The Fast Merging Module (FMM) for readout status processing in CMS DAQ

E. Cano[a], S. Cittolin[a], A. Csilling[a], S. Erhan[b], D. Gigi[a], F. Glege[a], M. Gulmini[a], J. Gutleber[a], C. Jacobs[a], M. Kozlovszky[a], H. Larsen[a], I. Magrans[a], F. Meijers[a], E. Meschi[a], S. Murray[a], A. Oh[a], L. Orsini[a], L. Pollet[a], A. Racz[a], D. Samyn[a], P. Scharff-Hansen[a], C. Schwick[a], P. Sphicas[a,c], J. Varela[a]

[a]CERN, Div. EP, Meyrin CH-1211 Geneva 23 Switzerland
[b]University of California, Los Angeles, USA
[c]University of Athens, Athens, Greece

Abstract

The Fast Merging Module (FMM) is part of the Trigger Throttling System (TTS). The TTS adapts the trigger frequency to the DAQ capacity in order to avoid congestions and overflows. The states of all data sources (~640 in the case of CMS) are read out and merged by the FMM to obtain the status of each detector partition. The functionality and the design of the FMM are presented in this paper.

I. Introduction

The TTS [1] is an element of the CMS data acquisition system [2]. It regulates the Level 1 Accept trigger rate (LV1A) to prevent overloading of any electronic devices in charge of moving, processing, and storing the data from the very front-end electronic down to the storage media. A global view of the TTS is shown in Figure 1.

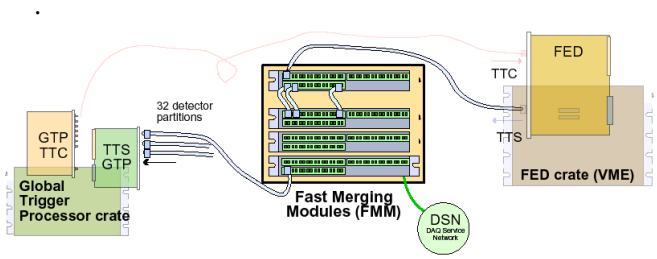

Figure 1: TTS global view

The Fast Merging Module receives and concentrates the states of Front-End Drivers (FED). The FMMs produce a single state per detector partition (see Table 1). The Trigger Control System (TCS) is able to handle 32 detector partitions. Based on the states provided by the FMMs, the TCS will adapt the LV1A rate if necessary.

Table 1: Sub-systems and FMMs

Detector (# of FEDs)	Partition (# of FEDs per partition)	# of FMMs per partition	# of FMMs per detector
Pixel (38)	barrel (32), forward (6)	1 1	2

Table 1: Sub-systems and FMMs

Detector (# of FEDs)	Partition (# of FEDs per partition)	# of FMMs per partition	# of FMMs per detector
Tracker[a] (440)	inner (114), outer (134), endcap+ (96) endcap- (96)	4+1 5+1 3+1 3+1	19
Preshower (~50)	SE+ (25) SE- (25)	1 1	2
Electronic Calorimeter (54)	EB+ (18) EB- (18) EE+ (9) EE- (9)	1 1 1 1	4
Hadronic Calorimeter (32)	HB+ (5) HB- (5) HE+ (5) HE- (5) HO+ (6) HO- (6)	1 1 1 1 1 1	6
Muon DT (5)	Barrel+ Barrel-	1 1	2
Muon RPC (6)	Barrel+ Barrel- Endcap+ Endcap-	1 1 1 1	4
Muon CSC (8)	Endcap+ (4) Endcap- (4)	1 1	2
Calo-trig Glob-Muon, Glob-trig	na na na	1 1 1	3
Muon trig	CSC trig DT trig	1 1	2
Total	31 (636)	46	46

a. one FMM is used to merge the first layer of FMMs

FEDs can be in one of the six possible states: Ready, Busy, Out-of-sync, Warning-Overflow, Failure, Disconnected (Figure 2). At startup, the device is in the Ready state. If the trigger rate is too high, after some time depending on its buffering capacity, the device will go in the Warning-Overflow state,

indicating to the Trigger Control System (TCS) that the rate is too high. If the trigger rate is not reduced, the device will enter the Busy state indicating that any further trigger will be lost leading to a loss of data with a possible loss of synchronization (i.e. event counters located in the device are no more synchronized with the central event counters located in the global trigger logic). Some devices are designed such that they can drop the data for a specific event but still stay synchronized. If the device will go in the Out-of-sync state, it will remain there until a resynchronization procedure is performed.

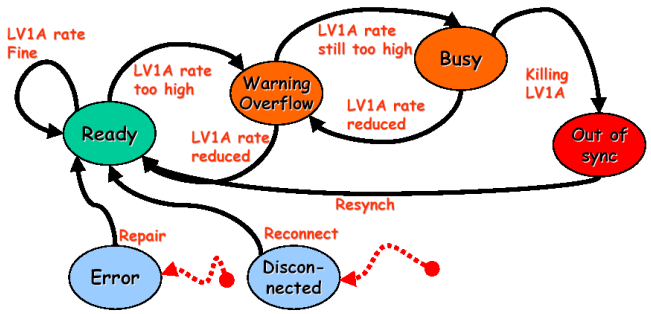

Figure 2: TTS state transition diagram

II. FMM FUNCTIONALITIES

As mentioned previously, the FMM is in charge of elaborating the detector partition state from the device state.
Dead time generated by each device is monitored in real time.
A history memory stores the state changes along with a time-tag: for monitoring purposes, a detailed analysis can be performed.
Each input can be masked in the computation of the partition state.

The output state of an FMM is computed from the inputs and the associated merging function. The merging function depends on the state to which it is applied. Currently, two functions are used:

- a logical OR
- an arithmetic sum followed by a variable threshold

The logical OR is used when one device in a given state is enough to set the whole partition in the same state. For example, when a device of a partition is busy, all the partition is declared to be busy.

The arithmetic sum combined with a threshold is used when an action is required (i.e resynchronization procedure) only when more than one device requires it. For example, it would be inappropriate to resynchronize a partition when a single device is out-of-sync.

These merging functions can be changed on request since their logic is implemented in a Field Programmable Gate Array (FPGA).

III. FMM PROTOTYPE IMPLEMENTATION

The logic of the FMM (see Figure 3) is implemented in an FPGA from the Xilinx Spartan family. External to the FPGA are only the Input/Output connectors, the history memory and the control interface.

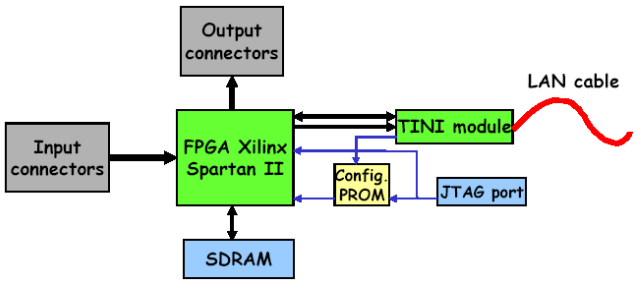

Figure 3: FMM block diagram

A. *Input/Output connector*

The connector is a standard RJ-45 network connector chosen for its low-cost and high reliability. The pinout of the connector is such that standard ethernet network cables can be used to connect a device with the FMM. The signaling level on the cable is LVDS. Built-in indicators allow to read directly the status of the attached device.(see Figure 4).

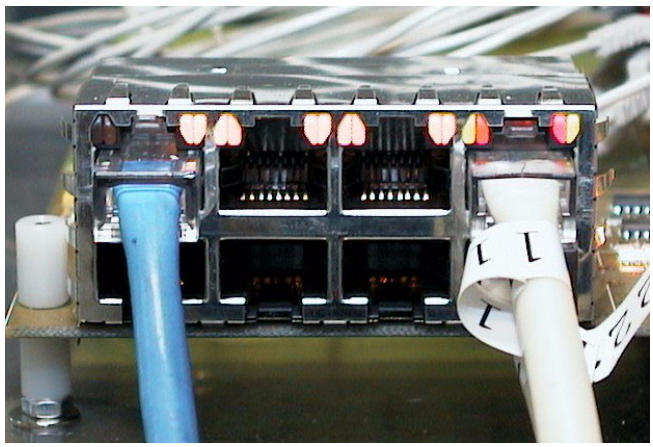

Figure 4: Input connector with status indicators

B. *History memory management*

The history memory uses SDRAM components. To optimize the usage of memory space, only state changes are written into the memory along with a time tag: a module in the FPGA continuously monitors the 128 input bits (32 devices giving each 4 bits) sampled by a 40 MHz clock. At a given point in time, if a 128-bit sample differs from the previous one, the current 128-bit status is written into the memory along with the 32-bit time tag. So for each state change, 20 bytes are written into the history memory. The current memory on the FMM is 16 MB or about 840 k-transitions. The time tag resolution is 6.4 µs: the delay before overwriting the history is ~ 7.5 hours.

C. FPGA logic

The FPGA block diagram is shown in Figure 5. From the input states, the FPGA computes the output state according to the merging functions. Its detects any state changes and fills up the history memory. Before actually writing into the external SDRAM, the states and the time-tag are first written to a small internal FIFO queue of 15 events deep. The FIFO queue is emptied to the SDRAM if no concurrent access from the control interface is taking place. This FIFO is also useful as a buffer when bursty state transitions occur.

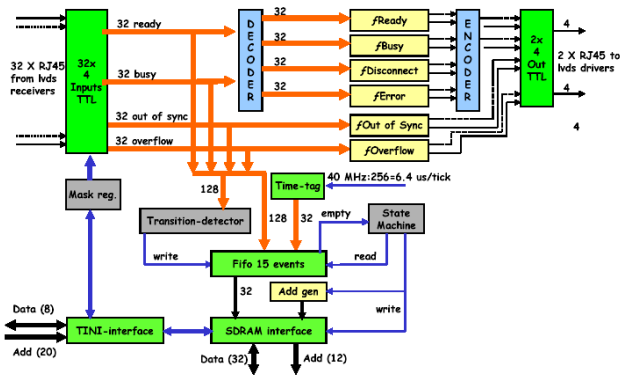

Figure 5: FPGA block diagram

D. Control interface

The control interface allows the user interaction with the FMM: configuration of the internal registers (mask, threshold, control/status), readout of the history memory, access to the deadtime monitors. The control interface is based on the Tiny InterNet Interface (TINI) [3]. The TINI platform is a combination of a chipset based on 8051 architecture and a Java programmable runtime environment. TINI's networking capability extends the connectivity of any attached device by allowing interaction with remote systems through standard network applications such as Web browsers.

The registers of the FPGA and the access to the history memory are mapped into the TINI microcontroller memory space.

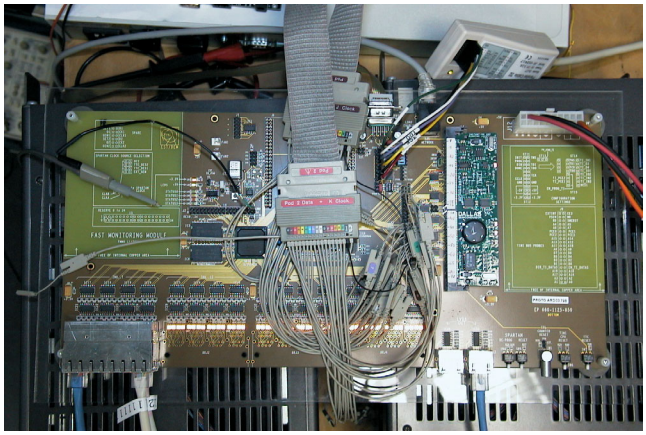

Figure 6: Prototype picture

E. Control software

A Java monitor (TiniMon) is running on the TINI processor and accepts commands either through a TCP socket based custom protocol or as http RPC requests (TiniMon is implemented as a Web Server).
On the client side, a Java debugger (TiniDebugger) provides basic FMM debugging facilities through a command line interface. Commands can be written in a script file.
A C library has also been implemented to communicate with TiniMon.

F. Performances

The measured transfer rate between the FPGA history memory and the microcontroller is about 700 kB/sec and 20 kB/sec between the client side and the TINI module.
The time to access an FPGA register is about 1 second. These limited performances are due to the TINI microcontroller processing speed.

The hardware transmission delay from the input connector to the output connector is 100 ns.

IV. Future plans

The current prototype will be used by the CMS DAQ group in the DAQ demonstrator [1]. A new prototype will be designed including new requirements:

- access latency to the FPGA registers and specifically the mask register must be in the range of a micro second. This is driven by the need to mask quickly a device generating too much deadtime, and hence penalizing the whole experiment. This leads to a redesign of the board with a faster control interface.

- real time state monitoring of a given device is an important feature to detect potential or real problems. Hardware monitoring modules will be incorporated but the FPGA logic resources are no more adapted: a bigger FPGA will be used for the next version.

V. SUMMARY

The first function of the Fast Merging Module is to perform logical and arithmetic operations on the input status information and to provide an output result that can be merged further to obtain the status of a detector partition. The second function is to keep a history of the state changes and to allow a monitoring of the data sources or a post-mortem analysis in the event of a serious system error.
A second prototype will implement a faster control interface and hardware monitoring engines for deadtime measurement.

VI. REFERENCES

[1] Trigger Throttling System for CMS DAQ, A. Racz
Proceedings of the sixth Workshop on electronics for LHC experiments, Cracow, 11-15 September 2000.

[2] CMS: The TriDAS project
Technical Design Report Volume I, Volume II

[3] TINI home page
http://www.ibutton.com/TINI/index.html