SWATCH: Common software for controlling and monitoring the upgraded Level-1 trigger of the Compact Muon Solenoid experiment

S. Bologna, G. Codispoti, G. Dirkx, L. Kreczko, C. Lazaridis, E. Paradas, A. Rose, A. Thea, T. Williams

***Abstract*—The Large Hadron Collider at CERN restarted in 2015 with a higher centre-of-mass energy of 13 TeV. The instantaneous luminosity is expected to increase significantly in the coming years. An upgraded Level-1 trigger system has been deployed in the Compact Muon Solenoid experiment, in order to maintain the same efficiencies for searches and precision measurements as those achieved in the previous run. This system consists of the order of 100 electronics boards connected by the order of 3000 optical links, which must be controlled and monitored coherently through software, with high operational efficiency. In this paper, we present the design of the software framework that is used to control and monitor the upgraded Level-1 trigger system, and experiences from using this software to commission the upgraded system.**

I. Introduction

Since 2009, the Compact Muon Solenoid (CMS) experiment [1] has been analysing the particles produced from the proton–proton and heavy ion collisions of the Large Hadron Collider (LHC), in order to investigate the predictions of the Standard Model of particle physics, and search for physics beyond the Standard Model. The Level-1 trigger system selects 100 kHz of the most interesting collisions from the 40 MHz proton–proton bunch crossing rate [2]. This Level-1 accept decision must be made within 3.8 μs of each bunch crossing, using coarse data from the calorimeters and muon detectors, whilst the full-resolution data is held in pipeline memories in the front-end electronics. The full-resolution data for the events selected by the Level-1 trigger is read out of the custom detector electronics into a computer farm, which implements further selection algorithms in order to reduce the event rate to the order of 100 Hz for archival storage.

The legacy Level-1 trigger system is composed of approximately 4000 data processor boards, of several custom application-specific designs [2]. These boards have been controlled and monitored by a medium-sized distributed system of over 40 computers and 200 processes. The legacy trigger was organised into several subsystems. Only a small fraction of the control and monitoring software was common between the different subsystems; the configuration data was stored in a database, with a different schema for each subsystem. This large proportion of subsystem-specific software resulted in high long-term maintenance costs, and a high risk of losing critical knowledge through the turnover of software developers in the Level-1 trigger project.

After a two-year shutdown, the LHC restarted collisions in 2015, with higher proton–proton centre-of-mass energy (13 TeV) and increased instantaneous luminosity. So the Level-1 trigger system has been upgraded during 2015 and early 2016, in order to improve its efficiency for searches and precision measurements, compared with the previous run [3]. The upgraded system is composed of a set of general-purpose boards, that follow the MicroTCA specification, connected by high-speed serial optical links, resulting in a more homogeneous system. This system will contain the order of 100 boards connected by 3000 optical links, which must be controlled and monitored coherently. The data flow within the upgraded Level-1 trigger system is shown in Fig. 1. The experiment's off-detector electronics transmit reduced-granularity data (trigger primitives) from the calorimeters and muon detectors to the trigger electronics over optical links. The trigger primitive data from different fiducial regions of the muon detectors — the barrel, end-cap, and overlap regions — are processed in three separate track finder subsystems; the resulting three sets of muon track candidates are combined in the micro-Global Muon Trigger. The calorimeter data is processed to reconstruct electrons, tau leptons and jets in a separate data path, consisting of two layers of electronics: a first layer that time multiplexes the data, and a second layer in which each electronic board processes the data from the entire calorimeter for every 9th event, in a round robin sequence. The particles reconstructed in the calorimeter and muon pathways are then combined in the micro-Global Trigger (μGT), which takes the decision of whether or not a proton–proton collision is a sufficiently interesting event, by applying kinematic and quality selection criteria on the received particle candidates.

In this paper, we present the design of the software that is used to control and monitor the upgraded Level-1 trigger system, and experiences from using this software to commission the upgraded system.

Manuscript received May 30, 2016.

S. Bologna is with University of Milan-Bicocca and INFN, Milano, Italy.
G. Codispoti is with University of Bologna and INFN, Bologna, Italy, and CERN, Geneva, Switzerland.
G. Dirkx is with CERN, Geneva, Switzerland.
L. Kreczko is with University of Bristol, Bristol, U.K.
C. Lazaridis is with Dept. of Physics, University of Wisconsin-Madison, U.S.A.
E. Paradas is with University of Ioannina, Greece.
A. Rose is with Imperial College, London, U.K.
A. Thea and T. Williams are with STFC Rutherford Appleton Laboratory, Harwell Oxford, U.K.
E-mail: t.williams@cern.ch

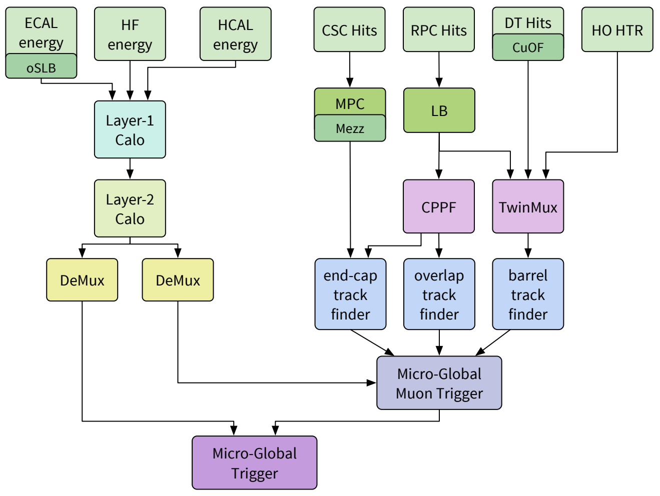


Fig. 1. Diagram illustrating the overall architecture of the Level-1 trigger system. The black arrows represent the flow of trigger primitive data and local trigger objects through the system. The back-end electronics for the detectors (electromagnetic calorimeter, hadronic calorimeter and muon detectors) are shown at the top of the diagram; the boxes below represent the trigger electronics that analyses the calorimeter and muon detector trigger primitive data, on the left and the right respectively. Each box corresponds to one or more crates of electronics boards.

II. Requirements on the control and monitoring software

The Level-1 trigger system must be controlled and monitored coherently through software, with high operational efficiency. The control and monitoring software must have reliable and predictable behaviour under all circumstances, and must fulfil the trigger system's operational requirements both in its initial commissioning period, and for the next several years.

A. *Functional requirements: Control*

The Level-1 trigger system must be configured through the online software; during configuration, the electronic boards' programmable logic must be set up correctly — for example, in each board:

- The clock infrastructure must be configured correctly.
- Optical input ports must be configured, and input data aligned with appropriate delays.
- Data processing algorithm parameters (such as energy thresholds, and calibration factors) must be set to appropriate values according to the current running conditions of the experiment and the LHC.

The online software must be able to configure the trigger hardware in multiple scenarios:

1) Configuration of the entire trigger system by the shift crew, along with all of the experiment's detectors, for normal data-taking runs. In this scenario, the trigger will be controlled via SOAP (Simple Object Access Protocol) messages sent from the top node of the experiment's run control hierarchy. To ensure traceability of the system's configuration during data-taking periods, the values of configuration parameters used in each global run must be stored in a database.
2) Configuration of the entire trigger system by experts, without the experiment's detectors, through a graphical user interface (GUI).
3) Configuration of individual subsystems by experts, without interfering with the operation of other subsystems. This mode of operation is essential, in particular during the commissioning period, and during LHC downtime, for example in order to independently validate the functionality of new firmware or software in that subsystem.

After configuring the hardware, the online software should perform appropriate checks to ensure that the operation has been completed successfully. In case of failure — for example, inability to lock onto the clock from an external source, or errors in input ports — the online software must report back detailed messages that allow the source of problem to be identified promptly either by the shift crew, or by on-call experts.

Finally, the configuration status of all subsystems must be summarised in a single graphical interface.

B. *Functional requirements: Monitoring*

Once the Level-1 trigger is configured for a data taking run, if any problem occurs in the trigger's data processing chain — for example, the failure of an optical transmitter — then it must be detected and fixed in a timely manner, in order to minimise the experiment's downtime. As a result, the online software must periodically query monitoring registers in the trigger electronics, and promptly alert the shift crew when a problem is detected. Furthermore, the graphical interfaces must allow rapid access to sufficient in-depth information in order for the shift crew, or on-call experts, to be able to fix the problem quickly and thereby minimise downtime.

In addition, the values of some of the monitoring data read from the hardware must be periodically stored in a database. A small amount of this archived monitoring data is critical, i.e. the experiment should not continue to run if the data storage mechanism fails. For example, the prescale factors used by the μGT subsystem must be recorded every 23 seconds. It is also useful to periodically archive a large proportion of the monitoring data values from each subsystem, for post-mortem analysis; however, this more extensive archiving of monitoring data is not critical to the experiment's operation in the short term.

C. *Non-functional requirements*

Commissioning The implementation of the control and monitoring software must be sufficiently flexible such that the developers are able to rapidly add new features, and improve existing functionality, based on evolving operational needs and experience gained in the commissioning period. In order to minimise downtime during this short commissioning period, such rapid improvement of the online software must be achievable without introducing bugs or regressions into existing functionality.

Testing The online software should be designed so that as much as is practical of its functionality can be validated by unit tests or integration tests without access to the trigger hardware.

Personpower Any common control and monitoring software framework should be designed to minimise the required personpower for both development and maintenance of the online software of the Level-1 trigger upgrade project as a whole.

Maintainable lifetime The upgraded Level-1 trigger system could be operating for several years; its control and monitoring infrastructure should have same maintainable lifetime.

Existing control and monitoring libraries The online software should make use of the existing functionality provided by the XDAQ [4] and Trigger Supervisor (TS) [5] libraries, in particular the infrastructure for distributed control and monitoring systems, running programs as services and implementing web servers.

III. Common processor and system models

In order to minimise the personpower required for the development of the Level-1 trigger online software overall, the common online software framework should provide a subsystem-agnostic interface that reflects the common structure of the subsystems in the upgraded Level-1 trigger. A uniform in-depth graphical interface for controlling and monitoring subsystems will also simplify the training of the shift crew, on-call experts, and other operation personnel. Therefore, in the early stages of designing this common online software framework, significant effort was dedicated to identifying an in-depth generic description of the upgrade hardware that is applicable to all subsystems.

A. Common processor model

In each subsystem, the data processing nodes are all AMCs (Advanced Mezzanine Cards) following the MicroTCA specification [6]. Each AMC uses an FPGA (Field Programmable Gate Array) for processing logic, and transmits/receives the trigger data on high-speed serial optical links. The common structure of the logic in each data processing board is reflected in the common processor model, shown in Fig. 2. In this model, the firmware on each data processor board consists of:

- A TTC block, that receives the clock and fast (fixed-latency) control commands from TCDS (Trigger Control and Distribution System) [3], the experiment's central clock and trigger distribution system.
- Zero or more optical input ports
- Zero or more optical output ports
- An algorithm block, that processes the data received on the optical input ports into data for the optical output ports.
- A readout block, that sends data from the board's input and output buffers to the experiment's Data Acquisition (DAQ) system. This recorded I/O data is later used to validate that the trigger algorithm firmware is functioning correctly.

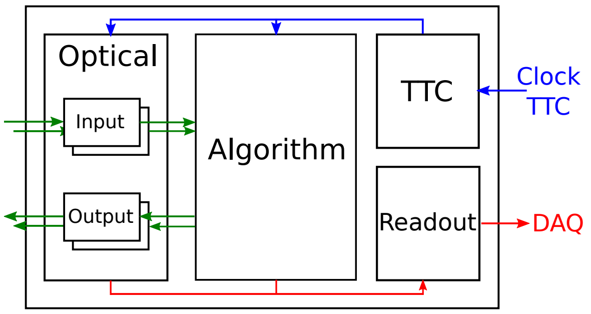


Fig. 2. Diagram showing the common model for data processor boards. The outer black box represents a single data processor board, and the other black boxes each represent a logical firmware block with distinct functionality. The green arrows represent the flow of the trigger primitive/object data that is sent and received over optical links. The blue arrows represent the paths of the clock signals and fixed-latency control commands. The red arrows represent the flow of readout data to the DAQ system.

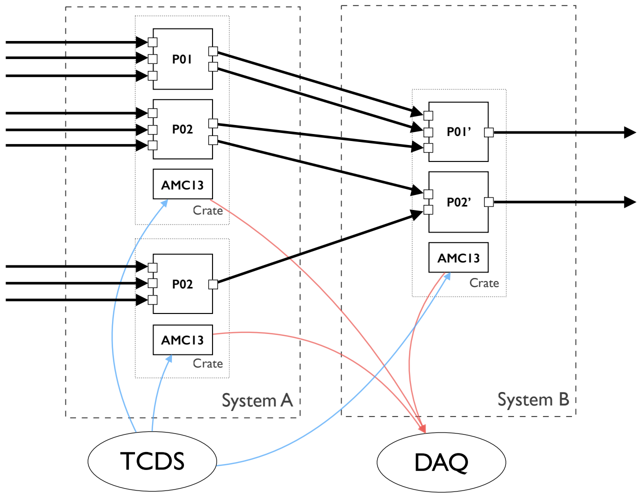


Fig. 3. Diagram showing the common model for the structure of systems of processor boards within the Level-1 trigger. Each black box represents a single electronics board, each grey box represents a MicroTCA crate, and each dashed box represents a trigger subsystem. The black arrows represent optical links carrying trigger primitives or trigger objects. The blue arrows represent the paths of the clock signals and fixed-latency commands sent by TCDS. The red arrows represent the flow of readout data to the DAQ system.

B. Common system model

Each subsystem consists of one or more processor boards housed in MicroTCA crates. A common module, the AMC13 [7], provides the clock, timing and DAQ services (i.e. TTC, TTS and DAQ backplane links) for all data processor AMCs within each MicroTCA crate. It distributes the clock and fast commands that it receives from TCDS onto high-speed point-to-point links on the MicroTCA backplane. In the opposite direction, it receives readout data over backplane links from each slot, and forwards that data into the CMS DAQ network. The common model for the structure of the trigger subsystems is shown in Fig. 3; in this abstract model, the AMC13 is referred to as a *DAQ-TTC manager*.

IV. The SWATCH framework

The SWATCH (SoftWare for Automating the conTrol of Common Hardware) framework provides a set of abstract C++ interfaces for controlling and monitoring the hardware of the upgraded Level-1 trigger system. Its design closely follows the common processor and system models, whilst remaining independent of the driver software.

Within SWATCH, the structure of each subsystem — i.e. the locations and names of processors, optical I/O ports and AMC13s, as well as their interconnections — are stored in subsystem-agnostic data structures. Each component from the common processor and system models is represented by an abstract interface class. The subsystem-specific software for controlling and monitoring each of these components is implemented in subsystem-specific classes that inherit from the common interface classes. The subsystem-specific classes for processors, systems and DAQ-TTC managers are registered and created using the factory pattern.

The SWATCH framework provides a generic interface for controlling each individual electronics board (both processors and DAQ-TTC managers), based on three concepts:

- *Command* A stateless action, represented by an abstract base class that is customised for subsystem-specific drivers by subtype polymorphism. Commands form the fundamental modular block of hardware control.
- *Command sequence* A stateless action, consisting of a list of commands that are executed in sequence.
- *Finite state machine (FSM)* Stateful actions; each FSM consists of a set of named states connected by transitions. Each transition between states is a list of commands that are executed in sequence. The structure of the standard run control FSM for processors and DAQ-TTC managers are automatically defined; their transitions are later populated with subsystem-specific command sequences in the constructors of subsystem-specific classes.

The parameters required for each command are registered through a generic interface; the values of parameters can be automatically retrieved by the framework, through the *gatekeeper* interface, an abstract representation of a pool of named configuration parameters. The gatekeeper interface was designed to be independent of the source of the configuration parameter values, in order to allow subsystem hardware to be easily configured and tested using file-based configurations, in scenarios when accessing the configuration database is not practical (for example, when testing hardware in a laboratory outside of CERN). During the commissioning period, this source-agnostic gatekeeper interface also ensured a seamless transition when the control software switched from using reading configuration parameters from files, to reading them from the database.

The SWATCH framework also provides a generic interface for monitoring the status of each board, based on two concepts:

- *Metric* Represents an individual item of monitoring data that is read from the hardware. Each metric can have associated error/warning conditions, that determine whether the metric's current value results in an error/warning monitoring state.
- *Monitorable object* Can contain metrics and other monitorable objects as children. The monitoring state of a monitorable object is the combined status of all its child metrics and monitorable objects. Each of the common firmware components within processors and DAQ-TTC managers (for example, optical I/O ports, TTC block, readout block) are represented in SWATCH as a monitorable object.

The framework allows both common metrics to be registered in the abstract interface classes, and additional subsystem-specific metrics to be registered in subsystem-specific classes.

A comprehensive suite of unit and integration tests for the SWATCH framework have been developed; these were implemented using the BOOST unit test framework, and are run regularly during development. We have found these unit and integration tests to be essential for quickly developing new features in the commissioning phase, without suffering from downtime due to bugs or regressions. Notably, since the SWATCH framework is hardware-agnostic, the framework's entire functionality can be tested without access to any subsystem's hardware; instead the unit and integration tests use dummy classes that represent processors and DAQ-TTC managers locally within computer memory.

V. Distributed control and monitoring

The SWATCH framework described in the previous section provides a C++ API (Application Programming Interface) for controlling and monitoring hardware whose structure matches the common processor and system models, whilst allowing the configuration sequence and the mechanism for retrieving monitoring data to be defined by each subsystem as required. However, the online software must also provide a graphical user interface that shows the configuration status and monitoring status of each subsystem. In addition, each subsystem must be integrated into the CMS run control hierarchy so that the trigger system can be configured coherently by the shift crew, in global runs, along with all of the experiment's detectors.

The Trigger Supervisor cell [5] is a XDAQ application that contains a generic interface to register stateful and stateless actions that can be initiated through a graphical user interface (webpages), or by SOAP messages received over the network — for example, from the CMS run control applications or from other TS cells. In the legacy trigger system, each subsystem was controlled by a unique hierarchy of cells, since each crate of electronics could only be directly controlled and monitored from a single computer. However, the majority of the hardware in the upgraded trigger system is controlled and monitored using IP-based communication protocols, and hence the host computers for online software applications can be de-localised with respect to the hardware. Therefore, to simplify the operation of the entire upgraded trigger system, and the independent operation of individual subsystems, each subsystem is controlled and monitored by a single TS cell. Through the SWATCH framework's subsystem-agnostic API, we have been able to develop subsystem-agnostic implementations for a significant fraction of the cell's hardware-agnostic functionality, such as the network-based run control

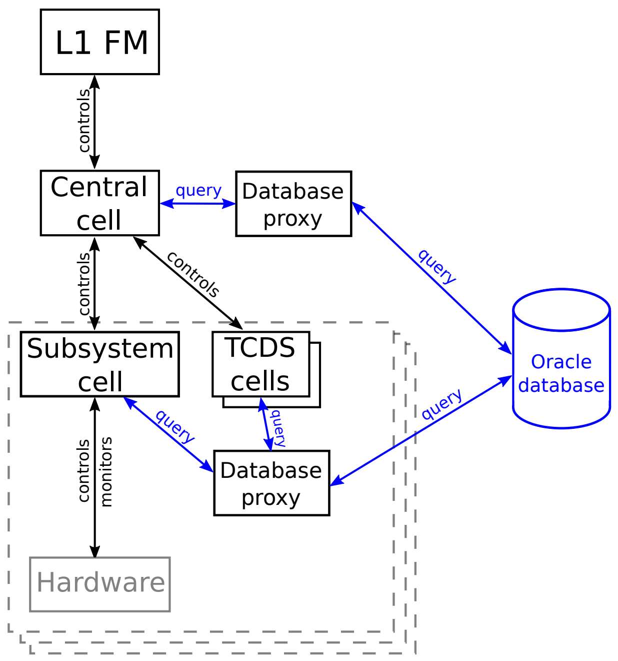


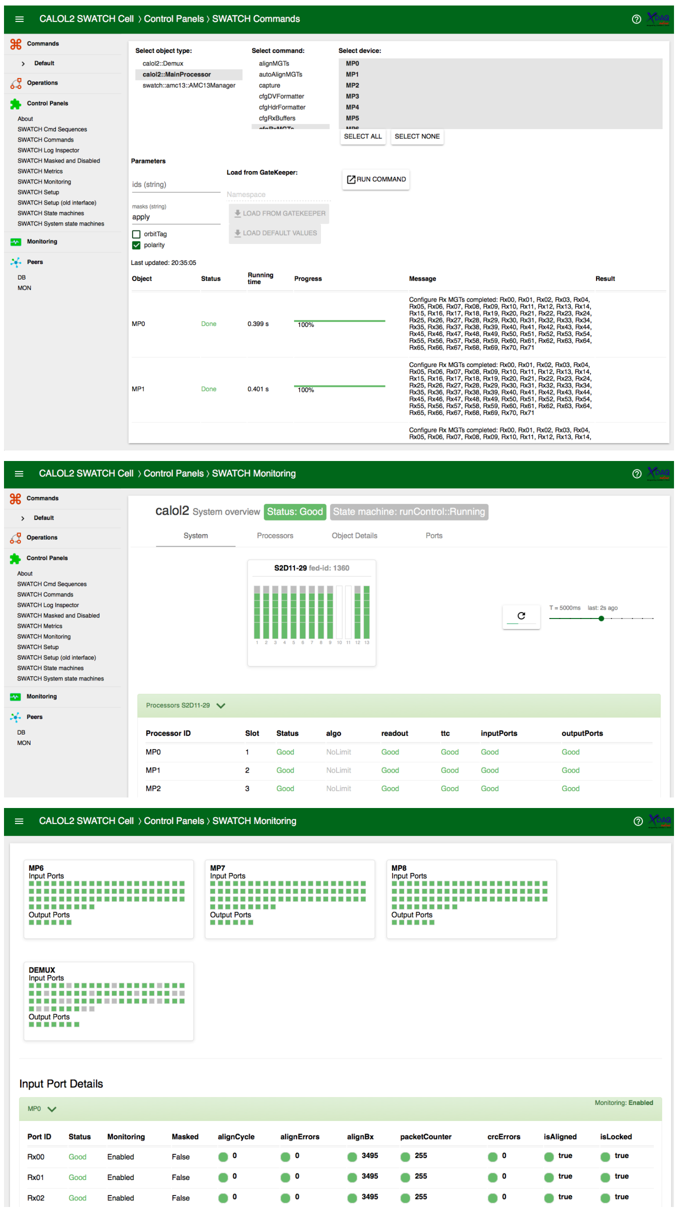
Fig. 4. A diagram illustrating the hierarchy of applications and services that are used to configure the upgraded Level-1 trigger system.

FSM, threads that periodically collect monitoring data from the hardware, and the retrieval of configuration parameters from database. The configuration of all subsystems for global running is coordinated by another TS cell, the *central cell*. The central cell is in turn controlled by the Level-1 trigger's Function Manager, a java application that lies in the 2nd layer of the central CMS run control hierarchy. The full run control hierarchy for the upgraded Level-1 trigger system is shown in Fig. 4. These applications have been successfully controlling and monitoring the upgraded Level-1 trigger since February.

A. Graphical user interface

The C++ API of the SWATCH framework both reflects the common processor and system model, and exposes the common and subsystem-specific monitoring/control primitives through subsystem-agnostic interfaces. As a result, it was feasible to develop common graphical user interfaces for the subsystem TS cell that allow subsystem experts, on-call personnel and shift crew to (among other procedures):

- Control one or more boards by executing commands, sequences or FSM transitions, and then view the detailed status and results of those actions.
- View the monitoring status of an entire system, or of individual processors, I/O ports and DAQ-TTC managers.
- Inspect the current values of any user-defined subset of monitoring data (metrics) and their error/warning conditions.
- Plot the values of any user-defined subset of monitoring data (metrics) in real time.



Select object type: calol2::Demux, calol2::MainProcessor, swatch::amc13::AMC13Manager

Select command: alignMGTs, autoAlignMGTs, capture, cfgDVFormatter, cfgHdrFormatter, cfgRxBuffers

Select device: MP0, MP1, MP2, MP3, MP4, MP5

Parameters

ids (string)

masks (string): apply

☐ orbitTag
☑ polarity

Load from GateKeeper: Namespace

Last updated: 20:35:05

Object	Status	Running time	Progress	Message	Result
MP0	Done	0.399 s	100%	Configure Rx MGTs completed: Rx00, Rx01, Rx02, Rx03, Rx04, Rx05, Rx06, Rx07, Rx08, Rx09, Rx10, Rx11, Rx12, Rx13, Rx14, Rx15, Rx16, Rx17, Rx18, Rx19, Rx20, Rx21, Rx22, Rx23, Rx24, Rx25, Rx26, Rx27, Rx28, Rx29, Rx30, Rx31, Rx32, Rx33, Rx34, Rx35, Rx36, Rx37, Rx38, Rx39, Rx40, Rx41, Rx42, Rx43, Rx44, Rx45, Rx46, Rx47, Rx48, Rx49, Rx50, Rx51, Rx52, Rx53, Rx54, Rx55, Rx56, Rx57, Rx58, Rx59, Rx60, Rx61, Rx62, Rx63, Rx64, Rx65, Rx66, Rx67, Rx68, Rx69, Rx70, Rx71	
MP1	Done	0.401 s	100%	Configure Rx MGTs completed: Rx00, Rx01, Rx02, Rx03, Rx04, Rx05, Rx06, Rx07, Rx08, Rx09, Rx10, Rx11, Rx12, Rx13, Rx14, Rx15, Rx16, Rx17, Rx18, Rx19, Rx20, Rx21, Rx22, Rx23, Rx24, Rx25, Rx26, Rx27, Rx28, Rx29, Rx30, Rx31, Rx32, Rx33, Rx34, Rx35, Rx36, Rx37, Rx38, Rx39, Rx40, Rx41, Rx42, Rx43, Rx44, Rx45, Rx46, Rx47, Rx48, Rx49, Rx50, Rx51, Rx52, Rx53, Rx54, Rx55, Rx56, Rx57, Rx58, Rx59, Rx60, Rx61, Rx62, Rx63, Rx64, Rx65, Rx66, Rx67, Rx68, Rx69, Rx70, Rx71	
				Configure Rx MGTs completed: Rx00, Rx01, Rx02, Rx03, Rx04, Rx05, Rx06, Rx07, Rx08, Rx09, Rx10, Rx11, Rx12, Rx13, Rx14,	

calol2 System overview Status: Good State machine: runControl::Running

Processors S2D11-29

Processor ID	Slot	Status	algo	readout	ttc	inputPorts	outputPorts
MP0	1	Good	NoLimit	Good	Good	Good	Good
MP1	2	Good	NoLimit	Good	Good	Good	Good
MP2	3	Good	NoLimit	Good	Good	Good	Good

Input Port Details

MP0 Monitoring: Enabled

Port ID	Status	Monitoring	Masked	alignCycle	alignErrors	alignBx	packetCounter	crcErrors	isAligned	isLocked
Rx00	Good	Enabled	False	● 0	● 0	● 3495	● 255	● 0	● true	● true
Rx01	Good	Enabled	False	● 0	● 0	● 3495	● 255	● 0	● true	● true
Rx02	Good	Enabled	False	● 0	● 0	● 3495	● 255	● 0	● true	● true

Fig. 5. Some of the generic GUIs for controlling and monitoring the subsystems of the upgraded Level-1 trigger.

Some of the graphical user interfaces are shown in Fig. 5. These generic GUIs are used regularly by subsystem experts to control subsystems in standalone tests, and by shift crew or on-call personnel in order to rapidly diagnose problems observed during global running. The ability to implement these GUIs only once, using the subsystem-agnostic SWATCH API has significantly reduced the personpower required to develop online software during the commissioning period.

VI. Conclusions

The software framework (SWATCH) that provides a generic interface for controlling and monitoring the upgraded Level-1 trigger's hardware has been presented. This new framework has controlled and monitored the upgraded trigger system coherently and reliably since February. The design of this framework is based on common models of the data processor boards, and the subsystems. By defining a generic hardware description of significantly finer granularity, the SWATCH framework has enabled the creation of a more uniform graphical interface across the different subsystems compared with the legacy system, simplifying the training of the shift crew, on-call experts, and other operation personnel. A common database schema is now used to store configuration parameters and a description of the hardware in each subsystem. The architecture of online software services is now identical for all subsystems. The introduction of the SWATCH framework has increased the proportion of common online software in the Level-1 trigger project. In long term, this increased fraction of common software should reduce the maintenance costs and risk of losing critical knowledge through turnover of software developers.

Acknowledgment

We would like to thank the Science and Technology Facilities Council (STFC) for funding the U.K. contribution to this research.

References

[1] The CMS collaboration, "The CMS experiment at the CERN LHC," *Journal of Instrumentation*, vol. 3, no. 08, p. S08004, 2008. [Online]. Available: http://stacks.iop.org/1748-0221/3/i=08/a=S08004

[2] The CMS Collaboration, *CMS TriDAS project: Technical Design Report, Volume 1: The Trigger Systems*, ser. Technical Design Report CMS. Geneva: CERN, 2000.

[3] The CMS collaboration, "CMS Technical Design Report for the Level-1 Trigger Upgrade," CERN, Geneva, Tech. Rep. CERN-LHCC-2013-011, CMS-TDR-12, Jun 2013, http://cds.cern.ch/record/1556311.

[4] XDAQ developers, "XDAQ website," https://svnweb.cern.ch/trac/cmsos.

[5] I. Magrans de Abril, C. E. Wulz, and J. Varela, "Concept of the CMS trigger supervisor," *IEEE Trans. Nucl. Sci.*, vol. 53, pp. 474–483, 2006.

[6] The PCI Industrial Computer Manufacturers Group (PICMG), "MicroTCA: PICMG MTCA.0 R1.0 Short Form Specification," PICMG, Tech. Rep. PICMG MTCA.0 R1.0, Sep 2006, http://www.picmg.org/pdf/MicroTCA_Short_Form_Sept_2006.pdf.

[7] E. Hazen, A. Heister, C. Hill, J. Rohlf, S. X. Wu, and D. Zou, "The AMC13XG: a new generation clock/timing/DAQ module for CMS MicroTCA," *Journal of Instrumentation*, vol. 8, no. 12, p. C12036, 2013. [Online]. Available: http://stacks.iop.org/1748-0221/8/i=12/a=C12036