Tektronix®

データ・アクイジション・システムにソリッド・ステート・スイッチングを導入する3つの理由

ホワイト・ペーパ

KEITHLEY
A Tektronix Company

はじめに

電気機械式（EM）リレーは、多チャンネルデータ収集システム（DAQ）の被試験物（DUT）やセンサに接続するマルチプレクサ・モジュールで使用される最も一般的なタイプのスイッチです。これらのリレーは、信号のルーティングと測定プロセスを容易にします。EMリレーは、ほとんどの状況で十分な性能とコスト効果を発揮しますが、いくつかの制限がある場合もあります。この記事では、ソリッドステート・リレーを使用したマルチプレクサ・モジュールまたはスイッチング・モジュールを使用することで、アプリケーションによりよく適合する3つの典型的な例を取り上げます。

スイッチ接点の長寿命化

DAQシステムは、複数の電気的特性（温度、直流電圧、抵抗など）を、数日、数週間、数ヶ月、あるいはそれ以上の長期間にわたって監視するアプリケーションによく採用されます。DAQシステムは、スイッチングを伴う生産テストにも使用されます。EMリレーの接点寿命は、一般的に数千万サイクル以上ですが、用途によってはこれを割り引かなければなりません。例えば、ケースレー7700 20チャネル差動マルチプレクサ・モジュール（DAQ6510データロガー／データ収集システムと組み合わせて使用）の仕様では、無負荷状態で少なくとも1億サイクルのEMリレーの寿命が期待できるとしています。最大信号レベルで動作させた場合、この仕様は最低10万サイクルとなり、1000倍も低下します。しかし、DAQ6510と互換性のあるケースレー7710 20チャネルソリッドステート差動マルチプレクサ・モジュールの仕様では、ソリッドステート・リレーは最大信号レベルで少なくとも100億サイクルを維持できることが示されています。したがって、ソリッドステート・リレーの寿命は、無負荷スイッチング条件では電気機械式リレーの少なくとも100倍、最大負荷スイッチング条件では100,000倍になります。スイッチング要求の高いアプリケーションでは、ソリッドステート・リレー・マルチプレクサ・モジュールを使用することで、消耗したEMマルチプレクサ・モジュールを交換する際のダウンタイムを短縮し、テスト・システムの寿命までにマルチプレクサを交換するコストを節約することができます。

より高速なスイッチング

7710および3724（3706Aシステムスイッチ／マルチメータに使用されるデュアル1×30 FETマルチプレクサカード）ソリッドステートモジュールは、スイッチの作動時間が短いため、7700 20チャネル・マルチプレクサおよび3720 デュアル1×30チャネル・マルチプレクサ（3706A用）EMリレーカードよりも高速にスキャンできます。ソリッドステート・カードは、電気機械式のスイッチング・モジュールに比べて5～10倍のスイッチング速度を実現します。付録AとBには、スキャンの設定とスイッチング時間の計算方法を示すサンプルコードがあります。付録Aのコードを使用した場合、DAQ6510はソリッドステート・カードでは800チャネル／秒のスキャンレートを達成しましたが、EMカードではわずか80チャネル／秒でした。付録Bのコード例では、3706Aはソリッドステート・カードで1600チャネル／秒のスキャンレートを達成しましたが、EMカードではわずか120チャネル／秒でした。

接触汚染の回避

電気機械式リレー（**図1**）には、磁気で開閉される物理的なスイッチ接点があります。時間が経つと、スイッチの接点に残留物がたまり（特に低レベルの信号を扱う場合）、スイッチがうまく接触しなくなったり、強い磁力が働いても接触抵抗が大きくなったりすることがあります。より高いエネルギーレベルの信号を扱う場合には、電圧や電流によって接点に蓄積された汚れを取り除くことができますが、これは前述の接点スイッチの寿命低下の原因となります。

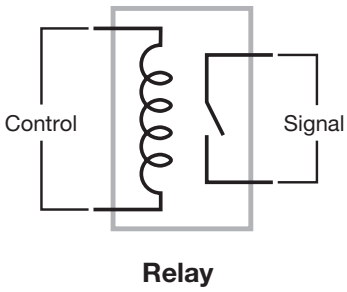

図1. 単純な電気機械式リレー

一方、ソリッドステート・リレー（**図2**）は、一般的に電界効果トランジスタ（FET）で構成されており、不純物による汚染や他のタイプのマテリアル異常を引き起こす可能性のある機械的なスイッチがありません。

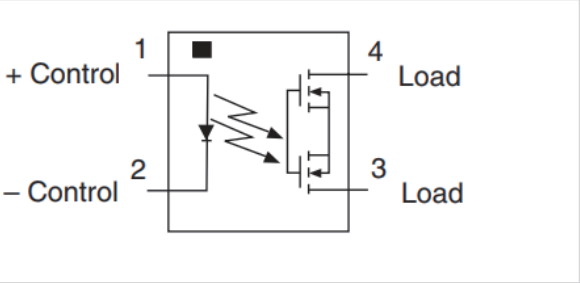

図2. シンプルなソリッドステート・リレー

ソリッドステートスイッチングの限界

ソリッドステートスイッチング・モジュールがEMスイッチング・モジュールよりも常に好まれるのは当然のことのように思われます。しかし、ソリッドステート・スイッチング・モジュールには、ソリッドステート・スイッチで使用可能な最大信号に制限があります。例えば、7710のソリッドステートマルチプレクサ・モジュールの最大信号レベルは、60VDCまたは42V $_{rms}$、100mAです。一般的なEMカードの最大容量は、300VDCまたは300V $_{rms}$、1Aです。EMリレーモジュールは、より広い範囲の電圧と電流で信号を切り替えることができます。ソリッドステート・カードは、FETなどのソリッドステートデバイスのオフステートリーク電流があるために、チャネルアイソレーションが低くなります。また、ソリッドステートのスイッチは、接触抵抗が大きくなります。これは、低抵抗の素子をテストする際の制限となります。最後に、ソリッドステート・スイッチ・モジュールは、同等のEMスイッチ・モジュールよりもコストが高くなります。ソリッドステート・スイッチは、スイッチング速度が速く寿命も長い利点がありますが、使用するスイッチの決定には他の要因も考慮する必要があります。

ユースケース例 - 高速スキャン
蓄電システムのターンオフ直流電流を監視するための高速スキャン

多くの電源電圧が蓄電モジュール（例えば、大きなコンデンサを備えた降圧コンバータ）によって保持されているシナリオを考えてみましょう。このモジュールは、主電源が失われたときに、依存するシステムがメモリの内容をオンボードのNANDフラッシュデバイスに保存している間、約10秒間電力を供給します。各サプライモジュールのパワーレール（今回は3本）の入力に流れる電流を0.1Ωの抵抗シャントで測定し、すべてのサプライモジュールの出力電圧を監視しています。パワーダウン（または損失）が発生すると、DAQ6510と7710マルチプレクサがトリガーされ、10秒間のすべての情報を収集するために可能な限り速い速度でスキャンします。設計者は、貴重なエネルギーを節約するために、最小の電流が引かれていることを観察して、可能な限り多くの機能がオフになっていることを確認するために詳細を確認します。また、設計者は、デザインの10年寿命の要件とメモリの転送が0℃と60℃の両方の温度で達成されていることを確認する必要があります。

図3のテスト回路は、3つの異なる電源と3つの独立した回路負荷を示しており、これらは（入力ではなく）個々の電源の出力にある大きな容量によって支えられています。この例では、DAQ6510は3つの抵抗負荷それぞれにかかるDC電圧と、3つの0.1Ωシャント抵抗（これにより電流を計算できる）をスキャンするように構成されています。もう1つの測定チャンネルは、周囲の環境の温度をサンプリングします。

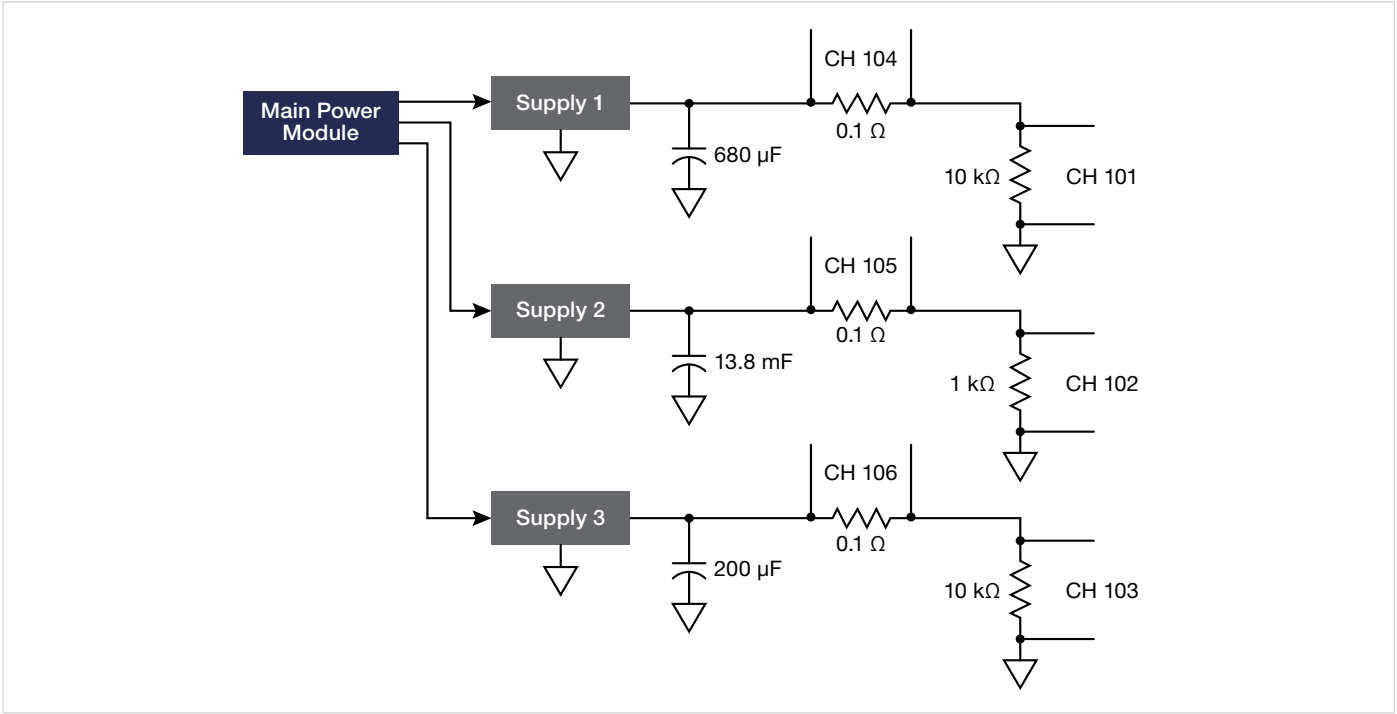

図3. 蓄電システムの停電後の直流電流をモニタする試験回路

次のテスト例は、電源を切った後や停電後のデバイスの特性を分析するために使用できます。この種のテストでは、システムの電流または電圧の引き込みを分析しその製品の電源オフに至る過程を観測します。これは、デバイス内のメモリの保存状態とともに、システムが電源喪失にどのように反応するかを判断するのに役立ちます。

1. DAQ6510の前面にある "**TERMINALS**" スイッチを "**REAR**" の位置にします。
2. DAQ6510の電源を入れてください。
3. "**Build Scan**" ボタンをタッチします。
4. "**+**"ボタンをタッチして "Add a group of channels"を選択します。
5. 表示されたポップアップダイアログでチャンネル**101**、**102**、**103**、**104**、**105**、**106**を選択し、"OK" をタッチします。
6. 測定機能として"**DC Voltage**" を選択します。
7. "settings" タブで、以下のように設定を変更します。
   - "Range" を "**10V**" に変更します。
   - "Auto Delay" を"**OFF**" にします。
   - "NPLC" を"**0.0005**" に変更します。
   - "Auto Zero" を"**OFF**" にします。
8. ディスプレイの左上隅で、"**MENU**" ボタンをタッチし、オプションのリストから"**Expand Group**" を選択します。
9. チャンネル104～106について、レンジを**100mV**に変更します。
10. ディスプレイの左上隅で "**MENU**" ボタンに触れ、オプションのリストから"**Collapse Groups**" を選択します。
11. "**+**" ボタンをタッチして、チャンネルの別のグループを追加します。
12. チャンネル **110**を選択して、"**OK**" をタッチします。
13. 測定機能として"**Temperature**" を選択します。
14. **設定**タブで、チャンネル**110**の設定を以下のように変更します。
   - "Open Lead Detector" を"**OFF**" にします。
   - "Auto Delay" を"**OFF**" に設定します。
   - "NPLC" を**0.0005**に変更します。
   - "Auto Zero" を"**OFF**" に設定します。

15. “**Scan**” タブをタッチして、スキャンの設定を行います。走査時間インジケータが“< 1 second” となっていることを確認してください。
16. “**Scan Count**” ボタンをタッチし、値を1300に変更して、**OK**をタッチする。“Scan Duration” インジケータが “~00:10” と表示され、10秒の実行時間であることを示していることに注意してください。
17. テスト回路の電源が入っていて、安定していることを確認する。
18. DAQ6510ディスプレイの“**Start Scan**” ボタンをタッチし、すぐにテスト回路の主電源を切ります。
19. “**View Scan Status**”ボタンをタッチします。これにより、スキャンのステータスと実行中の進行状況が表示されます。
20. スキャンが完了したら、“**Watch Channels**” ボタンをタッチし、チャンネル**110**の選択を解除し、次にチャンネル**104**、**105**、**106**を選択してください。
21. “**MENU**” キーを押してください。
22. “**Graph**” を選択します。これらの波形は、シャント抵抗にかかる電圧降下を示していることに注意してください。

図4～7は、主電源がない状態でシャントにかかる電圧が減衰していく様子を個別に示したものです。それぞれのレギュレータデバイスの出力におけるキャパシタンスと負荷抵抗の値により、すべてが異なっています。

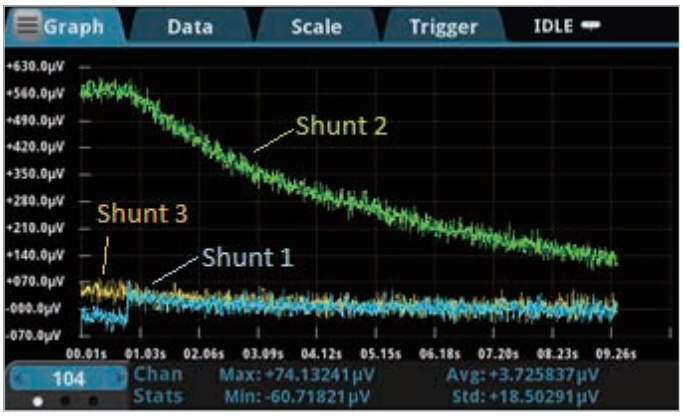

図4. シャント抵抗の測定値をまとめたプロット

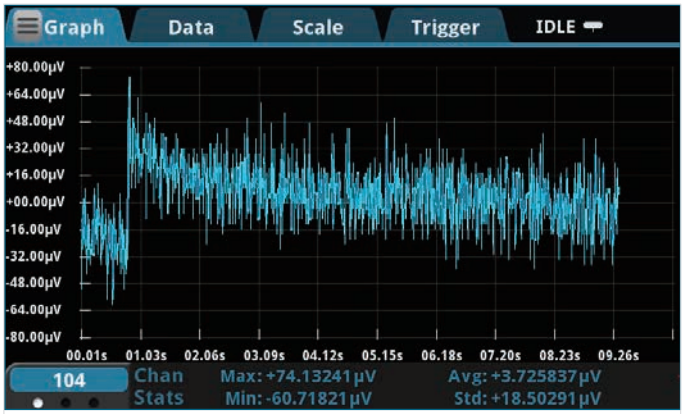

図5. CH104でモニターしたシャントで得られた測定値のプロット

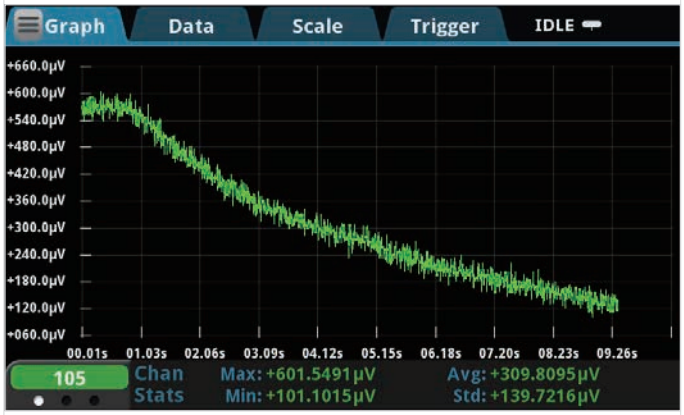

図6. CH105でモニターしたシャントで得られた測定値のプロット

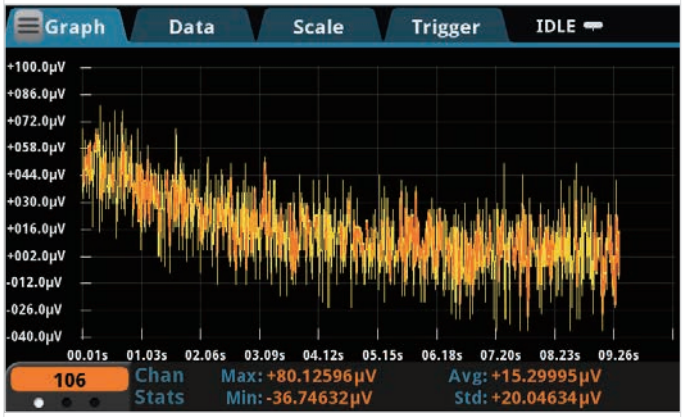

図7. CH106でモニターしたシャントでの測定値のプロット

負荷にかかる電圧降下（**図8**）や温度（図示せず）についても同様の表示が可能です。

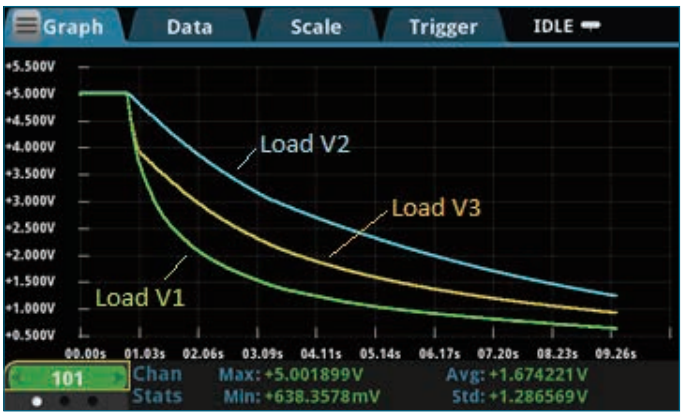

図8. 負荷にかかる電圧降下のコレクティブプロット

データはいくつかの異なる方法でエクスポートすることができ、PCを使ってデータの操作や分析を行うことができます。しかし、この作業は、テスト例のすべてのステップを実行するテストプログラムで自動化することができます。付録CおよびDでは、以下を実行するテストプログラムとその基礎となるスクリプトを紹介しています。

- 前述のように、スキャンの設定を行います。
- DAQ6510を使用して、イーサネット接続を介して2280Sの電源に制御コマンドを発行します。
- 2280Sの電源を設定し、オンにします。
- スキャンを開始します。
- 2280Sの電源をオフにします。
- スキャンの完了を監視します。
- DAQ6510では、シャントに接続された3つのチャンネルの電流を計算し、これらの値を保存するために新しいバッファを作成します。
- 各チャンネルの電圧、電流、または温度データを返します。

結論

DAQ6510と3706Aのソリッドステート・カードは、一般的に使用されているEMカードと比較して、明らかに速度面での優位性を示しています。EMカードに比べて最大10倍の速度差があるため、システムオペレータは一定の時間でより多くのチャンネルやデバイスをスキャンすることができます。テストスピード、大量生産、長期モニタリングが必要な場合、電流電圧のスペック範囲内であればソリッドステートスイッチング・モジュールを検討することをお勧めします。

付録A：DAQ6510および7710マルチプレクサによる高速スキャン

以下の例は、800チャネル／秒のチャネルスキャンレートを達成するために、DAQ6510と7710ソリッドステートマルチプレクサを構成する方法です。コード本体とプログラムコンソールに表示されるコメントは、スキャン実行のタイミングとは別に、スキャン設定のタイミングを詳細に示しています。

このコードはPython3.7で作成されており、測定器と制御用PCとの通信にPyVisa拡張ライブラリを使用しています。

```
import visa
import struct
import math
import time

doDebug = 1
rm = 0
myDaq = 0
printCmds = 0

# ============================================================
# DEFINE FUNCTIONS BELOW...
# ============================================================
def KEI_Connect(rsrcString, doIdQuery, doReset, doClear):
    myInstr = rm.open_resource(rsrcString)
    if doIdQuery == 1:
        print(KEI_Query(myInstr, "*IDN?"))
    if doReset == 1:
        KEI_Write(myInstr, "*RST")
    if doClear == 1:
        myInstr.clear()
    myInstr.timeout = 10000
    return myInstr

def KEI_Write(myInstr, cmd):
    if printCmds == 1:
        print(cmd)
    myInstr.write(cmd)
    return

def KEI_Query(myInstr, cmd):
    if printCmds == 1:
        print(cmd)
    return myInstr.query(cmd)

def KEI_Query_Binary_Values(myInstr, cmd):
    if printCmds == 1:
        print(cmd)
    return myInstr.query_binary_values(cmd, datatype = 'f', is_big_endian = False)

def KEI_Disconnect(myInstr):
    myInstr.close()
    return


#============================================================================
#
#    MAIN CODE STARTS HERE
#
#============================================================================
DAQ_Inst_1 = "USB0::0x05E6::0x6510::04340543::INSTR"
# Instrument ID String examples...
#       LAN -> TCPIP0::134.63.71.209::inst0::INSTR
#       USB -> USB0::0x05E6::0x2450::01419962::INSTR
#       GPIB -> GPIB0::16::INSTR
#       Serial -> ASRL4::INSTR
```

```
# Capture the program start time...
t1 = time.time()

# Opens the resource manager and sets it to variable rm then
#    connect to the DAQ6510
rm = visa.ResourceManager()
myDaq = KEI_Connect(DAQ_Inst_1, 1, 1, 1)

# Reset and start from known conditions
KEI_Write(myDaq, "*RST")

# Set up the reading buffer
KEI_Write(myDaq, "TRACe:MAKE 'mybuf', 1000")
KEI_Write(myDaq, "TRACe:CLEar 'mybuf'")
KEI_Write(myDaq, "FORM:ASC:PREC 0")

# Configure the channel measurement settings to optimize for speed
#    a. Setting a fixed range
#    b. Disabling auto zero
#    c. Disabling auto delay
#    d. Turn line sync off
#    e. Disable filtering and limits
#    f. Decreasing the power line cycles (PLC) to the minimum
KEI_Write(myDaq, "SENS:FUNC 'VOLT', (@101:110)")
KEI_Write(myDaq, "SENS:VOLT:RANG 1, (@101:110)")
KEI_Write(myDaq, "SENS:VOLT:RANG:AUTO 0, (@101:110)")
KEI_Write(myDaq, "SENS:VOLT:AZER OFF, (@101:110)")
KEI_Write(myDaq, "DISP:VOLT:DIG 4, (@101:110)")
KEI_Write(myDaq, "SENS:VOLT:NPLC 0.0005, (@101:110)")
KEI_Write(myDaq, "SENS:VOLT:LINE:SYNC OFF, (@101:110)")
KEI_Write(myDaq, "CALC2:VOLT:LIM1:STAT OFF, (@101:110)")
KEI_Write(myDaq, "CALC2:VOLT:LIM2:STAT OFF, (@101:110)")

# Configure the scanning attributes
KEI_Write(myDaq, "ROUT:SCAN:COUN:SCAN 100")
KEI_Write(myDaq, "ROUT:SCAN:BUFF 'mybuf'")
KEI_Write(myDaq, "ROUT:SCAN:INT 0.0")
KEI_Write(myDaq, "ROUT:SCAN:CRE (@101:110)")

# Change to processing the screen
KEI_Write(myDaq, "DISP:SCR PROC")

# Start the scan...
t2 = time.time()                    # Capture the time when the scan begins...
KEI_Write(myDaq, "INIT")

# Check the state of the scan (via the trigger model), if running
#    or waiting, then continue to hold; if idle then exit the
#    loop and extract the data.
rcvBuffer = KEI_Query(myDaq, "TRIG:STAT?")
while (("RUNNING" in rcvBuffer) or ("WAITING" in rcvBuffer)):
    time.sleep(0.01)
    rcvBuffer = KEI_Query(myDaq, "TRIG:STAT?")
t3 = time.time()                    # Captured the time when the scan ends...

# Change to HOME the screen
KEI_Write(myDaq, "DISP:SCR HOME")

# Extract the data
print(KEI_Query(myDaq, "TRACe:DATA? 1, 1000, 'mybuf'"))

t4 = time.time()                    # Capture the time when the test is complete...

# Terminate the instrument and resource sessions
KEI_Disconnect(myDaq)
rm.close
```

```
# Notify the user of completion and the data streaming rate achieved.
print("done\n")
print("Elapsed Total Test Time: {0:0.3f} s".format(t4-t1))
print("Elapsed Test Configuration Time: {0:0.3f} s".format(t2-t1))
print("Elapsed Scan Time: {0:0.3f} s".format(t3-t2))
print("Elapsed Data Extraction Time: {0:0.3f} s".format(t4-t3))
print("Calculated Scan Rate: {0:0.3f} chan/s".format(1000/(t3-t2)))
print("Calculated Scan Rate with Data Extraction: {0:0.3f} chan/s".format(1000/(t4-t2)))

input("\nPress Enter to continue...")
exit()
```

付録B：3706Aシリーズと3724マルチプレクサによる高速スキャン

以下の例では、3706Aと3724マルチプレクサを組み合わせて、1秒間に1000チャネルを超えるチャネルスキャンレートを実現する方法を示しています。コード本体とプログラムコンソールに表示されるコメントには、スキャンの実行タイミングとは別に、スキャンの設定タイミングの詳細が記載されています。

このコードはVisual Studio 2017のC#を使用して生成され、測定器と制御用PCの間でソケット通信を使用しています。

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Net.Sockets;
using System.Diagnostics;   // for timing tools
using System.IO;
using System.Threading;     // for delay

namespace Series_3706A_Speed_Scanning
{
    class Program
    {
        static public bool echoCommands = true;

        static void Main(string[] args)
        {
            string ipAddress = "192.168.1.37";
            int portNum = 5025;
            TcpClient myClient = null;
            NetworkStream netStream = null;
            string rcvBuffer = "";
            Stopwatch myStpWtch = new Stopwatch();
            myStpWtch.Start();

            // Get the elapsed time as a TimeSpan value.
            TimeSpan ts = myStpWtch.Elapsed;
            string elapsedTime = "";
            int cardSlot = 1;
            String sndBuffer = "";

            InstConnect(ref myClient, ref netStream, ipAddress, portNum, true, false, ref
rcvBuffer);

            // Reset the instrument to the default settings and clear existing system errors...
            InstSend(netStream, "*rst");
            InstSend(netStream, "errorqueue.clear()");

            // Check the interlock state and reset any existing scan attributes...
            InstQuery(netStream, "print(slot[1].interlock.state)", 32, ref rcvBuffer);
            InstSend(netStream, "*cls");
            InstSend(netStream, "scan.reset()");

            // Build the script that will...
            //  a. Configure the measurement channel attributes
            //  b. Clear and size the scan buffer,
            //  c. Establish the scan configuration
            //  d. Execute the scan
            //  e. Provide timers that allow us to monitor
            //      i. Scan setup time
            //      ii. Scan execution time
            InstSend(netStream, "loadscript SCAN_3724");
            InstSend(netStream, "timer.reset()");
            sndBuffer = String.Format("if slot[{0}].interlock.override == 0 then slot[{1}].
interlock.override = 1 end", cardSlot, cardSlot);
```

```
            InstSend(netStream, sndBuffer);
            InstSend(netStream, "channel.open(\"allslots\")");
            InstSend(netStream, "dmm.reset('all')");
            InstSend(netStream, "dmm.func = dmm.DC_VOLTS");
            InstSend(netStream, "dmm.nplc = 0.0005");
            InstSend(netStream, "dmm.displaydigits = dmm.DIGITS_7_5");
            InstSend(netStream, "dmm.autorange = dmm.OFF");
            InstSend(netStream, "dmm.autodelay = dmm.OFF");
            InstSend(netStream, "dmm.autozero = dmm.OFF");
            InstSend(netStream, "dmm.limit[1].enable = dmm.OFF");
            InstSend(netStream, "dmm.limit[2].enable = dmm.OFF");
            InstSend(netStream, "format.data = format.SREAL");  // Use binary data transfer for
readings...
            InstSend(netStream, "dmm.range = 10");
            InstSend(netStream, "dmm.measurecount = 1");
            InstSend(netStream, "scan.scancount = 100");          // used to be measurecount
            InstSend(netStream, "dmm.linesync = dmm.OFF");
            InstSend(netStream, "dmm.configure.set('dcv')");
            InstSend(netStream, "scan_buf = dmm.makebuffer(1000)");
            InstSend(netStream, "channel.connectrule = channel.BREAK_BEFORE_MAKE");
            //InstSend(netStream, "dmm.measure()");

            sndBuffer = String.Format("dmm.setconfig('1001:1010','dcv') scan.
create('1001:1010')");
            InstSend(netStream, sndBuffer);
            InstSend(netStream, "timeLapseSetup = timer.measure.t()");

            InstSend(netStream, "timer.reset()");
            InstSend(netStream, "scan.execute(scan_buf)");
            InstSend(netStream, "timeLapse = timer.measure.t()");
            InstSend(netStream, "endscript");

            // Call the script (on the instrument) that executes the scanning...
            InstSend(netStream, "SCAN_3724()");
            //Extract all data...
            float[] fltData = new float[100];
            int start_index = 1;
            int end_index = 100;
            int chunk_size = 100;
            int mm = 0;
            for (int n = 0; n < 10; n++)
            {
                sndBuffer = String.Format("printbuffer({0}, {1}, scan_buf.readings)", start_
index, end_index);
                InstQuery_FloatData(netStream, sndBuffer, chunk_size, ref fltData);  // scan_
buf.readings,
                start_index += chunk_size;
                end_index += chunk_size;
                for (int m = 0; m < fltData.Length; m++)
                {
                    Console.Write("Rdg {0} = {1},\n", (mm++) + 1, fltData[m]);
                }
            }

            // To get channels per sec scan speed, must divide 30 (the # of chans in a scan) by
elapsed time
            InstSend(netStream, "format.data = format.ASCII");
            InstQuery(netStream, "print(timeLapseSetup)", 128, ref rcvBuffer);
            Console.WriteLine("Time Lapse for scan script configuration: {0:E}", rcvBuffer);

            InstQuery(netStream, "print(timeLapse)", 128, ref rcvBuffer);
            Console.WriteLine("Time Lapse for internal scan execution: {0:E}", rcvBuffer);

            Double testResults = 1000 / Convert.ToDouble(rcvBuffer);
            Console.WriteLine("Calculated Channels/Sec: {0:E}", testResults);

            InstDisconnect(ref myClient, ref netStream);
```

```
            myStpWtch.Stop();

            // Get the elapsed time as a TimeSpan value.
            ts = myStpWtch.Elapsed;

            // Format and display the TimeSpan value.
            elapsedTime = String.Format("{0:00}:{1:00}:{2:00}:{3:00}.{4:000}",
                ts.Days, ts.Hours, ts.Minutes, ts.Seconds,
                ts.Milliseconds / 10);
            Console.WriteLine("Total Program Run Time " + elapsedTime + "\n");

            Console.WriteLine("Press any key to continue...");
            char k = Console.ReadKey().KeyChar;
        }

        static public int InstConnect(ref TcpClient myClient, ref NetworkStream netStream,
string ipAddress, int portNum, bool echoIdString, bool doReset, ref string strId)
        {
            int status = 0;
            try
            {
                myClient = new TcpClient(ipAddress, portNum);
                Console.WriteLine("Connected to instrument......");
                myClient.ReceiveTimeout = 20000;
                myClient.ReceiveBufferSize = 35565;
                netStream = myClient.GetStream();
                if (echoIdString)
                {
                    InstQuery(netStream, "*IDN?", 128, ref strId);
                }
                if (doReset)
                {
                    InstSend(netStream, "reset()");
                }
            }
            catch (Exception e)
            {
                status = -1;
                Console.WriteLine(e.Message);
            }
            finally
            {
                // Nothing to close
            }
            return status;
        }

        static public void InstDisconnect(ref TcpClient myClient, ref NetworkStream netStream)
        {
            netStream.Close();
            myClient.Close();
        }

        static public int InstSend(NetworkStream netStream, string cmdStr)
        {
            try
            {
                byte[] byteBuffer;
                if (echoCommands == true)
                {
                    Console.WriteLine("{0}", cmdStr);
                }
                byteBuffer = Encoding.ASCII.GetBytes(cmdStr + "\r\n");
                netStream.Write(byteBuffer, 0, byteBuffer.Length);
                Array.Clear(byteBuffer, 0, byteBuffer.Length);
                return 0;
            }
            catch (Exception e)
```

```
            {
                Console.WriteLine("{0}", e.Message);
                Console.WriteLine("{0}", e.ToString());
                return -9999;
            }

        }

        static public int InstRcv(NetworkStream netStream, int byteCount, ref string rcvStr)
        {
            try
            {
                byte[] rcvBytes;
                rcvBytes = new byte[byteCount];
                int bytesRcvd = netStream.Read(rcvBytes, 0, byteCount);
                rcvStr = Encoding.ASCII.GetString(rcvBytes, 0, bytesRcvd);
                Array.Clear(rcvBytes, 0, byteCount);
                return 0;
            }
            catch (Exception e)
            {
                Console.WriteLine("{0}", e.Message);
                return -9999;
            }
        }

        static public int InstRcv_FloatData(NetworkStream netStream, int chunkSize, ref
float[] fltData)
        {
            byte[] rcvBytes;
            rcvBytes = new byte[chunkSize * 4 + 3];
            int bytesRcvd = netStream.Read(rcvBytes, 0, rcvBytes.Length);
            // Need to convert to the byte array into single or do
            Buffer.BlockCopy(rcvBytes, 2, fltData, 0, fltData.Length * 4);
            Array.Clear(rcvBytes, 0, rcvBytes.Length);
            return 0;
        }

        static public int InstQuery(NetworkStream netStream, string cmdStr, int byteCount, ref
string rcvStr)
        {
            int status = 0;
            status = InstSend(netStream, cmdStr);
            if (status == 0)
                status = InstRcv(netStream, byteCount, ref rcvStr);
            return status;
        }

        static public int InstQuery_FloatData(NetworkStream netStream, string cmdStr, int
byteCount, ref float[] fltData)
        {
            int status = 0;
            status = InstSend(netStream, cmdStr);
            status = InstRcv_FloatData(netStream, byteCount, ref fltData);
            return 0;
        }
    }
}
```

付録C：DAQ6510と2280S-32-6のテストスクリプトを使ったモニタリング電圧の減衰

このTestScriptProcessor（TSP）コードは、ケースレーのTSP開発環境であるTestScriptBuilderで作成されています。提供するコードは、デバイスをTSP-Netを使用してLAN接続でリンクします。注:DAQ6510は、LAN接続とUSB接続を同時に使用することはできません。このコードは、電源出力がオフのときにカードの7チャネルの高速スキャンを実行し、電源を切ったときのレギュレータの特性を示します。

```
--[[ GLOBAL VARS DEFINED HERE ]]--
gInstPort = 5025
gPsuInstId = nil
gShuntVal = 0.1

--[[ SYSTEM FUNCTIONS DEFINED HERE ]]--
function DAQ_ChanConfig(voltChans, currChans, tempChan)
    --[[
        Configure the channel measurement settings to optimize for speed
            a. Setting a fixed range
            b. Disabling auto zero
            c. Disabling auto delay
            d. Turn line sync off
            e. Disable filtering and limits
            f. Decreasing the power line cycles (PLC) to the minimum
    ]]--

    reset()

    -- Configure channels measuring output voltage
    channel.setdmm(voltChans, dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_DC_VOLTAGE)
    channel.setdmm(voltChans, dmm.ATTR_MEAS_AUTO_DELAY, dmm.DELAY_OFF)
    channel.setdmm(voltChans, dmm.ATTR_MEAS_RANGE, 10)
    channel.setdmm(voltChans, dmm.ATTR_MEAS_RANGE_AUTO, dmm.OFF)
    channel.setdmm(voltChans, dmm.ATTR_MEAS_AUTO_ZERO, dmm.OFF)
    channel.setdmm(voltChans, dmm.ATTR_MEAS_DIGITS, dmm.DIGITS_4_5)
    channel.setdmm(voltChans, dmm.ATTR_MEAS_NPLC, 0.0005)
    channel.setdmm(voltChans, dmm.ATTR_MEAS_LINE_SYNC, dmm.OFF)
    channel.setdmm(voltChans, dmm.ATTR_MEAS_LIMIT_ENABLE_1, dmm.OFF)
    channel.setdmm(voltChans, dmm.ATTR_MEAS_LIMIT_ENABLE_2, dmm.OFF)

    -- Configure channels measuring current by way of the shunt
    channel.setdmm(currChans, dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_DC_VOLTAGE)
    channel.setdmm(currChans, dmm.ATTR_MEAS_AUTO_DELAY, dmm.DELAY_OFF)
    channel.setdmm(currChans, dmm.ATTR_MEAS_RANGE, .1)
    channel.setdmm(currChans, dmm.ATTR_MEAS_RANGE_AUTO, dmm.OFF)
    channel.setdmm(currChans, dmm.ATTR_MEAS_AUTO_ZERO, dmm.OFF)
    channel.setdmm(currChans, dmm.ATTR_MEAS_DIGITS, dmm.DIGITS_4_5)
    channel.setdmm(currChans, dmm.ATTR_MEAS_NPLC, 0.0005)
    channel.setdmm(currChans, dmm.ATTR_MEAS_LINE_SYNC, dmm.OFF)
    channel.setdmm(currChans, dmm.ATTR_MEAS_LIMIT_ENABLE_1, dmm.OFF)
    channel.setdmm(currChans, dmm.ATTR_MEAS_LIMIT_ENABLE_2, dmm.OFF)

    -- Configure channel measuring temperature
    channel.setdmm(tempChan, dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_TEMPERATURE)
    channel.setdmm(tempChan, dmm.ATTR_MEAS_OPEN_DETECTOR, dmm.OFF)
    channel.setdmm(tempChan, dmm.ATTR_MEAS_AUTO_DELAY, dmm.DELAY_OFF)
    channel.setdmm(tempChan, dmm.ATTR_MEAS_AUTO_ZERO, dmm.OFF)
    channel.setdmm(tempChan, dmm.ATTR_MEAS_NPLC, 0.0005)
end

function DAQ_ScanConfig(scanchan, myScanCnt)
    --[[
        Establish the scan and buffer settings
    ]]--
    scan.scancount = myScanCnt
    scan.scaninterval = 0.0
```

```
    scan.create(scanchan)

    defbuffer1.clear()
    format.data = format.ASCII

    -- Note that scan.stepcount should only be used after
    -- scan channels have been defined per scan.create()
    -- or scan.add(), otherwise this system attribute will
    -- be set to zero.
    defbuffer1.capacity = scan.scancount * scan.stepcount
end

function DAQ_Trig()
    --[[
        Trigger the start of the scan
    ]]--
    trigger.model.initiate()
end

function DAQ_ParseReadingBuffer(bufSize)
    --[[
        This utility function is used to break apart the default
        buffer where the collection of all readings is stored
        and separate them out into individual accessible buffers
        for each test point of interest.

        Note that for the buffers which hold current values, we
        not only extract, but also calculate based upon a known
        shunt resistance value (defined as a global above).
    ]]--

    -- Create a series of writable buffers to hold data from each point
    voltBuff1 = buffer.make(bufSize, buffer.STYLE_WRITABLE)
    voltBuff2 = buffer.make(bufSize, buffer.STYLE_WRITABLE)
    voltBuff3 = buffer.make(bufSize, buffer.STYLE_WRITABLE)
    currBuff1 = buffer.make(bufSize, buffer.STYLE_WRITABLE)
    currBuff2 = buffer.make(bufSize, buffer.STYLE_WRITABLE)
    currBuff3 = buffer.make(bufSize, buffer.STYLE_WRITABLE)
    tempBuff = buffer.make(bufSize, buffer.STYLE_WRITABLE)

    -- Establish the fill mode
    voltBuff1.fillmode = buffer.FILL_CONTINUOUS
    voltBuff2.fillmode = buffer.FILL_CONTINUOUS
    voltBuff3.fillmode = buffer.FILL_CONTINUOUS
    currBuff1.fillmode = buffer.FILL_CONTINUOUS
    currBuff2.fillmode = buffer.FILL_CONTINUOUS
    currBuff3.fillmode = buffer.FILL_CONTINUOUS
    tempBuff.fillmode = buffer.FILL_CONTINUOUS

    -- Define the buffer format
    buffer.write.format(voltBuff1, buffer.UNIT_VOLT, buffer.DIGITS_4_5)
    buffer.write.format(voltBuff2, buffer.UNIT_VOLT, buffer.DIGITS_4_5)
    buffer.write.format(voltBuff3, buffer.UNIT_VOLT, buffer.DIGITS_4_5)
    buffer.write.format(currBuff1, buffer.UNIT_AMP, buffer.DIGITS_4_5)
    buffer.write.format(currBuff2, buffer.UNIT_AMP, buffer.DIGITS_4_5)
    buffer.write.format(currBuff3, buffer.UNIT_AMP, buffer.DIGITS_4_5)
    buffer.write.format(tempBuff, buffer.UNIT_CELSIUS, buffer.DIGITS_4_5)

    -- Iterate through the main system buffer to extract specific
    -- readings per buffer.
    for i = 1, defbuffer1.n, 7 do
        -- Extract voltage values
        holder1 = defbuffer1.readings[i]
        buffer.write.reading(voltBuff1, holder1)

        holder2 = defbuffer1.readings[i+1]
        buffer.write.reading(voltBuff2, holder2)
```

```
        holder3 = defbuffer1.readings[i+2]
        buffer.write.reading(voltBuff3, holder3)

        -- Extract current values per I = V/R
        holder4 = defbuffer1.readings[i+3]
        holder4 = holder4 / gShuntVal           -- calculate I
        buffer.write.reading(currBuff1, holder4)

        holder5 = defbuffer1.readings[i+4]
        holder5 = holder5 / gShuntVal           -- calculate I
        buffer.write.reading(currBuff2, holder5)

        holder6 = defbuffer1.readings[i+5]
        holder6 = holder6 / gShuntVal           -- calculate I
        buffer.write.reading(currBuff3, holder6)

        -- Extract temperature values
        holder7 = defbuffer1.readings[i+6]
        buffer.write.reading(tempBuff, holder7)
    end
end

function PSU_Configure(ipAddress, vLevel, iLevel, outState)
    gPsuInstId = PowerSupply_Connect(ipAddress, gInstPort)
    PowerSupply_SetVoltage(gPsuInstId, vLevel)
    PowerSupply_SetCurrent(gPsuInstId, iLevel)
    PowerSupply_OutputState(gPsuInstId, outState)
    PowerSupply_SetDisplayText(gPsuInstId, "Start Test")
end

function PSU_Disable()
    PowerSupply_OutputState(gPsuInstId, 0)
    PowerSupply_SetDisplayText(gPsuInstId, "End Test")
    PowerSupply_Disconnect(gPsuInstId)
end

function PSU_Off()
    PowerSupply_OutputState(gPsuInstId, 0)
end

function PowerSupply_Connect(instAddr, remote_port)
    psuId = tspnet_init(instAddr, remote_port)
    return psuId
end

function PowerSupply_Disconnect(instId)
    tspnet_destroy(instId)
end

function PowerSupply_SetVoltage(instId, vLevel)
    sndBuffer = string.format("SOURce:VOLTage %f", vLevel)
    tspnet_send(instId, sndBuffer)
end

function PowerSupply_SetCurrent(instId, iLevel)
    sndBuffer = string.format("SOURce:CURRent %f", iLevel)
    tspnet_send(instId, sndBuffer)
end

function PowerSupply_OutputState(instId, myState)
    if myState == 0 then
        tspnet_send(instId, "OUTP OFF")
    else
        tspnet_send(instId, "OUTP ON")
    end
end
```

```
function PowerSupply_GetOutputState(instId)
   return tspnet_query(instId, "OUTP?")
end

function PowerSupply_SetDisplayText(instId, myText)
   sndBuffer = string.format("DISP:USER:TEXT \"%s\"", myText)
   tspnet_send(instId, sndBuffer)
end

-- Initialize connection between DAQ and controlled instrument
function tspnet_init(remote_ip, remote_port)
   tspnet.timeout = 5.0
   tspnet.reset()
   tspnet_instID = tspnet.connect(remote_ip, remote_port, "*RST\n")
   if tspnet_instID == nil then return nil     end
   tspnet_ipaddress = remote_ip
   tspnet.termination(tspnet_instID, tspnet.TERM_LF)

   tspnet_send(tspnet_instID, "*RST")
   return tspnet_instID
end

-- Send command to controlled remote instrument
function tspnet_send(tspnet_instID, command)
   tspnet.execute(tspnet_instID, command)
end

-- Query data from the controlled instrument
function tspnet_query(tspnet_instID, command, timeout)
   timeout = timeout or 5.0       --Use default timeout of 5 secs if not specified
   tspnet.execute(tspnet_instID, command)
   timer.cleartime()

   while tspnet.readavailable(tspnet_instID) == 0 and timer.gettime() < timeout do
      delay(0.1)
   end
   return tspnet.read(tspnet_instID)
end

-- Terminate the connection between the master and subordinate instrument
function tspnet_destroy(tspnet_instID)
   if tspnet_instID ~= nil then
      tspnet.disconnect(tspnet_instID)
      tspnet_instID = nil
   end
end

print("Done...")
```

付録 D：供給監視スクリプトを実行するためのプログラムファイル

以下のサンプルコードは、付録Cのスクリプトで定義された関数を呼び出すために使用されます。DAQ6510にロードされた関数をどのように呼び出しているかに注目してください。関数ではスキャンの設定と実行、電源の状態の制御、そして必要な計算を施したバッファの数値の読み出しを行います。

このコードは、Visual Studio 2017のC#を使用して生成されており、測定器と制御用PCとの間の通信にVISA COMドライバーリファレンスを使用しています。

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Threading;
using System.Diagnostics;       // needed for stopwatch usage
using Ivi.Visa.Interop;

namespace Example_DAQ6510_Monitor_Energy_Storage_Module
{
    class Program
    {
        static Boolean echoCmd = true;

        static void Main(string[] args)
        {
            ResourceManager ioMgr = new ResourceManager();
            string[] resources = ioMgr.FindRsrc("?*");

            foreach (string n in resources)
            {
                Console.Write("{0}\n", n);
            }

            FormattedIO488 myInstr = new Ivi.Visa.Interop.FormattedIO488();
            ////////////////////////////////////////////////////////////////////////////////////////////
/////////////
            myInstr.IO = (IMessage)ioMgr.Open("TCPIP0::192.168.1.165::inst0::INSTR", AccessMode.
NO_LOCK, 20000);
            // Instrument ID String examples...
            //       LAN -> TCPIP0::134.63.71.209::inst0::INSTR
            //       USB -> USB0::0x05E6::0x2450::01419962::INSTR
            //       GPIB -> GPIB0::16::INSTR
            //       Serial -> ASRL4::INSTR
            ////////////////////////////////////////////////////////////////////////////////////////////
/////////////
            myInstr.IO.Clear();
            int myTO = myInstr.IO.Timeout;
            myInstr.IO.Timeout = 20000;
            myTO = myInstr.IO.Timeout;
            myInstr.IO.TerminationCharacterEnabled = true;
            myInstr.IO.TerminationCharacter = 0x0A;

            Stopwatch myStpWtch = new Stopwatch();
            Stopwatch CHANTIME = new Stopwatch();

            myStpWtch.Start();

            // Clear any script local to the DAQ6510 which has the name "loadfuncs"
            instrWrite(myInstr, "if loadfuncs ~= nil then script.delete('loadfuncs') end\n");
            // Build the new "loadfuncs" script by defining it then extractin all the functions
            // defined within the test script file local to this program executable.
            instrWrite(myInstr, "loadscript loadfuncs\n");
            string line;

// Load the script file from the path where the Program.cs file resides
```

```
            System.IO.StreamReader file = new System.IO.StreamReader("..\\..\\
myTestFunctions.tsp");
            while ((line = file.ReadLine()) != null)
            {
                instrWrite(myInstr, line);
            }
            file.Close();
            instrWrite(myInstr, "endscript\n");
            //  To ensure all the functions written to the instrument become active, we
            //  call the "loadfuncs" script which holds the definitions.
            Console.WriteLine(instrQuery(myInstr, "loadfuncs()\n"));

            // Configure the DAQ6510 channel measure attributes DCV and Temperature.
            // Note that we will calculate current after the scan is complete.
            String sndBuffer = String.Format("DAQ_ChanConfig(\"{0}\", \"{1}\", \"{2}\")",
"101:103", "104:106", "110");
            instrWrite(myInstr, sndBuffer);

            // Configure the DAQ6510 scan attributes.
            Int16 scanCount = 1300;
            sndBuffer = String.Format("DAQ_ScanConfig(\"{0}\", {1})", "101:106,110", scanCount);
            instrWrite(myInstr, sndBuffer);

            // Tell the DAQ6510 to make a LAN connection to the power supply and configure
            // it to set the output on and supplying 9V at 1.5A.
            sndBuffer = String.Format("PSU_Configure(\'{0}\', {1}, {2}, {3})", "192.168.1.28",
9.0, 1.5, 1);
            instrWrite(myInstr, sndBuffer);

            //start timer for scan time
            CHANTIME.Start();

            // Trigger the scanning to start.
            instrWrite(myInstr, "DAQ_Trig()");

            // Turn the power supply output off.
            instrWrite(myInstr, "PSU_Off()");

            // Loop until the scan has successfully completed.
            CheckScanProgress(myInstr);

            // Scanning timer ending
            CHANTIME.Stop();

            // Ensure that the supply is turned off and the socket connection
            // is closed.
            instrWrite(myInstr, "PSU_Disable()");

            // Split the main buffer (defbufer1) into separate buffer items where the
            // individual channel measurments are warehoused.
            sndBuffer = String.Format("DAQ_ParseReadingBuffer({0})", scanCount);
            instrWrite(myInstr, sndBuffer);

            // Extract each buffer's contents and make them local to the controlling PC. Note
            // that the values for the current channels will hold the current values calculated
            // local to the DAQ6510.
            Console.WriteLine(instrQuery(myInstr, "printbuffer(1, voltBuff1.n, voltBuff1)"));
            Console.WriteLine(instrQuery(myInstr, "printbuffer(1, voltBuff2.n, voltBuff2)"));
            Console.WriteLine(instrQuery(myInstr, "printbuffer(1, voltBuff3.n, voltBuff3)"));
            Console.WriteLine(instrQuery(myInstr, "printbuffer(1, currBuff1.n, currBuff1)"));
            Console.WriteLine(instrQuery(myInstr, "printbuffer(1, currBuff2.n, currBuff2)"));
            Console.WriteLine(instrQuery(myInstr, "printbuffer(1, currBuff3.n, currBuff3)"));
            Console.WriteLine(instrQuery(myInstr, "printbuffer(1, tempBuff.n, tempBuff)"));

            // Output block for the time it took to run just the scan (not including the
            // output of the buffer)
            TimeSpan dt = CHANTIME.Elapsed;
```

```
            double dts = dt.Seconds;
            double dtms = dt.Milliseconds;
            dtms = dtms / 1000;
            double totalt = dts + dtms;
            Console.WriteLine("Scan time elapsed: " + totalt + " Second");

            double chanpersec = (7 * 1300) / totalt;  // number of channels times number
of scans,
                                                      //   then divide by scan time
            Console.WriteLine("Channels scanned per second: " + chanpersec);

            myInstr.IO.Close();

            myStpWtch.Stop();

            // Get the elapsed time as a TimeSpan value.
            TimeSpan ts = myStpWtch.Elapsed;

            // Format and display the TimeSpan value.
            string elapsedTime = String.Format("{0:00}:{1:00}:{2:00}:{3:00}.{4:000}",
                ts.Days, ts.Hours, ts.Minutes, ts.Seconds,
                ts.Milliseconds / 10);
            Console.WriteLine("Total Test Run Time " + elapsedTime);

            Console.WriteLine("Press any key to continue...");
            char k = Console.ReadKey().KeyChar;
        }

        static void instrWrite(FormattedIO488 instr, string cmd)
        {
            if (echoCmd == true)
            {
                Console.WriteLine("{0}", cmd);
            }
            instr.WriteString(cmd + "\n");
            return;
        }

        static string instrQuery(FormattedIO488 instr, string cmd)
        {
            instr.WriteString(cmd);
            return instr.ReadString();
        }


        static void CheckScanProgress(FormattedIO488 instr)
        {
            string trgrcheck = "";
            bool triggercheck = false;
            do
            {
                trgrcheck = instrQuery(instr, "print(scan.state())");
                //Console.WriteLine(trgrcheck);  //uncomment to see the current trigger state
                if (trgrcheck.Contains("SUCCESS"))
                {
                    triggercheck = true;
                }
            } while (triggercheck == false);
            return;
        }
    }
}
```

お問い合わせ先：

オーストラリア 1 800 709 465
オーストリア 00800 2255 4835
バルカン諸国、イスラエル、南アフリカ、その他ISE諸国 +41 52 675 3777
ベルギー 00800 2255 4835
ブラジル +55 (11) 3530 8901
カナダ 1 800 833 9200
中央／東ヨーロッパ、バルト海諸国 +41 52 675 3777
中央ヨーロッパ／ギリシャ +41 52 675 3777
デンマーク +45 80 88 1401
フィンランド +41 52 675 3777
フランス 00800 2255 4835
ドイツ 00800 2255 4835
香港 400 820 5835
インド 000 800 650 1835
インドネシア 007 803 601 5249
イタリア 00800 2255 4835
日本 81 (3) 6714 3086
ルクセンブルク +41 52 675 3777
マレーシア 1 800 22 55835
メキシコ、中央／南アメリカ、カリブ海諸国 52 (55) 88 69 35 25
中東、アジア、北アフリカ +41 52 675 3777
オランダ 00800 2255 4835
ニュージーランド 0800 800 238
ノルウェー 800 16098
中国 400 820 5835
フィリピン 1 800 1601 0077
ポーランド +41 52 675 3777
ポルトガル 80 08 12370
韓国 +82 2 565 1455
ロシア +7 (495) 6647564
シンガポール 800 6011 473
南アフリカ +41 52 675 3777
スペイン 00800 2255 4835
スウェーデン 00800 2255 4835
スイス 00800 2255 4835
台湾 886 (2) 2656 6688
タイ 1 800 011 931
イギリス、アイルランド 00800 2255 4835
アメリカ 1 800 833 9200
ベトナム 12060128

2022年2月現在

KEITHLEY
A Tektronix Company

www.tek.com/ja

テクトロニクス／ケースレーインスツルメンツ

各種お問い合わせ先： https://www.tek.com/ja/contact-tek

技術的な質問、製品の購入、価格・納期、営業への連絡、修理・校正依頼

〒108-6106 東京都港区港南2-15-2 品川インターシティB棟6階

2022年10月　1KZ-61547-0