スイッチ／測定ソリューションの比較

Application Note

このアプリケーション・ノートは、ファンクション・テストやデータ収集で使用するスイッチ／測定ソリューションの機能、実行速度、ソフトウェア開発の容易さを比較したものです。ここでは、Keithley 27xx、Racal 1256、Agilent 34970A、Agilent 34980Aの各スイッチ／測定ユニットとアプリケーション開発環境Visual Studio.NETでのAgilent 3499A/34401Aの組み合わせ、Agilent E1411B/ E1476A VXIの組み合わせ、National Instruments PXI-4070/SCXI-1128 PXIの組み合わせについて比較しています。

スイッチ／測定システムの特長

テスト・システムには、様々な測定機能がありますが、デジタル・マルチメータ（DMM）、リレー（スイッチ）という2種類のコンポーネントは常に存在する基本的なコンポーネントです。この機能は、以下の3種類の方法で実装できます。

- Agilent 3499A/B/CスイッチボックスやRacal 1256スイッチボックスとスタンドアローンのAgilent 34401A DMM、E1411B VXI DMMなどの個々の測定器を使用。
- Agilent E1411A VXI DMMとE1476A VXIスイッチ・カード、NI PXI-4070 DMMとSCXI-1128スイッチ・カードなどのVXIやPXI、PXIハイブリッドのメインフレームを使用。（後者はハイブリッド構成で、制御信号はPXIバックプレーンとSCXIバックプレーンで共有されます。これは、コンボのシャーシ内、またはPXIシャーシからスタンドアローンのSCXIシャーシへのケーブル配線で行われます。）
- DMMやスイッチ・カードを内蔵できるAgilent 34980A、Agilent 34970A、Keithley 2701などの専用の測定器を使用。

本書で詳しく説明するように、各ソリューションのすべてに利点と欠点があります。しかし、その使用モデルは、一般にデータ収集とファンクション・テストという2つのカテゴリーのいずれかです。どのソリューションを選択するかは、使用目的に大きく依存します。

データ収集とファンクション・テストの違い

同じ電圧計とスイッチを、以下の2種類のテスト・システムで使用できます。

- **データ収集（DAQ）**。機械／電気／電子装置の性能を評価するために、多数の測定値を収集します。例えば、大気圏に再突入する際の、宇宙船の数千箇所の温度を測定します。
- **ファンクション・テスト（EFT）**。被測定デバイス（DUT）と呼ばれる電子モジュールに信号を印加して、その出力をモニタし、リミットと比較します。例えば、自動車のエンジン制御モジュールが適切に動作しているかどうかを測定します。

Agilent Technologies

これらの2種類の環境では、同じハードウェアでも要求されるスループットが大きく異なります。データ収集では、電圧計とリレーは一緒にスキャン測定を行うようにプログラムされます。スイッチボックスにスイッチの開閉状態のリストがダウンロードされ、DMMによる測定と適切なスイッチ・セットアップを、ハードウェアのハンドシェークによりリンクします。スイッチの開閉リストを実行するために、電圧計からの測定完了トリガを使用したり、スイッチをフリーランにできます。DMMは、測定完了トリガを送出した後で、プログラム遅延の間ウエイトしたり、スイッチからのハードウェア信号を受信するまでウエイトできます。この信号はアドバンス・トリガと呼ばれ、スイッチがプログラムされた状態に進んで読取りOKであることを表します。このハードウェアのハンドシェークにより、各測定器間で転送されるI/Oの数を最少限に抑えることが可能です。さらに、測定のたびにセットアップ時間を繰り返す必要もありません。セットアップ情報をダウンロードしておけば、1回の開始コマンドで全体の測定を開始できます。そのため、非常に高速に実行できます。必要な測定分解能やスイッチング速度にもよりますが、1,000回/sの測定が可能です。このようなシステムでは多くのデータを転送する必要があるので、VXI/PXIシステムが有利です。

ファンクション･テストでは状況は異なります。信号を印加するたびに1回の測定を行い、このプロセスを、DUT内の全ピンに対して繰り返す必要があります。個々の測定のオーバヘッドと、信号と測定器の再構成（リレーをクローズし、読取りを行って、リレーをオープンする）が何回も必要になります。多くの場合、読取りが可能かどうかを確認するポーリングを行うので、これもI/Oの実行時間を増加させます。そのため、実行速度として100～500回/sの読取りが一般的です。測定回数が少ないので、この環境では高速のバックプレーンはあまり意味がありません。低コストの専用スイッチ／測定システムが適しています。

カードケージ

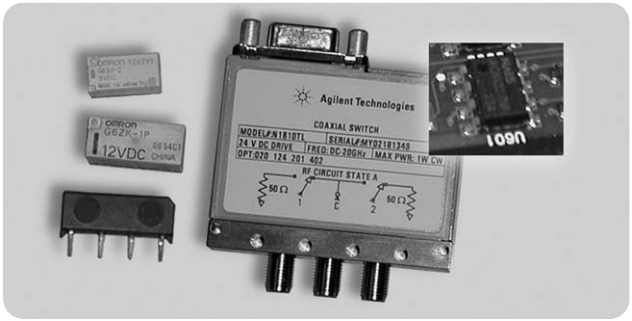
市販のスイッチング・システムはすべてカードケージとして提供され、様々なタイプのカードがあります。VXIやPXIのように、オープン・アーキテクチャとしてバックプレーン・インタフェースが文書化され、複数のベンダから製品が提供されているものもあります。NI社のSCXI、Racal社の1256、Keithley社の27XX、Agilentの3499ファミリ、34970A、新製品の34980Aなど、ベンダ独自の内部バックプレーンを持っているものもあります。独自のデザインを多くのメーカ（とユーザ）が選択する理由は、オープン規格が要求する余分な機能にコストをかけなくて済むからです。必要十分なだけのパワー、冷却、バックプレーン速度をボックスに実装して、そのスイッチ・カード用に最適化できます。ベンダ固有のアーキテクチャでは、ブレッドボード・カードを使用したカスタマイズが可能な場合もあります。このようなカードは、一般にアドレス・デコーダが内蔵されていて、ユーザが独自の回路を追加することができます。

リレーとDMMに加えて、一緒に使用されることの多い以下のようなその他の機能もスイッチ／測定システムに実装されています。

- デジタルI/O（DIO）
- DAコンバータ（DAC）。低周波の波形発生器として使用するのに十分な速度を持っている場合があります。
- 周波数カウンタ／トータライザ
- 等温ターミナル・ブロック
- カスタマイズ（ブレッドボード）カード（シグナル・コンディショニング用に便利）。一般に歪みゲージ用に必要。

スイッチ・カードにも様々なタイプがあり、マルチプレクサ、マトリックス、汎用スイッチなどがあります。

- 低周波（DC～100 MHz）
  - FETリレー（高スイッチング速度、高オン抵抗、低電圧、低電流）
  - リード・リレー（中スイッチング速度、低オン抵抗、中電圧、中電流。低熱起電力タイプも利用可能）
  - アーマチュア・リレー（低スイッチング速度、低オン抵抗、高電圧、高電流。低熱起電力タイプも利用可能）
- 高周波（～18 GHz）
  - RFリレー（低スイッチング速度、低オン抵抗、低電圧、低電流）
  - マイクロ波リレー（低スイッチング速度、低オン抵抗、低電圧、低電流）



プログラミングの容易さ

アプリケーション開発環境

どのようなテスト・システムでも測定器のプログラミングが必要です。一般に、以下のようなアプリケーション開発環境（ADE）が使用されます。

- グラフィカル言語
  - Agilent Technologies VEE Pro
  - National Instruments LabVIEW
- テキスト言語
  - Microsoft Visual Studio.NET（VB/VC++ 7.0、C#など）
  - National Instruments LabWindows/CVI
  - 以前のバージョンのMicrosoft言語（VB/VC++ 6.0など）
  - Rocky Mountain Basic（“HP Basic”）

これらの中でもっとも広く使用されている3つの環境はVEE Pro、LabVIEW、Visual Studio 6.0ですが、Visual Studio.NETの使用も増加しています。Visual Studio.NET環境に対しては、AgilenはT&M Programmers' Toolkitというソフトウェア製品を、NI社はMeasurement Studioを提供しており、どちらも測定器の使用が簡単になります。本書のベンチマーク・テストでは、Agilent T&M ToolkitおよびNI社 Measurement Studioのいずれを使用してもVisual Studio.NET環境下ですべての測定器をプログラミングすることができました。

ドライバとSCPI

一般にスタンドアローンの測定器は、GPIB、USB、LANなどのインタフェース経由でASCIIコマンドを送り、操作します。このようなコマンド言語は、SCPI（Standard Commands for Programmable Instruments）として標準化されています。このコマンドは、適切なインタフェースと直接通信する下位レベルの関数コール、または上位レベルの関数コールを使用して測定器に送ります。上位レベルのコールは、ドライバと呼ばれるソフトウェアに含まれています。これらのドライバはユーザによる自作が可能ですが、メーカやサードパーティが提供するドライバを使用することもできます。

VXIやPXIなどのカードケージを使用するシステムは、ASCIIコマンドではなくデータ・レジスタを介した制御がよく行われます（VXI規格の一部として、内蔵ワード-シリアル・プロトコルに依存するメッセージ・ベースのカードも一部にはあります）。必要なデータをレジスタに送出するために、前述の下位レベルのインタフェース関数コールの使用も可能です。しかし、そのようなプログラムを書くにはかなりの手間が必要で、通常はメーカが提供するドライバを使用します。レジスタ・ベースの測定器には、SCPIコマンドを受け取れるものもあります。Agilentは、一部のVXIカード用にD-SCPI（Downloadable SCPI）ドライバを用意しています。このドライバは、VXIメインフレームのスロット0に配置されたGPIBコントローラ（E1406Aコマンド・モジュール）のフラッシュROMにあります。Agilentでは、VXI FireWireインタフェース用のI-SCPI（Interpreted SCPI）も提供しています。I-SCPIはPCによる高速なコマンド解析を使用するので、通常はD-SCPIよりもかなり高速に動作します。PXIについては同様の機能はなく、PXIカードはすべてレジスタ・ベースであり、メーカが提供するドライバが必要です。VXI/PXI測定器用のドライバはコンパイルされたDLLとして提供され、どの言語でも使用できます。また、Agilent VEE ProやNI社のLabVIEW用の特殊なドライバとして提供されることもあります。

ドライバが必要なPXIカードの場合を除いて、ドライバの多くは測定器の全機能をサポートしているわけではありません。「パススルー」機能というのがあり、ドライバに実装されていないSCPIコマンドをドライバから測定器に送れるようになっています。

Visual Studio環境では、MicrosoftのIntelliSense機能によりドライバの使用が非常に簡単になります。プログラマは分かりやすい名前のついた機能をドロップダウン・リストから選択するだけで、その機能の説明やパラメータを見ることができます。

少なくてもSCPIかドライバかの選択が可能な場合は、ドライバの主な利点として以下があります。

1 プログラムの移植性がより高く、他のプログラマが理解しやすくなります。SCPI構文は広く理解されていますが、正確にどのような文字列を書くかは、測定器のマニュアルを見なければほとんど分かりません。また、ドライバは最終的にSCPIコマンドを生成するので、アプリケーションのデバッグを行う際に結局はコマンドを勉強する必要があります。

2. IntelliSenseにより、開発時に内蔵ヘルプを参照できます（VB6と.NET）。
3. 実装されている場合は、内蔵のステート・キャッシュにより実行速度が向上します（次のセクションで詳述）。
4. 測定器の特殊性が考慮されている場合があり、サポートの必要性が減少します。

ドライバの欠点は、前述のPXIの場合を除き、すべての機能がサポートされていない場合があることです。また、ドライバが正しく機能しない場合は、デバッグが困難なことがあります。Agilent T&M Toolkitが提供するI/O Monitorを使用すると、実際に測定器に送られるSCPIコマンドを見ることができるので、デバッグが容易になります。しかしそのためには、SCPIを理解できることが必要ですが、その必要がないからドライバを選択しているはずです。以上は、サードパーティではなく測定器のメーカが提供しサポートするドライバを使う理由でもあります。

インタフェース・ドライバ

プログラムが測定器にコマンドを渡すためには、LAN、USB、GPIB、MXI-3、FireWireなどのハードウェア・インタフェースを制御するためのドライバが必要です。このようなハードウェアを制御する下位レベルのドライバは、メーカのI/Oライブラリの一部として提供されます。NI社はそのようなインタフェース・ドライバをPassportドライバと呼び、AgilentはTulipドライバと呼んでいます。これらを直接使用する理由はないので、本書では詳細は省略します。しかしその存在は、特にNI社とAgilentのハードウェア／ソフトウェアを同じ環境内で使用する場合は、知っておくと便利です。NI社のソフトウェアがAgilentのインタフェース・ハードウェアに、またAgilentのソフトウェアがNI社のインタフェース・ハードウェアにアクセスすることがあるからです。これに関しては、Agilent I/Oライブラリのオンライン資料に詳述されています。

VISA、VXIplug&play、IVI

ソフトウェア・ドライバは、レイヤから構成されています。そのうち、もっともシンプルで標準化されたレイヤは、VISA（Virtual Instrument Software Architecture）と呼ばれています。（VISAは標準化され過ぎているので真のドライバと呼べないかもしれませんが、ここではそれを論じません。）これにより、viReadやviWriteなどのシンプルなC関数コールを使って、SCPIコマンドを測定器に送ることができます。また、同様のC関数コールviPeekやviPokeを使ってバイナリ・コマンドを測定器に送ることができます。AgilentとNI社は、このソフトウェア・ドライバを提供する主要な2つのメーカです。AgilentはSICL（Standard Instrument Control Library）というライブラリも提供しています。これはVISA以前のもので、Agilentが提供するVISAのベースとなるプログラムとして使われています。Agilentは業界標準のVISA-COMアーキテクチャも採用しており、Visual StudioなどのCOM環境のためにVISA C関数コールを提供しています。（COMはComponent Object Modelの頭文字。ライブラリをカプセル化するMicrosoftの仕様で、ライブラリ間のインタフェースや保守を容易にします。）

開発者が前述のviPeekやviPokeコマンドを使用しない限り、多くのVXI機器やすべてのPXI機器を含むレジスタ・ベースの機器はドライバが必要です。これらのCコマンドは冗長なプロセスなので、それへの対処としてVISAの上位レイヤとしてVXIplug&playが策定されました。名前にはVXIが入っていますが、そのコンセプトは多くのスタンドアローンのプログラマブル測定器を含め、VXIでない測定器にも拡張されています。

VXIplug&playドライバは測定器固有のもので、測定器の名前が関数名の中で使われています。代表的なCの例を以下に示します。

```
hp34401_read_Q (vi, readings, numReadings);
```

（多くのAgilentドライバは、AgilentがHewlett-Packardの一部であった時代に書かれました。したがって、下位互換性のためにドライバ名は変更されていません。）

同様の機能を持つ製品が多いので、VXIplug&play標準に起因する関数の無用な重複が生じました。これへの対処として、測定器をDMM、オシロスコープ、ファンクション・ジェネレータ、スイッチなどのクラスにまとめて、それらのクラス用のドライバがIVI（Interchangeable Virtual Instruments）と名付けられました。これらのドライバの定義は、IVI Foundationという組織によって仕様化されています。IVIには以下の2つの形式があります。

- IVI-C：主にNI社によって提供され、言語（C、LabVIEWなど）に応じて異なるバージョンが必要です。代表的なCの例：

  IviDmm_ReadMultipoint (vi, maxTime, arraySize, readingArray, &actualPoints);

- IVI-COM：主にAgilentによって提供され、環境（VEE、LabVIEW、Visual Studioなどを含む、Microsoftのすべての COM環境）ごとに異なるバージョンが必要ないので、多くの開発環境に適用できます。代表的なCの例：

  myDMM.Measurement.ReadMultiPoint (maxTime, readings)

いずれの場合も、測定器固有の関数コールを使用できます。これにより、クラス固有の関数が測定器の持つ機能に対応していない場合でも対処できます。このような機能はIVIの「互換性」を損なう場合がありますが、他にはない機能を持つことは、その測定器を選択する上での理由でもあります。測定器の互換性が必要で、それに応じたプログラムを書くことは、ユーザが判断する必要があります。

フロントパネルを持たない測定器用のVXIplug&play/IVIドライバには、ソフト・フロントパネルを作成するプログラムが付属しており、スタンドアローンの実行プログラムとなっています。これをユーザが作成した測定器制御プログラムに組み込むことはできません。しかし、プログラムの断片を作成して、開発環境にカットアンドペーストできるものがあります。

VXIplug&playドライバには、Agilent VEE ProおよびLabVIEWからのC関数コールを簡素化する関数パネルも付属しています。

LabVIEWドライバとVEE Proドライバ

LabVIEWは、NI社に固有の特殊なグラフィック・ドライバ（正式にはG Framework VXIplug&playドライバと呼ばれます）を使用しています。またGドライバ、G-Winドライバ、LabVIEWドライバ、LabVIEW認定プラグアンドプレイ・ドライバと呼ばれることもあります。しかし、LabVIEWでは、Windowsフレームワーク（Cベース）のVXIplug&playドライバも使用できます。そのため、ダウンロードする場合に、実際にオープンして中身を見るまでどちらか分からないことがあります。

AgilentのVEE Proも、VXIplug&play（Windowsフレームワーク）、IVI-C、IVI-COMの各ドライバを含め、様々なドライバを使用できます。過去には、Agilentは「パネル」ドライバというVEE Proに固有の特殊なタイプのドライバを開発したことがありました。今日では、Agilentはすべての測定器で、COMおよび.NETフレンドリなドライバに標準化しているので、ユーザは多くの開発環境を選択できるようになっています。

速度について

本書のベンチマーク・テストに使用したハードウェアでは、VXIplug&playおよびIVIの各ドライバは、今日のコンピュータ上ではSCPI（VISA上）と速度の点で遜色ありませんでした。これに関して異論もあるかもしれませんが、実行速度だけの理由でドライバを避ける理由はありません。

いずれの方法を採るにせよ、測定器のプログラミングにはより大きな問題があります。それは、コマンドのシーケンシングです。SCPIを使う場合は、他の測定器ステートを変化させるコマンドがあることを理解する必要があります。多くの問い合わせコマンドを送るという犠牲を払って、ドライバは、測定器がどのようなステートにあるかを知る必要があります。また、測定器をあるステートにするためのコマンドを送ることもあります。これにより、実行時間がかなり増加します。この問題を解決するのがステート・キャッシングで、これは通常はIVIドライバに実装されており、その気になればユーザ自身で実装することも可能です。ステート・キャッシング・ドライバは、測定器の現在のステートを常に追跡しています。そして、ステートを変化させないコマンドが来れば、そのコマンドを測定器には送りません。これにより、コマンドの転送時間と解釈時間を節約することができます。

ステートの問題は、一般に測定器にリセット・コマンドを送って、測定器を既知のステートに回復すれば解決できます。しかし、測定器のリセット・コマンドの実行にはかなりの時間がかかるので、最初にプログラムを実行するときに1度だけリセット・コマンドを送るのが良い方法です。その後は、プログラムの進行に従ってステートを注意深く管理して、プログラムの終了時に、測定器をリセット・コマンドと等価なステートにします。また、エラー時にリセット・コマンドを送るのも一般的な方法です。

実行時間を増加させる冗長なコマンドの例をここで紹介します。SCPIでプログラムして、コマンド“CONF:VOLT DC 10,.001”を送ったとします。これはDMMをDCファンクションに設定し、レンジを10 V、分解能を1 mVに設定します。GPIBの最大速度である1 MB/sで、この21文字（復帰改行文字を含む）の文字列を転送するのに21 μsかかります。さらに、関数コールのオーバヘッドとして、今日のPCでは10 μsほどかかります。測定器はそれ自身のかなり遅めのプロセッサを使用して、このコマンドを解釈しなければなりません。例えば34401A DMMでは、12 MHzプロセッサで約21 msがかかります。DMMがすでにそのステートにあった場合は、21.031 msもの不必要な時間が消費されません。これはたいした時間でないように思えますが、代表的なテスト・プログラムでは何百、何千回というDMMの読取りを行うことがあります。そのためI/Oと、それに関連する測定器内での処理を効率化することが重要となります。ステート・キャッシングが実装されたIVIドライバでは、測定器がすでにそのステートにあることを知っているので、わざわざコマンドを送りません。プログラミング時間に余裕があれば、ユーザは自分でステート・キャッシングを実装することも可能です。またこの例は、少量のデータ転送に対しては、LANやUSBはGPIBよりも特に高速でないことを示しています。どの場合でもI/O転送速度は測定器内の処理時間よりも非常に速いので、インタフェースの選択は大きな意味を持ちません。

Agilent 34980A（下図）は、きわめて高速な内部プロセッサを搭載した新しいクラスの測定器です。本機は前述の文字列を約1.5 msで実行することができます。そのため、ステート・キャッシングの必要性が減少します。

共通の開発環境

本書が取り上げた様々な測定器をベンチマークする上で、同種の測定器を比較検討するためには共通の開発環境が必要でした。LabVIEWとVEE Proは人気のあるグラフィック環境ですが、多くの製造テストはテキスト環境で行われています。この分野では、VB6とC、およびその派生言語（VC++、LabWindows/CVI）が多く利用されています。しかし、MicrosoftのVisual Studio.NET環境を使用して新たなアプリケーションを作成するための多くの理由があります。AgilentもNI社も、この環境でテスト・システム開発者がプログラムを容易に作成するためのツールを発表しています（AgilentのT&M Toolkit、NI社のMeasurement Studio）。Microsoftによる古いプログラムのサポートが終了するのにともない、開発者は.NETへのアップグレードが不可欠となるでしょう。.NETに付属の移植ツールを使えば、VB6プログラムもVB.NET環境に移植することが可能です。これらの理由によって、共通の開発環境としてVB.NETを選択しました。

また、NI社、Keithley社、Racal社、およびAgilentの各ハードウェアが、実際のテスト・システム内でともに動作することを示すことが必要でした。そのために、これらの全製品を使用できるように図1（7ページ）のようなテスト・システムを構築しました。Agilent 34980AとKeithley 2701は、LANハブを介してLAN接続を行いました。PXIケージをPCに接続するために、MXI-3を使用しました。VXIケージとPCの接続には、FireWireを使用しました。VXIケージはGPIBコマンド・モジュールによっても制御できるので、専用のPCI GPIBカードを使用してこれを行いました。その他の機器は、Agilent 82357A USB/GPIBコンバータを使用して、GPIBで制御しました。

図1は、テストした各測定器の相対的な高さも示しています。これらは、34401A DMMの2 EIA単位から（1 EIA単位＝1「ラック単位」は4.4 cm）、“3U”PXI/SCXIカードケージの4単位までにわたっています（PXIカードは3Uですが、それを収容するカードケージは4Uです）。VXIラック・サイズの比較は行いませんでした。便宜上4スロットのフレーム（3U）を使用しましたが、大規模なテスト・システムでは13スロットのフレーム（7U）が使用されます。今日のシステムはVXIでないハードウェアを使用して構築される傾向がありますが、VXIだけが提供する機能を使用するための一時的なカードの使用では、4スロットのフレームで十分です。

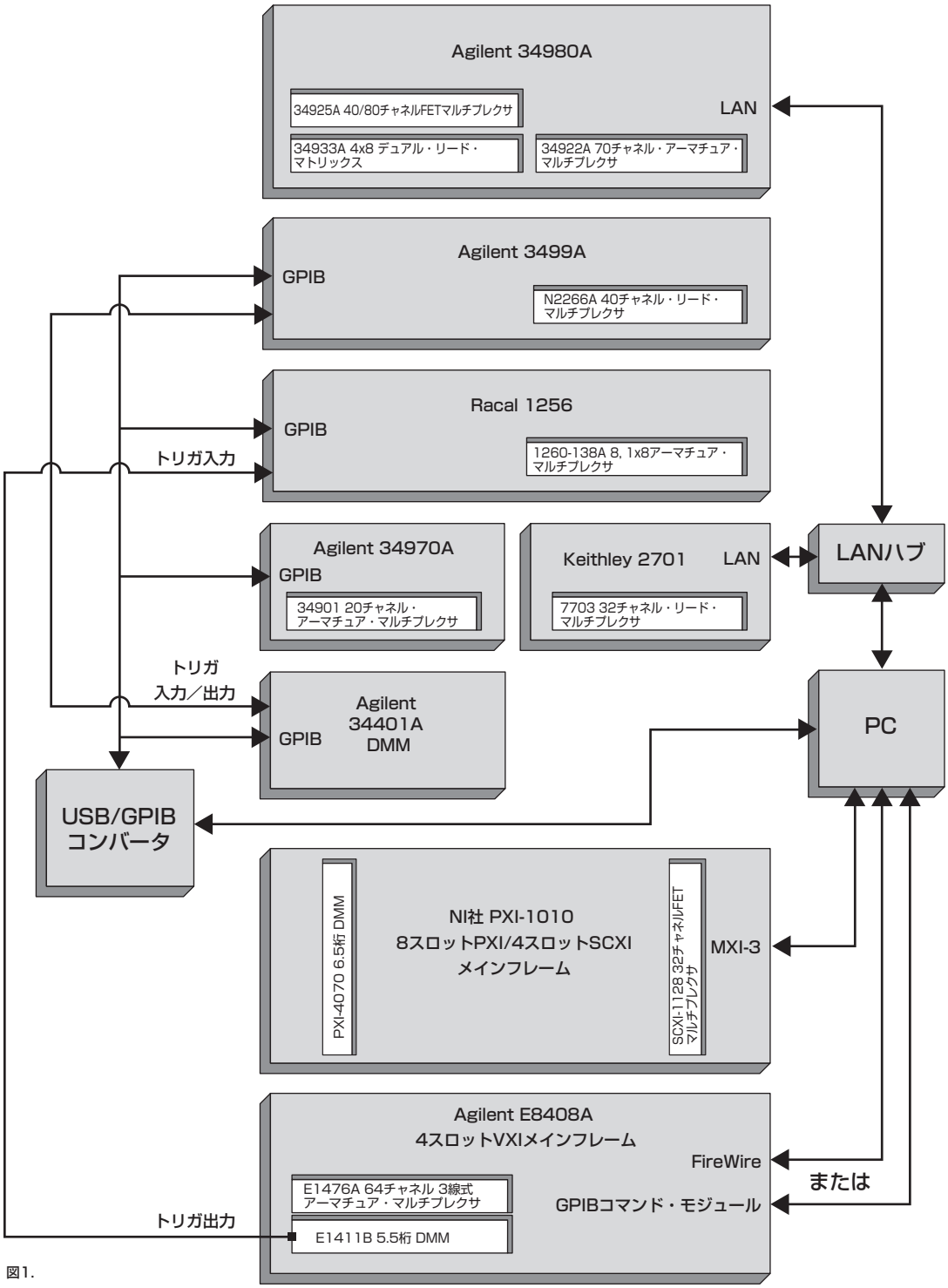

Agilent 34980A
34925A 40/80チャネルFETマルチプレクサ
34933A 4x8 デュアル・リード・マトリックス
34922A 70チャネル・アーマチュア・マルチプレクサ
LAN
Agilent 3499A
GPIB
N2266A 40チャネル・リード・マルチプレクサ
Racal 1256
GPIB
1260-138A 8, 1x8アーマチュア・マルチプレクサ
トリガ入力
Agilent 34970A
GPIB
34901 20チャネル・アーマチュア・マルチプレクサ
Keithley 2701
LAN
7703 32チャネル・リード・マルチプレクサ
LANハブ
トリガ
入力／出力
Agilent
34401A
DMM
GPIB
PC
USB/GPIB
コンバータ
PXI-4070 6.5桁 DMM
NI社 PXI-1010
8スロットPXI/4スロットSCXI
メインフレーム
SCXI-1128 32チャネルFET
マルチプレクサ
MXI-3
Agilent E8408A
4スロットVXIメインフレーム
FireWire
または
E1476A 64チャネル 3線式
アーマチュア・マルチプレクサ
GPIBコマンド・モジュール
トリガ出力
E1411B 5.5桁 DMM

図1.

ベンチマーク結果

プログラムはVB.NETで作成し、このプログラムでデータ収集（スキャンによる電圧測定）とファンクション･テスト（単一の測定と対応するスイッチング）の両方のすべての測定器を制御しました。Compaq Evo（2.4GHz Pentium 4、512MB RAM）には以下のソフトウェアをインストールしました。

- Agilent T&M Toolkit 1.2
- 上記のAgilentソフトウェアは、SICLおよびAgilent VISAを含むI/Oライブラリをインストールします。また、VXIplug&playドライバを持つすべてのAgilent測定器（E1411B VXI DMM、E1476A VXIスイッチなど）用の.NETラッパーをインストールしました。
- E1411B DMMおよびE1476Aスイッチ用のVXIplug&playドライバは、ADN（Agilent Developer Network）Webサイトからダウンロードしました。
- E1411B DMMおよびE1476Aスイッチ用のD-SCPIドライバは、ADN（Agilent Developer Network）Webサイトからダウンロードしました。
- 34401A DMM、3499Aスイッチ・ユニット、34970Aスイッチ／測定ユニット用のIVI-COMドライバは、ADN（Agilent Developer Network）Webサイトからダウンロードしました。
- D-SCPIコールとの併用のためのGPIBコマンド・モジュールを初期化するためのHP VIC（名前は、HPとAgilentに分割後も変更されていません）は、ADN（Agilent Developer Network）Webサイトからダウンロードしました。
- NI社 Measurement Studio.NET 7.2
- 2701スイッチ／測定ユニット用のIVI-Cドライバは、Keithley社のWebサイトからダウンロードしました。
- 1256スイッチ・ユニット用のIVI-Cドライバは、RacalのWebサイトからダウンロードしました。
- NI-SWITCH、NI-DAQ、NI-DAQmx、NI-DMMの各IVI-Cドライバは、NI社ドライバ・インストール・ディスクからインストールしました。.NET互換コールの作成のために、Toolkit Driver Wrapper Wizardを実行しました。

ベンチマークのタイミングの結果：

EFT（ファンクション･テスト）時間＝1つのリレーをオープンしてクローズし、1つのDMMをトリガしてそのDCV測定値を読み取る時間（レンジ10 V、分解能1 mV、4.5桁）。レポートされる時間は20回の測定の平均値です。

DAQ（データ収集）時間＝20チャネルのリストをスキャンして読み取る時間。各リレーをクローズするときに測定を実行します。

	ドライバ		SCPI	
	EFT	DAQ	EFT	DAQ
Agilent 34980A				
70チャネル・アーマチュア・マルチプレクサ（34922A）	16.9 ms	10.4 ms	15.1 ms	10.1 ms
デュアル4x8リード・マトリックス（34933A）	9.6 ms	*1*	8.4 ms	*1*
40/80チャネルFETマルチプレクサ（34925A）	10.0 ms	2.7 ms	7.9 ms	2.7 ms
Keithley 2701-7703				
32チャネル差動リード・マルチプレクサ	440 ms	68 ms	—	—
Racal 1256-138A/E1411B				
8 1x8 2線式アーマチュア・マルチプレクサ	4.13 ms	113 ms[2]	—	—
Agilent 34970A-34901A				
20チャネルアーマチュア・マルチプレクサ	52.2 ms	22.8 ms	70.0 ms	25.2 ms
Agilent 3499A-N2266A/34401A				
40チャネル・リード・マルチプレクサ	29.5 ms	21.4 ms	35.3 ms	27.1 ms
Agilent VXI E1476A/E1411B				
64チャネル3線式リード・マルチプレクサ	30.1 ms	11.9 ms	46.6 ms[3]	2.94 ms[3]
NI社 SCXI-1128/PXI-4070				
32チャネルFETマルチプレクサ	12.8 ms	2.91 ms[4]	不可	不可

注記：

1　マトリックスを使ってスキャン測定は実行できません。これはマトリックスの一般的な使用法ではなく、マトリックスはほとんどの場合にEFTテストのために使用されます。

2　Racal 1256は「スキャン・リスト定義」コマンドの処理に2sかかりました。これは、データ収集時間が非常に長いためです。それがなければ、実行時間は約13 msとなったでしょう。そのうち10 msは「トリガ遅延」パラメータによるもので、これは高度なトリガ出力が動作しなかったからです。この遅延がなければ実行時間は約3 msとなり、通常のDAQモードに対応します。

3　VXI SCPIの測定値は、E1406A GPIBコマンド・モジュールを使用して収集しました。

4　SCXI-1128はDMMにトリガを返すことができず、完全なハンドシェークはできませんでした。したがって、SCPI測定は同期タイプであり、DMMはサンプリング間隔で20回の測定を行い、スキャンを次のチャネルに進めるためにその「測定完了」信号を使用しました。そのため、実行時間はDMMのサンプリング間隔に依存します。レポートされる値は0.0のサンプリング間隔での値で、オーバヘッドを表しています。サンプリング間隔が増加するに従って、レポートされる値はサンプリング間隔と等しくなります。

不可 PXI機器はドライバが必要です。

このデータには、いくつかの興味深い点があります。前述のようにデータ収集（DAQ）は非常に高速な結果を示し、1桁違うこともあります。また、実行時間にかなりの差が表れています。例えば、Keithleyで高速リード・リレーを使用してもあまり改善はありません。これは、この測定器が読取りを行うのが非常に遅いためです。PXIは最高速でしたが、この測定はFETスイッチのみを使用して行いました。また、その性能は34980A FETスイッチと同程度のものでした。リード・リレーやアーマチュア・リレーを使用していたら、それらのスイッチング時間によって実行時間は増加していたでしょう。また、ドライバの実装方法によっては、ステート・キャッシングを使用するとSCPIの場合よりも高速になることが考えられます。

製品の幅

以下の表は、各プラットフォームで使用可能なモジュールの種類を示しています。

	Agilent 34980A	Agilent 34970A	Agilent 3499A	Agilent VXI	NI社 PXI/SCXI	Keithley 2701	Racal 1256
マトリックス・リレー							
FET	なし	なし	なし	なし	4x6 2w 4x8 2w	なし	なし
リード	2, 4x8 2w	なし	なし	4x32 2w	4x16 2w 4x32 2w 4x64 2w	なし	なし
アーマチュア	2, 4x8 2w 2, 4x16 2w	4x8 2w	4x4 2w 4x8 2w	16x16 4x16 8x32 8x8 4x16 2w	4x6 2w 8x16 2w	6x8 2w	12x12 2w （7構成）
マルチプレクサ・リレー							
FET	40チャネル 2w	なし	8チャネル 2, 4チャネル 4, 2チャネル se	16チャネル 3w 32チャネル 16チャネル 3w	24チャネル 2w 32チャネル 2w	20チャネル	なし
リード	40チャネル 2w	20チャネル 2w	40チャネル se	16チャネル 3w 48チャネル se 64チャネル 3w 256チャネル	64チャネル 2w 128 2w 256 2w	32チャネル 2w	42チャネル （ドライ／ 水銀接点）
アーマチュア	40チャネル 2w 27チャネル 2w	20チャネル 2w 40チャネル 1w	10チャネル 2w 20チャネル 2w 40チャネル 2w	64チャネル 2w		20チャネル 2w 32チャネル 2w 40チャネル 2w	23チャネル 2w 16, 1x4 8, 8チャネル 2w
RF	4, 1x4 50 Ω/75 Ω	2, 1x4 50 Ω/75 Ω	2, 1x4 2, 1x6 1x9 50 Ω/75 Ω	2, 1x4 6, 1x4 50 Ω/75 Ω	4, 1x4 8, 1x4 1x16 1x32 50 Ω	2, 1x4	10, 1x4 2, 1x4 1x6 2, 1x6
GPリレー							
アーマチュア	28-C/4-A, 20チャネル A	20チャネル C	7/8/10/ 20チャネル C, 40チャネル A	16チャネル C 32チャネル C 40チャネル A 64チャネル A	8/16/32/40 C 16/31/100 A	40チャネル A	12 A 12 B 12 C 20 DPDT 52 C 80 A, ロー／ ハイ・パワー・ バージョン
RF	2チャネル SPDT, 3チャネル SPDT		3チャネル SPDT, リレー・ドライバ	18 GHz, 3チャネル SPDT, リレー・ドライバ	32, 64, リレー・ ドライバ	なし	20チャネル SPDT 50 Ω/75 Ω 2 SPDT 5 SPDT 2x2 XFERスイッチ

10ページに続く

略語：
A＝Form A
B＝Form B
C＝Form C
1w＝1線式
2w＝2線式

製品の幅（続き）

	Agilent 34980A	Agilent 34970A	Agilent 3499A	Agilent VXI	NI社 PXI/SCXI	Keithley 2701	Racal 1256
DMM	6.5桁内蔵	6.5桁内蔵	外部	5.5桁/6.5桁	5.5桁/6.5桁	6.5桁内蔵	外部
デジタルI/O（DIO）	64チャネル	マルチファンクション・カード上	16ビット, 32ビットTTL	4, 8ビット 72チャネル 出力 96チャネル DIO 64チャネル ISO入力	11種類のカード	マルチファンクション・カード上	48 oc, 96（ttl, cmos, oc）
DAC	4チャネル ISO波形	マルチファンクション・カード上	マルチファンクション・カード上	4チャネル, 8/16チャネル		マルチファンクション・カード上	なし
カウンタ／トータライザ	DIOカード上	マルチファンクション・カード上	マルチファンクション・カード上	E1333A：2チャネル @100 MHz+ 1チャネル @1 GHz	4/8チャネル ～ 125 MHz	マルチファンクション・カード上	なし
マルチファンクション	32 DIO, 2±12V DAC 100 KHz トータライザ	16ビットDIO, 26ビット・カウンタ, 2, 16ビットDAC	15 GP/16 DIO 4x4マトリックス /16 DIO 2 DAC/16 DIO	Mモジュール	ディジタイザ, カウンタ, DIO, DAC	20チャネル2線式マルチプレクサ, 2 DAC, 16 DIO, 1カウンタ 10チャネル2線式マルチプレクサ, 32 DIO	なし
アナログ・バス	内部4バス 2線式	なし	なし	なし	PXI：なし SCXI：3バス 2線式	DMM ハイ／ロー	DMM ハイ／ロー
その他	ブレッドボード	なし	なし	オシロスコープ、任意波形発生器など	オシロスコープ、ディジタイザ、任意波形発生器など	なし	ブレッドボード

コスト

いくつかのスイッチ／測定システムの価格を右の表に示します。DMMとスイッチボックスが別途必要なものは省きました。PXI/VXIでは高価なカードケージを買わなければならず、またこれらは高速測定を目的としています。また、コンピュータ側とカードケージ側でインタフェースも必要になります。これは、使用する各スロットに加えて余分なコストとなります。さらに、別々のカードを使用するシステムを構築するのは、スイッチ／測定ボックスを使用する場合に比べてかなり時間がかかる場合があります。これについては、次のセクションでより詳細に検討します。VB.NETを使用することにより、34980Aおよび34970Aスイッチ／測定ユニットでスキャン測定を実装するのに数分間しかかかりませんでした。それに対して、PXIおよびSCXIを使用して同様の測定を行うのには、2週間と、サポート電話を何回もかける必要がありました。

測定器	定価（米国ドル、2004年9月1日現在）		
Agilent 34980A	$2350	DMM、LAN、USB、GPIB、8スロット含む	**（スロットあたり$294）**
Agilent 34970A	$1477	DMM、GPIB、RS232、3スロット含む	**（スロットあたり$492）**
Keithley 2750	$2995	DMM、GPIB、RS232、5スロット含む	**（スロットあたり$600）**
NI社 PXI	$5485	PXI-1042 8スロット・ケージ（$1995）、6スロット使用可能 PXI-4070 DMM（$1995）（1スロット使用） MXI-4インタフェース（PXI-PCI8331）（$1495）（1スロット使用）	**（スロットあたり$914）**

Agilent 34980A用のカードは$495～$2000
Agilent 34970A用のカードは$340～$501
Keithley 2701/2750用のカードは$445～$995、1台のマイクロ波モジュールは$1995
NI社のPXI用スイッチ・カードは$495～$1995、1枚の高密度カードは$4795

34980Aは他の様々なソリューションと比べても多くのカードを収容でき、高周波用を含めて様々なカードを使用でき、コンピュータ・インタフェースの選択の幅が広く、スロットあたりの価格がもっとも安価です。また、多くのプログラミング環境にも対応しています（LabVIEW/IVI-COMドライバを使用）。

評価中に遭遇した問題

3週間にわたった評価期間の間に、様々な問題に遭遇しました。その概要をここに示します。

1. **Keithleyではファームウェア・アップグレードが必要。** Keithley 2701で、プログラムが中断したり「クローズ」を実行しないで終了すると、Keithley 2701は電源を入れ直すまで使えませんでした。Keithley WebサイトのFAQを見ると、ファームウェア・バージョンA06に関する項目がありました。A06ではコマンド“KI2701”を送るためにセカンド・ポート(2701)をオープンでき、これによってポート1394がリセットされて再び使用可能になるということです。テストに使用したユニットのバージョンはA04だったため、KeithleyのWebサイトからA06をダウンロードし、ユニットを再フラッシュしました。この実行には問題はなく、上述の問題も解決しました。

2. **異常な動作。** Keithley 2701は、ROUT:MULT:CLOS(マルチ・リレーのクローズ)コマンドを送ると、DMMアパーチャ・タイムがSLOW(5 PLC)にリセットされることが分かりました。このため、アパーチャ・タイムを0.01 NPLCにセットし直すコマンドを追加する必要がありました。これは、DAQモードでConfigureステートメントが分解能を設定している場合でも同様でした。他の測定器でこのような動作をするものはなく、読取りがなぜこれほど遅いのか理解するのにしばらくかかりました。

3. **NI-VISAのセットアップ変更が必要。** NI社のVISAを使用する場合は、MAX(Measurement Automation Explorer)も同時に実行して“Passport to Tulip”インタフェース・ドライバをイネーブルする必要があります。これは、Tools->NI-VISA->VISA Options->Passportsにあります。これをイネーブルしないと、AgilentのVXI/GPIB機器が認識されません。しかし、NI Measurement Studioをインストールしていると、このインタフェースが使用できなくなります。これは、NI Passportインタフェース・ドライバがAgVisa32.dllをコールし、その終了時に例外を発生させることが原因でした。NI社に問い合わせてから、解決するまでに2週間かかりました。NI社のTulipパスポート・ドライバの作成者によると、これは既知の問題だったそうです。解決法として、NI VisaTulip.dllのパッチ・バージョンをインストールしました。問題は解決されましたが、最新のNI社のインストール・ディスクでもまだ古いバージョンが収録されてたままになっています。

4. **NI IVIの適合性の問題。** AgilentのDriver Wrapper Wizardでは、IVI-CとVXIplug&playの各ドライバのみがラップ可能で、それらはVI Foundation仕様に基づいて適切にインストールされている必要があります。NI-DMMドライバと、おそらく他のNI IVI-Cドライバは、インストール時に“LabWindows/CVI examples”ボックスにチェックしていないと適切にインストールされませんでした。本書の評価時では、NI社はその測定器の.NET互換性を提供していませんでしたが、Webサイトではダウンロードして手動でプログラムに追加できる.NETラップ・コードを提供していました。NI-DMMドライバについてはこれを行いました。しかし、NI社はそのラッパーで“InstrumentDriverInterop”のネームスペースを選択していましたが、これがAgilent Toolkitが作成したものとコンフリクトを起こしました。NIネームスペースを“InstDvrInterop”にリネームすると、このコンフリクトは解決しました。もう1つの方法は、ショートカットを使用せずに、使用の度にネームスペースを明示的に指定することでした。例えば、プログラムの最初に“Imports Agilent.TMFramework”を指定する場合は、そのフレームワーク内のサブカテゴリはすべてプリフィックスなしで使用できます。例えば、そのようなカテゴリにInstrumentDriverInteropがあり、

   Agilent.TMFramework.InstrumentDriverInterop.xxx

   と指定することも、単に

   InstrumentDriverInterop.xxx

   と指定することも可能です(xxxは次のレベルの機能)。しかし、NIラップされたドライバを同じネームスペースで付加すると、コンフリクトは解決されるはずです。

5. **不完全なマニュアル。** PXI-4070 DMMを使用するためのNI社のサンプルは、VB6、VC++、LabVIEWで書かれています。.NETのDAQmxスイッチのサンプルはソフトウェア・トリガだけで、ハードウェア・トリガは含まれていません。ハードウェア・トリガはバックプレーン・トリガ・バスの使用が必要ですが、これを使用するパラメータは文字列が必要です。しかし、どういう文字列を使用するかのサンプルがありません。IntelliSenseヘルプのポップアップはjavascriptを実行してヘルプを表示しようとしますが、動作しませんでした。目的のヘルプを検索するには、Start->Programs->NationalInstruments->NI-DAQ->DAQmx Documentationを手動で実行する必要がありました。しかし、トリガ・バス用のネームが必要なプログラム内で使用しようとしても、動作するネームがありませんでした。何回もエラーが出ました。PXITRig0、LBR_Trig0、TTL0、NI_VAL_TRIG_TTL0など、様々なネームを試みました。

次に、LabVIEWサンプルをロードしてどのようなネームが使われているかを調べました。これにより、新しいネーミング規約が必要なことが分かりました。これは、例えば“/SC1Mod3/TrigIn”など、すべてパス・ベースでした。しかしLabVIEWサンプルのネームにも、動作するものがありませんでした。今回もサポートに問い合わせて、結局返ってきた答はSCXI-1128はハンドシェークをサポートせず、同期モード（片方向トリガ）だけということでした。また、より新しいカードを使うか、従来のDAQを使ってくださいということでした。事前にどうしてそれを知ることができるのかと聞いたところ、次のような答が返ってきました。「残念ながら、トリガの制限の問題は購入の前に誰かに尋ねるか、ヘルプ・マニュアルを十分に読んでいないと分からない難しい問題です。」

6. **PXIでカードを入れ替えるのにPCの電源オフが必要。**PXIカードケージ内でカードを移動するには電源を入れ直す必要があり、PCの電源がオンのままでは行えません。PXIバックプレーンは、コンピュータのPCIの延長だからです。したがってPCのリブートが必要であり、時間がかかります。

7. **バージョン間のコンフリクト。**バージョン間のコンフリクトが何度も発生しました。例えば、SCXIスイッチングの問題をトラブルシュートしているときに、Measurement Studio 7.0上にNI-DAQ 7.1をインストールしていました。これにより多くの問題が発生し、解決するにはNI社のソフトウェアをアンインストールしてインストールし直す必要がありました。これに1日の大半を費やしました。

8. **AgilentとNI社のハードウェアを同じプログラムから制御する方法は？** NI MAXはすべてのNIインタフェース上のデバイスを認識できますが、VXI FireWireインタフェース、USB/GPIBコンバータ、PCI GPIBカードなどのAgilentのインタフェースは直接制御できません。またAgilentのInstrument Explorerは、現在のところPXIデバイスを認識できません。両方のメーカのデバイス間の通信が必要なテスト・システムで、この問題をどのようにして最もうまく解決できるでしょうか？ その答は、Agilent I/Oライブラリを“side-by-side”モード（I/Oライブラリのヘルプ・ファイルで説明）でインストールすることでした。その上で、前述のNI MAXのPassport-Tulipインタフェース・ドライバをイネーブルします。これで、これらのインタフェースに対するVISAコールはNI VISAからAgilent VISAを経由することになり、関連するAgilentインタフェースの制御が可能になります。この場合も、MXI-3などのNIインタフェースは、NI VISAドライバおよびPassportドライバを通して直接動作させることが可能です。

サンプル・プログラム

ここでは、DCV、レンジ10 Vに設定したDMMと20チャネル・マルチプレクサを使用して、簡単なEFT（クローズ／測定／オープン）測定およびDAQ（スキャン）測定を行います。そのためのVisual Basic.NET環境でのプログラミング要件を示します。

プログラムを見るうえで､以下の背景を考慮してください。

1. DMMを内蔵するスイッチ／測定ユニットは、DAQ測定の他に多くの作業を行います。これは、トリガ機能を提供しなければならないからです。これらの測定器に対して動作するプログラムを作るのに、数分しかかかりませんでした。これに対して、PXI/SCXI測定が動作するのに2週間かかりました。これは主に、トリガのための要件を理解するのが困難だったからです。

2. SCPI文字列は連結によって、長い文字列にすることができます。複数の関数コールによる余分なオーバヘッドがなくなるので、実行時間を少し短縮することが可能です。しかし前述のように、もともと文字列を送る必要がなければ、低速の測定器でコマンドを解釈する時間が少なくなり、関数コールの実行を高速化できます。アプリケーションで高スループットが必要な場合は、ステート・キャッシング・ドライバとSCPIとを比較検討する必要があるでしょう。

3. IVI-C（.NETラッパー付き）、IVI-COM、VXIplug&playの各ドライバは、.NET環境での使用法が非常によく似ています。これらすべてに対して、様々なレベルのIntelliSenseヘルプが提供されています。以下のプログラムでは明らかではありませんが、IVI-COMのヘルプは他の2つに比べて非常に分かりやすくなっています。PXI/SCXIのヘルプは分かりにくく、何度もオンライン・ヘルプを参照したり、NI社との電子メールによるサポート・セッションが必要でした。

Agilent T&M Toolkitを使用したVB.NETプログラムはすべて、以下を自動的に挿入します。

```
Imports Agilent.TMFramework
Imports Agilent.TMFramework.DataAnalysis
Imports Agilent.TMFramework.DataVisualization
Imports Agilent.TMFramework.InstrumentIO
Imports Agilent.TMFramework. InstrumentDriverInterop
```

NI Measurement Studioを使用する場合、以下を手動で挿入する必要があります。

```
Imports NationalInstruments
Imports niDMM_32.NI社DMMMeasurementConstants
```

Toolkitでは、宣言が自動的に“Public Class”に追加されます。

VXIplug&playドライバの宣言：

```
Dim myHpe1476 As InstrumentDriverInterop.VxipnpWrappers.Hpe1476
Dim myHpe1411 As InstrumentDriverInterop.VxipnpWrappers.Hpe1411
Dim myKe2700 As InstrumentDriverInterop.VxipnpWrappers.Ke2700
Dim myRi1256 As InstrumentDriverInterop.VxipnpWrappers.Ri1256
```

Agilent ToolkitのDriver Wrapper WizardによるIVI-Cドライバの宣言のラッピング

```
Dim myniSwitch As InstrumentDriverInterop.IviCWrappers.NiSwitch
```

ダイレクトI/O（SCPI）の宣言

```
Dim myDSCPI As InstrumentIO.DirectIO
Dim my34980 As InstrumentIO.DirectIO
Dim my34970 As InstrumentIO.DirectIO
Dim my3499 As InstrumentIO.DirectIO
Dim my34401 As InstrumentIO.DirectIO
```

IVI-COMドライバの宣言

```
Dim myAgilent34401 As Agilent.Agilent34401.Interop.Agilent34401
Dim myAgilent34970 As Agilent.Agilent34970.Interop.Agilent34970
Dim myAgilent3499 As Agilent.Agilent3499.Interop.Agilent3499
```

NI Measurement Studioの使用時に、手動で追加したIVI-Cの宣言

```
Dim PXIDMM As InstDvrInterop.Ivi.niDMM
Dim SCXI As InstDvrInterop.Ivi.niSwitch
```

測定器の初期化方法：

E1411/E1476コンボ用のVXIplug&playドライバの初期化：

DMMの別の使用法：

myHpe1411 = New InstrumentDriverInterop.VxipnpWrappers.Hpe1411("VXI0::24::INSTR", True, True)

DMM/Switchスイッチの使用法：

myHpe1411and1476 = New InstrumentDriverInterop.VxipnpWrappers.Hpe1411("VXI0::(24,25)::INSTR", True, True)

GPIBコマンド・モジュールを使用したE1411/E1476 VXI用のD-SCPIセッション：

myDSCPI = New InstrumentIO.DirectIO("GPIB1::9::3::INSTR", False)

Ke2701用のVXIplug&play（ラッパー）ドライバ・セッション：

myKe2700 = New VxipnpWrappers.Ke2700("TCPIP0::169.254.105.002::1394::SOCKET", False, False)
myKe2700.Reset()

3499用のIVI-COMドライバ・セッション：

myAgilent3499 = New Agilent.Agilent3499.Interop.Agilent3499Class
myAgilent3499.Initialize("GPIB0::9::INSTR", True, True, Nothing)
myAgilent3499.Utility.Reset()

3499用のSCPIセッション：

my3499 = New InstrumentIO.DirectIO("GPIB0::9::INSTR", False, False)
my3499.Timeout = 2000

34401用のIVI-COMドライバ・セットアップ：

myAgilent34401 = New Agilent.Agilent34401.Interop.Agilent34401Class
myAgilent34401.Initialize("GPIB0::22::INSTR", True, True, Nothing)

34401用のSCPIセッション：

my34401 = New InstrumentIO.DirectIO("GPIB0::22::INSTR", False, False)
my34401.Timeout = 2000

Racal 1256用のVXIplug&play（ラッパー）ドライバ・セッション：

myRi1256 = New InstrumentDriverInterop.VxipnpWrappers.Ri1256("GPIB0::14::INSTR", True, True)

34970用のIVI-COMドライバ・セッション：

myAgilent34970 = New Agilent.Agilent34970.Interop.Agilent34970Class
myAgilent34970.Initialize("GPIB0::8::INSTR", True, True, Nothing)

34970用のSCPIセッション：

my34970 = New InstrumentIO.DirectIO("GPIB0::8::INSTR", False, False)

34980用のIVI-COMドライバ・セッション：

myAgilent34980 = New Agilent.Agilent34980.Interop.Agilent34980Class
myAgilent34980.Initialize("TCPIP0::169.254.9.80::INSTR ", True, True, Nothing)
myAgilent34980.Reset

34980用のSCPIセッション：

my34980 = New InstrumentIO.DirectIO("TCPIP0::169.254.9.80::INSTR")
my34980.WriteLine("*RST")

PXI/SCXI用のIVI-Cドライバ・セッション:

PXIDMM = New InstDvrInterop.Ivi.niDMM("DAQ::8::INSTR", True, True)
myniSwitch = New IviCWrappers.NiSwitch("SCXI1::3", False, True)

Keithley 2701のサンプル

```
Dim index As Integer
Dim rdgArray(80) As Double
Dim reading As Double
Dim numPts As Integer
myKe2700.ConfigureAutoZeroMode(VxipnpWrappers.Ke2700.AutoZeroModeEnum.Off)
myKe2700.ConfigureMeasurement(VxipnpWrappers.Ke2700.MeasFunctionEnum.ValDcVolts, 10.0, 0.001)
```

EFTモード（クローズ／測定／オープン）：

```
For index = 1 To numChannels
    myKe2700.ConfigureSwitches( _
s101, _
VxipnpWrappers.Ke2700.SwitchModeEnum.ValCloseSingleOpenOtherChannels)
    myKe2700.ConfigureApertureTimeInfo(.01, VxipnpWrappers.Ke2700.ApertureTimeUnitEnum.Nplc)
    myKe2700.Read(5000, reading)
    myKe2700.ConfigureSwitches( _
s101, _
VxipnpWrappers.Ke2700.SwitchModeEnum.ValOpenMultiple)
Next
```

DAQモード（スキャン）：

```
myKe2700.SetChannelList(s101120)
myKe2700.ConfigureMultiPoint( _
    1, _
    numChannels, _
    myKe2700.TriggerSourceEnum.Immediate, _
    0.0)
myKe2700.ConfigureApertureTimeInfo(.01, VxipnpWrappers.Ke2700.ApertureTimeUnitEnum.Nplc)
myKe2700.ReadMultiPoint(5000, 80, rdgArray, numPts)
```

RACAL 1256/外部DMMのサンプル

```
Dim index As Integer
Dim readings(20) As Double
Dim trigDelay As Double = 0.02
myHpe1411.CalZeroAuto(False)
myHpe1411.VoltDcRang(False, _
    VxipnpWrappers.Hpe1411.RangeEnum2.VoltDcRang64V)
myHpe1411.VoltDcRes(VxipnpWrappers.Hpe1411.VoltDcResEnum.VoltRes488Micro)
myHpe1411.Trigger(1, False, trigDelay, VxipnpWrappers.Hpe1411.SourceEnum1.Immediate)
myHpe1411.Sample(1, VxipnpWrappers.Hpe1411.SourceEnum.Immediate, 0.0)
```

EFTモード（クローズ／測定／オープン）：

```
For index = 1 To numChannels
    myRi1256.OperateSingle138( _
        x.ModuleAddressEnum1._1, _
        x.OperationEnum.Close, _
        x.RelayTypeEnum.Mux0, _
        1)
myHpe1411.ReadQ(readings, 80)
myRi1256.OperateSingle138( _
        x.ModuleAddressEnum1._1, _
        x.OperationEnum.Open, _
        x.RelayTypeEnum.Mux0, _
        1)
Next
```

DAQモード（スキャン）：

```
myHpe1411.Trigger(1, False, trigDelay, VxipnpWrappers.Hpe1411.SourceEnum1.Immediate)
myHpe1411.Sample(numChannels, VxipnpWrappers.Hpe1411.SourceEnum.Timer, trigDelay)
myRi1256.ConfigOutputTrigState(VxipnpWrappers.Ri1256.OutputTrigStateEnum.Off)
myRi1256.ConfigInputTrigSource(VxipnpWrappers.Ri1256.TriggerSourceEnum1.TrigExt)
myRi1256.ArmTrigger(VxipnpWrappers.Ri1256.ArmTypeEnum.Cont)
myRi1256.DefScanList(New System.Text.StringBuilder("1(0:7,10:17,20:23)"))
myRi1256.TriggerImmediate() ' go to the first relay in the scanlist
myHpe1411.ReadQ(readings, numChannels)
```

ドライバを使用したAgilent 34980Aのサンプル

```
Dim index As Integer
Dim rdgArray(80) As Double
```

EFTモード（クローズ／測定／オープン）：

```
myAgilent34980A.Scan.ScanList = ""
myAgilent34980A.Voltage.DCVoltage.Configure("", 10.0,
Agilent.Agilent34980A.Interop.Agilent34980AResolutionEnum.Agilent34980AResolutionLeast)
myAgilent34980A.Voltage.DCVoltage.AutoZero("") =
Agilent.Agilent34980A.Interop.Agilent34980AAutoZeroEnum.Agilent34980AAutoZeroONCE
myAgilent34980A.Display.DisplayEnabled = False
myAgilent34980A.Trigger.Configure(
Agilent.Agilent34980A.Interop.Agilent34980ATriggerSourceEnum.Agilent34980ATriggerSourceImmediate, 1, 0, 1)
For index = 1 To numChannels
        myAgilent34980A.Route.Close("1001")
        myAgilent34980A.Measurement.Initiate()
        rdgArray = myAgilent34980A.Measurement.FetchNumbersOnly
        myAgilent34980A.Route.Open("1001")
Next
```

DAQモード（スキャン）：

```
myAgilent34980A.Scan.ScanList = "1001:1020"
myAgilent34980A.Voltage.DCVoltage.Configure("1001:1020", 10.0,
Agilent.Agilent34980A.Interop.Agilent34980AResolutionEnum.Agilent34980AResolutionLeast)
myAgilent34980A.Voltage.DCVoltage.AutoZero("1001:1020")
Agilent.Agilent34980A.Interop.Agilent34980AAutoZeroEnum.Agilent34980AAutoZeroONCE
myAgilent34980A.Display.DisplayEnabled = False
myAgilent34980A.Trigger.Configure(
Agilent.Agilent34980A.Interop.Agilent34980ATriggerSourceEnum.Agilent34980ATriggerSourceImmediate, 1, 0, 1)
myAgilent34980A.Route.Delay("1001:1020") = 0
myAgilent34980A.Measurement.Initiate()
rdgArray = myAgilent34980A.Measurement.FetchNumbersOnly
```

ドライバを使用したAgilent 34970Aのサンプル

```
Dim index As Integer
Dim readings(80) As Double
Dim rdgs(80) As String
myAgilent34970.Display.DisplayEnabled = False
myAgilent34970.Voltage.AutoZero(101) =
Agilent.Agilent34970.Interop.Agilent34970AutoZeroEnum.Agilent34970AutoZeroONCE
```

EFTモード（クローズ／測定／オープン）：

```
myAgilent34970.Voltage.DCVoltage.Configure("101:101", 10, 0.001)
For index = 1 To numChannels
    rdgs = myAgilent34970.Scan.Read()
Next
```

DAQモード（スキャン）：

```
myAgilent34970.Voltage.DCVoltage.Configure("101:120", 10, 0.001)
rdgs = myAgilent34970.Scan.Read()
```

ドライバを使用したAgilent 3499Aのサンプル

```
Dim index As Integer
Dim readings(80) As Double
Dim numRdgs As Integer
myAgilent34401.Advanced.AutoZero = _
        Agilent.Agilent34401.Interop.Agilent34401AutoZeroEnum.Agilent34401AutoZeroOnce
myAgilent34401.Display.Enabled = False
myAgilent34401.DCVoltage.Configure(10.0, 0.001)
```

EFTモード（クローズ／測定／オープン）：

```
myAgilent34401.Trigger.Source =
    Agilent.Agilent34401.Interop.Agilent34401TriggerSourceEnum.Agilent34401TriggerSourceImmediate
For index = 1 To numChannels
    myAgilent3499.Route.Close("@200")
    readings(0) = myAgilent34401.Measurement.Read(5000)
    myAgilent3499.Route.Open("@200")
Next
```

DAQモード（スキャン）：

```
myAgilent3499.Configure.ExtTriggerSource = 0
myAgilent3499.Configure.ExtTriggerOutput = True
myAgilent3499.Scan.ArmSource = _
    Agilent.Agilent3499.Interop.Agilent3499SourceEnum.Agilent3499SourceImmediate
myAgilent3499.Scan.ArmCount = 1
myAgilent3499.Scan.TriggerSource = _
    Agilent.Agilent3499.Interop.Agilent3499SourceEnum.Agilent3499SourceMIX
myAgilent3499.Configure.MuxFunction(2) = _
    Agilent.Agilent3499.Interop.Agilent3499WireEnum.Agilent3499WireWire2
myAgilent3499.Scan.ScanList = "@200:219"
myAgilent34401.Trigger.Source = _
    Agilent.Agilent34401.Interop.Agilent34401TriggerSourceEnum.Agilent34401TriggerSourceExternal
myAgilent34401.Trigger.MultiPoint.Count = numChannels
myAgilent34401.System.WaitForOperationComplete(5000)
myAgilent34401.Measurement.Initiate()
myAgilent3499.Scan.Initiate()
myAgilent3499.System.IO.WriteString("*TRG", True)
myAgilent3499.System.WaitForOperationComplete(5000)
myAgilent34401.Measurement.FetchMultiPoint(5000, readings)
```

ドライバを使用したAgilent E1411B/E1476Aのサンプル

```
Dim index As Integer
Dim opc As Short
Dim readings(80) As Double
myHpe1411and1476.CalZeroAuto(False)
myHpe1411and1476.VoltDcRang(False, VxipnpWrappers.Hpe1411.RangeEnum2.VoltDcRang64V)
myHpe1411and1476.VoltDcRes(VxipnpWrappers.Hpe1411.VoltDcResEnum.VoltRes488Micro)
```

EFTモード（クローズ／測定／オープン）：

```
For index = 1 To numChannels
    myHpe1476.ClosCardChan(1, 0)
    myHpe1411.ReadQ(readings, 80)
    myHpe1476.OpenCardChan(1, 0)
Next
```

DAQモード（スキャン）：

```
myHpe1411and1476.ConfigureList( VxipnpWrappers.Hpe1411.FuncEnum1.ConfListVoltDc, "100:119")
myHpe1411and1476.InitImm()
myHpe1411and1476.TimedFetchQ(5000, readings, 20)
```

NI社 PXI-4070/SCXI-1128のサンプル

```
Dim index As Integer
Dim readings(80) As Double
Dim numRdgs As Integer
Dim SampInt as Double
```

EFTモード（クローズ／測定／オープン）：

```
PXIDMM.ConfigureAutoZeroMode(NI社DMM_VAL_AUTO_ZERO_ONCE)
PXIDMM.ConfigureMeasurement(NI社DMM_VAL_DC_VOLTS, 10.0, 0.001)
PXIDMM.ConfigureTrigger(NI社DMM_VAL_IMMEDIATE, 0.0)
For index = 1 To numChannels
    myniSwitch.Connect("ch0", "com0")
    PXIDMM.Initiate()
    PXIDMM.Fetch(5000, readings(0))
    myniSwitch.Disconnect("ch0", "com0")
Next
```

DAQモード（スキャン）：

```
PXIDMM.ConfigureAutoZeroMode(NI社DMM_VAL_AUTO_ZERO_ONCE)
PXIDMM.ConfigureMeasurement(NI社DMM_VAL_DC_VOLTS, 10.0, 0.001)
PXIDMM.ConfigureMeasCompleteDest(NI社DMM_VAL_LBR_TRIG_0)
PXIDMM.ConfigureMultiPoint(1, numChannels, NI社DMM_VAL_INTERVAL, SampInt)
myniSwitch.ConfigureScanList("sc1!md3!ch0:19->com0;", _
    IviCWrappers.NiSwitch.ScanModeEnum.BreakBeforeMake)
myniSwitch.ConfigureScanTrigger(0.0, _
    IviCWrappers.NiSwitch.TriggerInputEnum.TtlO, _
    IviCWrappers.NiSwitch.ScanAdvancedOutputEnum.None)
myniSwitch.InitiateScan()
PXIDMM.Initiate()
PXIDMM.FetchMultiPoint(5000, numChannels, readings, numRdgs)
```

SCPIを使用したAgilent 34980Aのサンプル

```
Dim index As Integer
Dim readings As String
my34980.WriteLine("Rout:Scan (@)")
my34980.WriteLine("CONF:VOLT:DC 10,.001")
my34980.WriteLine("ZERO:AUTO ONCE;:DISP OFF;:TRIG:DELAY 0;:TRIG:SOUR IMM;:TRIG:COUN 1;:SAMP:COUN 1")
```

EFTモード（クローズ／測定／オープン）：

```
For index = 1 To numChannels
    my34980.WriteLine("ROUT:CLOS (@6001)")
    my34980.WriteLine("INI社T;FETC?")
    readings = my34980.Read()
    my34980.WriteLine("ROUT:OPEN (@6001)")
Next
```

DAQモード（スキャン）：

```
my34980.WriteLine("Rout:Scan (@6001:6020)")
my34980.WriteLine("Conf:volt:dc 10,.001,(@6001:6020)")
my34980.WriteLine("ZERO:AUTO ONCE;:DISP OFF;:TRIG:DELAY 0;:TRIG:SOUR IMM;:TRIG:COUN 1;:SAMP:COUN 1")
my34980.WriteLine("INI社T;FETC?")
readings = my34980.Read()
```

SCPIを使用したAgilent 34970Aのサンプル

```
Dim index As Integer
Dim Readings As String
Dim Points As Integer
Dim replyString As String
my34970.WriteLine("DISP OFF;: TRIG:COUN 1;SOUR IMM")
```

EFTモード（クローズ／測定／オープン）：

```
my34970.WriteLine("Conf:volt:dc 10,.001,(@101:101)")
For index = 1 To numChannels
    my34970.WriteLine("INI社T;FETC?")
    Readings = my34970.Read()
Next
```

DAQモード（スキャン）：

```
my34970.WriteLine("Conf:volt:dc 10,.001,(@101:120)")
my34970.WriteLine("INI社T;FETC?")
Readings = my34970.Read()
```

SCPIを使用したAgilent 3499A/34401Aのサンプル

```
Dim index As Integer
Dim Readings As String
Dim dummy As String
my34401.WriteLine("DISP OFF; :ZERO:AUTO ONCE;:CONF:VOLT:DC 10,.001")
```

EFTモード（クローズ／測定／オープン）：

```
my34401.WriteLine("TRIG:SOUR IMM;COUN 1;:SAMP:COUN 1")
For index = 1 To numChannels
    my3499.WriteLine("ROUT:CLOS (@200)")
    my34401.WriteLine("INI社T;FETC?")
    Readings = my34401.Read()
    my3499.WriteLine("ROUT:OPEN (@200)")
Next
```

DAQモード（スキャン）：

```
my3499.WriteLine("CONF:EXT:SOUR 0;OUTP 1")
my3499.WriteLine("ARM:SOUR IMM;COUN 1")
my3499.WriteLine("TRIG:SOUR MIX")
my3499.WriteLine("FUNC 2,2") ' 2-wire mode
my3499.WriteLine("SCAN (@200:219)")
my34401.WriteLine("TRIG:SOUR EXT;COUN " & Str(numChannels))
my34401.WriteLine("*OPC?")
dummy = my34401.Read
my34401.WriteLine("INI社T")
my3499.WriteLine("INI社T")
my3499.WriteLine("*TRG")
my3499.WriteLine("*OPC?")
dummy = my3499.Read
my34401.WriteLine("FETC?")
Readings = my34401.Read
```

D-SCPIを使用したAgilent VXI E1411B/E1476Aのサンプル

```
Dim index As Integer
Dim Readings As String
myDSCPI.WriteLine("ZERO:AUTO ONCE;: TRIG:SOUR IMM;DELAY 0")
```

EFTモード（クローズ／測定／オープン）：

```
myDSCPI.WriteLine("CONF:VOLT:DC 10,.001 (@100)")
myDSCPI.WriteLine("TRIG:COUN 1")
myDSCPI.WriteLine("SAMP:COUN 1")
For index = 1 To numChannels
    myDSCPI.WriteLine("INI社T; FETC?")
    Readings = myDSCPI.Read()
Next
```

DAQモード（スキャン）：

```
myDSCPI.WriteLine("CONF:VOLT:DC 10,.001 (@100:119)")
myDSCPI.WriteLine("INI社T;FETC?")
Readings = myDSCPI.Read()
```

サポート、サービス、およびアシスタンス

アジレント・テクノロジーが、サービスおよびサポートにおいてお約束できることは明確です。リスクを最小限に抑え、さまざまな問題の解決を図りながら、お客様の利益を最大限に高めることにあります。アジレント・テクノロジーは、お客様が納得できる計測機能の提供、お客様のニーズに応じたサポート体制の確立に努めています。アジレント・テクノロジーの多種多様なサポート・リソースとサービスを利用すれば、用途に合ったアジレント・テクノロジーの製品を選択し、製品を十分に活用することができます。アジレント・テクノロジーのすべての測定器およびシステムには、グローバル保証が付いています。アジレント・テクノロジーのサポート政策全体を貫く2つの理念が、「アジレント・テクノロジーのプロミス」と「お客様のアドバンテージ」です。

アジレント・テクノロジーのプロミス

お客様が新たに製品の購入をお考えの時、アジレント・テクノロジーの経験豊富なテスト・エンジニアが現実的な性能や実用的な製品の推奨を含む製品情報をお届けします。お客様がアジレント・テクノロジーの製品をお使いになる時、アジレント・テクノロジーは製品が約束どおりの性能を発揮することを保証します。それらは以下のようなことです。

- 機器が正しく動作するか動作確認を行います。
- 機器操作のサポートを行います。
- データシートに載っている基本的な測定に係わるアシストを提供します。
- セルフヘルプ・ツールの提供。
- 世界中のアジレント・テクノロジー・サービス・センタでサービスが受けられるグローバル保証。

お客様のアドバンテージ

お客様は、アジレント・テクノロジーが提供する多様な専門的テストおよび測定サービスを利用することができます。こうしたサービスは、お客様それぞれの技術的ニーズおよびビジネス・ニーズに応じて購入することが可能です。お客様は、設計、システム統合、プロジェクト管理、その他の専門的なサービスのほか、校正、追加料金によるアップグレード、保証期間終了後の修理、オンサイトの教育およびトレーニングなどのサービスを購入することにより、問題を効率良く解決して、市場のきびしい競争に勝ち抜くことができます。世界各地の経験豊富なアジレント・テクノロジーのエンジニアが、お客様の生産性の向上、設備投資の回収率の最大化、製品の測定確度の維持をお手伝いします。

アジレント・テクノロジー株式会社

本社〒192-8510 東京都八王子市高倉町9-1

計測お客様窓口

受付時間 9:00-19:00
(12:00-13:00もお受けしています。土・日・祭日を除く)
FAX、E-mail、Webは24時間受け付けています。

TEL ■■ 0120-421-345
(0426-56-7832)
FAX ■■ 0120-421-678
(0426-56-7840)
Email contact_japan@agilent.com
電子計測ホームページ
www.agilent.co.jp/find/tm

● 記載事項は変更になる場合があります。
ご発注の際はご確認ください。

Copyright 2005
アジレント・テクノロジー株式会社

電子計測UPDATE

www.agilent.co.jp/find/emailupdates-Japan

Agilentからの最新情報を記載した電子メールを無料でお送りします。

Agilent電子計測ソフトウェアおよびコネクティビティ

Agilentの電子計測ソフトウェアおよびコネクティビティ製品、ソリューション、デベロッパ・ネットワークは、PC標準に基づくツールによって測定器とコンピュータとの接続時間を短縮し、本来の仕事に集中することを可能にします。詳細については**www.agilent.co.jp/find/jpconnectivity**を参照してください。

Agilent Technologies

February 28, 2005
5989-1929JAJP
0000-00DEP