CONTROLLING THE EXCALIBUR DETECTOR

J. A. Thompson, I. Horswell, J. Marchal, U. K. Pedersen, Diamond Light Source, Oxfordshire, UK.
S. Burge, J. D. Lipp, T. Nicholls, Science and Technology Facilities Council, Oxfordshire, UK.

Abstract

EXCALIBUR is an advanced photon counting detector being designed and built by a collaboration of Diamond and the STFC. It is based around 48 CERN Medipix3 chips arranged as an 8 x 6 array. The main problem addressed by the design of the hardware and software is the uninterrupted collection and safe storage of image data at rates up to one hundred 2048 x 1536 frames per second. This is achieved by splitting the image into 6 'stripes' and providing a parallel data path for each stripe all the way from the detector chips to the storage. This architecture requires the software to control the configuration of the stripes in a consistent manner and to keep track of the data so that the stripes can be subsequently stitched together into frames.

INTRODUCTION

A 2D position sensitive detector is required for use in a range of imaging experiments at Diamond Light Source. Initially the detector will be applied on photon beam line I13, but the resulting design may be applied on other beam lines and in other applications.

The primary application will be in the technique of Coherent Diffraction Imaging (CDI) [1]. The detector will also find applications in Photon-correlation Spectroscopy (XPCS) [2], full-field microscopy and holo-tomography.

The Medipix3 Device

The Medipix3 device is a 256 x 256 pixel photon counting detector. It has three basic operating modes, single-pixel, charge-summing and colour. The EXCALIBUR instrument will initially support single-pixel and charge-summing modes.

The charge that is collected by a pixel due to an impinging photon is measured and compared to a programmable threshold. Only charge pulses that meet the threshold's requirements are registered by the 12 bit counter. There are two 12 bit counters associated with each pixel. These are normally used so that one is counting photons while the other is being read out, to eliminate dead time between frames.

Targeted Capabilities

Table 1 summarises the capabilities of the EXCALIBUR detector and draws comparison with the Pilatus [3].II and XFS detectors.

Table 1: Specification And Comparison With Competitors

Parameter	EXCALIBUR	Competitors
Frame size	2k x 1.5k pixels	
Maximum frame rate	100Hz, 12 bits continuous 1kHz, 12 bits burst 30kHz, 1 bit histogrammed	Pilatus II < 300Hz Pilatus XFS > 10kHz
Dead time between frames	0	Pilatus II: ~2ms defined by chip read out time.
Pixel size	55 um	Pilatus II 172um Pilatus XFS 75um
Dynamic range	12 bits	Pilatus XFS 12 bits
Quantum efficiency	~50% at 15keV	~50% at 15keV

HARDWARE ARCHITECTURE

The hardware architecture is shown in Fig. 1. It consists of a sensor array, a number of front-end modules (FEMs) and a cluster of read-out node PCs.

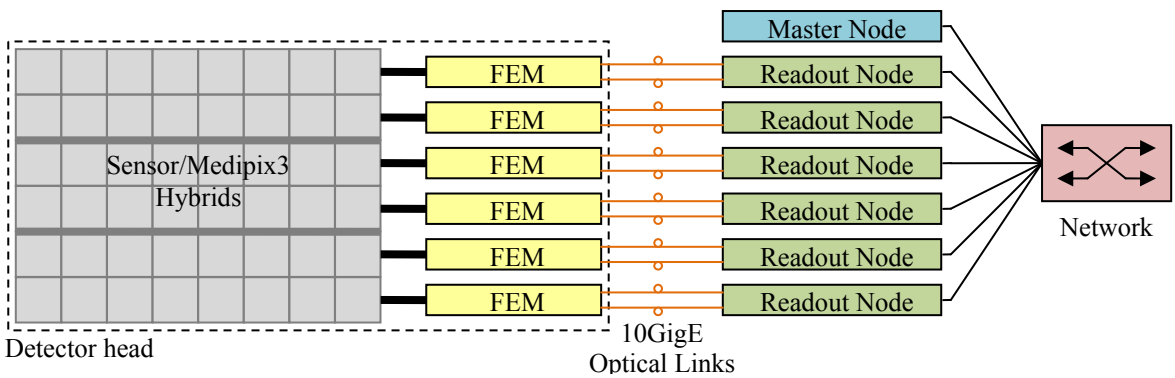

Figure 1: Hardware architecture.

Copyright © 2011 by the respective authors — cc Creative Commons Attribution 3.0 (CC BY 3.0)

Sensor Assemblies

The sensor assembly consists of three hybrid modules, each containing an 8 by 2 array of sensors and Medipix3 chips. The detector assembly consists of three hybrid modules, each containing a large silicon sensor bonded to an array of 8 x 2 Medipix 3 chips. The gaps between the chips on one module are 3 pixels (the minimum possible); the sensor pixels that cover these gaps are larger, as shown in Fig 5. A 124 pixel wide inactive region is present between modules due to the presence of wire bond pads connecting the chips to their read-out electronics.

Front End Modules

A FEM is provided for each horizontal row of sensors, giving a total of 6 FEMs. Each FEM co-ordinates the acquisition of data by a row of Medipix3 chips and also provides access to most of the registers of the 8 Medipix3 chips for the embedded software. From the point of view of the software, a FEM provides an image 'stripe' that is 2069 by 256 pixels. The seven inter-chip gaps (each of 3 pixels) are included in the output data but do not contain valid data.

Computing Resource

Six read-out nodes and one master node make up the computing cluster for EXCALIBUR. The read-out nodes communicate with the FEMs through 10G Ethernet fibre optic links. Connection to the Diamond Light Source network backbone is then made through six 1G Ethernet links to a switch with a 10G upstream connection.

Data Storage

The primary storage for data captured by EXCALIBUR will be a Lustre [4] distributed file system. Files may also be stored locally on the read-out node's file systems if required.

EMBEDDED SOFTWARE ARCHITECTURE

The embedded software is based on EPICS [5] using the area detector module [6]. Figure 2 shows an overview of the data path software; the classes ADDriver, NDPluginDriver and NDPluginFile are area detector base classes supplied by the module library.

Image Stripe Handling

The AdFem class is responsible for interfacing to the FEM. It receives image stripes, places them into standard area detector buffers and then passes them on for further processing. In addition, it provides EPICS process variables that allow the control of acquisition and the configuration of the Medipix3 and FEM devices.

The pixel arrangement in the stripes is different for each of the pair of FEMs that service a sensor module, due to the rotation of the Medipix3 chips. This is corrected by the AdFemFix class. This function is provided as a separate area detector plug-in to allow it to run on a different core to the FEM readout, keeping the throughput high.

Summary Image Production

The AdSlaves and AdMaster classes together provide the mechanism for transferring at a low rate (configurable

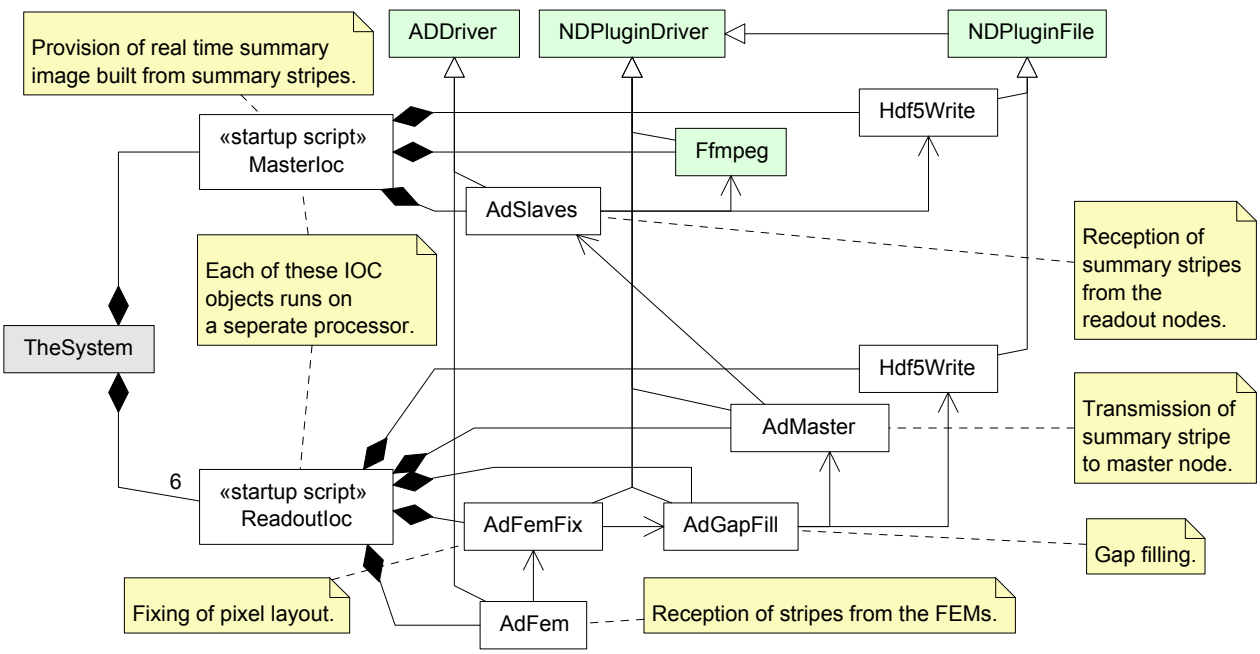

Figure 2: UML class diagram showing a summary of the data path software.

Copyright © 2011 by the respective authors — cc Creative Commons Attribution 3.0 (CC BY 3.0)

as 1 in N) a summary image stream from the read-out nodes to the master node. A class diagram is shown in Fig. 3.

The AdMaster class transfers every Nth stripe to the AdSlaves class through a TCP socket. The stripes are written into the correct place in a single area detector buffer to stitch them together. Once a stripe is received from each read out node, the complete frame is passed on for further processing on the master node. This will normally include an MPEG streaming plug-in to provide data for a suitable viewer.

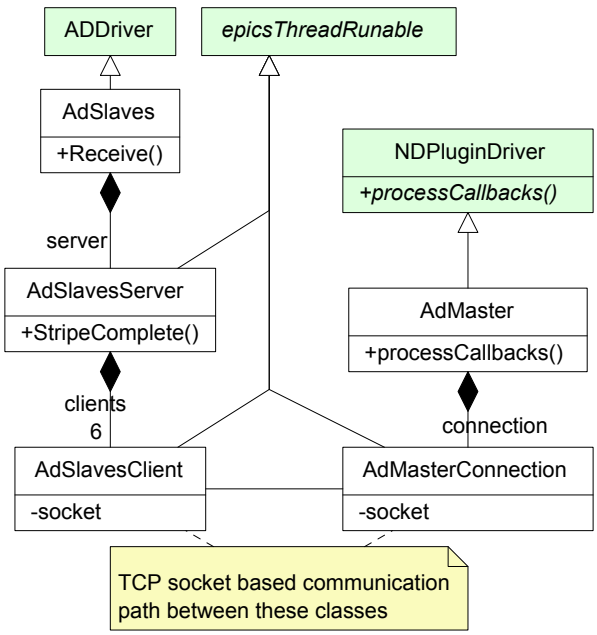

Figure 3: UML class diagram of the summary image processing.

Parallel File Handling

The read-out nodes each open the same file on the Lustre file system and write to the correct part of the file to assemble the stripes into the frame. Not only does this avoid processing by the instrument; it also utilises the multiple parallel write stream capability of Lustre to keep the data rate high.

The files are written in the HDF5 format [7]. The file writing plug-in utilises the version of the HDF5 software that uses the openMPI parallel processing library [8].

Filling the Gaps

The gaps between the adjacent Medipix3 chips are required to be filled, either by a constant value or by interpolation between the pixels surrounding the gap. A class diagram of the plug-in responsible for this task, AdGapFill, is shown in Fig. 4. The chip layout and gap details are shown in Fig. 5.

The large gaps between the modules are always filled with a constant value. The HDF5 file format provides the ability to fill automatically gaps in the recorded data with a constant. Advantage is taken of this feature to fill the large gaps without increasing the data rate of the system.

The small gaps may be filled with a constant value or interpolated, according to configuration. The small vertical gap pixels are already present in the stripe data, so the gap filling plug-in writes the constant or interpolated value as appropriate. The gap filling feature of HDF5 is used again for the horizontal gaps in constant fill mode.

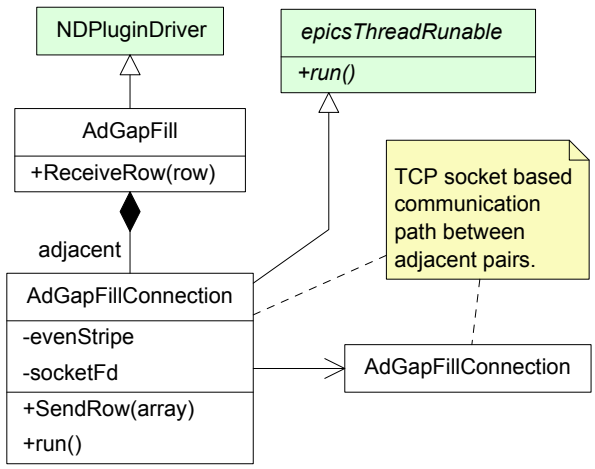

Figure 4: UML class diagram of the gap filling processing.

Interpolating across the small gap between stripes is not straightforward; it requires data communication between the adjacent gap filling plug-ins. The plug-in on the upper side of the gap receives the top row of pixels from the plug-in on the lower side. It then generates extra rows of interpolated pixels which extend the stripe downwards, over the gap.

The sensor areas of the pixels adjacent to the small gaps extend over the gap, as shown in the right of Fig 4. This means that photons landing in the pixel gaps are still captured. Interpolation is therefore concerned with sharing captured photons between the edge pixels and the gap pixels.

Configuration Control

The read-out nodes each provide EPICS process variables that control the operation of a single FEM and its associated Medipix3 chips. These are all available to allow the individual control necessary during many engineering operations, especially during calibration.

For normal user operation, co-ordinated control of the entire instrument is required. This is provided by a separate set of EPICS process variables residing on the master node that model the instrument in a user-oriented manner. These process variables are then expanded and distributed to the read-out nodes by the configuration control state machine. The same entity also monitors the operating state of the whole instrument and audits the distributed configuration.

Copyright © 2011 by the respective authors — cc Creative Commons Attribution 3.0 (CC BY 3.0)

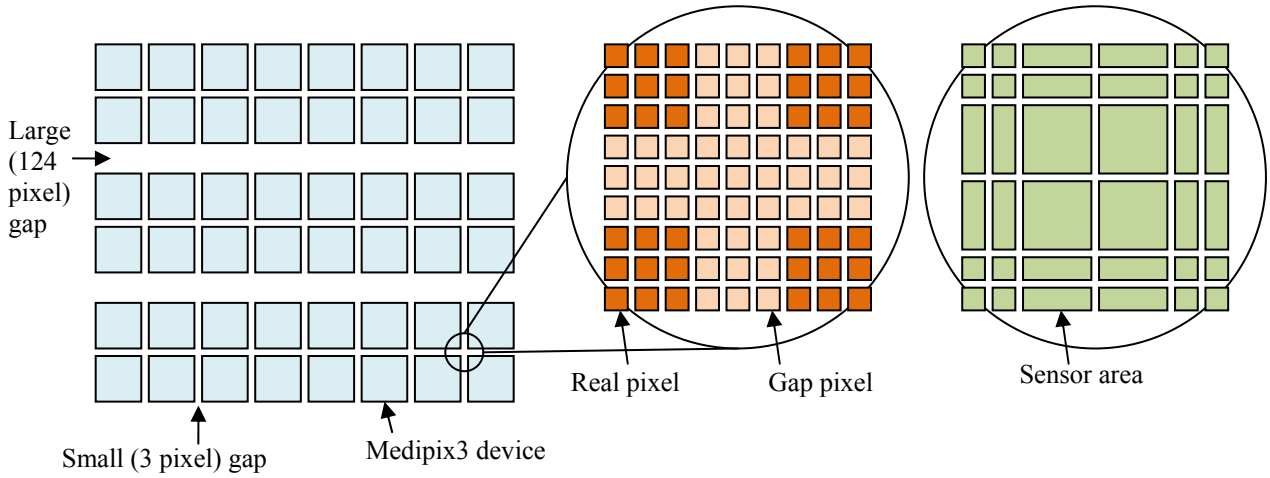

Figure 5. Medipix3 device layout and gap details.

Instrument Calibration

There are many calibration procedures to be undertaken on the instrument to provide the necessary information for correct operation. These include pixel threshold trimming, flat field correction and bad pixel maps. In general, the procedures will be performed by external scripts and will result in calibration files that are loaded onto the instrument and activated at the appropriate time.

SOFTWARE TESTING

The testing of the software was carried out in two phases. The first phase used a software simulation of the hardware with an extensive set of automatic test routines. This provided an automatic record of the testing undertaken and an easy way of repeating the tests at any stage during instrument development.

The second phase was a period of integration and testing with the instrument hardware. This necessarily involved rather more manual intervention and grew as the various parts of the instrument were brought together.

Instrument Simulator

To allow unit testing of the embedded software before the hardware was available, a simulation of the hardware was produced. This consists of three parts, the Medipix3 chip simulation, a FEM simulation and a back door through which the test suite can control the simulation.

The Medipix3 chip simulation is reasonably detailed from a software point of view. It includes the ability to generate pixel data that responds to the settings of the chip's registers in a reasonably appropriate manner.

The FEM simulation is rather simpler. Its main job is getting data into and out of the Medipix3 chips, so many of its functions are invisible to the software.

The simulation connects to the embedded software at a point where the FEM driver library connects to the embedded software proper. This allows the simulation to take the form of a library that is linked in place of the FEM driver library.

Automatic Test Suite

The tests written covered low level Medipix3 register access, configuration management and image frame processing. They were written in Python [9] utilising the PyUnit standard library. Cases are able to control the simulation through its backdoor, allowing both valid and fault conditions to be tested.

CONCLUSIONS

An instrument capable of sustained capture of 2k x 1.5k pixel images at 100 frames per second has been built. This corresponds to a throughput of just over 5Gbps to the Lustre file system in 6 coordinated, parallel data streams.

REFERENCES

[1] http://en.wikipedia.org/wiki/Coherent_diffraction_imaging
[2] http://sector7.xor.aps.anl.gov/~dufresne/UofM/xpcs.html
[3] http://www.dectris.ch/
[4] http://wiki.lustre.org/index.php/Main_Page
[5] http://www.aps.anl.gov/epics/about.php
[6] http://cars9.uchicago.edu/software/epics/areaDetector.html
[7] http://www.hdfgroup.org/HDF5/
[8] http://www.open-mpi.org/
[9] http://www.python.org/doc/

Copyright © 2011 by the respective authors — cc Creative Commons Attribution 3.0 (CC BY 3.0)