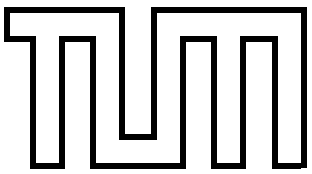

Technische Universität München

Fakultät für Informatik
Forschungs- und Lehreinheit Informatik VI

Übung zur Vorlesung Echtzeitsysteme

Aufgabe 6 – Modellbasierte Entwicklung

Nadine Keddis
keddis@fortiss.org

Dominik Sojer
sojer@in.tum.de

Stephan Sommer
sommerst@in.tum.de

Michael Geisinger
geisinge@in.tum.de

Wintersemester 2011/12

Aufgabe 6: 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, indem sie das Abstraktionsniveau anheben und somit den Entwicklungsprozess beschleunigen. Eine möglicher Ansatz besteht darin, dass zur Entwicklung von Anwendungen Komponenten ausgewählt und in einer grafischen Benutzeroberfläche (*GUI*) parametrisiert und verschaltet werden. Mit formalen Analysen des Modells oder Simulationen können Entwickler ihren Ansatz bereits in einer sehr frühen Designphase überprüfen. Mit Hilfe von Codegeneratoren kann dann Code erzeugt werden, der die spezifizierte Funktionalität implementiert. Des Weiteren kann ein solches Werkzeug beim Ausrollen der Software (*Deployment*) und der Fehlersuche (*Debugging*) unterstützen.

EasyLab

EasyLab ist ein modellbasiertes Entwicklungswerkzeug, das im Rahmen des Forschungsprojekts *EasyKit*[1] am *Lehrstuhl für eingebettete Systeme und Robotik* entwickelt wurde. Der Schwerpunkt liegt dabei auf der Entwicklung von Steuerungssystemen für *mechatronische Anwendungen*, d.h. für Anwendungen bei denen Elektronik, Mechanik und Software zur Lösung der Problemstellung geeignet zusammenwirken müssen.

Bitte beachten Sie: Die Entwicklung von EasyLab ist noch nicht abgeschlossen. Bitte nehmen Sie uns den einen oder anderen Fehler oder gar Absturz nicht übel. Falls Sie einen Fehler finden sollten, teilen Sie uns bitte mit, wie dieser reproduziert werden kann. 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.

   Abbildung 1 zeigt ein *SFC*-Programm, das einen Regler für ein sich selbst aufrichtendes invertiertes Pendel implementiert. Das Programm enthält einen (eindeutigen) Anfangszustand, der durch den doppelten gezeichneten Rahmen gekennzeichnet ist. Der aktuelle Zustand wird verlassen, sobald die nachfolgende Transitionsbedingung

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

erfüllt ist, die jeder Ausführung des dem Zustand zugeordneten Programms ausgewertet wird (das Programm 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 ε ist, d.h. falls der Stab sich schon fast in der vertikalen Position befindet. Der `balance`-Zustand wird nie verlassen, was durch die Bedingung $\bot$ angedeutet wird. In EasyLab werden Transitionsbedingungen als Boolesche Ausdrücke behandelt, d.h. $\bot$ entspricht hier dem konstanten Booleschen Ausdruck `false`. Der Pfeil deutet einen Sprung in einen anderen Zustand an.

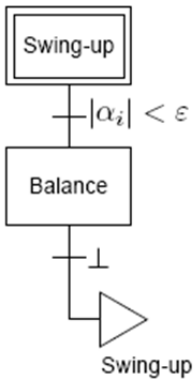

Abbildung 1: *SFC*-Programm für das invertierte Pendel

Zusätzlich muss jedem Zustand natürlich ein Programm zugeordnet werden, das angibt, welche Befehle in dem jeweiligen Zustand abgearbeitet werden sollen. Dazu referenziert jeder Zustand eines *SFC*-Programms ein *Unterprogramm*. In EasyLab werden die Unterprogramme durch die synchrone Datenflusssprache (siehe unten) beschrieben.

2. *Synchroner Datenfluss*: Die Synchrone Datenflusssprache (engl. *synchronous data flow, SDF*) definiert formal einen gerichteten Multigraphen (ein Graph, in dem zwei Knoten auch durch mehrere Kanten verbunden sein können).

   Die Knoten des Graphen werden als Aktoren bezeichnet und sind jeweils Instanzen eines bestimmten Aktortyps. Ein Aktortyp wird durch die Menge seiner typisierten Eingänge, Ausgänge und internen Zustandsvariablen beschrieben, sowie durch eine ihm zugeordnete Berechnungsvorschrift, die die Berechnung des Aktors in Abhängigkeit der Werte an den Eingängen darstellt. In EasyLab liegt eine große Zahl an vorgefertigten Aktortypen in der so genannten *Funktionsblockbibliothek* vor.

   Die Kanten des Graphen bezeichnen den Datenfluss zwischen den Aktoren und verbinden immer Ausgänge von Aktoren mit Eingängen. An einen Ausgang eines Aktors kann eine beliebige Zahl an Kanten angeschlossen werden. An einen Eingang eines

Aktors kann maximal eine Kante angeschlossen werden. Die Semantik ist dabei die Folgende: Jedes Mal, wenn am Ausgang eines Aktors ein Wert ausgegeben wird, wird dieser an alle Eingänge kopiert, die durch Kanten mit dem entsprechenden Ausgang verbunden sind. Ist an einem Eingang keine Kante angeschlossen, so wird stattdessen ein festgelegter Vorgabewert verwendet.

Im Gegensatz zur Ablaufsprache, bei der jeweils nur ein Zustand aktiv ist, werden alle Aktoren in einem Datenflussgraphen in jedem Programmschritt ausgeführt. Die Reihenfolge der Ausführung ist dadurch definiert, welche Aktoren „bereit zur Ausführung“ sind: Zunächst werden die Aktoren ausgeführt, die keine Abhängigkeiten gegenüber anderen Aktoren haben. Das sind solche ohne Eingänge bzw. mit Eingängen, an denen keine Kanten angeschlossen sind (für Letztere ist ja ein Vorgabewert definiert). Dadurch werden weitere Aktoren „bereit zur Ausführung“, bis alle Aktoren ausgeführt wurden. Kreise im Graphen sind ebenfalls erlaubt, hierzu muss für jeden Kreis angegeben werden, welcher Aktor zuerst ausgeführt werden soll (dieser Aktor wird dadurch sozusagen künstlich „bereit zur Ausführung“ gemacht). Aufgrund ihrer synchronen Semantik werden Datenflussgraphen üblicherweise dazu verwendet, Signale zu verarbeiten und Regelungsaufgaben durchzuführen.

Ausführung von Programmen

EasyLab unterstützt zwei Ausführungsmodi: *Simulation* und *Codegenerierung*.

- Im Simulations-Modus werden die Aktoren „virtuell“ auf dem Hostrechner ausgeführt, was zur Überprüfung des Designs der Anwendung 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 automatisch übersetzt und auf die Zielhardware übertragen werden.

Um den Programmierer mit einer komfortablen Möglichkeit zu unterstützen, die Funktionalität der Anwendung auf der realen Hardware zu überprüfen, 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 Auswirkungen der Änderungen sofort sichtbar werden.

Easykit Starter

Das Produkt *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 Hardware-Baukastensystem zurückgegriffen, mit dem schnell Prototypen einer Steuerungen aufgebaut werden können. Auf dem vorliegenden Basisboard stehen zwei Sockel zur Verfügung, auf die die kleinen schwarzen Bausteine des Baukastensystems (vgl. Abbildung 2) aufgesteckt werden können. Die Bausteine ordnen dabei E/A-Pins des Controllers den Ein- und Ausgängen auf den Klemmleisten zu und fungieren zusätzlich als Schutzbeschaltung.

Abbildung 2: Baustein aus dem EasyKit-Baukastensystem

Bitte beachten Sie: Die vorliegenden Platinen und Bausteine von EasyKit Starter sind teilweise Prototypen und in der Funktionalität und Ausführung nicht vollständig identisch.

Im vorliegenden Aufgabenblatt werden wir uns mit den beiden folgenden Hardware-Bausteinen befassen:

- `AIO1.1` bzw. `AIO1 V2`: Dieser Baustein schaltet den Analogeingang Nr. 1 (`A-In 1`) sowie den Analogausgang Nr. 1 (`A-Out 1`) des jeweiligen Sockels frei.
- `DIO1.1` bzw. `DIO1 V2`: Dieser Baustein schaltet den Digitaleingang Nr. 1 (`D-In 1`) sowie den Digitalausgang Nr. 1 (`D-Out 1`) des jeweiligen Sockels frei.

Mehrere Blöcke unterschiedlichen Typs können aufeinandergesteckt werden, so dass sich *Stapel* ergeben. Werden Bausteine auf das Board gesteckt, so erkennt EasyLab diese von selbst und zeigt die dadurch freigeschaltete Funktionalität in der Funktionsblockbibliothek an.

Falls Sie mehr über EasyKit, EasyLab und EasyKit Starter erfahren wollen, besuchen Sie bitte die Webseite `http://www.easy-kit.de/` oder setzen Sie sich mit der Übungsleitung in Verbindung. EasyLab Starter kann von der angegebenen Webseite kostenlos heruntergeladen werden.

Anwendungsentwicklung mit EasyLab

Einführungsaufgaben

a) Starten Sie EasyLab und verbinden Sie das USB-Kabel mit dem EasyKit Starter. Die Software sollte nach kurzer Zeit das Board erkennen und einen zusätzlichen Eintrag in der Funktionsbibliothek (rechte Seite) anbieten, der dessen Funktionalität beschreibt.

Laden Sie nun zunächst das Beispielprojekt `ButtonLED.easy` aus dem Verzeichnis `Q:\Uebung05` und vollziehen Sie das einfache *SFC*- und *SDF*-Programm nach.

b) Wechseln Sie in den Simulationsmodus und starten Sie die simulierte Programmausführung durch Klicken auf den grünen Pfeil in der Symbolleiste. Die Funktionsblöcke auf der linken Seite des Datenflussprogramms entsprechen den Tastern `TA1` bis `TA3` auf dem Basisboard. *Simulieren* Sie Tastendrücke durch Klicken auf die entsprechenden Schaltflächen und beobachten Sie die Veränderung. Klicken Sie zum Stoppen der Simulation auf das rote Rechteck in der Symbolleiste.

c) Wechseln Sie nun in den Codegenerierungs-Modus und starten Sie die Programmausführung auf der Zielhardware durch Klicken auf den grünen Pfeil in der Symbolleiste. Bitte warten Sie, bis der Vorgang abgeschlossen ist (ca. 15 Sekunden). Sobald die Debugging-Verbindung steht, können Sie den aktuellen Zustand des Programms, das auf dem EasyKit Starter Board abläuft, in EasyLab betrachten. Drücken Sie zum Test die Taster `TA1` bis `TA3` (*nicht* jedoch `RESET`) auf dem Board und achten Sie auf die Veränderung im Datenflussprogramm in EasyLab. Klicken Sie zum Stoppen der Programmausführung auf das rote Rechteck in der Symbolleiste.

d) Machen Sie sich weiter mit der Anwendung vertraut und implementieren Sie ein paar einfache Anwendungen, z.B. zum Ein- und Ausschalten von LEDs. Durchsuchen Sie dazu die Funktionsblockbibliothek bzw. die Online-Hilfe.

Geschwindigkeitsregelung eines Motors

Das Ziel dieser Aufgabe ist es, die aus der Vorlesung bekannte Geschwindigkeitsregelung für einen Elektromotor nachzuvollziehen. Als zusätzliche Referenz können Sie im Folgenden die Webseite `http://www.easy-kit.de/EkAnwendungEasyKitStarter` verwenden.

a) Vorbereitungen:

- Verbinden Sie die Masse- und Versorgungsleitung des EasyKit Starter mit der Motorplatine (`GND_D` mit `GND`, `5V_D` mit `+5V`).
- Der Motor wird über ein analoges Signal angesteuert. Stecken Sie hierzu einen AIO-Block auf einen der Sockel 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*) zwischen 0 und 100 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/transparent gefärbten Scheibe erzeugt und das dann an einem digitalen Eingang des Controllers anliegt (DIO-Block). Setzen Sie dabei den Jumper `JP2` auf dem Motorboard auf die Position 1-2. Zur Auswertung steht der Funktionsblock *Digitale Flankenerkennung* zur Verfügung, der die Zeit in μs (millionstel Sekunden) zwischen zwei aufeinander folgenden steigenden Flanken ausgibt.

b) Erstellen Sie nun ein Programm, das den Motor mit einem fest vorgegebenen Tastverhältnis antreibt und die Drehgeschwindigkeit (in Umdrehungen/min) misst. Verwenden Sie zur Umrechnung geeignete Funktionsblöcke aus der Sektion „Mathematik“ der Funktionsblockbibliothek. Beachten Sie, dass manche Encoderscheiben an den Motoren eine andere Anzahl an schwarz/transparent-Durchgängen haben als andere.

c) Entwickeln Sie nun ein Programm, das den Motor mit einer vorgegebenen Geschwindigkeit antreibt. Verwenden Sie hierzu einen PID-Regler aus der Funktionsblockbibliothek. Wählen Sie geeignete Anfangskoeffizienten (z.B. $c_p = 0{,}05$, $c_i = 0$ und $c_d = 0$) und führen Sie eine experimentelle Auslegung des Reglers mit Hilfe der Debugging-Funktionalität durch.

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 eingeschaltet ist. Dabei soll das Licht nur eingeschaltet werden, falls es „dunkel genug“ ist.

Verwenden Sie in dieser Aufgabe alle drei Taster des Basisboards als Schalter (für die drei Stockwerke), die drei LEDs symbolisch als Beleuchtung des jeweiligen Stockwerks und den Fototransistor auf der Motorplatine (`T1` bzw. `E1`), um die Helligkeit zu messen. Der Fototransistor setzt dabei die Helligkeit in eine Spannung von ca. 1,5 V bis 4,5 V um (höhere Spannungen bedeuten eine höhere Helligkeit).

a) 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 dagegen die Lichtschranke aus).

b) Entwickeln Sie eine Anwendung, die das Licht einschaltet, sobald der Fototransistor vom Licht (z.B. Fenster) abgewandt wird.

c) Erweitern Sie nun Ihr Programm um die oben beschriebenen Lichtsteuerung. Verwenden Sie jedoch nicht die Ablaufsprache zur Repräsentation des Programmzustands, sondern verwenden Sie geeignete Funktionsblöcke aus der Sektion „Signalverarbeitung“ → „Flip-Flops“ (siehe auch EasyLab Online-Hilfe).

Zustandsbasiertes Programmieren

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

Transitionsbedingungen 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` arithm. Variablen)
- Globale Variablen können über den Variablenmanager des *SFC*-Programms angelegt und verwaltet werden. Klicken Sie auf „Hinzufügen“ am unteren Rand des Fensters, um eine neue Variable anzulegen und parametrieren Sie diese dann in der Liste. Danach erscheinen in der Funktionsblockbibliothek des Datenflussprogramms in der Sektion „Variablen“ Aktoren zum Lesen und Schreiben der entsprechenden Variable. Wenn Sie beispielsweise eine Variable namens `value` anlegen, können Sie diese z.B. in einer Transitionsbedingung mit dem Wert `value > 2.5` verwenden und die Variable mit dem Funktionsblock *value Schreiber* vom Datenflussprogramm aus setzen.

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

- Verwenden Sie das gleiche *SDF*-Programm wie oben – die aktuell gewünschte Geschwindigkeit soll jedoch aus einer globalen Variable ausgelesen werden.
- Verwenden Sie die Taster, um zwischen den verschiedenen Geschwindigkeitsstufen (z.B. 3000, 3500 und 4000 U/min) weiter- bzw. umzuschalten, 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.