On the Level1 Trigger Decision Sorter Implementation

LHCB Technical Note

Issue: 1
Revision: 1

Reference: LPHE 2005-05 / LHCB 2004-049 DAQ
Created: 21 March 2004
Last modified: 21 June 2004

Prepared By: A. Barczyk, B. Jost, CERN and N. Neufeld, LPHE EPFL

Abstract

The implementation of the Level 1 trigger decision sorter in the LHCb DAQ system, based on Network Processors, is presented.

Document Status Sheet

1. Document Title: On the Level1 Trigger Decision Sorter Implementation			
2. Document Reference Number: LHCb 2004-049 DAQ			
3. Issue	**4. Revision**	**5. Date**	**6. Reason for change**
Draft	1	21 March 2004	First version
1	1	21 June 2004	Comments

Table of Contents

List of Figures

1. Introduction

The LHCb software trigger system will consist of two stages, called Level 1 and HLT (High Level Trigger). Both will execute algorithms on the same computer farm, the common implementation is described in [1]. Figure 1 shows the basic layout of the system. The front-end modules will send Level1 trigger data to the farm, while keeping the complete event data in memory. After processing the event, the farm node sends the decision to the Readout Supervisor for distribution to the front-end modules. Due to parallel execution of the trigger algorithms in the farm nodes and the fact that the processing time varies with the event contents, the decisions have to be sorted according to the event ID, before they reach the Readout Supervisor.

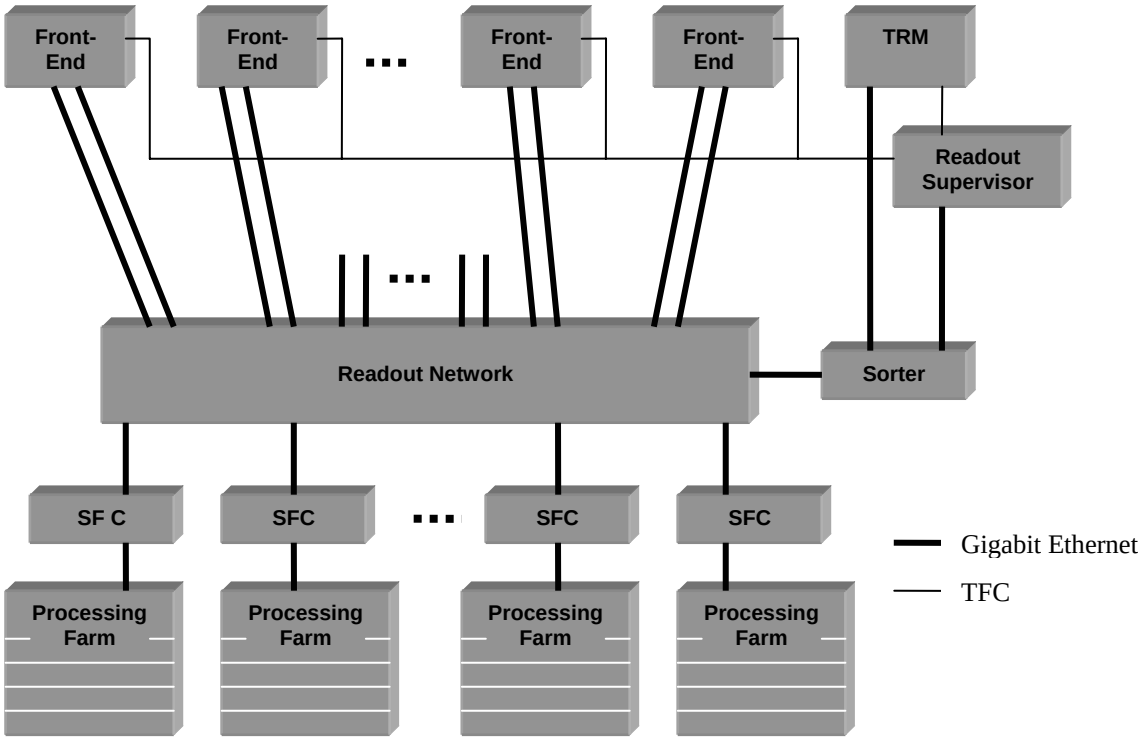

Figure 1: Schematic layout of the LHCb online system.

The detailed description of the complete LHCb trigger system can be found in [2].

1.1. Level1 Decision Frame format

As described in [1], the Level 1 trigger data will be embedded in so-called Multi-Event Packets (MEP). It seems therefore only natural to deliver the decisions in a similar structure, called Multi-Decision Packet (MDP). The MDP format has been defined in [4]. Packing Level 1 trigger decisions in the same way as

event fragment data has several advantages as compared to the single-decision frames – in particular significantly lower frame rate and lower memory bandwidth requirements in the Decision Sorter as well as in the Readout Supervisor. The frame rate is lowered by the packing factor (i.e. the number of decisions embedded in a single frame), e.g. is reduced from 1 MHz to 40 kHz for a packing factor of 25.

1.2. The Level1 Decision Sorter

Although the TELL1 boards will inject events into the readout system in sequential order according to the event ID, this order will not be maintained by the Level 1 trigger system. The main reason for this is the variable processing time of the trigger algorithm. The function of the Level 1 Decision Sorter is therefore to receive decision frames from the trigger farm and deliver them to the Readout Supervisor (RS) in sequential order of the Event ID, i.e. as the name implies to sort the decision frames arriving from the trigger farm. It also takes measures to prevent buffer overflow in the TELL1 boards, in case of non-delivery of a decision frame from the trigger farm.

1.3. The Trigger Receiver Module

In order to prevent buffer overflow in the TELL1 boards, the Decision Sorter needs to know the number of events that have entered the readout system. For this purpose, a Trigger Receive Module (TRM) will be added to the system. It will operate synchronously to the TELL1 boards, but interface directly to the Decision Sorter (c.f. Figure 1). With each MEP sent to the processing farm by the TELL1 boards, the TRM will deliver an MDP to the Decision Sorter. This MDP will carry the Level 1 IDs of the corresponding MEP events.

The TRM will be implemented using a TELL1 board.

2. Decision Sorter Implementation

Two possible implementations have been envisaged for the Level 1 Decision Sorter: One based on the NP4GS3 Network Processor [6] from IBM, and one implemented in software on a PC. In Sec. 2.1 we will describe the NP based solution as the base-line implementation, and give a brief description of the back-up solution in Sec. 2.2.

2.1. Network Processor based implementation

Network Processor technology was born with the aim to efficiently process data frames at high rate, and is therefore well suited for the problem at hand. The particular NP used to implement the Decision Sorter has been studied in detail by the LHCb online team and is described in [5].

For the implementation of the Decision Sorter, we have chosen the Copernicus card, commercially available from the S3 company [7]. The Copernicus is a PCI card hosting the IBM NP4GS3 Network Processor, and provides 3 copper Gigabit Ethernet ports.

The sorter code makes full use of the multi-threading at hardware level provided by the NP. The original code has been written in the framework of a summer student project [3], where its performance was evaluated in an instruction level simulation using the development tools provided by IBM.

The Decision Sorter code implements three threads: a decision receiver, a TRM receiver and a dispatcher thread. Several instances of the two receiver threads can execute concurrently, only the dispatcher thread has to be unique at all times. Also, the receiver threads are spawned by the NP hardware upon reception of a frame, while the dispatcher thread is started by one of the former threads.

The function of the decision receiver thread is to receive frames from the trigger farm, examine the event number and compare it to the next event number in sequence. If the frame carries the expected next event ID it is forwarded to the dispatcher thread, otherwise it is stored in a circular buffer in one of the NP's memories.

The dispatcher thread forwards the current frame, then reads the memory content of the circular buffer in order to verify if the next frame expected has already arrived earlier. If so, this frame is fetched from the memory, and forwarded to the output queue; otherwise the thread terminates.

The TRM receiver thread's function is to prevent overflow of the circular buffer, in case a decision is very late or missing. This way the Decision Sorter also provides a protection of the TELL1 buffer against overflow. The thread compares the event number from the TRM with the next event number to be sent to the Readout Supervisor. If the difference is close to the maximum number of events that can be stored in the TELL1 buffer, an MDP frame is generated, filled with default decisions for the missing events, and forwarded to the dispatcher thread.

2.2. PC based implementation

The main challenge for the sorting algorithm needed in our context is the relatively high frame rate. Clearly a specialised processing unit such as the NP is best suited for this task. Nevertheless, an implementation of the sorting algorithm on a PC would have a few advantages: the code can be written in a high level language, monitoring and problem recovery is simpler, and the implementation is easier to maintain over a long period of time. The main question to ask here is whether modern PC hardware can reliably process data frames at sustained high rate as needed by this application.

Simple tests have shown that a high-end PC equipped with a good Gigabit Ethernet NIC can handle up to ~500 kHz frame rate, if tuned correctly. This is significantly higher than the estimated MDP rate. One has to keep in mind, however, that in addition to the incoming MDP traffic, frames from TRM will arrive at the same rate, which doubles the aggregated input traffic.

The implementation of the Decision Sorter code on a PC is currently under development, and will be tested for performance in the coming weeks. It follows the main ideas described in Sec. 2.1.

2.3. Performance tests

We have implemented the decision sorter code on the NP4GS3 Network Processor (NP). The performance tests were carried out using the LHCb DAQ Test-bed, the layout of the Decision Sorter test setup shown in Figure 2.

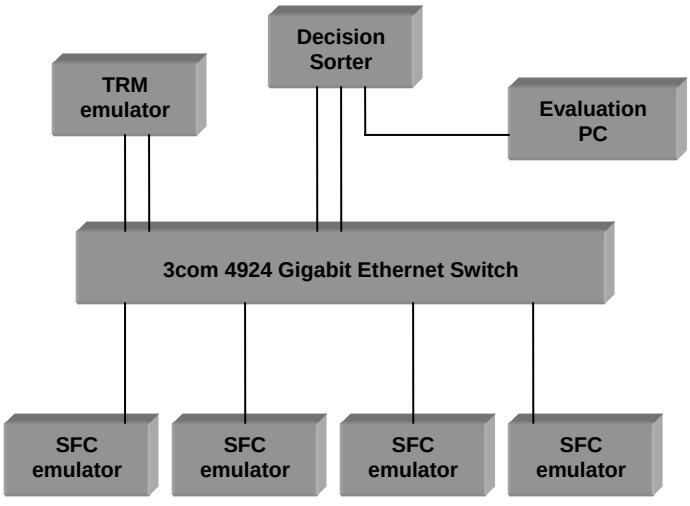

Figure 2: Level 1 Decision Sorter Test Stand

Network processor units were used for all components apart from the final destination. One NP board was programmed to act as the TRM emulator, distributing MEP frames to the SFC emulators on one port, and MDP packets to the Decision Sorter on a second port. Both ports were connected to a Gigabit Ethernet switch. Four NPs were used as sub-farm emulators, where the received frames were held back for a random

amount of time, such as to emulate the execution time of the Level 1 Trigger algorithm. After this parameterised delay, an MDP was sent to the Decision Sorter.

The correct operation of the Decision Sorter was verified using a PC. A single-threaded application was receiving the decision frames from the sorter unit, and verifying the sequence of the event IDs.

The NP based implementation has proved to work correctly with rates up to ~140 kHz with a packing factor of 25. This is a factor 3.5 above the needed rate.

The SFC emulators, in addition to modifying the order of the frames, were also programmed to discard frames at a certain rate. This way, the Decision Sorter had to react to the TRM information and generate an MDP in order to substitute the missing frame. In a test run, the complete system was operational for 4 days at a rate of 140 kHz, during which time ~ 30000 frames were discarded. The destination has received all frames in correct order, with the correct number of substituted frames. In this run, ~5×10^{10} MDP frames have been transmitted. With an expected transmission error rate of less than 10^{-14}, the emulated frame loss was several orders of magnitude higher than expected during normal operation of the experiment.

3. Conclusions

A baseline implementation of the Decision Sorter based on network processor technology has been successfully tested. Using Multi-Decision Packets, the frame rate has been reduced significantly. The performance of the NP4GS3 Network Processor from IBM has been evaluated, and found to be sufficient for the purpose of decision sorting. A back-up solution, with the sorter code running on a PC platform will be evaluated during summer 2004.

4. References

[1] "A common implementation of Level 1 trigger and HLT Data Acquisition", A. Barczyk, J-P. Dufey, B. Jost, N. Neufeld, LHCb 2003-079

[2] "LHCb Trigger System TDR", LHCb, CERN LHCC 2003-31, 2003

[3] "LHCb Level-1 trigger Real Time Sorter", J. Marecki, Summer Student Report, 2002 (unpublished)

[4] "Implementing the L1 trigger path", R. Jacobsson, LHCb 2003-80

[5] "LHCb Online System TDR", LHCb, CERN LHCC 2001-40, 2001

[6] IBM NP4GS3 Datasheet, http://www-3.ibm.com/chips/techlib/techlib.nsf/productfamilies/PowerNP_Network_Processors

[7] S3 group, http://www.s2group.com/network_processing/copernicus