SEU handling in ECAL

Pedro Parracho

Pedro.ferreira.parracho@cern.ch

Abstract

This document will describe all that is related with SEU (token ring problem) in ECAL. It describes what the problem is, how it is detected and how it is dealt with in the software.

Conventions

Software states: **bold**
Software signals: *italic*
Hardware states: CAPITALIZED
Hardware signals: *CAPITALIZED ITALIC*
C++ code: `Fixed width font`

1 Introduction

CMS is going to implement a method to fix SEU during running.

ECAL had several occurrences [1] where many the data links failed simultaneously. It was observed that the data links were all in the same token ring; in these events either all or half of the links controlled by the affected token ring failed.

These events were attributed to SEUs and we can use the CMS SEU recovery procedure to recover these links.

1.1 Problem identified as SEU

From the available information 4 different types if problems were identified:

1. Single extra word (usually a first word only from FE), this problem is expected to be spurious and recoverable with *TTS_RESYNC*, that will clear the error detected in the DCC and realign all the counters.

2. Full extra L1A data from a token ring, this error looks as if this and only this token ring received an extra L1A, sending to the DCC the corresponding event. This problem is expected to be recoverable with *TTS_RESYNC*, that will clear the error detected in the DCC and realign all the counters.

3. Reset of Token ring parameters, the token ring parameters are put back to turn-on values, as if a *RESET* was issued to the token ring. The CCS Supervisor will need to reconfigure the token ring registers, it should take $\sim 5\,\mathrm{s}$.

4. Loss of control over the token ring, to recover from this state CCS Supervisor has to regain control over token ring, this process might take $\sim$ 60s to finish.

Information

It is unknown if type 1 or 2 affect the timing of the token ring.

1.2 Possible recovery from SEU

We expect that the problem type 1 and 2 can be solved by a *TTS_RESYNC*.

According to the TTS specifications [2] a *TTS_RESYNC* is issued to allow a system out of the TTS_OUTOFSYNC state, we do not plan to use this state, but to use instead the state TTS_ERROR, to this state the TTS will send a *TTS_HARDRESET*, in the end of a *TTS_HARDRESET* a *TTS_RESYNC* is always issued.

So, if the TTS status of ECAL is TTS_ERROR ECAL will receive the *TTS_RESYNC* to recover from SEU type 1 and 2.

For SEU type 3 and 4 a software action has to be taken. This implies that the software identifies correctly the situation and changes it state from **Running** to **RunningSEU**, this state change is transmitted to the Level 0 function manager, that will pause the run and issue the FixSEU command.

1.3 Token ring problems detection

ECAL can assess the token ring status through three of the off-detector boards: CCS, DCC and TCC.

- The CCS can in principle read the token ring directly and detect changes in the configuration (due to reset) or loss of communication. This will allow to detection of type 3 and 4 problems, but not type 1 and 2.
- The DCC monitors the data stream and if the links served by a token ring show problems simultaneously, we can infer that that token ring had a problem. To the DCC all type of problems have the same symptom; error in the data stream, so we will assume that at problem first detected is type 1 or 2, fixable with a *TTS_HARDRESET*. If the *TTS_HARDRESET* fails to recover the links the software-based process, **RunningSEU**, will have to be triggered. As the Response to he state ERROr is automatic, the DCC supervisor just has to check the DCC for a permanent ERROR state to change to **RunningSEU** state.
- The TCC can also monitor the trigger stream and if it sees links served from the same token ring showing errors simultaneously, a token ring problem can be inferred. It is currently not clear if the TCC would be able to identify type 1 and 2 problems, as type 1 would be a single trigger primitive (TP) and type 2 is not applicable to the trigger link (in the trigger link every 25 ns a word is sent). Type 3 and 4 could very probably be seen either as the links going to error, or random words being transmitted to the trigger system (possibly causing high trigger rate).

The workflow the DCC needs to adhere to detect SEU problems, should be the following (Fig.1):

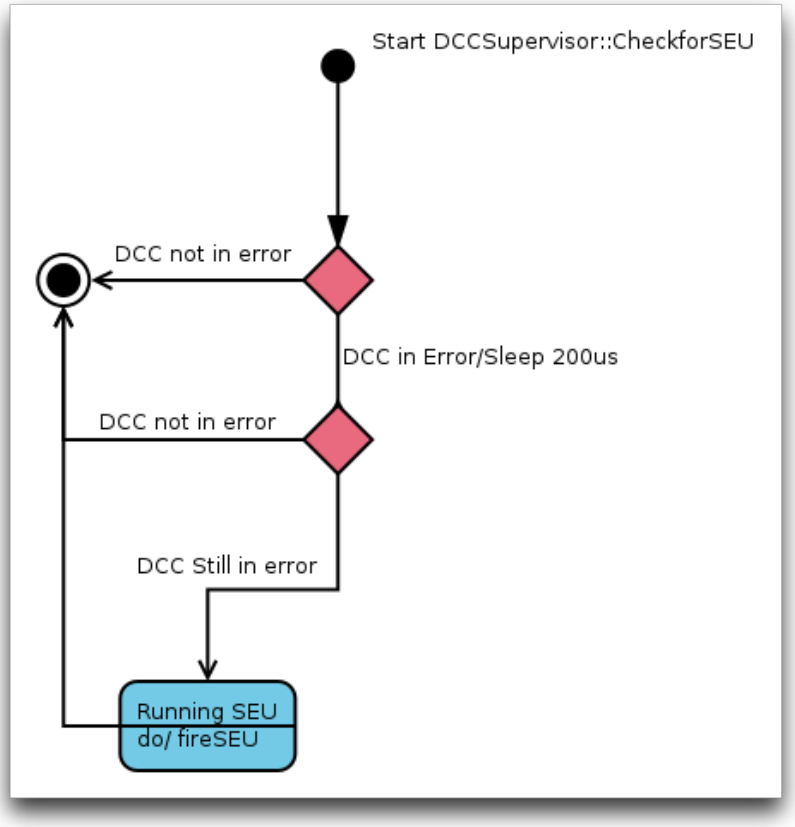

Figure 1: DCC Supervisor flowchart for SEU detection

2 SEU recovery procedure in ECAL Software

2.1 ECAL Function Manager (ECAL FM)

The function manager will have to implement the following states:

- **Running with SEU** : required by run control (RC), this state is used to indicate to level 0 run control that a subsystem identified a SEU problem and it is ready to fix it.
- **Recovering SEU**: required by RC, this is the state the subsystem changes to after receiving authorization to fix SEU. When the SEU is recovered the subsystem should either go to **Running**, indicating the recovery was successful and the run can continue, or go to **Error** indicating the recovery failed and the run needs to be stopped.

2.2 ECAL application

The new states and transitions need to be implemented in the ECAL state machine.

The new states are:

- **Running_SEU**: This is the target state for the action *FoundSEU*, it is a state required by Run Control, this state is used to indicate to level 0 run control that a subsystem identified a SEU problem and it is ready to fix it.
- **Fixed_SEU**: This is the target state for the action *FixingSEU*, State indicating that the supervisor finished successfully the SEU recovery process. If the recovery failed, the supervisor

should go to **Error**.

The new transitions are:

- *FoundSEU*: (c++ function: `SeuFoundAction`) transition between **Running** and **RunningSEU**, used to indicate to the function manager that ECAL identified a SEU problem and it is ready to recover from it.
- *FixSEU*: (c++ function: `SeuFixAction`) transition indicating that the recovery of SEU can be performed, at reception of this signal the supervisor should start the SEU recovery and go to **FixedSEU** if the process was successful or Fail if there was a problem.
- *ResumeRun*: (c++ function: `SeuResumeAction`) transition sent to all supervisors, indicating all the supervisors reached **FixedSEU** state, meaning the recovery finished, at the reception of this signal the Supervisor should cleanup any problems due to SEU recovery procedure and change state from **FixedSEU** to **Running**.

Each supervisor will also keep track of which FEDs request the recovery, so that infinite recovery loops are avoided. For that, the supervisor it will keep track of the number of recoveries and the time since last recovery. The algorithm will be as follows, before sending a SEU recovery request, the supervisor will check if it has register of previous requests from this FED, if the previous request was in a 5 mn(configurable) window, the number of requests is incremented and the timestamp remains the same. If the last request is longer that 5 mn the timestamp is incremented and the counter reset to 1. If the counter is bigger than 2 and the time difference smaller than 5 mn, the SEU is not triggered.

2.3 ECAL Supervisor

ECAL supervisor should manage the whole SEU recovery process.

It will receive a notification that an application changed its state to **RunningSEU**; this notification should have the information of which FED and channels present the problem. At that point it will change its state to **RunningSEU**, this will notify the ECAL Function manager that will inform level 0 and the SEU recovery procedure is started.
From the ECAL perspective the SEU recovery starts at the reception of the signal *Recover SEU* from the lvl0. When the *Recover SEU* is received, the ECAL supervisor sends the signal *FixSEU* with the list of (FEDs, channels in error) with problems to all the supervisors that will start the *Recover SEU* process and change to **FixedSEU** when the supervisors recovery process is finished. When all the supervisors report the state **FixedSEU**, ECAL supervisor will send the command *ResumeRun* to the supervisors that will clean up any problems the recover SEU process caused and go back to **Running**. When all the supervisors are in **Running**, ECAL supervisor will change to **Running** and the run is resumed.

The schematic of all the state changes and messages exchanged can be found in Fig. 2

2.3.1 HTTP interface to request SEU

The ECAL Supervisor also has a interface that allows to create the request for SEU recovery.

1. One method `ECALSupervisor::fakeSEU()` responds to the "... /fakeSEU" page, and generates immediately a **Running with SEU** for FEDs 696 to 699 in several channels. the only real use of this method is to test the software response, to make sure the connection ECAL Supervisor → ECAL FM → CMS lvl0 .

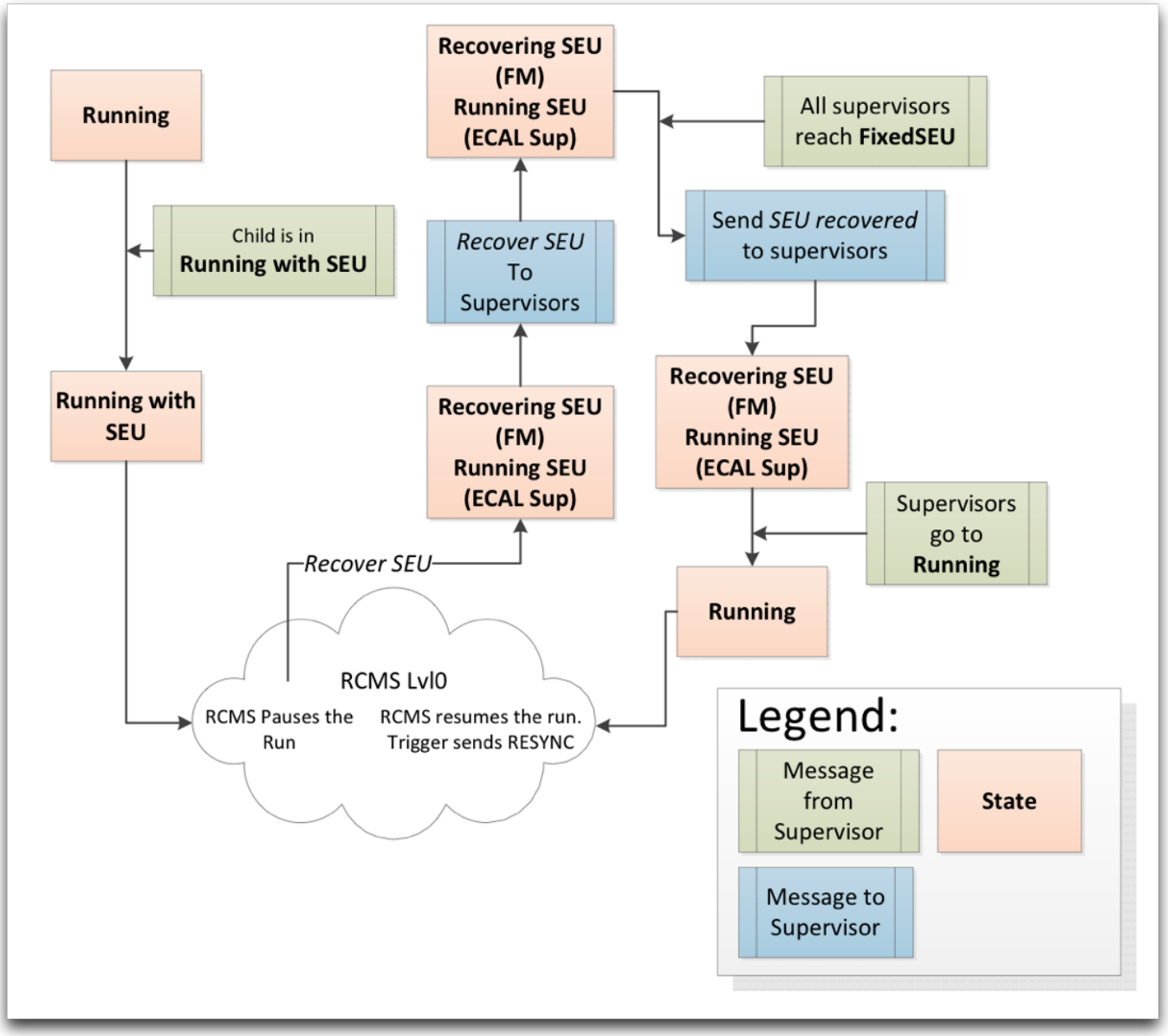

Figure 2: Schematic of ECAL Supervisor view of the SEU recovery procedure

2. The other method `ECALSupervisorUI::SEUform` responds to the url "…/SEUform" and will produce a form were the user can specify for which FED and channels he wants to trigger the SEU recovery. The channels must be specified by a column separated list with no spaces. This interface is useful to manually trigger a recovery if the automatic mechanism was unable to detect the token ring problem.

2.4 DCC Supervisor

The DCC Supervisor can read from the DCC the channels states, and it can use the channel state information to infer if a token ring has problems.

All the token rings control more than 8 channels, and from these more that 4 are seen by the same input handler. Currently if more than 4 channels are in error in a input handler the whole DCC will change its TTS status to ERROR. This means that a problem in the token ring that affects the data structure, will make the DCC to change it status to ERROR. We expect that a SEU type 3 or 4 will keep the DCC in permanent error, so if we find the DCC stuck in ERROR we can trigger the soft recovery by going to state **RunningSEU**.

2.5 DCC driver

A few methods were implemented in the DCC driver, documented here, but these methods are currently not used for the SEU recovery.
`ecal::status::DeviceStatus setRunningSEU()`: Sets the register that changes the TTS status to ERROR.

`ecal::status::DeviceStatus clearHardResetCounter()`: Clears counter for *TTC_HARDRESET* received.

`bool hardResetDuringSEU()`: This method checks if the Hardware SEU recovery was done, this is the value of bit *TTC_HARDRESET* in the STATUS register.

2.6 CCS Supervisor

The CCS Supervisor will receive a *FixSEU* together with the information of the FED/channels with problem, and it will take actions to recover the token ring. When the token ring is recovered, it will go to **FixedSEU**. If for some reason it fails to recover the SEU the CCS Supervisor will go to **Error**.

3 Code details

3.1 SEU Detection and reporting

3.1.1 ecalDCC

`pair<bool,unsigned int> DCCSupervisor::CheckForSEUInFED(ecal::FEDBase<DCC> *theFed) :`
This function checks for SEU problems in given FED. It checks for the DCC stuck in ERROR status (DCC stuck = two checks separated by 200us return DCC in ERROR).
If the DCC is stuck in ERROR, it reads channel states for given FED and it will produce a pair¡boolean,vector¿ if the more than `seuMaxChannels_` are found. The value of `seuMaxChannels_` is configure via a configurable `xdata::UnsignedInteger` with name `seuMaxChannels` and default value 5.

`DCCSupervisor::CheckForSEU():` When this method loops over all the FEDs using `CheckForSEUInFED` to detect SEU in individual FEDs, when a SEU is detected in a FED it will trigger the soft

SEU recovery. If the number of channels in a FED is bigger than 20 the SEU is not raised, as it is probably a LV problem.

`DCCSupervisor::FakeSEU(xgi::Input *in, xgi::Output *out)` The goal of this method is to test the SEU mechanism in the software, it is the callback to the *FakeSEU* request. It reports SEU problem in controlled FEDs in channels 98, 99 and 100.

3.1.2 ecalBase

`Application::fireSEUEvent(vector<pair<ecal::type::fedNumber_t,ecal::type::chsVect_t>> chs)` This method is used by any Supervisor to send a SEU notification to the ECAL Supervisor, the method receives a vector of pairs (fed number, channels with error), reformats this vector into a vector of `ecal:type::ChannelsBag` to be shipped to the ECAL Supervisor and changes the state **RunningSEU**, sending the information to the ECAL Supervisor. This method will also create an exception reporting the feds and channels with SEU, to be handled by the xmas alarming mechanism. The raised alarms can be found in the coldspot application `https://xdaq.web.cern.ch/xdaq/xmas/11/coldspot/coldspot.swf?spotlighturl=http%3A%2F%2Fsrv-c2d04-20.cms%3A9968&sysmodelurl=https%3A%2F%2Fxdaq.web.cern.ch%2Fxdaq%2Fsetup%2F11%2Fdefaultmodel.xml` and shown in Fig. 3

`bool Application::tooManySEUs( ecal::type::fedNumber_t fed )` Method that keeps track of the number of SEUs in a time windows and will return true if too many SEU recoveries were requested. The method is configurable via the configurables:

- `xdata::UnsignedInteger seuMaxTries`, default 3 tries
- `xdata::UnsignedInteger seuWindowTime`, default 300 seconds

If too many request are detected an exception is raised to the alarming system.

`StateHandler::update(xoap::MessageReference msg)` This method automatically extracts information from the received messages, for the SEU it will extract the vector with FEDs and channels in error and it will place it in a variable `channelsSEU_`.
The report for given application can be recovered using `channelsBags_t StateHandler::getChsWithSEU ( xdaq::ApplicationDescriptor * x)`.

`StateHandler::getChsWithSEU(xdaq::ApplicationDescriptor* x)` Returns the vector listing the channels with SEU reported by application x, the table is kept up-to-date by `StateHandler::update`.

3.1.3 ecalSupervisor

`ECALSupervisor::StateChangedCallback(xoap::MessageReference msg)` Receives spontaneous notifications from applications and looks into these notifications for: **Error** and **RunningSEU** states, calling:
`std::pair<bool,std::string> ECALSupervisor::checkForState(std::string state)` and `std::pair<bool, xdata::Table> ECALSupervisor::checkForSEU()`.
If `checkForSEU` detects the **RunningSEU**, we get the list made from the lists sent by the `fireSEUEvent` from the several applications that detected SEU. It will then call the `ECALSupervisor::fireSEUEvent(channelsBags_t chs)`, changing the ECAL Supervisor state to **RunningSEU** and reporting the information to ECAL Function Manager.

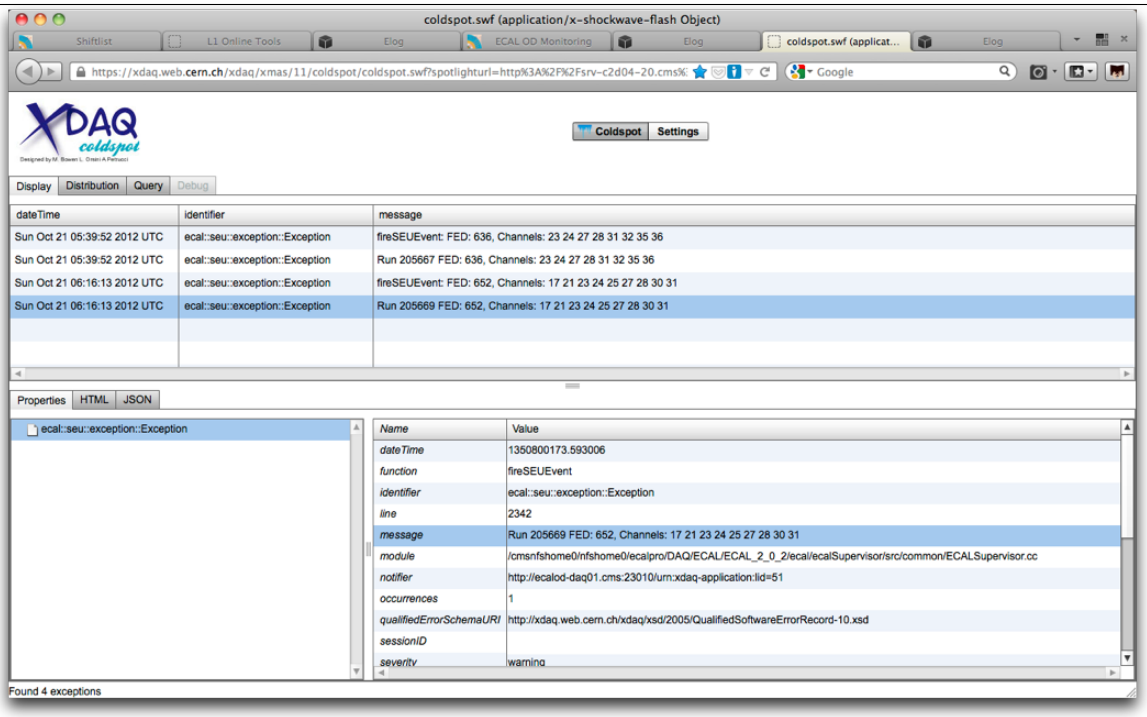

Figure 3: Screenshot of the coldspot application for ECAL, where the exceptions reported by the software recovery can be retrieved.

`std::pair<bool, channelsBags_t > ECALSupervisor::checkForSEU()` The method `checkForSEU` is called by `StateChangedCallback` and compiles the information from all the Supervisors that reported SEU. It gets the list of all applications and its states. If the state is **RunningSEU** then we get the vector with the channels list from the message, using the method `runManager_.getChsWithSEU`(method: `ecal::type::channelsBags_t StateHandler::getChsWithSEU(xdaq::ApplicationDescriptor* x)`).

`ECALSupervisor::fireSEUEvent( channelsBags_t chs)` This method receives the vector with the identification of channels with problems (fed Id, channel number).
It then stores the list of FEDs that reported SEU in `fedsWithSEU_`, this variable is shared in the infospace, under the name "FEDsWithSEU", so that the ECAL FM can read it and inform the shifter. This method will also publish a vector of coma separated values in the infospace, the variable holding the vector is called `strFEDsWithSEU_` and the infospace value is under "strFEDsWithSEU". The strings are formatted in the following way: FED: <fed number>, Channels: <ch>, <ch>, <ch>,...
After the infospace update, the the vector of `ecal::type::channelsBags_t` kept by the `runManager_` is also updated, using `RunManager::setChsSEU` method.
At this point it reports an exception to the alarming system, writes a line to the logfile `/var/log/ecalSupervisor.log` and fires the state change to go to state **RunningSEU**.

`RunManager::setChsSEU(channelsBags_t trs)` Method used by `ECALSupervisor::fireSEUEvent` to pass the list of token rings with problems to the `RunManager`.

`channelsBags_t RunManager::getChsSEU()` Method used by `ECALServiceConfigurator::FixSeu` to get the list of FEDs and channels with problems to pass to the applications.

3.2 SEU Recovery and resuming run

3.2.1 ecalBase

`Application::SeuFixAction (toolbox::Event::Reference e)` This method is the callback to move from the state **RunningSEU** to **FixedSEU**. Each supervisor should overload this method to deal with its specific needs.

`Application::SeuResumeAction (toolbox::Event::Reference e)` This method is the callback to move from the state **FixedSEU** to **Running**. Each supervisor should overload this method to deal with its specific needs.

`Application::fireSEURecovered()` This method is used by the applications to indicate it changed its state to **FixedSEU**.

3.2.2 ecalSupervisor

`ECALSupervisor::SeuFixAction (toolbox::Event::Reference e)` Called when the FM sends the *Recover_SEU*. `SeuFixAction` sends the messages to the supervisors involved in the SEU recovery, by group order: DAQ, trigger, and Control. When all the supervisors are done, it will change the state to **FixedSEU** and will call `ECALSupervisor::fireSEURecovered` that will trigger the process to change the state change to **Running**.

`ECALSupervisor::fixResumeSeuServices(vector<ServiceProperties> group, vector...)` This method is called by `ECALSupervisor:`
`:SeuFixAction` and `ECALSupervisor::SeuResumeAction` and it is responsible for getting the message to and from all the application belonging to the given group and whose controlled FED reported SEU. It calls `PartitionManager::getApplicationsByService(service,feds)` to get the list of applications to send message to.

`PartitionManager::getApplicatiosByService(string service, xdata::Vector<...)` This method is part of the `PartitionManager` and serves as interface between the `RunManager` and `ServiceDescriptc` it simply calls the method `ServiceDescriptor::getApplicationsByFEDs(feds)` that will do most of the work.

`ServiceDescriptor::getApplicationsByFEDs(xdata::Vector<ecal::type::fedNumberSer_t> feds)` Method searches for the applications controlling the feds, and returns the corresponding application descriptors.

`ServiceConfigurator::FixSeu()` Virtual function to be overloaded by inheriting classes.

`ECALServiceConfigurator::FixSeu()` Re-implementation of `ServiceConfigurator::FixSeu`, gets the list of applications and sends them the command to fix SEU together with the list of <FED,Channels> with problems (type `ecal::type::channelsBags_t`) under the name ChannelsSEU, this list is given to `ECALServiceConfigurator` from `Runmanager` class.

`ECALSupervisor::checkStatusWfeds( std::vector<std::string> services, ...)` Call after sending messages to each group to verify that targeted application are in **FixedSEU** state. The method simply loops over the list os services calling `ECALSupervisor::checkStatusWfeds( std::vector<std::string> services, const std::string& state, xdata::Vector<ecal::type::fedNumberSer_t> feds)` for each of the services.

`std::string ECALSupervisor::checkStatusWfeds( const std::string& service,const ...)` Checks in all applications belonging to service "`service`" and controlling FEDs identified in "`feds`" for the state "`state`".

`ECALSupervisor::SeuResumeAction (toolbox::Event::Reference e)` Similar to method `ECALSupervisor::SeuFixAction`, but it sends the messages in a different order, to control then trigger and, in the end DAQ services.

`ServiceConfigurator::ResumeFromSeu()` Pure virtual function to be overloaded by inheriting classes.

`ECALServiceConfigurator::ResumeFromSeu()` Re-implementation of `ServiceConfigurator::ResumeFromSeu`, it gets the list of applications and sends them the command to resume running after fixing SEU together with the list of <FED,Channels> with problems (type `ecal::type::channelsBags_t`) under the name "ChannelsSEU", this list is given to `ECALServiceConfigurator` from `Runmanager` class.

4 TODO in the software

4.1 Disable SEU recovery mechanism

Creating a way so that the SEU recovery mechanism can be disabled by a configuration. Add a field in the run configuration key?
This filed would have to be passed to all supervisors and the ECAL application class should be the class responsible for not raising SEU if it is not authorized.

4.2 Protect against new SEU found during SEU recovery

The SEU mechanism is expected to be relatively rare, so, no mechanism is implemented to deal with the arrival of new SEU problems during the SEU recovery steps.

4.3 Communication with the DAQ shifter

Communication with the shifter should be implemented in the case the SEU recovery requires a lengthy procedure to recover.

5 Additional monitoring

5.1 FMM logs

Currently the transition of the DCC from the READY to ERROR state is the best indicator that a problem in the token ring happened, as the TTS will react immediately to the ERROR state it is not guaranteed that the software will be able to detect the ERROR status. To complement the detection done by the DCC Supervisor we are using the logs produced by Fast Merging Modules (FMM), hardware modules dedicated to trace all the TTS state transitions for CMS.
During the run the FMM produces log files were all the TTS state changes are registered, a script, called *fmmReader.py* (locatde in */nfshome0/ecaldev/utils/fmmReader*) was written and it is able to parse the FMM logs and send an email if the ERROR state is detected in a ECAL FED, this script is automatically run every hour.

The FMM logs are also written to a DB, and it is in preparation an icinga alarm that will monitor the DB for ERROR state a ECAL FED, this monitoring will run every few seconds, and a email and SMS will be sent if the ERROR is detected.

5.2 ecalSupervisor log

The ecalSupervisor when triggers the SEU recovery writes to file a line with information about the run number, time, FED and channels, to automate the alarming, a script called *sendSEUAlarm.sh* (located in */nfshome0/ecaldev/utils/cron/sendSEUAlarm*), this script monitors the log file *vmepcS2F19-22:/var/log/ecalSupervisor.log* and will send an email if a new entry is detected, this script is run every hour.

6 Other resources

Presentation: Brief History of SEUs , `https://indico.cern.ch/conferenceDisplay.py?confId=197430`

Presentation: Thoughts on SEUs , `https://indico.cern.ch/getFile.py/access?contribId=8&resId=0&materialId=slides&confId=163127`

Presentation: Token rings problems during data taking , `https://indico.cern.ch/getFile.py/access?contribId=2&resId=0&materialId=slides&confId=150079`

Presentation: SEUs with TPLLs reset in 2012 , `https://indico.cern.ch/getFile.py/access?contribId=2&resId=1&materialId=slides&confId=184335`

Presentation: Token rings problems during data taking , `https://indico.cern.ch/getFile.py/access?contribId=2&resId=0&materialId=slides&confId=150079`

Presentation: Token rings problems during data taking. Part 2 , `https://indico.cern.ch/getFile.py/access?contribId=5&resId=1&materialId=slides&confId=154939`

Presentation: DAQ improvements for 2012 , `https://indico.cern.ch/getFile.py/access?contribId=3&sessionId=0&resId=1&materialId=slides&confId=169716`

Twiki: Tracking by Jose , `https://twiki.cern.ch/twiki/bin/viewauth/CMS/EcalESFE`

Twiki: Token Ring tables , `https://twiki.cern.ch/twiki/bin/view/CMS/TRTable`

References

[1] *TokenRing Incidents Table*, `https://twiki.cern.ch/twiki/bin/viewauth/CMS/EcalESFE`

[2] CMS L1 Trigger Control System, *CMS Trigger/DAQ Group*, `http://cdsweb.cern.ch/record/687458`