NEWS 2010

LAUTERBACH DEVELOPMENT TOOLS

DEBUGGER, REAL-TIME TRACE, LOGIC ANALYZER

グルノーブル（フランス）で開催されたIP/ESCにてパネルディスカッションに登壇したStephan Lauterbach

コミュニケーション: 成功への鍵

ローターバッハの成功は、長年にわたってお客様やパートナー各社と堅固な関係を築いてきた成果を示すものです。パートナー各社の最新技術と常に足並みをそろえ、お客様からのご要望に果敢にお応えすることで、当社では最高の開発ツールを最適な時期に投入してくることができました。ローターバッハでは、「適切なコミュニケーションには積極的な関わりが不可欠である」という理念を2010年も社是として掲げ続けます。

エキスパートセミナー

お客様の製品知識を広げるために、当社はドイツ本社の新社屋において2009年10月、業界別のTRACE32エキスパートフォーラムを初めて設立しました。このフォーラムの基本姿勢は、システムユーザーの皆様と当社のTRACE32開発者との間で情報交換を行うことにありました。当社では、皆様からの建設的なフィードバックに基づき、2010年にも同様のイベントを計画しています。

標準化への取り組み

組み込み業界のパートナー各社と対話を続ける一方、ローターバッハは標準化委員会にも参加し、情報交換と親交の重要な場として活用しています。長年にわたり、当社は標準化委員会で得られた成果を数多く当社製品に取り込んできました。このニュースレターでは、標準化への取り組みについても詳しく説明しています。

パネルディスカッション

展示会のパネルディスカッションは、現在および将来における市場の要望についてお客様およびパートナー各社と議論する格好の場です。たとえば、2009年11月には、フランスのグルノーブルで開催されたIP/ESCでStephan Lauterbachがパネルディスカッションに登壇しました。ARM社が主催したこのディスカッションでは、デバッグおよびトレース技術の将来について論じられました。

サンノゼで開催させる次回のESC Silicon Valleyで、皆様とお話できることを楽しみにしています（ブースNo.1910）。

コンテンツ

Windows 7対応の新しいTRACE32インストーラ

2009年12月に発表した新しいTRACE32　DVD（ビルド20817/リリース）で、ローターバッハはWindows　7に公式に対応するようになりました。

TRACE32インストーラが強化され、Windows　7が動作しているシステムでのTRACE32　USBドライバのインストールが容易になりました。Windows　7では、ドライバはWindows Update　Serverを使用するか、または事前にインストールされているドライバパッケージを使用して自動的にインストールするしかないため、この措置が必要でした。

新しいDVDを利用できない場合には、以下のリンクから新しい*TRACE32　USB　driver　installer　for Windows XP/Vista/7 (32-bit and 64-bit)*をダウンロードすることできます。

http://www.lauterbach.com/faq/t32usb_setup.exe

TRACE32の処理とルックアンドフィールは、Windows　7が動作するシステムでも同じです。Windowsセキュリティモデルへのサポートをさらに向上し、自動インストールを容易にするために、2009年12月からDVD上のTRACE32実行可能ファイルにも署名が付くようになりました。なお、USBドライバの署名は、2007年から使用しています。.

シリアルフラッシュデバイスのプログラミング

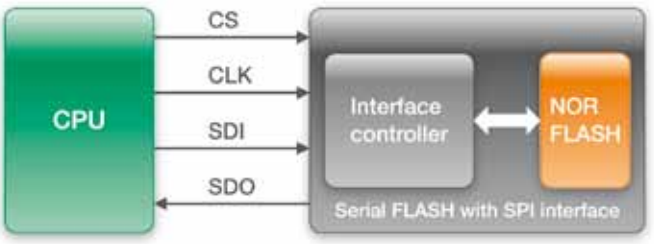


図1: SPIインタフェースでのシリアルフラッシュ

組み込みシステムの設計には、シリアルフラッシュメモリデバイスが使用されることが多くなっています。ローターバッハはこのトレンドを早くから認識し、2009年中頃からTRACE32デバッガでシリアルフラッシュデバイスのプログラミングに対応しています。また、2010年には、このサポートをさらに推し進める予定です。

シリアルフラッシュデバイスは、コンパクトな形態でピン数が少なく、エネルギー消費量も抑えられるため、コスト効率の点でNOR/NAND型フラッシュデバイスに替わるものとして注目されています。シリアルフラッシュの設計コンセプトは単純で、インタフェースコントローラがNOR/NAND型フラッシュデバイスよりも上流に接続されるため、SPIバスまたはMMCバス上でコンポーネントのプログラミングと読み出しが可能になります（図1を参照）。

TRACE32は、プログラミングだけでなく内容の読み取りや表示も含めてシリアルフラッシュデバイスをサポートしています。フラッシュの内容は一般的な16進ダンプで表示されるため、プログラムデータを迅速に確認することができます（図2を参照）。

ご使用のシリアルフラッシュに関するTRACE32の対応状況については、以下のリストをご確認ください。

NAND/シリアルフラッシュコントローラのサポート状況
http://www.lauterbach.com/ylistnand.html

NOR/NAND/シリアルフラッシュデバイスのサポート状況
http://www.lauterbach.com/ylist.html

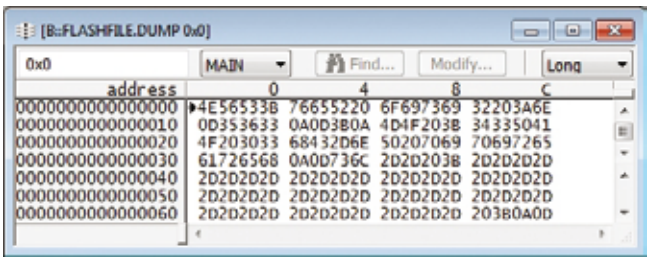


図2: シリアルフラッシュの内容をビジュアルに表示

Intel® Atom™ マイクロアーキテクチャ対応のデバッガ

2009年10月から、ローターバッハは Intel® Atom™ 向けの開発ツールを提供しています。Linuxのデバッグにはすでに完全に対応し、2010年前半にはWindows　CEへの対応が予定されています。

既にサポートされているCPU

Intel®	LA-3776 (Atom)	
	• 230	• N270
	• 330	• N280
	• D410	• N450
	• D510	• Z5XX

他の派生プロセッサにも対応予定。

ディファレンシャルロード

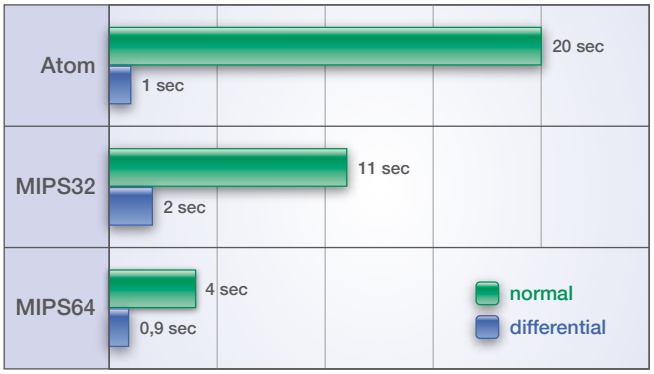


図3: ディファレンシャルロードにより4MBファイルのロード時間が大幅に短縮

デバッグサイクルは、プログラムのデバッグ → エラーの特定 → エラーの修正 → プログラムのコンパイル → プログラムのリロードという手順で進められるのが一般的です。ツールチェーンは、この各ステップを遅延なく迅速に実行できるものでなければなりません。

低速なJTAGインタフェースを介してサイズの大きいプログラムをターゲットRAMにダウンロードすると、待機時間が長くなります。この問題を解消するのがディファレンシャルロードです。ディファレンシャルロードにより、新たにコンパイルしたプログラムと、以前にロードしたプログラムとの差がわずかしかない場合に、ロード時間が大幅に短縮されます。

ディファレンシャルロードは、すでにロードしたプログラムのコピーをデバッガが保持するという仕組みを基本的なコンセプトとしています。新たにコンパイルしたバージョンをロードすると、ディファレンシャルファイルが作成されます。このディファレンシャルファイルには、元のファイルからコンパイル後の新しいバージョンへの更新に必要なあらゆる情報が圧縮された形で収められているため、デバッガはディファレンシャルファイルをターゲットシステムにダウンロードするだけで済みます。プログラムを完全に新しくコンパイルした場合と比べると、ディファレンシャルファイルは通常30分の1～100分の1のサイズになるため、ダウンロードが非常に高速になります。ダウンロードが終わると、ターゲットシステム上のロードエージェントがディファレンシャルファイルを展開し、新しいコンパイルがターゲットメモリに格納されます。

図3は、プログラムを1%だけ変更した場合のテストファイルについて実行した測定結果を示したものです。

1. Atomアーキテクチャ
   CPU: Intel Z530P
   プロセッサ周波数: 1.6GHz
   JTAG周波数: 20MHz
   通常のダウンロード: 204KB/s

2. MIPS32アーキテクチャ
   CPU: Broadcom BCM7325
   プロセッサ周波数: 167MHz
   JTAG周波数: 20MHz
   Turbo Modeでの通常のダウンロード: 370KB/s

3. MIPS64アーキテクチャ
   CPU: Cavium OCTEON Plus CN58XX
   プロセッサ周波数: 950MHz
   JTAG周波数: 50MHz
   Turbo Modeでの通常のダウンロード: 1MB/s

新サポートプロセッサ

New Derivatives	
ARC	LA-3750 (ARC) • ARC 601 / ARC 630
ARM	LA-7843 (Cortex-A / R) • Cortex-A5 / Cortex-A5MPCore • Cortex-A9 / Cortex-A9MPCore LA-7844 (Cortex-M) • Cortex-M0 / Cortex-M1
ATMEL	LA-3779 (AVR32) • AVR32 (Q2 / 2010) LA-7844 (Cortex-M) • AT91SAM3U
Broadcom	LA-7760 (MIPS32) • BCM3380 • BCM56xxx / 5836 • BCM6362 / 6368 / 6550 • BCM7401 / 7402 / 7403
CEVA	LA-3711 (CEVA-X) • CEVA-X1641 • CEVA-XC (Q2 / 2010) LA-3774 (TeakLite-III) • TeakLite-III
Cavium	LA-7761 (MIPS64) • Octeon CN54xx / CN56xx • Octeon CN63xx LA-7765 (ARM11) • ECONA CNS3XXX
Cortus	LA-3778 (APS) • APS-IP
Energy Micro	LA-7844 (Cortex-M) • EFM32
Freescale	LA-7736 (MCS12X) • MC9S12G LA-7742 (ARM9) • i.MX23 / i.MX25 LA-7732 (ColdFire) • V1 ColdFire Core LA-7753 (MPC55xx) / LA-7630 (NEXUS MPC55xx) • MPC5643L LA-7764 (PowerQUICC III) • QorIQ P1013 / P1022 / P4080
Infineon	LA-7756 (TriCore) • TC1167 / TC1197 / TC1337 • TC1367 / TC1387 / TC1387ED • TC1782 / TC1782ED
Infineon (Cont.)	LA-7759 (XC2000 / C166S V2) • XC2000ED • XC2200 / XC2300 Family • XC2700 Family • XE166 / XGOLD110
LSI	LA-7834 (StarCore) • StarPro25xx / 26xx
Marvell	LA-7742 (ARM9) • 88AP128 / 162 / 166 / 168 • MV76100 / 78100 / 78200 LA-7765 (ARM11) • 88SV581X-V6 LA-7843 (Cortex-A / R) • 88SV581X-V7 LA-7762 (XScale) • PXA93x / PXA950
MIPS	LA-7760 (MIPS32) • MIPS32 1004K / 1004KF • MIPS32 1004K CPS • MIPS32 M14K / M14Kc
NEC	LA-3777 (78K0R) • 78K0R / Fx3, 78K0R / Kx3 LA-7835 (V850) • V850E2 / Px4 •
NXP	LA-7844 (Cortex-M) • LPC13xx LA-7742 (ARM9) • LPC29xx
ST Microelectronics	LA-7753 (MPC55xx) / LA-7630 (NEXUS MPC55xx) • SPC56EL60 LA-7844 (Cortex-M) • STM32F105 / STM32F107
Tensilica	LA-3760 (Xtensa) • Xtensa 8
Texas Instruments	LA-7847 (TMS320C28X) • TMS320F28232 • TMS320F28234 / F28235 LA-7838 (TMS320C6400) • TMS320TC6424 • TMS320TCI6482 / I6488 LA-7843 (Cortex-A / R) / LA-7838 (TMS320C6400) • AM3505 / AM3517 (Sitara) • OMAP4430 / OMAP4440 LA-7742 (ARM9) / LA-7841 (TMS320C6700) • OMAP-L137 / OMAP-L138

AMPおよびSMPシステムのデバッグ

マルチコアプロセッサの多くは、AMPまたはSMPのいずれかのシステムとして使用できるため、操作のモードに応じてデバッグとトレースのコンセプトも変わってきます。このパートでは、TRACE32を使用してARM Cortex-A9 MPCoreをデバッグする際に、このコンセプトがどう適用されるかについて説明します。

デバッグのコンセプト

お客様と意見交換を行う中で、当社は以下の2つの用語がいかに多様に解釈されているかを何度も実感してきました。

- AMP - 非対称型マルチプロセッシング
- SMP - 対称型マルチプロセッシング

そのため、ここではローターバッハでの各用語の使い方と、TRACE32デバッガの設定や使用方法への影響について説明します。

「マルチプロセッシング」という用語が示すように、組み込みシステムでは複数のコアが同時に動作しています。デバッグの際に重要なのは、システムタスクをこの個々のコアに分散する方式です。

AMPシステムにおけるデバッグのコンセプト

AMPシステムの場合、各コアに特定のタスクが割り当てられます。タスクの分散方法はシステムの設計段階で決定され、標準的なコントローラ（通常はRISCアーキテクチャ）に加え、特殊化したアクセラレータ（DSPあるいはカスタムコア）もよく利用されます。

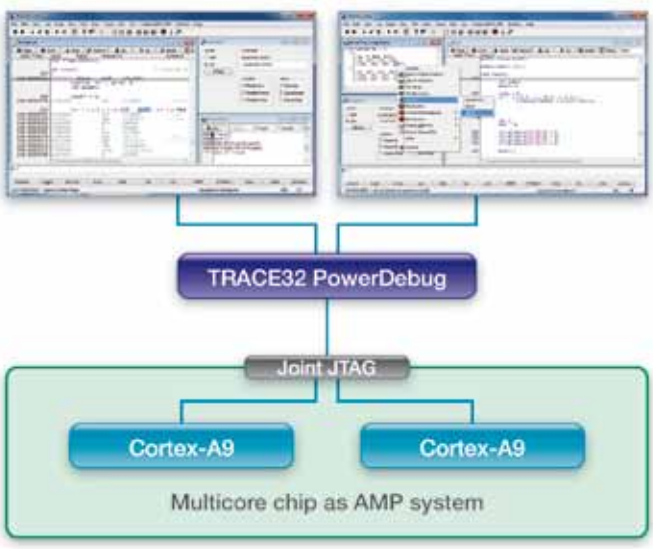


図4: AMPシステムのデバッグでは、コアごとに各TRACE32インスタンスが起動

AMPシステムをデバッグする際には、コアごとに各TRACE32インスタンスが起動します（図4を参照）。これには2つの理由があります。

1. AMPシステムには種類の異なるコアアーキテクチャが混在できる。

2. 各コアがアプリケーションの個々の部分を別々に処理する。つまり、シンボルとデバッグ情報の大部分はそれぞれ対応するコア専用として割り当てられる。

ただし、各コアは独立して動作するわけではなく、アプリケーションタスクを同時に並行して実行するため、すべてのコアが開始と停止を同時に行わなくてはなりません。コア間の相互動作をテストし、アプリケーション全体を監視および制御するには、この方法しかないのです。また、すべてのコアの開始と停止を同時に行うには別の方法もあります。マルチコアプロセッサが内部の同期ロジックを通じてこれに対応していれば理想的ですが、このロジックがない場合には、TRACE32が同期プロセスを引き受けます。そして、特殊なアルゴリズムがJTAGコマンドのシーケンスを計算し、可能な限り速やかにすべてのコアを制御します。

SMPシステムにおけるデバッグのコンセプト

各コアに割り当てるタスクが事前に定義されているAMPシ »

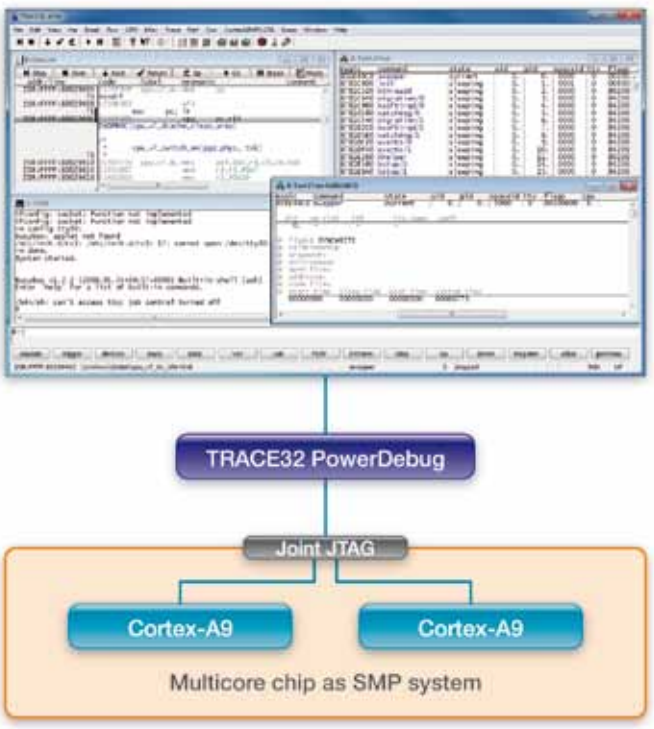


図5: SMPシステムのデバッグでは、すべてのコアに対して1つのTRACE32インスタンスが起動

ステムとは対照的に、SMPでは柔軟に割り当てが行われます。SMPシステムの場合、システム設計者がタスクをコアに割り当てるのではなく、SMPオペレーティングシステムが割り当てを行います。なお、必要に応じて各コアに自由にタスクを割り当てるには、すべてのコアが同じタイプでなければなりません。

タスクの割り当てが動的に実行されるため、割り当ては現在のシステム状態によって異なることになります。オペレーティングシステムによって割り当てが可能な処理の単位を「タスク」または「スレッド」と呼びます。簡単に言うと、処理が必要なタスクは現在空いているコアに割り当てられます。

SMPシステムのデバッグでは、TRACE32インスタンスが1つだけ起動し、すべてのコアがそこから一元的に制御されます（前ページの図5を参照）。デバッグの際に開発者が扱うのは、通常1つのタスクだけであるため、TRACE32のユーザーインタフェースには、その単一のタスクから見た、あるいはタスクが実行されるコアから見たSMPシステム全体の状態が表示されます。もちろん、必要に応じて別のタスクやコアに表示を切り替えることも可能です。

TRACE32にはSMPオペレーティングシステムに類似した機能が備わっており、全コアのデバッグが編成されるため、開発者はSMPシステムの詳細まで調べる必要はありません。例えば、ブレークポイントを1つ設定すると、TRACE32は全てのコアにブレークポイントを設定します。これは、ブレークポイントを設定する時点では、そのブレークポイントを含むプログラムのセクションがどのコアで実行されるかわかっていないためです。コアがブレークポイントで停止すると、他のコアもすべて自動的に停止し、TRACE32の表示は、ブレークポイントに遭遇したタスクまたはコアに切り替わります。プログラムを再開すると、すべてのコアが同時に実行を開始します。

TRACE32によるSMPシステムのデバッグは容易です。TRACE32インスタンスが起動し、SMPシステムに応じて設定されれば、開発者は基本的に1つのコアをデバッグするときと同様にTRACE32を使用することができます。

トレースのコンセプト

TRACE32では、トレースデータがAMPシステムとSMPシステムのどちらで生成されたかに応じて、トレース情報の解析と表示の仕方が異なります。AMPシステムの場合、ほとんどのトレース解析は各コアに対して独立して実行されます。一方、SMPシステムのトレース情報はクエリーのタイプに応じて1つのタスク、1つのコア、またはシステム全体に対して解析することができます。

AMPシステムにおけるトレースのコンセプト

AMPシステムの各コアに対するデバッグは、それぞれのTRACE32インスタンス上で実行されるため、トレース情報も個別のユーザーインタフェース上に表示されます。AMPシステムは、様々なタイプのコアで構成できるため、複数のトレースプロトコルが必要になる場合もあります。各トレースのストリームは個別のユーザーインタフェースで表示されるため、デコードと解析は個別に行うことができます。

また、コアの相互動作をテストし、複雑なシステムエラーを短時間で特定するために、各トレースのビューとその相互の関係を時間軸に沿って表示することができます。そのために、TRACE32 PowerTraceには共通の時間ベースが用意され、開発者は1つのユーザーインタフェース上でトレースビューの時間軸上の点を選択すれば、ほぼ同じ時点に他のコアでどのようなコマンドが実行されたかを正確に知ることができます（図6を参照）。

SMPシステムにおけるトレースのコンセプト

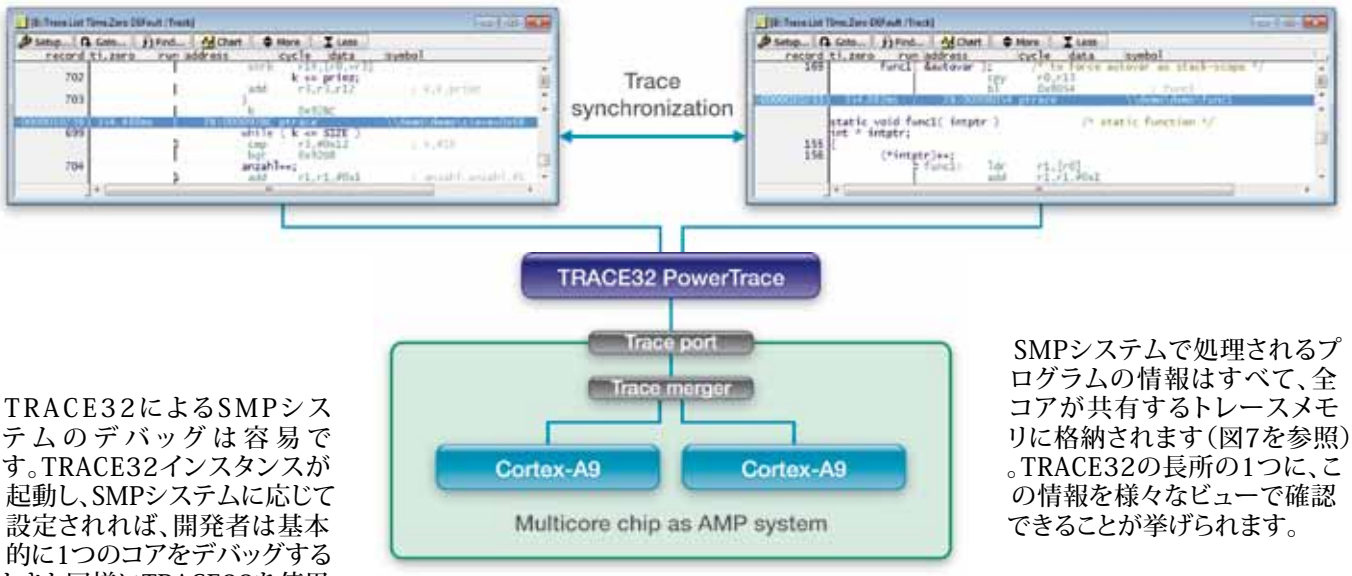


図6: AMPシステムのトレースでは、各コアのトレース情報が個別のユーザーインタフェース上に表示される。ユーザーインタフェースの時間的な同期が可能

SMPシステムで処理されるプログラムの情報はすべて、全コアが共有するトレースメモリに格納されます（図7を参照）。TRACE32の長所の1つに、この情報を様々なビューで確認できることが挙げられます。

ここでは、タスクにおけるエラー » や、タスク固有の実行時の

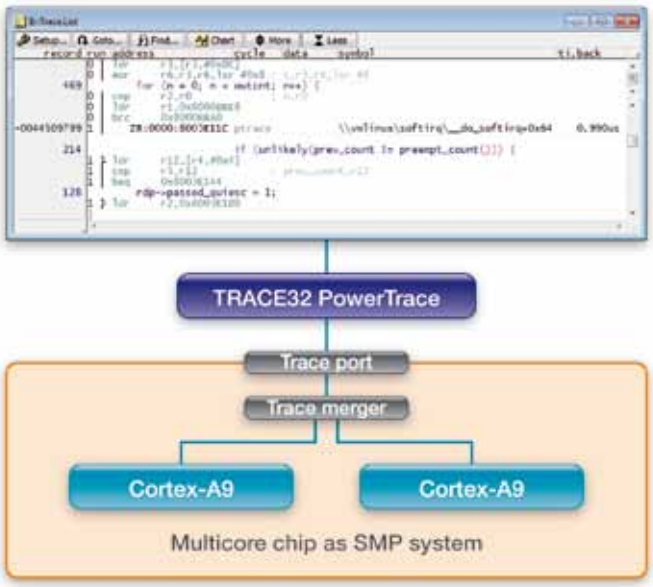


図7: SMPシステムのデバッグでは、すべてのコアのトレース情報が共有のトレースメモリに格納される

測定値を特定するために、個々のタスクを指定してトレース情報を表示することができます。

「自分のタスクはどのコアで処理されたか」、「自分のコアの実行時の負荷はどのくらいか」といった情報を知りたい場合には、すべてのコアのトレース情報を同時に表示する方法が適しています。図8に、このビューの例を示します。ここで、コア番号(0、1)は、各プログラムセクションが実行されたコアを示しています。

SMPシステム全体を調べる場合、タスクやプログラムセクションがどのコアで処理されたかを知る必要はありません。TRACE32では、このようなSMPシステムのビュー

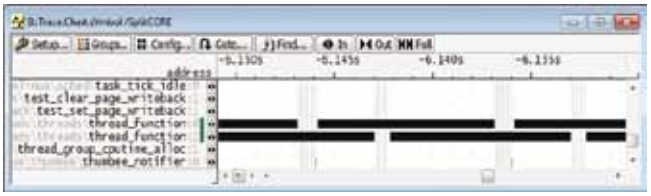


図8: トレース解析では、それぞれのプログラムセクションをどのコアが処理したかが示される

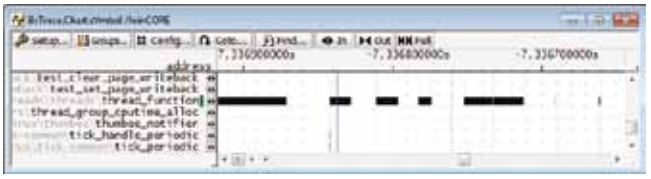


図9: トレース解析はSMPシステムを全体的に解析する。ここでは、どのコアがどのプログラムセクションを処理したかは重要ではない

を示す表示オプションもあります(図9を参照)。

2010年、ローターバッハではSMPシステムからトレース情報を作成して表示する機能を引き続き強化する予定です。これには、既存ユーザーからのフィードバックに基づいた新しい解析機能や、現在開発を進めている新コンセプトも含まれています。

RTOSデバッグの最新情報

新サポートRTOS	
FAMOS for ARM	available
Linux for Atom	available
SMP Linux for MIPS64	available
OKL4 for ARM	available
OS21 for ARM	available
SMP Symbian OS for ARM	available
Windows CE for Atom	Q1/2010

新しいRTOSバージョン向けの更新

- LynxOS 5.0 for PowerPC
- MQX 3.x for ColdFire
- OSE 5.4
- QNX 6.4
- VxWorks 6.4
- µC/OS-III

機能拡張

- µC/OS-IIでパーティションとMPU/MMUをサポート
- Symbian OSのページブレークポイント
- VxWorksでRTPをサポート

標準化委員会

今日、デバッグおよびトレース技術に対して、高いレベルでの標準化を望むお客様が増えています。適切な標準の策定において積極的な役割を果たすため、ローターバッハは、長年にわたり多くの国際委員会に参加してきました。そのような積極的な参加経験を活かして、ローターバッハでは新しい標準の承認後、その標準に準拠したフルサポートをすぐにTRACE32に採用しています。

ORTI標準

ローターバッハが初めて積極的に策定を支援したのはORTI標準でした。ORTI標準とは、RTOS対応のデバッグでOSEKオペレーティングシステムのオブジェクトの構造とメモリマッピングを記述し、それをファイルに格納するための記述言語に関する標準です。ORTI標準はAUTOSARコンソーシアム内で策定され、2003年以降には、ETAS GroupやVector社などのOSEKオペレーティングシステムの全プロバイダが採用するようになりました。

ORTIファイルの処理は以下のように行われます。:
まず、OSEK System Builderが、OSEKオペレーティングシステムのユーザー設定からORTIファイルを作成します。 次に、OSEK対応のデバッグが可能になるように、ORTIファイルがTRACE32デバッガにロードされます（図10を参照）。.

TRACE32により、開発者は以下の機能を使用できるようになります（OSEK System Builderが必要な情報をORTIファイルに保存していると仮定）。

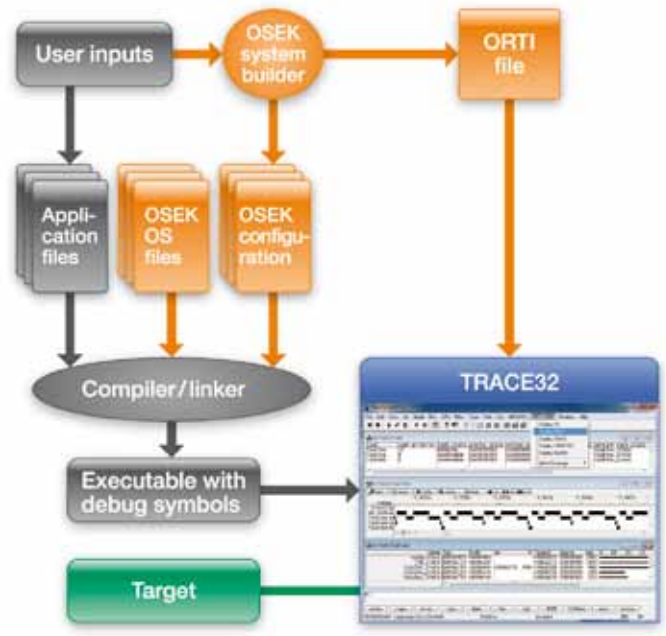


図10: OSEK System BuilderがORTIファイルを作成し、TRACE32へのファイルのロード後にOSEK対応のデバッグが可能になる

AUTomotive Open System ARchitecture

www.autosar.org

Working group
OSEK/VDX Debug Interface Working Group

Standard
OSEK Run Time Interface Version 2.2, Nov 2005
http://portal.osek-vdx.org/files/pdf/specs/orti-a-22.pdf
http://portal.osek-vdx.org/files/pdf/specs/orti-b-22.pdf

- OSEKリソースの直感的な表示（図11はアラームリスト）
- タスクごとのブレークポイント
- タスクのスタック解析（図12を参照）
- タスクのコンテクスト表示
- タスク実行時間解析（図14を参照）
- サービス実行時間の解析

NEXUS標準

1998年のNEXUS 5001フォーラムにおいて、組み込みプロ

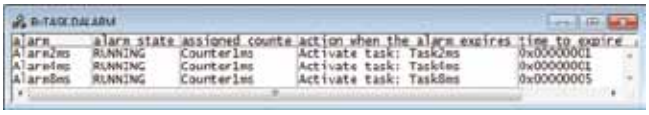


図11: TRACE32におけるOSEKオペレーティングシステムのアラームリスト

図12: TRACE32におけるOSEKオペレーティングシステムのスタック解析

セッサのデバッグインタフェースとトレースインタフェースに関する標準化に向けて第一歩が踏み出されました。NEXUS標準は、その後2003年に承認されました。NEXUSには以下のような内容が含まれています。

- JTAGインタフェース（一般的にはIEEE1149.1）
- プログラムを実行したままデバッガがメモリに対して読み書きを実行できるメカニズム
- メッセージベースのトレースプロトコル（プログラムフローとデータトレース）
- マルチコアプロセッサのデバッグとトレース
- ハードウェアレイヤー
- 標準のNEXUSコネクタ

ローターバッハは2001年以降、各種のプロセスアーキテクチャでNEXUS標準をサポートしています。その中で最新かつ最も重要なのが、自動車業界で広く利用されているFre- »

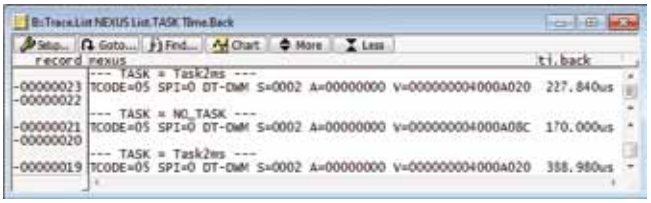


図13: 記録タスク切り替え時のNEXUSメッセージ

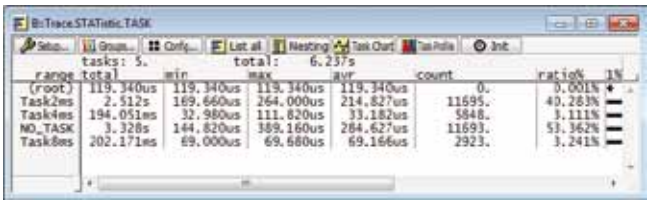


図14: OSEKオペレーティングシステムのタスク実行時の解析

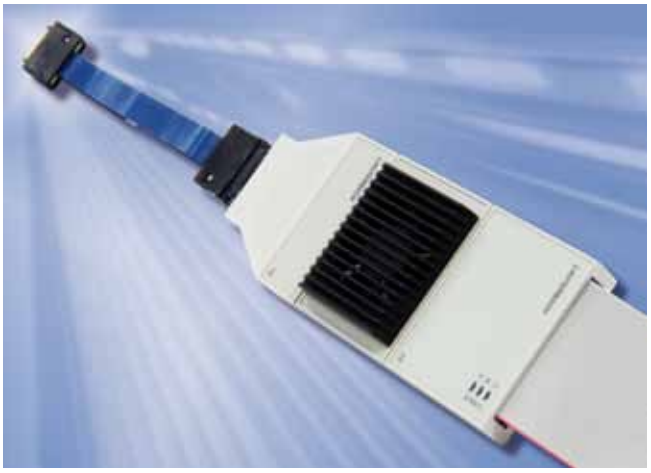


Fig. 15: TRACE32のQorIQ用高速シリアルトレース

escale社およびST Microelectronics社のMPC55xx/MPC56xxシリーズプロセッサです。図12と13は、OSEKタスクの実行時間を測定する際にNEXUSのトレースログを使用している様子を示したものです。

NEXUS標準は実装の自由度が高いため、ローターバッハは2008年にNEXUSハードウェアを完全に新しく設計し直しました。FPGAベースの新しいハードウェアはきわめて柔軟に設計されているため、NEXUSのあらゆる実装に対応できます。新しい機能には、トレース信号のサンプリングポイントの最適化や、新しいJTAGプロトコルであるIEEE 1149.7のサポートなどが含まれます。IEEE 1149.7の詳細については、10ページを参照してください。

2003年版のNEXUS標準は現在改訂が進められています。改訂版はまだ承認には至っていませんが、MPC56xxシリーズのプロセッサの一部はすでにこの新標準を実装しています。TRACE32での新しいプロトコルへの対応には、現在のところソフトウェアアップデートを利用しています。

QorIQ用シリアルトレース

ローターバッハは、2010年4月のESC Silicon Valleyにおいて、新しいQorIQ用高速シリアルトレース（図15を参照）を発表する予定です。QorIQ P4xxxのトレース技術は、NEXUSプロトコル、AURORAベースのハードウェアレイヤー、そしてPower.org組織のワーキンググループで承認されたコネクタに基づいています。

最初にサポートされるプロセッサはFreescale社のQorIQ P4080となる予定です。P4080 SoCはPower Architecture e500mcの8コアを採用し、SMP設定またはAMP設定で動作しますが、SMPとAMPのグループを組み合わせた設定も可能です。「AMPおよびSMPシステムのデバッグ」の詳細については、このニュースレターの5ページを参照してください。

パラレルトレースインタフェースと比較すると、シリアルトレースインタフェースには、シリアル転送のためにピン数が少なくなる、データ転送にディファレンシャル信号を利用するためにスループットが高くなる、という利点があります。

生成されるトレースデータが大量になれば、それに応じて大きいトレースメモリのバッファサイズが必要になります。これを解決するのが、4GBのメモリを持つローターバッハのPowerTrace IIです。

QorIQ用の高速シリアルトレースは、最大4つの高速チャネルを想定して設計されています。サポートされる転送速度は以下の通りです。

- 3チャネルまでの場合、チャネルあたり最大6.25GBit/s
- 4チャネルまでの場合、チャネルあたり最大3.125GBit/s

トレースデータは、Samtec社製のコネクタシステムを介して集録されます。ローターバッハでは、各種コネクタ用のアダプタを提供しています。

IEEE 1149.7

IEEE 1149.7とは、最新の技術要件に対応するために従来のJTAG標準であるIEEE 1149.1を更新した標準 »

Nexus 5001™

www.nexus5001.org

Standard
Standard for a Global Embedded Processor Debug Interface Version 2, December 2003
http://www.nexus5001.org/st/ieee_isto_5001_2003.pdf

Power.org
www.power.org
Working group Common Debug Interface Technical Subcommittee
Standard Power.org™ Standard for Physical Connection for High-Speed Serial Trace, July 2008 http://www.power.org/resources/downloads/Power_CDI_Physical_Connection_for_HSST_APPROVED_v1.0.pdf

IEEE Standards Association
www.standards.ieee.org
Working group: 1149.7
Standard Official approval of the standard is planned for Q1/2010

で、cJTAG（compact　JTAG）とも呼ばれます。業界をリードするデバッガメーカーとして、ローターバッハはこの新しい2ピンインタフェースの策定にも深く関わってきました。

IEEE 1149.7標準に準拠すると、デバッガとコアの間の通信全体がTCKピンとTMSピンで実行されます。標準のデバッグ通信と比べ、新しい標準には以下のような特徴があります。

- IEEE 1149.7では、外側の2ピンコネクタしか使用しません。チップ上では、1149.7コントローラが通信を標準のJTAGに再変換します（図16を参照）。
- 簡単に言うと、IEEE 1149.7は標準のJTAGインタフェースとは異なり、シリアル化されます（図17を参照）。

標準のJTAGにおけるTMS信号は単方向性であるため、ローターバッハは双方向性のTMSに対応するようにデバッグケーブルを適応させる必要がありました。

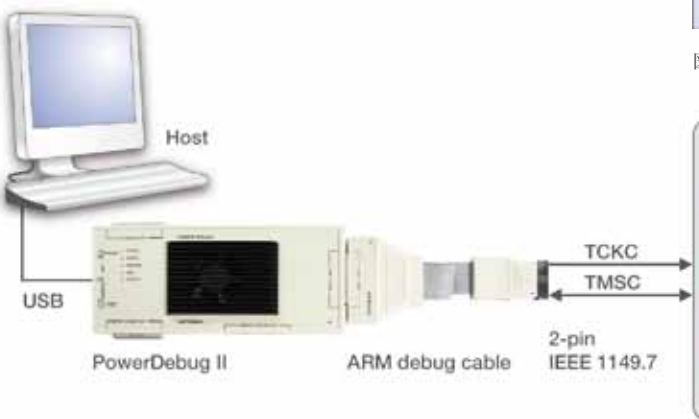


図16: IEEE 1149.7プロトコルは、チップ上で標準のJTAGプロトコルに再変換

- ARM/Cortexアーキテクチャ向けのデバッグケーブルはすべて、2008年の最初に変更されました（Debug Cable V4）。
- MPC55xx/MPC56xxアーキテクチャ向けデバッグケーブルの最新版は、2009年9月から利用できるようになりました（シリアルJTAG付きOnCE Debug Cable）。

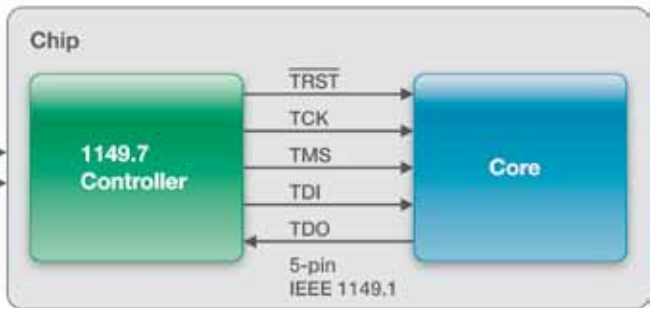
もちろん、新しいデバッグケーブルでも従来の標準が引き続きサポートされています。

TRACE32は2009年10月にcJTAGテストに合格しましたが、リリースは1149.7標準の公式承認後になります。

MCD API

MCD　APIでは、マルチコアシステムのデバッグにおけるCインタフェースが定義されています。ここではマルチコアシステムが実際のハードウェアとソフトウェアシミュレーションのどちらで構成されるかは問われません。ローターバッハでは現在、MCD APIの応用例を2つ確認しています。 »

図17: 1149.7プロトコルの簡略図



Open SoC Design Platform for
Reuse and Integration of IPs

SPRINT

www.sprint-project.net

Working group: Debug and Analysis

Standard: Multi-Core Debug API v1.0, April 2009
http://www.lauterbach.com/sprint_mcd_api_v1_0.zip

1. TRACE32への標準インタフェースとしてのMCD APIである、新しいTRACE32 MCDリモートAPI（図18を参照）
2. 仮想プロトタイプの標準インタフェースとしてのMCD API

1. 更新されたTRACE32リモートAPI

ローターバッハのリモートAPIでは、外部アプリケーションでTRACE32を制御することができます。たとえば、回帰テストの自動化に利用することが可能です。

当社のリモートAPIは、長年にわたりお客様から多くのご要望を受けて開発が進められてきましたが、現在マルチコアシステムの増加に伴い、より総合的な変更が求められています。

ローターバッハは、現行のリモートAPIを改訂するのではなく、MCD標準に基づいてリモートAPIを設計し直すことを決定しました。MCD APIはTRACE32の要件をすべては網羅しない可能性も高いため、TRACE32に固有の機能が追加される予定です。

新しいTRACE32 MCDリモートAPIの最初のバージョンは2010年中頃のリリースを予定しており、現時点で従来のレガシーリモートAPI上での開発はすべて凍結されています。レガシーリモートAPIは引き続きサポートされますが、新しい機能は今後実装されませんので、お客様はその点にご注意ください。

図18: リモートAPIを介してTRACE32を外部アプリケーションで制御可能.

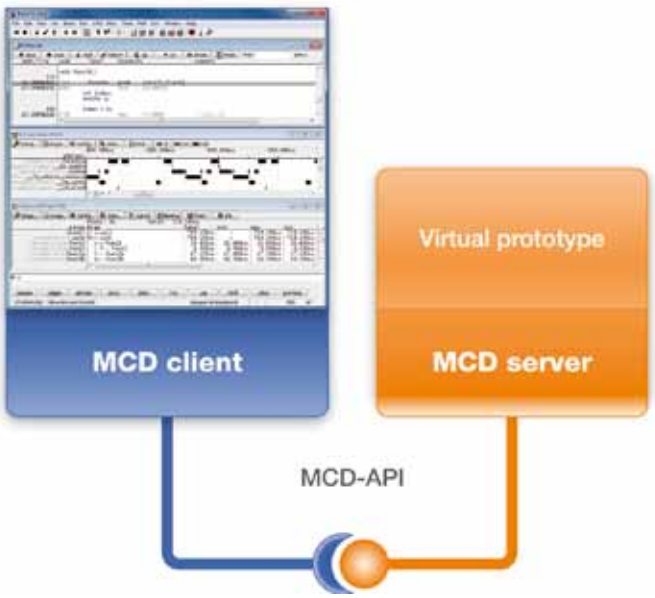


図19: 仮想プロトタイプのデバッグはMCD-APIを介して実行

2. 仮想プロトタイプ用の標準インタフェース

過去5年間にも、TRACE32は仮想プロトタイプのデバッグを行うことが可能でした。通常、仮想プロトタイプのメーカーは、この目的で独自のデバッグAPIを提供しています。MCD APIは、デバッガと仮想プロトタイプの間の標準インタフェースとなり（図19を参照）、これには以下のような利点があります。

- 新しい仮想プロトタイプに迅速に適応
- 十分にテストされた高性能な機能を豊富に提供

ヒント - Read/Writeブレークポイントの追加

オンチップのブレークポイントはデバッグリソースとしてきわめて貴重であるため、ほとんどのプロセッサでは数が非常に限定されています。これまでも、コアにEmbedded Trace Macrocell(ETM)が組み込まれていればARMコアに2つの読み書きブレークポイントを追加するという設定は可能でした。これらのブレークポイントを利用すると、高い精度でブレーク条件を定義できるというオプションが追加されますが、残念なことに、この便利なオプションを意識している開発者はまだ多くはありません。

追加の Read/Write ブレークポイント	
ARM7/9/10/11 Cortex-A5/-A8/-R4	追加で2つの read/writeブレークポイント(データ値指定も可)
Ceva-X/TeakLite	追加で2つの read/writeブレークポイント(データ値指定も可)
TMS320C6400	追加で最大4つの read/writeブレークポイント(データ値指定も可)

表1:トレースロジックベースのブレークポイントが可能なアーキテクチャ

効率的なデバッグのためには、特定のデータ値が変数に書き込まれたときにプログラムを一時停止できることが重要です。残念なことに、一部のARMまたはCortexコアでは、必要な読み書きブレークポイントを提供していません。たとえば、Cortex-A8のプロセッサデバッグロジックでは、ブレークポイントがこの機能に対応していません。

ただし、Cortex-A8にETMロジックがあれば、ETMで提供される2つのアドレスおよびデータコンパレータを使用して、TRACE32でこの機能を提供することが可能です。これにより、このコンパレータは、トレース情報をフィルタリングしてトリガーを生成するという当初の機能には利用できなくなりますが、デバッグ時にはこの機能が使用されることはほとんどないため、この点はあまり問題にはなりません。

「ETM.ReadWriteBreak」のオプションを設定すると、TRACE32のリソース管理を再設定して、ETMの2つのアドレス/データコンパレータを標準の読み書きブレークポイントとして使用できるようになります。

これらのブレークポイントは、Cortex-A8のプロセッサデバッグロジックで用意されている標準のブレークポイントと比べ、優れた機能を発揮します。たとえば、プロセッサデバッグロジックの読み書きブレークポイントではビットマスクを使用してアドレス領域をマーキングするだけですが、ETMのブレークポイントではアドレス領域を正確に定義することができます。そのため、TRACE32はETMベースのブレークポイントを優先します。表2は、この2種類のブレークポイントを比較したものです。

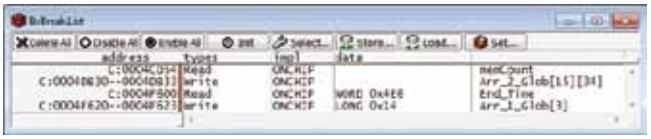


Fig. 20: 先にETMベースの2つのread/writeブレークポイント、その後Cotex-A8のブレークポイントが使われる

Cortex-A8のプロセッサデバッグロジック
1～16 個のread/write ブレークポイント ビットマスクでのアドレス指定 データ値指定なし
Cortex-A8のETMベースのブレークポイント
2 個のread/write ブレークポイント 正確なアドレス指定 正確なデータ値指定でのread/writeブレークポイント

表2: ETMベースのブレークポイントを追加して機能を強化

ここで、使用中のデバッガがTRACE32 JTAGデバッガのみの場合でもETMブレークポイントは使用できることに注意してください。これは、標準のJTAGインタフェース上でもETMコンパレータを構成できるためです。

また、Cortex-A8について述べたように、トレースロジック機能を他のアーキテクチャで使用すると、ブレークポイントを追加して機能を強化することができます(表1を参照)。

世界の支社

- ドイツ
- フランス
- イギリス
- イタリア
- アメリカ
- 中国
- 日本

その他の国でも経験豊富な販売代理店が対応させていただきます。