PARTNER-JetとEclipseによるT-Engineソフトウェア開発

辻 邦彦（つじ くにひこ）
京都マイクロコンピュータ株式会社

図1　PARTNER-Jet（接続されているのはTeaboard）

T-Engine開発キットには、それを購入するだけでT-Engineボード（ハードウェア）だけでなく、GNUベースのコンパイラ、リンカ、デバッガなどソフトウェア開発に必要なツールがひととおり付属してきます。付属するのはCUI（Character User Interface）のコマンドラインコンパイラ、コマンドベースのソースデバッガGDBであり、近頃のGUI（Graphical User Interface）ベースの開発環境に比べるともの足りないものがあります。また組込み開発ではデバッグのためにJTAG-ICEを用いて高度なデバッグを行うことが多いのですが、そういったこともできません。

京都マイクロコンピュータ（KMC）のJTAG-ICE、PARTNER-JetはT-Engineソフトウェアのデバッグにも対応しており、またオープンソースの統合開発環境Eclipse[注1)]にも対応しています。ここでは、PARTNER-Jet／Eclipseを用いたT-Engineソフトウェアの開発、デバッグについて解説します。なお、PARTNER-JetのEclipse対応版の正式リリースは2006年春の予定で、本稿はベータ版を用いて解説を行っています。製品出荷時には一部の変更の可能性もあることをご了承ください[注2)]。

PARTNER-JetのEclipse対応

●Eclipse

Eclipseはもともとは IBMが開発していたもので、当初はJavaの統合開発環境として提供されていました。Eclipseそのものも Javaで開発されており、使いやすいGUI、Plug-inによる拡張などさまざまな特徴があります。また近年IBMからオープンソースでの開発に移行し、このオープンソース化も多くの支持を集めている大きな要因と思われます。EclipseはJavaの統合開発環境として強力なのはもちろんなのですが、プラグインでの拡張でC/C++の統合開発環境の機能も提供されています。余談ですが、数百ものプラグインが存在し、中にはインターネットに接続し株価チャートを表示する、というようなものまで存在します。

C/C++言語の開発には、Eclipse本体のほかにC/C++ Development Toolkit（以後CDT）というプラグインをインストールする必要があります。

●PARTNER-Jet

京都マイクロコンピュータが開発、販売するPARTNER-Jetは、JTAGを利用した高性能、高機能なエミュレータです。従来の普通の組込みソフトウェアのデバッグだけでなく、MMUを用いた大規模なシステムのデバッグ（T-EngineやLinuxなど）や、またマルチコアCPUのデバッグなどを特徴としています。また統合開発環境Eclipseにもいち早く対応し、使い勝手の部分でも進化しています。

●標準Eclipse/CDTと、PARTNER-JetのEclipse/CDT

標準配布されているEclipse/CDTではあくまで統合開発環境のGUIが配布されているだけで、これにコンパイラやリンカ、またデバッガは含まれていません。プロジェクト管理の画面や、エディタ、またビルダやデバッガの画面などが含まれます。デバッグするにはほかにデバッガが、ビルドするにはほかにmakeやantなどのツール、そしてコンパイラが必要になります。これに関し

注1） http://www.eclipse.org/
注2） 変更内容、本原稿の出荷バージョンへの対応は、当社WEBページ（http://www.kmckk.co.jp）に掲載する予定です。
注3）http://www.cygwin.com/
注4）http://www.t-engine4u.com/products/tbarm920mx1.html

ては私どもが対応したPARTNER-Jet用のEclipse/CDTでも同じです。

標準のEclipse/CDTでは、付属していないコンパイラやデバッガについては、gccやgdbなどのGNUのツールチェインが利用されることが想定されているようです。特にデバッガに関してはEclipse/CDTがGDB/MIというgdbが提供するプロトコルで呼び出しており、簡単には他のデバッガは使えません（図2）。

CDTとgdbとの間では、メモリやレジスタの読み書き、ブレークポイントの設定／解除やプログラムの実行制御などデバッガとして動作に関する制御が行われるだけでなく、デバッグ対象プログラムに関するデバッグ情報の管理などもやりとりしています（アドレス0x00001234番地はソースファイルfoo.cの33行目である、などの情報をCDTがgdbに問い合わせし、gdbがCDTに返します）。CDTのデバッガ画面とデバッガとの間のインタフェースは新たにプラグインを開発して別のデバッガ（PARTNERなど）に対応させるということも考えられますが、私どもは可能な限りEclipse/CDTには手を加えずに対応を行うために、今回新たにPARTNERにコマンドラインモードとGDB/MIプロトコルを実装しました。Eclipse/CDTはオープンソースの世界で日々進化拡張していき、それを利用者が享受できるためにはEclipse/CDT側を変更するのでなく、PARTNERにGDB/MIを実装するほうが良いと判断しました。

上記のような実装を行った結果、Eclipse/CDT＋PARTNERではデバッグ対象としてはPARTNERが認識できるオブジェクトフォーマットが扱えることになります。gccなどのGNUツールチェインで生成されたものだけでなく、ARM純正コンパイラやルネサス純正コンパイラなどもEclipseからデバッグすることが可能になっています。またgdbが持っていないICE固有機能（リアルタイムトレースやプロファイル、ハードウェアブレークなど）は、PARTNERのコマンドもEclipseのデバッガ画面から通るようにできていますので、それらも利用可能です。コマンドだけでなく、GUIからも操作が可能になっています。

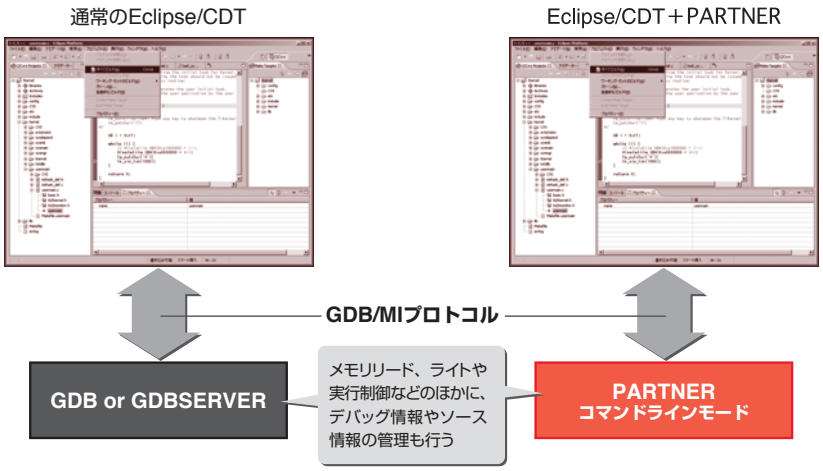

図2　EclipseとPARTNERの関係

T-Engine開発キットとEclipse/CDT（付属のサンプルをEclipseでビルドする！）

今のT-Engine開発キットのビルド環境をEclipse/CDTの統合開発環境に移行するのは非常に簡単です。必要なものは、

・Cygwin[注3]：ver1.5以降
・T-Engine GNU開発環境（Windows版）：T-Engine開発キットに付属
・Eclipse SDK、win32/x86：ver3.1以降
・CDT、win32/x86：ver3.0以降

になります。原稿執筆時点での最新バージョンはCygwin ver1.5.19-4、Eclipse SDK ver3.1.2、CDT ver3.0.2となっています。上記のほか、Eclipse統合開発環境を日本語化するためのLanguage Packesなどがあります。必要に応じてインストールします。本稿執筆にあたっては、Cygwin ver1.5.18、Eclipse SDK ver3.1.1、CDT ver3.0.1を用いています。今回はTeaboard/ARM920-MX1[注4]をターゲットのT-Engineとして説明をしますが、他のボードでもおおむね同じです。まずはこれらを正しくインストールしてください。環境変数などが正しく設定されているかを確認するために、付属のT-Engine開発キットのサンプルをCygwinコンソールでmakeしておくことをお勧めします。

●Eclipse環境にサンプルを登録

T-Engine開発キット付属のサンプルプログラムfileio（プロセスベースアプ

リケーション。デフォルトは/usr/local/te/bapp l /fileio にインストールされます）をEclipseでビルドしてみましょう。Eclipseで開発するときには、プロジェクトという単位で開発し、そのプロジェクトはワークスペースというディレクトリに保管されます。ワークスペースのディレクトリは最初のEclipse起動時にたずねられるので、適時指定してください。起動したらメニューから［ファイル］→［新規］→［プロジェクト］→［Standard C Make Project］を選択してください。プロジェクトを新規作成するダイアログが出るので、プロジェクト名のところにfileioと入力して［終了］ボタンでダイアログを閉じます。ダイアログを閉じると、C/C++ Projectウィンドウの中にfileioという新しいプロジェクトが生成されます。この状態ではソースファイルも何もない空のプロジェクトです。T-Engine開発環境に付属するサンプルプログラムのファイルは、インポートという方法で取り込みます。fileioプロジェクトを選択し、［ファイル］→［インポート］を選択するとダイアログが出ます。ここで「ファイル・システム」を選択し、「次へ」を押下します。インポートの詳細を設定するディレクトリが出てくるので、ソースディレクトリにサンプルプログラムfileioが存在するディレクトリ（C:¥cygwin¥usr¥local¥te¥bappl¥fileio）を入力します。そして、その下のツリービューに登場するfileioの前のチェックボックスをチェックし、［終了］ボタンを押下します。これでサンプルfileioのすべてのファイルが取り込まれます（図3）。

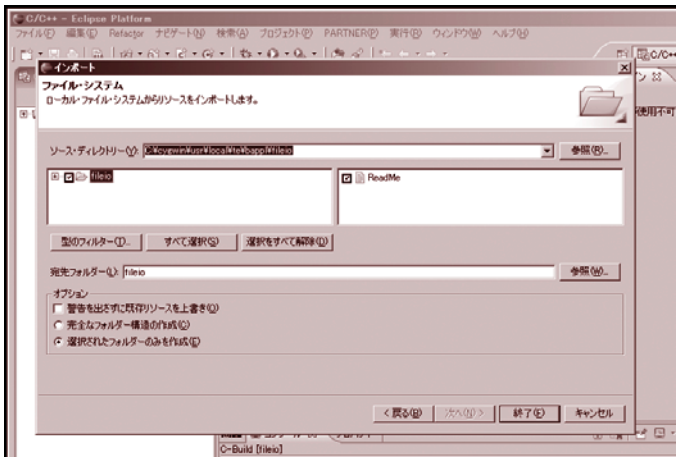

図3　インポートの設定

これで必要なすべては基本的にそろっているのですが、このままではビルドでエラーが発生します。そのために以下のことを行います。

・src¥Makefileをビルドするディレクトリ（tbmx1¥など）にコピーする

これはCygwin環境上で各ビルドディレクトリのMakefileがシンボリックリンクで ./../src/Makefileを指していることに関する問題の対策です。Eclipseからは普通のWindowsのファイルI/Oが用いられるため、Cygwinのシンボリックリンクは認識されません。そのために、この問題は回避するためにMakefileの実体をコピーします。

・c:¥cygwin¥usr¥local¥te¥bappl¥etcをワークスペースにコピーする

上記Makefileの中では ../../etc/makerulesがincludeされています。このetcディレクトリはインポートしたfileioディレクトリと同じレベルにあるためインポートされていなく、このままではビルド時にエラーが発生します。仮にワークスペースがc:¥work¥workspace¥であるならば、プロジェクトはc:¥work¥workspace¥fileio¥で、c:¥work¥workspace¥etc¥となるようにディレクトリの中身を含めてコピーします。

●Eclipseでビルド

プロジェクトを生成した状態でのEclipseでのビルドはmakeを呼び出すことで行っています。makeを単純に呼び出すだけでは足りない場合のことも考慮して、ビルドを行うときなどの設定が多く行うことができるようになっています。ただし、普通に作られたmakeであれば（make clean, make allなどでビルドができるmakefile）、大きく変更しなくても使えるようになっています。今回のT-Engineのプログラムをビルドするためには、2つの設定を追加で行います。設定はプロジェクトfileioを選択し、「プロパティ」ダイアログの「C/C++ Make Project」画面で行います。

●環境変数の登録

普通のWindowsアプリケーションであるEclipseから、Cygwin上でのgnu makeを利用することを前提としたMakefileをそのまま用いるために、EclipseからCygwinの環境が動作するように環境変数を設定します。「C/C++ Make Project」画面の「Environment」を表示し、「New」から

Name：PATH

Value：c:¥cygwin¥bin

を設定します。

●ビルドディレクトリの設定

makeを実行するディレクトリを指定します。「C/C++ Make Project」画面の「Make Builder」の「Build Directory」に指定します。指定するときに［Browse］ボタンを押下すると、プロジェクトのトップディレクトリが出てきます。T-Engineのサンプルは、その下にある各ターゲット用のディレクトリでmakeしますので、そのディレクトリ（tbmx1¥やsh7727¥）を選択します（図4）。

●ビルドする！

これで設定はすべて完了です。メニューの［プロジェクト］→［プロジェクトのビルド］を選択すると、自動的にビルドが行われます（図5）。今までの設定が間違っていなければ、問題なくビルドが完了するはずです。ここまでの設定だけで、統合開発環境としての機能は有効になっており、ビルドでワーニングやエラーが出た場合にはその内容とソースコードが連携し、修正が容易にできるようになっています。

これでT-Engine付属のサンプルはEclipse上で作成できるようになりました。今まではエディタでソースを作成し、コマンドラインでmakeを行い、エラーが発生したlogファイルなどからタグジャンプをして、などとしていた部分が、この開発環境の中だけで完結して行えます。

Eclipseは自分自身でビルドを行わないで、makeなどを呼び出す形式、しかも呼び出し方法が強力にカスタマイズして呼び出すことが特徴で、他の今までの統合開発環境の独自のビルドのしくみとは大きく異なります。そのために、T-Engineのサンプルなど既存のソフトウェアビルド環境も簡単に利用することが可能になっています。サンプルではなく、すでにソフトウェアを開発されている方でもmakeでビルドしていたなら簡単な設定でeclipseに取り込むことが可能と思われます。

さて、ここまででも便利なのですが、これだけではデバッグができません。次からはデバッグについて説明しますが、ここからの機能を試すにはフリーでダウンロードできるEclipseだけでなく、PARTNER-Jetが必要になります。

PARTNER-JetでT-Engineソフトウェアをデバッグする

●T-EngineとPARTNER-Jet

Eclipseでのデバッグの前に、T-EngineとICEの関係を整理します。

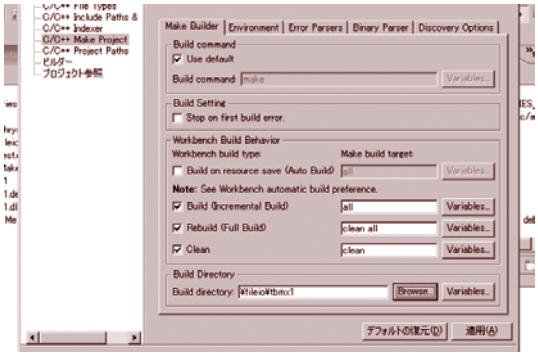

図4　ビルドの設定

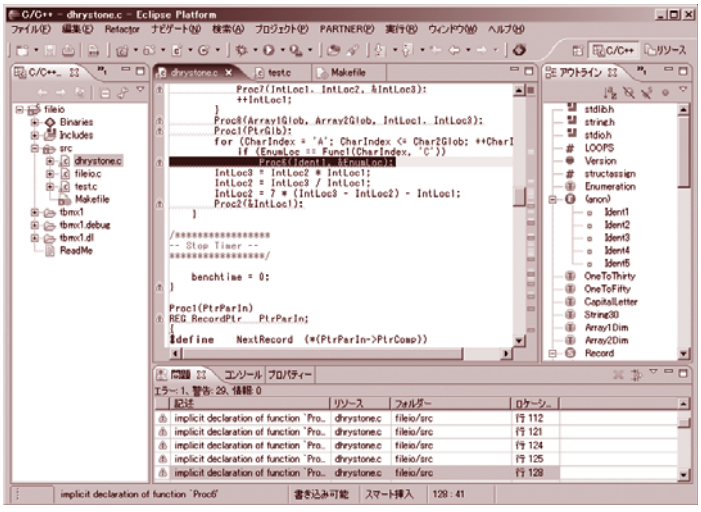

図5　Eclipseでビルドした様子（エディタ画面の波線はコンパイルで警告が出ている箇所）

注5、6）T-Engineデバッグに関する説明書やサンプルなどは、2006年4月以降に弊社WEBページで公開予定です。

T-Engine上で動作するソフトウェアは、デバッガ的には表1のように分類されます。

①のソフトウェアは従来の組込みソフトウェアのデバッグと同じ手法が使えます。そのためにここでは特に説明しません。②と③のデバッグに関しては、従来のICEそのままではデバッグができません。②はカーネルがファイルから読み込み配置さいたアドレス（リロケーションアドレス）をICEが知る必要があります。また③はMMU上の多重になっている論理空間をICEが認識する必要があります。PARTNER-Jetはこのどちらにも対応しており、デバッグが可能です。ただし、②と③のデバッグ方法は異なりますので、開発しているソフトウェアによってデバッグの方法を変える必要があります（とはいっても、デバッグ開始の手順のところが違うだけで、実際のデバッグの方法や機能は同じです）。

表1　T-Engine上で動作するソフトウェア（デバッガ的な分類）

ソフトウェアの種類	空間	実行アドレス	実行
❶・T-Monitor ・T-Kernel ・T-Monitorベースのプログラム	論理・物理一致の空間 （論理アドレスと物理アドレスが一対一でマッピングされる）	コンパイル・リンク時にアドレスが決定	ROMに焼き込み、もしくはICEから転送を行い、指定の番地から直接起動
❷・デバイスドライバ ・T-Kernelベースのプログラム	論理・物理一致の空間 （論理アドレスと物理アドレスが1対1でマッピングされる）	lodspgコマンドを用いて初期化するときに実行アドレスが決定	ファイルシステム上からカーネルを介して起動する
❸・プロセスベースのプログラム	論理・多重空間	コンパイル・リンク時にアドレスが決定	ファイルシステム上からカーネルを介して起動する

●統合開発環境Eclipseでデバッグするときの現時点での問題点

Eclipseは統合開発環境なのですが、今のT-Engineの環境ではデバッグや実行の部分が統合開発環境としては扱いにくい問題があります。統合開発環境では、エディット→ビルド→実行／デバッグというサイクルが、その中で完結して簡単に実施できるようになっていることが一般的です。これはPC上のアプリケーションを開発しているセルフ開発の場合には何も問題がないのですが、クロス開発となると、少しめんどうなことが発生してきます。まずビルドしたファイルをターゲット上にどのように転送するのか？という問題が発生します。また、プログラムのデバッグや実行の開始など、そのプログラムをキックする方法も解決する必要があります。後者の場合には、最悪はコンソールからプログラム名を入力するなどの方法でも我慢ができますが、前者のビルドしたファイルの転送は、ここの自動化を実現しないと統合開発環境としての威力は半減してしまいます。NFSなどターゲットと開発ホストでファイルシステムを共有できるようなしくみがあると簡単なのですが、現時点のT-Engine開発環境ではそのような機能はありません。統合開発環境でビルドしても、実行のたびにxmodemプロトコルでシリアルケーブルでファイル転送、というのも現実的ではありません。

●ICEを利用した簡易ファイル転送

前途の問題を解決するために、ICEを使った簡易ファイル転送のしくみを実装してみます。ICEはホストPCとターゲットデバイスを接続していて、メモリのリード／ライトなどができます。デバッグでなく、少し応用すれば、簡単にファイル転送ぐらい行えます。

実装のしくみは簡単です。ICEからはファイルのイメージを特定のメモリアドレスに書き込み、ターゲット上でそのメモリをファイル化するプログラムを動作させるだけです。もちろん、このファイル化するプログラムもICEでデバッグできる状態にしておき、転送先のメモリアドレスはICEがデバッグ情報から調べ、安全なメモリアドレスに転送を行います。またICE上で動作させると、ブレークポイントを設定して、任意に実行したり止めたりすることができるので、ターゲット上で動作するファイル化プログラムの必要な機能を必要なだけ実行することが可能です。そしてPARTNERの提供するマクロ機能と組み合わせれば、任意のファイルを自動的に転送するしくみを実現できます。

今回はこのファイル化プログラムをT-Kernelベースのプログラムとして作成しました。このプログラムそのものをここで解説するには誌面が足りないので割愛させていただきます[注5]。

以降、PARTNER-JetでT-Engineソフトウェアのデバッグの様子を少しだけ紹介します。ここではデバッグのしかたを簡単に説明しますので、実際に使われる方はPARTNER-JetのT-Engineデバッグに関する説明書を参考にしてください[注6]。

●PARTNER-Jetでカーネルレベルの常駐プログラムをデバッグする

PARTNER-JetでT-Kernelベースアプリケーションやデバイスドライバ、サブシステムなどの常駐プログラムをデバッグするのは簡単です。

一般的な常駐アプリケーションのmain()の部分はリスト1のようになっています。これをリスト2のように少しのコードを追加し、そしてmakefileなどを少し設定（デバッグ情報の追加など）することで準備は完了します。正しく設定してビルドしたなら、ターゲットのコンソールでlod_spgコマンドを用いて常駐プログラムをロードすると、デバッガPARTNERで自動的にブレークしてデバッグが開始できます。図6はデバッグが開始できたときの様子です。このときには、すでにリロケーションなどは解決済みで、すぐにデバッグを行うことができます。

リスト1

```
EXPORT ER main(INT ac, UB *av[]) {

        if (0 > ac) {
                // ドライバアンロード
        } else {
                // ドライバロード
        }

        . . . . . .
```

リスト2

```
#include "kmc-ice.h"                          ←追加

EXPORT ER main(INT ac, UB *av[]) {

        if (0 > ac) {
                // ドライバアンロード
#ifdef __KMC_ICE_DEBUG__
                __kmc_driver_end();           ←追加
#endif
        } else {
                // ドライバロード
#ifdef __KMC_ICE_DEBUG__
                __kmc_driver_start();         <←追加
#endif
        }

        . . . . . .
```

●PARTNER-Jetでプロセスベースのソフトウェアをデバッグする

プロセスベースのソフトウェアのデバッグも難しくありません。デバッグしたいプログラムの開始のところにデバッグ用のフック関数呼び出しを入れるだけです（このフックを入れずに、デバッグ用でないリリース用のコードでデバッグすることもPARTNER-Jetでは可能です。しかし、ここでは手順が簡単になるので、フックを入れる方法で説明します）。フック関数を入れ込み、そしてデバッグ情報を付加してビルドします。デバッグの開始は常駐プログラムと少し異なり、あらかじめデバッグ情報をPARTNER-Jetで読み込んでおきます。そしてターゲットでプロセスを実行すれば、埋め込んだデバッグフックの次の行で停止して、プロセスのデバッグが可能になります。

図7はICEフックの入れ方と、そして実際にデバッグ開始時にICEフックの次の行でブレークした様子です。

上記のようにブレークしてしまえば、JTAG-ICEの機能をフルに使ってデバッグすることが可能です。

●EclipseとPARTNER-Jetでプロセスベースのソフトウェアをデバッグする

EclipseでPARTNER-Jetを使ってデバッグするには、KMCがリリースするPARTNER-Jet対応のEclipseを用いる必要があります。これはEclipse/CDT本体はほとんど変更されていませんが、デバッガとしてPARTNER-Jetを使うための設定画面などのプラグインが含まれています。

EclipseでPARTNER-Jetを使ってPARTNER-Jetをデバッグするのも準備は簡単です。デバッグ開始のフックを入れる変更などは、先ほど説明した内容と同じです。そしてビルドするところまでは、先に説明したこととまったく同じです。

そしてデバッグを行うのですが、デ

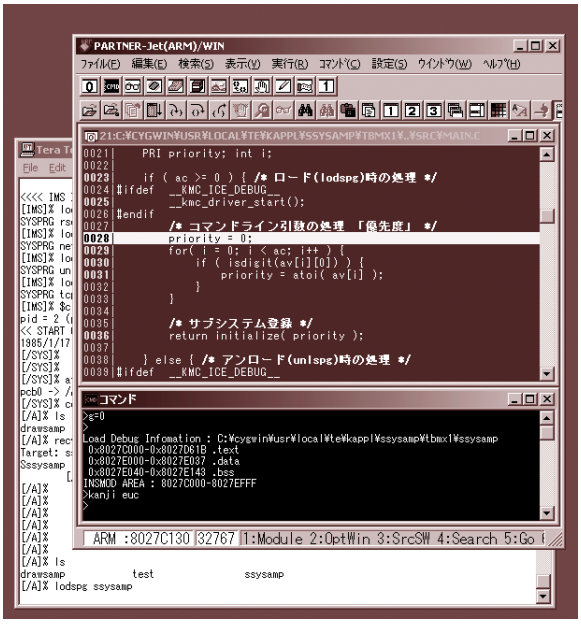

図6　常駐プログラムをPARTNERでデバッグ開始

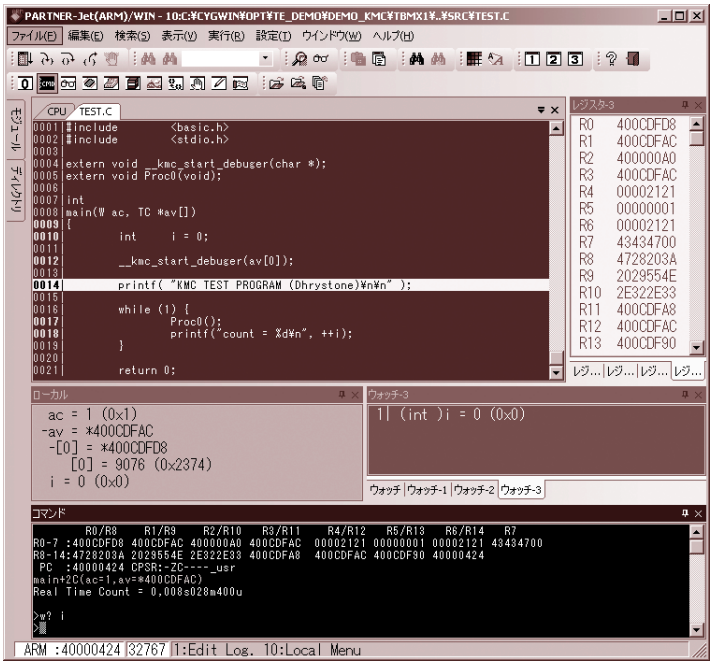

図7　プロセスベースアプリケーションをPARNERでデバッグ開始　※画面はPARTNER ver5(ベータ)のスナップショットで、2006年春出荷予定です。

バッグを始める前に利用するデバッガの設定を行う必要があります。［メニュー］→［実行］→［構成およびデバッグ］のダイアログで設定します。KMC版では、これを選ぶと図8のように「C/C++ PARTNER-Jet Target Application」という設定があるので、ここに新規にこのプロジェクトのデバッグ設定を作成します。そして、右側の設定画面でPARTNER-Jetに関する各種設定を行います。

そして、この設定が完了すると［適用］を押下して保存し、［デバッグ］押下でデバッグを開始します。なおデフォルトではビルドにエラーがあると、［デバッグ］を押下してもデバッグを開始しないので注意してください。

デバッグを開始すると、PARTNER-Jetの初期化が終われば画面が「デバッグパースペクティブ」というものに切り替わります（ちなみに、今までビルドで使っていたのは「C/C++パースペクティブ」と言います）。デバッグパースペクティブに切り替わり、［再開］ボタンを押下するとターゲットがリセットベクタから起動します。ここで先に説明したファイル転送プログラムを組み込んであると、Eclipseでビルドしたアプリケーションが自動的に転送されます。そして、ターゲットコンソールで転送したアプリケーション名を入力して起動すると、EclipseでもPARTNERの時と同じようにデバッグフックの次の行でブレークします。その様子を図9に示します。

同じところでブレークしているのですが、ずいぶんと雰囲気が違います。よく見ると、コンソールのところにはPARTNERのデバッガコンソールと同じ内容が出力されているのがわかります。デバッグパースペクティブでPARTNER-Jetを使ってデバッグしているときには、ここはPARTNER-Jetのコンソールにもなります。表示だけでなく、PARTNERのコマンドを入力して制御することも可能です。

後は、Eclipseの優れたGUIを用いて変数を見たり実行したりデバッグするだけです。またこの画面でもソース変更は可能なので、バグに気がついたらその場で変更し、即ビルドして再度デバッグすることが可能です。ビルド&デバッグを繰り返すときには統合化初環境が便利なことが改めてわかります。

Eclipseにも便利なデバッグ機能が多くあります。関数コールツリーとソー

スコード、そしてローカル変数表示の連携表示などはなかなかわかりやすく表示してくれます。また、優れたGUIのおかげで、特にマニュアルがなくても、おおよその機能は使えると思います。いろいろと便利な機能があるので、ぜひ使って体感してみてください。

終わりに

PARTNER-JetとEclipseを用いてT-Engineソフトウェアの開発について説明してきました。しかし、今の方法は現段階でできること、という制限の中での解説になっています。特に統合開発環境ということを考えた場合には、開発したプログラムの転送と実行の開始にはついては、まだまだ検討・改良の余地があります。NFSのようなしくみを用いてターゲット側とホストPCでファイルシステムが共有できるようなしくみがあれば、このあたりは非常に便利になるでしょう。PARTNER-Jetは、その高速なJTAGを応用したVirtual LinkとJTAG-Etherという技術があります。このような技術を使えば、ファイルシステムの共有など実装できそうです。今後はこのような技術の対応もしていきたいと思います。

京都マイクロコンピュータは「Eclipse Foundation Add-in Provider」に参加して積極的に Eclipseでの組込み開発環境での応用を強化していきます。それの成果として、ここで解説した内容を簡単に使えるようなものをセットにして、T-Engine向けの開発環境として提供する予定です。

T-Engineプラットフォームが今以上に普及して成長していくには、ソフトウェアの開発環境（開発用ツールソフトウェアや開発者向け情報の整備など）の進化は重要です。Eclipseなどの登場で開発ツールは強化されてきていますが、私どもも組込みソフトウェアのツールメーカーとしてT-Engineの開発環境を進化させるようにしていき、T-Engineソフトウェア開発がより活発になるよう協力していきたいと思います。

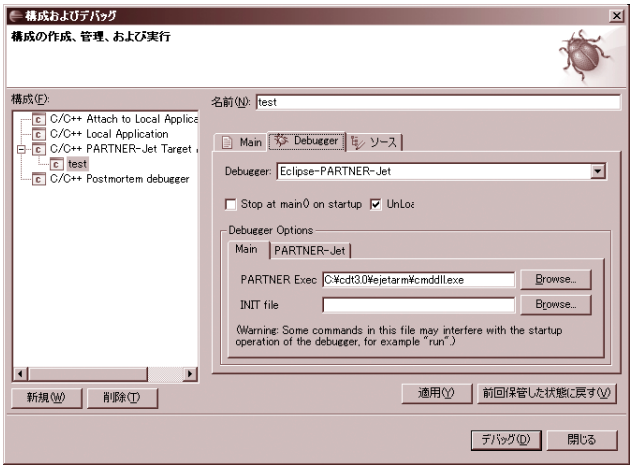

図8　EclipseでのPARTNER-Jetの設定

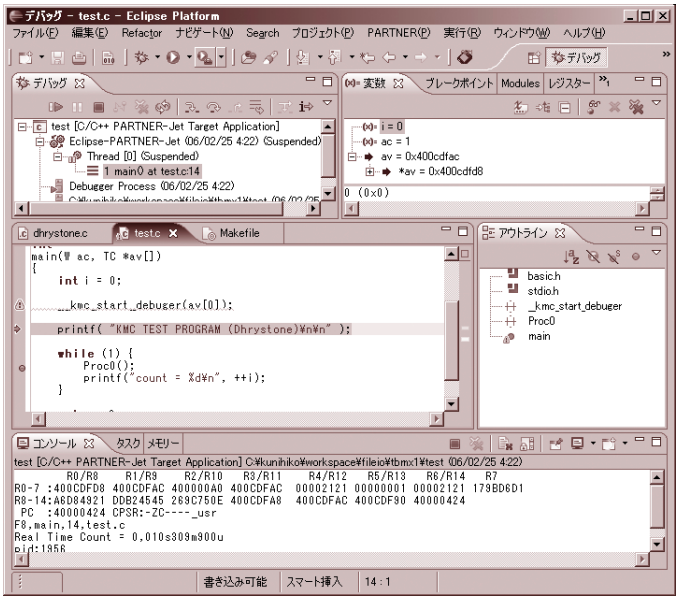

図9　EclipseとPARTNER-Jetを使ってデバッグ