

高輝度 LED のパルス幅変調試験：
2651A 型の応用

はじめに

エネルギーの節減へ向けて世界の注目が集まる中、大きな電力を消費する従来型光源へ注がれる目が厳しさを増すとともに、各国政府も照明光源のエネルギー効率向上に向け要求を強めています。このような要求に応えるため、白熱電球に代わり得る光源の開発へ向けて大規模な投資が行われています。現在では、これまでの電球よりもはるかに効率的で寿命の長い電球型蛍光灯 (CFL) が市場に広く行き渡っています。しかし、これらのランプも理想的な代替品と言うにはほど遠いものです。すでに実証されているように、高輝度発光ダイオード (HBLED: High Brightness Light Emitting Diode) の方が、はるかに優れたオプションです。高輝度発光ダイオードは、白熱電球と同様にほぼ瞬間的に最高輝度に達し、廃棄の困難な化学物質を含みません。白熱電球と比較して寿命が驚くほど長く、効率レベルがさらに向上しつつあることも大きな利点です。

エネルギー消費の大幅な削減が可能になるとは言え、HBLED のコストはまだ高いため、残念ながら大部分の消費者が白熱電球よりも HBLED を選択する状況にはまだ至っていません。消費者のコスト負担を下げるため、製造業者は収率改善と効率レベルの向上を目指して絶え間ない努力を重ねています。これらの要求に応えるためには正しい試験が必要であり、そのためには正しい試験装置が必要になります。高輝度 LED の適正な試験のために製造業者が要求する項目は多岐にわたります。このアプリケーションノートでは、電気的試験の幾つかの要件について解説し、2651A 型ハイパワーシステムソースメータ®を適用してこれらの要求を満たす方法を説明します。

より大きなパワーへの要求

HBLED は一般的に 1W またはそれ以上のパワーで動作する LED と定義されており、最も標準的な動作範囲は 1W から 3W までです。一般の LED が 3mm または 5mm のドーム型プラスチックパッケージに封入されて 10~30mA の電流で動作するのに対して、HBLED は LED 接合部の発熱を効率的に散逸させるように設計された熱伝導性の良い小型基板上にマウントされて 300mA~1A、もしくはそれ以上の電流で動作します。1 個の HBLED の輝度も非常に高いのですが、多くの照明アプリケーションは 1 個の HBLED だけで光量をまかなうことはできません。そのため、交換用の LED 電球であれ、照明器具そのものであれ、複数の HBLED を束ねて 1 つの照明光源とするのが普通です。実際の応用では複数の HBLED を組み合わせ使用しますが、製造試験は通常、個別パッケージレベルで実施されるため、試験装置にはそれに対応できるパワー供給能力が要求されます。最新の計測器ベースで構成されたソースメジャーユニット (SMU)、たとえば、ケースレーの 2400 型、2600A 型ソースメータシリーズは、この要求を十分にカバーできる能力を持っています。

多くのアプリケーションでは光が多方向へ広がるのが望ましく、かつ、照明光源のサイズにも十分な空間的余裕がありますから、複数の LED を束ねて 1 つの光源を構成する方式でうまく行きます。しか

し、別なアプリケーション、たとえば、スペースに余裕がないとか、光の指向性が要求されるようなアプリケーションでは、この方法が望ましくないか、あるいは単に機能しないというケースもあり得ます。小さなパッケージから大きな光量を取り出したいという要求が、ハイパワー LED モジュール (1 個以上の大型ダイ LED から構成されます) の開発へと繋がりました。複数のダイが存在する場合は、アプリケーションと利用できる電力源に応じて、ダイを並列または直列に接続します。ハイパワー LED のダイは標準的な HBLED のダイよりもはるかに大きく、それに応じて流すことのできる電流もはるかに大きくなります。1 つのダイが 10A もの電流に耐えなければならないという状況がよく普通に起こります。

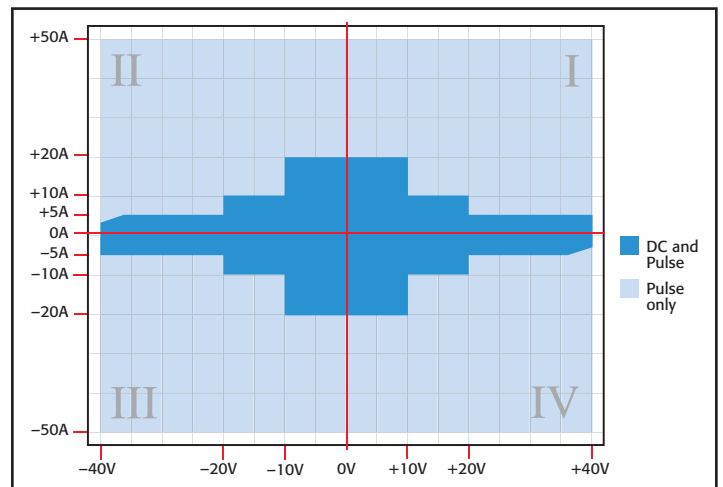


図 1: 2651A 型ハイパワーソースメータが扱うパワー範囲

ハイパワー HBLED モジュールの適正な試験を実施するためには、大きなパワーを DUT に供給できる試験装置を使用しなければなりません。一台の装置で印加と測定の両方を実行できる SMU は LED 試験に最適な試験装置とすることができますが、現在販売されている SMU の大部分は必要なレベルのパワーを供給することができません。ハイパワー HBLED モジュールが 100W 以上のパワーを必要とするケースが多いのに対して、計測器ベースの SMU が供給できるパワーは多くの場合 40W またはそれ以下に限定されています。しかし、ケースレーの 2651A 型ハイパワーソースメータは DC パワーならば連続的に 200W まで、パルスパワーならば 2000W まで供給する能力を備えています。現状および近い将来に現れてくるハイパワーモジュール試験を考えるならば、十分に強力な試験装置とすることができます (図 1)。

パルス幅変調

パルス幅変調は LED の明るさをコントロールするために標準的に用いられている方法の 1 つです。この方法を使用するときは、一定のレベルと一定の周波数を持つ電流パルスを、パルス幅だけを変化させながら LED に流します (図 2)。パルスの幅を変化させると LED が ON 状態となる時間の長さが変化し、これによって目が感じる明るさのレベルも変化します。この駆動方式で動作する LED は実際には点滅動

作を繰り返しているのですが、点滅周波数が非常に高いため、人の目には一定レベルで発光しているように見えます。順方向駆動電流のレベルを下げることによって LED の輝度をコントロールすることも可能ですが、多くの理由によってパルス幅変調の方が選択されています。

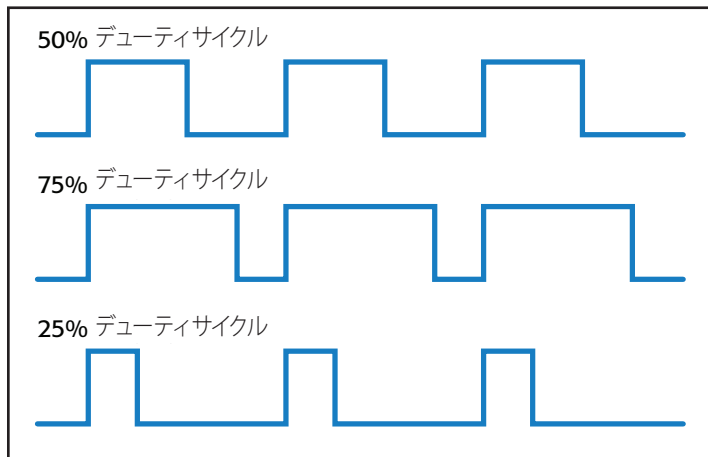


図 2: パルス幅変調では、パルスのレベルと周波数を一定に保ってデューティサイクルを変化させる

パルス幅変調を使用する第 1 の、そして明らかに最も重要な理由は、LED の輝度を低下させたときの色調を一定に保つことです。LED の場合、発生する光の色は LED がその時点で動作している順方向電圧に関係します。LED の順方向電流を変化させたときの順方向電圧は比較的一定に保たれるのですが、それでも数十から数百ミリボルトのレベルで変化します。この変化は、特に低電流レベルでの動作で大きくなります。(図 3)。順方向電圧のわずかな変動によって光の色合いもわずかに変化します。エンドユーザにとって、これは望ましくない現象です。加熱による効果を考えないとすれば、パルス幅変調法では個々のパルスごとに正確に同一レベルの電流が流れますから、順方向電圧もすべてのパルスで同じになります。そのため、発光の色調も変化しません。

パルス幅変調を選択するもう一つの重要な理由は、明るさを直線的にコントロールできることです。LED の発光量は LED の駆動に使用される電流の量に対して直線的な比例関係にはありません。言い換えると、駆動電流を 50% に減らしたとしても光出力の値が 50% 低下するのではなく、それ以外の他の値を示します。このような特性があるため、電流を変化させて明暗調節を行う方式には問題があります。なぜならば、光出力の順方向電流に対する特性を個々の LED につい

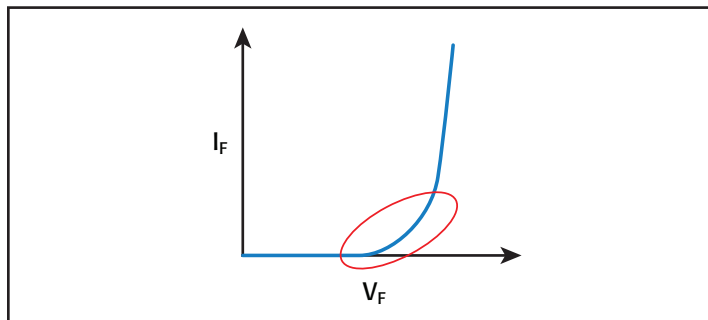


図 3: 順方向電流が低いときは順方向電圧の変化が大きく、順方向電流が大きくなるにしたがって順方向電圧が比較的になる

て評価し、得られた曲線に合わせて駆動法を調節しなければならぬからです。輝度を線形にコントロールしたいのであれば、パルス幅変調を使用するのがはるかに簡単です。パルス幅変調方式ならば、パルスのデューティサイクルを 50% に減らすだけで LED の出力光を 50% に低下させることができます。LED の ON 時間が半分になれば発生する光量も半分になるといふことで、非常に単純です。

パルス幅変調の別な利点としてパワー効率の良さを挙げるができます。パルス幅変調では個々のパルスの電流レベルを一定にしますから、LED が最も効率的に動作するように、すなわち、ワットあたりのルーメン (光束強度) 出力が最大となるようにパルス電流を選択することができます。これは、LED を、どの輝度レベルでも最大効率で動作させられることを意味します。パルス幅変調が効率向上に寄与するもう一つの要素として、ある駆動電流レベルが与えられた場合、LED をパルス駆動した方が DC 駆動よりも大きな光量が得られます。多くの製造メーカのデータシートには駆動電流と光束強度の関係を示すグラフが示されています。製造メーカがパルス駆動と DC 駆動の両方で LED の特性評価を行っている場合、両方式のグラフを比較すれば、パルス駆動の特性曲線が DC 駆動の曲線よりも高い位置にあることが分かります。パルス駆動の方が自己発熱を低くすることができるというのが、効率に違いが発生する理由です。最後に、駆動回路自体のパワー効率を考えるとパルス幅変調の方が優れています。PWM が使用するスイッチング回路は、ほとんどパワーを無駄に消費しません。スイッチ回路がオフのときは実質的に全く電流が流れませんから、消費するパワーもほぼゼロです。スイッチがオンになったときは、オン状態での抵抗が非常に小さいため、ほとんど全てのパワーが LED にそのまま伝達されて、スイッチ自体ではほとんどパワーが消費されません。それに対して、電流を変化させて駆動する方式の場合、パワーの一部が回路の他の部分で消費されるため、LED に伝達されるパワーを減らすことがしばしば起こります。

2651A 型ハイパワーシステムソースメータでのパルス幅変調

記: このアプリケーションノートでは 2651A 型を対象としてパルス幅変調波形の出力方法を説明しますが、この方法は 2600A 型システムソースメータシリーズの全製品で共通に使用できます。

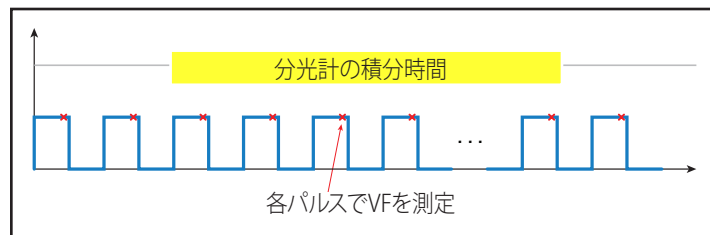


図 4: LED をパルス列で駆動しながら、それぞれのパルスごとに Vf 測定を行う。並行して分光計が光学測定を行う

LED がパルス幅変調で使用されることが多いので、試験にもパルス幅変調法を使用するのが適当です。PWM 試験の一部として通常用いられる試験法では、LED にパルス列を印加しながら、分光計を用いて複数のパルスにまたがる光出力を総合的に測定します。この測定は完了までに数十から数百ミリ秒を要することがあります。パルス出力が継続している期間中は、パルスごとに順方向電圧を測

定して LED の温度上昇に伴う変化を調べます。この試験を模式的に図 4 に示します。

2651A 型ハイパワーシステムソースメータは、100% までのデューティサイクルで 0~20A のパルス変調波形を出力する能力を持っており、さらに 50% のデューティサイクルならば 20~30A、35% のデューティサイクルならば 30~50A を出力することができます。2651A 型の先進的なトリガモデルを使用すれば、精密なパルス幅とデューティサイクルに加えて、他の測定装置との緊密な同期が可能です。このような同期機能を使用して 2 台の 2651A 型を連動させると、1 台で可能な電流レベルの 2 倍の電流を同じデューティサイクルで供給することができます。このノートでは、2651A 型を使用して 30A、50% デューティサイクルの波形を発生させ、同時にデジタル I/O 出力を使用して分光分析計にトリガをかける方法を説明します。また、同じ試験で 2 台の 2651A 型を連動させることにより、電流を 60A に増大させる方法についても説明します。

使用する装置

この試験を実施するためには以下の装置が必要となります：

- GPIB または Ethernet アダプタを備えた PC
 - GPIB ケーブル、または RJ45 LAN クロスオーバー Ethernet ケーブル
 - 2651A 型ハイパワーソースメータ
 - 2651A-KIT-1 型低インピーダンス/高電流同軸ケーブル
 - 8ピン信号コントロールケーブル
 - 2ピン端子ブロックエクステンダ (2651A-KIT-1 型用として使用)
 - 8ピン端子ブロックエクステンダ (8ピン信号コントロールケーブル用として使用)
 - デジタル I/O 用 DB-25 (オス) コネクタキット ハードウェア
 - 12 AWG 以上の太さの電線 (端子ブロックとデバイスを接続)
- 35% デューティサイクルで 100A まで、または 50% デューティサイクルで 60A までの試験を実施する場合は以下の装置を追加してください。
- 2651A 型ハイパワーソースメータ
 - 2651A-KIT-1 型低インピーダンス/高電流同軸ケーブル
 - 8ピン信号コントロールケーブル
 - 2ピン端子ブロックエクステンダ (2651A-KIT-1 型用として使用します)
 - 8ピン端子ブロックエクステンダ (8ピン信号コントロールケーブル用として使用)
 - TSP-Link RJ45 LAN クロスオーバーケーブル

通信

1 台のソースメータの場合

2651A 型ソースメータを 1 台使用して試験回路を構成する場合は、図 5 に示すように、GPIB または Ethernet を使用して計測器とコンピュータを接続します。

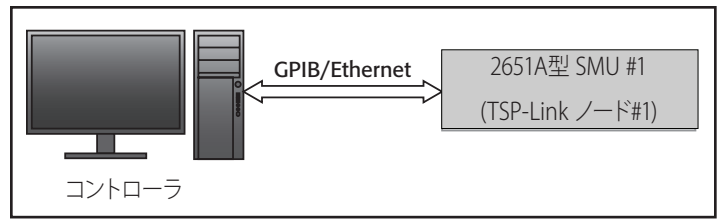


図 5: ソースメータを 1 台使用する試験の通信セットアップ

2 台のソースメータの場合

2651A 型ソースメータを 2 台使用してシステムを構成する場合は、1 台の 2651A 型を GPIB または Ethernet を介してコンピュータへ接続し、2 台目の 2651A 型へは TSP-Link で最初の 2651A 型と接続します。最初の 2651A 型にノード番号 #1 を割り付け、2 番目の 2651A 型のノード番号を #2 とします。これらの接続方法を図 6 に示します。

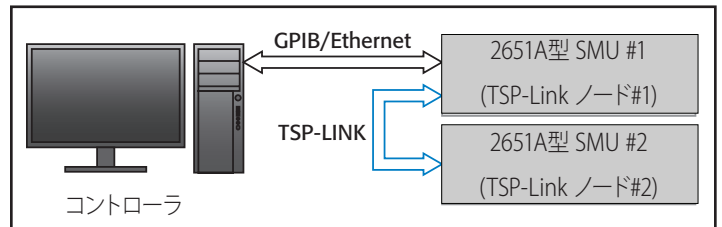


図 6: 2 台の 2651A 型を使用するシステムの通信セットアップ

分光計とデジタル I/O の接続

2651A 型から分光計にトリガを掛けられるようにするには、分光計の試験開始トリガラインを 2651A 型のデジタル I/O ポートへ接続しなければなりません。分光計からのトリガラインを 25ピン D-Sub コネクタ (2651A 型背面パネル) のピン #1 へ接続してください。ピン #15 から #21 はいずれもグラウンドラインですから、どれを GND としてもかまいません。正しい接続の例を図 7 に示します。

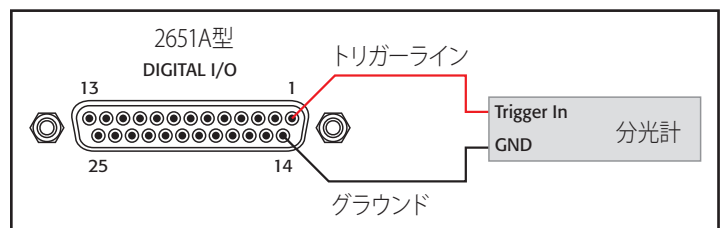


図 7: 2651A 型のデジタル I/O から分光計への接続

記: ソースメータを 2 台使用する構成の場合は、SMU #1 のデジタル I/O のみに接続し、SMU #2 のデジタル I/O は使用しません。

デバイス接続

ソースメータから被試験 LED デバイスへの接続方法を図 8 と図 9 に示します。図 8 はソースメータ 1 台で構成されたシステム、図 9 はソースメータ 2 台構成のシステムの接続を示します。

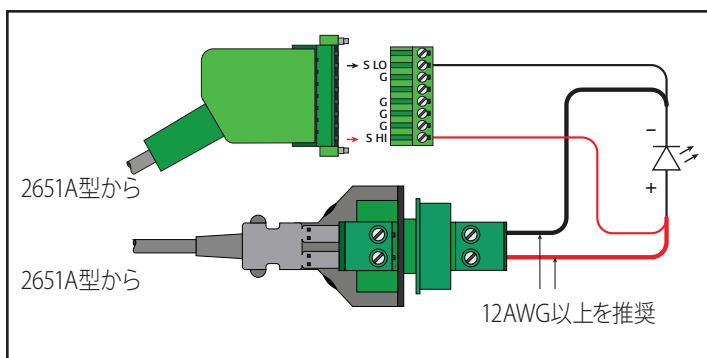


図 8:1 台の 2651A 型ソースメータから被試験 LED デバイスへの接続

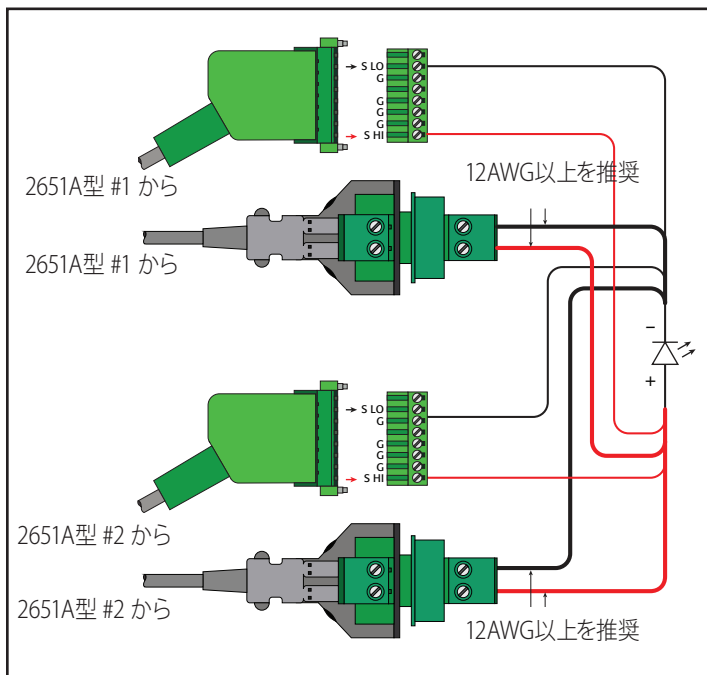


図 9:2 台の 2651A 型ソースメータの構成から被試験 LED デバイスへの接続

2 台の 2651A 型ソースメータを使用するシステムの場合、2 台の SMU を並列接続することによって 2 倍の電流容量を持つ 1 つのソースを構成することができます。このような試験を実施する場合は、“Model 2651A High Power System SourceMeter Instrument Reference Manual” の “Combining SMU outputs (SMU 出力の結合)” セクションの内容を理解してから試験を実行してください。

記: 低インピーダンス/高電流同軸ケーブル (2651A 型に付属) を接続した 2ピン端子ブロックエクステンダから被試験デバイスまでの接続には大きな電流が流れます。この部分の接続には 12 AWG、またはそれよりも太い電線を使用してください。また、インダクタンスを小さくするため、この配線はできるだけ短くしてください。

トリガモデルの構成

2651A 型の DC 動作領域で利用できるレベルを超える電流を印加したい場合は、高度なトリガモデルを使用する必要があります。このトリガモデルを使用することにより、2651A 型の精密なパルス幅と緊密な同期が得られ、正確なパルス幅変調を実行することができます。以下

のセクションでは、このトリガモデルを構成してパルス幅変調波形を出力し、2651A 型から分光計にトリガを掛ける方法を説明します。

1 台の 2651A 型ソースメータを使用するトリガモデルの構成

1 台の 2651A 型を使用して LED のパルス幅変調試験を実施するトリガモデルを図 10 に示します。この構成では、タイマ 1 がパルス周期をコントロールします。タイマ 2 はパルス幅をコントロールし、タイマ 3 はパルスのスタートから測定開始までの遅延時間を決めます。タイマ 4 は波形出力開始から分光計測定開始までの遅延時間を設定します。タイマ 4 の設定時間終了に同期して、デジタル I/O から Trigger 1 が出力され、これが分光計にトリガを掛けます。

2 台の 2651A 型ソースメータを使用するトリガモデルの構成

2 台の 2651A 型ソースメータを使用して LED のパルス幅変調試験を実施するトリガモデルを図 11 に示します。この構成では 2 台の 2651A 型の出力を結合して並列動作させます。この構成では 2651A #1 のタイマ 1 が両方の SMU のパルス周期をコントロールします (具体的には、2651A #1 の出力トリガ信号を TSP-Link Trigger 1 を中継して、2651A #2 へ伝達します)。TSP-Link を使用するトリガの待ち時間は極めて短いですから、2651A #2 は 500ns 以内の遅れ時間で 2651A #1 に同期して動作します。この信号はタイマ 1 の設定時間が終了する度に毎回送信され、2 台の SMU は毎回のパルスサイクル開始時に同期が取られますから、長期的な同期のずれが生ずることはありません。

SMU が 1 台の構成と同じように、タイマ 2 はパルス幅をコントロールし、タイマ 3 はパルスのスタートから測定開始までの時間遅延を決めます。パルス幅と測定遅延時間については、個々の SMU がそれぞれのタイマ 2 とタイマ 3 を使用してコントロールします。タイマの精度が高く、かつ毎回のパルスの先頭で同期が取られることから、両方の SMU のパルス幅の同一性と測定の同時性が保証されます。最後に、ここでも 2651A #1 のタイマ 4 がパルス波形の開始から分光計測定開始までの (デジタル I/O Trigger 1 の出力を遅らせる) 遅延時間を作り出します。

周波数とデューティサイクルの設定

SMU を 1 台または 2 台使用する構成のいずれにおいても、特定の周波数とデューティサイクルを持つ波形を設定するには、そのトリガモデルに適切なパルス間隔とパルス幅を指定しなければなりません。周波数 (f) とパルス間隔 (P) は式 $P = 1 / f$ で関係付けられますから、適切なパルス間隔を設定することによって周波数をコントロールすることができます。たとえば、1kHz の波形であれば、 $P = 1 / 1\text{kHz} = 1\text{ms}$ となります。デューティサイクル (D.C.) は 1 つのパルス間隔 (P) の中でパルスの幅 (PW) が占める割合、すなわち、 $\text{D.C.} = \text{PW} / P$ となります。したがって、パルス幅として適当な値を設定すれば、式 $\text{D.C.} * P = \text{PW}$ を使ってデューティサイクルを計算することができます。

パルス幅の変調

図 10 および図 11 のトリガモデルダイアグラムが示すように、パルス幅はタイマ 2 によってコントロールされ、このタイマは固定されたタイムアウト値を持っています。この関係は ICL コマンドを使用して次のように表現されます

`trigger.timer[2].delay = pulseWidth`

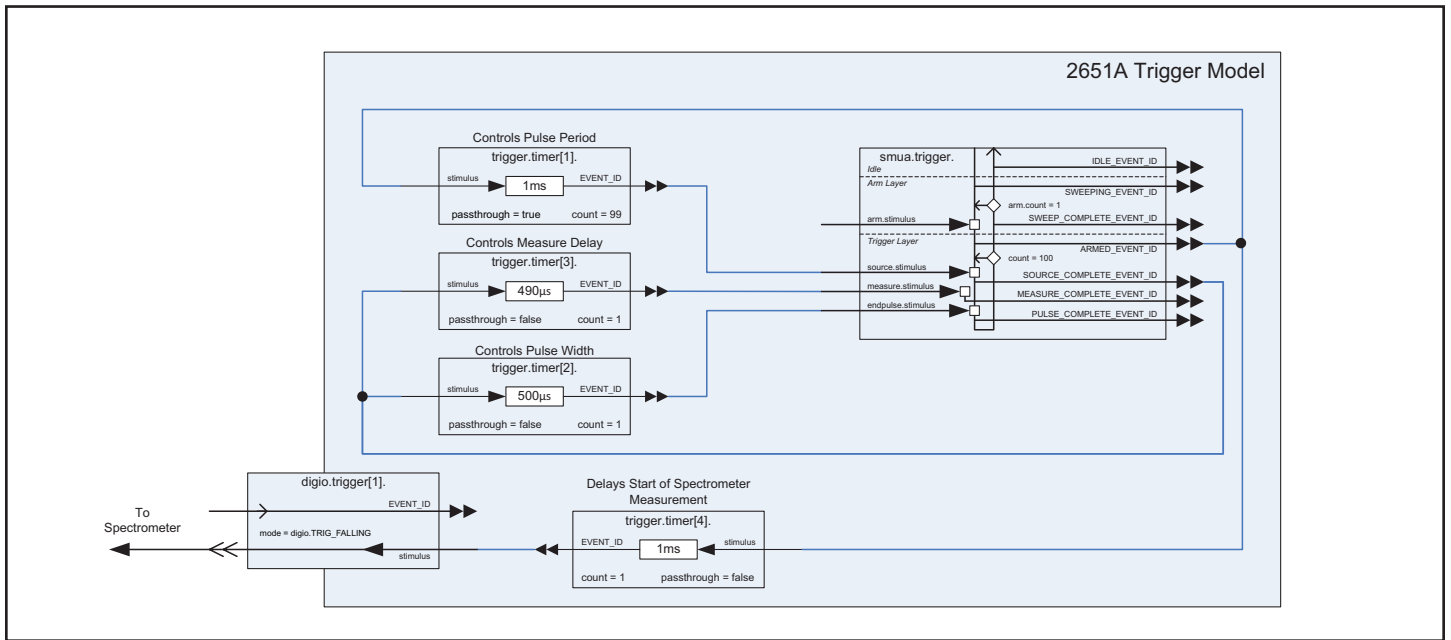


図 10: 1 台の 2651A 型システムソースメータを使用して LED の PWM 試験を行うトリガモデル

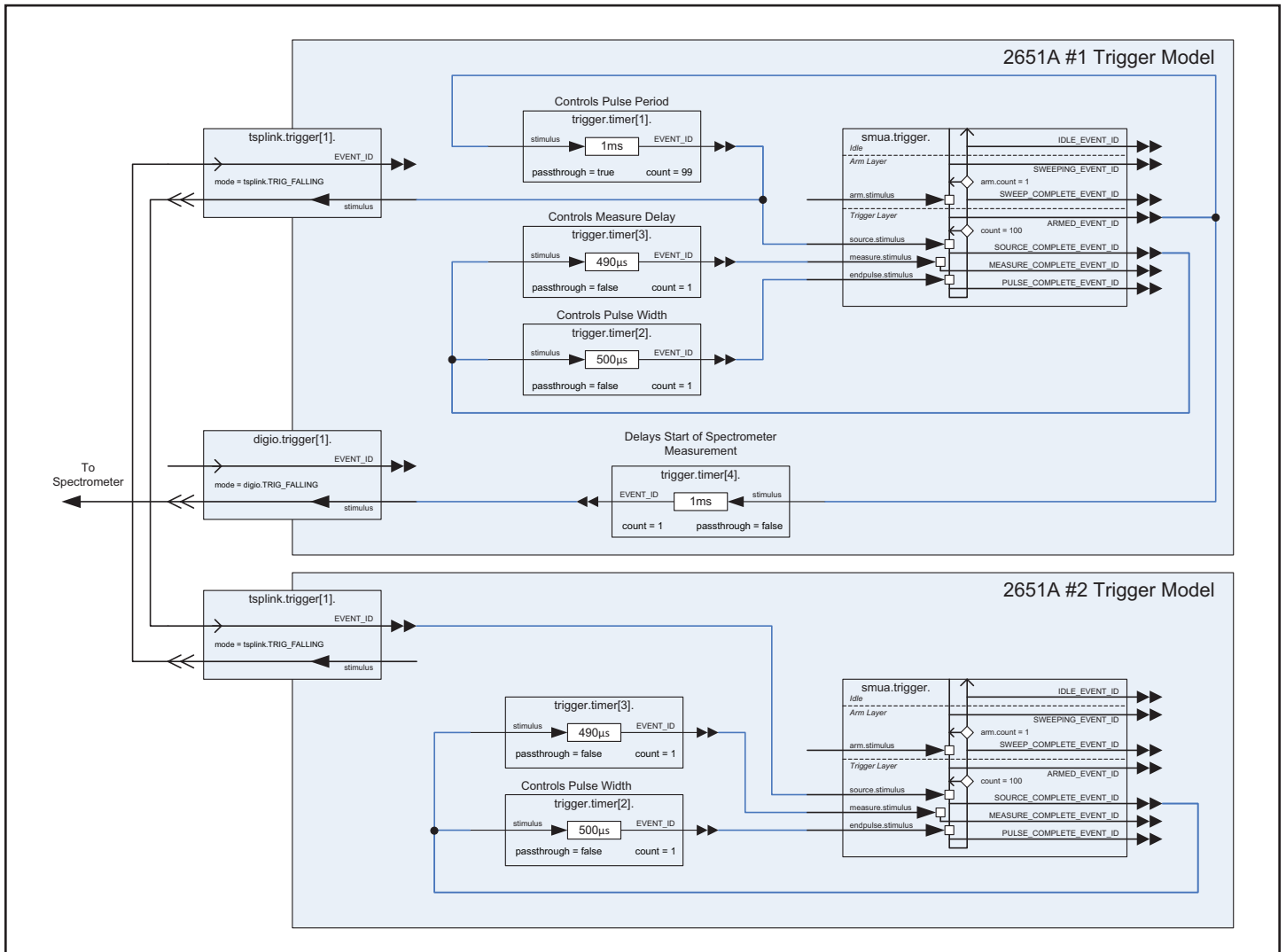


図 11: 2 台の 2651A 型システムソースメータを使用して LED の PWM 試験を行うトリガモデル

ここに、pulseWidth は固定された遅延の値 (秒単位) です。固定されたタイムアウト値を持つということは、1 つの波形出力が続く全範囲のすべてのサイクルにおいてパルス幅が同じ値を持つことを意味します。1 つの波形の出力の途中でパルス幅を変調させるためには、タイム 2 のタイムアウト値が変数でなければなりません。これを実現するため、トリガモデルが使用するタイムは、1 つの値に限定されず、複数の遅延時間リストを割り付けることができます。時間幅を変調するには次の ICL コマンドを呼び出します。

```
trigger.timer[2].delaylist = pulseWidthTable
```

ここに、pulseWidthTable は複数の遅延時間 (秒単位) を格納したテーブルです。タイム 2 に遅延時間リストを割り付けることによって、波形に含まれる個々のパルスに異なるパルス幅を指定することができます、すなわち、波形がパルス幅変調されます。

記: このアプリケーションノートに示されるスクリプト例は固定および可変パルス幅の両方に対応しています。詳しくは関数のドキュメントを参照してください。

プログラムコード例

記: このアプリケーションノートで説明する TSP スクリプトはあくまでも使用法の説明を目的としたものであり、製造スループット向上のために最適化されたものではありません。システムスループット最適化のための検討事項については、ケースレーのアプリケーションエンジニアにお問い合わせください。

記: このアプリケーションノートで紹介する TSP スクリプトは Test Script Builder から実行することを想定しています。それ以外のプログラミング環境、たとえば Microsoft® Visual Studio や National Instruments LabVIEW® から実行可能ですが、その場合には若干の修正が必要となります。

このアプリケーションノートの TSP スクリプトは、高輝度 LED を対象として、光学測定を含むパルス幅変調試験を実行するために必要なすべてのコードを含んでいます。また、2651A 型ハイパワーシステムソースメータを 1 台または 2 台使用するシステムの両方に対応しています。このスクリプトのコードは、付録 A「ソースコード」に記載されています。

スクリプトは以下の機能を実行します:

- TSP-Link 接続の初期化 (ユニットを 2 台使用するシステムにのみ該当)
- SMU のレンジと測定条件の設定
- トリガモデルの設定
- データ読み込みバッファの準備
- PWM 波形を出力
- 収集したデータを計測器コンソールに読み込みます (データ形式は Microsoft Excel® スプレッドシートへの直接コピー/ペーストに対応)。

スクリプトは、インラインコードの単一ブロックを使用するのではなく、TSP 関数を使用して記述されます。TSP 関数は他のプログラム言

語、たとえば C 言語や Visual Basic が提供する関数に類似した形式を持っており、関数が内蔵するコードを実行する前に呼び出さなければなりません。そのため、スクリプトだけを実行しても試験が実施されることにはなりません。試験を実施するには、まずスクリプトを実行して必要な関数を Test Script へローディングしてから関数を呼び出さなければなりません。スクリプトの実行法、および計測器コンソールを使用してコマンドを入力する方法については、Test Script Builder のドキュメントをご覧ください。

スクリプトには、コードラインがどのような操作を実行するのか、およびスクリプト内に含まれる関数を説明したコメント行が含まれています。node[2] で始まるラインは、TSP-Link インタフェースを介して 2651A #2 へ送信されるコマンドです。それ以外のすべてのコマンドは 2651A #1 上で実行されます。

プログラム例の使用法

このスクリプトはパルス幅変調波形を出力するために 2 つの関数を備えています。1 つは 1 台の 2651A 型ハイパワーシステムソースメータで使用するための関数であり、他方は 2 台の 2651A 型ハイパワーシステムソースメータで使用するための関数です。これらの関数は波形を設定するためのパラメータを含んでいるので、ユーザはコードを書き換えることなく、これらのパラメータ値を調節することができます。以下のセクションでは、それぞれの関数とそのパラメータについて説明します。

PWM_Test_Single()

PWM_Test_Single(pulseLevel, pulseLimit, frequency, dutyCycle, numpulses, specDelay)

1 台の 2651A 型ハイパワーソースメータを使用してパルス幅変調波形を出力するための関数です。波形に含まれる毎回のパルスごとに高速 ADC を用いて順方向電圧を取り込みます。この測定はパルスの立下りの $10\mu\text{s}$ 前に実行されます。PWM 波形特性の設定、および、波形の始点から分光計測定開始までの遅延時間設定のために、この関数のパラメータを使用します。パラメータ値が入るべき位置を空白にしておく、デフォルト値が使用されます。

パラメータ	単位	説明
pulseLevel	Amps	試験実行中のパルスの電流レベル 最小値: -50 最大値: +50 デフォルト値: 1 コメント: ここに設定される値によって SMU の動作領域が規定されます。したがって、誤差を発生させずに使用できるデューティサイクルの最大値にも影響を与えます。
pulseLimit	V	試験実行中に使用できるパルスの電圧リミット 最大値: 40 デフォルト値: 1 コメント: ここに設定される値によって SMU の動作領域が規定されます。したがって、誤差を発生させずに使用できるデューティサイクルの最大値にも影響を与えます。
frequency	Hz	1 秒あたりのパルス数 最小値: 0.1 最大値: 10,000 デフォルト値: 100 コメント: 波形のパルス幅は周波数とデューティサイクルによって決定されます。試験に使用する動作範囲が DC 動作領域の外である場合は、誤差を発生させずに使用できる最小周波数がここに指定される値よりも高いことがあります。
dutyCycle	%	パルスのオン時間をパルス周期のパーセント値として表した値 最小値: 0.01 最大値: 99 デフォルト値: 1 コメント: このパラメータには単一の値、もしくは値を列記したテーブルを割り付けます。単一の値を割り付けた場合は、波形のすべてのパルスのデューティサイクルは同一になります。値を列記したテーブルを割り付けた場合は、それぞれのパルスのデューティサイクルはテーブルの対応する項の値によって決定されます。たとえば、テーブルに 50、25、および 40 という値が記載されていたとすると、波形の最初のパルスのデューティサイクルは 50% となり、2 番目および 3 番目のパルスのデューティサイクルはそれぞれ 25%、40% となります。テーブルに指定した値の数を超える個数のパルスが波形に含まれる場合は、テーブルの最後の値に達した後、次のパルスはテーブルの先頭に戻って、テーブルの最初から値を再使用します。上の例で説明すると、もし出力するパルスの数が 5 であったとすれば、パルスのデューティサイクルは次の順で変化します: 50%、25%、40%、50%、および 25% 波形のパルス幅は周波数とデューティサイクルによって決定されます。パルスとデューティサイクルが取り得る値は、SMU がその中で動作しているパワーの動作可能領域に応じて制限されます。試験で使用する動作領域や選択する周波数によっては、誤差を発生させることなく使用可能な最小/最大デューティサイクルの値がここに示す値よりも大きくなることもあり、低くなることもあります。指定されたデューティサイクルを適用した場合に、その結果としてパルス幅/デューティサイクルがその時点での SMU の動作領域に対して過大となるのであれば、SMU はパルス幅/デューティサイクルを SMU の最大許容値内に収まるように制限します。 このような制限が適用された場合、出力波形のパルスが短縮される、あるいは欠落するといった事象が発生します。 最大パルス幅とデューティサイクルの値については、2651A 型の製品仕様を参照してください。
numpulses	N/A	波形が含むパルス数 最小値: 2 最大値: 100,000 以上 デフォルト値: 10
specDelay	秒	PWM 出力開始から分光計測定開始 (デジタル I/O トリガ発生) までの時間。 デフォルト値: 0

関数 PWM_Test_Single() の呼び出し例を次に示します:

PWM_Test_Single(30, 10, 1000, 50, 100, 1e-3)

このパラメータ設定で呼び出された関数は、パルスレベル 30A、電圧リミット 10V、周波数 1kHz、デューティサイクル 50% のパルス 100 個で構成されたパルス幅変調波形を出力します。波形出力開始後 1ms が経過した時点で分光計の測定が始まります。試験が完了すると、SMU は出力を停止し、試験中に収集した順方向電圧の値を装置コンソールに表示します (Microsoft Excel スプレッドシートにコピー/ペースト可能な形式で表示されます)。この出力例を図 12 に示します。

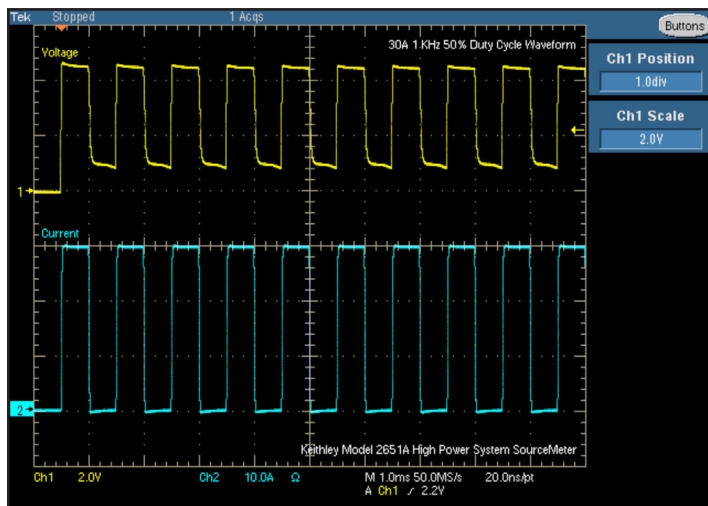


図 12: ケースレー 2651A 型がハイパワー LED モジュールに供給するパルス波形の例: 30A、1kHz、デューティサイクル 50%

先に示す例では、波形のデューティサイクルが 50% に固定されています。パルスごとにパルス幅が変化する波形を作り出す場合は、デューティサイクルテーブルを関数に引き渡さなければなりません。次に示す関数呼び出しがその例です。

```
dutyTable = {20, 40, 60, 80, 60, 40, 20, 40, 60 }
```

```
PWM_Test_single(20, 10, 1000, dutyTable, 9, 1e-3)
```

このパラメータ設定で呼び出された関数は、パルスレベル 20A、電圧リミット 10V、周波数 1kHz のデューティサイクルが変化するパルス 9 個で構成されたパルス幅変調波形を出力します。テーブルから読み取った値が個々のパルスのデューティサイクルを決定します。最初のパルスのデューティサイクルは 20%、2 番目のパルスのデューティサイクルは 40%、3 番目のパルスのデューティサイクルは 60% という具合です。波形出力開始後 1ms が経過した時点で分光計の測定

が始まります。試験が完了すると、SMU は出力を停止し、試験中に収集した順方向電圧の値を装置コンソールに表示します (Microsoft Excel へコピー/ペースト可能な形式で表示されます)。この出力例を図 13 に示します。

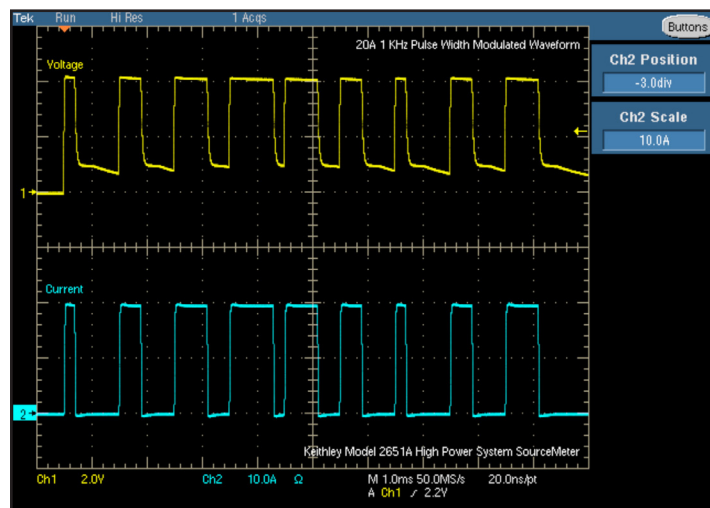


図 13: ケースレー 2651A 型がハイパワー LED モジュールに供給するパルス幅変調波形の例: 20A、1kHz

PWM_Test_Dual()

PWM_Test_Dual(pulseLevel, pulseLimit, frequency, dutyCycle, numpulses, specDelay)

この関数は、2 台の 2651A 型ハイパワーシステムソースメータを使用して、1 台で可能な電流出力の 2 倍の大きさを持つパルス幅変調波形を出力します。

この関数が使用する 2 台の 2651A 型ハイパワーシステムソースメータは TSP-Link を介して相互に接続されています。SMU の出力を並列接続することによって、1 台の 2651A 型で被試験デバイスに供給可能な量の 2 倍の電流を出力します。1 台の SMU を使用する場合と同様に、波形に含まれる毎回のパルスごとに高速 ADC を使用して順方向電圧を取り込みます。電圧値の読み込みはパルスの立下りの 10 μ s 前に実行されます。PWM 波形特性の設定、および、波形の始点から分光計測定開始までの遅延時間設定のために、この関数のパラメータを使用します。パラメータ値が入るべき位置を空白にしておくと、デフォルト値が使用されます。

パラメータ	単位	説明
pulseLevel	A	試験実行中のパルスの電流レベル 最小値: -100 最大値: +100 デフォルト値: 1 コメント: ここに設定される値によって SMU の動作領域が規定されます。したがって、誤差を発生させずに使用できるデューティサイクルの最大値にも影響を与えます。
pulseLimit	V	試験実行中に使用できるパルスの電圧リミット 最大値: 40 デフォルト値: 1 コメント: ここに設定される値によって SMU の動作領域が規定されます。したがって、誤差を発生させずに使用できるデューティサイクルの最大値にも影響を与えます。
frequency	Hz	1 秒あたりのパルス数 最小値: 0.1 最大値: 10,000 デフォルト値: 100 コメント: 波形のパルス幅は周波数とデューティサイクルによって決定されます。試験に使用する動作範囲が DC 動作領域の外である場合は、誤差を発生させずに使用できる最小周波数がここに指定される値よりも高いことがあります。
dutyCycle	%	パルスのオン時間をパルス周期のパーセント値として表した値 最小値: 0.01 最大値: 99 デフォルト値: 1 コメント: このパラメータには単一の値、もしくは値を列記したテーブルを割り付けます。単一の値を割り付けた場合は、波形のすべてのパルスのデューティサイクルが同一になります。値を列記したテーブルを割り付けた場合、それぞれのパルスのデューティサイクルはテーブルの対応する項の値によって決定されます。たとえば、テーブルに 50、25、および 40 という値が記載されていたとすると、波形の最初のパルスのデューティサイクルは 50% となり、2 番目および 3 番目のパルスのデューティサイクルはそれぞれ 25%、40% となります。テーブルに指定した値の数を超える個数のパルスが波形に含まれる場合は、テーブルの最後の値に達した後、次のパルスはテーブルの先頭に戻って、テーブルの最初から値を再使用します。上の例で説明すると、もし出力するパルスの数が 5 であったとすれば、パルスのデューティサイクルは次の順で変化します: 50%、25%、40%、50%、および 25% 波形のパルス幅は周波数とデューティサイクルによって決定されます。装置のパルスとデューティサイクルが取り得る値は、SMU がその中で動作しているパワー包絡線の領域に応じて制限されます。試験で使用する動作領域や選択する周波数によっては、誤差を発生させることなく使用可能な最小/最大デューティサイクルの値がここに示す値よりも高くなることもあり、低くなることもあります。指定されたデューティサイクルを適用した場合に、その結果としてパルス幅/デューティサイクルがその時点での SMU の動作領域に対して過大となるのであれば、SMU はパルス幅/デューティサイクルを SMU の最大許容値内に収まるように制限します。このような制限が適用された場合、出力波形のパルスが短縮される、あるいは欠落するといった事象が発生します。最大パルス幅とデューティサイクルの値については 2651A 型の製品仕様を参照してください。
numpulses	N/A	波形が含むパルス数 最小値: 2 最大値: 100,000 以上 デフォルト値: 10
specDelay	秒	PWM 出力開始から分光計測定開始 (デジタル I/O トリガ発生) までの時間。 最小値: 0 デフォルト値: 0

関数 PWM_Test_Dual() の呼び出し例を次に示します:

```
PWM_Test_Dual(60, 10, 1000, 50, 100, 1e-3)
```

このパラメータ設定で呼び出された関数は、パルスレベル 60A、電圧リミット 10V、周波数 1kHz、デューティサイクル 50% のパルス 100 個で構成されたパルス幅変調波形を出力します。波形出力開始後 1ms が経過した時点で分光計の測定が始まります。試験が完了すると、SMU は出力を停止し、試験中に収集した順方向電圧の値を装置コンソールに表示します (Microsoft Excel へコピー/ペースト可能な形式で表示されます)。この出力例を図 14 に示します。

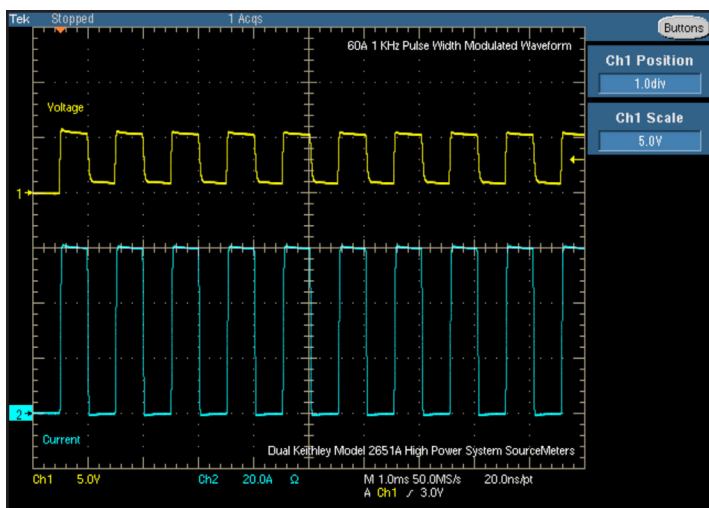


図 14: 2 台の2651A 型を使用してハイパワー LED モジュールに供給するパルス波形の例: 60A、1kHz、デューティサイクル 50%

1 台の SMU の場合と同様に、この関数を使用して可変デューティサイクルを発生させることができます。パルスごとにパルス幅が変化する波形を作り出すためには、デューティサイクルテーブルを関数に引き渡さなければなりません。次に示す関数呼び出しがその例です。

```
dutyTable = {20, 40, 60, 80, 60, 40, 20, 40, 60}
```

```
PWM_Test_Dual(40, 10, 1000, dutyTable, 9, 1e-3)
```

このパラメータ設定で呼び出された関数は、パルスレベル 40A、電圧リミット 10V、周波数 1kHz のデューティサイクルが変化するパルス 9 個で構成されたパルス幅変調波形を出力します。テーブルから読み取った値が個々のパルスのデューティサイクルを決定します。最初のパルスのデューティサイクルは 20%、2 番目のパルスのデューティサイクルは 40%、3 番目のパルスのデューティサイクルは 60% という具合です。波形出力開始後 1ms が経過した時点で分光計の測定が始まります。試験が完了すると、SMU は出力を停止し、試験中に収集した順方向電圧の値を装置コンソールに表示します (Microsoft Excel ヘコピー/ペースト可能な形式で表示されます)。この出力例を図 15 に示します。

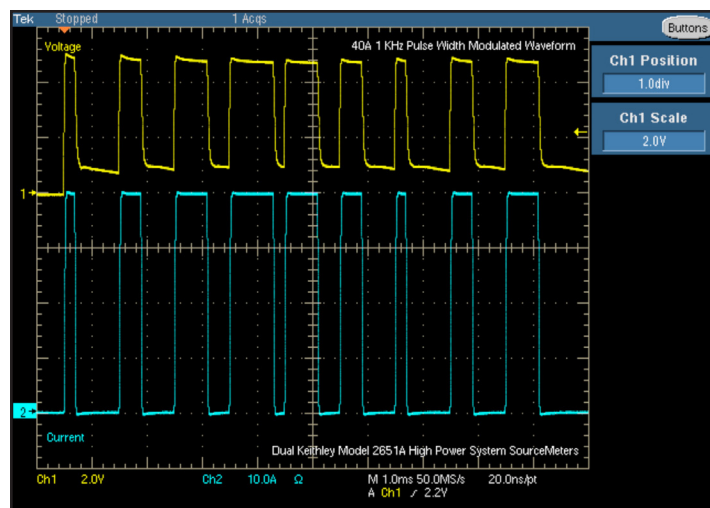


図 15: 2 台の2651A 型を使用してハイパワー LED モジュールに供給するパルス幅変調波形の例: 40A、1kHz

結論

HBLED は驚くほどの速さで進歩を遂げつつあり、製造メーカはこれを将来における光源の主流とすべく多大の努力を払っています。しかし、この目的を達するために LED 製造メーカは幾つかのハードルを飛び越えなければなりません。LED の製造コストをさらに低下させると同時に、効率の向上と光出力のさらなる増大を図らなければなりません。このような目標の達成を図る上で中心的な役割を果たすのが、精密で信頼性が高く、再現性に優れた印加/測定装置です。製造メーカはこのような条件を満たし、かつ、変化し続ける試験ニーズに対応できるパワーと柔軟性を備えた試験装置を必要としています。

2600A 型システムソースメータシリーズは、LED 製造メーカの変化する試験ニーズに余裕を持って追いつく性能と柔軟性を備えています。高度トリガモデルを始めとする革新的な機能を持つ 2600A 型シリーズは精密かつ優れた信頼性で反復測定を行い、パルス幅変調波形や AC 波形、さらに任意形状波形などの複雑な駆動方法に対応します。この製品ラインに新たに 2651A 型が加わりました。この製品は、従来からの低電流測定確度を維持しながら、現在最高の輝度を持つハイパワー LED モジュールにも対応できるようにパワーが強化されています。2600A 型システムソースメータシリーズは、柔軟性に富む出力機能に加えて、1 台の装置で精密な印加と測定操作を実行します。まさに、HBLED 試験のための理想的な選択肢と言えます。

付録A:ソースコード

記:このスクリプトのプログラムコードは、2651A型ハイパワーシステムソースメータであれば、そのまま動作させられます。2600Aシリーズシステムソースメータの他の測定器でも若干の変更をすることで動作させられます。

```
--[[
Title:          Pulse Width Modulation Script
Description:    The purpose of this script is to generate a pulse width
                modulated waveform for use in testing High Brightness LED modules.
                Users of this script should call the functions in the User Functions
                section.  Functions in the Utility Functions section are used by the
                User Functions to execute the test.

System Setup:
  PWM_Test_Single()
    1x Model 2651A
  PWM_TEST_Dual()
    2x Model 2651A
    1x TSP-Link Cable

    Node 1: 2651A #1 (Master)
    Node 2: 2651A #2 (Slave)
]]--

=====
-- User Functions
=====
--[[ PWM_Test_Single()

    This function uses a single SMU to output a pulse width modulated waveform.
--]]
function PWM_Test_Single(pulseLevel, pulseLimit, frequency, dutyCycle, numPulses, specDelay)
    if (pulseLevel == nil) then pulseLevel = 1 end
    if (pulseLimit == nil) then pulseLimit = 1 end
    if (frequency == nil) then frequency = 100 end
    if (dutyCycle == nil) then dutyCycle = 1 end
    if (numPulses == nil) then numPulses = 10 end
    if (specDelay == nil) then specDelay = 0 end
    local pulsePeriod
    local pulseWidth
    local measDelay
    -- Calculate the timing parameters from the frequency and duty cycle
    pulsePeriod,pulseWidth,measDelay = CalculateTiming(frequency, dutyCycle)

    -- Do a quick check on the input parameters
    f,msg = SimpleRegionCheck(pulseLevel, pulseLimit, dutyCycle, 1)
    if (f == false) then
        print(msg)
        quit()
    end

    reset()
    smua.reset()
    smua.source.func          = smua.OUTPUT_DCAMPS
    smua.sense                = smua.SENSE_REMOTE
    smua.source.autorangei    = 0
    smua.source.rangei        = pulseLevel
    smua.source.leveli        = 0
    -- Set the DC bias limit.  This is not the limit used during the pulses.
    smua.source.limitv        = 1

    smua.measure.autozero     = smua.AUTOZERO_ONCE
```

```

smua.measure.autorangev          = 0
smua.measure.rangev              = pulseLimit
-- The fast ADC allows us to place the measurements very close to the falling edge of
-- the pulse allowing for settled measurements even when pulse widths are very small
smua.measure.adc                  = smua.ADC_FAST
smua.measure.count                = 1
smua.measure.interval            = 1e-6
-- Uncomment the following lines to turn on measure filtering.  When enabled, the SMU
-- will take multiple measurements and average them to produce a single reading.
-- Because the Fast ADC can take one measurement every microsecond, several measurements
-- can be aquired in a small time to produce an averaged reading.
--smua.measure.filter.count       = 5
--smua.measure.filter.enable      = smua.FILTER_ON

-- This measure delay sets the delay between the measurement trigger being received
-- and when the actual measurement(s) start.  This is set to 0 because we will be
-- delaying the trigger itself and do not need additional delay.
smua.measure.delay                = 0

-- Setup the Reading Buffers
smua.nvbuffer1.clear()
smua.nvbuffer1.appendmode         = 1
smua.nvbuffer1.collecttimestamps= 1
smua.nvbuffer2.clear()
smua.nvbuffer2.appendmode         = 1
smua.nvbuffer2.collecttimestamps= 1

-- Configure the Trigger Model
=====

-- Timer 1 controls the pulse period
trigger.timer[1].count            = numPulses > 1 and numPulses - 1 or 1
trigger.timer[1].delay            = pulsePeriod
trigger.timer[1].passthrough= true
trigger.timer[1].stimulus         = smua.trigger.ARMED_EVENT_ID

-- Timer 2 controls the pulse width
trigger.timer[2].count            = 1
if (type(pulseWidth) == "table") then
    -- Use a delay list if the duty cycle will vary for each pulse
    trigger.timer[2].delaylist     = pulseWidth
else
    -- else every pulse will be the same duty cycle
    trigger.timer[2].delay         = pulseWidth
end
trigger.timer[2].passthrough= false
trigger.timer[2].stimulus         = smua.trigger.SOURCE_COMPLETE_EVENT_ID

-- Timer 3 controls the measurement
trigger.timer[3].count            = 1
if (type(measDelay) == "table") then
    -- If the duty cycle is variable then the measure delay will be as well
    trigger.timer[3].delaylist     = measDelay
else
    trigger.timer[3].delay         = measDelay
end
trigger.timer[3].passthrough= false
trigger.timer[3].stimulus         = smua.trigger.SOURCE_COMPLETE_EVENT_ID

-- Configure SMU Trigger Model for Sweep
smua.trigger.source.linear1(pulseLevel, pulseLevel, numPulses)
smua.trigger.source.limitv       = pulseLimit

```

```

smua.trigger.measure.action          = smua.ASYNC
smua.trigger.measure.iv(smua.nvbuffer1, smua.nvbuffer2)
smua.trigger.endpulse.action= smua.SOURCE_IDLE
smua.trigger.endsweep.action= smua.SOURCE_IDLE
smua.trigger.count                   = numPulses
smua.trigger.arm.stimulus             = 0
smua.trigger.source.stimulus= trigger.timer[1].EVENT_ID
smua.trigger.measure.stimulus        = trigger.timer[3].EVENT_ID
smua.trigger.endpulse.stimulus       = trigger.timer[2].EVENT_ID
smua.trigger.source.action            = smua.ENABLE

-- Configure the Digital I/O trigger
ConfigureSpectrometerTrigger(specDelay)

-- Start the Test
=====
-- Turn the output on
smua.source.output                   = 1
-- Start the trigger model execution
smua.trigger.initiate()

-- While the trigger model is outputting the waveform and collecting the
-- measurements, the script will scan the status model for any overruns
-- that may occur as a result of using improper settings.
local ovr = false
local msg = ""
while ((status.operation.sweeping.condition ~= 0) and (ovr == false)) do
    ovr, msg = CheckForOverRun(localnode)
end
if (ovr == true) then
    smua.abort()
    print(msg)
end
-- Turn the output off
smua.source.output                   = 0
-- Return the data
PrintData()
end

--[ PWM_Test_Dual()

This function uses two SMUs connected together in parallel to output a pulse width
modulated waveform. By using two SMUs higher current levels/duty cycles can be achieved.
--]]
function PWM_Test_Dual(pulseLevel, pulseLimit, frequency, dutyCycle, numPulses, specDelay)
    if (pulseLevel == nil) then pulseLevel = 1 end
    if (pulseLimit == nil) then pulseLimit = 1 end
    if (frequency == nil) then frequency = 100 end
    if (dutyCycle == nil) then dutyCycle = 1 end
    if (numPulses == nil) then numPulses = 10 end
    if (specDelay == nil) then specDelay = 0 end

    local pulsePeriod
    local pulseWidth
    local measDelay

    -- Calculate the timing parameters from the frequency and duty cycle
    pulsePeriod,pulseWidth,measDelay = CalculateTiming(frequency, dutyCycle)

    -- Do a quick check on the input parameters
    f,msg = SimpleRegionCheck(pulseLevel, pulseLimit, dutyCycle, 2)
    if (f == false) then

```

```

    print(msg)
    quit()
end

-- Initialize the TSP-Link
errorqueue.clear()
tsplink.reset()
errcode,errmsg,stat = errorqueue.next()
if (errcode ~= 0) then
    print(errmsg)
    exit()
end
reset()
ConfigureLocalSMU(pulseLevel, pulseLimit, pulsePeriod, pulseWidth, measDelay, numPulses)
ConfigureRemoteSMU(pulseLevel, pulseLimit, pulsePeriod, pulseWidth, measDelay, numPulses)

-- Start the Test
=====
-- Turn the output on
smua.source.output = 1
node[2].smua.source.output = 1
-- Start the trigger model execution
node[2].smua.trigger.initiate()
smua.trigger.initiate()

-- While the trigger model is outputting the waveform and collecting the
-- measurements, the script will scan the status model for any overruns
-- that may occur as a result of using improper settings.
local ovr1 = false
local ovr2 = false
local msg1 = ""
local msg2 = ""
-- Loop until the sweep is either complete, or an overrun condition is detected
while (((status.operation.sweeping.condition ~= 0) or (node[2].status.operation.sweeping.condition ~=
0)) and (ovr1 == false) and (ovr2 == false)) do
    ovr1, msg1 = CheckForOverRun(localnode)
    ovr2, msg2 = CheckForOverRun(node[2])
end
if ((ovr1 == true) or (ovr2 == true)) then
    smua.abort()
    node[2].smua.abort()
    print("SMU#1:", msg1)
    print("SMU#2:", msg2)
end
-- Turn the output off
node[2].smua.source.output = 0
smua.source.output = 0
-- Return the data
PrintDataDual()
end

=====
-- Utility Functions
=====
function ConfigureLocalSMU(pulseLevel, pulseLimit, pulsePeriod, pulseWidth, measDelay, numPulses)
    smua.reset()
    smua.source.func = smua.OUTPUT_DCAMPS
    smua.sense = smua.SENSE_REMOTE
    smua.source.autorangei = 0
    smua.source.rangei = pulseLevel/2
    smua.source.leveli = 0
end

```

```

-- Set the DC bias limit.  This is not the limit used during the pulses.
smua.source.limitv                = 1
smua.source.offmode                = smua.OUTPUT_NORMAL
smua.source.offfunc                = smua.OUTPUT_DCVOLTS
smua.source.offlimiti              = 1e-3

smua.measure.autozero              = smua.AUTOZERO_ONCE
smua.measure.autorangev            = 0
smua.measure.rangev                = pulseLimit
-- The fast ADC allows us to place the measurements very close to the falling edge of
-- the pulse allowing for settled measurements even when pulse widths are very small
smua.measure.adc                   = smua.ADC_FAST
smua.measure.count                  = 1
smua.measure.interval              = 1e-6
-- Uncomment the following lines to turn on measure filtering.  When enabled, the SMU
-- will take multiple measurements and average them to produce a single reading.
-- Because the Fast ADC can take one measurement every microsecond, several measurements
-- can be aquired in a small time to produce an averaged reading.
--smua.measure.filter.count         = 5
--smua.measure.filter.enable        = smua.FILTER_ON

-- This measure delay sets the delay between the measurement trigger being received
-- and when the actual measurement(s) start.  This is set to 0 because we will be
-- delaying the trigger itself and do not need additional delay.
smua.measure.delay                  = 0

-- Setup the Reading Buffers
smua.nvbuffer1.clear()
smua.nvbuffer1.appendmode           = 1
smua.nvbuffer1.collecttimestamps= 1
smua.nvbuffer2.clear()
smua.nvbuffer2.appendmode           = 1
smua.nvbuffer2.collecttimestamps= 1

-- Configure the Trigger Model
=====

-- Timer 1 controls the pulse period
trigger.timer[1].count              = (numPulses > 1) and numPulses - 1 or 1
trigger.timer[1].delay               = pulsePeriod
trigger.timer[1].passthrough= true
trigger.timer[1].stimulus            = smua.trigger.ARMED_EVENT_ID

-- Timer 2 controls the pulse width
trigger.timer[2].count              = 1
if (type(pulseWidth) == "table") then
    -- Use a delay list if the duty cycle will vary for each pulse
    trigger.timer[2].delaylist       = pulseWidth
else
    -- else every pulse will be the same duty cycle
    trigger.timer[2].delay            = pulseWidth
end
trigger.timer[2].passthrough= false
trigger.timer[2].stimulus            = smua.trigger.SOURCE_COMPLETE_EVENT_ID

-- Timer 3 controls the measurement delay
trigger.timer[3].count              = 1
if (type(measDelay) == "table") then
    -- If the duty cycle is variable then the measure delay will be as well
    trigger.timer[3].delaylist       = measDelay
else
    trigger.timer[3].delay            = measDelay

```

```

end
trigger.timer[3].passthrough= false
trigger.timer[3].stimulus      = smua.trigger.SOURCE_COMPLETE_EVENT_ID

-- TSP-Link Trigger 1 is used to synchronize the SMUs by telling
-- the second SMU when to pulse.
tsplink.trigger[1].clear()
tsplink.trigger[1].mode        = tsplink.TRIG_FALLING
tsplink.trigger[1].stimulus    = trigger.timer[1].EVENT_ID

-- Configure SMU Trigger Model for Sweep
smua.trigger.source.linear(pulseLevel/2, pulseLevel/2, numPulses)
smua.trigger.source.limitv     = pulseLimit
smua.trigger.measure.action     = smua.ASYNC
smua.trigger.measure.iv(smua.nvbuffer1, smua.nvbuffer2)
smua.trigger.endpulse.action= smua.SOURCE_IDLE
smua.trigger.endsweep.action= smua.SOURCE_IDLE
smua.trigger.count             = numPulses
smua.trigger.arm.stimulus      = 0
smua.trigger.source.stimulus= trigger.timer[1].EVENT_ID
smua.trigger.measure.stimulus = trigger.timer[3].EVENT_ID
smua.trigger.endpulse.stimulus = trigger.timer[2].EVENT_ID
smua.trigger.source.action     = smua.ENABLE
end

function ConfigureRemoteSMU(pulseLevel, pulseLimit, pulsePeriod, pulseWidth, measDelay, numPulses)
node[2].smua.reset()
node[2].smua.source.func          = node[2].smua.OUTPUT_DCAMPS
node[2].smua.sense                 = node[2].smua.SENSE_REMOTE
node[2].smua.source.autorangei     = 0
node[2].smua.source.rangei        = pulseLevel/2
node[2].smua.source.leveli        = 0
-- Set the DC bias limit. This is not the limit used during the pulses.
node[2].smua.source.limitv        = 1
node[2].smua.source.offmode       = node[2].smua.OUTPUT_NORMAL
node[2].smua.source.offfunc       = node[2].smua.OUTPUT_DCAMPS
node[2].smua.source.offlimitv     = 40

node[2].smua.measure.autozero     = node[2].smua.AUTOZERO_ONCE
node[2].smua.measure.autorangev   = 0
node[2].smua.measure.rangev      = pulseLimit
-- The fast ADC allows us to place the measurements very close to the falling edge of
-- the pulse allowing for settled measurements even when pulse widths are very small
node[2].smua.measure.adc          = node[2].smua.ADC_FAST
node[2].smua.measure.count        = 1
node[2].smua.measure.interval     = 1e-6
-- Uncomment the following lines to turn on measure filtering. When enabled, the SMU
-- will take multiple measurements and average them to produce a single reading.
-- Because the Fast ADC can take one measurement every microsecond, several measurements
-- can be aquired in a small time to produce an averaged reading.
--node[2].smua.measure.filter.count = 5
--node[2].smua.measure.filter.enable = node[2].smua.FILTER_ON

-- This measure delay sets the delay between the measurement trigger being received
-- and when the actual measurement(s) start. This is set to 0 because we will be
-- delaying the trigger itself and do not need additional delay.
node[2].smua.measure.delay        = 0

-- Setup the Reading Buffers
node[2].smua.nvbuffer1.clear()
node[2].smua.nvbuffer1.appendmode = 1
node[2].smua.nvbuffer1.collecttimestamps= 1

```

```

node[2].smua.nvbuffer2.clear()
node[2].smua.nvbuffer2.appendmode      = 1
node[2].smua.nvbuffer2.collecttimestamps= 1

-- Configure the Trigger Model
-----

-- Timer 2 controls the pulse width
node[2].trigger.timer[2].count          = 1
if (type(pulseWidth) == "table") then
  -- Use a delay list if the duty cycle will vary for each pulse
  node[2].trigger.timer[2].delaylist= pulseWidth
else
  -- else every pulse will be the same duty cycle
  node[2].trigger.timer[2].delay        = pulseWidth
end
node[2].trigger.timer[2].passthrough = false
node[2].trigger.timer[2].stimulus     = node[2].smua.trigger.SOURCE_COMPLETE_EVENT_ID

-- Timer 3 controls the measurement delay
node[2].trigger.timer[3].count          = 1
if (type(measDelay) == "table") then
  -- If the duty cycle is variable then the measure delay will be as well
  node[2].trigger.timer[3].delaylist= measDelay
else
  node[2].trigger.timer[3].delay        = measDelay
end
node[2].trigger.timer[3].passthrough = false
node[2].trigger.timer[3].stimulus     = node[2].smua.trigger.SOURCE_COMPLETE_EVENT_ID

-- TSP-Link Trigger 1 is used to synchronize the SMUs.  SMU #2 receives
-- its trigger to pulse from SMU #1
node[2].tsplink.trigger[1].clear()
node[2].tsplink.trigger[1].mode        = node[2].tsplink.TRIG_FALLING
-- Release the trigger line when the pulse is complete
node[2].tsplink.trigger[1].stimulus    = 0

-- Configure SMU Trigger Model for Sweep
node[2].smua.trigger.source.lineari(pulseLevel/2, pulseLevel/2, numPulses)
node[2].smua.trigger.source.limitv     = pulseLimit
node[2].smua.trigger.measure.action    = node[2].smua.ASYNC
node[2].smua.trigger.measure.iv(node[2].smua.nvbuffer1, node[2].smua.nvbuffer2)
node[2].smua.trigger.endpulse.action   = node[2].smua.SOURCE_IDLE
node[2].smua.trigger.endsweep.action   = node[2].smua.SOURCE_IDLE
node[2].smua.trigger.count              = numPulses
node[2].smua.trigger.arm.stimulus       = 0
node[2].smua.trigger.source.stimulus    = node[2].tsplink.trigger[1].EVENT_ID
node[2].smua.trigger.measure.stimulus   = node[2].trigger.timer[3].EVENT_ID
node[2].smua.trigger.endpulse.stimulus  = node[2].trigger.timer[2].EVENT_ID
node[2].smua.trigger.source.action      = node[2].smua.ENABLE
end

function ConfigureSpectrometerTrigger(specDelay)
  -- Digital I/O line 1 triggers the spectrometer measurements
  -- Timer 4 puts a delay between the start of the pulse train and the
  -- output of the digital IO trigger on Digital I/O line 1
  digio.trigger[1].clear()
  digio.trigger[1].mode                  = digio.TRIG_FALLING

  -- If the delay value is > 0 then configure a timer to provide the delay
  if specDelay > 0 then
    trigger.timer[4].count                = 1
  end
end

```

```

trigger.timer[4].delay           = specDelay
trigger.timer[4].passthrough    = false
trigger.timer[4].stimulus       = smua.trigger.ARMED_EVENT_ID

digio.trigger[1].stimulus       = trigger.timer[4].EVENT_ID
else
  -- Else bypass the timer and trigger the digital I/O immediately
  -- Configure the Digital I/O pin that will trigger the spectrometer
  digio.trigger[1].stimulus     = smua.trigger.ARMED_EVENT_ID
end
end
end

function CheckForOverRun(pNode)
  -- Check SMUA Trigger Overruns
  if (bit.band(pNode.status.operation.instrument.smua.trigger_overrun.condition, 2) == 2) then
    return true, "smua arm trigger is overrun"
  end
  if (bit.band(pNode.status.operation.instrument.smua.trigger_overrun.condition, 4) == 4) then
    return true, "smua source trigger is overrun"
  end
  if (bit.band(pNode.status.operation.instrument.smua.trigger_overrun.condition, 8) == 8) then
    return true, "smua measure trigger is overrun"
  end
  if (bit.band(pNode.status.operation.instrument.smua.trigger_overrun.condition, 16) == 16) then
    return true, "smua endpulse trigger is overrun"
  end
  end

  local CFORi = 0
  -- Check Timers for Overrun
  if (pNode.status.operation.instrument.trigger_timer.trigger_overrun.condition > 0) then
    return true, string.format("Timer trigger is overrun: 0x%x", CFORi)
  end
  end

  -- Check Blenders for Overrun
  if (pNode.status.operation.instrument.trigger_blender.trigger_overrun.condition > 0) then
    return true, string.format("blender trigger is overrun: 0x%x", CFORi)
  end
  end

  -- Check TSP-Link Triggers for Overrun
  if (pNode.status.operation.instrument.tsplink.trigger_overrun.condition > 0) then
    return true, string.format("TSP-Link trigger is overrun: 0x%x", CFORi)
  end
  end

  -- Check DIGIO Triggers for Overrun
  if (pNode.status.operation.instrument.digio.trigger_overrun.condition > 0) then
    return true, string.format("digio trigger is overrun: 0x%x", CFORi)
  end
  end

  -- Check LAN Triggers for Overrun
  if (pNode.status.operation.instrument.lan.trigger_overrun.condition > 0) then
    return true, string.format("LAN trigger is overrun: 0x%x", CFORi)
  end
  end

  return false, "no overrun detected"
end

function PrintData()
  print("Timestamp\tVoltage\tCurrent")
  for i=1,smua.nvbuffer1.n do
    print(smua.nvbuffer1.timestamps[i], smua.nvbuffer2[i], smua.nvbuffer1[i])
  end
end
end

```

```

function PrintDataDual()
    local voltage
    local current
    print("Timestamp\tVoltage\tCurrent")
    for i=1,smua.nvbuffer1.n do
        voltage = (smua.nvbuffer2[i] + node[2].smua.nvbuffer2[i])/2
        current = smua.nvbuffer1[i] + node[2].smua.nvbuffer1[i]
        print(smua.nvbuffer1.timestamps[i], voltage, current)
    end
end

function CalculateTiming(frequency, dutyCycle)
    local pulsePeriod = 1/frequency
    local pulseWidth
    local measDelay

    -- If duty cycle was a table then we need to create delay lists for the timers
    if (type(dutyCycle)=="table") then
        pulseWidth = {}
        measDelay = {}
        for i=1,table.getn(dutyCycle) do
            if ((dutyCycle[i] > 99) or (dutyCycle[i] < 0.01)) then
                print(string.format("Error: dutyCycle[%d] must be between 0.01% and 99%.", i))
                exit()
            end
            -- Calculate pulse width from period and duty cycle. Subtract 3us of overhead
            pulseWidth[i] = pulsePeriod * (dutyCycle[i]/100) - 3e-6
            -- Set measure delay so measurement happen 10us before the falling edge of the pulse
            measDelay[i] = pulseWidth[i] - 10e-6
        end
    else -- Duty cycle was a single value so we only need a single delay value for the timers
        if ((dutyCycle > 99) or (dutyCycle < 0.01)) then
            print("Error: dutyCycle must be between 0.01% and 99%.")
            exit()
        end
        pulseWidth = pulsePeriod * (dutyCycle/100) - 3e-6
        measDelay = pulseWidth - 10e-6
    end
    return pulsePeriod, pulseWidth, measDelay
end

function SimpleRegionCheck(pulseLevel, pulseLimit, dutyCycle, SMUs)
    -- This function only serves as a quick check that the entered parameters are
    -- within the max allowable duty cycles for the operating regions. This function
    -- does not check that the pulse widths are within the maximums as well.

    local pLev = math.abs(pulseLevel)
    f = true
    msg = "Checks passed."
    if ((pulseLimit >= 10e-3) and (pulseLimit <= 10)) then
        if ((pLev > 30*SMUs) and (dutyCycle > 35)) then
            msg = string.format("Duty Cycle too high for pulse region 5. Duty cycle must be 35% or less
for pulse levels above %dA.", 30*SMUs)
            f = false
        elseif ((pLev > 20*SMUs) and (pLev <= 30*SMUs) and (dutyCycle > 50)) then
            msg = string.format("Duty Cycle too high for pulse region 2. Duty cycle must be 50% or less
for pulse levels between %dA and %dA.", 20*SMUs, 30*SMUs)
            f = false
        end
    elseif ((pulseLimit > 10) and (pulseLimit <= 20)) then
        if ((pLev > 20*SMUs) and (dutyCycle > 10)) then

```

```

    msg = string.format("Duty Cycle too high for pulse region 6. Duty cycle must be 10%% or less
for pulse levels above %dA.", 20*SMUs)
    f = false
    elseif ((pLev > 10*SMUs) and (pLev <= 20*SMUs)) and (dutyCycle > 40)) then
        msg = string.format("Duty Cycle too high for pulse region 3. Duty cycle must be 40%% or less
for pulse levels between %dA and %dA.", 10*SMUs, 20*SMUs)
        f = false
    end
    elseif (pulseLimit > 20) and (pulseLimit <= 40) then
        if ((pLev > 10*SMUs) and (dutyCycle > 1)) then
            msg = string.format("Duty Cycle too high for pulse region 7. Duty cycle must be 1%% or less
for pulse levels above %dA.", 10*SMUs)
            f = false
            elseif ((pLev > 5*SMUs) and (pLev <= 10*SMUs)) and (dutyCycle > 40)) then
                msg = string.format("Duty Cycle too high for pulse region 4. Duty cycle must be 40%% or less
for pulse levels between %dA and %dA.", 5*SMUs, 10*SMUs)
                f = false
            end
        else
            msg = "Error: pulseLimit out of range. pulseLimit must be between 10mV and 40V."
            f = false
        end
    end

    return f,msg
end

```

```

--PWM_Test_Single(1, 2, 100, 1, 10, 0)
-- duty = {20, 40, 60, 80, 60, 40, 20, 40, 60}
--PWM_Test_Single(20, 10, 1000, duty, 9, 1e-3)
--PWM_Test_Dual(40, 10, 1000, duty, 9, 1e-3)

```

KEITHLEY

ケースレーインストゥルメンツ株式会社

本社 ■ 〒105-0022 ■ 東京都港区海岸1-11-1 ニューピア竹芝ノースタワー13F ■ TEL: 03-5733-7555 ■ FAX: 03-5733-7556
大阪オフィス ■ 〒564-0052 ■ TEL: 06-6396-1630 ■ FAX: 06-6396-1634
■ Web site : www.keithley.jp ■ Email : info.jp@keithley.com

© Copyright 2010 Keithley Instruments, Inc
Printed in China