ピコ秒パルスレーザ基板ライブラリ（ｔｍＨＩＤＬＤ．ｄｌｌ）説明書

株式会社光響　2016/08/30

概要

ピコ秒パルスレーザ基板を制御するための.Net Framework 4.0 ライブラリについて説明します。

<table>
<tr><td colspan="2">ユーザアプリケーション
(VB.net VC.net VC#.net　LabVIEW など)</td></tr>
<tr><td></td><td>tmHIDLD.dll</td></tr>
<tr><td>.net framework4.0</td><td>setupapi.dll</td></tr>
</table>

本パルスボードシステムは、ＵＳＢによる制御が可能ですが、以下の特徴があります。

- Windows 標準装備のＵＳＢドライバを使用するので、ドライバインストールが不要。
- ハードウェア制御ソフトを.Net Framework 4.0 ライブラリにまとめてあるため、ユーザが容易にアプリケーションを構築できる。
- １つのユーザアプリケーションで複数のボードの制御ができる。

本説明書は上記.Net Framework 4.0 ライブラリである、ｔｍＨＩＤＬＤ．ｄｌｌについて説明します。

対応ＯＳ

.Net Framework 4.0　をインストールした Windows XP
.Net Framework 4.0　をインストールした Windows Vista
Windows 7　32bit,　64bit

ピコ秒パルスレーザ制御基板のハードウェ構成は図１，２のようになっています。
PC と USB２．０フルスピード（１２MHｚ）で接続。HID クラスを使用し、PC 側に USB ドライバは必要ありません。

図１．ハードウェアブロック図

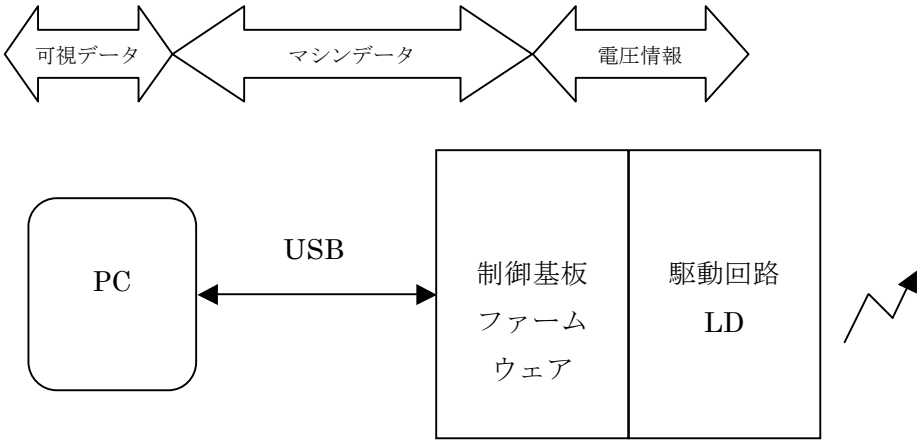
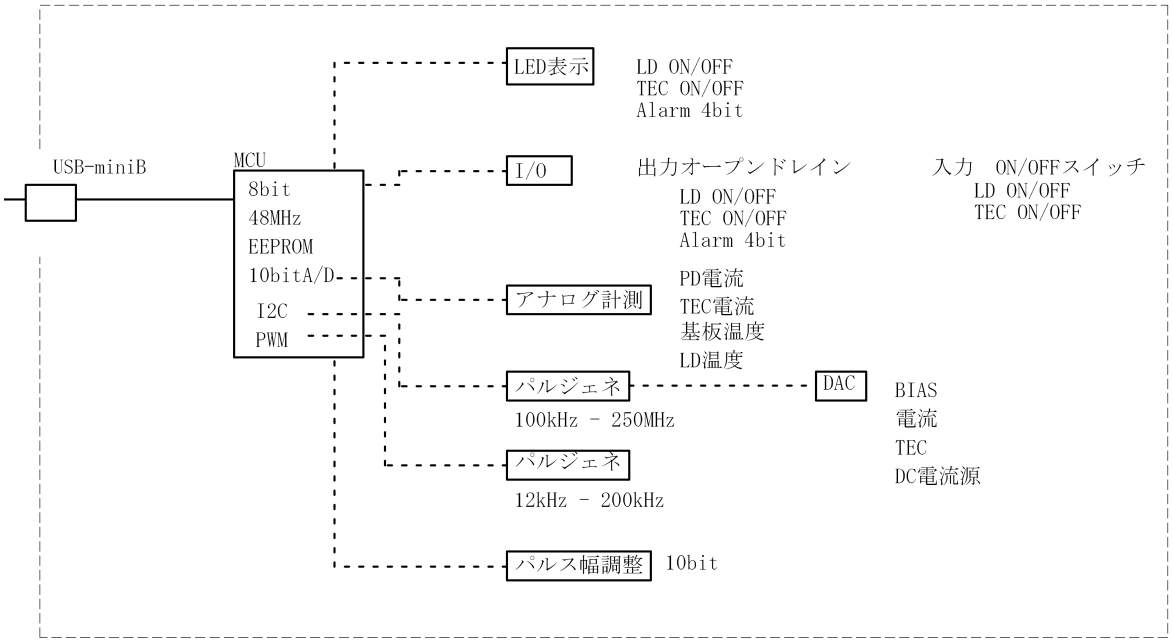
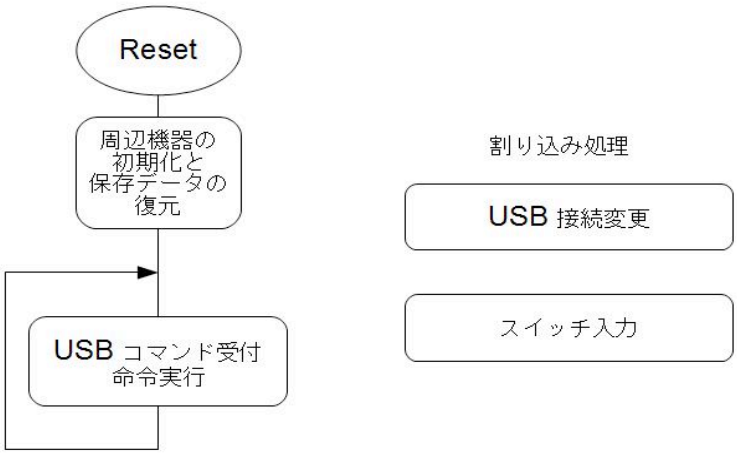


図２．制御基板の機能リスト



ファームウェアの動作



電源 ON、または、リセットスイッチ ON で自身と周辺機器を初期化し、EEPROM に保存されたデータを復元します。その後、命令があれば実行を行う無限ループとなります。割り込み要因としては、USB 接続変更とスイッチ入力があります。USB 接続変更は USB コネクタで PC と接続された時と、切断された時に発生します。切断されるとスタンドアローンモードとなります。LD オン状態で USB 切断が起きると LD も TEC も OFF にします。スイッチ入力は、LD ON、TEC ON があります。

割り込み処理

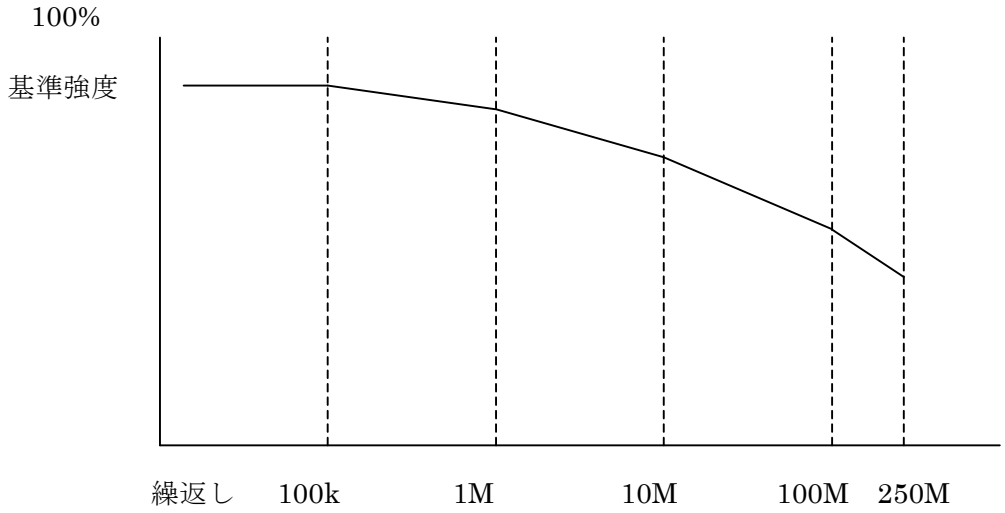
項目	処理内容
USB 接続	USB モードに移行。LD,TEC を OFF する。
USB 切断	スタンドアローンモードに移行。LD,TEC を OFF する。
LD スイッチ ON/OFF	LD スイッチの変化
TEC スイッチ ON/OFF	TEC スイッチの変化

また、各動作モードでは、下記の処理になります。

動作モード	処理内容の違い
USB	USB から制御を行う。但し、LD と TEC の ON はハードウェアスイッチとの AND 動作となる。
スタンドアローン	各設定値は USB により設定されたものを使用する。LD と TEC の ON はハードウェアスイッチのみで ON する。

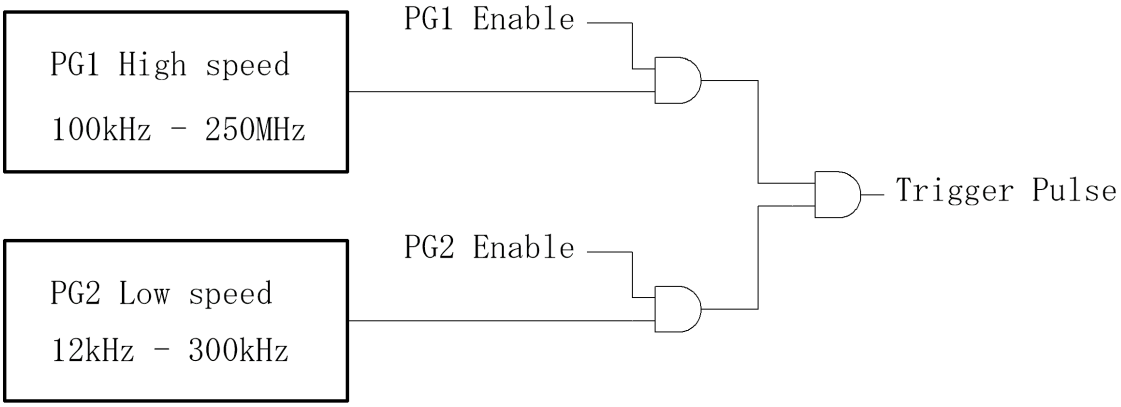
自動光強度補正について（特許出願中）

本LDボードは、内部パルスジェネレータにより、12kHzから250MHzまでの繰返し設定を行うことができますが、ある周波数で最適なピコ秒パルス波形を得るためには、最適な光強度があり、これを外れるとパルス形状に乱れが生じます。繰り返しを設定したときに、この最適な光強度を自動的に設定するのが、自動光強度補正です。補正は下記の図のように行われます。



繰返しが100kHz以下では基準強度となります。これはLDの個体差によりますが、だいたい８０％前後の強度になります。100k 1M 10M 100M 250MHzの各点を基準にして直線近似で補正を行います。ただし、自動補正されたあとでも各種パラメータの変更は自由にできます。
また、各点の補正値はLDの個体差がありますので、各基板で異なる値となります。

パルスジェネレータ（内部トリガ発信器）
パルスジェネレータは２種類あり、３ｋＨｚから２５０ＭＨｚまでをカバーします。



Visual Basic によるプログラムの作成

以下では、最も簡単なアプリケーションを作成します。tmHIDLD.dll のバージョンを読み込むプログラムです。
まず、Visual Studio 2010 Express などで、Windows アプリケーションなどの新しいプロジェクトを作成し、その中のフォルダに tmHIDLD.dll をコピーします。
プロジェクト＞参照の追加　で、上記 tmHIDLD.dll を指定します。

次にコードを表示して、下記のコードを追加します。

```
Dim usb As New tmHIDLD.tmHIDLD(True)
MsgBox(usb.getVersion.ToString())
```

コンパイルして実行すると、下記のようなバージョン情報が表示されます。

Visual C++の場合は、下記のコードになります。

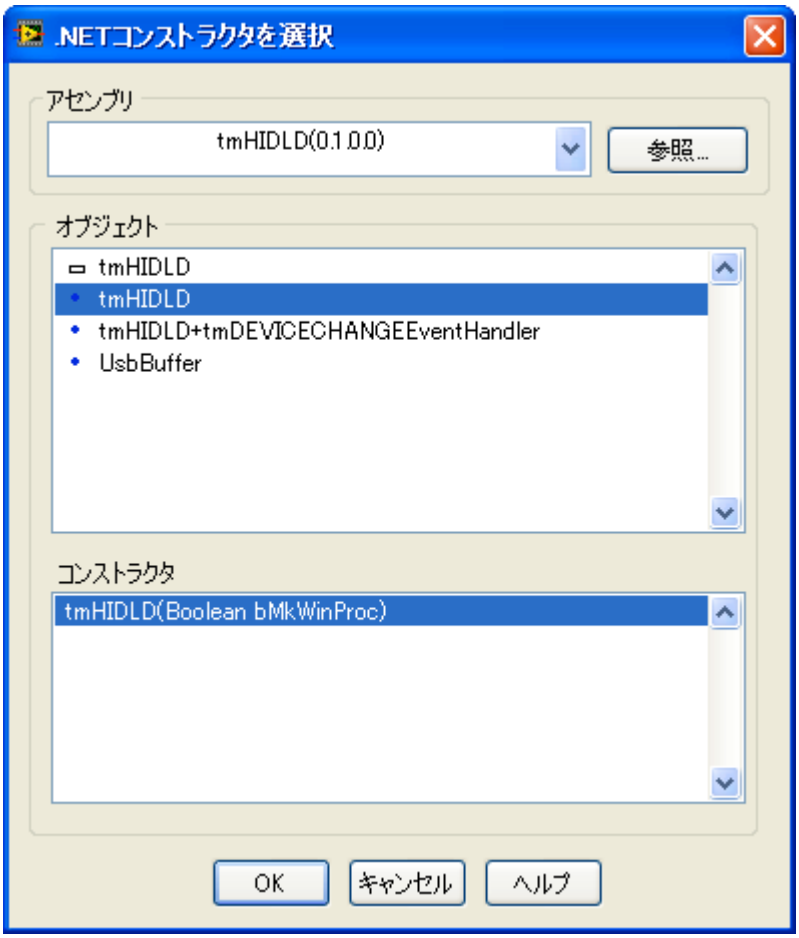
```
tmHIDLD::tmHIDLD^ usb = gcnew tmHIDLD::tmHIDLD(true);
MessageBox::Show(usb->getVersion());
```

Visual C#の場合は、下記のコードになります。

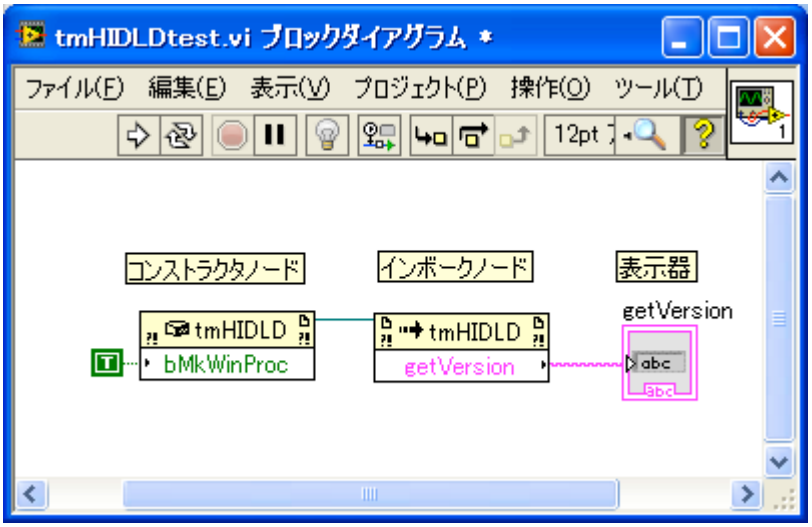
```
tmHIDLD.tmHIDLD usb = new tmHIDLD.tmHIDLD(true);
MessageBox.Show(usb.getVersion());
```

LabVIEW によるプログラムの作成

レーザを制御するプログラムのブロックダイアグラムを開き、コンストラクタノードを追加し、ダブルクリックして下記のダイアログを表示させます。



左のように選択してＯＫで閉じます。



左のようにプログラムします。

実行すると、下記のようにバージョンを表示します。

参考資料：LabVIEW に.NET 4.0 アセンブリをロードする

http://zone.ni.com/reference/ja-XX/help/371361H-0112/lvhowto/configuring_clr_version/

ユーザアプリケーションからのＵＳＢデバイス管理

ＵＳＢデバイスは、ユーザにより任意のタイミングで接続されたり、取り外されたり（切断）します。DLL ライブラリを通じて、このイベント情報を受け取り、制御する方法について説明します。

まず、tmHIDLD.dll は、上記のようなＵＳＢの接続変更イベントに対して、tmDEVICECHANGE をスローします。このイベントはパルス LD ボードが接続されても、切断されても、他のＵＳＢデバイスが接続されても、とにかくＵＳＢ接続に変化があれば発生します。ユーザアプリケーションでは、このイベントを受けたら、ＵＳＢ接続情報を更新するようにプログラムして下さい。
具体的には、下記のコード例のように CheckUSB ファンクションを実行します。
また、何からの理由からイベントを受け取れないユーザアプリケーションのために、Updates プロパティがあります。このプロパティを一定時間ごとに読出し、True になったらイベントとしてＵＳＢ情報を一新します。このプロパティは True を読み出すと False になりますので重複動作しない仕組みになっています。

USB 制御プログラム例

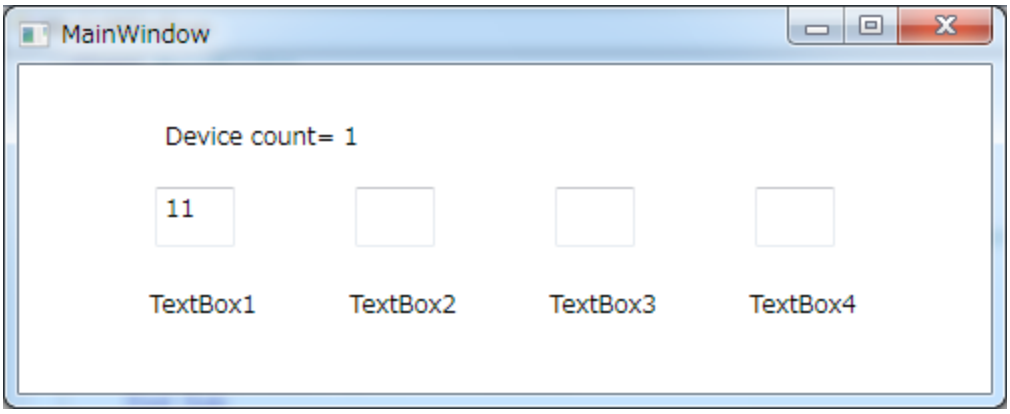
```
Class MainWindow
    'イベント付きのインスタンスを定義します。
    Public WithEvents usblib As tmHIDLD.tmHIDLD

    'プログラムの初期化時
    Private Sub Window_Loaded(ByVal sender As System.Object, _
                              ByVal e As System.Windows.RoutedEventArgs) _
                          Handles MyBase.Loaded
        usblib = New tmHIDLD.tmHIDLD(True)
        usblib.CheckUSB()    '最初のUSB更新

        UpdateUSBIF()        'USB更新
    End Sub

    'tmDEVICECHANGEのイベントハンドラ
    Private Sub sDeviceChange(ByVal sender As Object, _
                              ByVal e As System.EventArgs) _
                          Handles usblib.tmDEVICECHANGE
        usblib.CheckUSB()    'イベント時のUSB更新

        UpdateUSBIF()        'USB更新
    End Sub

    Private Sub UpdateUSBIF()
        usblib.CheckUSB()    'USB更新

        'DevCountプロパティで接続されているボードの数を取得
        Label1.Content = "Device count= " + usblib.DevCount.ToString

        Dim i As Integer
        TextBox1.Text = ""       '表示のクリア
        TextBox2.Text = ""
        TextBox3.Text = ""
        TextBox4.Text = ""
        'CardNo(*)プロパティでそれぞれのカードナンバーを取得
        For i = 0 To usblib.DevCount - 1
            If i = 0 Then TextBox1.Text = usblib.CardNo(i)
            If i = 1 Then TextBox2.Text = usblib.CardNo(i)
            If i = 2 Then TextBox3.Text = usblib.CardNo(i)
            If i = 3 Then TextBox4.Text = usblib.CardNo(i)
        Next

    End Sub
End Class
```

プログラム表示部の例

```
<Window x:Class="MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="MainWindow" Height="202" Width="502">
    <Grid Height="143" Width="450">
        <TextBox     Height="30"     HorizontalAlignment="Left"     Margin="50,50,0,0"
Name="TextBox1" VerticalAlignment="Top" Width="40" />
        <TextBox     Height="30"     HorizontalAlignment="Left"     Margin="150,50,0,0"
Name="TextBox2" VerticalAlignment="Top" Width="40" />
        <TextBox     Height="30"     HorizontalAlignment="Left"     Margin="250,50,0,0"
Name="TextBox3" VerticalAlignment="Top" Width="40" />
        <TextBox     Height="30"     HorizontalAlignment="Left"     Margin="350,50,0,0"
Name="TextBox4" VerticalAlignment="Top" Width="40" />
        <Label      Content="TextBox1"      Height="25"      HorizontalAlignment="Left"
Margin="42,96,0,0" VerticalAlignment="Top" />
        <Label      Content="TextBox2"      Height="25"      HorizontalAlignment="Left"
Margin="142,96,0,0" VerticalAlignment="Top" />
        <Label      Content="TextBox3"      Height="25"      HorizontalAlignment="Left"
Margin="242,96,0,0" VerticalAlignment="Top" />
        <Label      Content="TextBox4"      Height="25"      HorizontalAlignment="Left"
Margin="342,96,0,0" VerticalAlignment="Top" />
        <Label   Content="Device   count=   0"   Height="28"   HorizontalAlignment="Left"
Margin="50,12,0,0" Name="Label1" VerticalAlignment="Top" />
    </Grid>
```

上記プロがラムの実行結果

カード番号１１番のボードを１枚 USB 接続したときの結果です。



制御用ファンクションとプロパティ

MaxUnit プロパティ

Integer で使用するボードの最大数を指定します。これにより内部の変数の割り当て数が決まります。

UseUnit プロパティ

Integer で実際に使用するボードの数を指定します。

Updates プロパティ

ReadOnly Boolean で tmDEVICECHANGE イベントが発生すると True になります。

True を読むと False になります。tmDEVICECHANGE イベントを直接トリガに使用できない場合に、この信号をポーリング処理することでイベントを検知します。

CardNo(n)プロパティ　　n:ゼロから始まるボードの通し番号

ReadOnly Integer で n+1 番目のボードのカード番号を取得します。CheckUSB を実行すると、ライブラリは、カード番号を読み取り、この番号により USB の内部情報を入れ替えます。従って CardNo の結果は必ず小さい順になります。たとえば、CardNo = 2, 1, 8, 6 があるとすると、下記のように並べ替えられます。

CardNo(0) = 1
CardNo(1) = 2
CardNo(2) = 6
CardNo(3) = 8

DevCount プロパティ

ReadOnly Integer で現時点で接続されているボードの数を取得します。

getVersion ファンクション

String 型で tmHIDLD.dll のバージョンを取得します。

SerialNumber(n)プロパティ

Byte 型配列(8 要素)でシリアル番号を読み書きします。書き込む場合は EEPROM の書き換え警告を表示します。

getFirmVersion(n)ファンクション　　n:ゼロから始まるボードの通し番号

String 型でファームウェアのバージョンを取得します。

“FF FF” の形式の半角５文字からなる１６進数になります。

GetEEPROM(n, add, cnt)　　n:ゼロから始まるボードの通し番号(Integer)
add: EEPROM アドレス(Byte)
cnt:　読み出すデータ数(Byte)

Byte 型配列(64 要素)で EEPROM の内容を読み出します。

SetEEPROM(n, add, cnt,　dat, mes)　n:ゼロから始まるボードの通し番号(Integer)
add: EEPROM アドレス(Byte)
cnt:　書き込むデータ数(Byte)
dat:　書き込むデータ(Byte 配列)
mes:　警告を出す／出さない(Boolean)

EEPROM に書き込みを行います。dat のデータ数が cnt より少ないと書き込みは行われません。
書き込み実行／未実行を Boolean 型で返します。

LDOnOff(cmd, n)　　cmd:　コマンド
n:ゼロから始まるボードの通し番号(Integer)

LD オン／オフを制御します。Cmd は 0: off　1:　on
戻り値はありません。

TECOnOff(cmd, n)　　cmd:　コマンド
n:ゼロから始まるボードの通し番号(Integer)

TEC オン／オフを制御します。Cmd は 0: off　1:　on
戻り値はありません。

GetPD(n)　ファンクション　　n:ゼロから始まるボードの通し番号(Integer)

Single 型で PD 電流を取得します。単位 pA
また、同時に下記の値もボードから PC に転送します。この命令実行後、下記のプロパティ名で読み出すことができます。プロパティの読み出しは USB 転送を行いません。

パラメータ	プロパティ名	型	単位
LD on / off	LDOn	Boolean	なし
TEC on / off	TECOn	Boolean	なし
アラーム信号０	GetAlarm0	Boolean	なし
アラーム信号１	GetAlarm1	Boolean	なし
アラーム信号２	GetAlarm2	Boolean	なし
アラーム信号３	GetAlarm3	Boolean	なし
LD 温度	LD_Temp	Single	℃

基板温度	BD_Temp	Single	℃
TEC 制御電流	TEC_current	Single	1000=3.5A
TEC 制御電流(LDB-80)	TEC_current80	Single	183=1A
バイアス電流	Bias	Single	mA
パルス電流	Pulse	Single	mA
SOA 電流	SOA_current	Single	mA
PD 電流	PD_current	Single	pA
PD 電流(LDB-80)	PD_current80	Single	pA
パルス幅	PulseWidthB	Single	psec
IO 信号	IOInfo	Byte	なし
ポテンショ値	RV1Value	Single	%

SetTemp(n, temp) ファンクション n:ゼロから始まるボードの通し番号(Integer)

temp: 温度０～４０（Single)

tmHIDLD40.dll を利用する場合は、０～４０℃

tmHIDLD60.dll を利用する場合は、０～６０℃

TEC 温度を設定します。0.1℃刻みで設定可能です。

戻り値：設定された温度(Single)

GetTemp(n) ファンクション n:ゼロから始まるボードの通し番号(Integer)

TEC 設定温度を取得します。

戻り値：設定された温度(Single) 単位：℃

GetLdTemp(n) ファンクション n:ゼロから始まるボードの通し番号(Integer)

TEC 測定温度を取得します。

戻り値：測定された温度(Single) 単位：℃

SetPGOnOff(n, PG1, PG2, Ext) n:ゼロから始まるボードの通し番号(Integer)

PG1: ON/OFF (Boolean)

PG2: ON/OFF (Boolean)

Ext: ON/OFF (Boolean)

パターンジェネレータを制御します。PG1 と PG2 は同時に ON することが可能ですが、Ext（外部トリガ）を選択すると PG1,PG2 は自動的に OFF になります。

戻り値はありません。

SetPG2SingleShot(n)　　n:ゼロから始まるボードの通し番号(Integer)

この命令で、PG2 からシングルショットトリガが出ます。

GetPGOnOff(n)　　n:ゼロから始まるボードの通し番号(Integer)

Byte 型でパターンジェネレータの状態を取得します。

戻り値　　Bit0: PG1 on/off

Bit1: PG2 on/off

Bit2: Ext　on/off

SetPG1Repetition(n,　freq)　　n:ゼロから始まるボードの通し番号(Integer)

freq:　0-250,000,000　(Uinteger)

PG1 の繰り返しを設定します。単位 Hz。

戻り値はありません。

GetPG1Repetition(n)　　n:ゼロから始まるボードの通し番号(Integer)

UInteger 型で繰り返しを取得します。単位　Hz。

SetPG2Repetition(n,　freq)　　n:ゼロから始まるボードの通し番号(Integer)

freq:　0-250,000,000　(Uinteger)

PG2 の繰り返しを設定します。単位 Hz。

戻り値はありません。

GetPG2Repetition(n)　　n:ゼロから始まるボードの通し番号(Integer)

UInteger 型で繰り返しを取得します。単位　Hz。

ImHere(n)　　n:ゼロから始まるボードの通し番号(Integer)

インターフェース確認用に LED を点滅させます。

戻り値はありません。

SetBias(n,　val　)　　n:ゼロから始まるボードの通し番号(Integer)

val: 0 – 200[mA]

バイアス電流を設定します。パルス振幅と合計して 200mA を超えないように注意願います。

戻り値はありません。

SetBiasAdj(n,　val)　　n:ゼロから始まるボードの通し番号(Integer)
val: 0 – 200[mA]

バイアス電流を設定しますが、EEPROM に書き込みません。調整用などで一時的にバイアス電流を制御する場合に使用します。

GetBias(n)ファンクション　　n:ゼロから始まるボードの通し番号(Integer)

Single 型でバイアス電流を取得します。単位 mA。

SetLDCurrent(n,　val　)　　n:ゼロから始まるボードの通し番号(Integer)
val: 0 – 200[mA]

パルス振幅を設定します。バイアス電流と合計して 200mA を超えないように注意願います。

戻り値はありません。

SetLDCurrentAdj(n,　val)　　n:ゼロから始まるボードの通し番号(Integer)
val: 0 – 200[mA]

パルス振幅を設定しますが、EEPROM に書き込みません。調整用などで一時的にパルス振幅を制御する場合に使用します。

GetLDCurrent(n)ファンクション　　n:ゼロから始まるボードの通し番号(Integer)

Single 型でパルス振幅を取得します。単位 mA。

SetDCCurrent(n,　val　)　　n:ゼロから始まるボードの通し番号(Integer)
val: 0 – 300[mA]

SOA 用定電流源を設定します。

戻り値はありません。

SetDCCurrentAdj(n,　val)　　n:ゼロから始まるボードの通し番号(Integer)
val: 0 – 300[mA]

SOA 用定電流源を設定しますが、EEPROM に書き込みません。調整用などで一時的に SOA 用定電流源を制御する場合に使用します。

GetDCCurrent(n)ファンクション　　n:ゼロから始まるボードの通し番号(Integer)

Single 型で SOA 用定電流源を取得します。単位 mA。

HeaterMode(n,　ICT）ファンクション　　n:ゼロから始まるボードの通し番号
ICT:　ヒーターON/OFF (Boolean)

温度安定用ヒータの動作モードを制御します。

ICT True：30℃

ICT False：84℃

Boolean 型で設定された ICT を返します。

PowerDownMode(n,　PDN1）ファンクションn:ゼロから始まるボードの通し番号
PDN1:　パワーON/OFF (Boolean)

レーザ駆動回路の電源 ON/OFF を制御します。

PDN1 True：電源 ON

PDN1 False：電源 OFF

Boolean 型で設定された PDN1 を返します。

AutoAdjustmentMode(n,　NoAuto）ファンクション n:ゼロから始まるボードの通し番号
NoAuto:　調整 ON/OFF (Boolean)

パルス振幅の自動調整機能の ON/OFF を制御します。

NoAuto True：調整機能 OFF

NoAuto False：調整機能 ON

Boolean 型で設定された NoAuto を返します。

HeaterControl(n)プロパティ　　n:ゼロから始まるボードの通し番号

Byte 型で ICT、PDN1、NoAuto を設定、取得します。設定しないで読み出しのみを行うことができます。

設定時のビット

bit0 = 1：ICT 30℃

bit1 = 1：ICT 84℃

bit2 = 1：PDN1 True

bit3 = 1：PDN1 False

bit4 = 1：NoAuto True

bit5 = 1：NoAuto False

□　bit0 と bit1 を立てた場合、bit1 が優先になります。

取得時のビット

bit0 = 0：ICT 84℃

bit0 = 1：ICT 30℃

bit1 = 0：PDN1 False

bit1 = 1：PDN1 True

bit2 = 0：NoAuto False

bit2 = 1：NoAuto True

GetVR(n)ファンクション　　n:ゼロから始まるボードの通し番号

Single 型でポテンショの値を取得します。単位　%

０～１００の値を返します。

このポテンショは、表示用で、ハードには影響しません。

InitialPG1(n)プロパティ　　n:ゼロから始まるボードの通し番号

Boolean 型で電源投入時のパターンジェネレータ 1 の状態を設定、取得します。

True：PG1 を ON にします。

False：PG1 を OFF にします。

InitialPG2(n)プロパティ　　n:ゼロから始まるボードの通し番号

Boolean 型で電源投入時のパターンジェネレータ 2 の状態を設定、取得します。

True：PG2 を ON にします。

False：PG2 を OFF にします。

InitialPG1 と InitialPG2 を両方とも Off にすると、外部トリガ入力になります。

InitialLDON(n)プロパティ　　n:ゼロから始まるボードの通し番号

Boolean 型で電源投入時の LD On/Off の状態を設定、取得します。

True：LD を ON にします。

False：LD を OFF にします。

InitialTECON(n)プロパティ　　n:ゼロから始まるボードの通し番号

Boolean 型で電源投入時の LD On/Off の状態を設定、取得します。

True：TEC を ON にします。

False：TEC を OFF にします。

InitialLongShort(n)プロパティ　　n:ゼロから始まるボードの通し番号

Boolean 型で電源投入時のパルス幅設定の状態を設定、取得します。ハードウェアにより、設定できない場合があります。Long パルスはトリガ信号のパルス幅になります。

True： パルス幅設定を Long にします。

False： パルス幅設定を Short にします。

CheckUSB ファンクション

USB 接続情報を更新します。USB 接続が変更された場合このファンクションを実行することで、DevCount CardNo などの DLL の内部情報が更新されます。

PulseWidth(n, add, dat)　　n:ゼロから始まるボードの通し番号

add：パルス幅の選択（０～９）Integer

dat：パルス幅データ（１０～１０２３）UShort

dat×10 ピコ秒がパルス幅となります。

パルス幅は、EEPROM に１０セット記憶することができ、そのアドレスは０～９で表現されます。メモリーアドレスを add に、データを dat に指定します。これらのデータはハードウェアと EEPROM に書き込まれ、次回電源 ON 時に有効になります。アドレスに１２８を加えると、アドレス選択はハードウェアにも EEPROM にも書き込みされず、選択の読出しのみになります。dat をゼロにすると、データは EEPROM に書き込まれず、読み出しのみになります。ハードウェア設定を変更しないで、EEPROM の内容のみ読み出したい場合は、add に１２８をプラスして dat をゼロにします。また、ハードウェア設定を変更しないで EEPROM にデータを書き込みたい場合は、add に１２８をプラスして dat に書き込みたいデータを指定します。

読み出しデータは、Byte 型の配列で、３Byte になります。

読み出しデータの内容は下記となります。

1Byte 目　：アドレス　０～９

2Byte 目　：上位 2bit　０～３

3Byte 目　：下位 8bit　０～２５５

dat = 上位 2bit ×　２５６　＋　下位 8bit

コンストラクタ説明

tmHIDLD(MkWinProc)

MkWinProc (Boolean)

True：Win32 Windowメッセージを受ける。

False：Win32 Windowメッセージを受けない。

tmDEVICECHANGE を利用してUSB制御する場合、Trueにします。USBケーブルが抜き差しされると、このイベントがスローされますので、イベントハンドラーを利用してCheckUSBと、その他、必要な処理を実行します。

イベントを利用しない場合は、False にします。この場合は、一定時間ごとに Updates プロパティを読み出すことで、USB ケーブルの抜き差しを検出できます。

以上

改定履歴

2013/05/10

AutoAdjustmentMode 追加。

2013/08/12

PowerLock 追加。

2014/05/21

GetTemp 説明を変更。TEC 温度→TEC 設定温度

コンストラクタ説明を追加

GetLdTemp 追加

PulseWidth 追加

2014/10/15

GetPD(n)で、TEC 制御電流(LDB-80)と PD 電流(LDB-80)を追加。

2016/08/30

InitialLDON～InitialLongShort の部分を追加。PowerLock を削除。

PulseWidth の内容を修正。