KEITHLEY

Application Note Series

Number 3116

Measuring Pulsed Waveforms with the High Speed Analog-to-Digital Converter in the Model 2651A High Power System SourceMeter® Instrument

Green initiatives and energy efficiency standards worldwide have motivated engineers to find ways to design more efficient semiconductor devices and integrated circuits. In industrial applications, engineers are trying to improve the efficiency of switching power supplies and power inverters. In commercial and residential applications, the push for LEDs (light emitting diodes) drives the design of AC-DC converters to make these DC devices operate on AC power and use pulse width modulation as a light dimming technique.

For these inherently pulsed applications, it is important to test the discrete components that make up the end product under pulsed conditions. Test instruments with only DC capability can deliver an amount of power to a device that causes enough heat dissipation to alter its characteristics. The desire to measure the true state of the device without the effects of self-heating is another motivation for pulsed characterization.

The use of a pulsed stimulus demands faster measurements. The Model 2651A meets this need with its high speed ADCs (analog-to-digital converters). Coupled with the ability to measure asynchronously from the source, this feature makes the Model 2651A suitable for many transient characterization applications. The following demonstrates how to configure the ADCs to perform measurements on pulsed waveforms and considers techniques to obtain optimal results.

High Speed ADC vs. Integrating ADC

Traditional precision SMUs (source-measure units) use integrating ADCs. An integrating ADC averages the signal over a certain time interval known as the integration time. ***Figure 1*** depicts a simplified dual-slope integrating ADC. This type of ADC operates by charging a capacitor with the unknown signal and then discharging the capacitor using a reference voltage. The ratio of the charge and discharge times is proportional to the ratio of the unknown signal to the reference signal.

While having the advantage of high accuracy and excellent noise immunity, this ADC technology does not lend itself to high speed measurements. The charge-discharge cycles on the capacitor result in long inter-measurement intervals. For example, though the smallest integration interval for the Model 2651A is 0.001PLC (16.67µs for 60Hz, 20µs for 50Hz), the smallest inter-measurement interval is 50µs.

In addition to the two integrating ADCs for voltage and current, the Model 2651A also includes two high speed ADCs with the capability of sampling signals at burst rates of up to

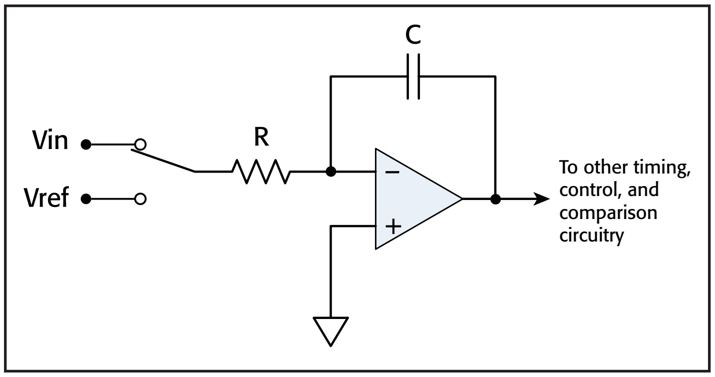

Figure 1. Simplified diagram of dual-slope integrating ADC.

1MHz.[1] These ADCs use sampling technology similar to an oscilloscope and take snapshots of the signal over time. A high speed ADC of the Model 2651A has higher resolution (18 bits) than an oscilloscope (typically 8 bits) resulting in more precise transient characterization in comparable bandwidths.

Figure 2 illustrates the difference between the integrating and high speed ADCs. While returning more readings, the measurements performed by the high speed ADC are less accurate and less repeatable than those performed by the integrating ADC. For applications that demand higher throughput, the lower accuracy can be tolerated, or if needed, improved by averaging several readings. Typically, integrating

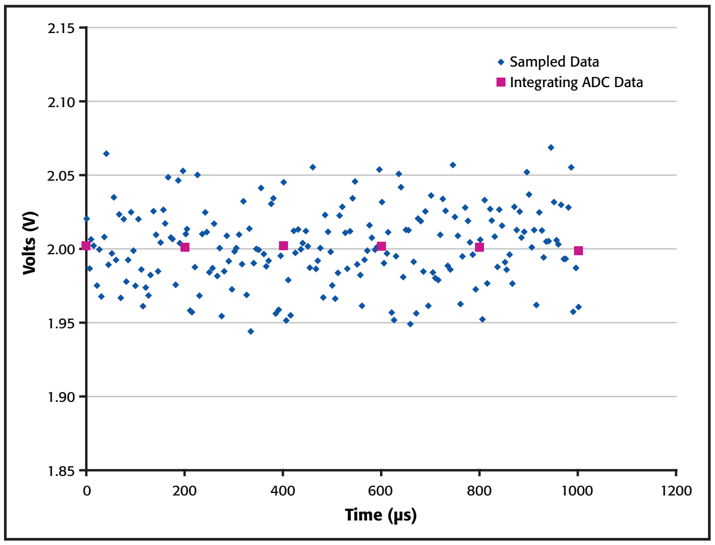

Figure 2. Comparison of possible results from integrating and sampling ADC technologies.

1 Up to 5,000 readings can be acquired at the maximum acquisition rate.

ADC measurements with integration rates of 0.01PLC or faster can be made with similar accuracy using the high speed ADC.

Having two high speed ADCs ensures that voltage and current measurement can be made simultaneously. The ability to sample current is a unique feature of the Model 2651A and may replace the need for a current probe and an oscilloscope in some applications.

The combination of the Model 2651A's high speed ADCs and trigger model supports precisely timed measurements on pulsed signals. Additionally, the Model 2651A introduces a feature that allows the user to trigger measurements asynchronously from source operations, such as before, during, or after a pulse. This capability can also be used with the integrating ADCs. ***Figure 3*** diagrams five examples of pulsed signals and measurements that can be made with the Model 2651A. The sections below discuss how to configure the Model 2651A to perform each of these examples.

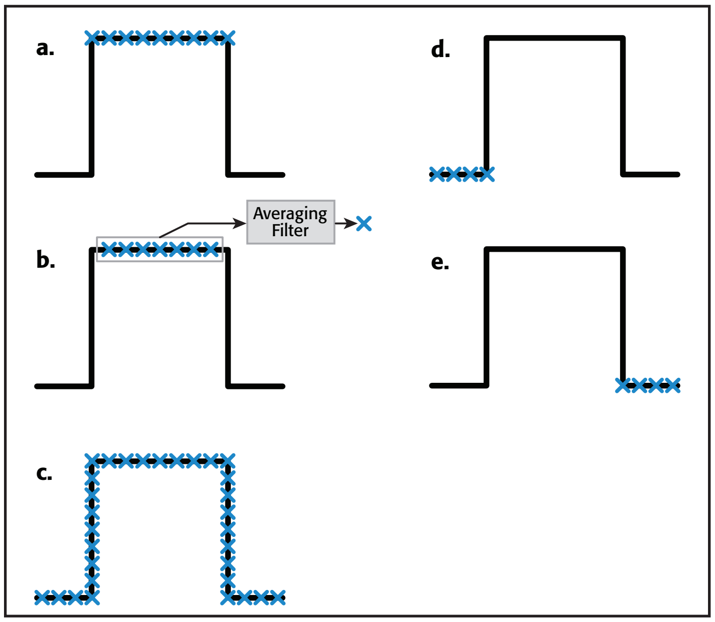

Figure 3a. Example #1, measuring at the top of the pulse.

Figure 3b. Example #2, performing a spot mean measurement at the top of the pulse.

Figure 3c. Example #3, digitizing the entire pulse.

Figure 3d. Example #4, triggering measurements to begin before the pulse.

Figure 3e. Example #5, triggering measurement to begin after the pulse.

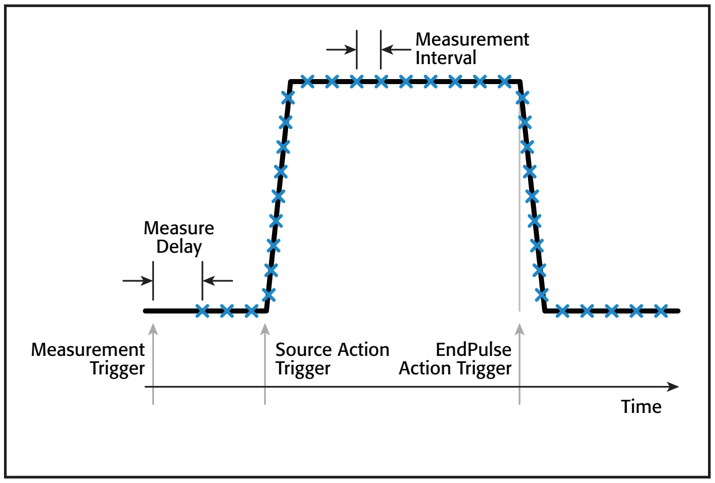

Figure 4. Specifying a pulse using the Model 2651A.

Figure 4 illustrates the relationship between a pulse, the triggering conditions that create the pulse, and the definition of high speed ADC measurement parameters.

General Information Regarding the Examples

In the sections that follow, each of the examples makes use of a few common functions. The code for these functions is in ***Appendix A***.

Example #1: Digitizing the Top of a Pulse

Potential Uses

For some applications, such as thermal impedance of power diodes and LEDS, characterizing the slope of the measured voltage at the top of the pulse is important. This capability is also useful for characterizing pulse amplitude flatness. The high speed ADCs can digitize the top of the pulse when the measurements are made synchronously with the source.

How the Measurement is Made

The pulses are timed using trigger timers. Measurements are triggered at the beginning of the pulse, but delayed to the settled part of the pulse by programming a measure delay.

The trigger model setup is depicted in ***Figure 5***. The Test Script Processor (TSP®) script for Example #1 is located in ***Appendix B*** along with the commands for executing the test and obtaining the results. The sample results taken using a 0.1Ω resistor are shown in ***Figure 6***.

Example #2: Performing a Spot Mean Measurement at the Top of the Pulse

Potential Uses

Often, analysis software is used to average sampled data to improve accuracy. The Model 2651A can automatically perform averages on measurements.

How the Measurement is Made

The averaging and median filters of the Model 2651A can be used on the high speed ADC readings, making it possible to return spot mean measurements. The same test performed in Example #1 can be modified to return a spot mean measurement instead of the raw sample data by changing a few lines of code.

The trigger model configuration is the same as used for Example #1. ***Appendix C*** contains the TSP script for Example #2 and the commands for executing the test and obtaining the

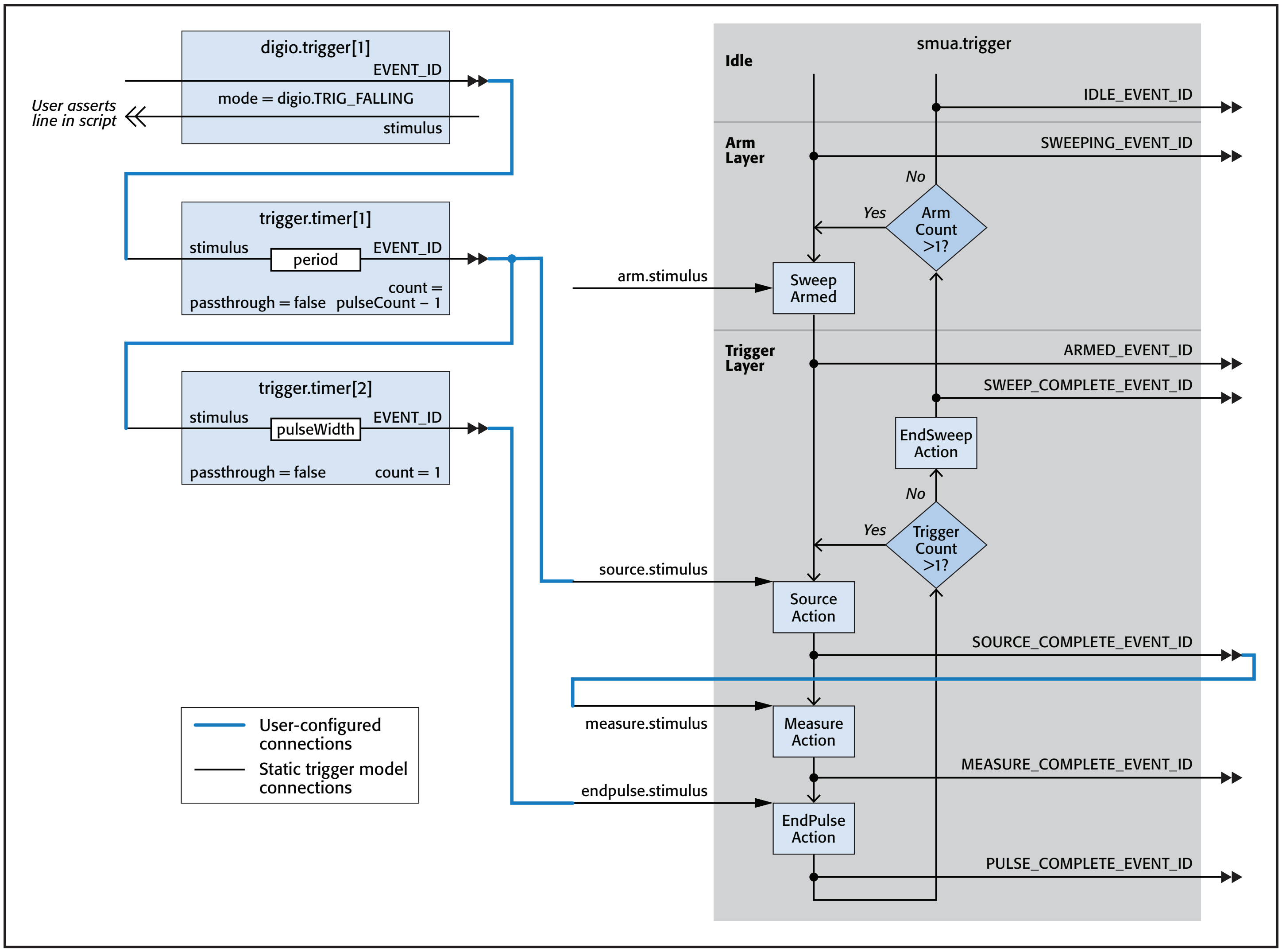

Figure 5. Trigger model configuration for Examples #1 and #2.

results. The test results are shown in ***Figure 6*** next to the raw sample data and are also listed in ***Table 1***.

Table 1. Spot mean measurement results for Example #2.

Current (A)	Voltage (V)
19.7816	1.97854
19.8421	1.98492

Example #3: Digitizing the Entire Pulse Including the Rising and Falling Edges

Potential Uses

At times, it is useful to characterize how a pulse is transmitted through a device or system. These applications require that the entire pulse be digitized, including the rising and falling edges. This measurement is possible using the high speed ADCs to measure asynchronously to the source operation.

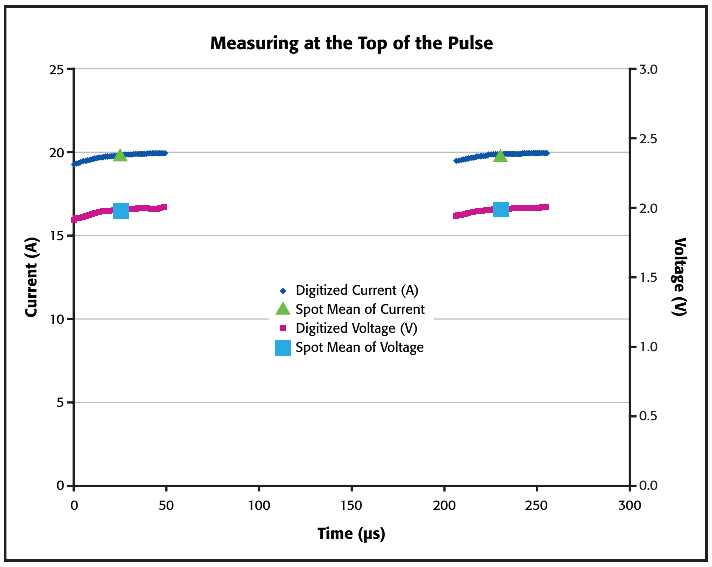

Figure 6. Results of Examples #1 and #2. The load is a 0.1Ω resistor.

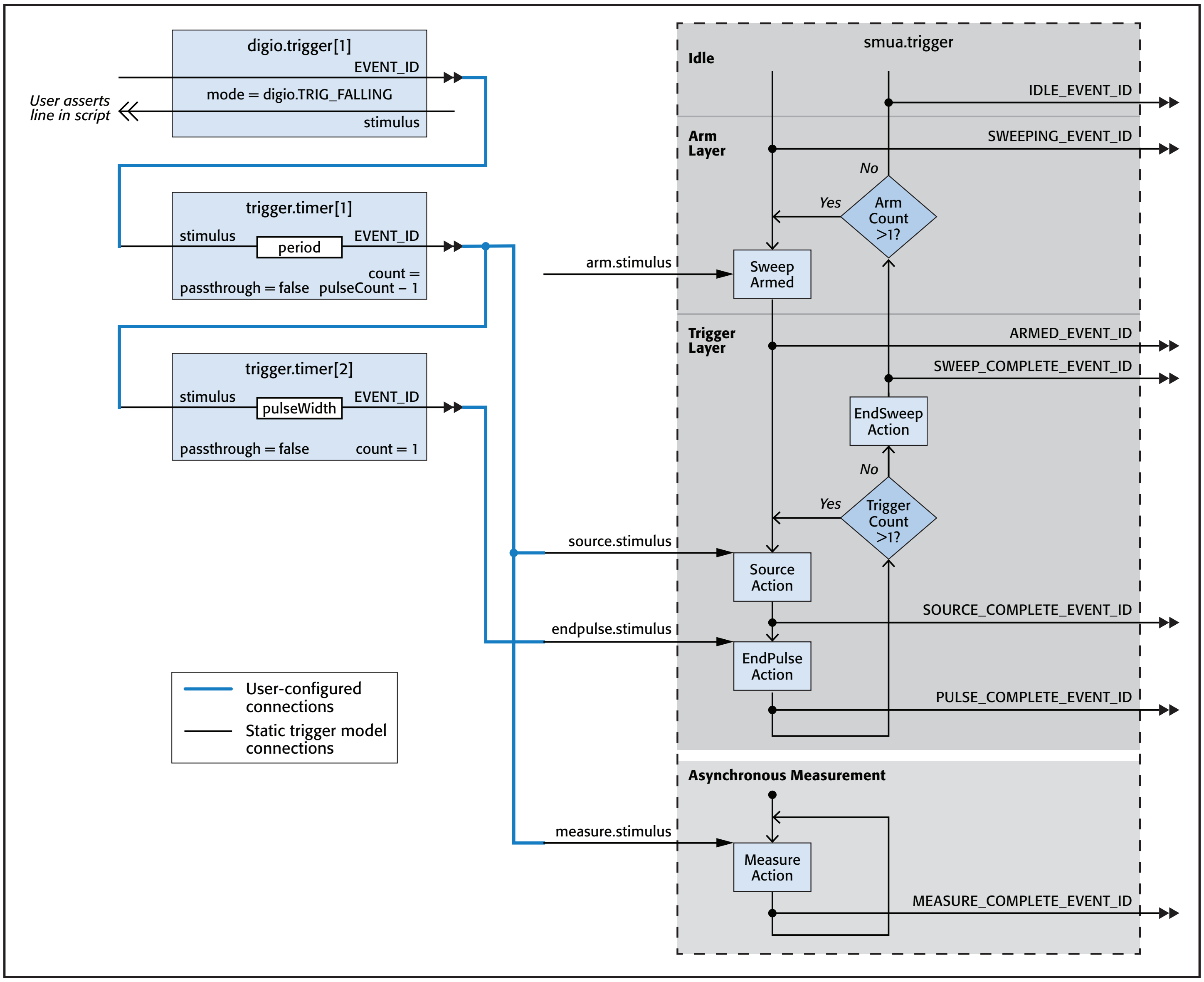

Figure 7. Trigger model configuration for Example #3.

How the Measurement is Made

Timers are again used to trigger the start and end of the pulse. The trigger used to start the pulse is also used to start the measurement process.

Figure 7 diagrams the trigger model configuration. *Appendix D* contains the TSP script for Example #3 and the commands for executing the test and obtaining the results. The results for the example data are shown in *Figure 8*.

Example #4: Pre-Pulse Characterization: Triggering Measurements before the Pulse

Potential Uses

Pulses are sometimes used to provide power stresses to the device. It is useful to note the device state before the stress is applied. This can be done by programming a pulse with a

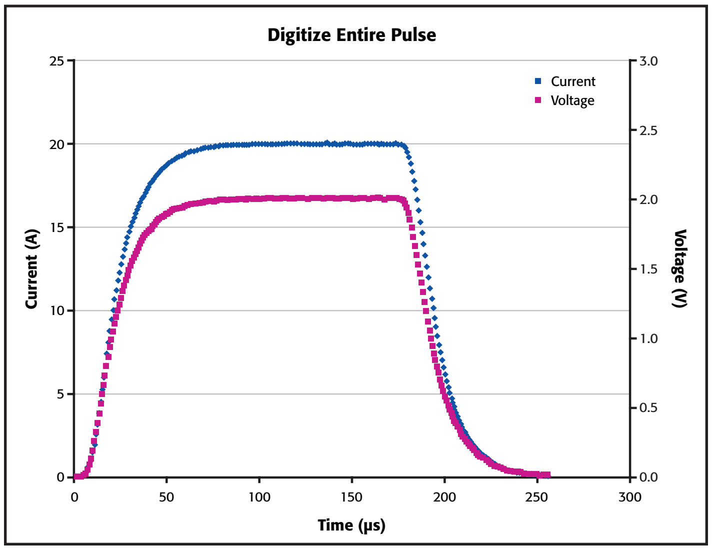

Figure 8. Results from Example #3 showing the voltage and current measurements as performed by the high speed ADCs. There is a 0.1Ω resistive load.

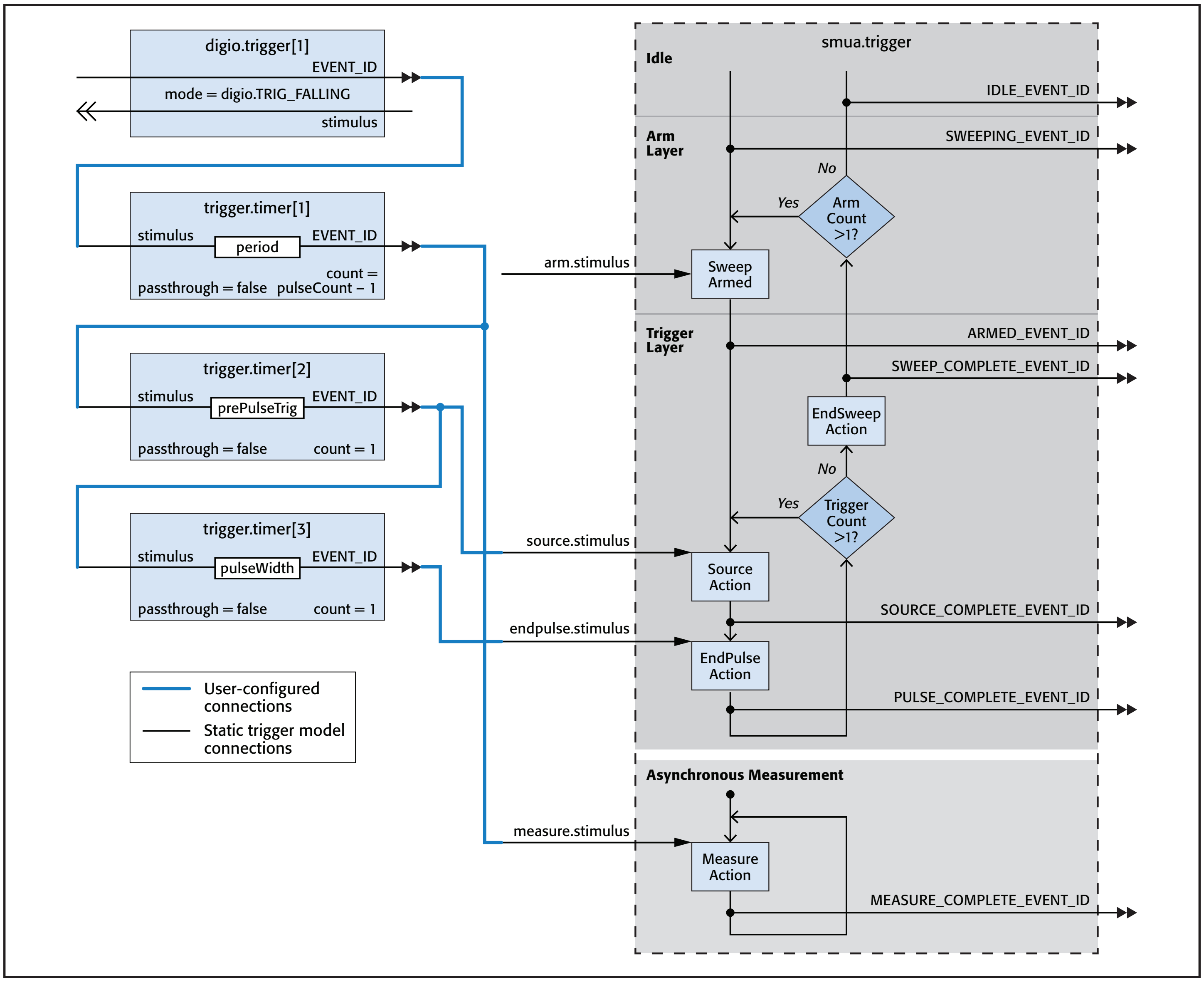

Figure 9. Trigger model configuration for Example #4.

non-zero idle level and triggering the measurements before the pulse.

How the Measurement is Made

This example is arranged so that the user can specify how long before the pulse the measurements should occur. Timers are used to program the start of the measurement and the beginning and end of the pulse.

The trigger model configuration for this example is shown in ***Figure 9***. ***Appendix E*** contains the TSP script of Example #4 and the commands for executing the test and obtaining the results. The results for the example code are shown in ***Figure 10***.

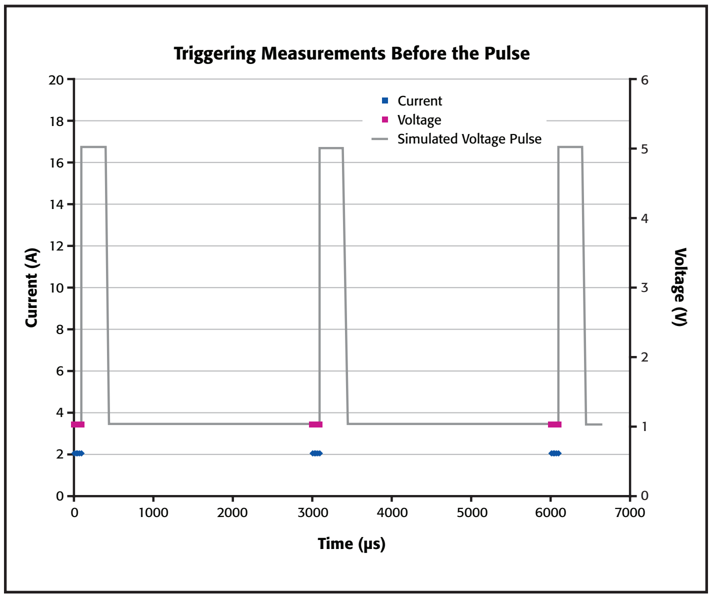

Figure 10. Results from Example #4. There is a 0.5Ω resistive load.

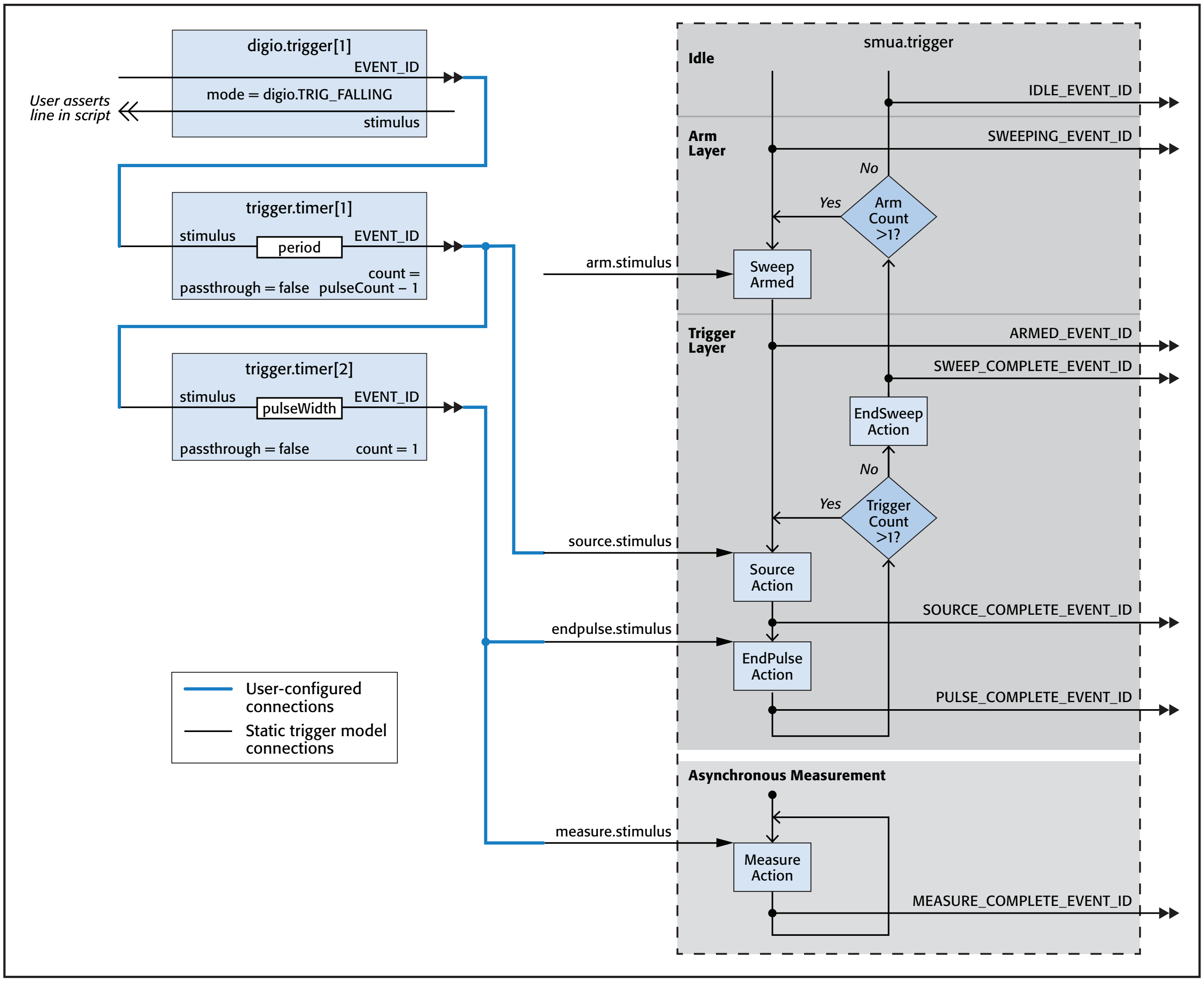

Figure 11. Trigger model configuration for Example #5.

Example #5: Post-Pulse Characterization: Triggering Measurements after the Pulse

Potential Uses

When using pulse testing to stress a device, the device must also be characterized after the stress is applied. This is typically done by sourcing a pre-defined test voltage or current after the pulse. The test level is chosen so as not to cause any additional thermal or electrical stress to the device. The measurement can be made by sourcing a pulse with a non-zero idle level and using the high speed ADCs to perform the measurement. The results from the high speed ADCs indicate how the device recovers from the stress.

How the Measurement is Made

The pulses are timed using trigger timers. The measurement is triggered by the EndPulse event of the trigger model,

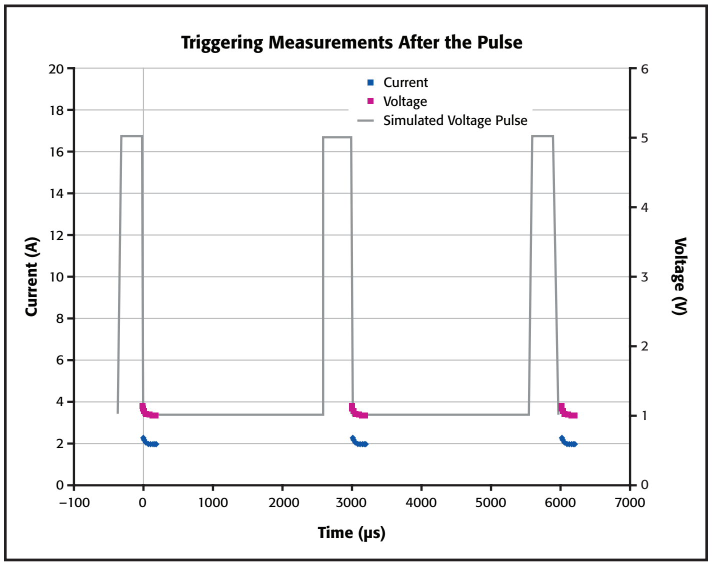

Figure 12. Results from Example #5. There is a 0.5Ω resistive load.

which causes the pulse to return to the idle level. If desired, a measurement delay can be used to postpone the start of the measurements until after the falling edge occurs.

Figure 11 diagrams the trigger model configuration for this example. ***Appendix F*** contains the TSP script of Example #5 and the commands for executing the test and obtaining the results. The results for the example code are shown in ***Figure 12***.

Getting Pulses and Measurements on Time

The ability to pulse in pulse-only current and voltage regions coupled with the ability to make measurements with the high speed ADCs means that the user must carefully consider test timing. In the Model 2651A, the taking of measurements is given priority over source and display operations. Therefore, display updates may not occur and source timing can be compromised if the system is busy making or processing measurements. Consider the following precautions in order to avoid erratic pulse timing. These precautions can also help to prevent damaging devices from the excessive power that can be generated from the Model 2651A's ability to output very high currents in pulse and DC regions.

- For precise timing, always use fixed ranges for source and measurement functions. (Note: Asynchronous measurements mandate the use of fixed ranging for both the integrating and high speed ADCs.)
- When operating in the extended pulse-only operation region, obey all maximum duty cycle and pulse width limitations. The instrument will turn off the output to prevent thermal runaway. However, continuing to issue triggers for source or EndPulse actions may result in source and/or EndPulse action overruns.
- Sustained high speed data acquisition rates may result in undesired pulse timing. This is more likely when the measure interval for the high speed ADC is smaller than 10μs and when a pulse train has a high duty cycle. If a particular configuration results in undesired pulse timing, then abort the present test and perform one of the following to obtain the desired timing: reduce the measurement trigger frequency, reduce the measurement count, or increase the measurement interval.
- In asynchronous operation, all expected measurements must be triggered before the End Sweep action occurs. Refer to Example #5 for an example of how to use a measure delay to postpone the start of measurements after the trigger.
- Any source polarity changes during a sweep incur a 100μs delay before the source level is set. The number 0 is considered a positive value.
  - For negative-going pulses that start from zero, use a 'negative zero', which is a negative number that is very near zero, e.g., -1e-12. See the code for Examples #1, #2, and #3.
  - When the source polarity must be changed during a sweep, the user must account for the polarity change delay in the TSP script in order to obtain proper pulse timing.

A user can monitor the state of the instrument using its status model. Action overruns set bits in different registers of the status model according to the trigger object in which the action overrun is generated. (Each time a register is read, all bits are cleared.) For the SMU trigger object, an action overrun occurs when a new input trigger is detected before the previously triggered action has been started[2]. ***Figure 13*** shows the SMU trigger overrun operation status register set. Note that each action block (Arm, Source, Measure, and EndPulse) has a corresponding overrun bit in the status model. ***Appendix G*** contains the TSP script demonstrating how to monitor the Model 2651A's operation overrun Event register for overruns to the Source, Measure, or EndPulse action blocks.

Each of the SMU action blocks latches (or remembers) one trigger, even if it cannot immediately act upon that trigger. An action overrun is not generated by the latched trigger. However, if multiple triggers are issued while an action is in process, then overruns are generated. For instance, if the Model 2651A is busy making a large number of high speed measurements, it may not be able to respond to the EndPulse event. The first time the EndPulse event is triggered, the SMU will hold off ending the pulse and the result is a pulse with a longer width than expected. If the SMU is still making or processing measurements when the next EndPulse event is triggered, an EndPulse action overrun is generated and the next pulse does not end.

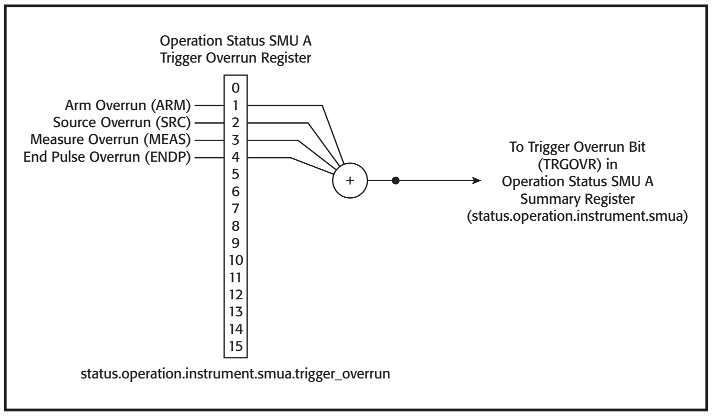

Figure 13. SMU trigger overrun operation status register set.

Conclusion

The high speed ADCs in Keithley's Model 2651A provide a more detailed look at measurements of pulsed waveforms. They can be used for a variety of transient characterization applications previously not possible with an SMU.

2 For asynchronous measurement mode, a measurement action overrun is generated each time a new measurement is triggered while the SMU is taking measurements. No triggers are latched for asynchronous measurements.

Appendix A. Common Functions, runPulse() and printData()

The examples in the following appendices use common functions. The TSP scripts for these functions are listed below. These lines of code may be copied, saved, and run as a script so that the functions are available.

```
--Runs the pulse after it is configured
function runPulse()
   smua.source.output = 1
   delay(0.001)
   smua.trigger.initiate()
   digio.trigger[1].assert()
   waitcomplete()
   smua.source.output = 0

end
--Prints the results from the reading buffers.
function printData()
   if smua.nvbuffer1.n == 0 then
      print("No readings in buffer")
   else
      print("Timestamps\tCurrent\tVoltage")
      for i = 1, smua.nvbuffer1.n do
print(string.format("%g\t%g\t%g", smua.nvbuffer1.timestamps[i], smua.nvbuffer1.readings[i], smua.nvbuffer2.readings[i]))
      end
   end
end
```

Appendix B. Code for Example #1, Digitizing the Top of a Pulse

This is the TSP script that is used to configure the Model 2651A SourceMeter instrument to digitize the top of the pulse when the measurements are made synchronously with the source. The commands for executing the code and obtaining the results are listed below the TSP script.

```
function digitizeTopSync(iAmpl, vLimit, pulseWidth, period, pulseCount, sampleInterval, measCount, measDelay)
   reset()
   errorqueue.clear()
   status.reset()

   local l_iZero --Variable for the idle level of the sweep

   --[[
   The ‘if’ statement below controls the idle level of the sweep. Set to 0 for positive pulses. Set to ‘negative zero’ for
   negative pulses. Eliminates polarity change delay. See “Getting Pulses and Measurements On Time” for more information.
   ]]
   if iAmpl <0 then
      local l_iZero = -1e-13
   else
      local l_iZero = 0
   end

   -- Prepare the reading buffers
   smua.nvbuffer1.clear()
   smua.nvbuffer1.appendmode = 0
   smua.nvbuffer1.collecttimestamps = 1
   smua.nvbuffer1.collectsourcevalues = 0
   smua.nvbuffer2.clear()
   smua.nvbuffer2.appendmode = 0
   smua.nvbuffer2.collecttimestamps = 1
   smua.nvbuffer2.collectsourcevalues = 0

   --Set up the source
   smua.source.func = smua.OUTPUT_DCAMPS
   smua.sense = smua.SENSE_REMOTE --Enables remote sense (4W measurements)
   smua.source.rangei = iAmpl
   smua.source.leveli = l_iZero --Sets the pulse off value.
   smua.source.limitv = vLimit -- Sets the DC voltage limit
   smua.trigger.source.listi({iAmpl}) --Sets the values for the source sweep
   smua.trigger.source.limitv = vLimit --Sets the voltage limit
   smua.trigger.source.action = smua.ENABLE --Enables the source sweep

   --Set up the measurements
   smua.measure.rangev = vLimit -- Set the measure voltage range
   smua.measure.adc = smua.ADC_FAST --Configures the SMU to use the high speed ADC
   smua.measure.interval = sampleInterval --Sets the measurement interval
   smua.measure.delay = measDelay --Sets the initial measurement delay
   smua.measure.count = measCount --Sets the measurement count
   smua.trigger.measure.iv(smua.nvbuffer1, smua.nvbuffer2) -- Configure the SMU to measure both voltage and current
   smua.trigger.measure.action = smua.ENABLE -- Enables measurements in sync with the source sweep

   --Set up a digital I/O line to trigger the start of the period timer
   digio.trigger[1].mode = digio.TRIG_FALLING
   digio.trigger[1].clear()

   -- Timer 1 controls the pulse period by triggering the pulse to begin
   trigger.timer[1].delay = period
   --The timer should always have a count of 1. The ‘if’ statement below configures the timer count.
   if pulseCount > 1 then
      trigger.timer[1].count = (pulseCount - 1)
   else
      trigger.timer[1].count = 1
   end
   trigger.timer[1].passthrough = true --Timer issues an event at the start of the first interval.
   trigger.timer[1].stimulus = digio.trigger[1].EVENT_ID -- Period timer is triggered by a digital I/O line
   trigger.timer[1].clear()

   -- Timer 2 controls the pulse width
   trigger.timer[2].delay = pulseWidth
   trigger.timer[2].count = 1
   trigger.timer[2].passthrough = false --Pulse width timer only issues event when interval elapses
   trigger.timer[2].stimulus = trigger.timer[1].EVENT_ID --Timer starts when period timer issues event
   trigger.timer[2].clear()
```

```
    -- Return to the bias level at the end of the pulse/sweep
    smua.trigger.endpulse.action = smua.SOURCE_IDLE
    smua.trigger.endsweep.action = smua.SOURCE_IDLE
    smua.trigger.count = pulseCount --Sets the number of pulses to generate
    smua.trigger.arm.count = 1 --Sets the number of times to iterate through the entire sweep
    smua.trigger.arm.stimulus = 0
    smua.trigger.source.stimulus = trigger.timer[1].EVENT_ID --Source starts when period timer generates event
    smua.trigger.measure.stimulus = smua.trigger.SOURCE_COMPLETE_EVENT_ID -- Start measuring when the source action is complete
    smua.trigger.endpulse.stimulus = trigger.timer[2].EVENT_ID -- Start EndPulse action as soon as the pulse width timer ends

end
```

Executing the Code

Send the following lines to configure and execute two 20A, 100µs pulses with a 200µs period and a voltage limit of 10V. Fifty measurements are requested, taken at 1µs intervals with an initial measurement delay of 50µs.

```
digitizeTopSync(20, 10, 100e-6, 200e-6, 2, 1e-6, 50, 50e-6)
runPulse()
```

Results

The raw sample data is stored in the dedicated nonvolatile buffers, smua.nvbuffer1 and smua.nvbuffer2. Use the following command to print the data through the remote command interface:

```
printData()
```

Sample results taken using a resistor are shown in ***Figure 6***.

Appendix C. Code for Example #2, Performing a Spot Mean Measurement at the Top of the Pulse

Below is the example code to configure the Model 2651A to perform a spot mean measurement at the top of the pulse. Highlighting is used to distinguish the command lines that are different from those in Example #1 (Appendix B). The commands for executing the code and obtaining the results are listed below the TSP script.

```
function spotMeanTopSync(iAmpl, vLimit, pulseWidth, period, pulseCount, sampleInterval, filtCount, measDelay)
   reset()
   errorqueue.clear()
   status.reset()

   local l_iZero --Variable for the idle level of the sweep

   --[[
   The 'if' statement below controls the idle level of the sweep. Set to 0 for positive pulses. Set to 'negative zero' for
   negative pulses. Eliminates polarity change delay. See "Getting Pulses and Measurements On Time" for more information.
   ]]
   if iAmpl <0 then
      local l_iZero = -1e-13
   else
      local l_iZero = 0
   end

   -- Prepare the reading buffers
   smua.nvbuffer1.clear()
   smua.nvbuffer1.appendmode = 0
   smua.nvbuffer1.collecttimestamps = 1
   smua.nvbuffer1.collectsourcevalues = 0
   smua.nvbuffer2.clear()
   smua.nvbuffer2.appendmode = 0
   smua.nvbuffer2.collecttimestamps = 1
   smua.nvbuffer2.collectsourcevalues = 0

   --Set up the source
   smua.source.func = smua.OUTPUT_DCAMPS
   smua.sense = smua.SENSE_REMOTE --Enables remote sense (4W measurements)
   smua.source.rangei = iAmpl
   smua.source.leveli = l_iZero --Sets the pulse off value.
   smua.source.limitv = vLimit -- Sets the DC voltage limit
   smua.trigger.source.listi({iAmpl}) --Sets the values for the source sweep
   smua.trigger.source.limitv = vLimit --Sets the voltage limit
   smua.trigger.source.action = smua.ENABLE --Enables the source sweep

   --Set up the measurements
   smua.measure.rangev = vLimit -- Set the measure voltage range
   smua.measure.adc = smua.ADC_FAST --Configures the SMU to use the high speed ADC
   smua.measure.interval = sampleInterval --Sets the measurement interval
   smua.measure.delay = measDelay --Sets the initial measurement delay
   smua.measure.count = 1 --Sets the number of measurements to take each time the measure action block is triggered
   smua.measure.filter.type = smua.FILTER_REPEAT_AVG
   smua.measure.filter.count = filtCount
   smua.measure.filter.enable = smua.FILTER_ON
   smua.trigger.measure.iv(smua.nvbuffer1, smua.nvbuffer2) -- Configure the SMU to measure both voltage and current
   smua.trigger.measure.action = smua.ENABLE --Enables measurements in sync with the source sweep

   --Set up a digital I/O line to trigger the start of the period timer
   digio.trigger[1].mode = digio.TRIG_FALLING
   digio.trigger[1].clear()

   -- Timer 1 controls the pulse period by triggering the pulse to begin
   trigger.timer[1].delay = period
   --The timer should always have a count of 1. The 'if' statement below configures the timer count.
   if pulseCount > 1 then
      trigger.timer[1].count = (pulseCount - 1)
   else
      trigger.timer[1].count = 1
   end
   trigger.timer[1].passthrough = true --Timer issues an event at the start of the first interval.
   trigger.timer[1].stimulus = digio.trigger[1].EVENT_ID -- Period timer is triggered by a digital I/O line
   trigger.timer[1].clear()

   -- Timer 2 controls the pulse width
   trigger.timer[2].delay = pulseWidth
```

```
    trigger.timer[2].count = 1
    trigger.timer[2].passthrough = false --Pulse width timer only issues event when interval elapses
    trigger.timer[2].stimulus = trigger.timer[1].EVENT_ID --Timer starts when period timer issues event
    trigger.timer[2].clear()

    -- Return to the bias level at the end of the pulse/sweep
    smua.trigger.endpulse.action = smua.SOURCE_IDLE
    smua.trigger.endsweep.action = smua.SOURCE_IDLE
    smua.trigger.count = pulseCount --Sets the number of pulses to generate
    smua.trigger.arm.count = 1 --Sets the number of times to iterate through the entire sweep
    smua.trigger.arm.stimulus = 0
    smua.trigger.source.stimulus = trigger.timer[1].EVENT_ID --Source starts when period timer generates event
    smua.trigger.measure.stimulus = smua.trigger.SOURCE_COMPLETE_EVENT_ID -- Start measuring when the source action is complete
    smua.trigger.endpulse.stimulus = trigger.timer[2].EVENT_ID -- Start EndPulse action as soon as the pulse width timer ends

end
```

Executing the Code

Send the following lines to configure and execute two 20A, 100μs pulses with a 200μs period and a voltage limit of 10. Two measurements are requested, each with an average of 50 readings taken at 1μs intervals with an initial measurement delay of 50μs.

```
spotMeanTopSync(20, 10, 100e-6, 200e-6, 2, 1e-6, 50, 50e-6)
runPulse()
```

Results

The raw sample data is stored in the dedicated nonvolatile buffers, smua.nvbuffer1 and smua.nvbuffer2. Use the following command to print the data through the remote command interface:

```
printData()
```

The results are shown in ***Figure 6*** next to the raw sample data and are also listed in ***Table 1***.

Appendix D. Code for Example #3, Digitizing the Entire Pulse Including the Rising and Falling Edges

The example code below lists the commands necessary to digitize one to two pulses at the maximum sample rate. (Note: To increase the pulse count, the sample interval or pulse period must be increased to allow sufficient time for data processing and to avoid erratic source timing. For more details, see "Getting Pulses and Measurements on Time" in this application note.) The commands for executing the code and obtaining the results are listed after the TSP script.

```
function digitizePulse(iAmpl, vLimit, pulseWidth, period, pulseCount, sampleInterval)
    reset()
    errorqueue.clear()
    status.reset()

    --[[
    The 'if' statement below controls the idle level of the sweep. Set to 0 for positive pulses. Set to 'negative zero' for
    negative pulses. Eliminates polarity change delay. See "Getting Pulses and Measurements On Time" for more information.
    ]]
    local l_iZero

    if iAmpl <0 then
        l_iZero = -1e-13
    else
        l_iZero = 0
    end

-- Prepare the reading buffers
    smua.nvbuffer1.clear()
    smua.nvbuffer1.appendmode = 0
    smua.nvbuffer1.collecttimestamps = 1
    smua.nvbuffer1.collectsourcevalues = 0
    smua.nvbuffer2.clear()
    smua.nvbuffer2.appendmode = 0
    smua.nvbuffer2.collecttimestamps = 1
    smua.nvbuffer2.collectsourcevalues = 0

    --Set up the source
    smua.source.func = smua.OUTPUT_DCAMPS
    smua.sense = smua.SENSE_REMOTE --Enable remote sense (4W measurements)
    smua.source.rangei = iAmpl
    smua.source.leveli = l_iZero --Sets the pulse off value.
    smua.source.limitv = vLimit -- Sets the DC voltage limit
    smua.trigger.source.listi({iAmpl}) -- Sets the amplitude values for the pulse sweep
    smua.trigger.source.limitv = vLimit --Sets the voltage limit
    smua.trigger.source.action = smua.ENABLE --Enables the source sweep

    --Set up the measurements
    smua.measure.rangev = vLimit -- Set the measure voltage range
    smua.measure.adc = smua.ADC_FAST -- Configures SMU to use high speed aDC
    smua.measure.interval = sampleInterval --Set measurement interval
    smua.measure.delay = 0 --Configures no initial measurement delay
    -- Set the measure count to be 50% greater than the width of the pulse to ensure we capture rising and falling edges of pulse
    smua.measure.count = pulseWidth / smua.measure.interval * 1.5
    smua.trigger.measure.iv(smua.nvbuffer1, smua.nvbuffer2) -- Configure the SMU to measure both voltage and current
    smua.trigger.measure.action = smua.ASYNC -- Configure SMU measurements to occur asynchrnously with source sweep

    --Set up a digital I/O line to trigger the start of the period timer
    digio.trigger[1].mode = digio.TRIG_FALLING
    digio.trigger[1].clear()

    -- Timer 1 controls the pulse period by triggering the pulse to begin
    trigger.timer[1].delay = period

    --The timer should always have a count of 1. The 'if' statement below configures the timer count.
    if pulseCount > 1 then
        trigger.timer[1].count = (pulseCount - 1)
    else
        trigger.timer[1].count = 1
    end

    trigger.timer[1].passthrough = true --Timer issues an event at the start of the first interval.
    trigger.timer[1].stimulus = digio.trigger[1].EVENT_ID -- Period timer is triggered by a digital I/O line
    trigger.timer[1].clear()

    -- Timer 2 controls the pulse width
    trigger.timer[2].delay = pulseWidth
    trigger.timer[2].count = 1
    trigger.timer[2].passthrough = false --Pulse width timer only issues event when interval elapses
    trigger.timer[2].stimulus = trigger.timer[1].EVENT_ID --Timer starts when period timer issue event
    trigger.timer[2].clear()

    -- Return to the bias level at the end of the pulse/sweep
    smua.trigger.endpulse.action = smua.SOURCE_IDLE
    smua.trigger.endsweep.action = smua.SOURCE_IDLE
    smua.trigger.count = pulseCount -- Configures number of pulses to complete
    smua.trigger.arm.count = 1
    smua.trigger.arm.stimulus = 0
    smua.trigger.source.stimulus = trigger.timer[1].EVENT_ID -- Trigger source sweep using events generated by period timer
    smua.trigger.measure.stimulus = trigger.timer[1].EVENT_ID -- Start measuring when the period timer starts the source action
    smua.trigger.endpulse.stimulus = trigger.timer[2].EVENT_ID -- Start EndPulse action as soon as the pulse width timer ends

end
```

Executing the Code

Send the following lines to configure and execute a single 20A, 200μs pulse with a voltage limit of 10V and take high speed ADC measurements at 1μs intervals.

```
digitizePulse(20, 10, 200e-6, 2e-3, 1, 1e-6)
runPulse()
```

Results

The raw sample data is stored in the dedicated nonvolatile buffers, smua.nvbuffer1 and smua.nvbuffer2. Use the following command to print the data through the remote command interface:

```
printData()
```

The results for the example data are shown in ***Figure 8***. The load is a 0.1Ω resistor.

Appendix E. Code for Example #4, Pre-Pulse Characterization: Triggering Measurements before the Pulse

Below is the TSP script for Example #4. The commands for executing the code and obtaining the results are listed after the TSP script.

```
function measThenPulse(vPulse, vIdle, iLimit, pulseWidth, period, pulseCount, prePulseTrig, measCount, sampleInterval)
    reset()
    errorqueue.clear()
    status.reset()

    --If the programmed pulse is negative and biased from zero, then program negative zero
    if vIdle == 0 and vPulse <0 then
        vIdle = -1e-12
    else
        vIdle = vIdle
    end


    -- Prepare the reading buffers
    smua.nvbuffer1.clear()
    smua.nvbuffer1.appendmode = 0
    smua.nvbuffer1.collecttimestamps = 1
    smua.nvbuffer1.collectsourcevalues = 0
    smua.nvbuffer2.clear()
    smua.nvbuffer2.appendmode = 0
    smua.nvbuffer2.collecttimestamps = 1
    smua.nvbuffer2.collectsourcevalues = 0

    --Set up the source
    smua.source.func = smua.OUTPUT_DCVOLTS --Set source function to DC Volts
    smua.sense = smua.SENSE_REMOTE -- Enable remote sense (4W measurements)
    smua.source.rangev = vPulse
    smua.source.levelv = vIdle --Sets the pulse off value.
    --The 'if' statement below sets the DC current limit
    if iLimit > 20 then
        smua.source.limiti = 20
    else
        smua.source.limiti = iLimit
    end
    smua.trigger.source.listv({vPulse}) -- Configures amplitude levels for source sweep
    smua.trigger.source.limiti = iLimit --Sets current limit during pulsing
    smua.trigger.source.action = smua.ENABLE --Enables source sweep

    --Set up the measurements
    smua.measure.rangei = iLimit -- Set the measure current range
    smua.measure.adc = smua.ADC_FAST -- Configures SMU to use high speed ADC
    smua.measure.interval = sampleInterval -- Sets measurement interval
    smua.measure.delay = 0
    smua.measure.count = measCount
    smua.trigger.measure.iv(smua.nvbuffer1, smua.nvbuffer2) -- Configure the SMU to measure both voltage and current
    smua.trigger.measure.action = smua.ASYNC -- Configures measurements to occur asynchrnously from source sweep

    --Set up a digital I/O line to trigger the start of the period timer
    digio.trigger[1].mode = digio.TRIG_FALLING
    digio.trigger[1].clear()


    -- Timer 1 controls the start of the measurement and serves as system clock
    trigger.timer[1].delay = period
    --Timer must always have a count of 1 or more. The following 'if' statement sets the timer count appropriately.
    if pulseCount > 1 then
        trigger.timer[1].count = (pulseCount - 1)
    else
        trigger.timer[1].count = 1
    end

    trigger.timer[1].passthrough = true --Timer issues an event at the start of the first interval.
    trigger.timer[1].stimulus = digio.trigger[1].EVENT_ID -- Period timer is triggered by a digital I/O line
    trigger.timer[1].clear()

    -- Timer 2 controls the delay from the start of measurement to the start of the pulse
    trigger.timer[2].delay = prePulseTrig
    trigger.timer[2].count = 1
    trigger.timer[2].passthrough = false --Only issue event when timer interval has elapsed
    trigger.timer[2].stimulus = trigger.timer[1].EVENT_ID
    trigger.timer[2].clear()

    --Timer 3 controls the pulse width
    trigger.timer[3].delay = pulseWidth
    trigger.timer[3].count = 1
    trigger.timer[3].passthrough = false -- Only issue event when timer interval has elapsed
    trigger.timer[3].stimulus = trigger.timer[2].EVENT_ID
    trigger.timer[3].clear()


    -- Return to the bias level at the end of the pulse/sweep
    smua.trigger.endpulse.action = smua.SOURCE_IDLE
    smua.trigger.endsweep.action = smua.SOURCE_IDLE
    smua.trigger.count = pulseCount --Sets number of pulses to perform
    smua.trigger.arm.count = 1
    smua.trigger.arm.stimulus = 0
    smua.trigger.source.stimulus = trigger.timer[2].EVENT_ID -- Start next pulse once prePulseTrigger time interval has elapsed
    smua.trigger.measure.stimulus = trigger.timer[1].EVENT_ID -- Start measuring when the period timer issues an event
    smua.trigger.endpulse.stimulus = trigger.timer[3].EVENT_ID -- Start EndPulse action as soon as the pulse width timer ends

end
```

Executing the Code

Send the following lines to configure and execute three pulses which pulse from 1 to 5V with a width of 300µs and a period of 3ms with a current limit of 20A. Ten measurements taken at 10µs intervals are taken 100µs before the pulse begins.

```
measThenPulse(5, 1, 20, 300e-6, 3e-3, 3, 100e-6, 10, 10e-6)
runPulse()
```

Results

The raw sample data is stored in the dedicated nonvolatile buffers, smua.nvbuffer1 and smua.nvbuffer2. Use the following command to print the data through the remote command interface:

```
printData()
```

The results for the example code are shown in ***Figure 10***. The load is a 0.5Ω resistor.

Appendix F. Code for Example #5, Post-Pulse Characterization: Triggering Measurements after the Pulse

Below is the TSP script that can be used to program the Model 2651A to perform Example #5. The commands for executing the code and obtaining the results are listed after the TSP script.

```
function pulseThenMeas(vPulse, vIdle, iLimit, pulseWidth, period, pulseCount, measDelay, measCount, sampleInterval)
   errorqueue.clear()
   reset()
   status.reset()

   --If the programmed pulse is negative and biased at zero, then program negative zero
   if vIdle == 0 and vPulse <0 then
      vIdle = -1e-12
   else
      vIdle = vIdle
   end

   -- Prepare the reading buffers
   smua.nvbuffer1.clear()
   smua.nvbuffer1.appendmode = 1
   smua.nvbuffer1.collecttimestamps = 1
   smua.nvbuffer1.collectsourcevalues = 0
   smua.nvbuffer2.clear()
   smua.nvbuffer2.appendmode = 1
   smua.nvbuffer2.collecttimestamps = 1
   smua.nvbuffer2.collectsourcevalues = 0

   --Set up the source
   smua.source.func = smua.OUTPUT_DCVOLTS --Set source function to DC Volts
   smua.sense = smua.SENSE_REMOTE -- Enable remote sense (4W measurements)
   smua.source.rangev = vPulse
   smua.source.levelv = vIdle --Sets the pulse off value.
   --The ‘if’ statement below sets the DC current limit
   if iLimit > 20 then
      smua.source.limiti = 20
   else
      smua.source.limiti = iLimit
   end
   smua.trigger.source.listv({vPulse}) -- Configures amplitude levels for source sweep
   smua.trigger.source.limiti = iLimit --Sets current limit during pulsing
   smua.trigger.source.action = smua.ENABLE --Enables source sweep

   --Set up the measurements
   smua.measure.rangei = iLimit -- Set the measure current range
   smua.measure.adc = smua.ADC_FAST --Configures SMU to use high speed ADC
   smua.measure.interval = sampleInterval -- Sets measurement interval
   smua.measure.delay = measDelay --Set the time from the end of the pulse to the start of measurements using measure delay
   smua.measure.count = measCount --Sets number of measurements
   smua.trigger.measure.iv(smua.nvbuffer1, smua.nvbuffer2) -- Configure the SMU to measure both voltage and current
   smua.trigger.measure.action = smua.ASYNC -- Configures measurements to occur asynchronously with the source sweep

   --Set up a digital I/O line to trigger the start of the period timer
   digio.trigger[1].mode = digio.TRIG_FALLING
   digio.trigger[1].clear()

   -- Timer 1 controls the start of the measurement and serves as system clock
   trigger.timer[1].delay = period
   --Timer must always have a count of 1 or more. The following ‘if’ statement sets the timer count appropriately.
   if pulseCount > 1 then
      trigger.timer[1].count = (pulseCount - 1)
   else
      trigger.timer[1].count = 1
   end

   trigger.timer[1].passthrough = true --Timer issues an event at the start of the first interval.
   trigger.timer[1].stimulus = digio.trigger[1].EVENT_ID -- Period timer is triggered by a digital I/O line
   trigger.timer[1].clear()

   -- Timer 2 controls the pulse width
   trigger.timer[2].delay = pulseWidth
   trigger.timer[2].count = 1
   trigger.timer[2].passthrough = false --Only issue events after timer interval elapses
   trigger.timer[2].stimulus = trigger.timer[1].EVENT_ID -- Start timer interval when period timer issues event
   trigger.timer[2].clear()
```

```
    -- Configure SMU Trigger Model for Sweep

    -- Return to the bias level at the end of the pulse/sweep
    smua.trigger.endpulse.action = smua.SOURCE_IDLE
    smua.trigger.endsweep.action = smua.SOURCE_IDLE
    smua.trigger.count = pulseCount --Set number of pulses to complete
    smua.trigger.arm.count = 1
    smua.trigger.arm.stimulus = 0
    smua.trigger.source.stimulus = trigger.timer[1].EVENT_ID --Trigger source action when period timer generates event
    smua.trigger.measure.stimulus = trigger.timer[2].EVENT_ID -- Trigger measurements when the end pulse action is started
    smua.trigger.endpulse.stimulus = trigger.timer[2].EVENT_ID -- Start EndPulse action as soon as the pulse width timer ends
end
```

Executing the Code

Send the following lines to configure and execute three pulses that pulse from 1V to 5V with a width of 300μs and a period of 3ms with a current limit of 20A. Twenty measurements taken at 10μs intervals are taken 100μs after the pulse ends.

```
pulseThenMeas(5, 1, 20, 300e-6, 3e-3, 3, 100e-6, 50, 10e-6)
runPulse()
```

Results

The raw sample data is stored in the dedicated nonvolatile buffers, smua.nvbuffer1 and smua.nvbuffer2. Use the following command to print the data through the remote command interface:

```
printData()
```

The results for the example code are shown in ***Figure 12***. The load is a 0.5Ω resistor.

Appendix G. Monitoring the Operation Overrun Event Register

The following TSP script demonstrates how to monitor the Model 2651A’s operation overrun Event register for overruns to the Source, Measure, or EndPulse action blocks.

```
function checkSwpStatus()

   trigOvrEvent = status.operation.instrument.smua.trigger_overrun.event
   print(“Overall smu trigger overrun event register value”, trigOvrEvent)

   --For more detail look at bits 2, 3, 4
   --bit 2 = source, bit 3 = measure, bit 4 = endpulse
   --The bit.test function returns a boolean: true if set, false if not set
   sourceOvr = bit.test(trigOvrEvent, 3)
   print(“Source Overrun bit:”, sourceOvr)
   measOvr = bit.test(trigOvrEvent, 4)
   print(“Measure Overrun bit:”, measOvr)
   endPulseOvr = bit.test(trigOvrEvent, 5)
   print(“EndPulse Overrun bit:”, endPulseOvr)

   --If any bit in the register is set, then return a Boolean set to true for overrun occurred.
   if trigOvrEvent > 0 then
      ovrBool = true
   else
      ovrBool = false
   end
   return ovrBool
end
```

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 www.keithley.com	NETHERLANDS Gorinchem 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 Zürich 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. 3116 03.23.11