CORE
Metadata, citation and similar papers at core.ac.uk
Provided by CERN Document Server

Trigger Throttling System for CMS DAQ

A. Racz

CERN, Div. EP, Meyrin CH-1211 Geneva 23 Switzerland

Abstract

This document is a first attempt to define the basic functionnalities of the TTS in the CMS DAQ. Its role is to adapt the trigger pace to the DAQ capacity in order to avoid congestions and overflows at any stage of the readout chain. The different possibilities for the TTS to measure the load on parts of the chain are examined. It clearly appears that one part of the chain needs fast reaction time (few tens of microseconds) whereas the rest of the chain can afford longer reaction time, available to nowadays processors.

I. Introduction

The trigger throttling system (TTS) is a critical element of the CMS data acquisition system. Its role is to ensure that trigger logic is not overloading any of the electronic devices in charge of moving, processing, and storing the data from the very front-end electronic down to the storage media.

By design, the DAQ is able to process events at 100 KHz (maximum average). However, due to the random nature of the trigger, the LV1-A signal can exceed these 100 KHz for some time. Although the DAQ is able to handle higher instantaneous LV1-A frequencies, data loss and DAQ failure may occur if the LV1-A frequency is not monitored and regulated according to the DAQ load.

As inputs, the TTS receives warnings/status from all the devices that can suffer from overload and as output, the TTS produces a throttling signal that will reduce the trigger rate according to strategies to be defined later. These strategies are out of the scope of the present document.

II. Front End resources

According to the front-end electronic logical model[1], the front-end derandomizers (see Fig. 1.) are the first devices to overflow when too fast LV1-A are issued. Space and power constraints in the front-end lead to small derandomizer depth and hence, these queues are very sensitive to bursty LV1-A even if the 100 kHz average is met. In general, these derandomizers behave like a first-in-first-out queue: the input frequency is directly the LV1-A and the output frequency (detector dependent) ranges from 3us to 7us (see table in the appendix).

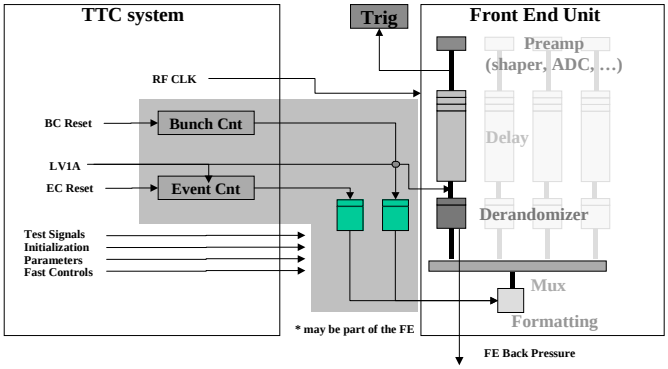

Fig. 1. Front-End logical structure

Simulations (see Fig. 2.) have been made showing the relation linking the depth, the service time and the probability to overflow. The best way to avoid overflow is to design for robustness: that means implement a big memory depth along with a short service time compared to the average time interval between two triggers (10us). When this is not possible since severe constraints on power consumption for example, other alternatives exist to minimize the overflow probability and to limit the consequences of it when it happens.

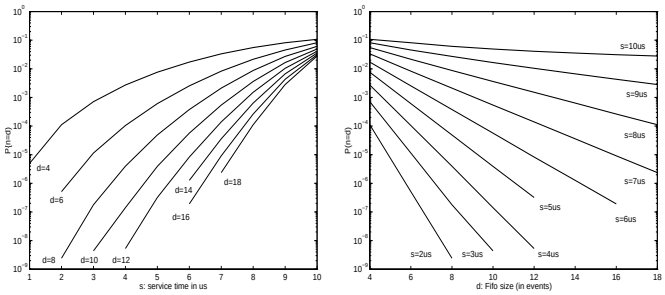

Fig. 2. Graphs showing the overflow probability versus the service time and the queue depth

A. Overflow consequences

Practically, an overflow means that a data related to a given trigger is overwritten in the readout channel or depending on the implementation, a data corresponding to a trigger cannot be introduced in the readout channel. At the level of the derandomizer, as the data is usually not time-stamped, the data of the next event will be readout as being the data of the current event: the data belonging to two different beam crossings are mixed and this is a real disaster! But the worst is not yet there... After an overflow, that channel (or group of channel) is misaligned with respect to the other channels of the detector and will continue to provide data belonging to a different trigger: the misaligned channel must be re synchronized to be in

step with the others. A resynchronization on-the-fly of a given channel (or a group of channels) would be the most efficient regarding the system availability. Technically-wise, there are three ways to perform this operation:

- the Timing Trigger and Control system (TTC)[2] must provide the right bunch and event number to the misaligned channel to be used at restart time. Due to the random nature of the trigger, the TTC cannot know what event number will be associated to which bunch number. So the only possibility is to send only the current event number reached at the end of the orbit gap of the LHC beam. During the duration of the gap (127BX or 3.175 us), there will be no event and at the first bunch of the new orbit, the bunch counter restarts at zero by definition
- the second way uses the special commands of the TTC. When the misalignment is detected and quantified, one could use 4 special commands to increment/decrement on-the-fly the event or bunch counter of a given channel. The misalignment monitoring process would detect sometime after the return-to-sync of the channel
- finally, the third way is to use a general reset broadcasted by the TTC that will flush all the data in all the front-ends (pipeline, derandomizers,....), initialize the bunch and event counter to zero and hold the system until the beginning of the next LHC orbit

The first alternative affects only a given channel but that channel can be unavailable up to an entire LHC orbit (~88us). The second alternative offers the smallest impact on the rest of the DAQ and on the un-availability time. The third one is the brute force, affects the whole DAQ, can result in an un-availability time of one orbit but the simplest to implement! The two first set severe requirements on the front-end functionnalities at the current design stage.

If none of these solutions are possible, the current run must be stopped in order to perform the resynchronisation. Then a new run can be started. If the overflows are frequent, this will reduce by a big factor the availability of the DAQ and consequently, the physics performances of CMS.

B. Overflow detection

There is no way to detect an overflow in the front-end without implementing dedicated hardware that chases this specific condition. Should the overflow happen (whatever we do, it will!), the status word for that event must include this information and the front-end electronic must be able to recover from that situation without a general reset. In other words, an overflowed detector must not loose the synchronicity of its event/bunch counter with respect to the other detectors.

C. Overflow prevention

In the front-end area, the storage resources are generally small. Therefore, the reaction time of a feed back loop must be short, i.e. within a few trigger intervals. Let us look at a simple reaction loop model (see Fig. 3 .)to better visualize the issues.

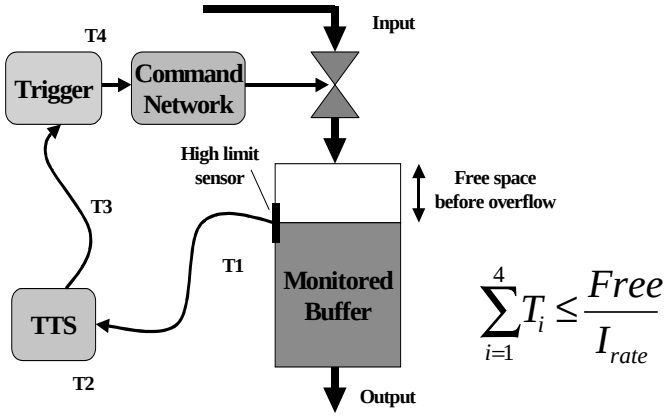

Fig. 3. Overflow model

Realistic durations can be given to the parameters:

- Sensor transmit time (T1) = 30us
- TTS processing time (T2) = 4us
- TTS transmit time (T3) = 1us
- Trigger setup time (T4) = 5us

The total response time is 40us or 4 events in average (assuming an input rate of 100 kHz), the high limit sensor must set at N-4, (N is the buffer capacity). During these 40us, 4 events will be readout (assuming an output rate of 100 kHz) and 4 events will enter the buffer. But these are only average values. The buffer would overflow if 9 triggers or more are generated. According to the Poisson distribution, the probability to have 9 or more triggers when an average of 4 is expected is still 2.1%. If by design, the output service time is shorter (i.e. 7us, ~6 events can be removed from the buffer), the situation is more comfortable: 11 triggers during 40us are needed to overflow the same buffer. The Poisson probability of this situation is 0.3%. The main problem with this solution is to use a significant part of the buffer as safety margin and of course the need of a complex alarm transmission network.

A second alternative to reduce the overflow probability is to reduce the reaction time. If hardware emulators (state machines) are used to know the buffer status at any time, the overflowing trigger can be inhibited and hence the overflow avoided. Applied to the previous model, T1 is now 0us: the total response time is 10us. If the threshold is set to N-1, 3 triggers are needed to overflow the buffer. The Poisson probability of having 3 events or more when 1 is expected is close to 8%! That means that the threshold must be set lower: for N-2, the overflow probability is 1.9% and for N-3, 0.4%. With this alternative, a complex network is avoided but emulation hardware has to be developed. Regarding the buffer usage, the situation is still not satisfactory.

A last alternative is to setup rules on the trigger: the idea is to identify the overflowing trigger sequence for each front end systems and implement in the trigger logic a sequence monitor that will regulate the burstiness of the trigger and hence inhibit the killing trigger of a sequence that overflows one front end

system. For example, these rules can be: at least 2 bunch crossings between 2 consecutive triggers, no more than 8 triggers per LHC orbit, etc.... These rules are of course highly dependent on sub-systems and require a deep knowledge of them. The drawback of this option is that the global trigger efficiency may be reduced by an inacceptable factor. Studies and simulation should allow the estimation of this inefficiency.

When numbers smaller than a percent are written, usually, one could think that the occurrence of that event is rare and marginal. In the case of the front-end, with an average time between triggers of 10us, 1% overflow probability means one overflow every millisecond of beam (!!!) and that is dramatic: if, by design, the only solution to recover from a misalignment is a reset of the front end chips, events located in the delay line and derandomizers are lost. Let's take a typical derandomizer depth of 8 events: 8 events are lost every millisecond (assuming no event in the delay line). For an average of 100 possible events during 1 ms, 8 of them are lost... In other words, this 1% overflow probability creates 8% inefficiency for the DAQ! Other queue depth and service times are represented in the graphs below (see Fig. 4 .).

This simple calculation does not consider neither the time to detect the overflow, time during which the data are corrupted and hence useless nor the fact that if the front-ends of a sub-detector are independent from each other, the overflow probability of that sub-detector is the overflow probability for a single front-end multiplied by the number of front-ends in that sub-detector.

Whatever the overflow reduction techniques will be, it is vital to maintain its probability in the 10^{-3} region or well below (if independent front-ends) and to implement efficient recovery procedures

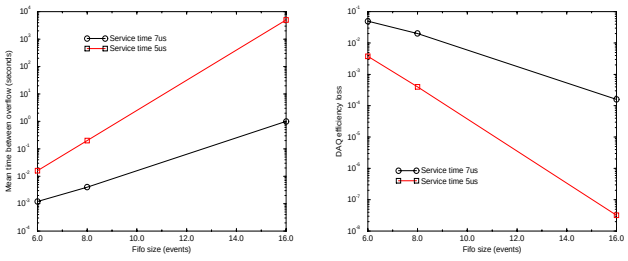

Fig. 4. Graphs showing the mean time between overflow and the DAQ efficiency loss for different derandomizer depth and for different service time when the recovery procedure is a general reset

During 1999, a TRIDAS workshop[1] addressed extensively this critical front-end issue: the table of the appendix summarizes the situation of every front-end regarding the buffer overflow. More information is available in the proceedings[2]. Without making any assumptions on the final choice, it is very likely that a combination of the last two alternatives will be used.

For obvious technical reasons, the physical location of the state machines and the trigger rule sequencer is within the global trigger logic.

III. Readout Unit resources

The storage devices of the Readout Unit (RU) are the next elements of the acquisition chain subject to overflow. These storage elements are the Detector Dependent Unit or Data Concentrator Card (DDU/DCC) and the Readout Unit Memory (RUM) (see Fig. 5 .).

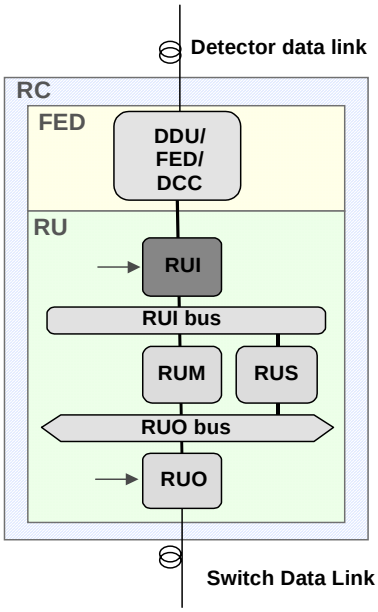

Fig. 5. Readout Unit block diagram

The physical location of the DDU is in the underground counting rooms whereas the RUM is located in the surface building: 100m (200m cable path) are in-between. The DCC provides more or less 2 KB of data per trigger: this makes 200 MB/sec. average data rate, assuming 100 kHz average trigger rate. Currently, as far as the protocol is known, the event data are removed from the DCC by a transparent data link at 400 MB/sec.: this significantly higher throughput is needed to absorb the fluctuations in the event size and also avoids the overflows in the final DCC storage buffer when nominal working conditions are met. If the data transfer is interrupted due to some failure (hardware or software), a classical avalanche of "full flags" will be issued.

Regarding the RUM, the situation is quite different from the front-end: almost no constraint on the memory size, I/O processors able to run complex monitoring/recovery tasks, standard fast network access (i.e. Gigabit Ethernet). By design, the memory is able to handle up to 100000 events (1 second of data taking). Given this capacity, the reaction time can be in

1.Front End and Readout Unit Workshop June 15-16, 1999 at CERN

2.http://cmsdoc.cern.ch/ftp/distribution/Meetings/TriDAS.workshops/99.06.15/Proceedings.html

the region of the processor times to signal a problem to the TTS.

So regarding the Readout Unit, assuming no failure in the data transfer, there is no intrinsic weakness requiring a special care.

IV. Routing network resources

As far as the architecture is defined, the routing network will be assembled from commercial units provided by the telecom industry. Although the final product is far from being chosen, they all include (with no exception) built-in monitoring capabilities and hence the TTS can be notified in the event of congestions.

V. Filter Farm resources

For what is concerning the filter farm, a distinction between the global load of the farm and the individual load of a single Filter Unit (FU) must be made.

A. Filter Unit load

The Filter Unit is made out of a Builder Unit (BU) and a commercial-off-the-shelf computer or cluster of computers (called “workers” in the following) running dedicated code. The BU receives the data to process from the routing network under the control of the Builder Manager (BM). The task of the BM is to assign a given event (or a set of events) to an individual BU. When an event is fully re-assembled, one of the “workers” starts the data processing. When the process is complete, the worker starts another event and so on. As the events are processed, the event queue of the BU is emptied. The BU can then requests another set of events. In principle, the load on the BUs and the workers should be balanced but given the event size distribution, some devices can be more loaded than others. Monitoring processes will run concurrently to ensure the proper operation of the FU. Should an overload occur (storage or processing overload), the FU can signal the problem to the “Farm Manager” or the BM who, in turn, will take corrective actions (to be defined later).

To know the load of the farm, status of each individual FUs can be collected but this will result in an important traffic on the Computing Service Network (CSN). A global load measurement can be performed at the BM level.

B. Filter Farm load

As described in the above paragraph, the BM grants a particular event of the RU to a particular FU. As the RU can handle 100000 events, the BM has a pool of 100000 “names” with which it will name the outcoming event from the RU. When a name is in-use, it cannot be assigned to another event: it is busy until the event analysis completion. At the end of the event processing, the concerned FU returns a status message with the “name” of the processed event. This name can again be re-used for another event of the RUs. With this mechanism, the BM can easily measure the global load of the farm by counting the number of names in-use.

VI. Conclusion

The role of the CMS Trigger Throttling System has been described in global terms. The situation regarding the different stages of the data acquisition chain is quite different. Intrinsically, the most problematic parts are the front-end systems where full custom solutions must be developed. For the rest of the chain, standard and well known solutions can be used.
After this first analysis, it has been shown that the TTS can be split logically and physically into two parts:

- a first one featuring quick reaction time, custom hardware, located in the global trigger logic
- a second one with slower reaction time, running on the BM processors

Finally, depending on the overflow recovery procedure, the DAQ availability time can be reduced by an inacceptable factor.

VII. References

[1] Front End Logical Model in CMS DAQ
S. CITTOLIN, J.F GILLOT, A. RACZ
CMS Technical note CMS-TN 96/015
available on http://cmsdoc.cern.ch

[2] Timing Trigger and Control reference WEB site
http://www.cern.ch/TTC/intro.html

VIII. APPENDIX

	Buffer size (events)	Service time (us)	Overflow probability	Emulation	Warning signal	Trigger rules
Pixel Detector	NA	NA	?	not possible (all the front-ens are independent	status bit in the data stream (3 levels of warning)	orbit gap extended from 127BX to 128BX and empty orbit every second
Trackers	6	7	0.8%[a] (APV6)	APV logic itself or FPGA (see CMS note 1999/028	no	at least 3 BX between consecutive triggers, other rules seems inefficient
Preshower	8	7.2 (5.4)	0.2% (4.10^{-5})	easy state machine	yes	no more than N+7 triggers during N x service time
ECAL	16	7	1.10^{-5}	easy emulation if needed	no (considered to be too slow)	none
HCAL	?	?	estimated to zero[b]	not needed	not needed	no more than 22 triggers per orbit
Muon DT	?	?	very low	not possible (occupancy dependent)	?	none
Muon RPC	?	?	very low	not possible (occupancy dependent	?	none
Muon CSC	~85	0.3/3/26	0.3event/ 24hours	?	?	?

a.extracted from fig.1
b.If the trigger rule is applied