Testen von Echtzeitsystemen

Mathias Ehret

Hochschule Offenburg
Angewandte Informatik
Badstraße 24, 77625 Offenburg
mehret@stud.fh-offenburg.de

ABSTRACT Das Testen ist ein sehr teurer Schritt in der Entwicklung von Real-Time Systems. Während für reine Softwaresysteme mit moderaten Anforderungen an das Laufzeitverhalten schon länger moderne Testverfahren und Paradigmen angewandt werden, sind deren Varianten für das Testen von Echtzeitsystemen noch relativ unbekannt.

Echtzeitsysteme können günstig als Timed Automata beschrieben und als Black-Box Systeme nach digitaltechnischen und formalen Gesichtspunkten verifiziert werden. Weitere Möglichkeiten ergeben sich durch adaptive Testgenerierung und Robustheitsprüfungen. Für die Spezifikation der Testfälle existiert eine Echtzeiterweiterung der TTCN Notation.

0 Einführung

Das Testen von Software-basierten Systemen beansprucht typischer Weise mehr als ein Drittel der für ein Projekt zur Verfügung gestellten Ressourcen. Dennoch sind oft unüberlegte und heuristisch gestützte Vorgehensweisen hoher Fehleranfälligkeit zu beobachten. Besonders aufwändig gestaltet sich die Testphase wenn nicht nur verifiziert werden muss, dass alle Eingaben zu den korrekten Ausgaben führen, sondern wenn auch für die bis dahin verstrichene Zeit Vorgaben bestehen. Man spricht hier von Echtzeitsystemen. [Lar04]

Das typische Setup eines Echtzeitsystems beinhaltet ein Programm, das Eingaben und Daten von angeschlossenen Geräten liest, diese berechnet und dann Ausgaben und Steuersignale an eine entsprechende Peripherie zurückgibt. Im einfachsten Fall geschieht dies in festen, unveränderlichen Intervallen. Die Eingaben können dabei in Form von Interrupts, also asynchron erfolgen oder durch ein periodisches Abfragen (Polling) der Peripherie zyklisch eingelesen werden. Die Zeit, die zwischen dem Eintreffen des Signals und dem Ende seiner Verarbeitung verstreicht, wird als Antwortzeit des Systems bezeichnet.

Um sichere Aussagen über dieses Zeitintervall zu treffen, reicht es aber nicht aus, mit statistischen Methoden Durchschnitts- oder Maximalwerte aus einem Set von tatsächlich durchgeführten Zeitmessungen zu berechnen, da das Verhalten des Systems unter moderater Auslastung nicht auf das Verhalten des Systems bei maximaler Last und damit nicht auf die maximale Antwortzeit (**W**orst **C**ase **E**xecution **T**ime) schließen lässt. [Jos86]

In dieser Ausarbeitung sollen nun verschiedene Testansätze für die genannte Problematik vorgestellt werden. Ziel ist es, einen Überblick über das Testen von Echtzeitsystemen zu erhalten.

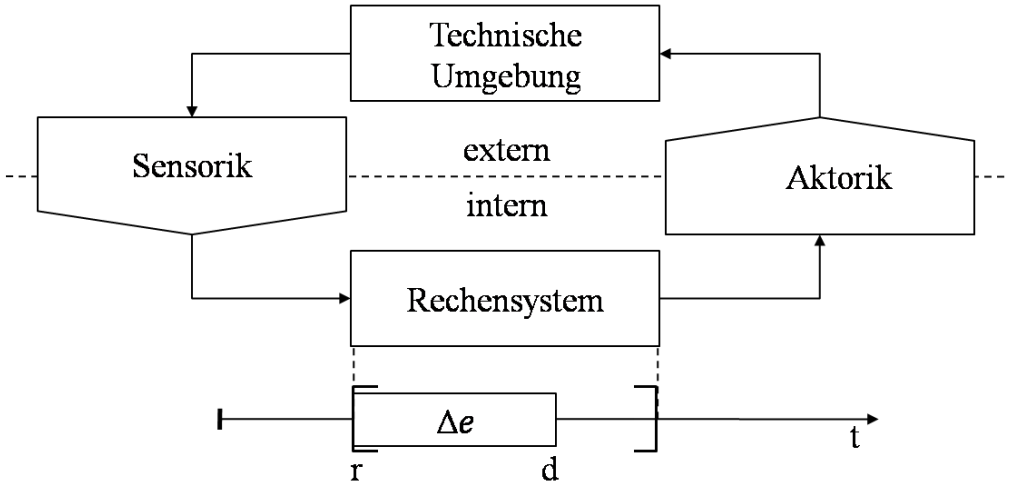

Abbildung 1: Grundmodell eines Echtzeitsystems; nach [Zöb08]

1 Terminologie

Hier sollen die wichtigsten Begriffe und Formalismen für das Beschreiben und Testen von Echtzeitsystemen erläutert werden.

1.1 Harte und weiche Echtzeit

Man unterscheidet zwischen harter und weicher Echtzeit, wobei die Definition der weichen Echtzeit weniger strenge Restriktionen an das Antwortverhalten eines Systems stellt als die der harten Echtzeit. Formal lässt sich Folgendes über eine harte Zeitbedingung A aussagen:

$$A \equiv r + \Delta e \leq d$$

r stellt den Startzeitpunkt dar, d die Deadline und Δe die von der Ausführung der Aufgabe in Anspruch genommene Zeitspanne. Ein hartes Echtzeitsystem zeichnet sich dadurch aus, dass die Wahrscheinlichkeit die Zeitbedingung A einzuhalten 1 ist:

$$P(A) = 1$$

Wie bereits erwähnt, sollen Systeme hier nicht statistisch untersucht, sondern ihr tatsächliches Echtzeitverhalten getestet werden. Daher ist in dieser Ausarbeitung immer von harter Echtzeit die Rede.

1.2 Modellierung

Modelliert man ein typisches Echtzeitsystem, so kann als Basis die Verknüpfung einer technischen Umgebung mit einem Rechensystem, wie in Abbildung 1 dargestellt, zu Grunde gelegt werden. Der Datenfluss von der Umgebung zum Rechensystem wird über eine Sensorik realisiert, die Steuersignale werden über eine Aktorik an die Umgebung umgesetzt.

Sowohl Sensorik als auch Aktorik arbeiten mit großen Datenmengen, während die eigentliche Berechnung aus Gründen der Effizienz auf dem kleinstmöglichen Datenumfang erfolgt. Datenreduktion und Datenexpansion kompensieren diese Differenzen. Abbildung 2 zeigt die eigentliche Berechnung (**C**ontrol **A**ction) im Schnittbereich beider Vorgänge. [Zöb08]

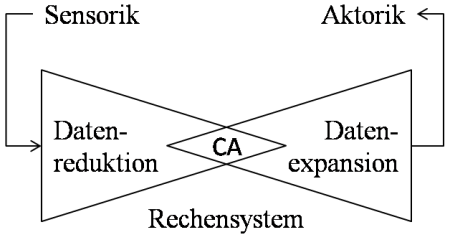

Abbildung 2: Steuerfunktion (CA), Reduktion und Expansion; nach [Zöb08]

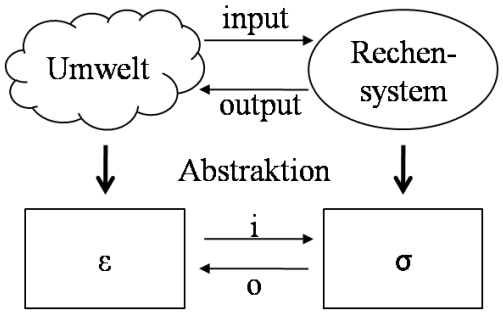

Abbildung 3a: Abstraktion eines RTS nach [Lar04]

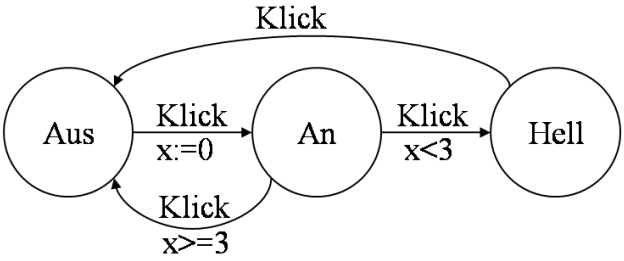

Abbildung3b: Beispiel „Doppelklicktaster“ nach [Hof06]

1.3 Abstraktion des Grundmodells

Um modellbasierte Testweisen zu ermöglichen, muss zunächst eine formale Abstraktion des zu testenden Rechensystems und seiner Umgebung vorgenommen werden. Wie in Abbildung 3a verdeutlicht, werden hierzu die Verhaltensweise der Umgebung und die Arbeitsweise des Rechensystems durch die Verhaltensmodelle ε und σ beschrieben, die über abstrakte Signale i und o miteinander interagieren. [Lar04]

Um diese Verhaltensmodelle zu beschreiben, eignen sich Timed Automata hervorragend, da sie auch im Hinblick auf die Verifikation von Echtzeitsystemen gute Chancen bieten. [Hof06]

1.4 Timed Automata

Timed Automata sind als Erweiterung der endlichen Automaten um eine Zeitkomponente zu verstehen. Zusätzlich zu den bereits bekannten States, Actions und Transitions wird ein Uhrenmodell aus einer Menge innerhalb des Automaten synchron laufender Uhren hinzugefügt. Jede Uhr wird durch reelle, positive Zahlen beschrieben, kann auf 0 gesetzt werden und dient als Eingang für Clock constraints. Man unterscheidet dabei zwischen Clock Constraints an Transitionen (Guards) und Clock Constraints innerhalb von Zuständen (Invarianten). Um zu beurteilen ob ein Statusübergang erfolgt, reicht es nun nicht mehr aus, nur die Aktionen zu betrachten. Nur wenn der Guard der Transition sowie die Invarianten des aktuellen und folgenden Zustands erfüllt sind, erfolgt ein Zustandsübergang. Auch die Betrachtung der Transitionen muss angepasst werden. Es wird zwischen diskreten Übergängen (Events) und ΔT-Übergängen (Delays) unterschieden. Beim Event erfolgt der Übergang von einem Zustand in den Anderen, ohne dass Zeit verstreicht, wohingegen beim Delay Zeit vergeht, ohne dass der Zustand sich ändert. [Hof06]

Als Beispiel wollen wir hier einen Taster betrachten, der eine Lampe schalten soll. Auf eine einfache Betätigung soll die Lampe ein und ausgeschaltet werden; ein „Doppelklicken“ auf den Taster bei ausgeschalteter Lampe soll eine hellere Lichtstufe einschalten. Diese Aufgabe ist durch ihre Zeitkomponente nur in einem Timed Automaton (siehe Abbildung 3b) durch den Einsatz zweier Guards zu lösen. Der erste Switch setzt die Uhr auf 0, der Automat wechselt nach ON; beim zweiten Switch entscheiden die Guards darüber, ob es sich um die zweite Hälfte eines Doppelklicks handelt (es sind ≤ 3 Zeiteinheiten vergangen) oder ob es sich um einen erneuten einzelnen Switch handelt. [Hof06]

2. RTS Black-Box Testing

Da es nahezu unmöglich ist echtzeitfähige Software-Systeme auf Instruction Level anhand ihres Quelltextes zu verifizieren, muss das Hauptaugenmerk beim Testen von Echtzeitsystemen auf Online Black-Box Testverfahren liegen. [Lan05]

2.1 Betrachtung des Timing-Verhaltens

Betrachtet man ein Echtzeitsystem als zwischen einem Sender und einem Empfänger geschaltetes einfaches System (Device), so bleiben zunächst nur die aus der Nachrichtentechnik bekannten Größen, um ein System nach seiner Spezifikation zu verifizieren. Das folgende Beispiel verdeutlicht einen Test auf diese Eigenschaften:

Gegeben sei ein einfacher Mikrocontroller nicht näher spezifizierter Bauart der über zwei Timer, einen **D**irect **M**emory **A**ccess Controller, zwei **D**igital **A**nalog **C**onverter und zwei I/O Ports verfügt. In der Hauptschleife des Systems wird der erste Pin des Ports0 P0.0 zunächst auf 1 gesetzt. Nach einer kurzen Wartezeit von einigen µs wird der Pin wieder auf 0 gesetzt und nach der gleichen Wartezeit beginnt die Schleife von vorne. So lässt sich der Jitter des Systems messen, da die Hauptschleife mit jeder DMA Aktion und jedem Interrupt kurz unterbrochen wird. Timer A0 erzeugt alle 25,5µs einen Trigger, der den DMA Controller anstößt. Dieser kopiert Daten (Tabelle einer Sinus Funktion) auf den DAC da0. So entsteht an da0 eine Sinuswelle. Diese Routine erhält die höchste Priorität. Außerdem erzeugt der DMA alle 16 Transfers einen Interrupt mit niedriger Priorität welcher P1.0 umschaltet. Eine **I**nterrupt **S**ervice **R**outine ist an A1 geknüpft und stößt den DMA ebenfalls alle 25,5 µs an; hier werden Daten für eine Sägezahnspannung auf da1 kopiert. [Lan05]

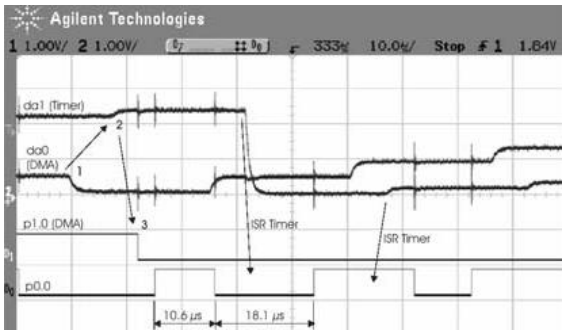

Abbildung 4: Messung mit Scope [Lan05]

Der oberste Graph auf Abbildung 4 zeigt die Spannung an da1, der folgende zeigt da0, der dritte P1.0 und der vierte P0.0. Hier lassen sich die Verzögerungen erkennen, die durch den DMA Prozess und die ISRs bedingt wurden. Die erste Verzögerung zwischen den ersten „Flanken“ auf da0 und da1 ergibt sich aus den unterschiedlichen Prioritäten des DMA und Timer Interrupts. Die zweite Verzögerung entsteht durch die Laufzeit der ISR, die den DMA Controller anstößt, die Sägezahn-Daten zu kopieren. Da die DMA-ISR erst angestoßen wird nachdem die A1-ISR abgearbeitet wurde, ergibt sich ein weiteres Delay dass sich in der überlangen Low Phase des P0.0 Pins äußert. [Lan05]

Hieraus lässt sich ein deterministisches Verhalten ableiten, mit dessen Hilfe sich alle Delays über die gesamte Laufzeit des Systems vorausbestimmen lassen. Vergleicht man diese nun mit der Spezifikation des Systems, lässt sich leicht sagen, in wie weit es seine Echtzeit-Anforderungen erfüllt. Allerdings funktioniert diese Methodik nur bei einer sehr reduzierten Sichtweise auf das DUT und nur wenn das System in einer deterministisch bestimmbaren Umgebung betrieben wird. Außerdem erfordert der Prozess des deterministischen Bestimmens der Delays eine differenzierte Kenntnis des Sourcecodes, die dem eigentlichen Gedanken des Black-Box Testens wiederspricht.

2.2 Adaptive Conformance Testing

Das Conformance Testing beschreibt formal das Testen eines Systems als auf der Implementierung von einem Agenten ausgeführte Experimente. Zunächst bedarf hierzu aber der Begriff der Timed Automata einer Erweiterung indem Actions nach Input und Output Actions unterschieden werden. Input Actions werden dabei im Graphen mit einem „?“ gekennzeichnet, Output Actions erhalten ein „!“. Wurde ein Automat entsprechend erweitert, spricht man von einem TAIO, also einem Timed Atomaton with Inputs and Outputs. [Kri04]

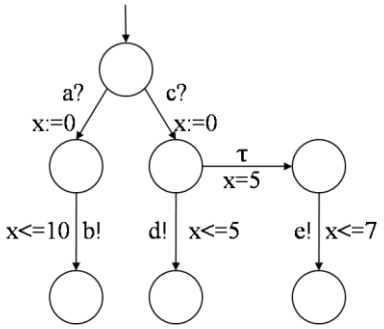

Abbildung 5: Bsp. TAIO; nach [Kri04]

Es werden zwei verschiedene Arten von Tests unterschieden:

Analog-Clock Tests messen exakt die verstrichene Zeit zwischen zwei betrachteten Aktionen. Ein Agent, der durch das Analog-Clock Verfahren testet, kann der Implementierung jederzeit einen Input liefern.

Digital-Clock Tests zählen lediglich wie viele „Ticks“ zwischen zwei Aktionen aufgetreten sind und können nur direkt nach dem Auftreten einer Aktion oder eines Ticks einen Input an die Implementierung schicken. Die Uhr ist zu Beginn des Tests 0 und ihre Periodizität beträgt 1. Außerdem erhält Tick eine höhere Priorität als Action; so ist sichergestellt, dass eine nach dem i-ten und vor dem i+1-ten Tick aufgezeichnete Action auch wirklich im Zeitintervall [i, i+1] aufgetreten ist. [Kri04]

Adaptives Testen beschreibt, dass die vom Agenten ausgeführten Actions von der bisherigen Testhistorie abhängen. Adaptive Tests können also als Bäume betrachtet werden, die die Strategie des Agenten in der Interaktion mit der Implementierung bestimmen. [Kri04]

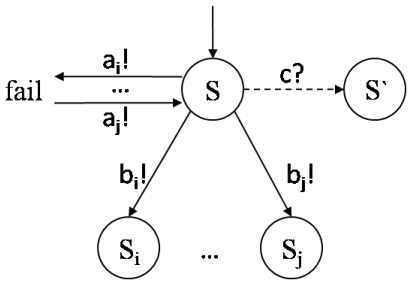

Abbildung 6: Generischer Knoten im Adaptiven Baum; nach [Kri04]

Ein Knoten in diesem Baum ist aus mehreren States zusammengesetzt und symbolisiert das Wissen, das der Agent über den augenblicklichen Teststatus hat. Der adaptive Algorithmus baut einen Baum auf, indem er Folgezustände gemäß des Schemas in Abbildung 6 an den bestehenden Knoten anhängt. Für alle illegalen Outputs a_i, a_j, …führt der Test zu einem „fail“ während die legalen Outputs b_i, b_j, … zu einem Übergang in den jeweilig spezifizierten Folgezustand S_i, S_j, … führen. Sollte der aktuelle Zustand einen Input c zulassen, so kann der Agent diesen Input liefern; der Automat geht dann in den Zustand S‘ über. Der Algorithmus kann bei jedem Knoten stoppen und diesen als „pass“ markieren. Der Algorithmus ist nicht vollständig beschrieben, da die Entscheidungen „Stopp oder Weiter“, „Warten oder Input emittieren“ und „Welchen Input emittieren“ über Benutzer-definierte Parameter, logisch oder zufällig getroffen werden. Das Treffen dieser Entscheidungen kann vor oder während der Testfall-Generierung, aber selbst während der Laufzeit der Tests (on-the-fly) getroffen werden. [Kri04]

2.3 Testen der Robustheit eines RTS

Bei dem Testen eines Systems auf seine Robustheit versucht man herauszufinden wie es auf Gefährdungen reagiert. Gefährdungen werden hier als von der Umgebung kommende, unerwartete Ereignisse definiert und als illegale Inputs bzw. fehlerhafte Clock Constraints modelliert. Hier muss das System zunächst als TAIO beschrieben sein (Nominalspezifikation) und um eine Minimalspezifikation erweitert werden. Diese Minimalspezifikation beschreibt die absolut notwendigen Minimalanforderungen die das System im Worst Case noch erfüllen muss. Da das System unter dem schweren Einfluss von Fehleingaben nicht mehr durchgängig korrekt arbeiten kann, reicht es zu verifizieren welcher Anteil der Nominalspezifikation noch erfüllt wird. Robustheit wird so zum quantifizierbaren Begriff. Die Minimalspezifikation ist als absolute Grenze zu verstehen; erfüllt das System diese Anforderungen nicht mehr, gilt es als nicht ausreichend robust. [Rol03]

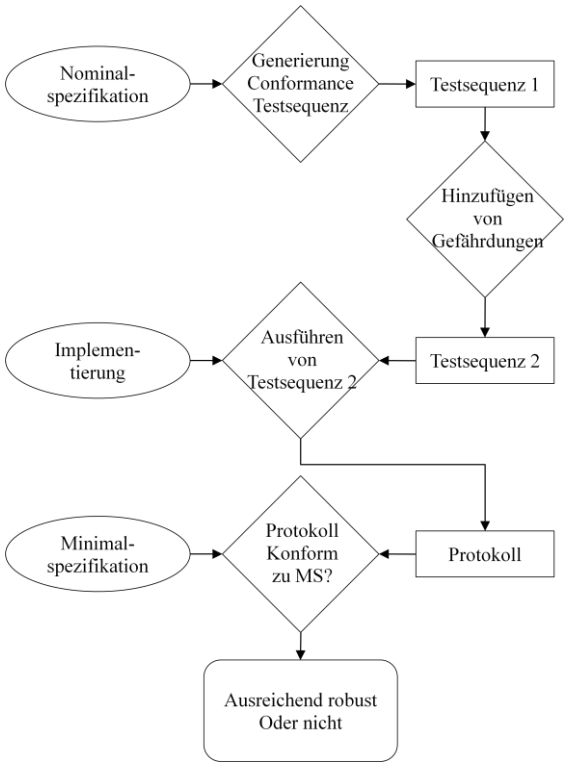

Abbildung7: Testprinzip; nach [Rol03]

3 Testspezifikation mit TTCN

TTCN (Tree and Tabular Combined Notation) ist eine Notation, um Protokolle und Systeme zu beschreiben, für die im Allgemeinen keine Zeitbedingungen gelten. Sie wird gewöhnlich eingesetzt, um Conformance Testfälle zu beschreiben. Allerdings bietet sie nur sehr begrenzte Möglichkeiten, Echtzeitbedingungen darzustellen und es existiert kein echter Standard für die Beschreibung von zeitabhängigen Testfällen. Daher bedarf es einiger Erweiterungen um effizient Testfälle für Echtzeitanwendungen beschreiben zu können.

Als Basisansatz werden funktionale Anforderungen und harte Echtzeitanforderungen von einander getrennt. Im Wesentlichen lassen sich alle Echtzeitbedingungen durch Tupel von Timestamps beschreiben.

Um solche Timestamps zu gewinnen, muss ein Testlauf rein funktionaler Anforderungen mit zusätzlichen Logging-Funktionen instrumentiert werden, um so während der Ausführung des Tests die relevanten Zeitinformationen zu gewinnen. Die Auswertung kann dann nach dem Ausführen des funktionalen Tests oder auch gleichzeitig mit der Ausführung erfolgen. Ersteres hat den Vorteil, den Test nicht durch zusätzliche Last zu verzögern und so die gewonnenen Zeitdaten weiter zu verfälschen. Allerdings müssen die Zeitdaten zur Laufzeit ausgewertet werden, wenn es Abhängigkeiten zwischen den funktionalen und zeitlichen Anforderungen gibt. [Dai03]

Diese gesamte Methodik erscheint als nicht besonders geeignet, da sich ein RTS genau durch die Verknüpfung von funktionalen mit zeitlichen Anforderungen auszeichnet und Tests nicht besonders aussagekräftig sind, wenn dazu der Programmablauf gestört werden muss.

4 Fazit

Das Testen von Echtzeitsystemen gestaltet sich komplex und sehr teuer, da neben den üblichen logischen Tests noch eine Zeitkomponente mit potentiell schwer abzusehendem Verhalten getestet werden muss und diese Systeme oft in komplexen, schwer zu berechnenden Umgebungen eingesetzt werden. Daher erfordert die Verifikation von Echtzeitsystemen sehr disziplinierte und erfahrene Tester.

Die relativ neuen Möglichkeiten zur Formalisierung, Berechnung und Generierung von Testfällen für Echtzeit Systeme ermöglichen eine wesentlich bessere Planung und Übersicht über die Tests und schaffen so neue Möglichkeiten, hohe Abdeckungen und hohe Testqualität zu erreichen.

Allein die Zahl der Publikationen zu diesem Thema in den letzten Jahren ist ein sicheres Zeichen dafür, wie schnell dieser Sektor des IT-Marktes wächst und wie groß der Druck ist, neue, sichere Vorgehensweisen zur Verbesserung des Testprozesses in diesem Sektor zu etablieren. In jedem Fall bietet sich hier noch großes Potential.

Neben den angesprochenen Themen und Paradigmen bieten sich auch interessante Möglichkeiten durch das evolutionäre Testen von Echtzeitsystemen (vgl.: Wegener: Testing Temporal Correctness of Real-Time Systems by Means of Genetic Algorithms) oder die Automatisierungstechniken (ARTT) aus Peters: „Automated Testing of Real-Time Systems“.

5 Literaturverzeichnis

[Dai03] Dai, Zen Ru, Grabowski, Jens and Neukirchen, Helmut. Real-time test specification with TTCN-3. 2003.

[Hof06] Hoffmann, Andreas. Timed Automata. : Reactive Systems Group - Universität Saarland, 2006.

[Jos86] Joseph, M. and Pandya, P. Finding Response Times in a Real Time System. *The Computer Journal.* 1986, 5, pp. 390-395.

[Kri04] Krichen, Moez and Tripakis, Stavros. BlackBox Conformance Testing for Real-Time Systems. *In 11th International SPIN Workshop on Model Checking of Software.* : Springerverlag, 2004.

[Lan05] Langer, Joseph and Kloppenberger, Klaus. Debugging and Verification for Embedded Real Time Systems. *Embedded World.* 2005.

[Lar04] Larsen, Kim G., Mikucionis, Marius and Nielsen, Brian. Online Testing of Real Time Systems. Aalborg, Denmark : 2004.

[Rol03] Rollet, Antoine. Testing Robustness of Real Time Embedded Systems. *Proceedings of Workshop On Testing Real-Time and Embedded Systems.* Reims, France : Université de Reims, 2003.

[Zöb08] Zöbel, Dieter. *Echtzeitsysteme - Grundlagen der Planung.* Berlin Heidelberg : Springerverlag, 2008.