Ein Kamel geht durchs Nadelöhr

Einblick in die Funktionsweise integrierter Trace-Schnittstellen

Trace Schnittstellen wie NEXUS oder ARM-ETM sind bei der Programmentwicklung von Embedded Anwendungen mit modernen CPU Architekturen kaum mehr weg zu denken. Ohne viel Aufwand öffnen sie den Blick auf das Echtzeit Programmgeschehen und in Verbindung mit dem entsprechenden Trace-Tool wird die Fehlersuche und Messung von Laufzeiten besonders effektiv. So sehr sich die Schnittstellen auch in ihren mechanischen, elektrischen und Protokoll-Eigenschaften unterscheiden, alle müssen zwei konträre Bedingungen erfüllen. Einerseits entsteht beim Programmlauf eine gewaltige Datenmenge die in Echtzeit über die Trace-Schnittstelle übertragen werden soll, andererseits steht für die Übertragung eine, bedingt durch die geringe Anzahl der verfügbaren CPU Pins, begrenzte Bandbreite zur Verfügung. Für bestimmte Szenarien gelingt es den CPU Herstellern sehr gut diesen Widerspruch aufzulösen. Probleme gibt es aber, wenn vom Anwender ungünstige Trace Einstellungen gewählt werden. Deshalb ist es von Vorteil, die Funktionsweise der Schnittstelle, deren Grenzen, Möglichkeiten und besonders deren Seiteneffekte zu kennen.

Hohe Prozessorfrequenzen und Preisdruck erfordern jedoch heutzutage den Einsatz von On-Chip Cache, On-Chip RAM oder On-Chip FLASH (System-On-Chip). Adress- und Datenbus werden aus Gründen der Geschwindigkeit, aber auch um Pins am Chip zu sparen, meistens nicht mehr nach außen geführt. Selbst für Mikrocontroller die noch ein externes SDRAM-Interface besitzen, macht es nur bedingt Sinn, an dieses ein Tracetool anzuschließen. Einerseits kann die Kapazitive Last von Trace-Steckern am Address-, Daten-Bus die Funktion der Applikation gefährden, andererseits ist es oftmals schwierig, aufgrund tiefer Prozessor Pipelines sowie Burst-Zyklen auf den Speicher, eindeutige Aussagen über die Aktivitäten im Prozessorkern zu machen. Wird etwa ein Programmteil im internen Cache ausgeführt, so sind am externen Businterface keinerlei Aktivitäten sichtbar. Wesentliche Teile des Programm- und Datenflusses bleiben dann für die Trace Aufzeichnung verborgen.

Address- und Daten-Bus des CPU Kerns sind also im Chip Gehäuse eingekapselt und deshalb muss ein neuer Zugang zu diesem eröffnet werden.

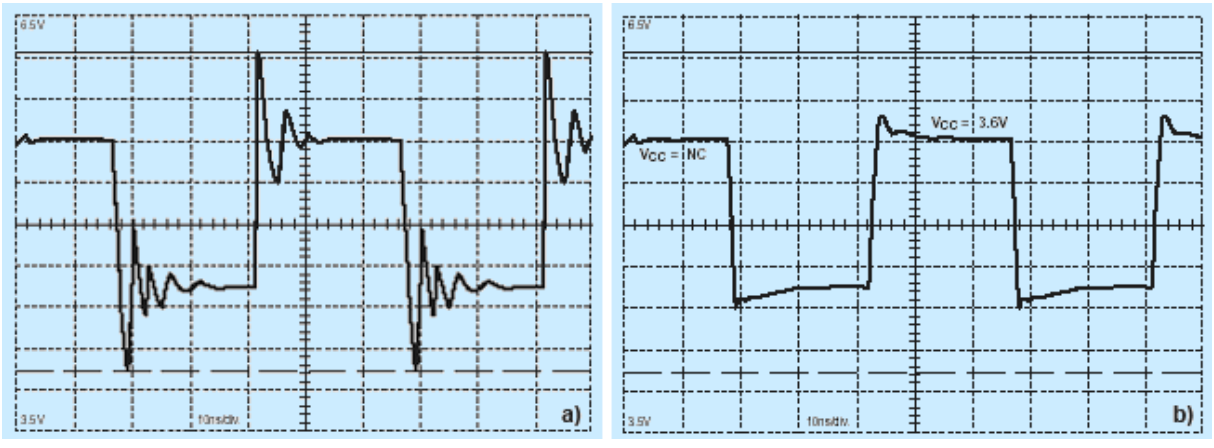

Bild 1: Signal an einer Trace-Schnittstelle – ohne (a) und mit Terminierung (b). Die Terminierung verlängert die Gültigkeit der Nutzerdaten.

Das Wissen um die Programmvergangenheit ist eine große Hilfe wenn es darum geht Fehlerursachen aufzuspüren oder sich Klarheit über das Laufzeitverhalten der Applikation zu verschaffen. Traditionell war und ist es üblich zur Programmlaufzeit die Aktivitäten auf dem Adress-, Daten- und Kontoll-Bus der CPU aufzuzeichnen. Bei dieser Methode sind alle Buszyklen und deren Reihenfolge bekannt, der Programmfluss kann deshalb eindeutig rekonstruieren werden.

Bondout-Chips mit vielen Nachteilen behaftet

Ein Lösungsansatz besteht darin, den kompletten internen Bus über zusätzliche Gehäusepins nach außen zu führen. Seit vielen Jahren kommen dafür so-genannte Bondout-Chips zum Einsatz. Die Praxis hat aber gezeigt, dass Bondout-Chips mit einigen Nachteilen behaftet sind.

Lösungsbedingt erhöht sich die Gehäusepinzahl. Ein größeres Gehäuse bedeutet aber höhere Kosten und deshalb kommen Bondout-Chips für die Serienproduktion nicht in Frage. Es werden also zwei Varianten des gleichen Chips benötigt, ein Serienchip und ein Entwicklungs-Chip (Bondout-Chip) und genau hier entstehen die Probleme. Für den CPU Hersteller bedeutet es einen erheblichen Aufwand zwei Varianten des gleichen Chips zu produzieren. Dabei kommt es häufig vor, dass Serien- und Bondout-Chip nicht auf dem gleichen Revisionsstand sind. Das bedeutet für den Entwickler, der Bondout-Chip kann noch mit Fehlern behaftet sein, die im Serienchip bereits behoben sind. Als Folge müssen unterschiedliche Softwareversionen gepflegt werden und so mancher Test mit dem Serienchip wiederholt werden. Und letztlich kann es auch noch zu Problemen mit der Verfügbarkeit von Bondout-Chips kommen. Das ist besonders dann von Bedeutung, wenn während der Entwicklung ein solcher zerstört wurde und schneller Ersatz benötigt wird.

All diese Erfahrungen haben zu der Forderung geführt, dass bereits auf dem Serienchip eine entsprechende Trace- und Debug-Schnittstelle integriert sein muss.

Feilschen um jeden Pin

Die mechanischen und elektrischen Details der Trace-Schnittstellen sind vom CPU Hersteller vordefiniert und werden so von den Trace-Tool Herstellern unterstützt. Die Anzahl der Schnittstellen Signale wird dabei möglichst gering gehalten, damit die Chipkosten nicht steigen.

Typischer Weise benötigt die Trace-Schnittstelle ein Clock-Signal, ein Synchnonisations-Signal oder Status-Bus und schließlich einen Daten-Bus mit 4, 8 bzw. 16bit Breite. Fügt man noch GND hinzu, so sind in der minimal Konfiguration nur sieben Verbindungen zwischen CPU und Trace-Tool ausreichend.

Meist sind diese Trace Signale an eigens dafür reservierten CPU Pins verfügbar. Es gibt aber auch den Fall, bei dem die CPU Pins alternativ für andere Funktionen verwendet werden können, zum Beispiel als Port. Beim Platinenentwurf sollte unbedingt vermieden werden, sich die Trace-Möglichkeit zu verbauen. Selbst wenn aktuell nicht geplant ist ein Trace-Tool einzusetzen, so kann es durchaus vorkommen, dass zu einem späteren Projekt Zeitpunkt diese Notwendigkeit entsteht. Die Pins sollten also Funktionen bedienen auf die zur Not verzichtet werden kann, und es sollten zumindest Lötpads auf der Platine vorgesehen werden damit nachträglich eine Kabelverbindung zum Trace-Tool hergestellt werden kann.

Gleiches gilt für CPUs die es erlauben die Trace-Busbreite frei zu konfigurieren. Selbst wenn im ersten Designschritt die 4bit Trace-Schnittstelle bereits eingeplant ist, so sollten trotzdem die zusätzlichen Trace-Signale abgreifbar sein, damit der Datendurchsatz bei Bedarf erhöht werden kann.

Wie einleitend erwähnt, benötigt man die Adress-, Daten- und Kontroll-Information vom internen Bus um daraus den Programmablauf zu rekonstruieren. Aber wie überträgt man die Trace-Informationen von einem 64bit Datenbus und 32bit Addressbus in Echtzeit über eine nur 4-bit breite Schnittstelle? Die CPU Hersteller müssen dazu an allen Parametern der Übertragung optimieren um dieses Problem zu lösen.

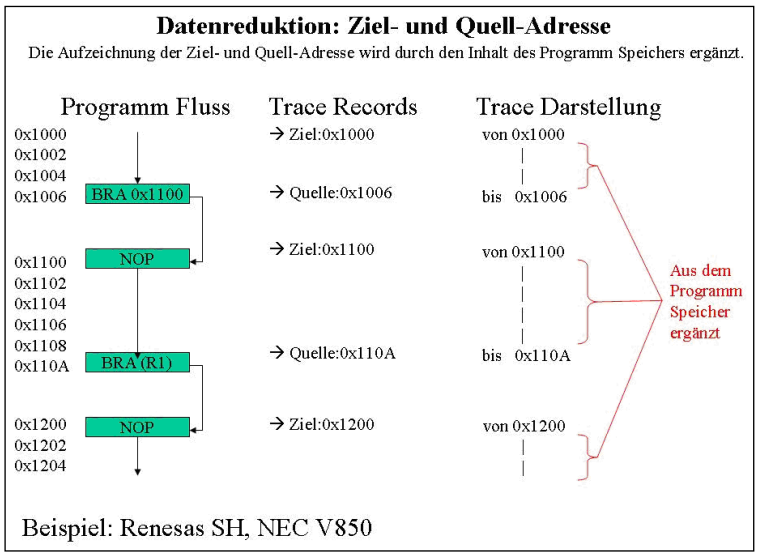

Bild 2: Beim Programmfluss ist es nicht nötig, sämtliche Adressen aufzuzeichnen, da Programme linear abgearbeitet werden. Nur die Sprünge müssen aufgezeichnet werden – der Rest kann aus dem Programmspeicher rekonstruiert werden.

Optimieren der Signale

Trace Schnittstellen werden in der Regel mit dem Systemclock der CPU betrieben. Frequenzen von über 300MHz bzw. Kanal-Bitraten von über 300Mbit/sec und DoubleDataRate Übertragung kommen zum Einsatz um einen möglichst hohen Datendurchsatz zu erreichen.

Beim Platinendesign muss dem natürlich Rechnung getragen werden. Oberstes Ziel ist es die Signalqualität und das Timing der Trace-Signale zu erhalten. So sollte der Trace-Stecker möglichst nahe an der CPU platziert werden. Um Laufzeitunterschiede zwischen den Signalen zu vermeiden, sollten die Leitungslängen zwischen CPU und Trace-Stecker identisch sein. Ebenso ist es ratsam die Clock-Leitung zu schirmen bzw. Bauteile für die Signalterminierung vorzusehen.

Bei hohen Frequenzen und entsprechenden Störeinflüssen wird aus so manchem Rechteck- ein verformtes Sinus-Signal. Deshalb wird auf Seiten der Tool Hersteller erheblicher Aufwand betrieben die Signalqualität und die Abtastung der Trace Signale zu optimieren. Dazu stehen drei Parameter zur Verfügung.

- Die Signal Terminierung
- Die Abtastschwelle und
- Der Abtastzeitpunkt

Bild zeigt die typischen Signalformen mit und ohne Terminierung. Die typischen Effekte von Signal Reflektionen im Besonderen Über-und Unter-Schwinger werden mit Hilfe der Terminierung so weit als möglich eliminiert. Durch variable Terminierungsspannungen kann für jede Applikation ein Optimum gefunden werden. Im Ergebnis verlängert sich die Gültigkeit der Nutzdaten (High bzw. Low-Pegel) . Man spricht in diesem Zusammenhang auch von Datenaugen, deren Größe ein Qualitätskriterium für die sichere Signal Abtastung ist.

Im nächsten Schritt wird die Abtastschwelle optimiert. Dabei wird diese so variiert, dass das Puls-Pausen-Verhältnis bei der Abtastung erhalten bleibt. Auch hier ist das Ziel möglichst breite Datenauge für die Abtastung zu erhalten. Und im letzten Schritt wird für jeden einzelnen Trace-Kanal der optimale Abtastzeitpunkt im Zentrum des Datenauges eingestellt.

In der Regel wird diese Optimierung automatisch beim Einschalten des Trace-Tools durchgeführt, kann aber optional auch vor jeder Aufzeichnung erfolgen. Gerade für kritische Applikationen, deren Signalqualität z.B. Temperaturabhängig ist, oder für Applikationen deren Systemtakt variiert wird, ist dies oft unerlässlich.

Trotz dieser aufwändigen Optimierungsmethoden kommt es dennoch vor, dass keine fehlerfreie Abtastung der Trace-Daten möglich ist. Deshalb bieten praktisch alle CPUs die Möglichkeit den Trace-Clock zu reduzieren. Damit wird eine fehlerfreie Aufzeichnung wieder möglich, der Performance Verlust ist aber erheblich und sollte wenn möglich vermieden werden.

Datenreduktion für den Programmfluss

Besonders effektiv wird die Trace-Schnittstelle entlastet, wenn die Datenmenge reduziert werden kann. Das betrifft sowohl die relevanten Trace-Informationen als auch das Protokoll mit dem diese übertragen werden. Die CPU Hersteller verfolgen dabei eine mehrstufige Strategie.

Im **ersten Schritt** der Datenreduktion geht man davon aus, dass sich der Programmcode im Speicher nicht verändert. Es ist also völlig ausreichend nur die Adressen aufzuzeichnen. Der zugehörige Programmcode stehen ja noch im Speicher (z.B. Flash) und kann bei Bedarf für die Trace Darstellung ausgelesen werden.

Für den **zweiten Schritt** der Datenreduktion nutzt man die Programmeigenschaft, dass immer eine Reihe von Befehlen linear aus dem Speicher abgearbeitet werden, bis dann ein Sprung-Befehl oder Interrupt den linearen Programmablauf unterbricht. Das bedeutet, es ist völlig ausreichend zu wissen von welcher Adresse (Quell-Adresse) gesprungen wurde und wohin gesprungen wurde (Ziel-Adresse). Der lineare Programmablauf zwischen den Sprüngen kann aus dem Speicherinhalt ermittelt werden.

Im Gegensatz zur Ziel-Adresse, die bei allen CPU Herstellern als ‚richtige' Adressinformation übertragen wird, wird die Quell-Adresse je nach verwendeter CPU auf unterschiedliche Art dem Trace-Tool mitgeteilt.

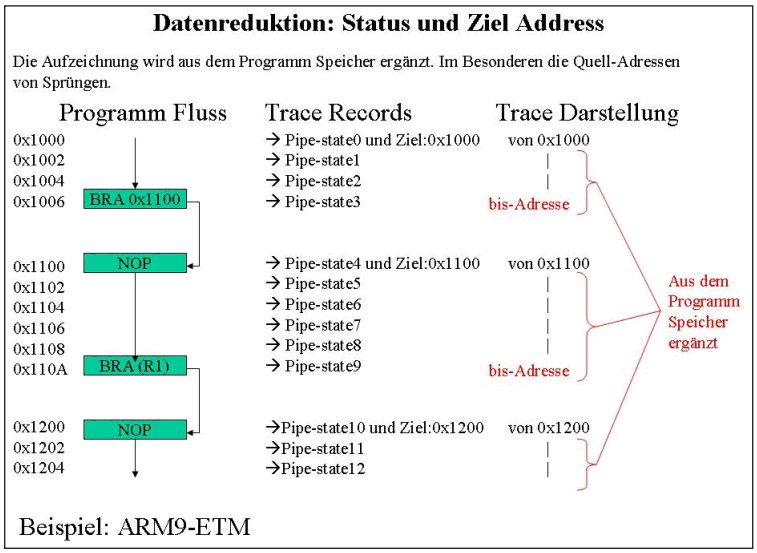

Bild 3: Eine andere Möglichkeit, die Trace-Daten zu reduzieren, besteht darin, nur die Sprungziel-Adressen und den Pipeline-Status zu übertragen. Die Quelladressen der Sprünge und der lineare Programmablauf werden rekonstruiert.

Bei der ersten Variante, wird die Quell-Adresse genauso wie die Ziel-Adresse als ‚richtige' Adressinformation übertragen. Nachteilig ist hier, dass immer zwei Adressen (Quelle, Ziel) übertragen werden was zur Überlastung der Trace-Schnittstelle führen kann. Als Vorteil ist zu nennen, dass nur Nutzdaten übertragen werden, der Speicher im Trace-Tool wird optimal ausgenutzt und ermöglicht deshalb lange Aufzeichnungszeiten.

Bei der zweiten Variante wird zusätzlich zur Ziel-Adresse bei jedem CPU Takt der CPU interne Pipelinestatus auf der Trace-Schnittstelle übertragen. Das gilt auch für die linearen Programmteile zwischen den Sprüngen. Bei der Rekonstruktion des Programmflusses wird dann der aufgezeichnete Pipelinestatus mit dem Speicherinhalt zusammengeführt. Damit kann genau ermittelt werden welcher Befehl (Quell-Adresse) zum Sprung geführt hat. Als Vorteil ist zu nennen, dass keine Quell-Adressen übertragen werden und deshalb eine Überlastung der Trace-Schnittstelle vermieden wird. Nachteilig ist hier, dass mit jedem Takt der Pipelinestatus aufgezeichnet werden muss. Der Speicher im Trace-Tool wird also mit der Taktgeschwindigkeit der Trace-Schnittstelle gefüllt auch wenn keine Sprünge im Programm auftreten. Die maximale Aufzeichnungszeit ist also durch die Taktgeschwindigkeit und die Tiefe des Trace-Speichers bestimmt.

Die Aufzeichnung des Pipelinestatus kann aber optimiert werden. So ist es möglich den Pipelinestatus mehrerer Befehle zu sammeln bzw. zu komprimieren und im Ergebnis die Anzahl der linear ausgeführten Befehle in einem Trace-Datenpaket zu übertragen.

Bis zu diesem Level der Datenreduktion wird jede Art von Sprung oder Interrupt auf der Trace-Schnittstelle übertrage. Das Trace-Tool kennt also zu jedem Zeitpunkt die Programm-Adressen (Bereiche) die aktuell vom Programm ausgeführt werden. Mit dem entsprechenden Trace-Tool ist es deshalb auch möglich in Echtzeit einen Adress-Trigger auszulösen oder eine Hardware basierte CodeCoverage Analyse durchzuführen.

Im **dritten Schritt** der Datenreduktion wird die Übertragungen von ‚direkten Sprüngen' eliminiert. 'Unbedingte direkte Sprünge' werden immer an der gleichen Adresse (Quell-Adresse) ausgeführt und springen immer zur gleichen Adresse (Ziel-Adresse). Beide Adressen lassen sich leicht aus dem Programmcode ermitteln und müssen deshalb nicht übertragen werden. Das gleiche Prinzip lässt sich auf ‚bedingte direkte Sprünge' erweitern, wenn der Pipelinestatus im Trace-Protokoll übertragen wird. Durch den Pipelinestatus kann genau ermittelt werden, ob ein ‚bedingter direkter Sprung' zur Ausführung gekommen ist oder nicht. Damit ist die Quell- und Ziel-Adresse bekannt.

Bis zu diesem Level der Datenreduktion lässt sich immer noch der komplette Programmfluss rekonstruieren. Echtzeit Trigger und CodeCoverage dagegen sind nicht mehr möglich.

Im **vierten Schritt** der Datenreduktion werden nur noch ausgewählte Sprung-Typen übertragen. Werden z.B. nur noch ‚Interrupt-Sprünge' aufgezeichnet so kann daraus noch eine Statistik über die Interrupt Aktivitäten in der Applikation ermittelt werden. Der Programmfluss kann aber nicht mehr ermittelt werden.

Ein generelles Problem gibt es noch mit kurzen Programmschleifen. Diese generieren innerhalb weniger CPU Takte sehr viele Sprünge und überlasten deshalb die Trace-Schnittstelle. Aber auch dafür gibt es Lösungen wie das NEXUS Trace Protokoll zeigt. Wird von diesem erkannt, dass mehrfach hintereinander die gleiche Sprunginformation übertragen wird, so wird ein Schleifenzähler aktiviert und anstelle der Sprungadresse nur noch die Anzahl der Schleifendurchläufe übertragen.

- keine Daten-Zugriffe
- alle Daten-Zugriffe
- Lese-Zugriffe
- Schreib-Zugriffe
- bestimmte Zugriffsbreiten
- Zugriffe auf definierte Adressfenster

Kombinationen der Zugriffstypen sind natürlich möglich.

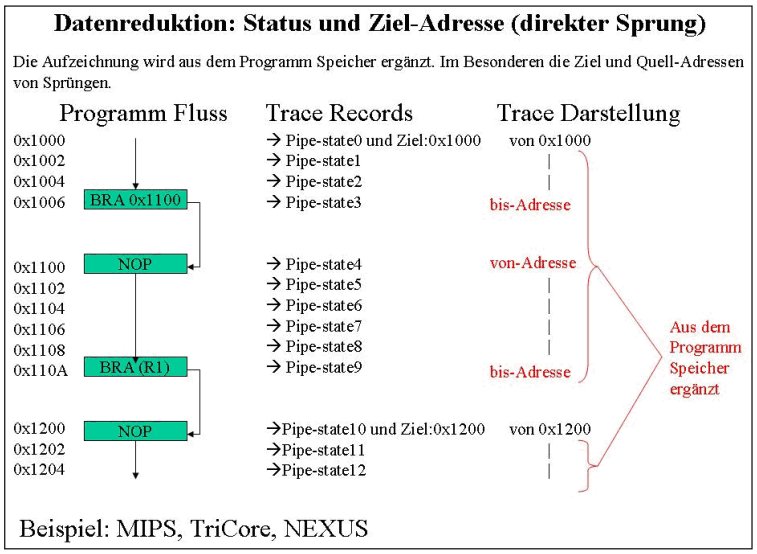

Bild 4: Eine weitere Datenreduktion ist möglich, wenn die Zieladresse von unbedingten direkten Sprüngen nicht mit übertragen wird, denn auch dies lässt sich aus dem Programmspeicher rekonstruieren. Selbst bei bedingten direkten Sprüngen ist die Rekonstruktion möglich, wenn der Pipeline-Status übertragen wird.

Datenreduktion für Daten-Zugriffe

Alle Reduktionsmechanismen die beim Programmfluss benutzt werden, können bei Daten Zugriffen nicht funktionieren. Das liegt einfach daran, dass es keine allgemeine Systematik für das Auftreten der Zugriffsadressen und deren Datenwerte gibt. Man ist deshalb auch gezwungen bei jedem Daten-Zugriff eine Adresse und einen Datenwert zu übertragen.

Anstelle von Datenreduktion tritt eine gezielte Daten-Auswahl. Alle CPUs sind deshalb mit Übertragungsfiltern ausgestattet, mit deren Hilfe der Entwickler selbst entscheidet welche Zugriffstypen an der Trace-Schnittstelle sichtbar werden.

Datenreduktion auf Protokoll Ebene

Der letzte Schritt der Datenreduktion geschieht beim Übertragungsprotokoll. Das gilt sowohl für den Programmfluss als auch für die Datenzugriffe. Bei den Adressen werden z.B. nur die Addressbits übertragen die sich im Vergleich zur letzen Übertragung geändert haben. Gerade für den Programmfluss ist dies sehr effektiv, da Programmsprünge meiste relativ ‚kurz' sind.

Für die Übertragung von Daten werden meist die führenden Nullen des Wertes abgeschnitten.

FIFO-Puffer

Die Erfahrung zeigt, dass selbst mit Datenreduktion und Highspeed Schnittstellen das Trace-Interface oftmals zu langsam ist. Es kommt gerade dann zu Überlastungen, wenn in kurzer Zeit viele Sprungbefehle im Programm ausgeführt werden. Die CPU Hersteller begegnen diesem meist temporären ‚Ansturm' mit FIFO Puffern. Die Trace-Informationen werden also erst in einen FIFO-Puffer geschrieben und dort gehalten, bis sie übertragen worden sind. Der Zeitversatz zwischen Programmausführung und Trace-Aufzeichnung ist dann nicht mehr eindeutig.

Programm-Bremse

Es gibt aber immer noch Szenarien, die selbst den FIFO-Puffer überfordern. Besonders die Übertragung von Daten-Zyklen führt regelmäßig zu einem

‚FIFO-Overflow' und zwangsläufig zu Lücken in der Trace-Aufzeichnung. Die CPUs bieten deshalb die Option die Programmausführung zu bremsen (Stall-Mode). Jedes mal wenn ein ‚FIFO-Overflow' droht, wird die CPU angehalten bis die Trace-Daten übertragen sind. Viele Applikationen verkraften diesen Performance Verlust ohne Probleme, von Echtzeitverhalten kann aber nicht mehr gesprochen werden.

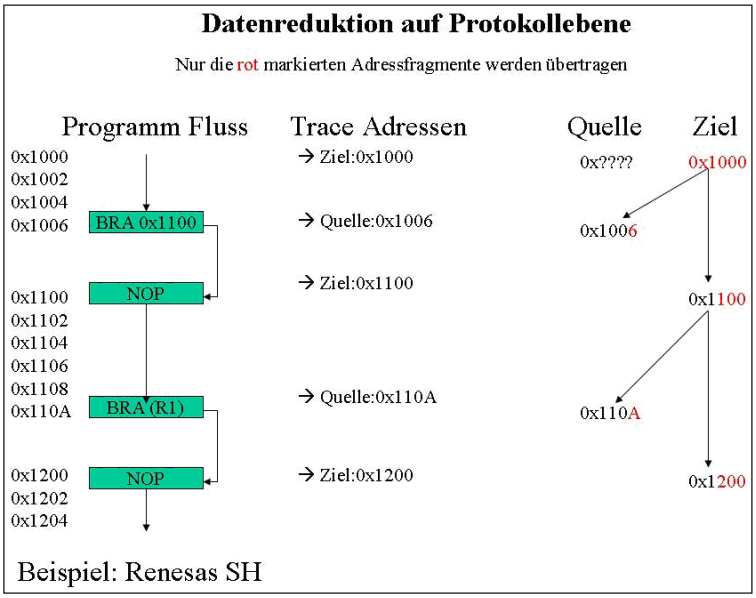

Bild 5: Datenreduktion auf Protokollebene. Bei Programmsprüngen werden nur die Bits übertragen, die sich geändert haben. Sie sind hier rot markiert.

Übertragung von Daten will gut überlegt sein

Integrierte Trace-Schnittstellen haben inzwischen einen hohen Standard erreicht. Mit den oben beschriebenen Algorithmen ist es möglich die Aktivitäten vom CPU internen Bus nach außen sichtbar zu machen. Wird nur der Programmfluss übertragen so gelingt dies meist sogar in Echtzeit. Einzig die Aufzeichnung von Datenzyklen führt regelmäßig zur Überlastung der Schnittstelle, bei aktiviertem Stall-Mode sogar zu längeren Programmlaufzeiten. Durch gezielte Auswahl der Datenzyklen kann aber dieser Effekt vom Anwender reduzieren werden.

Die mechanischen und elektrischen Details der Trace-Schnittstelle sind vom CPU Hersteller vordefiniert und werden so von den Trace-Tool Herstellen unterstützt. Der Anwender muss nur noch dafür sorgen, dass der entsprechende Trace-Stecker auf der Applikation vorhanden ist und dass die Trace Signale mit guter Qualität dort ankommen.

Trace-Tools setzen spezifische Trace-Adapter ein um den Programm- und Daten-Fluss aufzuzeichnen. Automatische Kalibrierung stellt dabei die optimale Signalqualität und Abtastung sicher. Tiefe Trace-Speicher ermöglichen lange Aufzeichnungszeiten, so dass auch weit zurück liegende Programmfehler aufgespürt werden können. Damit die große Datenmenge beherrschbar wird, sind schnelle Schnittstellen zum Bedienrechner und Leistungsfähige Such- und Filter-Funktionen erforderlich. Komfortablen Laufzeitmessungen, Statistiken und Cache-Analysen steht dann nichts mehr im Wege. Der Anwender erhält mit dem Trace-Tool eine Plug-and-Play Lösung die so gut wie keine bzw. nur einfache Setups erfordert.

Zukunftsfähige Konzepte gefragt

Integrierte Trace-Schnittstellen bieten derzeit die mit Abstand komfortabelste und leistungsfähigste Trace-Möglichkeit. Höhere CPU Taktraten werden das Bandbreitenproblem zukünftig noch weiter verschärfen.

Die Mechanismen der Datenreduktion erscheinen ausgereizt. So sind derzeit nur zwei Lösungsansätze in Sicht. Entweder wird die Übertragungsrate der physikalischen Trace-Schnittstelle erhöht oder der Trace-Speicher muss auf dem CPU Chip integriert werden.

Was die aktuelle Übertragungstechnik betrifft, so kann deren Taktrate kaum noch erhöht werden. Bereits jetzt ist die Abtastung der Trace-Signale problematisch. Höhere Datenraten können deshalb nur mit mehr Trace-Kanäle erreicht werden. Erst der Einsatz völlig anderer High-Speed Schnittstellen (z.B. LVDS) könnte dieses Problem dauerhaft lösen.

Erste CPUs mit Trace-Speicher On-Board sind bereits auf dem Markt. Bei diesen werden die gleichen

Reduktionsmechanismen eingesetzt wie oben beschrieben. Diesmal aber nicht wegen der zu geringen Bandbreite, sonder um Trace-Speicher zu sparen. Aus Kostengründen werden diese Trace-Speicher im Serienchip relativ klein bleiben müssen. Deshalb gibt es bereits wieder spezial Debug-CPUs mit grossem Trace-Speicher, mit allen Vor- und Nachteilen wie man sie von Bondout-CPUs kennt.

Vermutlich wird sich eine Mischung beider Konzepte durchsetzen. Das externe Trace-Interface kommt zum Einsatz, wenn auf die Echtzeit verzichtet werden kann bzw. lange Aufzeichnungszeiten benötigt werden. Ist aber Echtzeit gefordert so kommt der interne Trace zum Zug auch wenn die Aufzeichnungszeit relativ kurz ist.

Literaturhinweis:

„Debugger mit Rückspiegel, Tracetechniken im Überblick“