Technische Universität Ilmenau
Fakultät für Informatik und Automatisierung
Fachgebiet Rechnerarchitektur
Betreuer: Dr. Nützel

Hauptseminar
Wintersemester 2002/2003
zum Thema

# Software Entwurf für den RCX von Lego Mindstorms

Bearbeiter: Axel Schmiegel
Termin: 09. Februar 2003

# 1. Einleitung

Am Anfang meines Hauptseminars stand die Suche nach geeigneten Designtools zur Programmierung des RCX eines Legoroboters. Als Ergebnis dieser Suche fand ich etliche verschiedene Programmiersprachen zur Programmierung des RCX und 2 Tools die auf der Ebene der Modellbildung arbeiten. Dies ist zuerst einmal das Robolab. Vorteil dieses Tools ist das es speziell zur Programmierung des RCX designt ist. Einen Eindruck dieses Tools kann man durch die downloadbare Demo gewinnen. Nachteil ist das es Geld kostet. Deswegen wurde entschieden die weiteren Bemühungen auf das ECRobot zu richten.
Dies ist einer Erweiterung des Real-Time-Workshops und der Simulink Blockbibliothek, beides aus der Entwicklungsumgebung Matlab. Das ECRobot selber ist kostenlos und kann auf den Seiten von Mathworks herunter geladen werden, beziehungsweise ist in der neuen Matlab Version 6.5 R13 bereits enthalten.
Im nächsten Bild ist einmal grob dargestellt welche Komponenten am Entwicklungsprozess beteiligt sind. Im Einzelnen soll dann in den nächsten Kapiteln mehr oder weniger detailliert auf sie eingegangen werden. Sowie auf die Möglichkeiten die damit theoretisch zur Verfügung stehen, auch wenn ein Großteil derzeit nicht praktisch umgesetzt wurde.
Mit diesem Bericht soll hauptsächlich erreicht werden das jemand der daran interessiert ist sich mit Simulink zur Programmierung des RCX zu beschäftigen einen Überblick über die Möglichkeiten dieser Umgebung bekommt.

# 2. Simulink

Simulink ist natürlich mehr als eine Möglichkeit Legoroboter zu programmieren. Deswegen werde ich teilweise auch auf Aspekte von Simulink eingehen die nicht direkt mit der Programmierung eines RCX in Beziehung stehen.
Simulink ist ein Software Packet zum Modellieren, Simulieren und Analysieren dynamischer Systeme. Es unterstützt lineare und nichtlineare Systeme, modelliert zeitkontinuierliche und zeitdiskrete Systeme oder auch Hybridsysteme aus beiden.
Zum Modellieren stellt Simulink eine graphische Benutzeroberfläche zur Verfügung mit der Modelle als Blockdiagramm per Click-and-Drop zusammengebaut werden können. Mit diesem Interface ist es möglich Modelle wie mit Stift und Papier zu zeichnen. Dadurch wird man solange die vorhanden Blöcke zur Problemlösung genügen, voll von der in den Blöcken steckenden, teilweise komplizierten Funktionalität, verschont die man sonst selbst Programmieren müsste. Wenn die umfangreiche Blockbibliothek jedoch einmal nicht ausreichen sollte ist es mögliche eigene Blöcke zu erzeugen und einzubinden. Darauf wird im Kapitel S-Funktionen noch eingegangen.
Die Modelle sind hierarchisch aufgebaut, so kann man sich sowohl per Top-Down als auch Bottom-UP Methode der Problemlösung nähern. Per Doppelclick auf einen Block kann man die in ihm steckende Blockfunktionalität sehen, wenn es bereits ein Basisblock ist die einstellbaren Parameter. Dieser Aufbau ermöglicht eine übersichtliche Darstellung der Verbindungen/Interaktionen zwischen den Blöcken und so der dem Modell inne wohnenden Organisation.
Nach der Erstellung eines Modells, kann es Simuliert werden. Dazu stehen umfangreiche Methoden zur Signal-/Inputerzeugung zur Verfügung. Sowie diverse Möglichkeiten der Signalauswertung an den Ausgängen, außerdem ist es möglich die Ergebnisse auch direkt in den Matlab Workspace zu übernehmen und so beliebig weiterzuverarbeiten.
Man kann die Simulation kontinuierlich durchlaufen lassen, man kann aber auch im Schrittbetrieb Takt für Takt das Modell debuggen.
Im Folgenden wird nur noch auf bestimmte Konzepte von Simulink, die zum Verständnis der Zusammenarbeit mit dem ECRobot wichtig sind, eingegangen werden. Für diejenigen die sich intensiver mit dieser Materie beschäftigen wollen gibt es etliche Bücher zum Thema, die zum Bsp. in der Unibibliothek erhältlich sind.

## 2.1. Ein Beispielmodell

Bevor jetzt gleich in die Vollen gegangen wird möchte ich einmal kurz ein Beispielmodell vorstellen mit dem der so genannte ‚Acrobot' aus dem Lego Modellbau programmiert werden kann.

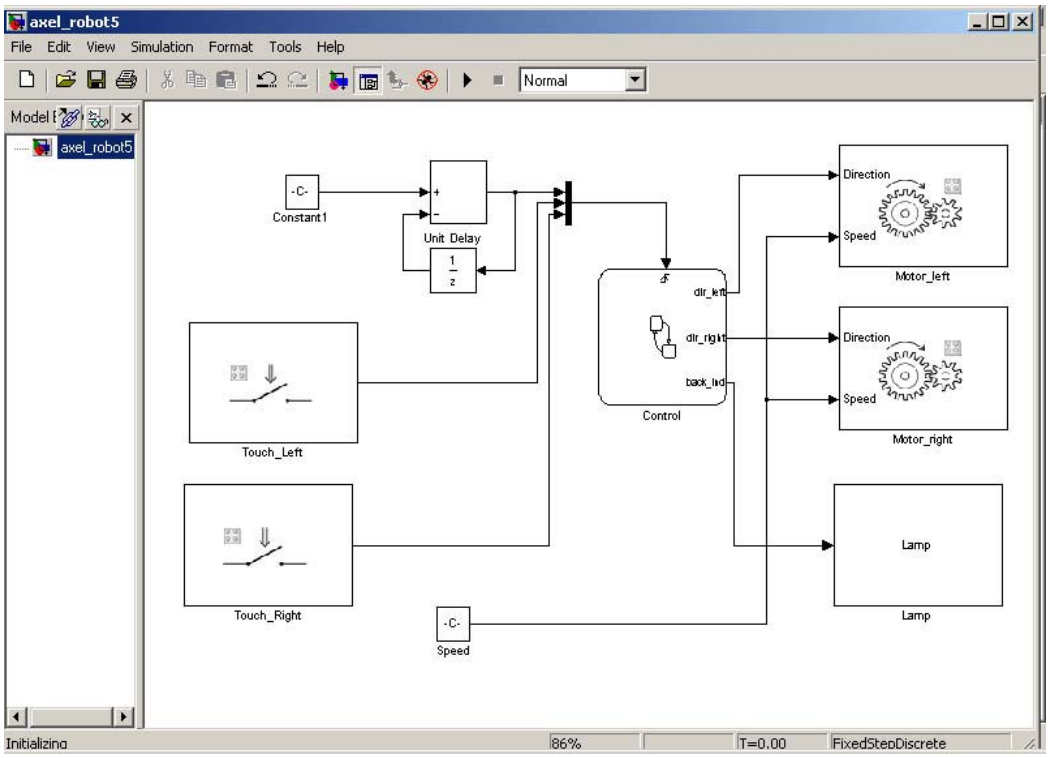


**Abb. 2.1/1 Robot5 Modell**

Zu sehen sind hier Eingangsblöcke(Touchsensoren), die Modellierung eines Softwaretimers, der Block eines Zustandsgraphen und Ausgangsblöcke(Motoren).
Nun werfen wir noch einen Blick auf das innere des Zustandsgraphen Blockes, um dann erst einmal mit den Erläuterungen der Simulink Komponenten Fortzufahren und dann später wieder auf dieses Modell und seine Details zurück zu kommen.

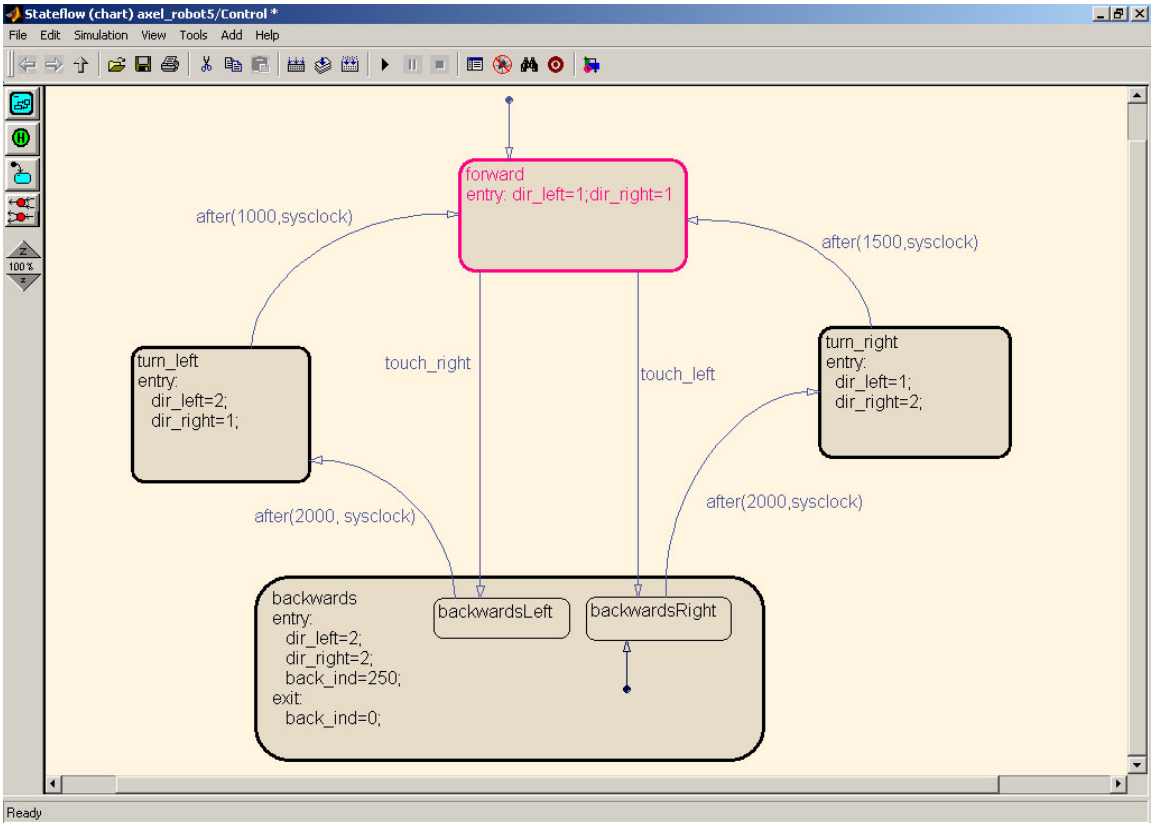


**Abb. 2.1/2 Stateflow zu Robot5**

## 2.2. Erläuterung einiger Begriffe

In diesem Kapitel werden Stichpunktartig einige der Begriffe die in der Simulinkwelt verwendet werden erläutert um so das Verständnis der folgenden Kapitel zu verbessern.

**Block Diagramm**
Ein Block Diagramm ist ein bildliches Modell eines dynamischen Systems. Es besteht aus einer Reihe von Symbolen, Blöcke genannt, verbunden durch Linien. Jeder Block repräsentiert ein elementares dynamisches System das eine kontinuierliche/diskrete Ausgabe erzeugt. Die Linien repräsentieren Verbindungen von Blockeingängen zu Blockausgängen. Der Typ eines Blocks bestimmt die Beziehung zwischen Ein- und Ausgängen.
Zu sehen war ein Block Diagramm bereits in Abb. 2-1.

**Blöcke**
Blöcke repräsentieren elementare dynamische Systeme, die durch Simulink simuliert werden können. Ein Block beinhaltet einen oder mehrere Sätze von Eingaben, Zuständen und Ausgaben. Der Blockausgang ist bestimmt durch eine Funktion abhängig von Eingängen, Zuständen und Zeit.
Ein Beispiel für einen Block ist der Motorblock aus dem Beispiel.

**Block Parameter**
Die Schlüsseleigenschaften von vielen Standardblöcken sind parametrisiert um möglichst flexible Aufgabenlösung zu gewährleisten. Jeder parametrisierbare Block hat einen Block Dialog in dem die Parameter geändert werden können. Wenn Parameter als ‚tunable' angelegt so können sie auch während einer Simulation geändert werden.
Im nächsten Bild sind die Blockparameter für den Motorblock zu sehen. Hier kann einmal die Sample Time und der Output Port eingestellt werden. Da bisher keine Unterstützung für die Rückkanalfähigkeit des RCX gegeben ist gibt es bei den Lego Blöcken keine tunebaren Parameter.

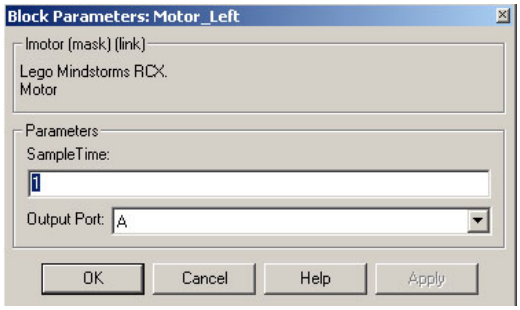


**Abb. 2.2/3 Motorblock Parameter**

**Sample Time**
Die Sample Time muss bei diskreten Blöcken so eingestellt werden das verbundene Blöcke dieselbe besitzen. Was genau mit Sample Time gemeint ist wird später noch genauer beleuchtet.

**Subsysteme**
Simulink erlaubt es ein komplexes System von verbundenen Subsystemen, welche durch ein eigenes Blockdiagramm repräsentiert werden, als Block darzustellen. Dazu erzeugt man einen Subsystemblock in dem man das Subblockdiagramm erstellt und mit den festgelegten Ein- und Ausgängen verbindet. Subsysteme könne beliebig Tief hierarchisch gestaffelt werden um Übersichtliche Modelle zu kreieren.

**Selbsterstellte Blöcke**
Es ist möglich in Simulink eigene Blöcke zu erstellen und in die Bibliotheken einzubinden. Dies kann grafisch oder über Programmierung geschehen. Grafisches erstellen ist dann ähnlich den Subsystemen nur das hier noch ein Parameterdialog dazukommt. Welche Möglichkeiten es hierfür gibt wird im Kapitel S-Funktionen noch genauer erläutert.

**Signale**
Simulink benutzt den Begriff Signal zur Bezeichnung der Ausgabe. Diese besteht aus einem breiten Spektrum von Signal Attributen, beinhaltend Datentyp(z.B. 8,16,32-bit integer), Zahlentyp(real oder komplex) und Dimensionalität(Arraygröße). Zu beachten ist das einige Blöcke nicht alle Signaltypen verarbeiten können.
Letztendlich schränkt natürlich meist das Zielsystem die Datentypen ein. Der einfache Anwender muss jedoch lediglich die in den Blöcken festgelegten Signaltypen beachten.

**Daten Typen**
Bei den Signalen sind die Datentypen durch die Verarbeitungsmöglichkeiten der Blöcke eingeschränkt. Bei den extern genutzten Variablen ist jedoch im Rahmen der Matlab Möglichkeiten jedoch freie Hand gelassen.
Die Nutzung externer Variablen macht für die Programmierung des RCX natürlich nur Sinn solange eine Rückkanalfähigkeit vorhanden ist.

## 2.3. Simulation dynamischer Systeme

Um die Funktionalität eines Modells zu überprüfen unterstützt Simulink den Nutzer mit einem breiten Spektrum an Debugging-Tools. Sowie Datendisplays und Blöcke zum mitloggen.
Die Simulierungsfähigkeit eines RCX-Modells ist derzeit dadurch eingeschränkt das die Device-Blöcke wie zum Beispiel der Motorblock nicht simuliert werden können. Prinzipiell ist dies aber durchaus möglich, natürlich nicht seiner Physik sondern nur in Form einer z.B. Drehzahlangabe. Derzeit müsste man den Motorblock für die Zeit der Simulierung z.B. durch ein Datendisplay ersetzen.

### 2.3.1. Solvers

Ein dynamisches System wird von Simulink, in aufeinander folgenden Zeitschritten über eine vorgegebene Zeitspanne, berechnet. Unter Nutzung der vom Modell gegebenen Informationen. Der Prozess der Berechnung der Folgezustände eines Systems mit seinem Modell wird Solving(Lösen) genannt. Es existiert keine bestimmte Methode ein Modell zu lösen deswegen kann man unter „Simulation Parameters“ unter verschiedenen Solvern wählen. Für den RCX ist zum Beispiel ein fixed-step Solver einzustellen.

### 2.3.2. Datenobjekte

In Datenobjekten werden die Informationen zum Modell oder der Simulation gespeichert. Wenn man direkt Zugriff auf diese nimmt dann ist es möglich noch mehr Einfluß zu nehmen als es aus der graphischen Oberfläche heraus möglich ist.

### 2.3.3. Erzeugen von Subsystems

Subsysteme sind logische Zusammenfassungen von Teilen des Modells um eine bessere Übersichtlichkeit zu erhalten.

Man kann ein Subsystem auf 2 Wegen erzeugen:
Indem man einen Subsystemblock in das Modell setzt diesen öffnet und die Blöcke einfügt die es enthalten soll.
Oder man erzeugt erst die Blöcke die das Subsystem ausmachen und gruppiert sie dann in ein Subsystem.

Für die erste Methode folgt man diesen Schritten:
1. Den Subsystemblock von Signals&Systems Bibliothek ins Modell kopieren
2. Den Subsystemblock per Doppelklick öffnen.
3. Im leeren Subsystemfenster kann man nun sein Subsystem unter Verwendung von Ein- und Ausgangsblöcken erzeugen.

Für den zweiten Weg markiert man die betreffenden Blöcke indem man einen Auswahlrahmen um sie zieht und schiebt sie dann per Edit->Create Subsystem ins Subsystem.

### 2.3.4. Nutzen von Callback Routinen

Man kann Matlab-Ausdrücke definieren, die ausgeführt werden wenn ein Blockdiagramm oder Block benutzt wird, in einer bestimmten Weise. Dies Ausdrücke Callback Routinen genannt sind verbunden mit dem Block, Port oder Modellparameter. Zum Beispiel der Callback der mit dem Block-OpenFcn-Parameter verbunden ist wird ausgeführt wenn der Modell-Nutzer eine Doppelklick auf den Block durchführt.

### 2.3.5. Bibliotheken

Bibliotheken ermöglichen es dem Nutzer Blöcke von externen Bibliotheken zu kopieren. Außerdem werden die Blöcke automatisch aktualisiert wenn der Quellblock verändert wird. Dies stellt sicher das in den Modellen die diese Bibliothek verwenden immer die aktuellsten Blockversionen genutzt werden.

**Terminologe**
Bibliothek :
ist eine Sammlung von Bibliotheksblöcken. Sie muss explizit durch „New Library“, aus dem File-Menü, erzeugt werden.
Bibliothek Block :
ein Block in einer Bibliothek
Referenz Block:
ist die Kopie eines Bibliotheksblockes.
Link:
Die Verbindung zwischen Referenz Block und Bibliotheksblock die es Simulink ermöglicht automatisch zu updaten.
Kopieren:
Die Funktion die eine Referenz des Blockes, von einem Bibliotheksblock oder von einem Referenzblock erzeugt.

**Simulink Blockbibliotheken**
Es gibt 2 Simulink Blockbibliotheken. Die Erste gibt die aktuelle Organisation der Blöcke an und kann durch „simulink3“ aufgerufen werden. Die Zweite, Aufrufbar mit „simulink“ zeigt die Organisation der Simulink Version 2 und früher.

**Bibliothek Links**

Der Link zum Bibliotheksblock wird auch hergestellt wenn man von einem Referenzblock kopiert.
Der Link wird über den Namen des Bibliothekblocks, während des Kopierens, hergestellt.
Wenn Simulink nicht in der Lage ist den Library Block zu finden, wird dies durch eine Fehlermeldung kenntlich gemacht. Außerdem steht im problematischen Block „Bad Link“.

**Abschalten von Bibliothek Links**
Simulink erlaubt die Links ab-/einzuschalten über Edit->Link Options.

**Modifizieren eines verlinkten Subsystems**
Simulink ermöglicht es Subsysteme zu modifizieren die Bibliothek Links haben. Modifizieren heißt hier strukturelle Änderungen durchzuführen, bei Parameteränderungen gibt es nichts zu beachten. Jedenfalls muss der Link abgeschaltet werden um die Änderungen im Modell durchführen zu können

**Brechen eines Links zum Library Block**
Das brechen des Links führt dazu das es keine Verbindung des Kopierten Blockes mehr zur Bibliothek gibt. Auf diese Weise kann man Stand-Alone-Modelle erzeugen die nicht mehr auf die Bibliothek angewiesen sind.

**Finden des Bibliothek Blocks über den Link**
Über „Link Options“->“Go to Library Link” kann man dies erreichen.

**Anzeigen von Bibliothek-Links**
Man kann das bestehen eines Links in der unteren linken Ecke eines Blocks anzeigen lassen. Dies geschieht durch Auswahl über das Menü „Format“->“Library Link Display“.

**Browsen in Block Bibliotheken**
Der Bibliothek Browser ermöglicht es auf einfachen und übersichtlichen Weg die Block-Bibliotheken zu durchsuchen. Erreichen kann man ihn über die Menüleiste oder durch eingeben des Befehls „simulink“ in die Kommandozeile.

### 2.3.6. Analyse von Simulations-Resultaten

**Betrachten von Ausgaben**
Ausgabe der Simulationsergebnisse kann auf 3 Wegen erfolgen:
- Signale in einen Scope- oder XY-Graph-Block führen.
- Die Ausgabe in eine Variable schreiben und die Matlab Plot Kommandos benutzen.
- Die Ausgabe in den Workspace schreiben.

### 2.3.7. Bedingte Ausführugn von Subsystemen

Ein bedingt ausgeführtes Subsystem ist ein Subsystem dessen Ausführung vom Wert eines Input-Signals abhängig ist. Dieses Signal wird Kontrollsignal genannt.
Simulink unterstützt 3 Typen von bedingt auszuführenden Subsystemen:
- Das **enabled subsystem** wird ausgeführt wenn das Kontrollsignal positive ist.
- Das **triggered subsystem** wird ausgeführt zu jedem Zeitpunkt an dem ein Trigger-Ereignis auftritt. Ein Trigger-Ereignis kann die fallende oder steigende Signalflanke sein.

-Ein **control flow statement** führt C-ähnlich Kontrollflusslogik unter der Beaufsichtigung eines Kontrollflussblockes durch. Damit kann man beispielsweise if-then oder switch Konstrukte modellieren.

## 2.4. S-Funktionen

S-Funktionen erlauben es eigene Blöcke zu Simulink Modellen hinzuzufügen. Man kann seine eigenen Blöcke erstellen im MATLAB®(die Matlab eigenen Programmiersprache), C, C++, Fortran oder Ada. Durch das befolgen von ein Paar einfachen Regeln kann man so seine eigenen Algorithmen in S-Funktionen implementieren. Nachdem man seine S-Funktion geschrieben und den Namen in einem S-Funktionen-Block abgelegt hat, kann man das Userinterface durch Maskieren individuell gestalten.
S-Funktionen können mit dem Real-Time-Workshop benutzt werde. Man kann die Codegenerierung, mit dem Real Time Workshop® für S-Funktionen, individualisieren. Durch das schreiben eines Target Language Compiler (TLC) Files.
S-Funktionen werden kompiliert als MEX-Files, durch Benutzung der Mex-Utility, beschrieben im Application Program Interface Guide. Zusammen mit anderen MEX-Files, werden sie dynamisch in Matlab gelinkt wenn benötigt.
Eine spezielle Aufruf-Syntax wird genutzt, damit die Simulink S-Funktionen mit dem equation solvers interagieren können. Diese Interaktion ist sehr ähnlich zur Interaktion die stattfindet zwischen dem Solver und built-in Simulink Blöcken. Die Form einer S-Funktion ist sehr allgemein und kann kontinuierliche, diskrete und hybride Systeme beinhalten.

### 2.4.1. Nutzen von S-Funktionen in Modellen

Um eine S-Funktion in ein Simulink Modell zu integrieren zieht man einen S-Funktion Block von der Functions & Tables Block Bibliothek in das Modell. Dann trägt man den Namen der S-Funktion in das dafür vorgesehen Feld in der Dialog-Box ein.
Die S-Funktion kann sowohl als C mex-File als auch als m-File eingebunden werden. Wenn beide Filetypen existieren, mit dem selben Namen, dann wird das C mex-File benutzt
Die Parameter die der Funktion mitgegeben werden sollen, kann man durch Komma getrennt in das Parameterfeld im Dialog eintragen.

**Wann sollten S-Funktionen genutzt werden**
Am gebräuchlichsten ist die Nutzung der S-Funktion zur Erzeugung angepasster Simulink Blöcke. Man kann S-Funktionen für eine Vielfalt von Anwendungen nutzen, inbegriffen:

- hinzufügen allgemeiner Zweck-Blöcke zu Simulink
- hinzufügen von Blöcken die Gerätetreiber für Hardware repräsentieren
- aufnehmen von existierendem C-Code in Simulationen
- Beschreiben eines Systems als mathematische Satz von Gleichungen
- Nutzen von graphischen Animationen

Der Nutzen von S-Funktionen ist, dass man Blöcke bauen kann die mehrmals im Modell genutzt werden, mit variierenden Parametern bei jeder Instanz des Blockes.

### 2.4.2. Wie S-Funktionen arbeiten

**Mathematik von Simulink Blöcken**
Ein Simulink Block besteht aus einer Menge von Inputs, Zuständen und Outputs. Wobei der Output von der Samplezeit den Inputs und Zuständen abhängig ist.

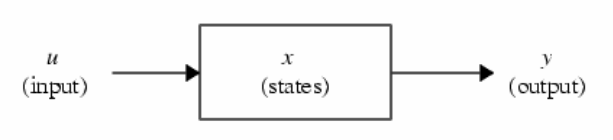


**Abb. 2.8.2/ 4 Blockgraph**

Die folgenden Gleichungen drücken die mathematischen Beziehungen zwischen Inputs, Outputs und Zuständen aus.

$$y = f_0(t, x, u) \qquad \text{(output)}$$

$$\dot{x}_c = f_d(t, x, u) \qquad \text{(derivative)}$$

$$x_{d_{k+1}} = f_u(t, x, u) \qquad \text{(update)}$$

$$\text{where} \quad x = x_c + x_d$$

**Abb. 2.8.2/5 Blockformeln**

**Simulationsphasen**

Die Ausführung eines Simulink Modells passiert in Phasen. Zuerst kommt die Initialisierungsphase. In dieser Phase bindet Simulink Bibliotheksblöcke ein, legt Daten Typen und Sample Zeit fest, wertet Blockparameter aus, setzt die Blockausführungsreihenfolge fest und weist den benötigten Speicher zu.
Dann betritt Simulink eine Simulationsschleife, bei der jeder Schleifendurchgang einem Simulationsschritt entspricht. Während jedes Simulationsschrittes führt Simulink jeden der Modellblöcke in der bei der Initialisierung bestimmten Reihenfolge aus. Für jeden Block, ruft Simulink Funktionen, die die Block Zustände, Ableitungen und Outputs für die aktuelle Sample Zeit ausführen, auf. Dies wird fortgesetzt bis die Simulation vollständig ist.

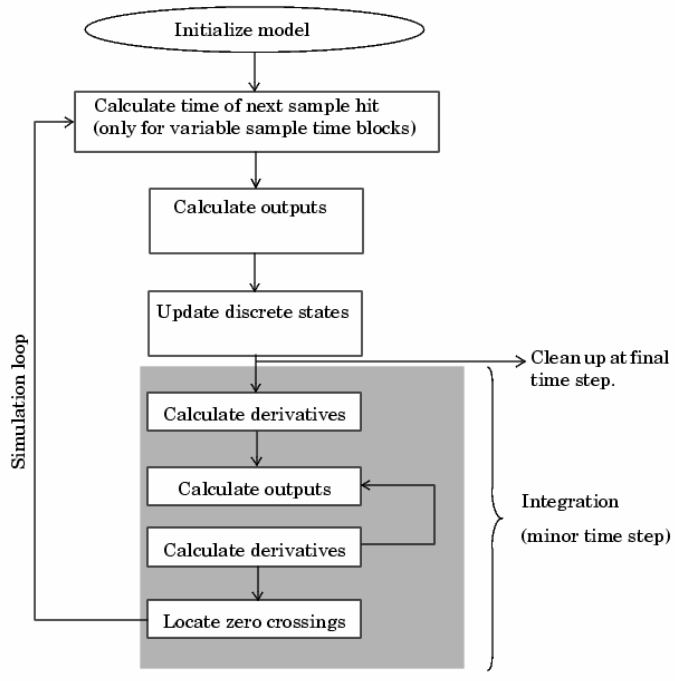


**Abb. 2.8.2/6 Simulationsphasen**

**S-Funktion Callback Methoden**
Eine S-Funktion umfasst eine Menge von S-Funktion-Callback-Methoden die zur Ausführung der Aufgaben gebraucht werden. Während der Simulation des Modells, zu jeder Simulationsphase, ruft Simulink die passenden Methoden für jeden S-Funktions-Block im Modell, auf. Die Aufgaben der S-Funktions-Methoden beinhalten:

- Initialisierung: Vor dem ersten durchlauf der Simulationsschleife initialisiert Simulink die S-Funktionen. Während dieser Phase, Simulink:
    - initialisiert den SimStruct, eine Simulationsstruktur die Informationen über die S-Funktionen enthält
    - die Anzahl von Input und Outputports setzt
    - die Block-Sample-Zeit setzt
    - den Speicher und die Größe der Arrays zuweist
- Berechnung des nächsten Sample Hits: Wenn man einen variablen Sample-Time-Block konstruiert hat, berechnet diese Phase den Zeitpunkt für den nächsten Sample Hit, es wird also die Schrittweite des nächsten Schrittes berechnet.
- Berechnung der Outputs im Major-Zeitschritt: Nach diesem Aufruf, sind alle Output-Ports dieses Blockes für den aktuellen Zeitschritt festgelegt.
- Updaten der diskreten Zustände im Major-Zeitschritt: In diesem Aufruf, sollten alle Blöcke die once-per-time-step Aktivitäten ausführen. Wie z.B. updaten der diskreten Zustände für die nächste Zeit des Simulationsschleifendurchlaufs.
- Integrieren: Dies wird für Modelle mit kontinuierlichen und/oder nonsampled Zero Crossings benutzt. Wenn die S-Funktion kontinuierliche Zustände hat, ruft Simulink die Output und abgeleiteten Anteile der S-Funktion zu jedem Minor-Zeitschritt auf. Ist es so, kann Simulink die Zustände der S-Funktion berechnen. Wenn die S-Funktion (nur bei C MEX) nonsampled zero crossings hat, dann wird Simulink die Outputs und Zero Crossing Anteile der S-Funktion in Minor-Zeitschritten berechnen, so das der genaue Zeitpunkt des Zero crossings bestimmt werden kann.

### 2.4.3. Implementieren von S-Funktionen

Man kann S-Funktionen als M-File oder als Mex-File implementieren. Im Folgenden werden diese alternativen Implementierungsformen beschrieben und ihre Unterschiede diskutiert.

**M-file S-Funktionen**
Eine M-File S-Funktion besteht aus einer Matlab-Funktion der folgenden Form.

```
[sys,x0,str,ts]=f(t,x,u,flag,p1,p2,...)
```

wobei f der Funktionsname ist, t die aktuelle Zeit, x der Zustandsvektor des zugehörigen S-Funktionsblock und u der Blockinput. *Flag* weist eine auszuführende Aufgabe zu und p1, p2,… sind Blockparameter. Während der Simulation ruft Simulink wiederholt f auf und benutzt *flag* um die auszuführende Aufgabe zu spezifizieren. Jedes Mal wenn die S-Funktion ihre Aufgabe ausgeführt hat gibt sie das Ergebnis in einer Struktur zurück, wie im Syntax-Beispiel zu sehen.

| Simulation Stage | S-Function Routine | Flag |
|---|---|---|
| Initialization | `mdlInitializeSizes` | `flag = 0` |
| Calculation of next sample hit (variable sample time block only) | `mdlGetTimeOfNextVarHit` | `flag = 4` |
| Calculation of outputs | `mdlOutputs` | `flag = 3` |
| Update discrete states | `mdlUpdate` | `flag = 2` |
| Calculation of derivatives | `mdlDerivatives` | `flag = 1` |
| End of simulation tasks | `mdlTerminate` | `flag = 9` |

**Abb. 2.8.3/7 M-File S-Funktionen**

Diese Tabelle bezieht sich auf die S-Funktion sfuntmpl.m, diese stellt eine Schablone zur Erstellung eigener S-Funktionen dar.

**MEX-file S-Functions**

Ähnlich der M-File S-Funktion, besteht die Mex-File-Funktion aus einer Menge von callback Routinen die Simulink aufruft um die verschiedenen Blockaufgaben während der Simulation auszuführen. Signifikante Unterschiede existieren jedoch. Als erstes, Mex-Files Funktionen werden in anderen Programmiersprachen geschrieben: C, C++, Ada oder Fortran. Außerdem ruft Simulink Mex S-Funktions-Routinen direkt anstelle über Flags auf. Weil Simulink die Funktionen direkt aufruft müssen Mex-File-Funktionen von Simulink standardisierten Namenskonventionen folgen. Andere Schlüsselunterschiede existieren ebenfalls. Erstens, die Menge der callback Funktionen die Mex-Funktionen implementieren können ist viel größer als die mit M-Files implementierbaren. Außerdem hat eine Mex-Funktion direkten Zugriff auf die interne Datenstruktur, SimStruct genannt, die Simulink zur Verwaltung der Informationen der S-Funktionen benutzt. Mex-File-Funktionen können außerdem Matlab's Mex-File API nutzen um direkt auf den Matlab Workspace zuzugreifen.

Eine Vorlage für eine C Mex-File S-Funktion findet man mit der sfuntmpl_basic.c. Diese Schablone beinhaltet skelettartig die benötigten und optionalen callback Routinen die ein C Mex-File S-Funktion implementieren kann. Eine umfangreicher kommentierte Version der Schablone kann man in der Datei sfuntmpl_doc.c finden.

**MEX-file Versus M-file S-Funktionen**

M-File und Mex-File S-Funktionen habe beide Vorteile. Der Vorteil von M-File S-Funktionen ist die Geschwindigkeit der Entwicklung. Die Entwicklung von M-File S-Funktionen vermeidet die Zeitaufwendigen compile-link-execute Zyklen die benötigt werden in einer zu compilierenden Sprache. M-File S-Funktionen haben außerdem einen einfacheren Zugriff auf Matlab und Toolbox Funktionen.
Der primäre Vorteil von Mex-File Funktionen ist Vielseitigkeit. Die größere Anzahl der Callbacks und der Zugriffe auf die SimStruct ermöglichen Mex-File-Funktionen Funktionalität zu implementieren die M-File-Funktionen nicht zugänglich ist. Solche Funktionalität beinhaltet die Fähigkeit andere Datentypen zu nutzen als double, complex inputs, matrix inputs und so weiter.

# 3. Real-Time-Workshop

Der Real-Time-Workshop erzeugt Programme von Simulink Entwürfen zum Prototypen, Testen und Einsetzen von Real-Time Systemen auf einer Vielzahl von Ziel-Plattformen. Nutzer des Real-Time-Workshops können direkt Source Code generieren, der den Compiler enthält, Input und Output Geräte, Speicher-Modelle, Kommunikationsknoten und andere Charakteristiken die eine Applikation möglicherweise fordert.

### 3.1.1. Komponenten und Features

Die prinzipiellen Komponenten und Features des Real-Time-Workshops sind:

- Simulink Code Generator: generiert automatisch C-Code aus dem Simulink-Modell
- Make Process: Der Real-Time-Workshop Nutzer erweiterbare make Prozess ermöglicht es ein eigenes Produktions oder Rapid Prototyping Target zu kreieren.
- Simulink external Mode: ermöglicht Kommunikation zwischen Simulink und einer Real-Time-Testumgebung, oder mit einem anderen Prozess auf derselben Maschine.
  Der External-Mode ermöglicht Real-Time Parameter Tuning und Datenbeobachtung mit Simulink als Benutzerinterface.
- Targeting Support: Wenn man die Targets dem Real-Time-Workshop beigelegt sind nutz, kann man Systeme für eine Anzahl von Umgebungen, einschließlich Tornado und Dos erzeugen. Die generischen Real-Time und Embedded Real-Time Targets sorgen für einen Rahmen zum entwickeln persönlich angepasster Rapid Prototyping oder Produktions Target Umgebungen.
- Rapid Simulation: Die Nutzung des Simulink Beschleunigers, dem S-Funktion Target oder dem Rapid Simulation Target ermöglicht 5 bis 20mal so schnell zu simulieren. Executables werden, mit diesen Targets erzeugt, die den normalen interpretativen Simulinkcode umgehen. Der entstehende Code ist hochoptimiert für genau die genutzten Funktionen.

### 3.1.2. Fähigkeiten und Vorteile

**Code Generator für Simulink Modelle**

- Generiert optimierten, personalisierten Code. Es gibt mehrere Stiele von generierbarem Code, welche entweder als Eingebettet oder Rapid Prototyping klassifiziert werden können.
- Unterstützt alle Simulink Features, einschlieslich 8, 16, 32 bit Integers und Float Datentypen.
- Die Fixed-point Fähigkeit vom Real-Time Workshop erlaubt die Vergrößerung der Integer Größe von 2 bis 128 bits. Die Code-Generierung ist Beschränkt durch die Implementation von char, short, int, und long in Embedded C Compiler Umgebungen(gewöhnlich 8, 16, and 32 bits).
- Der generierte Code ist Prozessor unabhängig. Er repräsentiert das Modell exakt. Ein separates Run-Time-Interface wird genutzt um diesen Code auszuführen. Es sind etliche Beispiele für Run-Time-Interfaces, ebenso für Produktions Run-Time-Interfaces, bereits mitgeliefert.
- Unterstützung von einigen Single und Multitasking Systemen.

- Die flexiblen Skriptmöglichkeiten des Target Language Compilers ermöglichen volle Umsetzung der eigenen Codes.
- Effizienter Code für S-Funktionen kann erzeugt werden durch Nutzung der Target Language Comiler Instruktionen (TLC Scripts genannt) und kann automatisch, mit generierten Code, integriert werden.

**Umfassende Modell basierte Debugging Unterstützung**

- Externe Modi erlauben es zu überprüfen was der generierte Code tut, wie das auslesen von Daten von dem graphischen Displayelementen im Modell, Es besteht keine Notwendigkeit einen konventionellen Source-Level-Debugger zu Nutzen um den generierten Code zu betrachten.
- Der Außen-Mode ermöglicht es ebenso den gerierten Code via Simulink Modell zu tunen. Wenn man die Parameter Werte, in den Blöcken im Modell, verändert wird der neue Wert zum generierten Code durchgereicht, der auf dem Ziel läuft, und die korrespondierende Target Speicher Stelle wir geupdatet. Noch mal, es ist nicht nötig einen Embedded Compiler Debugger zu nutzen um die Form von Operationen auszuführen. Das Modell ist das Debugger User Interface.

**Integration mit Simulink**

- Code Überprüfung: Man kann Code für ein Modell generieren und ein stand-alone Executable das den generierten Code benutzt und ein MAT-File erzeugt, das die Ausführungsergebnisse beinhaltet. Der generierte Code enthält System und Block Indentifications Tags um bei der Identifikation des Blocks zu helfen, im Source Modell, dass die gegebenen Zeile Code erzeugt hat. Das Matlab *command highlite system* erkennt diese Tags und hebt den korrespondierenden Block im Modell hervor. Unterstützung für Simulink Daten Objekte ermöglicht zu definieren wie die Signale und Blockparameter mit der externen Welt verbunden sind.

**Rapid simulations**

- Der Real-Time Workshop unterstützt etliche Wege die Geschwindigkeit der Simulation zu erhöhen durch optimierte, Modell-Spezifische Executables.

**Target Support**

- Schlüssellösungen für Rapid Prototyping reduzieren wesentlich Design Zyklen und erlauben schnelle Änderungen von Design Wiederholungen.
- Gebündelte Rapid Prototyping Beispiel Targets sorgen für Code den man modifizieren und schnell nutzen kann.
- Add-on Targets (Real-Time Windows Target und xPC Target) für PC basierte Hardware sind verfügbar von MathWorks. Diese Targets erlauben es einen PC zu verändern mit schneller, hoch qualitativer, low-cost hardware in ein Rapid Prototyping System.
- Es unterstützt eine Vielfalt von Dritt-Anbieter Hardware und Tools, mit erweiterbarer Gerätetreiber Unterstützung.

**Extensible make process**

- Erlaubt einfache Integration mit beliebigen Compilern und Linkern.
- Sieht vor einfaches linken mit Hand geschriebenen bewachenden oder unterstützenden Code.

### 3.1.3. Rollen der MathWorks Produkte in der Software Entwicklung

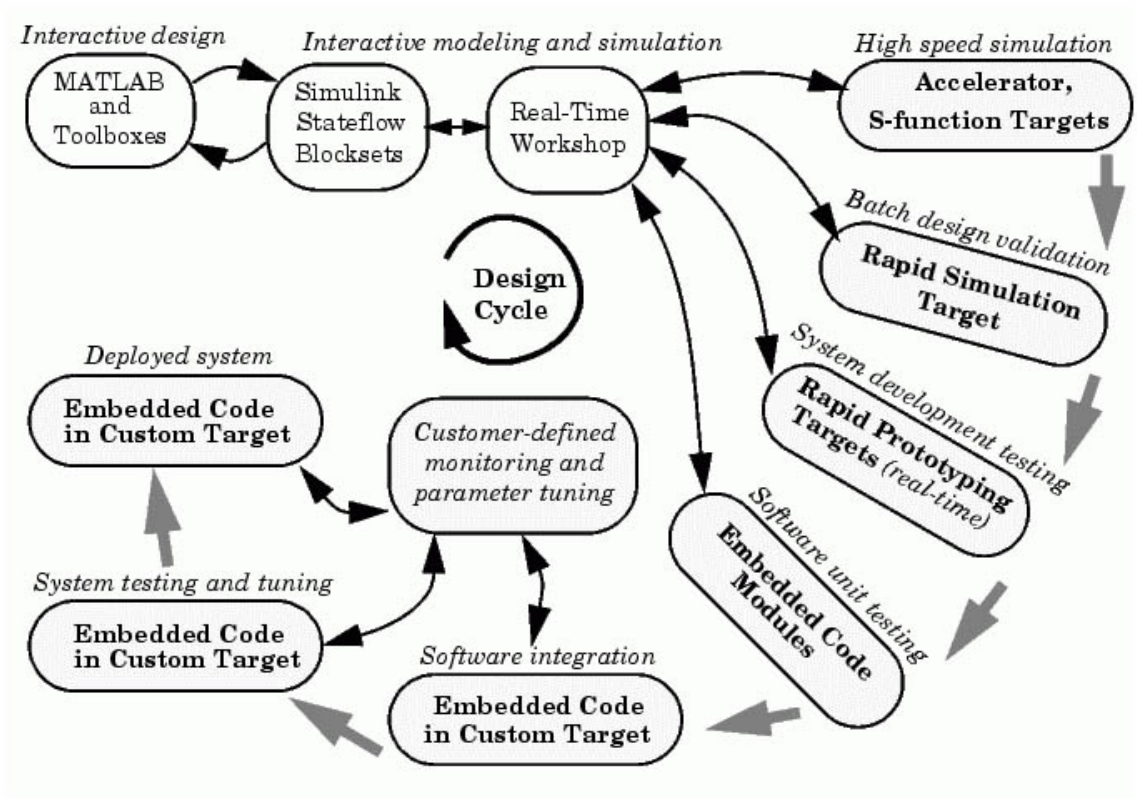


**Abb. 3.1.3/8 Entwicklungszyklus**

### 3.1.4. Unterstützte Compiler

Unter Windows werden die Compiler von Borland, LCC, Microsoft Visual C/C++ und Watcom unterstützt. Unter Linux ist als Default der cc eingestellt, der gcc wird ebenfalls unterstützt.

# 4. Stateflow

Stateflow ist ein graphisches Design und Entwicklungstool zur Simulation komplexer reaktiver Systeme basierend auf der finiten Zustandsmaschinen Theorie.
Auf den Stateflow soll jedoch nicht tiefer eingegangen werden, wie das bei Simulink der Fall war. Da dieses Paper sowieso schon viel zu umfangreich geworden ist. Stattdessen soll an dieser Stelle auf den im Beispiel verwendeten Stateflow Graphen eingegangen werden.

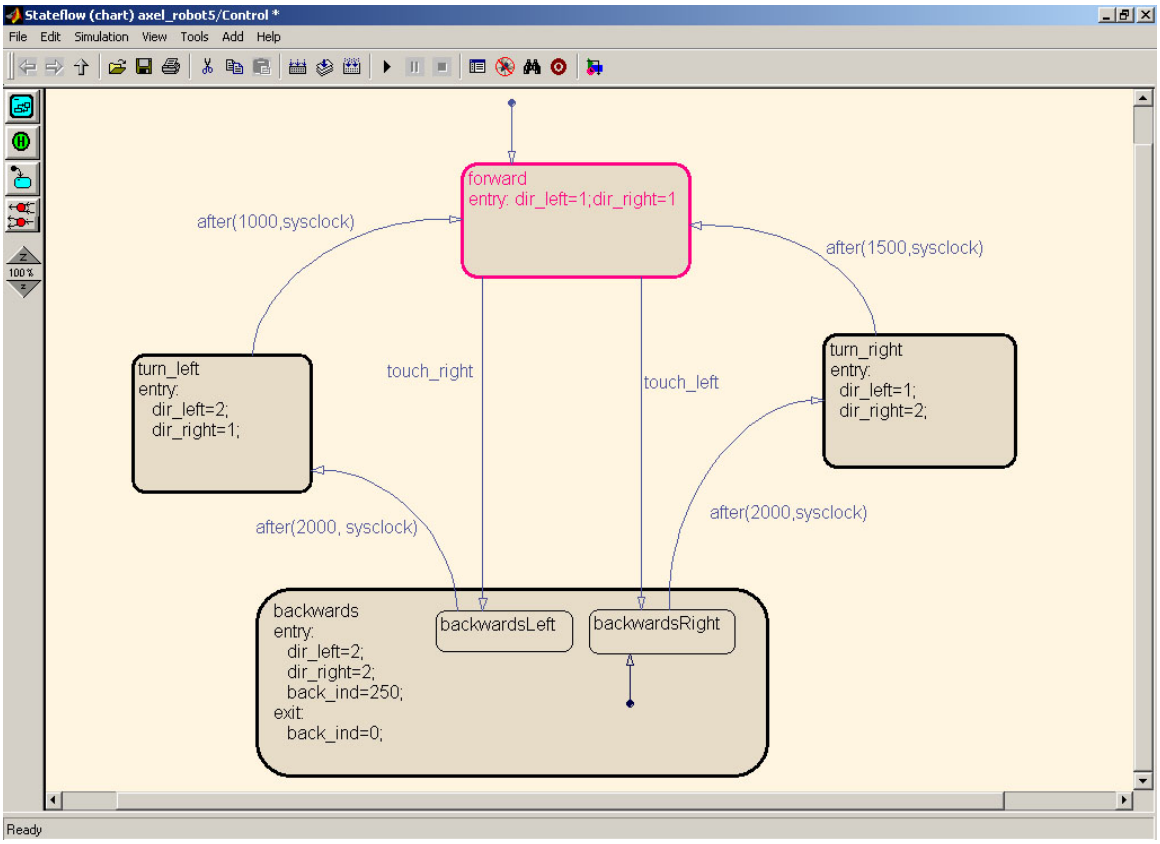


**Abb. 4/9 Stateflow Robot5**

Um zu verstehen wie dieser Stateflow funktioniert braucht man natürlich noch einige Definitionen, diese sind im nächsten Bild zu sehen.

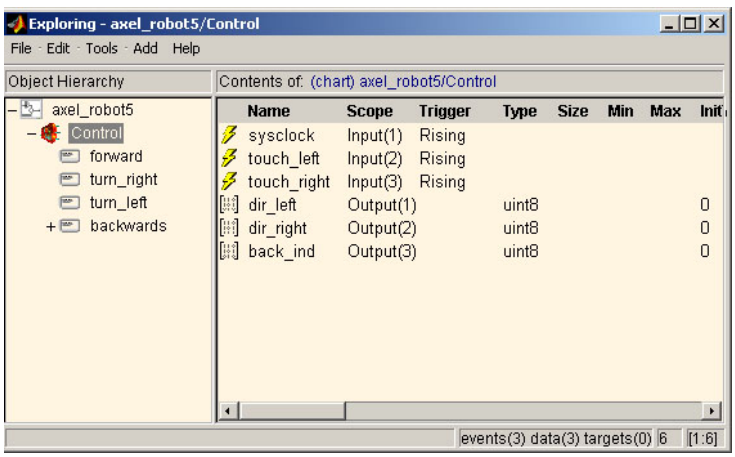


**Abb. 4/10 Stateflow Definitionen**

Nun sollte es recht gut erkennbar sein, wie die Inputs das durchlaufen des Statefllows bestimmen und dadurch die Ausgangswerte gesetzt werden. Zum Beispiel wenn der rechte Berührungsensor betätigt wird wandert man aus dem Initialen Zustand Oben in den unteren Zustand wandert wo für 2000 Zeittakte der rechte und linke Motor auf Rückwärts (=2) gesetzt werden, ebenso wie die Lampe eingeschaltet wird. Dann wird für 1000 Zeittakte der rechte Motor auf vorwärts gesetzt während der Linke auf rückwärts bleibt. Dann wird in den Initialzustand zurückgewandert in dem beide Motoren vorwärts fahren.

# 5. ECRobot

In diesem Kapitel soll, nachdem etliche der beteiligten Komponenten im Einzelnen beleuchtet worden, darauf eingegangen werden in welcher Form sie zusammen arbeiten.

## 5.1. Die beteiligten Komponenten und ihre Einordnung

Das ECRobot Target wurde hauptsächlich durch das modifizieren von Code und Control Files, die der Real-Time-Workshop bereits mitbringt, erstellt. Diese Files fallen in 2 Kategorien:

- Das *Run-Time-Interface* besteht aus Code der die Ausführung von generiertem Modell Code überwacht und Unterstützt. Im ECRobot Target, beinhaltet das Run-Time Interface das Haupt Programm, eingebundene Geräte Treiber und Header Files die die Anbindung an den LegOS Kernel gewährleisten.
- *Control Files*. Die Code Generierung und der Build Prozess werden gesteuert durch ein angepasstes System Target File und Template Makefiles. Die angepassten Kontroll Files starten einen Target spezifischen Cross-Compiler und (optional) downloaden des generierte Programms ins Zielsystem.

LegOS ist das Betriebssystem das auf dem RCX läuft und ist eine unabhängige Freeware Entwicklung. Der verwendete Compiler ist der GCC Cross-Compiler für den Hitachi h8300 Mikrocontroller. Zum übertragen des LegOS auf den RCX und dem benutzen des Compilers wird das Programm Cygnus benötigt. Dies emuliert eine Unix-Umgebung die nötig ist um mit den genannten Elementen arbeiten zu können. Cygnus ist ebenfalls frei erhältlich.

### 5.1.1. Main Program

Das Hauptprogramm für das ECRobot Target ist ECRobot_main.c. Das Hauptprogramm ist abgeleitet von ert_main.c , dem Template Embedded Coder Main Programm.

### 5.1.2. Geräte Treiber

Das ECRobot Target beinhaltet Geräte Treiber S-Funktions Blöcke und C MEX-File Komponenten zur Nutzung in Simulationen, ebenso wie korrespondierende TLC Files für die Generierung von Inline Code. Für jeden Treiber sind die zugehörigen Files:

- *driver.c*: Implementierung der Treiber S-Funktion für die Simulation
- *driver.dll*: MEX-File Komponente
- *driver_img.tif*: Icon für den Block
- *driver.tlc*: Implementation des Blocks für Inline Code Generierung

Die ECRobot Geräte Treiber Blöcke sind in einer Bibliothek gesammelt (ECRobot.mdl). Ein Treiberblock ist z.B. der Motorblock.

### 5.1.3. System Target File

Das System Target File für das ECRobot Target ist ECRobot.tlc, abgeleitet von ert.tlc. Anmerkung: Das ECRobot Target unterstützt keinen external Mode. Das heißt das die Rückkanalfähigkeit des RCX derzeit nicht unterstützt wird.

### 5.1.4. Code Generierungs Optionen

Die Codegenerierungsoptionen sind in der Datei *ecrobot_settings.tlc* abgelegt.

### 5.1.5. Template Makefile

Das Template Makefile für das ECRobot Target ist ECRobot.tmf. Es ist eine angepasste Version des ert_unix.tmf.
Die wichtigsten Anpassungen sind:

- Der Build Prozess ruft gmake auf, ein Make Utility das mit dem Real-Time-Workshop installiert wird.
- Das generierte Makefile startet den GCC Cross-Compiler für den Hitachi h8300 Mikrocontroller und spezifiziert Kommandozeilenparameter für den Compiler.
- Das generierte Makefile hat den Namen model.lx. Optionell, wird der generierte Code in den RCX herunter geladen via dem dll-Utility (hat nichts mit den *.dll zu tun)

Das Dll-Utility ist das Downloadtool für den RCX. Es wird sowohl zum Einspielen des Betriebsystems als auch der Programme benötigt. Es kann ebenfalls nur in der von Cygnus emulierten Unix Umgebung genutzt werden.

Das nächste Bild zeigt die Struktur eines vom ECRobot Target generiertem Programms wenn es auf dem RCX arbeitet.

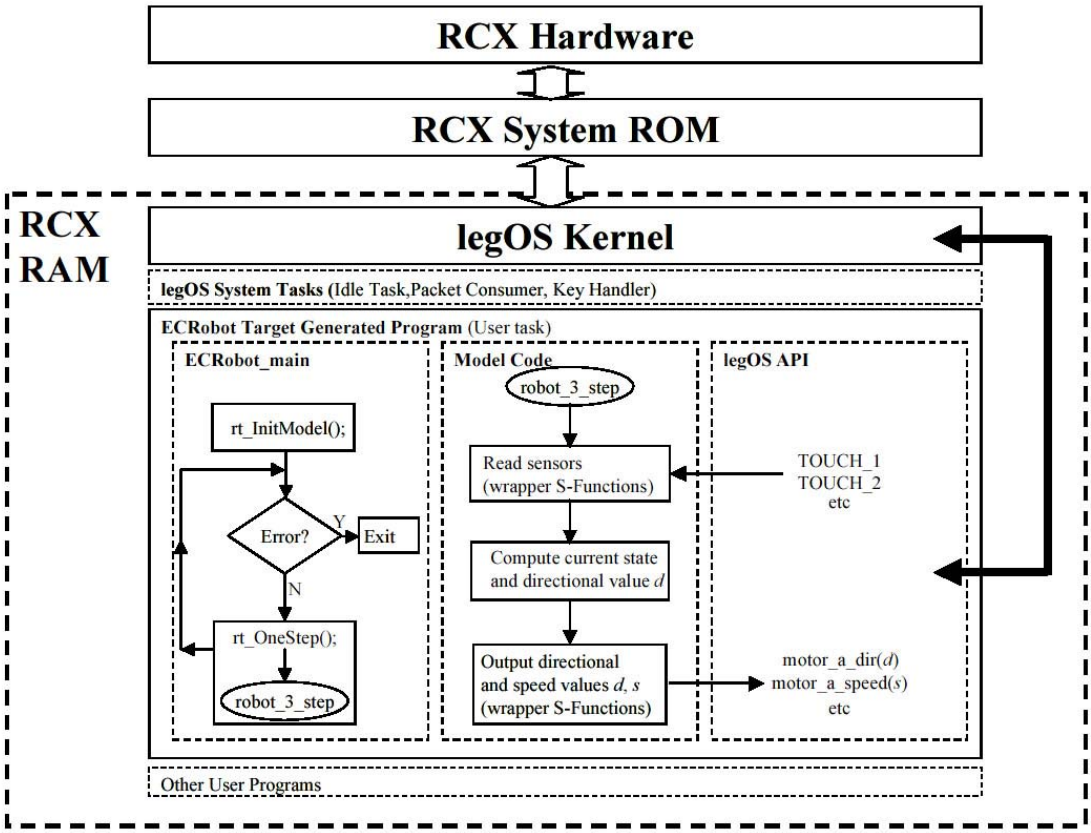


**Abb. 5.1.5/11 RCX**

Der RCX Ram ist zwischen 0x8000 und 0xFFFF lokalisiert, im RCX Adressraum. Der LegOS Kernel liegt im niedrigsten Teil des Ram's. Der verbleibende RAM ist zugeteilt zu:

- LegOS System Tasks. Wenn LegOS gebootet ist, dann startet es die System Tasks, welche laufen solange LegOS läuft. Ein Beispiel für einen System Task ist der *packet consumer* , welcher die IR Kommunikation mit dem Host Computer handelt und Nutzer Programme downloaded und in den Speicher legt.
- Reservierte Plätze für Speicher-Mapped Register, I/O-Ports und Interrupt Vektoren.
- Nutzer Programme sind auswechslebare Code Module, deren Adresse bestimmt wird, wenn sie durch den packet handler geladen werden.

### 5.1.6. Übersicht

Im folgenden Bild sind die beteiligten Komponenten und ihre Einordnung im Coderzeugungsprozess einmal vereinfacht dargestellt.

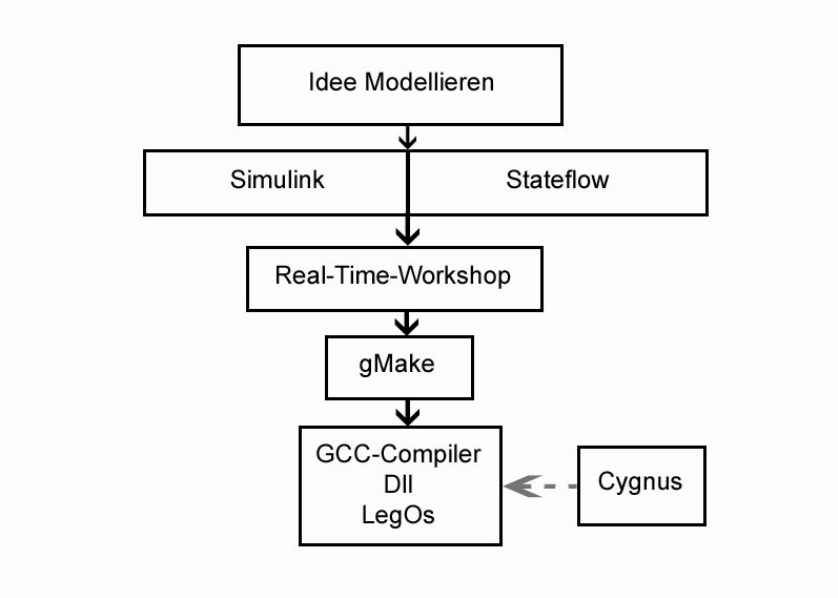


**Abb. 5.1.6/12 Übersicht der beteiligten Komponenten**

## 5.2. Die Blockbibliothek

In der folgenden Grafik sind die auf den RCX bezogenen Blöcke dargestellt, die durch das ECRobot eingebracht werden.

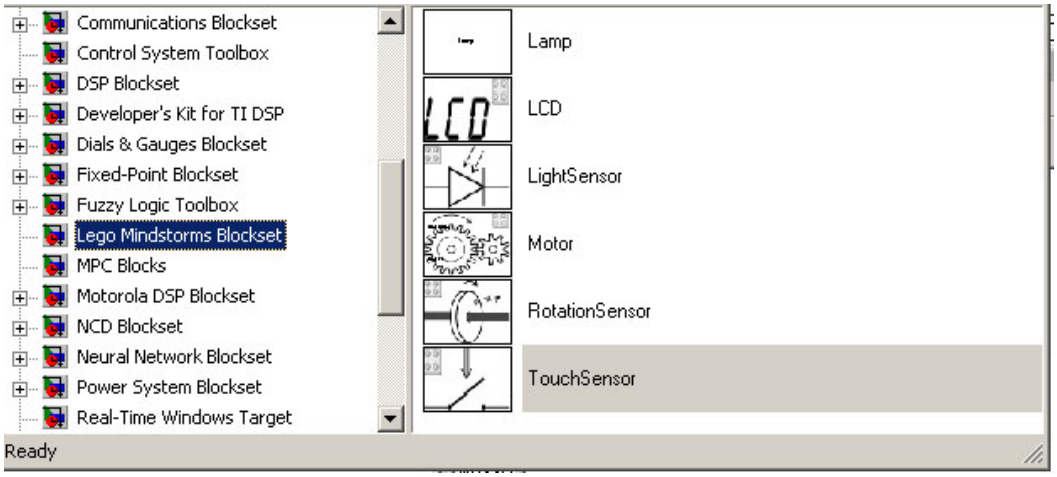


**Abb. 5.2/13 Legoblöcke**

# 6. Praktische Erfahrungen

## 6.1. Installation des ECRobot und der benötigten Komponenten

Theoretisch sollte die Einrichtung des ECRobot und der Zusatzsoftware nicht sonderlich schwer sein. In der von der MathWorks Webseite downloadbaren Version ist ein Pdf mit Schritt für Schritt Anleitung hinterlegt sowie die ECRobot Files. Die anderen Komponenten wie LegOS, Cygnus eca. Müssen jedoch unabhängig geladen werden, Dies hat rechtliche Ursachen. Im neuen Matlab 6.5 R13 sollte es sogar noch einfacher sein da hier das ECRobot bereits integriert ist und ein automatisches Installationsscript die anderen Komponenten automatisch einspielen sollte. Leider alles Puste Kuchen, weder das Installationsscript noch die Anleitung brachten den gewünschten Erfolg. Vermutlich hat dies etwas mit den verschiedenen Entwicklungsversionen der vielen unabhängigen beteiligten Komponenten zu tun. Glücklicherweise stellte ein Entwickler von Mathworks, freundlicherweise auf Anfrage, eine funktionsfähige Komplettversion zur Verfügung wie sie dort vor Ort benutzt wurde.
Da das einspielen des LegOS aus Mathlab heraus nicht funktioniert ist es noch notwendig das LegOS selbst zu installieren. Dazu ist die Anleitung auf der Webseite http://legos.sourceforge.net/cygwin/INSTALL-cygwin.html sehr gut geeignet. Mann sollte jedoch statt dem legos-0.2.6 das 0.2.5.2 aus dem Entwickler-Pack benutzen. Dieses in den Cygwin-Pfad kopieren und ohne die Make-Schritte gleich übertragen. In Matlab muss dann nur noch der Suchpfad mit den Pfaden des Entwicklerpakets erweitert werden. Außerdem funktioniert das ganze auch nur unter Matlab 6.1 R12. Noch einige Hinweise:

- Es hat sich gezeigt das beim auftreten von Make-Fehlern ein Neustarten von Matlab helfen kann.
- Pfade die im Suchpfad stehen können nicht gelöscht werden (zumindest unter WinXP).
- Das Modell Robot3 ist nicht funktionsfähig.
- Achtung! Das benutzen des legos-0.2.6 führt zu einfrieren des RCX beim nutzen aufgespielter Programme, die mit dem ECRobot erzeugt wurden. Es hilft dann nur noch das herausnehmen der Batterien.

## 6.2. Ein eigenes Modell erzeugen

Es hat sich herausgestellt das es eine Menge Fallstricke gibt die beim Erstellen eines eigenen Modells zu beachten sind. Deshalb ist es Empfehlenswert sich zuerst einmal an der Reimplementation eines der mitgelieferten Beispielprojekte zu versuchen.
Die Erfahrungen damit sind im folgenden in kurzen Stichpunkte niedergelegt:

- Neues Modellfile anlegen
- View->Show Library Browser um nötige Blöcke einzufügen
- Um den Stateflow einzusetzen den Block chart aus der Library kopieren
- Nun baut man alles so zusammen wie im Vorbild
- im nächsten schritt werden die Events, Variablen und Targets eingetragen Tools->Explore
- zuletzt müssen die simulations parameter korrekt gesetzt werden
    - fixed step solver und fixed size = 1
    - zero crossing detection off
    - inline parameter
    - system target file = rcx
    - stateflow options

Es gibt allerdings noch wichtige Unterschiede die nicht so leicht zu finden sind. Deshalb kann das Tool Merge noch etliche Unterschiede aufspüren und beseitigen. Da dies natürlich keine Lösung ist die bei einem neuen Projekt weiterhilft ist es günstig das eigene Modell mit vielen verschiedenen Zwischenständen zu speichern um sich so an die Fehlerhaften unterschiede heranzutasten. Dummerweise kam diese Idee bei mir zu Spät und zu viele Parameter auf einmal wurden verändert um sagen zu können welche der Veränderung letztendlich ausschlaggebend waren für die Funktionalität. Vermutet sind die Probleme allerdings im Stateflow Teil.

# 7. Schlussbemerkungen

Die Aufgabe einer Programmierumgebung auf hohem Abstraktionsniveau erfüllt das ECRobot sicherlich ganz gut. Wie aber die praktische Erfahrung gezeigt hat stellt gerade die sehr umfangreiche Funktionsvielfalt der Matlab Umgebung und die Kombination mit vielen anderen Hilfsprogrammen ein nicht unerhebliches Entwicklungshemmnis dar.
So ist zwar das ECRobot, für den Anwender der sich auf bestehende Blöcke beschränkt, eine nützliches Werkzeug um einen Lego Roboter aus der Sicht der Automatisierung zu programmieren. Für die Weiterentwicklung in bisher nicht abgedeckte Bereiche der RCX-Funktionalität sehe ich allerdings schwarz. Diese Vermutung stützt sich darauf, dass wie sich gezeigt hat die Verwendung vieler unabhängiger Tools zu erheblichen Kombatibilitätsproblemen führt und außerdem in dem Verlauf eines Jahres in dem ich die Entwicklung des ECRobot beobachte kaum Weiterentwicklungen gezeigt haben. Daraus Schlussfolgere ich das bei MathWorks nur eine Person an diesem Programm arbeitet. Eine Webrecherche zeigte außerdem auf, dass es außerhalb von MathWorks bisher keine weiteren Erwähnungen des ECRobot gibt und somit keine Weiterentwicklung des ECRobot durch eine größere „Fangemeinde“, wie das bei anderen Tools zum RCX programmieren der Fall ist, gibt. Dies Führe ich darauf zurück das es einerseits ein erheblicher Einarbeitsaufwand notwendig ist, der keinesfalls im Verhältnis zum erzielbaren Nutzen steht. Umschreiben möchte es mit dem Spruch „Mit Kanonen auf Spatzen schießen“. Ebenso dürfte dem Erfolg des ECRobot aber im Wege stehen das die Basis, also Matlab, sehr wohl kostenpflichtig ist und das in ganz erheblichem Maße und so sicher keinen Anreiz für eine Freeware Entwicklergemeinde hat.

## 7.1. Vergleich mit Robotics Invention V2.0

Letztendlich sollte man die ECRobot Variante noch mit der von Lego mitgelieferten Entwicklungsumgebung Vergleichen.
Fakt ist, dass das Robotic Invention derzeit eine wesentliche breitere Unterstützung der Funktionalität des RCX zur Verfügung stellt. Gebündelt in einer Entwicklungsumgebung die dafür ausgelegt ist auch Kindern das Programmieren des RCX zu ermöglichen. Also eine sehr intuitive Arbeit ermöglicht. Allerdings wird hier per Drag & Drop eine Art Ablaufplan aufgebaut während man im Simulink ja eine Modellbeschreibung einsetzt. Für ECRobot spricht, dass es jedem, der sich in das komplexe System einarbeitet, möglich ist die volle Funktionalität des RCX auszuschöpfen indem er passende S-Funktionen schreibt. Eine entsprechend motivierte Entwicklergemeinde währe also sehr wohl in der Lage in kurzer Zeit ein breites Funktionsangebot, auf Blockebene, zu erstellen. Allerdings hat sich wie bereits gesagt eine solche bisher nicht entwickelt.
Unterm Strich ist ECRobot also eine Methode um jemanden Modellierungskonzepte näher zu bringen. Vom Aufwand her ist es auf der Ebene der abstrakten Programmierung sicherlich nicht mit der von Lego mitgelieferten Software Konkurrenzfähig. Es bleibt also dabei, wer individuelle Ideen außerhalb der unterstützten Bereiche verwirklichen will muss bei den, inzwischen in großer Zahl vorhanden, Programmiersprachen bleiben. Als Beispiel wäre hier NQC zu erwähnen.

# 8. Abbildungsverzeichnis



# 9. Literaturverzeichnis

Die Inhalte dieses Papers sind größtenteils Übersetzungen aus den Hilfen von Matlab 6.1 R12 und 6.5 R13, sowie aus dem ecrobot.pdf.

# 10. Inhaltsverzeichnis des Anhangs

- Robolab_Demo_20.zip (Webseite: http://www.ni.com/company/robolab.htm)
  Die Demo zum kostenpflichtigen Robolab.
- rcx_0.7.zip
  Enthält das Entwicklerpaket von ECRobot bei dem bereits alles integriert ist.
- ecrobot.pdf
  Enthält die Beschreibung zu ECRobot.
- cygwin.zip
  Enthält die zum installieren benötigten Dateien von Cygwin.
- legOS-0.2.5.tar.gz
  Die LegOS Version sollte mit Cygwin nach der Anleitung auf der Webseite (http://legos.sourceforge.net/cygwin/INSTALL-cygwin.html) installiert werden.