Debugger mit Rückspiegel - *Trace-Techniken im Überblick*

Während der Entwicklung wird eine CPU heute vorwiegend mit so genannten On-Chip-Debuggern überwacht, die über JTAG oder ähnliche Schnittstelle mit der CPU verbunden werden. Die zentralen Aufgaben eines On-Chip- Debuggers bestehen darin, das Programm zu starten und an einem mehr oder weniger komplexen Breakpoint anzuhalten. Bei angehaltenem Programm kann der Entwickler dann den aktuellen Systemzustand auslesen. Einfache Fehler lassen sich mit dieser Debug-Technik gut auffinden.

Die Grenzen dieser Technik zeigen sich aber sehr schnell, wenn es darum geht, komplexe Fehler zu Lokalisieren, die nur unter bestimmten Laufzeitbedingungen auftreten. Gleiches gilt für Fehler deren Auswirkungen entweder zeitlich oder in der Programmhierarchie so weit auseinander liegen, dass der Zusammenhang zwischen Ursache und Wirkung nicht mehr erkennbar ist. Darüber hinaus ist eine Analyse des Zeitverhaltens nur bedingt möglich.
Die Steuerung der CPU alleine ist also für die effektive Fehlersuche nicht ausreichend.

Hier bringen Trace-Tools Licht ins Dunkel. Zur Programmlaufzeit zeichnen diese alle oder nur bestimmte Programmzyklen auf und können daraus je nach verwendeter Trace-Methode den groben Programmablauf oder den kompletten Programmfluss bis auf Assembler bzw. Hochsprachenebene rekonstruieren. Der Entwickler kann dann mit Hilfe dieser Aufzeichnung analysieren, ab welchem Zeitpunkt das Programm anders abgelaufen ist als gewünscht und kommt damit der Fehlerursache schon sehr nahe.

Das Ziel ist also klar: Ein Trace-Tool muss irgendeine brauchbare Aussage über die Programmvergangenheit liefern. Es stellt sich nur noch die Frage: Welche Methoden gibt es und wo sind deren Stärken und Schwächen?

Bus-Trace: Wenn alles offen liegt

Die Trace-Methoden unterscheiden sich grundsätzlich danach, ob sie zusätzliche Trace-Hardware erfordern oder mit dem Debugger alleine auskommen und keine Zusatz-Hardware benötigen. Trace-Hardware ist immer dann erforderlich, wenn bestimmte Signalgruppen der CPU aufgezeichnet werden sollen. Bei Mikroprozessoren ohne interne Peripherie und Speicher ist es üblich den externen Adress/Datenbus und einige Steuersignale aufzuzeichnen. Damit können der Programmfluss und alle Variablen dargestellt und deren Zustand zu einer Triggerbedingung verknüpft werden. Man spricht von der Bus-Trace-Methode (Bild 1).

Erste Voraussetzung für diese Methode ist der physikalische Zugang zum Speicherinterface der Applikation. Typischerweise kommen hier Adapter zum Einsatz, die entweder direkt an der CPU oder direkt am Speicherbaustein die Signale abgreifen. Unter Verwendung von PressOn

Adaptern für Leiterbahnpads oder Steckern wie z.B. MICTOR können die Signale auch auf der Strecke zwischen CPU und Speicher abgegriffen werden.

Ist eine physikalische Verbindung in dieser Art und Weise möglich, so können alle Speicherzugriffe aufgezeichnet werden und daraus der Programmfluss bzw. alle Datenzugriffe ermittelt werden.

Einschränkungen gibt es nur dann, wenn die CPU mit einem Cache oder internem RAM ausgestattet ist. Hier kann es vorkommen, dass auf dem externen Businterface lange Zeit kein Buszyklus erscheint weil alle benötigten Daten bereits im internen Speicher vorhanden sind und dort ausgeführt werden. In diesem Fall kann das Trace-Tool nichts aufzeichnen, das Programmverhalten bleibt verborgen. Um dennoch eine brauchbare Aufzeichnung zu erhalten können folgende Wege beschritten werden

Cache ausschalten: Diese Lösung ist sehr einfach und besteht meist nur aus einer Bit Manipulation im Cache Control Register. Der Performance Verlust kann dabei aber erheblich sein.

Cache auf WriteThrough schalten: Mit dieser Lösung sind alle Schreibzugriffe auf Variablen am externen Bus-Interface sichtbar. Auch hier ist meist eine Bit Manipulation im Cache Control Register ausreichend. Ist der Cache bereits auf WriteThrough Mode eingestellt so entsteht kein Performance Verlust. Beim Wechsel von CopyBack Mode zu WriteThrough Mode ist der Performance Verlust akzeptabel.

Code Instrumentierung: Hier wird die Applikation vom Entwickler oder vom Debug-Tool so verändert, dass sie auf Non-Cachable Adressbereiche schreibt. In der Applikation selbst ist festgelegt welche Register- oder Variableninhalte am externen Speicher-Interface sichtbar sind und somit aufgezeichnet werden.

Für die Bus Trace-Methode kommen meist Logic-Analysatoren zum Einsatz die eine große Anzahl von CPU Signalen aufzeichnen müssen. Bei einem 64bit Datenbus und entsprechend großem Adressraum können das schnell über 100 Signale werden. Wird das Businterface mit Double-Data-Rate betrieben, dann wird bei den meisten am Markt erhältlichen Logic-Analysatoren die doppelte Kanalzahl, also über 200 Kanäle benötigt, damit die Daten der fallenden und steigenden Clock-Flanke aufgezeichnet werden können. Es ist der hohen Integrationsdichte heutiger FPGAs zu verdanken, dass solche Logic-Analysatoren mit hoher Kanalzahl, großem Speicher und hoher Auflösung nicht größer als die Debugger Hardware selbst sind. Bei entsprechend enger Ankopplung an die Debugger Software, entsteht aus diesem Gespann eine handliche und komfortable Entwicklungsumgebung, wie man sie von Emulatoren kennt.

Altlaufstraße 40 • D-85635 Höhenkirchen-Siegertsbrunn • Tel. +49 8102 9876-0 • Fax +49 8102 9876-999 • sales@lauterbach.com

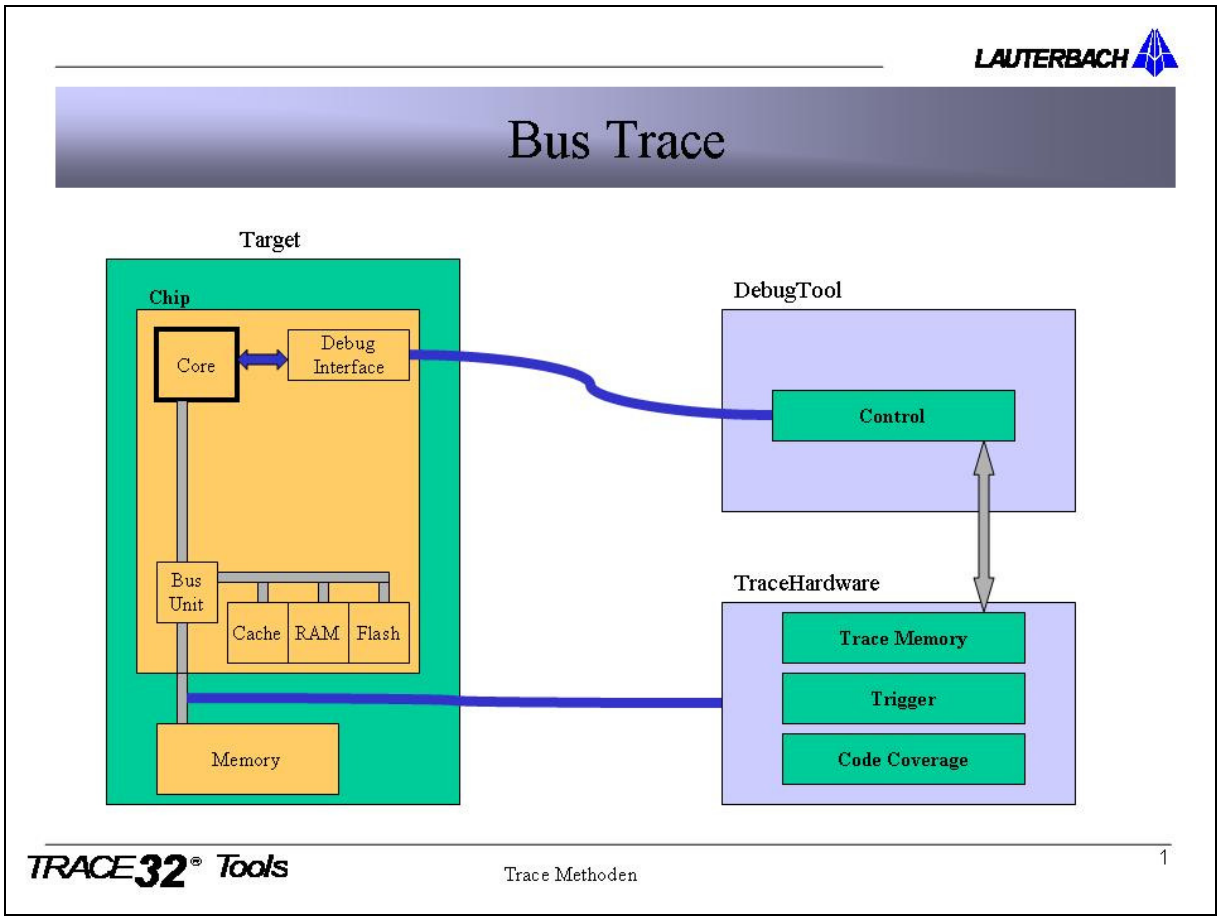

Bild 1. Der Bus-Trace wird verwendet, wenn ein physikalischer Zugang zum Speicher-Interface besteht. Wenn allerdings Programminhalte bereits in den Cache geladen wurden, tut sich am äußeren Bus-Interface nichts mehr und die internen Vorgänge bleiben verborgen.

Flow-Trace: Wenn der Chip mitmacht

Immer höhere Integrationsdichten und Preisdruck haben dazu geführt, dass bei vielen Prozessoren der CPU-Kern, Cache, Peripherie, FLASH- und RAM-Speicher in nur einem Gehäuse integriert sind (System-On-Chip). Oftmals verfügen dieser Prozessoren nicht einmal mehr über ein externes Speicher-Interface. Die Bus Trace-Methode kann deshalb nicht angewendet werden. Neuere CPU Architekturen haben deshalb neben der Debug-Schnittstelle eine spezielle Trace-Schnittstelle auf dem Serienchip. Über diese wird in komprimierter Form der Programmfluss nach außen sichtbar gemacht (Bild2).Man spricht von der Flow Trace Methode.
Als Trace-Schnittstelle kommt meist ein 4, 8 bzw. 16bit breiter Trace-Bus zum Einsatz, über den Programmflussdaten und/oder Datenzugriffe mit bis zu 400MHz Busfrequenz in komprimierter Form übertragen werden. Dabei werden die Informationen des Adressbus/Datenbus so übertragen wie sie direkt am CPU-Kern auftreten. Das bedeutet, dass auch Zugriffe auf Chip internen FLASH oder RAM-Speicher - insbesondere auf den Cache- aufgezeichnet werden können. Wenn es das Trace-Protokoll erlaubt, kann dieser komprimierte Datenstrom vom

Trace-Tool in Echtzeit zu kompletten Adressen rekonstruiert werden und damit für die Echtzeit Triggerung bzw. Echtzeit Code-Coverage Analyse genutzt werden. Die Funktionalität ist dann einem Emulator durchaus ebenbürtig.

Ohne Echtzeit Adress-Rekonstruktion ist es nur bedingt möglich die Trace-Daten zu filtern, also selektiv nur bestimmte Programmteile oder Variablen Zugriffe aufzuzeichnen. Nur wenige Trace-Schnittstellen unterstützen gezieltes Ein/Ausschalten oder die Definition von Addressfenstern innerhalb derer Trace-Daten generiert werden. Da hilft dann nur ein Trace-Tool mit großem Trace-Speicher in dem alle Trace-Daten ungefiltert aufgezeichnet werden. Bei Aufzeichnungszeiten von mehreren Sekunden, ist es sehr wahrscheinlich, dass der gesuchte Fehler in der Aufzeichnung zu finden ist. Außerdem stehen mehrere Programmdurchläufe für eine Laufzeitstatistik und/oder Code-Coverage Analyse zur Verfügung.

Für die Flow-Trace-Methode kommen CPU spezifische Trace-Adapter und Auswerte-Algorithmen zum Einsatz. Die mechanischen und elektrischen Details der Trace-Schnittstelle sind meist vom CPU Hersteller vordefiniert und werden so von den Trace-Tool Herstellern unterstützt. Der Anwender erhält eine Plug-and-Play Lösung die so gut wie keine bzw. nur einfache Setups erfordert. Wird dieses Interface von der CPU angeboten, dann stellt es mit Abstand die komfortabelste Trace-Möglichkeit dar.

Einziger Wermutstopfen dieser Trace-Technik ist das Bandbreitenproblem. Wenn also chipintern mehr Trace-Daten anfallen als über das Trace-Interface übertragen werden können. Die Chip Hersteller begegnen diesem Problem mit FIFO Buffern und Datenreduktion. Das funktioniert gut wenn nur der Programmfluss aufgezeichnet wird. Geht es aber darum Daten Zugriffe im Trace aufzuzeichnen, dann ist das Trace-Interface schnell überlastet.
Um das zu verhindern ist es meist möglich, die Programmausführung zu bremsen, was aber zu Performanceverlust führt.

Bisher wurden zwei Trace-Methoden beschrieben, die nur deshalb funktionieren, weil die CPU über Signalleitungen den Programmfluss nach außen sichtbar macht. Aber welche Trace-Methoden gibt es z.B. für Single-Chip Systeme, die weder über externen Adress/Datenbus noch über eine spezielle Trace-Schnittstelle verfügen? Auch für diesen Fall gibt es Lösungen, die durch die Nutzung CPU interner Trace-Optionen bzw. durch geschickten Einsatz des Debuggers, zu brauchbaren Trace-Ergebnissen führen.

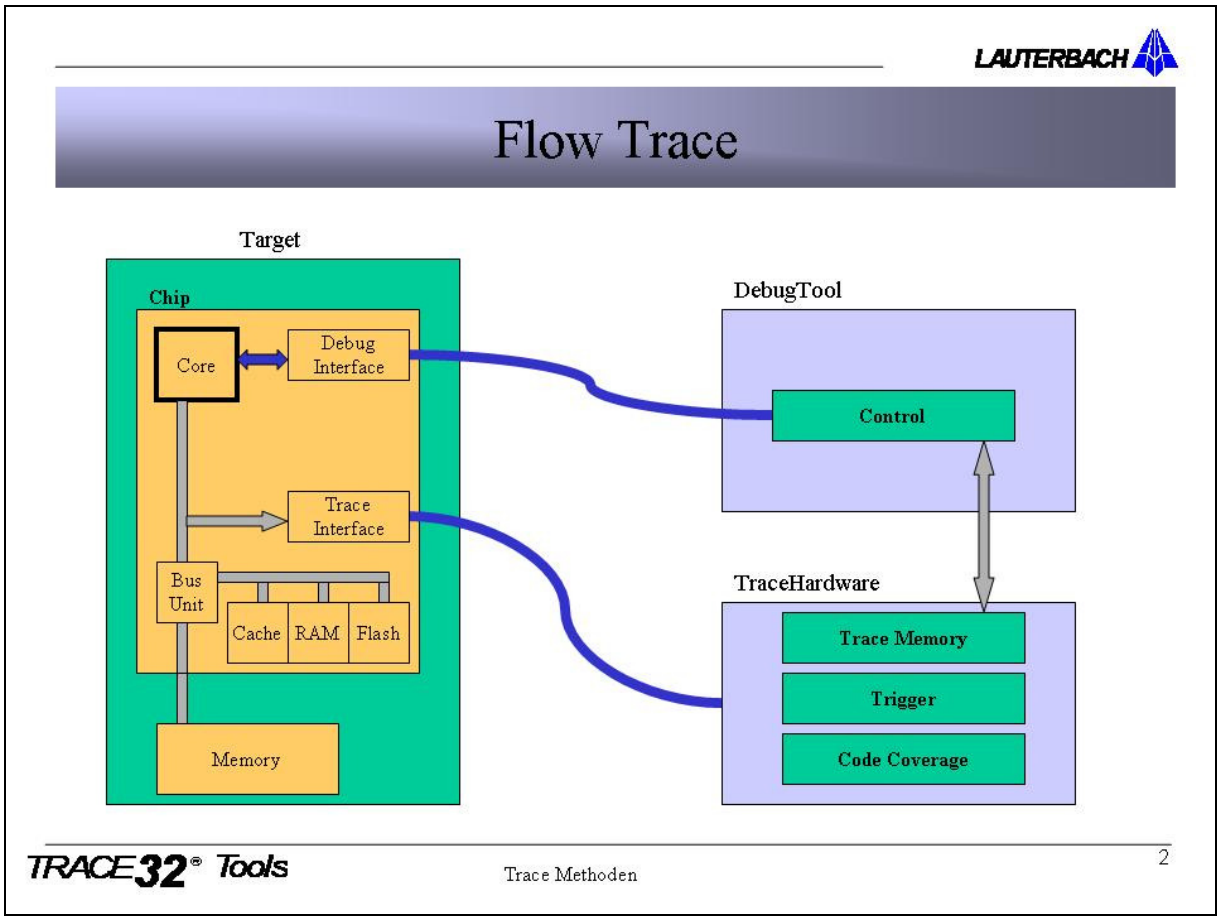

Bild 2. Neuere System-on-Chip-Systeme mit integriertem Speicher haben neben dem Debug-Interface auch eine Trace-Schnittstelle. Dann kann die Flow-Trace-Methode angewendet werden, bei der der Programmfluss aus dem Trace-Protokoll rekonstruiert wird.

On-Chip-Trace: mehrere Varianten

Manche CPUs enthalten einen Trace Buffer in den die Adressen von ausgeführten Sprungbefehlen und deren Sprungziel aufgezeichnet werden. Nachdem das Programm angehalten hat wird der Trace-Buffer via Debugger ausgelesen und der Programmfluss rekonstruiert. Die Trace-Aufzeichnung erfordert keine Änderungen in der Applikation, läuft in Echtzeit und ist sehr einfach zu bedienen. On-Chip Trace Buffer werden meist klein gehalten um den Preis des Chips bzw. die benötigte Siliziumfläche klein zu halten (Bild 3). Bei vielen CPUs können deshalb nur die letzten vier bis acht Sprünge aufgezeichnet werden. Nur sehr wenige CPUs bieten die Option für Aufzeichnungen von mehr als 1000 Sprüngen.

Extended On-Chip Trace

Um den kleinen Trace-Buffer der On-Chip-Methode zu erweitern, bieten manche CPUs die Möglichkeit bei vollem On-Chip Trace Buffer einen Interrupt auszulösen. Die entsprechende Interrupt Service Routine kopiert den On-Chip Trace Buffer in den Arbeitsspeicher des Zielsystems und springt anschließend zurück zur Applikation. Nach und nach füllt sich so der Trace-Speicher des Zielsystems. Abhängig von der verwendeten CPU steigt die Programmlaufzeit erheblich an.

Häufig kann die Interrupt Service Routine bereits in der Debugger Software integriert werden. Der Anwender muss dem Debugger nur noch den freien Trace-Speicher im Target RAM bekannt geben, der Rest läuft dann automatisch. Als Variante dieser Trace-Methode gibt es CPUs die per DMA Zugriff die Trace-Daten direkt in das Target RAM schreiben. Die Echtzeitverletzung ist dabei minimal.

Software-Trace-Methode (Trace-Puffer im Zielsystem)

Bei dieser Methode wird der Trace-Speicher durch die Applikation selbst gefüllt. Die Handhabung ähnelt der bei vielen Entwicklern bekannten ‚printf' Methode. Der Unterschied besteht darin, dass die Meldungen nicht auf einem Terminal ausgegeben, sondern in eine Trace-Struktur im Target-RAM eingetragen werden. In die Applikation wird vom Entwickler eine Trace-Funktion integriert die einen Trace-Speicher im Target RAM verwaltet. Der Entwickler selbst entscheiden welche Informationen (Variablen- oder Registerinhalt, Tasknummer ...) in den Trace-Speicher geschrieben werden. Nachdem die Applikation angehalten hat, wird der Trace-Speicher via Debugger ausgelesen und steht dann allen Analysetools zur Verfügung. Wird am Anfang und am Ende von ausgewählten Funktionen ein Trace-Aufruf eingefügt, so kann damit genau ermittelt werden in welcher Reihenfolge die Funktionen abgearbeitet wurde. Wird innerhalb des Trace-Aufrufes auch noch ein Timerwert in die Trace-Struktur geschrieben, dann sind sogar Laufzeitmessungen und Statistiken von Funktionen möglich.

Software-Trace-Methode (Tracebuffer im Entwicklungssystem)

Auch bei dieser Methode wird der Trace-Speicher durch die Applikation selbst gefüllt. Der Unterschied besteht darin, dass sich der Tracebuffer nicht im Target befindet, sonder auf dem Host. Dazu werden die Trace-Daten bereits zur Programmlaufzeit über spezielle Schnittstellen und Protokolle von der Applikation zum Host übertragen. Die Applikation muss also erweitert werden, damit das entsprechende Protokoll und deren physikalische Schnittstelle benutzt werden können.
Als physikalische Schnittstelle könnte z.B. ein serieller Kanal der Applikation verwendet werden. Da aber der Debugger bereits am Target angeschlossen ist, ist es naheliegend diese Schnittstelle für die Übertragung der Trace-Daten zu nutzen. Über einen gemeinsamen Kommunikationsbuffer im Targetspeicher werden dann die Trace-Daten von der Applikation zum Debug/Trace Tool übertragen.

Altlaufstraße 40 • D-85635 Höhenkirchen-Siegertsbrunn • Tel. +49 8102 9876-0 • Fax +49 8102 9876-999 • sales@lauterbach.com

Optimal ist diese Art der Übertragung wenn der Debugger auf den Target Speicher zugreifen kann, ohne die Programmausführung anzuhalten. Diese Option wird von einigen CPU Familien unterstützt. Alternativ muss das Programm periodisch angehalten werden, damit der Debugger auf den Kommunikationsbuffer zugreifen kann.
Als Variante zum Kommunikationsbuffer kann bei manchen DSP Bausteinen auch der Data-Communication-Channel (DCC) zur Datenübertragung dienen. Der DCC ist Teil der Debug Schnittstelle, deshalb werden keine Target Ressourcen benötigt und der Debugger kann darauf zugreifen ohne die Programmlaufzeit zu beeinflussen.
Der Aufwand kann bei dieser Trace-Methode natürlich noch weiter getrieben werden. So kann es für zeitkritische Applikationen sinnvoll sein, die Kommunikation noch durch eine Software Ringbuffer Struktur zu pipelinen (Fast-Data-Exchange FDX Protokoll). Dadurch wird vermieden, dass die Applikation unnötig lange warten muss bis einzelne Trace-Daten übertragen sind.

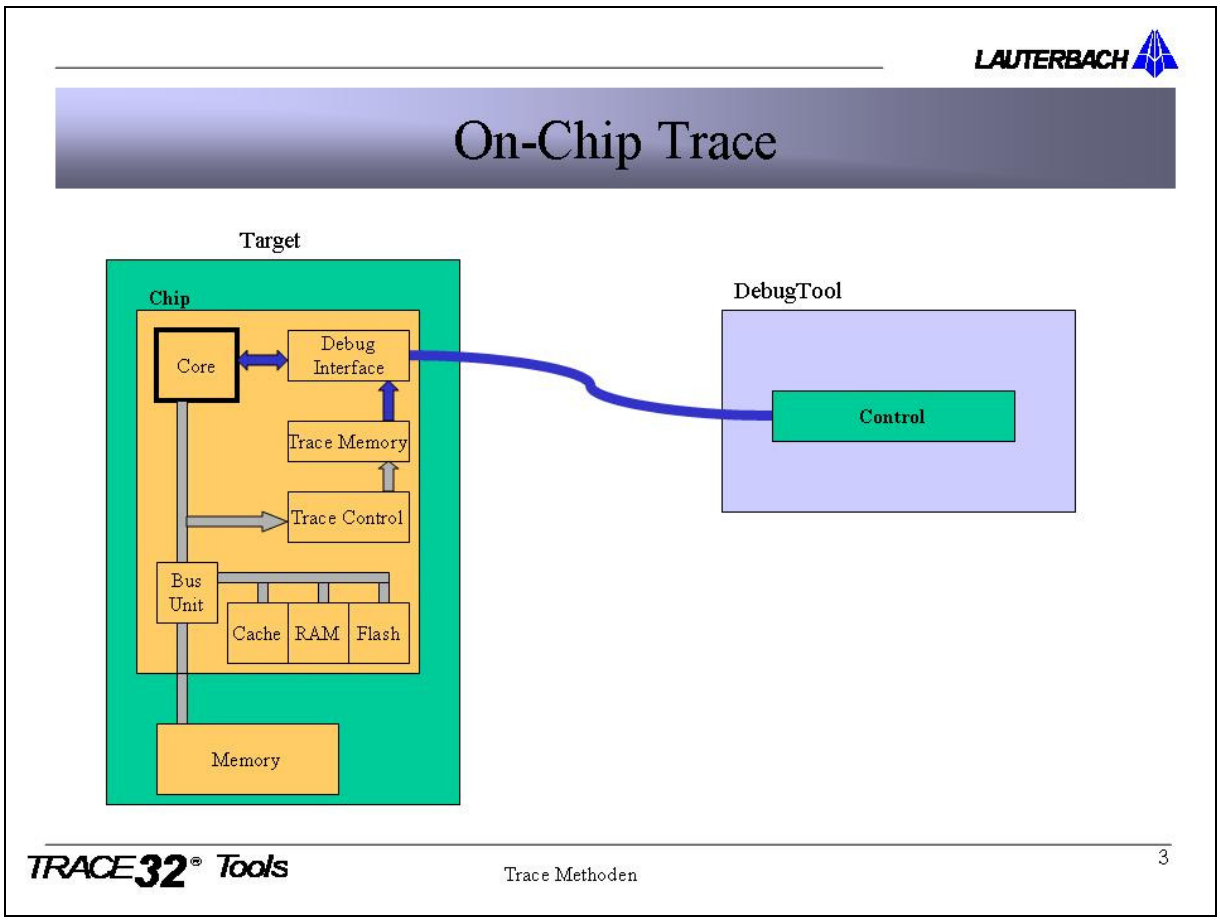

Bild 3. System-on-Chip-CPUs mit großem internen Speichern sind nahezu „undurchschaubar“. Einige CPUs haben deshalb die Trace-Infrastruktur gleich auf dem Chip integriert. Das ist für den Entwickler besonders einfach, erlaubt aber nur sehr geringe Trace-Tiefen.

Snooper Trace: verlustbehaftet
Diese Trace-Methode ‚schnuppert' periodisch in den Programmablauf und kann so die Veränderungen von Variablen bzw. den aktuellen Program-Counter aufzeichnen.
Wie bereits oben erwähnt erlauben manche Debug-Schnittstellen den Speicherzugriff auch während der Programmausführung ohne das Programm anhalten zu müssen. Der Einfluss auf die Programmlaufzeit ist dabei minimal. Variablen und Speicherinhalte können schnell gelesen werden und in einen virtuellen Trace auf dem Host-System abgelegt werden. Die Wiederholrate ist natürlich limitiert und ist auch davon abhängig wie viele Daten jeweils ausgelesen werden. Es kommt daher zwangsläufig zu Aufzeichnungslücken bzw. Datenverlusten im Trace.
Bei Aufzeichnungsintervallen von ca. 20 bis 30 us können mit dieser Trace-Methode sich ‚langsam' verändernde Variablen gut aufgezeichnet werden. Sie erfordert keine Programmänderung und ist sehr leicht zu bedienen.

Advanced Register Trace: Schritt für Schritt
Hier handelt es sich um einen simulierten Trace, so wie er auch in Debug-Simulatoren eingesetzt wird. Bei jedem Assembler Step, GO oder Break wird der aktuelle ProgramCounter, der ausgeführte Befehl und die Registerinhalte in einen virtuellen Trace auf dem Host-System gespeichert. Lässt man den Debugger oder Debug-Simulator mehrere hundert oder tausend Assembler Steps auszuführen so füllt sich der virtuelle Trace-Speicher nach und nach. Anschließend kann daraus der komplette Programfluss als Trace-Listing dargestellt werden. Diese Trace-Methode könnte man auch als Step-, GO-, Break-History bezeichnen. Läuft die Applikation auf einen Fehler, so kann man im Trace nachsehen an welcher Adresse man das Programm zuletzt gestartet hat, oder anders gesprochen, bis zu welcher Adresse die Applikation noch funktioniert hat.

Wie gezeigt steht dem Entwickler eine breite Palette an Trace-Methoden zur Verfügung. Welche davon zum Einsatz kommt, ist von der Prozessor Architektur, der Applikation und von dem untersuchten Problem abhängig Ist der Programmfluss aber erst einmal bekannt, so können darauf weitere Analysen- und Optimierungsmethoden angewandt werden, wie:

- Zeitmessung von Funktionen
- Statistik über Laufzeiten
- Verschachtelung von Funktionen bzw. Tasks
- Rekonstruktion von Variablen und Registerinhalten
- Cache Analyse und Optimierung
- Code-Coverage

Obwohl ein Trace-Tool, das zur Programmlaufzeit Informationen über die Reihenfolge der ausgeführten Instruktionen und ihre Resultate aufzeichnet, für das Auffinden komplexer Fehler bzw. zur Analyse des Laufzeitverhaltens geradezu ideal ist, ist nur etwa jeder zehnte Entwicklungsarbeitsplatz mit einem Trace ausgestattet. Eine Ursache dafür ist, dass viele Entwickler zu wenig über die vorhandenen Trace-Methoden und deren Nutzen Bescheid wissen. Trace-Tools erweitern die Effizienz bei der Softwareentwicklung erheblich, konsequent eingesetzt sparen sie viel Zeit und Energie bei der Programmentwicklung.

Diese Tabelle gibt einen Überblick über die verschiedenen Trace-Techniken. Auch Mischformen zwischen einzelnen Techniken sind möglich.

	Bus Trace	Flow Trace	On-Chip Trace	Extended On-Chip Trace	Software Trace Target	Software Trace Host	Snooper Trace	Advanced Register Trace
TraceHardware	ja	ja	nein	nein	nein	nein	nein	nein
TraceGröße	Tool RAM (groß)	Tool RAM (groß)	On-Chip RAM (klein)	Target RAM	Target RAM	Host RAM	Host RAM	Host RAM
	gering . . groß	minimal	minimal	gering	gering .. erheblich	erheblich	minimal	minimal
Applikationsänderung Erforderlich?	ja/nein	nein	nein	nein	ja	ja	nein	nein
Echtzeitverletzung?	nein .. erheblich	nein .. gering	nein	gering .. erheblich	gering	gering	gering	nein
CPU Beispiele	68K PPC	ARM PPC SHx V850	SHx Tricore PPC	SHx	Alle CPUs	Alle CPUs	Alle CPUs	Alle CPUs

Altlaufstraße 40 • D-85635 Höhenkirchen-Siegertsbrunn • Tel. +49 8102 9876-0 • Fax +49 8102 9876-999 • sales@lauterbach.com