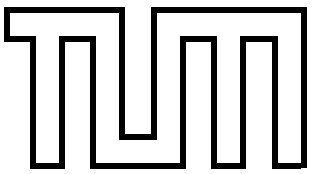

Technische Universität München

Fakultät für Informatik
Forschungs- und Lehreinheit Informatik VI

Übung zur Vorlesung Echtzeitsysteme

Aufgabe 7 – Modellbasierte Entwicklung

Simon Barner
barner@in.tum.de

Irina Gaponova
gaponova@in.tum.de

Stephan Sommer
sommerst@in.tum.de

Wintersemester 2009/10

Aufgabe 7: Modellbasierte Entwicklung mit EasyLab

Einleitung

Die Programmierung von (eingebetteten) Echtzeitsystemen ist aufwendig, fehleranfällig und erfordert viel spezifisches Detailwissen über die jeweilige Zielplattform.

Modellbasierte Entwicklungswerkzeuge bieten die Möglichkeit, die Programmierung solcher Systeme zu vereinfachen, in dem sie das Abstraktionsniveau anheben und somit den Entwicklungsprozess beschleunigen. Eine möglicher Ansatz besteht daran, dass zur Entwicklung von Anwendungen Komponenten ausgewählt und in einer graphischen Benutzeroberfläche (GUI) parametrisiert und verschaltet werden. Mit formalen Analysen des Models oder Simulationen können Entwickler ihren Ansatz bereits in einer sehr frühen Designphase überprüfen. Mit Hilfe von Codegeneratoren kann dann effizienter und robuster Code erzeugt werden, der die spezifizierte Funktionalität implementiert. Desweiteren kann ein solches Werkzeug beim Ausrollen der Software (Deployment) und der Fehlersuche (Debugging) unterstützen.

EasyLab

EasyLab ist ein modellbasiertes Entwicklungswerkzeug, das im Rahmen des Forschungsprojektes *EasyKit*[1] am *Lehrstuhl für eingebettete Systeme und Robotik* entwickelt wird. Sein Schwerpunkt liegt auf der Entwicklung von Steuerungen von mechatronischen Anwendungen.

Eine wichtige Bitte: EasyLab befindet sich derzeit noch in der Entwicklung. Bitte nehmen Sie uns zum einen den einen oder anderen Fehler oder gar Crash nicht übel.

Bitte teilen Sie uns vor allem auch mit, wie der Fehler reproduziert werden kann (hierzu stehen entsprechende Formulare zur Verfügung). Vielen Dank für Ihre Mithilfe!

Graphische Modellierungssprachen

EasyLab unterstützt derzeit zwei grafische Modellierungssprachen:

1. *Ablaufsprache*: Die Ablaufsprache (engl. *sequential flow chart, SFC*) beschreibt die Zustände und Zustandsübergänge eines Programms. Zustände von *SFC*-Programmen referenzieren Unterprogramme, die in unserem Fall Datenflussprogramme (siehe unten) sind. Abbildung 1 zeigt ein *SFC*-Programm, das einen Regler für ein selbstaufrichtendes invertiertes Pendel implementiert. Das Programm enthält einen (eindeuti-

[1] http://www.easy-kit.de/

gen) Anfangszustand, der durch den doppelten gezeichneten Rahmen gekennzeichnet ist. Der aktuelle Zustand wird verlassen, sobald die nachfolgende Transitionsbedingung erfüllt ist, die nach dem Zustand ausgewertet wird (der Zustand wird also mindestens einmal ausgeführt). Im Beispiel wird der Aufschwung-Zustand (`swing-up`) verlassen und der Balancier-Zustand (`balance`) betreten, sobald der Winkel α_i kleiner als eine gewisse Toleranz ist, d.h. falls der Stab sich schon fast in der vertikalen Position befindet.

2. *Synchroner Datenfluss*: Die Synchrone Datenflusssprache (*SDF*) besteht auf einem gerichteten Multigraphen, dessen Knoten als Aktoren eines bestimmten Aktortypen bezeichnet werden. Ein Aktortyp wird durch die Menge seiner typisierten Eingänge, Ausgänge und internen Zustandvariablen beschrieben. Die Kanten des Graphen bezeichnen den Datenfluss zwischen den Aktoren. Üblicherweise werden Datenflussgraphen dazu verwendet, Daten zu verarbeiten und Reglungsaufgaben durchzuführen.

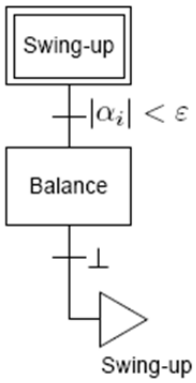

Abbildung 1: *SFC*-Programm für invertiertes Pendel

Ausführung von Programme

EasyLab unterstützt zwei Ausführungsmodi: Simulation und Codegenerierung. Im Simulations-Modus werden die Aktoren „virtuell“ auf dem Hostrechner ausgeführt, was zur Überprüfung eines Designs verwendet werden kann. Im Codegenerierungs-Modus wird automatisch *C*-Code für die jeweilige Zielplattform generiert, der auf Codevorlagen (Templates) basiert, die für jeden Aktortypen hinterlegt sind. Der erzeugte Code kann direkt übersetzt und auf die Zielhardware übertragen werden.

Um den Designer mit einer komfortablen Debugging-Möglichkeit zu unterstützen, bietet EasyLab die Möglichkeit zum *modellbasierten Debugging*, bei den die aktuellen Werte aller Ein- und Ausgänge sowie aller Variablen „live“ auf dem Host-Rechner angezeigt werden. Darüber hinaus besteht die Möglichkeit, Parameter zu modifizieren, während das Programm auf der Zielhardware läuft, so dass die Auswirkung der Änderung sofort beachtet werden kann.

Easykit Starter-Board und Match-X-Bausteine

EasyKit Starter wurde im Rahmen von *EasyKit* für den Einsatz in der Lehre entwickelt. Es basiert auf einem 32-Bit *ARM Cortex-M3* (*STM32F103* der Firma *ST microelectronics*). Da es eine der grundlegenden Ideen von *EasyKit* ist, auch die Hardware-Entwicklung zu beschleunigen, wird auf ein Baukastensystem zurückgegriffen, mit dem schnell Prototypen einer Steuerungen aufgebaut werden kann. Auf dem vorliegendne Board stehen nun zwei Sockel zur Verfügung, auf die *Match-X-Bausteine* aufgesteckt werden kann, die als Beschaltung für die E/A-Pins des Mikrocontrollers dienen.

Im vorliegendne Aufgabenblatt werden wir uns mit den beiden folgenden Bausteinen befassen:

- AIO1.1: Dieser Baustein schaltet den Analogeingang 1 sowie den Analogausgang 1 des jeweiligen Stapels frei.
- DIO1.1: Dieser Baustein schaltet den Digitaleingang 1 sowie den Digitalausgang 1 des jeweiligen Stapels frei.

Mehrere Blöcke unterschiedlichen Typs können aufeinander gesteckt werden, so dass sich *Stapel* ergeben.

Anwendungsentwicklung mit EasyLab

Einführungsaufgaben

- Starten Sie EasyLab und verbinden Sie das USB-Kabel mit dem EasyKit Starter. Die Software sollte nun das Board erkennen und eine zusätzliche Funktionsbibliothek anbieten, die dessen Funktionalität beschreibt. Laden Sie nun zunächst das Beispielprojekt `BottonLED.easy` aus dem Verzeichnis `C:\Program Files\EasyLab\examples` und vollziehen Sie das einfache *SFC*- und *SDF*-Programm nach.
- Wechseln Sie nun in den Codegenerierungsmodus, aktivieren Sie die Debugging-Unterstützung und laden Sie das Projekt auf die Zielhardware. Sobald die Debugging-Verbindung steht, können Sie den aktuellen Zustand des Programmes, das auf dem Starter Board abläuft, in EasyLab betrachten.
- Machen Sie sich weiter mit der Anwendung vertraut und implementieren Sie ein paar einfache Anwendungen, z.B. zum Ein- und Auschalten von LEDs. Probieren Sie auch den Simulationsmodus aus.

Geschwindigkeitsregelung eines Motors

Das Ziel dieser Aufgabe ist es, die aus der Vorlesung bekannte Geschwindigkeitsregelung für einen Elektromotor nachzuvollziehen.

- Vorbereitungen:
  - Verbinden Sie die Masse- und Versorgungsleitung des EasyKit Starter mit der Motorplatine.
  - Der Motor wird über ein analoges Signal angesteuert. Stecken Sie hierzu einen AOI-Block auf einen der Stabel und verbinden Sie den entsprechenden Ausgang mit den *Motor Analog/PWM*-Port. Hierbei wird das analoge Ausgangssignal über einen PWM-Generator erzeugt, für den das entsprechende Tastverhältnis (*duty cycle*) angegeben werden muss.
  - Die Drehgeschwindigkeit des Motors kann über die *Input Capture*-Einheit des Mikrocontrollers bestimmt werden. Hierbei wird das Signal ausgewertet, das die Lichtschranke beim Durchlauf der schwarz-weiss gefärbten Scheibe erzeugt und das dann an einem digitalen Eingang des Controllers anliegt (DIO-Block). Zur Auswertung steht der *Digital Edge Detection* Funktionsblock zur Verfügung, der die Zeit zwischen zwei aufeinder folgenden steigenden Flanken ausgibt (in μs).
- Erstellen Sie nun ein Programm, das den Motor mit einem fest vorgegebenen Tastverhältnis antreibt und das die Drehgeschwindigkeit (Umdrehungen / min) mißt.
- Entwickeln Sie nun ein Programm, das den Motor mit einer vorgegebenen Geschwindigkeit antreibt. Verwenden Sie hierzu einen PID-Regler aus der Funktionsblockbibliothek. Wählen Sie hierzu geeignete Anfangskoeffizienten (z.B. $c_p = 0.1$, $c_i = 0$ und $c_d = 0$) und führen Sie eine experimentelle Auslegung des Reglers mit Hilfe der Debugging-Funktionalität durch. Beachten Sie hierbei die Spezifikation der Eingangsgröße des PID-Regelers (Online-Hilfe!).

Treppenhaus-Beleuchtung

Das Ziel des Aufgabe ist es, eine Steuerung für die Beleuchtung des Treppenhauses eines dreistöckigen Hauses zu entwerfen, wobei sich in jedem der Stockwerke ein Lichtschalter befindet. Falls das Licht nicht brennt, soll ein Druck auf einen der Schalter das Licht einschalten. Umgekehrt soll ein Druck auf einen der Schalter das Licht ausschalten, falls es bereits an ist. Dabei soll das Licht nur eingeschaltet werden falls es dunkel genug ist.

Verwenden Sie in dieser Aufgabe die Taster des EasyKit Starter's als Schalter, eine LED als Treppenhaus-Licht und den Phototransistor auf der Motorplatine (`T1`), um die Helligkeit zu messen. Der Phototransistor setzt dabei die Helligkeit in eine Spannung von $0\,V$ bis $5\,V$ um (höhere Spannungen bedeuten eine höhere Helligkeit).

- Verbinden Sie den `Verstärker OUT`-Port der Motorplatine mit einem analogen Eingang des EasyKit Starter und setzen Sie den Jumper `JP2` auf die Position 2-3 (Position 1-2 wählt die Lichtschranke aus).
- Entwicklen Sie eine Anwendung, die das Licht einschaltet, sobald der Phototransistor vom Licht (z.B. Fenster) abgewandt wird.
- Erweitern Sie nun Ihr Programm um die oben beschriebenen Lichtsteuerung.

Zustandsbasiertes Programmieren

In den vorangegangenen Aufgaben wurde immer nur ein einzelner Programmzustand verwendet. In realen Anwendungen werden allerdings üblicherweise mehrere Zustände (Ausführungsmodi) verwendet. In diesem Fall kann in EasyLab die *SFC*-Sprache dazu verwendet werden, Zustände und *Transitionsbedingungen* zwischen diesen zu modellieren. Hierbei können derzeit Zustandssequenzen, Alternativ-Verzweigungen und Sprünge verwendet werden – die Implementierung paralleler Verzweigungen ist noch nicht abgeschlossen.

Transitionsbedinungen sind dabei Boolesche Terme sowie Vergleiche von arithmetischen Termen – jeweils über Konstanten und (globalen) Variablen, die die Verbindung zwischen den Werten der *SDF*-Diagramme und der *SFC*-Sprache darstellen. Die Syntax ist dabei an die Programmiersprache *C* angelehnt.

- Boolesche Verknüpfungen/Konstanten: `&&`, `||`, `!`, `true`, `false`
- Arithmetische Verknüpfungen/Konstanten: `+`, `-`, `*`, `/`, `0, 1, ...`
- Vergleichsoperatoren: `<`, `<=`, , `>`, `>=`, `==`, `!=`
- Beispiel: `(b || x < 2) && (y > 3)` (`b` Boolesche Variable, `x, y` arith. Variablen)
- Globale Variablen können über den Variablenmanager des *SFC*-Programmes angelegt und verwaltet werden. In *SDF*-Programmen steht für den Zugriff auf Variablen jeweils ein Leser und Schreiber zur Verfügung – desweiteren können per Rechts-Klick auf das Variablengerät Variablen hinzugefügt, entfernt sowie umbenannt werden.

Erweitern Sie nun die Motorregelung aus der obigen Aufgabe, so dass verschiedene Sollgeschwindigkeiten vorgegeben werden können.

- Verwenden Sie das *SDF*-Programm mit der Motorregelung wieder – die aktuell gewünschte Geschwindigkeit soll aus einer Variable ausgelesen werden.
- Verwenden Sie die Taster, um zwischen den verschiedenen Geschwindigkeitsstufen (z.B. 3000, 3500 und 4000) weiterzuschalten, d.h. die Variable entsprechend zu setzen. Danach soll dann jeweils das Regler-Unterprogramm aufgerufen werden, so dass die Änderung der Geschwindigkeit vollzogen werden kann.

Fehlerberichte

Auf diesem Blatt können Fehlerberichte gesammelt werden – es wird am Ende der Übung eingesammelt. Bitte gestalten Sie Ihre Berichte so spezifisch wie möglich!

- Fehler 1
  - Welche Auswirkung hat der Fehler (Crash, Fehlverhalten, ...)?
  - Wie kann der Fehler reproduziert werden?. Es besteht die Möglichkeit, EasyLab-Projeke im Ordner `Q:\Blatt6\Fehler` abzulegen. Um eindeutige Dateinahmen zu erhalten, verwenden Sie bitte für die Dateinahmen das Präfix `ez_<x>` (wobei `<x>` Ihre Gruppennummer ist) und geben Sie den Dateinamen hier an.
- Fehler 2
  - Welche Auswirkung hat der Fehler (Crash, Fehlverhalten, ...)?
  - Wie kann der Fehler reproduziert werden?. Es besteht die Möglichkeit, EasyLab-Projeke im Ordner `Q:\Uebung06\Fehler` abzulegen. Um eindeutige Dateinahmen zu erhalten, verwenden Sie bitte für die Dateinahmen das Präfix `ez_<x>` (wobei `<x>` Ihre Gruppennummer ist) und geben Sie den Dateinamen hier an.

- Fehler 3
  - Welche Auswirkung hat der Fehler (Crash, Fehlverhalten, ...)?

  - Wie kann der Fehler reproduziert werden?. Es besteht die Möglichkeit, EasyLab-Projeke im Ordner `Q:\Uebung06\Fehler` abzulegen. Um eindeutige Dateinahmen zu erhalten, verwenden Sie bitte für die Dateinahmen das Präfix `ez_<x>` (wobei `<x>` Ihre Gruppennummer ist) und geben Sie den Dateinamen hier an.

- Fehler 4
  - Welche Auswirkung hat der Fehler (Crash, Fehlverhalten, ...)?

  - Wie kann der Fehler reproduziert werden?. Es besteht die Möglichkeit, EasyLab-Projeke im Ordner `Q:\Uebung06\Fehler` abzulegen. Um eindeutige Dateinahmen zu erhalten, verwenden Sie bitte für die Dateinahmen das Präfix `ez_<x>` (wobei `<x>` Ihre Gruppennummer ist) und geben Sie den Dateinamen hier an.