ENTWICKLUNGS-TOOLS

Tracing von eingebetteten Multicore-Systemen

Multicore-Prozessoren bieten zwar mehr Rechenleistung als Single-Core-Architekturen, doch ist die Wahrscheinlichkeit von schwer erkennbaren und durch Nebenläufigkeit bedingter Fehler höher. Dieser Artikel präsentiert eine neue Technologie, die zur Messung des Zeitverhaltens von Programmen auf Mehrkern-Architekturen, zur Messung von Testüberdeckung (Coverage) im Zielsystem und zur Analyse komplexer Fehler verwendet werden kann.

Bild 1: CEDARtools ist ein Werkzeug für die Analyse komplexer eingebetteter Systeme entwickelt, mit dem das Problem der begrenzten Größe der Trace-Buffer gelöst wird. Wichtige OEMs und Tier1 evaluieren und verwenden das System bereits. © Accemic Technologies

Seit vielen Jahren übernehmen Steuergeräte im Automobil immer mehr und rechenintensivere Aufgaben – und kein Ende dieses Trends ist in Sicht. Mit der zunehmenden Komplexität steigt natürlich die Fehlerwahrscheinlichkeit und es sinkt die Fehlerbeseitigungseffizienz [1]. Die stetig steigende Rechenlast macht die Verwendung von Multicore-Prozessoren für viele Anwendungen unvermeidbar. Der Übergang von der sequentiellen Abarbeitung zu parallelisierter Abarbeitung kann aber zu sehr schwer reproduzierbaren Fehlern führen, wenn der Code-Bestand nie für eine parallele Ausführung gedacht war.

Zweifelsfrei ist in der Automobilbranche aus Haftungsgründen besondere Sorgfalt bei der Software-Erstellung und beim Test notwendig. Tests und Debug-Umgebungen machen aber nur dann Sinn, wenn man zu Fehlersymptomen auch Fehlerursachen zuordnen kann. Der Schlüssel liegt also in der Beobachtbarkeit. Im Gegensatz zu traditionellen Singlecore-Systemen, stellen uns aktuelle Multicore-Architekturen vor besondere Herausforderungen. Je höher integriert ein solches System ist, desto schwieriger wird es, interne Vorgänge zu verstehen. Genau an dieser Stelle unterstützt die in diesem Artikel vorgestellte Technologie.

Bewährte und neue Hilfsmittel

Weit verbreitet und etabliert zur Beobachtung von internen Vorgängen ist die Software-Instrumentierung. Hier protokolliert (automatisch) hinzugefügter Code die Ausführung des Programms. Zum Beispiel um Laufzeiten von Code-Blöcken zu messen oder um die Überdeckung von Tests zu ermitteln. Instrumentierung benötigt aber Platz und beeinflusst das zeitliche Verhalten der Programmausführung. Doch es gibt eine Alternative: Fast alle modernen Prozessoren verfügen serienmäßig über eine Embedded-Trace-Einheit (z. B. Intel Proces-

sor Trace [2]), die über eine spezifische (z. B. AGBT) oder eine Standard-Schnittstelle (z. B. USB DCI, PCIe) kommuniziert. Diese gibt Informationen über den abgearbeiteten Programmfluss preis, ohne dass dieser beeinflusst wird. Je nach Architektur und Trace-Konfiguration können auch das zeitliche Verhalten sowie Datenzugriffe rekonstruiert werden. Zudem ermöglichen viele Trace-Einheiten die leichtgewichtige (und damit im Release-Code akzeptable) hardwareunterstützte Instrumentierung (z. B. PTWrite, MIPI STP [4]) sowie das Tracen von Peripherieeinheiten (Speicherkontroller, Kommunikationseinheiten).

Bei heute üblichen Embedded-Trace-Lösungen werden die sehr breitbandigen Trace-Daten (einige Gbit/s) in einem Pufferspeicher abgelegt. Nach dem Ende des Testlaufs wird auf einem PC der Programmfluss rekonstruiert und die strukturelle Überdeckung ermittelt. Die Grenzen dieses Verfahrens liegen in der durch den notwendigen Zwischenspeicher begrenzten Beobachtungsdauer sowie der zusätzlich erforderlichen Rechenzeit für die Offline-Rekonstruktion des Programmablaufs.

Die Live-Analyse stellt eine Weiterentwicklung der zuvor vorgestellten Offline-Analyse der aufgezeichneten Trace-Daten dar. Hierbei sind zwei technische Herausforderungen zu meistern. Einerseits muss der hochkomprimierte Trace-Datenstrom vorverarbeitet und der Kontrollfluss der CPU(s) rekonstruiert werden. Dies muss auch für schnelle CPUs (>1 GHz) und für verschiedene Betriebssysteme funktionieren. Andererseits ist der generierte Ereignisstrom zu analysieren, beispielsweise durch die Aufzeichnung von Sprunginformation zur Analyse der strukturellen Testüberdeckung oder durch das dynamische Monitoring einer Vielzahl von Eigenschaften

```
17                                  // Unfold Collatz sequence and return its length.
18                                  // - n <= 1 will not execute the while loop at all.
19                                  // - n = 2^k will never trigger the 3*n+1 path.
20                          1970    unsigned collatz_depth(unsigned n) {
21                          1970    unsigned depth = 0;
22              [+,+]       13987   while(n > 1) {
                            ↪ 1970    40089B: eb 24 jmp 4008c1
                              13987   4008C1: 83 7d ec 01 cmpl $0x1,-0x14(%rbp)
                [+,+]       1970 ↪
                              12017   4008C5: 77 d6 ja 40089d
23              [+,+]       12017   n = (n&1)? 3*n+1 : n/2;
                              12017   40089D: 8b 45 ec mov -0x14(%rbp),%eax
                              12017   4008A0: 83 e0 01 and $0x1,%eax
                              12017   4008A3: 85 c0 test %eax,%eax
                [+,+]       3152 ↪ 8865  4008A5: 74 0e je 4008b5
                              3152    4008A7: 8b 55 ec mov -0x14(%rbp),%edx
                              3152    4008AA: 89 d0 mov %edx,%eax
                              3152    4008AC: 01 c0 add %eax,%eax
                              3152    4008AE: 01 d0 add %edx,%eax
                              3152    4008B0: 83 c0 01 add $0x1,%eax
                            ↪ 3152    4008B3: eb 05 jmp 4008ba
                              8865    4008B5: 8b 45 ec mov -0x14(%rbp),%eax
                              8865    4008B8: d1 e8 shr %eax
                              12017   4008BA: 89 45 ec mov %eax,-0x14(%rbp)
24                            12017   depth++;
25                                    }
```

Bild 2: Object-Code-Coverage und ihr Mapping auf Source-Code-Ebene.

INFO

Anwendungsfall: Dynamische Softwarearchitektur

Seit einigen Jahren steigt die Relevanz von Software im Fahrzeug und es ist davon auszugehen, dass dieser Trend sich weiter fortsetzt, besonders beim hochautomatisierten Fahren sind viele Innovationen softwarebasiert [9] [10]. Im Zusammenspiel mit ebenso steigenden Time-To-Market-Anforderungen sehen sich OEMs einer Vielzahl von Herausforderungen bei der Absicherung der Systeme gegenübergestellt. Zu nennen ist neben der steigenden Komplexität der Softwarearchitekturen vor allem die Einbindung von Legacy-Anteilen bei gleichzeitiger Verteilung auf Multicore-Plattformen.

Das LET-Konzept (Logische Ausführungszeit [1]) [11] bietet ein Gerüst um den genannten Herausforderungen zu begegnen, hat jedoch eine notwendige Voraussetzung: Es muss sichergestellt werden, dass jeder Task sicher innerhalb eines zugewiesen Zeitfensters gerechnet wird. Statische Analysen liefern zurzeit keine zufriedenstellende Präzision, daher werden für die Absicherung der Deadline-Einhaltung zusätzlich Messverfahren eingesetzt. Besonders interessant sind in diesem Zusammenhang Methoden, die ohne Instrumentierung der Software auskommen, also den tatsächlichen Ist-Stand messen.

[1] Das LET-Konzept abstrahiert von der physischen Ausführungszeit auf einer bestimmten Plattform bis hin zu den Zeiten der Lese- und Schreibzugriffe, um den Entwurfsprozess zu vereinfachen. Die tatsächliche Ausführungszeit ist beliebig, solange sie innerhalb der beiden Zeitpunkte liegt.

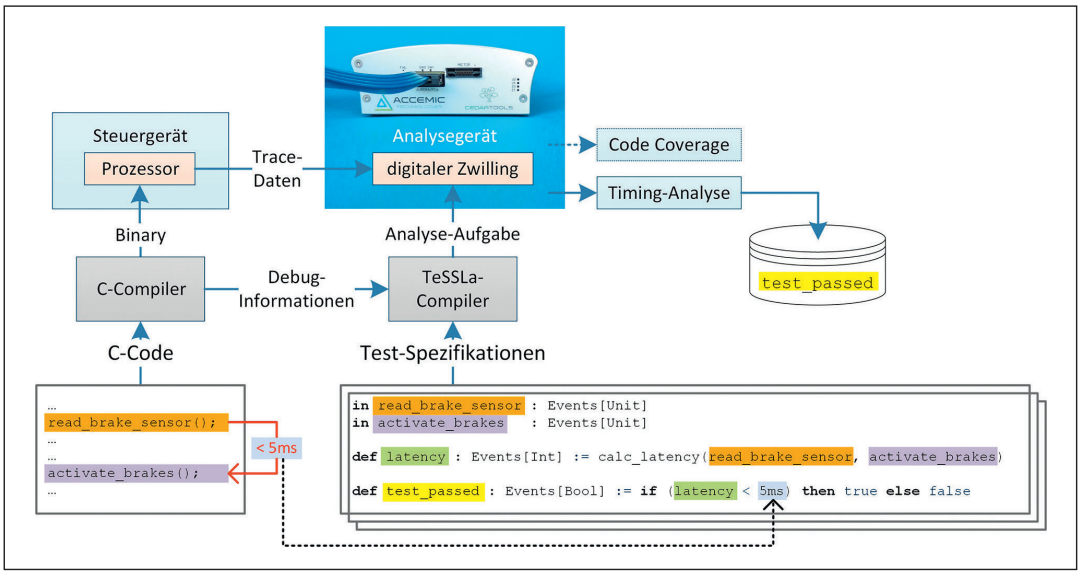

Bild 3: Beispiel der dynamischen Ereignisanalyse. © Accemic Technologies

mittels hochsprachlich konfigurierbaren Datenflussprozessoren.

Die Standards für die Entwicklung von sicherheitskritischen Systemen (z. B. [4]) definieren konkrete Anforderungen an den Testprozess, die anzuwendenden Testtechniken und den Nachweis der Vollständigkeit dieser Tests mittels der Messung der strukturellen Überdeckung. Bei letzterer muss je nach Kritikalität der Anwendung gezeigt werden, dass während der Tests alle Instruktionen (Statement Coverage), alle Sprünge (Branch Coverage) oder relevante Bedingungskombinationen für Verzweigungen (MC/DC) durchlaufen wurden.

Anwendungsfall Testüberdeckung

Generell lassen die Normen es weitgehend offen, auf welcher Testebene (Systemtest, Integrationstest, Modultest) die Messung der strukturellen Überdeckung erfolgen soll. Idealerweise wird die Anforderungsspezifikation in den Test mit einbezogen, sodass mit der strukturellen Überdeckung auch eine Aussage über die Qualität der Anforderungen getroffen werden kann.

Die Messung der strukturellen Abdeckung auf Object-Code-Ebene liefert, wie ausführlich in [5] diskutiert, andere Informationen als eine Messung auf Quellcode-Ebene. Durch Auswertung der vom Compiler generierten Debug-Informationen kann, wie in Bild 2 illustriert, die gemessene Object-Code-Coverage auf die von vielen Tools unterstützte Source-Code-Coverage abgebildet werden. Diese Auswertung der Debug-Information kann nun mit der Live-Analyse kombiniert werden: ohne jede Instrumentierung (und damit auch ohne Performanz-Einbußen) wird am Zielsystem die Testüberdeckung des Quellcodes ermittelt. Ein erstes Beispiel für solch ein Analysesystem ist das in Bild 1 dargestellte CEDARtools-System, welches die Live-Analyse mittels FPGA vornimmt.

Es spricht nun nichts dagegen die strukturelle Überdeckung von Tests nicht mehr, wie bisher üblich, beim Unit-Test zu erheben, sondern beim Systemtest. Auf diese Weise können Lücken im Systemtest gefunden und Zeit beim Unit-Test gespart werden.

INFO

Mit **CEDARtools** (Bild 1) hat Accemic Technologies ein Werkzeug für die Analyse komplexer eingebetteter Systeme entwickelt, mit dem das Problem der begrenzten Größe der Trace-Buffer gelöst wird. Wichtige OEMs und Tier1 aus dem Automobil- und Luftfahrtbereich evaluieren und verwenden das System bereits. Anstatt die Trace-Daten zu speichern und später offline zu analysieren, wird der Multi-Gbit-Datenstrom mittels ausgefeilter Hardware live analysiert. Dabei sind folgende Herausforderungen zu meistern:

- Live-Rekonstruktion des Kontroll- und Datenflusses einer oder mehrerer CPUs anhand von Trace-Daten –
- hochsprachlich konfigurierbare Live-Analyse des erzeugten Ereignisstroms, gleichzeitige Untersuchung einer Vielzahl von komplexen Eigenschaften (minimale und maximale Laufzeiten in Wirkketten, Statistik, Reihenfolgen, Überprüfung von Wertebereichen etc.)
- kontinuierliche Live-Messung der strukturellen Testabdeckung

Der Paradigmenwechsel besteht nun darin, nicht mehr Unmengen an Trace-Rohdaten über lange Zeiträume zu sammeln und später zeitraubend auszuwerten, sondern die Trace-Daten on-the-fly anhand vordefinierter und hochsprachlich beschreibbarer Eigenschaften zu analysieren. Dies ist über Minuten, Stunden und Tage möglich. Dennoch muss auf die Rohdaten nicht verzichtet werden: Mittels komplexer Trigger kann genau definiert werden, welche für die Analyse tatsächlich relevanten Rohdaten sowie Ereignisse in einem vorhandenen Trace-Buffer (4 GByte) gespeichert werden. Unterstützte Prozessoren (teilweise noch in Entwicklung) sind u.a. ARM Cortex, Infineon Aurix, Power Architecture und Intel Atom. Unterstützte Trace-Schnittstellen (teilweise noch in Entwicklung): NEXUS, HSSTP, AGBT, PCIe, USB, Mictor, NEXUS, und serielle High-Speed-Schnittstellen (Aurora).

Anwendungsfall Dynamische Analyse

Ein wichtiges Element im Entwicklungsprozess ist die statische Analyse. Aufgrund fehlender oder zu stark vereinfachter Modelle stößt diese Methode zunehmend an ihre Grenzen, so dass die ergänzende dynamische Analyse eingebetteter Systeme immer wichtiger wird. Diese Methode kann ebenfalls mit Hilfe der zuvor skizzierten Hardware implementiert werden.

Dazu werden bei der kontinuierlichen Rekonstruktion des Kontrollflusses bestimmte Instruktionsadressen markiert. Bei der Ausführung dieser Instruktionen werden Elemente in den emittierten Ereignisstrom eingefügt, welche dann online auf bestimmte Eigenschaften untersucht werden können. Die verwendeten Ereignisverarbeitungseinheiten sind hochsprachlich (www.tessla.io, [6]) konfigurierbar, und es kann eine Vielzahl von temporalen Eigenschaften (z. B. mittels AUTOSAR TIMEX [7] oder Amalthea definiert) parallel überwacht werden. Dabei erfolgt im FPGA nur eine Parametrierung der Ereignisverarbeitungseinheiten, eine individuelle Synthese von Logikstrukturen ist nicht erforderlich. Somit kann eine Änderung der hochsprachlichen Eigenschafts-Beschreibung binnen Sekunden auf einen Trace-Datenstrom angewandt werden. Ein Beispiel hierfür ist in Bild 3 angegeben.
Die dynamische Analyse ist auch ein mächtiges Werkzeug bei der Fehlersuche, besonders bei komplexen nichtdeterministischen Fehlerbildern. Je nach Auftrittswahrscheinlichkeit und dem Anwendungsgebiet kann die Suche nach einem dieser Defekte schnell zu Kosten in sechsstelliger Höhe führen [8].

Designvorkehrungen

Bereits im Lastenhaft sollten geeignete Vorkehrungen definiert werden, um eine umfassende Beobachtbarkeit des Steuergerätes sowohl während der Tests als auch nach dem Release zu erreichen. Die Trace-Schnittstelle sollte sowohl während der Testphase als auch im Serienfahrzeug in bestimmten Steuergeräteversionen verfügbar sein. Zudem muss für Untersuchungen an Vorserien- und Serienfahrzeugen die störsichere Übertragung der Trace-Daten vom Steuergerät zum Analysegerät (welches meist im Fahrzeuginneren platziert ist) sichergestellt sein.

Zusammenfassung

Die mit zunehmender Komplexität kontinuierlich steigende Anzahl von Post-Release-Defekten erfordert die Anwendung neuer Testmethoden. Dazu zählen die Messung der strukturellen Testabdeckung sowie die automatisierte Ausführung von Laufzeitanalysen im voll integrierten System. Diese Analyse ist dank einer neuen Technologie nun ohne Software-Instrumentierung und damit ohne Beeinflussung des Laufzeitverhaltens möglich.

Zudem lässt sich damit auch nach dem Release eines Systems effizient die Ursache von komplexen Fehlerbildern analysieren. Um diese technische Möglichkeit nutzen zu können, muss der breitbandige Zugriff auf die vom Prozessor ausgegebenen Trace-Daten gegeben sein. ■ (oe)

www.accemic.com

Diese Arbeiten wurden mit Mitteln aus dem EU-H2020-Projekt 732016 „COEMS“ und aus dem BMBF-Projekt „ARAMiS 2“ mit der FKZ 01IS16025 gefördert. Eine tiefgreifende Untersuchung der Messung der strukturellen Testabdeckung in einem integrieren System erfolgt im Forschungsprojekt „CoCoSI“ (BMBF KMU Innovativ, FKZ 01IS19044) mit den Projektpartnern Accemic Technologies, Fraunhofer IESE, Heicon und Intel.

Quellenverzeichnis

[1] C. Jones and O. Bonsignour, The Economics of Software Quality. Addison-Wesley, 2011.

[2] Intel® 64 and IA-32 Architectures Software Developer's Manual. Intel Corporation, 2016.

[3] 'Specification for System Trace Protocol (STP)'. MIPI Alliance, Inc., 2015.

[4] 'ISO 26262:2018. Road vehicles – Functional safety'. International Organization for Standardization Std., 2018.

[5] 'Position Paper CAST-17: Structural Coverage of Object Code'. FAA Certification Authorities Software Team (CAST), Jun-2003.

[6] L. Convent, S. Hungerecker, M. Leucker, T. Scheffel, M. Schmitz, and D. Thoma, 'TeSSLa: Temporal Stream-Based Specification Language', in Formal Methods: Foundations and Applications, Cham, 2018, pp. 144–162.

[7] 'AUTOSAR-TIMEX: Specification of Timing Extensions'. [Online]. Available: http://www.autosar.org/.

[8] B. Hanke and F. Schulz, 'Master Thesis: Assessment of multi-core integration infrastructure', University of German Armed Forces Munich, Munich, 2014.

[9] Y. Dajsuren and M. van den Brand, Eds., 'Automotive Software Engineering: Past, Present, and Future', in Automotive Systems and Software Engineering – State of the Art and Future Trends, Springer, 2019, pp. 3–8.

[10] P. Mallozzi, P. Pelliccione, A. Knauss, C. Berger, and N. Mohammadiha, 'Autonomous Vehicles: State of the Art, Future Trends, and Challenges', in Automotive Systems and Software Engineering – State of the Art and Future Trends, Y. Dajsuren and M. van den Brand, Eds. Springer, 2019, pp. 347–367.

[11] T. A. Henzinger, B. Horowitz, and C. M. Kirsch, 'Giotto: a time-triggered language for embedded programming', Proc. IEEE, vol. 91, no. 1, pp. 84–99, 2003.

Max Jonas Friese ist Software-Architekt bei der Mercedes-Benz AG.

Dr. Stephan Grünfelder ist Test-Trainer bei embedded-test.webs.com.

Dr. Michael Paulitsch ist Dependability Research Architect, Principal Engineer bei der Intel Deutschland GmbH.

Dr.-Ing. Alexander Weiss ist Geschäftsführer der Accemic Technologies GmbH.