KEITHLEY

Application Note Series

Number 3129

Testing High Brightness LEDs under Pulse Width Modulation Using the Model 2651A High Power System SourceMeter® Instrument

Introduction

With a worldwide focus on energy conservation, the high power consumption of traditional light sources has come under heavy scrutiny and governments are demanding improvements in the energy efficiency of lighting sources. This demand has lead to heavy investment in and development of alternatives to the incandescent bulb. Compact Fluorescent (CFL) bulbs have become prevalent in the marketplace and, although they are much more efficient and last longer than traditional bulbs, they still fall short of being ideal replacements. High Brightness Light Emitting Diodes (HBLEDs), however, have proven themselves to be a much better option. Like incandescent bulbs, they reach full brightness immediately and do not contain any difficult-to-dispose-of chemicals. They also offer advantages over incandescents: they have incredibly long lifetimes and efficiency levels are continuously being increased.

Unfortunately, the cost of HBLEDs is still too high for most consumers to select them over cheaper technologies, despite the large reduction in energy consumption they offer. In order to reduce the cost of HBLEDs to consumers, manufacturers are constantly working to improve yields and further increase efficiency levels. Meeting this demand requires proper testing and proper testing requires the proper test equipment. Manufacturers demand a lot from their test equipment today to test high brightness LEDs properly. This application note explores some of the electrical test requirements and how the Model 2651A High Power System SourceMeter® instrument can be applied to meet these demands.

The Demand for More Power

HBLEDs are commonly defined as LEDs that operate at 1W of power or higher, with typical operating ranges of from 1W to 3W. Instead of running on 10–30mA of current and being packaged in small 3mm or 5mm plastic domes, HBLEDs run at 300mA–1A or more and are mounted on a small, thermally conductive board designed to draw heat away from the LED's junction. Despite how bright a single HBLED can be, it is typically not bright enough to be used by itself in most lighting applications. Instead, multiple HBLEDs are commonly combined to create a single luminaire, whether for an LED light bulb for a retrofit application or an entire lighting fixture. Even though they are combined in actual use, production testing is typically performed at the individual package level, requiring modest power delivery capability, which is well within the capabilities of most modern instrument-based source-measure units (SMUs), such as the Keithley Series 2400, 2600B and 2651A SourceMeter instruments.

For many applications, combining multiple LEDs into a single luminaire works well when it's desirable to have the light spread in multiple directions and the size of the luminaire provides sufficient space to fit multiple LEDs. However, in other applications where space is limited and/or the light must be directional, this approach is either undesirable or simply will not work. This demand for a lot of light in a small package has lead to the development of high power LED modules, which consist of one or more large-die LEDs. When multiple die are present, they're either wired in parallel or in series, depending on the application and the available power source. The die of these LEDs are much larger than the die of typical HBLEDs and can handle much larger currents as well. It's common for a single die to be required to withstand current levels as high as 10A.

Testing high power HBLED modules properly demands test equipment that can deliver a lot of power to the DUT. Although SMUs, given their ability to source and measure in a single instrument, are the best type of test equipment for testing LEDs, most SMUs on the market simply can't deliver the level of power required. High power HBLED modules often require 100W of power or more, but most instrument-based SMUs are capable of delivering only 40W or less. Keithley's Model 2651A High Power System SourceMeter instrument is capable of delivering up to 200W of continuous DC power and up to 2000W of pulsed power, making it more than powerful enough to test both today's and tomorrow's high power modules (***Figure 1***).

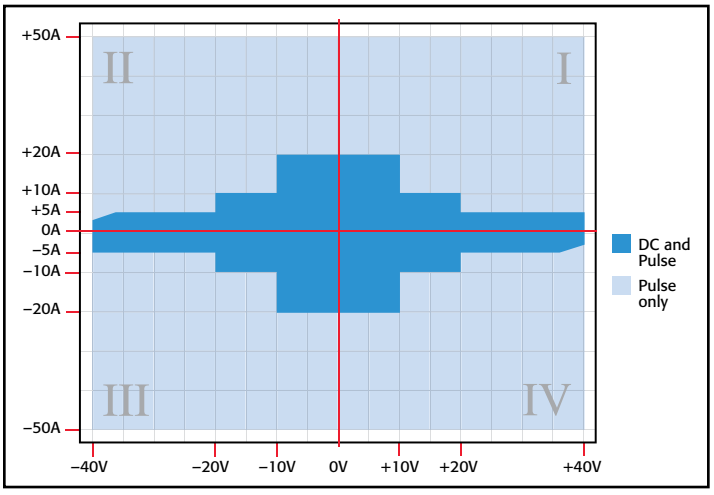

Figure 1: Power envelope for the Model 2651A High Power System SourceMeter Instrument.

Pulse Width Modulation

Pulse width modulation is a common method of controlling the brightness of LEDs. When using this technique, the current through the LED is pulsed at a constant frequency with a constant pulse level, but the width of the pulse is varied (***Figure 2***). Varying the width of the pulse changes the amount of time the LED is in the ON state, as well as the perceived level of brightness. In this drive scheme, the LED is actually flashing, but the frequency of the flashing is so high that it's indistinguishable to the human eye from a constant light level. Although it's possible to control the brightness of an LED simply by lowering the forward drive current, pulse width modulation is the preferable technique for a number of reasons.

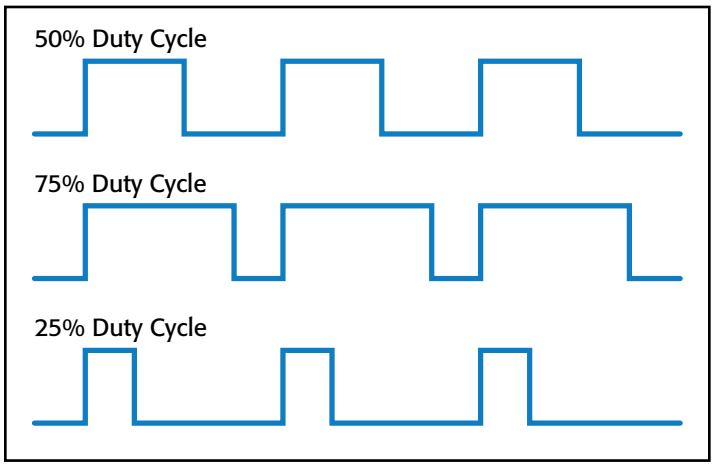

Figure 2: In pulse width modulation, the pulse level and frequency remain constant but the duty cycle is varied.

The first and arguably the most important reason for using pulse width modulation is to maintain consistency of the color of the light as the LED's brightness is reduced. In an LED, the color of the light it emits is related to the forward voltage at which it operates. Although the forward voltage of an LED will remain relatively constant as the forward current is changed, it actually does vary by as much as tens to even hundreds of millivolts. This occurs especially at lower current levels (***Figure 3***). This slight variation in forward voltage equates to a slight variation in light color, which is undesirable for the end user. If heating effects are ignored, in the pulse width modulation technique, the LED is pulsed using exactly the same current level on every pulse, so the forward voltage is the same for every pulse; therefore, the color of the light emitted will not vary.

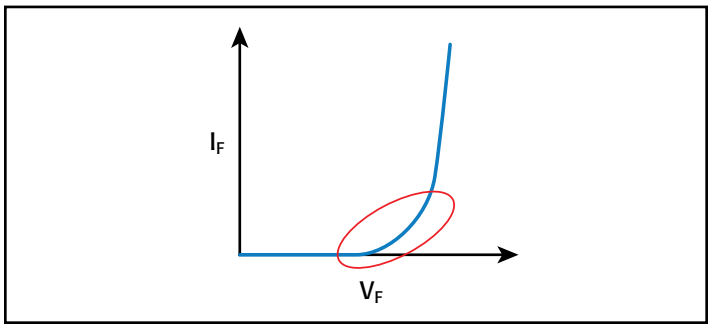

Figure 3: Forward voltage changes significantly when the forward current is low and becomes relatively constant as forward current becomes large.

Another important reason pulse width modulation is preferable is because this technique provides linear control over brightness. The amount of light an LED emits is not linearly related to the amount of current used to drive it. In other words, reducing the drive current by 50% will not cut the light output by 50%; instead, it will drop by some other amount. This would make a dimming scheme based on varying current difficult to apply because it would be necessary to characterize each LED's light output vs. forward current then calibrate the drive scheme to that curve. Using pulse width modulation is a much simpler way to get linear control over brightness. With pulse width modulation, in order to make the LEDs output 50% as much light, all that's necessary is to reduce the duty cycle by 50%. If the LEDs are only ON for half as long, only half as much light will be produced.

Power efficiency is another advantage of the pulse width modulation approach. Because pulse width modulation uses a constant current level for each pulse, it's possible to select a pulse level where the LED operates most efficiently, that is, where the lumen output per watt is the greatest. That means the LED is operating at maximum efficiency no matter what brightness level is used. Another way in which pulse width modulation enhances efficiency is that LEDs will actually output more light for a given drive current when pulsed rather than at DC. Many manufacturers' datasheets include a graph of forward current vs. luminous flux. If the manufacturer has characterized the LED under both pulsed and DC drive currents, one can observe that the pulsed characterization curve lies above the DC characterization curve. This is due to the reduced self-heating that the pulsed drive current produces. Finally, pulse width modulation enhances power efficiency even in the circuitry that drives it. The switching circuitry used in Pulsed Width Modulation wastes very little power. When the switch circuitry is turned off, virtually no current flows and virtually zero power is being used. When the switch is turned on, due to the very low on state resistances, nearly all the power is delivered to the LED and very little is consumed by the switch. In a variable current drive scheme, power to the LED is often reduced by consuming the excess power elsewhere in the circuit.

Pulse Width Modulation with the Model 2651A High Power System SourceMeter Instrument

NOTE: The techniques described here for outputting a pulse width modulated waveform with the Model 2651A are applicable to all members of the Series 2600B System SourceMeter instrument family.

Given that LEDs are often used with pulse width modulation, it's only appropriate that they be tested with pulse width modulation techniques. As part of Pulsed Width Modulation testing, an LED is usually tested by running a series of pulses through it while using a spectrometer to take an integrated measurement of the light output over the course of many pulses. This measurement may take tens or hundreds of milliseconds to complete. During the pulsed output, the forward voltage is measured on every pulse to look for changes as the temperature of the LED rises. ***Figure 4*** illustrates this test.

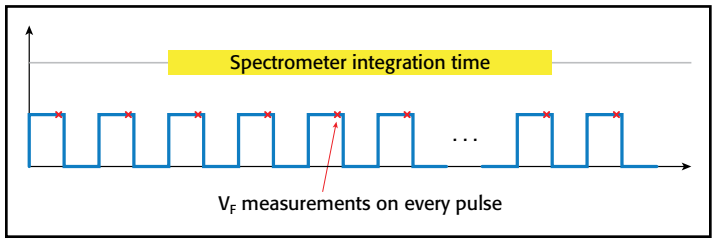

Figure 4: The LED is pulsed with a series of pulses and a V_F measurement is taken at each pulse. A spectrometer concurrently takes optical measurements.

The Model 2651A High Power System SourceMeter instrument is capable of outputting a pulse width modulated waveform with up to 100% duty cycle from 0–20A, 50% duty cycle from 20–30A, and 35% duty cycle from 30–50A. The Model 2651A's advanced trigger model allows for precision pulse widths and duty cycles and tight synchronization with other instruments. These synchronization features can be used to combine two Model 2651As to achieve a Pulsed Width Modulation waveform with pulse current levels twice as high as a single Model 2651A allows with the same duty cycle. This note details how to configure the Model 2651A to output a 30A, 50% duty cycle waveform with a digital I/O output trigger for triggering a spectrometer. It also details how to combine two Model 2651As to increase the current to 60A for this same test.

Required Equipment

Performing this test requires the following equipment:

- PC with GPIB or Ethernet adapter
- GPIB cable or RJ45 LAN Crossover Ethernet cable
- Model 2651A High Power System SourceMeter Instrument
- Model 2651A-KIT-1 Low-Impedance/High-Current Coaxial Cable
- 8-pin signal control cable
- 2-pin terminal block extender for use with the Model 2651A-KIT-1
- 8-pin terminal block extender for use with 8-pin signal control cable
- Digital I/O DB-25 Male Connector Kit Hardware
- 12 AWG or thicker cabling to connect from terminal blocks to device

Add the following equipment to perform this test at up to 100A with a 35% duty cycle or at up to 60A with a 50% duty cycle:

- Model 2651A High Power System SourceMeter Instrument
- Model 2651A-KIT-1 Low-Impedance/High-Current Coaxial Cable
- 8-pin Signal Control Cable
- 2-pin terminal block extender for use with the Model 2651A-KIT-1
- 8-in terminal block extender for use with 8-pin signal control cable
- TSP-Link RJ45 LAN Crossover Cable

Communications

Single SourceMeter Instrument

To perform this test using a single 2651A SourceMeter instrument configuration, connect the instrument to the computer via GPIB or Ethernet as illustrated in ***Figure 5***.

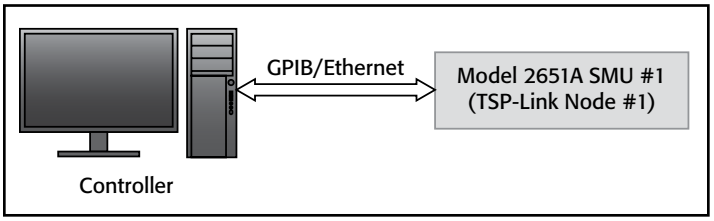

Figure 5: Communications setup for a single SourceMeter instrument.

Dual SourceMeter Instruments

For a dual 2651A SourceMeter instrument configuration, connect the first Model 2651A to the computer via GPIB or Ethernet. Connect the second Model 2651A to the first Model 2651A via the TSP-Link connection. Assign the first Model 2651A to Node #1 and the second Model 2651A to Node #2. These connections are illustrated in ***Figure 6***.

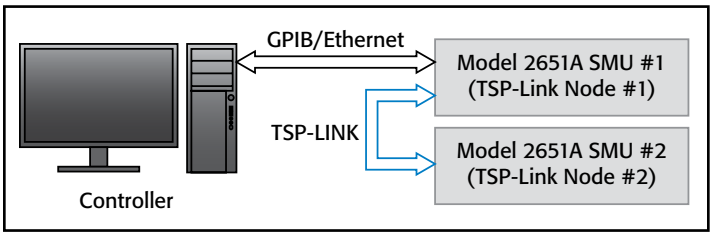

Figure 6: Communications setup for two Model 2651A instruments.

Connecting the Spectrometer to the Digital I/O

In order for the Model 2651A to trigger the spectrometer, the spectrometer's start of test trigger line must be connected to the Model 2651A's digital I/O port. Connect the trigger line from the spectrometer to pin #1 of the 25-pin D-Sub connector on the back panel of the Model 2651A. Ground connections can be found on any of pins 15 through 21. The proper connections are illustrated in ***Figure 7***.

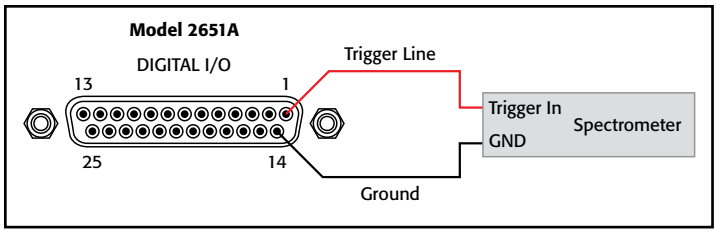

Figure 7: Connections from Model 2651A digital I/O port to spectrometer.

NOTE: In dual SourceMeter instrument configurations, connect only to the digital I/O port of SMU #1. The digital I/O port of SMU #2 is not used.

Device Connections

Figures 8 and ***9*** illustrate the connections from the SourceMeter instruments to the LED device under test. ***Figure 8*** illustrates the connections for a single SourceMeter instrument; ***Figure 9*** illustrates the connections for dual SourceMeter instruments.

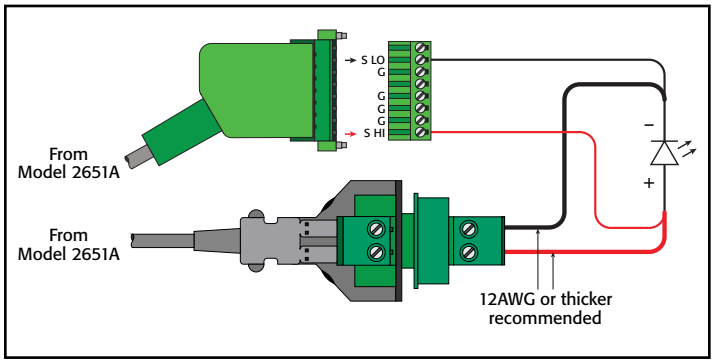

Figure 8: Connections from a single Model 2651A SourceMeter instrument to LED device under test.

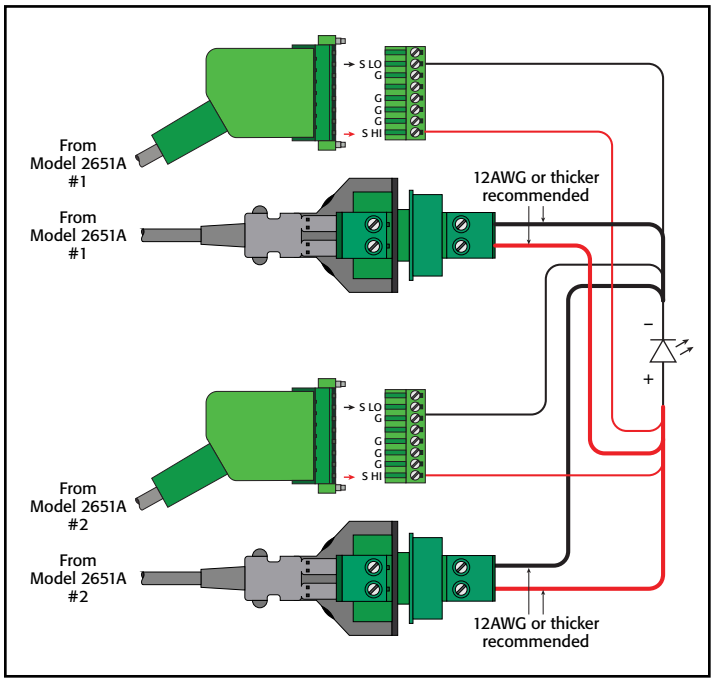

Figure 9: Connections from dual Model 2651A SourceMeter instruments to LED device under test.

When using dual Model 2651A SourceMeter instruments, the two SMUs are connected in parallel to create a single source with twice as much current capacity. Read and understand the section titled "Combining SMU outputs" in the *Model 2651A High Power System SourceMeter Instrument Reference Manual* before performing this test.

NOTE: Wiring from the 2-pin terminal block extender that is connected to the end of the low-impedance/high-current coaxial cable provided with the Model 2651A to the device under test should be made with 12 AWG wire or heavier in order to support the high current levels. Also, the length of these wires should be kept to a minimum to minimize inductance.

Configuring the Trigger Model

The advanced trigger model must be used to source current levels above those available in the DC operating regions of the Model 2651A. This same trigger model gives the Model 2651A its precise pulse widths and tight synchronization and makes accurate pulse width modulation possible. The following sections illustrate how to configure the trigger model to output a pulse width modulated waveform and trigger a spectrometer with the Model 2651A.

Configuring the Trigger Model for a Single Model 2651A SourceMeter Instrument

The trigger model shown in ***Figure 10*** will perform a pulse width modulation test on an LED using a single Model 2651A. In this configuration, Timer 1 controls the pulse period, Timer 2 controls the pulse width and Timer 3 inserts a delay between the start of the pulse and the start of the measurement. Timer 4 creates a delay between the start of the waveform output and the start of the spectrometer measurement. When Timer 4 expires, it triggers Digital I/O Trigger 1, which sends the start trigger to the spectrometer.

Configuring the Trigger Model for Dual Model 2651A SourceMeter Instruments

The trigger model shown in ***Figure 11*** will perform a pulse width modulation test on an LED using two Model 2651A SourceMeter instruments with their outputs connected together in parallel. Timer 1 of Model 2651A #1 controls the pulse period of both SMUs. It does this by relaying its output trigger signal on TSP-Link® Trigger 1, which is then sent to Model 2651A #2. Due to the extremely low latency of TSP-Link triggering, Model 2651A #2 is kept in sync with Model 2651A #1 to within 500ns. Because this signal is sent once every time Timer 1 expires, long-term synchronization between the SMUs is ensured because the two SMUs are synchronized at the start of every pulse cycle.

As in the single SMU configuration, Timer 2 controls the pulse width and Timer 3 inserts a delay between the start of the pulse and the start of the measurement. Each SMU uses its own

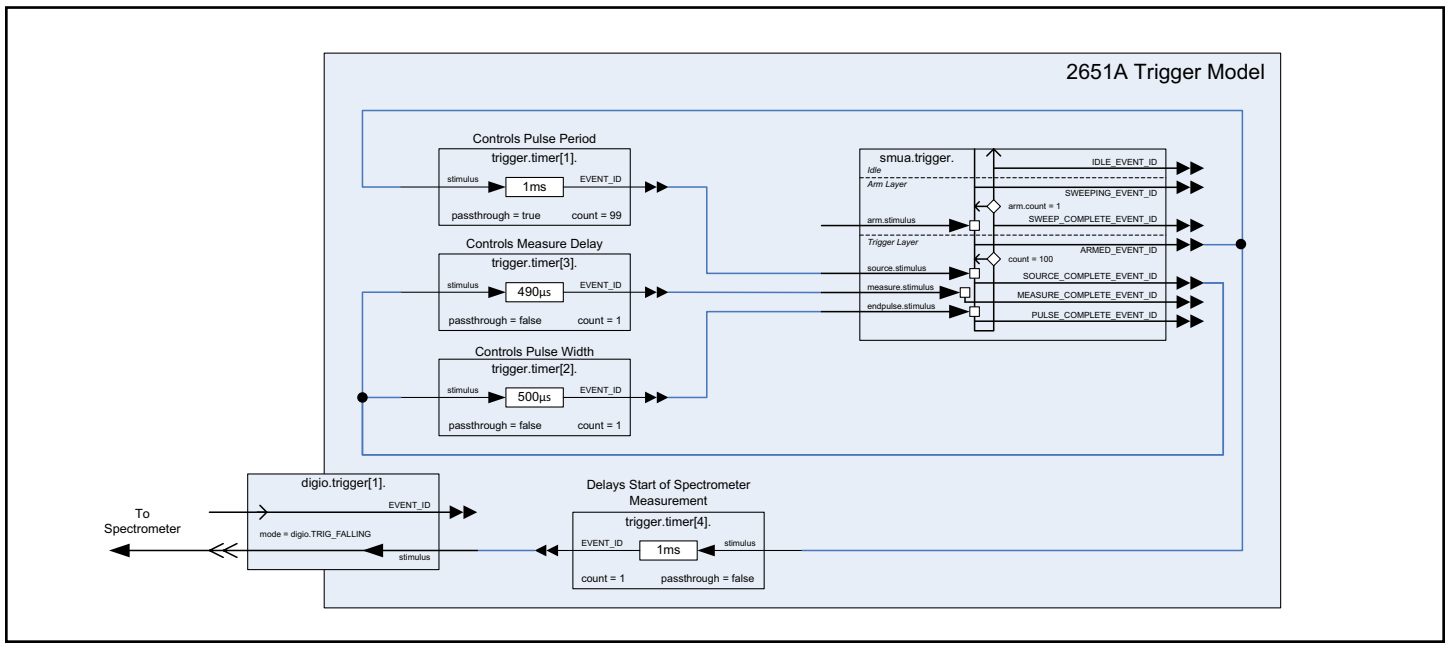

Figure 10: Trigger model for pulsed width modulation test on LED with a single Model 2651A System SourceMeter instrument.

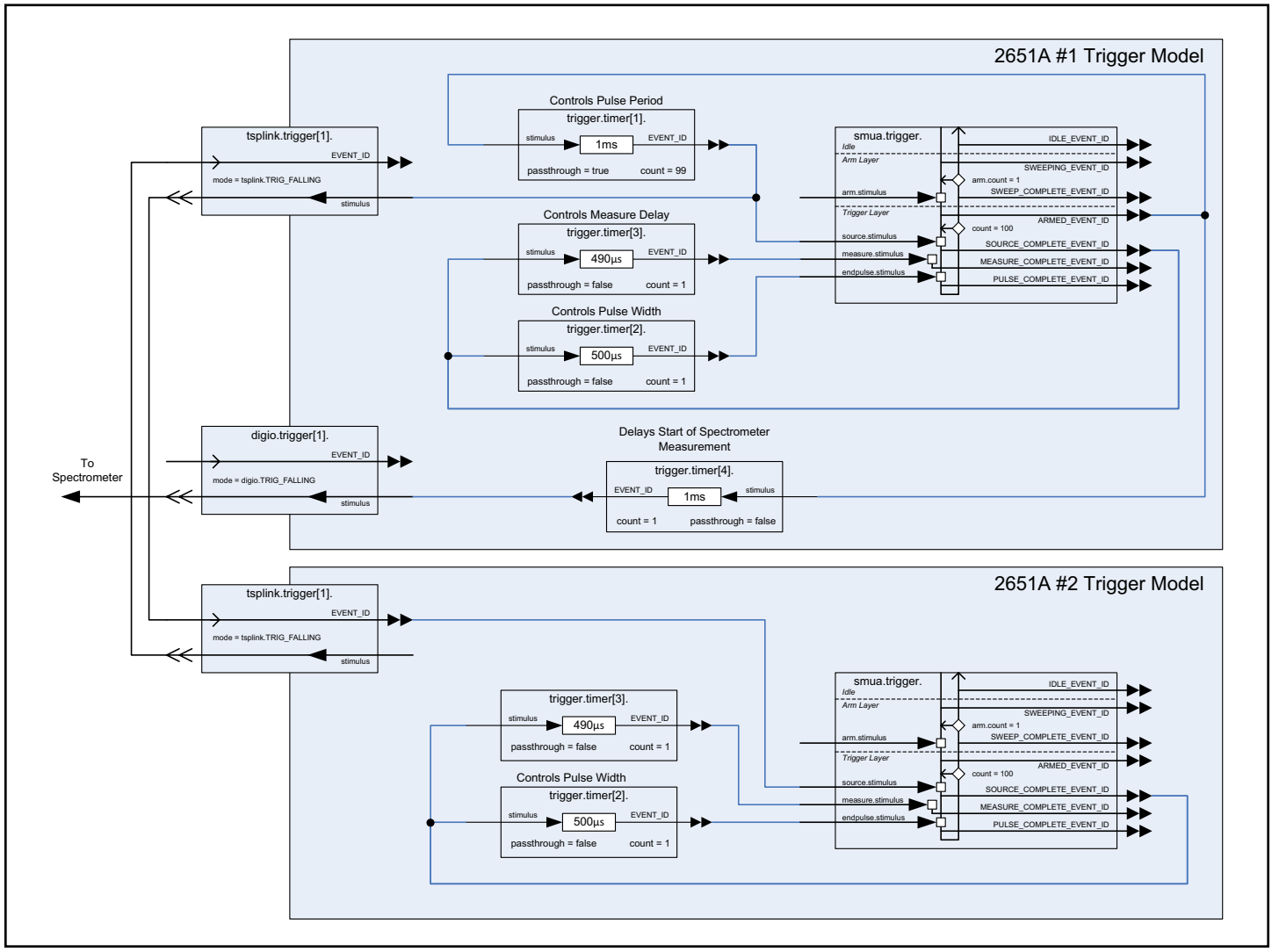

Figure 11: Trigger model for pulsed width modulation test on LED with dual Model 2651A System SourceMeter instruments.

Timer 2 and Timer 3 to control its pulse width and measure delay. Given the accuracy of the timers and the synchronization at the start of every pulse, it's certain that both SMUs will output a pulse with the same pulse width and take measurements at the same time. Finally, once again Timer 4 on Model 2651A #1 creates a delay between the start of the pulse waveform and the start of the spectrometer measurement by delaying the output of Digital I/O Trigger 1.

Configuring the Frequency and Duty Cycle

In both the single and dual SMU configurations, configuring the waveform for a particular frequency and duty cycle requires setting the appropriate pulse period and pulse width in the trigger model. Frequency (f) is related to pulse period (P) by the equation $P = 1 / f$; therefore, it's possible to control the frequency of the waveform by setting the appropriate pulse period. For a 1kHz waveform, P = 1/1kHz = 1ms. Duty cycle (D.C.) is the ratio of pulse width (PW) over pulse period (P) or D.C. = PW / P; therefore, the duty cycle can be set by setting the appropriate value for pulse width; this value can be calculated using the equation D.C. * P = PW.

Modulating the Pulse Width

The trigger model diagrams in ***Figures 10*** and ***11*** show the pulse width being controlled by Timer 2, which has a fixed timeout value. This is represented in code by calling the ICL command

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

where **`pulseWidth`** is a fixed delay value in seconds. Having a fixed timeout value means that the pulse width for every cycle in a single waveform output will be the same for the entire length of the waveform. Modulating the pulse width during a single waveform output requires that the timeout value of Timer 2 be variable. To facilitate this, the timers of the advanced trigger model can be assigned a delay list rather than a single value. This can be done by calling the ICL command

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

where **`pulseWidthTable`** is a table containing multiple delay values in seconds. With a delay list assigned to Timer 2, each pulse in the waveform can be assigned a different pulse width; therefore, the waveform can be pulse width modulated.

NOTE: The example script in this application note allows for both fixed and variable pulse widths. See the function documentation for details.

Example Program Code

NOTE: The Test Script Processor (TSP®) Script in this application note is for demonstration purposes only and is not optimized for fastest production throughput. Please contact a Keithley Applications Engineer for system throughput optimization considerations.

NOTE: The TSP Script in this application note is designed to be run from Test Script Builder. It can be run from other programming environments such as Microsoft® Visual Studio or National Instruments LabVIEW®; however, modifications may be required.

The TSP script provided in this application note contains all the code necessary to perform a pulse width modulation test with optical measurement on a high brightness LED using one or two Model 2651A High Power System SourceMeter instruments. The code for this script can be found in Appendix A: Source Code.

The script performs the following functions:

- Initializes the TSP-Link connection (only when using two units)
- Configures the SMU(s) ranges and measurement settings
- Configures the trigger model(s)
- Prepares the readings buffers for data
- Outputs the PWM waveform
- Returns the collected data to the instrument console in a format that can be copied and pasted directly into a Microsoft Excel® spreadsheet.

The script is written using TSP functions rather than a single block of inline code. TSP functions are similar to functions in other programming languages such as C or Visual Basic and must be called before the code contained in them is executed. Because of this, running the script alone will not execute the test. To execute the test, first run the script to load the functions into Test Script memory and then call the functions. Refer to the documentation for Test Script Builder for directions on how to run scripts and enter commands using the instrument console.

Within the script, there are several comments describing what is being performed by the lines of code, as well as documentation for the functions contained in the script. Lines starting with

```
node[2].
```

are commands that are being sent to Model 2651A #2 through the TSP-Link interface. All other commands are executed on Model 2651A #1.

Example Program Usage

This script contains two functions for outputting a pulse width modulated waveform: one for use with a single Model 2651A High Power System SourceMeter instrument and one for use with two Model 2651A High Power System SourceMeter instruments. These functions contain parameters whose values are used to configure the waveform, allowing the user to set properties of the waveform such as the frequency and duty cycle without needing to rewrite any code. The following sections provide explanations of each function and its parameters.

PWM_Test_Single()

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

This function will output a pulse width modulated waveform using a single Model 2651A High Power System SourceMeter instrument. A forward voltage measurement will be taken with the Fast ADC on every pulse in the waveform and measurements will be placed 10μs before the falling edge of the pulse. Use the parameters of this function to configure the properties of the Pulsed Width Modulation waveform and to set the delay between the start of the waveform and the start of the spectrometer measurement. If any parameters are left blank, a default value will be used.

Parameter	Units	Description
pulseLevel	Amps	The current level to pulse to during the test Min: –50 Max: +50 Default: 1 ***Comments:*** The value set here goes into determining what operating region the SMU is in and will therefore have an effect on the maximum duty cycle value that can be set without error.
pulseLimit	Volts	Voltage limit of the pulses during the test Max: 40 Default: 1 ***Comments:*** The value set here goes into determining what operating region the SMU is in and will therefore have an effect on the maximum duty cycle value that can be set without error.
frequency	Hz	The number of pulses per second Min: 0.1 Max: 10,000 Default: 100 ***Comments:*** Frequency and Duty Cycle determine the pulse width of the waveform. The minimum frequency that can be used without error may be higher than the value shown if the operating region for the test is outside of the DC operating area.
dutyCycle	%	The on time of the pulse as a percentage of the pulse period Min: 0.01 Max: 99 Default: 1 ***Comments:*** This parameter may be assigned a single value or a table of values. If it is assigned a single value, then all pulses in the waveform will have the same duty cycle. If this parameter is assigned a table of values, then the duty cycle for each pulse will be determined by a corresponding entry in the table. For example, if the table has the values 50, 25, and 40, then the first pulse in the waveform will have a duty cycle of 50%, the second pulse will have a duty cycle of 25%, and the third pulse a duty cycle of 40%. If there are more pulses in the waveform than there are values in the table, then when the last value in the table is reached, the values in the table will be reused from the beginning of the table starting with the next pulse. Using the previous example, if the number of pulses to output is 5, then the duty cycles of the pulses will be 50%, 25%, 40%, 50%, and 25%. Frequency and Duty Cycle determine the pulse width of the waveform. Pulse and duty cycle of the instrument are limited depending on the region of the power envelope the SMU is operating in. The minimum and maximum duty cycle that can be used without error may be higher or lower than the values shown depending on the operating region and selected frequency for the test. If the duty cycle specified results in a pulse width/duty cycle that is too large for the operating region, the SMU will limit the pulse width/duty cycle itself to its maximum allowable value. This will appear in the output waveform in the form of pulses that are cut short or are missing. See the Model 2651A Specification for details on maximum pulse widths and duty cycles.
numpulses	N/A	The number of pulses in the waveform Min: 2 Max: 100,000 or greater Default: 10
specDelay	Seconds	The time between the start of the PWM output and the output of the Digital I/O trigger to start the spectrometer measurement. Min: 0 Default: 0

The following is an example call to function **`PWM_Test_Single()`**.

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

This call will output a pulse width modulated waveform of 100 pulses with a pulse level of 30A, a 10V voltage limit, a frequency of 1kHz, and a duty cycle of 50%. Spectrometer measurements will begin 1ms after the start of the waveform output. At the completion of the test, the SMU output will be turned off and the forward voltage measurements collected during the test will be printed to the instrument console in a format compatible with copying and pasting into a Microsoft Excel spreadsheet. An example of this output can be seen in ***Figure 12***.

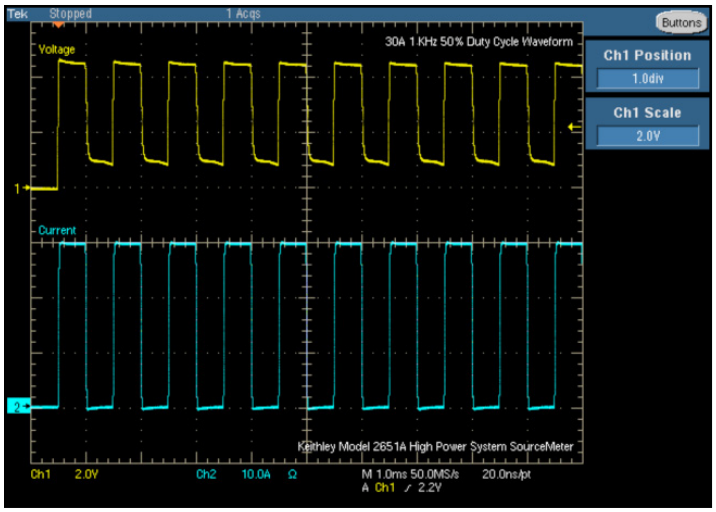

Figure 12: 30A 1kHz pulsed waveform with 50% duty cycle from a Keithley Model 2651A into a high power LED module.

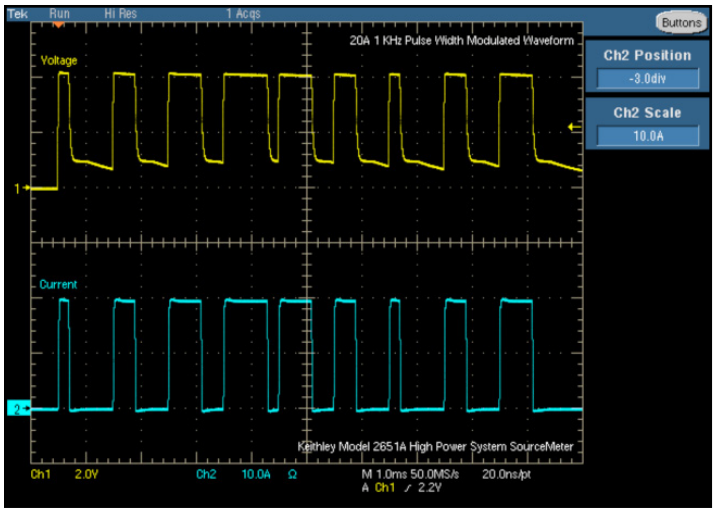

Figure 13: 20A 1kHz pulse width modulated waveform from a Keithley Model 2651A into a high power LED module.

In the previous example, the duty cycle of the waveform was fixed at 50%. To create a waveform where the pulse width varies for each pulse, it's necessary to pass the function a table of duty cycles. The following example calls demonstrate this.

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

This call will output a pulse width modulated waveform of 9 pulses with a pulse level of 20A, a 10V voltage limit, a frequency of 1kHz and a varying duty cycle. The duty cycle for each pulse will be read from the table. The first pulse will have a duty cycle of 20%, the second pulse a duty cycle of 40%, the third pulse a duty cycle of 60% and so on. Spectrometer measurements will begin 1ms after the start of the waveform output. At the completion of the test, the SMU output will be turned off and the forward voltage measurements collected during the test will be printed to the instrument console in a format compatible for copy and pasting into Microsoft Excel. An example of this output can be seen in ***Figure 13***.

PWM_Test_Dual()

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

This function uses two Model 2651A High Power System SourceMeter instruments to output a pulse width modulated waveform with as much as twice as much current as a single Model 2651A is capable of.

This function will output a pulse width modulated waveform using two Model 2651A High Power System SourceMeter instruments connected together via TSP-Link. By combining the SMU's outputs in parallel, twice the current of a single Model 2651A can be delivered to the device under test. Just like the single SMU test, a forward voltage measurement will be taken on every pulse in the waveform and measurements will be taken with the Fast ADC, placed 10μs before the falling edge of the pulse. Use the parameters of this function to configure the properties of the Pulsed Width Modulation waveform and to set the delay between the start of the waveform and the start of the spectrometer measurement. If any parameters are left blank, a default value will be used.

Parameter	Units	Description
pulseLevel	Amps	The current level to pulse to during the test Min: –100 Max: +100 Default: 1 ***Comments:*** The value set here goes into determining what operating region the SMU is in and will therefore have an effect on the maximum duty cycle value that can be set without error.
pulseLimit	Volts	Voltage limit of the pulses during the test Max: 40 Default: 1 ***Comments:*** The value set here goes into determining what operating region the SMU is in and will therefore have an effect on the maximum duty cycle value that can be set without error.
frequency	Hz	The number of pulses per second Min: 0.1 Max: 10,000 Default: 100 ***Comments:*** Frequency and Duty Cycle determine the pulse width of the waveform. The minimum frequency that can be used without error may be higher than the value shown if the operating region for the test is outside of the DC operating area.
dutyCycle	%	The on time of the pulse as a percentage of the pulse period Min: 0.01 Max: 99 Default: 1 ***Comments:*** This parameter may be assigned a single value or a table of values. If it is assigned a single value then all pulses in the waveform will have the same duty cycle. If this parameter is assigned a table of values then the duty cycle for each pulse will be determined by a corresponding entry in the table. For example, if the table has the values 50, 25, 40 then the first pulse in the waveform will have a duty cycle of 50%, the second pulse will have a duty cycle of 25% and the third pulse a duty cycle of 40%. If there are more pulses in the waveform than there are values in the table, then when the last value in the table is reached, the values in the table will be reused from the beginning of the table starting with the next pulse. Using the previous example, if the number of pulses to output is 5 then the duty cycles of the pulses will be 50%, 25%, 40%, 50%, 25%. Frequency and Duty Cycle determine the pulse width of the waveform. Pulse and duty cycle of the instrument are limited depending on the region of the power envelope the SMU is operating in. The minimum and maximum duty cycle that can be used without error may be higher or lower than the values shown depending on the operating region and selected frequency for the test. If the duty cycle specified results in a pulse width/duty cycle that is too large for the operating region, the SMU will limit the pulse width/duty cycle itself to its maximum allowable value. This will appear in the output waveform in the form of pulses that are cut short or are missing. See the Model 2651A Specification for details on maximum pulse widths and duty cycles.
numpulses	N/A	The number of pulses in the waveform Min: 2 Max: 100,000 or greater Default: 10
specDelay	Seconds	The time between the start of the PWM output and the output of the Digital I/O trigger to start the spectrometer measurement. Min: 0 Default: 0

The following is an example call to function **PWM_Test_Dual()**.

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

This call will output a pulse width modulated waveform of 100 pulses with a pulse level of 60A, a 10V voltage limit, a frequency of 1kHz, and a duty cycle of 50%. Spectrometer measurements will begin 1ms after the start of the waveform output. At the completion of the test, the SMU output will be turned off, and the forward voltage measurements collected during the test will be printed to the instrument console in a format compatible with copy and pasting into Microsoft Excel. An example of this output can be seen in ***Figure 14***.

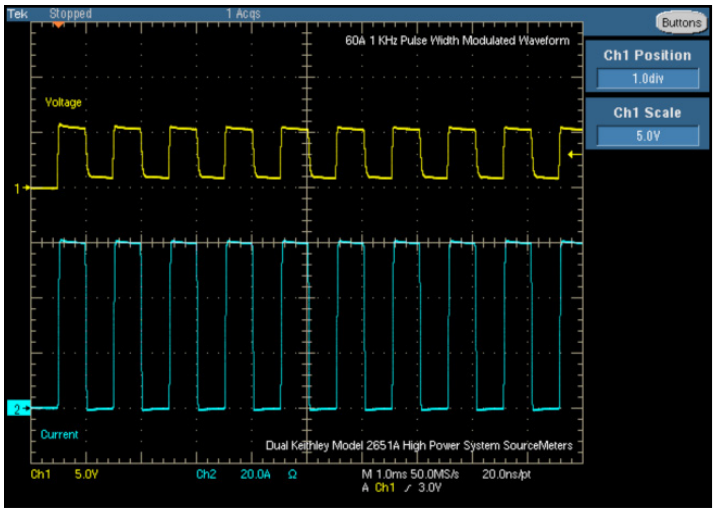

Figure 14: 60A 1kHz pulsed waveform with 50% duty cycle from dual Keithley Model 2651As into a high power LED module.

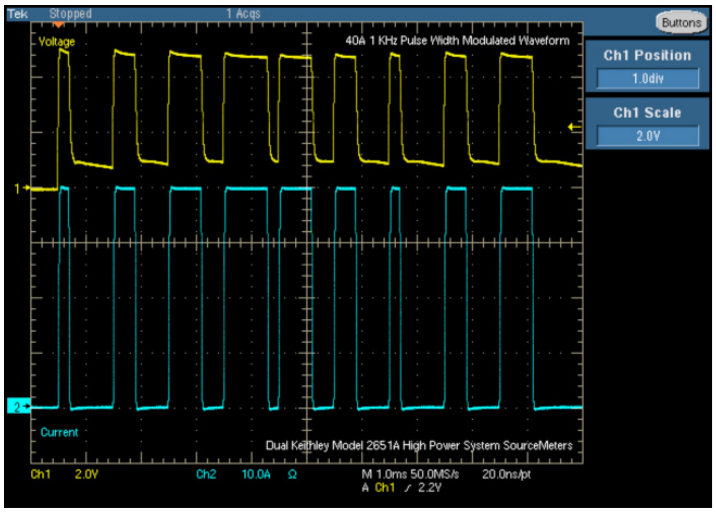

Figure 15: 40A 1kHz pulse width modulated waveform from dual Keithley Model 2651As into a high power LED module.

Just like the single SMU function, this function allows for a variable duty cycle as well. To create a waveform where the pulse width varies for each pulse it's necessary to pass the function a table of duty cycles. The following example calls demonstrate this.

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

This call will output a pulse width modulated waveform of 9 pulses with a pulse level of 40A, a 10V voltage limit, a frequency of 1kHz, and a varying duty cycle. The duty cycle for each pulse will be read from the table. The first pulse will have a duty cycle of 20%, the second pulse a duty cycle of 40%, the third pulse a duty cycle of 60% and so on. Spectrometer measurements will begin 1ms after the start of the waveform output. At the completion of the test, the SMU output will be turned off and the forward voltage measurements collected during the test will be printed to the instrument console in a format compatible for copy and pasting in to Microsoft Excel. An example of this output can be seen in ***Figure 15***.

Conclusion

HBLEDs are advancing at an incredible pace and manufacturers are working hard to make them the lighting source choice of the future. In order to get there, LED manufacturers must jump several hurdles, ever trying to reduce the cost of manufacturing LEDs while simultaneously trying to increase their efficiency and light output. At the center of meeting these objectives, manufacturers require accurate, reliable, and repeatable source and measurement equipment with the power and flexibility to adapt to their ever-changing testing needs.

Series 2600B System SourceMeter instruments offer the features and flexibility to keep up with LED manufacturers' testing needs. Innovative features like the advanced trigger model allow Series 2600B instruments to make repeatable measurements accurately and reliably using complex drive schemes such as pulse width modulated waveforms, AC waveforms, and even arbitrary waveforms. The newest addition to this product line, the Model 2651A High Power System SourceMeter Instrument, provides additional power to handle even the brightest high power LED modules while still maintaining low current accuracy. Series 2600B System SourceMeter instruments, with their flexible output capabilities and their ability to source and measure accurately in a single instrument, make them the perfect choice for testing HBLEDs.

Appendix A: Source Code

NOTE: The code in this script will work without modification only with the Model 2651A High Power System SourceMeter instrument. However, this script can also work with other Series 2600B System SourceMeter instruments with only minor modifications.

```
--[[
   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 Tektronix Company

A Greater Measure of Confidence

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

BENELUX
+31-40-267-5506
www.keithley.nl

BRAZIL
55-11-4058-0229
www.keithley.com

CHINA
86-10-8447-5556
www.keithley.com.cn

FRANCE
+33-01-69-86-83-60
www.keithley.fr

GERMANY
+49-89-84-93-07-40
www.keithley.de

INDIA
080-30792600
www.keithley.in

ITALY
+39-049-762-3950
www.keithley.it

JAPAN
81-120-441-046
www.keithley.jp

KOREA
82-2-6917-5000
www.keithley.co.kr

MALAYSIA
60-4-643-9679
www.keithley.com

MEXICO
52-55-5424-7907
www.keithley.com

RUSSIA
+7-495-664-7564
www.keithley.ru

SINGAPORE
01-800-8255-2835
www.keithley.com.sg

TAIWAN
886-3-572-9077
www.keithley.com.tw

UNITED KINGDOM
+44-1344-39-2450
www.keithley.co.uk

For further information on how to purchase or to locate a sales partner please visit **www.keithley.com/buy**

 Printed in the U.S.A No. 3129 4.9.14