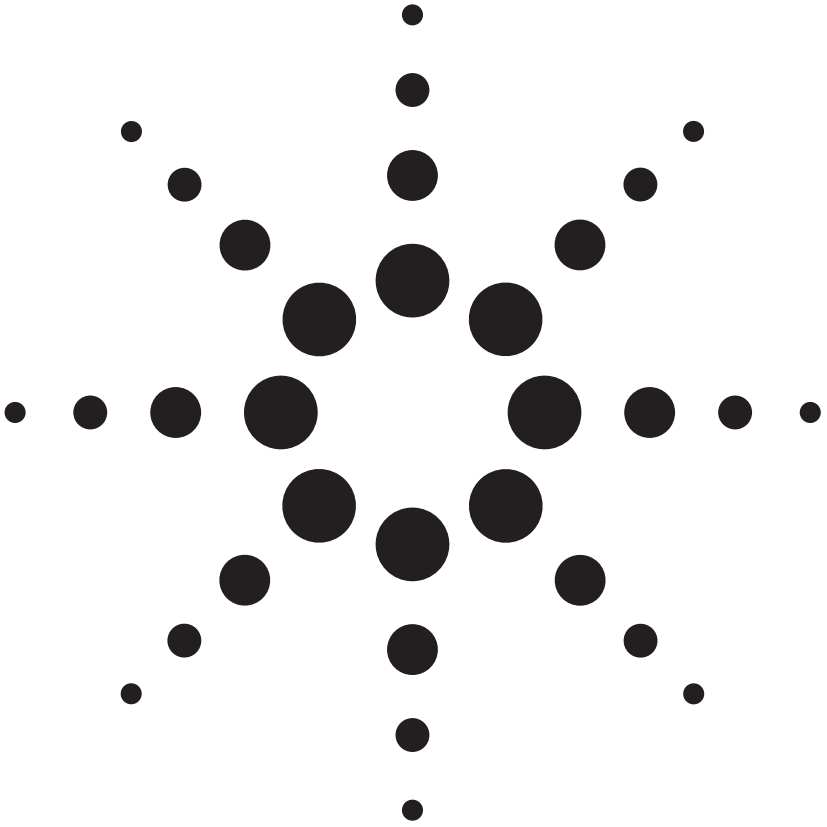
Linux を使用した LXI 測定器の制御：TCP の使用

Application Note 1465-29



Agilent Open では、I/O インタフェースとして、PC 標準の I/O インタフェースを採用しています。これにより、ハードウェア、I/O、ソフトウェア・ツールを柔軟に組み合わてシステムの構築、拡張、保守が可能になります。たとえば、OS として Linux を使用している場合、LAN や USB インタフェースを有効活用できます。本アプリケーション・ノートでは、Linux 環境でテスト機器を制御する方法を解説しています。サンプル・コードは、**http://www.agilent.co.jp/find/linux** からダウンロードできます。

目次

Agilent Technologies

LXI 測定器と LAN ベースの測定器

Agilent は長年にわたって、LAN インタフェースを備えた測定器を提供してきました。2004 年の LXI Consortium[1] の発足とともに、LAN ベースの測定器は急速に普及し始め、テスト業界に広く受け入れられるようになりました。

イーサネットには、コストの安さや、分散／リモート・アプリケーションへの適合性などの、いくつかの明白な利点があります。これほど明白ではないにしても、同程度に重要ないくつかの機能もあります。例えば、ギガビット・イーサネットのきわめて高い性能や、マルチキャスト（1 対多）、ピアツーピア、準同時通信によって実現される柔軟性などです。

イーサネットへの移行は、Linux（およびその他の非 Windows）ユーザにとって大きな利点があります。オペレーティング・システムに内蔵された標準 API を使って測定器を制御できるからです。GPIB や MXI、あるいは PCI カードなどのインタフェースには、使用するオペレーティング・システムのフレーバに対応した特殊なドライバ・ソフトウェアが必要であり、そのようなソフトウェアが使用できない場合もあります。

測定器制御に用いられる TCP/IP プロトコル

2000 年に、VXI*plug&play* Alliance[2] は、LAN ベースの測定器のサポートを VISA 仕様に追加しました。イーサネットによる測定器制御の 2 つの方法が VISA に採用されました。1 つは VXI-11[3] で、もう 1 つはダイレクト TCP ソケット通信です（図 1 を参照）。

VXI-11 は、もともと GPIB の機能をシミュレートするために設計されました。これには、サービス・リクエスト（SRQ）、シリアル・ポール、デバイス・トリガ、デバイス・クリアなどの、ハードウェアに基づいた機能も含まれています。ネイティブ LAN ベースの測定器以前には、LAN-GPIB ゲートウェイで使用されていました。VXI-11 は、リモート・プロシージャ・コール（RPC）に基づいたものです。LAN-GPIB ゲートウェイなどのサーバにより、ゲートウェイにつながれた GPIB 測定器など複数の論理デバイスへのアクセスを実現できます。VXI-11 は LAN-GPIB ゲートウェイ用に設計されたものですが、ネイティブ LAN ベースの測定器の多くでも互換性のためにサポートされています。このタイプの接続の詳細については、Agilent Application Note 1465-28「Linux を使用した LXI 測定器の制御：VXI-11 の使用」で詳細に説明しています。

測定器制御のもう 1 つの方法は、ソケット通信です。これは、ダイレクト TCP ソケット接続経由で、ストリーム方式で測定器を制御するものです。ディスク・ファイルの読み書きに似た方法です。

VXI-11 と TCP ソケット：どちらを使用すべきか？

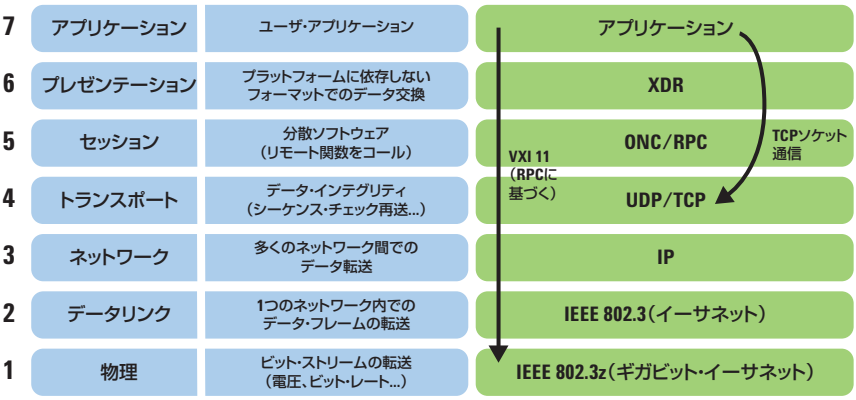
Agilent E5810A などの LAN-GPIB ゲートウェイで GPIB 測定器にアクセスする場合、PC をゲートウェイとして使用する場合、使用できるのは VXI-11 だけです。一方、ネイティブ LAN 測定器の多くは、TCP VXI-11 とソケット通信の両方をサポートしています。どちらを使用する方がよいのでしょうか？

多くの場合は、単に好みの問題です。ただし、VXI-11 の方が複雑な（上位レイヤの）プロトコルです（図 1 を参照）。したがって、ダイレクト・ソケット通信の方が多くの場合高い性能が得られます。特に、実際の測定時間が短く、個別のトランザクションを数多く実行する場合にはダイレクト・ソケット通信のほうが適しています。また、このアプリケーション・ノートの例からわかるように、ソケットの方がずっと使い方が簡単です。したがって、測定器がソケットをサポートする場合は、ネイティブ LAN 測定器に対してはソケットを使用することをお勧めします。

ソケットとトランスポート・オプション

ソケットは 2 つのシステム間の「通信のエンドポイント」であり、郵便箱や電話番号のようなものです。ソケット間の通信は通常双方向（デュプレックス）であり、デバイス・ドライバやその他の通信方式と同様に、ストリーム・モデルに基づいています。データはバイト（文字）のストリームとして伝送され、使用する API の動作はディスク・ファイルの読み書きの場合とよく似ています。

図 1.　TCP/IP の各層と測定器制御での使用



面倒な詳細はオペレーティング・システムが処理してくれます。例えば、バイト・ストリームのデータのまとまりをIPパケットに格納して、宛先システムに届けてくれます。ネットワーク層にIPを使用した場合は、図1に示すように、トランスポート層で使用できるプロトコルにはTCPとUDPの2種類があります。

TCPは「コネクション・ベース」です。すなわち、パケットが正しく伝送されること(確認応答の使用)と、正しい順序で届くこと(シーケンス番号の使用)が、オペレーティング・システムによって保証されています。これに対して、UDPはコネクションレスです。理論的には、パケットが消失したり(確認応答がない)、間違った順序で宛先に届いたりすることがありえます。このような性質から、UDPに比べて多少オーバヘッドは大きいのですが、測定器制御には通常TCPが用いられます。

慣習として、Agilentの測定器やその他のメーカの製品では、TCPとポート5025が制御用に用いられています。

ソケット通信用のAPIコール

前述のように、ソケット通信はLinuxオペレーティング・システムでサポートされています。しかも使い方は簡単です。表1に、ソケット通信用の基本的なシステム・コールを示します。

図2のサンプル・プログラムは、測定器とのTCP接続を確立し、測定器のID文字列を読み取るものです。

最初のステップでは、`socket()`のコールにより、特定のプロトコル・ファミリ(この例では、`PF_INET`すなわちIPv4)とセッション・タイプ(ここでは、`SOCK_STREAM`すなわちTCP)のソケットを作成します。これにより、TCPリンクに必要なシステム・リソースが予約され、初期化されます。

`connect()`コールは、サーバとの接続を確立します。これは、必要な接続の詳細を含むデータ構造体へのポインタを受け取ります。サーバのポート番号とIPアドレスは「ネットワーク・バイト・オーダ」で指定する必要があります(下記参照)。

ここで、測定器と通信する準備が整いました。`send()`をコールしてコマンドを送信します。SCPI文字列の末尾には"\n"(改行)文字が付くことに注意してください。測定器の応答を読み取るには、`recv()`コールを使用します。応答も改行文字で終わります。応答をC文字列に変換するために、末尾にゼロをアペンドする必要があります(後処理で`printf()`などのC文字列関数を使用する場合)。

最後に、`close()`をコールしてソケットを閉じます。

`recv()`コールは、データが使用可能な場合のみ(データが使用可能になった後で)戻ります。すなわち、タイムアウト・メカニズムは組み込まれていません。このため、適切なタイムアウト処理を実装する必要があります。このための1つの方法は、後述する`select()`コールを使用する方法です。

ネットワーク・バイト・オーダ

イーサネット通信は、設計上プラットフォームから独立しています。この独立性は、イーサネット・パケットの全体的なフォーマットと、パケット内の個々のデータ・フィールド(IPアドレス、ポート番号など)のバイト順序を定義することで実現されています。ここで規定されたバイト順序(ビッグ・エンディアン[4])のことを「ネットワーク・バイト・オーダ」とも呼びます。

`connect()`などのAPIコールを使用する場合は、IPアドレスやポート番号として渡すデータは、システムが適切なTCPメッセージを構築するために直接使用されます。このため、パラメータはネットワーク・バイト・オーダでAPI関数に渡す必要があります。

データ型をホスト・バイト・オーダからネットワーク・バイト・オーダに(およびその逆に)変換するために、いくつかの関数が用意されています。例えば`htons()`は、ポート番号(unsigned short)をネットワーク・バイト・オーダに変換します。また、`inet_addr()`は、IPアドレス(ドット記法)を32ビットの符号なし整数に変換する関数であり、結果をネットワーク・バイト・オーダで返します。

表1. ソケット通信用の基本的なLinuxシステム・コール

システム・コール	概要
`socket()`	クライアント(コントローラ)上で、特定のプロトコル(IPv4、IPv6など)とプロトコル・タイプ(コネクションレス/UDPまたはコネクション・ベース/TCP)に対するソケットを作成します。 詳細については socket(2) man ページを参照してください。
`connect()`	与えられたサーバ(IPアドレスとポート番号で参照)に対するソケット接続を開始します。 詳細については connect(2) man ページを参照してください。
`send()`	測定器にメッセージ(SCPIコマンドなど)を送信します。 詳細については send(2) man ページを参照してください。
`recv()`	測定器からデータ(測定結果など)を読み取ります。 詳細については recv(2) man ページを参照してください。
`close()`	connect() で開始された接続を閉じます。 詳細については close(2) man ページを参照してください。

図 2.　TCP による基本的な SCPI 通信

```
int MySocket;
if((MySocket=socket(PF_INET,SOCK_STREAM,0))==-1) exit(1);

struct in_addr {
   unsigned long s_addr;
};
struct sockaddr_in {
   short int sin_family; // アドレス・ファミリ
   unsigned short int sin_port; // ポート番号
   struct in_addr sin_addr; // インターネット・アドレス
   unsigned char sin_zero[8]; // パディング
};
struct sockaddr_in MyAddress;

// 構造体全体をゼロに初期化
memset(&MyAddress,0,sizeof(struct sockaddr_in));
// その後、個々のフィールドを設定
MyAddress.sin_family=PF_INET; // IPv4
MyAddress.sin_port=htons(5025); // ほとんどの測定器が使用するポート番号
MyAddress.sin_addr.s_addr=inet_addr("169.254.9.80"); // IP アドレス

// TCP 接続を確立
if(connect(MySocket,(struct sockaddr *)&MyAddress,
   sizeof(struct sockaddr_in))==-1) exit(1);

// SCPI コマンドを送信
if(send(MySocket,"*IDN?\n",6,0)==-1) exit(1);

// 応答を読み取る
char buffer[200];
int actual;
if((actual=recv(MySocket,&buffer[0],200,0))==-1) exit(1);
buffer[actual]=0; // ゼロを追加（C 文字列）
printf("Instrument ID: %s\n",buffer);

// ソケットを閉じる
if(close(MySocket)==-1) exit(1);
```

Nagle のアルゴリズム / TCP_NODELAY

Linux を含むほとんどのオペレーティング・システムは、Nagle のアルゴリズム（開発者 John Nagle にちなんだ名前。RFC896[5] を参照）を使用して、TCP 通信を効率化しています。このアルゴリズムでは、小さなパケットの送信がわずかな時間だけ遅延されます。

この理由は、複数の小さいデータを 1 つのパケットに結合して送信した方が、別々のパケットを送信するよりも効率的だからです（TCP とイーサネットのオーバヘッドのため）。

これは多くのアプリケーションでうまく動作します。しかし、測定アプリケーションでは、このアルゴリズムを使うと遅延が増加するので望ましくない場合があります。このような場合は、setsockopt() システム・コールを使って、使用するソケットに TCP_NODELAY オプションを設定します。これにより、システムは Nagle のアルゴリズムを使用しなくなります（小さいメッセージもただちに送信されます）。図 3 に示すコードは、TCP_NODELAY オプションを設定するものです。

setsockopt() を使うと、ソケット接続に割り当てられた送信／受信バッファ・サイズなどの、他のさまざまなパラメータも変更できます。詳細については、setsockopt(2)man ページを参照してください。

図 3.　Nagle のアルゴリズムを無効にして遅延を最小にする TCP_NODELAY オプション

```
#include <netinet/tcp.h>
#include <netinet/in.h>

int StateNODELAY = 1; // NODELAY をオンにする

setsockopt(MySocket,IPPROTO_TCP,TCP_NODELAY,
   (void *)&StateNODELAY,sizeof StateNODELAY);
```

制御ポート／デバイス・クリア

ここまで説明した通常のソケット接続の他に、ほとんどの Agilent 測定器はコントロール接続をサポートしています。この TCP リンクは、即座に届ける必要があるメッセージ、すなわちデバイス・クリア・メッセージと SRQ（サービス・リクエスト）のために用いられます。

コントロール接続に使用するポート番号は標準化されていません。ただし、コントロール接続用に使用される測定器のポート番号を問い合わせるために、特別な問合せコマンド SYST:COMM:TCPIP:CONTROL?（通常の接続で送信）が用意されています。

注記：測定器の中にはコントロール接続をサポートしないものもあります。したがって、測定器の応答を確認することが重要です。有効な（0 より大きい）ポート番号が返されると、その測定器はコントロール接続をサポートしています。

コントロール接続によって可能になる重要な機能の 1 つは、測定器の通信バッファをクリアする「デバイス・クリア」です。これは、通信の問題のために通信が中断したときに、測定器の制御を回復するために使用できる場合があります。

デバイス・クリアは、コントロール接続経由で文字列”DCL\n”を送信することによって開始されます。測定器は確認のためにコマンドをエコー・バックします。

図 4 のサンプル・コードは、制御ポートへのリンクをセットアップし、それを使ってデバイスをクリアします。

図 4.　制御ポートを使った測定器のクリア

```
void send_string(int MySocket,char string[])
{
   if(send(MySocket,string,strlen(string),0)==-1) {
      /* エラー処理をここで実行 */
   }
   return;
}

void read_string(int MySocket,char *buffer)
{
   int actual;
   if((actual=recv(MySocket,buffer,200,0))==-1) {
      /* エラー処理をここで実行 */
   }
   else buffer[actual]=0;
   return;
}

send_string(MySocket,"SYST:COMM:TCPIP:CONTROL?\n");
char buffer[200];
read_string(MySocket,buffer);
unsigned int ControlPort;
sscanf(buffer,"%u",&ControlPort);
printf("Control Port: %u\n",ControlPort);

int MyControlSocket;
if((MyControlSocket=socket(PF_INET,SOCK_STREAM,0))==-1) {
   /* エラー処理をここで実行 */
}

struct sockaddr_in MyControlAddress;
memset(&MyControlAddress,0,sizeof(struct sockaddr_in));
MyControlAddress.sin_family=PF_INET; /* IPv4 */
MyControlAddress.sin_port=htons((unsigned short)ControlPort);
MyControlAddress.sin_addr.s_addr=inet_addr("169.254.9.80");
if(connect(MyControlSocket,(struct sockaddr
*)&MyControlAddress,
   sizeof(struct sockaddr_in))==-1) {
   /* エラー処理をここで実行 */
}

send_string(MyControlSocket,"DCL\n");
read_string(MyControlSocket,buffer);
if(strcmp(buffer,"DCL\n")==0)
   printf("DCL\\n received back from instrument...\n");
else printf("Response: %s\n",buffer);

if(close(MyControlSocket)==-1) {
   /* エラー処理をここで実行 */
}
```

SRQ（サービス・リクエスト）

前述のように、制御ポートはSRQにも使用されます。測定器は、何か通知すべきことが起きたとき（エラーが発生したときや、測定器の出力バッファに測定結果が格納されたとき）に、SRQを使ってシステム・コントローラにシグナルを送ります。

測定器は、”SRQ”の後に測定器のステータス・バイトが付いた文字列（例：”SRQ+128\n”）を送信することにより、SRQを通知します。テスト・アプリケーションはこれに対して適切に対応する必要があります。

上記の仕組みには、何らかの形の非同期プログラミングが必要です。アプリケーションは測定器がいつSRQを発生するかを予測できないからです。これを実現するには、「SRQハンドラ」を使用するのが一般的な方法です。ハンドラは、セットアップされた後、コントロール接続からデータが得られる（すなわち、測定器がSRQを通知したことを示すまで、スリープ（サスペンド）状態に入ります。

これを実現する方法の1つは、`select()`[6]関数の使用です。`select()`は、ファイル・デスクリプタ（この例では制御ソケット接続のハンドル）のステータスが変化するまでスリープします。すなわち、この後の例のように、`select()`はデータが使用可能になったとき（またはタイムアウトが発生したとき）に戻ります。

図5に示すサンプル・コードでは、`select()`を使用してSRQを待ちます。

最初に、`select()`でモニタするファイル・デスクリプタをセットアップします。`FD_ZERO`と`FD_SET`は、`fd_set`構造体を操作するためのマクロです。この例では、制御ポートのハンドルだけを構造体に追加します。

次に、SRQを発生するように測定器をセットアップします。この例（34410Aマルチメータを使用）では、動作完了ビットを使ってSRQをトリガします。

その後、`select()`を使って、制御ソケット接続でデータが使用可能になるまで待ちます。

`select()`関数はデータが使用可能になるまで（またはタイムアウトが発生するまで）現在のスレッドをサスペンドすることに注意してください。制御ポートのモニタは通常別スレッドで行います。

まとめ

測定器でソケットが使用できる場合は、VXI-11プロトコルよりもソケットを使用することをお勧めします。ソケットのほうが、実現できる機能は同じで、より性能が高く、使い方も簡単です。

1 LXI (LAN Extensions for Instrumentation) とLXI Consortiumの詳細については、**http://www.lxistandard.org**をご覧ください。

2 VISAとVXI*plug&play* Allianceの詳細については、**http://www.vxipnp.org**をご覧ください。

3 VXI-11の詳細については、**http://www.vxibus.org/freepdfdownloads/vxi-11.pdf**をご覧ください。

4 ビッグ・エンディアンのシステムでは、最上位バイトが先（下位アドレス）に格納されます。Intelプロセッサはリトル・エンディアン（最下位バイトが先）を使用しています。

5 RFC896の詳細については、**http://www.ietf.org/rfc**をご覧ください。

6 詳細については select(2) manページを参照してください。

図5. select()を使ってSRQを待つ

```
fd_set MyFDSet;
struct timeval tv;
int retval;

tv.tv_sec=10; tv.tv_usec=0; // タイムアウト

FD_ZERO(&MyFDSet); // セットをクリア
FD_SET(MyControlPort,&MyFDSet); // 制御ポートを追加

// SRQの発生
send_string(MySocket,"*ESE 1\n"); // OPCは標準イベント・ビットをセット
send_string(MySocket,"*SRE 32\n"); // 標準イベントによりSRQが発生
send_string(MySocket,"CONF:FREQ\n"); // 何かの動作を実行...
send_string(MySocket,"*OPC\n"); // 終了したらOPCをセット

retval=select(MyControlPort+1,&MyFDSet,NULL,NULL,&tv);
if(retval==-1) {
   // エラー処理をここで実行
}
if(retval==1)
{
   // 1つの接続のステータスが変化...制御ポートのはず
   printf("Data available\n");
   read_string(MyControlPort,buffer);
   printf("Data read: %s\n",buffer);
}
```

Agilent の関連カタログ

1465 シリーズのアプリケーション・ノートは、テスト・システムの構築、テスト・システムで有効に LAN/ 無線 LAN/USB を使用する方法、RF/ マイクロ波テスト・システムの最適化と拡張についての豊富な情報を提供しています。

テスト・システム開発

- 『システム開発者ガイド : テスト・システムでの LAN の使用：基礎』AN 1465-9
（カタログ番号 5989-1412JA）
http://cp.literature.agilent.com/litweb/pdf/5989-1412JA.pdf
- 『テスト・システムでの LAN の使用 : ネットワークの設定』AN 1465-10
（カタログ番号 5989-1413JA）
http://cp.literature.agilent.com/litweb/pdf/5989-1413JA.pdf
- 『システム開発ガイド テスト・システムでの LAN の使用：PC の設定』
AN 1465-11
（カタログ番号 5989-1415JA）
http://cp.literature.agilent.com/litweb/pdf/5989-1415JA.pdf
- 『システム開発ガイド 計測環境での USB 使用』AN 1465-12
（カタログ番号 5989-1417JA）
http://cp.literature.agilent.com/litweb/pdf/5989-1417JA.pdf
- 『システム開発ガイド SCPI + ダイレクト I/O、ドライバの使用法』AN 1465-13
（カタログ番号 5989-1414JAJP）
http://cp.literature.agilent.com/litweb/pdf/5989-1414JAJP.pdf
- 『システム開発ガイド テスト・システムにおける LAN の使用法 : アプリケーション』AN 1465-14
（カタログ番号 5989-1416JAJP）
http://cp.literature.agilent.com/litweb/pdf/5989-1416JAJP.pdf
- 『システム開発ガイド テスト・システムでの LAN の使用：システム I/O のセットアップ』AN 1465-15
（カタログ番号 5989-2409JAJP）
http://cp.literature.agilent.com/litweb/pdf/5989-2409JAJP.pdf
- 『LXI による次世代テスト・システム』
AN 1465-16
（カタログ番号 5989-2802JAJP）
http://cp.literature.agilent.com/litweb/pdf/5989-2802JAJP.pdf

RF/ マイクロ波テスト・システム

- 『RF/ マイクロ波テスト・システムの構成要素の最適化』AN 1465-17
（カタログ番号 5989-3321JAJP）
http://cp.literature.agilent.com/litweb/pdf/5989-3321JAJP.pdf
- 『RF/ マイクロ波テストシステムのテスト品質向上のための 6 ヒント』
AN 1465-18
（カタログ番号 5989-3322JAJP）
http://cp.literature.agilent.com/litweb/pdf/5989-3322JAJP.pdf
- 『システムの信号経路の校正：ベクトルおよびスカラ補正法による測定精度の向上』AN 1465-19
（カタログ番号 5989-3323JAJP）
http://cp.literature.agilent.com/litweb/pdf/5989-3323JAJP.pdf

LXI（LAN eXtensions for Instrumentation）

- 『次世代 LXI テスト・システム』
AN 1465-20
（カタログ番号 5989-4371JAJP）
http://cp.literature.agilent.com/litweb/pdf/5989-4371JAJP.pdf
- 『LXI に移行する 10 の理由』
AN 1465-21
（カタログ番号 5989-4372JAJP）
http://cp.literature.agilent.com/litweb/pdf/5989-4372JAJP.pdf
- 『GPIB から LXI への移行』
AN 1465-22
（カタログ番号 5989-4373JAJP）
http://cp.literature.agilent.com/litweb/pdf/5989-4373JAJP.pdf
- 『PXI、VXI、LXI によるハイブリッド・テスト・システムの構築』AN 1465-23
（カタログ番号 5989-4374JAJP）
http://cp.literature.agilent.com/litweb/pdf/5989-4374JAJP.pdf
- 『テスト・システムにおけるシンセティック測定器の使用法 : 利点とトレードオフ』
AN 1465-24
（カタログ番号 5989-4375JAJP）
http://cp.literature.agilent.com/litweb/pdf/5989-4375JAJP.pdf
- 『GPIB から LXI への移行（システム・ソフトウェア編）』AN 1465-25
（カタログ番号 5989-4376JAJP）
http://cp.literature.agilent.com/litweb/pdf/5989-4376JAJP.pdf
- 『LAN/LXI を組み込むための GPIB システムの変更』AN 1465-26
（カタログ番号 5989-6824JAJP）
http://cp.literature.agilent.com/litweb/pdf/5989-6824JAJP.pdf

テスト・システムでの Linux の使用

サンプル・コードは **http://www.agilent.co.jp/find/linux** からダウンロードできます。

- 『Linux を使用したテスト・システム：Linux の基礎』AN 1465-27
（カタログ番号 5989-6715JAJP）
http://cp.literature.agilent.com/litweb/pdf/5989-6715JAJP.pdf
- 『Linux を使用した LXI 測定器の制御：VXI-11 の使用』AN 1465-28
（カタログ番号 5989-6716JAJP）
http://cp.literature.agilent.com/litweb/pdf/5989-6716JAJP.pdf

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 Direct

www.agilent.co.jp/find/agilentdirect

測定器ソリューションを迅速に選択して、使用できます。

Agilent Open

www.agilent.co.jp/find/open

Agilent は、テスト・システムの接続とプログラミングのプロセスを簡素化することにより、電子製品の設計、検証、製造に携わるエンジニアを支援します。Agilent の広範囲のシステム対応測定器、オープン・インダストリ・ソフトウェア、PC 標準 I/O、ワールドワイドのサポートは、テスト・システムの開発を加速します。

LXI

www.lxistandard.org

LXIは、GPIBのLANベースの後継インタフェースで、さらに高速かつ効率的なコネクティビティを提供します。Agilentは、LXIコンソーシアムの設立メンバです。

Remove all doubt

アジレント・テクノロジーでは、柔軟性の高い高品質な校正サービスと、お客様のニーズに応じた修理サービスを提供することで、お使いの測定機器を最高標準に保つお手伝いをしています。お預かりした機器をお約束どおりのパフォーマンスにすることはもちろん、そのサービスをお約束した期日までに確実にお届けします。熟練した技術者、最新の校正試験プログラム、自動化された故障診断、純正部品によるサポートなど、アジレント・テクノロジーの校正・修理サービスは、いつも安心で信頼できる測定結果をお客様に提供します。

また、お客様それぞれの技術的なご要望やビジネスのご要望に応じて、

- アプリケーション・サポート
- システム・インテグレーション
- 導入時のスタート・アップ・サービス
- 教育サービス

など、専門的なテストおよび測定サービスも提供しております。

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、測定器のメインテナンスをサポートいたします。詳しくは：

www.agilent.co.jp/find/removealldoubt

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 (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
(042-656-7832)

FAX■■ 0120-421-678
(042-656-7840)

Email contact_japan@agilent.com

電子計測ホームページ
www.agilent.co.jp

● 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 2008
アジレント・テクノロジー株式会社

Agilent Technologies

April 8, 2008
5989-6717JAJP
0000-00DEP