ツールを使って開発期間の短縮を図ろう

WinDriver を使ったデバイス・ドライバの開発

エクセルソフト株式会社　西 伸顕

Jungo 社製の WinDriver というツールを利用して、デバイス・ドライバを開発する手法を紹介します。

一般に、デバイス・ドライバを開発するためには、OS の内部構造 注1 や、ドライバ・インターフェース 注2 などの基礎知識が必要となります。また、開発には多くの手順を踏むうえに、各 OS に応じて開発する必要があるので、たくさんの作業と時間が必要になります。

WinDriver を利用すると、多くの手順が自動化できるうえに、上記のような OS の知識の習得が不必要になるために、開発期間の短縮が図れます。

さらに、Windows や Linux、Solaris、VxWorks といった OS 間で互換性のあるソース・コードを記述することも可能です。

WinDriver の評価版は、エクセルソフト(株)の Web サイトから無償でダウンロードできます。

http://www.xlsoft.com/jp/products/download/

1. WinDriver と開発手順の概要

● WinDriver の基本構造

WinDriver は、カーネル・モードではなく、ユーザ・モードで開発を行います（図 1）。また、ウィザードでハードウェアの診断を行い、自動的にハードウェア独自のドライバ・コードを生成できます。ハードウェア独自のドライバ・コード（アプリケーション）は、WinDriver が提供するカーネル・レベル（Ring-0）で動作する汎用的なカーネル・モジュールを使用します。このカーネル・モジュールが提供する、ハードウェアへアクセスする API を使用して、アプリケーションはハードウェアのレジストリ、メモリ範囲、I/O 範囲へのアクセス、ハードウェアの割り込みの処理（PCI/ISA の場合）およびデバイスのパイプのデータ転送（USB の場合）を行います。

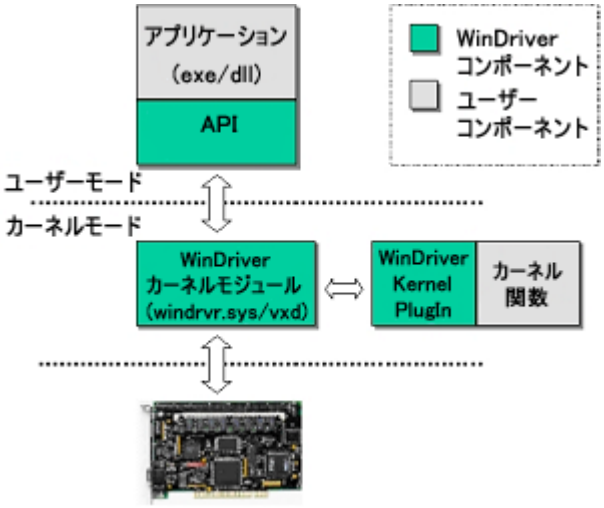


図 1　WinDriver のアーキテクチャ

● 従来までの手間と時間のかかる開発手法

今までの一般的なドライバの開発は、以下のような手順で行われてきました。

1）OS の内部構造を学習（Windows、Linux、VxWorks など）
2）各 OS でのデバイス・ドライバの記述方法を学習（DDK など）
3）カーネル・モードの開発や、デバッガ・

ツールの使用方法を習得
4）カーネル・モードのデバイス・ドライバを記述（基本的なハードウェアの I/O 部）
5）ユーザ・モードでアプリケーションを記述（カーネル・モードで記述されたデバイス・ドライバを介して、ハードウェアにアクセス）
6）対応する OS ごとに 1）～4）の手順を繰り返す

● WinDriver による開発手順

WinDriver を用いると、カーネル・モードの開発はもちろん、OS の内部構造やバス・プロトコルの知識の習得といった手順を省いても開発が行えるので、結果として、開発時間の短縮となります。

WinDriver での開発手順は、WinDriver の DriverWizard を使用してハードウェアの動作を検証した後、自動的にクロス・プラットホームな（互換性のある）ドライバ・コードを生成します。そのアプリケーションを雛形とし、開発者はコードに必要な機能のみを追加するだけです。DriverWizard では、一般的な開発環境（Microsoft Visual Studio、Borland Builder、Linux gmake など）用のメイク・ファイルを生成します。生成されたコードは修正せずにコンパイルし実行できます。

2. WinDriver を使ったデバイス・ドライバの開発手順

ここからは、WinDriver を使用するとどれくらいの作業が削減できるのかを知るために、実際に WinDriver を使って PCI カードの Windows 用のデバイス･ドライバを開発していきます。

今回、ターゲットとした PCI カード（VenderID 6809、DeviceID 8000）の動作概要は、以下のとおりです。

- ディップ・スイッチ入力レジスタを読み出すと、PCI ボード上に実装した 8 ビット・ディップ・スイッチの状態を読み出す。ディップ・スイッチの状態が ON でビット 1、OFF でビット 0
- LED 点灯データ出力レジスタに 1 を書き込むと､PCI ボード上の 8 ビット LED が点灯する
- PCI ボード上の 7 セグメント LED の隣に並ぶ 4 個のプッシュ・ボタンのいずれかを押下すると、割り込みが発生する

● ステップ 1――インストール

WinDriver のインストール・プログラム（WDxxx.EXE。xxx はバージョン番号）を起動し、インストーラの指示に従ってインストールを実行します。インストールは、システム管理者の権限のあるユーザで行ってください。

インストールの際には必要ありませんが、WinDriver で生成されたコードをコンパイルおよびビルドするために C/C++、Visual Basic､Delphi などの 32 ビット開発環境が必要となります。また、Kernel PlugIn 機能を使用する場合にのみ、DDK をインストールする必要があります。

● ステップ 2――ハードウェアの選択

次に、下記のような手順でハードウェアの選択を行います。

- DriverWizard を起動し､表示されたダイアログ・ボックスから［Create a new driver project］を選択
- ［Select Your Device］ダイアログ・ボックスで､Plag-and-Play カードがすべて表示されるので、対象のデバイス［Vendor ID 6809 Device ID 8000］を選択（図 2）

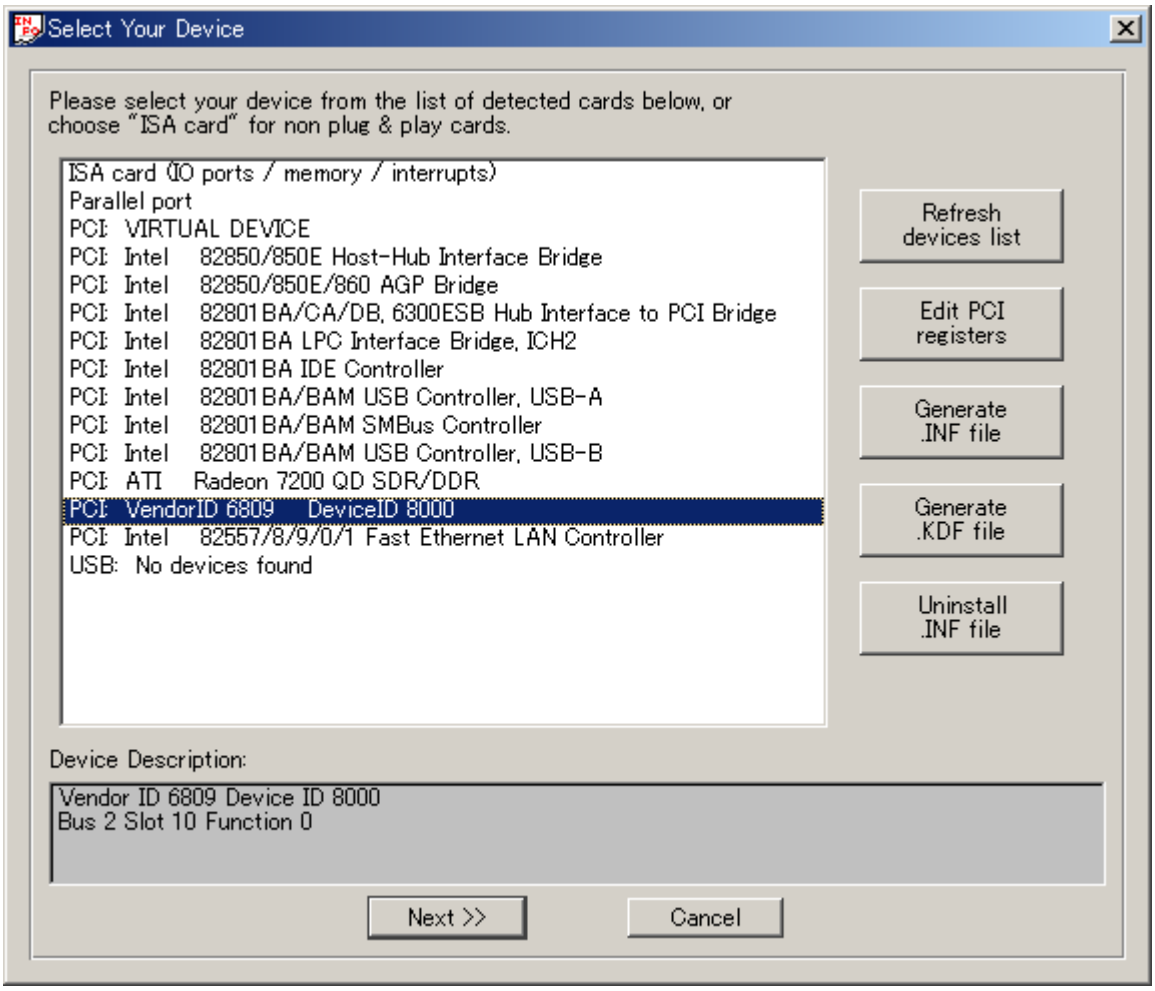


図２　対象となる PCI カードを選択

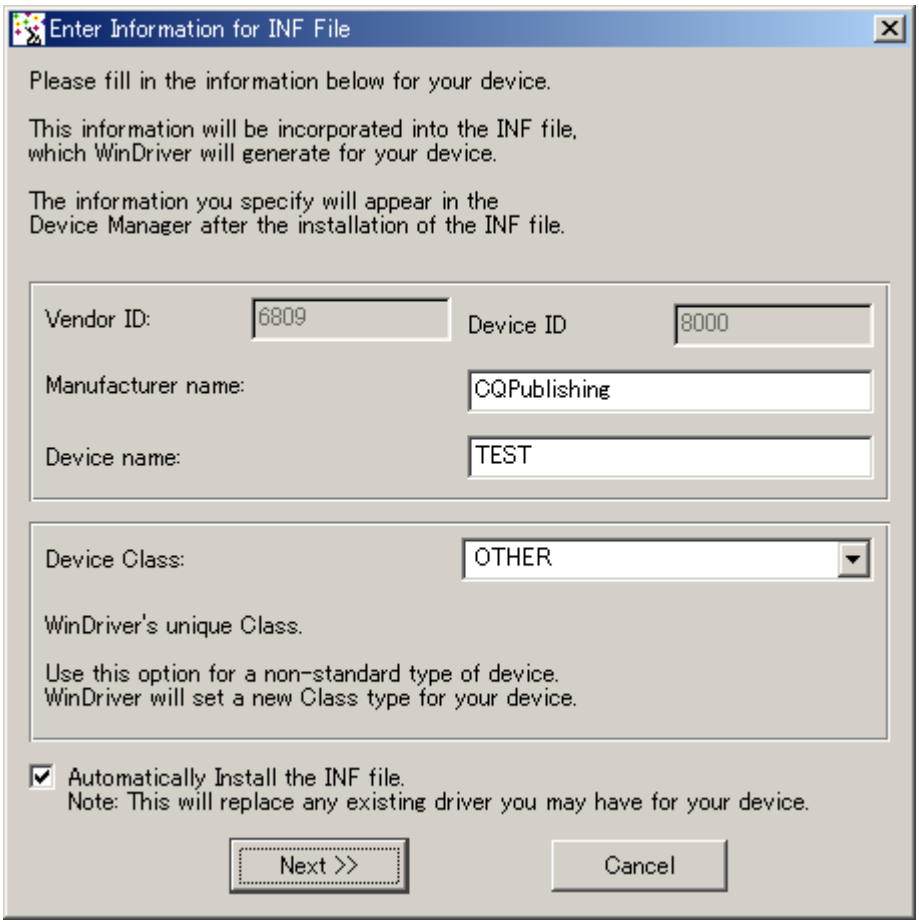


図 3　INF ファイルの生成

● ステップ 3――INF ファイルの生成

PCI デバイスのドライバがインストールされていない新規の Plag-and-Play カード用のドライバを開発する場合は、対象のデバイスの INF ファイルを生成してインストールする必要があります。DriverWizard では、これを自動的に行えます（図 3）。

- ［Generate .INF file］または［Next］をクリック
- 表示されたダイアログ・ボックスに必要な項目を入力
- ［Next］をクリックし、INF ファイルの保存先のディレクトリを選択。Windows 2000/XP/Server 2003 上では、［Automatically Install INF file］オプションをオンにすることによって DriverWizard が自動的に INF ファイルをインストールする
- INF ファイルのインストールが終了したら、ステップ 2 の［Select Your Device］ダイアログ・ボックスに戻り、再度、対象のデバイスを選択

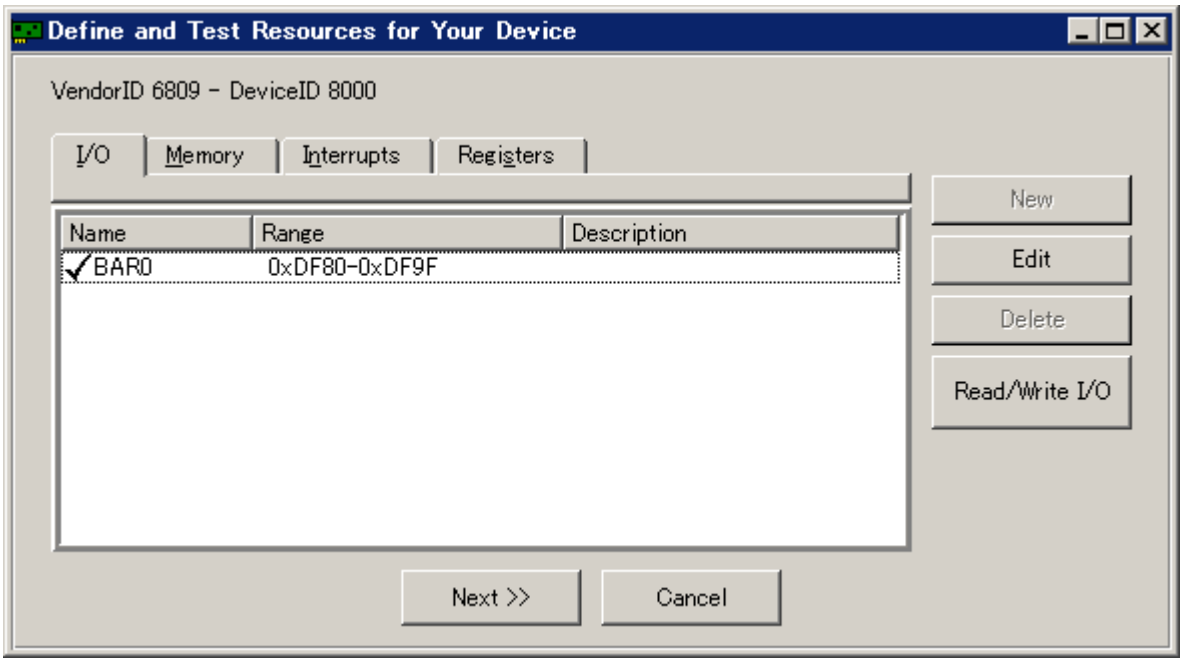


図 4　対象となる PCI カードのリソースの検出

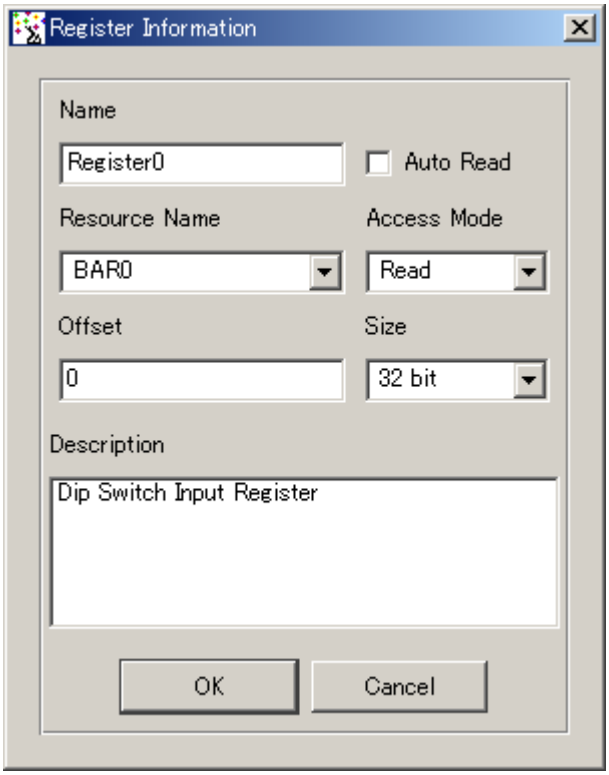


図 5　レジスタの定義

Register0 を定義。Register1～3 も同様に定義する。

● ステップ 4――ハードウェアの検出と定義

DriverWizard は、Plag-and-Play ハードウェアのリソース（I/O 範囲、メモリ範囲、割り込み）を自動的に検出します（図 4)。

I/O レジスタ（表 1）については、手動で定義します。［Registers］タブをクリックし、［New］ボタンをクリックすると、レジスタが定義できます（図 5)。

次に、割り込み情報の定義を行います（図 6)。

表 1　I/O レジスタの仕様

オフセット	アクセスサイズ	R/W	ビット	用途
+00h	32 ビット	R	ビット 7 – 0	ディップスイッチ入力レジスタ
+04h	32 ビット	R/W	ビット 7 – 0	LED 点灯データ出力レジスタ
+08h	32 ビット	R W	ビット 0 ビット 0	割り込みステータス・レジスタ (1 で割り込み発生) 割り込み要求クリア (1 で割り込みクリア)
+0Ch	32 ビット	R/W	ビット 0	割り込みマスク・ レジスタ (1 で割り込み解除)

● ステップ 5――ハードウェアの検証

デバイス・ドライバを記述する前に、ハードウェアが期待どおりに機能するかどうかを診断する必要があります。ドライバのコードを記述することなく、DriverWizard でハードウェアの診断を行えます。

- I/O、メモリ、レジスタへの読み書きを行う
- Register0 を検証する。ディップ・スイッチ入力レジスタを読み出す。PCI ボード上の 8 ビット・ディップ・スイッチの状態が ON でビットが 1（図 7）
- Register1～3 についても、検証していく
- ハードウェアの割り込みを Listen（確認）する。[Listen to Interrupts]をクリック（図 8）

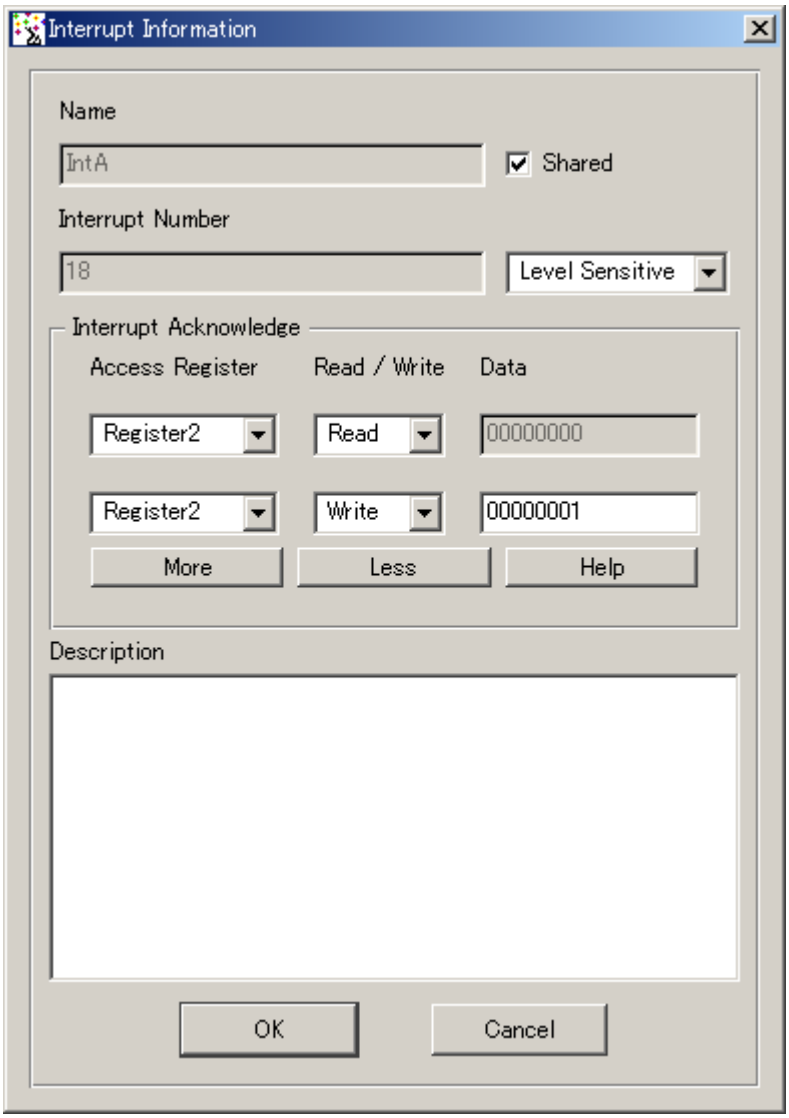


図 6　割り込みの定義

図 7　ハードウェアの検証

Register0 を検証。Register1～3 も同様に検証する。

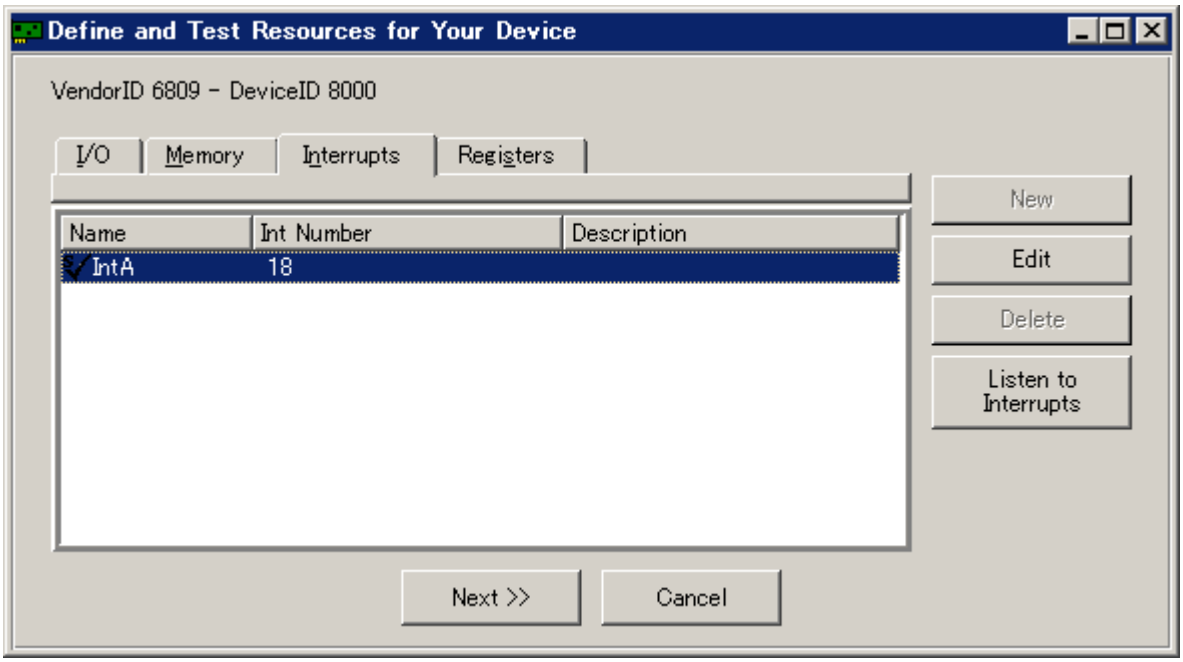


図 8　割り込みの確認

● ステップ 6――ドライバ・コードの生成

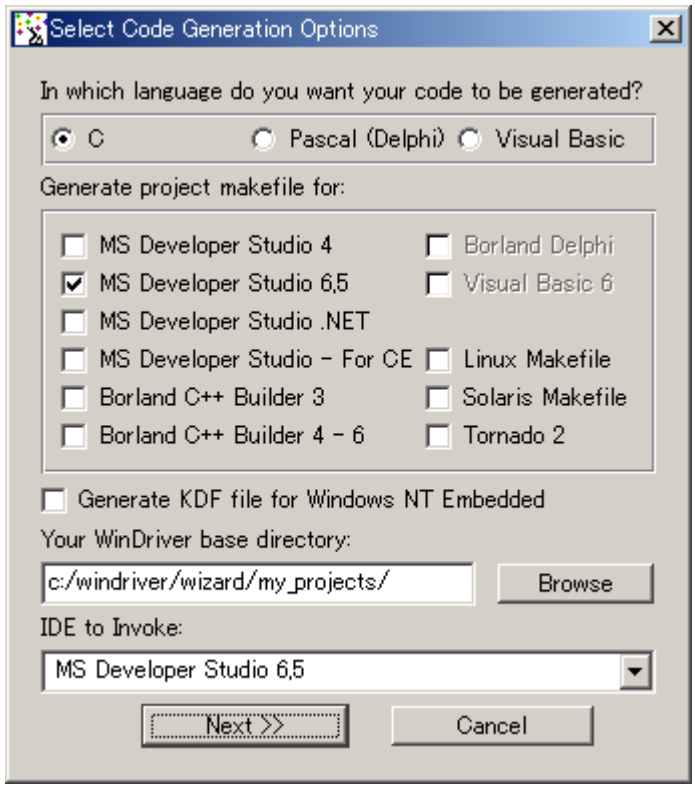
ハードウェアの検証後、DriverWizard で自動的にドライバの雛型となるコードを生成します。

- ［Next］をクリックするか、Build メニューから［Generate Code］を選択
- ［Select Code Generation Options］画面で、開発言語を選択し、作成するプロジェクトの開発環境を決める。ここでは、“ C 言語 ”で“ MS Developer Studio 6,5 ”を選択した〔図 9(a)〕
- ドライバ・コード内で Plag-and-Play と Power Management イベントを処理する場合と、KernelPlugIn コードを生成する場合には、図 9(b)の画面で選択（Kernel PlugIn 機能を使用する場合には、DDK をインストールする必要がある）
- ［Next］をクリック。DriverWizard が自動的にコードを生成し、選択したコンパイラを起動する

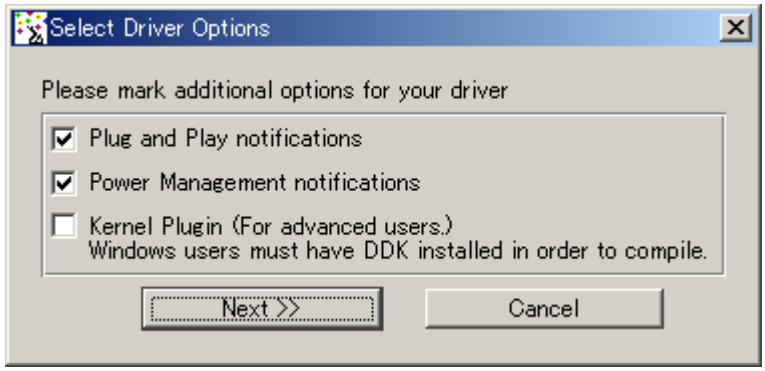
C/C++を使用する場合、DriverWizard は次のファイルを生成します（“ test ”はプロジェクト名）。

- test_files.txt――生成されたファイルの説明を記述した readme ファイル
- test_diag.c――ターゲット・デバイス用に DriverWizard が生成した雛形となるアプリケーション。ライブラリ関数の使用方法を明示
- test_lib.c――test_diag.c で使用するデバイスへのアクセスで使用するユーティリティ関数の一般的なライブラリ
- test_lib.h――ユーティリティ関数のヘッダ・ファイル
- 選択した開発環境用のプロジェクト・ファイル

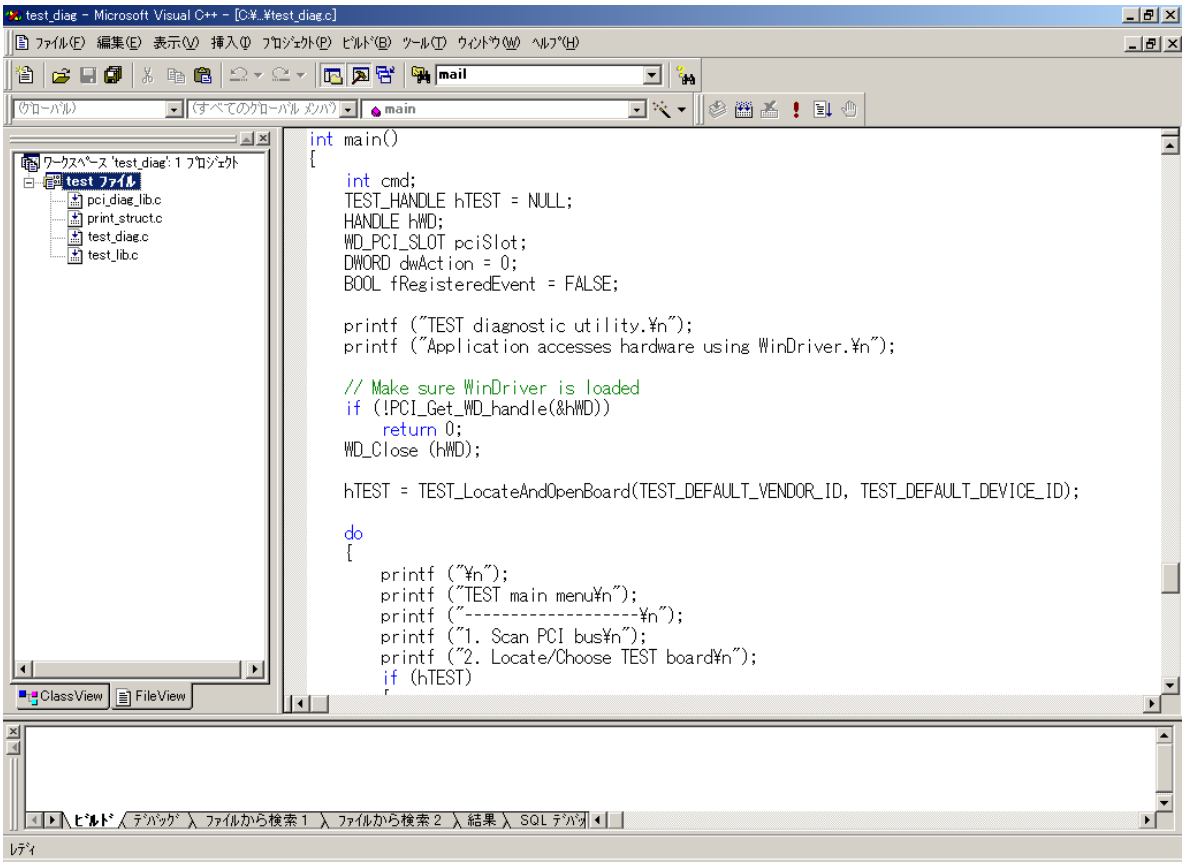
DriverWizard が生成したコードをビルドし、実行します。これで PCI カードの診断アプリケーションの完成です。この診断アプリケーションがドライバの雛型となります。



(a) 開発環境の選択



(b) オプションの選択



(c) コンパイラの起動

図 9　ドライバ・コードの生成

● DriverWizard が生成したコード

リスト 1 のコードは、DriverWizard が生成した API 関数です。ステップ 4 で定義したリソース Bar0 Range にあるレジスタ“ Register1 ”にアクセスし、Read/Write するためのコードです。

なお、コメント文も自動的に生成されますが、英語で書かれているため、ここでは日本語に訳して掲載しています。

そのほか、PCI カードを WinDriver のカーネル・モジュールと動作するようにレジスト/アンレジストするためのコードと、PCI カードの割り込みを有効/無効にするコードについては、長くなるのでここでは割愛します。主な WinDriver API の一覧を表 2 に示します。

リスト 1　DriverWizard が生成した API 関数

```
// 関数: TEST_ReadRegister1()
//   レジスタ Register1 からの読み込み
// 引数:
//   hTEST [in] - TEST_Open() 関数から受信したカードへのハンドル
// 戻り値:
//   レジスタから読み込んだ値
UINT32 TEST_ReadRegister1 (TEST_HANDLE hTEST)
{
   return  TEST_ReadDword(hTEST,  (TEST_ADDR)  TEST_Register1_SPACE,
TEST_Register1_OFFSET);
}
```

(a) Register1 からの読み込み

```
// 関数: TEST_WriteRegister1()
//   レジスタ Register1 への書き込み
// 引数:
//   hTEST [in] - TEST_Open() 関数から受信したカードへのハンドル
//   data [in] - レジスタへ書き込むデータ data [in]
// 戻り値:
//   なし

void TEST_WriteRegister1 (TEST_HANDLE hTEST, UINT32 data)
{
   TEST_WriteDword(hTEST,     (TEST_ADDR)     TEST_Register1_SPACE,
TEST_Register1_OFFSET, data);
}
```

(b) Register1 への書き込み

表 2　主な WinDriver API の一覧

開始 & 終了	PCI の I/O およびメモリアクセス	Plug-and-Play & パワー マネージメント
WD_Open()	WD_Transfer()	WD_EventRegister()
WD_Close()	WD_MultiTransfer()	WD_EventUnregister()
WD_Version()		WD_EventPull()
WD_License()	**PCI DMA**	WD_EventSend()
	WD_DMALock()	
PCI	WD_DMAUnlock()	**ユーティリティ & デバッグ**
WD_CardRegister()		
WD_CardUnregister()	**PCI 割り込み処理**	WD_Debug()
WD_CardCleanupSetup()	InterruptThreadEnable()	WD_DebugAdd()
	InterruptThreadDisable()	WD_DebugDump()
WD_PciScanCards()	低レベル:	
WD_PciGetCardInfo()	WD_IntEnable()	WD_LogStart()
WD_PciConfigDump()	WD_IntDisable()	WD_LogStop()
	WD_IntCount()	WD_LogAdd()
WD_IsapnpScanCards()	WD_IntWait()	
WD_IsapnpGetCardInfo()		WD_Sleep()
WD_IsapnpConfigDump()		

3. デバッグとパフォーマンスの向上、互換性

● デバッグ

従来のデバイス･ドライバのデバッグでは、カーネル・モードのデバッグが必要でした。デバイス・ドライバは、OS の一部として動作するので、デバッガによってドライバのプロセスを停止すると OS も停止し、ほかのプロセスも停止してしまいます。そのため、デバイス・ドライバをデバッグする際には、2 台のコンピュータをケーブルでつなぎ、1 台をデバッグするソフトウェアを実行するホストとし、もう 1 台をターゲットのマシンとしてデバッグを行います。

WinDriver ではユーザ・モードでドライバを開発するので、開発環境マシン上でユーザ・モードのデバッグ・ツールを使用してデバッグを行います。WinDriver にはデバッグ･ツールとしてデバッグ･モニタ･ユーティリティがあり（GUI ベースとコンソール・モードがある）、デバッグ・モニタで WinDriver のカーネル（windrvr.sys/windrvr.vxd/windrvr.dll/windrvr.o/wdpnp.sys）が処理するすべての動作を監視し、カーネルへ送られる各コマンドがどのように実行されるかを監視できます（図 10）。WinDriver で作成するドライバは、ユーザ・モードで動作するため、MS Developer Studio のユーザ・モード・デバッガも使用できます。

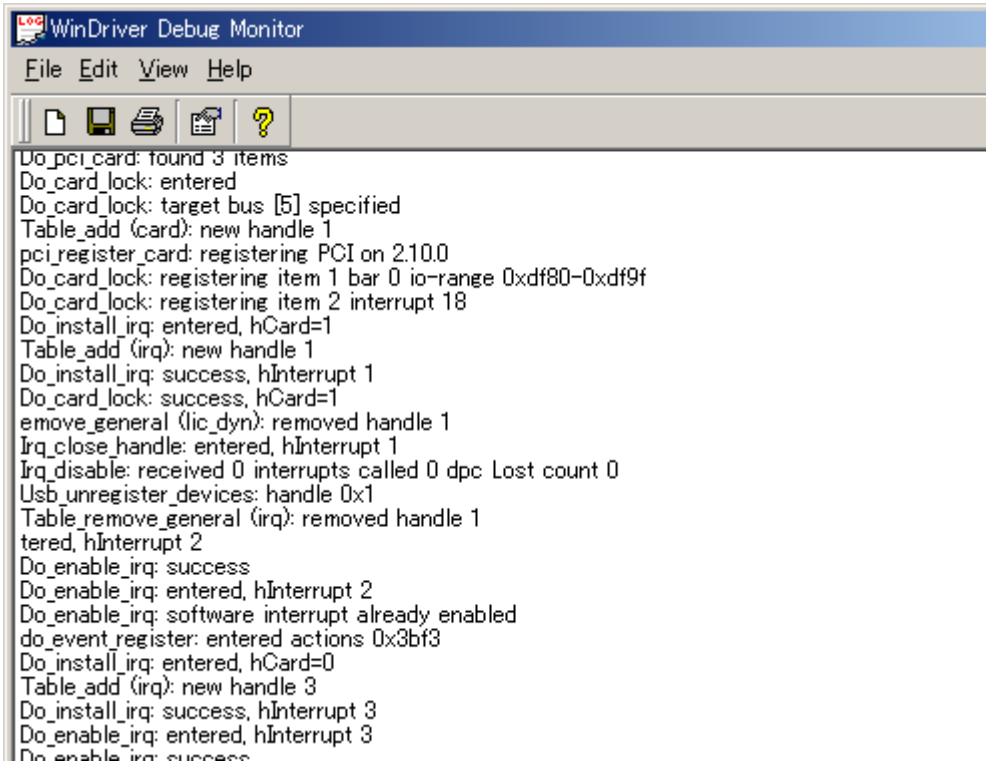


図 10　デバッグ・モニタの出力内容

● パフォーマンスの向上

アプリケーション・レベルでメモリおよび割り込みを処理する際に、カーネルからユーザ・モードへの関数を呼び出すところでオーバヘッドが発生します。この問題を解決するには、パフォーマンスの重要なコード部分をカーネル・レベルで実行できるアーキテクチャが必要です。開発者は、最初にユーザ・モードで簡単にすばやく開発し、パフォーマンスの重要な部分のコードを必要に応じて処理します。WinDriver には、Kernel PlugIn アーキテクチャがあり、これでコードのパフォーマンスの重要な部分をユーザ・モードからカーネル・モードに移行し、コードのパフォーマンスの最適化を行えます。ただし、Kernel PlugIn を使用する際には、DDK をインストールする必要があります。

WinDriver は、メモリ転送コマンドをカーネル・レベルで実行するので、通常は Kernel PlugIn を使用しません。Kernel PlugIn を必要とするのは、ハードウェアが高い割り込み速度を必要とする場合や、ハードウェアのメモリがメモリにマップされていない場合（たとえば I/O マップ）などです。PCI デバイスの場合、簡単なハードウェアの修正でハードウェアを I/O マップからメモリ・マップに変更できます。メモリ・マップのカードの場合、WinDriver はユーザ・モードのポインタを提供し、これを使用してユーザ・モードから直接カードのメモリからデータを転送でき、WD_Transfer() API 関数を呼ぶ必要がなく、かつパフォーマンスを向上します。

● クロス・プラットホーム――ドライバ・コードの互換性

WinDriver は、ハードウェアにアクセスするアプリケーション・レベルの API を提供します。その API は、カーネル・モジュールを呼び出し、OS 独自のカーネル API を使用してハードウェアにアクセスします。さまざまな OS のカーネル・モジュールを提供することによって、コードの修正をせずにドライバをほかの OS に移植できます。

WinDriver で開発したドライバは、対応する OS（Windows 98/Me/NT/2000/XP/Server 2003/CE、Linux、Solaris、VxWorks） 間でソース・コードで互換性があります。このうち、Windows の間ではバイナリ・レベルで互換性があります。UNIX システムの場合、ソースでの互換性があるので、再コンパイルのみが必要となります。WinDriver では、生成したコードを修正せずに、ほかの OS へも移行できる柔軟性を持っているのです。

＊　　＊

フル機能を備えた、機能制限のない WinDriver の評価版が、エクセルソフト(株)の Web サイトからダウンロードできるので、ぜひ、お試しください。

http://www.xlsoft.com/jp/products/download/

なお、製品版と評価版の違いは以下のようになっています。

- 評価版の有効期間はインストール後 30 日間です
- 評価版では、評価版であることを示すメッセージをつねに表示
- DriverWizard を使用中に評価版を実行していることを知らせるダイアログ・ボックスを表示
- Linux、Solaris、VxWorks、Windows CE 版では、60 分間動作した後、停止する。再度評価する際には、再ロードする必要がある

COLUMN
WinDriver のユーザ事例

本文中では PCI デバイス・ドライバの開発手法を紹介しましたが、WinDriver には、USB1.1/2.0 に対応したバージョンもあります。実際に WinDriver を使用して、USB のデバイス・ドライバ開発を行ったユーザの事例を紹介します。

● 製品概要――MSX ゲームリーダー

アスキーおよびマイクロソフトが提唱した 8 ビットのゲーム機「MSX」向けのゲーム ROM カートリッジを Windows が搭載された PC よりアクセスする USB 機器です。公式エミュレータである「MSXPLAYer」とセットで使用し、Windows 上で MSX のソフトを動作させることができます。

● WinDriver の使用

ルネサステクノロジー製「H8S/2215UF」を使用した USB デバイスである「MSX ゲームリーダー」のドライバ DLL の作成に WinDriver を使用しました。前述の公式エミュレータから、作成したドライバにアクセスしています。処理としては、ゲーム ROM カートリッジにアクセスするためのアドレス

とデータの受け渡しと、書き込み／読み出しデータの転送を行っています。

● 開発環境

- Windows 2000 Professional
- Visual Studio C++6.0（ドライバ側のみ。エミュレータ部は別環境で開発して組み合わせた）

● WinDriver を採用した理由

- 動作が安定していること
- 学習が容易なこと
- 開発効率が良いこと
- デバイスの着脱に対応していること

● WinDriver のメリット

- ウィザードに従って設定するだけで、INF ファイルと VC++での雛形を含んだプロジェクトが作成され、その時点でデバイスのテストを行うことができた
- USB デバイスの着脱に対応していた。デバイスのアクセス中にデバイスが抜かれた場合も適切な処理がなされた（アクセス関数が処理を中断し、エラー・コードを返した。着脱時の処理はコールバック関数として記述するだけ）
- Windows API の ReadFile／WriteFile 関数を使うような感覚で、特に違和感なくプログラミングできた
- ユーザ・モードで開発し、デバッグできたため開発効率が良かった
- 評価版があり、かつ評価版の試用期間を延長できたので、十分な評価を行ったうえで採用できた

にし・のぶあき エクセルソフト株式会社

注 1：デバイス・ドライバは、実行する OS と密接な関係があり、デバイス・ドライバを開発する際には、OS のアーキテクチャおよび内部構造の知識が必要となる。

注 2：ドライバ・インターフェースのセットには、Windows 98/Me/NT/2000/XP/2003 Server の DDK、Windows CE の ETK などがある。ドライバには、各 OS 独自の API を使用し、開発者は OS に応じたドライバを開発する必要がある。