erzeugt werden. Öffnen Sie dazu zunächst wieder eine Cygwin-Konsole (siehe Schritt 5) und geben Sie nacheinander folgende Befehle ein:

   cd /legOS

   make realclean

   make depend

   make

   Nach Abarbeitung dieser Befehle steht Ihnen nun in der Datei c:\legOS\boot\legOS.srec der Betriebssystem-Kernel zur Verfügung.

8. Nun muss der Kernel noch einmalig in den RCX geladen werden. Dazu stellen Sie den eingeschalteten RCX vor den Infrarot-Sender und geben innerhalb der Cygwin-Konsole aus Ihrem legOS-Verzeichnis die Befehlszeile

   util/firmdl3 boot/legOS.srec

   ein. Nach Übertragung des Kernels (dauert einige Sekunden) sollte auf dem LCD des RCX das Männchen und dahinter ein Bindestrich zu sehen sein. Um sicherzugehen, betätigen Sie die PRGM-Taste. Im Display sollte nun "NONE" zu sehen sein. Herzlichen Glückwunsch! Sie haben das *legOS*-Betriebssystem nunmehr erfolgreich installiert.

9. Legen Sie sich abschließend für Ihre eigenen Experimente am besten ein Unterverzeichnis innerhalb des *legOS*-Verzeichnisses an (z. B. c:\legOs\winfact). Kopieren Sie die Datei *MakeFile* aus dem Verzeichnis c:\legOS\demo in Ihr Unterverzeichnis. Kopieren Sie ebenfalls alle Dateien aus dem Unterverzeichnis *\AutoCode* Ihrer WinFACT-Installation in dieses Unterverzeichnis.

Tipp: Eine hervorragende Einführung in legOS und alle damit zusammenhängenden Themen bietet das Buch "Extreme Mindstorms - An Advanced Guide to LEGO Mindstorms" von Dave Baum, ISBN 1-893115-84-4, www.apress.com

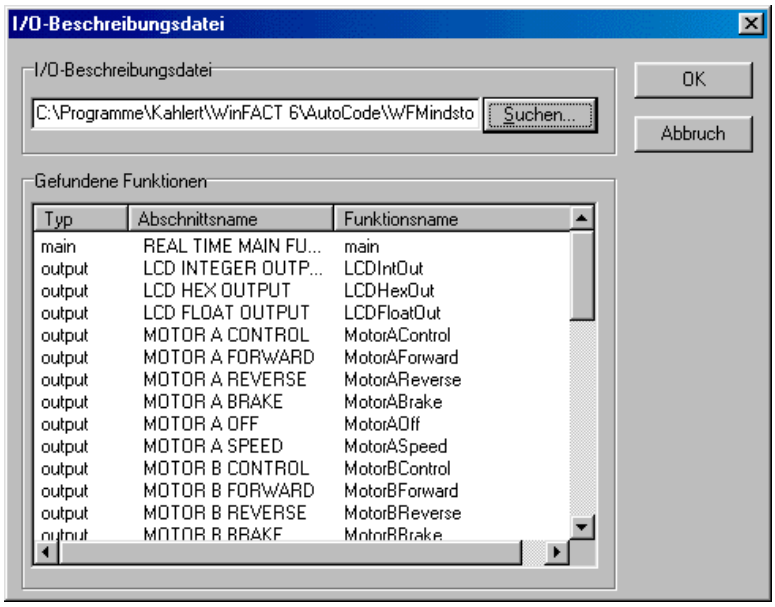
---

3. Programmierung des RCX aus BORIS

Nach der (zugegebenermaßen recht mühsamen) Installation und Konfigurierung des *legOS*-Betriebssystems wollen wir nun das Zusammenspiel mit BORIS anhand eines sehr einfachen Anwendungsbeispiels erläutern. Dazu soll der RCX aus BORIS heraus so programmiert werden, dass er die Motoren A und C nach dem Starten des Programms innerhalb von 10 Sekunden vom Stillstand auf maximale Geschwindigkeit im Vorwärtslauf bringt. Die aktuelle Motorgeschwindigkeit soll dabei zur Kontrolle auf dem LC-Display des RCX ausgegeben werden.

Schritt 1: Grundlegende Einstellungen des AutoCode-Generators

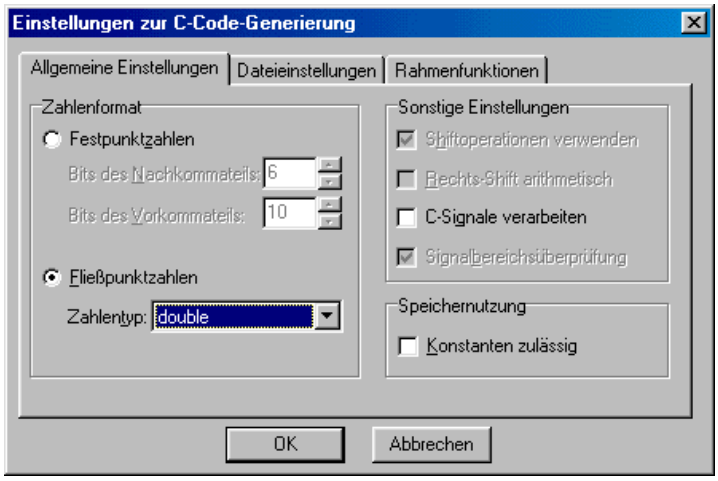
Bevor wir die BORIS-Struktur erstellen, wollen wir einige grundsätzliche Einstellungen für den BORIS-AutoCode-Generator vornehmen, die wir später dann für praktisch alle LEGO-Mindstorms-Projekte übernehmen können. Dazu wählen wir nach dem Starten von BORIS die Menüoption CODE-GENERIERUNG | I/O-BESCHREIBUNGSDATEI WÄHLEN... Über die *Suchen*-Schaltfläche können wir nun die erforderliche I/O-Beschreibungsdatei für die RCX-Programmierung suchen; sie befindet sich im *\AutoCode*-Unterverzeichnis Ihrer WinFACT-Installation und heißt *WFMindstorms.c*[2]. Nach Auswahl der Datei erscheinen in der Liste im unteren Dialogbereich alle Schnittstellenfunktionen der Toolbox (siehe nachfolgende Bildschirmgrafik).



Wahl der I/O-Beschreibungsdatei

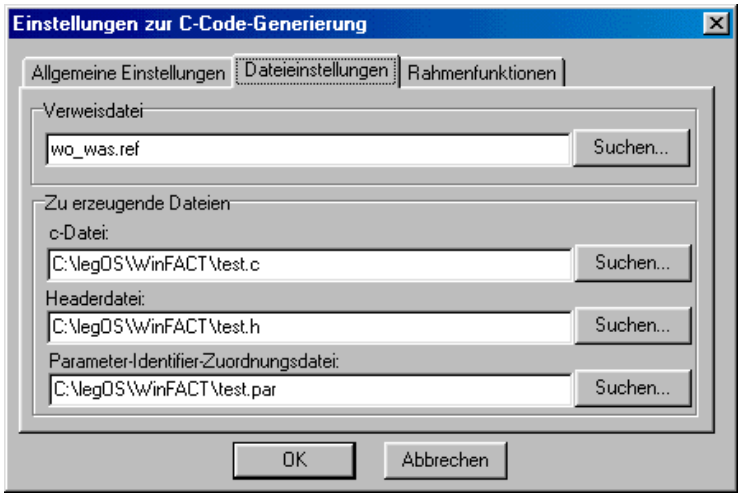
[2] Sofern Sie unter noch WinFACT 98 arbeiten, sollten Sie stattdessen die Datei *WFMindstorms98.c* wählen!

Danach wechseln wir über CODE-GENERIERUNG | CODE-GENERIERUNGS-EINSTELLUNGEN... in den Dialog zur Einstellung codespezifischer Parameter. Nehmen Sie zunächst die allgemeinen Einstellungen vor wie in der nachfolgenden Bildschirmgrafik dargestellt.



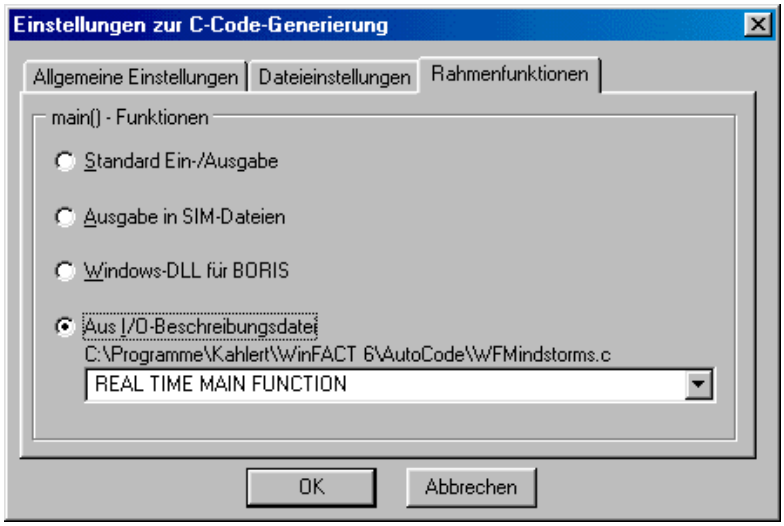
Allgemeine Einstellungen zur Code-Generierung

Anschließend müssen wir dem AutoCode-Generator mitteilen, wo er die generierten Dateien ablegen soll. Wir möchten unser Projekt mit *Test* bezeichnen und alle generierten Dateien im *\WinFACT*-Unterverzeichnis des *legOS*-Verzeichnisses ablegen und tragen daher die in nachfolgender Bildschirmgrafik angegebenen Dateinamen ein.



Dateieinstellungen zur Code-Generierung

Schließlich müssen wir für den generierten C-Code eine Rahmenfunktion (*main*-Routine) auswählen; diese befindet sich in der zuvor gewählten I/O-Beschreibungsdatei und heißt *Real Time Main Function* (siehe nachfolgende Bildschirmgrafik).

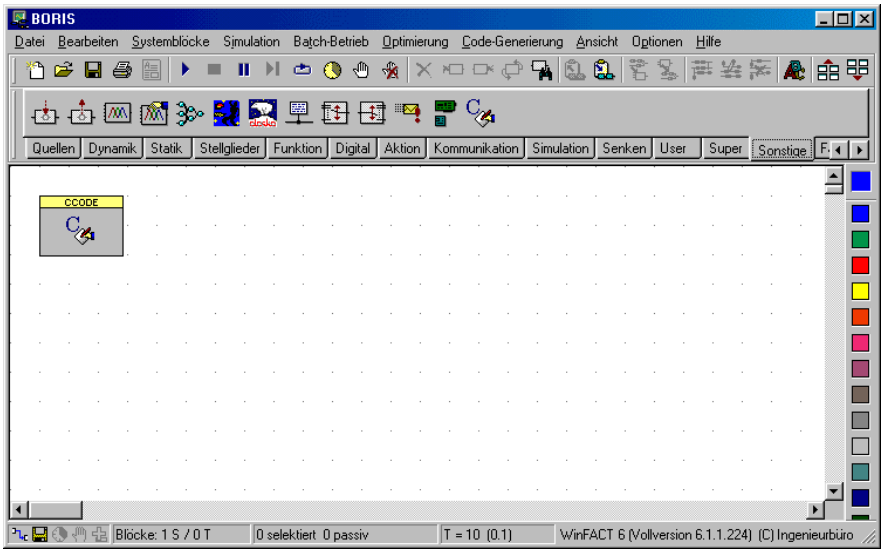


Wahl der Rahmenfunktion für den C-Code

Um alle bisher getätigten Einstellungen für spätere Projekte direkt wiederverwenden zu können, wollen wir die Konfiguration abspeichern. Dazu dient die Menüoption CODE-GENERIERUNG | KONFIGURATION SPEICHERN... Speichern Sie die Konfiguration z. B. unter dem Namen *Test.ccg* im *\WinFACT*-Unterverzeichnis des *legOS*-Verzeichnisses. Sie können sie dann später über CODE-GENERIERUNG | KONFIGURATION LADEN... wieder laden.

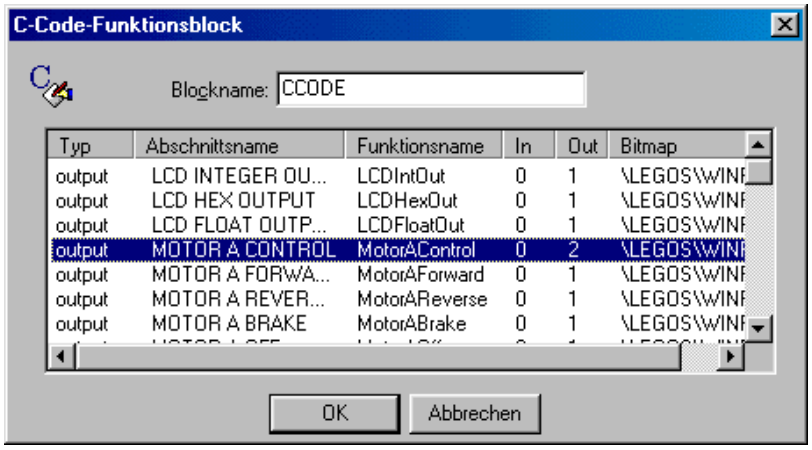
Schritt 2: Erstellen der Systemstruktur

Die "Kopplung" des RCX an die BORIS-Struktur findet über den C-Code-Schnittstellenblock statt; dieser Blocktyp bildet daher die Grundlage der LEGO Mindstorms Toolbox. Sie finden ihn in der Systemblock-Palette *Sonstige* ganz rechts (siehe nachfolgende Bildschirmgrafik). Nach dem Einfügen des Blocks ist dieser zunächst noch ohne Funktionalität, d. h. mit keiner Funktion der I/O-Beschreibungsdatei verbunden.



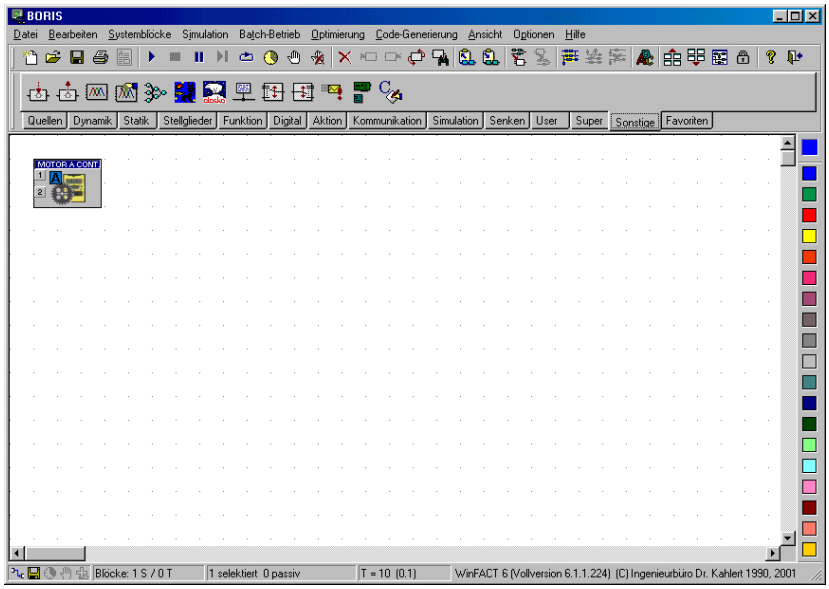
BORIS nach Einfügen eines C-Code-Schnittstellenblocks (vor Zuweisung einer I/O-Beschreibungsfunktion)

Um dem Block nun eine bestimmte Funktionalität zu geben (nämlich die Ansteuerung von Motor A des RCX), wechseln wir durch einen Doppelklick auf den Block in seinen Parameterdialog. Dort finden wir alle Funktionen der zuvor ausgewählten I/O-Beschreibungsdatei *WFMindstorms.c*. Wählen Sie die Funktion *MotorAControl* aus.



Zuweisen der Funktion MotorAControl *an den C-Code-Block*

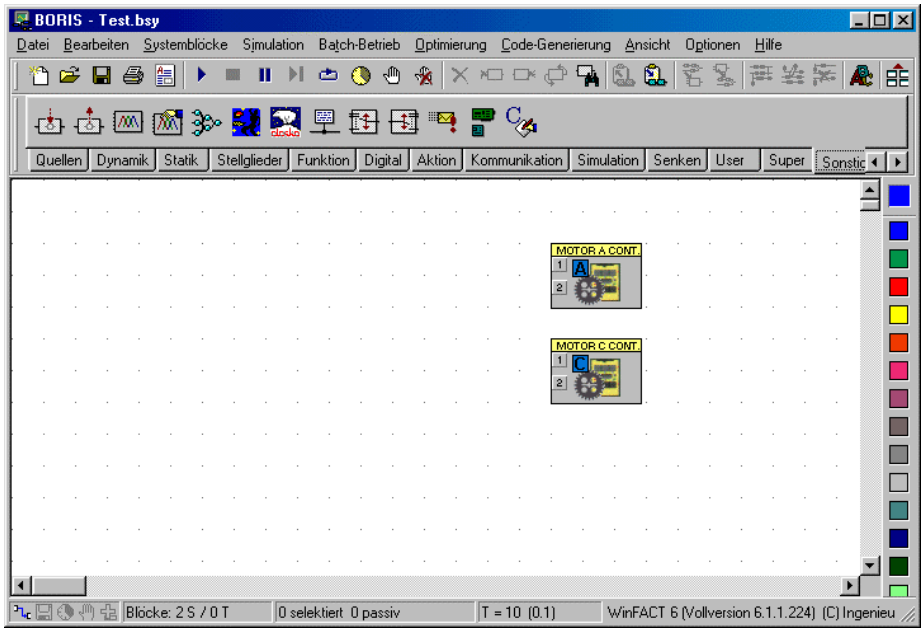
Nach dem Verlassen des Dialogs erhält der Block nunmehr ein charakteristisches Icon, welches seine Funktionalität unmittelbar deutlich werden lässt[3]. Außerdem besitzt er nun zwei Eingänge; der erste dient - wie sich dem Kapitel *Die I/O-Beschreibungsdatei WFMindstorms.c* entnehmen lässt - zur Vorgabe der Motor-Betriebsart und der zweite zur Steuerung der Motor-Drehzahl.



BORIS nach Verlassen des Parameterdialogs

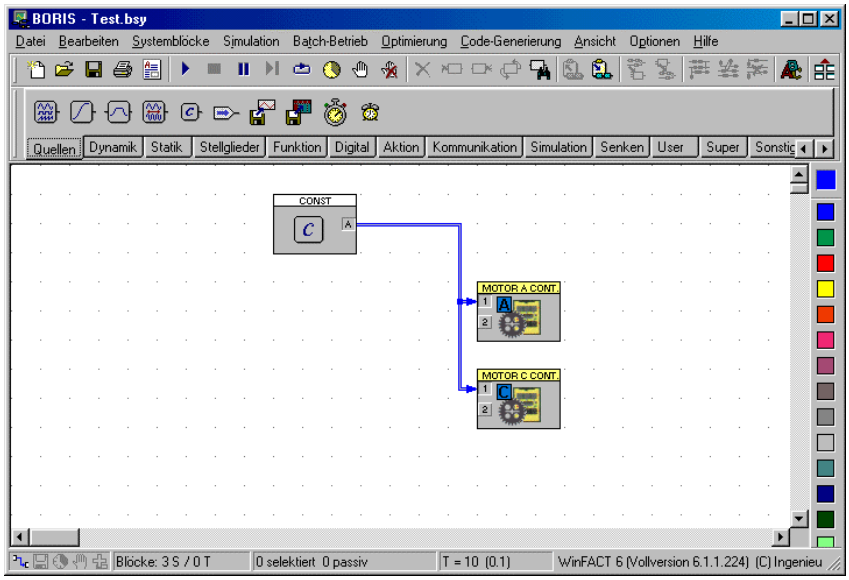
Da wir die Motoren A und C parallel ansteuern müssen, fügen wir einen weiteren C-Code-Block ein, dem wir im Parameterdialog entsprechend die Funktion *MotorCControl* zuweisen (siehe nachfolgende Bildschirmgrafik).

[3] Unter WinFACT 98 ist diese Option noch nicht implementiert.



BORIS nach EInfügen eines weiteren Blocks zur Ansteuerung von Motor C

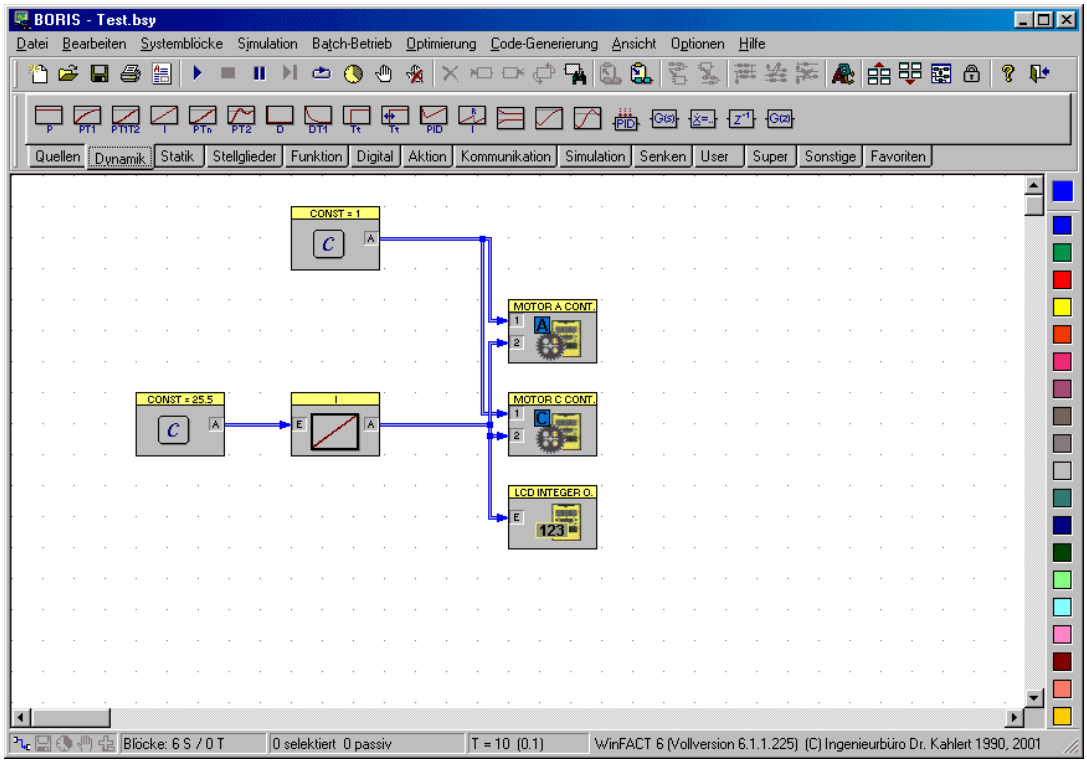
Da unser Roboter nur vorwärts fahren soll, können wir den ersten Eingang beider Blöcke jeweils auf einen konstanten Wert setzen; der Dokumentation zur Funktion *MotorAControl* bzw. *MotorCControl* entnehmen wir, dass für Vorwärtsfahrt am ersten Eingang ein Wert ≥ 1 anliegen muss. Wir benutzen zur Anstuerung daher einen CONST-Block aus der Systemblock-Palette *Quellen*, dessen Parameter wir auf 1 setzen.



Ansteuerung der Blockeingänge zur Steuerung der Motor-Betriebsart

Die Drehzahl der beiden Motoren soll innerhalb von zehn Sekunden linear von 0 auf ihren Maximalwert (255) ansteigen. Dazu können wir einen Integrierer mit einer Integrationszeit von 1 s benutzen, dessen Ausgang wir auf 255 begrenzen und dessen Eingang wir mit einem konstanten Wert von 255/10 = 25.5 speisen. Da die aktuelle Drehzahl parallel auf dem LCD des RCX ausgegeben werden soll, benötigen wir einen weiteren C-Code-Block, dem wir aus der I/O-Beschreibungsdatei die Funktion *LCDIntOut* zuweisen und dessen Eingang wir ebenfalls mit dem Ausgang des Integrierers verbinden. Unsere komplette Struktur zeigt dementsprechend nachfolgende Bildschirmgrafik. Damit auch alle Blöcke in C-Code überführt werden, selektieren wie die komplette Struktur, klicken dann mit der rechten Maustaste in den freien Bereich des BORIS-

Hauptfensters und wählen im daraufhin erscheinenden Kontextmenü die Option IN C-CODE-GENERIERUNGSLISTE EINTRAGEN. Alle Blöcke erhalten daraufhin im nicht selektierten Zustand einen gelben und im selektierten Zustand einen blauen Titelbalken.



Komplette Systemstrukur nach Eintragen aller Blöcke in die C-Code-Generierungsliste

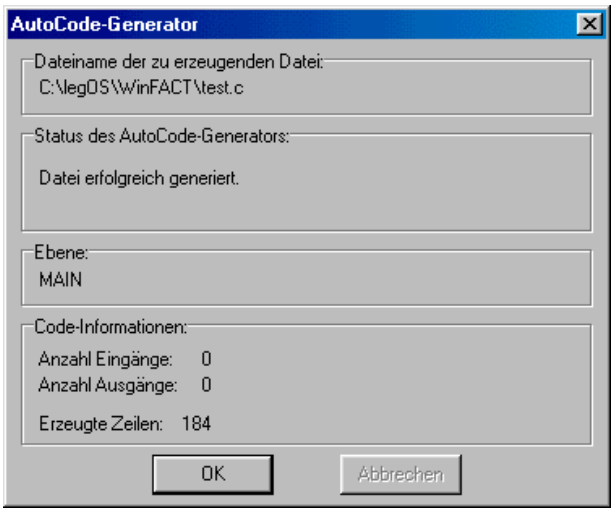
Schritt 3: Wahl der Simulationsparameter

Obwohl die auf Basis der LEGO Mindstorms Toolbox erstellten BORIS-Strukturen innerhalb von BORIS in der Regel nicht wirklich "simuliert" werden sollen, hat zumindest die in BORIS eingestellte Abtastschrittweite eine bestimmte Bedeutung; sie entspricht nämlich später auf dem RCX der Zykluszeit, d. h. der Abtastschrittweite. Für die meisten Fälle ist hier ein Wert von 0.1 (entsprechend 100 ms) sinnvoll. Die Simulationsdauer ist ohne Belang, da das generierte Programm später auf dem RCX ohnehin zyklisch abläuft und nur über den Run-Schalter des RCX abgebrochen werden kann[4]. Wir stellen für unser Beispiel daher eine Simulationsdauer von 10 und eine Schrittweite von 0.1 ein (entspricht den Voreinstellungen in BORIS).

[4] Dies gilt zumindest dann, wenn als *main*-Programm (Rahmenprogramm) wie in diesem Beispiel die Funktion *Real Time Main Function* aus der I/O-Beschreibungsdatei gewählt wird.

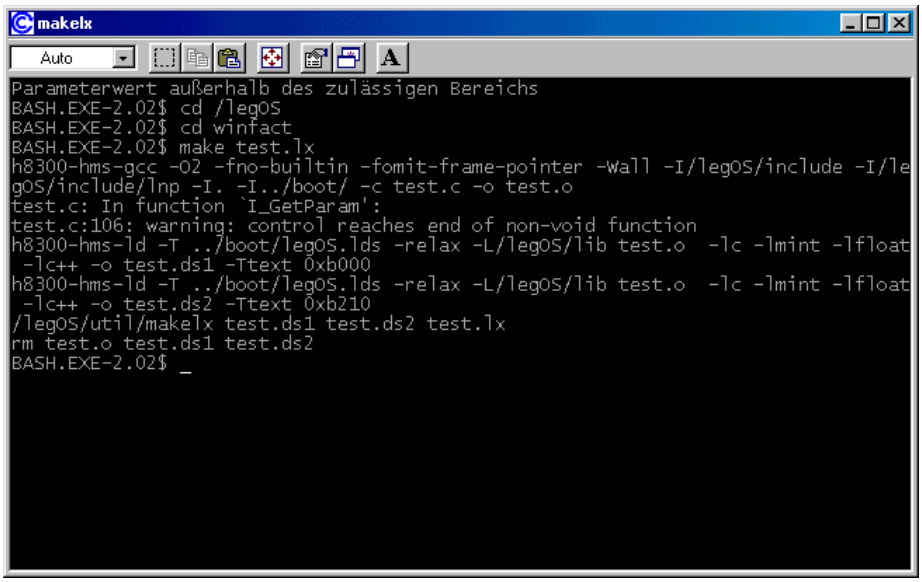
Schritt 4: Generierung des C-Codes, Compilieren und Übertragen auf den RCX

Nach der Erstellung der Systemstruktur und der Wahl der Simulationsparameter kann der zugehörige C-Code generiert werden. Dazu dient der BORIS-Menüpunkt CODE-GENERIERUNG | CODE GENERIEREN... Nach erfolgreicher Code-Generierung meldet BORIS wie in untenstehender Bildschirmgrafik Vollzug.



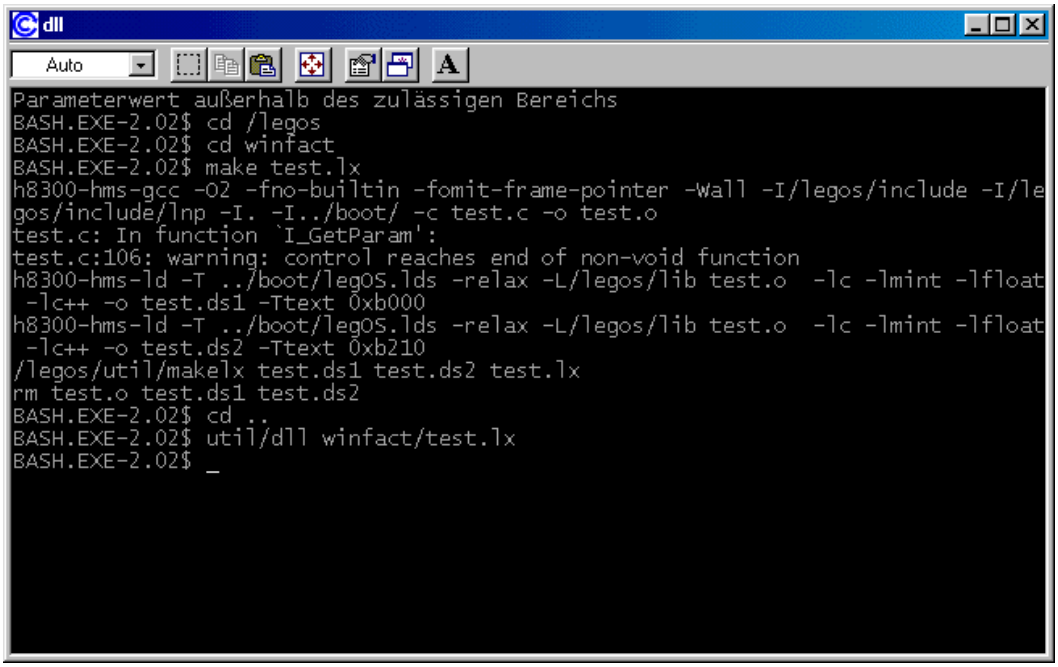
Meldung des Code-Generators nach gelungener Code-Generierung

In unserem *legOS*-Verzeichnis steht uns nunmehr eine C-Datei mit Namen *Test.c* zur Verfügung, die wir nur noch compilieren und dann auf den RCX übertragen müssen. Dazu öffnen wir wieder eine Cygwin-Konsole (zur Erinnerung: aus dem Windows-Startmenü über PROGRAMME | CYGNUS SOLUTIONS | CYGWIN B20). Dort wechseln wir zunächst *über cd /legOS* in das *legOs*-Verzeichnis und dann über *cd winfact* in unser Unterverzeichnis. Über *make test.lx* starten wir schließlich den Compiliervorgang, der aus unserer Datei *Test.c* die Datei *Test.lx* generiert (siehe nachfolgende Bildschirmgrafik).



Cygwin-Konsole nach Compilieren von Test.c. Die Warnungen des Compilers können Sie getrost ignorieren

Zum Herunterladen von *Test.lx* auf den RCX wechseln wir nun über *cd ..* wieder in das *legOS*-Verzeichnis und starten den Upload dann von dort über die Befehlszeile *util/dll winfact/test.lx*. Nach einigen Sekunden meldet sich das Cygwin-Prompt wieder kommentarlos zurück - ein Zeichen dafür, dass alles "glattgegangen" ist (siehe nachfolgende Bildschirmgrafik).



Cygwin-Konsole nach erfolgreicher Übertragung des Programms Test.lx auf den RCX

Schritt 5: Starten des Programms

Nach erfolgreicher Übertragung des Programms auf den RCX können wir unser Programm nun testen. Dazu genügt die simple Betätigung der *Run*-Taste auf dem RCX. Wenn alles korrekt verlaufen ist, sollten beide Motoren nun innerhalb von zehn Sekunden auf Maximaldrehzahl beschleunigen und dann auf diesem Wert verbleiben, wobei der aktuelle Wert der Drehzahl parallel auf dem LCD des RCX angezeigt wird.

Zusammenfassung der Vorgehensweise

Die zuvor beschriebene Vorgehensweise mag zunächst sehr komplex erscheinen; von den aufgeführten Arbeitsschritten ist jedoch später jeweils nur noch ein kleiner Teil erforderlich. Im wesentlichen sind - nach Starten von BORIS - für eine Neuentwicklung folgenden Schritte erforderlich:

1. Laden Sie am besten zunächst die AutoCode-Konfiguration (in unserem Beispiel TEST.CCG) über CODE-GENERIERUNG | KONFIGURATION LADEN und passen Sie anschließend ggfs. unter CODE-GENERIERUNG | CODE-GENERIERUNGS-EINSTELLUNGEN die Namen der Ausgabedateien an (in unserem Beispiel TEST.C, TEST.H und TEST.PAR, falls Sie diese nicht überschreiben wollen. Alle anderen Einstellungen können für die LEGO Mindstorms Projekte in der Regel belassen werden.

2. Nun können Sie sich an das Erstellen der BORIS-Struktur begeben. Vergessen Sie nicht, alle Blöcke wie beschrieben in die C-Code-Generierungsliste einzutragen und generieren Sie den C-Code über CODE-GENERIERUNG | CODE GENERIEREN...
3. Öffnen Sie eine Cygwin-Konsole. Wechseln Sie über die Befehle

   cd /legos

   cd winfact

   in das WinFACT-Unterverzeichnis des *legOS*-Betriebssystems und starten Sie die Compilierung des von BORIS generierten C-Codes über

   make dateiname.lx

   wobei *dateiname.c* der Name der von BORIS erzeugten C-Datei (in unserem Beispiel TEST.C) ist. Wechseln Sie dann über

   cd ..

   zurück ins *legOS*-Hauptverzeichnis und übertragen Sie das Programm über

   util/dll winfact/dateiname.lx

   in den RCX.

Selbstverständlich lassen sich vom "geübten" Anwender die in Schritt 3 einzugebenden Befehle auch über eine Batch-Datei automatisieren; sie lassen sich aber innerhalb der Cygwin-Konsole auch über die Cursor-Tasten (vergleichbar dem DOS-Fenster) jederzeit zurückholen.

4. Die I/O-Beschreibungsdatei `WFMindstorms.c`

main - Routine

Funktion:

Bildet die *main*-Routine zur Echtzeitverarbeitung der mit BORIS generierten Strukturen auf dem RCX. Als Abtastzeit wird die unter BORIS eingestellte Simulationsschrittweite benutzt

Code innerhalb der I/O-Beschreibungsdatei:

```
/* Real time main function */
#include "C:\legOS\include\unistd.h"
#include "C:\legOS\include\time.h"
int main(int argc, char *argv[])
{
############################################################
############################################################
############################################################
############################################################
############################################################
############################################################
############################################################
############################################################
}
/*end*/
```

LCD-Ausgabe

LCD Integer Output `LCDIntOut`

Funktion:

Gibt den am Blockeingang liegenden Signalwert als Ganzzahl auf dem LCD des RCX aus. Der Signalwert sollte im Integer-Bereich, d. h. zwischen -32768 und +32767 liegen.

Code innerhalb der I/O-Beschreibungsdatei:

```
/* LCD Integer Output */
#Outputs: 1
#include "C:\legOS\include\rom\lcd.h"
void LCDIntOut(SVartyp e1)
{
############################################################
}
/*end*/
```

LCD Hex Output `LCDHexOut`

Funktion:

Gibt den am Blockeingang liegenden Signalwert als Hexadezimalzahl auf dem LCD des RCX aus. Der Signalwert sollte im Integer-Bereich, d. h. zwischen 0 und $FFFF liegen.

Code innerhalb der I/O-Beschreibungsdatei:

```
/* LCD Hex Output */
#Outputs: 1
#include "C:\legOS\include\conio.h"
void LCDHexOut(SVartyp e1)
{
############################################################
}
/*end*/
```

LCD Float Output `LCDFloatOut`

Funktion:

Gibt den am Blockeingang liegenden Signalwert als Fließkommazahl mit einer Nachkommastelle auf dem LCD des RCX aus.

Code innerhalb der I/O-Beschreibungsdatei:

```
/* LCD Float Output */
#Outputs: 1
#include "C:\legOS\include\conio.h"
#include "C:\legOS\include\rom\lcd.h"
void LCDFloatOut(SVartyp e1)
{
############################################################
############################################################
############################################################
}
/*end*/
```

Soundausgabe

Beep `Beep`

Funktion:

Gibt auf dem internen Lautsprecher des RCX einen Signalton aus, wenn am Blockeingang HIGH-Pegel (d. h. ein Signalwert > 2.5) anliegt.

Code innerhalb der I/O-Beschreibungsdatei:

```
/* Beep */
#Outputs: 1
#include "C:\legOS\include\dsound.h"
void Beep(SVartyp e1)
{
############################################################
############################################################
############################################################
############################################################
############################################################
}
/*end*/
```

Motor-Steuerung

Motor A Control `MotorAControl`

Funktion:

Steuert die Betriebsart des Motors in Abhängigkeit vom ersten Blockeingang e_1 wie folgt an:

$e_1 = 0$:	Betriebsart *brake*	(Motor bremst)
$0 < \lvert e_1 \rvert < 1$:	Betriebsart *off*	(Motor aus)
$e_1 \geq 1$:	Betriebsart *forward*	(Drehrichtung vorwärts)
$e_1 \leq -1$:	Betriebsart *reverse*	(Drehrichtung rückwärts)

Über den zweiten Blockeingang wird die Drehgeschwindigkeit des Motors im Bereich 0..255 vorgegeben. Sollen Betriebsart und Drehgeschwindigkeit eines Motors mit getrennten Blöcken gesteuert werden, können dazu die nachfolgend aufgeführten Blöcke benutzt werden.

Code innerhalb der I/O-Beschreibungsdatei:

```
/* Motor A Control*/
#Outputs: 2
#include "C:\legOS\include\dmotor.h"
void MotorAControl(SVartyp e1, SVartyp e2)
```

```
{
############################################################
############################################################
############################################################
############################################################
############################################################
}
/*end*/
```

Motor A Forward `MotorAForward`

Funktion:

Liegt am Blockeingang logischer HIGH-Pegel (Signalwert > 2.5) an, wird der Motor in die Betriebsart *forward* versetzt, andernfalls behält er seine aktuelle Betriebsart bei.

Code innerhalb der I/O-Beschreibungsdatei:

```
/* Motor A Forward */
#Outputs: 1
#include "C:\legOS\include\dmotor.h"
void MotorAForward(SVartyp e1)
{
############################################################
}
/*end*/
```

Motor A Reverse `MotorAReverse`

Funktion:

Liegt am Blockeingang logischer HIGH-Pegel (Signalwert > 2.5) an, wird der Motor in die Betriebsart *reverse* versetzt, andernfalls behält er seine aktuelle Betriebsart bei.

Code innerhalb der I/O-Beschreibungsdatei:

```
/* Motor A Reverse */
#Outputs: 1
#include "C:\legOS\include\dmotor.h"
void MotorAReverse(SVartyp e1)
{
############################################################
}
/*end*/
```

Motor A Brake `MotorABrake`

Funktion:

Liegt am Blockeingang logischer HIGH-Pegel (Signalwert > 2.5) an, wird der Motor in die Betriebsart *brake* versetzt, andernfalls behält er seine aktuelle Betriebsart bei.

Code innerhalb der I/O-Beschreibungsdatei:

```
/* Motor A Brake */
#Outputs: 1
#include "C:\legOS\include\dmotor.h"
void MotorABrake(SVartyp e1)
{
```

```
############################################################
}
/*end*/
```

Motor A Off `MotorAOff`

Funktion:

Liegt am Blockeingang logischer HIGH-Pegel (Signalwert > 2.5) an, wird der Motor in die Betriebsart *off* versetzt, andernfalls behält er seine aktuelle Betriebsart bei.

Code innerhalb der I/O-Beschreibungsdatei:

```
/* Motor A Off */
#Outputs: 1
#include "C:\legOS\include\dmotor.h"
void MotorAOff(SVartyp e1)
{
############################################################
}
/*end*/
```

Motor A Speed `MotorASpeed`

Funktion:

Setzt die Drehgeschwindigkeit des Motors auf den am Blockeingang anliegenden Wert im Bereich 0..255. Die aktuelle Betriebsart des Motors (Laufrichtung) wird beibehalten.

Code innerhalb der I/O-Beschreibungsdatei:

```
/* Motor A Speed */
#Outputs: 1
#include "C:\legOS\include\dmotor.h"
void MotorASpeed(SVartyp e1)
{
############################################################
}
/*end*/
```

Motor B Control `MotorAControl`

Funktion:

Steuert die Betriebsart des Motors in Abhängigkeit vom ersten Blockeingang e_1 wie folgt an:

$e_1 = 0$:	Betriebsart *brake*	(Motor bremst)
$0 < \lvert e_1 \rvert < 1$:	Betriebsart *off*	(Motor aus)
$e_1 \geq 1$:	Betriebsart *forward*	(Drehrichtung vorwärts)
$e_1 \leq -1$:	Betriebsart *reverse*	(Drehrichtung rückwärts)

Über den zweiten Blockeingang wird die Drehgeschwindigkeit des Motors im Bereich 0..255 vorgegeben. Sollen Betriebsart und Drehgeschwindigkeit eines Motors mit getrennten Blöcken gesteuert werden, können dazu die nachfolgend aufgeführten Blöcke benutzt werden.

Code innerhalb der I/O-Beschreibungsdatei:

```
/* Motor B Control*/
#Outputs: 2
#include "C:\legOS\include\dmotor.h"
void MotorBControl(SVartyp e1, SVartyp e2)
{
############################################################
############################################################
############################################################
############################################################
############################################################
}
/*end*/
```

Motor B Forward `MotorBForward`

Funktion:

Liegt am Blockeingang logischer HIGH-Pegel (Signalwert > 2.5) an, wird der Motor in die Betriebsart *forward* versetzt, andernfalls behält er seine aktuelle Betriebsart bei.

Code innerhalb der I/O-Beschreibungsdatei:

```
/* Motor B Forward */
#Outputs: 1
#include "C:\legOS\include\dmotor.h"
void MotorBForward(SVartyp e1)
{
############################################################
}
/*end*/
```

Motor B Reverse `MotorBReverse`

Funktion:

Liegt am Blockeingang logischer HIGH-Pegel (Signalwert > 2.5) an, wird der Motor in die Betriebsart *reverse* versetzt, andernfalls behält er seine aktuelle Betriebsart bei.

Code innerhalb der I/O-Beschreibungsdatei:

```
/* Motor B Reverse */
#Outputs: 1
#include "C:\legOS\include\dmotor.h"
void MotorBReverse(SVartyp e1)
{
############################################################
}
/*end*/
```

Motor B Brake `MotorBBrake`

Funktion:

Liegt am Blockeingang logischer HIGH-Pegel (Signalwert > 2.5) an, wird der Motor in die Betriebsart *brake* versetzt, andernfalls behält er seine aktuelle Betriebsart bei.

Code innerhalb der I/O-Beschreibungsdatei:

```
/* Motor B Brake */
#Outputs: 1
#include "C:\legOS\include\dmotor.h"
void MotorBBrake(SVartyp e1)
{
############################################################
}
/*end*/
```

Motor B Off `MotorBOff`

Funktion:

Liegt am Blockeingang logischer HIGH-Pegel (Signalwert > 2.5) an, wird der Motor in die Betriebsart *off* versetzt, andernfalls behält er seine aktuelle Betriebsart bei.

Code innerhalb der I/O-Beschreibungsdatei:

```
/* Motor B Off */
#Outputs: 1
#include "C:\legOS\include\dmotor.h"
void MotorBOff(SVartyp e1)
{
############################################################
}
/*end*/
```

Motor B Speed `MotorBSpeed`

Funktion:

Setzt die Drehgeschwindigkeit des Motors auf den am Blockeingang anliegenden Wert im Bereich 0..255. Die aktuelle Betriebsart des Motors (Laufrichtung) wird beibehalten.

Code innerhalb der I/O-Beschreibungsdatei:

```
/* Motor B Speed */
#Outputs: 1
#include "C:\legOS\include\dmotor.h"
void MotorBSpeed(SVartyp e1)
{
############################################################
}
/*end*/
```

Motor C Control `MotorCControl`

Funktion:

Steuert die Betriebsart des Motors in Abhängigkeit vom ersten Blockeingang e_1 wie folgt an:

$e_1 = 0$:	Betriebsart *brake*	(Motor bremst)
$0 < \lvert e_1 \rvert < 1$:	Betriebsart *off*	(Motor aus)
$e_1 \geq 1$:	Betriebsart *forward*	(Drehrichtung vorwärts)

$e_1 \leq -1$: Betriebsart *reverse* (Drehrichtung rückwärts)

Über den zweiten Blockeingang wird die Drehgeschwindigkeit des Motors im Bereich 0..255 vorgegeben. Sollen Betriebsart und Drehgeschwindigkeit eines Motors mit getrennten Blöcken gesteuert werden, können dazu die nachfolgend aufgeführten Blöcke benutzt werden.

Code innerhalb der I/O-Beschreibungsdatei:

```
/* Motor C Control*/
#Outputs: 2
#include "C:\legOS\include\dmotor.h"
void MotorCControl(SVartyp e1, SVartyp e2)
{
############################################################
############################################################
############################################################
############################################################
############################################################
}
/*end*/
```

Motor C Forward `MotorCForward`

Funktion:

Liegt am Blockeingang logischer HIGH-Pegel (Signalwert > 2.5) an, wird der Motor in die Betriebsart *forward* versetzt, andernfalls behält er seine aktuelle Betriebsart bei.

Code innerhalb der I/O-Beschreibungsdatei:

```
/* Motor C Forward */
#Outputs: 1
#include "C:\legOS\include\dmotor.h"
void MotorCForward(SVartyp e1)
{
############################################################
}
/*end*/
```

Motor C Reverse `MotorCReverse`

Funktion:

Liegt am Blockeingang logischer HIGH-Pegel (Signalwert > 2.5) an, wird der Motor in die Betriebsart *reverse* versetzt, andernfalls behält er seine aktuelle Betriebsart bei.

Code innerhalb der I/O-Beschreibungsdatei:

```
/* Motor C Reverse */
#Outputs: 1
#include "C:\legOS\include\dmotor.h"
void MotorCReverse(SVartyp e1)
{
############################################################
}
/*end*/
```

Motor C Brake `MotorCBrake`

Funktion:

Liegt am Blockeingang logischer HIGH-Pegel (Signalwert > 2.5) an, wird der Motor in die Betriebsart *brake* versetzt, andernfalls behält er seine aktuelle Betriebsart bei.

Code innerhalb der I/O-Beschreibungsdatei:

```
/* Motor C Brake */
#Outputs: 1
#include "C:\legOS\include\dmotor.h"
void MotorCBrake(SVartyp e1)
{
############################################################
}
/*end*/
```

Motor C Off `MotorCOff`

Funktion:

Liegt am Blockeingang logischer HIGH-Pegel (Signalwert > 2.5) an, wird der Motor in die Betriebsart *off* versetzt, andernfalls behält er seine aktuelle Betriebsart bei.

Code innerhalb der I/O-Beschreibungsdatei:

```
/* Motor C Off */
#Outputs: 1
#include "C:\legOS\include\dmotor.h"
void MotorCOff(SVartyp e1)
{
############################################################
}
/*end*/
```

Motor C Speed `MotorCSpeed`

Funktion:

Setzt die Drehgeschwindigkeit des Motors auf den am Blockeingang anliegenden Wert im Bereich 0..255. Die aktuelle Betriebsart des Motors (Laufrichtung) wird beibehalten.

Code innerhalb der I/O-Beschreibungsdatei:

```
/* Motor C Speed */
#Outputs: 1
#include "C:\legOS\include\dmotor.h"
void MotorCSpeed(SVartyp e1)
{
############################################################
}
/*end*/
```

Licht-Ausgänge

Light A Out `LightAOut`

Funktion:

Setzt die Ausgangsleistung der am Ausgang A angeschlossenen Lampe auf den am Blockeingang anliegenden Wert im Bereich 0..255

Code innerhalb der I/O-Beschreibungsdatei:

```
/* Light A Out */
#Outputs: 1
#include "C:\legOS\include\dmotor.h"
void LightAOut(SVartyp e1)
{
############################################################
############################################################
}
/*end*/
```

Light B Out `LightBOut`

Funktion:

Setzt die Ausgangsleistung der am Ausgang B angeschlossenen Lampe auf den am Blockeingang anliegenden Wert im Bereich 0..255

Code innerhalb der I/O-Beschreibungsdatei:

```
/* Light B Out */
#Outputs: 1
#include "C:\legOS\include\dmotor.h"
void LightBOut(SVartyp e1)
{
############################################################
############################################################
}
/*end*/
```

Light C Out `LightCOut`

Funktion:

Setzt die Ausgangsleistung der am Ausgang C angeschlossenen Lampe auf den am Blockeingang anliegenden Wert im Bereich 0..255

Code innerhalb der I/O-Beschreibungsdatei:

```
/* Light C Out */
#Outputs: 1
#include "C:\legOS\include\dmotor.h"
void LightCOut(SVartyp e1)
{
############################################################
```

```
############################################################
}
/*end*/
```

Sensoren

Sensor 1 `Sensor1`

Funktion:

Liefert am Blockausgang den Rohwert von Sensor 1 als Hexadezimalwert, d. h. im Bereich 0..65535.

Code innerhalb der I/O-Beschreibungsdatei:

```
/* Sensor 1 */
#Inputs: 1
#include "C:\legOS\include\dsensor.h"
void Sensor1(SVartyp *a1)
{
############################################################
}
/*end*/
```

Sensor 2 `Sensor2`

Funktion:

Liefert am Blockausgang den Rohwert von Sensor 2 als Hexadezimalwert, d. h. im Bereich 0..65535.

Code innerhalb der I/O-Beschreibungsdatei:

```
/* Sensor 2 */
#Inputs: 1
#include "C:\legOS\include\dsensor.h"
void Sensor2(SVartyp *a1)
{
############################################################
}
/*end*/
```

Sensor 3 `Sensor3`

Funktion:

Liefert am Blockausgang den Rohwert von Sensor 3 als Hexadezimalwert, d. h. im Bereich 0..65535.

Code innerhalb der I/O-Beschreibungsdatei:

```
/* Sensor 3 */
#Inputs: 1
#include "C:\legOS\include\dsensor.h"
void Sensor3(SVartyp *a1)
{
```

```
############################################################
}
/*end*/
```

Touch Sensor 1 `TouchSensor1`

Funktion:

Liefert am Blockausgang den Zustand von Sensor 1 als logischen Wert mit einem Signalpegel von 5 (Sensor gedrückt) bzw. 0 (Sensor nicht gedrückt).

Code innerhalb der I/O-Beschreibungsdatei:

```
/* Touch Sensor 1 */
#Inputs: 1
#include "C:\legOS\include\dsensor.h"
void TouchSensor1(SVartyp *a1)
{
############################################################
############################################################
############################################################
############################################################
}
/*end*/
```

Touch Sensor 2 `TouchSensor2`

Funktion:

Liefert am Blockausgang den Zustand von Sensor 2 als logischen Wert mit einem Signalpegel von 5 (Sensor gedrückt) bzw. 0 (Sensor nicht gedrückt).

Code innerhalb der I/O-Beschreibungsdatei:

```
/* Touch Sensor 2 */
#Inputs: 1
#include "C:\legOS\include\dsensor.h"
void TouchSensor2(SVartyp *a1)
{
############################################################
############################################################
############################################################
############################################################
}
/*end*/
```

Touch Sensor 3 `TouchSensor3`

Funktion:

Liefert am Blockausgang den Zustand von Sensor 3 als logischen Wert mit einem Signalpegel von 5 (Sensor gedrückt) bzw. 0 (Sensor nicht gedrückt).

Code innerhalb der I/O-Beschreibungsdatei:

```
/* Touch Sensor 3 */
#Inputs: 1
#include "C:\legOS\include\dsensor.h"
void TouchSensor3(SVartyp *a1)
```

```
{
############################################################
############################################################
############################################################
############################################################
}
/*end*/
```

Light Sensor 1 `LightSensor1`

Funktion:

Liefert am Blockausgang die an Sensor 1 gemessene Lichtintensität als (skalierten) Integer-Wert etwa im Bereich zwischen 50 (dunkel) und 300 (hell).

Code innerhalb der I/O-Beschreibungsdatei:

```
/* Light Sensor 1 */
#Inputs: 1
#include "C:\legOS\include\dsensor.h"
void LightSensor1(SVartyp *a1)
{
############################################################
############################################################
}
/*end*/
```

Light Sensor 2 `LightSensor2`

Funktion:

Liefert am Blockausgang die an Sensor 2 gemessene Lichtintensität als (skalierten) Integer-Wert etwa im Bereich zwischen 50 (dunkel) und 300 (hell).

Code innerhalb der I/O-Beschreibungsdatei:

```
/* Light Sensor 2 */
#Inputs: 1
#include "C:\legOS\include\dsensor.h"
void LightSensor2(SVartyp *a1)
{
############################################################
############################################################
}
/*end*/
```

Light Sensor 3 `LightSensor3`

Funktion:

Liefert am Blockausgang die an Sensor 3 gemessene Lichtintensität als (skalierten) Integer-Wert etwa im Bereich zwischen 50 (dunkel) und 300 (hell).

Code innerhalb der I/O-Beschreibungsdatei:

```
/* Light Sensor 3 */
#Inputs: 1
#include "C:\legOS\include\dsensor.h"
void LightSensor3(SVartyp *a1)
```

```
{
############################################################
############################################################
}
/*end*/
```

Rotation Sensor 1 `RotationSensor1`

Funktion:

Liefert am Blockausgang den aktuellen Wert des Rotationssensors 1. Über HIGH-Pegel am Blockeingang (d. h. einen Signalwert > 2.5) wird der Sensor auf einen Anfangswert von 0 zurückgesetzt.

Code innerhalb der I/O-Beschreibungsdatei:

```
/* Rotation Sensor 1 */
#Inputs: 1
#Outputs: 1
#include "C:\legOS\include\dsensor.h"
void RotationSensor1(SVartyp e1, SVartyp *a1)
{
############################################################
############################################################
############################################################
############################################################
}
/*end*/
```

Rotation Sensor 2 `RotationSensor2`

Funktion:

Liefert am Blockausgang den aktuellen Wert des Rotationssensors 2. Über HIGH-Pegel am Blockeingang (d. h. einen Signalwert > 2.5) wird der Sensor auf einen Anfangswert von 0 zurückgesetzt.

Code innerhalb der I/O-Beschreibungsdatei:

```
/* Rotation Sensor 2 */
#Inputs: 1
#Outputs: 1
#include "C:\legOS\include\dsensor.h"
void RotationSensor2(SVartyp e1, SVartyp *a1)
{
############################################################
############################################################
############################################################
############################################################
}
/*end*/
```

Rotation Sensor 3 `RotationSensor3`

Funktion:

Liefert am Blockausgang den aktuellen Wert des Rotationssensors 3. Über HIGH-Pegel am Blockeingang (d. h. einen Signalwert > 2.5) wird der Sensor auf einen Anfangswert von 0 zurückgesetzt.

Code innerhalb der I/O-Beschreibungsdatei:

```
/* Rotation Sensor 3 */
#Inputs: 1
#Outputs: 1
#include "C:\legOS\include\dsensor.h"
void RotationSensor3(SVartyp e1, SVartyp *a1)
{
############################################################
############################################################
############################################################
############################################################
}
/*end*/
```

RCX-Bedientasten

View Button `ViewButton`

Funktion:

Liefert am Blockausgang HIGH-Pegel (d. h. einen Signalwert von 5), wenn der *View*-Taster auf dem RCX gedrückt ist, ansonsten LOW-Pegel (d. h. einen Signalwert von 0).

Code innerhalb der I/O-Beschreibungsdatei:

```
/* View Button */
#Inputs: 1
#include "C:\legOS\include\dbutton.h"
void ViewButton(SVartyp *a1)
{
############################################################
############################################################
############################################################
############################################################
}
/*end*/
```

Prgm Button `PrgmButton`

Funktion:

Liefert am Blockausgang HIGH-Pegel (d. h. einen Signalwert von 5), wenn der *Prgm*-Taster auf dem RCX gedrückt ist, ansonsten LOW-Pegel (d. h. einen Signalwert von 0).

Code innerhalb der I/O-Beschreibungsdatei:

```
/* Prgm Button */
#Inputs: 1
#include "C:\legOS\include\dbutton.h"
void PrgmButton(SVartyp *a1)
{
############################################################
############################################################
```

```
############################################################
############################################################
}
/*end*/
```

Sonstiges

Battery Voltage `BatteryVoltage`

Funktion:

Liefert die Batteriespannung des RCX als Integer-Wert in Millivolt.

Code innerhalb der I/O-Beschreibungsdatei:

```
/* BatteryVoltage */
#Inputs: 1
#include "C:\legOS\include\sys\battery.h"
void BatteryVoltage(SVartyp *a1)
{
############################################################
}
/*end*/
```