Fast Frame Access

Overview

- Leading-edge applications in quantum physics require fast access to frame data.
- The Teledyne Princeton Instruments PICam API offers direct control of our cameras, including fast access to live data via event callbacks.
- In-house tests demonstrate that full parsing of 1024x1024 frame data on the ProEM®-HS:1024BX3 camera (GigE interface) can be achieved in as little as 2 ms after readout ends.

Motivation:

Quantum research is at the cutting edge of modern science and is a field where the world's brightest and most driven researchers are continually coming closer to creating technology based on the application of quantum principles that will revolutionize information (and life) as we know it.

Leading-edge quantum applications, such as trapped ions and cold atoms, first and foremost require an imager that is up to the task of dealing with low-light and short-lived events. Teledyne Princeton Instruments' EMCCD and *em*ICCD cameras are built for such experiments, and our longstanding reputation for innovation and quality speaks for itself.

In addition to an extraordinary imager, these cutting-edge applications also require fast access to frame data so that real-time decisions can be made (e.g., whether or not to trigger a light source). Scientists may need to know the number of electron events at that time, or perhaps they want to know where the centroid is at that moment. For these extraordinarily precise applications, Teledyne Princeton Instruments offers our PICam API, which can be used to directly automate the camera using the C run-time, thus eliminating overhead that can occur when communicating through other development environments (e.g., National Instruments LabVIEW® or MathWorks MATLAB®).

Fast Frame-Data Access in PICam

The underlying mechanism in achieving fast access to frame data in PICam is an event callback. PICam allows one to register an event listener that runs asynchronously and alerts the system when there is any new frame data updated to the circular buffer. A user can then write a callback function that tells the system exactly what to do each time such data is received.

The PICam API command to register this callback is

```
PicamAdvanced_RegisterForAcquisitionUpdated(device_,AcquisitionUpdated);
```

The header of the callback function would be

```
PicamError PIL_CALL AcquisitionUpdated(
        PicamHandle device,
        const PicamAvailableData* available,
        const PicamAcquisitionStatus* status)
```

When trying to process data in real-time, it is of paramount importance that the delays in the system be understood. By using the chrono library in C++, we have been able to calculate timing on a ProEM-HS:1024BX3 camera, keeping in mind an application that would require the knowledge of all 1,048,576 (1024x1024) pixels as fast as possible.

1. The jitter from the end of readout to the image data becoming available to process using the event callback.
2. The time it takes to read all pixel data out into an array that can be manipulated by the user.

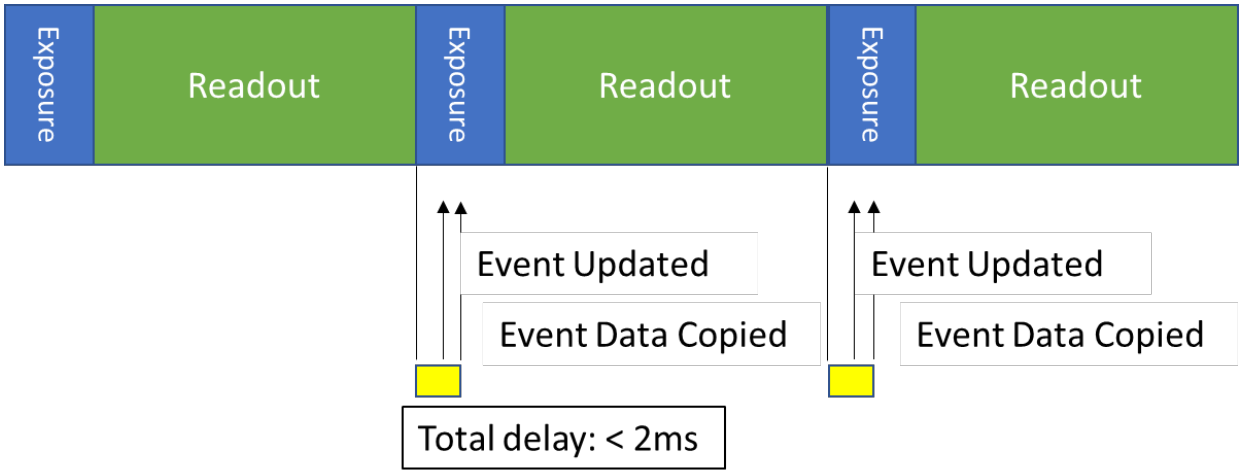

Figure 1. Example of a three-frame exposure + readout cycle, where an event callback is executed after each readout. Through use of the C++ chrono library, we have calculated the total delay (which includes jitter from readout end to image buffer becoming available, as well as time needed to copy each byte into array) to be just under 2 ms.

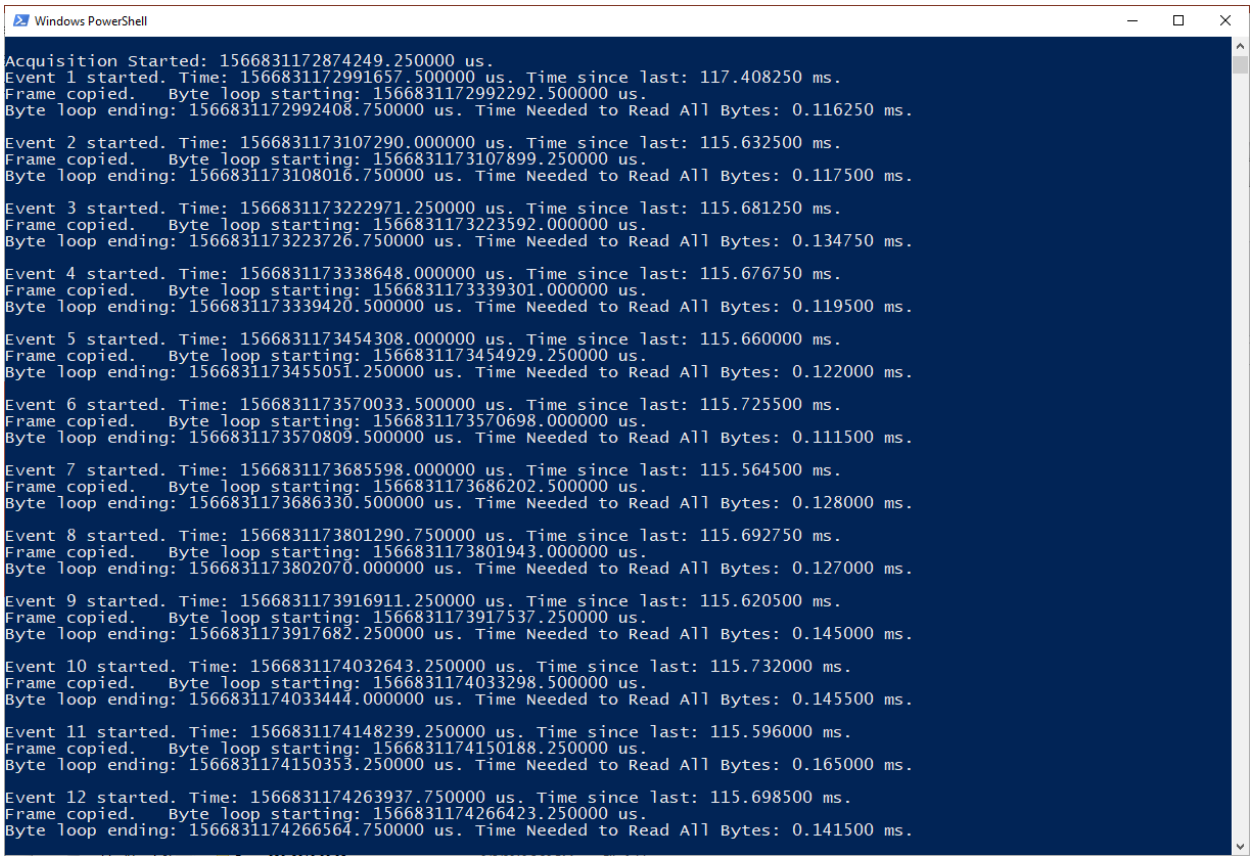

Figure 2. Timing output from a live-frame capture performed on a ProEM-HS:1024BX3 using an experiment with 0 ms exposure and 115.6 ms readout time. The jitter from readout end to event callback availability can be seen in the first two lines. System time was taken just after the asynchronous acquisition command was issued, and a subsequent stamp was taken as soon as the first event callback was run. This jitter was found to be 1.8 ms: 117.4 ms - 115.6 ms. Then, the time to grab the data from each pixel (using a simple naïve for loop) was consistently under 0.2 ms. The event-to-event timings from event #1 onward show a consistent 115.6 – 115.7 ms timing, which corresponds well to the expected readout rate.

Please refer to Figure 1 for a simple diagram of the timing described above. Figure 2 shows the output from a PICam script written to extract the timing delays that are discussed here.

Here, we show that the jitter from readout end to event callback beginning is 1.8 ms, the time taken to copy all 1,048,576 pixels (using a simple naïve for loop, see below) is consistently <0.2 ms, and the subsequent event-to-event timing (which is expected to be identical to the readout-to-readout timing in an experiment with 0 ms exposure) is consistently in the 115.6 ms range — the same 115.6 ms that is the reported readout rate of the ProEM-HS:1024BX3 used in this experiment with Frame Transfer readout, full-frame ROI, and 10 MHz ADC rate. When all these factors are considered, one can determine a total delay of **<2 ms*** with high confidence.

```
for (piint loop = 0; loop < framelength; loop += 2)
{
        frameptr = buf + (loop);
        pixVal = (pi16u*)frameptr;
}
```

Note: For loop used to loop through each pixel value (2 bytes) and copy value into an object the user can manipulate while the camera is capturing data.

You are not alone when programming!

One should never feel like they are alone and need to re-invent the wheel when using the PICam API.

Teledyne Princeton Instruments provides a vast library of documentation and sample code to help any user get started with communicating with our cameras via PICam. Even for advanced functions such as event callbacks, we have sample code that the user can follow in order to successfully register an event, set up a callback, and grab the frame data.

Should you get stuck even after following the sample code and cross-referencing functions with the Programmers Manual, you can drop an email to our support box at PI.Techsupport@teledyne.com and we will get back to you very quickly.

Our support staff includes Ph.D.-level engineers who are well versed in Teledyne Princeton Instruments cameras and control. Most important, we share an enthusiasm for the cutting-edge applications of our customer base across the world.

* As Microsoft® Windows® is non-deterministic with respect to thread timing, we can safely ensure that the results shown in this study would hold true on average over a large number of events, but it is possible that there will be some outlier events where the delays will be slightly longer.