BrightnESS

Building a research infrastructure and synergies for highest scientific impact on ESS

H2020-INFRADEV-1-2015-1

Grant Agreement Number: 676548

Deliverable Report: D5.4 Report on field data acquisition

1 Project Deliverable Information Sheet

BrightnESS Project	Project Ref. No. 676548	
	Project Title: BrightnESS - Building a research infrastructure and synergies for highest scientific impact on ESS	
	Project Website: https://brightness.esss.se	
	Deliverable No.: 5.4	
	Deliverable Type: Report	
	Dissemination Level: Public	Contractual Delivery Date: 31.08.2017
		Actual Delivery Date: 22-09-2017
	EC Project Officer: Mina Koleva	

2 Document Control Sheet

Document	Title: BrightnESS_Deliverable_5.4	
	Version: 1.0	
	Available at: https://brightness.esss.se	
	Files: 1	
Authorship	Written by	Jonas Nilsson (WP5)
	Contributors	Tobias Richter (WP5 leader)
	Reviewed by	Morten Jagd Christensen (WP5), Afonso Mukai (WP5), Mark Koennecke (WP5), Roy Pennings (WP1)
	Approved by	BrightnESS Steering Board

3 Terms and Abbreviations

Term/abbr.	Description
ADC	Analog to Digital Converter
DAC	Digital to Analog Converter
DMSC	Data Management & Software Centre (ESS division)
EPICS	Experimental Physics and Industrial Control System
ESS	European Spallation Source
EVG	Event Generator
EVR	Event Receiver
FIFO	First In, First Out
FPGA	Field-Programmable Gate Array
HDD	Hard Disk Drive
MRF	Micro Research Finland
NeXus	A neutron, X-ray and muon science data file format.
RTOS	Real-Time Operating System
SSD	Solid-State Drive
S/s	Samples per second
TTL	Transistor-Transistor Logic

4 List of Figures

Table of Contents

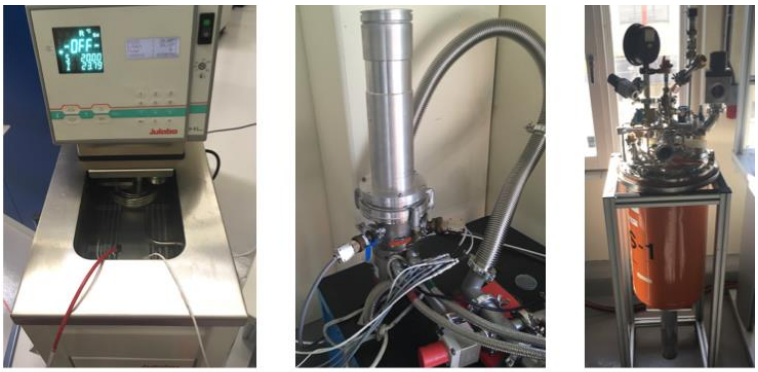
5 Executive Summary

To enable cutting edge materials research at the European Spallation Source (ESS), information about the sample and its environment conditions, like temperature, strain, or electric and magnetic field are as important as the collection of neutron event data. This BrightnESS task 5.2 deals with the collection of sample environment information at a higher rate than is routinely done at existing neutron scattering facilities, in order to be prepared for the demands of the high flux source that ESS will provide. This first deliverable of the task lays out the work that has been done so far to gather requirements, identify potential hard- and software solutions, as well as prepare for integrating the sample environment data stream into the general data acquisition chain. It concludes with a roadmap to get to a working demonstrator system by the end of the project. The main challenges are the accurate absolute timestamping of the sampled data, which makes full use of the advanced ESS timing system, as well as the need to support continuous data sampling at high rates, which is a rare operations mode for fast ADCs. The work has been carried out in close collaboration with the other tasks in WP5, the ESS Detector Group (largely in WP4), and the ESS Integrated Control Systems division (ICS).

6 Background and Requirements

One of the primary goals of the European Spallation Source (ESS) is to enable research into materials science. This includes examining samples in a wide range of sample environment conditions, e.g. changing the temperature or strain of the sample. Figure 1 shows examples of devices for measuring and controlling the environments of samples undergoing measurements. For analysis, these sample environment parameters have to be measured, stored and be correlated to the data from the neutron detectors. Examples of some of the parameters measured in the sample environments shown in this figure as well as a few others are:

- Strain sensors
- Pressure sensors
- Magnetic field sensors
- Electric field sensors
- Light sensors
- Humidity sensors



(a) (b) (c)

Figure 1. From left to right: a) Device for controlling the temperature of samples submerged in a liquid. b) Vacuum chamber with pressure sensors and sample temperature control capability. c) Cryostat with cooling controlled by adjusting the evaporation of liquid Helium.

Many environmental parameters will only change slowly (on the order of seconds to hours) and can thus be handled by EPICS (Experimental Physics and Industrial Control System), the standard system used for command and control. Some parameters, however, will be rapidly changing or alternating at high frequencies and will require measurement (or sampling) rates on the order of ~100 to 1,000,000 samples per second (S/s) as well as robust timestamping of the data. At high sampling rates and timestamping requirements, more specialised solutions or modifications to EPICS will be required.

This report examines acquisition control, aggregation and storage of data produced from the sampling of environmental sensors at a high rate (i.e. more than ~ 100 S/s). Most environmental sensors provide an output voltage (either directly or indirectly) which commonly has a linear relationship with the parameter being measured. Thus, measurement of a parameter is simply done by feeding this voltage into an Analogue to Digital Converter (ADC). By knowing the relation between the output voltage from the sensor and its sensitivity, the original parameter can be calculated.

The facility-wide timing system at ESS will ensure the synchronisation of instruments and other equipment to the accelerator pulses and will enable timestamping of events and measurements (with a resolution of ≈10 ns) for correlation purposes. There are several ways in which the timing system can be used to determine the timestamps of ADC samples. The different alternatives, as well as their advantages and disadvantages, will be discussed in this document.

The maximum required sampling rate can be approximated by comparing the maximum speed of the neutrons and the sample size. Given that the fastest neutrons will have a speed of 10,000 m/s [1] and that the maximum sample size is no larger than ~1 cm, it will take the neutrons ~1 µs to pass through the sample. Assuming that a sample environment parameter remains static during this time, this yields a maximum required sampling rate of 1 MS/s. As many beam lines will make use of a large wavelength spectrum, events will be collected for the entire duration between proton pulses. This requires that the sample environment parameters are monitored continuously as well.

Based on similar projects in development at other research facilities [2] the target resolution of the data is 16 bits. The data type to be used for storage as well as at what point in the data processing pipeline conversion to this type will be done has not been decided. Storage of the data as a floating-point voltage is preferred for usability reasons. As single precision floating point numbers use 23 bits for storing the significand, usage of this type will result in zero loss

of precision. If high resolution ADCs (e.g. 24 or 32 bits) are required at a later stage, the converted values will have to be stored using double precision floating point numbers.

Assuming a worst case scenario of 1 MS/s and that the value as well as the timestamp for each value is stored using double precision floating point values (64 bits each), this yields a data production rate of ≈61 MB/s per ADC channel. It must be stressed that this is a worst case approximation and that a more conservative approximation would result in a data rate of ≈6 MB/s per ADC channel.

In summary, there is a need for a sampling system that can perform continuous sampling (and storing) of an analogue signal at a rate of 1 MS/s with a resolution of timestamp bits or more. Every sample has to be timestamped using a facility wide timing system. The maximum (continuous) data production rate is ≈61 MB/s and thus the storage medium will have to be able to write data at this rate or higher.

7 Timing and ADC Hardware at ESS

In this section we will discuss the facility wide timing system as well as the two hardware alternatives (IFC14xx-based and the Detector Readout Electronics) for doing the actual sampling of the analogue signals and how these systems might interface with the timing hardware.

7.1 MRF Timing Hardware

The ESS facility will be using timing hardware developed by *Micro-Research Finland* (MRF) to distribute a global clock across the facility for synchronising and timestamping as decided by the accelerator division and ICS. The timing system developed by MRF uses a master clock provided by an *event generator* (EVG)[3] which is then fanned out to individual *event receivers* (EVR)[4] using a fibre optical connection (see Figure 2). Synchronisation of the EVRs to the EVG is done automatically. The EVG can also generate events which are then transmitted to all the connected EVRs [5]. For the ESS facility, the EVG will be set to automatically generate periodic events at several different frequencies including the 14 Hz rate at which the accelerator will be pulsed. Events can also be generated locally from TTL inputs by individual EVRs.

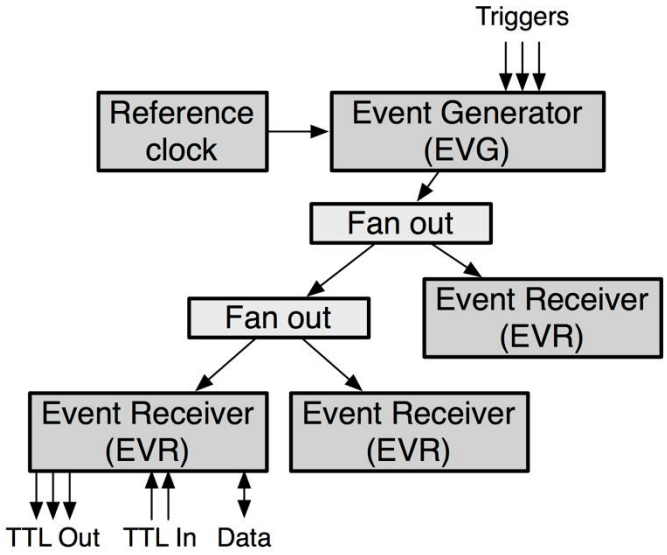


Figure 2. Simplified overview of the MRF timing system. The event generator distributes a master clock to the event receivers. The EVG also distributes timestamped events generated from triggering signals which are transmitted to the EVRs. Each EVR can also generate local events from inputs.

The MRF timing hardware needs to be controlled by a CPU and Linux drivers for doing this are provided by MRF. EVR hardware exists with a range of different interfacing standards, including the µTCA backplane connection standard and PCIe for use with desktop computers.

An event received or generated by an EVR will result in a timestamp for that event being created. The timestamp can be retrieved by the CPU to which the EVR is connected. The EVR can also be configured to output a TTL pulse, with a configurable delay and width, on an event. These features provide two alternatives for making sure that the sampled data is properly timestamped:

1. synchronisation of the clock of a device (e.g. sampling hardware) to the global clock used by the EVG/EVR. This is done by configuring the EVR to output a TTL pulse at a specific time. The sampled data can (using the TTL pulse) then be synchronised to the timing system either in hardware or in software;

2. trigger sampling of the analogue signal using the (periodically) generated events as provided by the EVG/EVR and get timestamps for those events.

7.2 IOxOS IFC14xx-Based Hardware

For complex experiment set-ups, there is a need for a modular (and thus flexible) hardware system for communication, controlling devices and sampling sensors in real time or near real time. The rack-based system that will be used at by the ICS at ESS, will implement the relatively recently developed µTCA backplane connection standard, which allows communication at a rate of more than 3 Gb/s between components.

The rack computer systems will use the IOxOS IFC14xx series of CPU cards [6]. The IFC14xx CPU cards have a 4-core 1.8 GHz processor for general purpose processing, 2 GB of RAM and an FPGA for hardware control (e.g. ADCs and DACs). The CPU card will be running Linux and has EPICS support available from the manufacturer for many of the hardware interfaces of interest. Communication between the IFC14xx CPU card and systems external to the rack system can be done using a 1 Gb Ethernet interface module. The operating system used by the IFC14xx CPU is the ELDK Linux distribution with the kernel patched with real-time support to reduce latencies.

The IFC1410 CPU card has two on-board sockets for connecting daughter cards, e.g. the ADC311x series of ADC cards [7]. The ADC311x cards have 8 analogue inputs and samples with 16 bits of resolution at a maximum of 250 MS/s. Alternatively, the IFC1420 CPU card has only a single daughter card socket but also 10 analogue inputs that can be sampled with 16 bits of resolution at 250 MS/s. Due to the flexibility of the FPGA on-board the IFC14xx card, triggering of the sampling can be done from software, TTL input (on the front panel or backplane connector) or from an FPGA implemented triggering algorithm. An ADC3110 card mounted on a IFC1410 CPU card next to an EVR is shown in Figure 3.

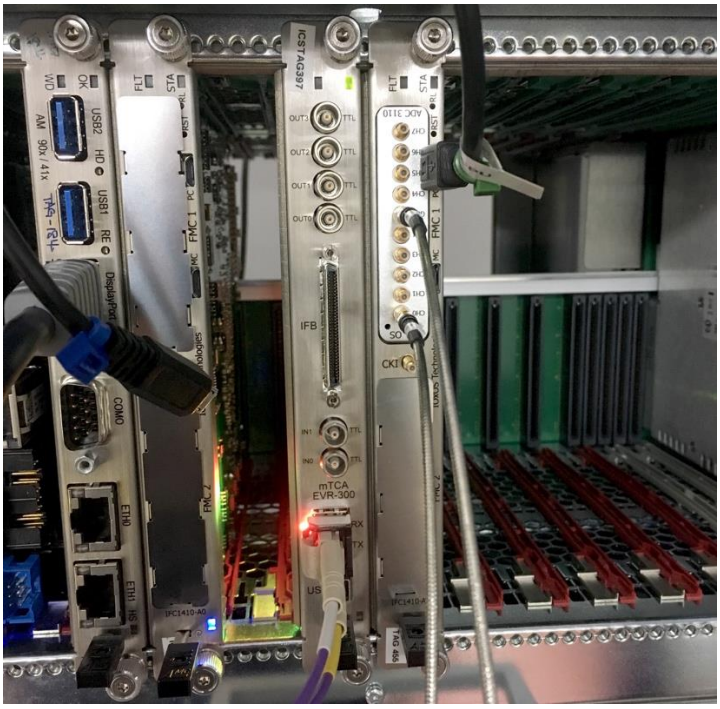


Figure 3. The ADC3110 card mounted on an IFC1410 CPU card in µTCA rack also containing an EVR and additional µTCA cards.

The IFC14xx series of CPU cards are currently under development and only prototypes of the IFC1410 model are available in small quantities as of August 2017. The software, including the FPGA firmware, is also under development. The FPGA firmware used to control the ADC311x series ADCs supports continuous sampling through the use of a FIFO buffer. However, this functionality has not been implemented in the software drivers and the bandwidth of the FIFO buffer is limited to 8 MB/s. This corresponds to e.g. a single analogue channel being sampled at ≈4 MS/s.

7.3 ESS Detector Readout Electronics

As the amount of unprocessed data that will be produced by the neutron detectors at ESS will be large, a system for aggregating and doing first-pass processing of this data is under development (part of BrightnESS Work Package 4). The system will use FPGA-based readout electronics for aggregating and timestamping the detector data, before sending it to general purpose computing hardware for further processing. The latter parts of this system will be data type agnostic and will thus be able to also aggregate and timestamp data produced by an ADC.

The timestamping of the neutron/ADC data will be done in the “Master FPGA” shown in Figure 4. The Master FPGA will use its own internal timestamping mechanism which will require synchronisation to the timing system. This is done by a CPU, which transmits the

timestamp of a future synchronisation pulse to the FPGA via a serial connection. When the synchronisations pulse is registered by the FPGA, the internal clock will be switched over to the time received via the serial connection.

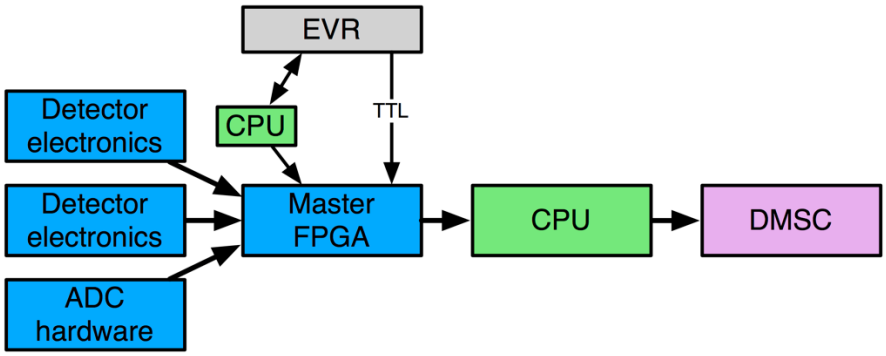


Figure 4. The general outline of the neutron processing system. The "Detector electronics" box represents another level of FPGAs that aggregates data from several detector front ends and the "DMSC" box represents data analysis and storage.

The Master FPGA will be interfaced to the detector electronics using a high-speed fibre-optic connection. The interface between the Master FPGA and the computer doing initial processing is expected to consist of one or several 10GbE fibre optical connections. Assuming that the processing of the data is not CPU-limited, this would allow for a theoretical maximum throughput between the Master FPGA and the computer of ≈1200 MB/s, which is in excess of what is required.

Prototypes for performing non-continuous and timestamped sampling of analogue data is available for testing purposes at ESS (see Figure 5). This sampling hardware uses a 4 channel 14-bit ADC which can sample signals at a rate of 100 MS/s. The high sampling rate makes it possible to perform oversampling which can increase the resolution of the data while still allowing for an effective sampling rate of >1 MS/s. The sampling hardware streams the data to a computer for further processing using a 1 Gb Ethernet connection. A Master FPGA simulator is also available for stress testing of the data processing pipeline (see Figure 6). The Master FPGA simulator is connected using a 100GB fibre optical link to a server, which is to be used for testing of high bandwidth streaming of data.

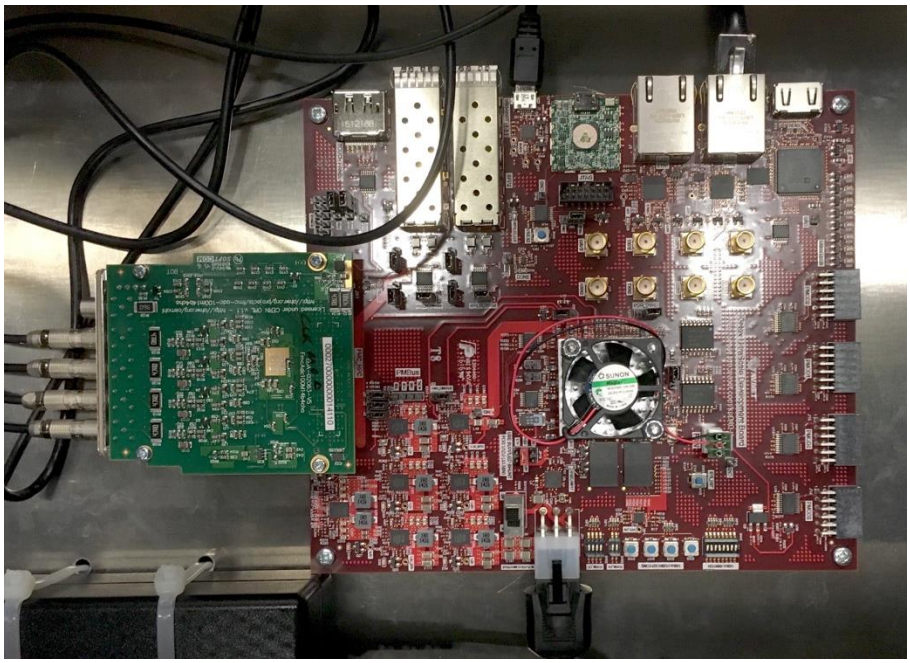


Figure 5. An FPGA-based detector readout electronics prototype used for timestamped sampling of an analogue signal.

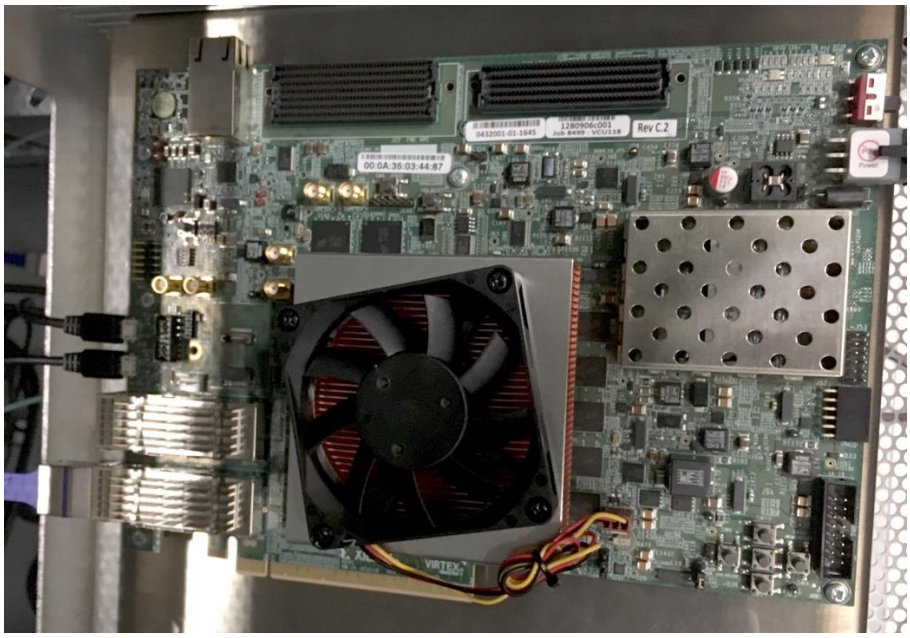


Figure 6. A Master FPGA simulator (i.e. data generator) for stress testing of the processing capabilities of general purpose computing hardware.

8 Comparison of Timestamping Strategies

Given the requirements and available hardware listed in the previous sections, three methods have been devised for sampling and timestamping of analogue data as a part of this project. These three methods are described below.

8.1 Timestamping in the Sampling Hardware

Sampling hardware that also supports timestamping of the samples still requires that the internal sample time clock counter of the FPGA is synchronised with that of the timing hardware. This is accomplished by making the clock signal of the timing hardware available to the FPGA for use by its own internal counter. The clock counter of the timing hardware is also periodically synchronised with the clock counter of the sampling FPGA. This is the method used by the ESS Detector Readout Electronics.

As the analogue signal is sampled, the data is moved to the next application for processing and long term storage. The movement of command signals, data and clock signals is illustrated in Figure 7.

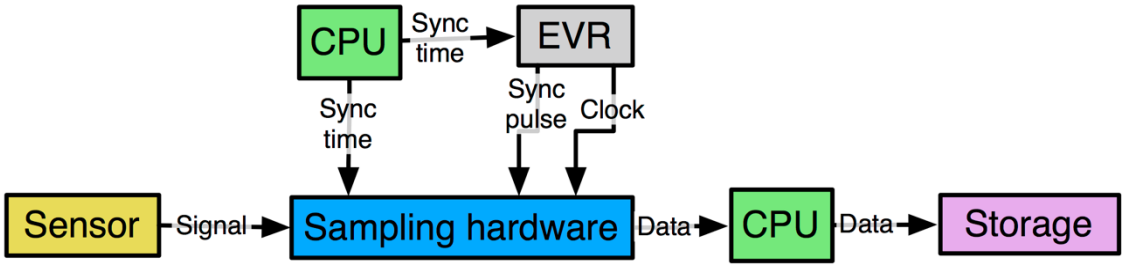


Figure 7. The components and the connections between them when using sampling hardware that also has the capability of timestamping the sample data. Note that an external CPU is tasked with making sure that the sampling hardware and EVR clocks are synchronized.

As the timestamped sampling in the set-up illustrated in Figure 7 is done entirely in the FPGA, the processing by the CPU is relatively light. This division of labour between components is illustrated in Figure 8.

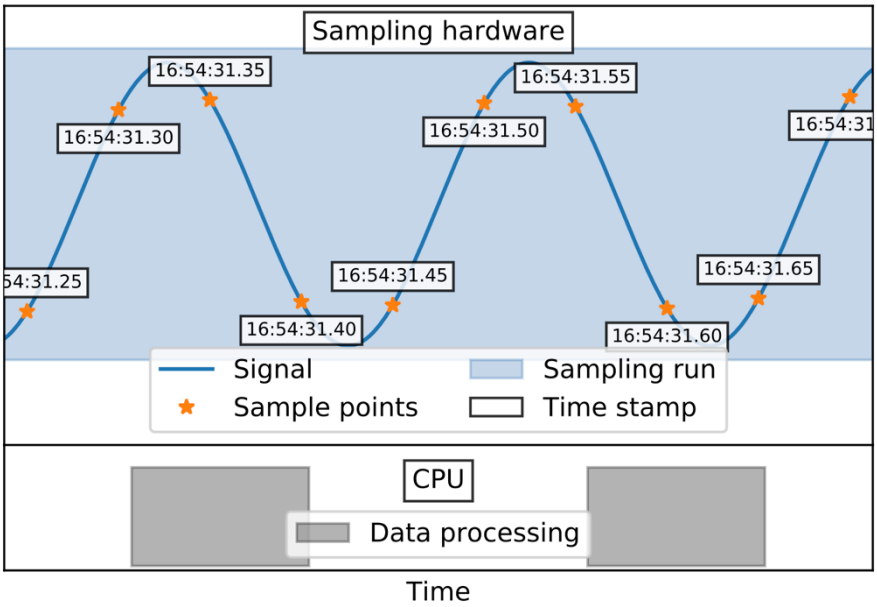


Figure 8. An illustration of the responsibilities of the different components (shown in Figure 7) when doing a sampling run using sampling hardware that is also capable of timestamping the data.

8.2 Timestamping in Software

Assuming that the data rates are sufficiently low (which is the case when sampling at 1 MS/s) timestamping can be done in software by sampling the TTL event output of the timing hardware simultaneously with the analogue signal. As the timestamps of these TTL pulses can be retrieved from the timing hardware, the timestamps of each sampling point can be re-created in software. The set-up for doing timestamping in software is illustrated in Figure 9. The sampling process when using this set-up is illustrated in Figure 10.

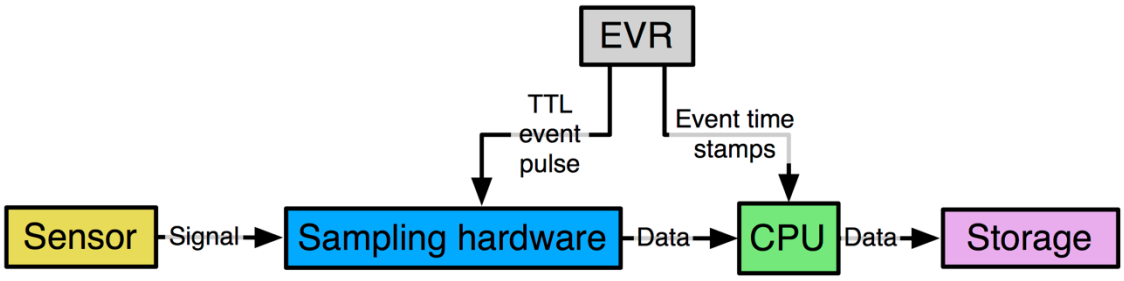


Figure 9. The hardware and connections required when doing timestamping in software. Note that the TTL event pulse is sampled as an analogue signal simultaneously with the actual analogue input signal.

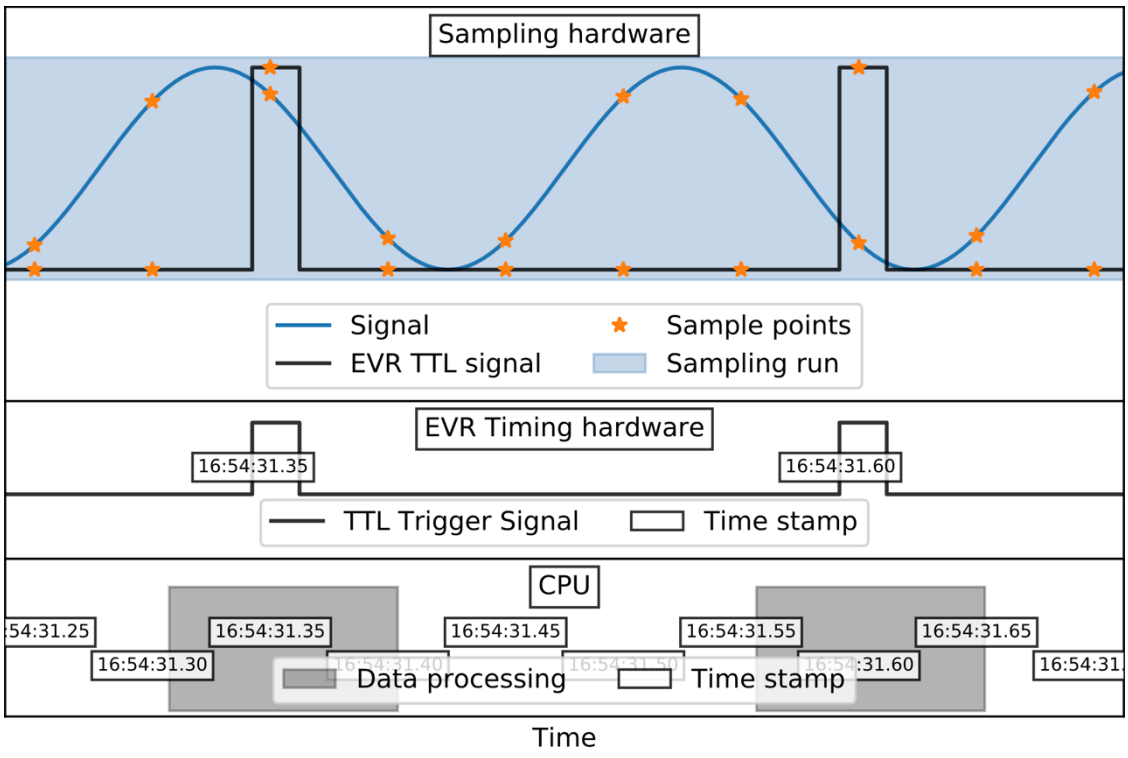


Figure 10. Division of labour between components when sampling and timestamping of the sampled data, when the latter is done in software as illustrated in Figure 9.

This method of timestamping is appropriate for the IFC14xx-based hardware solution due to its lack of FPGA support for timestamping. However, it should be noted that this method requires that one ADC channel is used for sampling of the TTL output of the timing hardware. This further limits the total throughput of the system as continuous sampling using the IFC14xx is bandwidth-limited to 8 MB/s. The limitations of the drivers that are currently available for the IFC14xx system (as mentioned in the hardware section) should also be considered. Furthermore, the CPU on the IFC14xx may not have enough processing power for doing

software-based timestamping, although this could be solved by processing the data with a more powerful CPU outside the rack.

8.3 Triggered Sampling

At low sampling rates, an approximation of continuous sampling can be used. If the sampling rate is low enough, the CPU will be able to retrieve the data from the last sampling run as well as the corresponding timestamps within a time span that is shorter than the time between two sample points (e.g. 1 ms at a sampling rate of 1 kS/s). The set-up for doing this is similar to that of the previous alternative, except that the TTL signal from the timing hardware is used to trigger sampling instead (see Figure 11).

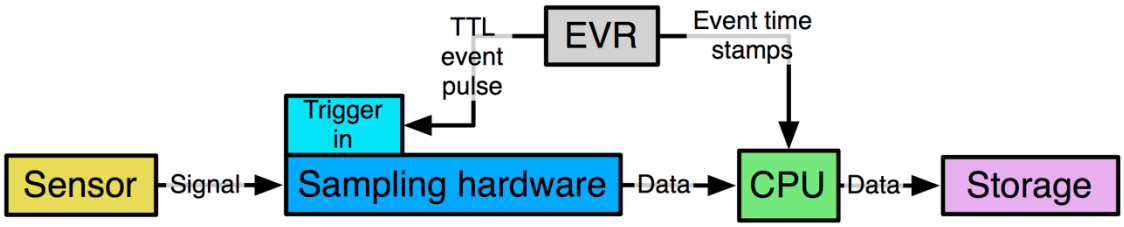


Figure 11. The set-up used for approximating continuous sampling by triggering sampling runs using the timing hardware.

Figure 12 illustrates how the CPU transfers the data produced from a sampling run before the next one is started. An important consideration when using this type of sampling is that latencies are short. The version of Linux running on the IFC14xx CPU card will have some RTOS functionality, giving maximum latencies of around 100 µs. The RTOS functionality will enhance this method of sampling, though tests will be required in order to determine maximum continuous sampling rates.

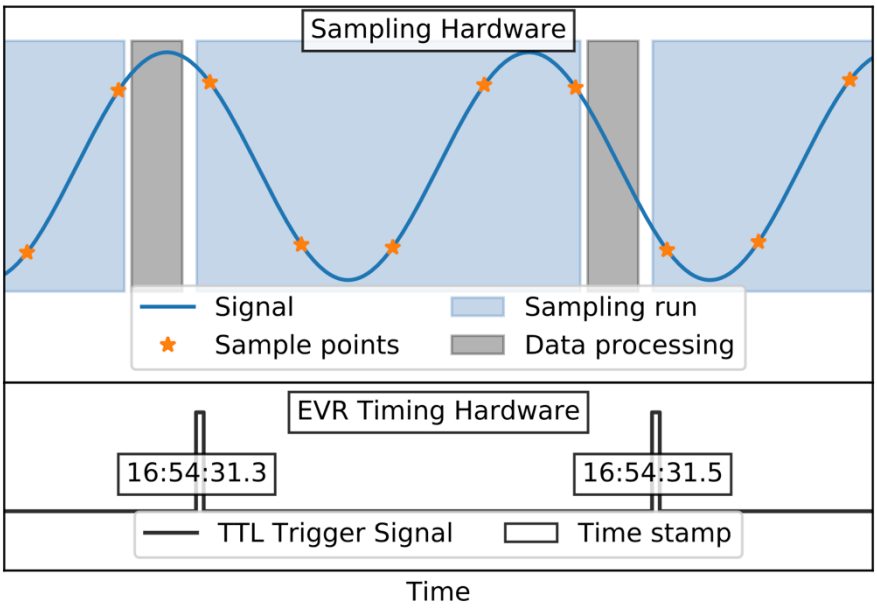


Figure 12. An illustration showing the processing of data and timestamps when an approximation of continuous timestamped sampling is used. Figure 11 uses the division of labour shown here.

9 Integration into the Data Acquisition Pipeline

Given the requirements mentioned in this document, the available hardware and the alternatives for doing timestamping, three methods for collecting the data and storing it to file have been devised and prototyped. These solutions are discussed in the following sections.

9.1 Using the areaDetector EPICS Framework

Given that an EPICS areaDetector (see [8], [9]) driver exists for the sampling hardware, one viable solution is to use its plugin system as a first step for collecting the data. The areaDetector framework supports a plugin system where the plugins loaded by an areaDetector driver gain access to the data collected by that driver. The data is shared between the driver and its plugins, using the structure illustrated in Figure 13.

```
struct NDArray {
  int               uniqueId;
  double            timeStamp;
  epicsTimeStamp    epicsTS;
  int               ndims;
  NDDimension_t     dims[10];
  NDDataType_t      dataType;
  size_t            dataSize;
  void             *pData;
  NDAttributeList  *pAtrributeList;
};
```

Figure 13. Data structure used to store area detector data when passed from an area detector driver to a plugin.

An areaDetector plugin, called *ADPluginKafka*, has been developed. It serialises data produced by an areaDetector driver and transmits the data via an Ethernet connection to a data aggregator. A sister software project was also developed in conjunction with the plugin, called *ADKafka*, which implements an areaDetector driver that receives data from a data aggregator. These two software projects are collected in the *ad-kafka-interface* code repository which is available on GitHub [10].

Aggregation of data from *ADPluginKafka* (and in general) is provided by the Apache Kafka open source project. Use of this software is discussed in more detail in the BrightnESS D5.1 report [11], though a short version of it follows below.

Apache Kafka[12] is a publish-subscribe messaging system built as a distributed commit log. It is designed to be scalable and robust to hardware failures. Persistence of the messages is configurable and the throughput performance is high. Kafka relies on a publisher, called *producer* in Kafka terminology, to publish data as messages to a Kafka cluster, which is comprised of one or more servers referred to as *brokers*. The *ADPluginKafka* project and most other projects developed at ESS which implement communication with an Apache Kafka broker, do this via an open source library called *librdkafka* [13].

The *ADPluginKafka* project and its communication with a Kafka broker has been performance tested using simulated data. In these tests, the transfer of data was limited by the write speed of the SSD used by the Kafka broker (in this case ≈120 MB/s). Hence the use of this project for transferring analogue data with the bandwidth requirements given earlier in the text will not be a problem.

Kafka does not define a messaging format. Messages have to be serialised using some method and the FlatBuffers [14] library was chosen for this, as discussed in the BrightnESS D5.1 report [11]. FlatBuffers is a cross-platform serialisation library, originally created at Google for performance-critical applications. It uses a simple interface description language

to describe serialisation schemas which are then used to represent complex data in a binary buffer.

The serialisation schema developed to be used by the *ADPluginKafka* project closely reflects that of the data structure provided to areaDetector plugins. This schema is shown in Figure 14.

```
enum DType: byte { int8, uint8,
    int16, uint16, int32, uint32,
    float32, float64, c_string }

table NDAttribute {
    pName: string;
    pDescription: string;
    pSource: string;
    dataType: DType;
    pData: [ubyte];
}

struct epicsTimeStamp {
    secPastEpoch: int;
    nsec: int;
}


table NDArray {
    id: int;
    timeStamp: double;
    epicsTS: epicsTimeStamp;
    dims: [ulong];
    dataType: DType;
    pData: [ubyte];
    pAttributeList: [NDAttribute];
}
```

Figure 14. The FlatBuffers schema used by the ADPluginKafka project for serialising areaDetector data.

For experimental data that is to be stored, a software project is underway in BrightnESS task 5.3. This activity is detailed in the report titled “Beta-version data aggregator software” [15]. The file writing application developed as a part of that activity receives data from a Kafka broker. This data is de-serialised and written to an HDF5 file using the NeXus data format [16].

As a part of the activity, the file writer was extended to write data serialised by the *ADPluginKafka* plugin. The NXlog NeXus class was selected for storing this data and Figure 15 shows the parts of the NXlog definition used by the file writing code that was also developed. Because the data does not contain timestamps for individual samples, only the timestamp of the sampling run is stored (in the *time* data set). The code for storing this data is available in the NeXus file writing code repository [17].

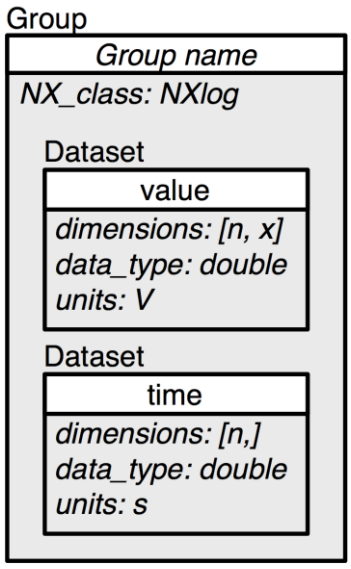


Figure 15. The group and data set structure used by the NeXus file writer for writing area detector data as described by Figure 14.

Although the *ad-kafka-interface* project provides a quick and easy solution for moving EPICS areaDetector data to a Kafka cluster, it requires an areaDetector driver which implements the communication with the sampling hardware and the timing hardware. No such driver has been developed so far, though there is a possibility that this will be done for IFC14xx-based hardware. Note that the data structure used by the areaDetector framework (shown in Figure 14) was not designed for storing analogue sampling data with a timestamp for each individual sample point. This can be solved by storing the sample points using e.g. the double precision floating point data type and interleaving the sampling data with the timestamps for each sample point. The structure was also not designed for storing meta data related to the set-up of the sampling hardware though this can be done using the *pAttributeList* field.

9.2 Using the EPICS Base Libraries

An alternative to using the areaDetector framework is to write an EPICS driver using the base libraries provided by EPICS. This method provides greater control of the data structures in use, enabling the storage of e.g. timestamps for every individual sample. For the purpose of prototyping this alternative, an existing EPICS driver was modified to provide better support for timestamped sampled data. This proof-of-concept driver exposes the data structure illustrated in Figure 16. The code for this modified driver is available on Bitbucket in the *m-epics-ifcdaq* repository [18].

```
ess:fsd/ifcdaq:1.0
    double[] value
    double[] time
    time_t timeStamp
        long
    secondsPastEpoch
        int nanoseconds
        int userTag
    int uniqueId
```

Figure 16. A proof of concept data structure for providing timestamped data over EPICS.

Moving data made available over EPICS to an Apache Kafka broker is done using the EPICS Forwarder project. The EPICS Forwarder is developed as a part of Brightness task 5.3. More information about it can be found in the report “Beta-version data aggregator software” [15] and the EPICS Forwarder code can be found on GitHub [19].

For this specific project, the EPICS Forwarder functionality was extended in order for it to support serialisation of the data structure illustrated in Figure 16. Likewise, a file writing module was added to the NeXus file writer for dealing with this FlatBuffers schema. The file writing module implements file writing using the NeXus NXlog class and as timestamps for individual samples are provided, these are stored as well. This file structure is illustrated in Figure 17.

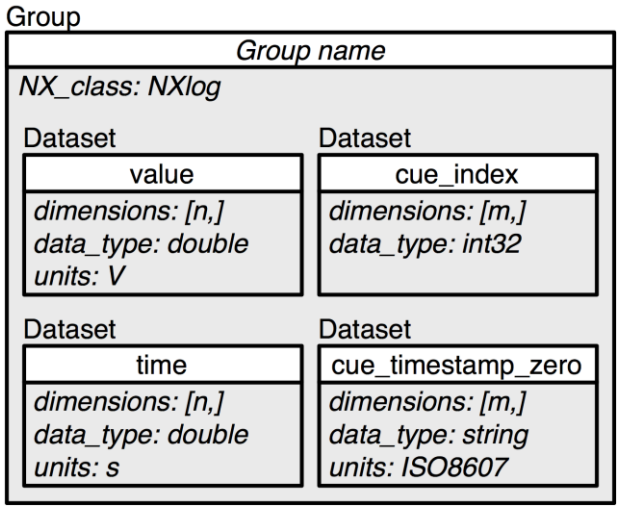


Figure 17. The group and data set structure used by the NeXus file writer for writing timestamped sampling data structured as shown in Figure 16.

As EPICS does not provide any guarantee of data delivery, the probability of data loss would be higher using this method if the EPICS driver does not also implement Apache Kafka producer functionality. Making the timestamped data available over EPICS (instead of only Apache Kafka) would be useful for monitoring, debugging and setting up experiments. The development of a driver using the base EPICS libraries will be one outcome if an IFC14xx-based solution, where timestamping is done in software, is used.

9.3 Using a Stand-alone Kafka Producer

An option for reading out the data from the sampling hardware is to use software which does not depend on EPICS. For transmission of data, support for Apache Kafka would be required instead however. This is the case for the software which will implement readout of the ESS Detector Readout Electronics.

An initial version of this software already exists and has been developed as part of BrightnESS task 5.1 [20]. It has been tested successfully with data from neutron detectors developed by the ESS Detector Group, showing a high data throughput. This software is easy to modify to also accept timestamped sampled data, although this has not yet been done due to a lack of hardware available for testing.

Due to fewer dependencies, software that does not require EPICS will have a shorter development cycle. However, this software will then also be less useful to scientists using the sampling hardware as it will be harder to access the data. Though this option is intended for a solution based on the ESS Detector Readout Electronics, it would be possible to adapt it for a IFC14xx-based solution as well.

10 Conclusion and Roadmap

The integration of the fast sample environment data into the main data acquisition chain is well developed and tested. Any hardware solution that will be picked for sampling and timestamping can be accommodated with no or very few additional modifications.

The relatively low data rates produced per analogue channel (i.e. a maximum of ≈61 MB/s) are simple for a modern computer system to process when this is done in an application written in a compiled language (e.g. C/C++). Even when 8 channels are present, the storage medium will only need to write data at a maximum of 250 MB/s. Storage of this amount of data does not pose a problem, as consumer grade SSDs and HDDs with sequential write speeds on the order of hundreds of MB/s are already available. Software prototypes have already well exceeded the data processing rates required.

On the hardware side, as expected, no hardware with working firmware and drivers exists with the capability of taking MRF timestamped measurements. Sampling hardware with support for non-continuous timestamped sampling of analogue signals is available at the time of writing with the IFC14xx-based hardware. However, drivers for continuous sampling require additional work. Another solution is to adapt ESS Detector Readout hardware for ADC sampling of analogue signals. As both hardware alternatives for fast sampling of analogue data are undergoing hardware and software development, they are both being considered as viable alternatives in the short term. As the required functionality is implemented, development will be focused on one of the hardware solutions.

Currently, the main risk is the development and availability of the hardware, firmware and drivers. Hence the strategy to arrive at a working demonstrator will be adapting existing software prototypes for hardware as it becomes available. This should require limited effort and will allow full evaluation of the hardware in a timely manner. Due to additional low-level driver work required for a IFC14xx-based MRF timestamped continuous sampling solution, the preference would be to make use of the ESS Detector Readout card instead. Adding support for continuous sampling there, is more straight forward and makes better use of existing collaborations. If this is met with unexpected problems, we can still refocus on the IFC14xx drivers for continuous sampling.

11 References

[1] F. Piscitelli, “Neutron detectors for ESS instrument projects.” August-2017 [Online]. Available: https://confluence.esss.lu.se/display/DG/Neutron+Detectors+for+ESS+Instrument+Projects
[2] G. Granroth, “Private communication.” August-2017.
[3] M.-R. Finland, *VME event generator (vme-evg-230) technical reference*. 2009 [Online]. Available: http://www.mrf.fi/dmdocuments/EVG-230TREF-002.pdf
[4] M.-R. Finland, *VME event receiver (vme-evr-230) and vme event receiver with cml outputs (vme-evr-230RF) technical reference*. 2009 [Online]. Available: http://www.mrf.fi/dmdocuments/EVR-230TREF-005.pdf
[5] J. Pietarinen, “MRF timing system.” Feb-2008 [Online]. Available: http://www.mrf.fi/pdf/presentations/MRF.CERN.Feb2008.pdf
[6] IOxOS, “IFC1410 qoriq t2081 & kintex ultrascale amc: Building a comprehensive mtca.4 ecosystem.” Apr-2017 [Online]. Available: http://www.ioxos.ch/images/pdf/01_datasheet/IFC_1410_DS_A3.pdf
[7] IOxOS, “ADC 3110/3111 – eight channel 16-bit adc.” Apr-2017 [Online]. Available: http://www.ioxos.ch/images/pdf/01_datasheet/ADC_3110_DS_A1.pdf

[8] A. Johnson, “EPICS home page.” August-2017 [Online]. Available: http://www.aps.anl.gov/epics/
[9] M. Rivers, “areaDetector: EPICS software for area detectors.” August-2017 [Online]. Available: http://cars.uchicago.edu/software/epics/areaDetector.html
[10] J. Nilsson, “Area detector Kafka interface.” August-2017 [Online]. Available: https://github.com/ess-dmsc/ad-kafka-interface
[11] A. Mukai and T. Richter, “Deliverable report: D5.1 design report data aggregator software,” technical report, Aug. 2016 [Online]. Available: https://brightness.esss.se/sites/default/files/deliverables/
[12] A. S. Foundation, “Apache Kafka website.” August-2017 [Online]. Available: http://kafka.apache.org
[13] M. Edenhill, “Librdkafka - The Apache Kafka C/C++ Client Library.” August-2017 [Online]. Available: https://github.com/edenhill/librdkafka
[14] Google, “FlatBuffers website.” August-2017 [Online]. Available: https://google.github.io/flatbuffers/
[15] A. Mukai and T. Richter, “Deliverable report: D5.3 beta-version data aggregator software,” technical report, Jul. 2017.
[16] N. I. A. Committee, “The NeXus common data format for neutron, X-ray, and muon science.” 2017 [Online]. Available: http://nexusformat.org
[17] D. Werder, M. Brambilla, and M. D. Jones, “NeXus file writer code repository.” August-2017 [Online]. Available: https://github.com/ess-dmsc/kafka-to-nexus
[18] “IFCDAQ code repository.” August-2017 [Online]. Available: https://bitbucket.org/europeanspallationsource/m-epics-ifcdaq/branch/dmsc_epicsv4_experiment
[19] D. Werder, “EPICS forwarder code repository.” August-2017 [Online]. Available: https://github.com/ess-dmsc/forward-epics-to-kafka
[20] M. Shetty and M. Christensen, “Deliverable report: D5.2 processing choices for detector types,” technical report, May 2017.