Messung der Anwendungsperformance auf Mikroprozessoren für den Automobilbereich

Praxisgerechte Messtechnik für aktuelle und zukünftige Mikroprozessoren

Dipl.-Ing. (BA) **P. Gliwa**, Gliwa GmbH, Weilheim i.OB.;
Dr.-Ing. **A. Mayer**, Infineon Technologies AG, München;

Kurzfassung

Performance- und Speicherbedarf sind die ausschlaggebenden Kriterien bei der Auswahl eines Mikroprozessors. Doch ist die Messung der Anwendungsperformance und die Bestimmung freier Ressourcen alles andere als trivial. Fehlerhafte Messungen und Abschätzungen des Laufzeit- und Speicherbedarfs führen unter Umständen zu Fehlentscheidungen mit Auswirkung auf Entwicklungskosten, Entwicklungszeiten und/oder Stückkosten. In laufenden Projekten erlaubt die richtige Laufzeitmesstechnik sowohl gezielte Optimierung als auch stichhaltige Absicherung der Laufzeitanforderungen.

Abstract

Performance and memory consumption are the key selection criteria for a microcontroller. It is vital to have sufficient performance and memory and yet over-capacity causes unnecessarily high production costs. Inappropriate measurements and estimations of performance and memory consumption lead to poor hardware selection with expensive consequences. Even with well-chosen hardware, performance and memory consumption must be assessed progressively to ensure that problems are detected early and any optimisations are correctly targeted. Accurately measuring the performance of modern, multi-threaded applications on sophisticated microcontrollers is a complex task and we present a systematic methodology for such performance analysis. In this methodology, tools are used to permit powerful analyses with acceptable effort.

1. Einleitung

Die Steuergeräteentwicklung sieht sich nicht nur der Forderung nach immer *mehr* Funktionalität, sondern meist auch der Forderung nach immer *schnellerer* Funktion gegenübergestellt. Schnellere Busse erfordern schnellere Bearbeitung der Daten. Neue Anwendungsgebiete wie die Elektromobilität bringen neue Anforderungen an das zeitliche Verhalten mit sich. Da-

zu ein Beispiel. Bei einem Motor- oder Fahrstrategiesteuergerät stellt die ungewollte Beschleunigung einen der wesentlichen kritischen Fehlerzustände dar. Wird also z.B. aufgrund eines Softwarefehlers fälschlicherweise eine maximale Drehmomentanforderung ausgegeben, so muss durch Mechanismen des Sicherheitskonzeptes eingegriffen werden, bevor das Fahrzeug eine kritische Beschleunigung erfährt. In der Welt der Verbrennungsmotoren gilt hier seit Jahren eine Reaktionszeit von rund 300ms als ausreichend. Mit den immer wichtiger werdenden Elektroantrieben stellt sich die Situation völlig anders da. Ein Elektromotor hat im Gegensatz zum Verbrennungsmotor im Stillstand sein maximales Drehmoment. Auch müssen weniger Motormassen beschleunigt werden, der Elektromotor „spricht schneller an“. Untersuchungen haben gezeigt, dass hier schon nach rund 70ms eine Abschaltung erfolgen muss, damit der Geschwindigkeitszuwachs oder die aus dem Stand heraus zurückgelegte Strecke im ungefährlichen Rahmen bleibt. Sicherheitsmechanismen wie Coretest und RAM-Test müssen trotz komplexerer Cores und größerer Speicher schneller reagieren können.
Neben den Systemen selbst ist die Art und Weise der Zusammenarbeit bei der Steuergeräteentwicklung komplexer geworden. Mehrere Zulieferer und der OEM entwickeln jeweils Teile der Software und beim Austausch muss IP Schutz gewährleistet sein. Dennoch soll der Ressourcenverbrauch, also Laufzeit- und Speicherbedarf, für das gesamte Projekt a) jederzeit einfach ermittelt werden können und b) nicht wesentlich höher ausfallen als bei einer Entwicklung unter einem Dach.
AUTOSAR hat sich zum Ziel gesetzt, die Steuergeräteentwicklung zu vereinfachen und wird mittel- bis langfristig dieses Ziel erreichen. Doch derzeit überwiegen die Kosten für Einführung und Umstellung noch die angestrebten Kostenersparnisse durch Vereinheitlichung der Schnittstellen. Viele AUTOSAR Konzepte sind zwar seit längerem stabil, doch sind sie noch nicht in den Serienprojekten etabliert und AUTOSAR Methoden werden noch nicht „gelebt“.

Um diesen Gegebenheiten optimal begegnen zu können, müssen Prozessoren, Entwicklungsumgebungen und Werkzeuge besondere Features bieten. Features, die einerseits eine möglichst einfache und kostengünstige Entwicklung erlauben, andererseits aber ein hohes Maß an Sicherheit und Verfügbarkeit des Produktes gewährleisten.
Im Verlauf dieser Abhandlung werden einige dieser Features näher beleuchtet.

2. Prozessorressourcen: Laufzeit und Speicher

Wurde zu Projektbeginn die Entscheidung für einen bestimmten Prozessor gefällt, steht damit auch fest, welche Rechenleistung und wie viel Speicherplatz zur Verfügung stehen, das Fehlen eines externen Speichers vorausgesetzt. Stellt sich im weiteren Projektverlauf her-

aus, dass entweder Laufzeit oder Speicher knapp werden, gibt es nur wenige Möglichkeiten zu reagieren:

- Codeoptimierung, Umstrukturierung
- Prozessorwechsel
- Reduktion der Funktionalität

In der Praxis kommt nicht selten leider noch eine vierte Option hinzu, nämlich eine Umsetzung der vorgegeben Funktionalität auf der gegebenen Hardware auf Kosten der Sicherheit und/oder Verfügbarkeit. In solchen Fällen werden z.B. Aussetzer aufgrund von selten auftretenden Laufzeitproblemen mangels bezahlbarer Alternativen einfach hingenommen und erst nach Serienanlauf richtig angegangen.

Der Prozessorwechsel ist meist zu aufwendig und zu teuer und eine Reduktion der Funktionalität in der Regel nicht durchzusetzen. Bleibt die Optimierung, bei der die richtigen Werkzeuge entscheidend helfen können und die umso einfacher ist, je besser der Prozessor die Ressourcenanalyse unterstützt. Bevor auf die Verschiedenen Messtechniken eingegangen wird, sollen zunächst die Ressourcen „Laufzeit“ und „Speicher“ näher betrachtet werden.

Unter Laufzeit wird Rechenzeit verstanden. Ein Singlecoreprozessor kann immer nur eine Anweisung gleichzeitig bearbeiten, muss also sequentiell alle anstehenden Aufgaben erledigen. Je höher die Taktfrequenz und je leistungsfähiger Befehlssatz, Cache, Pipelines und Speicherzugriffe, desto mehr Aufgaben können in einer gegebenen Zeit erledigt werden.

Für einen gegebenen Codeabschnitt, z.B. eine Schleife, eine Funktion, eine Task oder einen Interrupthandler ergeben sich abhängig vom durchlaufenen Pfad und dem Pipeline- und Cachezustand des Prozessors unterschiedliche Laufzeiten, die mit verschiedenen Wahrscheinlichkeiten auftreten, siehe Bild 1.

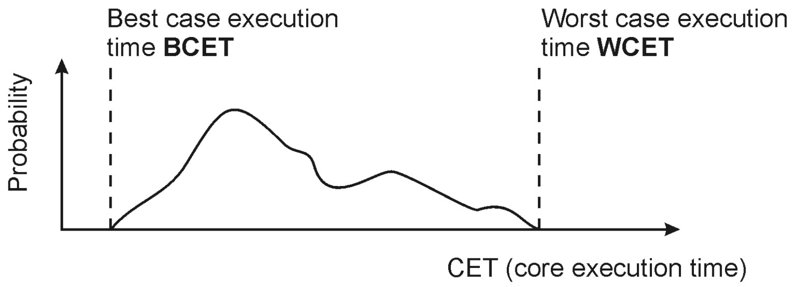

Bild 1: Wahrscheinlichkeitsverteilung der Laufzeit eines Codeabschnitts

Bei dieser Betrachtung wird schon impliziert, dass der betrachtete Codeabschnitt nicht unterbrochen wurde beziehungsweise dass jede Unterbrechung herausgerechnet wurde, es handelt sich bei diesen Laufzeiten um sogenannte Nettolaufzeiten. Die Nettolaufzeit ist die zentrale Größe bei der Laufzeitoptimierung eines Codeabschnitts, die es zu reduzieren gilt.

Neben der Nettolaufzeit interessieren auch weitere zeitbezogene Größen, z.B. die Antwortzeit (im Englischen „response time“, RT). Sie repräsentiert die Zeitspanne zwischen dem Erforderlichwerden einer Ausführung bis zum Abschluss der Ausführung.

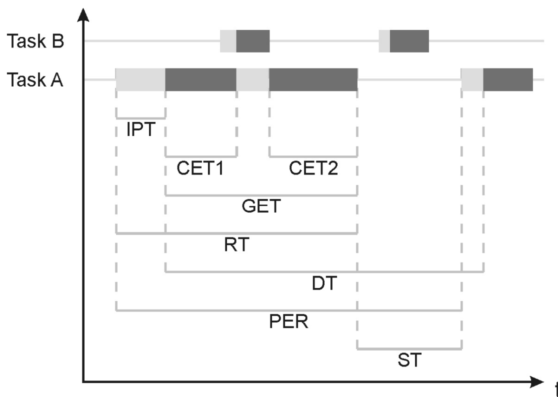

Bild 2: Timinggrößen veranschaulicht an einer Laufzeitsituation

Tabelle 1: Übersicht der wichtigsten Timinggrößen

Abk.	Name EN	Name DE	Beschreibung
IPT	initial pending time	Initialwartezeit	Aktivierung bis Start (in der Regel nicht messbar bei Interrupts)
CET	Core execution time	Nettolaufzeit	Laufzeit, bei der eventuelle Unterbrechungen herausgerechnet wurden
GET	gross execution time	Bruttolaufzeit	Laufzeit inklusiv der Zeit, die eventuelle Unterbrechungen in Anspruch genommen haben
RT	response time	Antwortzeit	Aktivierung bis Terminierung bzw. Ende (in der Regel nicht messbar bei Interrupts)
DT	delta time	Deltazeit	Start bis Start („Laufzeitperiode“)
PER	period	Periode	Aktivierung bis Aktivierung („konfigurierte Periode“)
ST	slack time	Restzeit	Verbleibende Zeit bis zur nächsten Aktivierung bei Tasks bzw. bis zum nächsten Start bei Interrupts
JIT	Jitter	Jitter	Abweichung von der angestrebten Periode

Sie hängt stark davon ab, ob und in welchem Maße der betrachtete Code unterbrochen und somit verzögert wurde. Die Antwortzeit ist eine wesentliche Größe für die Formulierung von Laufzeitanforderungen, da sie typischerweise gegebenen Deadlines gegenübergestellt wird.
Bild 2 veranschaulicht neben der Netto- und Antwortzeit weitere, in der Regel weniger wichtige Zeiten.
Die Ermittelung der Laufzeiten ist teilweise sehr kompliziert und mit einigem Aufwand verbunden. Kapitel 3 beschreibt einige Techniken und deren Vor- und Nachteile.

Soweit die Ressource *Laufzeit*. Im Folgenden soll auf die Ressource *Speicher* näher eingegangen werden. Anders als die Laufzeiten ist der statische Speicherbedarf sehr einfach zu ermitteln, die beim Bauen der Anwendung generierte Linker-Map enthält diese Information

mehr oder weniger gut leserlich. Es empfiehlt sich, diese Datei oder eine andere geeignete Quelle bei jedem Buildprozess auszuwerten und eine Zusammenfassung der Speichernutzung in die Ausgabe des Buildprozesses einzufügen. Eine geringfügig erweiterte Linkerkontrolldatei und ein einfaches Perlskript ermöglichen dieses Vorgehen ohne großen Aufwand. Die Ausgabe kann wie in Bild 3 gezeigt aussehen.

```
C:\WINDOWS\system32\cmd.exe
Memory usage (generated by perl script "memStat.pl")
Input file: temp.map
(c) by Gliwa GmbH, Muenchen

EFLASH:   255.46 KByte used ( 12.5% of  2048.00 KByte; start: 00000000)
ERAM:      13.07 KByte used (  5.1% of   256.00 KByte; start: 00820000)
IFLASH:   500.33 KByte used ( 97.7% of   512.00 KByte; start: 00400000)
IRAM:      26.48 KByte used ( 82.8% of    32.00 KByte; start: 007F8000)


Explanations:
-------------
EFLASH: external flash; no dynamic memory allocation
```

Bild 3: Ausgabe der Speichernutzung am Ende des Buildprozesses bei einem Serienprojekt

Aufwendige dynamische Speicherverwaltung ist in der Automobilbranche unüblich und somit ist oft der Stack der einzige dynamische Speicherbereich einer Anwendung. Um den Stackbedarf zu ermitteln, gibt es dynamische oder statische Verfahren. Das am häufigsten verwendete dynamische Verfahren beschreibt in der Initialisierungsphase den gesamten Stack mit einem bestimmten Pattern und überprüft zur Laufzeit – meist im Hintergrund – wie viel vom Pattern überschrieben, wie viel Stack also genutzt wurde. Statische Verfahren kommen gänzlich ohne Messung aus und untersuchen lediglich den Code. Auf diese Weise arbeitet z.B. der ***StackanAlyzer*** von der Firma AbsInt.

Praktisch jeder heutige Mikroprozessor bietet verschiedene Speicher mit verschiedenen Zugriffszeiten. Es leuchtet ein, dass im Sinne der Laufzeitoptimierung möglichst die Symbole in die schnellen Speicher lokatiert werden, auf die häufig zugegriffen wird. Es wird hier bewusst von Symbolen gesprochen, da Code wie Daten gleichermaßen betroffen sind. Dies ist einer der wenigen Berührungspunkte der beiden Ressourcen Speicher und Laufzeit. Die Frage, welches Symbol wie häufig verwendet wird, ist zum Teil gar nicht so leicht zu beantworten. Bei den Tasks und Runnables ist es noch leicht, Zahlen zu nennen, doch schon bei den Diensten der Treiberschicht wird es etwas komplizierter. Und fragt man einen Integrator nach der Häufigkeit, mit der implizite Castfunktionen des Compilers ausgeführt werden, erntet man meist Schulterzucken. Dabei sind es gerade solche Symbole, für die sich niemand

richtig verantwortlich fühlt, die viel Optimierungspotential bieten und nicht selten mehrere tausend Mal pro Sekunde verwendet werden.
Hier bieten moderne Architekturen wie der Infineon TriCore zusammen mit den entsprechenden Werkzeugen wie z.B. ***T1*** von der Gliwa GmbH die Möglichkeit, ohne jedes Wissen um die Funktionsweise einer Anwendung die Verwendungshäufigkeit aller Symbole automatisiert auszumessen. Eine solche Analyse kann sehr einfach dazu verwendet werden, die Speichernutzung zu optimieren, ohne funktionale Einschränkung hinnehmen oder größere Eingriffe in die Software machen zu müssen.
Ein zweiter Berührungspunkt zwischen den Ressourcen Speicher und Laufzeit sind die Compilerschalter. Die Optimierungen, die der Compiler vornimmt, wirken sich bis zu einem gewissen Grad positiv sowohl auf Laufzeit als auch Speicherbedarf aus. Einige Optimierungen verbessern aber den einen Aspekt auf Kosten des anderen. Bei einer gezielten Optimierung sollte nicht jedes Modul über den gleichen Kamm geschoren, sondern die Auswirkungen im Einzelnen ausgemessen werden. Beispiel: es ist oft überraschend, wie wenig Speicherersparnis eine Einstellung „optimize for size“ bringt, die jedoch enorm negative Auswirkung auf die Laufzeit hat.

3. Timinganalysemethoden

Bild 4 gibt eine Übersicht über die verschiedenen Timinganalysemethoden und ordnet sie dem jeweiligen Anwendungsbereich zu. Die Einteilung erfolgt entsprechend den jeweiligen Analyseverfahren. Zum einen ist da die Unterscheidung nach Code- und RTOS- bzw. Systemebene. Wie weiter oben erwähnt, spielt es bei der Ermittelung von Timinggrößen eine entscheidende Rolle, ob der betrachtete Teil der Software von anderen Teilen unterbrochen werden kann oder nicht. Analysewerkzeuge auf der Codeebene gehen davon aus, dass keine Unterbrechungen stattfinden. Ein einfaches Beispiel ist das Setzen eines Portpins am Anfang und das Rücksetzen am Ende einer Task. Das entsprechende Rechteck auf dem Oszilloskop verrät nicht, ob die Task unterbrochen wurde oder nicht und lässt somit keinen direkten Schluss auf die Nettolaufzeit der Task zu.
Bei der Betrachtung der RTOS-Ebene werden genau diese Effekte in den Mittelpunkt der Betrachtung gerückt. Eine zentrale Frage lautet hier zum Beispiel: „Wie groß ist die maximale Antwortzeit meiner Task, wenn diese und jene Nettolaufzeiten angenommen werden und durch welche mögliche Unterbrechungen kommt es zu dieser Zeit?“.
Die Betrachtung der Systemebene geht noch einen Schritt weiter und berücksichtigt die Busse zwischen den Steuergeräten. Eine zentrale Fragestellung ist hier die nach der Dauer von Wirkketten, also wie lange es dauert, bis ein an einem Sensor eingelesenes, über Busse

transportiertes, in einem Steuergerät ausgewertetes Signal letztlich eine Auswirkung an einem Aktor hervorruft. Wichtig sind sowohl Worst-Case als auch Verteilungen der typischen Fälle.

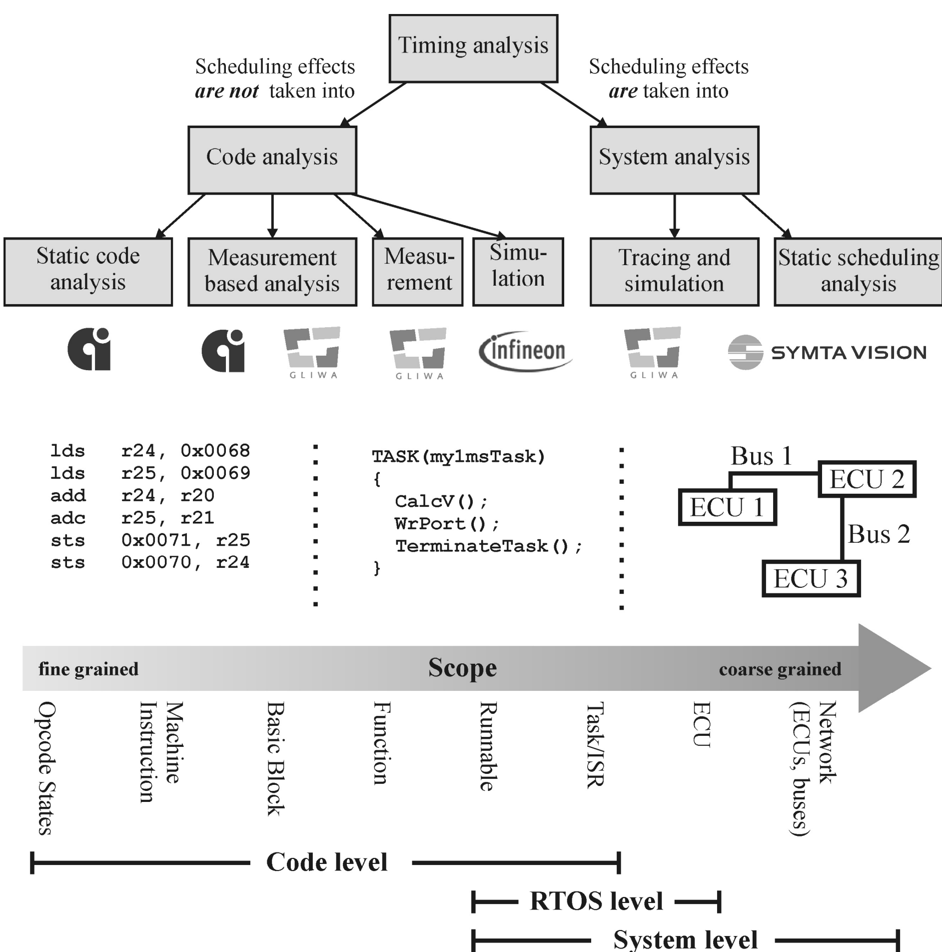

Bild 4: Timinganalysetechniken und Anwendungsbereiche

Eine weitere Unterscheidung teilt die Timinganalysemethoden in statische und mess- bzw. tracebasierte Ansätze auf. Statische Ansätze sind modellbasiert, bilden also die Realität mehr oder weniger genau ab. Sie erlauben Untersuchungen noch bevor Hardware verfügbar ist und sind somit in der frühen Projektphase sehr gut einsetzbar. Da sie unabhängig von Testvektoren sind, sind sie in der Lage, sowohl Worst- und Best-Cases als auch Verteilungen von typischen Fällen zu errechnen. Messbasierte Ansätze erfordern ein lauffähiges System und spiegeln die Realität wieder. Sie eignen sich zur Timingabsicherung, zum Timingdebuggen und nicht zuletzt zur Überprüfung modellbasierter Ansätze. Der Königsweg führt über die gewinnbringende Kombination der Techniken. So hat zum Beispiel BMW bei der Entwicklung der Aktivlenkung mittels Tracing sowohl Timingdebugging betrieben als auch

einen großen Teil der Timingabsicherung bestritten [1]. Um auch die seltenen theoretisch möglichen Worst-Cases zu erwischen, wurde basierend auf den Messergebnissen statische Schedulinganalysen durchgeführt – all dies automatisiert mittels HIL Tests.

Messlösungen erfordern praktisch immer eine Modifikation entweder der Hard- oder Software. Bei der Verwendung der HW-Traceschnittstelle Nexus z.B. muss die Nexus Schnittstelle rausgeführt sein, was bei Seriensteuergeräten praktisch immer eine Sondervariante der Hardware bedeutet. Wesentlich einfacher ist hier die Möglichkeit, die Infineon mit seinem TriCore bietet: zu den Serienchips gibt es sogenannte ED-Devices, die Tracefunktionalität (MCDS Multi-Core Debug Solution) nebst Tracespeicher schon auf dem Silizium mit sich bringen. Mit der gerade in Entwicklung befindlichen neuesten Generation, wird es möglich sein, nicht nur wie bisher die Cores, Busse und etwa 60 Performanceindikatoren (Cache Hits/Misses, Bus Konflikte, etc.) parallel zu tracen, sondern auch direkt Hardware Signale, z.B. solche die Interrupts auslösen. Damit lässt sich dann die in Bild 2 gezeigte IPT erstmalig auch für Interrupts messen.

Beim Mess- bzw. Traceansatz, der über die Instrumentierung der Software geht, ist es entscheidend, dass der Einfluss auf die Software so gering wie möglich ist. Pro Tracepunkt benötigt die Tracinglösung ***T1*** von der Gliwa GmbH auf einem MPC5604B mit 64MHz rund 0,65µs, beim TriCore TC1797 mit 90MHz sogar nur ca. 0,32µs.

Sowohl auf Code als auch auf Systemebene ist die Simulation möglich, doch erfordert sie erfahrungsgemäß einen großen Inbetriebnahme- und Pflegeaufwand, will man verlässliche und aussagekräftige Ergebnisse produzieren. Insbesondere müssen für belastbare Simulationsergebnisse repräsentative Traces zur Verfügung stehen. Da diese zunächst an einem realen System erfasst werden müssen, stellt sich die Frage, warum danach noch der zusätzliche Simulationsaufwand betrieben werden soll. Ein Grund kann sein, dass sich am Simulationsmodell leichter Änderungen vornehmen lassen als am echten System.

Im Rahmen des Forschungsprojektes ALL-TIMES [2] haben europäische Firmen und Universitäten Methoden, Werkzeuge und Schnittstellen im Timingumfeld entwickelt. Auf dieser Basis gründeten 2008 die drei Firmen AbsInt, Gliwa und Symtavision die REAL-TIME EXPERTS als Allianz von Echtzeitexperten mit dem Ziel, für jede Fragestellung aus dem Embedded Timingbereich eine Lösung bieten zu können [3]. AbsInt bietet unter Anderem den statischen Codeanalysator ***aiT*** zur Ermittelung von WCETs und einen ***StackAnalyzer*** als Verifikationswerkzeuge an. ***T1*** von Gliwa ist eine Tracing- und Messlösung, die on-target Messung und Überwachung ermöglicht und komplexes Laufzeitverhalten einfach verständlich visualisiert. ***SymTA/S*** von Symtavision erlaubt statische Schedulinganalyse (Worst-Case und Verteilungen von typischen Fällen) zur Planung und Absicherung des

Systemtimings. Zurzeit entwickeln Symtavision und Gliwa zusammen mit Infineon die nächste Generation der Timingwerkzeuge für Tracing und Schedulinganalyse, die dann speziell auf Multicore abgestimmt sind.

4. Beispiele für den Einsatz von Timingwerkzeugen

Die wenigsten Steuergeräteprojekte werden auf der grünen Wiese entwickelt sondern sind die Weiterentwicklung einer aktuell eingesetzten Generation. Bei der Weiterentwicklung können zwei unterschiedliche Wege eingeschlagen werden. Entweder die bestehende Hardware wird beibehalten, dann muss vorab sichergestellt werden, dass die zusätzlichen Funktionen die gegebene Hardware nicht überlasten. Oder aber es wird eine neue Hardware entwickelt. Die Prozessorauswahl wird dabei zu einem wesentlichen Teil aus der Erfahrung heraus vor dem Hintergrund der anstehenden Erweiterungen und einer meist groben Kenntnis der Leistungsfähigkeit des Prozessors entschieden.

In beiden Fällen können Timingwerkzeuge wichtige Entscheidungskriterien liefern. Im ersten Fall – nur die Software wird erweitert – muss der Funktionsentwickler abschätzen, mit welcher zusätzlicher Laufzeit zu rechnen ist. Dabei hilft es, die zu entwickelnde Funktion von der Komplexität her mit bestehenden Funktionen zu vergleichen. Mittels gezielter „Laufzeitbremsklötze“ in der bestehenden Software kann dann sehr pragmatisch ausgemessen werden, ob die zusätzliche Funktionalität das System überlastet. Alternativ oder ergänzend kann dies per statischer Schedulinganalyse „durchgespielt“ werden, wobei z.B. auch automatisch die zulässigen Grenzen für zusätzliche Laufzeiten bestimmt werden können.

Für den zweiten Fall – eine neue Hardware mit neuem Prozessor wird entwickelt – kann man mittels statischer Codeanalyse im Vorfeld sehr gut abschätzen, welche Laufzeitperformance bestehender Code auf dem einen oder anderen Prozessor bringen wird.

Ist die Entscheidung gefallen und liegt ein A-Muster vor, können mittels Tracing leicht unerwünschtes Timingverhalten untersucht bzw. das korrekte Verhalten der Software permanent überwacht werden.

5. Timingspezifikation

Mit der „Timing Extension“ der AUTOSAR 4.0 Release haben Entwickler erstmals die Möglichkeit, Timinganforderungen mittels AUTOSAR genau zu spezifizieren. Auch wenn einige der Möglichkeiten zu komplex sind, um sich in den nächsten Jahren in der Praxis durchzusetzen, sind die wichtigsten Aspekte enthalten und finden bereits Einzug in die ersten Lastenhefte. So können zum Beispiel Latenzzeiten von Wirkketten oder maximal zulässige Jitter, wie sie sich aus den physikalischen Anforderungen an einen Regelalgorithmus ergeben,

mittels AUTOSAR Timing Extension formuliert werden. Bisher wurden im Lastenheft vom OEM oft nur Forderungen nach einer maximalen CPU Auslastung für die verschiedenen Integrationsstufen festgehalten. Das ist sehr einfach und greifbar, doch eben nicht ausreichend.

Der beste Weg zur Übernahme der AUTOSAR Ideen in die Lastenhefte der Automobilwelt besteht darin, etabliertes Vorgehen mit den neuen Ansätzen zu verbinden. Vor diesem Hintergrund möchten wir empfehlen, die AUTOSAR Timing Extension in einer der nächsten Releases um die Möglichkeit einer Spezifikation der CPU Auslastung ergänzt werden. Inklusiv einer genauen Definition, was unter CPU Auslastung zu verstehen ist.

Zusammen mit der BMW Car IT hat Gliwa einen Prototyp entwickelt, der den Brückenschlag von der Spezifikation hin zur Verifikation darstellt und gewissermaßen die linke Seite des V-Modells mit der rechten verbindet. Mit ***Artime*** hat die BMW Car IT ein Artop Plug-in geschaffen, dass es Entwicklern ermöglicht, AUTOSAR 4.0 Timing Extension Timinganforderungen bequem zu formulieren. Diese Timinganforderungen werden vom Artop Plug-in ***ArT1*** eingelesen und in ***T1*** Timingconstraints übersetzt, die automatisch zur Laufzeit vom ***T1*** Targetcode überwacht werden.

6. Ausblick

Mit den kommenden Multicoreprozessorsystemen zeichnet sich eine Komplexität bei Timingfragen ab, die ohne Toolunterstützung nicht mehr wirtschaftlich in den Griff zubekommen sein wird. Für optimale Toolunterstützung ist es erforderlich, dass sich Prozessorhersteller, OS Anbieter, Debuggerhersteller, Compileranbieter und Timingtoolhersteller eng abstimmen. Dies kann gerne als Aufruf verstanden werden, die jetzt schon gute Zusammenarbeit weiter auszubauen.

Literaturangaben

[1] H. Sarnowski, P. Gliwa, M. Jersak, K. Richter. Laufzeitanalysen zur frühzeitigen Absicherung von Software. ATZelektronik, Nr. 1, 2009

[2] ALL-TIMES: Integrating European Timing Analysis Technology. Research project within the European Commission's 7th Framework Programme on Research, Technological Development and Demonstration. www.all-times.org

[3] Real-Time Experts, http://www.real-time-experts.com/

[4] M. Rudorfer, C. Knüchel, S. Voget, S. Eberle, and A. Loyer. Artop - an Ecosystem Approach for Collaborative AUTOSAR Tool Development. In Proceedings of ERTS2, 2010