Data Logging of Power Profiles from Wireless IoT and Other Low-Power Devices Using the DMM6500 6½-digit Bench/System Multimeter

KEITHLEY
A Tektronix Company

Tektronix®

Data Logging of Power Profiles from Wireless IoT and Other Low-Power
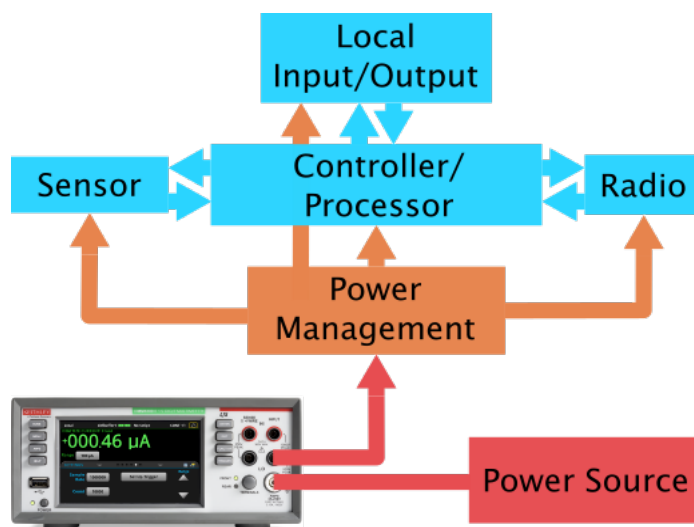Devices Using the DMM6500 6½-digit Bench/System Multimeter

APPLICATION NOTE

Introduction

Generally, data logging is the capture of data for a specific duration of time. The data is then analyzed to determine the performance of a circuit board, module, or a product. The duration of time can last for minutes to months and often involves testing the product in extreme environmental conditions to accelerate the aging process, such as with highly-accelerated life testing (HALT) and highly-accelerated stress testing (HASS). During the development and testing of electrical devices, current, voltage, power, and temperature are all common data points that may need to be stored. For simple devices with stable output under consistent load conditions, incremental data sampling at intervals of seconds to minutes is more than adequate. Devices with many, short duration power states may require much faster sampling.

For wireless devices, such as Internet of Things (IoT) devices, power consumption is of utmost importance. These devices may be left unattended for long periods of time, possibly in hard to reach locations and without access to line power. This will be familiar to designers of biosensors, remote detectors, or battery-powered home products. These wireless devices will operate for long periods in sleep or standby modes, ideally drawing only hundreds of nanoamps to tens of microamps of current. However, these same devices transition into active modes where the processor will wake the device and perform tasks such as reading sensors, executing self-check operations, flashing an LED, and sending a radio frequency transmission (via Bluetooth, LoRa, Wifi, etc.). These activities are more power-intensive and can require hundreds of milliamps or more, though for very short durations. Because of this, fast measurement sampling is necessary to test these devices so that short-duration current spikes do not go undetected, giving a misleading representation of the power consumption of the device. Understanding the power profile of IoT devices across varied states is essential not only to ensure proper behavior but also to optimize the battery life for the device.

While most digital multimeters (DMMs) can take static or slowly changing measurements, they cannot adequately capture the short duration load current bursts that occur when the wireless device transmits. The Keithley DMM6500 addresses this need by having a 16-bit sampling A-to-D converter for digitizing current or voltage measurements

Simplified block diagram of measuring power of a wireless IoT device

Data Logging of Power Profiles from Wireless IoT and Other Low-Power
Devices Using the DMM6500 6½-digit Bench/System Multimeter

APPLICATION NOTE

at up to 1 MSample/s. This 1 µs sampling interval allows capture of ultra-fast load current bursts and helps to show any unwanted transients in the load current. The digitizing functions cover the same measurement ranges as the DMM's traditional DC voltage and current functions of 100 mV to 1000 V or 10 µA to 3 A. Digitizing sensitivity is a low 10 µV and 10 nA for voltage and current, respectively. The DMM6500 also features a full 5 in. (12.7 cm) multi-touch screen display for interacting with recorded data on the instrument with fully-featured graphing and data analysis tools. Collecting logged data can be as simple as inserting a USB drive or transmitting readings during the test to a computer.

This application note illustrates different methodologies that can be used for capturing the load current characteristics of a device or the voltage discharge profile of a battery. We start with an example of a simple data logging method that shows a slower, simpler collection of measured data over time using one of the trigger model templates available on the DMM6500. We then use the DMM6500 in its high-speed digitizing mode, triggering on an active-state device waveform and show how to quickly determine the average current of the signal. We then investigate the idea of streaming measurement data from the DMM6500 directly to a PC and introduce a method to simplify some of the analysis that would be required

Performing Traditional Data Logging

Simple data logging applications are concerned with monitoring one or more electrical signals for extended periods at specific intervals. An engineer may be evaluating the stability of a design that performs just a few modest tasks and want to monitor its behavior long term.

The DMM6500 provides tools to help simplify the setup and execution of the data logging described above. The following example uses a built-in trigger model template to log a DC voltage for one hour, sampling the signal at 1-second intervals. Finally, the entirety of the data is exported to a USB drive for viewing on your PC.

Logging Voltage Readings from the DMM6500 Front Panel

For this example, we assume we are monitoring the voltage across a lithium-ion coin cell battery while it is inside a device under its fully-loaded conditions. We wish to observe the signal for a one-hour period, capturing a reading every second. The DMM6500 can be quickly setup to measure in the manner described above using one of its built-in trigger models. The graph function will be utilized to view the results, and you will capture a screenshot of the display image. Finally, you will export the data to a USB drive.

1. Press the **POWER** button on the front panel to turn on the instrument.

2. On the FUNCTIONS swipe screen, select **DCV** to select the DC Voltage function.

3. Press the **MENU** key.

4. Select **Templates** under the Trigger header.

5. Select the **SimpleLoop** as the Trigger Model.

6. Set the Count to 3600 (60 sec x 60 min, for a reading each second for an hour).

7. Set the Delay to 1 second.

8. Press the **TRIGGER** button. You will be prompted to change the measurement method – respond by selecting **Yes**.

9. From the dialog presented, select **Initiate Trigger Model**.

10. Press the **MENU** key.

11. Select **Graph** under the Views header.

12. The graph will be set to SmartScale by default, this scaling shows you a close-up of the most recently acquired data.

13. Select the **Scale** tab.

14. Set the X-axis Method to All.

Data Logging of Power Profiles from Wireless IoT and Other Low-Power
Devices Using the DMM6500 6½-digit Bench/System Multimeter

APPLICATION NOTE

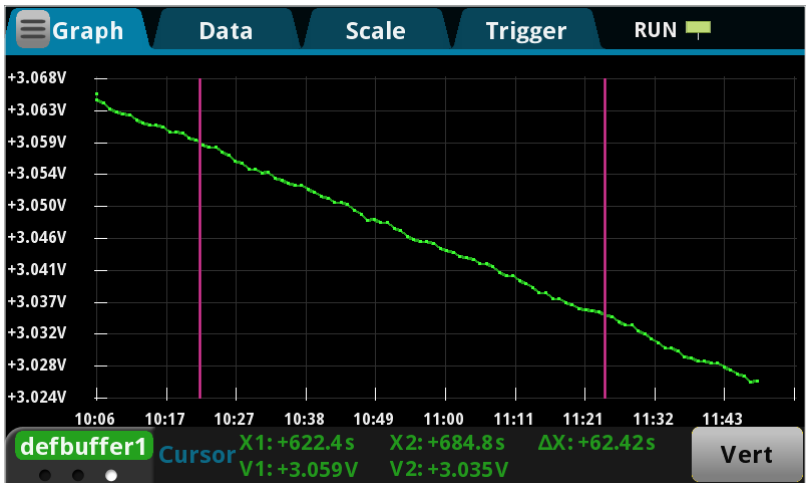| Style	Standard	
Append Mode	1	
Fill Mode	1	
Capacity	100000	
Count	2021	
Base Time Seconds	1517218170	
Base Time Fractional	0.598689284	
Base Time	29:30.6	
Reading	Relative Time	
3.301080526	0	
3.297693003	1.017782	
3.296889096	2.035536	
3.296503709	3.053297	

Figure 1: Portion of CSV file output.

Figure 2: Using the Vertical Cursors on the Graph.

15. Select the **Graph** tab to view the entire data set as it is acquired. You will see Buffer Stats, statistics of the acquired data, at the bottom of the graph. These are automatically generated for all graphed data.

16. Swipe right-to-left on the Buffer Stats information to see the Cursor information.

17. Select **None** to add vertical cursors.

18. Select one of the cursors to move it around the graph. Notice that the cursor statistics update automatically at the bottom.

19. Insert a USB drive to the front panel of the unit.

20. Hold the **HOME** key, and press the **ENTER** key to save a screen capture of the display to the USB drive.

21. Press the **MENU** key.

22. Select **Reading Table** under Views.

23. Drag across the image of the graph to the far right to see the numerical values of the acquired data as it is measured.

24. Select the menu icon, ☰, in the top left of the screen.

25. Select **Save to USB**.

26. Disable Reading Format, Status, and Channel.

27. You can change how time stamps for each reading are shown. Relative is the default and gives the amount of time elapsed from the start of the trigger model for each reading.

28. Select **OK**.

29. Unplug the USB drive and insert it into a computer you would like to use to analyze the data

The data is saved in a .csv format and accessible by most spreadsheet and analysis programs.

This method of data logging is effective in many cases, and the sampling rate for this example is established by the applied delay in the trigger model (1 second). For better resolution on the signal, you could remove the delay and adjust the aperture time to the minimum allowed while the DMM6500 is in integrated measurement mode, 8.333 µs. To sample even faster and provide even more details on the signal in question, you can use the digitizing functions of the DMM6500 to achieve 1 µs interval sampling.

Data Logging of Power Profiles from Wireless IoT and Other Low-Power
Devices Using the DMM6500 6½-digit Bench/System Multimeter

APPLICATION NOTE

Performing Data Logging with Digitizing Functions

Microprocessor-driven remote wireless devices interact with integrated sensors, radio frequency (RF) transceivers, and power management integrated circuits (PMICs) and have varying degrees of impact on the battery that powers them. The designer will want knowledge of transitions in to and out of all device operating states (sleep, standby, sense, transmit, etc.) to build a proper power profile applicable to the device and ensure that it meets their power budget.

A power profile can be simplified to determine a current profile (or average current) for each of the states in question. To properly measure a current profile, you will want to use a high-speed digitizer to observe and account for any transient behavior – intended or not. The DMM6500 is capable of digitizing current or voltage signals at a rate of up to 1 MSample/s and provides level-triggering tools for capturing specific, needed instances.

Some engineers want to observe the power profile of the device over the long-term, streaming all digitized measurement data to a computer for analysis. The Keithley DMM6500 can be controlled by a remote PC using a high-speed communications connection (usually USB or Ethernet) and perform this streaming operation. However, depending on the period over which you want to stream, this can generate gigabytes worth of data that you may not have time to compile and analyze. We recommend using the digitizer, but using the processing power of the DMM6500 to acquire and calculate the cumulative Amp- or Watt-Hours values that you hope to gain.

The following examples show how to perform waveform capture to determine the average current of the signal of interest and how TSP scripting can help to offset or eliminate the need for deep data analysis on streamed results.

Capturing a Current waveform with the Digitizing Function

As noted before, making use of a high-speed digitizer will help you to evaluate the current profile of each operating mode of your device, getting you closer to verifying that it will meet the target power budget. The digitizing functions of the DMM6500 offer the same current and voltage measurement ranges as the traditional DMM functions, but the sampling rate is much higher at 1MS/s. This will help to expose high resolution details on the select signals (both raw data and graphically) that will either give you confidence in your design or provide insight on what might need to be corrected to meet your requirements.

This example will set up a level trigger to automatically capture the current waveform of a Bluetooth device. The DMM6500 is connected in series to a wireless device and its battery as shown in the figure below. A trigger model will be setup to capture the current waveform event of Bluetooth device.
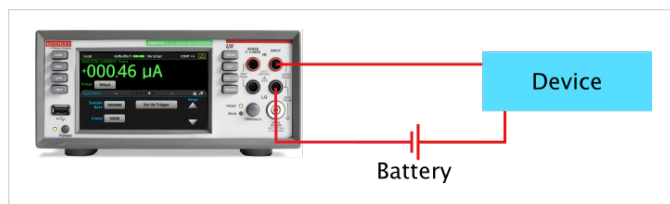

Figure 3: Connections between the DMM and the device

1. Press the **POWER** button on the front panel to turn on the instrument.

2. In the upper right corner of the display, select the annunciator reading **CONT**, and change the triggering to Manual Triggering Mode.

3. On the FUNCTIONS swipe screen select **Digi I** for the digitize current function.

4. Press the **MENU** key.

5. Select **Reading Buffers** under the Measure header.

6. Set the Capacity of defBuffer1 to 1,000,000.

7. Press the **MENU** key.

Data Logging of Power Profiles from Wireless IoT and Other Low-Power
Devices Using the DMM6500 6½-digit Bench/System Multimeter

APPLICATION NOTE

8. Select **Graph** under Views.

9. Select the **Trigger** tab.

10. Set the Event Source to **Waveform**.

11. Select **Analog Edge**.

12. Set the Level to an appropriate amount for the waveform you are trying to capture, in this case, 10 mA.

13. Press the **TRIGGER** key to initiate the digitizing.

14. Activate the event on your device and the DMM will capture the event, as well as half the buffer (500,000 readings) before the event.
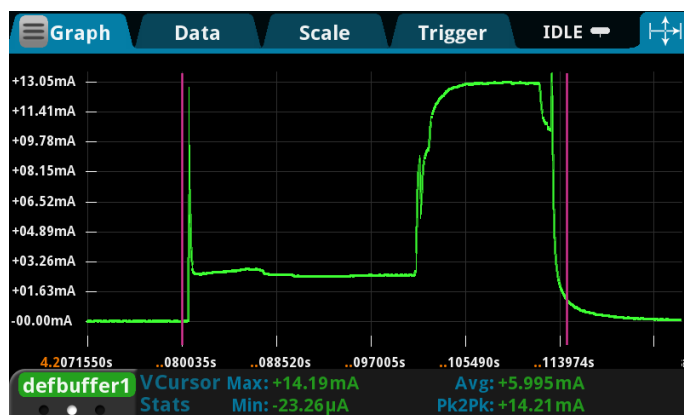

Figure 4: Use the vertical cursors to determine the average current over a portion of the waveform

Select the Graph tab to view the captured waveform. Pinch the graph to zoom into one of the individual waveforms. From here, you can use the cursors to gather more information or export the data just as in the previous example. By placing Vertical cursors around a portion of the waveform, you can use the automatically calculated VCursor Stats to find important information for just that portion, as seen in **Figure 3**. The fast sampling capabilities of the DMM means you can closely examine the features of power drain for all your devices. The extensive memory and robust graph feature make exploring power profiles for power transients, or anomalies, simple. You can see how acquiring this much data for a long period could be troublesome. It only took a couple seconds to capture 1,000,000 data points. One possible solution is to use an external computer to receive and process the data as it comes in.

Streaming Digitized Readings to Your PC

The details above provide insight on how to capture individual segments of average current for any of the given operating modes of your device design. You will also need to test how all those operating modes work together over long periods, very similar to how remote, wireless devices should operate in realistic conditions. A common method for achieving this is to let the device run for a prescribed amount of time while continuously collecting current readings at a high sample rate to derive a more comprehensive power profile. Because of the interest in both current (or power) and time for this method, it is presumed that you will be determining the Amp-hours or Watt-hours to prove that the battery selected for your design will survive the lifespan you planned for based on your power budget.

The DMM6500 supports this data streaming approach, achieving sampling rates up to 100 kS/s while concurrently transferring the digitized data directly to your computer up to 100 k readings per second, depending on the communications protocol selected (the highest using USB). Since the data analysis tends to vary from user to user, we leave that to your discretion. Instead, we present an example that provides insight into the measurement setup, triggering, and data extraction necessary to execute the data streaming using the DMM6500. The details of this example are as follows:

- Main control code is written using Python 3.5 or later and is provided in **Appendix A**.

- The communication protocol is Ethernet, and a TCP/IP socket is used to establish a connection to the DMM6500.

- Once connected, a script file containing TSP command functions is uploaded to the DMM6500. This allows the Python code to execute the more verbose remote setup, triggering and data extraction processes with simpler local function calls. This helps to push more of the processing over to your test instrument, leaving your computer free to perform other tasks. The contents of this script file, `functions.lua`, can be viewed in **Appendix B**.

Data Logging of Power Profiles from Wireless IoT and Other Low-Power
Devices Using the DMM6500 6½-digit Bench/System Multimeter

APPLICATION NOTE

- Data is retrieved from the DMM6500 in specific quantities (or "chunks" as defined in the code), received as binary floating-point data, and then converted to ASCII.

- The ASCII format readings are written to a data file saved on the controlling computer.

Logging Amp-Hour and Watt-hour readings from the DMM6500 Using TSP

While the data streaming method described above may produce gigabytes worth of information for you to analyze using your software of choice, some users want to avoid working with all that data. These users prefer an immediate, cumulative value to be presented to them indicating the Amp-hours or Watt-hours for the operating device. With the power of TSP scripting, we can customize the operation of the DMM6500 to do that.

An example script for accomplishing this with the DMM6500 is available on the web at www.tek.com/keithley. The script code is also presented in its raw form in **Appendix C**, with the highlights of what it accomplishes as follows:

- User-selectable digitizing from 1 k to 125 kS/s, with the higher rates better helping to capture and account for transient signals

- Choice of calculating Amp-Hours or Watt-Hours

- Amp- or Watt-hours displayed on the instrument front panel along with the average current of your device

- Graphical output of the Amp- or Watt-hours data

Whether you choose to download the script from the website in its entirety or copy and paste it into your own editor, we provide the following details on how to load and run the script.

Running the Script

This example uses a TSP Script and Keithley Instruments' Test Script Builder (available on the web at www.tek.com/keithley) to record the Amp-hours consumed by a wireless device for up to 30 days. The TSP Script will be loaded onto the DMM via USB drive but it can also be executed from an external computer running Keithley Instruments' Test Script Builder.

The code itself can be found in Appendix C at the end of this Application Note. It uses the DMM's digitize current function to measure current values to a recycled buffer. Once a second, the mean current from this recycled buffer and the elapsed time of the test are used to calculate the Amp-hours (or the watt-hours, if the source voltage is given) of the device, which is then logged to a permanent buffer. This method captures anomalous power draw caused by small peaks in current while keeping a more manageable number of data points.

1. Press the **POWER** button on the front panel to turn on the instrument.

2. Copy the script located in Appendix C of this Application Note onto a USB drive.

3. Insert the USB drive into the front panel of the instrument.

4. Select **No Script** at the top of the display and select the script copied from this Application Note (i.e. usb1/PowerHr_Meter).

5. Select **Amp-Hours**.

6. Select a Sample Rate of 50,000.

7. Select a current range that covers the maximum current draw for your device, even small current peaks should fit within the range to accurately characterize the device.

8. The measurement will start immediately, the current amp-hours consumed, and the average current are displayed on the HOME screen of the DMM.

Data Logging of Power Profiles from Wireless IoT and Other Low-Power
Devices Using the DMM6500 6½-digit Bench/System Multimeter

APPLICATION NOTE

9. Press the **MENU** key.

10. Select **Graph** under the Views header.

11. Select the **Data** tab.

12. Select **defBuffer1** then select **Remove**.

13. Select **Add** and then **ampHrsBuffer** to add the amp-hours measurement to the Graph.

14. Select the **Graph** tab to view the amp-hours reading as it arrives.

15. Press the **TRIGGER** key when you are finished acquiring data to stop the trigger model.
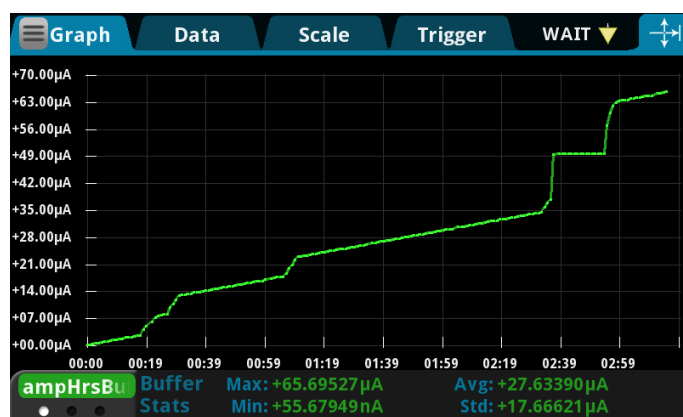

Figure 7: Example of an amp-hours measurement graph. Changes in slope equate to changes in power consumption.

The data can be exported to a .csv file for analysis on an external computer. The amp hour measurement may be compared against expected values or searched for anomalous power draw (which could be shown by a sudden change in slope). Some other common uses for this type of measurement would be determining the specifications for a battery that needs to power a device, or generating specifications for power consumption.

Data Logging Multiple Measurements at Multiple Places

In addition to the examples in this Application Note, the DMM6500 can be used to log data using any of its 15 measurement functions including functions such as temperature with thermocouples, RTDs, or thermistors and voltage ratio of two voltages. Using an optional switch card, the DMM6500 can utilize up to 10 measurement channels for logging data at different points on a device or on multiple devices. When even more channels are needed, a Keithley DAQ6510 can utilize up to 80 measurement channels with the same capabilities of the DMM.

Conclusion

With battery life so important to consumers, understanding and minimizing the load current of IoT devices is important to achieving success in the market. The Keithley DMM6500 6 ½-Digit Bench/System Multimeter offers the needed performance to measure the load current of a device in all its operating states, from the sleep state to the transmit state. With this measurement data and the DMM6500's analysis capability, the designer has all the insight needed to understand a device's total power profile.

Data Logging of Power Profiles from Wireless IoT and Other Low-Power
Devices Using the DMM6500 6½-digit Bench/System Multimeter

APPLICATION NOTE

Appendix A: Python script to execute data streaming

This python 3.5 script uses the TSP functions in Appendix B to stream digitized current readings over a LAN connection to an external computer. You can expect about 60,000 readings/second with this script, but this speed is limited by the Ethernet cable of the LAN connection. Higher streaming speeds are possible by using a faster data bus such as a USB connection with PyVISA. PyVISA information is available at https://pyvisa.readthedocs.io.

Before running this script, you should make some changes:

- Set the `ip_address` variable to match the ip address of your DMM6500, default 192.168.1.78.

- Set the `seconds_to_capture` variable to the number of seconds you wish to log data, default 10.

- Make sure the TSP functions from Appendix B are in a file called `functions.lua` and are in the same directory this script is in.

```python
#!/usr/bin/python
import socket
import struct
import math
import time

#settings
seconds_to_capture = 10 # Modify this value to adjust your run time.
#minutes_to_capture = seconds_to_capture * 60
sample_rate = 60000 # NOTE: 60kS/s is the max rate we have observed under
                    #       certain conditions/circumstances. To attain
                    #       higher sampling and data transfer rates, use
                    #       USB.
chunkSize = 249     # This value is the max binary format transfer value
                    # we can implement for data transfer, and is limited
                    # by the ethernet protocol where the max frame size
                    # is < 1500 bytes, and this includes header/trailer
                    # information for each of the networking layers
                    # involved in the TCP/IP (physical, data link, network,
                    # and transport). The "chunkSize" variable defines how
                    # many readings to to transfer for a given poll of the
                    # instrument.
ip_address = "192.168.1.71"     # Place your instrument's IP address here.
output_data_path = "data.txt"   # This is the output file that is created which
                                # will hold your readings provided in ASCII
                                # format in a text file.
functions_path = "functions.lua"    # This file holds the set of TSP (Lua-
                                    # based) functions that are called by
                                    # the Python script to help minimize the
                                    # amount of bytes needed to setup up and
                                    # more importantly, extract readings from
                                    # the instrument. The file is opened and
                                    # written directly to instrument memory.

#helpers
# implement "chunkSize" instead of a fixed value
chunks = math.floor((seconds_to_capture * sample_rate) / chunkSize)

def load_functions(s):
    # This function opens the functions.lua file in the same directory as
    # the Python script and trasfers its contents to the DMM6500's internal
    # memory. All the functions defined in the file are callable by the
    # controlling program.
    func_file = open(functions_path, "r")
    contents = func_file.read()
```

Data Logging of Power Profiles from Wireless IoT and Other Low-Power
Devices Using the DMM6500 6½-digit Bench/System Multimeter

APPLICATION NOTE

```python
    func_file.close()
    s.send("if loadfuncs ~= nil then "
                "script.delete('loadfuncs') "
            "end\n".encode())
    s.send("loadscript loadfuncs\n{0}\nendscript\n"
        .format(contents)
        .encode())
    s.send("loadfuncs()\n".encode())
    print(s.recv(100).decode())

def send_setup(s):
    # This function sends a string that includes the function
    # call and arguments that set up the DMM6500 for digitizing
    # current for the requested time and sample rate.
    s.send("do_setup({0}, {1})\n"
        .format(seconds_to_capture, sample_rate)
        .encode())
    s.recv(10)

def send_trigger(s):
    # This function sends a string that calls the function
    # to trigger the instrument.
    s.send("trig()\n".encode())
    s.recv(10)

def write_block(ofile, floats):
    # This function writes the floating point data to the
    # target file.
    for f in floats:
        ofile.write("{0:.4e}\n".format(f))

def get_block(s):
    # This function extracts the binaray floating point data
    # from the DMM65000.
    s.send("get_data()\n".encode())
    response = s.recv(1024)
    return response

def set_display(screen, state):
    # This function changes the display view and backlight settings
    s.send("disp_state({0}, {1})\n".format(screen, state).encode())
    s.recv(10)

#configure, trigger, transfer
s = socket.socket()                     # Establish a TCP/IP socket object
s.connect((ip_address, 5025))           # Connect to the instrument
ofile = open(output_data_path, "w") # Open/create the target data

load_functions(s)
send_setup(s)
set_display(16, 0)                          # Change to MENU screen; backlight off
send_trigger(s)

t1 = time.time()                            # Start the timer...
for i in range(0, int(chunks)):         # Loop to collect the digitized data
    write_block(ofile, get_block(s))# Write the data to file
    if i % 10 == 0:                         # This is here for debug purposes, printing
        print("{0:.1f}%".format(i/chunks * 100)) # out the % of run time elapsed
                                            # and technically it could be commented out.
t2 = time.time()                            # Stop the timer...

set_display(0, 1)                           # Change to HOME screen; backlight on
ofile.close()                               # Close the data file.
s.close()                                   # Close the socket.
```

Data Logging of Power Profiles from Wireless IoT and Other Low-Power
Devices Using the DMM6500 6½-digit Bench/System Multimeter

APPLICATION NOTE

```python
# Notify the user of completion and the data streaming rate achieved.
print("done")
print("{0:.0f} rdgs/s".format((chunks * chunkSize)/(t2-t1)))

input("Press Enter to continue...")
```

Appendix B: TSP script file to support data streaming

These functions are used by the python script in **Appendix A**. They should be put in a file named `functions.lua` in the same directory that the python script from Appendix A will be run from. The python script loads these functions onto the instrument's internal memory so they can be executed with minimal delay. This also reduces the communication required across the data bus, reducing chances for error.

```lua
readings_captured = 0

function do_setup(capture_time, sample_rate)
    reset()
    dmm.digitize.func = dmm.FUNC_DIGITIZE_CURRENT
    dmm.digitize.range = 1
    dmm.digitize.samplerate = sample_rate
    format.data = format.REAL32

    trigger.model.setblock(1, trigger.BLOCK_DIGITIZE,
                              defbuffer1,
                              trigger.COUNT_INFINITE)

    trigger.model.setblock(2, trigger.BLOCK_DELAY_CONSTANT,
                              capture_time)

    trigger.model.setblock(3, trigger.BLOCK_DIGITIZE,
                              defbuffer1,
                              trigger.COUNT_STOP)
    waitcomplete()
    print("ok")
end

function trig()
    readings_captured = 0
    trigger.model.initiate()
    print("ok")
end

function get_data()
    chunker = 249
    while buffer.getstats(defbuffer1).n - readings_captured < chunker do
        delay(0.001)
    end
    local index1 = math.mod(readings_captured, 100000) + 1
    local index2 = index1 + (chunker - 1)
    if index2 > 100000 then
        index2 = 100000
    end
    printbuffer(index1, index2, defbuffer1)
    readings_captured = readings_captured + chunker
end

function disp_state(screen, state)
    if screen == 0 then
        display.changescreen(display.SCREEN_HOME)
    elseif screen == 1 then
```

Data Logging of Power Profiles from Wireless IoT and Other Low-Power
Devices Using the DMM6500 6½-digit Bench/System Multimeter

APPLICATION NOTE

```
        display.changescreen(display.SCREEN_GRAPH)
    elseif screen == 16 then
        display.changescreen(16)
    end

    if state == 0 then
        display.lightstate = display.STATE_LCD_OFF
    else
        display.lightstate = display.STATE_LCD_100
    end
    print("ok")
end

print("functions loaded")
```

Appendix C: Logging Amp-hour or Watt-hour Measurements using TSP Script

This TSP script can be run on a Keithley Model DMM6500 for measuring Amp-hour or Watt-hours of a device. The script should be placed in a text file and given a name such as `PowerHr_Meter.tsp`. It can either be run from within Keithley Instruments Test Script Builder program on a PC connected to a Model DMM6500 or run directly from a USB drive inserted into the DMM's front panel.

When run, the script will first ask whether amp-hours or watt-hours are to be measured. The watt-hour option will further ask for the applied DC voltage, the amp-hour measurement is multiplied by this applied voltage to get watt-hours. You must then define the sampling rate, defaulted to 50 kS/s. Finally, the current range must be specified to simplify calculations.

```
-- create functions

function setup_DMM6500_buffer(BufSize)
    dciBuffer = buffer.make(BufSize, buffer.STYLE_STANDARD)
    dciBuffer.clear()
    buffer.clearstats(dciBuffer)
    dciBuffer.capacity = 1 * BufSize
    dciBuffer.fillmode = buffer.FILL_CONTINUOUS
end -- function

function setup_DMM6500_measure(sampleRate, measRange)
    -- setup our refilling buffer
    setup_DMM6500_buffer(sampleRate)   -- BufSize = sampleRate = 1 second of buffering
    opc()

     -- setup measure type, ranges, etc.
    dmm.digitize.func = dmm.FUNC_DIGITIZE_CURRENT
    opc()

    dmm.digitize.range = measRange

    dmm.digitize.samplerate = sampleRate

    dmm.digitize.aperture = dmm.APERTURE_AUTO
    --Changing count is optional.  The reading buffer capacity is the determining factor
    dmm.digitize.count = 1   -- CANNOT be zero; 1 to 55Million

    -- control the swipe screen
    display.clear()
    display.changescreen(display.SCREEN_USER_SWIPE)

    -- clear any existing trigger blocks
    trigger.clear()
```

Data Logging of Power Profiles from Wireless IoT and Other Low-Power
Devices Using the DMM6500 6½-digit Bench/System Multimeter

APPLICATION NOTE

```lua
    trigger.model.load("Empty")
    opc()

    --Define a trigger model that will capture until we push front panel trigger button
    trigger.model.setblock(1, trigger.BLOCK_BUFFER_CLEAR, dciBuffer)
    trigger.model.setblock(2, trigger.BLOCK_DELAY_CONSTANT, 0)
    trigger.model.setblock(3, trigger.BLOCK_DIGITIZE, dciBuffer, trigger.COUNT_INFINITE)
    trigger.model.setblock(4, trigger.BLOCK_WAIT, trigger.EVENT_DISPLAY) -- wait until the
TRIGGER key is pressed
    trigger.model.setblock(5, trigger.BLOCK_DIGITIZE, dciBuffer, trigger.COUNT_STOP) -- stop
making digitized measurements

    opc()
end -- function

function my_dmm6500_waitcomplete(useWattHrs, dcvVal)
    local i = 1
    local cbIndex = 1
    local tempVal = 0

    -- check trigger model state on Amp-Hr meter (DMM6500)
    present_state, n = trigger.model.state()  -- state, present block number

    --STATE_RUNNING, IDLE, WAITING, EMPTY, FAILED, ABORTING, ABORTED, BUILDING
    while present_state == (trigger.STATE_WAITING or trigger.STATE_RUNNING) do
        reading_stats = buffer.getstats(dciBuffer)
        i_avg = reading_stats.mean
        runtime = dciBuffer.relativetimestamps[dciBuffer.n]
        AmpHrs = i_avg * runtime/3600
        if useWattHrs == 0 then
            display.settext(display.TEXT1, string.format("Amp-Hrs: %.4e", AmpHrs));
            display.settext(display.TEXT2, string.format("Avg. I: %.6e A", i_avg));
            tempVal = AmpHrs
            buffer.write.reading(ampHrsBuffer, tempVal, runtime)
        else
            WattHrs = AmpHrs * dcvVal
            display.settext(display.TEXT1, string.format("Watt-Hrs: %.4e", WattHrs));
            display.settext(display.TEXT2, string.format("Avg. I: %.6e A", i_avg));
            tempVal = WattHrs
            buffer.write.reading(wattHrsBuffer, tempVal, runtime)
        end

        delay(1)
        i = i + 1
        present_state, n = trigger.model.state()  --update the trigger model state var
        end  -- while loop
end -- function

function get_amphrs()
    present_state, n = trigger.model.state()

    reading_stats = buffer.getstats(defbuffer1)
    runtime = defbuffer1.relativetimestamps[defbuffer1.n]
    i_avg = reading_stats.mean
    AmpHrs = i_avg * runtime/3600
end --function

function set_dci_range()
    optionID = display.input.option("Select current range", "1A", "100mA", "10mA", "1mA",
"100uA", "10uA")
    if optionID == display.BUTTON_OPTION1 then -- 1A
        return 1.0
    elseif optionID == display.BUTTON_OPTION2 then -- 100mA
        return 100e-3
```

Data Logging of Power Profiles from Wireless IoT and Other Low-Power
Devices Using the DMM6500 6½-digit Bench/System Multimeter

APPLICATION NOTE

```lua
    elseif optionID == display.BUTTON_OPTION3 then -- 10mA
        return 10e-3
    elseif optionID == display.BUTTON_OPTION4 then -- 1mA
        return 1e-3
    elseif optionID == display.BUTTON_OPTION5 then -- 100uA
        return 100e-6
    elseif optionID == display.BUTTON_OPTION6 then -- 10uA
        return 10e-6
    end
end

function set_output_hrs_format()
    optionID = display.input.option("Select Computation Option", "Amp-Hours", "Watt-Hours")
    if optionID == display.BUTTON_OPTION1 then -- Amp-Hrs
        return 0
    elseif optionID == display.BUTTON_OPTION2 then -- Watt-Hrs
        return 1
    end
end -- function

function get_user_sample_rate()
    return display.input.number("Sample Rate", display.NFORMAT_INTEGER, 50000, 1000, 125000)
end -- function

function get_user_dcv_value()
    -- for the watt-hours, have the user input the applied voltage to their device
    return display.input.number("DCV Level Applied", display.NFORMAT_DECIMAL, 3.25, 0.0, 24.0)
end — function
-- *********************** MAIN PROGRAM ***********************

reset()  --reset the DMM6500

eventlog.clear()

-- set default sample_rate and current_range
local sample_rate = 15e3
local DMMcurrentMeasRange = 0.01
local dcvVal = 0.0

-- downsize the default buffers to ensure room for the new ones
defbuffer1.capacity = 10
defbuffer2.capacity = 10

-- let us size this for 1 sample per second for up to 30 days: 60*60*24*30 = 2,592,000
local hrsFormat = set_output_hrs_format()
if hrsFormat == 0 then    -- provide semi-acceptable units to be visible on the graph and in the
reading table
    ampHrsBuffer = buffer.make(2592000, buffer.STYLE_WRITABLE)
    buffer.write.format(ampHrsBuffer, buffer.UNIT_AMP, buffer.DIGITS_6_5)
else
    wattHrsBuffer = buffer.make(2592000, buffer.STYLE_WRITABLE)
    buffer.write.format(wattHrsBuffer, buffer.UNIT_WATT, buffer.DIGITS_6_5)
end

if hrsFormat == 1 then
    dcvVal = get_user_dcv_value()
end

dmm.digitize.func = dmm.FUNC_DIGITIZE_CURRENT
dmm.digitize.range = DMMcurrentMeasRange
```

Data Logging of Power Profiles from Wireless IoT and Other Low-Power
Devices Using the DMM6500 6½-digit Bench/System Multimeter

APPLICATION NOTE

```lua
sample_rate    = get_user_sample_rate()-- let the user select the sample rate to use
DMMcurrentMeasRange  = set_dci_range()-- let the user select the current range to use
setup_DMM6500_measure(sample_rate, DMMcurrentMeasRange)

-- start our DMM6500 High Speed Digitizing
trigger.model.initiate()
delay(0.5)  -- allow some data to accumulate....

-- start the DMM6500 Amp-Hr status reporting loop
-- Press TRIGGER button to exit the loop
my_dmm6500_waitcomplete(hrsFormat, dcvVal)

-- clean up DMM6500 (Amp-Hr)
trigger.model.abort()
```

Contact Information:

Australia* 1 800 709 465
Austria 00800 2255 4835
Balkans, Israel, South Africa and other ISE Countries +41 52 675 3777
Belgium* 00800 2255 4835
Brazil +55 (11) 3759 7627
Canada 1 800 833 9200
Central East Europe / Baltics +41 52 675 3777
Central Europe / Greece +41 52 675 3777
Denmark +45 80 88 1401
Finland +41 52 675 3777
France* 00800 2255 4835
Germany* 00800 2255 4835
Hong Kong 400 820 5835
India 000 800 650 1835
Indonesia 007 803 601 5249
Italy 00800 2255 4835
Japan 81 (3) 6714 3086
Luxembourg +41 52 675 3777
Malaysia 1 800 22 55835
Mexico, Central/South America and Caribbean 52 (55) 56 04 50 90
Middle East, Asia, and North Africa +41 52 675 3777
The Netherlands* 00800 2255 4835
New Zealand 0800 800 238
Norway 800 16098
People's Republic of China 400 820 5835
Philippines 1 800 1601 0077
Poland +41 52 675 3777
Portugal 80 08 12370
Republic of Korea +82 2 565 1455
Russia / CIS +7 (495) 6647564
Singapore 800 6011 473
South Africa +41 52 675 3777
Spain* 00800 2255 4835
Sweden* 00800 2255 4835
Switzerland* 00800 2255 4835
Taiwan 886 (2) 2656 6688
Thailand 1 800 011 931
United Kingdom / Ireland* 00800 2255 4835
USA 1 800 833 9200
Vietnam 12060128

* European toll-free number. If not accessible, call: +41 52 675 3777

Rev. 02.2018

KEITHLEY
A Tektronix Company

Find more valuable resources at TEK.COM