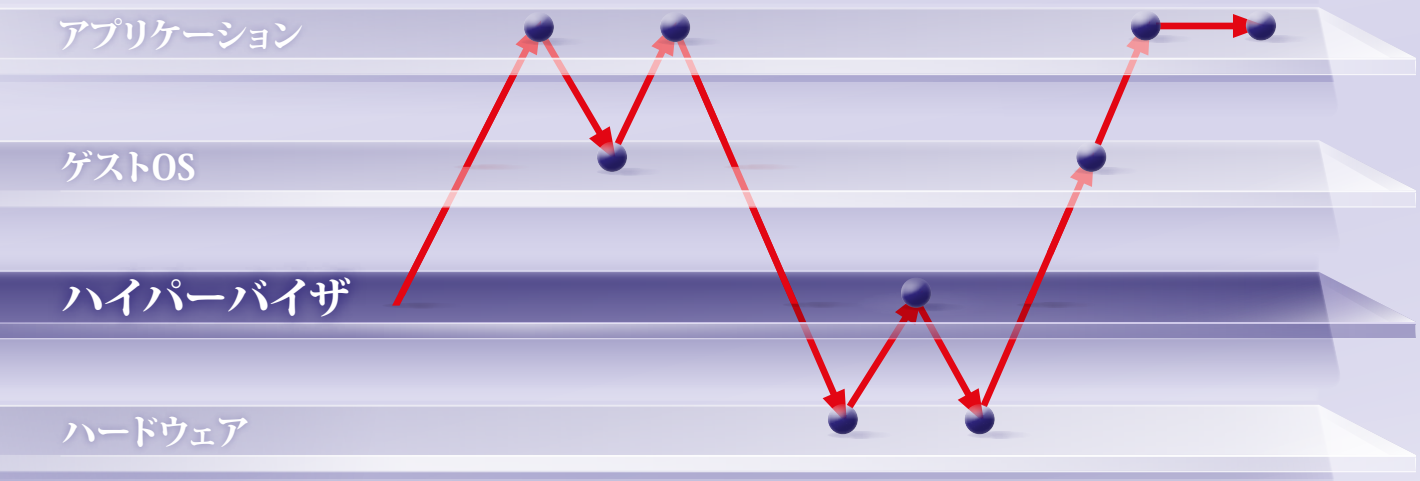
NEWS 2017

日本語版



すべてのソフトウェアレイヤを
シームレスにデバッグ

コンテンツ

LAUTERBACH
DEVELOPMENT TOOLS

ハイパーバイザデバッグ

2017年4月にローターバッハはより高度なハイパーバイザ・サポート機能を提供する予定です。本稿ではLeMaker版HiKeyボード上(Cortex-A53搭載)のXenハイパーバイザにゲストOSとして2つのLinuxが動作しているものをリファレンス実装として紹介しています。

組込みシステムにおける仮想化

仮想化の構想としては、単一のハードウェアプラットフォーム上で複数のオペレーティングシステムが並列に動作できます。仮想化は近年、より多くの組込みシステムに採用されるようになりました。例えば自動車の運転席では、Androidベースのユーザーインターフェースと並行して同一ハードウェア上でAUTOSARオペレーティングシステムで管理されたリアルタイムアプリケーションが動作しています。仮想化のコアであるハイパーバイザは、全体動作が正確かつ効率的に実行されるように制御します。

仮想マシンモニタとも呼ばれるハイパーバイザはソフトウェアレイヤであり、以下の2つの役割を果たします。

1. 仮想マシン(VM)の起動と管理
2. VMに対して物理的ハードウェア資源の仮想化

VM上で稼働しているオペレーティングシステムは、ゲストOSと呼ばれます。ゲストOSがアクセスする仮想化されたハードウェア資源はハイパーバイザにより物理的資源へ割り付けられます。

CPUの仮想化はデバッグにおいて重要です。

各仮想マシンには1個以上の仮想CPU(vCPU)が割り当てられます。vCPUの数はハードウェアプラットフォームで利用できるCPUコアの数と異なることもあります。

メモリの仮想化も同様に重要であり、VMは実際の物理メモリでなく、ゲストOSの物理メモリを仮想化メモリとして認識します。ハイパーバイザは各VM用に個別のページテーブルを管理し、物理メモリへのアクセスを制御します。少なくともLinuxのようなOS上のアプリケーションプロセスは仮想アドレス上で動作するため、デバッガは次のように2段階でアドレス変換処理をする必要があります。

- ゲストOSの仮想メモリからゲストOSの物理メモリ
- ゲストOSの物理メモリからホストOSの物理メモリ

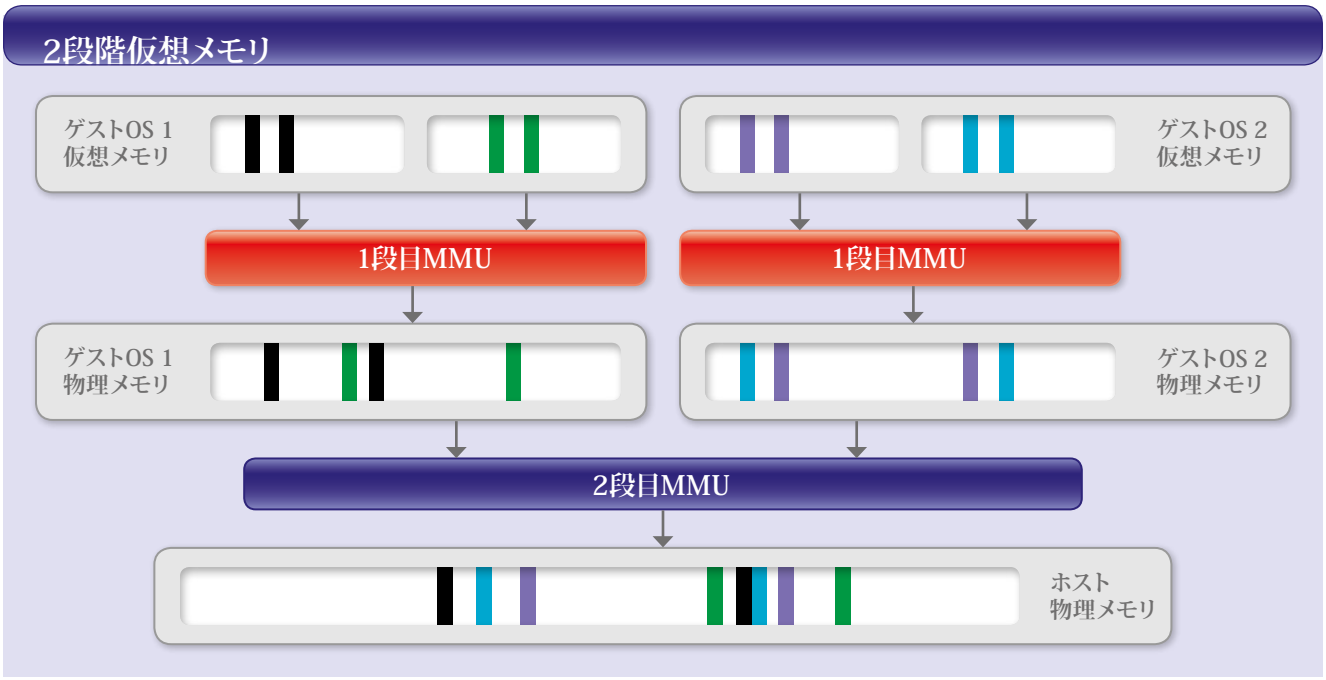
次ページの「2段階仮想メモリ」図から、

- 1段目のMMUのマッピング情報はゲストOSのページテーブルにより処理
- 2段目のMMUはハイパーバイザのページテーブルを使用

ということがわかります。

拡張デバッグ構想

2016年、TRACE32は体系的に拡張され、ハイパーバイザ対応の制限のないデバッグ機能を提供できるようになりました。追加されたのは、次の拡張機能です。



- マシンIDの追加でアクティブ/非アクティブVMコンテキストにアクセスできます。仮想マシンは実行のためのコアが割り当てられている場合にアクティブと考えられます。
- 最新ハイパーバイザ認識機能により、デバッガはハイパーバイザのVMを検出、可視化します。
- 単一のOSのみのデバッグではなく、複数のOSの同時デバッグが可能になりました。
- これまでデバッガはアクティブなゲストOSのページテーブルにしかアクセスすることができませんでしたが、非アクティブなゲストOSのページテーブルにもアクセスできるようになりました。

これら拡張の最も重要な点は、システム全体にわたるシームレスなデバッグ機能の提供です。つまり、システムがブレークポイントで停止した際、各プロセス、VM、ハイパーバイザ、そして実ハードウェアプラットフォームの現行状態を確認、変更することができます。また、コードのどの位置にでもプログラムブレークポイントを設定できます。

これら全ての拡張機能のスタートポイントだったLinux等のオペレーティングシステム用の制限のないデバッグ機能の提供を開始してから約20年が経過し、あらためてOSデバッグコンセプトの最重要機能を簡潔にまとめると:

プロセスはOSのプライベート仮想アドレス空間で実行されます。TRACE32 OS認識機能とTRACE32 MMUサポート機能により、プロセス境界を超えたシームレスなデバッグが可能です。

- スペースIDにより、各プロセスの仮想アドレス空間に直接アクセスできます。
- TASKオプションにより、各プロセスの現行のレジスタセットとスタックフレームを表示できます。

マシンID

オペレーティングシステムが仮想マシン上で稼働している場合、OSデバッグ構想はどのように拡張すれば良いのでしょうか。

1. 最初に各仮想マシンを一意的に識別する必要があるため、TRACE32は各VMにマシンIDを割り当てます。ハイパーバイザのマシンIDは0です。スペースIDがプロセスの仮想アドレス空間を識別するために使用される様に、マシンIDはVMのプライベートアドレス空間を識別するために使用されます。
2. プロセスのレジスタセットとスタックフレームを表示するには、デバッガはプロセスが実行されている VM とゲスト OS を認識する必要があります。このためにMACHINE オプションが導入されました。

TRACE32コマンド

従来のOS認識デバッグ	Data.dump	<space_id>:<virtual_address>
	Data.LOAD.Elf <file>	<space_id>:<virtual_address>
	Register.view	/TASK <process_name>
	Frame.view	/TASK <process_name>
< NEW > ハイパーバイザデバッグ	Data.dump	**<machine_id>:::**<space_id>::<virtual_address>
	Data.LOAD.Elf <file>	**<machine_id>:::**<space_id>::<virtual_address>
	Register.view	**/MACHINE <machine_id>** /TASK <process_name>
	Frame.view	**/MACHINE <machine_id>** /TASK <process_name>

この2つの拡張機能により、デバッガはプロセス境界を超えて全ての情報にアクセスすることができます。上記の「TRACE32コマンド」では、ハイパーバイザデバッグ向けに拡張したTRACE32コマンド構文とOS認識デバッグで使用する構文の違いを示しています。

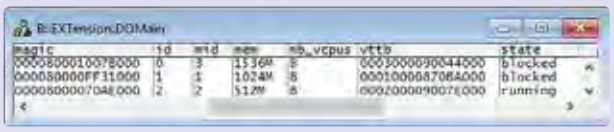
ハイパーバイザ認識機能

OS認識機能と同様にハイパーバイザ認識機能が用意されています。この機能はハードウェアプラットフォームで稼働しているハイパーバイザ上の全ての情報をデバッガに提供します。一方、ハイパーバイザにデバッグシンボルをロードする必要があり、ロードが済むとデバッガは各ゲストOSの一覧を作成できます。「ゲストOSリスト」画面例では、リファレンス実装：Cortex-A53上のXenを用いており、以下の情報が表示されています。

- VM IDとVMの状態、VM毎のCPUの数
- 2段目のページテーブルのスタートアドレス(vttb)

特定のハイパーバイザ用の認識機能はローターバッハが作成し、お客様に提供します。現在サポートしているハイパーバイザの一覧は、5ページ記載の「現在サポートしているハイパーバイザ」をご参照ください。

ゲストOSリスト



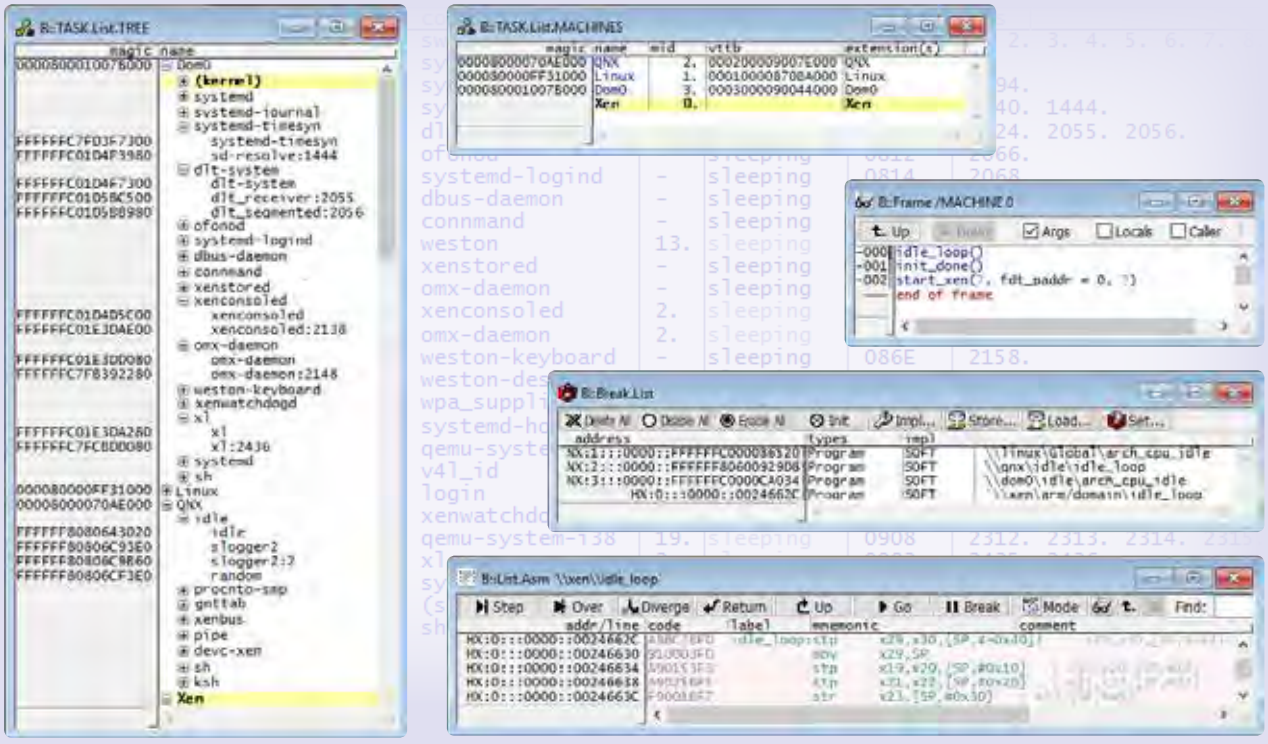
デバッガの設定

拡張デバッグ構想ではTRACE32を使用したデバッグはどのように変わったのでしょうか。最初に設定をみてみましょう。ハイパーバイザと各シングルゲストOSを以下の手順で設定します。

1. デバッグシンボルをロードします。
2. ページテーブル認識機能(MMU)を設定します。
3. TRACE32ハイパーバイザ認識機能をロードし、それぞれTRACE32 OS認識機能についても行います。

Cortex-A53上のXenハイパーバイザ

グローバルタスクリスト　　仮想マシンリスト



「デバッガの設定」図は、各段階の設定事項を示しています。

デバッグプロセス

デバッガにはしばしば矛盾した要求が課されます。シンプルで直感的な操作性が望まれることもあれば、最大限の柔軟性かつフルスクリプト機能が望まれることもあります。まず、直感的な操作性では基本的な考え方は非常に単純です。デバッガがブレークポイントで停止すると、GUIはブレークポイントのアプリケーションプロセスを可視化します。

また、TRACE32のグローバルタスクリストを開くだけで、システム全体の実行タスクすべてがリストされます。GUIで表示したいタスクを選択し、ダブルクリックします。グローバルタスクリストでは特定のタスクにプログラムブレークポイントを簡単に設定することもできます。.elfファイルがロードされる際、デバッグシンボルはマシンIDとスペースIDと関連付けられているため、従来のデバッグと同様に名前で関数と変数のアドレスを指定できます。

さらに、TRACE32のコマンドを使用すれば最大限の柔軟性とフルスクリプト機能が得られます。これらのコマンド用の拡張構文は上記の通りです。

まとめ

ハイパーバイザデバッグは、既に実績のあるOS認識デバッグの考え方を体系的に拡張する形で実現されているので、TRACE32ユーザの皆様は簡単に導入することができます。

現在サポートしているハイパーバイザ

KVM	Wind River Hypervisor 2.x
VxWorks 653 3.x	Xen

(継続追加予定)

Intel® x86/x64－ツールアップデート

今年の 1 月にローターバッハは CombiProbe Whisker MIPI60-Cv2 を新しく発売しました。TRACE32 CombiProbe と TRACE32 QuadProbe は、Intel® Converged MIPI60 コネクタ向けに同一のデバッグ機能を提供します。

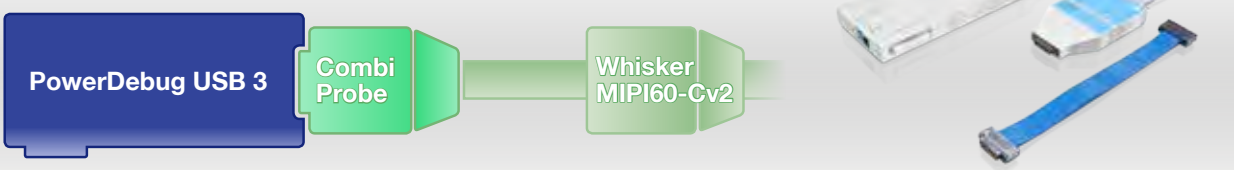
- 標準的なJTAG、PmodeのIntel® debug hooks、およびI2C バス
- 統合デバッグポート（JTAGチェイン 2つ）
- Intel® Survivability機能（閾値、スルーレート等）

この2つのデバッグツールの応用範囲は異なります。TRACE32 QuadProbeはサーバープロセッサ専用に設計されており、最大4個までのデバッグコネクタが装備されたターゲット上で何百個ものスレッドのSMPデバッグを可能にする専用デバッグツールです。

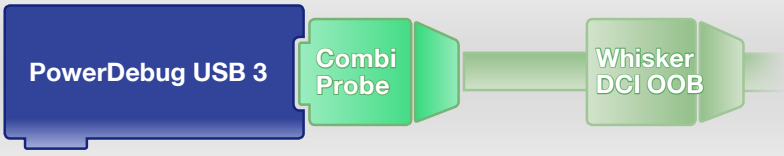
TRACE32 Combi Probeは、クライアントとモバイルプロセッサ用に設計されたMIPI60-Cv2 Whiskerと組み合わせてデバック機能を強化するだけでなく、システムトレースデータをキャプチャ、評価することができます。トレース機能は4ビットと8ビットトレースポートを1個ずつ、通常の帯域幅でサポートしています。

DCI OOB Whiskerと組み合わせたTRACE32 CombiProbeは、デバッグコネクタ装備のないフォームファクタデバイスのデバッグ＆トレース用に特別に設計されました。チップにDCIマネージャが搭載されている場合、ターゲットとデバッガはUSB3インタフェースを介してデバッグ&トレースメッセージを直接交換することができます。メッセージの交換に使用されるDCI規格はシステムトレース情報を記録するトレースメッセージのみでなく、標準的なJTAGとIntel® debug hooksをサポートしています。

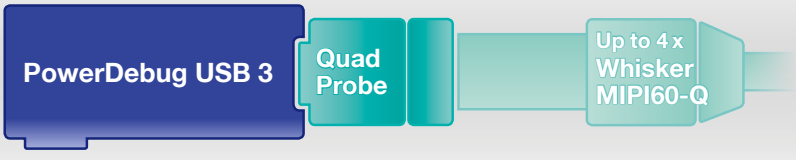
Intel® Converged MIPI60コネクタ用 デバッガ＆システムトレース
(デバイスとクライアントアプリケーション)

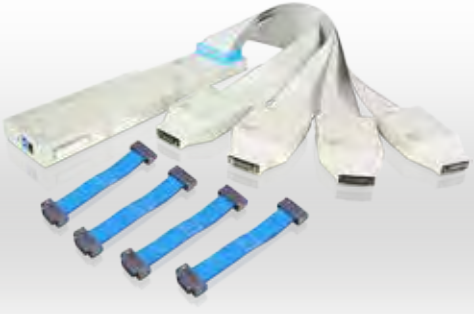


USB3コネクタ用 デバッガ＆システムトレース
(各種アプリケーション)



Intel® Converged MIPI60 コネクタ用 デバッガ
(サーバ)





TRACE32 CombiProbe TriCore DAP

DAPストリーミングとTRACE32ストリーミング

ローターバッハは2016年10月にInfineonのAURIX™ファミリ用にCombiProbe TriCore DAPの提供を開始しました。TRACE32はAGBTインタフェースが装備されていないターゲットハードウェアのAURIXユーザにも包括的なライン タイム解析を提供します。

DAPストリーミング

CombiProbeにはDAPストリーミングという新技術が実装されています。プログラムの実行中にオンチップトレースメモリのコンテンツが読み出され、CombiProbeの128MBトレースメモリにすべて転送されます。このため、チップ側に高速デバッグインタフェースが必要です。AURIX™ DAPインタフェースは、最高160MHzまでのDAP周波数、最高30MB/sまでのデータ転送速度という要件を満たしています。DAP帯域幅は全体のプログラムフローを転送するには不足していますが、それでも広範な解析を実行することができます。

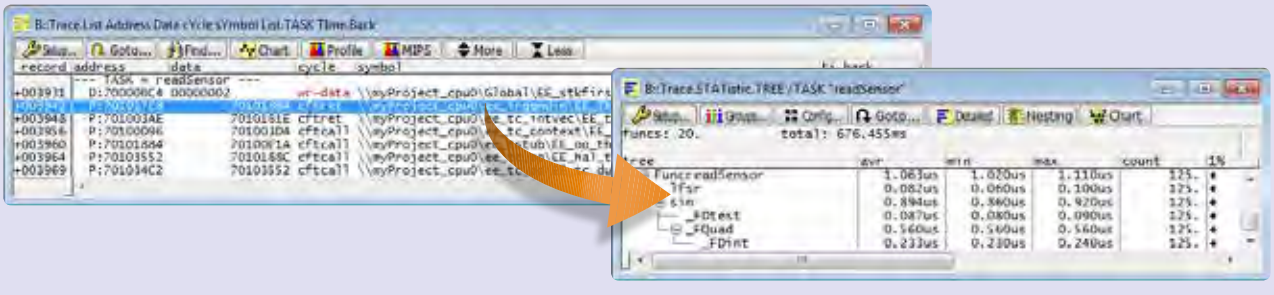
- Compact Function Trace (CFT)を使用したランタイム測定。このトレースは、関数呼び出し(cftcall)/戻り(cftret)にのみトレースデータが生成される特別なプログラムトレースモードです。「Compact Function Trace」図はコールツリーとTRACE32がこのトレースデータに基づいて測定したランタイムの詳細の例を示しています。
- 選択した変数値を時間軸で解析
- タスク、ISR、OSサービスのランタイム測定

TRACE32 ストリーミング

関連するトレースデータの記録にCombiProbeの128MBトレースメモリでは足りない場合、DAPストリーミングをTRACE32ストリーミングと組み合わせることができます。TRACE32ストリーミングでは、CombiProbeがトレースデータを受信するとすぐにホストコンピュータに転送し、ファイルに保存します。長期の測定用にトレースデータを連続的に数テラバイト分も記録することができます。

詳しい情報はホームページをご覧ください。
www.lauterbach.com/cobtricoredap_j.html

Compact Function Trace



ARTI－AUTOSARランタイムインターフェース

自動車業界の現行要件を満たすために、新しいARTI規格がOS認識デバッグ＆トレース向けに特別に策定される予定です。ローターバッハは、AUTOSARオフィシャル開発パートナーとして、この規格の策定に参加しています。2018年初旬に規格の発表が予定されています。

2003年以来自動車業界で使用されてきたORTI規格は、AUTOSARシステムのデバッグ＆プロファイリングにおいて非常に多数の開発者を支援してきました。しかし時代と共に古くなり、新たな要件に対応するために改訂が必要になりました。

目標

新規格にはソフトウェア開発、マルチコア、マルチECUシステムのための新手法、リアルタイムクリティカルシステムの検証おいて高まる要件が含まれます。ARTI規格に追加される新しいデバッグ、トレース、そしてプロファイリング機能の数多くは、独自のソリューションとして既に使用されており、その機能性は実証済みです。しかしながら、後述のとおり、開発プロセスにおいて使用される多様なツール間の標準インタフェースはまだありませんでした。

トレースデータのエクスポート

2014年以来、ローターバッハは車載ソフトウェアの検証、最適化に使用される様々なツールメーカーと密に連携を取ってきました。TRACE32はリアルタイムに記録したトレースデータをエクスポートし、外部のツールにロードして包括的な解析を行います。それでは、何が不足しているのでしょうか。

1. ビルドツールが作成するORTIファイルにはタスク、OSサービス、ISRに関する情報のみ含まれ、タスクの開始/終了場所やランナブルに関する情報は一切含まれません。エクスポートする前に不足している情報を手動でTRACE32に追加する必要があります。
2. トレースデータのエクスポートにおいて標準化されたフォーマットがなく、外部ツールはTRACE32独自のフォーマットを読み込む必要があります。

新しいARTI規格は以上の2点を解決します。ビルドツールが作成するARTIファイルには全てのAUTOSARオブジェクトに関する情報が含まれると共に、トレースデータのエクスポートも標準化されます。

まとめ

主要ツールメーカーとツールユーザが一体となりARTI規格の策定に携わっています。きっとORTIと同様に今後長期にわたり活用される規格が誕生するでしょう。