

Firmware lower-level discrimination and compression applied to streaming x-ray photon correlation spectroscopy area-detector data

T. Madden,^{1, a)} P. Fernandez,¹ P. Jemian,¹ S. Narayanan,¹ A. R. Sandy,¹ M. Sikorski,¹ M. Sprung,^{1, b)} and J. Weizeorwick¹

*X-Ray Science Division, Argonne National Laboratory,
9700 S. Cass Ave., Argonne, IL 60439 USA*

(Dated: 23 April 2010)

We present a data acquisition system to perform on-the-fly background subtraction and lower-level discrimination compression of streaming x-ray photon correlation spectroscopy (XPCS) data from a fast charge-coupled device (CCD) area detector. The system is built using a commercial frame grabber with a built-in field-programmable gate array (FPGA). The system is capable of continuously processing 60 CCD frames per second each consisting of $1,024 \times 1,024$ pixels with up to xx photon hits per frame.

PACS numbers: Valid PACS appear here

Keywords: x-ray photon correlation spectroscopy (XPCS), x-ray intensity fluctuation spectroscopy (XIFS), field-programmable gate array (FPGA)

I. INTRODUCTION

X-ray photon correlation spectroscopy (XPCS) performed with hard x-rays ($E \gtrsim 7$ keV, where E is the x-ray energy) has emerged as a powerful technique for characterizing the equilibrium or steady-state dynamics of condensed matter on length scales shorter than can be achieved with optical techniques and on longer time scales than can be achieved via neutron scattering. Even on optically accessible length scales, opaque and metallic samples are readily studied, providing new opportunities for studies of colloidal and other soft matter systems. Several recent review articles^{1, 2, 3} provide summaries of the scientific impact of the technique and the mechanics of performing such experiments.

A key factor enabling XPCS's recent significance has been the application of direct-detection area detectors to problems of interest—so-called multispeckle XPCS—because even at a third generation synchrotron source like the Advanced Photon Source (APS), XPCS experiments are brilliance limited. Area detectors allow x-ray speckle data to be simultaneously collected at equivalent wave-vector transfers (Q 's) permitting averaging of the auto-correlation functions thereby improving the signal-to-noise ratio (SNR) in an experiment. Simultaneously, area detectors collect data over a span of different wave-vectors greatly increasing the efficiency of such experiments. Despite recent successes, however, the use of such detectors remains challenging both because of the limited availability of suitable high-speed detectors and because of the need to gather, manage and reduce data at increasingly higher frame rates over increasingly long measurement durations. This paper addresses the second issue, namely development of a system to compress x-ray-speckle data on-the-fly so that XPCS-suitable detectors

can be run as fast as possible for as long as possible in an operationally-sensible manner.

As a specific example, we consider recent multi-speckle XPCS measurements that were performed at our beamline (8-ID) at the Advanced Photon Source (APS). Over a very narrow temperature range, a dense colloidal suspension in a binary mixture was found to display novel repulsive- and attractive-glass behavior with an unusual transition between these two phases⁴. The complex phase behavior was unraveled via a combination of small-angle x-ray scattering (SAXS) and XPCS measurements. The dynamics (XPCS) measurements required collecting instantaneous speckle patterns at 60 frames per second (fps) over collection periods extending to 1,000 seconds. Intensity-intensity time correlations, calculated according to

$$g_2(Q, \Delta t) = \frac{\langle I(Q, t)I(Q, t + \Delta t) \rangle_t}{\langle I(Q, t) \rangle_t^2},$$

where Δt is the delay time and $I(Q, t)$ is the intensity at wave-vector Q and time t , revealed a distinct 2-step decay of the correlation functions in the repulsive-glass phase followed by a fully arrested correlation function in the attractive-glass phase. In between these 2 phases, the time autocorrelation functions exhibited a very unusual logarithmic intensity decay. The unusual dynamic properties of this system, spanning nearly 5 decades in delay time, could not have been discovered and measured without an area detector running continuously at high frame rates over extended collection periods.

Despite the evident scientific need, the measurements described above are not operationally-sustainable without the developments described below. The detector used for the above measurements⁵ has $1,024 \times 1,024$ $14\text{-}\mu\text{m}$ -square pixels and outputs 2 bytes of data per pixel. Without compression, a single time sequence as described above would require more than 120 gigabytes (GB) of storage. Since, in the example above, 100's of time sequences were required to establish and confirm

^{a)}Electronic mail: tmadden@aps.anl.gov

^{b)}Current address: Petra-III, DESY, Hamburg, Germany

the phase diagram of the system, disk space and data-reduction bandwidth would be rapidly exhausted. A key realization is that i) we use direct-detection CCD's for XPCS so the signal above the background (dark noise) is relatively high meaning individual photons can be distinguished and ii) the scattered signal is weak so the number of recorded photons per frame is relatively small. Thus, each frame can be compressed by a significant amount—typically xx%.

During the first iteration of software development for the camera⁷, the frames were accumulated in computer memory and then compressed and written to disk when the memory was full. This allowed the camera to run at full speed (60 fps) but only accumulate $\approx 1,000$ frames or 3 decades in delay times. A second iteration of high-level software (C++) development performed compression and wrote the data to disk on-the-fly. But even with a relatively powerful workstation computer dedicated to this task, on-the-fly compression performed with this software was unable to process 60 fps so the short-time dynamic range of the detector was limited. As such, we were led to consider firmware thresholding and compression of rapidly streaming multi-speckle XPCS data.

The remainder of this paper describes the design, implementation and performance of this system. We used a field-programmable gate array (FPGA) hosted on a commercial frame grabber to realize live lower-level discrimination (LLD) and compression of multi-speckle XPCS data. The output of the system is a series of highly-compressed data frames each consisting of a listing of intensities above a user-determined threshold and their locations on the CCD sensor. Somewhat analogous work has been described recently⁷ but though the frame rate was significantly higher than for our detector, the mean number of events per frame was much smaller. Reference⁷ discusses direct-detection CCD's, dark noise and photon identification in the context of multi-speckle XPCS measurements.

II. SYSTEM DESIGN

A schematic overview of our multi-speckle real-time compression system is shown in Fig. 1. It consists of a fast area detector⁷, a signal translator (not shown) that converts the low-voltage differential signaling (LVDS) signal of the detector to Camera Link, a frame grabber and a host workstation with fast local storage. Aside from the detector, the frame grabber is the key component in the system. It is a Dalsa Anaconda PCI-X frame grabber with an on-board Xilinx Virtex-Pro XC2VP20 FPGA. Solid arrows indicate data flow. The arrow heads indicate the direction of data flow and the width of the arrow is proportional to the data rate. Dashed arrows indicate the control interface for the system which has been developed under the EPICS areaDetector⁷ framework.

The key step in the system that reduces the data flow is compression which is accomplished via implementation

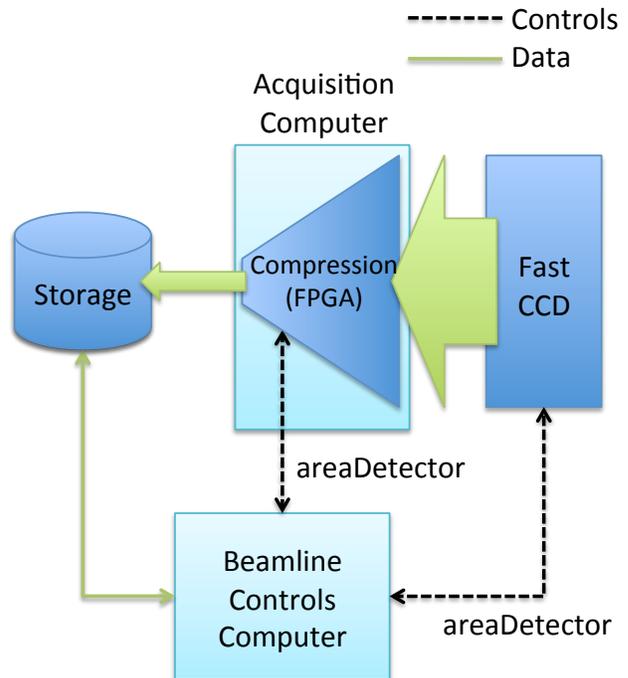


FIG. 1. Schematic representation of the data flow from detector to disk.

of an appropriate LLD⁷ in the FPGA. Briefly, as each frame is streamed through the FPGA, pixels are retained if they equal or exceed a defined LLD value and discarded otherwise. The LLD is specified on a pixel-by-pixel basis according to the following equation:

$$\text{THRESHOLD}(i, j) = C + DK_{\text{AVE}}(i, j) + \alpha DK_{\text{RMS}}(i, j), \quad (1)$$

where (i, j) are pixel indices, C is a pixel-independent constant threshold or offset value, DK_{AVE} is the pixel-dependent value of the average dark signal, and DK_{RMS} is the pixel-dependent value of the root-mean-square (rms) of the dark signal multiplied by a scaling factor β . In practice, the average dark value is first subtracted from all incoming data frames and then the dark-subtracted frames are compared to the LLD defined in Eqn. 1 with $DK_{\text{AVE}}(i, j) = 0$.

The LLD array can be created and updated on-the-fly in the FPGA on an as-needed basis. Figure 2 illustrates the procedure and the subsequent data compression. If only a constant LLD is required, then a LLD value specified by the user on a control screen is loaded into the FPGA. If, as is more typically the case, the LLD includes dark subtraction and a linear combination of a constant and the rms dark frame then the procedure is as follows.

1. The user specifies how many dark frames, $N_{\text{DARK}}^{\text{AVG}}$, should be collected to determine the average dark signal. $N_{\text{DARK}}^{\text{AVG}}$ dark frames are accumulated and recursively averaged and then stored on disk.

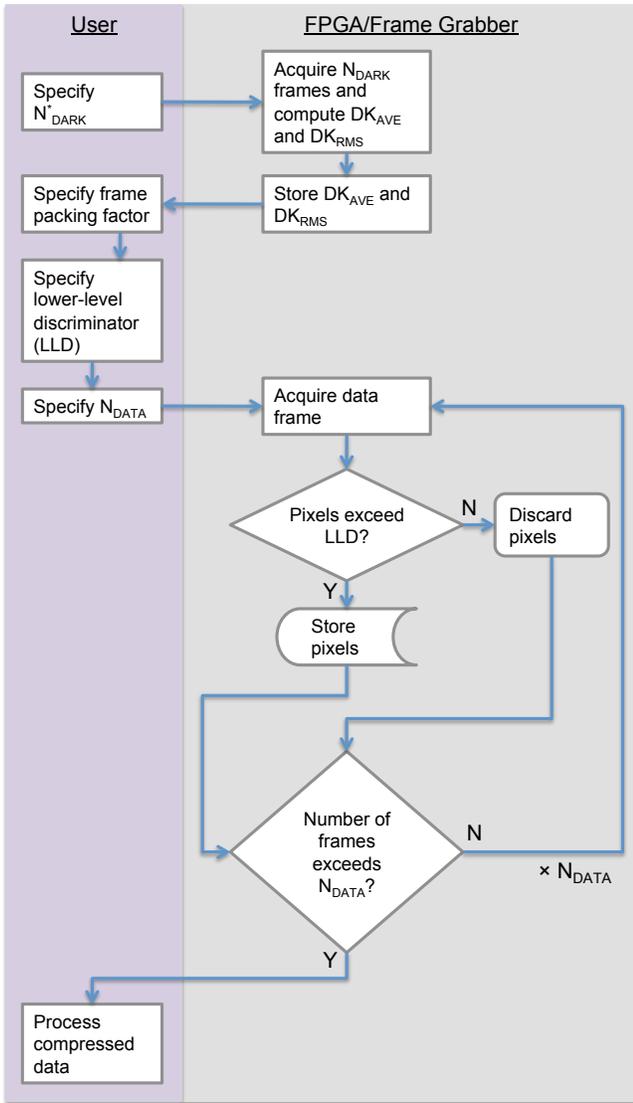


FIG. 2. Schematic of dark and data acquisition processing steps. For simplicity, the flow chart shows the number of dark frames being the same for the average and rms determination, but this need not be the case.

2. The user specifies how many dark frames, $N_{\text{DARK}}^{\text{RMS}}$, should be collected to determine the rms dark signal. (In practice, we use $N_{\text{DARK}}^{\text{RMS}} = N_{\text{DARK}}^{\text{AVG}} \equiv N_{\text{DARK}}$.) $N_{\text{DARK}}^{\text{RMS}}$ dark frames are accumulated and, using the mean dark frame determined in Step 1, the mean absolute deviation (MAD) is recursively determined and stored on disk. (The MAD is more easily calculated in the FPGA than the variance.) Provided that the variance in the dark signal is well approximated by a normal distribution, which we will show below to be the case for the detectors we use, then the standard deviation or rms is given by $\sqrt{(\pi/2)}\text{MAD}$.
3. The user specifies the frame packing ratio. Within

a continuous sequence of data acquisition, the frame grabber requires a fixed frame size so the FPGA can not output variably-sized compressed data frames to the frame grabber. Moreover, it is inconvenient to constantly change the frame buffer size in the image grabber as signal levels change. Instead, the FPGA packs the user-specified integer number, N_{pack} , of compressed frames into a single uncompressed frame and then passes the packed frames to the frame grabber. This procedure reduces the rate of data flow from the frame grabber to the computer bus by $N_{\text{pack}} \times$. Typical packing ratios used in our experiments are 4 or 8 or 16.

4. The user specifies the LLD according to Eqn. 1.

Steps 1–4 need only be repeated occasionally. Thereafter, the number of frames, N_{DATA} to accumulate in a sequence is specified and the detector accumulates compressed frames and writes them to local or network-attached storage.

III. FPGA SYSTEM ARCHITECTURE

In Fig. 3 is a diagram of the hardware layout of the FPGA circuitry the frame grabber. The FPGA is connected by 64 bit wide busses to two banks of Dynamic RAM (DDR) and two banks of Static RAM (SRAM). While the SRAM provides random access memory to the FPGA for processing images, the DDRs are designed for high speed block transfers and are not intended for random access. Because all busses to and from memory are 64 bits wide, four 16-bit pixels are processed at once. Raw image data from the camera is transferred to the FPGA over a 64 bit buss, while the FPGA sends processed data over a 64 bit buss to the CPU (via the PCI buss on the computer). The memory banks are needed for storing images to be used in dark subtraction, image accumulation and averaging, and compression. In our current implementation, only the DDR memory is used. As shown in the figure 64 bit busses link the FPGA to the camera interface and host computer's PCI buss for transfer to the host CPU.

Figure 4 shows a detailed view of the FPGA firmware design and how it connects to the hardware on the frame grabber. Each DDR bank is read and written in blocks. These blocks are typically 64 64-bit words, but can be set to arbitrary sizes based on the size of the images from the camera. Transfer logic controls the block transfers to and from the DDR banks. First-In-First-Out (FIFO) memories convert the discreet blocks of data from the DDR to continuous data streams that can be synchronized with camera data and data from the second memory bank. It is essential that data streams from memories and camera be properly synchronized so computation such as image averaging or dark subtraction can be done correctly. Note that each DDR bank features two FIFOs for reading memory to the FPGA and one FIFO for writing back

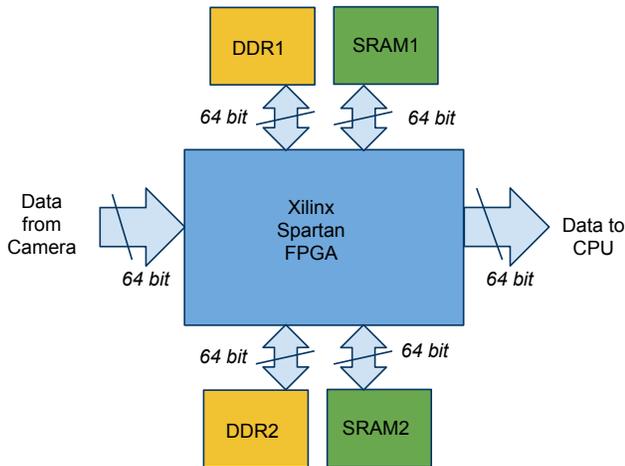


FIG. 3. Block diagram of FPGA architecture residing on Coreco Frame Grabber. Xilinx FPGA receives data from camera, processes data, then sends data to CPU. FPGA has access to DDR and SRAM for storing images. All busses are 64 bits wide and send 4 pixels at a time

the Memory. The purpose of the extra readout-FIFO is give the DDR a second data stream for reading out two separate images from the DDR. In principle, any number of FIFOs can be associated with a DDR to create the functionality of a multi-port RAM, or several DDR banks. One application of using multiple data streams from a DDR is in processing images with 32 bit precision. As data from the camera arrives in 64 bit words containing 4 pixels, 128 bit data words can be read out of the DDRs using multiple FIFOs representing four 32-bit pixels. This is necessary for accurate computation of image mean or standard deviations.

The Math Functions block in Figure 4 assigns a time stamp to each incoming image. The purpose of the time stamp is to record when the image was acquired. When performing XPCS experiments it is desired to measure the properties of a sample as they change with time. Also, time stamps can be used to detect missed frames, and hence incomplete datasets. The time stamp is a 48 bit binary number derived from a simple digital counter clocked at 132MHz. The C++ software driver converts the time stamp into microseconds. A second time stamp is generated by the software driver of the commercial frame grabber. In applications where the FPGA is not used for real time processing, the software time stamp is used. Additionally, the Math Functions block performs dark subtraction, image mean and mean absolute deviation.

Processing four pixels at once, the Compression Logic applies a lower level discriminator to each pixel, in which a threshold value is subtracted from each pixel value and the result is compared to zero, and stores the pixel value and its location in the image to a register. Originally the plan was to store one pixel value of 16 bits with one pixel location of 24 bits. Because a 36 bit data record does not

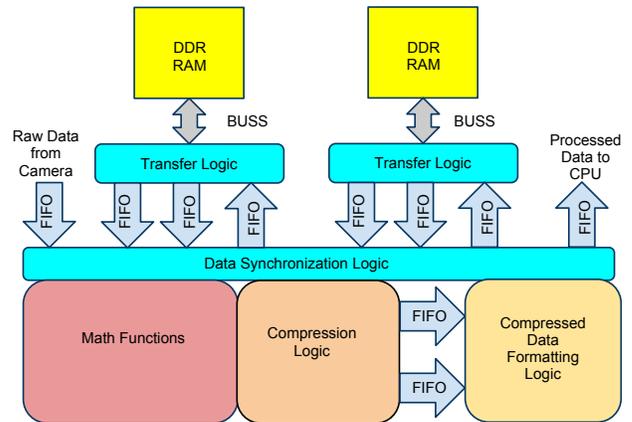


FIG. 4. Block diagram of FPGA firmware. lots of fifos to convert DDR block into continuous data streams. These streams must be as such to avoid scrambling th images. Logic for math compression. Compression FIFOs needed because compression alters the overall data rate

fit nicely into 64-bit DDR RAMs, it was decided to store a single 24-bit location with two pixel values, representing two adjacent pixels. Therefore, the 56 bit data record is nicely stored into the 64 bit word with 8 extra bits for future development.

Because the process of image compression lowers the overall data rate FIFOs, called "compression FIFOs", store and transfer compressed data from the Compression Logic to the Compressed Data Formatter. The Compressed Data Formatter formats and stores compressed data to the DDR memory for subsequent readout to the host CPU. Compressed data may be of any arbitrary size, depending on the actual information in the images. Because the commercial frame grabber is designed to handle images of fixed size, it is necessary to stuff compressed data into blocks the same size as the images coming from the camera. For example, if the camera supplies images of 1024x 1024 16-bit pixels, or 2097152 bytes, the compressed data must be sent to the host CPU in 2097152-byte blocks. To accomplish this, a compression rate C_r is set to an integer from 2 to 16 by the user. The compression rate determines how many compressed images from the camera are stuffed into a single 2097152 byte block. The Data Formatter formats C_r compressed images and appropriate header data containing time stamps and various FPGA settings into the 2097152 byte block. The frame grabber is programmed to reduce the frame rate by C_r . If C_r is set to 4, and the camera runs at 60fps, the FPGA/grabber outputs 15fps of compressed data the host CPU. If the raw images from the camera are sparse, the compressed data frames may contain mostly empty data records. In this case, the compression rate can be set to a larger value.

IV. FPGA MATH ALGORITHMS

Mathematical computations done on the FPGA include recursively averaging N dark images, recursively averaging N absolute offset images (for estimating standard deviation), and dark subtraction. Averaging is defined as

$$\bar{\mathbf{I}} = \frac{1}{N} \sum_{k=0}^{N-1} \mathbf{I}_k \quad (2)$$

To avoid storing N images the FPGA averages the images recursively with the formula,

$$\bar{\mathbf{I}}_k = \alpha \bar{\mathbf{I}}_{k-1} + \beta \mathbf{I}_{\text{raw}} \quad (3)$$

The coefficients α and β can be set arbitrarily, but are generally set to $\alpha = 1$ and $\beta = \frac{1}{N}$. In this case, the recursion is run N times to avoid blowing up to infinity. Because the coefficients are stored as 16-bit integers, with values from 0.0 to 1.0 represented as integers from 0 to 65536. If the number of images to average N is chosen arbitrarily, truncation to 16 bits will add bias to the estimated the mean. The 16 bit truncated coefficient β_{int} is defined as

$$\beta_{int} = \text{round}(65536 * \beta) = \text{round}\left(\frac{65536}{N}\right) \quad (4)$$

To assure that truncation of coefficients does not add bias to the mean calculation we must set N such that

1. N is an integer.
2. $\frac{65536}{N}$ is an integer.

By setting N to a power of 2 we assure $\beta_{int} = \text{round}\left(\frac{65536}{N}\right) = \frac{65536}{N}$, and add no bias to the mean estimation.

Once an estimated mean dark image is obtained and stored to memory, an estimate of standard deviation can be obtained. A commonly used estimator of Standard Deviation is defined as

$$\sigma = \sqrt{\frac{1}{N-1} \sum_{k=0}^{N-1} (\mathbf{I}_k - \bar{\mathbf{I}})^2} \quad (5)$$

Note that estimating standard deviation requires a square root operation, which is difficult to implement on an FPGA. It is simpler to estimate standard deviation by calculating absolute mean deviation defined as

$$\bar{\mathbf{M}} = \frac{1}{N} \sum_{k=0}^{N-1} |\mathbf{I}_k - \bar{\mathbf{I}}| \quad (6)$$

In² it is shown that for a Gaussian process $\sigma = \sqrt{\frac{pi}{2}} \bar{\mathbf{M}}$. We assume Gaussian noise in the images, and use the recursion

$$\bar{\mathbf{M}}_k = \alpha \bar{\mathbf{M}}_{k-1} + \beta |\mathbf{I}_{\text{raw}} - \bar{\mathbf{I}}| \quad (7)$$

to estimate standard deviation. The factor of $\sqrt{\frac{pi}{2}}$ is multiplied in the C software driver to convert to standard deviation. To assure the recursive equations do not accumulate error 32-bit precision fixed point arithmetic is used. Starting with 16-bit raw data and converting to 32-bit data requires twice as many reads and writes to the DDR memories. Our experience is that the FPGA can still process data fast enough to keep up with the XPCS data collection.

To calculate the compression thresholds on a pixel-by-pixel basis, a threshold image is defined as the mean dark image plus some number s of standard deviations:

$$\mathbf{T} = \bar{\mathbf{I}} + s \sqrt{\frac{pi}{2}} \bar{\mathbf{M}} + t, \quad (8)$$

where t is a user defined scalar threshold added to all pixels. It is especially useful when no mean dark image $\bar{\mathbf{I}}$ or mean absolute deviation image $\bar{\mathbf{M}}$ is acquired. When the images are compressed, the threshold image \mathbf{T} is subtracted from the raw image, and all processed pixels below zero are discarded. Processed pixels greater than zero are stored in the compressed data records.

XXX Speed of busses in the FPGA and grabber

V. FPGA COMPILATION AND SOFTWARE CONTROL

The firmware was written in VHDL and compiled with the Xilinx development tools into a bit file. When the frame grabber is configured, the bit file is loaded to set up the FPGA configuration. A C++ software driver was written to control the frame grabber using the commercial software API supplied by Dalsa. The software driver is hosted on an EPICS Input Output Controller (IOC) application, to interface to the beam line control system. The graphical user interface (GUI) for FPGA control is shown in Figure 5.

The controls on the window are described as follows. The FPGA control window allows the user to set the following options:

1. Acquisition Mode: Pulldown menu to set the mode of operation of the FPGA. Modes include
 - (a) Pass Mode: FPGA returns raw data to CPU.
 - (b) Background Subtract: FPGA subtracts background image in RAM and returns result to CPU
 - (c) Compression: FPGA subtracts background image including pixel-by-pixel thresholds,

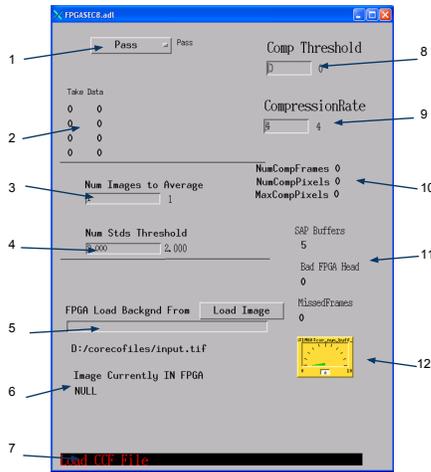


FIG. 5. Screen shot of EPICS user interface for FPGA-based system. This window controls the FPGA settings. Not shown are windows for viewing images, controlling camera, and saving files.

compares result to scalar threshold, and returns compressed images. Frame rate is reduced.

- (d) Integrate Mean Image: Recursively average user-specified number of frames. Store result to file.
 - (e) Integrate Standard Deviation Image: Recursive average mean absolute offset images, scale to standard deviation image. Store result to file.
 - (f) Return Background: Return to CPU data stored in background subtraction memory.
2. FPGA status flags: Integers displayed tell status of FPGA.
 3. Number of images to average- When acquiring mean or mean deviation images user sets number of images to accumulate
 4. Number of Standard Deviations: Once standard deviation of dark image is found, compression threshold LLD is set to some number of standard deviations.
 5. Threshold and background files can be loaded from disk. User enters filename.
 6. Filename of image in FPGA is shown
 7. Status of FPGA and Grabber is shown
 8. compression threshold: Scalar to be added to threshold file.
 9. Compression rate: Number of images to stuff into one data block. Compression rate of 4 reduces frame rate by 4.

10. Compression status: Displays number of frames in one data block, number of pixels in the frame, and Max number of pixels that can be stuffed into compressed frame.

11. Grabber Status: Tells how much memory for temporary storage of images is available, number of bad compressed images acquired, and missed frames.

12. Memory Meter: meter tells when system is about to run out of memory and miss frames.

VI. SYSTEM PERFORMANCE

Describe performance of the system.

VII. FPGA VERSUS GRAPHICS PROCESSING UNIT

GPU versus FPGA. Graphics Processing Units (GPU) originally designed for handling graphics on PCs are increasingly being used for scientific data processing. We plan to implement real time processing on a GPU for XPCS data. We do not feel the GPU will replace the FPGA or vice versa. As data comes from the camera it must be transferred over the PCI bus for processing, then over the PCI bus again to the CPU. The FPGA processes and compresses data before it reaches the PCI bus. In this way only compressed data is transferred. The use of a GPU and FPGA are complementary: the FPGA can compress the data and do some modest calculations while the GPU can perform complex calculations on compressed data. In this way one can have the advantages of both technologies. As GPUs become more integrated with the CPU, they will become even more useful for scientific data processing. However, the FPGA occupies a special niche for data collection for reducing the data rate before it is transferred into the PC's bus.

Itemize properties of incoming data and processed data

Results (and graphs) of compression factors and max frame rates versus compression factors.

VIII. CONCLUSIONS

We have successfully implemented an FPGA-based system for on-the-fly thresholding and compression of rapidly streaming multi-speckle XPCS data. The system allows us to acquire and compress 1 megapixel CCD frames at ≥ 60 fps allowing XPCS to measure delay times spanning at least 5 orders of magnitude. Future work will focus on extending the current 32-bit architecture to 64-bit architecture and physically-separating the FPGA unit from the frame grabber. The former will allow us to support newly emerging faster XPCS-suitable detectors⁷ and the latter will provide more flexibility with respect to choice of frame grabbers while, at

the same time, allowing a stable FPGA development environment. We are also considering implementing so-called “droplet” algorithms⁷ or performing multi-tau correlations directly in the FPGA⁷. A drawback of the latter though is that it would restrict its application to only stationary systems despite the significant progress that has been realized lately with respect to non-equilibrium and intermittent dynamics^{7, 8, 9}. In this regard, a more promising future development is to marry the pre-processing described in this paper, with correlations performed via high-performance computing (HPC). The inherently parallel nature of multi-speckle XPCS suggest that firmware thresholding combined with HPC

can yield autocorrelation functions in real time.

ACKNOWLEDGMENTS

M. Sikorski acknowledges support from a Laboratory Directed Research and Development (LDRD) project. Use of the Advanced Photon Source at Argonne National Laboratory was supported by the U. S. Department of Energy, Office of Science, Office of Basic Energy Sciences, under Contract No. DE-AC02-06CH11357.