KEITHLEY

Application Note Series

Number 3129

使用2651A高功率数字源表进行高亮度LED的脉宽调制测试

引言

随着全世界都重视节约能源，传统光源的高功耗受到了严格的审查而且政府要求提高光源的能源效率。提高能源效率的要求导致了大量的投资以及开发白炽灯泡的替代品。紧凑型荧光灯（CFL）灯泡已在市场上流行，虽然它们比传统灯泡效率更高更耐用，但是它们仍然不是理想的替代品。但是，高亮度发光二极管（HBLED）已经证实是一种更好的替代品。像白炽灯泡一样，HBLED能立即达到满亮度而且不含有任何难以处理的化学制品。相对于白炽灯，HBLED还具有下列优点：寿命很长而且效率能不断提高。

遗憾的是，虽然HBLED的能耗大大降低，但是HBLED的价格还是太贵了，大多数消费者不得不转而选择更便宜的技术。为了降低HBLED的价格，制造商不断努力提高产量，进一步提高效率水平。为实现此目标需要正确地测试，而正确地测试需要使用适合的测试设备。现在，为了正确测试高亮度LED，制造商要求测试设备具有很多功能。此应用笔记研究了一些电气测试要求以及如何用2651A高功率数字源表满足这些测试要求。

对更高功率的要求

HBLED通常定义为功率在1W及以上、工作范围典型值从1W至3W的LED。工作电流不是10～30mA而且没有采用小型3mm或5mm塑料圆顶封装，HBLED工作电流为300mA～1A或更高，安装在小型、热传导板上以便于LED结散热。不管一颗HBLED的亮度有多大，对于大多数照明应用而言，一颗HBLED的亮度通常不够。相反，通常将多颗HBLED组合制作一个灯具，或者作为改进应用的LED灯泡或作为整体照明装置。即使HBLED在实际应用中组合使用，但是通常在独立封装级上进行生产测试，因而需要适度的功率传输能力，这恰好是基于仪器的最先进源-测量单元（SMU）的功能之一，例如吉时利2400系列和2600A数字源表。

对于许多应用而言，如果希望光线朝多个方向传播而且灯具有足够的空间容纳多颗LED时，可以将多颗LED组成一个灯具。但是，在空间有限和/或光线必须定向的应用中，这种方法要么不满足要求要么不能工作。用小封装实现高亮度的需求推动了由一个或多个大管芯LED构成的高功率LED模块的发展。当存在多个管芯时，这些管芯采用并联连接或者串联连接，这取决于应用和可用电源。这些LED的管芯比典型HBLED的管芯大得多，还能承受更大电流。通常一个管芯要求承受的电流幅度高达10A。

正确测试高功率HBLED模块要求测试设备将大量功率传至DUT。虽然SMU是测试LED的最佳设备，假设一台SMU仪器具有源和测量功能，市面上多数SMU都不能提供所需的功率电平。高功率HBLED模块通常需要100W以上的功率，但是大多数基于仪器的SMU仅能提供40W以下的功率。吉时利的2651A高功率数字源表能提供高达200W连续直流功率以及高达2000W的脉冲功率，从而在测试现在和未来高功率模块应用中都绰绰有余（图1）。

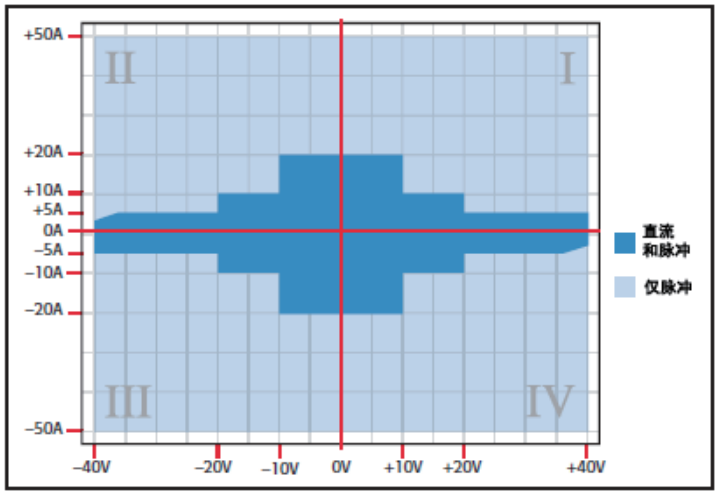

图1：2651A高功率数字源表的功率范围。

脉宽调制

脉宽调制是控制LED亮度的常规方法。采用此技术时，通过LED的电流脉冲具有固定的频率和固定的脉冲电平，但是脉冲宽度在变化（见图2）。脉冲宽度的变化改变了LED的导通态时间和能感觉的亮度水平。在此驱动方案中，LED实际上是闪烁的，但由于闪烁的频率非常高，人眼无法辨别这与恒定亮度级的区别。虽然通过降低正向驱动电流就能简单地控制LED的亮度，但是由于一系列的原因，脉宽调制仍然是一种更合适的技术。

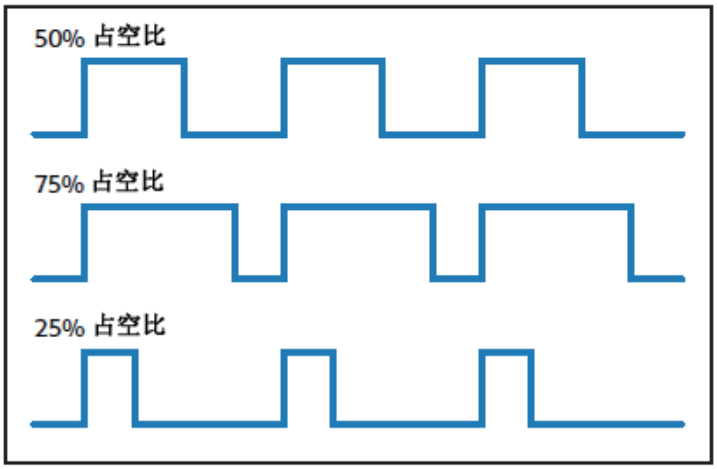

图2：在脉宽调制中，脉冲电平和脉冲频率保持恒定，但是占空比变化。

使用脉宽调制的首要、可以说是最重要的原因是在降低LED亮度时能保持光线颜色的一致性。在LED中，发光颜色与LED工作的导通电压有关。虽然随着导通电流的变化LED的导通电压将保持相对稳定，但实际上导通电压会改变几十至几百毫伏。在较低

电流电平上尤其如此（见图3）。导通电压的轻微变化相当于光线色彩的轻微变化，这是最终用户不想要的。如果忽略热效应，在脉宽调制技术中，向LED施加的每个脉冲都采用完全相等的电流幅度，所以导通电压对于每个脉冲都相等；因此，发光颜色不会变化。

脉宽调制更合适的另一个重要原因是此技术能对亮度进行线性控制。LED的发光量与驱动电流大小并非线性相关。换句话说，驱动电流降低50%将不会使光输出降低50%；而是降低另一个不同的光输出量。这会使基于改变电流的调光方案很难应用，因为必须分析每颗LED的光输出与导通电流的关系才能根据此关系曲线校准驱动方案。使用脉宽调制更容易实现对亮度的线性控制。使用脉宽调制，为了使LED的输出亮度变为50%，需要做的就是将占空比减小50%。如果LED导通时间缩短一半，那么产生的亮度将减少一半。

功率效率是脉宽调制法的另一个优点。因为脉宽调制对每个脉冲都使用固定的电流幅度，所以可以选用LED工作效率最高（即每瓦流明最大输出的位置）的脉冲幅度。这就意味着，无论采用何种亮度级，LED的工作效率都最大。脉宽调制提高效率的另一种方法是对于给定的驱动电流，用脉冲方式代替直流方式将使LED实际输出亮度更高。许多厂商的datasheet提供导通电流与光通量的关系曲线图。如果厂商分析了LED在脉冲驱动电流和直流驱动电流条件下的特性，我们就会看到脉冲特性曲线位于直流特性曲线之上。这是因为脉冲驱动电流降低了自发热。最后，脉宽调制甚至能提高驱动电路的功率效率。脉宽调制采用的开关电路功率损耗极低。当开关电路关断时，基本上没有电流并且功耗为零。当开关闭合时，由于导通态电阻很小，几乎全部功率都传递至LED而且开关功耗很低。在可变电流驱动方案中，通常在电路中功耗较大的部分降低了传至LED的功率。

采用2651A高功率数字源表的脉宽调制

注意：本文介绍的采用2651A输出脉宽调制波形的技术适用于2600A系列数字源表的全部产品。

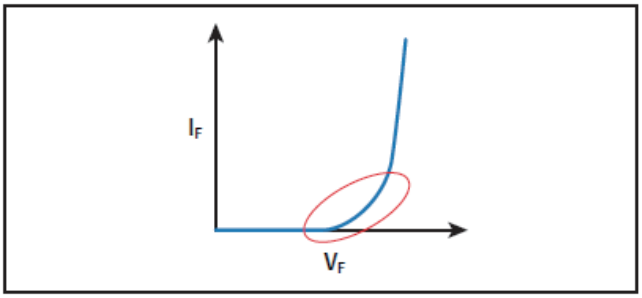

图3：当导通电流较低时导通电压会显著变化，并且当导通电流较高时导通电压比较稳定。

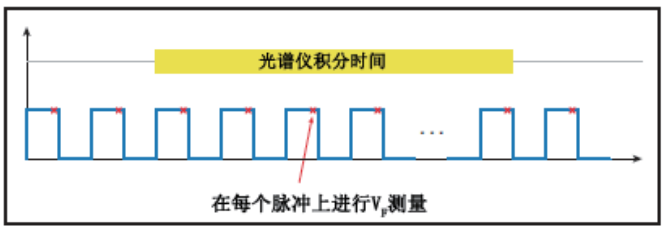

图4：向LED施加一系列脉冲并在每个脉冲上测量V_F。光谱仪同时进行光学测量。

假设LED常常配合脉宽调制使用，那么LED只适于用脉宽调制技术测试。作为脉宽调制测试的一部分，LED通常被施加一系列脉冲，同时用光谱仪测量在多个脉冲输入后光输出的积分。这种测量需要用几十或几百毫秒完成。在脉冲输出过程中，在每个脉冲上测量导通电压可以了解LED温度升高带来的变化。图4示出了此项测试。

2651A高功率数字源表能输出100%占空比0～20A电流、50%占空比20～30A电流、35%占空比30～50A电流的脉宽调制波形。2651A的先进触发器模型支持精密脉冲宽度和占空比以及与其它测量仪器的紧密同步。可以利用这些同步特性将两台2651A合并一起使用以实现占空比不变而脉冲幅值为单台2651A的2倍。

此笔记详细介绍了如何配置2651A输出30A、50%占空比波形和数字I/O输出触发器用于触发光谱仪。此外，还详细介绍了如何在同一个测试中组合两台2651A 将电流提高至60A。

所需设备

此项测试需要使用以下设备：

- 带GPIB或以太网适配器的PC
- GPIB电缆或RJ45 LAN交叉网线
- 2651A高功率数字源表
- 2651A-KIT-1低阻抗/大电流同轴电缆
- 8针信号控制电缆
- 2针终端模块延长器配合2651A-KIT-1使用
- 8针终端模块延长器配合8针信号控制电缆使用
- 数字I/O DB-25公头连接器硬件包
- 12 AWG或更粗的电缆用于终端模块和器件的连接

测试高达100A电流35%占空比或者60A电流50%占空比需要增加以下设备：

- 2651A高功率数字源表
- 2651A-KIT-1低阻抗/大电流同轴电缆
- 8针信号控制电缆
- 2针扩展端子配合2651A-KIT-1使用
- 8针扩展端子配合8针信号控制电缆使用
- TSP-Link RJ45 LAN交叉线

通信

单台数字源表

为了用一台2651A数字源表配置完成测试，应通过GPIB或以太网将测量仪器连接至计算机，如图5所示。

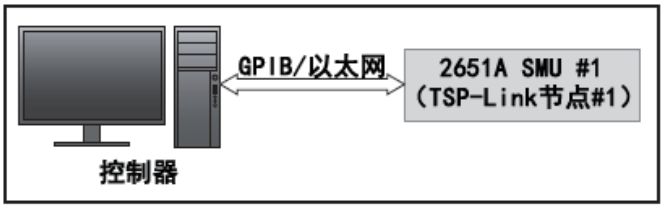

图5：单台数字源表的通信设置。

两台数字源表

对于两台2651A数字源表的配置，应将第一台2651A通过GPIB或以太网连接到计算机。再将第二台2651A通过TSP-Link连接至第一台2651A。将第一台2651A指定为节点#1并将第二台2651A指定为节点#2。连接如图6所示。

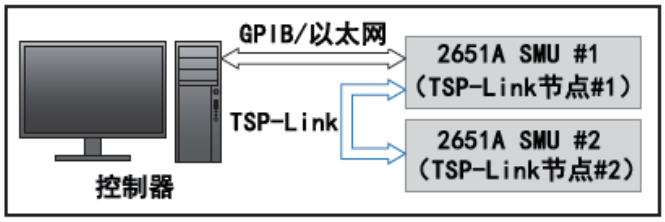

图6：两台2651A的通信设置。

将光谱仪连接至数字I/O

为了用2651A触发光谱仪，光谱仪的测试触发线开始必须连接至2651A的数字I/O端口。将光谱仪的触发线连接到2651A背板上25针D-Sub连接器的#1针。可在15至21针上找到接地连接。正确连接如图7所示。

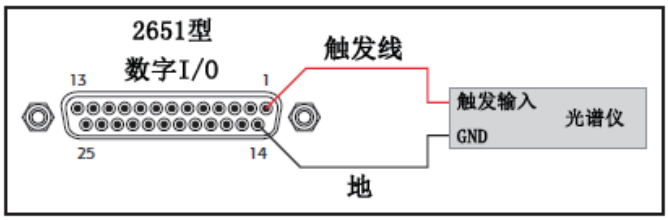

图7：从2651A数字I/O端口连接至光谱仪。

注意： 在两台数字源表的配置中，仅连接至SMU #1的数字I/O端口。SMU #2的数字I/O端口未使用。

设备连接

图8和图9举例说明了数字源表和LED被测器件的连接。图8示出了一台数字源表的连接；图9示出了两台数字源表的连接。

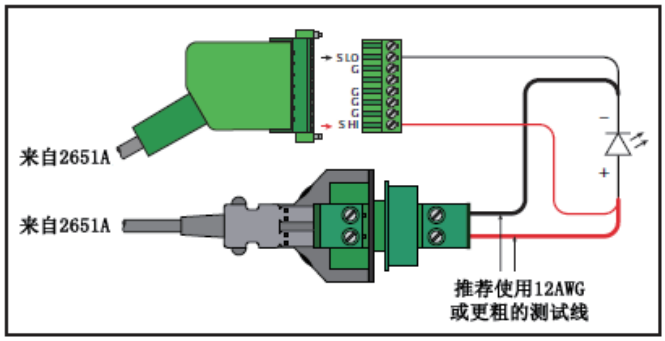

图8：一台2651A数字源表和LED被测器件的连接。

当使用两台2651A数字源表时，两台SMU并联可以实现具有两倍电流容量的一个源。在进行此项测试之前，请阅读并理解2651A高功率数字源表参考手册中题为“*组合SMU输出*”的部分。

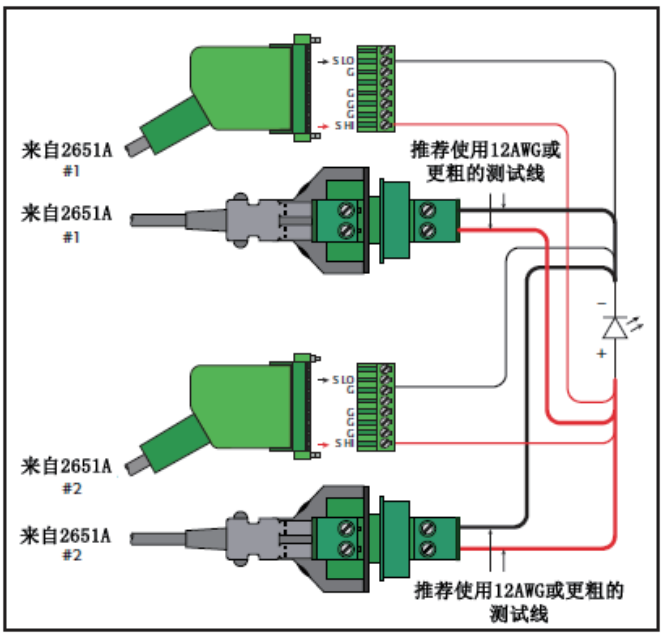

图9：两台2651A数字源表和LED被测器件的连接。

注意： 连接至2651A的低阻抗/大电流同轴电缆一端的2针扩展端子与被测器件的连线应当采用12AWG或更粗的线来支持更大电流幅度。而且，这些走线的长度应当最短使电感最小。

配置触发模型

先进的触发模型的源电流幅度必须高于2651A规格指标中的直流工作范围。同样的触发模型为2651A提供精密脉冲宽度和严格定时从而使准确的脉宽调制成为可能。接下来的内容举例说明了如何配置触发模型输出脉宽调制波形并用2651A触发光谱仪。

一台2651A源表的触发模型配置

图10示出的触发模型将用一台2651A对LED进行脉宽调制测试。在此配置中，定时器1控制脉冲周期，定时器2控制脉冲宽度并且定时器3在脉冲开始和测量开始之间插入一个延时。定时器4在波形输出开始和光谱仪测量开始之间建立一个延时。当定时器4时间一到，它就会触发数字I/O触发器1发送触发信号启动光谱仪。

两台2651A源表的触发模型配置

图11示出的触发模型将用输出并联的两台2651A数字源表对LED进行脉宽调制测试。2651A #1的定时器1用于控制两台SMU的脉冲周期。脉冲周期的控制通过继电器控制TSP-Link®触发器1输出的触发信号，然后将触发信号发送至2651A #2。由于TSP-Link触发的延时极短，2651A #2和2651A #1的同步能保持在500ns之内。因为每当定时器1时间一到就发送一次触发信号，由于在每个脉冲周期开始同步两台SMU因而确保了两台SMU之间的长期同步。

与一台SMU的配置相同，定时器2控制脉冲宽度，定时器3在脉冲开始和测量开始之间插入一个延时。SMU使用各自的定时器2和定时器3控制脉冲宽度和测量延时。

如果给出定时器的准确度并且在每个脉冲的开始进行同步，那么毫无疑问两台SMU将输出相等宽度的脉冲并且同时进行测量。最终，通过延迟数字I/O触发器1的输出，2651A #1的定时器4又在脉冲波形开始和光谱仪测量开始之间插入一个延时。

配置频率和占空比

在单和双SMU配置中，配置波形的特定频率和占空比需要在触发模型中设置适当的脉冲周期和脉冲宽度。频率（f）与脉冲周期（P）的关系式为P=1/f；因此，可以通过设定适当的脉冲周期来控制波形的频率。对于1kHz波形，P=1/1kHz=1ms。占空比（D.C.）是脉冲宽度（PW）与脉冲周期（P）的比值或者D.C.=PW/P；因此，可以设置适当的脉冲宽度来设定占空比；脉冲宽度值可以通过等式D.C.*P=PW计算。

控制脉冲宽度

图10和11的触发模型图示出了由定时器2（具有固定超时值）控制的脉冲宽度。通过调用ICL命令控制脉冲宽度用代码表示为

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

其中pulseWidth是用秒表示的固定延时值。具有固定超时值意味着在一个波形输出中每个周期的脉冲宽度与波形的总长度都是相等的。控制对一个波形输出的脉冲宽度进行调制要求定时器2的超时值可变。为了方便，先进触发模型的定时器可以指定为一个延时表而不是一个值。这可以通过调用ICL命令实现。

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

其中，pulseWidthTable是包含多个延时值（单位：秒）的一个表。由于延时表被分配给定时器2，在波形上的每个脉冲可以指定一个不同的脉冲宽度；因此，可以控制波形的脉冲宽度。

注意： 在此应用笔记中的示例脚本支持固定和可变脉冲宽度。请参见函数文件了解详细信息。

示例程序代码

注意： 在此应用笔记中的测试脚本处理器（TSP®）脚本仅用于示范，还未针对最快生产产量进行优化。请联络吉时利应用工程师了解系统吞吐量优化的考虑。

注意： 此应用笔记的TSP脚本通过测试脚本生成器运行。此脚本还可以通过其它编程环境运行，例如Microsoft® Visual Studio或国家仪器的LabVIEW®；但是可能需要修改。

此应用笔记提供的TSP脚本包含了用一台或两台2651A高功率数字源表进行高亮度LED光学测量和脉宽调制测试所需的全部代码。此脚本的代码可以在附录A：源代码中找到：

此脚本能执行以下功能：

- 初始化TSP-Link连接（仅当使用两台设备时）
- 配置SMU的量程和测量设置
- 配置触发模型
- 准备读数据缓冲
- 输出PWM波形
- 采集数据返回至测量仪器控制台，这些数据可以直接拷贝和粘贴为Microsoft Excel®电子表格的格式。

此脚本的编写采用TSP函数而不是一行一行的代码段。TSP函数类似于其它编程语言（例如C或Visual Basic）的函数，而且必须先调用函数才会运行函数中的代码。因此，单独运行脚本将不会执行测试。为了执行测试，首先要运行脚本将函数加载到测试脚本存储器然后调用函数。请参考测试脚本生成器文件了解如何运行脚本并且用测量仪器控制台输入命令。

在此脚本中，几个注释说明了这些代码的功能，以及脚本中的函数文件。始于

```
node[2].
```

的命令行是通过TSP-Link接口发送到2651A #2的命令。其它全部命令在2651A #1上运行。

程序使用举例

此脚本包含两个函数用于输出脉宽调制波形：一个函数配合一台2651A高功率数字源表使用，另一个函数配合两台2651A高功率数字源表使用。这两个函数包含的参数值用于配置波形，便于用户设定波形特性（例如频率和占空比）无需重写任何代码。下面内容解释了每个函数和参数。

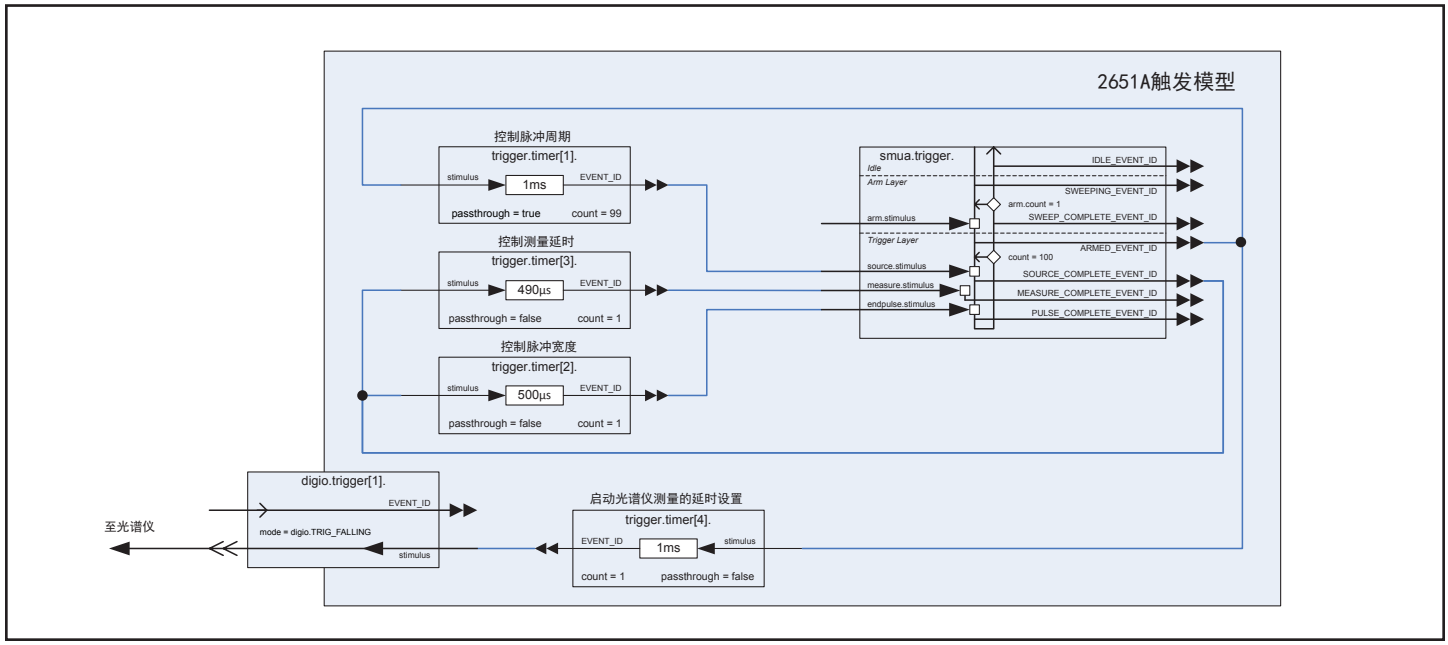

图10：用一台2651A数字源表实现LED脉宽调制测试的触发模型。

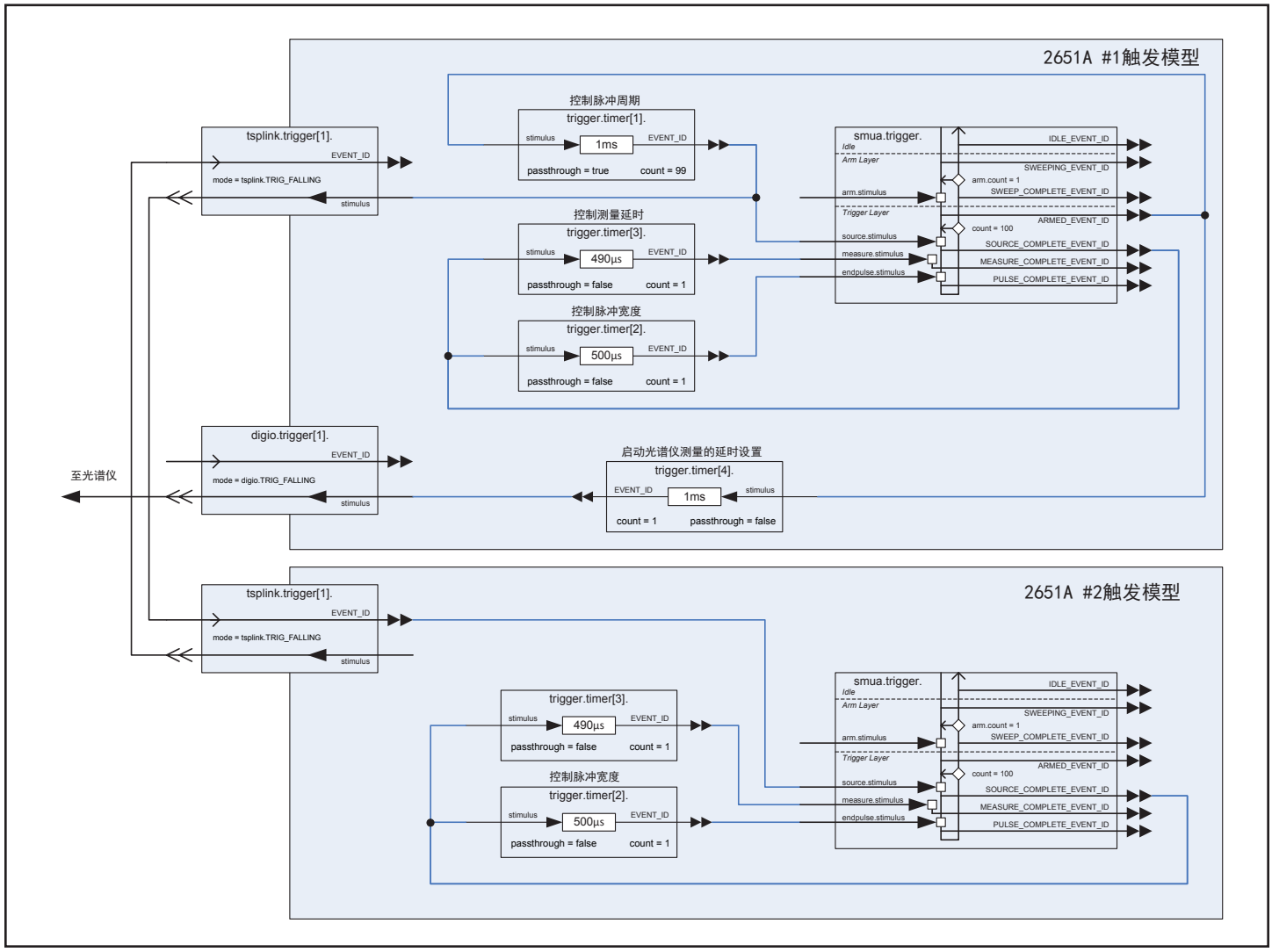

图11：用两台2651A数字源表实现LED脉宽调制测试的触发模型。

PWM_Test_Single()

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

这个函数将使用一台2651A高功率数字源表输出脉宽调制波形。针对波形的每个脉冲用快速ADC测量导通电压，并且在脉冲下降沿之前10 µs进行测量。用此函数的参数来配置脉宽调制波形特性，并且在波形开始和光谱仪测量开始之间插入延时。如果参数留空，将使用缺省值。

参数	单位	说明
pulseLevel	安培	测试期间的脉冲电流幅度 最小值：-50 最大值：+50 缺省值：1 注释： 这个设置值用于确定SMU的工作范围，因而会影响在没有误差条件下可以设定的最大占空比。
pulseLimit	伏特	测试期间的脉冲电压极限 最大值：40 缺省值：1 注释： 这个设置值用于确定SMU的工作范围，因而会影响在没有误差条件下可以设定的最大占空比。
frequency	Hz	每秒脉冲数 最小值：0.1 最大值：10,000 缺省值：100 注释： 频率和占空比决定了波形的脉冲宽度。如果测试的工作范围位于直流工作范围之外，没有误差的最小工作频率可能比 如上的最小值高。
dutyCycle	%	脉冲导通时间占脉冲周期的百分比 最小值：0.01 最大值：99 缺省值：1 注释： 此参数可以指定为一个值或数值表。如果指定为一个值，那么波形上全部脉冲的占空比都相等。如果此参数指定为数值表，那么每个脉冲的占空比将由表中对应元素决定。例如，如果表的元素分别为50、25和40，那么波形中第一脉冲的占空比为50%，第二脉冲的占空比为25%，第三脉冲的占空比为40%。如果波形的脉冲个数多于表的元素个数，那么当达到表的最后一个元素时，下一个脉冲将从表的第一个元素开始重用。借用前面的例子，如果输出脉冲个数为5，那么这些脉冲的占空比将分别为50%、25%、40%、50%和25%。 频率和占空比决定了波形的脉冲宽度。测量仪器的脉冲和占空比极限取决于SMU工作的功率包络范围。没有使用误差的占空比最小值和最大值可能比 如上值高或低，这取决于测试的工作范围和所选频率。如果指定占空比得到的脉冲宽度/占空比对于工作范围而言太大了，那么SMU会将脉冲宽度/占空比限制在最大容许值。这将在输出波形中表现为脉冲截断或脉冲丢失。 参见2651A说明书，了解最大脉冲宽度和占空比的详细内容。
numpulses	N/A	波形的脉冲数量 最小值：2 最大值：100,000或更多 缺省值：10
specDelay	秒	从PWM输出开始到数字I/O触发器输出启动光谱仪测量之间的时间。 最小值：0 缺省值：0

下面是调用PWM_Test_Single()函数的例子。

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

这个函数调用将输出含有100个脉冲的脉宽调制波形，其中脉冲电平为30A、电压限幅10V，频率为1kHz并且占空比为50%。波形输出开始1ms后启动光谱仪测量。测试完成后，SMU输出将被关断并且将测试过程中采集的导通电压测量值以兼容Microsoft Excel拷贝、粘贴的格式打印到仪器控制台。输出的例子如图12所示。

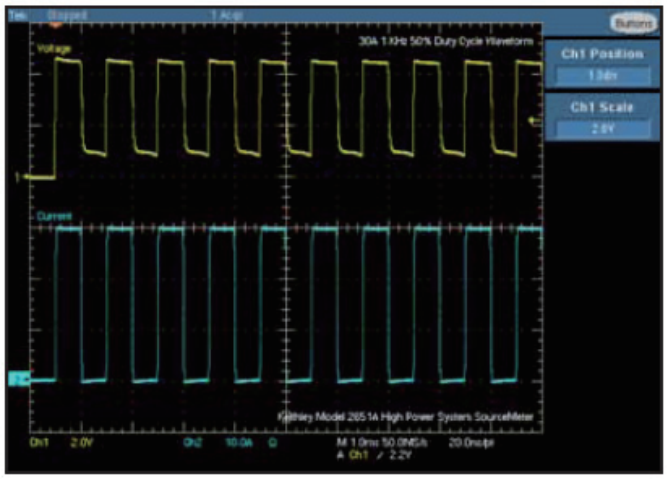

图12：50%占空比的30A 1kHz脉冲波形，从吉时利2651A输出至高功率LED模块。

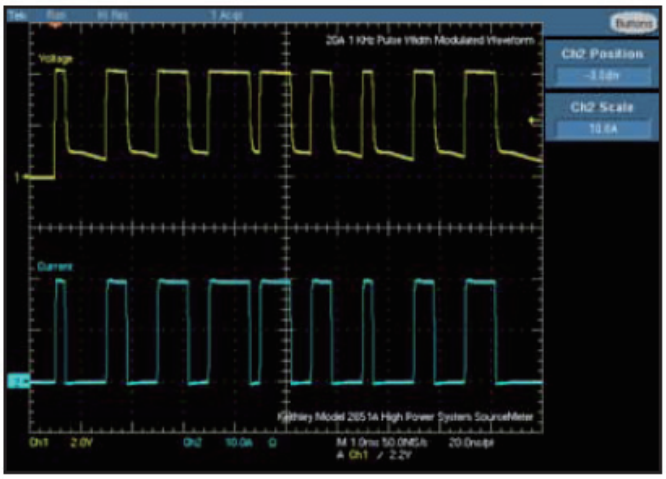

图13：20A 1kHz脉宽调制波形，从吉时利2651A输出至高功率LED模块。

在前面的例子中，波形的占空比固定为50%。为了建立每个脉冲宽度都可变的波形，需要向函数传递占空比的表。用下面的函数调用举例说明。

```
dutyTable={20, 40, 60, 80, 60, 40, 20, 40, 60}
PWM_Test_single(20, 10, 1000, dutyTable, 9, 1e-3)
```

这个函数调用将输出包含9个脉冲的脉宽调制波形，其中脉冲电平为20A、电压限幅10V，频率为1kHz并且占空比可变。每个脉冲的占空比从表中读取。第一个脉冲的占空比为20%，第二个脉冲的占空比为40%，第三个脉冲的占空比为60%等等。波形输出开始1ms后启动光谱仪测量。完成测试后，SMU输出将被关断并且将测试过程中采集的导通电压测量值以兼容Microsoft Excel拷贝、粘贴的格式打印到仪器控制台。输出的例子如图13所示。

PWM_Test_Dual()

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

此函数使用2台2651A高功率数字源表输出脉宽调制波形，输出电流是一台2651A的两倍。

此函数将采用两台经过TSP-Link连接的2651A高功率数字源表输出脉宽调制波形。通过并行组合两台SMU的输出，可以传送两倍于一台2651A的电流到被测器件。正如一台SMU的测试，将对波形的每个脉冲，在脉冲下降沿前10μs，用快速ADC测量导通电压。使用此函数的参数配置脉宽调制波形特性并设置波形开始和光谱仪测量开始之间的延时。如果参数留空，将使用缺省值。

参数	单位	说明
pulseLevel	安培	测试期间的脉冲电流幅度 最小值：-100 最大值：+100 缺省值：1 注释： 这个设置值用于确定SMU的工作范围，因而会影响在没有误差条件下可以设定的最大占空比。
pulseLimit	伏特	测试期间的脉冲电压极限 最大值：40 缺省值：1 注释： 这个设置值用于确定SMU的工作范围，因而会影响在没有误差条件下可以设定的最大占空比。
frequency	Hz	每秒脉冲数 最小值：0.1 最大值：10,000 缺省值：100 注释： 频率和占空比决定了波形的脉冲宽度。如果测试的工作范围位于直流工作范围之外，没有误差的最小工作频率可能比如上的值高。
dutyCycle	%	脉冲导通时间占脉冲周期的百分比 最小值：0.01 最大值：99 缺省值：1 注释： 此参数可以指定为一个值或数值表。如果指定为一个值，那么波形上全部脉冲的占空比都相等。如果此参数指定为数值表，那么每个脉冲的占空比将由表中对应元素决定。例如，如果表的元素分别为50、25和40，那么波形中第一脉冲的占空比为50%，第二脉冲的占空比为25%，第三脉冲的占空比为40%。如果波形的脉冲个数多于表的元素个数，那么当达到表的最后一个元素时，下一个脉冲将从表的第一个元素开始重用。借用前面的例子，如果输出脉冲个数为5，那么这些脉冲的占空比将分别为50%、25%、40%、50%和25%。 频率和占空比决定了波形的脉冲宽度。测量仪器的脉冲和占空比极限取决于SMU工作的功率包络范围。没有使用误差的占空比最小值和最大值可能比如上值高或低，这取决于测试的工作范围和所选频率。如果指定占空比得到的脉冲宽度/占空比对于工作范围而言太大了，那么SMU会将脉冲宽度/占空比限制在最大容许值。这将在输出波形中表现为脉冲截断或脉冲丢失。 参见2651A说明书，了解最大脉冲宽度和占空比的详细内容。
numpulses	N/A	波形的脉冲数量 最小值：2 最大值：100,000或更多 缺省值：10
specDelay	秒	从PWM输出开始到数字I/O触发器输出启动光谱仪测量之间的时间。 最小值：0 缺省值：0

下面是调用PWM_Test_Dual()函数的例子。

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

这个调用将输出包含100个脉冲的脉宽调制波形，其中脉冲电平为60A、电压限幅10V，频率为1kHz并且占空比为50%。波形输出开始1ms后启动光谱仪测量。测试完成后，SMU输出将被关断并且将测试过程中采集的导通电压测量值以兼容Microsoft Excel拷贝、粘贴的格式打印到仪器控制台。输出的例子如图14所示。

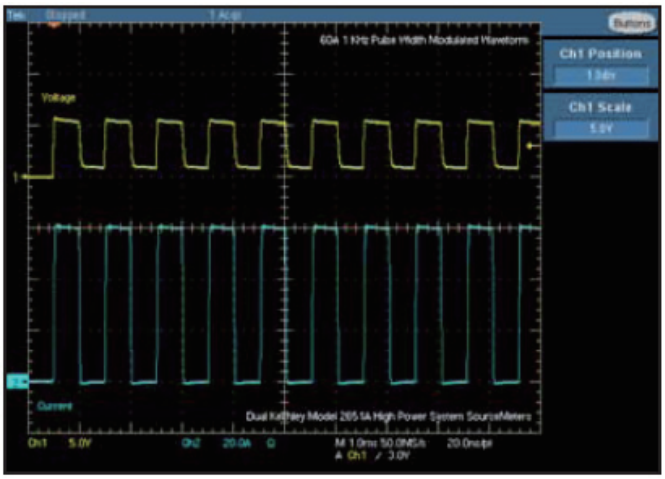

图14：50%占空比的60A 1kHz脉冲波形从两台吉时利2651A输出至高功率LED模块。

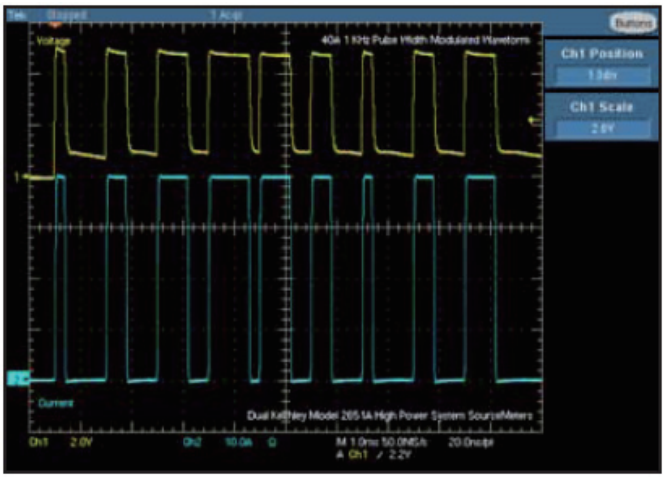

图15：40A 1kHz的脉宽调制波形从吉时利2651A输出至高功率LED模块。

类似一台SMU的函数，这个函数也能实现可变占空比。为了建立每个脉冲宽度都可变的波形，需要向函数传递占空比的表。用下面的函数调用举例说明。

```
dutyTable={20, 40, 60, 80, 60, 40, 20, 40, 60}
PWM_Test_Dual(40, 10, 1000, dutyTable, 9, 1e-3)
```

这个函数调用将输出包含9个脉冲的脉宽调制波形，其中脉冲电平为40A、电压限幅10V，频率为1kHz并且占空比可变。每个脉冲的占空比从表中读取。第一个脉冲的占空比为20%，第二个脉冲的占空比为40%，第三个脉冲的占空比为60%等等。波形输出开始1ms后启动光谱仪测量。完成测试后，SMU输出将被关断并且将测试过程中采集的导通电压测量值以兼容Microsoft Excel拷贝、粘贴的格式打印到仪器控制台。输出的例子如**图15**所示。

结论

目前，HBLED以难以置信的速度推进，制造商们努力让自己的照明光源成为未来的选择。为了实现这个目标，LED制造商必须克服诸多障碍，不断努力降低LED的制造成本，同时提高效率和光输出。实现这些目标的核心是制造商需要准确、可靠和可重复的源和测量设备以及适应不断变化测试需求的能力和灵活性。

2600A系列数字源表能提供满足LED制造商测试需求的特点和灵活性。先进触发模型的创新特点能让2600A系列测量仪器通过复杂驱动方案（例如脉宽调制波形、交流波形，甚至任意波形）准确、可靠地进行可重复测量。这条产品线最新增加的2651A高功率数字源表具有处理最大亮度高功率LED模块的同时保持低电流准确度的强大能力。2600A系列数字源表具有灵活的输出能力和准确的源和测量能力使它们成为HBLED测试的理想选择。

附录A：源代码

注意：此脚本代码只需配合2651A高功率数字源表使用，无需修改。但是，此脚本也能配合2600A系列其的它数字源表使用，只需稍做修改。

```
--[[
    Title:              Pulse Width Modulation Script
    Desription:         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.lineari(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 outputing the waveform and collecting the
    -- measurements, the script will scan the status model for any overruns
    -- that may occur as a result of using impropper 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 ouput a pulse width
    modulated wavform.  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 outputing the waveform and collecting the
    -- measurements, the script will scan the status model for any overruns
    -- that may occur as a result of using impropper 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
-- 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.lineari(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
        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

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

    -- 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

    -- 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

    -- 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

    -- 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

    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

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

    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

A GREATER MEASURE OF CONFIDENCE

KEITHLEY INSTRUMENTS, INC. ■ 28775 AURORA RD. ■ CLEVELAND, OH 44139-1891 ■ 440-248-0400 ■ Fax: 440-248-6168 ■ 1-888-KEITHLEY ■ www.keithley.com

BELGIUM
Sint-Pieters-Leeuw
Ph: 02-3630040
Fax: 02-3630064
info@keithley.nl
www.keithley.nl

CHINA
Beijing
Ph: 86-10-8447-5556
Fax: 86-10-8225-5018
china@keithley.com
www.keithley.com.cn

FRANCE
Saint-Aubin
Ph: 01-64532020
Fax: 01-60117726
info@keithley.fr
www.keithley.fr

GERMANY
Germering
Ph: 089-84930740
Fax: 089-84930734
info@keithley.de
www.keithley.de

INDIA
Bangalore
Ph: 080-26771071 , -72, -73
Fax: 080-26771076
support_india@keithley.com
www.keithley.com

ITALY
Peschiera Borromeo (Mi)
Ph: 02-5538421
Fax: 02-55384228
info@keithley.it
www.keithley.it

JAPAN
Tokyo
Ph: 81-3-5733-7555
Fax: 81-3-5733-7556
info.jp@keithley.com
www.keithley.jp

KOREA
Seoul
Ph: 82-2-574-7778
Fax: 82-2-574-7838
keithley@keithley.co.kr
www.keithley.co.kr

MALAYSIA
Penang
Ph: 60-4-643-9679
Fax: 60-4-643-3794
sea@keithley.com
sea.keithley.com

NETHERLANDS
Son
Ph: 0183-635333
Fax: 0183-630821
info@keithley.nl
www.keithley.nl

SINGAPORE
Singapore
Ph: 65-6747-9077
Fax: 65-6747-2991
sea@keithley.com
www.keithley.com

SWITZERLAND
Z ü rich
Ph: 044-8219444
Fax: 044-8203081
info@keithley.ch
www.keithley.ch

TAIWAN
Hsinchu
Ph: 886-3-572-9077
Fax: 886-3-572-9031
info_tw@keithley.com
www.keithley.com.tw

UNITED KINGDOM
Theale
Ph: 0118-9297500
Fax: 0118-9297519
info@keithley.co.uk
www.keithley.co.uk

Printed in the U.S.A. No. 3129 7.1.11