Scan control and data acquisition with Python

G. MOLDOVAN, M. ZABEL AND M. JOACHIMI
point electronic GmbH, Halle (Saale), Germany

BIOGRAPHY
Dr. Grigore (Greg) Moldovan is the Chief Technology Officer at point electronic GmbH – a supplier of electronics and software for electron microscopy. He manages technology, development and marketing. Greg has previously worked for Oxford Instruments Nano-Analysis, University of Oxford, and University of Cambridge; holds a PhD from University of Nottingham and has a Diploma in Physics. He is a scientist and a microscopist.

ABSTRACT
Development of new techniques for electron microscopy often requires advances in scan control and image acquisition. A need for development may arise because of new scan patterns, new synchronisation with detectors, or simply because of high-performance resolution and speed. This article aims to provide a guide on current state of technology, with practical examples on how this control is coded from Python. It is shown here that the latest technology can already provide deep and open access to the hardware over application programmable interfaces, and that image and point map scan patterns are easy to program.

CORRESPONDING AUTHOR
Erich-Neuss-Weg 15
D-06120 Halle (Saale)
Germany
Mobile: +49 157 58076085
Email: gm@pointelectronic.de

FIGURE 1 *Photo of DISS6 scan generator hardware. To assist software development, display on front panel shows parameters of running scan job.*

INTRODUCTION

For much of electron microscopy, scanning the electron beam on the sample has been fixed to the same pattern as that of scanning the electron beam inside the cathode ray tube (CRT), the display technology during the early days of microscopy. This used to provide a simple foundation for understanding magnification, as the scan size of the electron beam on the screen was fixed to the CRT physical size. This is no longer true, images can be displayed at any size, and magnification as was originally conceived does not make sense. But more importantly, whilst modern displays have inherited the CRT raster pattern from top-left to bottom-right, scanning of the electron beam on the sample can now be set free from this conventional pattern – far more is possible now, as presented here.

It should not be preconceived that scan patterns are limited the internal scan generator of the microscope. Indeed, these have a varying degree of complexity and performance, reflecting what the manufacturer has taken as requirements at the time. There are a number of practical considerations that limit specifications and access to this internal scan generator, not only cost and service, but also compatibility with other parts of the microscope hardware and software. However, this means not that a microscope is not limited by its internal scan generator, because it may be easily bypassed with the addition of an external scan generator to the external scan interface of the microscope. This gives safe access to the internal scan amplifier, therefore enabling the electron beam to follow any external scan pattern, however complex. Typical microscope limits are given therefore by the bandwidth of its internal scan amplifier and the hysteresis of its scan coils.

This architecture is a great asset in electron microscopy, not only because it provides open means to bring new features and value to microscopes, but also because it allows for different systems to be developed and controlled independently, not only with regards to the scan generator, but also the corresponding computer, operating system and software. We will make use of this architecture here – add an external scan generator with an open control library and use Python code to set scan parameters, map points and synchronisation signals.

HARDWARE

The external scan generator used in this work is the sixth generation of digital imaging scanning system (DISS6) by point electronic GmbH, see Figure 1. It provides an analog interface to the microscope external scan interface, as well as a digital interface for synchronisation and signal inputs for detectors. Key interfaces will be described briefly below in order to provide the basics necessary to understand the software commands

Microscopy and Analysis 36(5): 13-15 (EU), October 2022

used in the code examples.

EXTERNAL SCAN INTERFACE

Whilst there is no standard for the external scan interface in electron microscopy, it is available on most microscopes as a connector on a panel or inside the electronics rack. Some microscopes provide several such connectors, as it's common to have multiple external scan generators.

Microscopes generally also provide means to switch between scan generators, either manual switches requiring user intervention to hardware or software, or automatic switches linked to a hardware control signal included with the interface, or a switch command to the microscope control software. It is also possible to daisy chain multiple external scan generators on a single external scan interface, and then scan switching is independent from the microscope control.

Key signals on the external scan interface are X and Y voltage inputs into the microscope, which are used by the scan amplifier to provide the necessary currents to the scan coils and therefore place the electron beam at the corresponding XY position. Rotation and shift can be done to a limited extend by the scan generator, however these are generally done by the scan amplifier.

Note that the final XY position of the electron beam on the sample is not only a relation to these X and Y voltage signals, but also depends on the scan amplifier, the scan coils, and of course on the electron optics settings, such as magnification or alignment.

SYNCHRONISATION INTERFACE

Synchronisation with the beam scan is required for *in situ* experiments, imaging cameras or advanced scanning schemes. A hardware synchronisation interface is needed for the high speeds typical of scan generators, which can be down to 20 ns acquisition time per image pixel/ map point. Therefore, synchronisation is provided as Transistor-Transistor Logic (TTL) compatible signals on a digital input/output interface of the external scan generator. Software synchronisation is of course possible at the level of Python code as well – however it's inefficient at any speed, and simply at not possible at such short acquisition times.

The scan generator may be configured to output a pixel clock when the beam is stationary at each pixel/ point in the scan, and to wait for pixel synchronisation input at each pixel/ point in the scan pattern. Note that pixel clock output depends on the scan pattern and the corresponding acquisition times, which may not be constant, and therefore pixel clock may not be a regular signal.

Each scan job may be additionally set up to output a frame clock at the beginning of each scan, and to wait for an external signal synchronisation before starting a scan. This is useful to signal the beginning of an acquisition or an *in situ* experiment. For the case of image scans only, the scan job may be configured to output a line clock at the beginning of each line, and to wait for a line synchronisation input before starting a new line. This is generally used to prevent fixed pattern noise in images at the same frequency as the mains power.

Note that a scan generator with complete frame, line and pixel synchronisations does allow for master and slave configurations, though this is not covered here.

SOFTWARE

A software stack is used in order to bring DISS6 scan control and image acquisition into Python, over the available USB or ETH control interfaces. A USB connection is used, then the USB driver must be installed.

A driver is not necessary an ETH connection is used, but additional network IP configuration is required. The examples presented here use a USB connection to a Microsoft Windows PC. For clarity, variable declarations and error handling are not included in the code examples presented here.

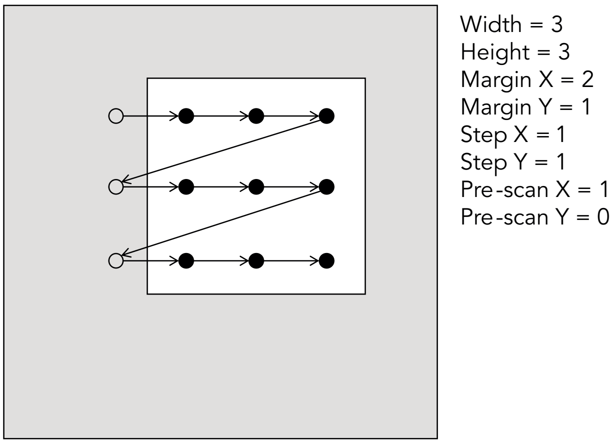

FIGURE 2 *Schematic diagram of image scan geometry. Scan pattern is predetermined and generated by the scan generator. Software provides scan parameters.*

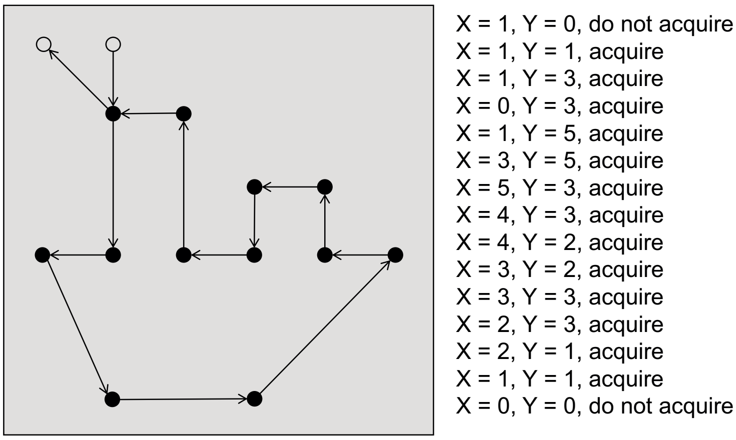

FIGURE 3 *Schematic diagram of pixel map scan. Scan generator runs through a list of points. Scan pattern is generated by the control software.*

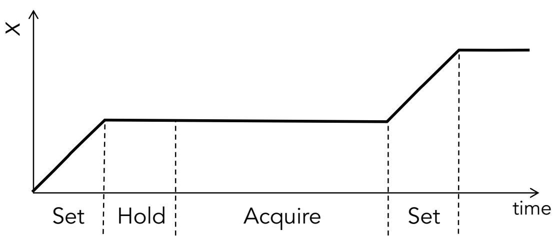

FIGURE 4 *Schematic diagram of set, hold and acquisition times. Time is displayed on horizontal axis, and horizontal scan direction is displayed on vertical axis.*

CONNECTING TO SCAN GENERATOR

A DISS6Control library is delivered as part of the system to encapsulate and abstract low-level control of the hardware. This means that in order to program complex patters, it is sufficient to familiarise with the Application Programmable Interface (API) of the control library, without a detailed knowledge of the internal architecture, functions and addresses. This architecture has been intentionally developed to reduce complexity of software development.

For example, establishing a remote connection to the device over USB is reduced to adding DLLs from current working directory, loading the 32- or 64-bit library and calling the USB initialisation function (Figure 5a).

SETTING UP AN IMAGE SCAN

This is the conventional top-left to bottom-right scan pattern, with regular spacings between pixels and constant acquisition time, as illustrated in Figure 2. Whilst this image geometry may appear simple, defined primarily by pixel resolution, more complex settings are added to enable region of interest (ROI) scans. Image scan settings therefore include margins from top and left, horizontal and vertical steps. Further adding to this, an image scan must define not only these pixels for acquisition, but also the pre-scan pixels used to manage scan distortions originating from hysteresis of the scanning coils – these are excluded from the acquired data.

Scan times are defined in part by pixel acquisition time, and in part by management of scan coil hysteresis, which again can result in complex settings. A second approach to minimise scan distortion is to add delay times at the beginning of each line and frame, which should be proportional to the return distances, as well as the scan speed. Beam return time can also be managed, including number of steps on the return path and the corresponding times on line return and frame return steps.

In code, a new image scan job is first created for the video channels needed for data acquisition, then geometry is specified. Example code below is for a one 512 x 512 scan over the entire scan space without pre-scan, 1 μs pixel acquisition time, with 20 μs line start delay, 512 return steps, and 10 ns/step for both line and frame (Figure 5b). Note that a scan job ID must be given and used throughout, and that a scan step of 128 is required to cover the entire scan space of 16-bit. Also note DISS6 constants for ease of use.

SETTING UP A POINT MAP

As illustrated in Figure 3, a much more general approach to scanning is for the scan generator to use a list of points with X and Y coordinates and acquisition times, rather than generate the scan points on the fly in a particular pattern. When used in this mode, the software takes over the role of generating the scan pattern, with the hardware keeping the role of generating the corresponding voltages for the scan amplifier. This approach must therefore be used with care, as no provisions are made in the scan generator for managing hysteresis, which now must be managed entirely in the software.

The code example (Figure 5c), creation of scan job, start of scan and retrieval of data remains the same. In order to reduce load time, software must bundle all information for a point map in a set of two 32-bit binary arrays. This binary includes point coordinates and acquisition time, as well as flags for acquisition and synchronisation.

SETTING UP SYNCHRONISATION

Setting triggers and synchronisation for a scan job is the same for image scan and point map – three values must be given, for pixel, line and frame. Clock lengths are configurable to match external equipment.

To account for time needed by the scan coils to move, and by the external equipment to reset, two additional settle and hold time can be set, in addition to the acquisition time. Pixel dwell time is given by the sum of hold and acquisition times, as illustrated in Figure 4.

In code we see there is an example where the scan generator initiates a scan by outputting a pixel trigger configured with 10 ns settling time to move the beam and 1 μs hold time to wait for external hardware after sync input (Figure 5d).

STARTING SCAN AND RETRIEVING DATA

The remaining work is to start the scan job, retrieve the data in chunks until end of frame, and make the image from the pixel buffer., see example code in Figure 5c.

Note that a frame buffer array must contain corresponding addresses of the frame buffers for each channel selected – this example shows one channel only. NumPy and Image packages are used here.

SUMMARY AND CONCLUSIONS

The current architecture of most electron microscopes is open with regards to beam scanning and data acquisition, as it includes an external scan interface to the internal scan amplifier. This can be easily used to add an external scan generator with new functions and advanced control, with DISS6 given here as an example. Programming such equipment has become less complex over time, with the simple Python examples given here illustrating that much can be achieved with only a few code lines. DISS6 point map scan mode moves the task of scan pattern generation to the software, thus enabling an increased degree of openness towards new developments. DISS6 digital synchronisation and triggering also allows for complex synchronisation schemes with external equipment.

Article, and references available online at: analyticalscience.wiley.com/publication/microscopy-and-analysis

A
```
os.add_dll_directory(os.getcwd())
diss6control = cdll.loadlibrary("diss6control64.dll")
rc = diss6control.InitUSB(none)
```

B
```
rc = diss6control.CreateImageScanjob(len(channels),
                                     byref(channels),
                                     byref(hScanJob))
rc = diss6control.SetImageGeometry(hScanJob, 512, 512, 0, 0, 128, 128, 0, 0)
rc = diss6control.SetAcquisitionTime(hScanJob, 1000)
rc = diss6control.SetLineStartDelay(hScanJob, DISS6.TIME_SCALE_20US)
rc = diss6control.SetBeamReturnTiming(hScanJob,
                                     512,
                                     DISS6.TIME_SCALE_10NS,
                                     DISS6.TIME_SCALE_10NS)
rc = diss6control.SetFrameCount(hScanJob, 1)
```

C
```
rc = diss6control.CreatePixelMapScanJob(len(channels),
                                     byref(channels),
                                     byref(hScanJob))
for y in range(512):
    for x in range(512):
        pixels[x+y*512]=Pixel_t(PixelComponent_t(x*128, 0b1000010000000000),
                   PixelComponent_t(y*128, 0))
rc = diss6control.SetPixelMapValues(hScanJob, len(pixels), byref(pixels))
rc = diss6control.SetFrameCount(hScanJob, 1)
```

D
```
rc = diss6control.SetHardwareTriggerEnabled(hScanJob, 1, 0, 0)
rc = diss6control.SetPixelClockLength(DISS6.TIME_SCALE_10NS)
rc = diss6control.SetSettlingTime(hScanJob, DISS6.TIME_SCALE_10NS)
rc = diss6control.SetHoldingTime(hScanJob, DISS6.TIME_SCALE_1US)
```

E
```
rc = diss6control.StartScanJob(hScanJob, DISS6.ABORT_SCAN_IMMEDIATELY)
while 1:
  rc = diss6control.ReadChannelData(hScanJob,
                                     byref(FrameBufferArray),
                                     byref(pixelCount),
                                     DISS6.READ_FLAG_USE_PIXEL_OFFSET,
                                     None,
                                     byref(pixelOffset),
                                     byref(status))
  if (status.value & DISS6.READ_STATUS_FRAME_END):
    break
data = numpy.frombuffer(frameBufferA1, dtype=numpy.uint16)
data.shape = (512,512,1)
im = Image.fromarray(data,mode='I;16')
```

FIGURE 5 *Example code*