Laufzeitmessung für SCCharts auf Lego Mindstorms

Dirk Sommerfeld

Bachelorarbeit
2015/16

Prof. Dr. Reinhard von Hanxleden
Real-Time and Embedded Systems group
Department of Computer Science
Kiel University

Advised by
Dipl.-Inf. Ass. iur. Insa Fuhrmann

Eidesstattliche Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Kiel,

Zusammenfassung

Diese Bachelorarbeit beschäftigt sich mit der Messung der Laufzeit von SCCharts auf LEGO Mindstorms. SCCharts sind eine Modelliersprache für sicherheitskritische reaktive Systeme. Die LEGO Mindstorms sind programmierbare Systeme, die beispielsweise in einen Roboter oder eine Maschine aus speziellen LEGO Bauteilen eingebettet werden können. KIELER ist ein Forschungsprojekt der Arbeitsgruppe Echtzeitsysteme und Eingebettete Systeme der Christian-Albrechts-Universität zu Kiel. Durch das Projekt KIELER kann man mit SCCharts ein reaktives System modellieren und es für eine Ausführung auf den LEGO Mindstorms übersetzen. Reaktive Systeme sind zyklisch aufgebaut. Sie lesen Eingaben, berechnen daraus eine Reaktion und erzeugen Ausgaben. Für einen fehlerfreien Ablauf ist es typischerweise wichtig, dass die Laufzeitanforderung der Verarbeitungsschritte eingehalten wird. Es wurde ein Mechanismus entwickelt, um bei der Ausführung auf den Mindstorms die Laufzeit zu messen. Das funktioniert mit Zeitstempeln, welche bei der Ausführung gesetzt werden. Im Laufe der Arbeit wurden mehrere Methoden zur Generierung von Zeitstempeln betrachtet und ihre Verwendbarkeit für eine Messung beurteilt. Dann wurde ein Mechanismus zur Laufzeitmessung auf LEGO Mindstorms in KIELER integriert.

Inhaltsverzeichnis

Abbildungsverzeichnis

Quellcodeverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

Brick programmierbarer Baustein eines LEGO Mindstorms

KiCo KIELER Compiler

KIELER Kiel Integrated Environment for Layout Eclipse RichClient

Prom Project Management in KIELER

SCCharts Sequentially Constructive Charts

TPP Timing Programm Point

WCET worst-case execution time

Kapitel 1

Einleitung

Im Bereich der eingebetteten Systeme befasst man sich mit Computern oder Software, welche fest in Geräte eingebaut und an eine bestimmte Funktion angepasst sind. Die Geräte reagieren meist auf Steuerbefehle und Sensoren, um dann Aktoren wie Motoren anzusteuern und werden oft im Dauerbetrieb eingesetzt. Deshalb ist bei der Entwicklung darauf zu achten, dass die Systeme fehlerfrei und ohne Abstürze arbeiten. Die Ausführungszeit der Programme ist dabei sehr entscheidend. Dabei gilt nicht „schneller ist besser“, sondern dass das System im richtigen Moment angemessen reagiert. Die Ausführungszeit hängt von vielen Faktoren ab. Die verwendete Hardware bestimmt, wie schnell ein einzelner Befehl ausgeführt wird und wie hoch der Durchsatz an Befehlen ist. Sie bestimmt auch die Befehle, die das System versteht. Diese sind in einem Instruktionssatz niedergeschrieben. Auch die umgebende Software, insbesondere das verwendete Betriebssystem, sowie die Art, wie ein Stück Code zur Ausführung gebracht wird oder wie hoch die Auslastung des Systems ist, sind Faktoren, die die Ausführungszeit beeinflussen können.

1.1 Worst-case execution time

Der Begriff Worst-Case Execution Time (WCET) bezeichnet die längst mögliche Ausführungszeit eines Codeabschnitts. Eine exakte Berechnung der WCET kann sehr aufwendig werden, in einigen Fällen ist es sogar praktisch unmöglich. Deshalb setzt man Techniken ein, die die WCET annähernd bestimmen. Es gibt zwei große Teilgebiete von Techniken, die WCET zu bestimmen[WEE+]. Das eine Gebiet umfasst statische Analysen. Ein verbreitetes Verfahren dafür ist das Arbeiten auf einem Kontrollflussgraphen des Programms. Dabei stellt man die Verzweigung eines Programms in einem Graphen dar und berechnet beispielweise den längsten Pfad des Graphen. Das zweite Gebiet sind messbasierte Verfahren. Das Ausführen auf der Zielhardware verbunden mit einem Messmechanismus ist hierfür eine Möglichkeit. Alle Verfahren haben Vor- und Nachteile. Bei einer statischen Analyse des Programms kann es schwierig werden, das restliche System einzubeziehen. Im System können beispielsweise Caches verbaut sein, deren zeitliches Verhalten nicht oder nur schwierig vorhersagbar ist. Beim Ausführen auf der Zielhardware ist es im Allgemeinen unmöglich alle Fälle zu testen. Deshalb kommt es oft zur Unterschätzung der WCET, da die längste Ausführungszeit nicht in der Testmenge liegen muss. Ein Vorteil von statischen

Analysen, im Vergleich zu Messmethoden, ist, dass die Zielhardware für die Berechnung nicht direkt eingesetzt werden muss.

Bei eingebetteten Systemen handelt es sich meist um reaktive Systeme. Das System liest eine Eingabe, berechnet eine Reaktion für diese und produziert entsprechende Ausgaben. Die Systeme lesen meist in regelmäßigen Abständen Eingaben ein. Dieser Zyklus wird Tick genannt. Bei der Berechnung der WCET müssen der Zustand und die Eingaben des Programms berücksichtigt werden. So kann die Dauer des längstmöglichen Tickintervalls berechnet werden. Mit dieser Information kann eine sichere Taktung des Systems bestimmt werden.

Die Sequentially Constructive Charts (SCCharts) [HDM+14] bilden eine grafische, synchrone Programmiersprache für sicherheitskritische Systeme. Ein SCChart ist typischerweise ein reaktives Programm. Es hat Eingaben, Ausgaben und verschiedene Zustände. Das SCChart kann zu einer Tickfunktion kompiliert werden. Bei der Berechnung des WCETs für ein SCChart wird die erzeugte Tickfunktion betrachtet. Es wird die langsamste Ausführungszeit dieser Funktion, unter der Berücksichtigung des Zustandsraums und der möglichen Eingaben des SCCharts, gesucht.

1.2 Analyse von SCCharts

Das Kiel Integrated Environment for Layout Eclipse RichClient (KIELER) ist ein Forschungsprojekt zur Verbesserung von grafischer, modellbasierter Entwicklung von komplexen Systemen. An der Christian-Albrechts-Universität zu Kiel wird die Modellierung von eingebetteten Systemen mittels SCCharts und der KIELER Entwicklungsumgebung auf LEGO Mindstorms gelehrt. Die LEGO Mindstorms werden außerdem am Lehrstuhl für Echtzeitsysteme und Eingebettete Systeme als Forschungsobjekt eingesetzt. Ein wesentlicher Aspekt von Echtzeit-Systemen ist die Einhaltung von Laufzeitanforderungen. Deshalb ist man an einer Laufzeitanalyse von SCCharts auf LEGO Mindstorms interessiert.

Ein aktiver Bereich der Forschung in KIELER ist das *Hotspot Highlighting*. Dabei werden Elemente des SCCharts hervorgehoben, die im langsamsten Tick besonders viel Zeit verbrauchen, die sogenannten Hotspots (Siehe Abbildung 1.2). Um ein solches Highlighting anzubieten, wird eine Analyse der Laufzeit benötigt. In KIELER wurde das Hotspot Highlighting mit einer Analyse des Analysewerkzeugs KTH's Time-aware Analyzer (KTA) des Royal Institute of Technology (KTH) in Schweden implementiert. Der Kieler Compiler kann ein SCChart unter anderem in die Sprache C übersetzen. Das Analysewerkzeug analysiert das SCCharts indem es bei dem generieten C-Code eine erschöpfende Suche einsetzt und eine Ausführung auf einem MIPS 32 Prozessor simuliert. Man möchte das

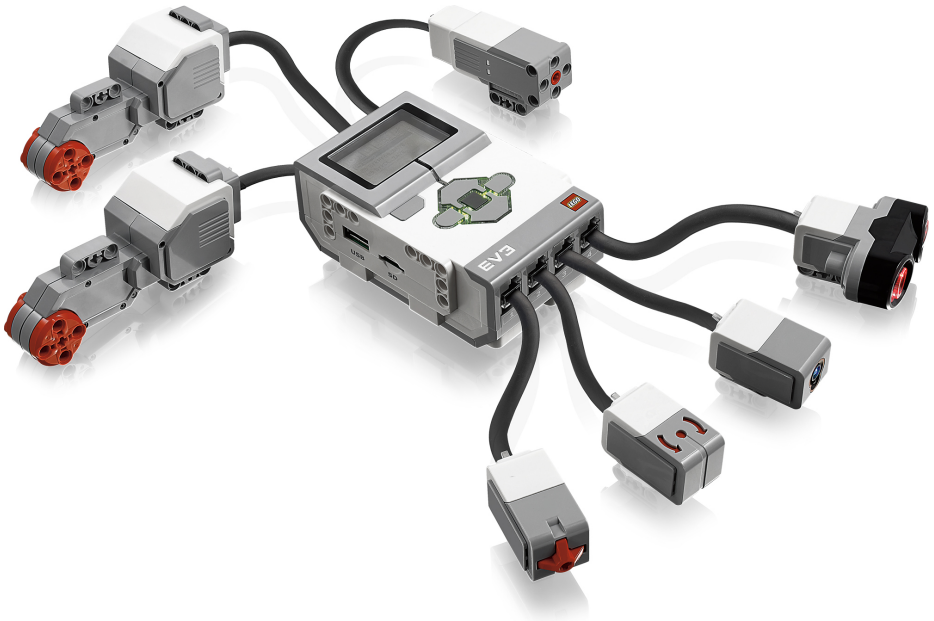


Abbildung 1.1. LEGO Mindstorms EV3 mit Sensoren und Aktoren[1]

Hotspot Highlighting auch für LEGO Mindstorms einsetzen. Dafür benötigt man die WCET des SCCharts auf diesem System.

Ein zusätzlicher Verwendungszweck einer Laufzeitanalyse auf LEGO Mindstorms ist einen Vergleich zwischen Hotspots eines SCCharts, erzeugt vom KTA für den MIPS 32 Prozessor und den Hotspots des SCCharts auf den LEGO Mindstorms machen zu können.

1.3 Problemstellung

In KIELER soll es möglich sein, eine Laufzeitanalyse von SCCharts auf LEGO Mindstorms zu erstellen. Dafür habe ich einen Mechanismus entwickelt, der aus der kompilierten Tickfunktion für das SCChart unter Zuhilfenahme von Informationen über die Zustände und Eingaben des SCCharts ein lauffähiges Programm erstellt und es auf dem LEGO Mindstorms ausführt. Bei der Ausführung werden Informationen über die Laufzeit erstellt und in einer Datei abgespeichert. Es werden alle Kombination aus Zuständen und Belegung von Eingaben gemessen und die Messergebnisse gespeichert. Das Messen der Laufzeit ist in die Software integriert und nicht beispielsweise über eine externe Schnittstelle der

[1] Quelle: `https://entwickler.de/wp-content/uploads/2015/07/loewenstein_robotik2_2.jpg`

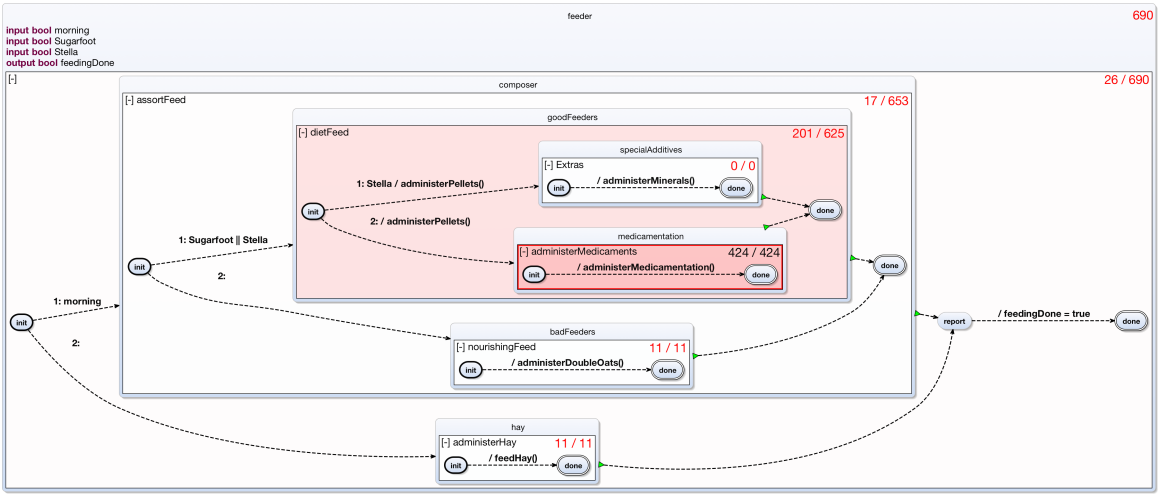


Abbildung 1.2. SCChart mit Laufzeitanalyse und Hotspot Highlighting

Hardware realisiert, damit die Messung ohne Modifizierung des Mindstorms möglich ist. Innerhalb der Ausführung werden Zeitstempel gesetzt. Hierfür wurden die verfügbaren Methoden zur Generierung von Zeitstempeln geprüft und eine Methode verwendet, die möglichst genau ist und wenig Einfluss auf das System hat. Dieser Mechanismus wurde dann in KIELER implementiert. In der Tickfunktion gibt es Variablen, die beschreiben, in welchem Zustand sich das SCChart befindet. Die Anzahl und Belegung der Variablen hängt dabei vom einzelnen SCChart ab. Meistens sind bei der Ausführung des SCCharts nur bestimmte Belegungen der Variablen gültig. Wenn keine Informationen über die gültigen Belegungen vorhanden sind, werden alle Belegungen der Variablen als gültig angenommen, um eine Messung durchzuführen. In der Implementierung dieser Arbeit werden nach einer Messung die ermittelten Laufzeiten in tabellarischer Form zur Verfügung gestellt. Eine Interpretation der Laufzeiten wurde nicht implementiert.

1.4 Aufbau der Arbeit

Im folgenden Kapitel wird auf verwandte Arbeiten eingegangen, die sich mit der Bestimmung von WCET beschäftigen oder eine Analyse der Hardware von LEGO Mindstorms liefern. In Kapitel 3 werden Technologien vorgestellt, auf welchen das Messverfahren aufbaut oder die es verwendet. Kapitel 4 behandelt den Aufbau und die Struktur des Messverfahrens. Bevor das Messverfahren entwickelt werden konnte, musste zunächst ein passendes System und eine angemessene Methode zur Zeitmessung gefunden werden. Wie der strukturelle Aufbau des gewählten Messverfahrens dann in KIELER umgesetzt ist, wird in Kapitel 5 beschrieben. In Kapitel 6 werden dann einige Ergebnisse der Mes-

sung präsentiert und analysiert. Im letzten Kapitel werden die Ergebnisse der Arbeit zusammengefasst und zukünftige Verwendungszwecke sowie offene Aufgabenstellungen vorgestellt.

Kapitel 2

Verwandte Arbeiten

In diesem Kapitel werden andere Arbeiten vorgestellt, die sich mit Laufzeitanalysen beschäftigen. Es wird eine Methode zur Messung der Laufzeit vorgestellt, welche über externe Hardware den Prozessor steuert. Danach wird auf verschiedene Techniken zur Laufzeitanalyse eingegangen und ein paar Werkzeuge analysiert, welche für die Bestimmung von statischen Laufzeitanalysen entwickelt wurden.

2.1 Joint Test Action Group Interface

Im Paper „Ada user guide for LEGO MINDSTORMS NXT" [BPZ11] wird beschrieben, wie man die LEGO Mindstorms NXT mit der Programmiersprache Ada verwenden kann. Dazu wird auch erklärt, wie man Debugging über ein externes Interface durchführt. Es handelt sich um ein Joint Test Action Group Interface (JTAG) und wurde 1985 für Test- und Debuggingzwecke entwickelt. An dieses Interface schließt man einen Test Access Port (TAP) Adapter an. Um dieses zu ermöglichen, muss man den Brick öffnen und die Leitung für das Interface auf der Hauptplatine verlöten (Abbildung 2.1, links). Danach ist man in der Lage, über den TAP Adapter den Mindstorm zu steuern. Dabei kann man nicht nur den Speicher manipulieren und Programme starten, sondern auch systeminterne Zustände auslesen. Man kann beispielsweise den Prozessor anhalten, seine Register auslesen und sogar neu setzen und anschließend die Ausführung fortsetzen. Über dieses Interface könnte man beispielsweise die Prozessorzyklen mitzählen, welche ein Programm zur Ausführung benötigt.

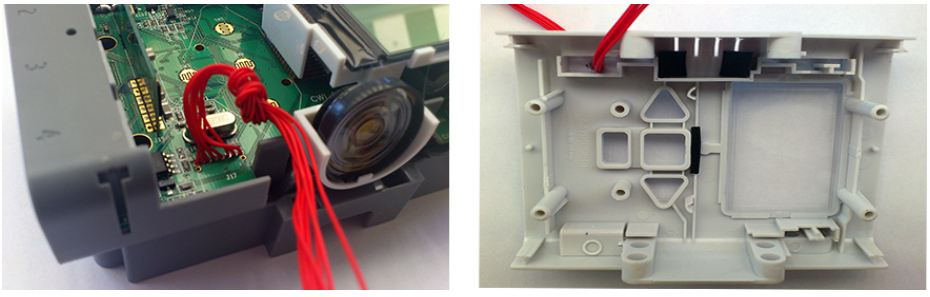


Abbildung 2.1. Geöffneter NXT mit Modifikation für JTAG Interface[1]

In dieser Bachelorarbeit soll eine Methode entwickelt werden, die bei der Entwicklung von Modellen, wenig Zusatzaufwand für die Durchführung der Laufzeitanalyse benötigt. Um das JTAG Interface nutzen zu können, wird zusätzliche Hardware in Form eines Adapters benötigt. Für die Verdrahtung des Adapters muss der Brick geöffnet, das Interface verlötet und nach außen geführt werden. Schließlich muss die Software für das Ansteuern des Interfaces auf dem Rechner konfiguriert werden, mit dem der Mindstorm angesteuert werden soll. Da es nicht abschätzbar ist, wie gut die vorhandene Software an eine Laufzeitmessung anpassbar ist und die Hardware modifiziert werden müsste, wurde dieser Ansatz in dieser Arbeit nicht weiter verfolgt.

2.2 Übersicht über Laufzeitanalysewerkzeuge

Das Paper „The Worst-Case Execution Time Problem - Overview of Methods and Survey of Tools"[WEE+] beschreibt allgemein das Vorgehen vieler Analysewerkzeuge und stellt einige Analysewerkzeuge vor. Es werden drei grundsätzliche Herangehensweisen an die Analyse beschrieben, beispielsweise ob eine Analyse direkt auf dee ganzen Ausführung oder nur auf Teilbereichen des Systems oder des Programms arbeitet. Die beiden Bereiche statische und messbasierte Methoden zur Laufzeitanalyse, welche in Abschnitt 1.1 beschrieben sind, werden in 5 Unterkategorien eingeteilt und diese genauer beschrieben. Beispielsweise die Simulation von Ausführungen eines Programms ist nicht klar in einen Bereich einteilbar. Die Vorteile und Nachteile verschiedener Analysetechniken werden vorgestellt und ein Vergleich zwischen statischen und messbasierten Analysen gezogen. Danach werden verschiedene Werkzeuge zur Laufzeitanalyse vorgestellt.

Unter anderen auch „aiT WCET Analyzers" von AbsInt[2]. Dieses Werkzeug wurde im Verlauf der Recherche dieser Arbeit auch betrachetet, um zu ermitteln, ob sich diese für eine Analyse von SCCharts auf einem LEGO Mindstorms eignet. Das Werkzeug unterstützt die Prozessoren der LEGO Mindstorms NXT und EV3. Es soll die WCET eines Programms für diese Architekturen berechnen können. Von der Verwendung in meiner Arbeit habe ich jedoch abgesehen, da es sich um ein kommerzielles Werkzeug handelt.

Das Werkzeug „Open Toolbox for Adaptive WCET Analysis"[3] (OTAWA) ist im Rahmen eines Forschungsprojektes an der University of Toulouse, Frankreich, entstanden. Im Paper „OTAWA: An open toolbox for adaptive WCET analysis"[BCR+10] werden die Elemente und Funktionen von OTAWA vorgestellt. Der Vorteil dieses Werkzeugs ist, dass es durch Abstraktion auf verschiedene Hardware anpassbar ist. Man kann über ein Interface eine

[1] Quelle: Ada user guide for LEGO MINDSTORMS NXT [BPZ11]
[2] http://www.absint.com/
[3] http://www.otawa.fr/

Beschreibung des Prozessors an das Werkzeug übergeben und damit die WCET-Analyse auf diesen Prozessor ausweiten. Damit könnte man OTAWA auf die LEGO Mindstorms anpassen. Ein mögliches Problem ist die Anpassung des Werkzeugs an die Eigenschaften eines SCCharts. OTAWA verwendet Bibliotheken, um verschiedene Analysetechniken anbieten zu können. Inwieweit diese für SCChart verwendbar bzw. erweiterbar sind, wurde im Rahmen dieser Arbeit nicht betrachtet, da das Hauptziel der Bachelorarbeit eine Messung auf der Hardware ist. Ein weiteres Problem ist, dass für SCCharts das Messen von einzelnen Codeabschnitten möglich sein soll und in OTAWA berechnet man die Laufzeit von Funktionen.

Kapitel 3

Verwendete Technologien

Dieses Kapitel beschreibt Technologien, welche bei der Messung benötigt oder verwendet werden. Dazu gehört die Softwarearchitektur KIELER, in welche der Mechanismus zur Zeitmessung integriert wurde. Des Weiteren wird die Hardware beschrieben, die für eine Messung in Betracht gezogen wurde. Dabei handelt es sich um verschiedene Versionen der LEGO Mindstorms.

3.1 Eclipse

Eclipse[1] ist eine iä.ntegrierte Entwicklungsumgebung (IDE). Sie wurde von IBM bis 2001 unter dem Namen „IBM Visual Age for Java 4.0“ entwickelt. Danach wurde der Quellcode offengelegt und Eclipse als frei verfügbare Software weiterentwickelt. 2004 wurde die Eclipse Foundation[2] gegründet, eine Open-Source-Gemeinschaft, die es sich zur Aufgabe gemacht hat, Eclipse zu betreuen. Eclipse wurde ursprünglich als Java Entwicklungsumgebung entwickelt, mittlerweile gibt es allerdings eine Vielzahl von Anwendungsgebieten. Durch einen modularen Aufbau kann man die Funktionalität von Eclipse erweitern, dafür wurde eine Plug-in-Architektur umgesetzt. Durch Plug-ins kann man die vorhandene Funktionalität erweitern oder neue Funktionen hinzufügen. Es ist dadurch auch möglich, eine eigene Sprache mit einem Editor in Eclipse umzusetzen.

3.2 Kiel Integrated Environment for Layout Eclipse RichClient

Das Forschungsprojekt Kiel Integrated Environment for Layout Eclipse RichClient[3] (*KIELER*) beschäftigt sich mit der Verbesserung von graphischen Modellierungen für die Entwicklung von eingebetteten und sicherheitskritischen Systemen. Dafür wurde auf Basis von Eclipse eine Entwicklungsumgebung aufgebaut. Diese kann zur Modellierung von SCCharts[HDM+14] verwendet werden. Das SCChart wird in einer textuellen Form modelliert. Zu diesem Modell wird dann automatisch eine visuelle Repräsentation generiert. Das erleichtert nicht nur den Modellierungsprozess, sondern unter anderem auch das

[1] https://eclipse.org/

[2] https://eclipse.org/org/foundation/

[3] http://rtsys.informatik.uni-kiel.de/confluence/KIELER/

Verstehen von unbekannten SCCharts. In KIELER ist nicht nur die Modellierung von SCCharts möglich, darüber hinaus es gibt noch weitere Funktionen für die Entwicklung von komplexen Systemen. Diese Funktionen sind in KIELER in Projekte aufgeteilt. An dieser Stelle werden Projekte von KIELER vorgestellt, die für die Implementierung der Laufzeitmessung relevant sind.

3.2.1 Kieler Compiler

Der Kieler Compiler (*KiCo*)[MSH14] wurde in KIELER für die Übersetzung von SCCharts entwickelt. Er funktioniert mit dem *Single-Pass Language-Driven Incremental Compilation (SLIC)* Prinzip. Das bedeutet, dass der Compiler in einzelne Transitionen untergliedert ist. Diese Transitionen übersetzen immer ein Element einer Sprache. Dadurch ist der Compiler modular in einzelne Transformationen aufgebaut und kann um weitere Transformationen erweitert werden. Für jede Transformation wird angegeben, welche Eingaben in dieser möglich sind und welche Ausgabe produziert wird. Dadurch kann man KiCo für verschiedene Zielplattformen einsetzen und Zwischenschritte der Kompilierung ersetzen oder weitere einfügen. Beim Aufruf von KiCo wird dabei in einem *Context* übergeben, welche Transformationsschritte benutzt werden sollen. Man kann KiCo unter anderem auch dafür verwenden, aus einem SCChart eine Tickfunktion zu erstellen. Die Tickfunktion kann dann in der Sprache C oder Java erzeugt werden.

3.2.2 Project Management in KIELER

In der Entwicklungsumgebung KIELER ist die Entwicklung von Systemen in Projekte eingeteilt. Um diese Projekte für die Modellierung von SCCharts vorzubereiten und eine bestimmte Zielplattform zu wählen, wurde die Erweiterung Project Management in KIELER (*Prom*)[Sta15] entwickelt.

Prom dient dazu das Projekt zu initialisieren, den benötigten *Context* für KiCo zu konfigurieren und die Tickfunktion des SCCharts in ein Programm einzubetten. Dafür muss ein Environment für diese Plattform vorhanden sein und bei der Erstellung des Projekts ausgewählt werden. Es folgt die Benennung des Projekts. Optional können noch viele weitere Einstellungen vorgenommen werden. Zu diesen zählen zum einem die Einstellung von Prom und zum anderen die Konfigurationsmöglichkeiten, welche Eclipse für Projekte bietet. Passend für das entsprechende Environment gibt es Snippets und eine Vorlage für das Programm, in welches die Tickfunktion eingebettet wird. In den Snippets stehen Codeabschnitte, die die Eingaben und Ausgaben des SCCharts verarbeiten können. Man kann anschliend über Annotationen im SCChart den Snippets die nötigen Informationen zukommen lassen, um bestimmte Abschnitte für das Programm zu verwenden. Bei der Ausführung des SCCharts wird ein vollständiges Programm mit Hilfe von KiCo aus der

Vorlage und den Snippets erstellt. Prom übernimmt die Einstellung von KiCo, erstellt die Dateien und startet entweder Kommandozeilenbefehle oder einen *launch shortcut*. Die Einstellungen dafür können auch über das Environment übergeben werden. Der *launch shortcut* ist ein Mechanismus in Eclipse, um aus einem Editor oder der Workbench Programme zu starten. Ein solcher launch shortcut wird beispielsweise vom Eclipse Plug-in leJOS angeboten. Mit diesem kann man Java Programme automatisch übersetzen lassen und auf den, mit leJOS Firmware bestückten Brick, hochladen. Um einen *launch shortcut* zu erstellen wird von Eclipse ein Interface bereitgestellt.

Um es zu ermöglichen, in Prom weitere Environments hinzuzufügen, wird ein Interface angeboten. Man kann in einem Projekt in KIELER dieses Interface implementieren. Dazu werden die Einstellungen für das Environment übergeben und die Programmvorlage und Snippets erstellt.

3.2.3 Timing Program Points

Das Projekt „Interactive Timing Analysis“[FBS+14] beschäftigt sich mit der Darstellung der Laufzeit eines SCCharts bei der Entwicklung von Systemen. Eine experimentelle Entwicklung, die daraus hervorgegangen ist, ist das *Hotspot Highlighting*. Um ein solches Highlighting anzubieten, wurde eine Infrastruktur implementiert, die eine Analyse des SCCharts möglich macht.

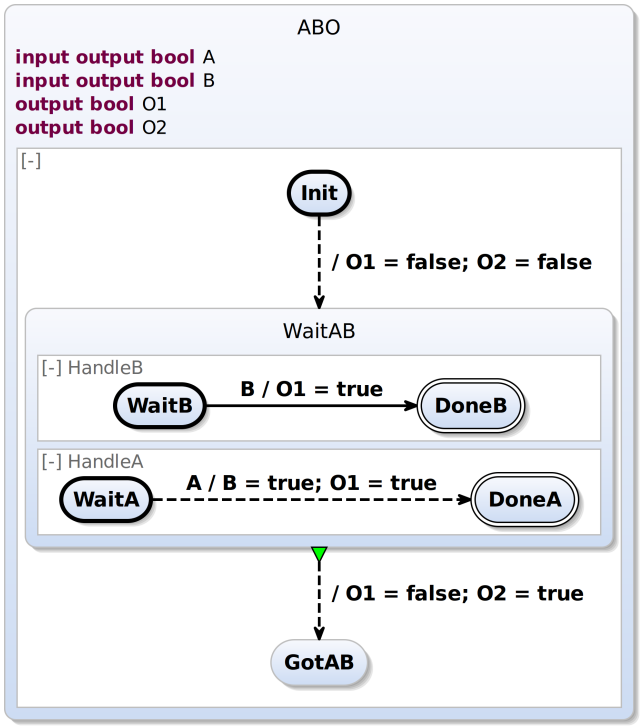
Das Tracing[Sch14] ist eine Erweiterung von KIELER, mit der man Beziehungen zwischen Modellelementen vor und nach einer Modelltransformation herstellen kann. In dem Projekt „Interactive Timing Analysis“ wurde eine Methode entwickelt, die aus dem Tracing eine Zuordnung von Elementen des SCCharts zu Abschnitten der Tickfunktion berechnet. Dabei wird die Tickfunktion so strukturiert, dass eine Zuordnung der Abschnitte zu Regionen des SCCharts möglich ist. Mit dieser Methode wurde dann eine Transition für KiCo entwickelt, die in einem Zwischenschritt sogenannte *Timing Program Points* (TPP) in die Tickfunktion einfügt.

Die *Timing Program Points* sind durchnummeriert und werden als Markierung in der Tickfunktion eingearbeitet. Diese Punkte trennen Abschnitte in der Tickfunktion voneinander. Wenn man nun die Laufzeit zwischen den TPPs kennt, kann man daraus die anteilige Laufzeit der Regionen des SCCharts berechnen. Ein Beispiel für eine Übersetzung von einem SCChart mit den TPPs ist in Abbildung 3.1 illustriert, dabei wird aus dem SCChart C-Code generiert. Die drei TPPs in Zeile 9, 17 und 25 teilen die Tickfunktion in vier Abschnitte. Diese Abschnitte sind verschiedenen Regionen des SCChart zugeordnet.

Damit man eine Laufzeitanalyse durchführen kann, werden neben den TPPs in der

Tickfunktion noch weitere Informationen zum SCChart benötigt. Das sind zum einen Informationen über die Verwendung von Variablen in der Tickfunktion, also ob sie als Ein- oder Ausgaben genutzt werden oder ob sie für die Speicherung des Zustands des SCCharts zuständig sind. Zum anderen benötigt man Informationen, welche Zustände des SCCharts gültig sind. Um diese Informationen zu erstellen, gibt es im Projekt eine Funktion, welche in einer Datei die Information erstellt. Diese Datei heißt timing-analysis-Datei (.ta-Datei). Die Funktion erstellt einen Teil der Daten aus der Übersetzung des SCCharts und den anderen Teil aus einer assumption-Datei (.asu-Datei). Die assumption-Datei enthält die gültigen Zustände des SCCharts und es können über sie Laufzeiten von Hostcode-Aufrufen integriert werden. Diese geben für eine Laufzeitanalyse die Möglichkeit, Hostcode-Aufrufe durch Dummies zu ersetzen und trotzdem für die Aufrufe Laufzeiten berechnen zu können. Eine Laufzeitanalyse eines SCCharts kann dann mit der passenden .ta-Datei und der generierten Tickfunktion, in der bei der Übersetzung die TPPs eingefügt wurden, durchgeführt werden.

Das *Hotspot Highlighting* ist eine mögliche Verwendung einer Laufzeitanalyse. In der Analyse wird der längste Tick des SCCharts berechnet, die anteiligen Laufzeiten der Regionen im SCChart durch die TPPs des Ticks bestimmt und diese Information in ein Highlighting wie in Abbildung 1.2 umgesetzt.



(a) Das SCChart *ABO*

```
1  void tick(){
2
3    {
4      g0 = _GO;
5      if(g0){
6        O1 = 0;
7        O2 = 0;
8      }
9      TPP(1);
10     g4 =(PRE_g3);
11     g2 =((g4&&A)||(_GO&&A));
12     if(g2){
13       B = 1;
14       O1 = 1;
15     }
16     g3 =((_GO&&(!(A))) ||
           (g4&&(!(A))));
17     TPP(2);
18     g7 =(PRE_g6);
19     g7b = g7;
20     g6 =(_GO||(g7b&&(!(B))));
21     g8 =(g7b&&B);
22     if(g8){
23       O1 = 1;
24     }
25     TPP(3);
26     g8_e2 =(!(g6));
27     g2_e1 =(!(g3));
28     g9 =((g2_e1||g2) &&
           (g8_e2||g8) &&
           (g2||g8));
29     if(g9){
30       O1 = 0;
31       O2 = 1;
32     }
33     g11 =(PRE_g10);
34     g10 =(g9||g11);
35   }
36   PRE_g3 = g3;
37   PRE_g6 = g6;
38   PRE_g10 = g10;
39   _GO = 0;
40
41 return;
42 }
```

(b) C-Code von *ABO* mit TPPs

Abbildung 3.1. Ein SCChart mit generiertem C-Code

(a) Der Mindstorms RCX[4]

(b) Der Mindstorms NXT[5]

Abbildung 3.2. Die ersten Mindstorms

3.3 LEGO Mindstorms

Die LEGO Mindstorms Produktserie besteht aus drei programmierbaren Bausteinen. Die sogenannten Bricks werden in Paketen verkauft, welches neben Aktoren und Sensoren auch andere Legosteine enthalten. Die Aktoren sind beispielsweise Motoren und Lampen. Bei den Sensoren gibt es Taster oder Geräte zur Umgebungserkennung, wie Ultraschallsensoren, für Entfernungsmessung oder Gyrosensoren um Drehungen zu erkennen. Man kann aus diesen Paketen verschiedenste Roboter oder Maschinen bauen und zur Steuerung Programme erstellen und auf den Brick hochladen. Der Brick besitzt Buchsen zum Anschließen von Aktoren und Sensoren.

Der erste Brick, mit dem Namen RCX (Robotics Command System), ist 1998 erschienen. Sein Hauptprozessor hat eine 8-Bit Architektur. Basierend auf diesem System wurde das zweite System mit dem Namen LEGO Mindstorms NXT im Jahr 2006 entwickelt. Der Brick besitzt drei RJ12-Anschlüsse für Aktoren und vier RJ12-Anschlüsse für Sensoren. Die Aktoren des NXT bestehen aus Glühlampen und Servomotoren. In die Servomotoren sind Rotationssensoren integriert, wodurch man den Motor gradgenau ansteuern kann. Die Sensoren bestehen aus einem Abstandsmesser auf Ultraschallbasis, einem Lichtsensor, der Helligkeit erkennt, einem Schallsensor, der Geräusche erkennt und Tastern, welche betätigt oder als Kontaktsensor verwendet werden können. Im Inneren des Bricks ist ein Atmel AT91SAM7S256 Mikroprozessor verbaut. Der Prozessor gehört in die ARM7TDMI Familie. Die Prozessorfamilie benutzt je nach Version einen anderen Instruktionssatz. Im Fall des AT91SAM7S256 ist es der ARMv4T Instruktionssatz.

[4] Quelle : `http://vignette2.wikia.nocookie.net/herofactory/images/5/57/LEGO_RCX.png/revision/latest?cb=20110118221326`
[5] Quelle : `https://entwickler.de/wp-content/uploads/2015/07/loewenstein_robotik2_2.jpg`

Der aktuellste Brick, der im Jahr 2013 erschienen ist, heißt EV3. Er besitzt vier RJ12-Anschlüsse für Aktoren und vier RJ12-Anschlüsse für Sensoren. Die Anschlussgeräte des NXT sind mit dem EV3 kompatibel. In dieser Version gibt es neben allen Sensoren und Aktoren, welche auch für den NXT zur Verfügung stehen, noch einen kleinen Motor und einen Gyrosensor, welcher Drehungen erkennen kann. Der Lichtsensor kann neben Hell und Dunkel auch Farben erkennen. Zur Steuerung des Bricks sind sechs Knöpfe angebracht. Die Micro-USB-Buchse dient als Kommunikationsschnittstelle. An der Seite hat der Brick eine USB-Buchse und ein Einschubfach für Micro-SD-Karten. Die Buchse kann verwendet werden, um weitere USB Geräte an den Mindstorm anzuschließen, wie beispielsweise einen WLAN-Empfänger. Das Einschubfach ist zur Erweiterung des Speichers des Bricks gedacht. Dabei kann man eine SD-Karte bis zu der Größe von 32 GB verwenden. In Abbildung 3.3 ist ein Roboter abgebildet, der aus einem EV3 gebaut wurde. An der Seite des Bricks sieht man die USB- und SD-Kartenschnittstelle.

Im Inneren des Ev3 ist, wie bei den Vorgängern auch, ein Ein-Chip-System (System on a Chip) eingebaut. Beim Ev3 ist es der von Texas Instruments hergestellte AM1808. Der Hauptprozessor auf diesem Chip gehört in die ARM9 Architektur-Familie. Es ist der ARM926EJ-S Kern mit einem Takt von 300 MHz (3 1/3 ns pro Prozessorzyklus). Er benutzt einen 32-Bit ARMv5 Instruktionssatz. Dieser Instruktionssatz ist 2003 erschienen. Der Ev3 hat einen Arbeitsspeicher von 64 MB RAM, sowie einen internen Speicher mit 16 MB Flash-Speicher und kann per SD-Karte um bis zu 32 GB erweitert werden.

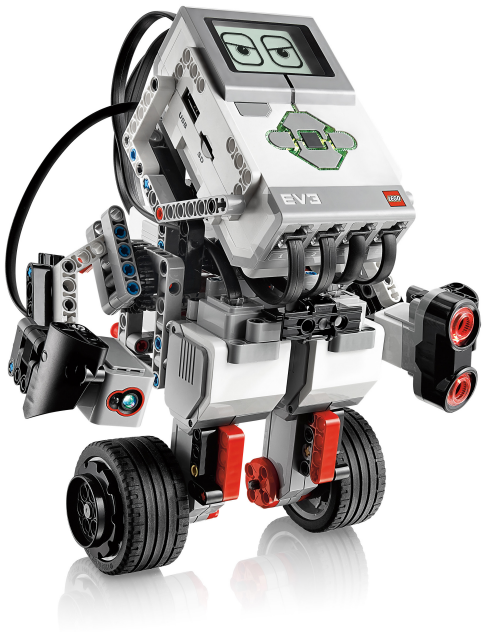


Abbildung 3.3. Der Gyro-Boy, ein Roboter aus einem EV3[6]

Der EV3 unterstützt, neben dem von LEGO entwickelten Betriebssystem LEGO EV3 Firmware, auch noch weitere Betriebssysteme. Es ist nicht nötig die Firmware auf dem Brick zu überschreiben, denn er ist so

[6] Quelle: `http://mms.businesswire.com/bwapps/mediaserver/ViewMedia?mgid=352951&vid=5`

konfiguriert, dass ein passendes bootbares Betriebssystem per SD-Karte eingesteckt werden kann. Der Brick priorisiert das Betriebssystem von der SD-Karte. Der Ev3 unterstützt unter anderem ein Linux-Debian-Kernel. Der in dieser Arbeit verwendete Kernel heißt ev3dev. Er bietet eine Netzwerkschnittstelle über den Micro-USB-Anschluss, mit der man den Brick per SSH ansteuern kann. Es gibt auch die Möglichkeit, den Brick per Bluetooth oder WLAN anzusprechen. Für WLAN muss man allerdings einen WLAN-Empfänger an die integrierte USB-Schnittstelle anschließen. Wenn der EV3 mit einem Rechner verbunden ist, kann er sich über diesen Rechner mit dem Internet verbinden, um beispielsweise Updates herunterzuladen. Der Kernel ev3dev ist kompatibel mit allen Programmen, die für ARM unter Linux zur Verfügung stehen. Außerdem gibt es einen Crosscompiler, der für alle gängigen Betriebssysteme bereit steht. Mit diesem Compiler können C-Programme für den Kernel übersetzt werden. So ist es nicht nötig Programme auf dem Brick zu kompilieren. Das Programm kann auf einem System übersetzt und auf den Ev3 hochgeladen werden.

Kapitel 4

Entwurf und Aufbau des Messverfahrens

In diesem Kapitel wird beschrieben, wie das Messverfahren aufgebaut ist. Dazu wird erklärt, welche Methoden zur Laufzeitmessung im Laufe der Arbeit betrachtet wurden. Die Laufzeitmessung soll auf LEGO Mindstorms durchgeführt werden, dafür wurden verschiedene Betriebssysteme analysiert. Danach werden für das eingesetzte Betriebssystem verschiedene Methoden für die Erstellung von Zeitstempeln vorgestellt. Der Ablauf der Messung wird im letzten Abschnitt beschrieben.

4.1 Betrachtung der Mindstorms und der Betriebssysteme

Für die Zeitanalyse sollte nach Möglichkeit ein Zielsystem gewählt werden, das Programme in einer Sprache versteht, die sich gut für eine Analyse eignet. Eine weit verbreitete Sprache für Analysen ist C, da sie hardware-nah übersetzt werden kann und wenige Hintergrundprozesse benötigt, um ausgeführt zu werden. In dieser Arbeit wurde der LEGO Mindstorms EV3 verwendet und als Betriebssystem ein Linux-Kernel. In Verlauf der Arbeit wurden noch weitere Betriebssysteme betrachtet. Diese stelle ich im folgenden vor.

LEGO Firmware: Für die vom Hersteller auf den Bricks installieren Betriebssysteme gibt es keine Möglichkeit ein SCChart auszuführen. Diese Betriebssysteme können nur Programme ausgeführen, welche mit der LEGO Software erstellt wurden. Dabei wird eine grafische Programmiersprache verwendet, die auf LabVIEW bassiert.

leJOS: leJOS ist ein Java-Betriebssystem. Es ist für die Programmierung mit Java ausgelegt. leJOS wurde sowohl für den EV3, wie auch für den NXT entwickelt. Java bietet bei der Entwicklung von Programmen den großen Vorteil der Plattformunabhängigkeit. Der Code wird in einer virtuellen Maschine ausgeführt, die für die Plattform angepasst sein muss. Ein weiterer Vorteil ist, dass man bei der Entwicklung von Programmen den Speicher nicht selbst verwalten muss. Die Verwaltung des Arbeitsspeichers ist in Java automatisiert. Ein daraus entstehender Nachteil ist, dass ein sogenannter *Garbage Collector* benötigt wird. Dabei handelt es sich um ein Programm, welches im Hintergrund läuft und Speicherbereiche bereinigt, wenn diese nicht mehr referenziert werden. Der Garbage Collector und die virtuelle Maschine sind schwer berechenbar und können bei einer Messung

die Ausführungszeit erheblich beeinflussen. Aufgrund dieser Nachteile wurde von einer Verwendung von Java für die Messung abgesehen.

4.1.1 Betriebssysteme des Lego Mindstorms NXT

Für den LEGO Mindstorms NXT gibt es eine Reihe von Betriebssystemen. Um das Betriebssystem zu wechseln, muss der Brick geflasht werden. Beim Flashen werden alle Daten auf dem Brick gelöscht und es muss ein neues Betriebssystem aufgespielt werden. Um ein neues Betriebssystem auf den Brick zu schreiben, benötigt man einen Rechner, bei dem der Treiber für den Mindstorms eingerichtet ist. Der Brick muss mit diesem Rechner über ein Micro-USB Kabel verbunden sein. Der LEGO Mindstorms NXT bietet kein Betriebssystem, welches C-Code verarbeitet und sich für die Messung eignet. Aber es gibt Betriebssysteme für den NXT welche C-ähnlichen Code benutzen, dazu gehören brickOS und nxtOSEK.

brickOS: Das Betriebssystem brickOS ist für eine Messung zu aufwendig. Es wird dafür ein Rechner benötigt, auf dem das „Bricx Command Center“ installiert ist. Die Einrichtung dieser Software ist für einige Systeme schwierig. So konnte bei einem Test für Linux die Software zwar gestartet werden, aber sie hat den Brick, welcher über USB verbunden war, nicht erkannt. Für Programme wird dann nicht die Sprache C verwendet, sondern die Programmiersprache „Not eXactly C“. Des Weiteren würde man einen speziellen Compiler für diese Programme benötigen.

nxtOSDEK: Ein weiteres Betriebssystem, das betrachtet wurde, ist nxtOSEK. Dieses bietet gegenüber brickOS den Vorteil, dass es eine GNU-ARM-Compiler-Chain (GCC) benutzt. Die Bedeutung von GCC hat sich im Laufe der Zeit sehr gewandelt. Die allgemeinste Bedeutung ist GNU-Compiler-Collection. Ursprünglich wurde der GCC als C-Compiler für Linux-Systeme entwickelt. Mittlerweile kann er eine Vielzahl von Sprachen übersetzen. Er ist auch für die Benutzung verschiedener Zielplattformen und Architekturen erweitert worden. Bei der Installation von nxtOSEK traten einige Probleme mit der Einrichtung des Bricks auf, da die Firmware nur als Quellcode vorliegt. Dieser muss zur Benutzung erst übersetzt werden. Zum Zeitpunkt dieser Bachelorarbeit konnte die Firmware auf einem aktuellen Ubuntu der Version 15.10 nicht kompiliert werden. Es werden nur Ubuntu 11.10 und Ubuntu 10.8 als verwendbare Betriebssysteme angegeben, um die Kompilierkette aufzubauen. Da es sich abzeichnete das nxtOSEK eine erhebliche Konfigurierungszeit des Entwicklungsrechners benötigt, wurde von einer Verwendung für die Zeitmessung abgesehen.

4.1.2 Lego Mindstorms EV3

Der LEGO Mindstorms EV3 kann flexibler mit verschiedenen Betriebssystemen umgehen. Es ist möglich das Betriebssystem zu wechseln, ohne den Brick neu beschreiben zu müssen. Man erstellt eine SD-Karte mit einem bootbaren Betriebssystem und steckt diese vor dem Starten des Bricks ein. Wenn man den EV3 dann startet erkennt er das Betriebssystem auf der SD-Karte und startet es anstelle der Firmware, welche intern im Brick installiert ist. Dabei wird das Betriebssystem im Flashspeicher nicht überschrieben. Stattdessen arbeitet der Brick von der SD-Karte aus. Für den EV3 gibt es auch verschiedene Betriebssysteme. Unter anderem gibt es eine Version von ROBOTC. Neu im Vergleich zu dem NXT ist, dass der EV3 mit einem Linux-Kernel kompatibel ist. Da sich heraus stellte, dass ein Linux-Kernel alle Kriterien für eine Messung erfüllt, wurden keine weiteren Betriebssysteme betrachtet.

ev3dev: Ev3dev ist ein Debian-Linux basiertes Betriebssystem und wurde speziell für den EV3 entwickelt. Durch die Benutzung von Linux stehen alle Programme zur Verfügung, welche sich für die ARM-Architektur kompilieren lassen. Außerdem kann der Cross-Compiler[1] (GNU C Compiler for armel architecture) verwendet werden. Ein Cross-Compiler ist eine Übersetzungsmaschine, welche den Quellcode eines Programms in ein ausführbares Programm übersetzen kann und dabei nicht auf der Zielhardware ausgeführt werden muss. Der LEGO Mindstorms EV3 ist ein eingebettetes System und für seinen Verwendungszweck leistungsstark, aber er ist nicht für komplexe Berechnungen ausgelegt. Der Rechner, der für die Entwicklung genutzt wird, ist im Allgemeinen deutlich leistungsfähiger als der eingesetzte EV3. Der Brick muss für die Messung das Programm lediglich ausführen. Der Compiler muss für eine Messung auf dem Entwicklungsrechner installiert sein und für KIELER zugänglich gemacht werden. Dafür muss man die Umgebungsvariable setzen, damit KIELER weiß, wo sich der Compiler befindet. Dadurch kann das Programm vollständig auf dem Rechner übersetzt werden. Es ist möglich, über ein Micro-USB-Kabel das Gerät mit einem anderen Rechner zu verbinden. Dann wird per USB eine Netzwerkverbindung aufgebaut. Im ev3dev läuft eine Software, welche über diese Schnittstelle eine Kommunikation per Secure Shell (SSH) ermöglicht. Man benötigt zur Steuerung des Bricks nur einen Client, der SSH unterstützt. Diese Clients sind in Linux standardmäßig installiert und es stehen auch Clients für Windows und MAC zur Verfügung. Die Messung benötigt also nur einen Rechner mit einem gängigen Betriebssystem und einem SSH-Client, welcher als kostenlose Software für alle zur Verfügung steht.

[1] https://launchpad.net/gcc-arm-embedded

4.2 Methoden zur Zeitmessung des EV3

Um eine Messung der Laufzeit durchführen zu können, sind in die Tickfunktion „Timing Program Points“ eingearbeitet. Zwischen diesen TPPs soll die vergangene Zeit gemessen werden. Dafür benötigt man eine Methode um Zeitstempel an den TPPs zu generieren. Es werden nun verschiedene Methoden vorgestellt, welche im Verlauf der Arbeit betrachetet wurden. Dabei hat sich die Methode *Timer* als geeignet herausgestellt und wird in der Laufzeitmessung dieser Arbeit eingesetzt.

Beim Testen verschiedener Methoden war es erforderlich, den Kernel des EV3 zu modifizieren. Gründe dafür waren zum einen, dass Funktionen betrachtet wurden, die nur dem Kernel zugänglich sind, zum anderen war zum Ausführen bestimmter Befehle eine Berechtigungsstufe erforderlich. Diese Befehle kann man entweder in einem Modul des Kernels ausführen oder in einer Kernelfunktion aufrufen. Um einem Linux-Betriebssystem ein neues dynamisches Modul hinzuzufügen, gibt es das Dynamic Kernel Module Support (DKMS) Framework. Um es zu benutzen, installiert man das *dkms* package und fügt dem Kernel die passenden Header hinzu. Da der ev3dev kein Standard-Kernel ist, kann man die Header nicht über den *package manager* beziehen. Man kann die Header im Git[2] von ev3dev finden und dem Kernel hinzufügen. Es war dennoch nicht möglich, auf dem Kernel neue dynamschine Module zu installieren. Zur Modifikation des Kernels musste er neu kompiliert werden. Dafür gibt es im Git vom ev3dev den kompletten Quellcode und Buildscripts zur Erstellung des Kernels. Diese kann man unter Ubuntu verwenden. Man kann den Quellcode des Kernels um weitere Module oder Systemfunktionen erweitern und ihn dann kompilieren. Es gibt zwei Methoden, den modifizierten Kernel zum Einsatz zu bringen. Man kann ein komplett neues Image vom Betriebssystem erstellen und dieses auf eine SD-Karte kopieren. Das hat den Nachteil, dass der Kernel sich beim ersten Starten neu initialisieren muss und alle Daten gelöscht werden. Die zweite Methode besteht darin, ein Updatepackage für das ev3dev zu erstellen. Dieses Package installiert man auf dem EV3. Dabei werden die Module und Systemfunktionen angepasst.

Bibliotheken: Die geläufigste Methode, um in einem Linux-Kernel einen Zeitstempel zu erhalten, ist die Funktion *gettimeofday(struct timeval *tv, struct timezone *tz)*. Das *struct timeval* besteht aus zwei Variablen. In der einen werden Sekunden und in der anderen Millisekunden gespeichert. Dieses struct wird von der Bibliothek time.h zur Verfügung gestellt. Beim Aufruf der Funktion werden die Sekunden und Millisekunden gespeichert, die seit dem 1.1.1970 vergangen sind. Diese Methode ist auch der Standard-Zeitstempel der Programmiersprache Java. Für eine Messung ist diese Methode ungeeignet, da sie nur Millisekunden genau messen kann, was für eine Messung zu grob ist. Ein weiteres struct

[2] https://github.com/ev3dev

in dieser Bibliothek ist *timespec*. Es speichert Sekunden und Nanosekunden. Man setzt einen Zeitstempel in timespec mit der Funktion *clock_gettime(clockid_t, struct timespec *)*. Als Parameter dieser Funktion übergibt man einen Pointer auf ein timespec struct und die Art des Zeitstempels, den man speichern will. Beispielweise gibt es als Art *CLOCK_MONOTINIC*. Damit bekommt man einen Zeitstempel von einem Zähler, der einen undefinierten Startpunkt hat und nicht gesetzt werden kann. Dadurch, dass man ihn nicht setzen kann, ist sichergestellt, dass niemand ihn zurücksetzen kann. Diese Methode ist nicht für eine Messung geeignet, da es sich bei Tests herausgestellt hat, dass der Jitter dieser Methode zu groß ist. Die gemessende Zeit zwischen zwei Zeitstempeln schwankte bei Wiederholung der Ausführung um mehr als fünf Millisekunden.

Neben der Möglichkeit, die Ausführungszeit über die Zeit zu messen, kann man die Prozessorzyklen zählen, welche ein Programm bei der Ausführung braucht. Dafür braucht man eine Unterstützung der Hardware, wie ein Register, welches bei jedem Prozessorzyklus inkrementiert wird. Außerdem muss sichergestellt sein, dass sich die Zyklen in eine Laufzeit umrechnen lassen. Bei einer Ausführung auf mehreren Prozessoren oder wenn der Prozessor einen variablen Takt hat, ist eine Berechnung der Laufzeit schwer oder gar unmöglich. Der EV3 besitzt nur einen Prozessor mit einem festen Takt von 300 Mhz.

In Linux gibt es eine Bibliothek, die Funktionen enthält, mit denen man die verbrauchten Prozessorzyklen eines Programms messen können soll. Bei Versuchen diese Bibliothek zu verwenden, stellte sich heraus, dass diese Bibliothek auf dem EV3 nicht funktionierte. Diese Bibliothek wird erst ab der ARMv6 Architektur unterstützt, wärend der EV3 eine ARMv5 Architektur benutzt.

Assembler: Unter x86 gibt es einen Standard-Assembler Befehl, mit dem man einen Zähler auslesen kann, der Prozessorzyklen zählt. Für ARM-Architekturen gibt es auch so einen Befehl. Es ist *MRC p15, 0, <Rd>, c9, c13, 0*. Dabei gibt das Schlüsselwort MRC an, dass man etwas aus dem Co-Prozessor bei ARM-Chips in ein Register des Hauptprozessors kopieren will. Das <Rd> ist das Zielregister im Hauptprozessor, die restlichen Optionen beschreiben, welches Register man aus dem Co-Prozessor kopieren möchte. Es gibt mehrere MRC-Befehle, auch für die ARMv5-Architektur, aber der Befehl *MRC p15, 0, <Rd>, c9, c13, 0* funktioniert nur für ARM Cortex-A8- und Cortex-A9-Architekturen. Da dies aber nicht gleich ersichtlich war, habe ich diesen Befehl auf dem EV3 ausprobiert. Um diesen Befehl ausführen zu dürfen, muss man sich im Kernelspace befinden. Ich habe über ein Modul ein Programm für den Kernel geschreiben, da sonst ein Fehler ausgegeben wird und das Betriebssystem die Ausführung des Befehls beendet. Nachdem die Erweiterung dem Kernel hinzugefügt wurde und der Befehl ausgeführt wird, wird das Programm beendet, weil die Hardware den Befehl nicht versteht.

Timer: Im Kernel des EV3 gibt es Mechanismen, welche mit einem Zeitstempel arbeiten. Aus den Informationen dieser Mechanismen wurde die Methode zur Generierung des Zeitstempels erstellt. Ein wichtiges Element des Kernels ist das Scheduling. Der Scheduler arbeitet dafür auf Basis eines Timers. Ich habe dem Kernel eine Erweiterung geschrieben, um das Verhalten dieses Timers zu analysieren. Dabei wurde festgestellt, dass der Scheduler auf einem „general purpose timer“ des Chips arbeitet. Er ist als 32-Bit-Zähler konfiguriert und läuft mit einer Frequenz von 24 MHz. Das bedeutet, ein hochgezähltes Bit in diesem Timer entspricht einer vergangenen Zeit von 41 $^2/_3$ Nanosekunden. Der Zähler hat einen Überlauf nach 178.956.969.942 Nanosekunden (Das entspricht etwa 2 Minuten und 58 Sekunden). Diese Konfigurierung ist für eine Zeitmessung geeignet. Die Schritteinheit ist klein genug, um die Messung einzelner Codeabschnitte zu ermöglichen. Zudem ist der Überlauf des Zählers groß genug, damit er nicht innerhalb der Messung eines Ticks mehrfach überläuft. Damit man auf den Timer zugreifen kann, ist er auf einer Speicheradresse hinterlegt. In der allgemeinen technischen Dokumentation der AM1808/AM1810[3] wird beschrieben, wie der Timer aufgebaut ist. Beispielsweise wie die Funktionen der einzelnen Register des Timers sich zusammensetzen. In der technischen Dokumentation des AM1808[4] wird dann beschrieben, dass der Timer an der Hex-Adresse 0x1c20010 im Speicher liegt. Um den Timer zu konfigurieren, müssen die Einstellungen in die passenden Register des Timers geschrieben werden. In der Implementierung wird der schon konfigurierte Timer verwendet, welchen auch der Scheduler nutzt. Dabei wird dem Programm ein lesender Zugriff auf den Teil des Speichers zugänglich gemacht, in dem der Timer liegt. Es ist also nicht möglich, den Timer zu setzen, um Fehler im Betriebssystem zu vermeiden. Der Timer wird im Betriebssystem auch von weiteren Funktionen verwendet. Der Vorteil dieser Methode ist, dass es nicht nötig ist den Kernel ev3dev zu modifizieren, um einen Timer zu konfigurieren und eine Messung durchzuführen.

4.3 Struktureller Aufbau des Messverfahrens

In diesem Teil wird nun beschrieben, wie die einzelnen Elementen des Messverfahrens zusammengefügt werden. Wir beginnen mit den SCChart, konstruieren daraus ein Programm und führen es auf dem Mindstorms aus. Aus dem SCChart muss zunächst die Tickfunktion erstellt werden. Dafür wird der *Kieler Compiler* verwendet. KiCo wird dafür so konfiguriert, dass die Tickfunktion in der Sprache C erzeugt wird und die Timing Program Points in der Tickfunktion enthalten sind.

[3] http://www.ti.com/lit/ug/spruh82a/spruh82a.pdf
[4] http://www.ti.com/lit/ds/symlink/am1808.pdf

4.3.1 Beschreibung der timing-analyse-Datei und assumption-Datei

Für die Laufzeitmessung eines SCCharts wird nicht nur eine einzelne Ausführung der Tickfunktion durchgeführt. Sie wird mit verschiedenen Eingaben und Zuständen aufgerufen. Die Information über Zustände und Eingaben werden durch eine Datei übergeben. In der sogenannten timing-analysis-Datei (.ta-Datei) werden die nötigen Daten für die Zeitanalyse geliefert. In der Datei werden die Informationen durch Schlüsselwörter markiert. Es gibt folgende Schlüsselwörter:

- Funktion <Funktionsname> : Die hier benannte Funktion ist diejenige, die analysiert werden soll. Typischerweise ist das die Tickfunktion.
- InitFunction <Funktionsname> : Die Funktion setzt das SCChart auf den initialen Zustand zurück.
- State <Variablenname>: Alle Variablen, in denen der Zustand des SCCharts gespeichert wird, werden mit dem Schlüsselwort *State* markiert.
- GlobalVar <Variablenname> <Wertebereich>: Mit diesem Schlüsselwort ordnet man den Eingaben des SCCharts einen Wertebereich zu. Es werden alle Werte aus diesem Bereich gemessen. In der aktuellen Implementierung kann man Boolean mit dem Wertebereich 0...1 angeben und Integer mit beispielsweile 3..67.
- FunctionWCET <Hostcode-Aufrufsname> <Wert>: Durch dieses Schlüsselwort kann man für Hostcode-Aufrufe eine Abschätzung der Laufzeit angeben.
- Combination: Nach diesem Schlüsselwort wird in den folgenden Zeilen eine gültige Belegung der Zustandsvariablen beschrieben. Die Zeilenanzahl entspricht dabei der Anzahl der durch State gekennzeichneten Variablen. In jeder Zeile steht ein Variablenname und der Wert dieser Variable.
- FWCET <Wert1> <Wert2>: Dieses Schlüsselwort beschreibt zwischen welchen TPPs die Zeit gemessen werden soll.
- WCP <Wert1> <Wert2>: Damit wird der Laufzeitanalyse mitgeteilt, dass man den Pfad zwischen Wert1 und Wert2 im längsten Tick wissen möchte.

Die meisten dieser Einstellungen können durch das KIELER Projekt Interactive Timing Analysis aus dem Result von KiCo erstellt werden. Zusätzlich zum Result wird noch eine assumption-Datei (.asu-Datei) eingelesen. Darin sind die restlichen Einstellungen gespeichert. Dazu gehören die Abschätzungen der Laufzeiten von Hostcode-Aufrufen, welche unter dem Schlüsselwort FunctionWCET zu finden sind und die gültigen Belegungen der Zustandvariablen mit dem Schlüsselwort *Combination*.

Die Informationen der assumption-Datei sind nicht immer vorhanden. Für Hostcode-Aufrufe gilt, dass sie nicht ersetzt werden, wenn keine Information über ihre Laufzeit vorhanden ist. Sie werden also normal ausgeführt und ihre Laufzeit mitgemessen. Wenn keine gültigen Kombinationen für die Belegung der Zustandsvariablen gegeben werden, werden alle Kombinationen als gültig angesehen. Es wird dann das Kreuzprodukt der Zustandsvariablen gebildet und gemessen.

4.3.2 Erstellung der Tick-Datei

Nun haben wir die Tickfunktion in C-Code und der timing-analysis-Datei. Aus den Combinations werden dann Setfunktionen erstellt. Diese funktionieren ähnlich wie die InitFunktion. Sie setzen das SCChart nur nicht auf den Startzustand, sondern in einen Zustand der analysiert werden soll. Hostcode-Aufrufe, deren Laufzeiten bekannt sind, werden in der Tickfunktion ersetzt. Dafür werden Variablen von Typ Integer angelegt und auf 0 initialisiert. Im Namen der Variablen ist der Hostcode-Aufruf und der Codeabschnitt codiert, in dem der Hostcode-Aufruf enthalten ist. Wenn der Aufruf in mehreren Codeabschnitten vorkommt, werden mehrere Variablen angelegt. Anstelle des Aufrufes in der Tickfunktion wird dann die entsprechende Variable inkrementiert. Damit ist später nachvollziehbar, wie oft die Hostcode-Funktion aufgerufen wurde. Damit der GCC das Inkrement wie einen Funktionsaufruf behandelt, also keine Optimierung an dieser Stelle durchführt, wird vor und nach dem Inkrement eine Barriere errichtet.

Eine Barriere besteht aus den Befehl *asm volatile(::: "memory")*. Es ist ein leerer Assembler-Befehl, d.h. an der Ausführung des Programms ändert sich nichts, aber durch die Schlüsselwörter *volatile* und *memory* beeinflusst man den Compiler. Wenn man ein Assembler-Befehl als *volatile* deklariert, optimiert der Compiler nicht über diesen Befehl hinweg, er behält also die Reihenfolge des Quellcodes an dieser Stelle bei. Steht bei einem Assembler-Befehl die Option *memory*, nimmt der Compiler an, dass sich die Variablen im Speicher durch den Assembler-Befehl geändert haben können und liest alle Variablen für die weitere Benutzung neu ein.

In der Tickfunktion müssen noch die TPPs durch den Zeitstempel ersetzt werden. Um den ersten und letzten Codeabschnitt messen zu können, müssen noch TPPs am Anfang und am Ende der Tickfunktion eingefügt werden. Dann wird pro TPP eine Variable angelegt. In der Tickfunktion wird dann der TPP ersetzt durch eine Zuweisung in der entsprechenden Variable. Die Zuweisung entspricht dem Wert des Zählregistes vom Timer. Damit sich die einzelnen Codeabschnitte nicht durch Optimierung mischen, werden auch hier Barrieren vor und nach dem Aufruf eingefügt. Zur Veranschaulichung habe ich für ein SCChart *secure* (Abbildung 4.1) modelliert. In Abbildung 4.2 ist die Tickfunktion vor

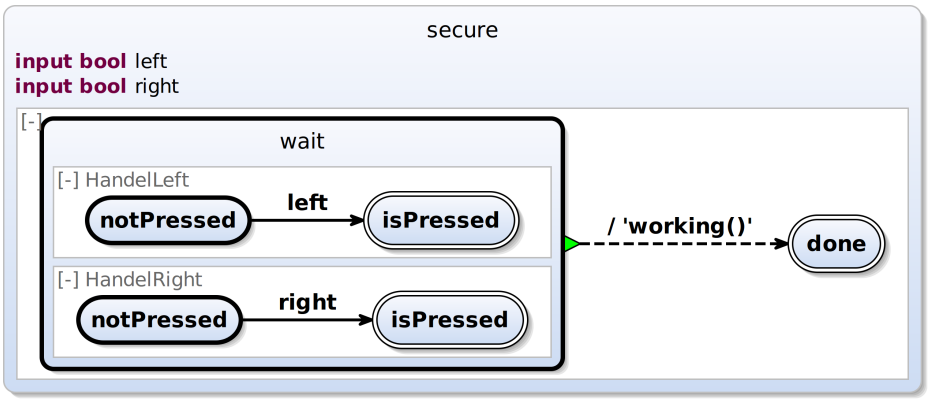


Abbildung 4.1. SCChart des Beispiels Abbildung 4.2

und nach der Bearbeitung dargestellt.

4.3.3 Erstellung der Main-Datei und des Headers

Als nächstes wird die Mainfunktion erstellt. In der Mainfunktion wird zunächst der Pointer für den Zeitstempel initialisiert. Dafür wird ein lesender Zugriff auf den Speicherbereich angefordert, in dem der Timer liegt. Der Timer wird in Abschnitt 4.2 vorgestellt. Es wird ein Pointer auf den Zähler des Timers gesetzt. Danach werden die Variablen für die Messung initialisiert. Für jede Eingabe, in der .ta-Datei gekennzeichnet durch GlobalVar, wird ein Schleifenzähler und ein Array angelegt. Das Array enthält die Werte, die im Wertebereich angegeben sind. Es wird noch eine Zählvariable für die Schleife angelegt, die den Zustand des SCCharts setzt. Des Weiteren wird ein Puffer für die Messergebnisse angelegt. Die Größe des Puffers ist so gewählt, dass die Messergebnisse eines Durchlaufes hinein passen. Da das SCChart vor jedem Aufruf in den passenden Zustand gesetzt wird, kann man außerhalb der Tickfunktion, in der Mainfunktion die Verwaltung für die Messung durchführen. Dazu gehören ineinander geschachtelte Schleifen, um die Tickfunktion für die Messung vorzubereiten. Die erste Schleife setzt dabei einen Zustand der Tickfunktion. Dann wird für jede Eingabe des SCCharts eine Schleife erstellt. Damit wird ein Kreuzprodukt aus den Zuständen und allen Belegungen der Eingaben gebildet. Da es sich um eine Messung handelt, werden für jede Konfiguration des SCCharts zehn Messungen durchgeführt. Im Inneren der Schleifen befindet sich der eigentliche Tick-Aufruf. Nach dem Tick-Aufruf werden die Konfiguration und Zeitstempel des Ticks in eine Datei geschrieben, bevor der nächste Tick-Aufruf vorbereitet wird.

Damit ein Austausch von Variablen in C zwischen der Mainfunktion und der Tickfunktion funktioniert, wird eine Header-Datei erstellt. In der Header-Datei stehen der

```
1
2
3  char left;
4  char right;
5
6
7
8
9  char _GO;
10 char g0;
11 char g2;
12 char PRE_g2;
13 char g3;
14 char g4;
15 char g6;
16 char PRE_g6;
17 char g7;
18 char g8;
19 char g9;
20 char g4_e1;
21 char g8_e2;
22
23 void tick(){
24
25
26   {
27     g0 = _GO;
28     TPP(1);
29     g3 =(PRE_g2);
30     g2 =(_GO||(g3&&(!(left))));
31     g4 =(g3&&left);
32     TPP(2);
33     g7 =(PRE_g6);
34     g6 =((g7&&(!(right)))||_GO);
35     g8 =(g7&&right);
36     TPP(3);
37     g4_e1 =(!(g3));
38     g8_e2 =(!(g7));
39     g9 =((g4_e1||g4) &&
           (g8_e2||g8) &&
           (g4||g8));
40     if(g9){
41       working();
42     }
43   }
44   PRE_g2 = g2;
45   PRE_g6 = g6;
46   _GO = 0;
47
48
49 return;
50 }
```

(a) Die originale Tickfunktion

```
1  #define TPP(LABEL) asm volatile("" ::: "memory");
       timingTPP##LABEL = *systemTimer; asm volatile(""
       ::: "memory")
2
3  char left;
4  char right;
5
6  unsigned long timingTPPStart, timingTPPEnd,
       timingTPP1, timingTPP2, timingTPP3;
7  int working_timing_exit = 0;
8
9  char _GO;
10 char g0;
11 char g2;
12 char PRE_g2;
13 char g3;
14 char g4;
15 char g6;
16 char PRE_g6;
17 char g7;
18 char g8;
19 char g9;
20 char g4_e1;
21 char g8_e2;
22
23 void tick(){
24 TPP(entry);
25
26   {
27     g0 = _GO;
28     TPP(1);
29     g3 =(PRE_g2);
30     g2 =(_GO||(g3&&(!(left))));
31     g4 =(g3&&left);
32     TPP(2);
33     g7 =(PRE_g6);
34     g6 =((g7&&(!(right)))||_GO);
35     g8 =(g7&&right);
36     TPP(3);
37     g4_e1 =(!(g3));
38     g8_e2 =(!(g7));
39     g9 =((g4_e1||g4) && (g8_e2||g8) && (g4||g8));
40     if(g9){
41       asm volatile("" ::: "memory");
           working_timing_exit++; asm volatile("" :::
           "memory");
42     }
43   }
44   PRE_g2 = g2;
45   PRE_g6 = g6;
46   _GO = 0;
47
48 TPP(exit);
49 return;
50 }
```

(b) Die modifizierte Tickfunktion

Abbildung 4.2. Überarbeitete Tickfunktion des SCCharts *secure*

Pointer auf den Zählers des Timers, die Variablen der Zeitstempel und der Eingaben des SCCharts. Die Mainfunktion und Tickfunktion importieren diesen Header. Damit kann die Mainfunktion die Eingaben der Tickfunktion setzen und nach dem Tickaufruf die Zeitstempel aus der Tickfunktion auslesen.

Nun wird aus dem generierten Programm ein ausführbares Programm mit dem GCC erstellt. Danach wird es dann per SSH auf den Mindstorms hochgeladen und gestartet. Nachdem alle Messungen abgeschlossen sind, wird die Datei mit den Ergebnissen zurückkopiert. Die Ergebnisse sind als Tabelle aufgebaut. Dabei entspricht eine Zeile in der Tabelle einem Tick des SCCharts. In der ersten Zeile der Tabelle wird beschrieben, was die einzelnen Spalten bedeuten. Die erste Spalte beschreibt welche Setfunktion benutzt wurde, um den Zustand des SCCharts zu setzen. Weiterhin gibt es für jede Eingabe des SCCharts eine Spalte in der beschrieben wird, mit welchem Wert die Eingabe beim Tick belegt ist. Es folgt für jeden TPP eine Spalte. Der Wert in dieser Spalte ist der gemessene Zeitstempel, der bei der Ausführung des Ticks gesetzt wurde. Danach können noch weitere Spalten folgen. Diese Spalten gehören zu den ersetzten Hostcode-Aufrufen. Dabei wird gezählt, wie oft ein Hostcode-Aufruf in einem Codeabschnitt aufgerufen werden würde. Es wird für einen Hostcode-Aufruf, der in einem Codeabschnitt vorkommt, eine Spalte erstellt. Durch die Abschätzung in der timing-analysis-Datei ist dann die Berechnung der Ausführungszeit des Ticks möglich ohne das die Funktion ausgeführt werden muss.

Kapitel 5

Implementierung in KIELER

In diesem Kapitel wird beschrieben, wie das Messverfahren aus Kapitel 4 in KIELER umgesetzt ist. KIELER wurde dafür um das Plug-in *ev3timing* erweitert. Durch dieses Plug-in ist eine Laufzeitmessung von SCCharts auf dem LEGO Mindstorms EV3 möglich. In Abbildung 5.1 sind die Dateien dargestellt, die für eine Messung erstellt werden und welche Projekte diese verarbeiten.

Die Erweiterung „Project Management in KIELER“ wird in Kapitel 3.2.2 vorgestellt. Die Verwaltung des Projektes für die Laufzeitanalyse wird von Prom übernommen. Das Projekt kann mit Hilfe von Prom erstellt werden. Nachdem das SCChart im Projekt modelliert wurde, kann es über „KiCo Compilation“ ausgeführt werden. In Abbildung 5.2 ist eine Möglichkeit dargestellt, wie nach der Modellierung das Programm gestartet werden kann. Prom erstellt die Tickfunktion und startet den *launch shortcut*, welcher von ev3timing bereit gestellt wird. Einen launch shortcut erstellt man über das Interface ILaunchShortcut. Es erlaubt, in der Entwicklungsumgebung aus dem Editor oder der Workbench das Programm zu starten. Dabei wird die entsprechende Implementierung des launch shortcuts aufgerufen und das Programm übergeben. An dieser Stelle soll das Plug-in ev3timing einsetzen und den restlichen Ablauf der Messung übernehmen. Wie Prom vorgehen muss, wird in einem Environment festgehalten. Das „EV3 Timing” Environment wird im Plug-in ev3timing über das Interface, welches Prom anbietet, erstellt. Bei der Erstellung des Environments übergibt man die Einstellungen, die dann im Projekt in KIELER gesetzt werden sollen.

Zu den Einstellung für das Environment gehört die Konfiguration von KiCo. KiCo wird aufgerufen, um aus dem SCChart die Tickfunktion in C mit eingefügten TPP zu erstellen. Dafür setzt man als targetLanguage im Environment die Zielsprache und zusätzliche Transformationen zum einfügen der TPPs. Die Transformation, welche die TPPs einfügt, setzt voraus, dass im Context von KiCo zwei Properties gesetzt wurden. Zum einen muss das Tracing eingeschaltet werden. Das geschieht indem man die boolesche Property Tracing.ACTIVE_TRACING auf „True” setzt. Zum anderen muss in der Property TimingAnalysis.INPUT_SCCHART das Modell des SCCharts übergeben werden. Das Tracing erzeugt bei Transformationen in KiCo Hilfsinformationen. Die Hilfsinformationen enthalten eine Zuordnung zwischen Elementen vor und nach einer Transformation. Daraus wird

eine Zuordnung von Elementen vom Anfang bis zum Ende der Kompilierung berechnet. Mit dieser kann man dann ein Mapping für Regionen des SCCharts erstellen. Aus diesem Mapping erkennt man die Positionen an denen die TPPs eingefügt werden müssen. Diese Properties sind vorläufig auf dem Branch in Plug-in Prom gesetzt. Damit Prom flexibel bleibt, sollte das Interface zur Erzeugung eines Environments um die Benutzung von Properties erweitert werden.

Im Environment wird auch der launch shortcut gesetzt, welcher von Prom nach der Generierung der Tickfunktion aufgerufen wird. Für die Messung ist im Plug-in ev3timing ein Shortcut erstellt und als *associated launch shortcut* zum Environment hinzugefügt worden. Der erste Schritt für die Laufzeitmessung ist Aufruf der Funktion *generateTimingRequestFile(String code, State scchart, String uriString, int highestInsertedTPPNumber)* aus dem Plug-in *tsccharts*. Die Funktion liest dann die Assumptions ein, wenn welche im Projekt vorhanden sind und erstellt dann die timing-analysis-Datei, welche die Informationen für die Messung enthält. Für diesen Aufruf werden Informationen aus dem Result von KiCo sowie das Modell des SCCharts benötigt. Prom übergibt dem launch shortcut nur die Main-Datei, wie es durch das Interface ILaunchShortcut vorausgesetzt wird. Aus diesem Grund ist dieser Aufruf vorläufig in Plug-in Prom implementiert. Eine Möglichkeit, diesen Aufruf aus Prom zu lösen und in das Plug-in ev3timing zu übertragen, wäre einen angepassten launch shortcut anzubieten, der neben dem Pro-

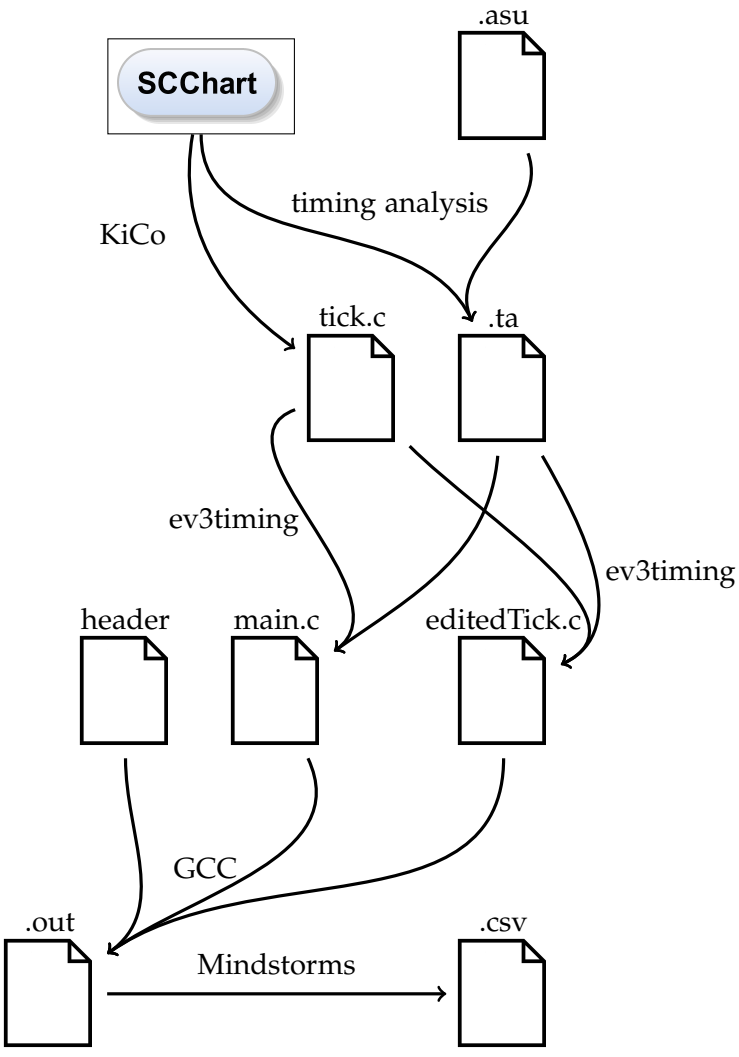


Abbildung 5.1. Erstellung der Programmdateien

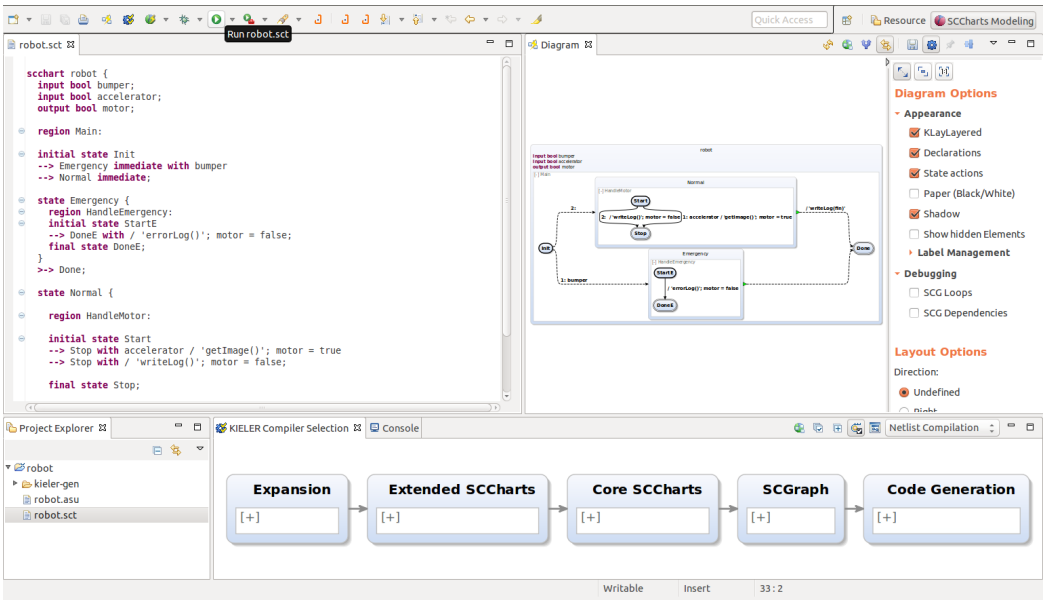


Abbildung 5.2. KIELER

gramm noch das Modell und das Result übergibt.

Nach der Erstellung der Informationen für die Messung wird eine Funktion in ev3timing benutzt, um das Programm zu erstellen. Dazu gehört das Editieren der Tick-funktion, das Erstellen der Setfunktionen, Mainfunktion und des Headers. Die Details dazu wurden in Kapitel 4 vorgestellt. Danach wird der ARM-Cross-Compiler aufgerufen, welcher das C-Programm zu einem ausführbaren Programm übersetzt. Damit das möglich ist, muss die GCC-ARM-Embedded-Toolchain[1] installiert sein und eine Umgebungsvariable für KIELER darauf gesetzt sein. Danach wird das Programm auf den EV3 hochgeladen. Die Bibliothek Java Secure Channel wird verwendet, um eine SSH-Verbindung zum EV3 herzustellen. Der EV3 ist dabei per USB-Kabel mit dem Rechner verbunden. Danach wird gewartet bis die Ausführung der Messung auf dem EV3 beendet ist und die Ergebnisse der Messung vorliegen. Die Datei, welche die Ergebnisse enthält, wird zum Abschluss, wieder per SSH, in das Projekt zurückkopiert.

[1] https://launchpad.net/gcc-arm-embedded

Kapitel 6

Evaluation

In diesem Kapitel wird der Ablauf einer Messung anhand eines Beispiels präsentiert und die Messergebnisse von verschiedenen SCCharts gezeigt und analysiert. Dabei werden die Vor- und Nachteile der Messung verdeutlicht.

6.1 Basisbeispiel SCChart *robot*

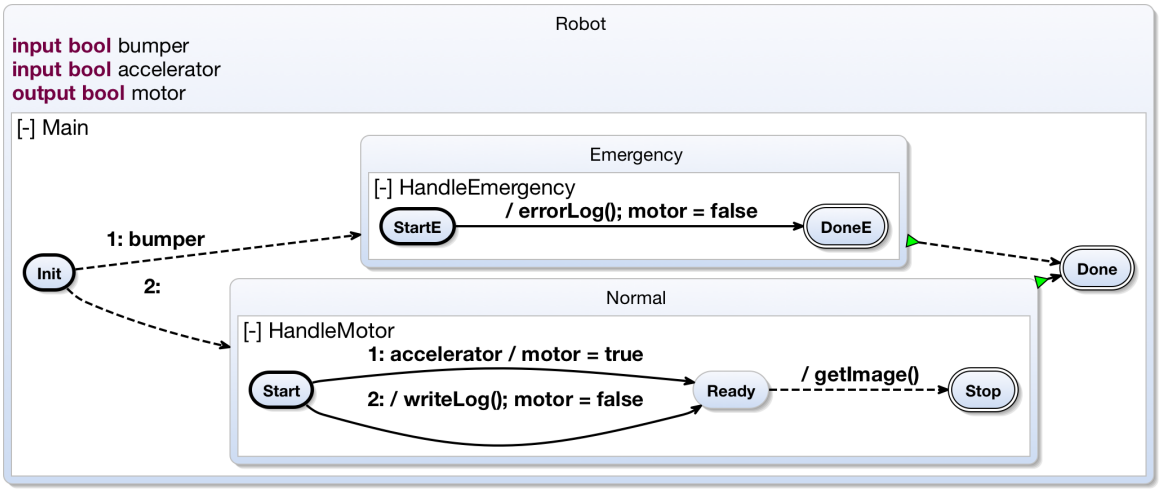


Abbildung 6.1. SCChart des Robots[1]

Das SCChart *robot* wurde in dieser Arbeit als Grundbeispiel verwendet, um die Messung zu entwickeln. Basierend auf diesem Beispiel wurde schon einmal eine Analyse durchgeführt[2], wodurch die gültigen Belegungen der Zustandsvariablen vorliegen. Es sind in diesem Fall drei Variablen, die den Zustand des SCCharts speichern. Es sind _GO, PRE_g1 und PRE_g4. Die Variable _GO gibt an ob es sich um den ersten Tick handelt und die Variablen

[1] Quelle : http://rtsys.informatik.uni-kiel.de/confluence/download/attachments/14516356/robot-synchron1.png?version=1&modificationDate=1452692490000&api=v2

[2] http://rtsys.informatik.uni-kiel.de/confluence/display/KIELER/Interactive+Timing+Analysis

PRE_g1 und PRE_g4 speichern, die Belegung der Variablen g1 und g4 im letzten Tick.

Als sogenannte *guards* werden die Variablen g1 und g4 bezeichnet, welche Bedingungen von Transitionen und aktive Zustände des SCCharts beschreiben. Von den acht (2^3) dadurch möglichen Kombinationen sind nur drei Kombinationen für das SCChart gültig. In Abbildung 6.1 ist das SCChart dargestellt. Im SCChart kann man die drei Zustände wiedererkennen, die gespeichert werden müssen. Diese sind der Init Zustand, der StartE Zustand und der Start Zustand. Die Zustände Emergency und Normal schließen sich gegenseitig aus. Die restlichen Zustände werden im gleichen Tick verlassen, in dem sie betreten werden, da sie ausgehende unbedinge immediate Transition haben oder Endzustände sind. Der Quellcode für das SCChart, also das SCChart textual, ist im Anhang A.1 enthalten. Für die Laufzeitanalyse ist in Abbildung A.2 auch die timing-analysis-Datei abgebildet. Aus der assumption-Datei werden die Daten mit den Schlüsselwörtern FunctionWCET und Combination importiert. Dann wird daraus die Mainfunktion (A.3) und Tickfunktion (A.4) erstellt. Danach wird das Programm ausgeführt und dabei die Laufzeit in eine „comma-separated values" Tabelle eingetragen. Eine Messung ist in Tabelle A.1 abgebildet. Man erkennt in den rechten Spalten die Anzahl der Aufrufe von den Hostcodefunktionen, welche für die Messung ersetzt wurden. Wenn durch die assumption-Datei Laufzeiten der Funktionen bekannt sind, können diese mit entsprechender Ausführungsanzahl eingerechnet werden.

SetNr	bumper	accelerator	Gesamtzeit in ns	errorLog	writeLog	getImage
0	0	0	8351	1	0	0
0	0	1	8269	1	0	0
0	1	0	8425	1	0	0
0	1	1	8302	1	0	0
1	0	0	8171	0	2	0
1	0	1	8099	0	1	1
1	1	0	8064	0	2	0
1	1	1	8269	0	1	1
2	0	0	8417	0	0	0
2	0	1	8339	0	0	0
2	1	0	8269	0	0	0
2	1	1	8380	0	0	0

Tabelle 6.1. Gesamtlaufzeit robot

Aus der Tabelle mit den Messergebnissen habe ich exemplarisch eine vereinfachte Version in Tabelle 6.1 erstellt. Dafür wurde ein Durchschnittswert der gemessenen Zeiten berechnet und dargestellt. Dabei wurden Werte, welche extrem abweichen, ignoriert.

Für die Hostcode-Aufrufe ist die Häufigkeit der Aufrufe in dem Tick aufgelistet. Die Gesamtzeit ist dabei ohne die Hostcode-Aufrufe berechnet. Dieses ist nur eine beispielhafte Interpretation zur Veranschaulichung. In dieser Arbeit wird keine Interpretation der Messung durchgeführt, dieses ist Teil der offenen Aufgabenstellung dieser Arbeit.

In dieser Messung gibt es beispielsweise 3 Werte die deutlich erhöht sind. Es wurde insgesamt 120 Ticks gemessen (3 $Zustaende * 2^2$ $Eingabenbelegungen * 10$ $Messung$). Den Wert 98.250 ns für einen Tick erhält man aus der Zeile: SetNr = 0, bumper = 1 und accelerator = 1 und von dieser Konfiguration die 8. Messung. Dieser Wert ist deutlich von den neun Vergleichswerten zu unterscheiden, da diese nur etwa 8000 ns brauchen. Eine mögliche Ursache dafür ist, dass es durch den Scheduler einen Kontextwechsel gegeben hat. Ein Überlauf des Timers ist ausgeschlossen, da der gesamte Ablauf der Messung nur wenige Sekunden gedauert hat und der Timer ungefähr drei Minuten für einen Überlauf benötigt. Wenn man die Zeit zwischen dem ersten und dem letzten Zeitstempel berechnet ergibt sich eine verbrauchte Zeit der Messung von 142.569.125 ns (0,1 Sekunden). Wenn der Zähler nach 2^{32} Bit auf 0 zurückspringt, erkennt man es durch eine negativen Laufzeit eines Ticks. Für diese Berechung gilt dann $(2^{32} - Endpunkt + Startpunkt)$.

6.2 Einfluss der Messung auf die Laufzeit

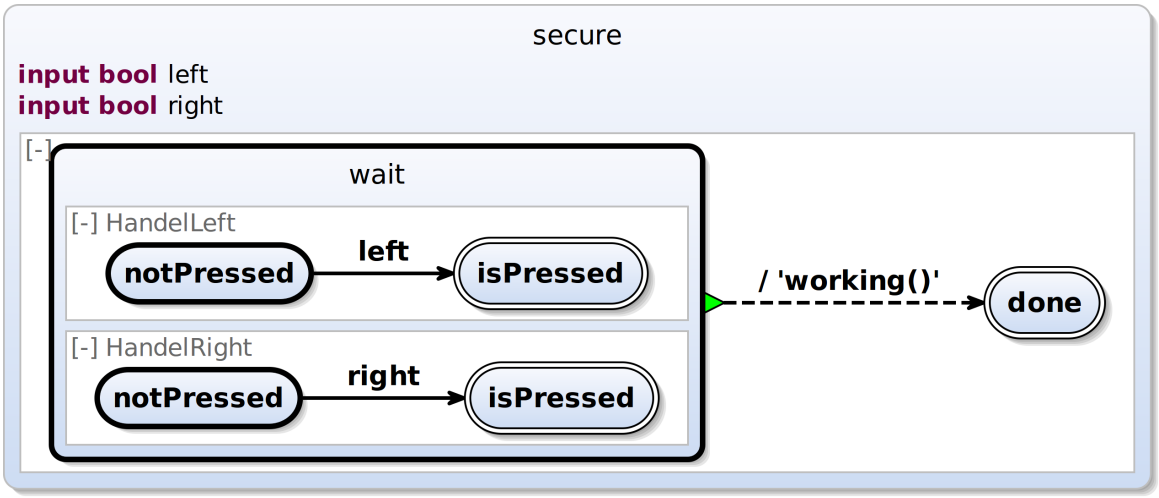


Abbildung 6.2. SCChart des *secure* Beispiels

Um den Einfluss der Messung auf die Laufzeit zu demonstrieren, wurde das SCChart *secure* mit unterschiedlicher Anzahl von TPPs durchgemessen. Für die Laufzeitanalyse sind

3 TPPs im Programm enthalten. Dazu kommen noch die impliziten TPPs, am Anfang und am Ende des Programmes. In der Tabelle 6.2 ist die Gesamtlaufzeit eines Ticks abgebildet. Die Anzahl der TPPs wird im Programm variiert. Somit kann man erkennen, dass die Anzahl der TPPs eine Rolle spielt. Die Laufzeit wird durch die TPPs erhöht. Dies erfolgt linear zur Anzahl der TPPs. Damit kann man über eine untere Schranke den Einfluss der TPPs ausgleichen.

SetNr	left	right	nur entry und exit	1 TPP	2 TPP	3 TPP
0	0	0	5477	6223	6778	7039
0	0	1	5473	5990	6756	7035
0	1	0	5391	5963	6664	7247
0	1	1	5481	6177	6646	6978
1	0	0	5522	5727	6662	6933
1	0	1	5694	5690	6531	6906
1	1	0	5608	5740	6564	7002
1	1	1	5678	5838	6519	7027
2	0	0	5002	5908	6367	6828
2	0	1	4940	5879	6428	6888
2	1	0	5416	6018	6445	6879
2	1	1	5469	6113	6469	7042
3	0	0	5248	5735	6375	6756
3	0	1	5223	5785	6432	6982
3	1	0	5682	5793	6482	6941
3	1	1	5703	5932	6564	6986
4	0	0	5174	6166	6658	7129
4	0	1	5383	5973	6678	7011
4	1	0	5166	6178	6547	6933
4	1	1	5298	5973	6609	6883
5	0	0	5514	6006	6601	7105
5	0	1	5543	5887	6535	6994
5	1	0	5384	5885	6601	7023
5	1	1	5507	5826	6551	6937
6	0	0	4895	5842	6301	6692
6	0	1	4891	5879	6309	6960
6	1	0	5194	5986	6650	7002
6	1	1	5330	5920	6523	6908
7	0	0	5186	5764	6211	6744
7	0	1	5100	5781	6207	6937
7	1	0	5481	6043	6551	7084

SetNr	left	right	nur entry und exit	1 TPP	2 TPP	3 TPP
7	1	1	5667	5899	6564	6863

Tabelle 6.2. Gesamtlaufzeit von *secure* verschiedener TPP-Anzahlen

6.3 Ersetzung von Hostcode-Aufrufen

Abbildung 6.3. Beispiel *print*

Ein weiteres Beispiel stellt das Ersetzen der Hostcode-Aufrufe dar. Dabei werden zwei Messungen mit der Linux-Systemfunktion *printf()* durchgeführt und zwei Messungen, bei denen der Aufruf mit Hilfe einer assumption-Datei ersetzt wurde. Jede Messung besteht aus 10 Aufrufen, dabei wird das SCChart immer zurückgesetzt. Das SCChart ist so minimal gehalten, dass es keine weitere Funktion besitzt, deshalb ist auch nur ein Zustand relevant. In der Tabelle 6.3 sind die vier Messungen abgebildet. Dabei ist deutlich der Nachteil von Systemfunktionen zu erkennen. Sie haben beim ersten Aufruf eine sehr hohe Laufzeit. Dazu schwankt ihre Laufzeit bei wiederholtem Aufruf immer noch deutlich. Bei den ersetzten Ergebnissen gibt es diesen Overhead nicht und man kann über die assumption-Datei angeben, mit welcher Laufzeit für den Aufruf gerechnet werden soll.

2. Messung mit printf	1. Messung mit printf	2. Messung mit ersetztem printf	1. Messung mit ersetztem printf
767889	574082	1886	1804
20787	20992	1681	1722
19639	19188	1845	1558
19270	24354	1681	1558
19844	20254	1681	1558
17917	19311	1640	1558
18737	18819	1845	1558

2. Messung mit printf	1. Messung mit printf	2. Messung mit ersetztem printf	1. Messung mit ersetztem printf
18737	19434	1845	1558
18286	19270	1845	1722
17302	19557	1845	1558

Tabelle 6.3. Messungen des SCCharts *print*, mit Hostcodeaufrufen und Ersetzungen für die Hostcodeaufrufe

6.4 Messung von größeren SCCharts

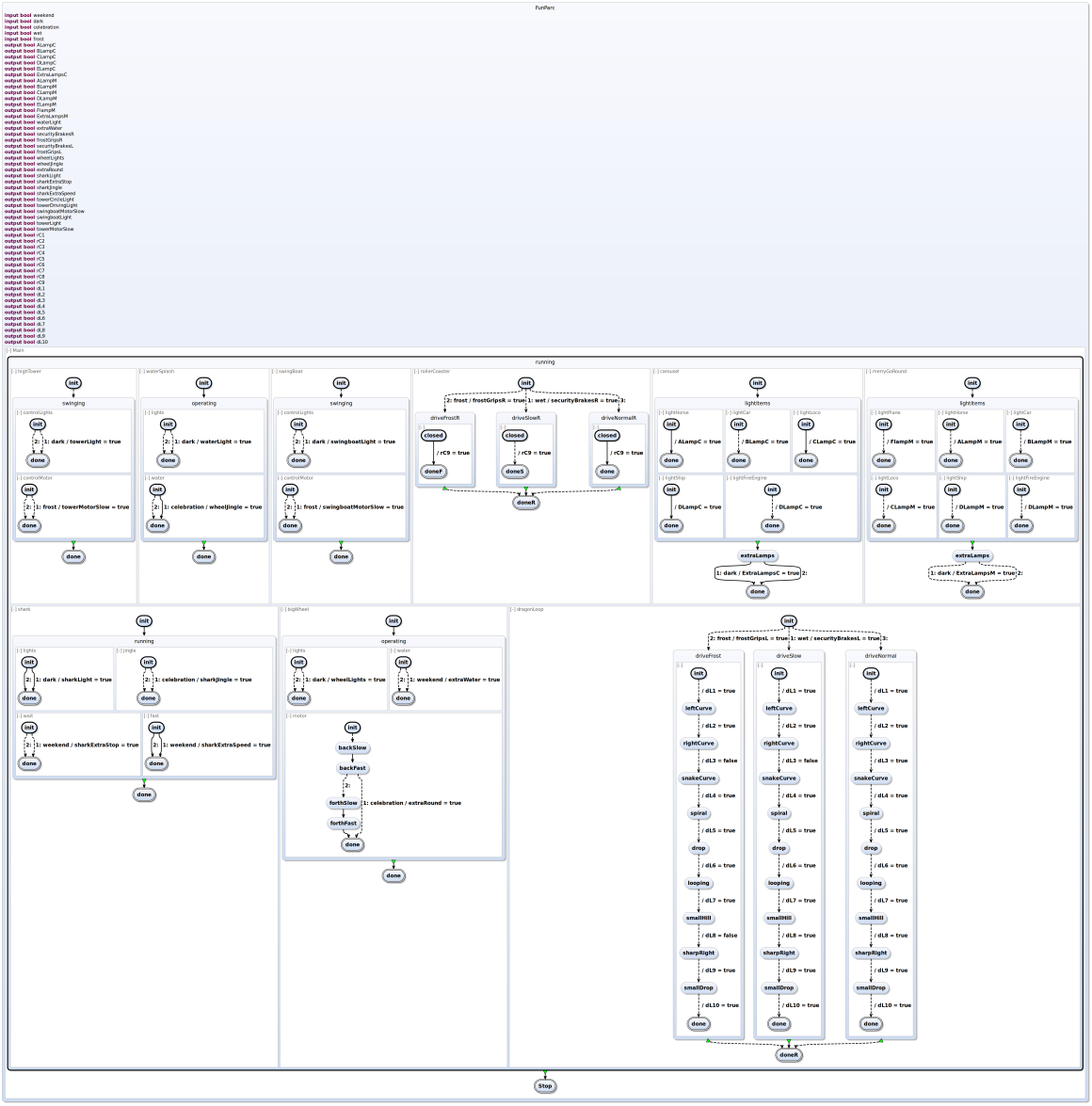


Abbildung 6.4. SCChart des FunParcs

Um die Skalierbarkeit zu demonstrieren, wurde das SCChart *FunParc* auf dem EV3 eine Messung durchgeführt. Das SCChart besitzt neun Variablen, welche den Zustand speichern, und fünf boolsche Eingaben. Jede Konfiguration wird zehnmal durchgemessen. Daraus ergibt sich eine Testmenge von 163.840 Ticks (2^9 *Zustaende* $*$ 2^5 *Eingabenbelegungen* $*$ 10 *Messung*), da keine Informationen über die gültigen Zustande des SCCharts vorlie-

gen. Die gesamte Durchführung der Messung dauerte 7 Minuten und 15 Sekunden. Die Testmenge für die Messung eines SCCharts verdoppelt sich mit jeder weitere boolsche Zustandsvariable oder Eingabe. Allerdings kann man durch Informationen über die gültigen Kombination die Testmenge stark einschränken. In der Tickfunktion des FunParcs sind 50 TPPs enthalten und einige dieser TPPs sind nur in bestimmten Zuständen erreichbar. Das erkennt man daren, dass für alle zehn Messticks dieser Konfiguration der entsprechende TPP den Wert 0 hat.

Kapitel 7

Fazit

7.1 Zusammenfassung

In dieser Arbeit wurde eine Methode entwickelt, um die Laufzeit von Ticks eines SCCharts zu bestimmen. Die Methode sollte einen geringen Aufwand zur Durchführung einer Messung haben. Für eine Messung wird automatisch aus dem SCChart ein Programm generiert und auf der Zielplattform, einem der LEGO Mindstorms ausgeführt. Im Laufe der Arbeit wurden verschiedene Mindstorms und Betriebssysteme betrachtet und beurteilt wie gut sie sich für eine Laufzeitmessung eignen. Der EV3 mit einem Debian-Linux erfüllt alle Vorraussetzung, die für die Arbeit vorgegeben waren. Für komplexe Systeme können, ohne das SCChart zu verändern, im Programm Aufrufe ersetzt werden. Dadurch benötigt man nur ein USB-Kabel und den Brick des EV3s mit einer SD-Karte, um die Laufzeitmessung durchzuführen. Um die Zeitmessung durchzuführen wird ein *general purpose timer* verwendet, welcher schon vorkonfiguriert ist. Diese Methode wurde dann anschließend in KIELER implementiert. Dabei wurden die vorhandenen Strukturen erweitert und deren Anpassung beschrieben. KIELER ist damit in der Lage, die Laufzeit eines SCCharts auf einem LEGO Mindstorms EV3 zu messen.

7.2 Ausblick

Für eine Messung benötigt man die gültigen Zustände des SCCharts. Der Zustand eines SCCharts wird in Variablen gespeichert. Welche Belegungen dieser Variablen im SCChart erreichbar sind, kann bisher nicht automatisch generiert werden. Um die Messung dahingehend automatisieren zu können, wird ein Algorithmus benötigt, der diese Kombinationen erzeugen kann. Ein breites Feld, dass ebenfalls nicht in dieser Arbeit abgedeckt wird, ist die Weiterverarbeitung der Messdaten. Aus den Messergebnissen könnte man den längsten Tick bestimmen und die anteiligen Laufzeiten dieses Ticks berechnen, um die Laufzeitmessung beispielsweise für das *Hotspot Highlighting* einzusetzen. Eine andere Verwendungsmöglichkeit der Laufzeitmessung wäre es, einen Vergleich zwischen verschiedenen Methoden zur Laufzeitanalyse zu erstellen, sofern weitere Techniken vorhanden sind. Man kann die Messdaten auch dafür verwenden, die Laufzeit verschiedener Prozessoren zu vergleichen.

Anhang A

Robot

```
scchart robot {
  input bool bumper;
  input bool accelerator;
  output bool motor;

  region Main:

  initial state Init
  --> Emergency immediate with bumper
  --> Normal immediate;

  state Emergency {
    region HandleEmergency:
    initial state StartE
    --> DoneE with / 'errorLog()'; motor = false;
    final state DoneE;
  }
  >-> Done;

  state Normal {

    region HandleMotor:

    initial state Start
    --> Stop with accelerator / 'getImage()'; motor = true
    --> Stop with / 'writeLog()'; motor = false;

    final state Stop;

   }
   >-> Done with / 'writeLog(fin)';

   final state Done;
}
```

Quellcodeverzeichnis A.1. SCCharttextual des Robots

```
Function tick
InitFunction reset
State _GO
State PRE_g1
State PRE_g4
HighestTPPNumber 5
GlobalVar bumper 0..1
GlobalVar accelerator 0..1

FunctionWCET errorLog 800
FunctionWCET writeLog 1800
FunctionWCET getImage 2000

Combination
_GO 1
PRE_g1 1
PRE_g4 0

Combination
_GO 1
PRE_g1 0
PRE_g4 1

Combination
_GO 0
PRE_g1 0
PRE_g4 0

FWCET entry 1
FWCET 1 2
FWCET 2 3
FWCET 3 4
FWCET 4 5
FWCET 5 exit
WCP entry exit
```

Quellcodeverzeichnis A.2. Timing Analysis Datei des Robots

Mainfunktion

```
#include "robot.c"
#include "robot.h"

#include <time.h>
#include <sys/time.h>
#include <stdlib.h>
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <unistd.h>

unsigned long *systemTimer = 0;
int run = 1;
int main(){



   //Address to timing register in Memory
   off_t offset = 0x1c20010;
   size_t len = 8;
   size_t pagesize = sysconf(_SC_PAGE_SIZE);
   off_t page_base = (offset / pagesize) * pagesize;
   off_t page_offset = offset - page_base;

   int fd = open("/dev/mem", O_SYNC);
   unsigned char *mem = mmap(NULL, page_offset + len, PROT_READ , MAP_SHARED, fd,
       page_base);

   if (mem == MAP_FAILED) {
      perror("Can't map memory");
      return -1;
   }

   //Pointer to the 32Bit timing register
   systemTimer = mem + page_offset + 4;

   //write header of csv
   FILE *fp;
   fp = fopen("./result.csv", "w+");
   fprintf(fp, "SetNr, bumper, accelerator, TPP(start), TPP(1), TPP(2), TPP(3), TPP(4),
       TPP(5), TPP(end), errorLog_timing_2, writeLog_timing_3, writeLog_timing_4,
       getImage_timing_3\n");
```

```
41    fclose( fp );
42
43    char buf[108];
44    int bufNeed = 0;
45    int currBufIndex = 0;
46    int stateLoopcount;
47    int bumperLoopcount;
48    int bumperArr[2] = {0, 1};
49    int acceleratorLoopcount;
50    int acceleratorArr[2] = {0, 1};
51    int rerun = 0;
52
53    for(stateLoopcount = 0; stateLoopcount < 3; stateLoopcount++){
54
55       for(bumperLoopcount = 0; bumperLoopcount < 2; bumperLoopcount++) {
56
57          bumper = bumperArr[bumperLoopcount];
58
59          for(acceleratorLoopcount = 0; acceleratorLoopcount < 2; acceleratorLoopcount++) {
60
61             accelerator = acceleratorArr[acceleratorLoopcount];
62             rerun = 10;
63
64             while(rerun){
65
66                reset();
67                if(stateLoopcount == 0){
68                   set0();
69                }
70
71                if(stateLoopcount == 1){
72                   set1();
73                }
74
75                if(stateLoopcount == 2){
76                   set2();
77                }
78
79
80
81                tick();
82
83                bufNeed = sprintf (&buf[currBufIndex], "\%u", stateLoopcount);
84                currBufIndex = currBufIndex + bufNeed;
85                buf[currBufIndex] = ',';
```

```
86                currBufIndex ++;
87                bufNeed = sprintf (&buf[currBufIndex], "\%d", bumperLoopcount);
88                currBufIndex = currBufIndex + bufNeed;
89                buf[currBufIndex] = ',';
90                currBufIndex ++;
91                bufNeed = sprintf (&buf[currBufIndex], "\%d", acceleratorLoopcount);
92                currBufIndex = currBufIndex + bufNeed;
93                buf[currBufIndex] = ',';
94                currBufIndex ++;
95                bufNeed = sprintf (&buf[currBufIndex],"\%lu",timingTPPStart);
96                currBufIndex = currBufIndex + bufNeed;
97                buf[currBufIndex] = ',';
98                currBufIndex ++;
99                bufNeed = sprintf (&buf[currBufIndex],"\%lu",timingTPP1);
100               currBufIndex = currBufIndex + bufNeed;
101               buf[currBufIndex] = ',';
102               currBufIndex ++;
103               bufNeed = sprintf (&buf[currBufIndex],"\%lu",timingTPP2);
104               currBufIndex = currBufIndex + bufNeed;
105               buf[currBufIndex] = ',';
106               currBufIndex ++;
107               bufNeed = sprintf (&buf[currBufIndex],"\%lu",timingTPP3);
108               currBufIndex = currBufIndex + bufNeed;
109               buf[currBufIndex] = ',';
110               currBufIndex ++;
111               bufNeed = sprintf (&buf[currBufIndex],"\%lu",timingTPP4);
112               currBufIndex = currBufIndex + bufNeed;
113               buf[currBufIndex] = ',';
114               currBufIndex ++;
115               bufNeed = sprintf (&buf[currBufIndex],"\%lu",timingTPP5);
116               currBufIndex = currBufIndex + bufNeed;
117               buf[currBufIndex] = ',';
118               currBufIndex ++;
119               bufNeed = sprintf (&buf[currBufIndex],"\%lu",timingTPPEnd);
120               currBufIndex = currBufIndex + bufNeed;
121               buf[currBufIndex] = ',';
122               currBufIndex ++;
123               bufNeed = sprintf (&buf[currBufIndex], "\%u", errorLog_timing_2);
124               currBufIndex = currBufIndex + bufNeed;
125               buf[currBufIndex] = ',';
126               currBufIndex ++;
127               bufNeed = sprintf (&buf[currBufIndex], "\%u", writeLog_timing_3);
128               currBufIndex = currBufIndex + bufNeed;
129               buf[currBufIndex] = ',';
130               currBufIndex ++;
```

```
131                 bufNeed = sprintf (&buf[currBufIndex], "\%u", writeLog_timing_4);
132                 currBufIndex = currBufIndex + bufNeed;
133                 buf[currBufIndex] = ',';
134                 currBufIndex ++;
135                 bufNeed = sprintf (&buf[currBufIndex], "\%u", getImage_timing_3);
136                 currBufIndex = currBufIndex + bufNeed;
137                 buf[currBufIndex] = '\n';
138                 currBufIndex ++;
139 
140                 errorLog_timing_2 = 0;
141                 writeLog_timing_3 = 0;
142                 writeLog_timing_4 = 0;
143                 getImage_timing_3 = 0;
144                 rerun--;
145 
146                 buf[currBufIndex] = '\0';
147                 fp = fopen("./result.csv", "a");
148                 fputs(buf , fp);
149                 fclose( fp );
150                 currBufIndex = 0;
151 
152             }
153 
154 
155         }
156       }
157    }
158    return 0;
159 }
```

Quellcodeverzeichnis A.3. Main-Datei des Robots

Tickfunktion

```
/* Im edit the generated TickFunction */
/* It now ready to timing */
#include "robot.h"


#define TPP(LABEL) asm volatile("" ::: "memory"); timingTPP##LABEL = *systemTimer; asm volatile("" ::: "memory")

/*****************************************************************************/
/* G E N E R A T E D C C O D E */
/*****************************************************************************/
/* KIELER - Kiel Integrated Environment for Layout Eclipse RichClient */
/* */
/* http://www.informatik.uni-kiel.de/rtsys/kieler/ */
/* Copyright 2014 by */
/* + Kiel University */
/* + Department of Computer Science */
/* + Real-Time and Embedded Systems Group */
/* */
/* This code is provided under the terms of the Eclipse Public License (EPL).*/
/*****************************************************************************/
char bumper;
char accelerator;
char motor;

unsigned long timingTPPStart, timingTPPEnd, timingTPP1, timingTPP2, timingTPP3, timingTPP4, timingTPP5;
int errorLog_timing_2 = 0;
int writeLog_timing_3 = 0;
int writeLog_timing_4 = 0;
int getImage_timing_3 = 0;

char _GO;
char g0;
char g1;
char PRE_g1;
char g2;
char g3;
char g4;
char PRE_g4;
char g5;
char g6;
char g7;
```

```
42 char g8;
43
44 void set0(){
45 _GO = 1;
46 PRE_g1 = 1;
47 PRE_g4 = 0;
48 }
49 void set1(){
50 _GO = 1;
51 PRE_g1 = 0;
52 PRE_g4 = 1;
53 }
54 void set2(){
55 _GO = 0;
56 PRE_g1 = 0;
57 PRE_g4 = 0;
58 }
59
60 void reset(){
61   _GO = 1;
62   PRE_g1 = 0;
63   PRE_g4 = 0;
64   return;
65 }
66 void tick(){
67 TPP(Start);
68
69   {
70     g0 = _GO;
71     TPP(1);
72     g1 =(_GO&&bumper);
73     g2 =(PRE_g1);
74     if(g2){
75       asm volatile("" ::: "memory"); errorLog_timing_2++; asm volatile("" ::: "memory");
76       motor = 0;
77     }
78     TPP(2);
79     g5 =(PRE_g4);
80     g6 =(g5&&accelerator);
81     if(g6){
82       asm volatile("" ::: "memory"); getImage_timing_3++; asm volatile("" ::: "memory");
83       motor = 1;
84     }
85     g8 =(g5&&(!(accelerator)));
86     if(g8){
```

```
87        asm volatile("" ::: "memory"); writeLog_timing_3++; asm volatile("" ::: "memory");
88        motor = 0;
89      }
90      TPP(3);
91      g7 =(g6||g8);
92      if(g7){
93        asm volatile("" ::: "memory"); writeLog_timing_4++; asm volatile("" ::: "memory");
94      }
95      g3 =(g2||g7);
96      TPP(4);
97      g4 =(_GO&&(!(bumper)));
98      TPP(5);
99    }
100   PRE_g1 = g1;
101   PRE_g4 = g4;
102   _GO = 0;
103
104 TPP(End);
105 return;
106 }
```

Quellcodeverzeichnis A.4. Tick-Datei des Robots

Tabelle A.1. Messergebnisse robot

SetNr	bumper	accelerator	TPP(start)	TPP(1)	TPP(2)	TPP(3)	TPP(4)	TPP(5)	TPP(end)	errorLog _timing _2	writeLog _timing _3	writeLog _timing _4	getImage _timing _3
0	0	0	393994977	393995003	393995044	393995104	393995152	393995178	393995198	1	0	0	0
0	0	0	394038645	394038671	394038708	394038771	394038813	394038839	394038859	1	0	0	0
0	0	0	394203134	394203160	394203197	394203257	394203305	394203330	394203350	1	0	0	0
0	0	0	394249288	394249309	394249346	394249408	394249455	394249481	394249501	1	0	0	0
0	0	0	394271907	394271922	394271959	394272016	394272059	394272085	394272105	1	0	0	0
0	0	0	394293959	394293974	394294011	394294072	394294115	394294141	394294161	1	0	0	0
0	0	0	394316405	394316420	394316457	394316514	394316561	394316586	394316606	1	0	0	0
0	0	0	394341082	394341097	394341134	394341191	394341234	394341260	394341280	1	0	0	0
0	0	0	394365743	394365758	394365795	394365856	394365903	394365928	394365948	1	0	0	0
0	0	0	394393465	394393480	394393517	394393579	394393627	394393653	394393673	1	0	0	0
0	0	1	394418326	394418341	394418378	394418435	394418478	394418504	394418527	1	0	0	0
0	0	1	394443188	394443203	394443240	394443297	394443345	394443371	394443391	1	0	0	0
0	0	1	394489831	394489851	394489888	394489949	394489992	394490018	394490038	1	0	0	0
0	0	1	394512608	394512627	394512668	394512725	394512768	394512794	394512814	1	0	0	0
0	0	1	394535501	394535516	394535553	394535610	394535653	394535679	394535699	1	0	0	0
0	0	1	394557732	394557747	394557784	394557841	394557889	394557915	394557935	1	0	0	0
0	0	1	394582974	394582989	394583026	394583083	394583126	394583151	394583171	1	0	0	0
0	0	1	394607519	394607534	394607571	394607628	394607671	394607697	394607717	1	0	0	0
0	0	1	394655973	394655999	394656041	394656101	394656148	394656173	394656193	1	0	0	0
0	0	1	394681478	394681493	394681533	394681589	394681636	394681662	394681682	1	0	0	0
0	1	0	394727567	394727587	394727632	394727688	394727731	394727756	394727776	1	0	0	0
0	1	0	394750272	394750287	394750324	394750381	394750424	394750450	394750470	1	0	0	0
0	1	0	394773621	394773636	394773677	394773737	394773784	394773810	394773830	1	0	0	0
0	1	0	394796224	394796239	394796276	394796333	394796376	394796402	394796426	1	0	0	0
0	1	0	394821934	394821949	394821986	394822048	394822099	394822125	394822145	1	0	0	0
0	1	0	394846927	394846942	394846979	394847041	394847083	394847109	394847129	1	0	0	0
0	1	0	394875270	394875285	394875322	394875379	394875426	394875452	394875472	1	0	0	0
0	1	0	394900227	394900242	394900279	394900336	394900384	394900410	394900430	1	0	0	0
0	1	0	394925798	394925813	394925850	394925907	394925954	394925980	394926000	1	0	0	0
0	1	0	394971715	394971741	394971778	394971839	394971882	394971908	394971928	1	0	0	0
0	1	1	394995100	394995120	394995157	394995217	394995259	394995285	394995305	1	0	0	0
0	1	1	395017244	395017259	395017296	395017353	395017401	395017426	395017446	1	0	0	0
0	1	1	395039850	395039865	395039902	395039959	395040006	395040032	395040052	1	0	0	0
0	1	1	395064900	395064920	395064957	395065014	395065061	395065087	395065107	1	0	0	0
0	1	1	395092902	395092917	395092956	395093013	395093056	395093082	395093102	1	0	0	0
0	1	1	395117943	395117958	395117995	395118052	395118103	395118129	395118149	1	0	0	0
0	1	1	395143581	395143596	395143633	395143690	395143738	395143768	395143788	1	0	0	0
0	1	1	395168426	395168441	395168478	395168538	395170736	395170764	395170784	1	0	0	0
0	1	1	395215291	395215318	395215359	395215421	395215468	395215494	395215514	1	0	0	0
0	1	1	395237823	395237838	395237879	395237935	395237977	395238003	395238023	1	0	0	0
1	0	0	395260892	395260911	395260947	395261001	395261044	395261070	395261090	0	1	1	0
1	0	0	395283190	395283205	395283244	395283298	395283346	395283372	395283392	0	1	1	0
1	0	0	395308684	395308699	395308735	395308795	395308838	395308863	395308883	0	1	1	0
1	0	0	395335678	395335693	395335729	395335787	395335835	395335861	395335881	0	1	1	0
1	0	0	395361132	395361147	395361183	395361239	395361287	395361312	395361332	0	1	1	0
1	0	0	395386036	395386051	395386087	395386141	395386184	395386210	395386230	0	1	1	0
1	0	0	395432459	395432479	395432515	395432574	395432621	395432646	395432666	0	1	1	0

Tabelle A.1. Messergebnisse robot

SetNr	bumper	accelerator	TPP(start)	TPP(1)	TPP(2)	TPP(3)	TPP(4)	TPP(5)	TPP(end)	errorLog _timing _2	writeLog _timing _3	writeLog _timing _4	getImage _timing _3
1	0	0	395463353	395463368	395463408	395463462	395463505	395463530	395463550	0	1	1	0
1	0	0	395486151	395486166	395486202	395486258	395486301	395486326	395486346	0	1	1	0
1	0	0	395507656	395507671	395507707	395507762	395507812	395507838	395507858	0	1	1	0
1	0	1	395529217	395529232	395529268	395529323	395529374	395529400	395529420	0	0	1	1
1	0	1	395554405	395554420	395554459	395556617	395556660	395556686	395556706	0	0	1	1
1	0	1	395580820	395580835	395580871	395580931	395580974	395580999	395581019	0	0	1	1
1	0	1	395605325	395605340	395605376	395605431	395605474	395605500	395605520	0	0	1	1
1	0	1	395632019	395632034	395632079	395632138	395632181	395632207	395632227	0	0	1	1
1	0	1	395675455	395675475	395675515	395675570	395675613	395675639	395675659	0	0	1	1
1	0	1	395699180	395699195	395699231	395699290	395699333	395699359	395699379	0	0	1	1
1	0	1	395721892	395721907	395721947	395722002	395722049	395722075	395722095	0	0	1	1
1	0	1	395743784	395743799	395743835	395743890	395743933	395743961	395743981	0	0	1	1
1	0	1	395766148	395766163	395766199	395766254	395766302	395766328	395766348	0	0	1	1
1	1	0	395790547	395790562	395790598	395790656	395790699	395790725	395790745	0	1	1	0
1	1	0	395817955	395817970	395818006	395818061	395818108	395818133	395818153	0	1	1	0
1	1	0	395859570	395859596	395859638	395859696	395859743	395859769	395859789	0	1	1	0
1	1	0	395885309	395885329	395885365	395885419	395885466	395885492	395885512	0	1	1	0
1	1	0	395930834	395930854	395930890	395930948	395930991	395931017	395931037	0	1	1	0
1	1	0	395953589	395953604	395953640	395953694	395953745	395953771	395953791	0	1	1	0
1	1	0	395975255	395975274	395975310	395975364	395975412	395975437	395975457	0	1	1	0
1	1	0	395997456	395997471	395997507	395997561	395997604	395997629	395997649	0	1	1	0
1	1	0	396021934	396021949	396021985	396022040	396022083	396022108	396022132	0	1	1	0
1	1	0	396047021	396047036	396047072	396047127	396047170	396047195	396047215	0	1	1	0
1	1	1	396074080	396074095	396074131	396074191	396074241	396074267	396074287	0	0	1	1
1	1	1	396116365	396116385	396116421	396116480	396116527	396116553	396116573	0	0	1	1
1	1	1	396159897	396159917	396159953	396160013	396160060	396160088	396160108	0	0	1	1
1	1	1	396184123	396184138	396184174	396184234	396184281	396184306	396184326	0	0	1	1
1	1	1	396205808	396205823	396205859	396205914	396205957	396205982	396206002	0	0	1	1
1	1	1	396228174	396228189	396228225	396228280	396228323	396228349	396228369	0	0	1	1
1	1	1	396249615	396249630	396249666	396249723	396249766	396249792	396249812	0	0	1	1
1	1	1	396276631	396276646	396276686	396276741	396276784	396276810	396276830	0	0	1	1
1	1	1	396300867	396300882	396300918	396300972	396301015	396301041	396301061	0	0	1	1
1	1	1	396325636	396325651	396325687	396325742	396325792	396325818	396325838	0	0	1	1
2	0	0	396352573	396352588	396352624	396352680	396352730	396352756	396352776	0	0	0	0
2	0	0	396396001	396396021	396396057	396396118	396396168	396396194	396396214	0	0	0	0
2	0	0	396420161	396420176	396420212	396420273	396420319	396420344	396420364	0	0	0	0
2	0	0	396442671	396442686	396442722	396442779	396442825	396442851	396442871	0	0	0	0
2	0	0	396464440	396464455	396464491	396464548	396464594	396464620	396464640	0	0	0	0
2	0	0	396486698	396486713	396486749	396486806	396486851	396486876	396486896	0	0	0	0
2	0	0	396510921	396510936	396510972	396511038	396511083	396511109	396511129	0	0	0	0
2	0	0	396537736	396537751	396537787	396537844	396537894	396537920	396537940	0	0	0	0
2	0	0	396561774	396561790	396561826	396561887	396561937	396561963	396561987	0	0	0	0
2	0	0	396586574	396586589	396586625	396588718	396588768	396588794	396588814	0	0	0	0
2	0	1	396631584	396631599	396631637	396631698	396631744	396631770	396631790	0	0	0	0
2	0	1	396656567	396656582	396656618	396656677	396656727	396656753	396656773	0	0	0	0
2	0	1	396678740	396678758	396678795	396678856	396678902	396678928	396678948	0	0	0	0
2	0	1	396701380	396701395	396701431	396701488	396701534	396701560	396701580	0	0	0	0
2	0	1	396723424	396723440	396723476	396723533	396723578	396723604	396723624	0	0	0	0
2	0	1	396748665	396748680	396748716	396748777	396748823	396748849	396748869	0	0	0	0
2	0	1	396781785	396781800	396781836	396781896	396781942	396781967	396781987	0	0	0	0
2	0	1	396809594	396809609	396809645	396809702	396809748	396809773	396809793	0	0	0	0
2	0	1	396833724	396833739	396833779	396833835	396833880	396833906	396833926	0	0	0	0

Tabelle A.1. Messergebnisse robot

SetNr	bumper	accelerator	TPP(start)	TPP(1)	TPP(2)	TPP(3)	TPP(4)	TPP(5)	TPP(end)	errorLog _timing _2	writeLog _timing _3	writeLog _timing _4	getImage _timing _3
2	0	1	396877140	396877160	396877196	396877253	396877299	396877324	396877344	0	0	0	0
2	1	0	396900900	396900920	396900956	396901013	396901059	396901084	396901104	0	0	0	0
2	1	0	396923169	396923189	396923225	396923282	396923328	396923354	396923376	0	0	0	0
2	1	0	396944770	396944785	396944821	396944878	396944924	396944950	396944970	0	0	0	0
2	1	0	396967213	396967228	396967264	396967320	396967366	396967392	396967412	0	0	0	0
2	1	0	396992072	396992092	396992128	396992184	396992230	396992255	396992275	0	0	0	0
2	1	0	397019896	397019911	397019947	397020004	397020053	397020079	397020099	0	0	0	0
2	1	0	397061551	397061577	397061615	397061676	397061730	397061756	397061776	0	0	0	0
2	1	0	397090595	397090615	397090651	397090711	397090763	397090789	397090809	0	0	0	0
2	1	0	397134552	397134572	397134611	397134674	397134720	397134746	397134766	0	0	0	0
2	1	0	397157050	397157065	397157101	397157158	397157208	397157234	397157254	0	0	0	0
2	1	1	397178692	397178708	397178744	397178801	397178847	397178875	397178895	0	0	0	0
2	1	1	397200864	397200879	397200915	397200972	397201018	397201044	397201064	0	0	0	0
2	1	1	397224924	397224939	397224975	397225032	397225078	397225103	397225123	0	0	0	0
2	1	1	397251753	397251769	397251805	397251861	397251911	397251937	397251957	0	0	0	0
2	1	1	397276345	397276360	397276396	397276453	397276499	397276527	397276547	0	0	0	0
2	1	1	397301288	397301308	397301344	397301404	397301457	397301483	397301503	0	0	0	0
2	1	1	397325686	397325701	397325737	397325794	397325840	397325866	397325886	0	0	0	0
2	1	1	397372050	397372065	397372103	397372164	397372210	397372236	397372256	0	0	0	0
2	1	1	397394284	397394299	397394335	397394396	397394444	397394470	397394490	0	0	0	0
2	1	1	397416671	397416687	397416723	397416785	397416830	397416856	397416876	0	0	0	0

Literatur

[BCR+10] Clément Ballabriga, Hugues Cassé, Christine Rochange und Pascal Sainrat. „Otawa: an open toolbox for adaptive wcet analysis“. In: *Software Technologies for Embedded and Ubiquitous Systems*. Springer, 2010, S. 35–46.

[BPZ11] Peter Bradley Valdenebro, Juan Antonio de la Puente Alfaro und Juan Rafael Zamorano Flores. „Ada user guide for lego mindstorms nxt“. In: *Ada User Journal* 32.3 (2011), S. 194–203.

[FBS+14] Insa Fuhrmann, David Broman, Steven Smyth und Reinhard von Hanxleden. *Towards interactive timing analysis for designing reactive systems*. Reconciling Performance and Predictability (RePP'14), satellite event of ETAPS'14. Also as technical report: EECS Department, University of California, Berkeley, UCB/EECS-2014-26, April 2014. Apr. 2014.

[HDM+14] Reinhard von Hanxleden, Björn Duderstadt, Christian Motika, Steven Smyth, Michael Mendler, Joaquín Aguado, Stephen Mercer und Owen O'Brien. „SCCharts: Sequentially Constructive Statecharts for safety-critical applications“. In: *Proc. ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'14)*. Long version: Technical Report 1311, Christian-Albrechts-Universität zu Kiel, Department of Computer Science, December 2013, ISSN 2192-6274. Edinburgh, UK: ACM, Juni 2014.

[MSH14] Christian Motika, Steven Smyth und Reinhard von Hanxleden. „Compiling sccharts - a case-study on interactive model-based compilation“. In: *Leveraging Applications of Formal Methods, Verification and Validation. Technologies for Mastering Change*. Springer, 2014, S. 461–480.

[Sch14] Alexander Schulz-Rosengarten. „Framework zum tracing von emf-modelltransformationen“. Bachelor Thesis. Kiel University, Maerz 2014.

[Sta15] Andreas Achim Stange. „Comfortable sccharts modeling for embedded systems“. Bachelor Thesis. Kiel University, Okt. 2015.

[WEE+] Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann, Tulika Mitra u. a. „The worst-case execution time problem - overview of methods and survey of tools“. In: ().