Zukünftige Absicherung der Laufzeitanforderungen in komplexen E/E-Systemen

Peter Gliwa, Gliwa GmbH und Ralf Klein, Symtavision GmbH

peter.gliwa@gliwa.com, klein@symtavision.com

Abstract: Die ISO 26262 fordert neben einem nachhaltigen Softwarearchitekturdesign explizit die Absicherung eines korrekten Timings. Gerade bei komplexen Systemen ist eine fundierte Planung und deren Laufzeitabsicherung unumgänglich, zumal mit zunehmender Komplexität Laufzeitprobleme bei fehlendem Design und schlecht vorbereiteter Analysemöglichkeiten unvermeidbar sind.
Wie kann also in Zukunft das Laufzeitverhalten systematisch abgesichert werden? Bei näherer Betrachtung zeigt sich schnell, dass nur ein umfassender, alle Phasen des V-Modells abdeckender Ansatz zum Ziel führt.

1 Einleitung

Das V-Modell hat in der Softwareentwicklung im Automobilbereich seinen festen Platz. Seit Jahrzehnten orientieren sich Entwickler an der Methodik dieses Modells – zumindest bei der Funktionsentwicklung. Erst in den letzten Jahren setzt sich immer mehr durch, das V-Modell auch auf die Timingaspekte einer Software anzuwenden. Dass es nicht viel früher dazu kam, ist erstaunlich. Die Schritte Design, Implementierung und Test lassen sich sehr gut auch für das Timing durchführen. Und der Erfolg der Steuergeräteprojekte, die diesen Weg konsequent gegangen sind, bestätigen dies. Erfolg heißt hier, frühzeitig ein Timingkonzept definiert und verifiziert zu haben, das im weiteren Projektverlauf mit wenig Aufwand abgesichert und optimiert wird. Laufzeitengpässe werden frühzeitig erkannt und es können Gegenmaßnahmen ergriffen werden, *bevor* sie zu einem Problem werden.

2 Terminologie

Das Timing einer Anwendung wird quantitativ mit diversen Timingparametern beschrieben. Auf Systemebene ist das zum Beispiel die CPU Auslastung. Auf Task Ebene und darunter finden sich Parameter, wie sie in Tabelle 1 beschrieben und in Abbildung 1 visualisiert sind.

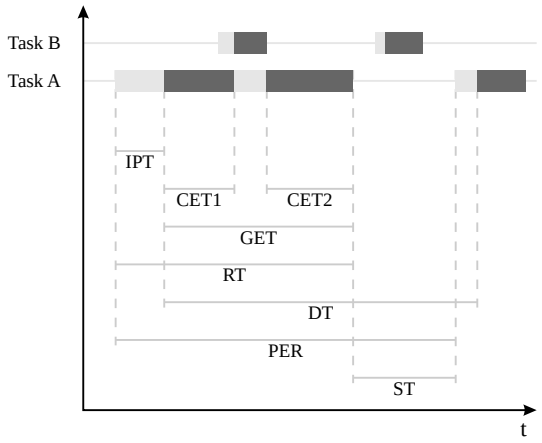

Abbildung 1: Timingparameter einer Task A dargestellt an einer Laufzeitsituation

ID	Abk.	Name EN	Name DE	Beschreibung
1	IPT	initial pending time	Initialwartezeit	Aktivierung bis Start
2	CET	core execution time (computation time)	Nettolaufzeit	Laufzeit abzüglich aller Unterbrechungen
3	GET	gross execution time	Bruttolaufzeit	Start bis Terminierung
4	RT	response time	Antwortzeit	Aktivierung bis Terminierung
5	DT	delta time	Deltazeit	Start bis Start („gemessene Periode“)
6	PER	period	Periode	Sollabstand Aktivierung bis Aktivierung (wie konfiguriert, nicht wie gemessen)
7	ST	slack time	Restzeit	„verbleibende“ Laufzeit: Terminierung bis Aktivierung bei Tasks bzw. Start bei Interrupts
8	JIT	jitter	Jitter	Abweichung der Deltazeit von der Periode

Tabelle 1: Timing information

3 Timinganalysetechniken

Bevor auf die Methodik bei der Anwendung des V-Modells auf das Timing eingegangen wird, sollen zunächst ein paar grundlegende Techniken der Timinganalyse beleuchtet werden [JGR10].

Grundsätzlich gibt es zwei technisch unterschiedliche Ansätze für die Bestimmung von zeitlichen Eigenschaften: Den messbasierten Ansatz sowie den modellbasierten Ansatz. Beim letzteren spielen die komplementären Techniken der Codeanalyse und Schedulinganalyse eine zentrale Rolle. Hinzu kommen hybride Verfahren. Abbildung 2 zeigt eine Strukturierung.

Unter Tracing bezeichnet man im Zusammenhang mit Laufzeitanalysen die Aufzeichnung von timingrelevanten Ereignissen, insbesondere von Start- und Endzeitpunkten von Tasks, Interrupts und Runnables. Diese Aufnahme kann über bestehende Hardwaretracing Schnittstellen wie NEXUS erfolgen, oder als Softwaretracing mit Instrumentierung umgesetzt werden. Die Timingsuite T1 der Gliwa GmbH [Weba] ermöglicht letzteren Ansatz über eine automatisierte Instrumentierung. Die aufgezeichneten Ereignisse können un-

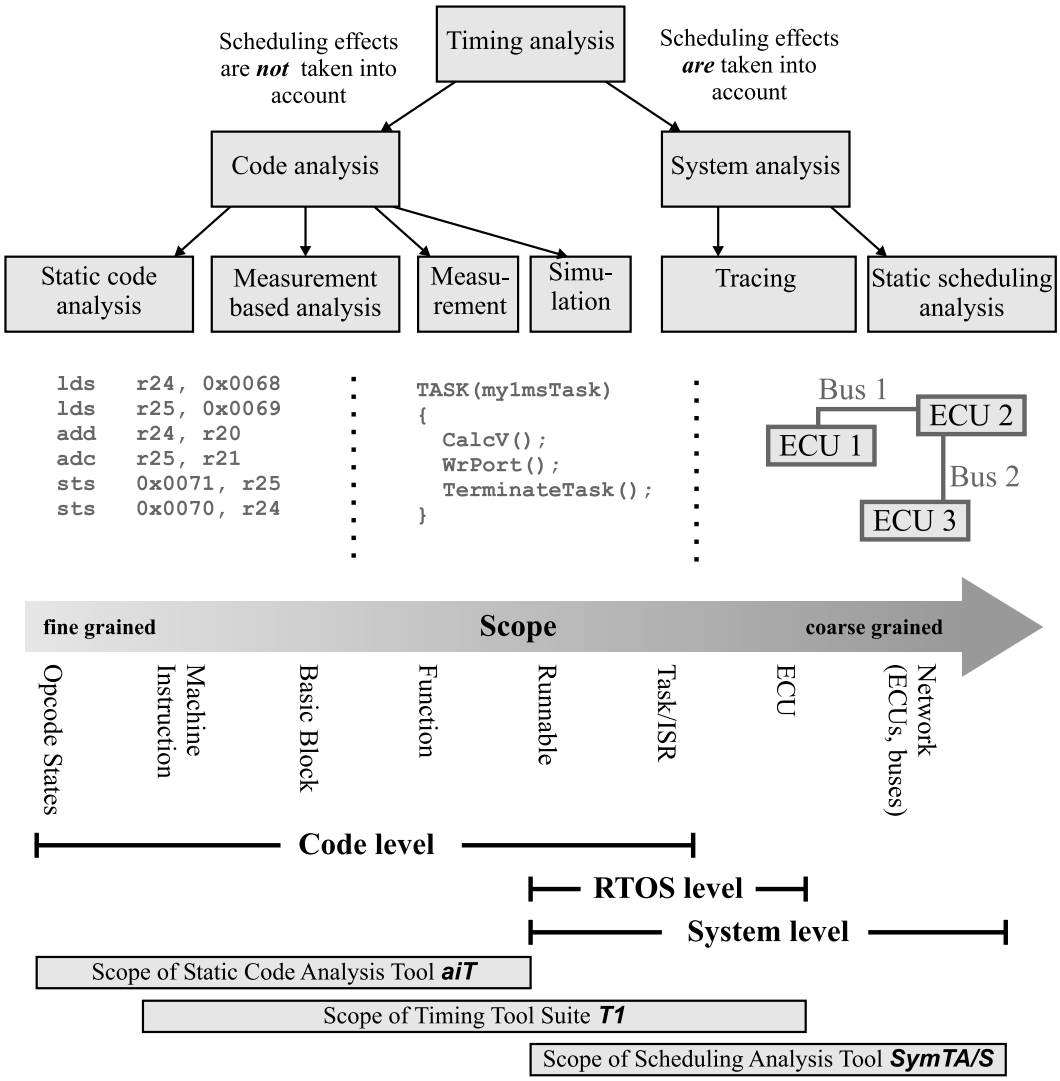

Abbildung 2: Übersicht der Timinganalysetechniken

abhängig vom PC auf dem Target ausgewertet werden (T1.cont) oder zum PC übermittelt und dort visualisiert und analysiert werden (T1.scope). Die Visualisierung kann als „Oszilloskop fürs Betriebssystem“ verstanden werden. Ergebnisse der Analyse sind neben anderen Laufzeitgrößen die Nettolaufzeit (Minimum, Maximum, Durchschnitt, Verteilung) der instrumentierten Objekte. Bei der Ermittelung der Nettolaufzeit macht T1 bei Runnables nicht halt, es können auch Funktionen oder sogar beliebige Codeabschnitte vermessen werden. Jede Codeadresse kann dabei als Start- oder Endpunkt der Messung dienen. Timinganforderungen – so genannte Timing Constraints – können für beliebige Timingparameter festgelegt und sowohl targetseitig als auch PC-seitig überwacht werden. So kann z.B. kontrolliert werden, ob die Nettolaufzeit Time einer Task A größer als 210us ist. Wird eine Timinganforderungen verletzt, kann z.B. über eine Callback ein Fehlerspeichereintrag erfolgen oder über T1.scope die Laufzeitsituation, die zur Verletzung geführt hat, festgehalten und dargestellt werden. Die optionale Komponente T1.flex erlaubt dem Anwender, zur Laufzeit Code zu instrumentieren - ohne dass die Anwendung neu kompiliert/geflasht werden muss. T1.test letztlich ermöglicht die vollständige Automatisierung

von Laufzeittests. So können zum Beispiel parallel zu funktionalen Tests an der HIL die Nettolaufzeiten aller Tasks, Interrupts, Runnables und Funktionen ausgemessen werden.

Die Anbindung des Targets an den PC erfolgt wahlweise über CAN, JTAG, Nexus oder Diagnose (ISO14229 oder ISO14230). Die Bandbreite ist frei wählbar und hat lediglich Einfluss auf die Übertragungsdauer aufgezeichneter Messwerte/Traces.

Grundsätzlich ist der Einsatz von Tracing immer dann erforderlich, wenn der Nachweis erbracht werden muss, dass das reale System auch tatsächlich den Laufzeitanforderungen genügt. Weiterhin ist Tracing das Mittel der Wahl beim Debuggen von Scheduling- bzw. Laufzeitproblemen am realen System.

Die statische Code-Analyse basiert nicht auf der tatsächlichen Ausführung des Codes sondern leitet aus dem Code und der Hardware-Beschreibung Laufzeitinformationen ab. Die Umsetzung im Werkzeug aiT von AbsInt besteht grundsätzlich aus drei Schritten [CFW01]. Zunächst wird der Kontrollflussgraph des Codes analysiert, um die möglichen Ausführungspfade und Schleifenobergrenzen zu bestimmen. Dann werden die Laufzeiten für die Basisblöcke in Abhängigkeit von Prozessor und Speicher-Architektur (Pipeline und Cache) bestimmt. Ein Basisblock ist ein Codeabschnitt, der rein sequentielle Teile enthält, also keinerlei Sprungbefehle. Abschließend wird auf Basis der Laufzeiten der Basisblöcke und der Schleifenobergrenzen ein kritischer Pfad bestimmt, der die Gesamtlaufzeit ergibt.

Auf der Systemebene werden mittels Schedulinganalysen die Antwortzeiten von Tasks und von Wirkketten über mehrere Tasks und ggf. Busse bestimmt. Das Werkzeug SymTA/S [Webb] von Symtavision bestimmt diese Response Time als Summe der eigenen Nettolaufzeit und der Laufzeiten aller anderen Tasks, Runnables und Interrupts, die die Ausführung unterbrechen und/oder verzögern können. Dazu müssen folgende Fragen möglichst genau beantwortet werden: Welche Tasks, Runnables, Interrupts können zeitgleich zum aktuell betrachteten aktiv sein? Wie lange dauert die Störung? Wie häufig kommt es zu Unterbrechungen? Erst dann kann beantwortet werden, wann die Ausführung des aktuell betrachteten Tasks tatsächlich abgeschlossen ist.

Zur Lösung müssen je nach Systemkomplexität viele hunderte bis tausende mögliche Taskabläufe überprüft werden. Dies macht SymTA/S konstruktiv, d.h. es ist nicht notwendig entsprechende Testmuster zur Verfügung zu haben. Neben den Einzellaufzeiten aller Tasks und Runnables und den Schedulingparameter (Periode, Preemption-Typ, Offset, etc.) sind insbesondere Aktivierungsmodelle notwendig, um die Frage nach „wann und wie oft werden Systemteile aktiviert“ zu beantworten, z.B. als Zykluszeit von periodischen Tasks, Delays / Offsets zwischen Trigger-Zeitpunkten, oder als Häufigkeit des Auftretens von Interrupts.

SymTA/S führt diese Analysen automatisch basierend auf einem virtuellen Systemtimingmodell aus. Symtavision bietet für OSEK, AUTOSAR OS, CAN und FlexRay sehr detaillierte Analyse-Bibliotheken für die Absicherung an. Es werden sowohl das Worst-Case Zeitverhalten als auch statistische Verteilungen bestimmt. Für die Optimierung bietet Symtavision ein Explorations-Paket und Analysen mit reduziertem Detail, die in der Frühphase für schnelle Vergleiche verschiedener Konfigurations-Optionen benutzt werden. Mittels Sensitivitätsanalyse können die Engpässe und Reserven einer möglichen Systemkonfiguration bestimmt werden.

4 Das V-Modell angewendet auf Timing

4.1 Frühe Phase - Timingdesign

Die Systemarchitektur wird in einer frühen Phase modelliert und Timingbudgets werden bereits dort vergeben. Somit lässt sich noch vor der Verfügbarkeit lauffähiger Softwarestände das Timingkonzept absichern. Mittels der in AUTOSAR 4.0 eingeführten Timing Extensions werden Laufzeitanforderungen spezifiziert. Diese Anforderungen ergeben sich einerseits aus der Systemarchitektur, andererseits aus funktionalen/physikalischen Anforderungen. Bereits Anfang 2011 wurden von OEMs Lastenhefte für Steuergeräte erstellt, in denen die Laufzeitanforderungen per AUTOSAR 4.0 Timing Extensions festgeschrieben wurden.

4.2 Integration und Timingdebugging

Im Projektverlauf kommt irgendwann der Punkt, an dem die Software das erste Mal auf dem Zielsystem läuft. Zunächst noch mit wenig Funktionalität, doch sukzessive werden Softwarekomponenten integriert, Treiber aktualisiert etc. Die Erstinbetriebnahme als auch der folgende Integrationsprozess kann wesentlich vereinfacht werden, wenn das Timing mittels Tracing überwacht wird. Unerwartete und somit im frühen Timingmodell nicht vorhandene Timinganomalien können so aufgedeckt und die Ursachen geklärt werden (Timingdebugging).

Zur gleichen Zeit wird das frühe Timingmodell mittels gemessener Ausführungszeiten verfeinert und die in der frühen Phase vergebenen Timingbudgets zur Laufzeit überwacht. Im Gegenzug wird mittels Schedulinganalysen mit dem aktuellen Modell der aktuelle Softwarestand abgesichert. Der Abgleich und die Absicherung erfolgt größtenteils automatisiert, benötigt somit wenig Entwicklungsressourcen.

4.3 Späte Phase - Timingtests

Die in der früher Phase definierten Laufzeitanforderungen müssen überprüfbar und messbar sein. Automatisierte Laufzeitmessungen können so umgesetzt werden, dass sie parallel zu funktionalen Tests laufen und praktisch keine zusätzliche Testzeit benötigen. Dadurch wird es möglich, fehlerhaftes Laufzeitverhalten zu debuggen und Messungen auch im Fahrzeug mit Seriensteuergeräten vorzunehmen. Die Testabdeckung für Fälle, die bei den Tests *nicht* auftraten, erfolgt dann mit Hilfe von Schedulinganalysen.

5 Beispielhaftes Serienprojekt mit AUTOSAR Laufzeitabsicherung

Anfang 2011 wurde bei BMW ein Fahrwerkssteuergerät zur Entwicklung ausgeschrieben. In dem dazugehörigen Lastenheft wurde der Großteil der Timinganforderungen mittels der AUTOSAR 4.0 Timing Extensions formuliert. Die Timinganforderungen ergaben sich aus dem geforderten physikalischem Verhalten einerseits sowie aus Netzwerkanforderungen andererseits. Die Tatsache, dass Timinganforderungen formal mittels AUTOSAR beschrieben wurden, ist allein schon bemerkenswert, da die AUTOSAR 4.0 Timing Extensions im breiten Seriengeschäft bisher wenig in der Anwendung sind. Das BMW Projekt ging aber noch einen zukunftsweisenden Schritt weiter. Die formal fixierten Anforderungen wurden automatisiert mittels T1 abgesichert. Der sonst beim Thema Timing zu beobachtende Bruch zwischen textuell formulierten Laufzeitanforderungen im Lastenheft und der manuellen Absicherung dieser Anforderungen in der Verifikationsphase war aufgehoben. Die Lösung wurde im Februar 2012 auf der ERTS in Toulouse erstmalig dem breiten Publikum vorgestellt [OSG12].

Die AUTOSAR 4.0 Timing Extensions erlauben die Formulierung von beispielsweise Latency/Offset Anforderungen, Periodizitätsanforderungen oder Anforderungen bezüglich einer definierten Reihenfolge von Ereignissen. Abgelegt wird all dies in AUTOSAR XML. Da AUTOSAR XML als Sprache für den Alltagsgebrauch kaum geeignet ist, hat die BMW Car IT GmbH eine intuitive und einfach zu verwendende Sprache entwickelt: *Artime*. Das gleichnamige Artop Plug-in erlaubt es, in der gewohnten Eclipse basierten AUTOSAR Oberfläche Artop Timinganforderungen für ein System zu spezifizieren.

Ein weiteres Artop Plug-in namens *ArT1* ermöglicht, Laufzeitanforderungen samt Systemkonfiguration in eine T1 Projektdatei zu exportieren. Es wird erstmals der „missing link“ eliminiert; AUTOSAR Projekte können so nahtlos von der Timingspezifikation den Weg hin zur Timingabsicherung beschreiten.

Der logische nächste Schritt besteht jetzt darin, die statische Schedulinganalyse mit in den Prozess einzubeziehen. Am gezeigten Projekt kann und soll dies an zwei Stellen erfolgen. Zum einen in der frühen Entwicklungsphase, um Schedulingkonzepte zu entwickeln und losgelöst vom Code abzusichern. Zum anderen zum Zeitpunkt der Systemabsicherung, wenn detaillierte Mess- und Traceergebnisse zuverlässige maximale Nettolaufzeiten aller Tasks, Interrupts und Runnables liefern. Hier kann mittels statische Schedulinganalyse auch solche Laufzeitsituationen überprüft werden, die während der Tests nicht beobachtet wurden aber aufgrund der Systemkonfiguration zumindest theoretisch denkbar sind. Abbildung 3 zeigt die jeweiligen Schritte am V-Modell.

6 Zusammenfassung

Der hier vorgestellte Ansatz für die Betrachtung von Timing in verschiedenen Entwicklungsphasen auf verschiedenen Ebenen mit Hilfe spezialisierter Tools zeigt deutlich den Nutzen und die Notwendigkeit zum Einsatz von Timing Werkzeugen.

Die Einsatzfähigkeit und das nahtlose Zusammenspiel der Werkzeuge wurde bei diversen Kunden in vielen Projekten bewiesen. Dabei erlaubt der modellbasierte Ansatz von

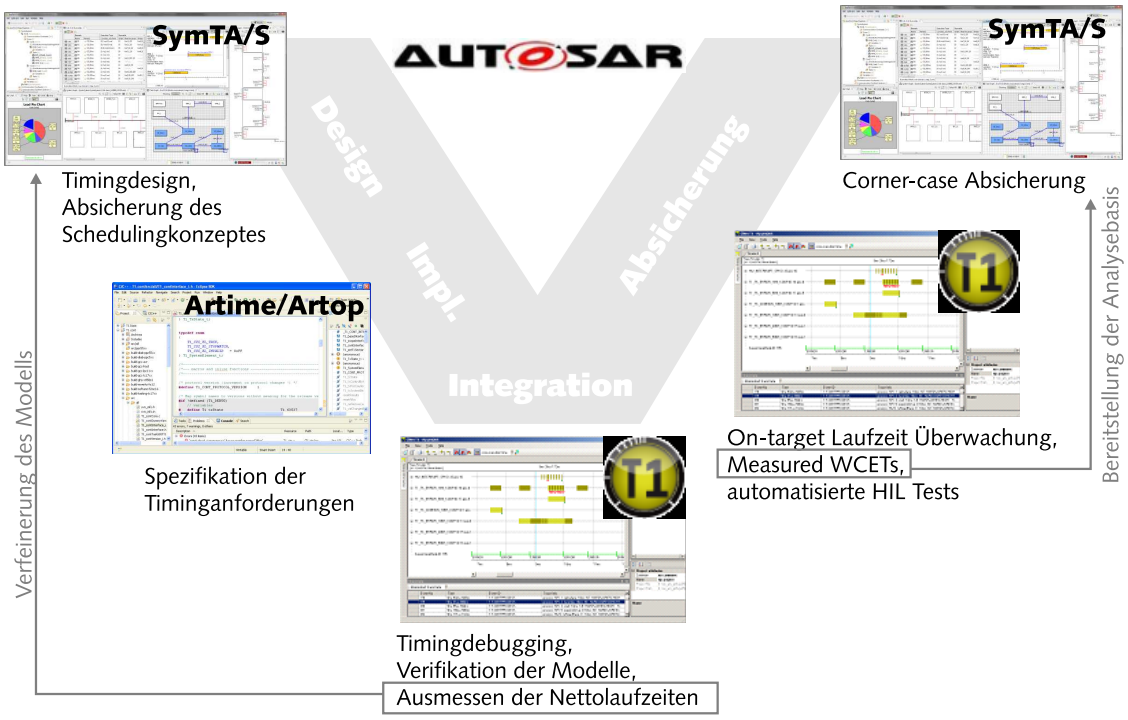

Abbildung 3: Timing durchgängig in AUTOSAR und am V-Modell

SymTA/S schon in der Frühphase die Ermittlung von kritischen Timingwerten, die als Timing Budgets übernommen werden können. Das Timing Modell wird dann im Laufe des Projektfortschritts immer weiter an die echte Implementierung angepasst und durch regelmäßige Analysen das einhalten der Timing Budgets garantiert. Die finale Absicherung zum Projektende erfolgt dann auf den reellen Daten.

Tracing Tools wie T1 erlauben das Erfassen und Bereitstellen von notwendigen Daten für die Schedulinganalyse und geben darüber hinaus einen Einblick in das tatsächliche implementierte Zeitverhalten. Des weiteren erlauben T1 Erweiterungen auch Target Stresstests und das Ermitteln von freien Kapazitäten für die Erweiterung von Funktionsumfängen.

Angesichts all dieser Möglichkeiten ist es verwunderlich, warum die Betrachtung von Timing auch schon in sehr frühen Entwicklungsphasen nicht viel stärkere Berücksichtigung erfährt, da dadurch viel Zeit, und somit Geld, durch das aufwändige Suchen von Timing Problemen weitestgehend vermieden werden können. Fest steht, dass mit der ISO 26262 das Thema Timingabsicherung nochmals Rückenwind bekommt, ja unumgänglich wird. Die ISO 26262 fordert explizit die Absicherung des Laufzeitverhaltens von Echtzeitsystemen [KF12].

Literatur

[CFW01] M. Langenbach F. Martin M. Schmidt H. Theiling S. Thesing C. Ferdinand, R. Heckmann und R. Wilhelm. Reliable and precise WCET determination for a reallife processor. In *First Workshop on Embedded Software (EMSOFT01)*, 2001.

[JGR10] Marek Jersak, Peter Gliwa und Kai Richter. Planung und Absicherung der Echtzeitfaehigkeit von Software und vernetzten Steuergeraeten. In *Simulation und Test in der Funktions- und Softwareentwicklung fuer die Automobilelektronik III*, 2010.

[KF12] D. Kästner und C. Ferdinand. Safety Standards and WCET Analysis Tools. *Embedded Real Time Software and Systems Congress ERTS* [2], 2012.

[OSG12] Christoph Ainhauser Oliver Scheickl und Peter Gliwa. Tool Support for Seamless System Development based on AUTOSAR Timing Extensions. In *Proceedings of Embedded Real-Time Software Congress (ERTS)*, 2012.

[Weba] Gliwa Website. http://www.gliwa.com.

[Webb] Symtavision Website. http://www.symtavision.com.