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.

1. Introduction

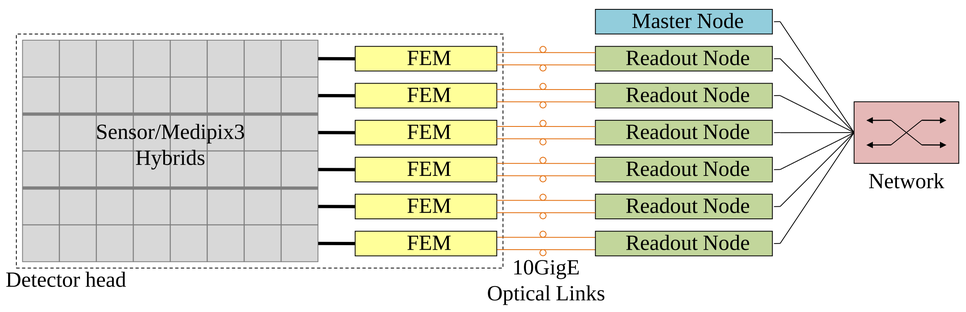
A 2D position sensitive detector is required for use in a range of imaging experiments at Diamond Light Source. Initially the detector will be applied on photon beam line I13, but the resulting design may be applied on other beam lines and in other applications.

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

The table right summarises the capabilities of the EXCALIBUR detector and draws comparison with the Pilatus II and XFS detectors.

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

2. Hardware Architecture



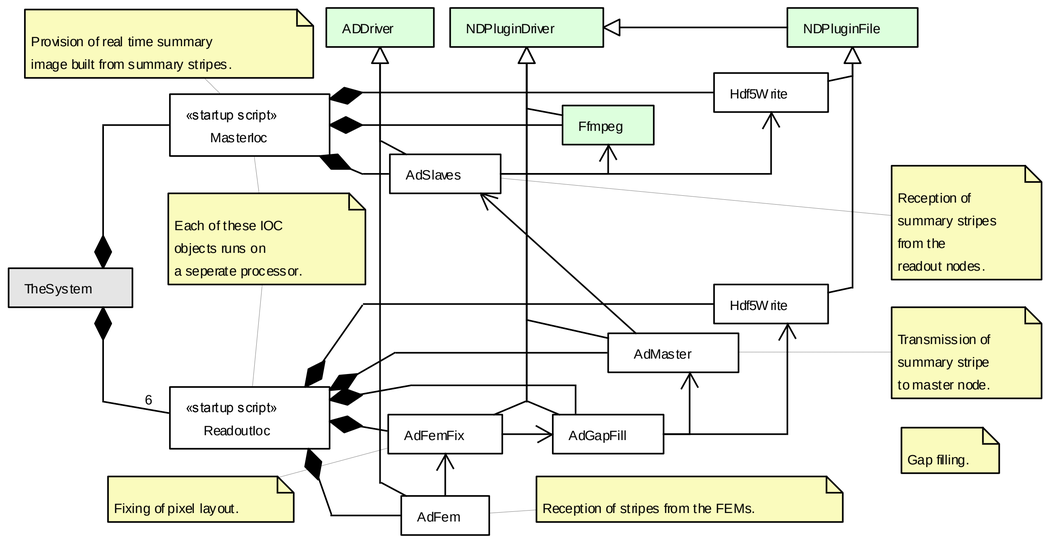
The hardware architecture consists of a sensor assembly, a number of front-end modules (FEMs) and a cluster of read-out node PCs.

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.

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.

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.

3. Software Architecture



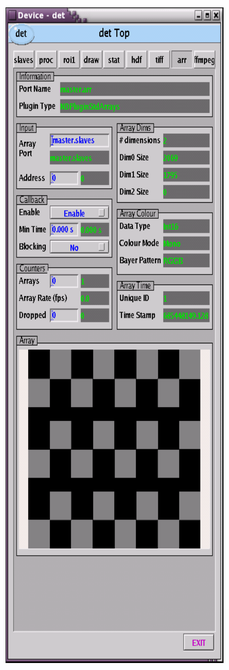
The embedded software is based on EPICS using the area detector module. Above is a UML diagram showing an overview of the data path software; the classes ADDriver, NDPluginDriver and NDPluginFile are area detector base classes supplied by the module library.

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.

Image data is written to file in the HDF5 format by the Hdf5Write class. 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 file writing plug-in utilises the version of the HDF5 software that uses the openMPI parallel processing library.

4. Summary Image

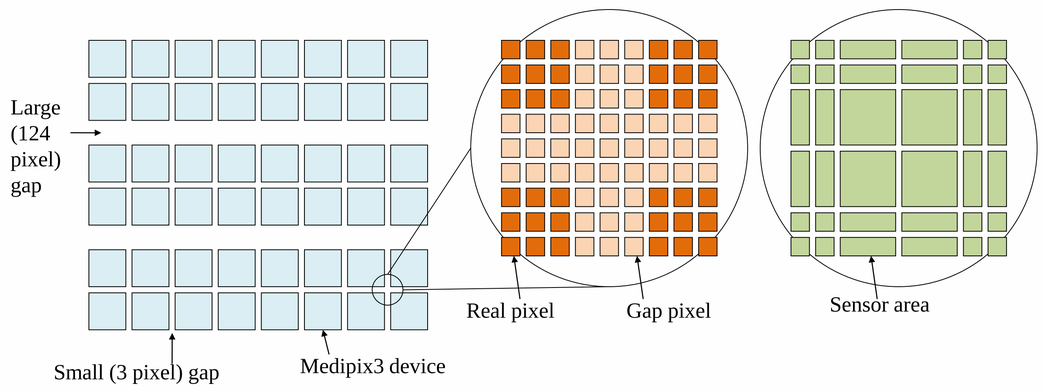


The AdSlaves and AdMaster classes (see Software Architecture) together provide the mechanism for transferring at a low rate (configurable as 1 in N) a summary image stream from the read-out nodes to the master node. The AdMaster class transfers every Nth stripe to the AdSlaves class through a TCP socket.

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.

On the left is an EDM screen showing the stitched summary image. The large horizontal gaps are clearly visible.

5. Filling the Gaps



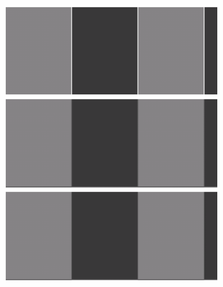
The gaps between the adjacent Medipix3 chips are required to be filled, either by a constant value or by interpolation between the pixels surrounding the gap. The chip layout and gap details are shown above. The class responsible for the work is AdGapFill (see Software Architecture).

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 small horizontal gaps are created by adding extra rows to the upper stripe of each pair.

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. The sensor areas of the pixels adjacent to the small gaps extend over the gap, as shown in green on the right of the diagram. 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.

The Composite image on the right shows the effect of the gap filling modes; the top stripe is constant fill, the middle is mean fill, the bottom is linear fill.



6. Testing

```
1..5
ok 1 - CaseConstant : Constant filling mode.
ok 2 - CaseScaleEdges : Scale edge pixels with constant filling mode.
ok 3 - CaseMean : Mean filling mode.
ok 4 - CaseLinear : Linear filling mode.
ok 5 - CaseSynchronisation : Adjacent row communication synchronisation.
# ============================
# Passed 5/5 tests, 100.00% okay, in 39.02s
#
```

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. An example test report for the gap filling module is shown above.

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.

October 2011. For more information please contact jonathan.thompson@diamond.ac.uk