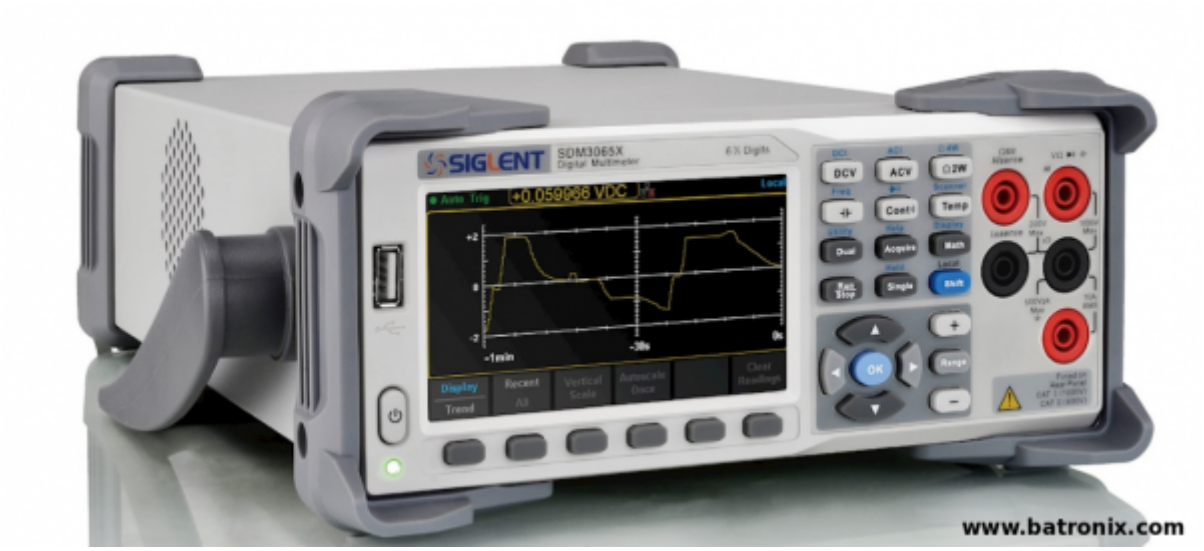
Siglent SDM3065x Bench Multimeter



SCPI and Fast Sampling

This bench multimeter needs a very special procedure, to get the advantage of fast measurements.

This article should be applicable with minor adjustments for the all the new Siglent DMM's: SDM3045X, SDM3055, SDM3055A, SDM3065X

Problem

The default command `Measure:Voltage:DC?` takes about 2s. What next came to mind, was the command `Read?`. It answers very fast, but is delivers measurements from the past - could be some minutes or even hours old. Even clearing the buffer with R? doesn't change that situation.

Another Problem is, that when the DMM got it's first `Measure` command, the display will not update any more, also mentioned by a user in the EEVblog Forum, see Links [2].

Solution

For both problems I gladly found the solution. The approach is like this:

- set the sampling counter to MAX (600 Mio)
- clear the buffer with R? (to avoid getting an old value on the first reading)
- set trigger source to 'BUS'
- initiate the DMM (arms the trigger)
- trigger it by sending *TRG
- read samples with R? 1

When the DMM isn't asked for any value, it will sample with the preset NPLC setting (200ms by default) and therefore also updates the LCD reading.
If we want the latest value from the memory, we query for it with R? 1.

Python Code

Below you can find a python class for communicating with it. It implements the upper approach.

A typical usage could be:

```
import siglent_sdm3065x
dmm = siglent_sdm3065x.SDM3065X('10.0.0.114')
dmm.reset()

# setup:
v = dmm.getVoltageDC('20V',0.05) # set NPLC to 0.05 which is 1ms in a 50Hz grid
print(v)

# further readings:
for i in range(10):
    print(dmm.read())
```

siglent_sdm3065x.py

```
# Python3 Class for controlling a Siglent SDM3065x Bench Multimeter via Ethernet and SCPI

# MIT License
# Copyright 2018 Karl Zeilhofer, Team14.at

# Permission is hereby granted, free of charge, to any person obtaining a copy of
# this software and associated documentation files (the "Software"), to deal in the
# Software without restriction, including without limitation the rights to use, copy,
# modify, merge, publish, distribute, sublicense, and/or sell copies of the Software,
# and to permit persons to whom the Software is furnished to do so, subject to the
# following conditions:
#
# The above copyright notice and this permission notice shall be included in all
# copies or substantial portions of the Software.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED,
```

```
# INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A
# PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS
OR COPYRIGHT
# HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER
IN AN ACTION OF
# CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION
WITH THE SOFTWARE
# OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

import decimal
import socket
import sys

import time

_timeoutCmd = 10  # seconds
_timeoutQuery = 10  # seconds

class SDM3065X:
    def __init__(self, ip):
        self.ipAddress = ip
        self._port = 5024 # TCP port, specific to these devices, the
standard would be 5025


        self.NPLC = ['100','10','1','0.5','0.05','0.005'] # available
Integration durations in number of powerline cycles
            # set with Voltage:DC:NPLC or similar
        self.voltageRangeDC =
['200mV','2V','20V','200V','1000V','AUTO'] # set with VOLTage:DC:RANGe
        self.voltageRangeAC = ['200mV','2V','20V','200V','750V','AUTO']
# set with VOLTage:AC:RANGe
        self.currentRangeDC =
['200uA','2mA','20mA','200mA','2A','10A','AUTO'] # set with
CURRent:DC:RANGe
        self.currentRangeAC =
['200uA','2mA','20mA','200mA','2A','10A','AUTO'] # set with
CURRent:AC:RANGe

        self._PrintDebug = True


    def _abort(self, msg):
        #  TODO 2: something like this:
traceback.print_stack(sys.stdout)
        print('Abort for device IP=' + self.ipAddress + ':' +
str(self._port))
        print(msg)
        print('ERROR')
        sys.exit(-1)
```

```
    def _debug(self, msg):
        if(self._PrintDebug):
            print(self.ipAddress + ':' + str(self._port) + ': ' + msg)

    def reset(self):
        self._runScpiCmd('*RST')


    def getVoltageDC(self, range='AUTO', integrationNPLC='10'):
        self._runScpiCmd('abort')  # stop active measurement
        self._runScpiCmd('Sense:Function "Voltage:DC"')

        if str(integrationNPLC) not in self.NPLC:
            self._abort('invalid integrationNPLC: ' +
str(integrationNPLC) + ', use ' + str(self.NPLC))
        self._runScpiCmd('Sense:Voltage:DC:NPLC ' +
str(integrationNPLC))

        if range not in self.voltageRangeDC:
            self._abort('invalid range: ' + range + ', use ' +
str(self.voltageRangeDC))
        self._runScpiCmd('Sense:Voltage:DC:Range ' + range)

        if range == 'AUTO':
            self._runScpiCmd('Sense:Voltage:DC:Range:AUTO ON')
        else:
            self._runScpiCmd('Sense:Voltage:DC:Range:AUTO OFF')

        self._runScpiCmd('Trigger:Source Bus')
        self._runScpiCmd('Sample:Count MAX') # continiuous sampling,
max. 600 Mio points
        self._runScpiCmd('R?') # clear buffer
        self._runScpiCmd('Initiate') # arm the trigger
        self._runScpiCmd('*TRG')  # send trigger (samples exact one
value)

        return self.read()

    def getCurrentDC(self, range='AUTO', integrationNPLC='10'):
        self._runScpiCmd('abort') # stop active measurement
        self._runScpiCmd('Sense:Function "Current:DC"')

        if str(integrationNPLC) not in self.NPLC:
            self._abort('invalid integrationNPLC: ' +
str(integrationNPLC) + ', use ' + str(self.NPLC))
        self._runScpiCmd('Sense:Current:DC:NPLC ' +
str(integrationNPLC))

        if range not in self.currentRangeDC:
            self._abort('invalid range: ' + range + ', use ' +
```

```
str(self.currentRangeDC))
        self._runScpiCmd('Sense:Current:DC:Range ' + range)

        if range == 'AUTO':
            self._runScpiCmd('Sense:Current:DC:Range:AUTO ON')
        else:
            self._runScpiCmd('Sense:Current:DC:Range:AUTO OFF')

        self._runScpiCmd('Trigger:Source Bus')
        self._runScpiCmd('Sample:Count MAX') # continiuous sampling,
max. 600 Mio points
        self._runScpiCmd('R?') # clear buffer
        self._runScpiCmd('Initiate') # arm the trigger
        self._runScpiCmd('*TRG')  # send trigger (samples exact one
value)

        return self.read()

    # read()
    # get latest value, with current settings
    # saves a lot of time!
    # the DMM aquires in the meanwhile, and read fetches the most
recent value
    # getVoltageDC() or getCurrentDC() must be called before!
    # typical session with netcat:
    # >>r? 1
    # #215+1.36239593E+00
    #  ^ 2 = number of digits with describe the packet length
    def read(self):
        ans = '>>'
        t0 = time.time()
        while ans == '>>':
            if (time.time()-t0)>5:
                self._abort('timeout in read(), forgot to start the
measurement?')
            ans = self._runScpiQuery('R? 1') # get latest data
            if ans == '>>':
                time.sleep(0.1)
        ans = str(ans)
        nDigits = int(ans[1])
        ans = ans[(2+nDigits):]

        return self.str2engNumber(ans)

    def _runScpiCmd(self, cmd, timeout=_timeoutCmd):
        self._debug('_runScpiCmd(' + cmd + ')')

        BUFFER_SIZE = 1024
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        s.settimeout(timeout)
        data = ""
```

```
        try:
            s.connect((self.ipAddress, self._port))
            data = s.recv(BUFFER_SIZE)  # discard welcome message
            s.send(bytes(cmd + '\n', 'utf-8'))
            data = s.recv(BUFFER_SIZE)
        except socket.timeout:
            self._abort('timeout on ' + cmd)

        s.close()
        retStr = data.decode()
        self._debug('>>' + retStr)
        return retStr

    def _runScpiQuery(self, query, timeout=_timeoutQuery):
        self._debug('_runScpiQuery(' + query + ')')

        BUFFER_SIZE = 1024
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        s.settimeout(timeout)
        data = b""
        try:
            s.connect((self.ipAddress, self._port))
            data = s.recv(BUFFER_SIZE) # discard welcome message
            s.send(bytes(query + '\n', 'utf-8'))
            data = s.recv(BUFFER_SIZE)
        except socket.timeout:
            self._abort('timeout on ' + query)

        s.close()
        retStr = data.decode()
        retStr = retStr.strip('\r\n')
        return retStr


    def str2engNumber(self, str):
        x = decimal.Decimal(str)
        return x.normalize().to_eng_string()
```

Links

[1] https://www.eevblog.com/forum/testgear/siglent-sdm3055-multimeter-scpi-commands-and-python/
[2] https://www.eevblog.com/forum/testgear/siglent-sdm3055-multimeter-scpi-commands-and-python/msg719901/#msg719901
[3] https://www.batronix.com/shop/multimeter/Siglent-SDM3065X.html

article, english, electronics, technical, howto, programming, remote control, script, technical, python