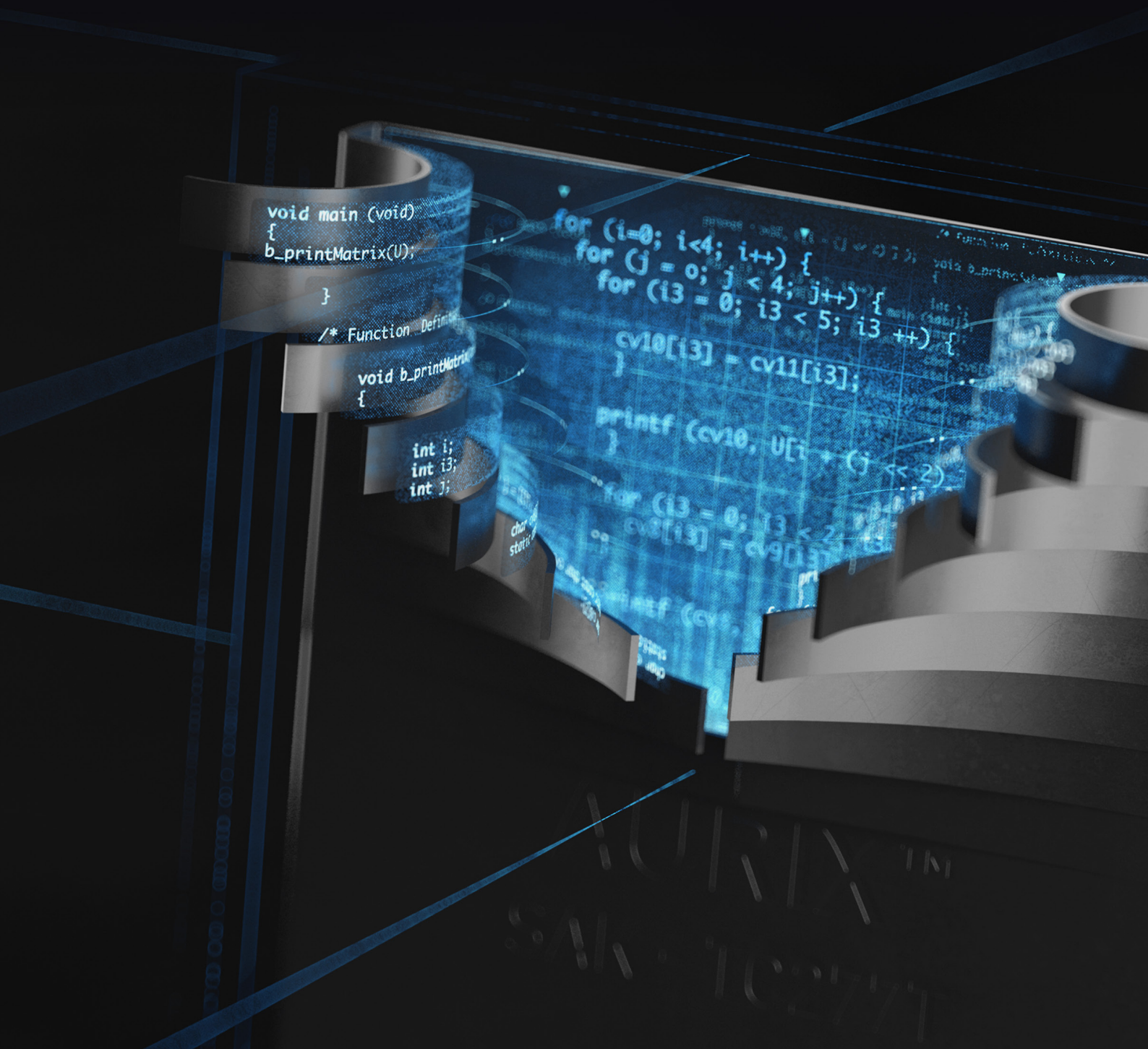
TASKING®

Embedded Debugger
製品概要



エンベデッド・デバッガー - 概要

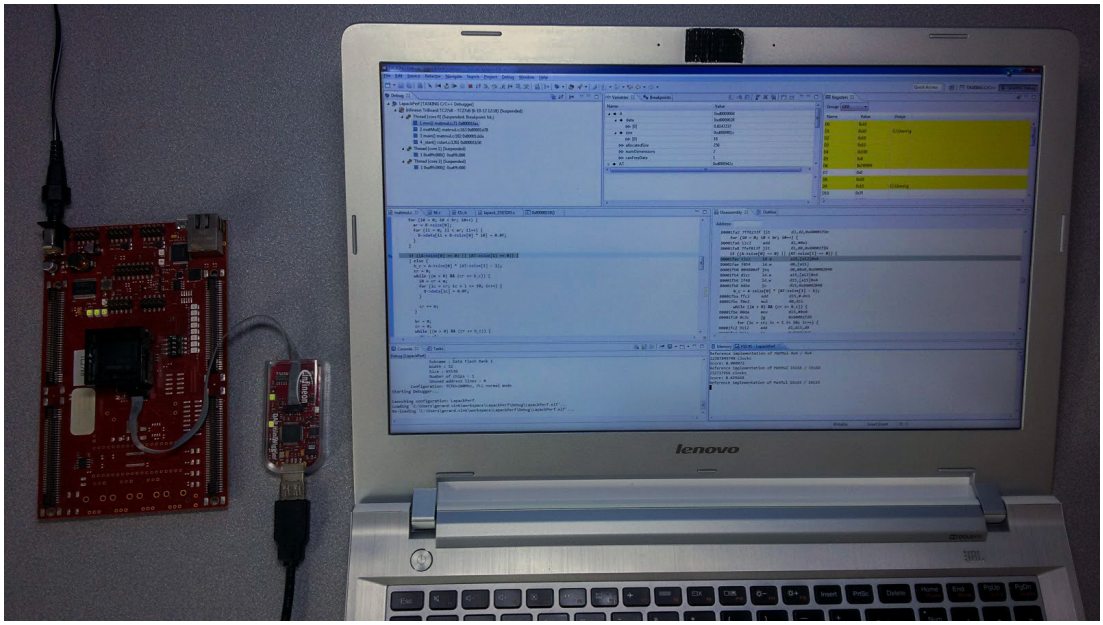
TASKINGは、インフィニオンのマイクロコントローラ TriCore™向けのデバッガーを20年以上提供しています。最近まで、このデバッガーはTASKING TriCore VXツールセットのコンポーネントとしてリリースされ、独立したツールとして利用することはできませんでした。ほとんどのデバッグ作業に対応する機能がこのデバッガーにすべて揃っていると考える多くの顧客からは、コンパイラのツールセットから独立したソフトウェア開発者向けのスタンドアロンのツールとして利用できるようにしてほしい、という要望が寄せられていました。それがきっかけで、最大6つのTriCoreでシングル/マルチコアの合理化されたデバッグのほか、GTM/MCS、SCR、HSM、PCPといった補助コントローラのデバッグを行えるTASKINGエンベデッド・デバッガーが提供されるようになりました。

エンベデッド・デバッガーはEclipseベースの独立したツールですが、既存のEclipse Mars環境ではプラグインとして統合することもできます。ここでは、RAMやFLASHメモリへのプログラムの読み込み、C/C++とアセンブリレベルでのシンボリックデバッグ（高度に最適化されたバイナリを含む）、コードのブレークポイントとデータのウォッチポイント、すべてのコアの同時デバッグ、マルチコア同期の開始/停止、Autosar OSを認識するデバッグ、すべてのデバイス制御レジスタ（SFR）への完全なシンボリックアクセスなど、デバッグのプロセスを合理化する機能が提供されています。さらに、すぐに利用できる評価ボード関連のサポート、プロジェクト設定とタスクの実行に活用できるウィザード、オンラインヘルプやユーザードキュメントを提供しています。

TASKINGエンベデッド・デバッガーでは、インフィニオンの低価格のminiWigglerを使用するDAPプロトコルまたはJTAGからターゲットデバイスへ高速にアクセスできます。詳細なトレーシングや複雑なタイミング分析などの高額のハードウェアプローブが必要になる機能はサポートされていません。物理的なハードウェアが使用できない場合は、パッケージに含まれるインストラクション・セット・シミュレーターに接続することが可能です。

ターゲットデバイスのアクセスとツールの接続性

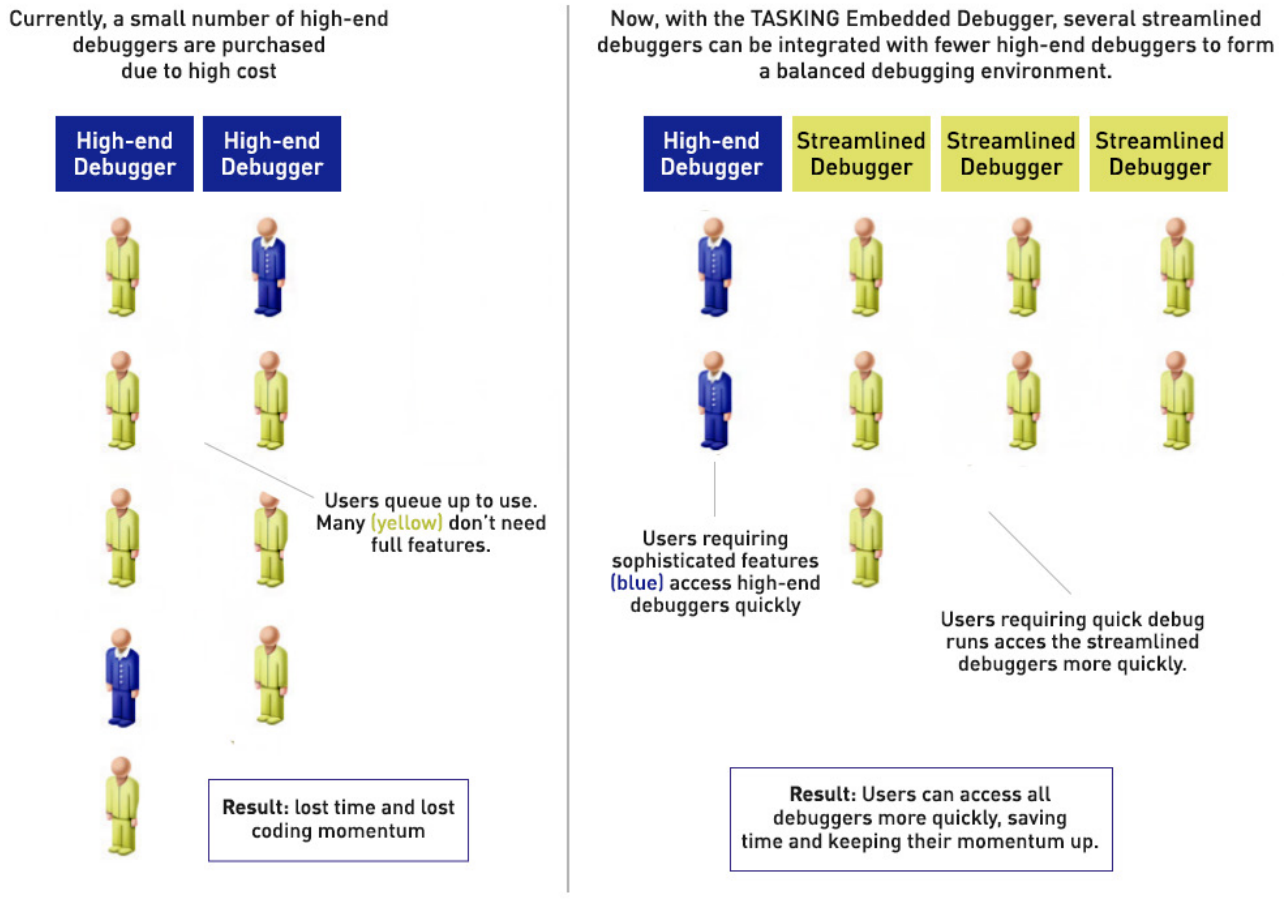
TASKINGエンベデッド・デバッガーは、物理的なターゲットのオンチップデバッグ機能を使用するか、シングルコア・インストラクション・セット・シミュレーター（別途記載）に接続します。



エンベデッド・デバッガー、*DAS miniWiggler*、ターゲットボード

デバッグとプローブ間の通信は、インフィニオンのDevice Access Server（DAS）プロトコルを使って実行されます。ここではDASによって、キャリブレーションツールなどのデバイスへのアクセスが必要な他のツールとデバッガー間での相互運用性が確保されます。また、リモートアクセスも可能なため、オフィスのデスクからラボにあるターゲットのデバイスに接続することもできます。ここでは、複数の開発者がデバイスをタイムシェアリングすることも可能です。

ここで実現するのは、プロジェクトの完了にかかる時間の短縮、コストの節約、スケジュールが遅れるリスクの削減、開発者の非稼働時間の排除です。開発者がデバッグに費やす時間の大半はコードの検証になり、複雑な問題は付随しません。高性能なデバッガーを購入して開発者が共有する場合と、少数の高性能なデバッガーに大半の問題を分析できる安価なデバッガーを併用する方が効率的です。デバッガーのライセンスが使用出来るまで待つことで、開発者の「編集、コンパイル、デバッグ」のサイクルを中断することは、非効率的なだけではありません。そのため、こうした中断があると、一連の考えを思い出すのに時間がかかってしまうことがあります。TASKINGエンベデッド・デバッガーを使用することで、開発コストの削減およびプログラミング作業の効率化を行えます。



全デスクにデバッガーを配置

エンベデッド・デバッガー - 機能と特長

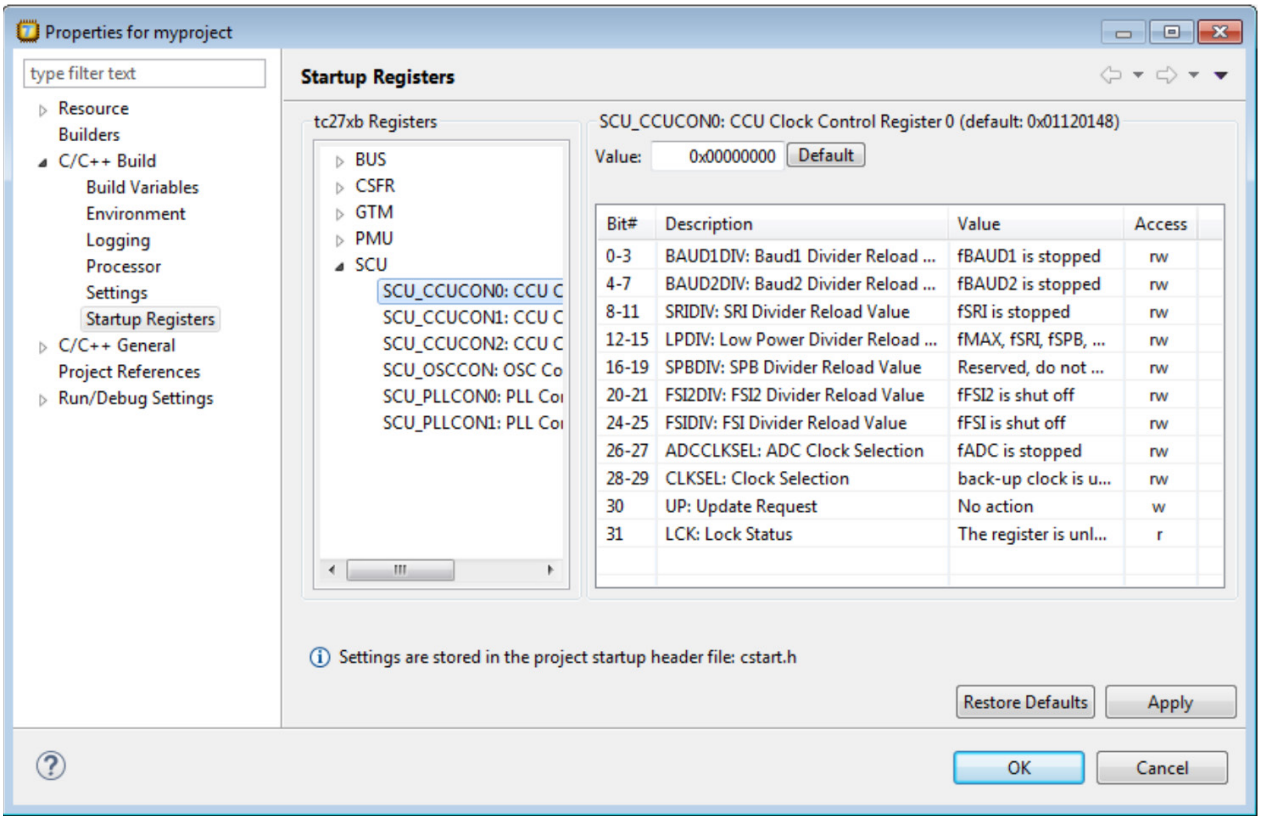
Eclipseベースのユーザーインターフェース

TASKINGエンベデッド・デバッガーは、業界標準のEclipse IDEに統合されています。デフォルトでは、独立したTASKINGのEclipse環境にインストールされるものの、ユーザーのEclipse Mars環境にプラグインとして統合することも可能です。

Eclipseでは、ソフトウェア開発者が馴染みのある標準化された作業環境が提供されるため、新しいツールについて習得する時間を短縮できます。生産性を向上させる機能は、デバッガーにすべて搭載されています。また、デバッガーではウィザードも提供されており、多くの場合に複雑で面倒なターゲットシステムの設定作業が容易になります。インフィニオンと複数のサードパーティーから開発ボード向けに提供されているサポートもすぐに利用できます。さらに、オンラインヘルプやプロジェクトのサンプルなど、詳細なユーザードキュメントも提供されています。

有効なスレッド、プログラムの場所、スタック、変数やレジスターの値など、プログラムを効率的にデバッグ（複数のコアで実行（任意））するために必要なデータはすべて、複数同時に表示されます。スレッドの実行が停止した時点で更新されます。ここでは、パースペクティブと呼ばれるデフォルトのウィンドウレイアウトが提供されています。ただし、特定のニーズに合わせて自由にレイアウトを作成し、Eclipseのパースペクティブとして保存し、後から再利用することもできます。

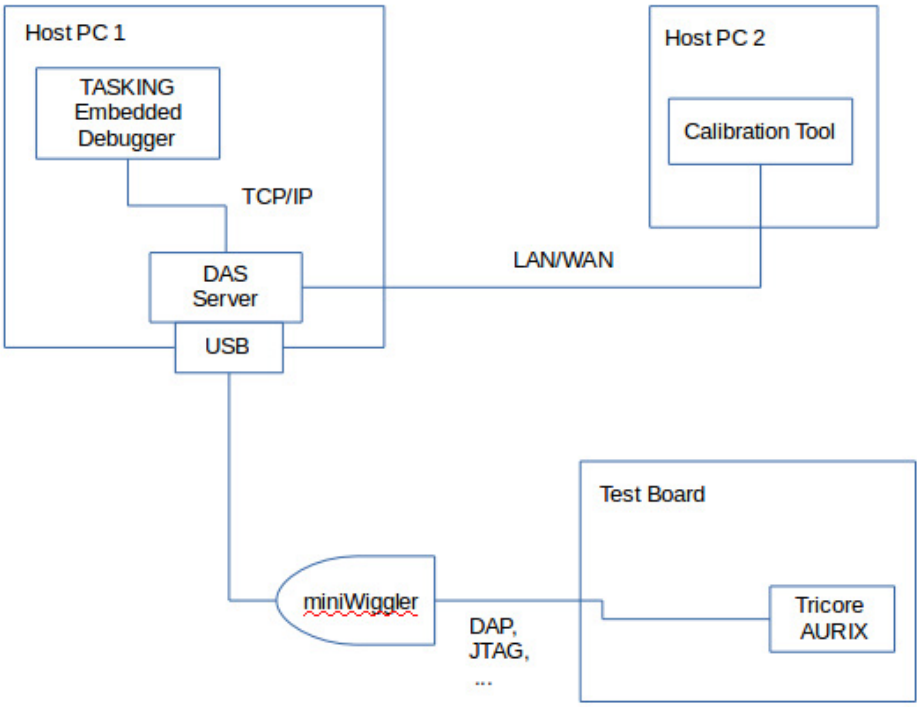
デバッガーのコマンドは、マウスまたはキーボードショートカットキーを使って実行します。操作性を上げるため、通常はショートカットキーを使用することが推奨されます。このデバッガーは、一般的な操作を行うキーボードショートカットキーに対応しています。



デバイスのレジスターの値がシンボル形式で表示される設定ウィザード

物理的なターゲットデバイスには、インフィニオンが提供するminiWigglerを介して接続されます。ここではTriCore、AUDO Future、AUDO MAX、AURIX™、AURIX2G™などのマイクロコントローラーにアクセスできます。またデバイスアクセスポート（DAP2）プロトコルがサポートされているため、既存のJTAGベースの通信チャネルよりも高速でデバッグ通信が行われます。

ターゲットとの論理的な接続は、インフィニオンのDASプロトコルを介して行われます。このプロトコルは、インフィニオンと同社のツールパートナーによって広く利用されている実績のある技術です。ここではDASによって、キャリブレーションツールなどのデバイスへのアクセスが必要な他のツールとデバッガー間での相互運用性が確保されます。さらに、リモートアクセスも可能なため、以下の図に示されるようにオフィスのデスクからラボにあるターゲットデバイスに接続することができます。ここでは、複数の開発者がデバイスをタイムシェアリングすることも可能です。



複数ツールのリモート操作

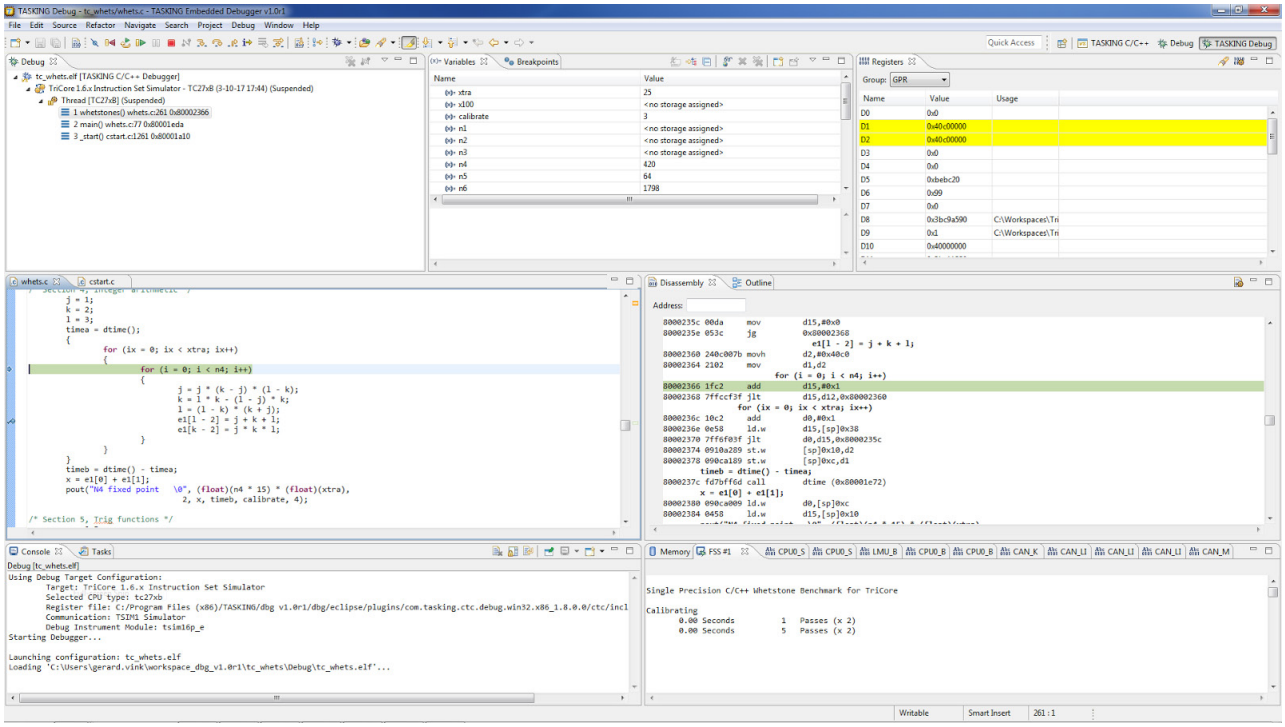
C/C++とアセンブリでのシンボリックデバッグ

デバッグはC/C++言語やアセンブラー、またはこれらを併用して行うことができます。ハイレベルな言語は逆アセンブルと混在して同じウィンドウで表示することも、別のウィンドウで表示することもできます。すべての変数は、関数呼び出しなどの式で表示、変更、使用できます。アドレスは、シンボリック、アブソリュートまたは行番号に基づきます。また、高度に最適化されたC/C++コードのシンボリックデバッグも実行できます。

デフォルトのデバッガーパースペクティブ（2ページを参照）では、下記のシンボルに関する情報を確認できます。

- [Debug] 有効なコア、実行中のスレッド、およびそれらのスタックにアクセスできるため、関数に関連付けられたコードにすばやく移動し、その復帰アドレスにブレークポイントを設定することが可能です。

- [Variables] ローカルおよびグローバルの変数や1つ以上のレジスターに一時的に保存されている変数を確認、修正し、それらが保存されているメモリの場所にデータのブレークポイント（ウォッチポイント）を設定できます。
- [Breakpoints] コードとデータのブレークポイントを確認、修正できます。
- [Register] ターゲットデバイスのレジスターを確認、修正できます。ここにはアドレスや汎用レジスターのほか、特殊機能レジスター（SFR）とも呼ばれるデバイス設定レジスターが表示されます。最後の更新後に変更された値はハイライト表示されます。デバイス設定レジスターのシンボルアクセスがサポートされているため、プロセッサーのユーザーガイドを参照しながらビットパターンを意味を成すものに手動で変換するというエラーの発生しやすい作業が除外されます。
- [Source] 上位のソースコード内の現在の場所が表示されるため、変数の詳細をすばやく確認してソースの行にブレークポイントを設定できます。
- [Disassembly] 分解された（ソースコードと結合された（任意））メモリの画像が表示されます。
- [Memory] ターゲットのメモリの詳細を確認、修正できます。このデバッガーでは、ダウンロードしたプログラムファイルの内容と最新のメモリ画像の差異を一覧化できるため、たとえば自己修正コードも検出することができます。
- [Heap] ヒープの詳細が表示されるため、メモリ割り当ての問題をすばやくデバッグできます。（デフォルトのパースペクティブには表示されません）



デフォルトのデバッグパースペクティブ

RAMとFlashへのプログラムのダウンロード

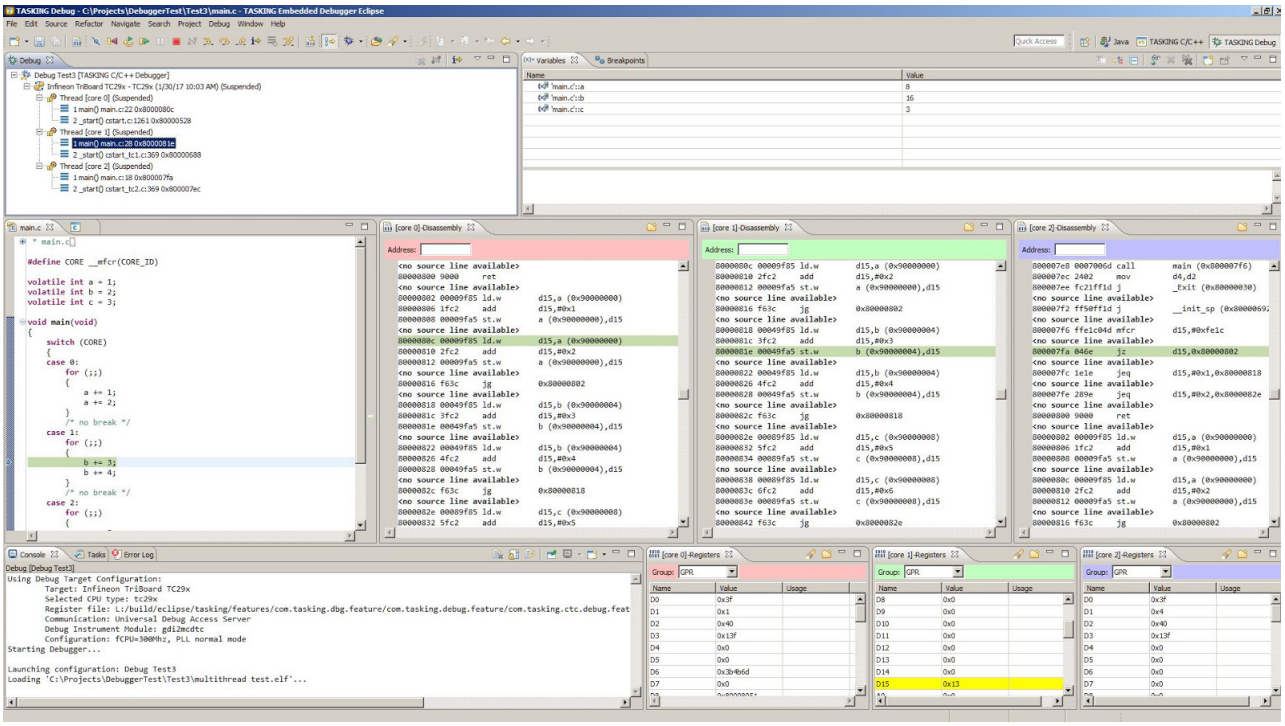
プログラムのコードとデータは、RAMや内部のFlashメモリにダウンロードできます。Flashのプログラミングはデバッガーによって制御されます。ユーザーが求めるように、Flashは高速でプログラムされ、複数のファイルのダウンロードが1回の操作で完了します。

ダウンロードしたコードは、デバイスでコアやコアの組み合わせによって実行できます。もちろん、コードはデバイスの初期化要件にも対応しています。

マルチコアデバッグのGUI

最大6つのTriCoreのマルチコアデバッグがサポートされています。以下の図には、1つのコア/スレッドと、他のコア/スレッドのソースコードと結合された逆アセンブルに関する詳細情報を表示するウィンドウのレイアウトが示されています。

[Debug] システム内のすべてのスレッドのスタックが表示されます。スレッドまたはスレッドの関数を選択するとすべてのウィンドウが自動的に更新され、そのスレッドと実行されるコアの状態が表示されます。次のセクションで解説されるマルチコアの開始/停止機能と組み合わせると、内部コアのインタラクションの原因になるバグを特定して修正する機能を使用できるようになります。



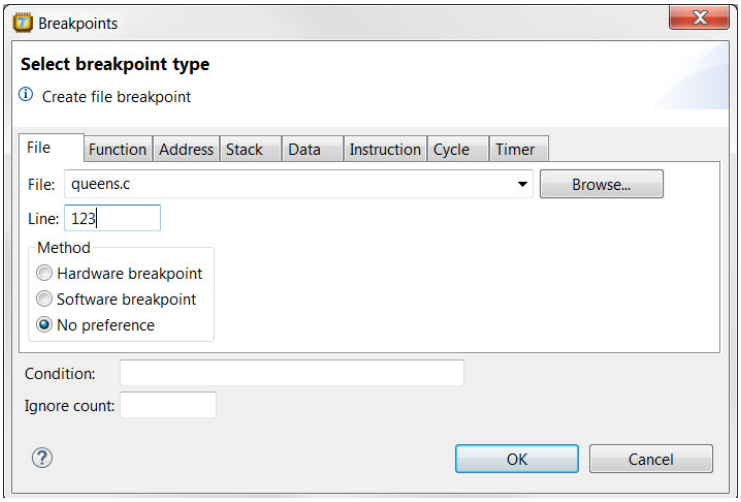
コアの状態（0、1、2）が表示されるマルチコアのパースペクティブ

マルチコア同期の開始/停止を詳細に制御

利用できる詳細な制御機能には、リセットと実行、再起動、再開、C/C++のほか、命令レベルのステッピング、ステップイン、ステップオーバー、現在の関数からの復帰、中断を認識するステッピングなどがあり、ユーザーは制御の実行コマンドをすべてのコアに適用するか、1つの特定のコアのみに適用するかを指定できます。
中断を認識するステッピングは非常に便利で、中断ハンドラーにステップインすることなく、バックグラウンドで中断を処理しながらスレッドをステップスルーできます。

コードのブレークポイントとデータのウォッチポイント

ブレークポイントとウォッチポイントは、プログラム実行の重要な時点でプログラムを停止するために使用します。これによって、実行中のプログラムについての理解を深めることができます。コードのブレークポイントは、所定のアドレスから命令が取得された時点でプログラムを停止させます。データのブレークポイント（ウォッチポイント）は、特定のメモリの場所でデータの読み込みや書き込みが行われる場合にプログラムを停止させます。データのブレークポイントを使用すると、コードを（シングル）ステップスルーすることなく、変数の使用や誤用を簡単に追跡できます。コードやデータのブレークポイントに加え、TASKINGエンベデッド・デバッガーでは経過時間（時刻、CPUのサイクル、実行された命令）に基づくブレークポイントも使用できます。



多くの種類のブレークポイントが表示されるブレークポイント設定ダイアログ

利用できるオンチップブレークポイント/ウォッチポイントの数は、使用するデバイスのリソースによって異なります。RAMに設定できるソフトウェアのブレークポイントの数に制限はありません。ブレークポイントやウォッチポイントに条件を紐づけることで、正しい戻り値を返した場合にのみプログラムを停止することもできます。

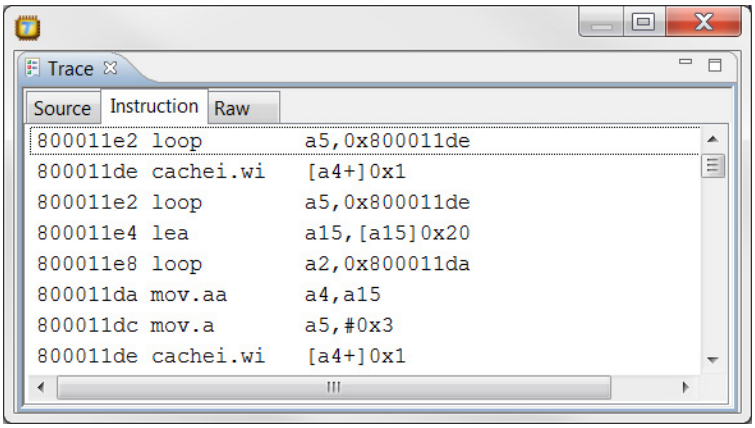
Autosar OSを認識するデバッグ

TASKINGエンベデッド・デバッガーにはリアルタイムオペレーティングシステム（RTOS）を認識するデバッグ機能が搭載され、オペレーティングシステムの内部リソース（タスクリストやメッセージのキューなど）へのシンボリックアクセスが提供されています。OSEK/ISO 17356に準拠するオペレーティングシステムのサポートはすぐに利用可能で、ORTI（OSEKランタイムインターフェース）で特定のOSの実装について設定できます。

ORTIは、OS自体を知らないデバッガーがOSを認識できるようにする仕様です。多くのAUTOSAR/OSEKシステムの構築では、OSのコンポーネントの必要な情報をすべて、デバッガーで読み込み可能な「ORTIファイル」というテキストファイルに抽出することができます。

基本的なトレースのサポート

デバッガーには、最近実行されたCステートメントやマシンの命令を表示する個別のウィンドウが用意されています。ここでは、実行ターゲットデバイスのトレースバッファとともに、コンパイル中に生成されたシンボルの情報が使用されます。これは、AURIXエミュレーターやシミュレーターに対応しています。トレースのウィンドウは、ターゲットが停止するたびに自動的に更新されます。



プログラムの実行が表示される命令トレースのウィンドウ

ファイルシステムの仮想化

ファイルシステムのシミュレーション機能では、ボードからホストシステムにデータを簡単に移動してさらに分析できるため、実際の入力/出力デバイスを入手する前にプログラムをデバッグすることが可能です。
ここではTASKING Cライブラリに組み込まれた機能を使用して、ターゲットで実行され

るすべてのファイルのIO呼び出しがホストシステムにリダイレクトされます。デバッガーでは、キーボードまたはファイルから入力データを読み込んだり、出力をウィンドウやファイルに送信したりして、さまざまな方法でデータを可視化できます。

TASKINGスクリプトデバッガーを使ったテストとデバッグの自動化

パッケージには、Eclipseデバッガーのほかにコマンドラインデバッガーも含まれています。このスクリプトデバッガーはインタラクティブなデバッガーではなく、テストを自動化するためのツールです。リグレッションテストやデバッグの操作は、使用しやすいうえに高度な独自の言語で記述されたユーザー指定のスクリプトを使って順序付けられます。ファイルシステムの仮想化など、上記のすべての機能はスクリプトデバッガーでサポートされています。

シングルコア命令セットシミュレーター

Infineon TriCore、AUDO、AURIXのデバイス（TriCore、Generic Timer Module（GTM/MCS）、Standby Controller（SCR）、Hardware Security Module（HSM）、Peripheral Co-Processor（PCP））にあるすべてのコアでは、パフォーマンスが最適化されたシングルコア・インストラクション・セットシミュレーターを使用できます。

このシミュレーターでは、ターゲットのハードウェアを入手する前にプログラムをデバッグできます。実行コードに加え、このシミュレーターではコードについての解析データや適時の情報が収集されます。これは、ホットスポット分析や命令を使用するための基本材料となり、異なるバージョンのアルゴリズム間での相対的パフォーマンスの測定が容易になります。さらに、トレース機能もサポートされています。

製品仕様

サポート対象のホストシステム

- Windows 7以上

サポート対象のTriCoreデバイス

- TC11シリーズ（TC1130、TC1164、TC1166、TC1167、TC1197）
- TC173シリーズ（TC1736）
- TC176シリーズ（TC1762、TC1766、TC1767）
- TC178シリーズ（TC1782、TC1784）
- TC179シリーズ（TC1791、TC1792、TC1793、TC1796、TC1797、TC1798）
- AURIX TC2シリーズ（TC21X、TC22X、TC23X、TC26X、TC27X、TC29X）
- AURIX TC3シリーズ

サポート対象のデバッグプローブとターゲットコネクタ

- インフィニオンのminiWiggler
- 10ピンDAPの自動車用20ピンJTAG

サポート対象のシングルコア命令セットシミュレーター

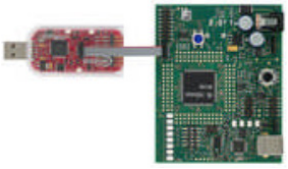
- TriCore、GTM/MCS、SCR、HSM、PCP（すべてパッケージに含まれる）

EMBEDDED DEBUGGER

概要

TASKINGエンベデッド・デバッガーは、大規模な開発チームによるコード検証のためのコスト効率に優れた完全ソリューションです。わずかな時間しか必要にならない高機能のデバッガーを1つ入手するのであれば、それと同じ金額で合理化されたデバッガーを複数購入したほうがよいでしょう。そうすることで、開発者は早い段階で論理的な問題を特定し、コーディングエラーを修正できるようになります。

参考文献



[1] DAS（Device Access Server）：www.infineon.com/DASから利用可能

[2] MCD（マルチコアデバッグ）API: www.infineon.com/DASから利用可能

[3] DAP miniWiggler: www.infineon.com/DASから利用可能