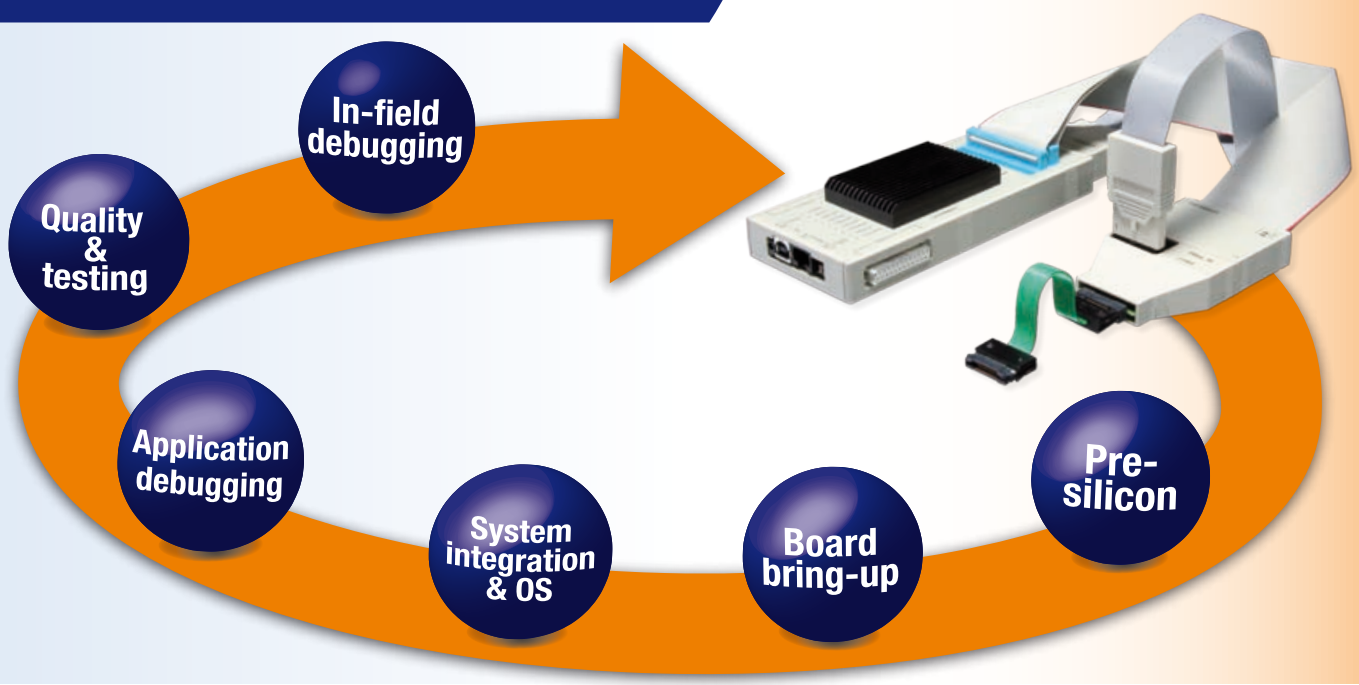
NEWS 2012

LAUTERBACH DEVELOPMENT TOOLS

DEBUGGER, REAL-TIME TRACE, LOGIC ANALYZER



Ein Debugger für alle Projektphasen

Embedded Designs werden immer komplexer. Der Zeitdruck wächst stetig. Um Projekte dennoch termingerecht fertigzustellen, setzen Projektleiter auf Debug- und Tracetools, die den Entwickler durch alle Projektphasen begleiten.

Die gleiche Bedienphilosophie, ein einheitliches Look-and-Feel, erweiterbare Skripte – all das verkürzt den Einarbeitungsprozess, festigt das Erfahrungswissen und schafft Raum für die eigentliche Entwicklungsarbeit. Bewährte Begleiter sind die TRACE32 Debug- und Tracetools von Lauterbach. Entwickler mit über 10 Jahren TRACE32-Erfahrung sind keine Seltenheit. Was aber zeichnet TRACE32 aus?

- Hardware- und Software-basierte Tools
- Frühzeitiger Support für neue Prozessoren
- Großes Portfolio an unterstützten Prozessoren
- Umfangreiche Test- und Analysefunktionen
- Nahtlose Integration in die Embedded Toolchain
- Kontinuität in der Bedienphilosophie, beim Look-and-Feel und bei der Skriptsprache

Hardware- und Softwaretools

Das Kerngeschäft von Lauterbach sind hardware-basierte Debug- und Tracetools, aber Lauterbach bietet seit über 20 Jahren auch Logikanalysatoren an. Hauptmerkmal der TRACE32 Logikanalysatoren ist ihre nahtlose Integration in die hardware-basierten Debug- und Tracetools. Ein typisches Anwendungsszenario für einen TRACE32 Logikanalysator stellt der Artikel „JTAG-Signale überprüfen“ auf Seite 6 vor.

Schnelle, leistungsfähige Rechner erfüllen zunehmend die Voraussetzungen für PC- oder Workstation-basierte Simulationen und Validierungen. Von besonderer Bedeutung für die Entwicklung von Embedded Produkten sind die Presilicon-Validierung von Prozessoren und SoCs, sowie die Presilicon-Software-Entwicklung mit virtuellen »

INHALT

Targets. Um auch für diese Phasen des Entwicklungsprozesses Tools zur Verfügung stellen zu können, bietet Lauterbach reine Software-Lösungen an.

Presilicon-Validierung

Für die Halbleiterhersteller ist es wichtig, ihre neuen Prozessoren oder SoCs vor der eigentlichen Herstellung zu validieren. Intensiv getestet werden einzelne Komponenten, wie beispielsweise die JTAG-Schnittstelle, der gesamte Core oder das Zusammenspiel zwischen Core und Peripherie.

Klassisch verwendet man dazu Emulatoren, wie Palladium oder FPGA-Prototypen, an die hardware-basierte TRACE32 Debugtools angeschlossen werden. Im Unterschied zum realen Prozessor läuft hier einfach alles sehr viel langsamer.

Erste Validierungen von Verilog- bzw. SystemC-Modellen führt man heute bereits direkt auf einem PC oder einer Workstation durch. Für eine solche Validierung kann keine Debug-Hardware benutzt werden. Lauterbach hat deshalb 2011 seiner Software ein Verilog Back-End hinzugefügt, das auf Signalebene eine JTAG-Schnittstelle simuliert (siehe Bild 1).

Der Einsatz von TRACE32 Tools in der Presilicon-Validierung bildet eine wichtige Grundlage für die frühzeitige Unterstützung neuer Prozessoren und SoCs:

- Getestete Tools stehen zur Verfügung, noch bevor das erste Silizium die Fabrik verlässt.
- Expertenwissen über den neuen Prozessor/SoC ist vorhanden und kann vom Kunden abgerufen werden.
- Start-up Skripte für den TRACE32 Debugger stehen bereit.

Virtuelle Targets

Um mit dem Start der Softwareentwicklung nicht auf die ersten Hardware-Prototypen warten zu müssen, wird heute zunehmend mit virtuellen Targets gearbeitet. Sobald das virtuelle Target fertig programmiert ist, kann mit dem Debuggen von Boot-Loader, Betriebssystem und den Anwendungen begonnen werden.

Für das Debuggen und Tracen stellen die meisten virtuellen Targets ihre eigene API bereit. Ist das nicht der Fall, kann auf die standardisierte MCD-API zurückgegriffen werden (**http://www.lauterbach.com/mcd_api.html**). Da viele neue Projekte heute Multicore-Chips einsetzen, hat Lauterbach sein Multicore-Debugging für virtuelle Targets 2011 weiter ausgebaut.

60+ unterstützte Prozessorarchitekturen

Es gibt keinen im Embedded Markt gängigen Prozessor oder SoC für den Lauterbach keine Tools bereitstellt. Für eine Reihe von Cores ist Lauterbach sogar der einzige Anbieter. Standardkontroller, DSPs, FPGA-Softcores, konfigurierbare Cores – alles lässt sich zu einem Multicore-Chip zusammenfügen und mit einem TRACE32 Tool debuggen.

Auch 2011 hat Lauterbach wieder Anpassungen für eine Vielzahl neuer Prozessoren und Multicore-Chips durchgeführt. Eine Übersicht dazu finden Sie in der Tabelle auf Seite 4.

Test- und Analysefunktionen

Jede Projektphase erfordert eigene Test- und Analysefunktionen. Die TRACE32 PowerView GUI stellt dazu eine

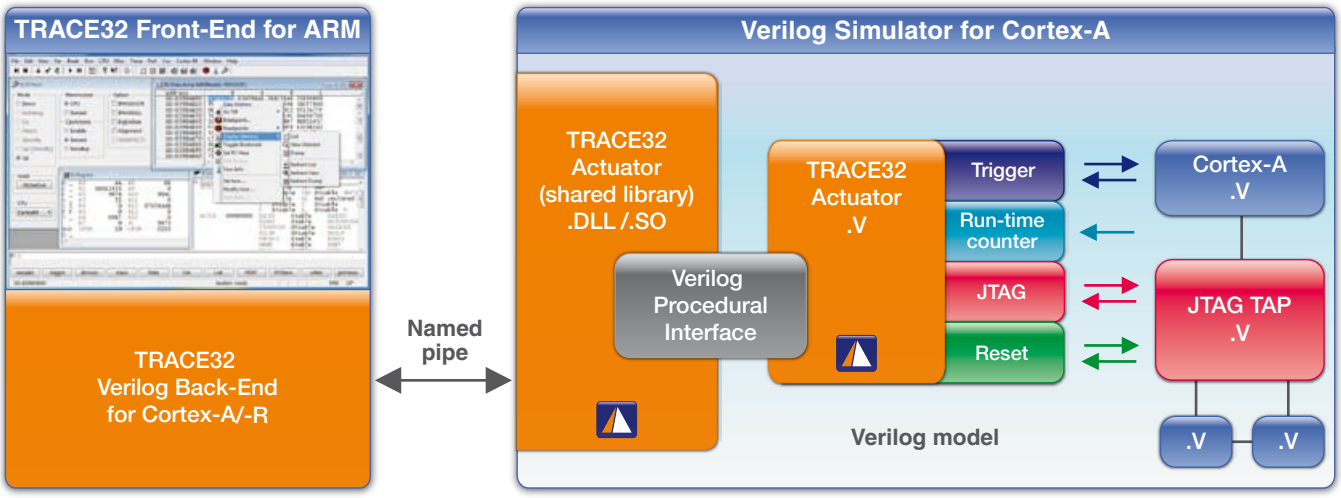


Bild 1: Für jede Benutzereingabe im TRACE32 Front-End erzeugt das Verilog Back-End die JTAG-Signale zur Validierung des Modells.

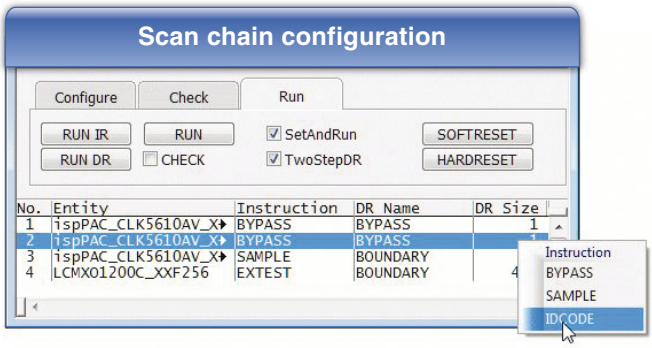


Bild 2: Für die Hardware-Inbetriebnahme kann der Entwickler Boundary-Scan Kommandos nutzen.

umfangreiche Auswahl von Kommandos und Menüs zur Verfügung. Boundary-Scan Kommandos (siehe Bild 2) und Kommandos zur Manipulation der JTAG-Pins sind Beispiele für Low-Level-Kommandos.

High-End-Kommandos unterstützen den Entwickler hauptsächlich bei der Qualitätssicherung. Typischerweise handelt es sich hier um hochentwickelte Kommandos zur Auswertung von Tracedaten. Code-Coverage-Analysen, Funktionslaufzeitmessungen oder Energy-Profiling sind typische Beispiele.

Seit Jahresbeginn 2011 bietet Lauterbach für alle Prozessorarchitekturen die Möglichkeit, die Traceinformationen zur Aufzeichnungszeit auf den Hostrechner zu streamen. Auf diese Weise lassen sich erheblich mehr Diagnosedaten aufsammeln. Qualitätssicherung wird so noch einfacher. Mehr Informationen zu diesem Thema finden Sie im Artikel „Code-Coverage - Einfach und aussagekräftig“ auf Seite 7.

Integration in die Embedded Toolchain

Die TRACE32 Software ist so offen gestaltet, dass sie mit allen gängigen Basiskomponenten eines Embedded Designs reibungslos zusammenarbeiten kann. Dies umfasst:

- Host-Betriebssysteme
- Programmiersprachen und Compiler
- Target-Betriebssysteme
- Virtuelle Maschinen wie die *Android VM Dalvik*

Die offene TRACE32 API erlaubt zudem ein nahtloses Zusammenspiel mit einer Vielzahl von Third-Party Tools. Beispiele sind spezielle IDEs wie Eclipse, graphische Programmierwerkzeuge und externe Profilingtools. Auch hier gab es 2011 einige Neuheiten.

Prism, das Parallelisierungs-Tool der schottischen Firma CriticalBlue unterstützt Entwickler bei der Parallelisierung von sequentiellen Funktionen. Es ermöglicht zunächst verschiedene Parallelisierungsstrategien ohne Änderungen am Funktionscode auszuprobieren. Wurde die optimale Strategie ermittelt, kann die Parallelisierung schrittweise, ebenfalls unterstützt von Prism, durchgeführt werden. Seit Juli 2011 bietet Lauterbach die Möglichkeit, Traceinformationen im Prism-Format zu exportieren und liefert so die Ausgangsdaten für die Parallelisierung.

Eine weitere Neuheit ist die Integration zwischen MATLAB Simulink® und TRACE32. Der Artikel „Simulation und Realität rücken näher zusammen“ auf Seite 14 stellt diese Integration ausführlich vor.

Hohe Kontinuität

Es ist bei Lauterbach Praxis, beim Umstieg auf eine neue Technologie, lange Übergangsphasen zu gewähren. Kein Kunde soll gezwungen sein, mitten in einer heißen Projektphase eine Technologieanpassung vornehmen zu müssen. Auch hierfür ein Beispiel:

Ab Mai 2012 wird Lauterbach seine graphische Bedienoberfläche TRACE32 PowerView auch in einer QT-basierten Version zur Verfügung stellen (siehe Bild 3). Damit kann eine zeitgemäße GUI für Linux, Mac OS X und andere Host-Betriebssysteme angeboten werden.

Damit jeder Kunde den passenden Zeitpunkt für einen Umstieg selbst festlegen kann, wird Lauterbach die Motif-Version von TRACE32 PowerView auch weiterhin unterstützen.

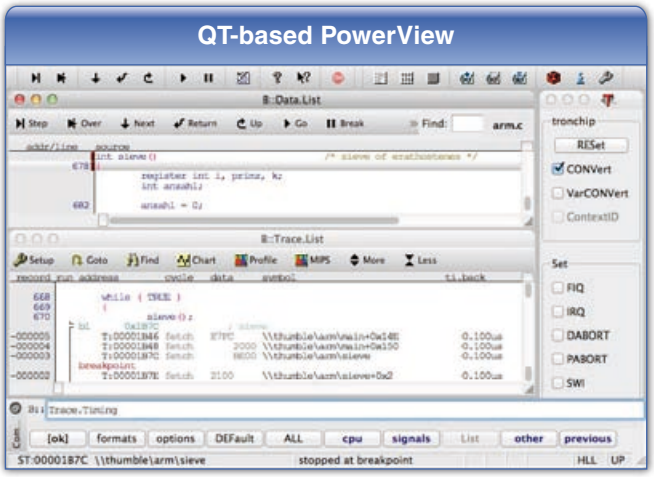


Bild 3: Die neue QT-basierte GUI für Linux, Mac OS X und andere Betriebssysteme.

Auf den nächsten Seiten unserer NEWS 2012 werden Sie sicher noch viel Nützliches für Ihr aktuelles oder zukünftiges Projekt finden. Wir würden uns freuen, wenn das eine oder andere Feature Sie voranbringt. Einige unserer Innovationen werden wir Ihnen auf der *embedded world 2012* live präsentieren. Über Ihren Besuch an unserem **Stand 210** in **Halle 4** würden wir uns freuen.

Neu unterstützte Prozessoren

Neue Derivate

Altera	**Cortex-A/-R** • FPGA mit Cortex-A9 MPCore als Hardcore **MIPS32** • MP32
AppliedMicro	**PPC44x** • 86290/491/791 Q2/2012
ARM	**Cortex-A/-R** • Cortex-A7/Cortex-A7 MPCore • Cortex-A15 • Cortex-A15 MPCore • Cortex-R5/Cortex-R5 MPCore • Cortex-R7/Cortex-R7 MPCore
Beyond Semiconductor	**Beyond** • BA22
Broadcom	**MIPS32** • BCM35230 • BCM63168, BCM63268 • BCM7231, BCM7358
Cavium	**MIPS64** • CN61XX/CN62XX/CN66XX • CN67XX/CN68XX
Ceva	**CEVA-X** • CEVA-XC
CSR	**ARM11** • QUATRO 4500
Cypress	**ARM9** • EZ-USB FX3
Energy Micro	**Cortex-M** • Giant Gecko
Freescale	**MCS08** • S08LG **MCS12X** • MC9S12VR, MC9S12XS • MM912F634 **Cortex-A/-R** • i.MX 6 Series **MPC55xx/56xx** • MPC5604E, MPC5675K, • MPC5676R **QorIQ 32-Bit** • P1010, P1020 • P2040, P2041 • P3041, P4040, P4080 • PSC9131
Freescale (Forts.)	**QorIQ 64-Bit** • P5010, P5020
Fujitsu	**Cortex-A/-R** • MB9DF126, MB9EF126
IBM	**PPC44x** • 476FP Q2/2012
Ikanos	**MIPS32** • Fusiv Vx185
Infineon	**TriCore** • TriCore Multi-Core Architecture
Intel®	**Atom™/x86** • Atom D2500, Atom N550 • Core i3/i5/i7 2nd Generation
Lantiq	**MIPS32** • XWAY xRX100
LSI	**PPC44x** • ACP344x Q2/2012
Marvell	**ARM9 Debug-Kabel** • 88E7251 **ARM11 Debug-Kabel** • 88AP610-V6, MV78460-V6 **Cortex-A/-R Debug-Kabel** • 88AP610-V7, MV78460-V7
Nuvoton	**Cortex-M** • NuMicro
NXP	**Cortex-M** • LPC12xx **Beyond** • JN5148
Qualcomm	**MIPS32** • AR7242 **Cortex-A/-R** • Krait
Renesas	**V850** • V850E2/Fx4: 70F3548..66 70F4000..70F4011 • V850E2/Fx4-L: 70F3570..89 • V850E2/Px4: 70F3503/05 70F3507/08/09 **78K0R/RL78** • 78K0R/Kx3-C/L • RL78/G14, RL78/G1A • RL78/F12, RL78/I1A **H8SX** • H8SX1725

Neue Derivate	
Renesas (Forts.)	**SH** • SH708x mit AUD/Onchip-Trace • SH7147
Samsung	**ARM7** • S3F4 **Cortex-A/-R** • S5PV310 **Cortex-M** • S3FM, S3FN
ST-Ericsson	**Cortex-A/-R** • A9500, A9540, M7400 **MMDSP** • A9500, A9540
STMicro-electronics	**MPC55xx/56xx** • SPC56A80, SPC56HK **Cortex-M** • STM32F2xx, STM32F4xx
Synopsys	**ARC** • ARC EM4, ARC EM6
Tensilica	**Xtensa** • BSP3, LX4, SSP16

Texas Instruments	**MSP430** • CC430Fxxx, MSP430FR5xxx • MSP430x1xx..MSP430x6xx **ARM9** • AM38xx • OMAP4460/4470 • TMS320C6A81xx • TMS320DM81xx **Cortex-A/-R** • AM335x, AM38xx • OMAP4460/4470/543x • RM48L950 • TMS320C6A81xx • TMS320DM81xx • TMS570LS3xxx **Cortex-M** • AM335x • OMAP4460/4470/543x • TMS470MFxxx **TMS320C28X** • TMS320C28346/F28069 **TMS320C6x00** • OMAP4460/4470/543x • TMS320C6A81xx • TMS320DM81xx • TMS320TCI6616/18
Xilinx	**Cortex-A/-R** • Zynq7000

Nexus-Trace auch für kleine Gehäuseformen

Die Nexuszelle, die in die MPC560xB/C-Controller von Freescale bzw. in die SPC560B/C-Controller von ST integriert ist, kann Tracedaten zur Instruktionsausführung generieren. Im Falle eines Betriebssystems kommen Informationen über die Taskwechsel hinzu.

Damit ein externes Tracetool wie TRACE32 diese Daten aufzeichnen kann, muss der Mikrocontroller über eine Traceschnittstelle verfügen. In ihren Standardgehäusen haben die Mitglieder der MPC560xB/C-Familie jedoch keine solche Schnittstelle. Um während der Entwicklung nicht auf wertvolle Daten über den Programmlauf verzichten zu müssen, werden silizium-kompatible Mikrocontroller in einem 208-Pin BGA *Development Package* angeboten. Diese Bauform verfügt über eine Nexus-Schnittstelle mit 4 MDO (*Message Data Out*) Pins.

Seit Mitte 2011 liefert Lauterbach MPC560xB/C-Adapter aus, die den Original-Controller auf dem Target durch einen 208-Pin Controller mit Nexus-Traceschnittstelle ersetzen. Diese Lösung muss man sich wie folgt vorstellen: Auf dem Target befindet sich ein Sockel der Firma Tokyo Eletech passend für eines der Standardgehäuse. Hier kann wahlweise der Original-Controller oder der Lauterbach MPC560xB/C-Adapter eingesetzt werden.

Der MPC560xB/C-Adapter selbst setzt sich aus folgenden Komponenten zusammen:

- Passender MPC560xB/C-Controller im 208-Pin BGA *Development Package* sowie Mictor-Stecker mit Nexus-Schnittstelle für den Anschluss eines TRACE32 Tracetools (blau dargestellt im Bild 4)
- Sockel-Adapter der Firma Tokyo Eletech

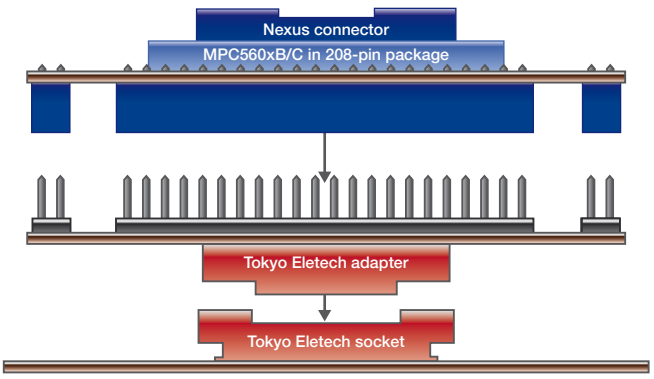


Bild 4: Der MPC560xB/C-Adapter erlaubt statt dem Original-Controller ein Development Package mit Nexus-Schnittstelle einzusetzen.

JTAG-Signale überprüfen

Der PowerTrace II von Lauterbach ist mit einem integrierten Logikanalysator ausgestattet. Der Lieferumfang des Tracetools enthält deshalb auch eine Standard-Probe. Damit lassen sich 17 digitale Kanäle mit einer Abtastrate von bis zu 200 MHz aufzeichnen.

Ein typisches Anwendungsszenario für diesen Logikanalysator mit einer Speichertiefe von bis zu 1024 KSamples ist die Überprüfung der JTAG-Signale während der Presilicon-Validierung und bei der Inbetriebnahme von Evaluation Boards (siehe Bild 6 und 7).

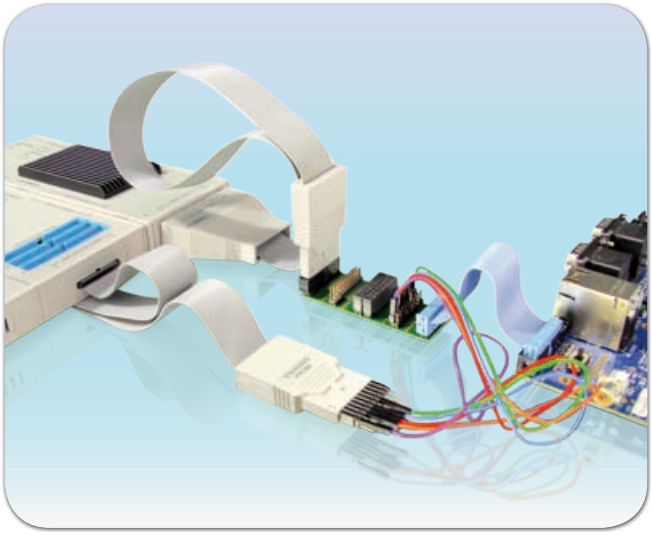


Bild 5: Messanordnung zur Aufzeichnung der JTAG-Signale.

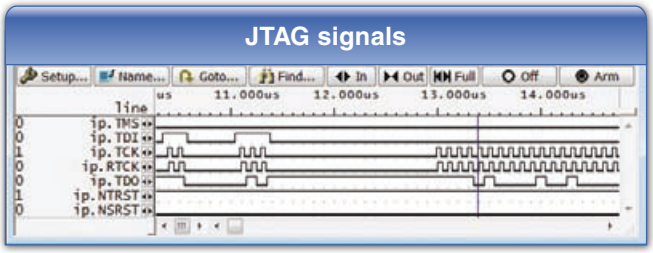


Bild 6: Die aufgezeichneten JTAG-Signale.

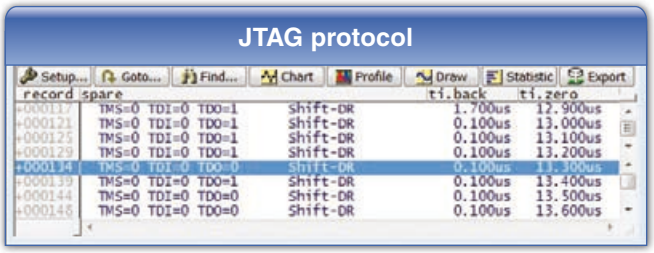


Bild 7: Die Protokoll-Darstellung der JTAG-Signale.

Erweiterungen der Target-OS Awareness

Folgende Versionsanpassungen wurden durchgeführt:

- eCos 3.0
- embOS 3.80
- FreeRTOS v7
- Linux v3.0
- MQX 3.6
- RTEMS 4.10
- SMX v4

- Der Inhalt des *QNX tracelogger* kann über die TRACE32 QNX OS-Awareness angezeigt werden. Eine graphische Darstellung von Taskwechseln ist über die TRACE32 Kommandogruppe LOGGER.Chart möglich.

- Die TRACE32 QNX OS-Awareness wurde für *position independent executables* angepasst.

Neu unterstützte Target-OS	
µC/OS-II für Andes	verfügbar
µC/OS-II für Blackfin	verfügbar
Elektrobit tresos (OSEK/ORTI)	verfügbar
Erika (OSEK/ORTI)	verfügbar
FreeRTOS für AVR32	verfügbar
Linux für Beyond	geplant
MQX für ARC	verfügbar
OSEK/ORTI SMP	geplant
PikeOS	verfügbar
PXROS-HR für TriCore	verfügbar
PXROS-HR Run Mode Debugging	verfügbar
RTEMS für Nios II	verfügbar
Sciopta 2.x	verfügbar
SYS/BIOS für ARM	verfügbar
VxWorks SMP	verfügbar

Code-Coverage – Einfach und aussagekräftig

Seit März 2011 kann TRACE32 Traceinformationen bereits während der Aufzeichnung auf die Festplatte streamen. Die große Menge an Daten über den Programmablauf, die auf diesem Weg erfasst werden kann, führt zu einer erheblichen Vereinfachung der Code-Coverage-Analyse.

Trace-basierte Code-Coverage-Analyse

Für die Qualitätssicherung von sicherheitsrelevanten Produkten ist ein Nachweis von *Statement Coverage* und *Condition Coverage* oft zwingend vorgeschrieben.

- **Statement Coverage** weist nach, dass jede Codezeile während des Systemtests ausgeführt wurde.
- **Condition Coverage** weist nach, dass für jede bedingte Anweisung sowohl der Pass-, als auch der Fail-Zweig mindestens einmal durchlaufen wurde.

Für viele Embedded Systeme ist zudem gefordert, dass hochoptimierter Code in Echtzeit getestet wird. Codeinstrumentierung und Laufzeitverfälschung sind hier tabu.

Um diesen Anforderungen nachkommen zu können, muss der verwendete Prozessor/SoC folgende Voraussetzungen erfüllen:

1. Die eingesetzten Cores müssen eine Core-Tracelogik besitzen (siehe Bild 8). Eine solche Logik generiert Informationen über die Instruktionsausführung auf den Cores. Je nach Ausgestaltung der Tracelogik können Informationen über Prozesswechsel und durchgeführte Load/Store-Operationen hinzukommen.

2. Der Prozessor/SoC muss über einen Traceport mit ausreichender Bandbreite verfügen, damit die Traceinformationen verlustfrei durch ein externes Tool aufgezeichnet werden können.

Das klassische Messverfahren

Bisher wurde die Code-Coverage-Analyse mit TRACE32 in folgenden Schritten durchgeführt:

1. Programmausführung starten und automatisch anhalten, wenn der Tracespeicher voll ist.
2. Tracespeicherinhalt in die Code-Coverage-Datenbank übertragen.
3. Programmausführung fortsetzen.

Für jeden Messschritt war die Anzahl der erfassbaren Daten durch die Größe des vom Tracetool bereitgestellten Speichers begrenzt.

Die Ergebnisse der Code-Coverage-Analyse konnten nach Abschluss der Messung, bei Bedarf auch nach jedem Zwischenschritt, überprüft werden.

Neu: Streaming

Wenn die Tracedaten bereits zur Aufzeichnungszeit auf den Hostrechner übertragen werden, können alle notwendigen Daten **in einem Messschritt** erfasst werden. Die gestreamten Daten werden in einer Datei auf der Festplatte zwischengespeichert. Um zu verhindern, dass die Festplatte vollständig mit Tracedaten gefüllt wird, stoppt TRACE32 das Streaming sobald weniger als 1 GByte freier Speicher übrig ist.

Um das Streaming nutzen zu können, müssen folgende technische Voraussetzungen erfüllt sein:

- 64-Bit Hostrechner und 64-Bit TRACE32 Executable
- Möglichst schnelle Schnittstelle zwischen Tracetool und Hostrechner
- Optimale Konfiguration der Tracequelle und des Tracetools »

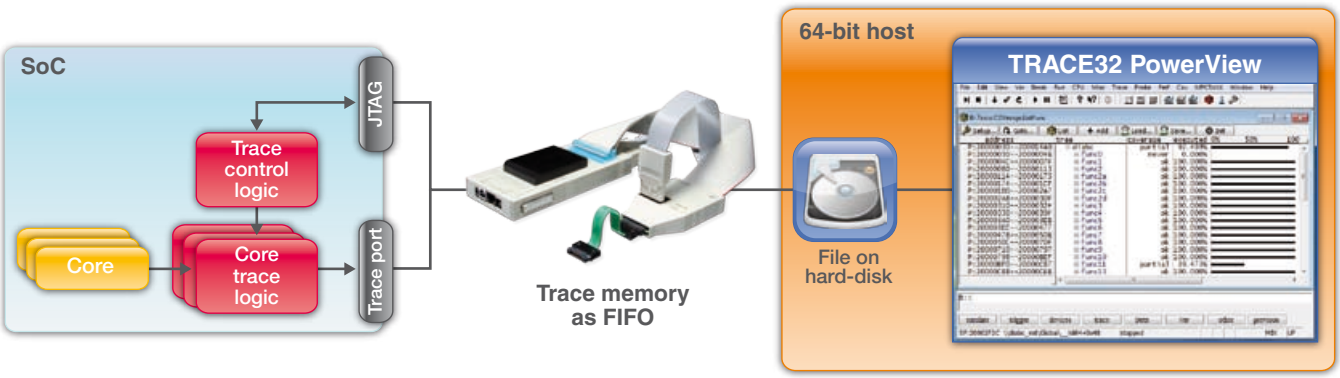


Bild 8: Für die Code-Coverage-Analyse können bis zu 1 TByte Tracedaten auf den Hostrechner gestreamt werden.

Schnelle Hostschnittstelle

Die Menge an Tracedaten, die über den Traceport exportiert wird, hängt zum einen von der Target-Hardware ab. Die Anzahl der Cores, die Anzahl der Traceport-Pins sowie die verwendete Traceclock sind wichtige Parameter, ebenso das von der Core-Tracelogik verwendete Protokoll. Das ARM PTM-Protokoll ist beispielsweise kompakter als das ARM ETMv3-Protokoll (siehe Bild 9).

Eine andere wichtige Einflussgröße ist die Embedded Software. Eine Software, die viele Sprünge durchführt und Daten/Instruktionen meist im Cache vorfindet, erzeugt mehr Tracedaten pro Sekunde als eine Software, die viele sequentielle Instruktionen abarbeitet und häufig auf die Verfügbarkeit von Daten/Instruktionen warten muss.

Die Datenmenge variiert, ist aber in jedem Fall groß. Das Streaming funktioniert nur dann, wenn die Übertragungsrate zwischen Tool und Hostrechner ausreicht, um alle am Traceport ausgegebenen Daten verlustfrei auf den Hostrechner zu übertragen. Für das TRACE32 Tool PowerTrace II ist deshalb die 1 GBit Ethernet-Schnittstelle die richtige Wahl.

Direkten Einfluss auf die Tracedatenmenge gewinnt man durch die Programmierung der Tracelogik. Diese sollte so programmiert sein, dass ausschließlich Informationen generiert werden, die für die Code-Coverage-Analyse relevant sind. Hierzu folgen nun zwei Beispiele.

ETM/PTM: Optimale Konfiguration

ETM und PTM sind unterschiedliche Implementierungen für die Core-Tracelogik der ARM/Cortex-Architekturen. Die ETM lässt sich so konfigurieren, dass nur Traceinformationen zur Instruktionsausführung erzeugt werden. Informationen über die durchgeführten Load/Store-Operationen sind für die Code-Coverage-Analyse nicht nötig. Da die PTM keine Informationen zu den Load/Store-Operationen erzeugen kann, muss diese nicht konfiguriert werden.

Beide Tracequellen codieren die Instruktionsadressen virtuell. Verwendet ein Embedded Design ein Betriebssys-

PowerTrace vs. PowerTrace II

TRACE32 Tracetools gibt es in zwei Bauformen, die sich vor allem bezüglich ihrer Kenndaten unterscheiden.

PowerTrace

- 256 oder 512 MByte Tracespeicher
- USB 2.x und 100 MBit Ethernet
- 80 MBit/s als maximale Übertragungsrate zum Hostrechner
- Software-Kompression der Tracedaten (Faktor 3)
- Speicherinterface mit 100 MHz

PowerTrace II

- 1/2/4 GByte Tracespeicher
- USB 2.x und 1 GBit Ethernet
- 500 MBit/s als maximale Übertragungsrate zum Hostrechner
- Hardware-Kompression der Tracedaten für ETMv3 und PTM (Faktor 6)
- Speicherinterface mit 233 MHz

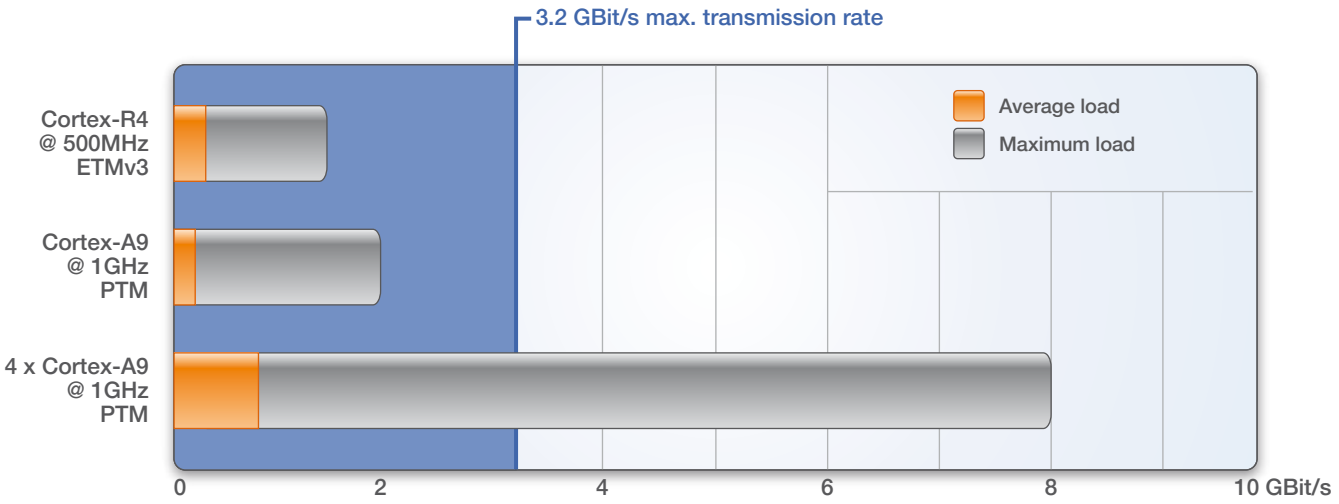


Bild 9: Eine Übertragungsrate von 3,2 GBit/s reicht in der Regel aus, um alle Informationen zum Programmablauf auf den Host zu streamen.

tem wie Linux oder Embedded Windows, sind die virtuellen Adressen nicht eindeutig auf physikalische Adressen umrechenbar. Die Tracequelle muss so konfiguriert sein, dass sie Informationen über den virtuellen Adressraum generiert, in dem eine Instruktion ausgeführt wurde.

Für die ARM ETM/PTM lässt sich die Tracedatenmenge weiter reduzieren:

- Die Code-Coverage-Analyse wertet keine Zeitinformation aus. Deshalb wird empfohlen, das TRACE32 Tracetool so zu konfigurieren, dass die Tracedaten ohne Zeitstempel auf den Host übertragen werden. Dadurch reduziert sich die Datenmenge um ein Drittel.

- Der PowerTrace II bietet zusätzlich noch eine FPGA-basierte Hardware-Kompression der Tracedaten. Damit lassen sich bis zu 3,2 GBit/s Tracedaten auf den Hostrechner übertragen. Bild 9 zeigt, dass diese Übertragungsrate für ein verlustfreies Streamen der ETM/PTM-Daten in der Regel ausreichend ist.

Nexus: Optimale Konfiguration

Die Core-Tracelogik nach dem Nexus-Standard ist unter anderem auf den Prozessoren der MPC5xxx Familie implementiert. Für die Code-Coverage-Analyse genügt eine Nexus Class 2 Tracezelle. Auch hier ist es ausreichend, Informationen über die Instruktionsausführung zu generieren. Die Tracedaten sind besonders kompakt, wenn man dazu *Branch History Messaging* verwendet. Eine Reduktion der Datenmengen um den Faktor 10 ist durchaus realistisch. Aktuell wird das Nexus-Streaming nur vom PowerTrace II unterstützt.

Selbstverständlich funktioniert das Streaming auch für alle anderen von TRACE32 unterstützten Prozessoren/SoCs mit Traceport.

Code-Coverage für Multicore-Systeme

TRACE32 unterstützt die Code-Coverage-Analyse auch für SMP-Systeme. Hier gilt es ebenso nachzuweisen, dass eine Instruktion ausgeführt wurde. Die Information, welcher Core dafür verantwortlich ist, spielt bei einem SMP-System keine Rolle. Bild 10 zeigt die Ergebnisse für ein SMP-System mit zwei Cortex-A9 MPCores.

Für das *Statement* und *Condition Coverage* ist hier eine bedingte Anweisung, bei der nur der Fail-Zweig durchlaufen wurde, gelb hinterlegt und mit *not exec* gekennzeichnet. *Detailed Coverage* listet in den Spalten *exec/notexec* auf, wie oft jede Anweisung bzw. jeder Zweig der Anweisung durchlaufen wurde.

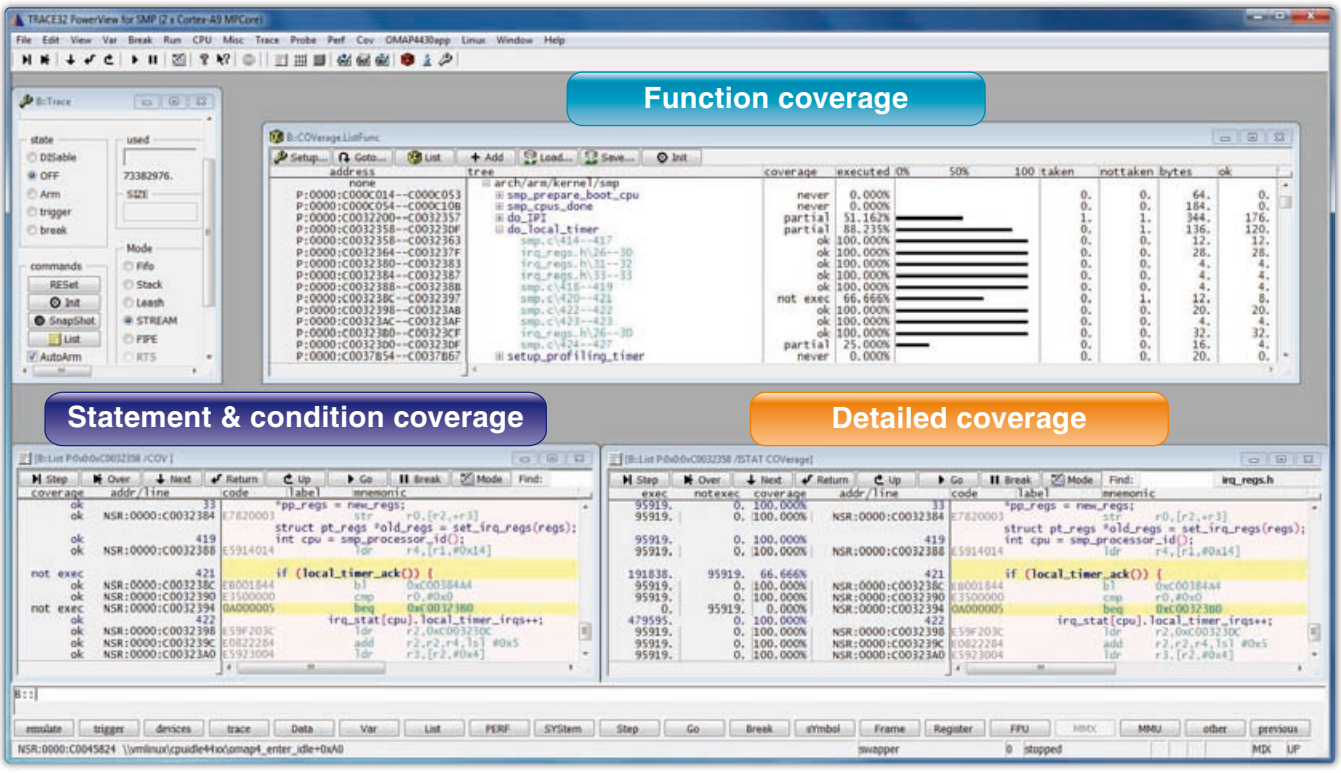


Bild 10: Code-Coverage-Analyse für ein SMP-System.

CoreSight Trace Memory Controller

Der neue CoreSight Trace Memory Controller – kurz TMC – bietet SoC-Designern mehr Gestaltungsfreiraum für die Trace-Infrastruktur. Erste Designs mit TMC werden bereits von TRACE32 unterstützt.

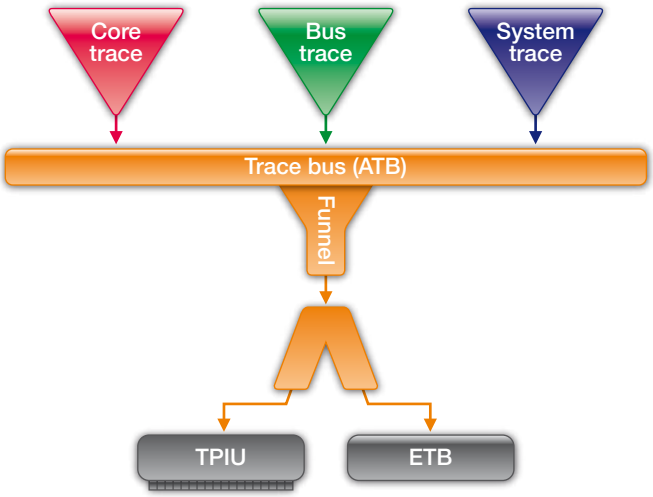


Bild 11: Alle von den Trace Macrocells erzeugten Tracedaten werden vom CoreSight Funnel zu einem einzigen Datenstrom zusammengefasst.

Unter CoreSight werden Diagnosedaten zur Analyse der SoC-internen Abläufe von so genannten *Trace Macrocells* erzeugt. Dabei können drei Typen unterschieden werden:

- **Core Trace Macrocells** sind einem Core zugeordnet und generieren Traceinformationen über die Instruktionsausführung auf diesem Core. Je nach Ausgestaltung der Tracezelle können Informationen über Prozesswechsel und durchgeführten Load/Store-Operationen hinzukommen.
- **Bus Trace Macrocells** sind fest einem Bus zugeordnet und generieren Traceinformationen über die dort stattfindenden Datentransfers.
- **System Trace Macrocells** generieren Traceinformationen für Hardware-Trigger *(system event tracing)* oder zu Diagnosedaten, die von der Anwendungssoftware durch Code-Instrumentierung erzeugt wurden.

Der *CoreSight Funnel* setzt alle Tracedaten zu einem Datenstrom zusammen (siehe Bild 11). Dieser Tracedatenstrom wird anschließend entweder in einem Onchip-Speicher abgelegt (ETB) oder über einen Traceport für die Aufzeichnung durch ein externes Tool bereit gestellt (TPIU). Die heute implementierten CoreSight Trace-IPs stoßen bei komplexen Multicore-SoCs, die viele *Trace Macrocells* enthalten, bisweilen an ihre Grenzen.

ARM CoreSight

ARM stellt mit CoreSight einen umfangreichen Satz von IP-Blöcken zur Verfügung, der es SoC-Designern erlaubt, eine maßgeschneiderte Debug- und Trace-Infrastruktur aufzubauen.

Eine einzige Debug-Schnittstelle genügt, um alle Cores des SoCs zu kontrollieren und zu koordinieren, sowie auf alle Speicher zuzugreifen.

Eine Traceschnittstelle reicht aus, um Diagnosedaten über die Abläufe innerhalb des SoCs ohne Echtzeitverletzung bereit zu stellen.

- **ETB:** Der Onchip-Tracespeicher ist oft zu klein, um eine ausreichende Menge an Tracedaten für die spätere Analyse aufzunehmen. Typische Größen für die ETB liegen heute immer noch zwischen 4 und 16 KByte.
- **TPIU:** Es entstehen immer wieder Systemzustände, in denen mehr Tracedaten generiert werden, als der Traceport ausgeben kann. CoreSight ist so konzipiert, dass Tracedaten aus den *Trace Macrocells* nur dann übernommen werden, wenn die TPIU sie auch ausgeben kann. Verbleiben die generierten Tracedaten zu lange in den *Trace Macrocells*, kommt es zu einem Überlauf der dortigen FIFOs und wichtige Daten können verloren gehen.

Für beide Szenarien soll der neue *CoreSight Trace Memory Controller* nun Abhilfe schaffen.

TMC als Embedded Trace Buffer

Um mehr Tracedaten Onchip für die Analyse bereitzustellen, kann der Chiphersteller theoretisch bis zu 4 GByte SRAM an den *Trace Memory Controller* anschließen (siehe Bild 12).

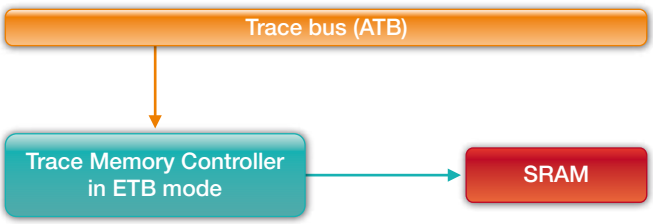


Bild 12: Im ETB-Mode kann der Trace Memory Controller bis zu 4 GByte Onchip-Tracespeicher zur Verfügung stellen.

TMC als Embedded Trace FIFO

Untersuchungen des von der TPIU exportierten Tracedatenstroms haben ergeben, dass die Bandbreite der meisten Traceports in der Regel ausreichend dimensioniert ist. Zu einer Überlastung und damit zum Verlust von Tracedaten kommt es nur, wenn Lastspitzen auftreten.

Der *Trace Memory Controller* lässt sich nun so in die Trace-Infrastruktur des SoCs integrieren, dass er als *Embedded Trace FIFO* Lastspitzen an der TPIU abpuffert (siehe Bild 13). Dabei ist der ETF so konzipiert, dass es dort nicht zum Verlust von Tracedaten kommen kann. Die Größe des ETF ist beim Design frei festlegbar (512 Bytes bis 4 GBytes).

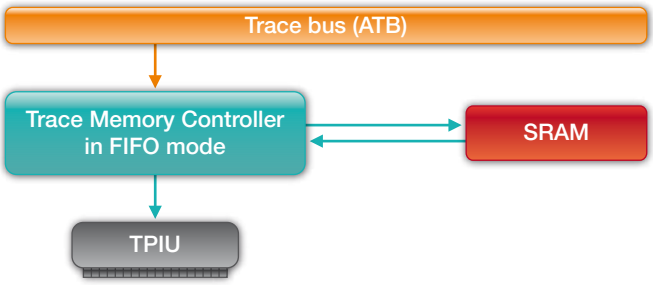


Bild 13: Im FIFO-Mode kann der Trace Memory Controller Lastspitzen an der TPIU abpuffern. Dadurch lässt sich der Verlust von Tracedaten verhindern.

Die beiden hier vorgestellten Integrationen des TMCs in die Trace-Infrastuktur eines SoCs sind einfache Beispiele. Selbstverständlich kann der TMC IP-Block wesentlich flexibler und komplexer in ein CoreSight-System eingebaut werden.

Anpassungen in TRACE32

Natürlich muss Lauterbach die TRACE32 Software für die Konfiguration und das Handling des *Trace Memory Controller* anpassen. Dies gilt insbesondere dann, wenn dieser auf neue, bisher noch nicht unterstützte Weise in den SoC integriert ist. Der TRACE32-Nutzer muss lediglich die Basisadresse für den TMC konfigurieren. Anschließend kann er alle bewährten Trace-Features wie gewohnt nutzen.

TMC als Router zum high-speed Link

Innerhalb der Embedded Community wird schon seit längerem die Möglichkeit diskutiert, sich von einem dedizierten Traceport zu lösen. Dafür gibt es sicher viele gute Argumente.

Mit dem *Trace Memory Controller* bietet ARM nun erstmals eine Anschlussmöglichkeit an high-speed Standard-

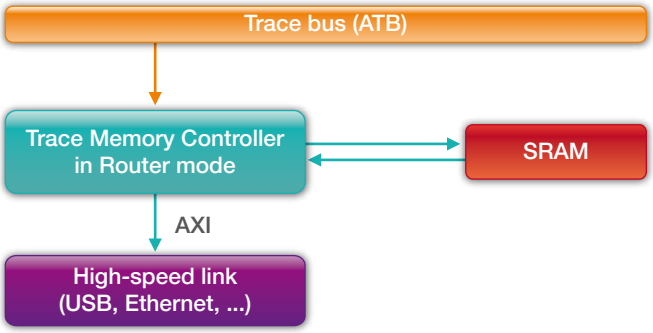


Bild 14: Im Router-Mode leitet der Trace Memory Controller die Tracedaten zum Export an eine schnelle Standard-Schnittstelle weiter.

Schnittstellen. USB bzw. Ethernet-Schnittstellen gelten als Favoriten, vor allem deshalb, weil sie in vielen Endprodukten sowieso zur Verfügung stehen. Idealerweise teilt sich das externe Tracetool die Schnittstelle dabei mit den anderen angeschlossenen Geräten.

Der TMC als so genannter *Embedded Trace Router* übernimmt innerhalb des SoCs die Aufgabe, die Tracedaten über den AXI-Bus für den Export an die IP der high-speed Schnittstelle weiterzureichen (siehe Bild 14).

Ein solcher Traceexport erfordert völlig neue Tracetools. Lauterbach steht bereits im intensiven Kontakt mit führenden Halbleiterherstellern, um für diesen Technologiewechsel passende Tools zu entwickeln.

TRACE32 CoreSight Features

- Offen für alle in CoreSight integrierbare Cores; Lauterbach bietet Debugger für alle ARM-/Cortex-Cores, für eine Vielzahl von DSPs, sowie für konfigurierbare Cores.
- Unterstützung für Asymmetrisches Multiprocessing (AMP) und Symmetrisches Multiprocessing (SMP)
- Debuggen über JTAG-Schnittstelle und 2-Pin *Serial Wire Debug*
- Synchrones Debuggen aller Cores und der Peripherie
- Unterstützung der CoreSight *Cross Trigger Matrix*
- Unterstützung für alle Arten von *Trace Macrocells* (ETM, PTM, HTM, ITM, STM und mehr)
- Tools für parallele und serielle Traceports
- Multicore-Tracing
- Umfassendes Profiling für Multicore-Systeme

Zielgenaue Traceauswertungen für Cortex-M3/M4

Fehlerbehebung, Performance-Tuning, Code-Coverage, all das lässt sich für ein Embedded System schnell durchführen, wenn einem die dafür notwendigen Traceauswertungen zur Verfügung stehen. Um für die Cortex-M3/M4 Prozessoren optimale Traceauswertungen anbieten zu können, hat Lauterbach 2011 neue Wege beschritten.

ETM und ITM zusammenführen

Für die Cortex-M3/M4 Prozessoren können Traceinformationen von zwei verschiedenen Quellen generiert werden (siehe Bild 17).

- **ETMv3:** Generiert Informationen zur Instruktionsausführung.
- **ITM:** Generiert Informationen über die Schreib-/Lesezugriffe mittels der *Data Watchpoint and Trigger Unit (DWT)*.

Die ITM-Tracepakete zu den Schreib-/Lesezugriffen enthalten folgende Informationen:

- Datenadresse
- Datenwert, gelesen bzw. geschrieben
- Program Counter

Unter Auswertung des *Program Counter* lassen sich die separat generierten Datenzugriffe nahtlos in den Programmablauf integrieren (siehe Bild 15), was zu einer wesentlichen Vereinfachung der Fehlersuche führt. Die Ursache dafür, dass ein falscher Datenwert auf eine Speicheradresse geschrieben wurde, lässt sich schneller ermitteln, wenn die Schreibzugriffe eingebettet in den gesamten Programmablauf dargestellt werden.

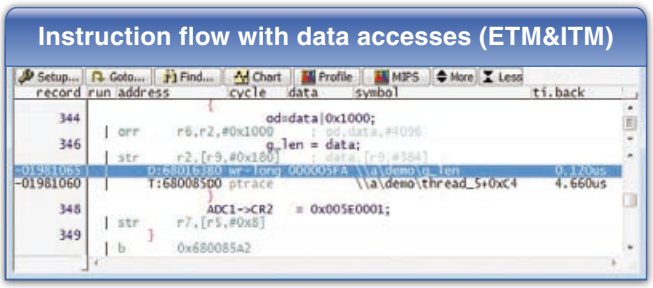


Bild 15: Durch das Zusammenführen von ETM- und ITM-Tracedaten lassen sich Schreib-/Lesezugriffe nahtlos in den Programmablauf integrieren.

OS-aware Tracen

Wenn auf dem Cortex-M3/M4 ein Betriebssystem läuft, sind Taskwechsel-Informationen für die Traceauswertung

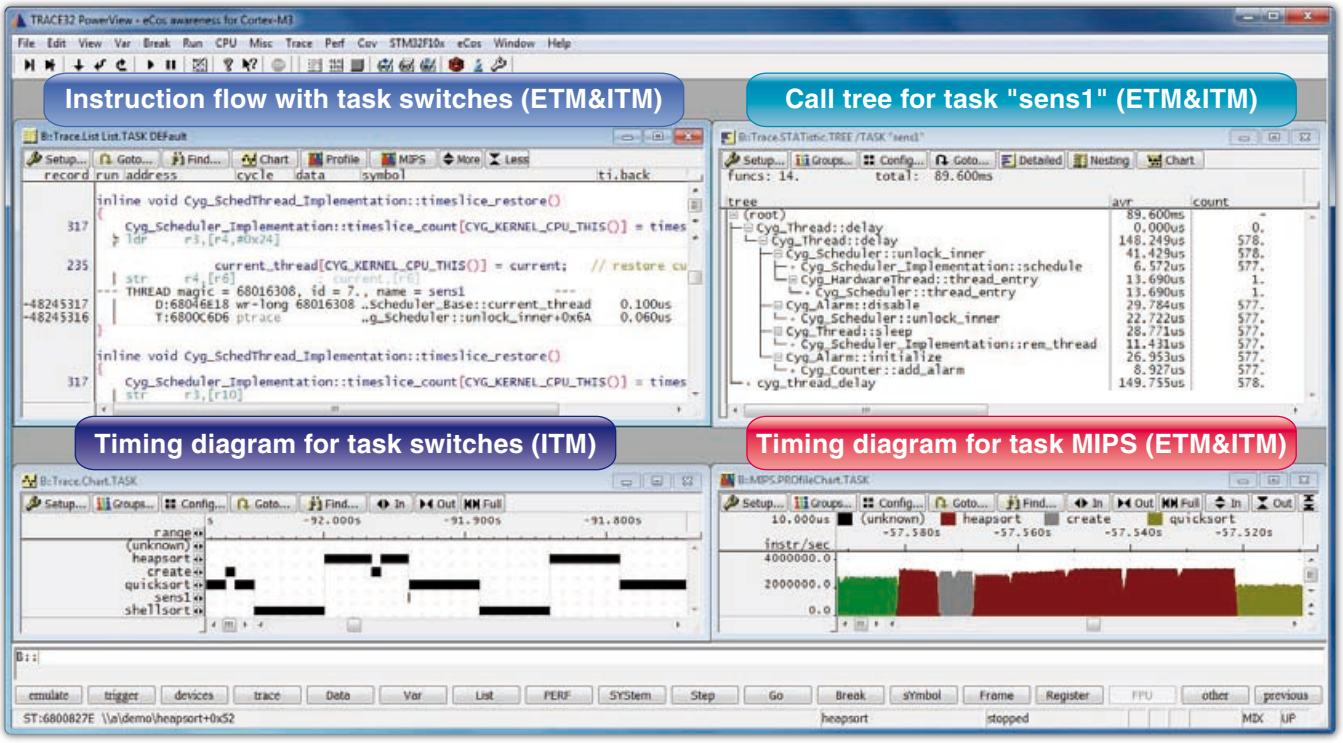


Bild 16: Durch das Zusammenführen von ETM- und ITM-Tracedaten können für das Betriebssystem eCos umfassende Traceauswertungen durchgeführt werden.

unverzichtbar. Eine Möglichkeit, Informationen über Taskwechsel zu erhalten, ist folgende: Mittels der ITM werden Traceinformationen für den Schreibzyklus generiert, in dem der Kernel die Kennung für den aktuellen Task auf die entsprechende OS-Variable schreibt. Als Schreibzugriff lässt sich der Taskwechsel, wie zuvor beschrieben, nahtlos in den Programmablauf integrieren. Dies verbessert die Lesbarkeit des Trace-Listings (siehe Bild 16). Die Integration der Taskwechsel in den Programmablauf bildet auch die Basis für die in Bild 16 gezeigten Laufzeitanalysen.

Drei Aufzeichnungsmodi

Für die Aufzeichnung der von den Cortex-M3/M4 Prozessoren generierten Traceinformationen unterstützt Lauterbach drei Modi:

- **FIFO-Modus:** Abspeichern der Information im 128 MByte Speicher der TRACE32 CombiProbe.
- **STREAM-Modus:** Streamen der Information auf die Festplatte des Hostrechners.
- **Real-time Profiling:** Die Traceinformation wird auf den Hostrechner gestreamt und dort live ausgewertet.

Bei den ersten beiden Aufzeichnungsmodi wird die Traceinformation zunächst erfasst. Die Traceauswertung kann man erst nach Abschluss der Aufzeichnung durchführen.

Jeder Aufzeichnungsmodus hat seine spezielle Zielsetzung. FIFO ist sicher der gebräuchlichste Modus. Er ist schnell und genügt in der Regel für die Fehlersuche und für Laufzeitanalysen.

Die auf Cortex-M3/M4 Prozessoren implementierte ETMv3 verfügt weder über Trigger noch Tracefilter. Daher ist es nicht möglich, die Tracequelle so zu konfigurieren, dass diese nur Informationen für den Programmabschnitt generiert, der von Interesse ist. Dies kann dazu führen, dass für die Fehlersuche Tracedaten eines relativ langen Zeitraums aufgezeichnet und ausgewertet werden müssen. Hier ist der STREAM-Modus die richtige Wahl. Der STREAM-Modus stellt allerdings höhere Anforderungen an die Debug-Umgebung:

- Die große Datenmenge, die durch das Streaming entsteht, erfordert ein 64-Bit TRACE32 Executable. Nur so lassen sich die Traceeinträge adressieren.
- Die Übertragungsrate zwischen CombiProbe und Hostrechner muss ausreichen, um alle Tracedaten verlustfrei zu streamen. Der 128 MByte Speicher der CombiProbe puffert dabei Lastspitzen am Traceport (TPIU) ab.

Real-time Profiling ist besonders geeignet, um *Statement* und *Condition Coverage* durchzuführen, da man die Analyse live auf dem Bildschirm mitverfolgen kann. Für bedingte Anweisungen, für die bisher nur der Fail-Zweig durchlaufen wurde (gelb unterlegt in Bild 17), kann man das Embedded System gezielt so stimulieren, dass auch der Pass-Zweig zur Ausführung kommt.

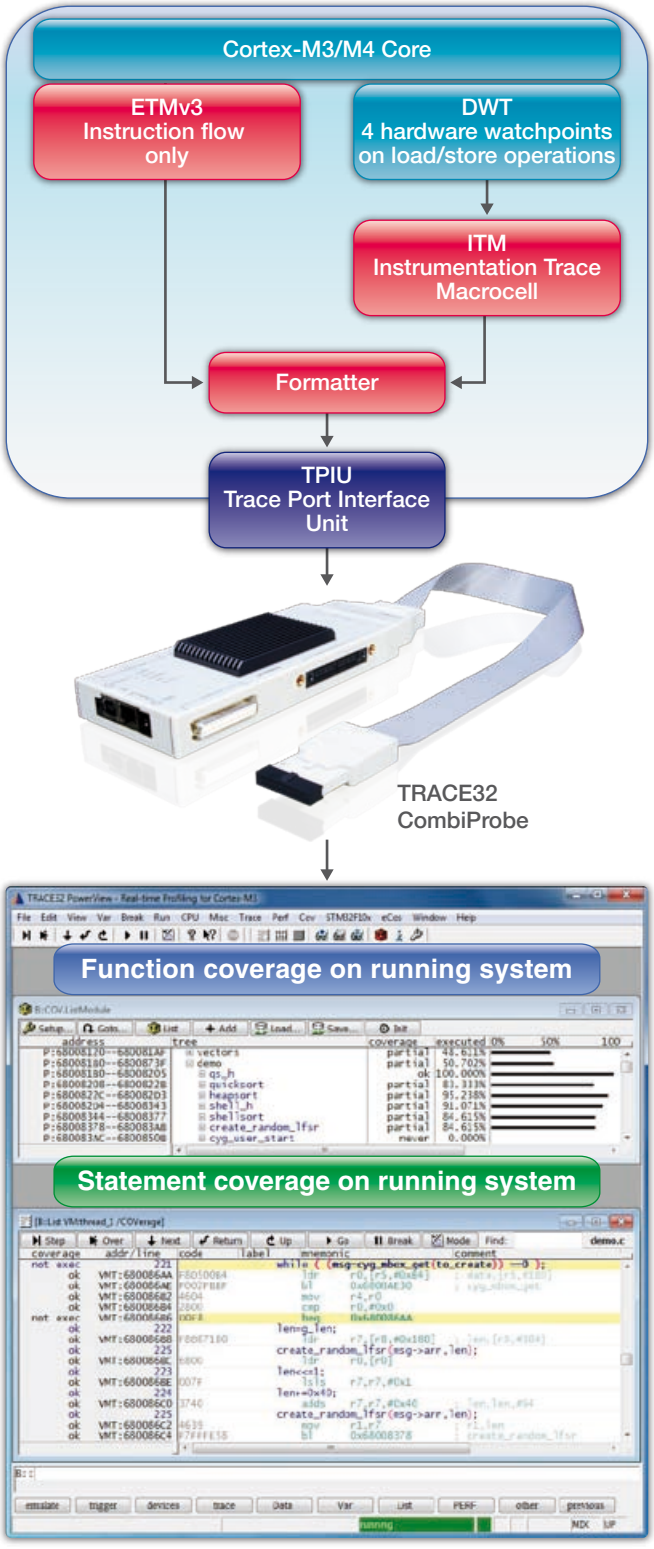


Bild 17: Real-time Profiling erlaubt es, die Code-Coverage-Analyse live auf dem Bildschirm mitzuverfolgen.

Simulation und Realität rücken näher zusammen

Es ist heute üblich, die Simulation und Verifikation möglicher Lösungen vorab durchzuführen. Gerade in der Regelungstechnik haben aus diesem Grund MATLAB® und Simulink® als Entwicklungssoftware Einzug gehalten. Wer seine Regelstrecke im Simulator testet und so mögliche Auswirkungen veränderter Parameter schnell und einfach voraussagen kann, spart viel Zeit und Aufwand.

Aber wie geht es weiter, wenn man seinen optimalen Regelalgorithmus in der Simulation gefunden hat? Wie bringt man diese Lösung auf die echte Steuerungshardware?

Simulink bietet dafür die Möglichkeit der automatischen Codegenerierung. Aber kann man sicher sein, dass das Programm auf der Steuerungshardware genauso reagiert wie die Simulation? Vertrauen ist gut, Kontrolle ist besser! Und oft ist ein Nachweis zwingend vorgeschrieben.

Verifikationsansatz der TU München

Der Lehrstuhl für Flugsystemdynamik der TU München hat in seiner Entwicklung eines Flight-Control-Systems für eine Diamond DA42 (siehe Bild 20) einen effizienten Lösungsweg aufgezeigt.

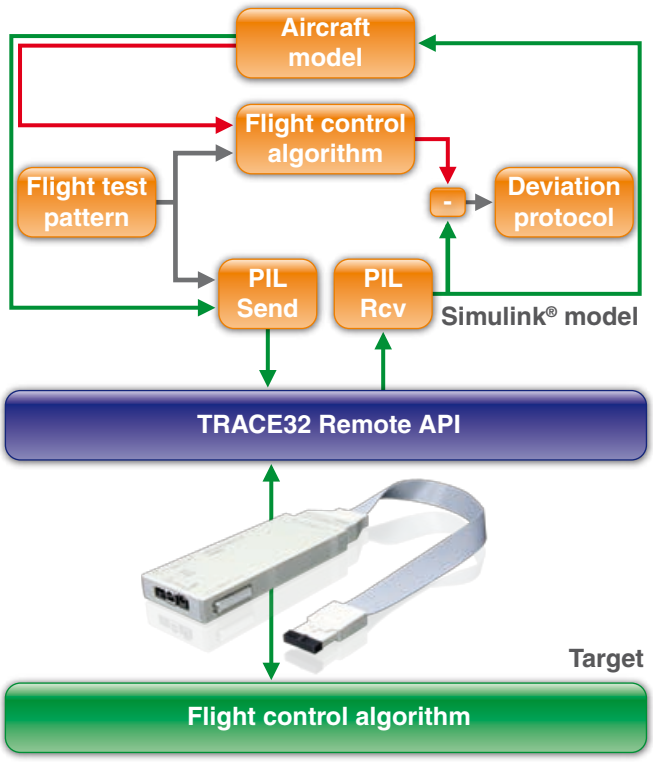


Bild 18: Das reale Reglerverhalten (grüner Pfad) und das simulierte Reglerverhalten (roter Pfad) werden abgeglichen.

Nachdem die Regelungsalgorithmen mit Simulink erstellt und funktional getestet waren, wurde mittels des *Embedded Coders* aus den Regelungsblöcken der entsprechende Programmcode für den Prozessor der Steuerungshardware erzeugt. Unter Verwendung eines TRACE32 Debuggers hat man den generierten Code auf die Steuerungshardware geladen und dort ebenfalls funktional getestet.

Um die Abweichung zwischen simuliertem Reglerverhalten (roter Pfad) und realem Reglerverhalten (grüner Pfad) zu ermitteln, vor allem aber um die numerische Genauigkeit der Steuerungshardware zu bestätigen, wählte man eine Processor-In-the-Loop-Simulation (PIL) (siehe Bild 18). Die PIL-Simulation basiert im Wesentlichen auf den eigens entwickelten Simulink Blöcken „PIL Send“ und „PIL Receive“. Diese wurden entworfen, um die Kommunikation zwischen Simulink und der *TRACE32 Remote API* zu realisieren.

In jedem Durchlauf führt der Flight-Control-Algorithmus auf der Zielhardware einen einzelnen Berechnungsschritt der zeitdiskret arbeitenden Flugregelung durch. Das Simulink Modell stellt die dazu benötigten Eingangsparameter bereit. Die berechneten Stellgrößen werden an das Simulink Modell zurückgeliefert und speisen dort das *Aircraft Model*. In einem parallelen Berechnungszweig ermittelt der simulierte Flight-Control-Algorithmus dieselben Stellgrößen. Durch Differenzbildung werden beide Ergebnisse abschließend verglichen.

Die Erprobung im Teststand ergab eine absolute Abweichung von 10^{-13}. Eine hohe Übereinstimmung, die durch diesen Lösungsansatz elegant und einfach nachgewiesen wurde.

Unter **www.lauterbach.com/intsimulink.html** finden sich weitergehende Informationen zum Projekt des Lehrstuhls für Flugsystemdynamik der TU München.

TRACE32 Integration für Simulink®

Auf der *embedded world 2012* in Nürnberg wird Lauterbach eine noch engere Kopplung zwischen Simulink und TRACE32 vorstellen.

Für diese Kopplung nutzt Lauterbach die folgende Eigenschaft der Codegenerierung: Der für einen Simulink Block generierte Code beginnt immer mit einer Kommentarzeile, die den Namen und den Modellpfad des Blocks enthält. Die Kommentarzeilen stehen nach dem Laden des generierten Codes im TRACE32 Debugger zur Verfügung. Sie erlauben eine einfache Zuordnung von Simulink Blöcken zu Sourcecodezeilen.

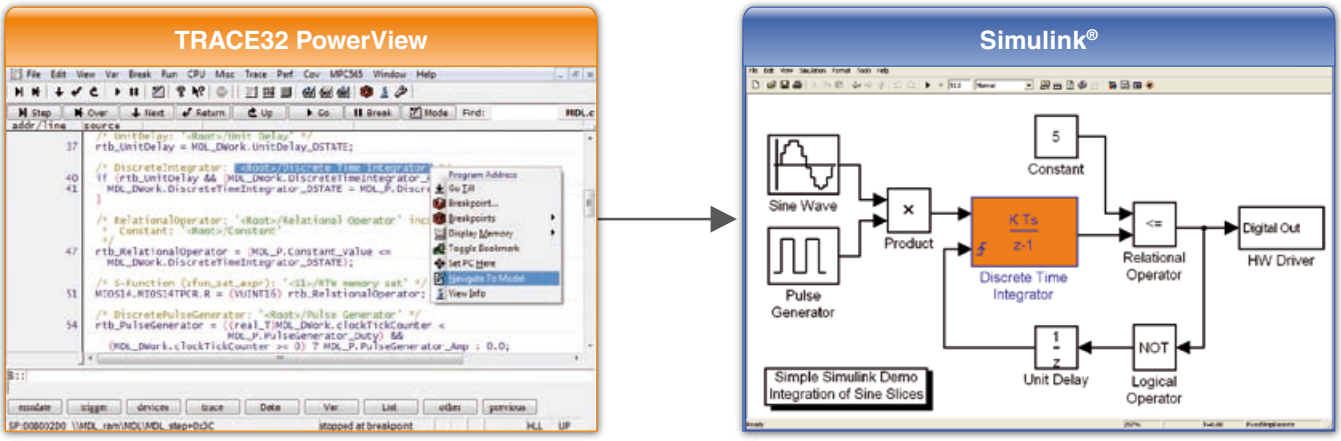


Bild 19: Der zur ausgewählten Sourcecodezeile gehörende Block wird in Simulink markiert.

Navigation von Simulink® zu TRACE32

Als so genannte *Simulink Customization Menus* werden ein globales TRACE32 Menü, sowie TRACE32 Menüs für Blöcke und Signale in Simulink integriert. Über diese Menüs lässt sich der TRACE32 Debugger aus Simulink heraus steuern. Folgende Funktionen sind möglich:

- Blockcode in TRACE32 darstellen
- TRACE32 *Variable Watch Window* für Signale öffnen
- Simulink Build in den TRACE32 Debugger laden
- Block-/Programm-Breakpoints in Simulink setzen und verwalten
- Signal-/Daten-Breakpoints setzen und verwalten
- Programmausführung auf der Steuerungshardware starten und stoppen

Navigation von TRACE32 zu Simulink®

Aus dem TRACE32 Debugger kann für eine ausgewählte Stelle des Quellcodes der dazugehörige Block in Simulink markiert werden (siehe Bild 19).

Ausblick

Sobald die *Simulink Release* 2012a verfügbar ist, wird Lauterbach die verbesserte Funktionalität der Simulink *rtiostream API* dazu nutzen, eine PIL-Simulation, Data-Logging sowie Parameter-Tuning zu integrieren.

MATLAB® und Simulink® sind eingetragene Warenzeichen von The MathWorks, Inc.

Bild 20: Erprobungsflugzeug Diamond DA42 (Quelle: www.diamond-air.at).

UEFI BIOS debuggen mit TRACE32

Eine neue TRACE32 Extension für die Atom™ Debugger erlaubt ein umfassendes Debuggen von Insyde's H2O UEFI BIOS.

UEFI ist der Nachfolger des traditionellen PC BIOS. Als Schnittstelle zwischen Firmware und Betriebssystem verwaltet es den Bootprozess. Von Power-On bis zur Übernahme durch das Betriebssystem durchläuft UEFI unterschiedliche, klar abgegrenzte Phasen (siehe Bild 21).

Als JTAG-basiertes Tool erlaubt TRACE32 das Debuggen ab dem *Reset Vector*. In jeder Phase des Bootprozesses unterstützt die Bedienoberfläche PowerView den Entwickler mit speziellen Windows zur Darstellung von UEFI-spezifischen Informationen. Vorgefertigte Skripte und Funktionen erlauben das Debuggen dynamisch nachgeladener Treiber ab der ersten Instruktion. Details zur neuen UEFI Extension finden Sie unter **www.lauterbach.com/uefi.html**.

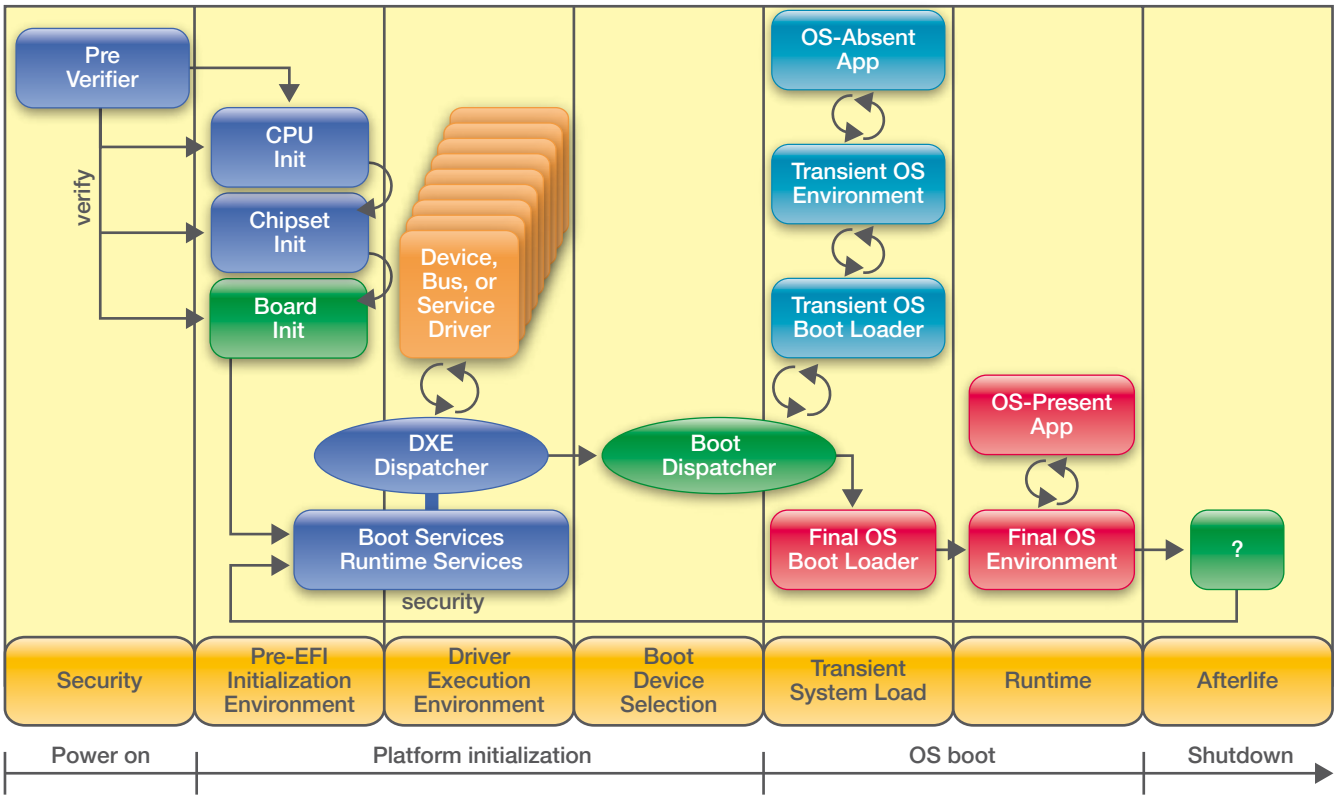


Bild 21: Ablauf eines System-Boots mit UEFI.

WELTWEITE NIEDERLASSUNGEN

- **Deutschland**
- Frankreich
- Großbritannien
- Italien
- USA
- China
- Japan

In allen anderen Ländern vertreten durch kompetente Partner

BENACHRICHTIGEN SIE UNS

Falls sich Ihre Adresse geändert hat oder Sie kein Mailing mehr von uns erhalten möchten, schicken Sie bitte eine E-Mail an **mailing@lauterbach.com**