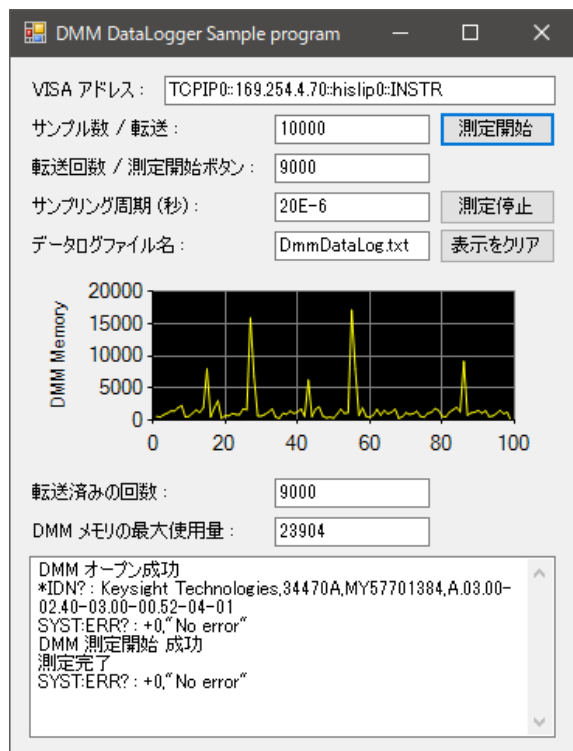


実用的な自動計測マルチメータ 編 サンプルプログラム

34465A/34470Aは、一般的なデジタル・マルチメータの機能に加えて、一定時間間隔で繰り返し測定をおこなう機能を持っています。PCから測定値を読み出すことにより、34465A/34470Aのメモリ以上の連続測定が可能です。

本セミナーのデジタルマルチメータ応用例でご紹介させていただきましたサンプルプログラムでは34465A/34470Aを使用して、一定の時間間隔で連続測定が可能です。

なおプログラムは、Visual Studio 2019 Professional C#、IO Libraries Suite 2020 (VISA.NET) で作成されております



大変申し訳ありませんが、弊社では**サンプルプログラムの保証は
おこなっておりません。**

実際のご利用にあたっては、プログラムの内容を十分にご理解いただいた上で、**お客様の責任**で実行してください。

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace DmmDataLoggerSample
{
    public partial class DmmDataLoggerSample : Form
    {
        public DmmDataLoggerSample()
        {
            InitializeComponent();
        }

        private void DmmDataLoggerSample_Load(object sender, EventArgs e)
        {
            // DMM 内蔵メモリに保存されている測定値の数を表示するグラフの初期化
            chtDmmMem.BackColor = Color.FromName("Control");
            chtDmmMem.Legends[0].Enabled = false;
            chtDmmMem.Series[0].ChartType = System.Windows.Forms.DataVisualization.Charting.SeriesChartType.Line;
            chtDmmMem.ChartAreas[0].BackColor = Color.FromName("Black");
            chtDmmMem.ChartAreas[0].AxisX.MajorGrid.LineColor = Color.FromName("Gray");
            chtDmmMem.ChartAreas[0].AxisX.Maximum = 100;
            chtDmmMem.ChartAreas[0].AxisY.MajorGrid.LineColor = Color.FromName("Gray");
            chtDmmMem.ChartAreas[0].AxisY.Title = "DMM Memory";
            chtDmmMem.Series[0].BorderWidth = 1;
            chtDmmMem.Series[0].Color = Color.FromName("Yellow");
        }

        private void btnClear_Click(object sender, EventArgs e)
        {
            chtDmmMem.Series[0].Points.Clear(); // チャートをクリア
            txtLog.Clear(); // ログをクリア
            txtMaxDmmMem.Text = "";
            txtNumberOfTxed.Text = "";
        }

        private bool StopFlag = false; // Stopボタンの検出用

        private void btnStop_Click(object sender, EventArgs e)
        {
            StopFlag = true; // Stopボタンが押された時の処理 (フラグを立てる)
        }

        private void btnStart_Click(object sender, EventArgs e)
        {
            // 測定パラメータの取得とDMMのオープン
            string systErr; // SYST:ERR?の応答用
            StopFlag = false; // Stopボタンの検出用

            // Startボタンでデータ転送を実行する回数の取得
            if (!int.TryParse(txtTxPerStart.Text, out int txPerStart))
            {

```

```

txtLog.AppendText($"転送回数 / Start ボタン : {txtTxPerStart.Text} を確認してください\r\n");
return;
}
// 1回に転送するサンプル数の取得
if (!int.TryParse(txtSamplePerTx.Text, out int samplePerTx))
{
txtLog.AppendText($"サンプル数 / 転送 : {txtSamplePerTx.Text} を確認してください\r\n");
return;
}

Ivi.Visa.IMessageBasedSession dmm; // VISA.NET DMM用IMessageBasedSession
try
{
dmm = (Ivi.Visa.IMessageBasedSession)Ivi.Visa.GlobalResourceManager.Open(txtVisaAddress.Text);
}
catch (Exception ex)
{
txtLog.AppendText($"DMM オープン失敗 : {ex.Message}\r\n");
return;
}

// DMMをデータログ用に設定し、測定を開始する
try
{
txtLog.AppendText($"DMM オープン成功\r\n");
dmm.FormattedIO.WriteLine("*RST"); // リセット
dmm.FormattedIO.WriteLine("*CLS"); // クリアステータス
dmm.FormattedIO.WriteLine("*IDN?"); // 名称問い合わせ
string idn = dmm.FormattedIO.ReadLine(); // 名称の取得
txtLog.AppendText($"*IDN? : {idn.Replace("\n", "\r\n")}"); // 名称の表示

dmm.FormattedIO.WriteLine("CONF:VOLT:DC 1,MAX"); // DC電圧測定 1Vレンジ、分解能最大
//
// 転送回数を TRIG:COUNに、サンプル数/転送を SAMP:COUNに設定しているが、実際には
// 測定する回数と転送するデータ数が等しければよい。
// TRIG:COUN x SAMP:COUN = 転送回数 * サンプル数/転送 であれば良い。
//
dmm.FormattedIO.WriteLine($"TRIG:COUN {txtTxPerStart.Text}");
dmm.FormattedIO.WriteLine($"SAMP:COUN {txtSamplePerTx.Text}");
dmm.FormattedIO.WriteLine("SAMP:SOUR TIM"); // サンプルソースは内蔵のタイマ
dmm.FormattedIO.WriteLine($"SAMP:TIM {txtSamplePeriod.Text}"); // タイマは最短20usecごと
dmm.FormattedIO.WriteLine("TRIG:DEL 0"); // トリガ遅延なし
dmm.FormattedIO.WriteLine("FORM:DATA REAL,64"); // データ転送はIEEEブロックフォーマット
dmm.FormattedIO.WriteLine("DISP OFF"); // DMMのディスプレイ表示はOFF

dmm.FormattedIO.WriteLine("SYST:ERR?"); // 初期設定後のエラーの確認
systErr = dmm.FormattedIO.ReadLine();
txtLog.AppendText($"SYST:ERR? : {systErr.Replace("\n", "\r\n")}"); // エラーの表示
if (systErr != "+0,\nNo error\r\n") // エラー発生時は中断
{
txtLog.AppendText("エラー検出 測定停止\r\n");
return;
}

dmm.FormattedIO.WriteLine("INIT"); // データロギングの開始
}
catch (Exception ex)
{

```

```
txtLog.AppendText($"DMM 測定開始 失敗 : {ex.Message}\r\n");
return;
}
```

// データロギング実施のループ

```
double[] rData; // 測定値を取得する変数
int dmmMaxMemory = 0; // ロギング中に使用したDMMの内蔵メモリの最大値を保持する変数
```

```
txtLog.AppendText($"DMM 測定開始 成功\r\n");
```

```
try
{
```

```
    for (int i = 0; i < txPerStart; i++)
```

```
    {
```

```
        Application.DoEvents(); // 画面の更新
```

```
        if (StopFlag) // Stopボタンが押されたときの処理
```

```
        {
```

```
            txtLog.AppendText($"測定停止ボタンが検出されました\r\n");
```

```
            dmm.FormattedIO.WriteLine("ABORT"); // 測定を中断
```

```
            break;
```

```
        }
```

```
        dmm.FormattedIO.WriteLine($"DATA:REM? {txtSamplePerTx.Text},WAIT"); // データ転送指示
```

```
        rData = dmm.FormattedIO.ReadLineBinaryBlockOfDouble(); // データ受信 (バイナリ)
```

```
        // クエストショナルレジスタでDMMのメモリのオーバーフローが検出できる
```

```
        dmm.FormattedIO.WriteLine("STAT:QUES:EVEN?"); // クエストショナルレジスタの取得
```

```
        short sqe = (short)dmm.FormattedIO.ReadLineInt64();
```

```
        if (sqe != 0) // メモリオーバーフローを含む、DMMの異常を検出
```

```
        {
```

```
            txtLog.AppendText($"STAT:QUES:EVEN? = {sqe}\r\n測定中断\r\n");
```

```
            dmm.FormattedIO.WriteLine("ABORT"); // 測定を中断
```

```
            break;
```

```
        }
```

```
        //
```

```
        // DMMの測定速度より、データ転送速度が遅い場合、DMMのメモリ内のデータ数が増加する
```

```
        // DMMのメモリ内のデータ数により、今後、オーバーフローするかどうかが予想できる
```

```
        //
```

```
        dmm.FormattedIO.WriteLine("DATA:POIN?"); // DMMのメモリ内のデータの個数を問い合わせ
```

```
        int dp = (int)dmm.FormattedIO.ReadLineInt64();
```

```
        dmmMaxMemory = dp > dmmMaxMemory ? dp : dmmMaxMemory; // データ個数の最大値を保持
```

```
        txtMaxDmmMem.Text = dmmMaxMemory.ToString();
```

```
        chtDmmMem.Series[0].Points.Add(dp); // データの個数をグラフ表示
```

```
        if ((i % 100) == 0) chtDmmMem.Series[0].Points.Clear(); // 100個おきにグラフを更新
```

```
        // 測定値をファイルに追記で保存する
```

```
        using (var writer = new System.IO.StreamWriter(txtFileName.Text, append: true))
```

```
        {
```

```
            foreach (var r in rData)
```

```
            {
```

```
                var s = r.ToString("0.0000E+00");
```

```
                writer.WriteLine(s);
```

```
            }
```

```
        }
```

```
        txtNumberOfTxed.Text = (i + 1).ToString(); // データ転送回数の表示
```

```
    }
```

```
}
```

```
catch (Exception ex)
```

```
{
    txtLog.AppendText($"DMM Data Logging Loop failed : {ex.Message}\r\n");
    return;
}

//
// データログの終了処理
//
txtLog.AppendText("測定完了\r\n");

try
{
    dmm.FormattedIO.WriteLine("SYST:ERR?"); // データロギング終了後のエラー確認
    systErr = dmm.FormattedIO.ReadLine();
    txtLog.AppendText($"SYST:ERR? : {systErr.Replace("\n", "\r\n")}");

    dmm.Dispose(); // DMMをクローズ
}
catch (Exception ex)
{
    txtLog.AppendText($"DMM 測定完了後 失敗 : {ex.Message}\r\n");
    return;
}
}
}
```