バグとの正しい付き合い方

組込みの特性を知って、楽なデバッグ生活

Appendix 2

❶ Windows CEのハードウェアデバッガ

小林 章　KOBAYASHI Akira　京都マイクロコンピュータ㈱　http://www.kmckk.co.jp/
ロボット、無線通信、組込みJava、デバッガと無節操にフィールドを変えつつ、ファームウェアからWindowsのUI、はたまたプロジェクトマネージャまでこなす便利屋エンジニア。

はじめに

Windows Embedded CEのデバッグは、通常のWindowsのアプリケーションを作るときと同様にMicrosoft社の統合開発環境である「Visual Studio」を使うのが一般的です。Visual Studioは基本的に「ソフトウェアデバッガ」と呼ばれ、OSにあらかじめ組み込まれたデバッグ機能とVisual Studioが通信することにより、デバッグを実現しています。

ソフトウェアデバッガの制限

このようなソフトウェアデバッガは特別なハードウェアを用意しなくてよい反面、次の制限が存在します。

- デバッグできない領域がある
  OSのデバッグ機能を使っているため、OSが動いていないブート部分や、OSの機能がほとんど使用できない割込み部分などは、デバッグが不可能、または制限されます。
- パフォーマンス面での低下が発生する
  OSのデバッグ機能が動くことにより、通常の動作とは別に追加の処理が実行されるため、全体の動作が遅くなる傾向があります。また、デバッグ機能を追加でターゲットに組み込む必要があるので、その分のメモリ領域（ROM、RAM）を消費します。
- デバッガとデバッグ対象が離れている場合、なんらかの通信路が必要

組込みでのデバッグはリモートデバッグが一般的です。デバッガとデバッグ対象は別のOS上で動作し、デバッガとデバッグ対象は通常USBやLANなどで接続されます。このためデバッグ時にはUSBドライバやLANドライバが正しく動作する必要があり、またデバッグに使用しているドライバはデバッグ対象にできません。

ハードウェアデバッガの特長

ソフトウェアデバッガの制限を回避するためには、「ハードウェアデバッガ」が有効です。組込み分野では「ICE」（In-Circuit Emulator）と呼ばれるハードウェアを使って、OSの機能を使わずに直接CPUを制御することでデバッグを行います。これにより次のように制限が軽減されます。

- どこでもデバッグ可能
  CPUを直接制御するため、ブートローダや割り込みの最下層でもブレーク処理、ステップ動作などが可能となります。
- デバッグ中のパフォーマンス低下がない
  OSのデバッグ機能を使用しないため、デバッグを行っていない部分は通常時と同じ動作となり、追加の処理が発生しません。
- 特別なインタフェースを使用し、追加のドライバが不要
  現在の組込み用のチップには「JTAG」と呼ばれるインタフェースが存在し、ハードウェアデバッグ時にはこのインタフェースを使って通信を行います。これはチップ内でハードウェア的

に組み込まれているもので、OSのドライバなどを使用しません。

しかし、従来のハードウェアデバッガでは、Visual Studioの出力するデバッグ情報に対応できず、またOSで管理される多重化された仮想メモリ空間を扱うことができませんでした。このためハードウェアデバッガを使うために、開発者は逆アセンブルされた出力を見ながら、かつ多重化されていないカーネル空間のみをデバッグしていました（表1）。

本稿ではWindows Embedded CEの最新デバッグ環境として、ハードウェアデバッガでありながら、C言語でのソースコードデバッグが可能、かつアプリケーション空間のデバッグも可能なPARTNER-Jetを使ったデバッグについて紹介します。

Windows Embedded CEの開発フェーズ

デバッグについて説明する前に、Windows Embedded CEでの開発フェーズについて触れておきましょう。Windows Embedded CEを搭載したデバイスを開発する場合、一般的に図1の手順で開発が行われます。

■ハード設計、開発、デバッグ

組込みの現場では、用途に合わせてハードウェアを新規に設計することは珍しくありません。また小規模の開発では、組込み用のボードなどと組み合わせて開発することもあります。たいていの場合、ハードウェア開発とソフトウェア開発はハードウェアがやや先行する形で並行して行われます。したがって、ソフトウェア開発中はハードウェアもデバッグ中であるため、ハード開発者へのフィードバックを行います。

■OS設計

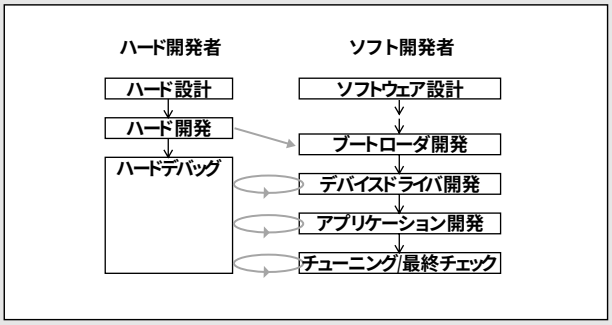
ソフトウェアを設計します。Windows Embedded CEを使った開発では、必要な機能をVisual Studioのカタログから選択することから始まります。設計フェーズですので基本的にデバッグは机上のチェックがメインです。

▼表1　デバッガの比較

対象	Visual Studio	一般ICE	PARTNER-Jet
ブートローダ	×	△*2	○
デバイスドライバ	△*1	△*2	○
アプリケーション	○	×	○

※*1：割込み部分などは不可、*2：ソースコードデバッグ不可

▼図1　開発フロー（概略）



バグとの正しい付き合い方

組込みの特性を知って、楽なデバッグ生活

■ブートローダ開発

ハードウェアの開発後、一番最初に必要となるのはブートローダです。Windows Embedded CEの場合、開発環境である「Platform Builder」を使って専用のブートローダ「EBOOT」を利用するのが一般的です。デバッグもEBOOTのデバッグがメインとなります。

■デバイスドライバ開発

ブートローダが実行され、OSが起動するようになると、今度はターゲットにある各デバイスのドライバを開発します。ドライバはチップベンダが提供するBSP（Board Support Package）やWindows Embedded CE用ドライバをベースに作成しますが、特殊なデバイスの場合は一から独自に作成することもあります。これらのデバッグはPlatform BuilderとVisual Studioを組み合わせて行います。また、デバイスドライバの検証には、CETK（Windows Embedded CEテストキット）を使います。

■アプリケーション開発

ドライバが動き出したら、実際のアプリケーションの開発を行います。Windows Embedded CEでは、通常の（組込みでない）Windowsと同じようにVisual Studioを使用してデバッグを行います。

■チューニング／最終チェック

ほぼ全体のシステムが動き出した時点で、パフォーマンスのチューニング（この場合のパフォーマンスは速度だけでなく、メモリ使用量なども行われます）や、全体を通したシステムテストが行われます。

ブートローダのデバッグ

Windows Embedded CEの標準ブートローダであるEBOOTは、初期動作部分がアセンブラで、残りはほぼC言語で記述されています。従来、このブートローダ部分は、前述したとおりVisual Studioではデバッグが難しい部分であり、開発者はデバッグ出力のみを頼りにトライ＆エラーを繰り返したり、ICEを使ってコンパイラから生成されたアセンブラ出力を眺めながら作業をしていました。

図2はPARTNER-Jetのデバッグ画面です。ハードウェアデバッガでありながら、デバッグ画面はVisual Studioと同様のものが表示されます。

❶モジュールウィンドウ

デバッグ用として読み込まれたモジュール（EXE、DLL）のソースおよび関数を表示します。表示はディレクトリによるツリー形式にすることもできます。

❷コードウィンドウ

デバッグ中のソースコードを表示します。逆アセンブルとソースコードのミックス画面も表示できます。

❹メモリウィンドウ

ターゲット上のメモリを表示するウィンドウです。複数のエリアのメモリを表示／変更できます。

❺ウォッチウィンドウ

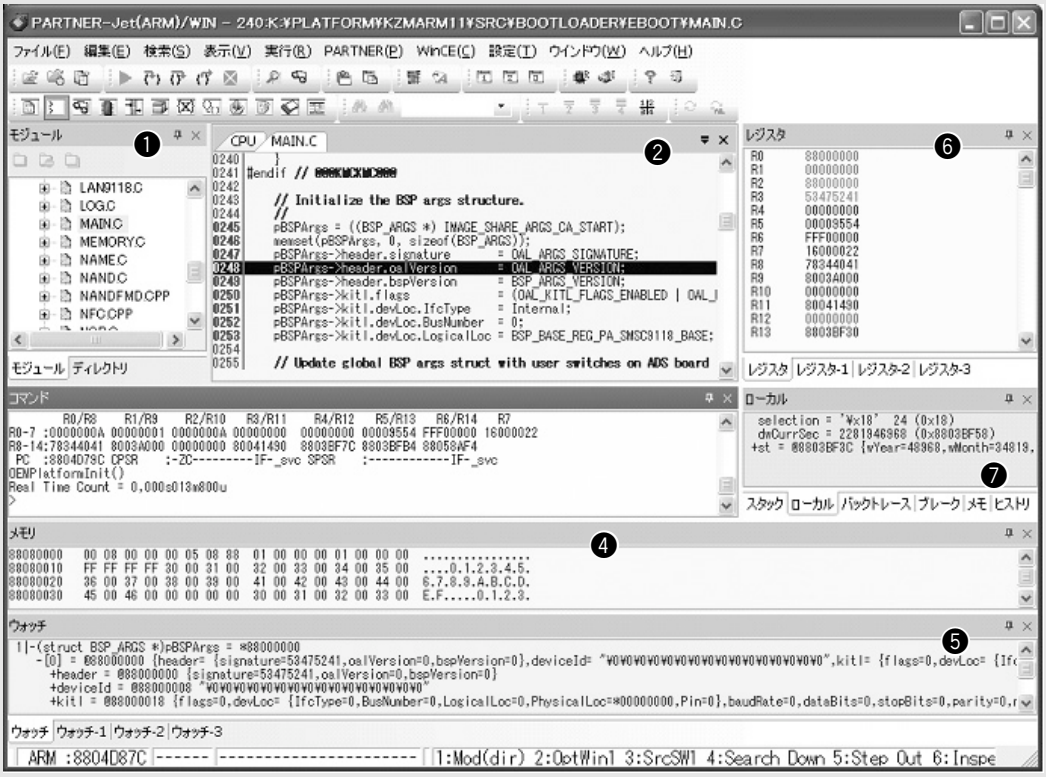
デバッグ中に監視したい変数などを登録すると、停止した時点の変数の内容を表示します。

❻レジスタウィンドウ

CPUのレジスタの状態を表示します。変化した部分は色が変わるなど、細かいレベルでの動作確認に有効[注1]です。

注1）コンパイラで最適化を行うと、変数などが使いまわしをされたり、レジスタに割りつけられて、参照ができないときがあります。このような場合は、逆アセンブル画面を見たり、ステップ実行などをしてレジスタの変化点を見るのが有効です。

▼図2　PARTNER-Jetのデバッグ画面



❼各種オプションウィンドウ

Visual Studioのデバッグ画面と同様に、スタック表示、ローカル変数の表示、コールスタック(バックトレース)、ブレークポイントなどのオプションウィンドウが存在します。また、ハードウェアデバッガは、CPUを直接操作できる利点を活かし、CPUに搭載されたデバッグやプロファイル支援の機能も使用できます。その1つのヒストリウィンドウは、CPUの実行履歴を表示するものです。これにより、プログラムがどの部分を通ってきたかをアセンブラ、およびCのソースレベルで辿ることができます。

デバイスドライバのデバッグ

デバイスドライバは、ほとんどがC言語で記述され、特定プロセスからロードされるDLLの形式をとります。従来のデバッグ方法では割込み処理の中はデバッグが非常に困難でしたが、ハードウェアデバッガでは任意の個所で処理を止め、CPUの状態やメモリの状態を確認できます。

デバイスドライバにありがちなミスとして、バッファのオーバフローによるメモリ破壊などがあります。ハードウェアデバッガはこのような特定エリアのメモリアクセスに対してハードウェアブレーク[注2]を設定することにより、メモリ破壊をし

注2）CPUや環境に依存します。

たコードを直接特定できます。

アプリケーションのデバッグ

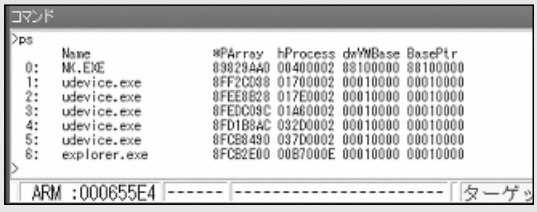
アプリケーションは、CやC++言語、ときには.NET Compact Frameworkを搭載してC#やVisual Basicで記述され、EXEとして実行されます。ハードウェアデバッガでは、ユーザプロセス空間のような多重化された空間のデバッグは困難と言われてきました。これは、ハードウェアデバッガが直接知り得る情報はCPUレベルのものであり、OSが管理する情報とのすり合わせが難しいことからきていました。

これを、ハードウェアデバッガが図3のようにOSが管理する情報を直接アクセスすることにより、多重化された仮想メモリ空間を意識することなく、ソースレベルでのデバッグを実現しています[注3]。

チューニング／最終チェック時のデバッグ

Windows Embedded CEでは、OSの環境全体を製品用のリリースビルドと、デバッグ用のデバッグビルドの2種類でビルドできます。チューニングや最終チェックのときには、全体をデバッグビルドではなく、リリースビルドを使用するのが通常です。

▼図3　デバッガがOS情報を直接アクセス



しかしリリースビルドでは、KITL（Kernel Independent Transport Layer）やカーネルデバッガなどのデバッグ用の機能が組み込まれない形でビルドされますので、デバイスドライバなどのデバッグはログ出力などに頼った形で行われてきました。

ハードウェアデバッガは、OSのデバッグ機能を使用しませんので、リリースビルドであっても同様にデバッグができます。また、ハードウェアデバッガは、CPUに搭載されたパフォーマンス計測機能[注4]を使って、実時間でのプロファイリングができます。

まとめ

駆け足でしたがハードウェアデバッガを使ったWindows Embedded CEのデバッグについて紹介しました。もっと詳しく知りたいという方は次の情報を参照してください。

■PARTNER-Jetに関する話
http://www.kmckk.co.jp/windows_ce/

■Windows Embedded CEのビルドやデバッグに関する話
『Windows Embedded CE 6.0組み込みOS構築技法入門』、松岡 正人 監修、伊藤 優／岩崎 平／江島 午郎／大場 孝仁／Kasar Mahesh／杉本 拓也／高根 英哉／田靡 哲也／中山 宏之／松井 俊訓／好井 智章 著、日経BPソフトプレス 刊、ISBN:978-4-89100-566-5

注3）マネージコード（C#およびVisual Basic）は非サポートです。
注4）CPUに依存します。