# An All-Software Implementation of the Vertex Trigger

## LHCb Technical Note

| | |
|---|---|
| Issue: | 1 |
| Revision: | 1 |
| Reference: | LHCb 98-022 TRIG |
| Created: | 28-Oct-97 |
| Last modified: | 5-Feb-98 |

**Prepared By:** Mike Koratzinos, Pere Mato

## Abstract

This note describes a possible implementation of the vertex topology trigger. This trigger will be part of the first level trigger of LHCb. The requirements for the vertex trigger are analysed and a conceptual design is presented with sufficient detail to allow an accurate assessment of its feasibility. All the important aspects of the problem have been looked at and solutions for the essential issues are proposed.

## Document Status Sheet

**Table A** Document Status Sheet

| **1. Document Title: An All-Software Implementation of the Vertex Trigger** | | | |
|---|---|---|---|
| **2. Document Reference Number: LHCb 98-022** | | | |
| **3. Issue** | **4. Revision** | **5. Date** | **6. Reason for change** |
| 1 | 1 | 5/2/98 | First version. |
| | | | |
| | | | |

# Table of Contents

# 1 Introduction

In this document we present a possible implementation model of the vertex topology trigger of LHCb. This trigger should provide a fast decision on wether there are secondary vertices at a given distance range from the primary vertex of the collision using the information collected by the silicon vertex detector. This topology is a clear signature of B particle decays that are of interest to the LHCb physics programme and provides a very good discrimination against background events.

We have started from the requirements for the vertex trigger system. These requirements are collected in the document [1]. The needs have been analysed in terms of data bandwidth and processing power. The results of these analyses are presented in section 3 and form the basis for the design of the possible solution that is described in some macroscopic detail in section 5. By presenting a conceptual implementation design, that is, decomposing the complete system into components, each of which can be easily realised using presently available technology, we intend to demonstrate the feasibility of the vertex trigger. The implementation design should also give us an idea of the cost of a system with those characteristics.

## 1.1 The detector

The silicon vertex detector of LHCb is described elsewhere [2]. For this report we will only briefly mention some relevant points. The detector sits in a magnetic field-free region inside a vacuum tank and comprises 17 identical 'stations' with an inter-station spacing of 4cm. Each station has two silicon planes, one with strips at constant radius (the r-strips) and one with strips at constant phi (the phi-strips). This r-phi geometry has been preferred over the more conventional x-y geometry (adopted for the LoI **Error! Reference source not found.**) due to the fact that R and phi are the natural coordinates of the problem and therefore such geometry simplifies some of the calculations that need to be performed. Each plane is further subdivided in 6 'sectors' each covering approximately 60 degrees in phi. The R strip pitch varies from 40 μm at the smallest radius, to 60 μm in the middle, and finally 80 μm for the outer region. Similarly, the phi strip width is 60/256 degrees for the inner part of the detector and 60/640 degrees for the outer part. A stereo angle is incorporated in the phi strip design. The number of channels per station is 7590 for the r plane and 5376 for the phi plane, taking the total number of channels to about 230000.

LHCb uses a special low-luminosity interaction point. Beam dimensions are about 80 μm in x and y, whereas in z, the interaction region spreads over several centimetres.

## 1.2 The triggering scheme

B particles produced at LHC energies travel on average 7 mm before decaying. A large percentage of the decays produce charmed particles which themselves decay downstream. Therefore, there are a large number of tracks with large impact parameter to the primary vertex, as well as a number of secondary vertices. In contrast, minimum bias events have only a small proportion of large impact parameter tracks and secondary vertices, mainly from $K^0$ decays. It therefore appears that a good discriminator for B events is the number of secondary vertices and/or the number of tracks with large impact parameters. The vertex detector trigger uses these topological considerations to select B events.

# 2 Vertex Trigger Requirements

## 2.1 General requirements

**Input rate**. The maximum input rate to the trigger system (L0 rate) has been taken to be 1MHz. The trigger system should be able to accept a new set of data every 1 μs and produce decisions at the same rate.

**Trigger data**. It is envisaged to use only binary information from the vertex detector for triggering purposes. A simple clustering algorithm, run in conjunction with the zero suppression, would give the addresses of r and phi clusters in units of (strip pitch)/2. A single 2-byte number is sufficient to uniquely identify a cluster in each one of the 17 detector stations.

**Output accept rate**. Here a reduction factor of 25 in minimum bias events has been assumed. This defines the L1 rate to be 40 kHz.

**Output decision order**. The trigger system should deliver the decisions to the LHCb global trigger in correct chronological order.

**The maximum allowed latency**. The depth of the L1 buffer defines the maximum latency. It has been assumed to be 256 μs. This assumption requires the use of digital L1 buffers for all sub-detectors, which in turn implies that the buffers will reside in a low radiation area away from the detector. The alternative solution, using analog L1 buffers on the detector, would imply a latency of the order of 25 μs and would probably require a different approach to the L1 vertex trigger implementation. It should be noted that if the digital L1 buffering solution is adopted, this latency could easily be increased to 512 μs or even 1024 μs.

**Trigger algorithm**. The system should implement the vertex trigger algorithm described in [4]. The performance of the trigger (efficiency for Bs and rejection power) should be the same as with the Montecarlo results. For the analysis of the requirements, especially for the CPU power needed, we have used the standard vertex trigger algorithm.

## 2.2 Other assumptions

We also have to make the assumption of the processing power for a commodity-standard CPU by the time of the construction of the experiment. Here, in accordance to standard policy, 1000 MIP machines have been assumed.

## 2.3 Physics requirements

The physics requirements are simple: One would like to use the L1 vertex trigger algorithm that maximises the B efficiency within the constraints noted in the previous section. This means using the most sophisticated algorithm that fits with in the latency budget. As an indication, a canonical 1000 MIPs CPU in the maximum latency allowed would be able to perform about 200,000 instructions, after allowing for I/O latency.

Although it is possible to work in the r-z plane and ignore phi information, studies have shown that phi information enhances the trigger performance and hence we intend to use it.

Another important consideration comes from detector alignment: Inter-detector misalignment or beam-detector misalignment degrades the detector performance, therefore a provision for the use of updatable alignment constants should be made.

# 3 Analysis of the Requirements

## 3.1 Data bandwidth requirements

The full detector simulation [3] has been used to obtain the expected number of clusters per event. The distribution of the number of r and phi clusters can be seen in Figure 1. The mean number of clusters per event is about 600. To produce estimates on data bandwidth we have used a more realistic number of clusters taking into account noise in the detector. For that, we have assumed 400 hits per event due to noise, giving a total of 1000 hits per event on average. Since a 2-byte number can easily represent each cluster within a station, the average size of an event is 2000 bytes and the average data rate to the vertex trigger system is 2 Gbytes/s.

The distribution of the number of clusters does not have long tails, as seen in Figure 1. The maximum event size has been taken to be 1600 clusters and with the added noise of 400 hits, this gives 2000 clusters per event and corresponds to 4000 bytes.

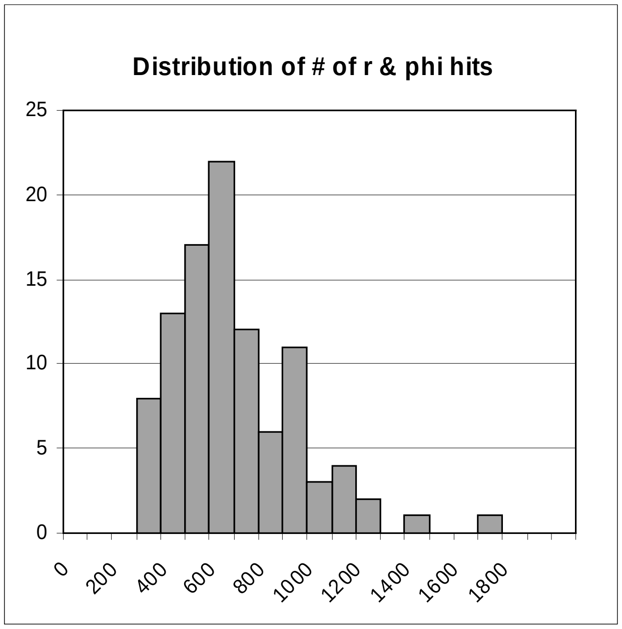


Figure 1 Distribution of the number of r & phi clusters per station

## 3.2 Processing requirements

The standard vertex trigger algorithm consists of the following steps: triplet finding, triplet removal, primary vertex finding, impact parameter calculation, 3D track finding, secondary vertex finding and trigger decision. The two steps comprise the 2D track finding algorithm. For more details of the algorithm please refer to [4]. Each of these steps has been benchmarked using a version of the code re-written in C in a style appropriate for real-time environment. Table 1 shows the average time of the different steps using an Intel 180 MHz Pentium-II based PC running Windows NT 4.0 and compiled with Visual C++ version 5.0.

| **Part of the algorithm** | **Time [μsec] (140 MIPs)** | **Time [μsec] (1000 MIPs)** |
|---|---|---|
| Triplet finding | 417 | 58 |
| Triplet removal | 117 | 16 |
| Primary vertex finding | 139 | 19 |
| Impact parameter 2D | 10 | 2 |
| 3D tracks | 108 | 15 |
| Secondary vertex finding | 52 | 7 |
| TOTAL | 843 | 117 |

Table 1 Results of benchmarking the vertex trigger algorithm in an Intel 180 MHz Pentium-II PC running Windows NT 4.0. The numbers are the average elapsed time for each algorithm step. The last column is the extrapolation to a 1000 MIPs processor.

The average processing time gives us an idea of how many processors we will need to implement a portion of or the complete algorithm using general-purpose processors. For example to implement the whole algorithm with an input rate of 1 MHz would require 117 processors with 1000 MIP processing power, if the design of the system ensures that the processors are kept busy all the time.

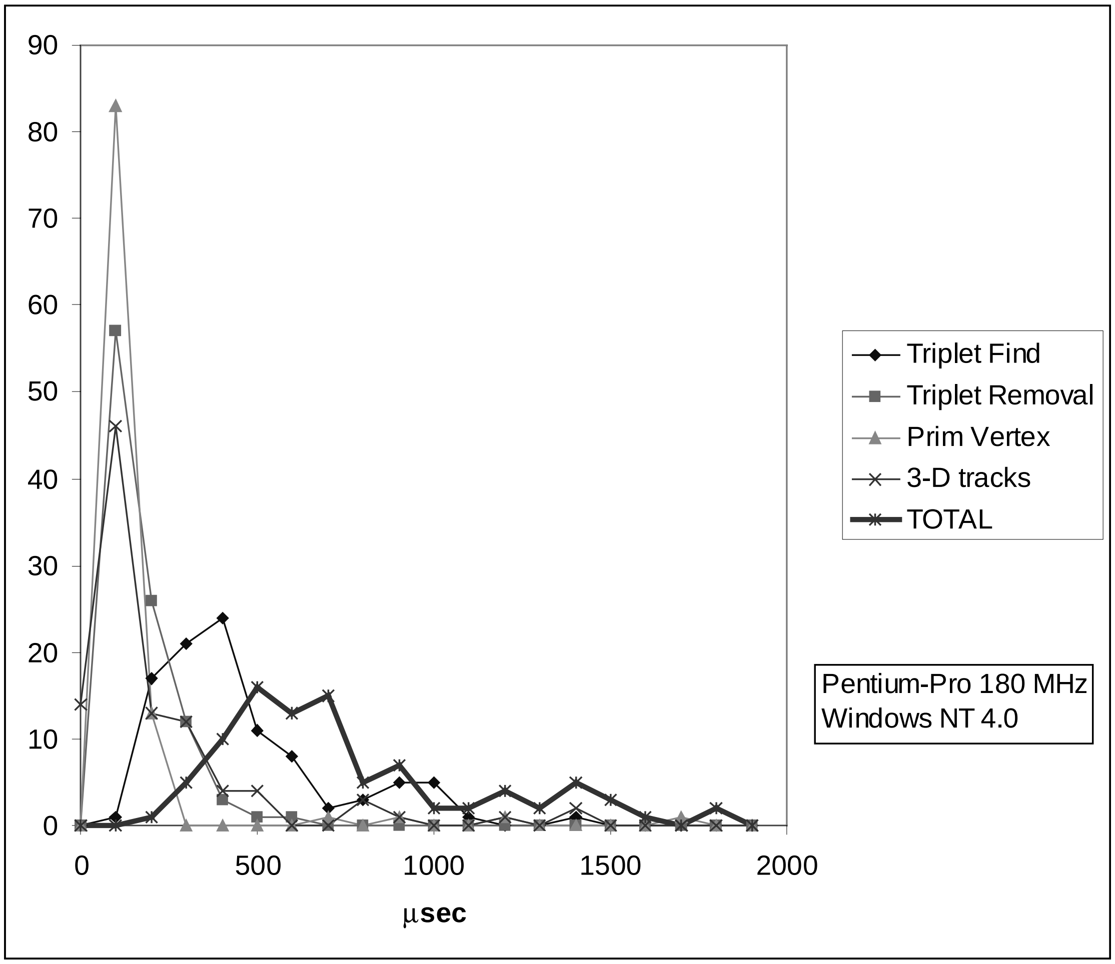


Figure 2 Distribution of the elapsed time of the different parts of the algorithm running in a 200 MHz Pentium-II processor.

To estimate the maximum latency, which is of importance for the maximum depth in the L1 buffers, the relevant parameters are the tails in the distributions of processing time. We can see in Figure 2 that there are tails almost everywhere. Moreover, there are tails in the parts of the algorithm which are difficult to process in parallel using local trigger data, e.g. the primary vertex, 3D tracks finding, etc. and these must be done sequentially using processors with a rich instruction set. Therefore taking into account these tails and extrapolating to 1000 MIP processors we can conclude that we need more that 50 μs maximum latency to perform the later parts of the trigger algorithm.

If one would consider running the complete algorithm in processors, then the maximum latency caused by the processing time for the vertex trigger should be of the order of 300 μs using 1000 MIP processors with the current version of the software. We have not yet finished the complete code optimization, but we envisage to be able to reduce this time by a sizable factor with no loss in performance.

## 4 Design considerations

The problem is effectively twofold: one part has to do with the processing of the trigger data using an algorithm with good B-tagging efficiency; the second has to do with collecting all the data from all the detector elements and sending them to the unit responsible for the trigger decision for that event (event building). Due to the complicated nature of the last steps of the vertex algorithm, the data have to end up in a memory of a processor that will complete the algorithm and take the decision. Whether or not it makes sense to perform a preprocessing before the transfer to the CPU depends on if the resulting bandwidth reduction justifies the extra cost of the preprocessing. The sustained data bandwidth required for transferring unprocessed r & phi cluster information is 2 Gbytes/s.

In the proposed implementation described in [5], the r- cluster information is pre-processed by an array of special purpose processors (FPGA) in parallel and the data volume is reduced by a factor 3 or 4 by sending the list of 2-D tracks instead of the initial r-clusters. However, the total bandwidth to the final processors required is still 1.2 -1.3 Gbytes/s since the phi data cannot be reduced. A more complicated method can be envisaged transferring only the phi information that is required on demand during the execution of the algorithm.

A potential advantage of finding 2-D track in hardware before continuing the execution of the algorithm with normal processors is to reduce the latency. We consider that the optimization of the latency for this first part of the algorithm should not be a driving force for the design, especially taking into account that the latency for the rest of the algorithm has substantial tails.

Furthermore, there are other important considerations. It is highly desirable for a design to be flexible, easy to build and maintain, and upgradeable. Flexibility is needed because our needs are going to change over time. The LHCb environment is still an unknown and we must be ready for surprises. The system should adapt to these unforeseeable changes in a smooth manner. Minimising the number of different type of components eases system implementation and long-term maintenance. And finally, commercial processors will evolve and faster ones will continue to appear after the commissioning of the system, so it would be desirable to be able to upgrade the system without much effort.

The proposed solution described in the next section tries to solve both main problems while having the extra virtues of flexibility, ease of maintenance and upgradability. From an engineering point of view the event building is the most challenging part, whereas from a physics point of view the challenge lies in developing an efficient B triggering algorithm that would fit within our latency budget.

# 5 Proposed design

## 5.1 Overview

The design we have adopted is to send all the trigger data belonging to the same event to a single processor in a farm of processors. The trigger algorithm will then be implemented as a software program running in the processor and having the complete event available in its memory. Therefore the main problem we are faced with is event building at a rate of 1MHz with an aggregated sustained throughput of 2 Gbytes/s. A diagram of the architectural design can be seen in Figure 3 where the event builder function has been implemented using a switch.

Two possibilities exist for the way that data are transferred to the processors: One is a *data on demand* scheme, where a processor requests the data it needs for a specific event during the execution of the trigger algorithm. For example, the processor could request the phi data for the sectors where 2D tracks have been found and not for the others. This approach minimizes data flow but is very heavy in control resources. Our calculations have shown that the gain in event bandwidth does not compensate for the resulting complexity of the design. We therefore propose the alternative scheme where all the data is always pushed through to the processors.

## 5.2 Data-Flow Protocol

Trigger data blocks containing the r and phi coordinates of the strip clusters are received by the trigger system from the front-end electronics sitting near the detector. For these studies we have assumed that the number of data sources is 17, that is equal to the number of stations. A header containing the event identifier trigger number and the station number will precede each data block. The average size of the data block is 100 bytes.

The idea is to push all the data blocks from the 17 sources into the processor's memory where the algorithm will be executed. The protocol is kept to a minimum by having enough data buffers in the final processor and in the intermediate stages to store sufficient events such that we will not need to worry about the available free space in an event by event basis. In the rare occasion of a buffer getting close to the maximum capacity a mechanism of indicating problems is foreseen. More quantitative statements on this will be given after the simulation of the trigger scheme is performed.

The destination processor for a given event is determined by a combination of a fixed algorithm based on the event number (destination port of the switch) and later with some kind of load balancing scheme to finally select the destination processor. In that way, the decision is made locally without the need of a global event supervisor or event manager taking into account which processor in the complete farm is ready to take new events. The minor drawback of this approach is that we have to slightly over dimension the processing capacity of the processor farm in order to absorb variations and fluctuations on the available CPU power of each processor.

Finally, the results of the execution of the trigger algorithm will be sent to a single specialized processor where the decisions will be re-ordered and a time-out will be applied for the events taking too long to process. The time-out cases should be minimized and whether the events will be accepted or rejected will depend on a careful study of the subject. However, as long as the time-out rate is much lower than the minimum bias retention rate (4%) the decision of what to do with time-outs is not an issue.

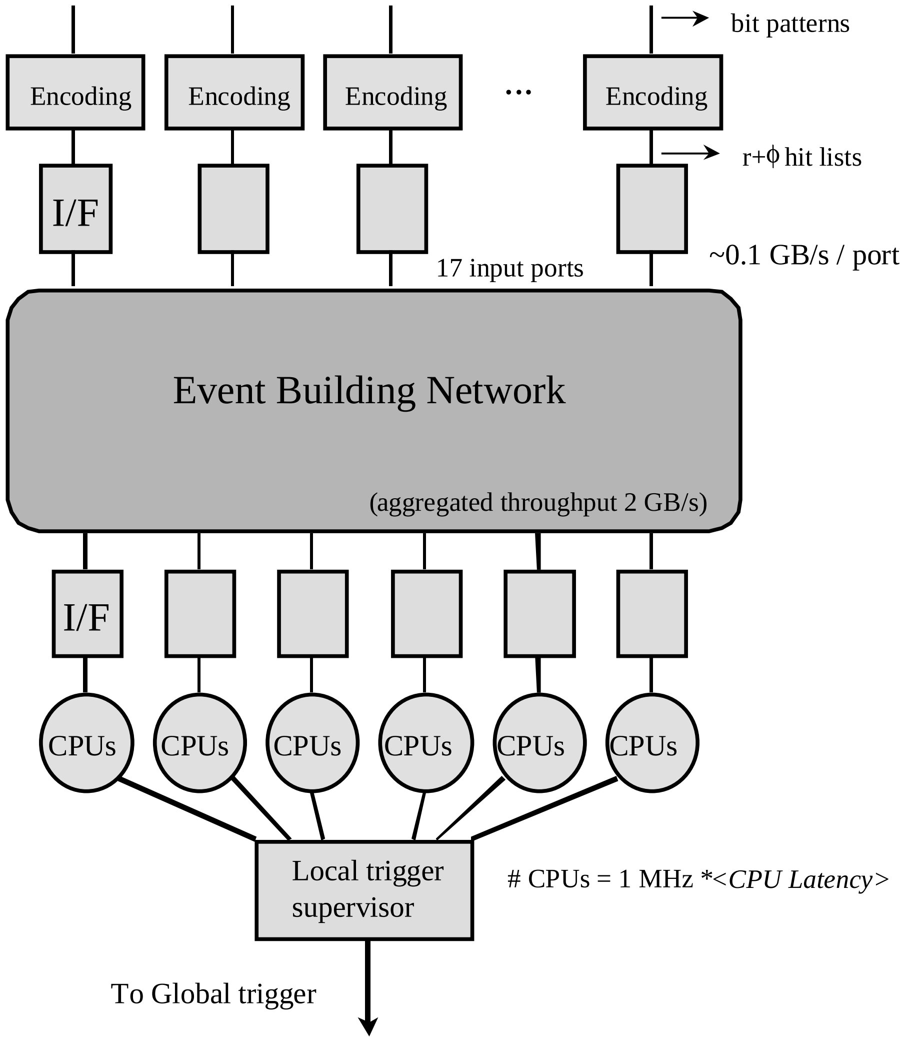


Figure 3 Overview of the vertex trigger implementation

## 5.3 Clustering & Coordinate encoding

Although, strictly speaking, this is not part of the trigger but rather of the front-end electronics, we will briefly discuss the address encoding in this section. Figure 4 shows a block diagram of the r and phi coordinate encoder function.

The output of the flash ADC is sent to both the L1 buffer and the coordinate encoding function. The full precision data (8 bits) which will be used for physics analysis waits in the L1 buffer until the L1 decision comes back from the global LHCb trigger. The trigger data will be corrected for common mode, which is obtained from adding the signal amplitude from 32 neighboring channels, and discriminated against a threshold per channel. The threshold could also take into account the position of the initial analog value in the L0 pipeline. The output of the discriminator is a 32-bit pattern, one bit per channel. Several 32-bit patterns will be input into a simple clustering and address encoding function. This last function delivers a data block with the coordinates of the physical hits in units of half of the maximum common strip pitch denominator (10 μm). The event identifier is added into the header of the hit list block.

Each of the steps of the described function should take less than 1 μs in order allow a maximum average input rate of 1 MHz. The output trigger data block will be transmitted to the electronics barracks by an optical connection with a sustained throughput of at least 1 Gbit/s.

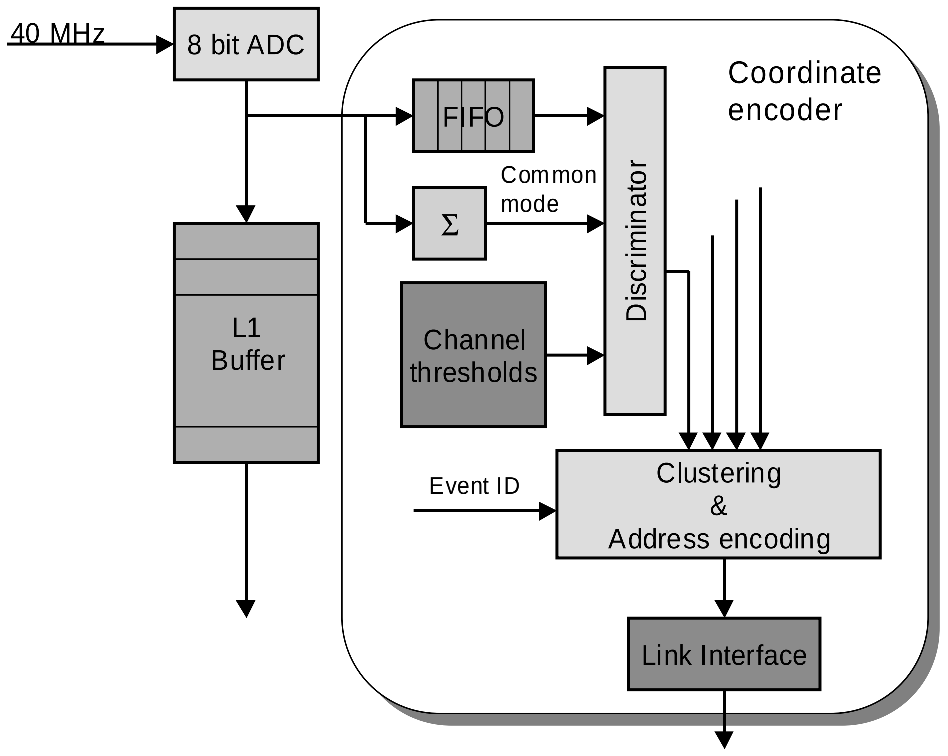


Figure 4 Block diagram of the clustering and r & phi coordinate encoding function in the front-end module.

## 5.4 The switch

The main requirement for the switch is that we need to send one event fragment from each data source (vertex detector station or another convenient partition) to a single destination at a rate of 1

MHz and with a sustained throughput of 2 Gbytes/s. The implementation we describe fulfills this requirement but other implementations based on other technologies could also be possible.[1]

The switch comprises a series of independent horizontal and vertical buses, with one interface per bus and a series of dual port memories (DPMs) at every bus intersection, as shown in Figure 5. There is one vertical bus per data source (in our case a coordinate encoder), and one horizontal bus per data sink (which, in the limiting case of very fast CPUs would be one processor). To keep the individual bus throughput to similar levels between horizontal and vertical buses, a square design is called for with similar number of horizontal and vertical buses.

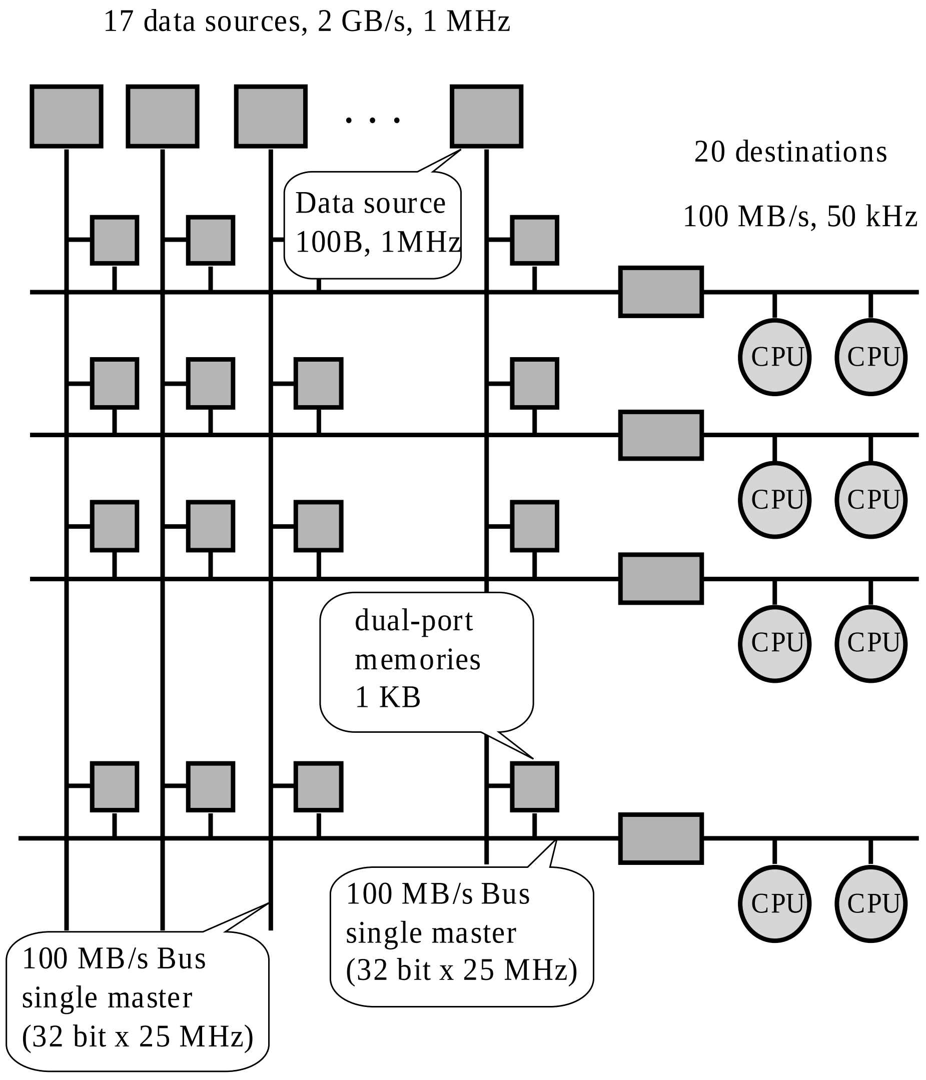

17 data sources, 2 GB/s, 1 MHz
. . .
20 destinations
100 MB/s, 50 kHz
Data source 100B, 1MHz
CPU
dual-port memories 1 KB
100 MB/s Bus single master (32 bit x 25 MHz)
100 MB/s Bus single master (32 bit x 25 MHz)

Figure 5 Architecture for the event builder switch with the main parameters for the different components.

[1] We do not know of any technology that could fulfill the fragment rate and data bandwidth requirements today. For example, Myrinet [6] could handle easily the bandwidth but not the fragment rate.

As seen in section 3.2, we envisage executing the trigger algorithm in about 100 µs on average. Since the input rate is 1 MHz, this would imply that the number of CPUs in the processor farm is about 100. This would in turn imply having 5 or 6 processors per horizontal bus.

The interface to the horizontal or vertical bus is the only master on the bus, thus making the design of the bus very simple. Both buses have similar requirements in terms of throughput. If one assumes 32 bit wide synchronous buses the required frequency is 25 MHz for a maximum data throughput of 100 Mbytes/s. Designing the buses for an average load of 50% would require speeds of 50 MHz, easily feasible with current technology[2].

The dual-port memories sitting at the crossings for each vertical and horizontal bus will be used to store temporarily the event fragments during the event building. The interface to the vertical bus pushes the event fragments into addresses which map different dual-port memories for each event in a round-robin fashion. Each vertical interface acts asynchronously with respect to the others. As soon data arrives from the front-end, it is pushed it into the appropriate dual-port memory (see Figure 6).

The role of the horizontal interface is to transfer all data fragments of the same event into a CPU memory. The horizontal transfer can be initiated as soon as the first data fragment is present or may wait until all fragments are there. The latter requires waiting for the longest fragment (maximum 3 times the average). Mechanisms for checking if the complete event is present can be implemented with a single bus cycle. A simple local algorithm using the number of events queued on each processor determines the final processor destination where the data is pushed.

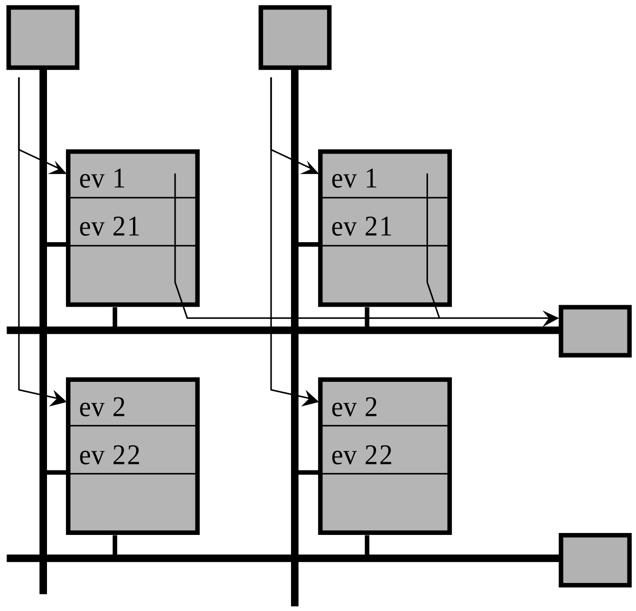


Figure 6 Detail of the operation during event building.

[2] The current PC motherboards run with a system bus at 66 MHz.

Each dual-port memory needs to be large enough to keep all data that has not yet been processed without overflowing. Its exact depth will be defined from the simulation studies. The size of the pre-allocated blocks depends on the maximum event size. At this point, we envisage allocating a 1024 byte block per event, and space for 100 events, thus, each dual-port memory will be 100 kbytes.

## 5.5 The processor farm

All the processors required to execute the algorithm are organized into 20 sub-farms. Each sub-farm runs independently of the others. The rate of input events to each sub-farm is 50 kHz, which is the original input rate divided by 20.

Each processor in the sub-farm will be a commercially available processor (PC like) and will be connected via a high performance link to the horizontal bus interface. The minimum required aggregated throughput for such an interconnect is 100 Mbytes/s. Several technologies are available at this moment that should be able to provide the required throughput. Examples are SCI, Myrinet, etc.

### 5.5.1 The sub-farm controller

The sub-farm controller acts as a bridge between a horizontal bus in the switch and the network connecting all the processors on the sub-farm. The main function is to assemble event fragments from the dual-port memories and send complete events to processors. It can also be used, as it is shown in Figure 8, to collect the results from the processors and put them into a FIFO to be picked up by the local trigger supervisor using a vertical bus. In this case, since it will know which processors have finished processing, the assignment of the processor could take into account the number of events scheduled and completed. This will offer a simple load-balancing scheme for each sub-farm.

### 5.5.2 The software program

The main task running in each processor will be the trigger algorithm, but it will probably not be the only task running there. Other tasks, for example the task monitoring the trigger process, will be required. For this reason, it will be adequate to run a real-time micro kernel in the processors to provide a multi-tasking environment.

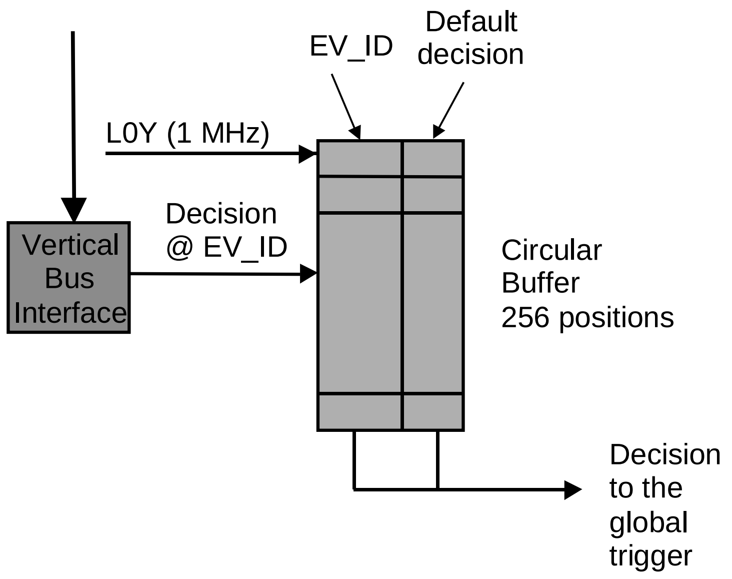
Handling time-outs is an important issue that needs to be studied in some detail. This is important because we can not afford a processor taking an excessively long time to process an event since other events will be queuing for it. If the processing is taking too long or it is predicted that it will take too long, the algorithm task needs to be interrupted and the current event processing aborted in an orderly manner.

The alignment constants required by the trigger algorithm will be downloaded at the start of a run. Nevertheless it might be important to monitor the alignment and eventually update the constants in real time. This could be another example of a low priority task running concurrently with the trigger algorithm.

## 5.6 Local trigger supervisor

The local trigger supervisor is a specialized hardware processor that receives trigger decisions from the processors and sorts them into order. It also provides an interface to the global trigger and to the DAQ system. The DAQ will read out the summary of the results of the trigger processing from the local trigger supervisor as any other sub-detector. This processor must work at 1 MHz input and output rate with a relatively small amount of data. The minimum data required is the event identifier and a bit for yes or no. The proposed implementation of the local trigger supervisor can be seen in Figure 7.

The required throughput into the local trigger supervisor is a few words at 1 MHz, that is some tens of Mbytes/s. One way of implementing the connection of all the CPUs is to put another vertical bus in the switch (see Figure 8). Each processor could deposit the results of the event being processed into a pre-defined memory location or simply into a FIFO. The local trigger supervisor reads the



results from the interface.

Figure 7 Implementation of the local trigger supervisor. Fixed-length circular buffer (FIFO) with random access to update event decision.

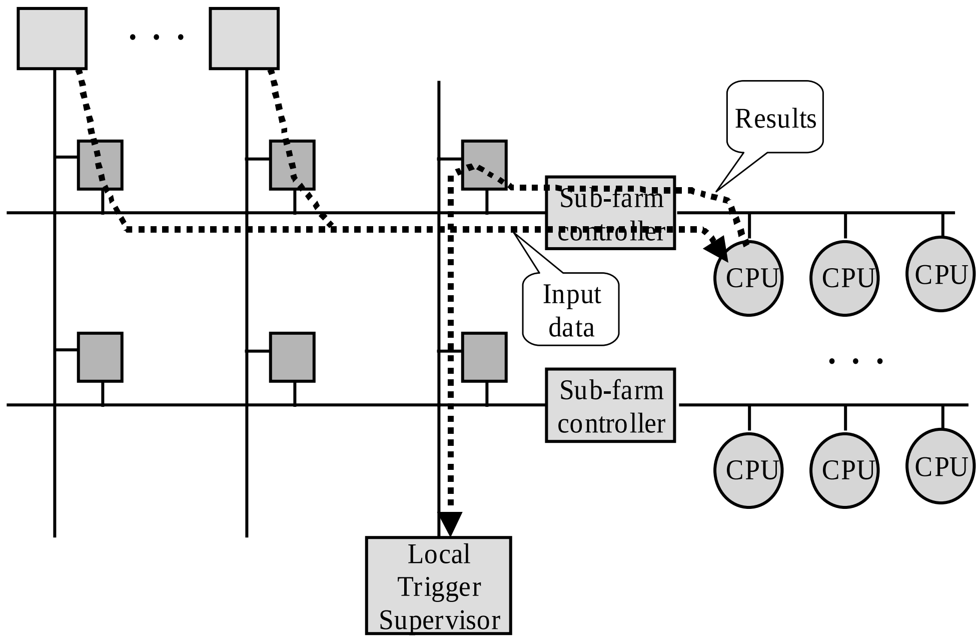


Figure 8 Connectivity of the local trigger supervisor. This diagram shows the possible implementation of the connection of all the processors using the same switch. The dotted lines show the path of the trigger data and of the results.

# 6 Performance and Cost

## 6.1 Trigger performance

The trigger performance of the system is basically given by the performance of the algorithm. Since everything is executed in software on standard processors there is no reason to think that the performance will be any different from that obtained using the algorithm off-line. Small discrepancies could come from the precision of the arithmetic operations or from the re-organization needed to optimize the algorithm or from the usage of special instructions to run as fast as possible.

## 6.2 Latency estimates

Table 2 shows our first estimates for the latency of the different steps in the trigger processing. More accurate numbers can be obtained using the simulation of the model that we are planning to produce using the Foresight modeling tool. This simulation should take into account the influence of previous events on the latency for the current event.

| | **Minimum** | **Typical** | **Maximum**[3] |
|---|---|---|---|
| Coordinate encoding | 3 μs | 3 μs | 3 μs |
| Writing data into memories | 0.2 μs | 1 μs | 3 μs |
| Wait complete event | 0.5 μs | 2 μs | 3 μs |
| Reading data from memories | 4 μs | 20 μs | 50 μs |
| Algorithm processing time | 30 μs | 120 μs | 250 μs |
| Collecting results | 5 μs | 5 μs | 5 μs |
| TOTAL | 43 μs | 151 μs | 314 μs |

Table 2 Estimates of the average latencies of the different phases

## 6.3 Flexibility, scalability and upgradability

One of the strengths of this design is its flexibility: There are no constraints as to what algorithm one uses in the trigger system since all data are available to the processor for every event. The design would work with x-y strips as well as R-phi strips, it could deal with more than one interaction per bunch crossing or with the presence of a magnetic field in the vertex region.

Such a modular design is also scalable, in that it can easily incorporate more data sources or more processing units without any redesign. The upgradability comes mainly from the fact that faster CPUs can easily be incorporated, either to reduce the number of processors or to be able to use a more efficient (and therefore longer) algorithm.

## 6.4 Cost estimates

Table 3 shows a first estimate of the cost for the vertex trigger. We have not included the cost of the coordinate encoding, which will need to be included in the front-end cost estimates of the vertex detector. The unit price is a mixture of current prices and extrapolated prices, in particular in the case of the cost for processors. Manpower cost is not included in the table.

[3] The maximum time will be in certain cases artificially cut to avoid very long tails. This truncation should affect a very small number of events (< 1%).

| | Units | Unit Cost | Cost [kCHF] |
|---|---|---|---|
| Optical links (1 Gb/s) | 17 | 1 | 17 |
| Switch input ports | 17 | 5 | 85 |
| Switch output ports | 20 | 5 | 100 |
| Crates 9U | 1 | 15 | 15 |
| CPU boxes (1000 Mip+1 GB) | 120 | 3 | 360 |
| CPU Interface (SCI) | 120 | 0.8 | 96 |
| Local Trigger Supervisior | 1 | 50 | 50 |
| Software licences | | 20 | 20 |
| **TOTAL** | | | **743** |

Table 3 Breakdown of the cost for the vertex trigger

## 7 Future work

A conceptual simulation of the design is underway. The plan is to go to higher levels of detail as the design matures. A careful error condition handling study will also be performed. In particular, buffer overflow exceptions, missing fragments of events, processing time-outs, etc. need to be studied in some detail and included in the simulation. Some of the issues here are common to the general DAQ design and we intend to collaborate closely with the DAQ group in areas of common interest.

A monitor system should be developed to spy on processor and switch performance. Part of the processor memory should be visible to the monitoring process as well as the buffer levels of the dual-port memories.

Alternative, offline-style trigger algorithms should be explored to study what we can gain in efficiency with a (substantial) increase in algorithm complexity. Then, working back, we can see how to simplify the algorithm until it fits into our available latency budget with minimal loss in performance.

## 8 Conclusions

A complete proposal for the implementation of the vertex detector first level trigger has been presented. It utilises an almost control-free crossbar switch for event building coupled to a processor farm that would run the trigger algorithm entirely in software. All data rates and other performance considerations are within easy reach of today's technologies and the whole system could be built today if needed. The design is also very flexible, scalable and upgradable.

# 9 References

**[1]** P. Mato, *"Vertex Trigger User Requirement Document"*, in preparation.

**[2]** H. Dijkstra, T. Bowcock, *"Vertex Detector*", Chapter 11 of LHCb Technical Proposal, 1998.

**[3]** LHCb simulation program based on GEANT 3.21.
M. Goosens et al.. *"GEANT - Detector Description and Simulation Tool"*, CERN program library long writeup W5013, CERN, 1995.

**[4]** H. Dijkstra, T. Ruf, *"The L1 Vertex Trigger Algorithm and its Performance"*, LHCb/98-006, 1998.

**[5]** H. Müller, *"Vertex Trigger Implementation using Shared Memory Technology"*, LHCb/98-xxx, 1998.

**[6]** Myrinet is fast network technology from Myricom Inc., Arcadia, CA. (USA). See http://www.myri.com