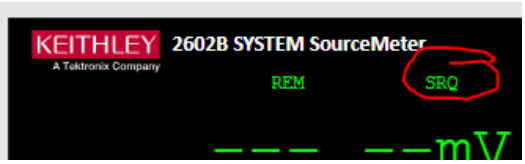
Detect End of a Sweep with 2600B Model

Chapter 12 in the 2600B Reference Manual details the status system of the instrument.

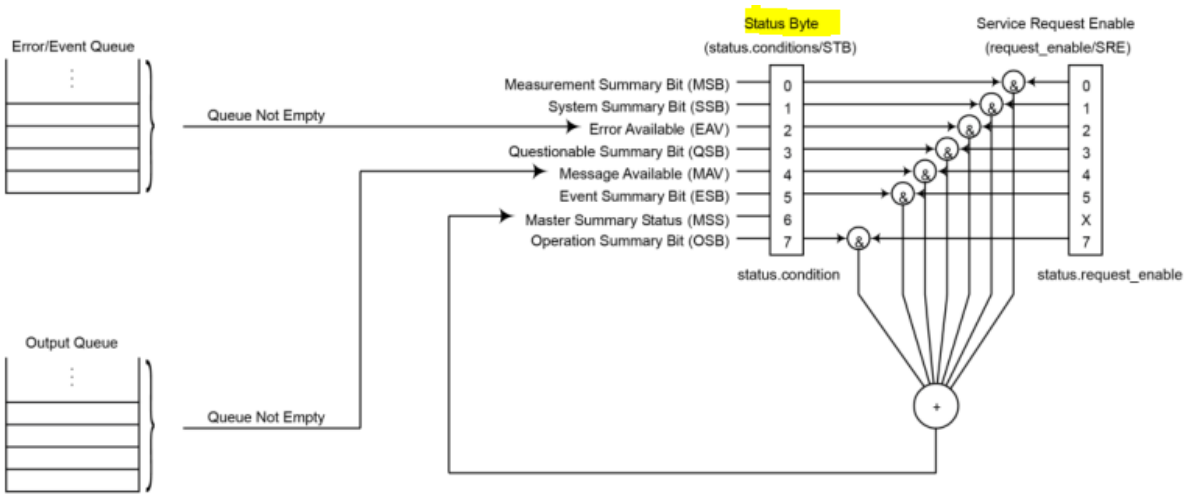
Here will we walk through using the status system to raise a Service Request when the SWEEPING bit changes from a 1 to 0 state which indicates the sweep just completed.



There are *many* other conditions that can be enabled.

When using VISA for instrument communication, there is a ReadSTB() command for reading the value of the status byte. The ReadSTB() will not suffer the same timeout errors that the VISA Read does if attempting to read *OPC? response during a long running sweep or other blocking operation.

The goal is to configure the status system so that the status byte will signal to your Python code that the event of interest has occurred.



Simple Events: Two of the bits (EAV and MAV) are directly influenced by error or output queues. If an error occurs or if the print() or printbuffer() are executed, the content placed in the queue will also cause the bit in the status register to transition from 0 to 1. In the corresponding request_enable register, your code can assign value of 1 to the bits that you'd like to monitor. The bit in condition register is logically ANDed with bit in enable register; and all the individual bits are ORed and control the MSS bit and SRQ state.

More Complex Events: To get access to information about SWEEPING, the Operation Summary bit (OSB) in status condition is fed by the multiple layers of the status.operation register sets.

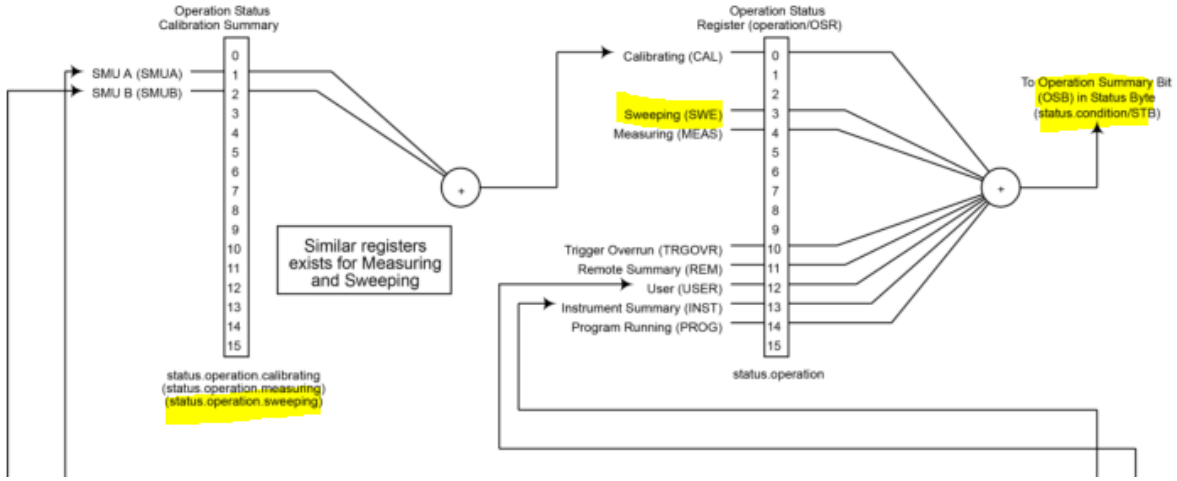
Below is Python code to configure instrument to tell us about SMUA completing a sweep:

```
#commands to configure SRQ when SWEEPING bit goes 1 to 0
# see Chapter 12 in 2600B Reference Manual
cmd_list2 = ["status.reset()",
             "status.operation.enable = status.operation.SWEEPING",
             "status.operation.sweeping.enable = status.operation.sweeping.SMUA",
             "status.operation.sweeping.ptr = 0",
             "status.operation.sweeping.ntr = status.operation.sweeping.SMUA",
             "status.request_enable = status.OSB"]

for cmd in cmd_list2:
    my_instr.write(cmd)
```

Illustrating the layers of the registers:

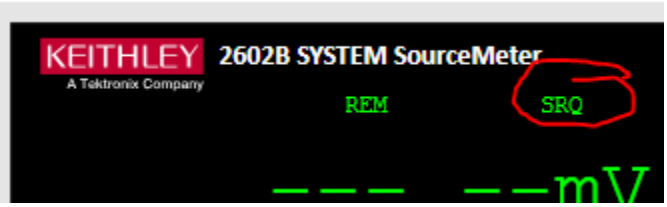
Figure 140: Operation status registers



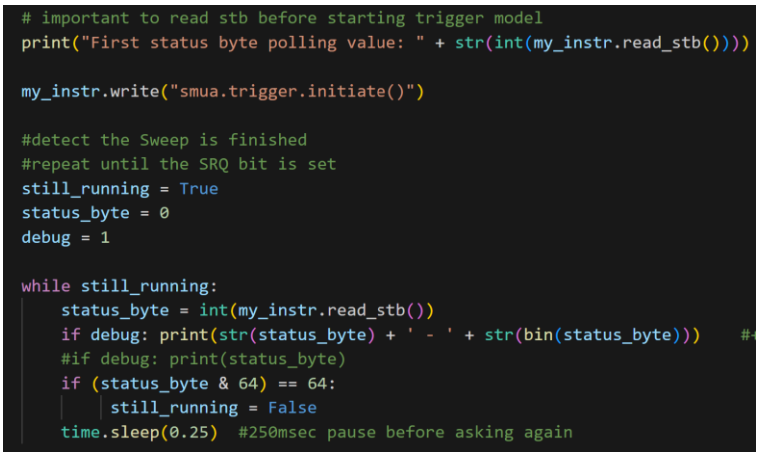
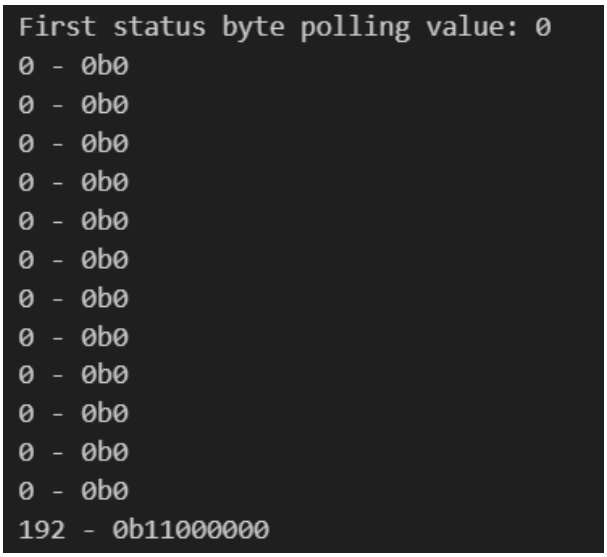
A condition register has a corresponding enable register. Write values to the enable register to cause the event (bit) of interest to be promoted up the register hierarchy.

Additionally, we are using the negative transition register to tell us about the 1 to 0 change (completion of sweep) and ignoring the 0 to 1 change (start of sweep).

The status.request_enable = status.OSB places a 1 in bit7 of the enable register. When the lower level SWEEPING bit change occurs, the OSB bit will go 0 to 1. This will be logical AND with the enabled register and cause the SRQ to occur.



Status Polling and Program output

Description	Code Samples
Python polling loop for detection of SRQ MSS = bit 6 = 2^6 = 64	
Program Output: Initially at zero MSS = bit 6 = 2^6 = 64 OSB = bit 7 = 2^7 = 128 Final Value of 192 = OSB and MSS at logic 1	

```
# important to read stb before starting trigger model
print("First status byte polling value: " + str(int(my_instr.read_stb())))

my_instr.write("smua.trigger.initiate()")

#detect the Sweep is finished
#repeat until the SRQ bit is set
still_running = True
status_byte = 0
debug = 1

while still_running:
    status_byte = int(my_instr.read_stb())
    if debug: print(str(status_byte) + ' - ' + str(bin(status_byte)))
    #if debug: print(status_byte)
    if (status_byte & 64) == 64:
        still_running = False
    time.sleep(0.25)  #250msec pause before asking again
```

```
First status byte polling value: 0
0 - 0b0
0 - 0b0
0 - 0b0
0 - 0b0
0 - 0b0
0 - 0b0
0 - 0b0
0 - 0b0
0 - 0b0
0 - 0b0
0 - 0b0
0 - 0b0
192 - 0b11000000
```

Full Code Listing

Assumptions: use VISA to get a session (my_instr) with 2600B series product

```
# set commands to setup a trigger model linear sweep
num_sweep_pts = 151
cmd_list = ["errorqueue.clear()",
            "smua.source.func = smua.OUTPUT_DCVOLTS",
            "smua.source.limiti = 100e-3",
            "smua.measure.nplc = 1",
            "smua.measure.delay = smua.DELAY_AUTO",
            "smua.measure.delayfactor = 1",
            "smua.measure.lowrangei = 100e-9",
            "smua.nvbuffer1.clear()",
            "smua.nvbuffer1.collecttimestamps = 1",
            "smua.nvbuffer2.clear()",
            "smua.nvbuffer2.collecttimestamps = 1",
            "smua.trigger.source.linearv(-0.5, 0.5, " + str(num_sweep_pts) + ")",
            "smua.trigger.source.limiti = 0.1",
            "smua.trigger.measure.action = smua.ENABLE",
            "smua.trigger.measure.iv(smua.nvbuffer1, smua.nvbuffer2)",
            "smua.trigger.endpulse.action = smua.SOURCE_HOLD",
            "smua.trigger.endsweep.action = smua.SOURCE_IDLE",
            "smua.trigger.count = " + str(num_sweep_pts),
            "smua.trigger.source.action = smua.ENABLE"]

for cmd in cmd_list:
    my_instr.write(cmd)
```

```
#commands to configure SRQ when SWEEPING bit goes 1 to 0
# see Chapter 12 in 2600B Reference Manual
cmd_list2 = ["status.reset()",
             "status.operation.enable = status.operation.SWEEPING",
             "status.operation.sweeping.enable = status.operation.sweeping.SMUA",
             "status.operation.sweeping.ptr = 0",
             "status.operation.sweeping.ntr = status.operation.sweeping.SMUA",
             "status.request_enable = status.OSB"]

for cmd in cmd_list2:
    my_instr.write(cmd)
```

```
#turn output on and run the sweep
my_instr.write("smua.source.output = smua.OUTPUT_ON")
```

```
t1 = time.perf_counter()

# important to read stb before starting trigger model
print("First status byte polling value: " + str(int(my_instr.read_stb())))

my_instr.write("smua.trigger.initiate()")

#detect the Sweep is finished
#repeat until the SRQ bit is set
still_running = True
status_byte = 0
debug = 1

while still_running:
    status_byte = int(my_instr.read_stb())
    if debug: print(str(status_byte) + ' - ' + str(bin(status_byte)))
    if (status_byte & 64) == 64:
         still_running = False
    time.sleep(0.25)  #250msec pause before asking again


#turn output off
my_instr.write("smua.source.output = smua.OUTPUT_OFF")

t2 = time.perf_counter()
print('Test time: {}'.format(t2 - t1))
print("Number of actual smua buffer pts: " +
str(my_instr.query('print(smua.nvbuffer1.n)')))

print("Polling loop done, status byte: " + str(int(my_instr.read_stb())))

print('Go get the data.')
```

Retrieve the data

```
import numpy as np
import matplotlib.pyplot as plt

#ask for voltage and current; buffer2=voltage
my_instr.write("printbuffer(1, smua.nvbuffer1.n,
smua.nvbuffer2.readings,smua.nvbuffer1.readings)")
raw_data = my_instr.read()            #one long comma delimited string
# split yields an array of strings alternating voltage, current, voltage, current
raw_data_array = raw_data.split(",")
```

```
volts = []
current = []
abs_current = []

# step through the array of strings, step size 2
# place the V and I into their own array of floats
for i in range(0, len(raw_data_array),2):
    volts.append(float(raw_data_array[i]))
    current.append(float(raw_data_array[i+1]))
    abs_current.append(abs(float(raw_data_array[i+1])))   #absolute value of
currents for log scale

#plot the absolute value of current vs. voltage
plt.plot(volts, abs_current, 'o-g')

x_min = min(volts)
x_max = max(volts) * 1.1
y_min = abs(min(abs_current)) / 10
y_max = abs(max(abs_current)) * 10
plt.axis([x_min,x_max,y_min,y_max])
plt.yscale('log')
plt.xscale('linear')
plt.title('Sweep V, Measure I, 100KOhm')
plt.margins(x=0.1, y=0.1)
plt.grid(True)

plt.show()
```

Output: