Chapter 11

The BTeV Trigger

11.1 Introduction

The BTeV experiment includes a sophisticated trigger system that rejects 99.9% of light-quark background events, while retaining large numbers of B decays for physics analyses. The design of the trigger supports BTeV's goal of studying a broad range of B decays using many different B-tagging techniques. To be competitive with other projects engaged in B physics (including those that will run concurrently with BTeV and those that are planned for the future), BTeV aims to maximize the number of B decays available for physics analyses by taking advantage of the high-resolution three-dimensional tracking data provided by the pixel vertex detector, by using a consistent trigger strategy throughout all stages of the trigger system, by analyzing every bunch crossing to search for evidence of a B decay, and by deploying a trigger system that is fault tolerant and fault adaptive.

In this chapter, we begin with an overview of the trigger system in Section 11.2 and the BTeV trigger requirements in Section 11.3. We provide technical descriptions of trigger algorithms in Section 11.4. The sections that follow, 11.5 and 11.6, provide details of the trigger hardware, trigger R&D and simulation results. Discussions of supervision and monitoring for the trigger system, and the RTES (Real Time Embedded Systems) Project are presented in Sections 11.7 and 11.8, respectively. We conclude with a discussion of our production plan in Section 11.9.

11.2 Overview

The trigger system is crucial for the success of BTeV. An important feature of the BTeV trigger (one that distinguishes our trigger from ones used in other experiments), is that it finds B events with high efficiency in the first stage of triggering (referred to as Level 1) by taking advantage of the key property that differentiates B (and charm) particles from other types of particles, namely their characteristic lifetimes. To achieve this, BTeV must analyze every bunch crossing by performing track and vertex reconstruction to search for

evidence of a particle-decay vertex within a few hundred microns to a few centimeters away from a primary interaction vertex. In practice, this is done by reconstructing all primary vertices and selecting events that have additional tracks with large impact parameters with respect to the nearest primary vertex. Other experiments [1] use a fairly simple "first level" of triggering. These types of triggers are able to reduce the number of events so that subsequent trigger levels have more time to deal with the surviving events, but they also restrict the types of final states that are accepted by the experiment, thereby limiting physics analyses. Trigger strategies that require the presence of specific final-state particles, such as muons, or demand the presence of a few high-p_t hadrons, are examples of this. By avoiding these restrictive trigger strategies at Level 1 and by exploiting the characteristic lifetimes of B particles at the first and subsequent trigger levels, the BTeV trigger is able to maintain high efficiency for B events throughout the entire event selection process.

The trigger system consists of three levels: L1, L2, and L3. Each level contributes to the reconstruction of events, and successive levels impose more and more refined selection criteria to select B events and reject light-quark background events. The trigger is designed to run at an initial (peak) luminosity of $2\times10^{32}\mathrm{cm}^{-2}\mathrm{s}^{-1}$ with a 132 ns, 264 ns, or 396 ns bunch crossing interval, corresponding to an average of 2, 4, or 6 interactions per crossing respectively. The L1 trigger operates at the Tevatron bunch-crossing rate and is the most demanding part of the trigger system. It consists of the L1 pixel trigger, L1 muon trigger, and Global Level 1 (GL1). The L1 pixel trigger reconstructs tracks and vertices for every bunch crossing. It is able to carry out track and vertex reconstruction at the bunch-crossing rate because of the very high-quality, low-noise, three-dimensional tracking information provided by the pixel detector. The data from the BTeV spectrometer are read out and processed in a large number of parallel data pipelines so the total processing time for data associated with a particular bunch crossing can be far greater than the bunch crossing interval, but the L1 trigger must produce trigger decisions with a time-averaged rate that is less than the bunch crossing interval.

The L1 trigger reduces the data rate by a factor of approximately 50 by rejecting background events while retaining B events with high efficiency for the next trigger level, L2. The L2 trigger improves the track and vertex reconstruction by reviewing the pixel data used at L1, and by including additional pixel hits in reconstructed tracks if necessary. L2 can also access additional data, such as data from the forward-tracking system, to further improve and refine track and vertex reconstruction. L2 reduces the data rate by rejecting at least 90% of bunch crossings that are sent to L2. At L3 all of the data for a bunch crossing are available. We perform a complete analysis of the data. This is comparable to the offline analysis performed by other high-energy physics experiments. L3 imposes the selection criteria for the final trigger decision and rejects at least 50% of the bunch crossings sent to L3.

The BTeV three-level trigger system is shown in Fig. 11.1. The figure shows a box at the top that represents the one-arm BTeV spectrometer. Data are read out from front-end electronics and are sent to the first-level trigger (L1) and to Level-1 buffers. An L2/3 crossing switch, which is part of the data acquisition system (see chapter **??**), is used to route data

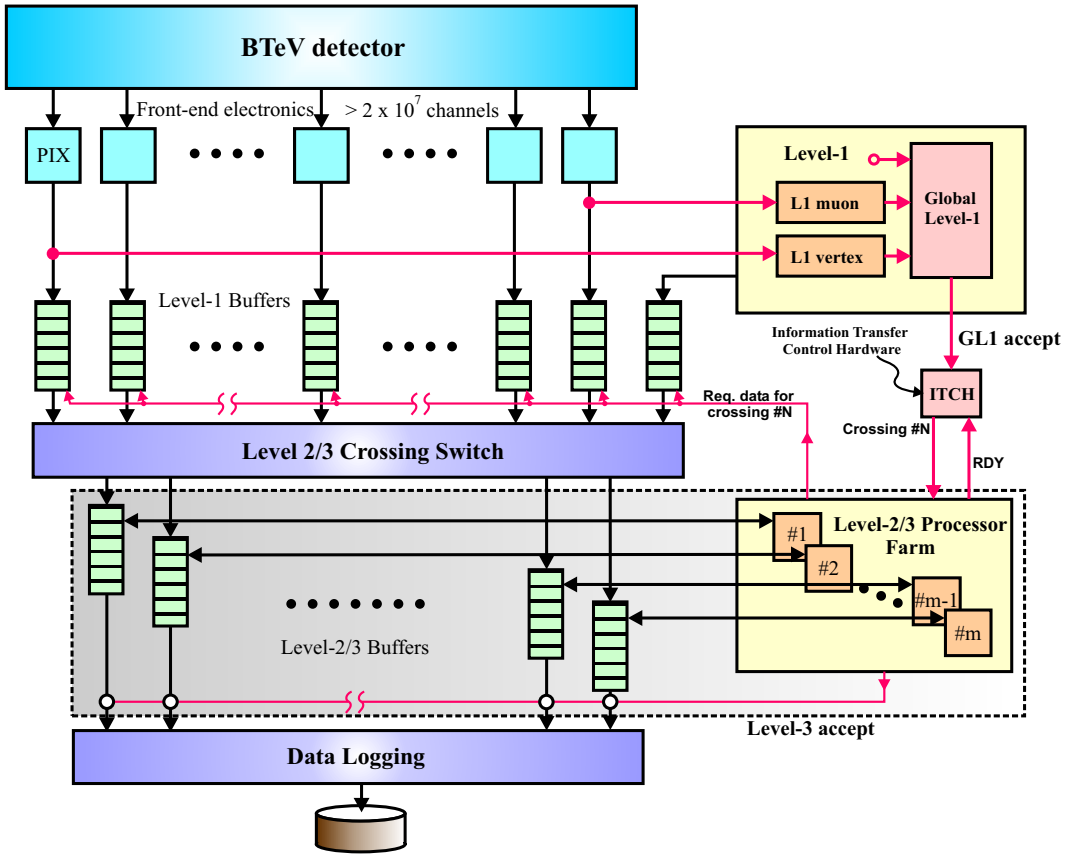

Figure 11.1: BTeV Three-Level Trigger Architecture

to the second and third level triggers for bunch crossings that satisfy the L1 trigger. Bunch crossings that satisfy L2 and L3 selection criteria are logged on archival media.

The trigger levels differ in the amount of time that is allotted for data processing, in the detector data that are available for processing (for L1, only the data from the pixel and muon detectors are available), and in the type of hardware used to process the data. L1 has the least amount of time available for processing and it uses a variety of hardware including field-programmable gate arrays (FPGAs) and digital signal processors (DSPs). L2 has more time to process data (due to data reduction that occurs at L1) and it uses a farm of Linux PCs. L3 has the longest amount of time available for processing and it uses the same farm of Linux PCs. Since the farm is used for both L2 and L3 it is referred to as the L2/3 trigger farm.

The distinction between the types of calculations that are performed at each trigger level are not rigid, and there is considerable flexibility in the design of our three-level scheme. This is especially true for calculations that are performed on data from the pixel detector. The pixel data, which are considered crucial for the success of the experiment, are used at all levels in our three-level scheme. The pixel data are analyzed by the L1 trigger in four pipelined stages: pattern recognition, track reconstruction, vertex reconstruction, and

impact-parameter calculations that form the basis for the L1 trigger decision. In our baseline design the bulk of the pattern recognition for pixel data is performed by FPGAs, which excel at performing large numbers of rudimentary calculations in parallel. The remaining L1 calculations are performed by DSPs, which offer more programming flexibility than FPGAs and have excellent I/O capabilities. The pixel data are also analyzed by subsequent trigger levels, L2 and L3. Here the I/O requirements are less critical (data rates are lower compared to rates in the L1 trigger), and we have decided to use general-purpose microprocessors (Linux PCs) with even greater programming flexibility. The flexibility of the overall design becomes apparent when one studies particular calculations and investigates how these calculations can be performed at different stages in the trigger architecture. For example, in Section 11.5 we describe two alternatives for pattern recognition that significantly reduce the processing time required by the DSPs. The first is a "hash sorter" that can be implemented in an FPGA that is already performing other functions (buffer management) while reducing the load on the DSPs. The second alternative further reduces the load on the DSPs by moving the remaining calculations that complete the pattern recognition for pixel data from DSPs to FPGAs. Additional studies of this kind involving the optimization of the trigger architecture are underway and will help to improve the overall design of the trigger system.

The design of the trigger has evolved over time. Many of the changes that have been made were introduced to accommodate changes in the pixel detector, and in some instances modified trigger algorithms have in turn led to significant improvements in the design of the pixel detector. An example illustrates the interdependence of the trigger and pixel detector and the importance of concurrent development of the two systems [2]. For the BTeV Proposal [3] the pixel detector was modified from three pixel planes per tracking station to two planes per station. The removal of an entire plane from each tracking station led to a reduction in cost, a reduction in material, and a pixel detector that would be easier to build. However, this change in the pixel detector was only adopted after it had been proven that a new L1 trigger algorithm was able to satisfy all trigger requirements without loss of physics capabilities. Other changes in the design of the pixel detector have been made, and each change was accompanied by considerable effort to modify trigger algorithms, repeat simulations, and tune algorithm parameters to maximize physics capabilities.

We anticipate using a variety of commercially available hardware for the BTeV trigger system, and we continue to monitor developments in commercial electronics. L1 uses a very large number of advanced commercial components in our baseline design, mainly FPGAs and DSPs. The hardware will only improve with time, and we can adapt our designs to take advantage of faster hardware with higher throughput. As processors get faster there will be a tendency to reduce the number of processors in the L1 trigger. This might require higher capacity network links at L1, but can also be addressed by moving calculations from a higher trigger level, such as L2, to L1. In this manner we will take advantage of our trigger architecture to find a good balance between the hardware that will be available when the experiment is ready to run, the trigger algorithms necessary to process the data, and the physics needs of the experiment.

Finding a balance between available hardware and required physics computations has

been an ongoing effort during the R&D and design phase of the BTeV trigger project. In the sections that follow we present our current baseline design for the BTeV trigger system. This design has evolved over time, and will continue to evolve as new hardware becomes available. Some of the strategies that have been considered, evaluated, and abandoned (for the time being) are the following:

- We considered using general-purpose processors for the entire trigger system, but found that the cost would be prohibitive. In particular, we found that the pattern recognition for the pixel data required far too many compute cycles on a general-purpose processor.
- We investigated the use of different types of DSPs for the L1 trigger. In particular, we studied the use of fixed-point DSPs for the pattern recognition for the pixel data, and found that we required a large number of DSPs for pattern recognition.
- We have studied other pattern-recognition algorithms that would be implemented largely with FPGAs. Our current algorithm (which is also based on an FPGA design) requires only a fraction of all pixel hits by finding two track segments for each particle trajectory: one track segment at the beginning of a trajectory (as the particle enters the pixel detector), and one track segment as the particle exits the pixel detector. Other algorithms that we have studied used all of the pixel hits associated with a particular track. One of the algorithms found three-station track segments throughout the pixel detector but suffered from significant increase in data volume as the data progressed through several stages in the trigger hardware.

We continue to explore alternatives to improve the design of the trigger system.

11.3 Requirements

This section describes the requirements for the BTeV trigger system. The requirements are based on detailed investigations of algorithms and technology believed to help the experiment achieve its physics goals while being both affordable and technically achievable. An understanding of the requirements presented in this section may require an understanding of details of the trigger system presented in subsequent sections in this chapter. We encourage the reader to refer to later sections for additional information.

11.3.1 Rate Requirements

The rate requirements for the trigger are determined by the running conditions of the Tevatron and the physics goals of the experiment.

- *L1 trigger rate:* The L1 trigger must make trigger decisions at the bunch-crossing rate every 132 ns, 264 ns, or 396 ns on average.

- *Number of interactions per bunch crossing:* The BTeV trigger must be able to handle an average of 2, 4, or 6 interactions per bunch crossing (depending on the bunch-crossing rate), and maintain good efficiency for B events accompanied by an average of 2, 4, or 6 minimum bias events, respectively.

11.3.2 Algorithm Requirements

BTeV requires the ability to study a broad range of B-decays including B_d, B_u, B_s, B_c, and all types of b-baryons. We intend to study these decays in a wide variety of final states including those that have only charged hadrons, those that have charged and neutral hadrons, photons and π^0s, electrons, and muons.

- *Trigger algorithm:* The BTeV trigger must base its decision on the characteristic properties of B decays so as not to limit the physics potential of the experiment. These properties must be determined well enough so that any needed corrections do not unduly limit the physics reach of the experiment.

11.3.3 Output Data Rate Requirements

Studies of the cost of data storage and retrieval have led to the conclusion that the output data rate should be no more than 200 MBytes per second. This corresponds to 2 PetaBytes per Snowmass year (10^7 seconds of running the experiment).

- *Output data rate:* The output data rate of the trigger must average to no more than 200 MBytes/sec at peak luminosity.

11.3.4 Rejection and Efficiency Requirements

The efficiency of the trigger depends on the particular parent B particle and its final state decay topology. The efficiency is defined for a particular analysis that is tuned to have acceptable signal to background to achieve the physics goals of the analysis. The events that survive the analysis cuts represent the sample of events used to determine the trigger efficiency. The efficiency is defined as the fraction of these events that survive the trigger. Extensive simulations have shown that an efficiency of greater than 50% can be achieved for most decay topologies with at least two charged hadrons associated with the B vertex, or the B vertex and an associated charm vertex. In cases with only one charged particle emerging from the B vertex and a non-charm decay such as a K_s, or a charm decay that has only one prong, the efficiency will be somewhat lower but should be at least greater than 20%. This determination of the trigger efficiency assumes 100% pixel efficiency and does not take into account the trigger livetime and uptime, which have their own requirements and are described in Section 11.3.6. Although we believe that pixel efficiencies close to 100% are achievable, we require that the trigger efficiency does not drop below 45% as long as the pixel detector is operating within its performance envelope.

Simulations have also shown that an average event size (after sparsification by the readout or front-end electronics) of less than 200 KBytes per bunch crossing is achievable at a 396 ns crossing time and an average of 6 interactions per bunch crossing. This represents the amount of data per bunch crossing for events that are analyzed by the L1 trigger. Further reduction in the event size can be obtained by eliminating some information associated with non-B interactions, summarizing and condensing the remaining information, and possibly applying a compression algorithm somewhere between the trigger and data storage. The result is that we expect the average amount of data per bunch crossing written to archival storage to be approximately 80 KBytes at a 396 ns crossing time and an average of 6 interactions per bunch crossing. This implies an output rate of 2500 bunch crossings per second. In this document we give numbers for both a 396 ns crossing time (an average of 6 interactions per crossing) and for a 132 ns crossing time (an average of 2 interactions per crossing). At a 132 ns crossing time we would expect the average amount of data per bunch written to archival storage to be about 50 KBytes, corresponding to an output rate of 4000 Hz. This implies that one must handle an output rate (see Section 11.3.3) up to 4000 bunch crossings per second.

- *Output crossing rate:* The BTeV trigger must accept bunch crossings at a rate compatible with the maximum output data rate (see Section 11.3.3). It is expected that the trigger will accept no more than 4000 bunch crossings per second.

- *Efficiency:* The efficiency of the trigger must be greater than 50% (not including livetime or uptime, and assuming 100% pixel efficiency) for nearly all B decays having two or more charged particles emerging from the B or daughter charm vertex. The trigger efficiency must be greater than 45% when the pixel detector is operating within its performance envelope.

- *Single prong efficiency:* The efficiency for less well defined states such as those with a single prong and a K_s must be greater than 20%.

- *L1 rejection:* The L1 rejection must be greater than 98% of all bunch crossings.

- *L1 efficiency:* The L1 efficiency, as defined above, must be greater than 55% for B decays having two or more charged decay particles. The efficiency must be greater than 50% as long as the pixel detector is operating within its performance envelope.

- *L2 rejection:* The L2 rejection must be greater than 90% of bunch crossings sent to the L2 trigger.

- *L2 efficiency:* The L2 efficiency must be greater than 90%.

- *L3 rejection:* The L3 rejection must be greater than 50% of bunch crossings sent to the L3 trigger.

- *L3 efficiency:* The L3 efficiency must be greater than 95%.

11.3.5 Global Level 1 Requirements

Global Level 1 (GL1) refers to the hardware and software that combines results from all L1 triggers (such as the L1 pixel and L1 muon triggers) and data from front-end electronics to select the bunch crossings that pass the first level trigger. GL1 also includes the Information Transfer Control Hardware (ITCH), which assigns L1 accepted bunch crossings to L2/3 processing nodes. The data that are sent to GL1 are referred to as trigger primitives. The trigger primitives are not required to arrive in any particular order. GL1 must decide when it has all the trigger primitives that it needs for a particular bunch crossing.

- *GL1 trigger rate:* GL1 must accept trigger primitives from L1 processors and front-end electronics at the full bunch-crossing rate, and must produce trigger decisions every 132 ns, 264 ns, or 396 ns on average (depending on the bunch-crossing rate).

- *GL1 trigger list:* GL1 must be able to inspect the trigger primitives for a bunch crossing and test against a group of conditions, called a *trigger list*, to see if the crossing satisfies one or more of the conditions. It must apply a prescale factor to each crossing that has satisfied a particular trigger. GL1 must then OR the results to determine whether the crossing satisfies the L1 trigger for that list.

- *GL1 partitioning:* To support partitioning of the trigger and data acquisition system, GL1 must be able to maintain multiple trigger lists and must be able to arbitrate if a particular bunch crossing satisfies more than one trigger list.

- *GL1 data packet:* GL1 must create a data packet, or packets, for each accepted bunch crossing. The data packets contain information that specifies which trigger lists were satisfied by the crossing, and contain the data from all trigger primitives suitably merged. The data packets must be buffered within GL1 and/or in a Level-1 buffer so that the data are available to higher-level triggers and recorded as part of the data written to archival storage.

- *GL1 prescale:* For diagnostic and monitoring purposes, GL1 must select events according to a prescale scheme and declare them as accepted. GL1 must record that these events were selected in this manner in the GL1 data packets.

- *GL1 signals to DAQ:* GL1 must issue any signals that might be needed by the data acquisition system (DAQ) to delete bunch crossings from Level-1 buffers for crossings that do not satisfy the L1 trigger. Similarly, GL1 must issue any signals needed by the DAQ to preserve crossings that satisfy the L1 trigger so that they are available to higher-level triggers.

- *GL1 statistics:* GL1 must maintain statistics required for the diagnosis of problems, for calculating deadtime, and for monitoring luminosity. It may need to obtain information about the luminosity from another source to determine whether dynamic prescaling is appropriate, but the current goal is to have GL1 keep statistics that can be used to determine the luminosity.

- *GL1 throttling:* GL1 must contain a mechanism for dynamically throttling and/or prescaling some triggers so that the triggers with a higher priority are taken. It must do this based on information concerning luminosity and the availability of processors and buffer memory.

- *GL1 latency:* In order to facilitate operation, it is permissible for GL1 to set a small "minimum L1 latency," which is guaranteed. This will set the time available for the formation of triggers and the generation of front-end trigger primitives. The absence of valid data from the front ends will not delay the trigger decision.

- *GL1 timeout:* In order to avoid problems with buffer memory and with GL1 itself, it is permissible to impose a "timeout" after which GL1 will make a decision based on whatever information it possesses. The system will accept some prescaled selection of crossings that fail to have all information available. These crossings are used for subsequent evaluation of the impact of these losses on the physics. If this situation is sufficiently rare, all such failures could be declared to pass the trigger. GL1 needs to record in the GL1 data packets that the trigger accepted a crossing due to a timeout, and must maintain statistics on the frequency and nature of such occurrences.

- *GL1 accounting:* GL1 must account for all bunch crossings ensuring that each was accepted, rejected, or reported as missing. This accounting must be done within an amount of time that allows corrective action to be taken if the rate of lost data becomes unacceptable.

- *GL1 assignment:* GL1 will assign each bunch crossing that satisfies a trigger list to one L2/3 processing node.

- *GL1 and run control:* GL1 must respond to Run Control commands and must be able to support partitioning (see Section 11.3.12).

11.3.6 Livetime and Uptime Requirements

The trigger performance can be affected adversely by factors both within the trigger and outside of it. Factors outside of the trigger system include an unusually "dirty" beam that produces extra background in the detector, badly imbalanced bunch populations, poorly performing or noisy detectors, etc. These "external factors" have the effect of reducing the "length" of the data pipeline as measured in bunch crossings and can therefore cause the front-end electronics to run out of buffer space during the trigger decision time, or can cause the trigger decision time to be longer than average due to noisier than average events. "Internal factors" include high failure rates for processors or bottlenecks in the data-transport network, which would prevent processors from being used efficiently. Any of these problems can cause the processing to fail to keep up with the bunch crossing rate and must inevitably result in data being discarded and therefore lost.

Terms used in this section are deadtime/livetime and uptime/downtime. During uptime the trigger is performing normal operations, processing incoming data with trigger algorithms and producing output. During downtime the system is unavailable. During livetime the trigger is performing normal operations on all assigned bunch crossings, processing normally within the time and bandwidth requirements. No data is left unprocessed, no data is lost, and no other system operation is hindered or impeded by the trigger operation. Deadtime occurs when data is permanently lost to the experiment due to overflows, timeouts, or errors.

- *Livetime:* With a suitably pipelined architecture, the trigger should impose almost no deadtime on the experiment. We set the livetime requirement based on factors *internal* to the trigger to be greater than 95%.
- *Uptime:* The trigger uptime must be greater than 95%.

11.3.7 Excess Capacity and Scalability Requirements

While extensive simulations have been and continue to be carried out to verify the trigger performance, it is impossible to predict exactly what reality will be like. It is therefore important that the trigger has extra capacity built in, and that the architecture be such that additional expansion is possible with incremental funding.

- *Capacity:* The trigger must have at least 50% extra capacity.
- *Scalability:* The initial implementation of the trigger architecture must permit an increase in capacity of at least a factor of 2 in processing and data throughput at every level, and a factor of 2 in the size of buffer memories.

11.3.8 Flexibility Requirements

The trigger architecture must be capable of including possible expansion to satisfy new physics goals that might arise in the future.

- *Flexibility:* It must be possible to add additional types of triggers.

11.3.9 Fault Tolerance and Security Requirements

Fault tolerance and redundancy must be designed into the architecture of the trigger system. Moreover, the trigger system must have adequate computer security for computing systems that are part of the trigger and are accessible from systems external to the trigger.

- *Component failure:* The trigger must continue operating, perhaps at reduced capacity and efficiency, in spite of the failure of a number of the processors, network connections, etc.

- *Trigger arbitration:* If the trigger system is having difficulty keeping up with the data rate, it must be able to drop less important triggers and calculations so that it can maintain high efficiency for the most important physics.

- *Purging:* The trigger system must be able to purge data when the data rate is too high. A mechanism that logs data loss due to purging must be implemented.

- *Timeouts:* Timeouts must be used to impose limits on excessive calculations that tie up resources. The system must monitor and record the frequency of timeouts as a warning and diagnostic tool.

- *Data duplication:* Within the trigger system, bunch-crossing data will be moved and processed without duplication.

11.3.10 Detector Performance Envelope

Detectors never perform perfectly. BTeV detectors will have specific performance envelopes that must be achieved and maintained. The trigger must be able to achieve its efficiency and rejection goals when all of the detectors are within their performance envelopes. There should also be enough overhead for the trigger to degrade gradually if one or two detectors move slightly outside of their worst acceptable performance.

- *Performance envelope:* The trigger must achieve its efficiency and rejection requirements (see Section 11.3.4) when all detectors that affect the trigger performance are within their performance envelopes.

11.3.11 Supervision and Monitoring Requirements

The trigger supervision and monitoring system is interfaced to the global control and monitoring system for the BTeV experiment. It is responsible for the supervision and monitoring of all hardware in the trigger system. Examples of functions performed by the trigger supervision and monitoring system include programming for Field Programmable Gate Arrays (FPGAs), programming for L1 and L2/3 processors, in-situ diagnostics and testing, performance monitoring, as well as status and error message reporting. In addition to performing these functions, the system must be able to receive commands from and report back to the BTeV control and monitoring system. It must record any unusual conditions or problems in the trigger and report them.

- *Supervision and monitoring:* The trigger supervision and monitoring system must receive commands from and report back to the BTeV control and monitoring system.

- *Read-back for programmable devices:* The supervision and monitoring system must be able to read data from programmable devices in L1, L2, and L3. This includes the

programs, parameters, device configurations, status and error messages, any temperature and voltage measurements, as well as processed data at useful probe points in the data stream.

- *Initialization:* The supervision and monitoring system must be able to configure trigger hardware and software.

11.3.12 Support for Commissioning and Debugging

The trigger system must be able to support not only steady-state operation but also commissioning and runtime troubleshooting. It will be necessary to test new trigger algorithms and new trigger hardware (for example, burning in a new set of processors). During commissioning, several detectors may be running almost autonomously for debugging or calibration purposes. The trigger system must be able to support partitioning. This means that it must simultaneously provide several different and independent triggers for different subsystems and allocate data to different groups of processors.

- *Partitioning:* The trigger system must be able to support partitioning.
- *Testing:* All programmable devices that are part of the trigger must be testable in-situ.
- *Database:* A resource identification and configuration database must be used to keep track of hardware and software used in the trigger system.

11.3.13 Electrical Requirements

The hardware that is designed and built, or purchased to implement the trigger will consist of digital electronics.

- *Electronics standards:* The trigger system must comply with the *BTeV Digital Electronics Standards* document.

11.3.14 Software Requirements

The "software" for the trigger system refers to algorithms, specialized operating system code, as well as diagnostic and testing code developed for programmable devices in the three trigger levels, Global Level 1, and the supervision and monitoring system. The software includes FPGA firmware, software developed for specialized processors, and software developed for general purpose processors. In addition, the Real Time Embedded Systems (RTES) project will contribute software that will provide error handling, reporting and recovery techniques to enhance the reliability and robustness of the system. This software will be integrated at several levels into the trigger system, but it will only be allowed to use a small amount of the available processing resources. All of the software must be developed using standard software tools that allow version tracking, assist code reviews, and help with maintainability.

- *Error detection:* Processes that regularly verify code purity and run standard datasets for testing purposes must be part of the development process, and must be used for testing the trigger after the development of the trigger has been completed.
- *Software Database:* Trigger parameters and constants (such as prescale factors, geometry and alignment constants) must reside in a database so that the particular values used to process data can always be identified.
- *Software repository:* All software must reside in a software repository that must be used to keep track of different versions of the software during development.
- *Version control:* The version numbers of software used to process data must be managed in such a way that the particular version that was used to process the data can always be identified and reproduced.
- *Processing resources:* RTES error detection, recovery and reporting software, as well as other types of control and monitoring software will use less than 10% of the processing resources of any trigger system unit on which it is installed.

11.3.15 Safety Requirements

The trigger system does not pose safety concerns beyond the usual and customary issues associated with large systems of low-voltage digital electronics.

- *Low voltage, high current safety:* Where the trigger electronics uses high-current (greater than 10 amps operating or 50 amps rated current) low-voltage (less than 50 volts) power supplies, the safety requirements for low-voltage, high-current power distribution systems must be followed. These requirements are detailed in the Fermilab ES&H Manual, Occupational Safety And Health section on Electrical Safety.
- *General safety:* The trigger system must comply with all applicable Fermilab ES&H safety standards.

11.4 Technical Description and Implementation

11.4.1 Overview

We begin this section with an architectural overview of the BTeV trigger system, describing the basic components of the 3-level trigger system and how they interact with each other [7]. We will also show how the trigger system integrates into the larger picture of the BTeV data acquisition system. For more details on various components of the data acquisition system relevant to the trigger system, please refer to Chapter **??**. This will then be followed by more detailed subsections focusing on each of the major trigger components: L1 pixel, L1 muon, GL1, and L2/3. These subsections will describe the algorithm and hardware aspects

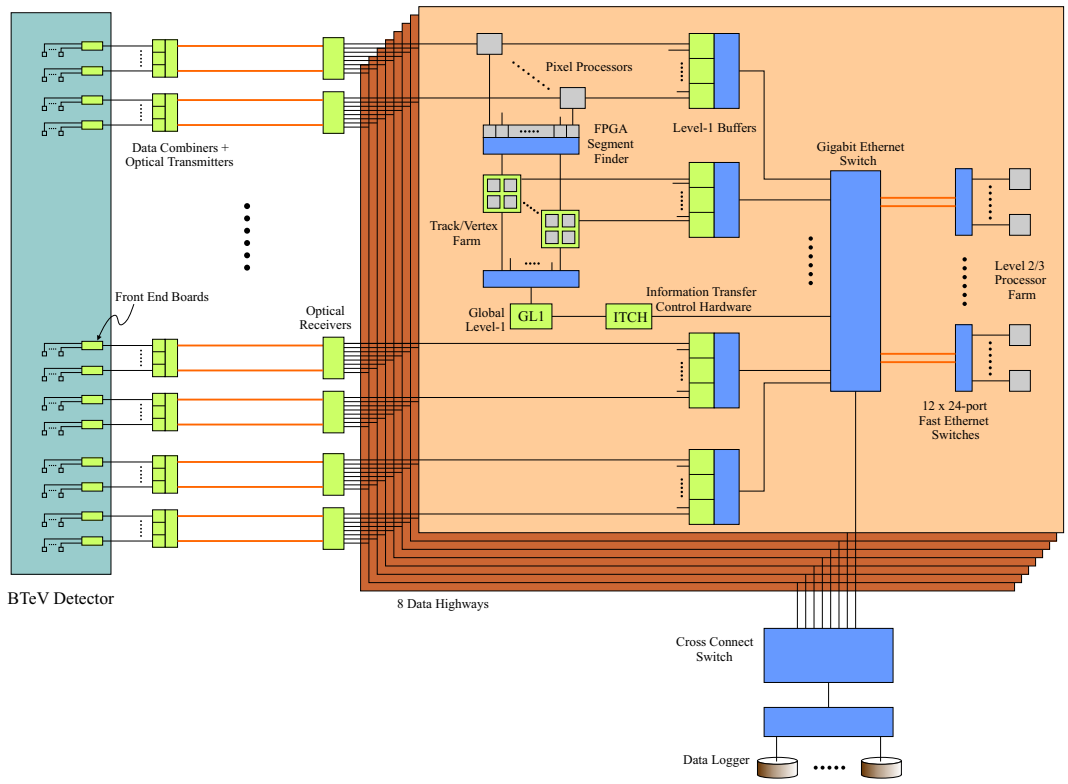

Figure 11.2: BTeV Three-Level Eightfold-way Trigger Architecture.

of each component. Finally, the next section will describe completed and ongoing R&D work on each of these trigger components.

The trigger system consists of five subsystems:

- *L1 pixel trigger:* The primary trigger for the experiment. It receives data from the pixel detector.
- *L1 muon trigger:* The trigger that selects J/ψ and prompt muon events and is used to calibrate the L1 pixel trigger. It receives data from the muon detector.
- *GL1 trigger:* Global Level 1 (GL1) receives data from the L1 pixel trigger, L1 muon trigger, and front-end electronics. It selects the bunch crossings that pass the first level trigger. It also includes the Information Transfer Control Hardware (ITCH) that assigns accepted bunch crossings to L2/3 processors.
- *L2/3 trigger:* The L2/3 trigger consists of the L2 and L3 algorithms running together on a processor farm (L2/3 trigger farm). A bunch crossing that is analyzed by a particular processor and satisfies L2 selection criteria is processed by L3 trigger algorithms without requiring a transfer to a different processor. Bunch crossings that fail are eliminated. The distinction between L2 and L3 is that L3 has access to all of the data for a particular bunch crossing, while L2 operates on a subset of the data.

- *Supervision and monitoring:* The supervision and monitoring system (not shown in Fig. 11.1) consists of separate subsystems for each of the four trigger subsystems: PTSM, MTSM, GL1SM, and L23SM for the L1 pixel, L1 muon, GL1, and L2/3 triggers, respectively.

BTeV's 3-level trigger architecture is shown in Fig. 11.2. Data from the detector's front-end electronics are sent at the full crossing rate from the collision hall to the counting room via high-speed optical links. Optical receivers in the counting room distribute the data for each bunch crossing uniformly to one of eight parallel data paths called "highways", each of which forms a complete and independent 3-level trigger system. The full data rate from the detector is ∼500 GB/s assuming a crossing rate of 2.5 MHz with an average of 6 interactions/crossing, and an average event size of 200 KB. The eightfold highway architecture reduces this full data rate to 62.5GB/s into each highway allowing the use of low-cost components such as commercially available ethernet switches. The choice of *eight* parallel data paths (as opposed to some other number) is used for the baseline design. The design is flexible and modular enough to reconfigure to other choices of numbers of parallel data paths if needed, due for example to possible changes in running conditions or to any changes based on bandwidth studies or cost considerations.

Note that in this document the default trigger and data rates we give are for a 396 ns crossing time (2.5 MHz crossing rate) and an average of 6 interactions per bunch crossing. The BTeV Trigger system has been designed to also handle a 132 ns crossing time so some rates are also given for a 132 ns crossing time (7.6 MHz crossing rate) and an average of 2 interactions per bunch crossing.

In each highway, data from all detector subsystems are temporarily stored in Level-1 buffers as trigger decisions are made. The Level-1 buffers consist of commodity SDRAM with enough memory to buffer ∼300 thousand crossings in each highway corresponding to ∼1000 ms (over 3 orders of magnitude more than the average L1 processing time) of L1 trigger decision time. Aside from going to the Level-1 buffers, data from the pixel and muon detector are also sent to the L1 pixel and muon triggers, respectively. Since the hardware for the L1 muon trigger is very similar to the L1 pixel trigger hardware, only the latter is shown in Fig. 11.2 for simplicity.

Before they are sent to the Level-1 buffers, pixel detector data pass through pixel processors that perform various operations on the data such as time-stamp expansion, pixel hit clustering, and x-y coordinate lookup. The processed data are also sent to an FPGA based segment tracker that executes the segment finding stage of the L1 trigger algorithm (see Section 11.4.2). Inner and outer track segments from the same crossing found by this stage are then routed by a network switch to one node in a farm of over 300 programmable embedded processors that execute a C version of the track and vertex finding stage of the L1 trigger algorithm. Complete processed results from each node are routed to the Level-1 buffers while summarized trigger results are sent to a GL1 processor responsible for the ultimate trigger decision. These decisions will be stored as a list of accepted crossing numbers in the Information Transfer Control Hardware (ITCH) which broadcasts accept messages to all Level-1 buffers.

The L1 muon trigger will be based on the same hardware used for the track and vertex farm of the L1 pixel trigger. The GL1 processor will combine L1 muon trigger decisions with those from the pixel trigger to form the final trigger decision.

The Level-1 buffers are grouped into 32 subsystems, each of which feed a port in the highway switch consisting of a commercial 64-port gigabit ethernet switch. 12 pairs of ports from this switch feed the paired gigabit uplink ports of a dozen 24-port fast-ethernet fanout switches. The fanouts, in turn, feed data to 96 commodity dual-CPU Linux-PC's that make up the L2/3 processor farm in each highway where the DRAM in each PC functions as an L2/3 buffer.

Requests for data are sent by an idle L2/3 node to the ITCH which responds by assigning one or more accepted crossing numbers to that node. Once it receives its assignment, the L2/3 node sends a request to a subset of the Level-1 buffers which respond by sending their data to that node. The actual number of crossings assigned for each transfer can depend on the running conditions, the optimal transfer packet size will be selected for each transfer. All requests and data transfers between the L2/3 farm and the Level-1 buffers and the ITCH are routed through the highway and fanout switches. Upon receiving the data, the L2/3 node executes the L2 trigger algorithm. If the event passes L2, the L3 algorithm is executed in the same node. L3 will be similar to the offline reconstruction, doing refined tracking and particle identification to improve upon the results from L2. L3 will perform some data reduction by dropping some raw data and summarizing some physics data, also some amount of data compression may be necessary.

If the event passes the L3 trigger, the processed results are propagated back up the fanout and highway switches to an external cross-connect switch that routes accepted events from all 8 highways to a small cluster of data-logging nodes for archival storage.

The L1 trigger rejects at least 98% of all crossings, reducing the input data rate of 62.5 GB/s into each highway to 1.6 GB/s into the L2/3 farm. The L2+L3 trigger rejects 95% of the L1-accepted crossings so that the data rate out of L2/3 is a mere 25 MB/s, with a reduction of a factor of $\approx$3 in the data size per bunch crossing. This 25 MB/s out of each highway amounts to a total of 200 MB/s going into the data-logging nodes.

Table 11.1 summarizes the trigger and data rates at each of the three levels of the BTeV Trigger System for running at a crossing every 396 ns and an average of 6 interactions per bunch crossing. The processing time (latency) per DSP/CPU is also given. The required maximum latencies imply a certain minimum number of DSP/CPUs, the table also gives these numbers of processing units when one assumes 100% usage of a processing unit for the trigger application. The baseline design is based on a larger number of processing units with a more realistic assumption for the DSP/CPU usage.

11.4.2 L1 Pixel Trigger

11.4.2.1 Algorithm

The L1 pixel trigger employs a two stage algorithm consisting of a segment-finding stage followed by a track and vertex finding stage. For a detailed description of the algorithm,

Table 11.1: Summary of trigger and data rates at each of the three levels of the BTeV Trigger System. The numbers are for running a crossing every 396 ns and an average of 6 interactions per bunch crossing. The processing time is per DSP/CPU per event (crossing). The actual processing is of course essentially deadtimeless with the use of pipelines.

Trigger Level	**Trigger Parameters**				
	Input rate	**Output rate**	**Reduction Factor**	**Processing Time (ms)**	**Minimum # DSP/CPU's**
Level 1	2.5 MHz 500 GB/s 200 KB/evt	50 KHz 12.5 GB/s 250 KB/evt	50	0.33	825 (@100%) 2500 (@33%)
Level 2	50 KHz 12.5 GB/s 250 KB/evt	5.0 KHz 1.25 GB/s 250 KB/evt	10	5	250 (@100%) 416 (@60%)
Level 3	5.0 KHz 1.25 GB/s 250 KB/evt	2.5 KHz 200 MB/s 80 KB/evt	2	134	672 (@100%) 1120 (@60%)

please refer to Ref. [2, 4]. The pixel vertex detector consists of an array of 30 stations of silicon pixel planes perpendicular to the beamline and distributed over about 125 cm along the interaction region. Each station contains two planes: one with the pixels oriented so the narrow pixel dimension is horizontal (bend plane) and one with the narrow pixel dimension in the vertical direction.

First Stage: Segment Finding

The segment finding stage, which is sometimes referred to as the pattern recognition stage, finds the beginning and ending segments of tracks in two separate regions of the pixel planes, an inner region close to the beam axis and an outer region close to the edge of the pixel planes. It restricts the search for the beginning and ending segments to these inner and outer regions, respectively. Since segments are found using hit clusters from three adjacent pixel stations in the defined regions, beginning segments are referred to as inner triplets while ending segments as outer triplets.

The search for inner triplets begins by looking for the two most upstream hits which will be referred to as inner doublets. These doublets are found using all combinations of hit clusters within defined regions of the bend plane of two adjacent stations. The upstream hit which is restricted to the inner region is paired with a hit from the downstream station and the straight line segment joining the two hits is projected upstream by a distance equal to the separation distance between pixel stations. If the x-y coordinates of the projection upstream come within or close to the edge of the beam hole in the center of the pixel station, the segment is then projected downstream to predict the position of a hit in the bend plane

of the third adjacent station. If a hit cluster lies within a given distance of the predicted position, the algorithm then checks to see if there are at least two confirming hits in the non-bend planes of the three adjacent stations. If all these requirements are satisfied, the hits are then grouped together as an inner triplet.

The search for exterior triplets proceeds in a similar fashion. Exterior doublets are found using combinations of hits from the bend plane of two adjacent stations where the downstream hit is restricted to the outer region of the pixel plane. The straight line segment joining the two hits is projected upstream by one station and the x-y coordinates required to lie within pixel plane boundaries. The segment is then projected downstream by one station and the x-y coordinates required to lie close to or beyond the edge of the pixel plane. For doublets that meet these requirements, the upstream projection is used to predict the position of a hit in the third station upstream of the doublet. If a hit cluster lies within a given distance of the prediction, the three hits are grouped together as an outer triplet without requiring confirming hits in the non-bend planes of the three stations.

Second Stage: Track and Vertex Finding

Once the inner and outer segments are found, the algorithm proceeds to the second stage which is referred to as the track and vertex finding stage. This stage consists of two separate phases. The first phase, referred to as the segment matching phase, attempts to match inner to outer triplets belonging to the same track. To do this, inner triplets are projected downstream based on their slopes in the non-bend plane and their curvatures (slope change) in the bend plane. Outer triplets are matched to inner triplets to form complete tracks based on their proximity to these projected trajectories.

After the segment matching phase, the algorithm then proceeds to the vertex finding phase. It begins this phase by processing the tracks found from the previous phase, calculating their momentum from the curvature and knowledge of the B-field and calculating its transverse distance from the beam axis. It then loops over all tracks with $p_T < 1.2$ GeV/c and which appear to originate close to the beamline to search for primary interaction vertices. The first track in the loop is considered a seed track for a new cluster of tracks. The second track is then paired with the first to see if they form a cluster by calculating their point of closest approach (vertex position) and requiring the track to come within a given distance of this point. If the track meets this requirement, it is added to the cluster. If not, the track is used as a seed for a new cluster. This procedure is repeated for all subsequent tracks. If more than one cluster exists, a new track is attached to that cluster for which its proximity to the vertex position is smallest. At the end of the search, the cluster with the most number of tracks is then considered a primary interaction vertex. The algorithm loops through all tracks attached to the non-primary clusters to see if they fit the primary interaction vertex attaching them to this vertex if they do.

All tracks attached to the primary vertex are then tagged and the whole procedure described above to search for primary vertices is repeated using the remaining tracks from the non-primary clusters. Once all the primary vertices are found, the algorithm searches for

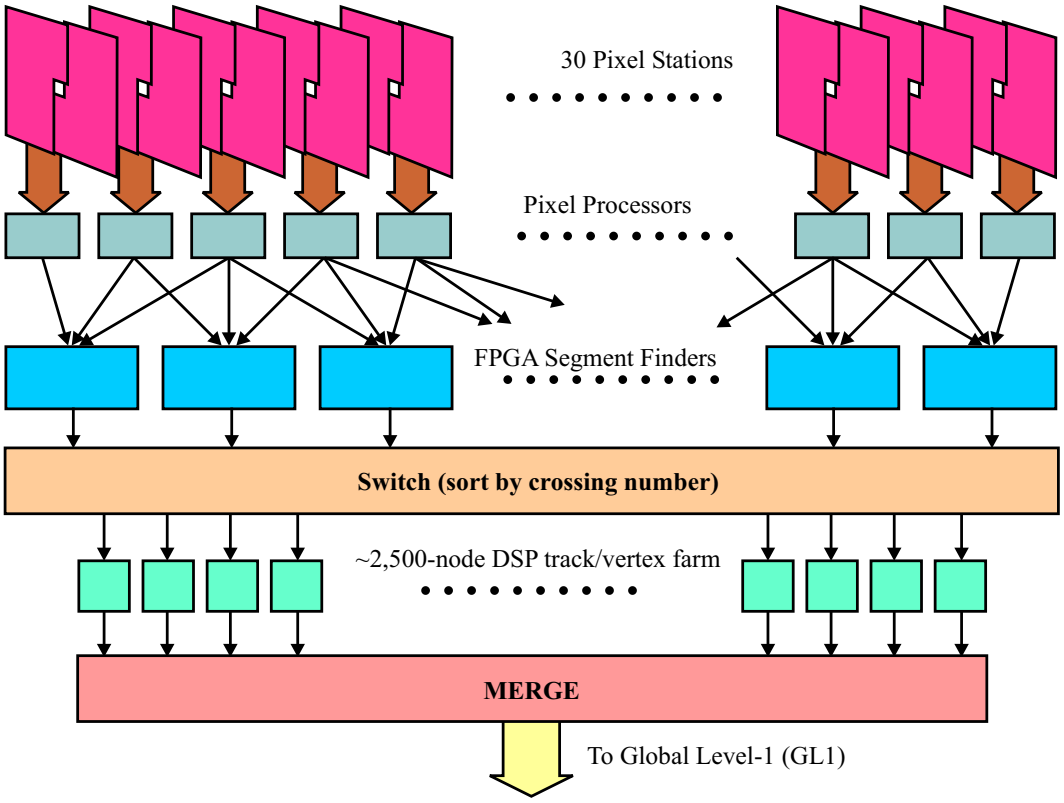

Figure 11.3: BTeV L1 Pixel Trigger Hardware Architecture

detached tracks by looping over all tracks not attached to any primary vertex and calculating its impact parameter from each primary vertex. The detached track is then assigned to that primary vertex for which its impact parameter is smallest. An L1 vertex trigger accept is generated if there are at least 2 detached tracks in the same arm of the BTeV detector, associated with the same primary vertex, and satisfying the following criteria: $p_T^2 \geq 0.25$ (GeV/c)^2, $b/\sigma \geq n$, and $b \leq 2$ mm where b is the impact parameter, σ is the error in b, and n is a preset value ranging from 2-7.

11.4.2.2 Hardware

The hardware architecture of the L1 pixel trigger hardware is shown schematically in Fig. 11.3. As described above, data from the pixel detector front end is sent to FPGA based pixel processors that group individual pixel hits into clusters and translate the row column information into x-y coordinates. Hit clusters from three neighboring pixel stations are then routed to FPGA based hardware that finds the beginning and ending segments of tracks in the segment finding stage of the L1 trigger algorithm.

The segments or triplets found in this stage are then sorted by a custom switch according to crossing number and routed to a programmable embedded processor in the track and vertex farm. This processor performs the track and vertex finding stage of the L1 algorithm.

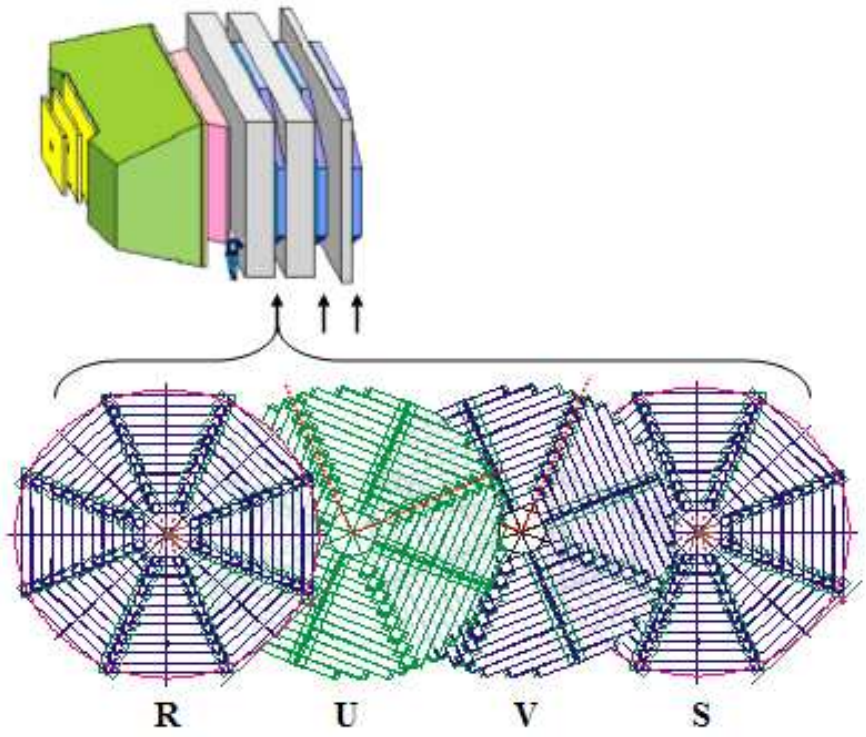

Figure 11.4: Schematic view of the BTeV Muon System. Divided into octants, each of the thee stations contains four views of detectors (R,U,V,S). Each view (per octant) contains 384 tubes in 12 planks.

Assuming an average processing time of 330 μs for the segment matching and vertex finding stage and an interaction rate of 2.5 MHz, a total of ~2,500 processors (~300 processors per highway) will be required for the track and vertex farm in order to examine every single bunch crossing with a 33% duty time on each DSP. Processed results will be sent to the Level-1 buffers while trigger summaries will be sent to the GL1 processor.

More details on the L1 pixel trigger hardware will be provided in subsequent sections where we describe the completed and ongoing R&D work on the L1 trigger.

11.4.3 L1 Muon Trigger

11.4.3.1 Introduction

The purpose of the L1 muon trigger is to look for tracks in the muon detector consistent with muons originating in the interaction region. The main job of this system will be to identify dimuon events in our quest to collect a huge sample of B decays containing J/ψ's, and to help calibrate the L1 pixel trigger.

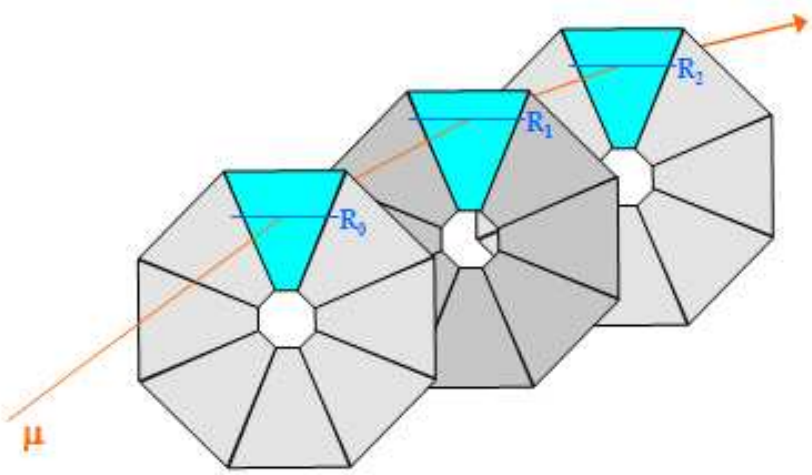

Figure 11.5: A muon track leaves hits in the R view of the top octant of all three muon detector stations.

11.4.3.2 Geometry

The layout of the muon system is shown schematically in Fig. 11.4. The entire system is segmented into equivalent octants. Each of the three muon detector stations contains four detector planes, referred to as views in what follows. Each view (per octant) is composed of 12 planks, each containing 32 tubes. Of the four views within each station/octant, two views (R and S) are oriented perpendicular to the radial direction, and the two other views (U and V) are rotated $\pm 22.5^o$ with respect to these.

11.4.3.3 Algorithm Overview

The purpose of the muon trigger is to identify muon tracks originating in the BTeV interaction region, and is accomplished by examining patterns of hit proportional tubes in the muon detector. The muon trigger system processes information from each octant independently, and within an octant, tracking is done independently on each of the four views.

To minimize combinations (i.e. processing time), adjacent hits within each view are sparsified by a simple algorithm that keeps only the "most central" tube (i.e. if tubes 11,12, and 13 are all asserted, the sparsification will keep only tube 12. If tubes 11 and 12 are asserted, only tube 11 is kept).

In order for track detection to be possible in a given octant/view, a track must leave hits in all three stations of that octant/view. The example illustrated in Fig. 11.5 shows a muon track leaving hits in all three R views of the uppermost octant.

Each muon which hits a specified octant/view in all three stations can be assigned a "co-ordinate" in the 3-dimensional space of this view. For example, the co-ordinate of the track in Fig. 11.5 is (R0,R1,R2), where R0 is the number of the proportional tube hit in

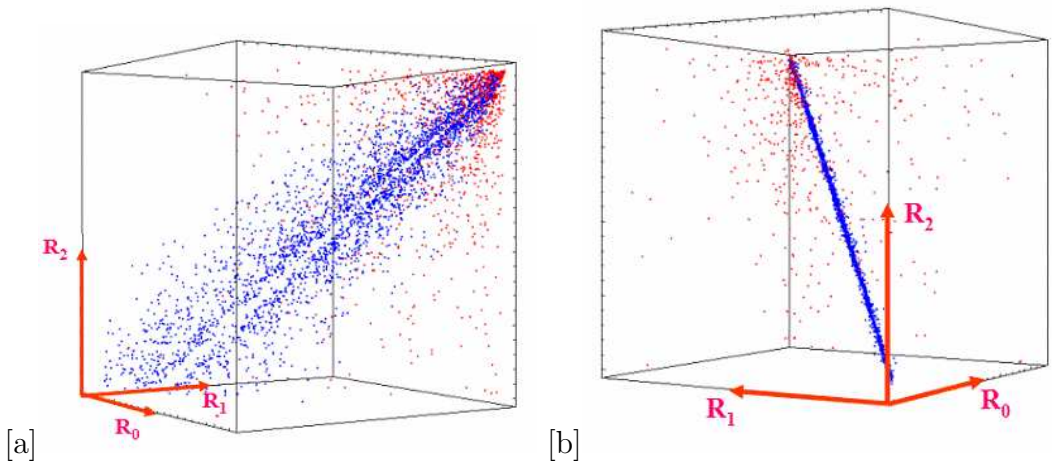

Figure 11.6: [a] Plot of (R0,R1,R2) for each of 2200 good muon tracks (blue dots) and the closest (R0,R1,R2) co-ordinate for each of 4300 minimum bias events (red dots). All octants are combined. [b] plot rotated to illustrate that all of the good muon tracks (blue dots) result in (R0,R1,R2) coordinates that lie on a well defined plane

station 0, etc. In what follows, the "R" view will be used as an example, but the identical approach is used to define tracks in the "U", "V" and "S" views as well.

Monte Carlo generated muons can be used to study the distribution of (R0,R1,R2) in the 3-dimensional R-space. The blue dots in Fig. 11.6[a] are the R-space coordinates (all octants combined) of about 2200 good muon tracks. These points lie in a very well defined plane - a fact that is more evident when examining Fig. 11.6[b], which is simply a rotated version of Fig. 11.6[a]. Note that the tubes are numbered from outside in: tube 0 is the outermost and tube 384 is the innermost in each view.

The equation of this plane can be found using a simple linear fit to the blue (good muon) points, and the perpendicular distance d to this plane from a given point (R0,R1,R2) can be easily calculated. Fig. 11.7 shows the d distribution for good muon events.

Given the minimal width of this distribution (the standard deviation of the entries in Fig. 11.7 is 1.5 tubes), we conclude that a simple plane in "R-space" does indeed provide a very good description of the (R0,R1,R2) coordinates of good muon tracks.

Although the results shown are for the R-view, nearly identical results are found when the same procedure is followed for the U, V and S views, hence we assume that tracking will be done in all 4 views using the same algorithm (albeit slightly different plane equations etc). For each view we define the normalized distance $D = d/\sigma$, where d is as described above and σ is the standard deviation of the d distribution for each view (i.e. the width of the peak in Fig. 11.7 for the case of the R view).

Cutting on D provides good rejection of minimum bias events because (R0,R1,R2) combinations from these do not tend to cluster around the "good muon plane" discussed above. This can be seen by examining the red points in Fig. 11.6[a] and Fig. 11.6[b], which indicate,

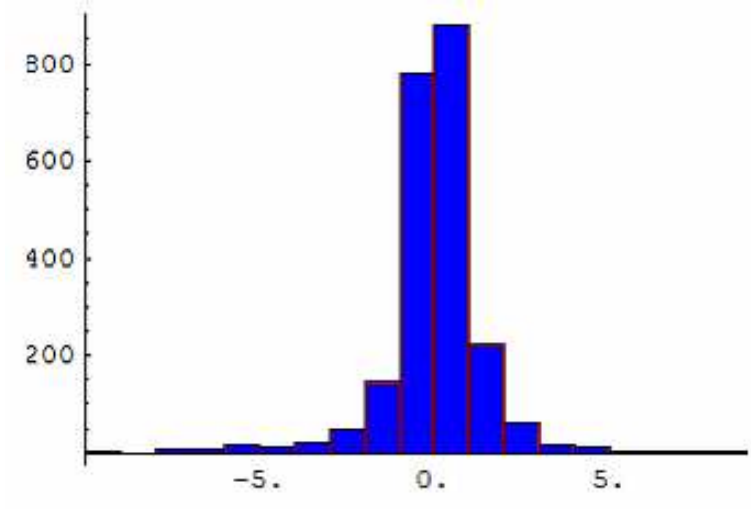

Figure 11.7: Distribution of distances d of each of the 2200 good muon (R0,R1,R2) coordinates from the plane defined by the best fit to the set of all such points, measured in tube number units.

for a sample of 4300 minimum bias events, the (R0,R1,R2) combination for each that is closest to the good muon plane.

Our strategy is thus to cut on D for each muon candidate, the tightness of this cut being a trade-off between efficiency and rejection (more on this below). Although not shown, the same discrimination is evident when examining the U-space, V-space, and S-space distributions.

11.4.3.4 Measuring Charge and Momentum

Upon closer examination of the good muon events (blue points) in Fig. 11.6[a], we find that useful kinematic information can also be extracted by simply considering the location of each point (i.e. track) within the plane discussed above. This is illustrated in Fig. 11.8[a], where the good muon tracks of Fig. 11.6[a] have been re-plotted such that the charge of each track is indicated by its color.

The charge separation is very clear, and indeed, nearly perfect charge identification can be obtained by considering only the (R0,R2) projection of this distribution. In other words, the correlation between the radial location of a track in the first and last stations gives good information about whether the track was bending inward or outward, which makes perfect sense. Fig. 11.8[b] shows the R2 vs R0 projection for the same events plotted in Fig. 11.8[a], along with the best fit line we used to determine the charge of muon candidates when simulating the dimuon trigger algorithm discussed below. Again, more or less identical results are found when examining the U, V and S views.

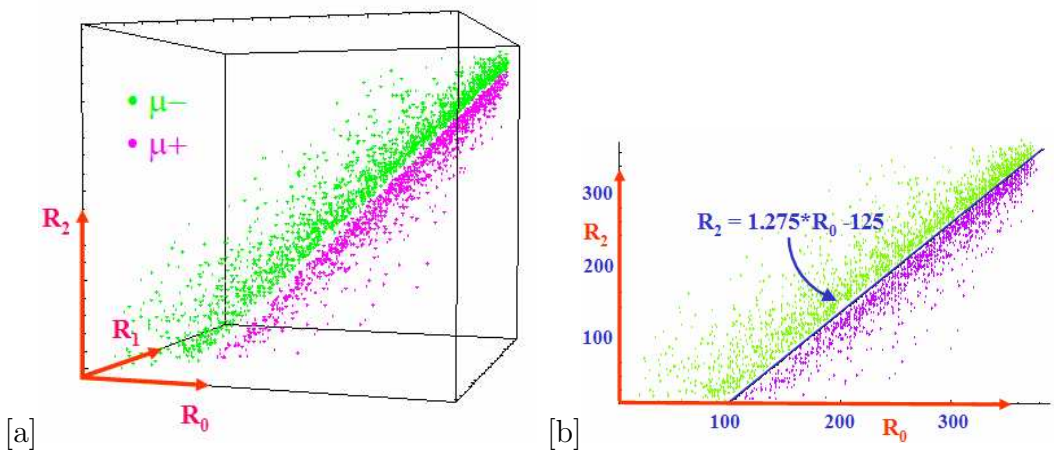

Figure 11.8: [a]Plot of (R0,R1,R2) for each of 1100 positive muon tracks (purple points) and 1100 negative muon tracks (green points). All octants are combined. [b]Plot of R2 vs R0 for each of 1100 positive muon tracks (purple points) and 1100 negative muon tracks (green points). All octants are combined. The blue line is a fit to the "gap" between positive and negative tracks.

Determining the charge of a track candidate is equivalent to measuring the sign of its curvature. The gap between positive and negative tracks that is evident in Fig. 11.8[a] and Fig. 11.8[b] is thus where tracks with zero curvature (i.e. infinite momentum) would live. We might therefore expect that the perpendicular distance of a point from this line is a measure of its curvature, hence inversely proportional to its momentum. Fig. 11.9, where the color of each dot indicates the total momentum of the good muon candidate it represents, gives qualitative evidence of this. No attempt has yet been made to use momentum discrimination in the muon trigger algorithm, although R&D to this end is planned.

11.4.3.5 Dimuon Algorithm

So far we have discussed the methodology for finding and determining the charge of single muon tracks. The thrust of the muon trigger will be to identify dimuon events, and the algorithm we have developed to accomplish this is discussed next.

Our preliminary (yet seemingly adequate) approach for triggering on $J/\psi \rightarrow \mu^+\mu^-$ is simply to look for events that contains at least one positive and one negative muon candidate, as illustrated in Fig. 11.10.

In each octant, the (R0,R1,R2) combination having the smallest absolute value of D is compared to a pre-determined D "cut", and if it passes the track-candidates charge is determined by examining (R0,R2) as discussed above. The same algorithm is used for all four views in an octant (R, U, V and S), and for each view the output from an octant is NO (no track found), POS (positive track found) or NEG (negative track found). If the charge determination within an octant is ambiguous (i.e. if different views give different answers),

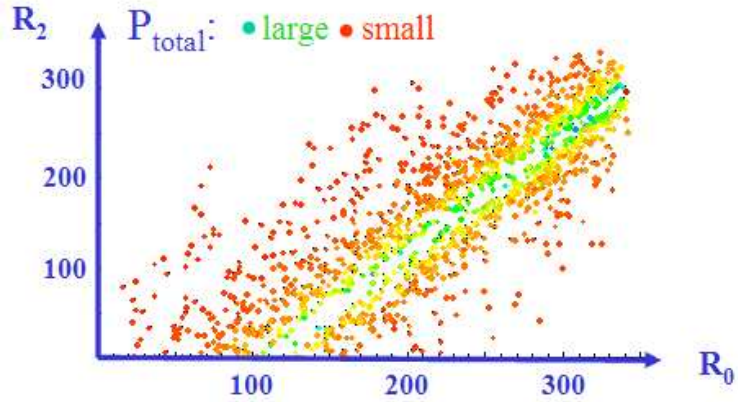

Figure 11.9: Plot of R2 vs R0 for 2200 good muon tracks. The color of the points indicate track momentum (red - small, green - large). All octants are combined.

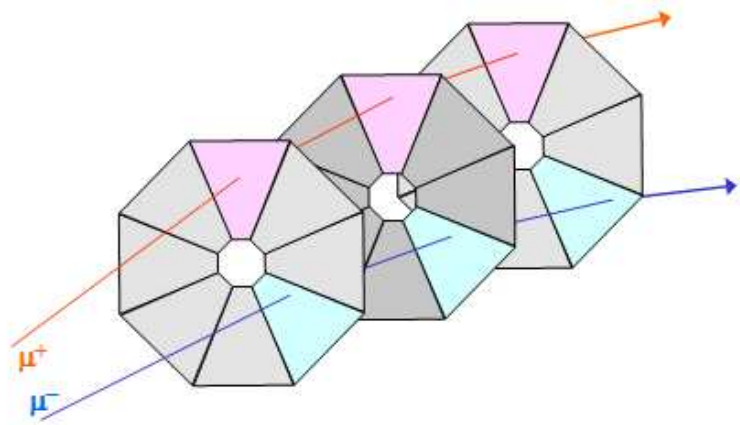

Figure 11.10: Schematic view of an event that would pass the dimuon trigger algorithm.

then the octant is allowed to count for either (but not both) charges required by the dimuon trigger.

11.4.4 Global Level 1 Trigger

The picture of the L1 trigger that we have presented is a simplified one involving only the vertex and muon triggers. The actual L1 trigger will be more complex involving a wider

variety of different triggers. This complexity is managed by the Global Level 1 Trigger (GL1).

First, both the vertex and muon triggers themselves can consist of several different triggers. For the vertex trigger, for instance, we have described a final selection that requires a minimum number of tracks to miss the primary vertex by a given number of standard deviations. In fact, there will be a variety of such triggers, some accepting events with a few tracks with large detachments and others accepting events with more tracks that have smaller detachments. We also want to record a sample of events that would otherwise have failed the trigger requirements in various ways in order to understand how the efficiency of the trigger "turns on." This last group of triggers will be prescaled.

Second, there will be triggers that combine information from more than one detector at GL1. For example, a single-high-p_t-muon trigger would be interesting but may have too high a rate. However, the GL1 trigger could accept a high-p_t single-muon trigger if the event also satisfies a relaxed vertex requirement — perhaps one track with a large impact parameter. (At L1, one would not know that the high-p_t muon corresponded to the single high-impact-parameter track).

Third, there will be special triggers. One example will be a variety of minimum- and low-bias triggers that will be heavily prescaled. Another example will be special alignment and calibration triggers. We will certainly collect special triggers or use minimum-bias triggers to do the quasi-real-time alignment of the pixel detector.

The main physics trigger will not be prescaled. GL1 must have the ability to prescale other less important physics triggers and calibration triggers. We also need the ability at L1 to adjust the prescale factors dynamically. For example, we want to be able to reduce the prescale factors on some of the triggers as the luminosity falls. We also want the ability to increase the number of alignment triggers taken at the beginning of a store and then reduce them once enough events have been collected to establish initial alignment constants for the store.

The operation of the GL1 trigger is as follows. The GL1 receives "trigger packets" from each trigger processor. These packets contain "trigger primitives" which are the data items from which GL1 makes the trigger decision. The packets, which have a format known to GL1, arrive asynchronously. The GL1 buffers these packets until it receives all the packets it is supposed to receive for a crossing, or until a timeout occurs for that crossing. In normal operation, as soon as all packets are received, GL1 extracts the primitives and generates all the various triggers from truth tables that have been downloaded. It then applies the appropriate prescale to each trigger and takes the OR of the result. If any of the triggers is satisfied, it issues an L1 accept. This results in the event data being transferred to the output buffers of the Level-1 buffers for eventual transfer to an L2/3 node. If no trigger is satisfied, the GL1 issues an L1 reject. This results in the Level-1 buffers being freed to be used for other events. In general, the trigger processors, which are exchanging messages with GL1, will have timeouts that are less than the GL1 timeout. If the allowed time for the arrival of an L1 trigger primitive expires before a trigger packet from a processor arrives,

an error flag will be generated. A prescaled sample of these events can then be recorded for further analysis and diagnostics. The rest of the GL1 decision proceeds as normal.

To minimize the number of designs that must be developed and maintained, the GL1 Trigger will be implemented using either hardware identical to that for the L1 track and vertex farms or conventional PC's.

11.4.5 L2/3 Trigger

The L2/3 trigger is a farm of commodity processors running Linux. After an L1 accept, all data for an event will be transferred from the Level-1 buffers to an L2/3 processor. If the event passes the L2 trigger it is then processed by L3 in the same farm node. The distinction between L2 and L3 is that L2 uses only the pixel data (or possibly pixel plus forward tracking) whereas L3 uses the full event data.

The input data to L2 consists of all L1 tracks and vertices and the raw pixel hits. The L2 algorithm performs a Kalman filter track fit on the L1 tracks and refits the primary vertices. It then searches for other primary vertices and detached secondary vertices. A secondary vertex must satisfy the following criteria: (i) tracks must have a confidence level greater than 2.5%, must be detached from the primary vertex by more than 3.5σ and must have transverse momentum > 0.5 GeV; (ii) all tracks must have the same sign p_z and be pointing away from the primary vertex; (iii)the vertex must have a confidence level greater than 2% and must be detached from the primary vertex by more than 3.5σ ; (iv) the vertex must have an invariant mass less than 7 GeV and more than 100 MeV outside the K_s mass. An event passes L2 if it has either a detached secondary vertex or a high p_T detached track. Current simulations show that we can achieve an L2 efficiency of about 90% for B decays of interest while rejecting more than 90% of light quark events.

The goal of L3 is to achieve another factor of 2 in background rejection and to reduce the size of the event by a factor of 3 at a 396 ns crossing time and an average of 6 interactions per bunch crossing. The L3 algorithm will be similar to the full offline reconstruction. We have prototype code for forward tracking, K_s reconstruction, particle ID in the RICH, and electron, photon and π^0 reconstruction in the calorimeter.

11.5 L1 Pixel Trigger Hardware and Simulations

11.5.1 Segment Finder Hardware

This section will discuss the completed and ongoing R&D work to implement the segment finding stage of the L1 trigger algorithm in hardware. We begin by presenting a detailed description of the segment finder algorithm within the context of hardware implementation.

11.5.1.1 Segment Finding Algorithm

As described in Section 11.4.2, the segment finding algorithm looks for track segments using hit clusters from the bend view of three adjacent pixel stations which we label planes *N-1*, *N*, and *N+1*. To find inner triplets, it begins by pairing hits from *N-1* that are within an inner region close to the beam axis with all the hits from *N*. The straight line connecting the two hits is projected back to *N-2* and its $x - y$ coordinates required to be within a window roughly corresponding to the beam hole of the pixel detector.

This first stage of the inner triplet phase is also known as the long doublet finding stage. For the hardware implementation, hits from *N-1* that are within an inner region and all hits from *N* are stored in two separate lists (memory locations). The requirement that the hits on *N-1* be confined to an inner region close to the beam axis is implemented in hardware with a cut operator that filters the hits going into the list. A reverse projection operator is then applied to each pair of hits on *N-1* and *N* to extrapolate back to *N-2*. Comparators are then used to see if the extrapolated position lies within the "beam hole". As will be described below, for each hit on N-1, these operations are done simultaneously with a whole list of hits on *N* using Associative Memory (AM) or Content Addressable Memory (CAM).

All pairs passing this stage are then sent to the second stage known as the triplet finding stage. In this stage, a forward projection operator extrapolates the straight lines connecting each pair downstream to predict the upper and lower limits of the window within which a hit on *N+1* must lie. A whole list of hits on *N+1* is then simultaneously checked against each set of window limits to find the third bend view hit in the triplet. Candidates for which a hit lies within the window on *N+1* are then sent to the third and final stage known as the short doublet finding stage. In this stage, projection operators are applied to the bend view hits found in the first two stages to predict the upper and lower limits of the windows within which hits must lie in these planes. Once again, each set of window limits is simultaneously compared with a whole list of hits in each non-bend view plane. Although this stage can be carried out in parallel for each of the three non-bend view planes, it is done sequentially to simplify the hardware for assembling the hits into a triplet. Since the basic operations carried out in each of the three stages describe above are identical, their hardware implementations are also identical.

The description above also applies to the search for outer triplets except that the last stage is not implemented since matching non-bend view hits are not required.

11.5.1.2 Implementing the Segment Finder in Hardware

We have made considerable progress in implementing the segment finding algorithm in hardware. The code for this algorithm is currently being written in Very High Speed Integrated Circuit (VHSIC) Hardware Description Language (VHDL). Taking the behavioral model of the segment finder algorithm and implementing it in VHDL is an important step toward determining its viability. It has been proven that the algorithm can be implemented in hardware and can meet the desired performance goals.

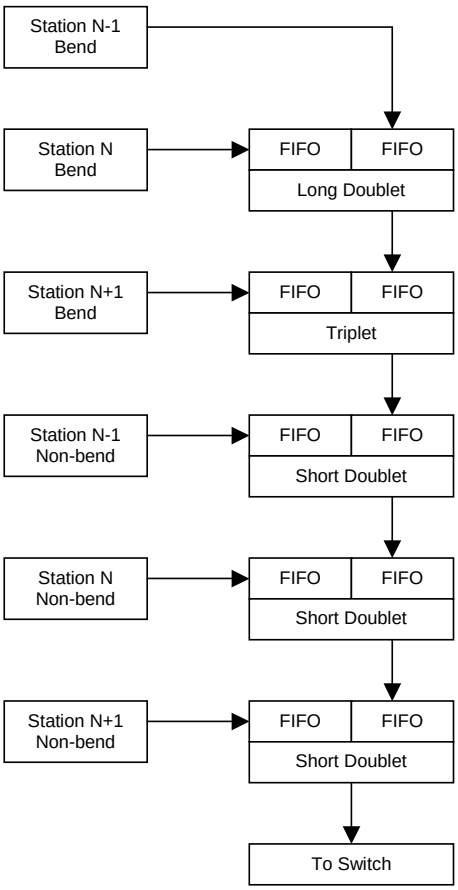

Figure 11.11: Block diagram of the segment finder algorithm as implemented in hardware.

VHDL was chosen for the implementation since it is a standard language and is widely supported by various electronic design automation (EDA) tools. The current plan is to use very high-density field programmable gate arrays (FPGA's) or complex programmable logic devices (CPLD's) for the design. Though the code is written in VHDL, the design also includes the Library of Parameterized Modules (LPM) and the manufacturer's Intellectual Property (IP) functions within the source. Incorporating LPM modules increases the portability of the design between different EDA tool vendors. Using IP functions makes efficient use of each manufacturer's specific architecture and takes advantage of custom resources available within the targeted device such as embedded Random Access Memory (RAM) which is essential to this design.

A block diagram of the Segment Tracker firmware implementation is shown in Fig. 11.11. As described above, the segment finding algorithm processes the data in three distinct stages called the long doublet, the triplet, and the short doublet finding stages. As pixel data enters from the detector, it is stored in FIFO memory. FIFO memory is also used between each

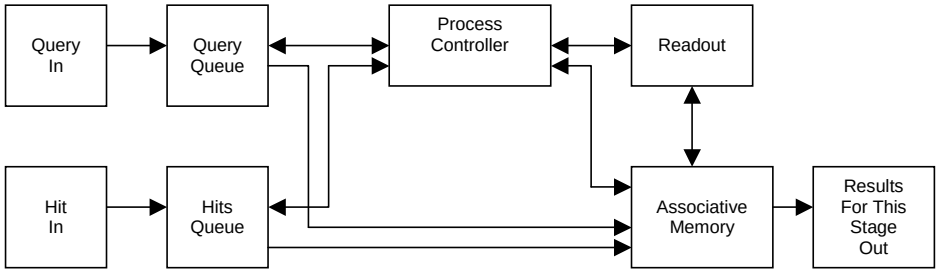

Figure 11.12: Block diagram of a single stage.

stage of the process to store results from the previous stage. Queuing both the input data as well as the intermediate results from each stage allows us to pipeline this process. When the pipeline contains sufficient data, each stage in this process will run independently of the others. A block diagram of a single stage is shown in Fig. 11.12. A general description of the process inside the long doublet, triplet, and short doublet finding stages follows.

The process starts when pixel data that meet certain constraints (e.g. hits within the inner region of a plane) enters the Query Queue from one station. We will refer to each of these data points as a "query". At the same time, pixel data from an adjacent station is entering the Hit Queue. We will refer to these data points as "hits". Thus, each query will be used to investigate whether there are hits from an adjacent station that can be assembled into a segment. The values from both the Query Queue and Hit Queue are then loaded into the Associative Memory according to the processing rate of each event. The Process Controller identifies the query and matches it to the corresponding hits using the Bunch Crossing Number (BCO). Events that don't match are discarded. In the Associative Memory, the query is stored in a single register and hits are stored in a register array that holds up to 16 values. This next step takes advantage of using an FPGA. A projection calculation is performed using the single query and all hits simultaneously in one clock cycle. Further, on the next clock cycle, all results from this calculation are simultaneously scanned and each candidate that meets the criterion is flagged for Readout. Readout then assembles the results and sends them on to the next stage of the process. This same sequence is repeated in each stage of the algorithm until the complete segment is finally passed on to the switch.

The FPGA Segment Tracker code is currently being implemented using Quartus II design software from Altera [5]. A design engineer has spent 0.3 FTE over the last 3 years writing, testing, and refining this VHDL code. Throughout the design process, the segment finding code has been repeatedly run through both synthesis and place-and-route tools. The tools are used in an iterative manner to provide positive feedback on how to further optimize the code to get the best results. The results from the place-and-route tool are shown in Table 11.2. The current design uses a maximum of 80% of the available resources in an

Family	APEX20KE		
Device	EPC20K1000EBC652-2X		
Resources	Used	Total	Percentage
Logic Elements	30,885	38,400	80%
Device Pins	127	488	26%
Memory Bits	204,800	327,680	62%
Phase Lock Loops	1	4	25%

Table 11.2: Segment finder hardware resource usage.

Altera EPC20K1000 part. The EPC20K1000 FPGA contains 38K total logic elements (LE's) and belongs to a family of high-density FPGAs that range up to 114K total LE's. This shows that the current design, together with future updates, can comfortably fit in various devices offered by Altera. Although Altera FPGA's have been used as the target device throughout the entire design cycle, similar devices in this density range are also offered by other manufacturers such as Xilinx. With some minor changes to the code, this design could be recompiled to fit in the Virtex II and Virtex II Pro family of devices from Xilinx. Our final objective is to target the device with the best cost versus performance characteristics regardless of the manufacturer.

Both the functional and timing simulations run on this code within the Quartus II design software use data that is extracted from Geant simulations of three adjacent pixel stations. The timing simulations have been successfully completed with the design running at 50 MHz. The results from these simulations match closely with those obtained from data flow analysis of the same data with Matlab [6].

In order to perform actual verification of the segment finding code, the design is currently being implemented in hardware. We will use a module called the Pixel Test Adapter (PTA) designed here at Fermilab as a firmware test and development platform. The PTA is a PCI based card that has an Altera EPC20K1000 FPGA onboard that is meant to be installed in the free slot of a personal computer (PC). Currently, we are conducting tests with this PTA using the same pixel data files from previous software simulations. Data is written to and results are read from the PTA card using simple C++ routines. The preliminary results from these tests show a great deal of consistency with those results obtained from both Matlab and Quartus II software simulations.

Initially, we plan to use the PTA card to process pixel data files one station at a time. The intermediate results of the track segments found for each station can be stored on the PC. When all the simulation data has been processed and collected, the resulting track segments can be sent directly to the L1 track and vertexing pre-prototype hardware to perform the second part of the L1 vertex trigger algorithm. In the future, additional PTA's identical to the first one can be built. Multiple PTA's can then be installed in a single PC representing a larger fraction of the total stations in the pixel detector. This would speed up the segment-

finding process. The resulting track segments can then be sent to the L1 track and vertex pre-prototype hardware for track and vertex reconstruction.

11.5.2 L1 Pixel Trigger Algorithm Timing Studies

11.5.2.1 DSP Timing Studies

This section will focus on timing studies done on the segment matching, tracking, and vertex finding portion of the L1 vertex trigger. Since this is the portion of the L1 vertex trigger algorithm that will run on the DSPs, it will be referred to as the DSP algorithm. We stress the importance of timing results since execution speed directly determines the total number of DSPs required in the L1 farm and therefore its cost and complexity. We will also describe work done to address two concerns raised by the PAC's Technical Review Committee in 2000, specifically (1) the fact that the custom assembly language code described in the proposal had not been run on a DSP or DSP simulator to demonstrate that it works, and (2) the potential problems involved in maintaining custom assembly code throughout the lifetime of the experiment.

The starting point of our studies was a variant of original C language version of the code on which the optimized assembly language version used in the proposal was based. (The code whose performance we describe here was optimized for triggering efficiency. The code used in the proposal made small sacrifices in triggering efficiency to achieve significant enhancements in computing speed. This will be discussed more below.) All DSP timing studies were done on a Texas Instruments (TI) C6711 DSP Starter Kit (DSK) board with a 150MHz TI TMS320C6711 DSP.

The initial step taken was to get the original, unoptimized C code running on the DSK board to address issue (1) above. Once the code was running on the DSK, optimizations were introduced in two major phases to reduce the execution times. In the first phase, costly calls to external library functions were replaced with C intrinsics that map directly to DSP instructions, and all double-precision floating point operations were replaced with single-precision operations. In the second phase, the segment matching portion of the algorithm which accounted for over half of the total execution time was completely rewritten in order to avoid the need to try all possible combinations of inner and outer segments in searching for complete tracks. Data structures of the code were also reduced in size to allow assignment of data memory sections in the DSP's internal Level 2 Cache/SRAM. This reduces CPU pipeline stalls due to cache misses requiring fetches from external SDRAM. In order to address issue (2) above, we also made it a point to do all optimizations in C, resorting to assembly language programming only when necessary.

Execution times after each phase of optimizations were measured using the built-in profiler of TI's Code Composer Studio to count CPU cycles accumulated in the DSP's on-chip performance monitors. Due to the reasons cited above, execution times are improved by hav-

Section of DSP algorithm	Proposal estimates in CPU cycles	DSP timing in CPU cycles per bunch crossing			
		Before opt. 10 BCO's	After Phase 1 opt. 10 BCO's	After Phase 2 opt.	
				10 BCO's	100 BCO's
Segment matching	24,200	1,296,778	503,012	164,836	168,113
Track processing	14,400	397,518	38,632	38,632	34,528
Vertexing	14,673	264,429	34,720	34,720	32,938
Total	53,273	1,958,725	576,364	238,188	235,579

Table 11.3: L1 Vertex Trigger Timing Results.

ing code and data memory sections reside in internal RAM. Since the DSP code[1] exceeded the 64KByte size of the internal memory, groups of functions were profiled in separate sessions wherein only those functions being profiled were assigned in internal RAM. This was done with the view that upcoming processors in TI's roadmap will have enough internal memory to hold the complete DSP code.[2] The measured execution times for both optimization phases are presented in Table 11.3 where they are compared with estimates in the BTeV proposal. These results were obtained with 10 simulated (Pythia/Geant3) minimum-bias bunch crossings with an average 2 interactions per crossing. For verification, the timing results for the second phase of optimizations was repeated with 10 times more statistics (Table 11.3, column 2). As these results show, execution times were reduced by nearly an order of magnitude ($1,958,725\ cycles \rightarrow 235,579\ cycles$) without resorting to custom assembly programming.

11.5.2.2 Optimizations with Custom Assembly Code

To see what performance gains could be achieved with custom assembly programming, code coverage of the segment matching routine was tested to identify sections on which optimizations would have the greatest impact. Using the intermediate assembler source code (totalling 1,329 lines) produced by the compiler for this routine as a starting point, custom assembly programming was done on the identified sections to make more efficient use of the DSP's 8 parallel execution units. 222 new lines of custom assembly code were written replacing roughly 20% of the compiler generated assembly code. Applying the procedure described above for measuring execution times on this optimized assembly version of the segment matching routine indicated a reduction of $\sim 2\times$ for this particular routine ($117,242\,cycles \rightarrow 63,618\,cycles$) and an overall reduction of over 20% for the complete DSP algorithm ($235,579\ cycles \rightarrow 181,955\ cycles$).

[1] roughly 2,500 lines of C source code representing on the order of $\sim$ 100KBytes in .data and .text memory sections.

[2] A new member of C671x family-TMS320C6713 is now available with 256KBytes of on-chip L2 SRAM (enough to hold the complete DSP algorithm) and higher core speeds of 225MHz (33% reduction in execution times).

Average number of interactions per bunch crossing	Without hash sorter		With hash sorter	
	Pentium III CPU cycles	TI C671x CPU cycles	Pentium III CPU cycles	TI C671x CPU cycles
$\langle 2 \rangle$	162,649	222,448	95,309	135,330
$\langle 4 \rangle$	412,702	563,164	214,539	305,247
$\langle 6 \rangle$	736,248	1,003,720	358,627	510,894

Table 11.4: L1 track and vertex algorithm CPU clock cycles for $\langle 2 \rangle$, $\langle 4 \rangle$, and $\langle 6 \rangle$interactions per bunch crossing (see text for more details).

11.5.2.3 Efficiency-Speed Tradeoffs

As mentioned above, the code used at the time of the proposal made some small sacrifices of ultimate triggering efficiency to achieve speed ups which explain in part the differences reported above. The version of the algorithm used here is aimed at maximizing the triggering efficiency. We are now going back and reviewing those tradeoffs. However, given the likely availability of faster DSPs, we do not see this as an urgent priority. Moreover, as will be seen below, alternatives to DSPs are now available which may already have provided us with a new approach which resolves nearly all remaining issues with respect to the L1 trigger speed.

11.5.2.4 Timing Studies on Other Processors

Investigations have also been done to measure execution times of the optimized C version of the DSP code on the following two general purpose processors: (a) 1.13 GHz Intel Pentium III-M, and (b) 500 MHz Motorola MPC7400 PowerPC G4. Additional optimizations done in C to utilize both processor's built-in vector execution units were introduced in the segment matching routine. Preliminary results are very encouraging indicating execution times roughly an order of magnitude better (Pentium III: $133,053\, cycles \times 0.88\, ns = 118\, \mu s$, PPC G4: $98,365\, cycles \times 2.0\, ns = 197\, \mu s$) than those for the optimized C code running on a 150 MHz TI TMS320C6711 DSP ($235,579\, cycles \times 6.67\, ns = 1,571\, \mu s$). Price/performance ratios of a PowerPC G4 based system seem quite promising compared to that based on a DSP. More work needs to be done, however, to investigate whether the I/O capabilities of these processors meet the requirements of the trigger.

11.5.2.5 Timing Studies for $\langle 4 \rangle$, and $\langle 6 \rangle$ Interactions per Bunch Crossing

The timing results presented above were all done using simulated data having an average of 2 interactions per bunch crossing. In this section, we present estimates for the DSP for averages of 4 and 6 interactions per bunch crossing.

To do these studies, 11 simulated triplet data files were created having a fixed number of interactions per bunch crossing ranging from 1 to 11. The number of CPU clock cycles to execute the L1 track and vertex algorithm was then measured and averaged over the $\sim 2,500$

Average number of interactions per bunch crossing	TI TMS320C6713 execution times in μs	
	with C version of segment matching routine	with hand-optimized assembly-lang. version of segment matching routine
$\langle 2 \rangle$	600.9	539.7
$\langle 4 \rangle$	1355	1205
$\langle 6 \rangle$	2268	2007

Table 11.5: L1 track and vertex algorithm execution times on a 225MHz C6713 DSP with the hash-sorter (see text for more details).

crossings in each of these data files on a 1.13 GHz Pentium III-M processor using the PAPI (Performance Application Programming Interface) libraries under Linux[3]. These libraries allow processor performance metrics to be measured by providing access to the processor's on-chip performance monitoring counters.

The number of Pentium III CPU clock cycles for $\langle 2 \rangle$, $\langle 4 \rangle$, and $\langle 6 \rangle$ interactions/bunch crossing were then determined from a Poisson weighted sum of the results with a fixed number of interactions per bunch crossing. These results are shown in column 2 of Table 11.4. In order to estimate the number of TI C671x CPU clock cycles for $\langle 4 \rangle$, and $\langle 6 \rangle$ interactions/bunch crossing, we normalized the PIII results for $\langle 2 \rangle$ interactions/bunch crossing to the measured TI C671x results in Table 11.3 totalling $222,448$ cycles[4]. This normalization procedure was done separately on the three sections of the algorithm shown in Table 11.3. It is assumed that the C671x and PIII times for each section differ only by a scale factor. The estimated C671x results are shown in column 3 of Table 11.4.

We have also conducted some studies to demonstrate how these timing results can be reduced considerably through the hash-sorter described in Section 11.5.3. The hash-sorter is custom hardware that can easily be implemented in the available space of the buffer manager FPGA to be described in Section 11.5.5. Inner and outer triplets are pre-sorted into bins of slope in the non-bend view, significantly reducing the execution time of the segment-matching routine of the L1 track and vertex algorithm. A C-implementation of the hash-sorter indicates an improvement of $\sim 4\times$ for the segment-matching routine ($87,385 - 20,045$ PIII cycles for$\langle 2 \rangle$ interactions/bunch crossing). Total PIII CPU clock cycles for the entire algorithm with the hash-sorter for $\langle 2 \rangle$, $\langle 4 \rangle$, and $\langle 6 \rangle$interactions/bunch crossing are shown in column 4 of Table 11.4. These results were arrived at using the same procedure described above for the results in column 2. The corresponding numbers for the TI C671x with the hash-sorter are estimated in a similar fashion and the results shown in column 5 of Table 11.4.

[3]These studies were done on a PIII rather than a DSP simply because they were much easier to do in a Linux/IA-32 environment

[4]These results are lower than those shown in Table 11.3 because we do not include the 13,131 cycles for triplet sorting routines that can easily be implemented in hardware

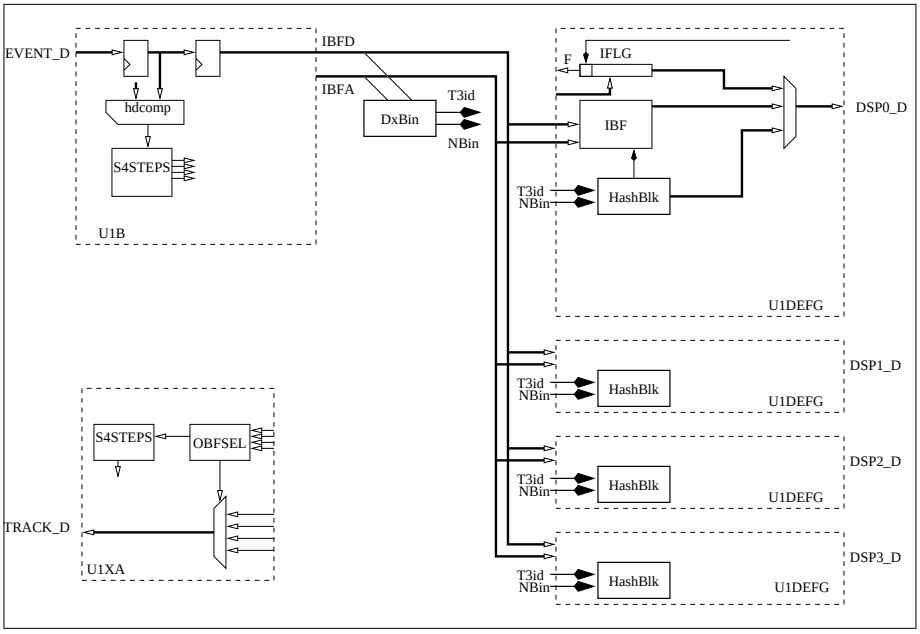

Figure 11.13: Hash sorter inserted into the Buffer Manager FPGA.

Using these results, we present estimates of the L1 track and vertex algorithm execution times on a currently available 225MHz TI C6713 DSP (4.44 ns/clock cycle) in column 2 of Table 11.5 for $\langle 2 \rangle$, $\langle 4 \rangle$, and $\langle 6 \rangle$interactions/bunch crossing. Assuming the execution time of the segment matching routine can be reduced by a factor of $\sim 2\times$ with hand-optimized assembly code (see Section 11.5.2.2), we present the corresponding estimates in column 3 of Table 11.5 if we use a hand-optimized assembly-language version of the segment matching routine together with the hash sorter.

11.5.3 L1 Hardware Hash Sorter

11.5.3.1 Hash Sorting for O(n^2) Algorithm Acceleration

In the BTeV L1 track and vertex algorithm, internal triplets are checked against an entire list of external triplets to find possible matches that form a complete track. This is an O(n^2) process that consumes a significant portion of CPU time. However, one could make use of the fact that external and internal triplets belonging to the same track have approximately equal slopes in the non-bend view. By sorting triplets into several bins based on their slopes, each internal triplet need only be checked against external triplets in one or two bins instead of an entire list, thereby reducing processing times significantly.

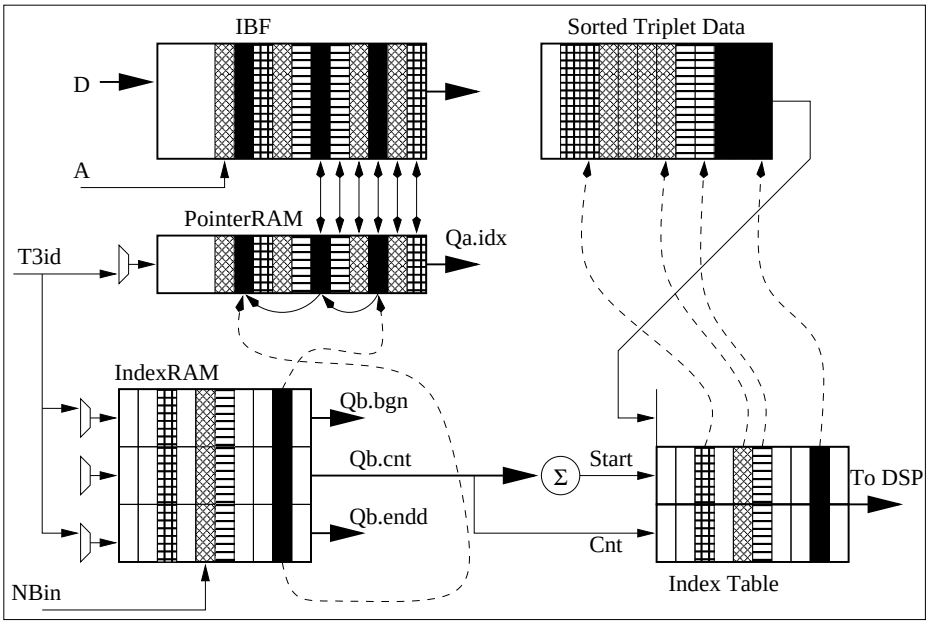

Figure 11.14: The hash sorting block (HashBlk).

11.5.3.2 Firmware Implementation

We have implemented a test design of the hash sorter parasitically on the Buffer Manager (BM) FPGA of the pre-prototype L1 track and vertex hardware described in Section 11.5.5. Two additional blocks, DxBin and HashBlk, are inserted into the original design of the BM as shown in Fig. 11.13.

As data fills the Buffer Manager's input buffer IBF, the DxBin block monitors the data (IBFD) and address (IBFA) lines, generating two numbers NBin and T3id. NBin identifies the hash bin where the triplet is entered representing its slope in the non-bend view. T3id is a unique identifier for the triplet composed of the higher bits of the IBF address where it is stored. These two numbers are used by the HashBlk block shown in Fig. 11.14 to perform the hash sorting function.

The three major components in this diagram are the IBF input buffer where the unsorted triplets are stored and the two main portions of the HashBlk consisting of the PointerRAM (Qa) and IndexRAM (Qb)[5]. Each unsorted triplet in the IBF has a corresponding location in the PointerRAM holding a pointer to the next triplet occupying the same hash bin. The IndexRAM, which stores information for each hash bin, contains the three fields Qb.bgn, Qb.endd, and Qb.cnt. The first two fields hold the ID's of the first and current triplet in the bin while the third field indicates the triplet count in that bin. These three fields are updated every time a triplet is entered into the IBF. If it is the first triplet in the bin, its ID

[5]It should be pointed out that the PointerRAM and the IndexRAM are implemented in different areas of the same dual-port RAM block making efficient use of the FPGA resources.

Resource	DxBin	HashBlk
Slices	56/5,120 (1%)	72/5,120 (1%)
Slice Flip Flops	73/10,240 (1%)	51/10,240 (1%)
4-input LUT's	35/10,240 (1%)	75/10,240 (1%)
Block RAM's	0	1/40 (2%)

Table 11.6: Silicon resource usage for the DxBin and HashBlk blocks in the hash sorter.

(T3id) is written into both Qb.bgn and Qb.endd. If it is not the first triplet, T3id is written into the PointerRAM location of the previous triplet which is pointed to by the current value of Qb.endd. Qb.endd is then updated with T3id of the current triplet while Qb.bgn is left unchanged and Qb.cnt is incremented by one.

The hash sorting action described above uses only 3 clock cycles given the available time of 4 clock cycles for filling the 4 32-bit data words for each triplet. The result is a single directional link list of hash bins in the PointerRAM which is available for later stages to use.

Once the hash sorting action is completed, the data is downloaded into the DSP's memory through a DMA process in two steps. In the first step, an index table is dumped into the DSP's memory. This index table allows the L1 track and vertex finding software running on the DSP to efficiently access the hash sorted triplet data block in the DSP's memory. Each entry in the table provides a pointer to the first triplet in a bin and the number of triplets in that bin. This table can easily be generated on the fly during the DMA process.

In the next step, the triplet data is transferred to the DSP's memory via DMA. The order in which the triplet data is presented at the BM FPGA's output port is controlled by the link list stored in the PointerRAM. This results in physically hash sorted data written into the DSP's memory. Downloading of both the index table and the hash sorted data block can be completed in a single DMA transaction. Once the DMA process begins, no further software intervention is needed.

11.5.3.3 Silicon Resource Usage

We have compiled the hash sorter function for the FPGA device (Xilinx Virtex II XC2v1000) on which the Buffer Manager is implemented. FPGA resource usage in shown in Table 11.6 indicating a total logic cell usage of about 7% (one DxBin block plus four HashBlk blocks) and memory block usage of about 10%. These numbers indicate that the hash sorter function can easily be accomodated in the available space of the Buffer Manager FPGA.

11.5.3.4 Timing Results

To test the hash sorter, we applied it to the track and vertex portion of the L1 trigger algorithm. Our timing studies were done on the 11 simulated triplet data files described in Section 11.5.2.5 with a fixed number of interactions per crossing ranging from 1-11. Execution

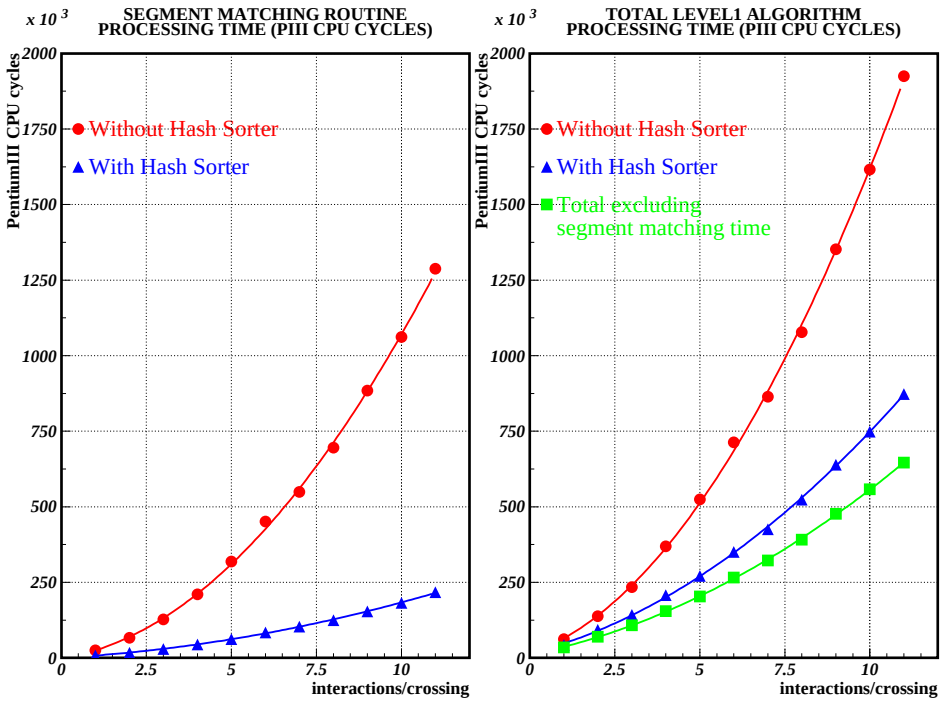

Figure 11.15: Execution times of the track and vertex portion of the L1 trigger algorithm with and without the hash sorter.

time of the trigger algorithm in CPU cycles on a 1.13 GHz Pentium III-M was measured and averaged over the $\sim 2,500$ crossings in each data file.

These measurements were done with and without the hash sorter and the results are shown in Fig. 11.15. Since the hash sorter affects only the performance of the segment matching routine of the algorithm, we show the results for this portion alone on the left-hand plot. Results for the entire algorithm are shown in the right-hand plot which also includes times for the entire algorithm excluding the segment matching routine. These results indicate an improvement of $\sim 5\times$ for the segment matching routine and an overall improvement of $\sim 2\times$ for the entire algorithm. They also show that, while total times are dominated by the segment matching routine when the hash sorter is not used, the time consumed by the other portions of the algorithm become more dominant when the hash sorter is used. Although these studies were done on a Pentium III, performance on other platforms should exhibit a similar trend of acceleration with hash sorting.

11.5.4 FPGA Segment Matcher

The L1 pixel trigger farmlets receive inner and outer triplets of pixel hits. These triplets are generated by the segment tracker, as explained in Section 11.5.1, and routed to the

farmlets by the L1 switch. The purpose of the tracking phase of the L1 algorithm is to reconstruct tracks using the inner and outer segments. The segment-matcher portion of the algorithm can be broken into a small number of sequential instructions that compute many independent pairs of data. Like the segment tracker, the segment matcher can be parallelized and considered for an FPGA implementation. FPGAs are more efficient than processors for performing multiple operations with fixed-point numbers, such as comparing a single query value to a list; or adding, subtracting, or multiplying one or more values to or from a list. If the entire list fits in the FPGA memory, these operations can be done in a single clock cycle.

The L1 pixel trigger timing analysis shows that about 50% of the farmlet processor execution time (not including the hash sorter described in Section 11.5.3) is spent in the segment-matching function, and the remaining 50% is spent doing the track and vertex reconstruction. If the segment matching is moved to an FPGA, then the average processing time is reduced by a factor of 2.

The segment matching has been simulated using a behavioral model in Matlab [6]. The design follows a timing model that is as close as possible to an FPGA implementation that will be done in VHDL. The current model suggests that a single FPGA segment matcher can be used for each farmlet. This FPGA would handle all of the data that is sent to the farmlet.

The behavioral simulation was performed for the statistical properties of the queuing model, the queue size, and service time distributions. In the simulated design, the FPGA is located before the Buffer Manager, which is the hardware that assigns bunch crossings to individual DSPs. Placing the segment matcher in front of the Buffer Manager reduces the number of segment matchers, since only one segment matcher is needed per farmlet. However, the segment matcher FPGA must be able to handle all of the data (inner and outer segments) for the farmlet. If necessary, one could also consider doubling the number of segment matcher FPGAs, but this doesn't appear to be necessary in our current design.

The segment matcher simulation used a conservative "worst case" scenario. The farmlet model included four DSPs. The service time distribution per DSP was modeled with an exponential distribution with $\mu = 46.8ns$ and $\sigma = 42.2ns$. These numbers were obtained by running the L1 track and vertex reconstruction algorithm on a 1.13GHz Pentium processor. The segment matcher was simulated using a conservative 50MHz clock for the FPGA. The average segment matcher service time obtained from the simulation was 7.1ns per bunch crossing, which represents 60% of the allowed time. The average size is $\mu = 1.7$ bunch crossings, and the standard deviation is $\sigma = 1.17$.

The option to implement the segment matcher in an FPGA gives us yet another method for reducing the number of DSPs needed for the L1 pixel trigger. The simulations have shown that the FPGA is able to handle all of the data sent to one farmlet. This implementation of the segment matcher is also compatible with the hash sorter described in Section 11.5.3. A VHDL implementation of the segment matcher is under development.

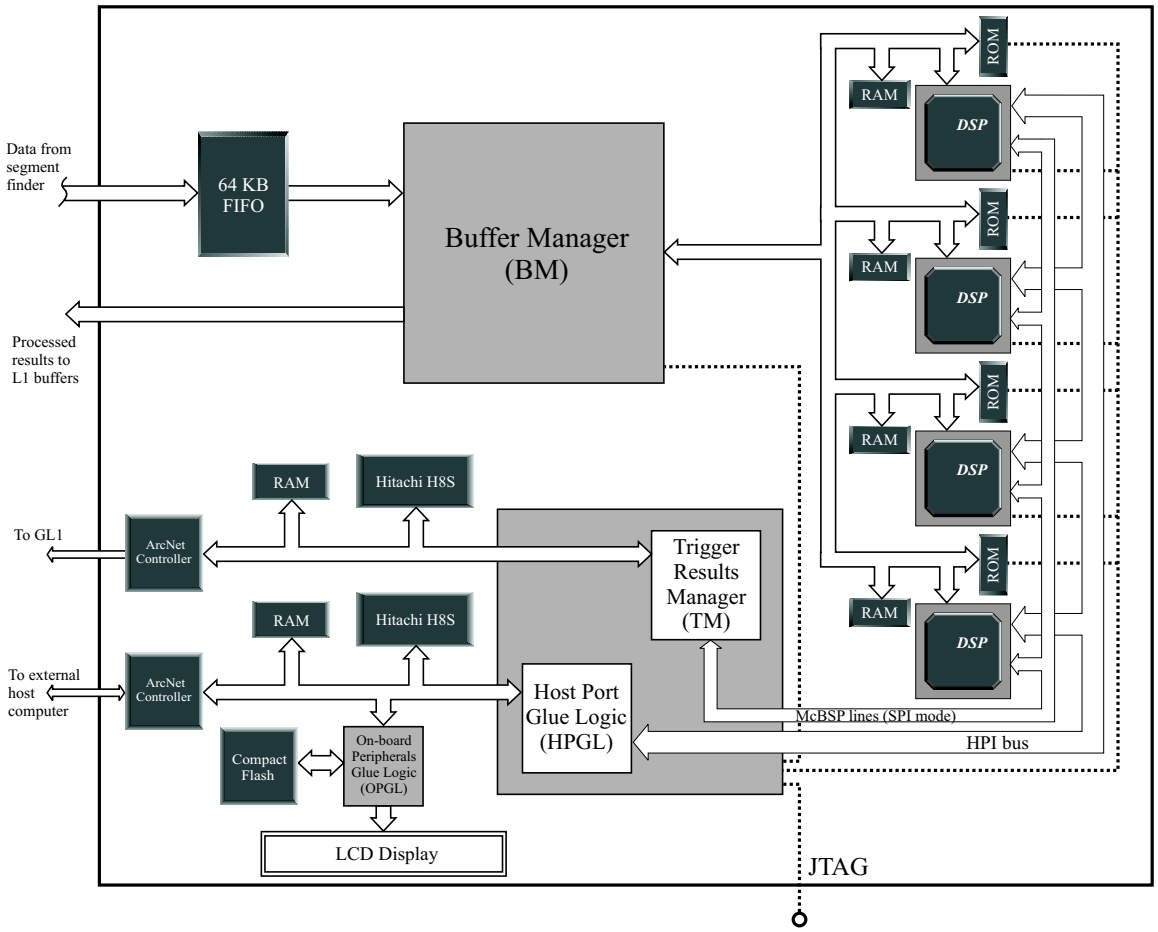

Figure 11.16: Block diagram of pre-prototype L1 track & vertex hardware

11.5.5 Pre-prototype L1 Track and Vertex Hardware

11.5.5.1 Overview

In this section, we describe the custom hardware we have developed that will serve as a reference design for a basic element or node in the track and vertex farm of the L1 trigger. It will also serve as a pre-prototype platform on which hardware and software investigations can be conducted to determine technical choices that will serve as input to the design of the prototype version. The prototype will form the basis for the pilot version of the L1 track and vertex hardware, which will serve as the basis for the production version of the hardware.

This discussion will begin with a brief overview of the pre-prototype hardware in terms of its major functional groups. It will be followed by a more detailed technical description of the main components in each of these groups. Finally, the discussion will conclude with a description of the performance of the pre-prototype hardware.

As mentioned in Section 11.4.2, the track and vertex farm of the L1 trigger performs the

second phase of the L1 trigger algorithm. It matches the inner and outer triplets found by the hardware segment finder to reconstruct complete tracks which are used to locate primary interaction vertices. A trigger accept is generated if tracks detached from these vertices meet certain criteria. Processed results are sent to the Level-1 buffers while summarized trigger results are sent to Global Level 1 (GL1) for the ultimate trigger decision.

We have designed the pre-prototype hardware to perform these basic functions. A description of the hardware can be broken down into three functional groups for 1) processing the data, 2) performing data input/ouput functions, and 3) providing control/monitoring functions. Physically, the hardware consists of an ATX-style motherboard and four daughtercards. A simplified block diagram of the pre-prototype hardware is shown in Fig. 11.16 The data processing elements consist of four digital signal processsors (DSPs) on the daughtercards that mate with the motherboard. Each DSP is independent of the others, performing the track and vertex portion of the L1 algorithm on all of the segment data for one bunch crossing.

Data I/O functions are handled by the buffer manager (BM) and the GL1 infrastructure. The two main functions of the buffer manager are to distribute data from the hardware segment finder to one of the four DSP's and to send processed results to the Level-1 buffers. The GL1 infrastructure consists of the trigger results manager (TRM), a 16-bit Hitachi microcontroller (GL1 Hitachi), and an ArcNet network controller. The trigger results manager receives trigger summaries from one of the DSP's multi-channel buffered serial port (McBSP) making them available to the Hitachi microcontroller. The Hitachi then sends these results to the GL1 processor through the ArcNet controller.

Control/monitoring functions are handled by Pixel Trigger Supervisor Monitor (PTSM) infrastructure. This consists of a second Hitachi microcontroller (PTSM Hitachi), host port glue logic (HPGL), on-board peripherals glue logic (OPGL), and a second ArcNet controller. The HPGL makes each DSP's registers and memory locations visible to the Hitachi through the DSP's Host Port Interface (HPI). This also allows the Hitachi to boot and reset any DSP. The OPGL provides the Hitachi with compact-flash memory and an alphanumeric LCD display. This allows it to access code and configuration files stored on the compact-flash memory and to display status and error messages on the LCD screen. The second ArcNet controller allows the Hitachi to communicate with a host computer on an external network which can be used to control and monitor the status of the preprototype hardware.

We describe each of the three functional groups of the pre-prototype hardware in greater detail below.

11.5.5.2 Technical Description of Pre-prototype Board

DATA PROCESSING

The pre-prototype hardware uses four 150 MHz Texas Instruments (TI) TMS320C6711 floating-point DSP's for executing the L1 track and vertexing algorithm. Each DSP sits on a daughtercard that conforms to the IEEE 1386 Common Mezzanine Card (CMC) standard. The daughtercard also supports TI TMS320C6211 fixed-point DSP's. Newer and

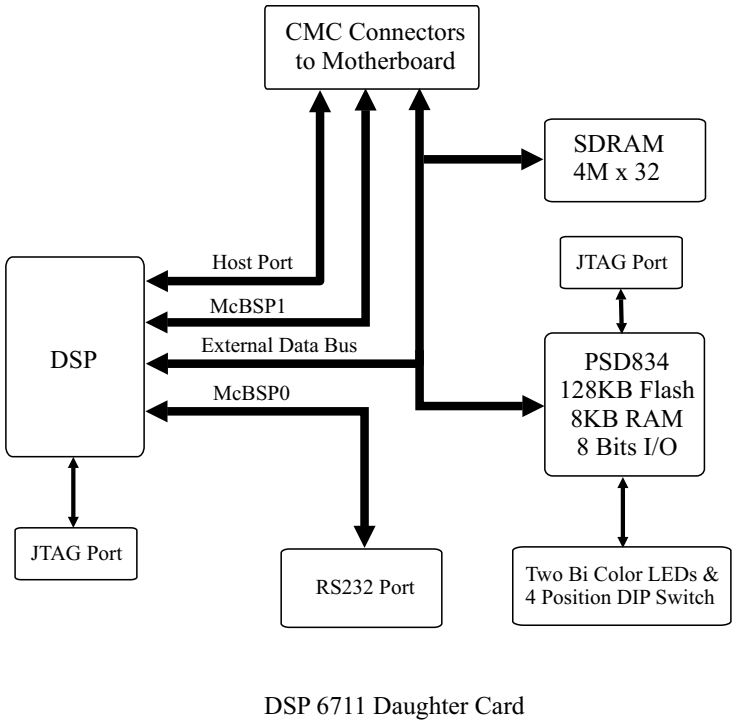

Figure 11.17: Block diagram of the DSP daughtercard

higher performance members of the C671x family such as as the C6713 can be supported with minor modifications to the board layout. A block diagram of the daughtercard is shown in Fig. 11.17. The daughtercard is very similar in functionality to a TI C6711 DSK (DSP Starter Kit) board and can be operated in standalone mode independently of the pre-prototype motherboard.

Hanging off the DSP's external memory interface (EMIF) are 16MB of 100 MHz SDRAM (2×4M×16 Micron MT48C4M16A2) and a Programmable System Device (PSD, STmicroelectronics PSD934F2) mapped, respectively, into its CE0 and CE1 address spaces. After powering up, the DSP boots from the main program flash of the PSD. The boot code, which performs a simple power-on self test (POST) and configures the SDRAM, can be directly programmed into the flash device through its JTAG port or through the DSP. User-defined LED's and switches are also available through the 8-bit ports of the PSD.

The first McBSP channel of the DSP interfaces with an RS-232 transceiver (Maxim MAX3221) which can be used to provide a serial console for the daughtercard. The CMC connectors route the EMIF bus, host port interface (HPI), and the lines for the second McBSP channel to the motherboard where they interface the DSP with the buffer manager, trigger results manager, and host port glue logic FPGA's to be described in the next section. A JTAG port connected to the DSP allows downloading of L1 code and real-time debugging of the DSP using TI's Code Composer Studio. L1 code can also be downloaded into the

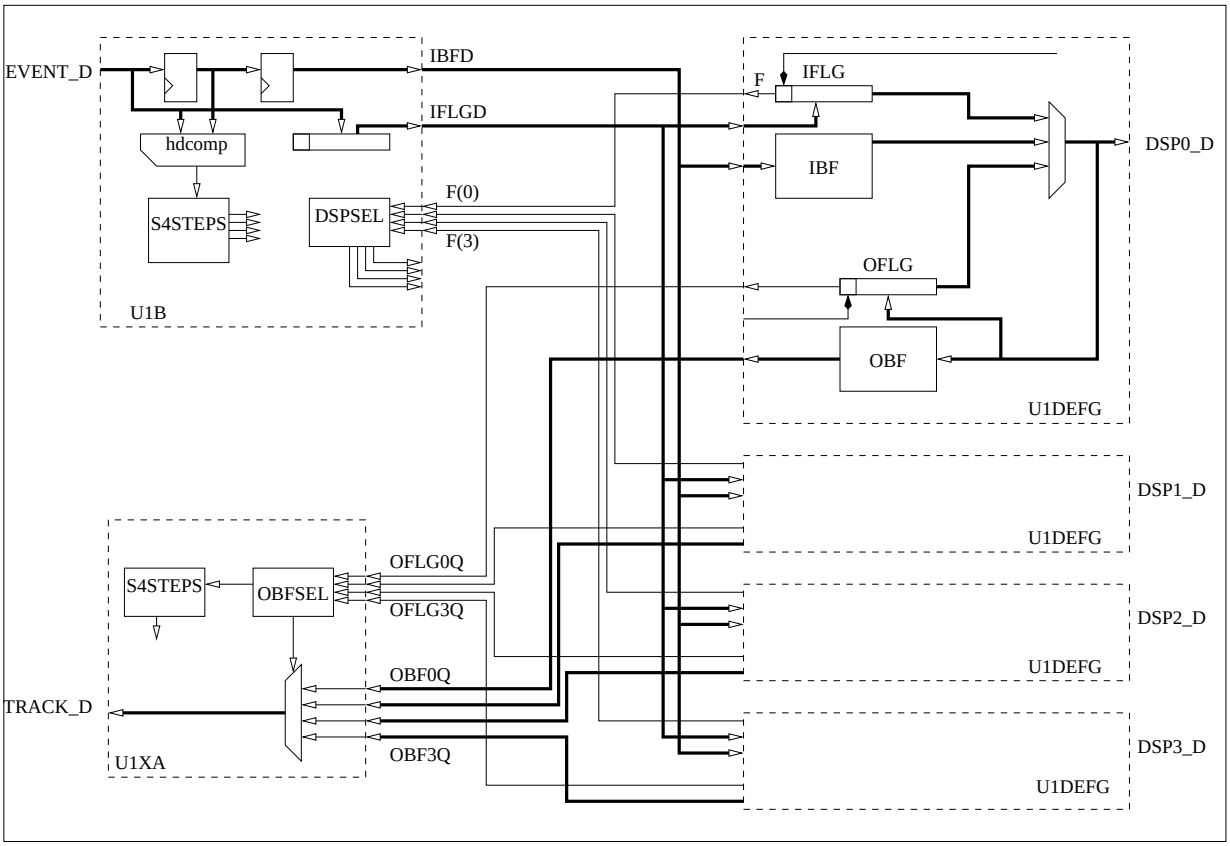

Figure 11.18: Block diagram of the buffer manager (BM)

DSP using its HPI interface which will be described below. Four logic analyzer headers allow hardware debugging of the various devices on the daughtercard. A unique serial number is provided for each daughtercard by a 1-wire EEPROM (Dallas Semiconductor DS2430AP) whose 1-wire bus is routed to the motherboard through the CMC connectors.

DATA INPUT/OUTPUT

Input Data from the Segment Trackers

Data received by the pre-prototype hardware through its 32-bit input ports is stored in a 64KB FIFO. The availability of data deasserts the empty flag of the FIFO which triggers the buffer manager to start transferring data to the DSP's.

The buffer manager (BM) is implemented in hardware on a Xilinx Virtex II FPGA. A block diagram of the buffer manager's architecture is shown in Fig. 11.18. Its task is to take all the triplet data stored in the FIFO for one bunch crossing and make it available to one of the four DSP's on the mezzanine cards. As soon as data is available, the first two words of the data block are read into the BM which checks to see if they are valid header words.

If the headers are valid, the sequencer block (S4STEPS), functioning as a state machine, initiates the data transfer to the DSP's.

The first step taken is to load the third word indicating the size of the data block (word count) into an internal register of the BM. Based on this word count, the entire block including the header words is transferred to the input buffer (IBF) of one of the four DSP's. DSP's are selected by checking their availability through a flag bit (MSB of the IFLG register) which can be reset by the DSP and by using a rotation priority scheme that ensures uniform loading of all four DSP's. The word count from the third header word is also loaded into the IFLG register and the flag bit set to indicate that the IBF is not empty.

The IFLG registers and IBF input buffers of the BM are accessed by the DSP's through their CE1 and CE2 address spaces. Each DSP constantly polls the flag bit of its associated input buffer to check the availability of data. If data is available, the DSP writes into the ESR register of its enhanced DMA engine (EDMA) to begin a CPU-initiated DMA transfer. DMA parameters loaded in the Parameter RAM (PaRAM) of the EDMA are used to transfer the data block into the DSP's internal L2 RAM through a 1-D source to 2-D destination, block-synchronized transfer. This allows contiguous blocks of data from the source to be loaded directly into selected members of C-structures that are non-contiguous in physical memory without CPU intervention. Each transfer actually consists of two separate transfers using two chained DMA channels for the interior triplets and the exterior triplets respectively. Exhaustion of DMA parameters on the first channel automatically triggers transfer on the second chained channel with its own set of parameters. As soon as the parameters are exhausted on either channel, they are automatically reloaded by the EDMA setting them up for the next input data block.

Processed data for Level-1 Buffers

After executing the L1 trigger algorithm on the input data, the DSP sends the processed data destined for the Level-1 buffers back out a 32-bit output port on the motherboard. To do this, the DSP executes a DMA transfer identical to that for the input data except that data flow is now from the internal L2 RAM to the OBF output buffers in the BM. Upon completion of the DMA transfer, the DSP loads the word count into the OFLG register and sets the flag bit (MSB of the OFLG) to indicate the availability of data in the output buffer. Once again the BM services the output buffers associated with each DSP by monitoring the flag bits and by using the same rotation priority scheme as in the case of input data. Based on the word count stored in the OFLG register, a similar sequencer block (S4STEPS) initiates the transfer of the processed data block from the OBF directly to the on-board 32-bit output ports.

Summarized trigger results for Global Level 1

The DSP also sends a summary of its trigger results to GL1. To accomplish this, the DSP utilizes its multi-channel buffered serial port (McBSP) to send these results to the trigger results manager (TRM) which interfaces the McBSP to the GL1 Hitachi microcontroller.

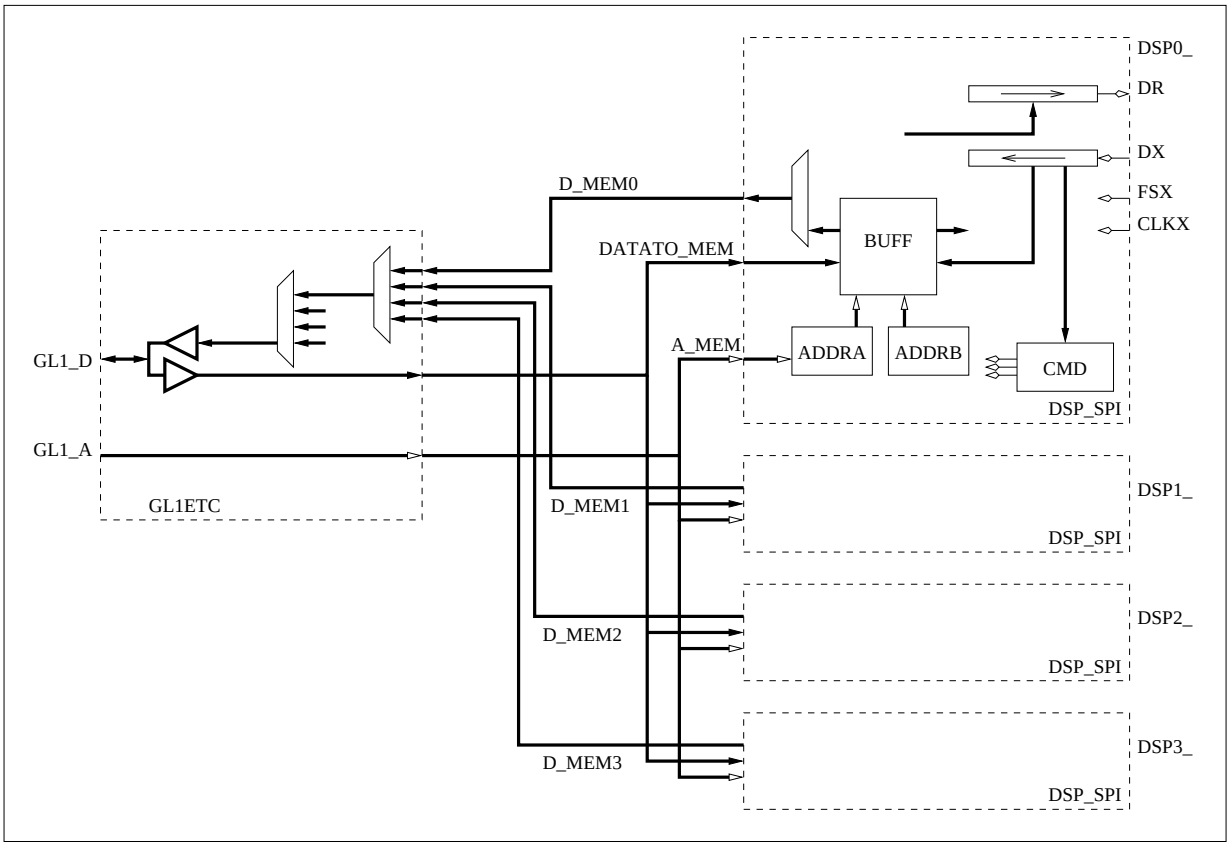

Figure 11.19: Block diagram of the trigger results manager (TRM)

The TRM is implemented in firmware on a Xilinx Virtex FPGA which is shared with the host port glue logic (HPGL) to be described below. A block diagram of the TRM is shown in Fig. 11.19. The H8S forwards this data to the on-board ArcNet interface which links the L1 preprototype hardware with the GL1 processor on an external network. In our implementation, the McBSP is configured as an SPI (Serial Peripheral Interface) master and the TRM as an SPI slave. The data transfer from the DSP's internal L2 RAM to the McBSP's data transmit register (DXR) is initiated by the CPU when it pulls the McBSP out of reset. This causes a 0 to 1 transition on the XRDY bit of the McBSP's serial port control register (SPCR) indicating the DXR's readiness to be loaded with new data. This transition provides the synchronization event (XEVT) to the EDMA engine which carries out the transfer.

Data from the McBSP is received at the TRM by a shift register in one of the four SPI blocks associated with a particular DSP. Here, the serial data is converted into parallel form before being stored in a dual-port memory buffer. Two address counters, ADDRB and ADDRA, are incremented whenever data is written into and read out of the buffer, respectively. A register on each SPI block keeps track of the difference between these counters

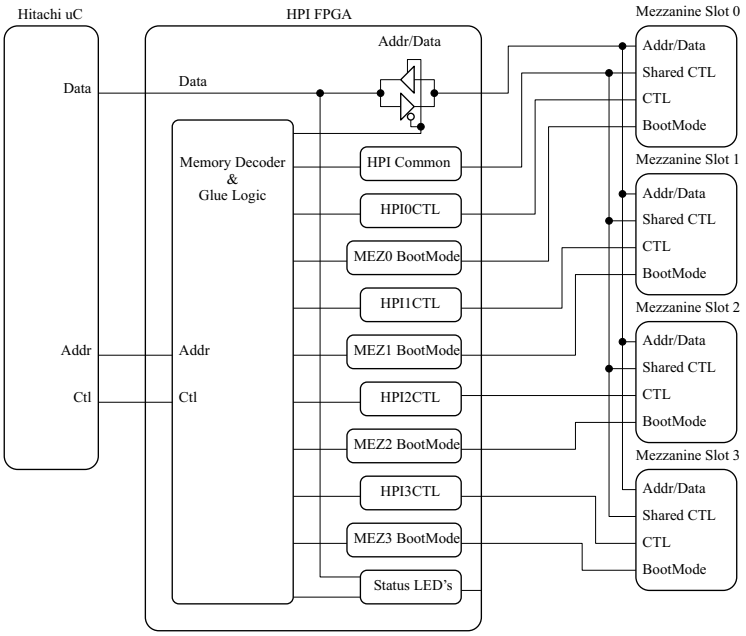

Figure 11.20: Block diagram of the Host Port Glue Logic (HPGL)

to indicate the availability of new data. A separate register keeps a summary of all four differences including a service suggestion (SS) flag pointing to the SPI block with the largest difference and hence requiring the most urgent attention. These registers and the data buffer are all mapped into the address space of the GL1 Hitachi which constantly monitors the registers for the availability of data. Once available, the Hitachi reads this data from the TRM's buffer and repackages it before sending it to the Global Level 1 host through an on-board ArcNet controller. The ArcNet controller used by the GL1 Hitachi is identical to that used by the PTSM Hitachi and is described in more detail in the next section.

Supervision and Monitoring (PTSM Infrastructure)

In addition to its main tasks of processing and handling data, the pre-prototype hardware has provisions that allow an external host computer, serving as a Pixel Trigger Supervisor Monitor, to access and control the 4 DSP's and other on-board peripherals. The hardware and software components that provide this functionality will be referred to as the PTSM infrastructure. Our description of this infrastructure will center on the two major hardware components that comprise it.

Host Port Glue Logic

The first component is the Host Port Glue Logic (HPGL) which interfaces the PTSM Hitachi microcontroller to the HPI bus of each of the 4 DSP's allowing it to initialize, control and

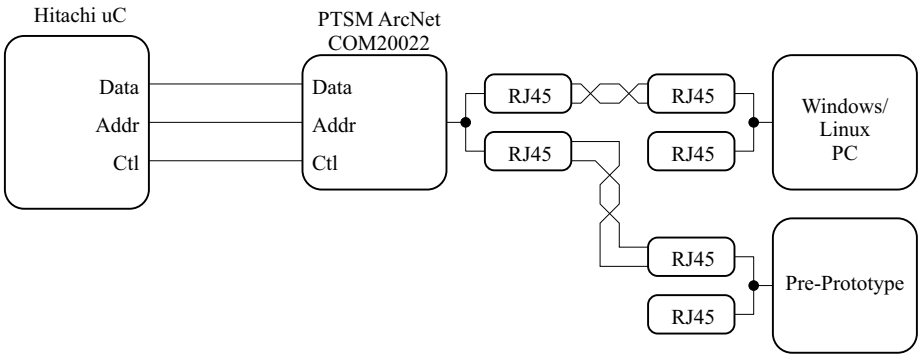

Figure 11.21: Hitachi to ArcNet interface

communicate with any DSP. The HPGL is implemented in hardware on the same Xilinx Virtex FPGA as the Trigger Manager described in the section above. The block diagram of the HPGL shown in Fig. 11.20 consists of the following four basic components: 1) a memory decoder and glue logic block, 2) an HPI common block, 3) an HPIxCTL block, 4) and a MEZxBOOTMODE block. The memory decoder and glue logic block decodes and maps the HPI bus of each DSP into a 64 Kbyte region of the Hitachi's address space allowing it to be accessed as memory. Control logic used by the Hitachi to access the 4 HPI buses is contained in the HPI common block and the 4 HPIxCTL blocks. Logic common to all 4 HPI buses is contained in the former while logic specific to a particular DSP's HPI bus in the latter. Finally, the 4 MEZxBOOTMODE blocks select the boot mode of each DSP to allow booting from the HPI bus, the DSP's on-board boot device (PSD934F2), or its JTAG port.

The 4 DSP's communicate with the Hitachi through a "mailbox" residing in the HPI memory map which can be used to indicate status or error conditions. A library of low-level C routines are available to allow access to the mailbox, select the DSP's boot mode, download DSP programs, and provide other useful functions.

An on-board ArcNet controller (SMSC COM20022) mapped into an 8 byte section of the Hitachi's memory space allows it to communicate with an external PTSM host computer on the network. ArcNet was chosen for the pre-prototype platform due to its low protocol overhead, adequate speed (10Mbps), low cost, and ease of implementation. A block diagram of the Hitachi to ArcNet interface is shown in Fig. 11.21. The Hitachi interfaces directly to the ArcNet cotroller without any glue logic, using only a chip select, three address lines, and a R/W line.

On-board Peripherals Glue Logic

The second component of the PTSM infrastructure is the On-board Peripherals Glue Logic (OPGL) which interfaces the Hitachi with an on-board CompactFlash (CF) memory card and a 4×20 character alphanumeric LCD display. The OPGL is implemented on a Xilinx

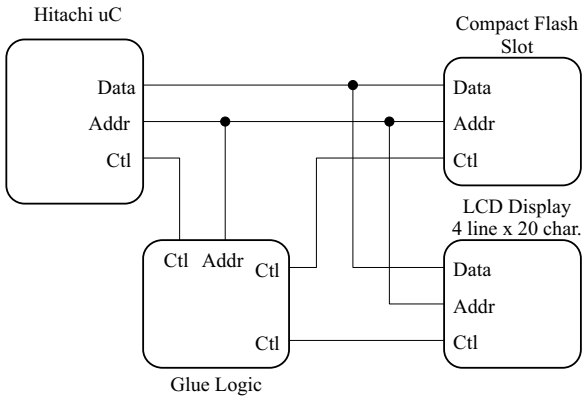

Figure 11.22: On-board Peripherals Glue Logic (OPGL)

FPGA and a block diagram is shown in Fig. 11.22. The OPGL maps the CF memory card and the LCD display into the Hitachi's memory space.

A library of low-level C routines has been implemented on the Hitachi to provide access to the CF memory card and the LCD display. Included in the library are functions for accessing and manipulating (read, write, delete etc.) the simple FAT32 file system implemented on the CF memory card. Also available are functions for controlling and displaying messages on the LCD display. With these routines, the CF memory card can be used to store FPGA boot code and DSP programs while the LCD display can be used to display status and debugging information.

11.5.5.3 Performance of Pre-prototype Board

This last section will discuss the performance of the pre-prototype hardware. It will focus on the ability of the hardware to handle the following three data I/O functions: getting raw segment data into the DSP's internal L2 RAM, sending the processed results out of the DSP, and sending the trigger summaries to GL1. The data processing ability will not be discussed here since they are addressed by the detailed timing measurements described in Section 11.5.2.1. These measurements were done on a TI DSK board with a TI C6711 DSP identical to the ones on the pre-prototype.

The Xilinx VirtexII used for the BM runs at a clock of 25MHz. Since the data path from the FIFO to the BM is 4 bytes wide, this means the raw segment data can be unloaded from the FIFO into the BM at 100 MB/sec. We have measured the time to transfer data via DMA from the BM's input buffers to the DSP's L2RAM by counting the number of clock ticks on the DSP's internal timer between the start and completion of the CPU-initiated DMA transfer. The start is defined by writing a "1" into the Event Set Register (ESR) and the completion by the Channel Interrupt Register (CIPR) getting set. It takes 377 ticks to transfer 128 32-bit wide words. Since the DSP's timer is incremented every 4 CPU clocks and the CPU runs at 150 MHz, this translates to a data rate of ∼51 MB/s for the input

path. We have also used this same procedure for the reverse path from the DSP's L2 RAM to the BM's output buffers. We measured 314 ticks for the same amount of data translating to ∼61 MB/s for the output path. These figures are more than sufficient to handle the estimated ∼13MB/s data rate into each DSP in the 2,500 node track/vertex farm of the baseline trigger design.

For the trigger results destined for GL1, we have successfully sent data out of the DSP with the McBSP running at ∼9 MHz ($(CPU/2) \times 1/(CLKGDV + 1)$, $CLKGDV = 7$). Since this is a serial stream, this translates to a data rate of ∼9Mb/s or ∼1MB/s. Assuming the trigger results sent to GL1 average at ∼20 bytes/crossing for each DSP, the required data bandwidth from each DSP to GL1, for a 2,500 node L1 track/vertex farm, is ∼60KB/s. This means that the actual bandwidth is more than an order of magnitude greater than the required bandwidth.

11.5.6 L1 Performance

11.5.6.1 Efficiency and Rejection Studies

All studies of the L1 trigger are performed using GEANT. We generate pixel clusters, and include hadronic reinteractions, photon conversions, decays in flight, and delta rays. All studies are performed with an average of two interactions per bunch crossing, except when we vary the number of interactions to study the trigger response for different running conditions.

We study the performance of trigger algorithms using minimum bias events and different types of B-events. We have studied a variety of cuts, implemented at various stages in the trigger, and have chosen to use a few cuts in addition to the final vertexing cuts to help reject minimum bias interactions. For example, the L1 pattern recognition eliminates low momentum tracks (tracks with $p < 3$ GeV/c) to avoid tracks that may suffer from excessive multiple scattering and could easily be reconstructed as having a large impact parameter with respect to a primary vertex. Moreover, all tracks are required to pass through at least four tracking stations to remove erroneous combinations of pixel clusters that can mimic what appear to be acceptable 3-station tracks (in these cases the interior and exterior triplets are usually constructed from identical pixel clusters). Lastly, a clean-up step removes all tracks that share pixel clusters with any other tracks. This method of removing fake tracks is simple, and perhaps overly severe, but it is effective in eliminating fake tracks at an early stage in L1. We have not performed an exhaustive study of possible trigger cuts, so we anticipate additional improvements resulting from future studies of the L1 trigger.

The L1 vertexing cuts are the final cuts that determine the L1 efficiency for B-events and the rejection of minimum bias events. These cuts are selected to provide 98% rejection for minimum bias crossings at a crossing time of 396 ns and an average of 6 interactions per bunch crossing. For a 132 ns crossing time and an average of 2 interactions per bunch crossing the cuts are selected to provide 99% rejection of minimum bias crossings. For historic reasons we describe in this section the cuts used for a crossing time of 132 ns, while the performance at 396 ns is described in a later section.

Table 11.7: L1 trigger efficiencies for minimum-bias events and various processes of interest that are required to pass off-line analysis cuts. The trigger efficiencies for all modes are determined for a bunch crossing time of 132 ns and with an average of two interactions per crossing. Results for some modes are also shown for a bunch crossing time of 396 ns and an average of 6 interactions per crossing.

Process	Efficiency	
	132 ns Crossing Time $\langle 2 \rangle$ Int/crossing	396 ns Crossing Time $\langle 6 \rangle$ Int/crossing
Minimum bias	1%	2%
$B_s \to D_s^+ K^-$	80%	66%
$B^0 \to J/\psi K_s$	65%	
$B^- \to K_s \pi^-$	45%	
$B^0 \to \phi K_s$	74%	
$B^0 \to$ 2-body modes $(\pi^+\pi^-, K^+\pi^-, K^+K^-)$	80%	

For a crossing time of 132 ns the vertexing cuts require that at least n tracks (all directed at one arm of the BTeV spectrometer) miss a primary vertex by at least $m\sigma$. The impact parameter is required to be less than 2 mm to exclude tracks that may be associated with other primary vertices in crossings with more than one interaction. The 2 mm cut also rejects daughter tracks from strange-particle decays. Fig. 11.23 shows the trigger response for minimum bias crossings for a range of vertexing cuts. There are four sets of points corresponding to the requirement of $n = 1$, 2, 3, or 4 detached tracks. The horizontal scale specifies the minimum impact parameter for detached tracks. For the results described in this section the requirement of 2 tracks at 2.8σ is used. This provides an L1 rejection for minimum bias events of a factor of 100. For the actual experiment we would likely run with a mix of prescaled triggers.

With the vertexing cuts set to achieve the desired minimum bias rejection, we can study the trigger efficiency for different types of B-events. For $B_s \to D_s^+ K^-$ we obtain the trigger efficiencies shown in Fig. 11.24. Our cut, requiring at least 2 tracks with a minimum impact parameter of 2.8σ, gives us a trigger efficiency of 80% for this decay mode. Trigger efficiencies for other B-decay modes are shown in Table 11.7 together with the efficiency when running at a crossing time of 396 ns and an average of 6 interactions per bunch crossing.

It is important to realize that the L1 trigger, which requires at least two detached tracks, is able to trigger on B events that involve decay modes with fewer than two charged tracks at the B-decay vertex. An example of this is the $B^- \to K_s\pi^-$ mode listed in Table 11.7. Since this mode has only one track associated with the B^- decay vertex, the majority of triggers come from detached tracks associated with the other B decay in the event.

Most of our studies of the L1 trigger efficiency are based on the decay mode $B_s \to D_s^+ K^-$. These include studies of pixel noise and inefficiencies, described in the next section. Although

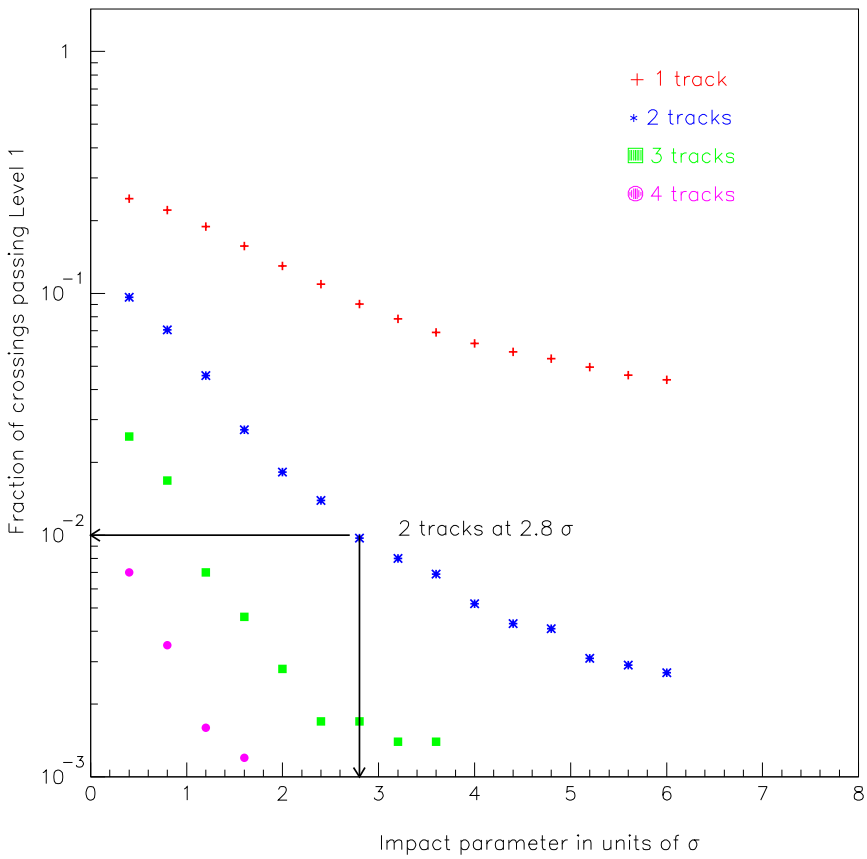

Figure 11.23: Trigger response for minimum bias events for a crossing time of 132 ns and with an average of two interactions per bunch crossing. The figure shows four sets of points requiring at least 1, 2, 3, or 4 detached tracks. The arrows show a cut that requires at least 2 detached tracks with an impact parameter that exceeds 2.8σ, and achieves 99% rejection.

we have not intentionally optimized cuts for this particular decay mode, it is conceivable that the current set of L1 cuts are more favorable for $B_s \rightarrow D_s^+ K^-$ than for other decay modes (such as the other modes listed in Table 11.7).

11.5.6.2 Pixel Noise and Inefficiency Studies

The L1 pattern recognition is exceptionally robust with respect to pixel inefficiencies and noise hits in the vertex detector. Fig. 11.25, which shows the trigger response for $B_s \rightarrow D_s^+ K^-$ and minimum bias crossings versus the number of noise hits in each pixel plane, summarizes the results from our noise and inefficiency studies. These studies were

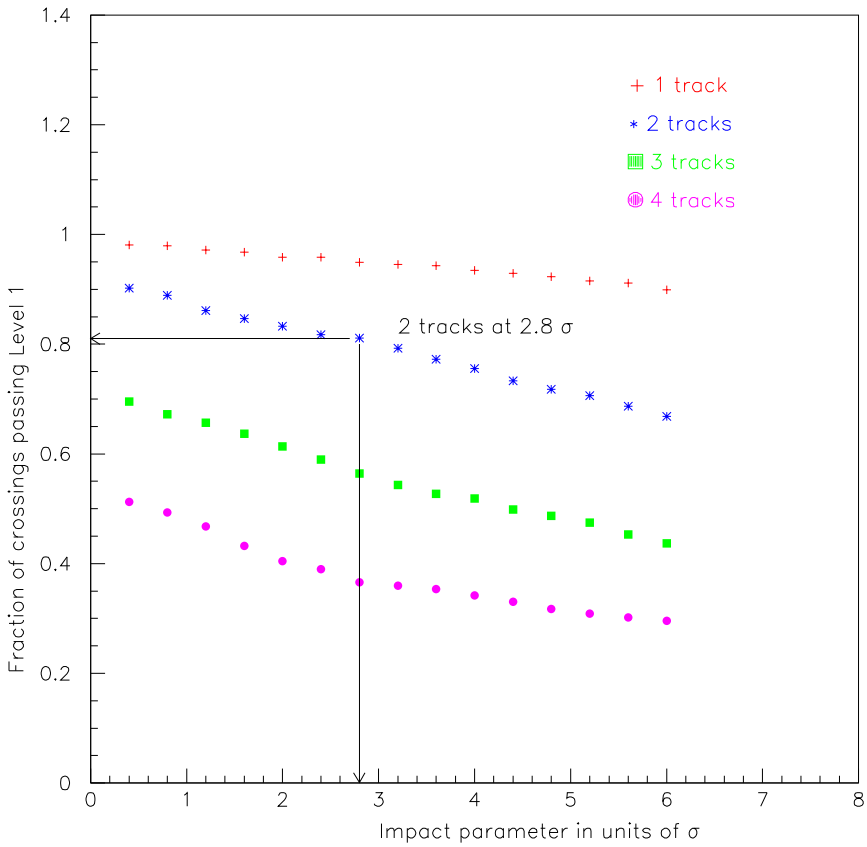

Figure 11.24: Trigger efficiency for $B_s \rightarrow D_s^+ K^-$ events with a crossing time of 132 ns and an average of two interactions per bunch crossing. The figure shows four sets of points requiring at least 1, 2, 3, or 4 detached tracks. The arrows show a cut that requires at least 2 detached tracks with an impact parameter that exceeds 2.8σ, and gives a trigger efficiency of 80%.

done for a crossing time of 132 ns and an average of two interactions per bunch crossing. There are three sets of points that correspond to three different pixel efficiencies. There is a noticeable decrease in the trigger efficiency for $B_s \rightarrow D_s^+ K^-$ with decreasing pixel efficiency. We expect to achieve a pixel efficiency that exceeds 99%, so the trigger efficiency for $B_s \rightarrow D_s^+ K^-$ at a crossing time of 132 ns and an average of two interactions per bunch crossing should exceed 75%—a trigger efficiency that is less than the first set of points (stars) and greater than the second set of points (triangles).

We add noise hits to the trigger simulation to study how sensitive the pattern recognition is to spurious pixel clusters. We have studied two types of noise distributions without

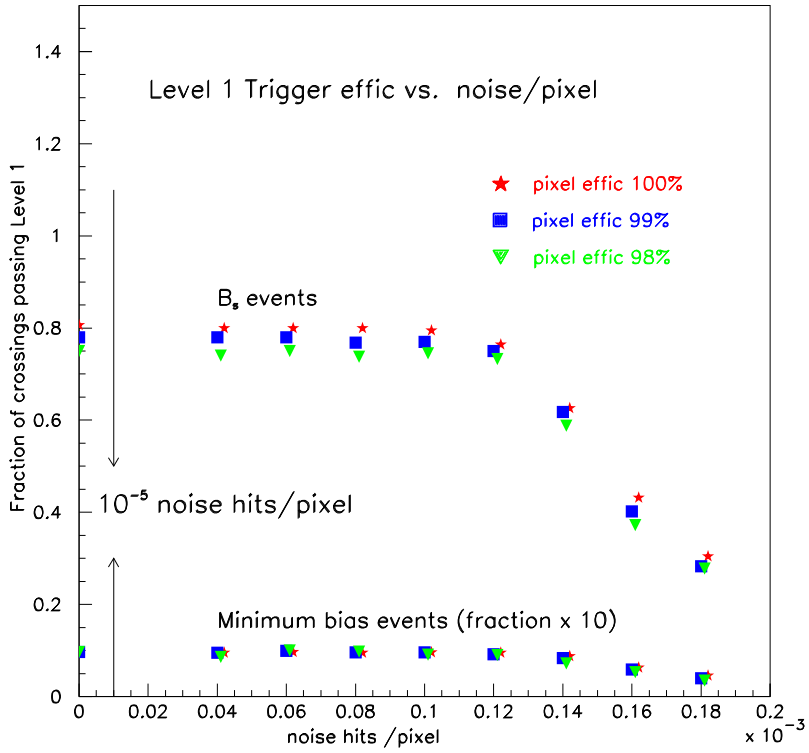

Figure 11.25: Fraction of $B_s \rightarrow D_s^+ K^-$ and minimum bias crossings, with a crossing time of 132 ns and an average of two interactions per crossing, that satisfy the L1 trigger for three different pixel efficiencies (three sets of points) vs. the number of noise hits/pixel that have been added in the vertex detector. Results for minimum bias crossings have been multiplied by 10.

observing any significant difference. We generate a uniform distribution of noise over an entire pixel plane (see Fig. 11.25). We expect the noise level in the detector to be less than 10^{-5} noise hits per pixel. In our studies we observe a decrease in the trigger efficiency for $B_s \rightarrow D_s^+ K^-$ when the noise level exceeds 10^{-4} hits per pixel. This decrease probably results from the clean-up step in the L1 trigger that removes tracks with shared hits.

For minimum bias crossings we get results that are similar to the results obtained for $B_s \rightarrow D_s^+ K^-$. Needless to say, L1 is exceptionally stable with respect to noise, and we do not expect any noticeable deterioration in the trigger performance for the amount of noise expected to occur in the pixel detector.

11.5.6.3 Performance at 396ns

This section describes studies for a range of BTeV running conditions with different numbers of interactions per bunch crossing, and the influence that these running conditions have on the BTeV Trigger System.

It is important to recognize that the performance of the trigger system depends on the quality of data from the detector, the performance of the Tevatron, and even some unverified physics assumptions about the largely unexplored forward rapidity region. The trigger must be able to cope with deviations from these assumptions. It is in this context that we have studied the performance of the trigger for a range of running conditions.

For the studies described in this document we have generated samples of minimum bias interactions, and interactions with the decay $B_s \rightarrow D_s K$.

BTeV was originally designed to operate with a 132 ns bunch-crossing interval at an initial luminosity (at the beginning of each store) of $2 \times 10^{32} cm^{-2} s^{-1}$. This corresponds to an average of 2 interactions per bunch crossing. The baseline running condition is now for a crossing time of 396 ns still at an initial luminosity of $2 \times 10^{32} cm^{-2} s^{-1}$, corresponding to an average of 6 interactions per bunch crossing. The BTeV Trigger system is designed to handle the range of crossing times from 132–396 ns and with 2–6 interactions per bunch crossing. We have investigated the behavior of the trigger system for different numbers of interactions per bunch crossing.

The results that we present for a range of 2-6 interactions per bunch crossing come from studies of the different hardware components used in the L1 pixel trigger, efficiency for minimum bias and B_s interactions, and bandwidth studies for data flowing into L1 and bandwidth into L2 for interactions that satisfy the L1 pixel trigger requirements. Results are summarized in a table at the end of this section.

Our studies of the FPGAs show that the time required to process pixel data increases almost linearly with the number of interactions per bunch crossing. Although the timing for this part of the L1 hardware is not an important factor (since processing times are significantly higher for the DSPs), our studies do confirm that the algorithm behaves in a robust manner as the number of interactions per bunch crossing increases. Our studies indicate that the requirements for memory resources for the FPGAs also increase linearly. This is not surprising, since each additional event in a bunch crossing adds an equal amount of data.

The results of timing studies for the DSPs are shown in the table below. These are results for a Texas Instruments C6713 DSP running at 225 MHz, which is available today.

Another study that we have performed for different numbers of interactions per bunch crossing considers the amount of data that is produced by the BTeV apparatus for each bunch crossing. This gives us an estimate of the bandwidth into L1. Furthermore, we have looked at the "size" of the interactions that are selected by the L1 trigger. There is a tendency for interactions with more data (more tracks per interaction) to be selected by L1. This determines the bandwidth into L2.

Table 11.8 shows trigger results for the L1 pixel trigger for an average of 2, 4, and 6 interactions per bunch crossing, with two different cut settings for 4 and 6 interactions per

Table 11.8: Results are shown for an average of 2, 4, and 6 interactions per bunch crossing (BCO). The results include trigger efficiencies, timing measurements, and bandwidth determinations.

Avg. ints/ BCO	L 1 cut(σ)	BCO (ns)	L1 eff B_s	L1 eff mbias	L1 time (ms)	# TIC6713 DSPs	Data into L1 kB/ BCO	Data into L1 GB/s	Data into L2 kB/ BCO	Data into L2 GB/s
$< 2 >$	1.9	132	0.79	0.020	0.540	4091	50	379	90	13.6
$< 4 >$	3.2	396	0.75	0.020	1.205	3043	100	253	150	7.6
$< 4 >$	2.3	396	0.78	0.035	1.205	3043	100	253	150	13.3
$< 6 >$	4.7	396	0.66	0.020	2.007	5068	150	379	180	9.1
$< 6 >$	3.6	396	0.71	0.030	2.007	5068	150	379	180	13.6

crossing. The second column shows the L1 detachment cut that we use to achieve the desired L1 efficiency for minimum-bias interactions (shown in column 5). The third column shows the bunch-crossing time, and the fourth column shows the trigger efficiency for $B_s \rightarrow D_s K$.

The sixth column shows our timing results for the Texas Instruments C6713 running at 225 MHz. The timing measurements determine the number of DSPs that are needed (column 7). The number that we achieve for 2 interactions per crossing shows that our latest results are close to the estimate (within a factor of 2) of 2500 DSPs that we presented in the BTeV Proposal. Other improvements (such as the hash sorter and segment matcher mentioned earlier) can reduce the number of processors even further.

Column 8 shows our estimates of the amount of data per bunch crossing coming from the detector, and is used to determine the bandwidth into L1 (column 9). We also determine the size of events for data selected by the L1 trigger (column 10), and this gives us an estimate of the bandwidth into L2 (column 11).

For 4 interactions per crossing we show the results that we get when we use a cut that gives us the same minimum bias efficiency (2%) that we have for 2 interactions. Since the bandwidth into L2 is lowered from 13.6 GB/s to 7.6 GB/s (due to the longer bunch crossing time of 396 ns), we can relax the L1 detachment cut to maintain the bandwidth that the system has been designed for. By relaxing the cut we increase the B_s efficiency to 78%. It is also noteworthy that for 4 interactions at 396 ns we reduce the number of DSPs from 4091 to 3043.

For 6 interactions per crossing we also show what happens for two different L1 detachment cuts. First we show results for a minimum bias efficiency of 2%. Then we show results for a minimum bias efficiency of 3%, which allows us to exploit the bandwidth in our design.

We have presented results for a range of interactions per bunch crossing for the most challenging aspects of the BTeV Trigger System, the L1 pixel trigger. The results show that we achieve acceptable performance for different running conditions for trigger timing and bandwidth. This shows that our baseline design for the L1 pixel trigger is able to handle these running conditions rather well, without requiring any significant changes in the design.

11.5.7 L1 Muon Trigger Hardware and Simulations

11.5.7.1 Simulation Overview

As should be clear from the discussion in Section 11.4.3, the efficiency and rejection for a given triggering scheme for the muon trigger will depend on a number of factors:

- Our choice of D-cut.
- How we choose to combine the information from the four views in each octant.
- The single-hit efficiency of the proportional tubes.
- The running conditions (i.e. non-muon occupancy).

In order to guide the design of the BTeV muon trigger and understand its performance as a function of the above factors (only some of which are under our control), we use a detailed GEANT based simulation of the BTeV detector.

In the baseline study, the minimum bias Monte Carlo events (used to study rejection) each contain N minimum bias interactions, where N is chosen from a Poisson distribution with an average $< N >= 2$. Each signal Monte Carlo event also contains N Poisson distributed minimum bias interactions, with the addition of one $B^0 \rightarrow J/\psi K_S$, where $J/\psi \rightarrow \mu^+\mu^-$ and $K_S \rightarrow \pi^+\pi^-$.

In addition to the overall fiducial acceptance defined by the BTeV muon detectors, the dimuon algorithm has two other small "geometrical" inefficiencies.

- Since we look for tracks in all octants independently, tracks that cross between octants are usually lost.
- Since we look for a single track in each octant, $J/\psi \rightarrow \mu^+\mu^-$ events where both muons enter the same octant are usually lost.

When combined, both of the above algorithmic effects reduce the trigger efficiency for $J/\psi \rightarrow \mu^+\mu^-$ events by about 7%. This is included in the muon trigger simulation, and reflected in all efficiency and rejection numbers presented below.

In the BTeV simulation used to generate the events that feed into our study of trigger performance, the proportional tube efficiency as a function of r/R (distance from the sense wire divided by the radius of the tube) is assumed to be 100% for $0 < (r/R) < 0.85$, and 0% for $0.85 < (r/R) < 1$. Since the tubes within a view are staggered, their coverage overlaps and the probability for a muon track to hit the efficient volume of at least one tube is 100% as long as the track enters the fiducial area of that view. To study trigger performance as a function of additional proportional tube inefficiency, we use a simple random number generator in our trigger simulation code to discard Monte Carlo generated hits and achieve whatever tube hit efficiency ϵ we desire.

11.5.7.2 Dimuon Trigger Rejection and Efficiency

When calculating trigger efficiency in what follows, the denominator is the number $J/\psi \rightarrow \mu^+\mu^-$ events for which both muons passed through the fiducial area of all views in all stations (in other words, events that could have been found by a perfect trigger), and the numerator is the number of dimuon events actually found by the trigger algorithm. Rejection is defined as the total number of bunch crossings divided by the number of crossings that passed the dimuon trigger.

Fig. 11.26 shows the $J/\psi \rightarrow \mu^+\mu^-$ (i.e. dimuon) efficiency and minimum bias rejection for two trigger configurations as a function of D-cut. The configurations, each studied for two values of proportional tube hit efficiency e, are as follows:

- 2/4 views: In each octant we look for tracks in all four views (R,U,V and S). A track is defined as at least two of the four views passing the D-cut.
- 3/4 views: In each octant we look for tracks in all four views (R,U,V and S). A track is defined as at least three of the four views passing the D-cut.

Examining Fig. 11.26 we can draw some very general (and fairly obvious) conclusions:

- As we tighten the D-cut, we will drive up the rejection and drive down the efficiency for any trigger scheme.
- The looser of the two trigger schemes ("2/4") has higher efficiency and lower rejection for any value of the D-cut.
- Lowering the proportional tube efficiency (from $\epsilon = 100\%$ to $\epsilon = 97\%$ in this case) causes a relatively larger change in the tighter trigger scheme ("3/4").
- The overall effect of lowering the tube efficiency to 97% is quite small.
- The tighter "3/4" scheme has good rejection (> 500) and good efficiency ($> 80\%$) for a range of D-cut values around 1.

These results are encouraging and indicate that we will have no problem triggering with this scheme if the assumptions that went into the simulation are valid. There is, however, some uncertainty in these assumptions. For example, although we expect it to be very high, we don't know what the proportional tube hit efficiency will be, hence a study of efficiency versus rejection for various values of ϵ is prudent.

Fig. 11.27 summarizes a study of dimuon trigger efficiency versus proportional tube hit efficiency for various triggering schemes. As above, "2/4" and "3/4" refer to the requirement that either 2 or 3 of the possible 4 views in an octant have tube combinations with $D < D$-cut. The schemes labeled "2/4(384)" and "3/4(384)" are directly comparable to the scheme used to generate the data shown in Fig. 11.26, and the others are discussed below.

The numbers shown in parentheses in the legend of Fig. 11.27, "(320)", "(352)", and "(384)", indicate the maximum tube number considered in each view. To motivate the

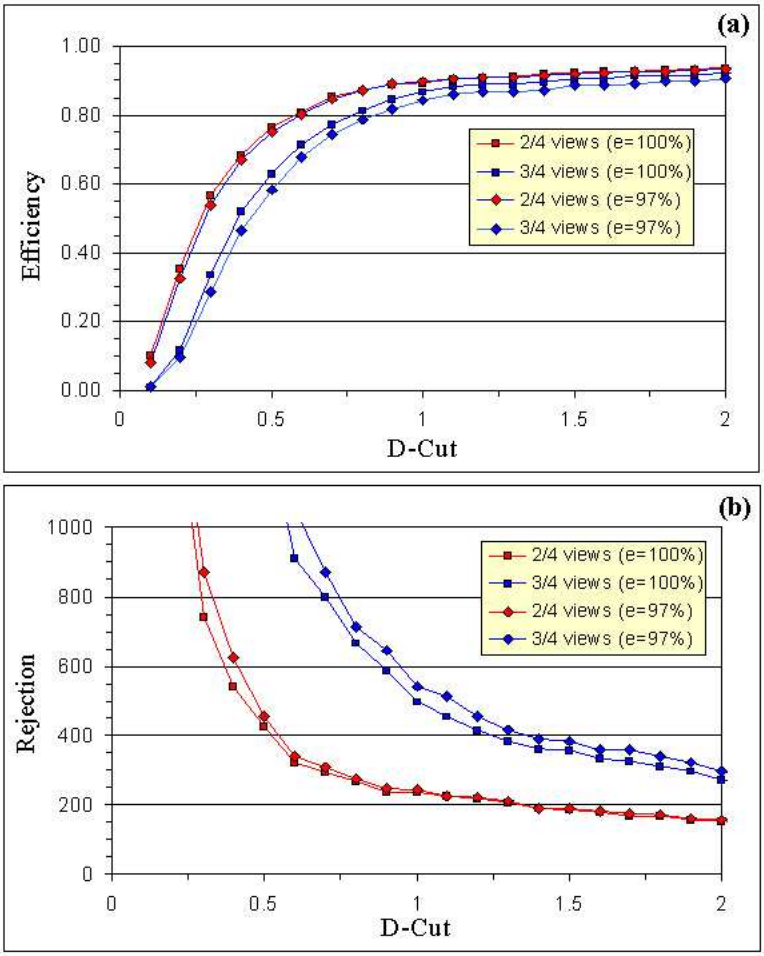

Figure 11.26: Dimuon trigger efficiency for $J/\psi \to \mu^+\mu^-$ events (a) and rejection for minimum bias bunch crossings (b), both as a function of D-cut.

study of this additional "geometric" constraint, examine the distribution of signal (blue) and background (red) tracks in Fig. 11.6. We see that most of the background is located in the innermost tubes, closest to the beampipe. Recall that in the BTeV numbering scheme, tubes are numbered from outside in, so that these inner tubes have the biggest tube numbers. Since there are 384 tubes in each view, the (384) classification means that all tubes are used, the (352) classification means that the innermost 32 tubes in each view are ignored, and the (320) classification means that he innermost 64 tubes in each view are ignored. Although the tactic of ignoring the innermost (i.e. hottest) tubes is not needed for the studies outlined in Fig. 11.26 and Fig. 11.27 where the average minimum bias occupancy $< N >$ is 2, we will see below that this approach might become desirable when $< N >$ is 3 or more.

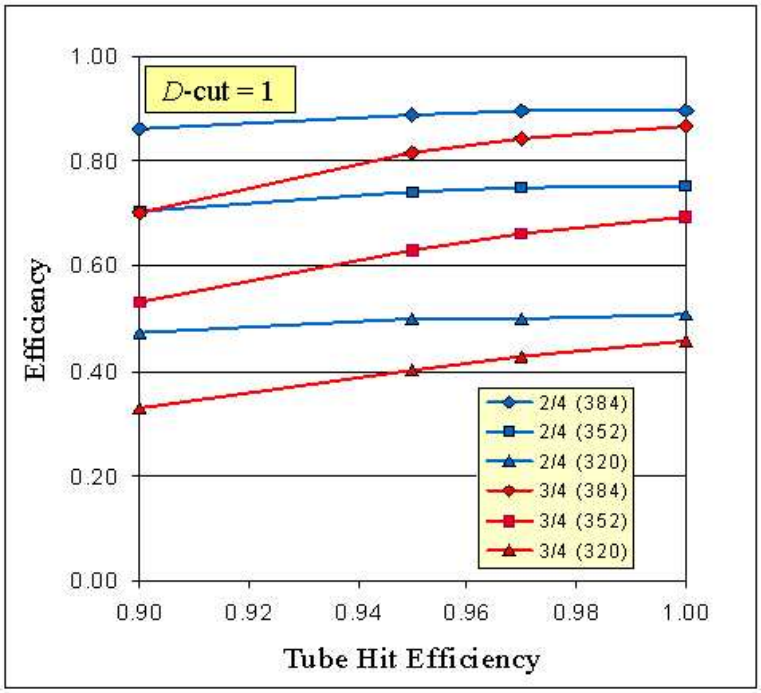

Figure 11.27: Efficiency for $J/\psi \rightarrow \mu^+\mu^-$ events for various trigger schemes as a function of proportional tube hit efficiency. A value of D-cut = 1 is used in all cases.

11.5.7.3 Performance at 396ns

As suggested above, a bigger concern is the potential uncertainty in the average number of minimum bias events per crossing that we will have to deal with. If the Tevatron runs at a luminosity of 2×10^{32} and a bunch spacing of $132ns$, we expect $< N >= 2$. If we run at $396ns$ bunch spacing at the same luminosity we would naively expect $< N >= 6$. For reasons we won't outline here (luminosity leveling, etc.) we expect that even if the Tevatron runs with a bunch spacing of $396ns$ we may not have to deal with minimum bias occupancy beyond $< N >= 3$.

We have simulated the efficiency and rejection of the "3/4" trigger scheme for $< N > =$ 2, 3, 4, and 5, assuming a proportional tube efficiency of 97%. The results are summarized in Figure 11.28.

We see that although efficiency is largely unaffected by increasing $< N >$, the minimum-bias rejection factor falls significantly. It is still true, however, that even for $< N >= 5$ we can achieve a rejection factor of 400 with 60% efficiency.

It is worth noting that significantly higher rejection as well as a much lower susceptibility to "non-muon" background hits can be achieved at the expense of efficiency by taking into account the very tight correlation between hit tubes in different views within a single station. This technique of "spacepoint" finding within some or all stations prior to correlating these

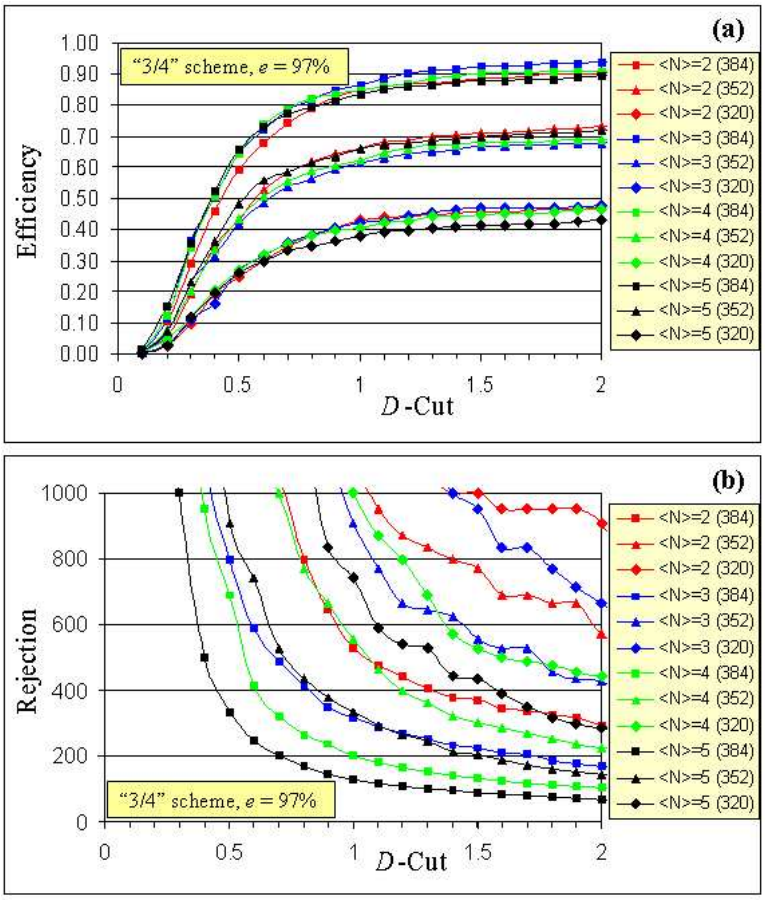

Figure 11.28: (a) Efficiency for $J/\psi \rightarrow \mu^+\mu^-$ events and (b) rejection for minimum bias bunch crossings using the "3/4" trigger scheme as a function of D-cut. Shown are the results for various average minimum bias interactions per crossing $< N >$, as well as maximum tube number requirement. The large scatter of the points at high rejection values simply reflects low statistics.

hits between stations (the latter step being the approach described in detail above), was in fact the first algorithm studied in depth when developing the baseline design. While we are not using the spacepoint method in the current baseline design because of its inherently lower efficiency and slower execution speed, this can be revisited if we are faced background rates that are much worse than anticipated.

11.5.7.4 DSP Timing Test Overview

In an effort parallel to the muon trigger algorithm design work described in the previous paragraphs, we have done significant R&D to develop a DSP test-stand and to port a speed-optimized version of the muon trigger code to this system. Fig. 11.29 is a photograph of

Figure 11.29: Texas Instruments TMS320C6711 DSP evaluation system.

the Texas Instruments TMS320C6711 DSP evaluation system used in these tests. Code developed by the University of Illinois at Urbana Champaign (UIUC) group as part of the BTeV-RTES effort was used to provide an efficient file-I/O path to the board, allowing us to test the DSP based code on the same large data-sample used in the efficiency and rejection studies described above.

Additional code developed at UIUC allows us to keep track of the number of clock cycles used in each part of the test code, hence accurate speed analysis is available to further guide code optimization.

11.5.7.5 DSP Timing Test Results

Running the DSP based trigger code we reproduce the results discussed above, and find remarkably good speed performance. Fig. 11.30 summarized these results for samples of both $< N >= 2$ minimum bias events (red diamonds) and $J/\psi \rightarrow \mu^+\mu^-$ events for which both muons were in the fiducial volume of the muon detector (blue squares). The vertical scale for the minimum bias events has been divided by a factor of 50 to allow this large sample to appear on the same plot as the smaller and $J/\psi \rightarrow \mu^+\mu^-$ sample.

The horizontal axis is in units of CPU "clock ticks", which simply needs to be divided by the clock speed of the processor to be converted to wall clock time. For a very modest 133MHz processor, for example, the average time for the trigger code run on samples of minimum bias and fiducial and $J/\psi \rightarrow \mu^+\mu^-$ events is 6 μs and 13 μs respectively. The time shown includes running on one view in all eight detector octants. To find the time required

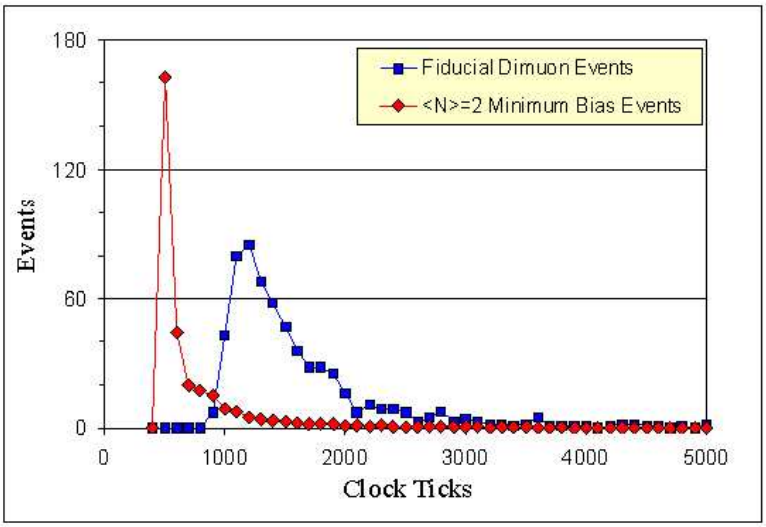

Figure 11.30: Timing results from running the muon trigger algorithm on one view in all eight octants, based on 16,000 minimum bias events (red diamonds x 1/50) and 631 fiducial $J/\psi \rightarrow \mu^+\mu^-$ events (blue squares). The average time for these distributions is 760 clock ticks and 1686 clock ticks respectively.

to run on all four views, as in the "3/4" scheme discussed above, a scale factor of four should be applied.

Queuing analyses and statistical resource simulations are in progress at this time. Preliminary results confirm that using "available-today" technology (150 MHz, C6711 processors), a farm of 220 DSPs can process all 4 views (R, U, V, S), if the workload can be perfectly distributed to any free processor. However, as it is by design our plan to employ the pixel trigger DSP farm to implement the muon trigger, there are architectural features which affect workload distribution. These are being studied, but all analyses to date confirm that a farm of 250 processors, with realistically projected performance improvements (e.g. 300 MHz) can accommodate the muon trigger workload.

11.5.8 GL1 Trigger

11.5.8.1 Introduction

The Global Level 1 trigger system (GL1) accepts trigger primitives from the L1 pixel and muon trigger systems and generates a trigger decision for every bunch crossing. The basic elements of the GL1 system are Input Matching Logic (IML), GL1 algorithm processor, GL1 Supervisor and Monitor (GL1SM) and the Information Transfer Control Hardware (ITCH). Incoming data from three data streams is matched by crossing number in the IML and queued for the GL1 algorithm processor. The first two streams come from the L1 pixel and muon trigger systems. The third stream is included for additional detector front-end

electronics or control system inputs but the current anticipated use of these auxiliary inputs will be commissioning and diagnostic triggers. The natural extension of the trigger decision-per-crossing is the requirement that the GL1 system account for every crossing. A timeout mechanism will be implemented to allow for lost events. Preceding subsystems are required to send a header for each crossing processed even if the header has no additional information. The incoming data will be filtered by the GL1 algorithm processor and used as an index into the current trigger table. The pre-scales are applied and a trigger decision is generated. The crossing numbers chosen for further processing are sent to the ITCH and queued to be dispatched to a L2/L3 processor over Ethernet. An accept message will be sent to all Level-1 buffers to cause them to reserve all the data associated with that crossing. Rejected crossing numbers will be handled in one of two ways. Either *A)* an L1 reject message will be sent to all Level-1 buffers to cause them to free the memory space associated with that crossing or *B)* no specific reject message will be sent and the Level-1 buffer will always write over the oldest data with the newest data and keep track of what crossing numbers are actually in the buffer. The decision between these two methods will be made based on the trade-off between the cost of additional memory in the Level-1 buffers needed for the longest trigger latency in *A* and the impact of many more messages in *B* on the decision network. The cost of the memory manager to *A)* garbage collect or *B)* manage crossing pointers is also a factor here.

11.5.8.2 GL1 Input Matching Logic (IML) and Incoming Data

The baseline architecture supports incoming event packets of 50 to 100 bytes from the L1 pixel and muon trigger systems. All events will have the BTeV standard bunch crossing number in the header and subsystem input will be matched using this number. It is required that auxiliary trigger inputs arrive at GL1 at the same time or before the arrival of the matching muon or pixel event packet. The exact time will be determined by the production hardware, but it will most likely be 1μs or less. The IML will collect information by crossing number until it has all three input packets (two if the auxiliary inputs are disabled) or until a timeout occurs. The GL1 system needs to have the same clock and timing inputs as the front-end DCBs so that it is synchronized with the front-ends. The IML will then queue the data for the GL1 processor(s) which will begin to process the information in the order of arrival.

11.5.8.3 GL1 Algorithm Processor

The GL1 processor will most likely be a small collection of processing elements because of the volume of information and data rate requirements. The matched packets from the three data streams will be filtered for rejected events and then indexed to the trigger table to produce an intermediate result. That result will be indexed through a prescale table to produce an ultimate crossing decision, that is, a go/no-go decision on whether the L2/L3 farm should process the crossing data. The decisions will be classified by type in order to allow simple optimizations later. The final decisions will be ordered by crossing number so some results

will be held up waiting for crossings that need more time. The time ordered results are queued into the ITCH. The processor will also produce and store some additional information that will help later processing stages. This may include portions of the incoming data that are not already saved in Level-1 buffers, intermediate results and/or trigger primitives to jump-start later processing. This information is added to the GL1 Level-1 buffer to make it accessible to L2/3.

11.5.8.4 Information Transfer Control Hardware (ITCH)

The ITCH must match the selected trigger data that needs further processing with resources that may be limited in availability, or algorithm. The processing resources are in the L2/3 farm and each processor may be off, doing diagnostics, be ready to process with a limited algorithm or be available for any type trigger. Those processors register with the ITCH as their processing resources become available. The registration mechanism allows for the L2/3 processor to indicate a desired trigger type. The ITCH maintains queues of crossings to be processed and resources available, matches the two and sends the processing element the crossing number to work on. It may also notify all of the Level-1 buffers of the assignment of data to resource although an alternative would be to have the resource request the data itself. That mechanism needs simulation to determine the trade off between latency and additional load on the network. If a processing element has considerable capability, it can continue to request crossings until it reaches an acceptable load. The ITCH must monitor the data queue sizes, report unacceptable levels and request resource additions or reductions from DAQ, L2/3SM, and/or Run Control. The ITCH is also the trigger accountant, reporting missing crossings and collecting the information needed to refine the trigger tables. The ITCH may be adaptively swapping trigger tables to optimize the trigger system to match the luminosity of the accelerator, to adapt to changing levels of processing resources or to accommodate to changes in other detector systems.

11.5.8.5 GL1 Supervisor and Monitoring (GL1SM)

The GL1SM will oversee the processing elements, distribute commands from the BTeV DAQ/Run Control system, assist with error mitigation, report and collect physics information, and account hardware diagnostic information. Some of this information can be displayed as histograms for the trigger operator and/or the experiment shift team. There will be a trigger expert terminal and display connected directly to the GL1SM, allowing a privileged operator to monitor, diagnose and adjust the GL1 trigger system independent of the rest of the experiment systems. More details on the GL1SM can be found in Section 11.7.

11.6 L2/3 Hardware, Software and Simulations

11.6.1 L2/3 Hardware

11.6.1.1 Overview of Baseline Design

The L2/3 Trigger will be implemented with an array of commodity CPU's and commercial network components. Fig. 11.31 shows an illustration of the layout of the components of the L2/3 Trigger. The 1536 CPU's in the baseline design consist of 768 1U rack mount dual-CPU PCs running Fermilab Linux. These are split equally between the 8 DAQ Highways, 96 PCs (192 CPU's) per Highway. Data from the L1 Trigger and the Front Ends are delivered to the L2/3 PCs via DAQ Fanout switches, 12 per Highway.

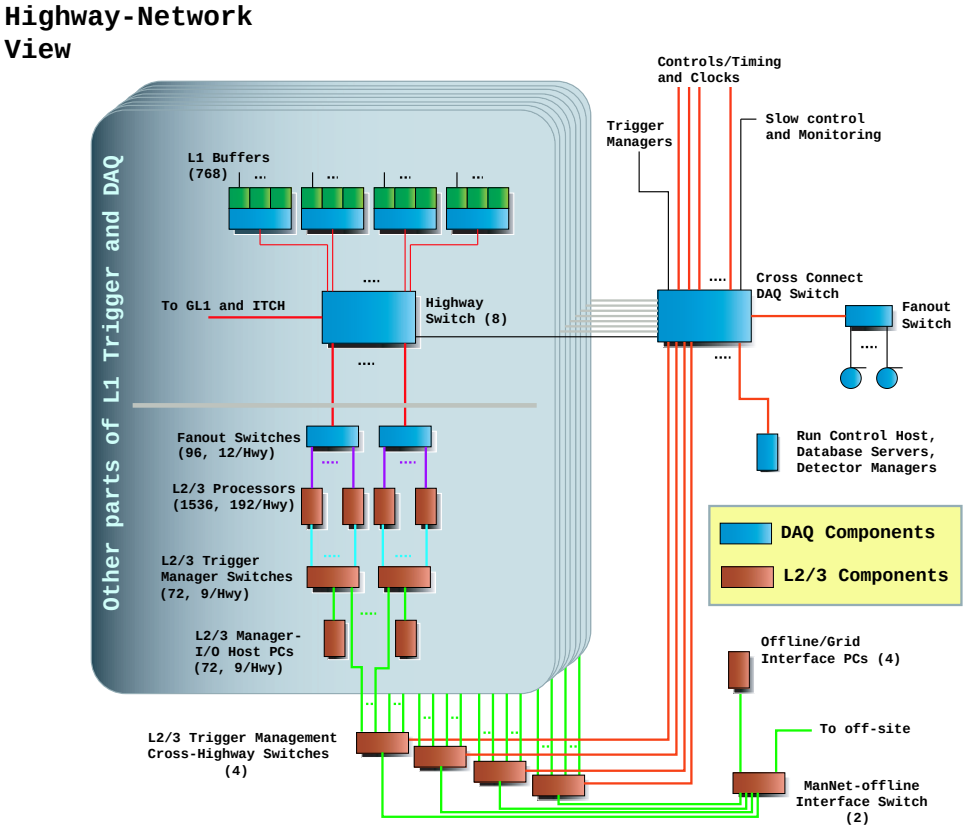

Figure 11.31: *Illustration of the layout of the L2/3 Trigger.*

The L2/3 Trigger CPU's are managed by a system of Manager-I/O Host PCs connected via a separate management network. This management system consists of 9 Manager PCs and 9 (Manager) switches per Highway. The Manager PCs from one Highway can communicate with the Manager PCs from other Highways via a Management Network (ManNet) Cross Connect switch. The Manager PCs are used for managing the L2/3 farm worker PCs,

as servers for database caches and data event pool caches as well as for use in monitoring the farm worker PCs, the data networks, and other components of the L2/3 Trigger system.

The L2/3 Trigger system will be located in a total of 30 racks on the second floor of the counting room, above where the hardware for the L1 Trigger will reside. The 96 farm worker PCs for each Highway will be housed in 3 racks, equally split between each rack. Each of these 24 racks will house 4 of the 24-port DAQ Fanout switches, and 3 of the Trigger Manager switches. Each farm worker PC will have two fast ethernet (100Base-T) connections to two of these Fanout switches. The management system will be housed in 4 additional racks while 2 racks will house L2/3 specific DAQ hardware.

Technologies using small single board computers running embedded Linux were investigated but these do not provide enough processing power. The relatively new blade servers are being investigated as a possible alternative to 1U dual processor PCs. Currently their cost per processing power is still too high. However they may become a viable alternative in the future. The advantages of the blade server are higher density and potentially lower power needs and better reliability and manageability. If enough power and cooling can be placed into each rack, with blade servers the 1536 farm worker CPU's could potentially be house in just 6–10 racks instead of 24 racks. In the future we expect the blade servers to become almost as much a commodity item as the 1U PCs are now, due to the current trend of processor improvement and mass market interest in blade servers. The L2/3 Trigger processing farm is expected to be heterogeneous.

Network Data Rate

In the baseline design the processing for the L3 Trigger takes place on the same PC as the processing for the L2 Trigger. The difference is that the L2 Trigger uses only a subset of the total available information, (the pixel tracking), while the L3 Trigger uses the entire information and basically performs all the processing that one normally associates with the offline reconstruction.

The rate of events (crossings) entering the L2 Trigger is 50 KHz. With an average event size of 250 KB/event, this means the average data rate entering L2 is 12.5 GB/s, or 1563 MB/s for each DAQ Highway. Each Highway has 12 DAQ Fanout switches, this means that each 24-port DAQ Fanout switch must handle a average data rate of about 130 MB/s, or 1042 Mbps. This can be obtained with 2 Gigabit fiber connections between the Highway Switch and each of the DAQ Fanout switches.

Each Fanout switch distributes data to 8 Farm worker PCs, giving about 16 MB/s per PC, or 8.1 MB/s per CPU. This corresponds to 130 Mbps per PC, or 65 Mbps per CPU. In the baseline design this is achieved with two fast ethernet (100Base-T, 100 Mbps) connections between each PC and Fanout switch. We expect that when the Farm Worker PCs are bought, these connections will most likely be implemented with Dual Gigabit ethernet. The price of Gigabit ethernet cards are already low enough and are included on many modern motherboards. However Gigabit switches are still expensive and we have chosen to not include them specifically for the DAQ Fanout switches in the baseline design.

The Manager PCs communicate with the Farm Worker PCs via a separate management network. This network consists of 24-port 10/100 switches with one of two Gigabit ports included. The Gigabit connection is from the Manager PC to the Manager switches, while single fast ethernet connections are used to connect the Farm Worker PCs to the Manager Switches. A separate management network helps achieve reliability and fault tolerance, as well as provide network resources for offline processing needs.

In the baseline design, after a L1 Trigger accept, all data for the event will be transferred to a L2/3 Trigger Farm Worker PC.[6] If an event passes the L2 Trigger, it is processed in the same PC through the L3 Trigger. Hence there is no data transfer through the network for entry into the L3 Trigger. The output of the L3 Trigger is 200 MB/s total at a rate of 2.5 KHz with an average event size of 80 KB. This corresponds to 130 KB/s per CPU or 1.0 Mbps per CPU. This is small compared to the incoming data rate and can be handled by the baseline DAQ network design without additional hardware on the Farm Worker PCs. The output data rate is 25 MB/s for each Highway, where the data is transferred from the Farm Worker PCs to the DAQ Highway Switch via the DAQ Fanout switches. From the Highway Switch the output goes out to data logging machines via the DAQ Cross Connect Switch at 200 MB/s.

CPU Speed

The L2 Trigger takes an input rate of 50 KHz and reduces it by a factor of 10 to 5 KHz. The L3 trigger further reduces this rate by another factor of about 2 to 2.5 KHz (as well as reducing the event size by a factor of 3.) To achieve this with 1536 CPUs requires, for example, the L2 Trigger to take less than 15 ms/event per CPU and the L3 trigger to take less than 154 ms/event per CPU. These latency times represent 100% CPU utilization which is not achievable. The actual baseline design therefore requires the L2 and L3 Trigger processing to take an average of 5 ms/event and 134 ms/event per CPU respectively. This corresponds to a CPU utilization of 60%. Note that 10% of the CPU is needed for DAQ event building and another 10% is allocated for RTES related and other monitoring. We assume that the overheads of the OS and of switching between tasks is 10%. Some allowance (10%) is also made for some inefficiency in a dual-CPU PC for two CPU's sharing the same system resources.

To estimate the likely performance of a CPU that would be used in the BTeV L2/3 Trigger Farm we looked at the CPU core speed trends in the last ten years. The CPU core speeds for INTEL and AMD CPU's is plotted against the release date of the CPU is given in Fig. 11.32. For later AMD CPU's where AMD quotes a performance rating (relative to a P4 CPU), this rating is used to provide alternative data points. Also shown on the plot is the Moore's Law curves for a doubling of CPU speed every 18 or 24 months. It can be seen that the data shows that the CPU speed doubling time to be between 18 or 24 months. The L2/3 farm CPU's will be bought over 3 years, 5% in FY07, 20% in FY08 and 75% in FY09. Using

[6]Note that in order to make efficient use of the network, in fact a buffer of many events rather than a single one will be sent to each PC for each data transfer request.

these fractions we can calculate an average CPU speed for a L2/3 Farm CPU. For CPU speed doubling times of 18, 24 and 30 months, the average L2/3 Farm CPU speed is projected to be 27 GHz, 16 GHz and 11 GHz respectively. For the baseline design requirements on the L2 and L3 trigger processing latencies, we have chosen that all processing time be compared against a CPU that will run the code 4 times faster than on a 3 GHz P4 Xeon CPU. We call this a "12 GHz P4 CPU" for the purposes of this document.

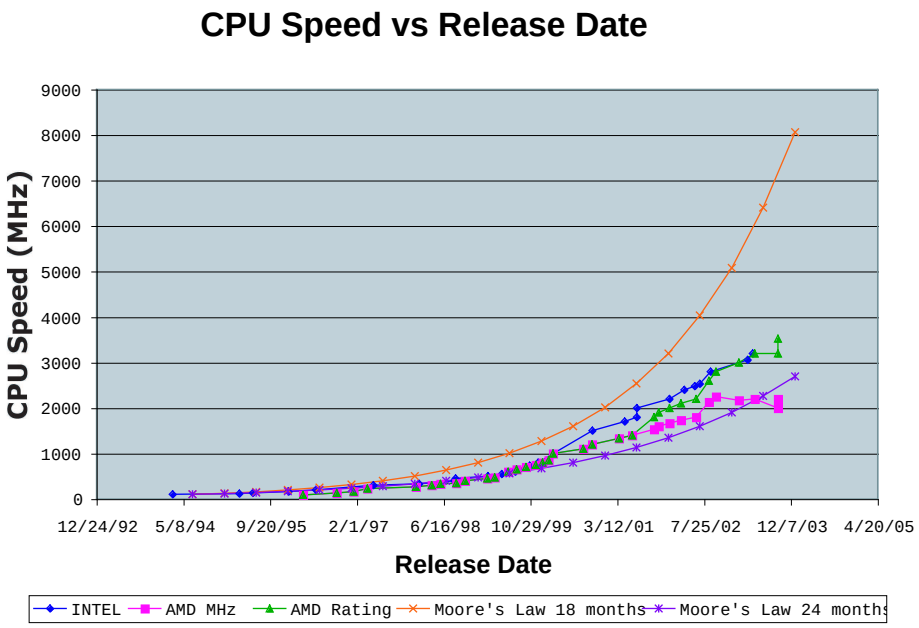

Figure 11.32: *CPU core speeds plotted against the release date of the CPU.*

The L2 Trigger code which achieves the required rejection was run on various PCs with different CPUs under Fermilab Linux. The results of this study are shown in Table 11.9. The data was generated using BTeVGeant. Both minimum-bias and $b\bar{b}$ events were run. It can be seen that we can satisfy the latency requirement using existing CPUs like those in Table 11.9. Originally the challenge looked to be in achieving the desired L2 Trigger efficiency and rejection with a CPU of reasonable speed. After work on improving the L2 algorithm, this has been shown to be achievable with existing, even low-end CPUs. We therefore expect that the CPU requirements will be set by the L3 Trigger code.

Besides the CPU speed and the actual Trigger code, the version of the compiler used and the optimization selected can also affect the execution speed. The L2 code was written in C++ and the compiler used was GNU gcc version 2.95.2. We expect that the compiler will improve in the future which will produce faster executing code.

There is currently no full L3 trigger code to benchmark. Although we have done many physics analyses of simulated data that includes reconstruction of neutrals and particle ID, we are not at the stage of having a full reconstruction package, unlike in a mature run-

Table 11.9: *The times taken for execution of the L2 Trigger code on various CPUs for various numbers of interactions per crossing, for both minimum bias only events (crossings) and those crossings also containing $b\bar{b}$ decays.*

CPU Type	**CPU Speed (GHz)**	**Time/Event (ms)**					
		2 Int/Cross		**4 Int/Cross**		**6 Int/Cross**	
		M.bias	$b\bar{b}$	**M. bias**	$b\bar{b}$	**M. bias**	$b\bar{b}$
INTEL P3 (SLOT 1)	0.5	9.5					
INTEL P3 (Coppermine)	1.0	4.3	5.2				
INTEL P3 (Coppermine)	0.866	5.0					
INTEL P4 (Xeon)	2.4	2.3		2.9		3.2	
AMD ATHLON	1.2	3.0	3.2	3.8	4.1	4.3	4.5
"12 GHz P4" (Projected)	12	0.46	0.49	0.58	0.63	0.64	0.67

ning experiment. However we show that the latency of 134 ms/event per CPU should be achievable with a CPU equivalent in speed to a 12 GHz Pentium IV Xeon CPU.

First we obtain the approximate time it takes to do the L3 tracking. We have already benchmarked the L2 code and found that pixel only tracking (for tracks associated with the L1 trigger vertex) takes less than 5 ms on a 2.4 GHz Pentium 4 CPU. To get an estimate of the L3 trigger using also the forward tracking detectors, a preliminary version of the L3 tracking was benchmarked that performed the full tracking including hits found in the forward tracking detectors, for all tracks found at L1. The execution times for this preliminary version of the full tracking is shown in Table 11.10. We are confident that from these results the charged particle tracking can be done confortably within the required maximum latency of 134 ms.

Table 11.10: *Execution or projected execution times for a preliminary version of the L3 charged particle tracking as described in the text.*

CPU	**Time/Event (msec) for bb**	
	2 Int/crossing	**6 Int/crossing**
Pentium III 866 MHz	201	433
AMD Athlon 1.2 GHz	135	298
"P4 12 GHz" (projected)	20	45

To estimate the amount of processing time required for the full L3 trigger we used the reconstruction code from FOCUS, a fixed-target charm photoproduction experiment that ran in 1996-7, and from E791, a fixed-target charm hadroproduction experiment that ran in 1990-1991. We used these as a benchmark of how long the L3 reconstruction code could take. Unlike for the real BTeV L3 code, these offline reconstruction code were not optimized

for execution speed, however it should give an idea of the likely execution time. Since the interactions and the spectrometer of BTeV is more complicated than that of FOCUS or E791, we must renormalized the FOCUS and E791 timings by the average number primary vertices produced, by the number of secondary particles produced and by the number of detector channels. This is done in Tables 11.11 and 11.12. The average of the projected execution times for FOCUS and E791 code is 134 ms on a "12 GHz P4 CPU". Note that the FOCUS and E791 offline reconstruction code was not optimized for execution speed as trigger code would be. We expect significantly faster executing reconstruction code when attention is paid to speed. We recognize this is an area of risk so we have allowed for a high contingency and in a later section we outlinethe steps to be taken if the L3 code is too slow.

Table 11.11: *Execution or projected execution times for FOCUS code. Renormalizations are made to try to project to a BTeV-like event. Each normalization is cumulative.*

Description	**Time/event (ms)**
1 million events in 8 hours (SGI Indy R5000 150MHz)	29
normalize to multiplicity at primary $\times\frac{17}{4}$	123
normalize to number of primary vertices $\times\frac{9}{2}$	554
normalize to number of detector channels/segment $\times 10$	5540
(BTeV/FOCUS: SSD 128/8; straws/PWC 27/13; ECAL 11/3; pixels in L2)	
normalize to 500 MHz Pentium III CPU (using g77)	2928
normalize to 1.0 GHz P4 CPU	1464
normalize to "12 GHz CPU"	122

Table 11.12: *Execution or projected execution times for E791 code. Renormalizations are made to try to project to a BTeV-like event. Each normalization is cumulative.*

Description	**Time/event (ms)**
Run on an SGI Indy R4000 100MHz	160
normalize to multiplicity at primary $\times\frac{17}{7}$	389
normalize to number of primary vertices $\times\frac{9}{2}$	1749
normalize to number of detector channels/segment $\times 10$	17490
normalize to 1.0 GHz P4 CPU (using Tiny)	1749
normalize to "12 GHz CPU"	146

As another comparison and to partly illustrate the gain in execution speed when more attention is paid to the execution time of the code we can also look at the existing CDF and D0 experiments which run at the Tevatron but with a central detector. Since the BTeV L3

Trigger is meant to do most of the traditionally offline reconstruction, we compare to the CDF and D0 offline (as opposed to their trigger code), see Fig. 11.33. For data taken at an average luminosity of 2.0×10^{31} cm^{-2}s^{-1}, the D0 offline processing takes about 25 sec/event on a "1 GHz CPU" This is much higher than their original specifications due to slow code not yet optimized for speed and due to software developer's appetite for including additional processing. We expect they could gain significant improvements in code speed in the future. The CDF offline code is more tied with their L3 trigger code and thus may represent a good comparison to show the gain in code execution speed when one is coding an online trigger rather than offline. The CDF offline code takes about 2.5 sec/event on a "1 GHz CPU" for data taken at an average luminosity of 2.0×10^{31} cm^{-2}s^{-1}. This is ten times faster than the D0 code. Part of this difference is due to the quite different numbers of tracking elements in the two detectors, but we believe part of this is also due to more attention being paid to execution speed. (At an average luminosity of 2.0×10^{32} cm^{-2}s^{-1} we project the CDF code to take 658 ms on a "12 GHz P4 CPU"). Thus although the CDF and D0 offline code runs much slower than our goal of 134 ms/event we think 134 ms/event is a challenging but achievable goal when proper care and consideration of code speed is taken into account right from the start.

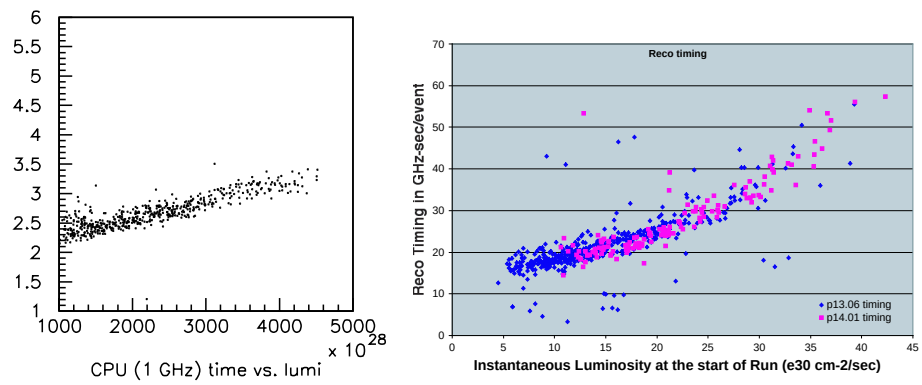

Figure 11.33: *Offline processing time per crossing for (left) CDF and (right) D0, plotted as a function of luminosity.*

It is reasonable to assume that this attention to code speed can be achieved since the traditionally offline reconstruction code for BTeV is all developed as part of the L3 Trigger project, rather than a much more open-ended offline project. At the beginning of data taking the emphasis will not necessarily be on the fastest processing speed for L3. At lower luminosity, as long as the efficiency and rejection goals of the L3 Trigger are met the processing could take longer than 134 ms/event. At this point the code would still be developed and tweaked for optimization of the physics goals. As the luminosity increases we can be quite certain of reducing the latency for L3 to 134 ms/event/CPU while keeping the necessary efficiency and rejection. The rejection requirement for L3 is only a factor of 2 reduction compared to the L2 rate. It is envisioned that the goal of 134 ms/event for performing full "offline" reconstruction at L3 would be achieved during the running of the

experiment as the code matures and the luminosity increases over time. Although the full "offline" reconstruction code is developed and written as part of the L3 Trigger project, we would not necessarily have to run all of it actually at the L3 Trigger stage. Especially at the beginning of the experiment when reconstruction code and calibration procedures are being ironed out, some offline processing or reprocessing will need to be done. Some of this processing could be done on free cycles of the L2/3 PC farm, since we assume a duty cycle of 33% for beam, or it could be done at PC farms at collaborating universities. As the code matures and is optimized for speed, more and more of the "offline" reconstruction would actually be run at the L3 Trigger stage. Some offline processing would probably always be desirable, like splitting off different physics data streams, or adding additional interesting physics streams. The data-taking duty cycle is low enough that the L2/3 PC farm should be available a fair fraction of the time for this sort of offline processing. In addition university groups have large PC processing farms that could be used.

Another point to note in this discussion of CPUs and L3 code processing speed is that the L2/3 PC farm is scalable. As part of the scalability requirement, additional L2/3 processing PCs can be added to increase the capacity of the L2/3 PC farm. To add more than 50% L2/3 processing PCs, additional DAQ Fanout switches would be needed unless these are already upgraded to Gigabit Fanout switches. In this case we could increase the number of L2/3 processing PCs by a factor of two without needing additional network hardware. However, due to space and power constraints, it would be more likely that one would actually replace slower old PCs with much faster up-to-date ones.

Data Storage in the L2/3 Trigger

One of the challenges in the BTeV Trigger system is to reduce the event size from an estimated 250 KB to 80 KB in the L3 Trigger. This is achievable because the objective is to run the equivalent of the offline processing at L3 and produce DST ("Data Summary Tape") as output. The DST contains all the information needed for physics analyses but does not contain enough information for full reprocessing (re-reconstruction) of the event. This event size reduction will be achieved in several stages over time as the luminosity increases. The real requirement on the L3 Trigger is an output rate of 200 MB/s or less. At full luminosity, this corresponds to an event rate of 2.5 KHz and an event size of 80 KB. A small percentage of (prescaled) triggers of larger event size with more event information will also be written out for monitoring and trigger studies. We expect that as the experiment (detectors and software) matures we will be able to write true DSTs at the L3 Trigger. Any reprocessing would at this stage only allow slight improvements in efficiency of physics analyses, or enable fewer and fewer new analyses.

The raw event size of 200 KB is a somewhat conservative estimate based on simulations of the BTeV spectrometer running at full luminosity. The occupancies and the projected numbers of bits per hit for each detector subsystem were used as input. With an average of 9 interactions per crossing, the average size of a crossing (event) input into the L1 Trigger is determined to be ≈ 100 KB, and the output from L1 is ≈ 108 KB/event. We have taken twice these numbers to get 200 KB and 250 KB for the event size input and output of L1.

This is thus a conservative estimate and it allows for possible extra noise hits and beam related backgrounds. For comparison, the average raw Run 2 D0 event size is about 250 KB, but about 75 KB of this is static "dictionary" information and is to be eliminated in future running. The CDF event size input into their offline farm is about 220 KB.

The average L3 Trigger output event size of 80 KB is thought to be achievable over the running of the experiment. It will be most important when the full luminosity is reached. As the experiment matures we will be able to drop all raw data and write out only high-level information (DSTs.) These DSTs will contain all the information necessary to do any physics analysis but will not contain enough information to reprocess or re-reconstruct the events. Although this might appear risky, in fact of course raw data is kept at the beginning of the experiment. Later as the Level 3 Trigger code and calibration procedures matures, less and less of this raw data will be needed for physics analyses. This is actually necessary since just the huge amount of data means that reprocessing is unlikely to be an easy option. In fact, in our experience, past fixed-target experiments that have produced vast amounts of data did not reprocess their full data sets.

As a comparison, the FOCUS experiment has DSTs with 3 KB/event. If we normalize to the primary multiplicity (17/4) and to the average multiplicities for B compared to charm decays (5/1.8), we would project a DST event size in BTeV of about 35 KB. Instead if we take a perhaps better comparison, the D0 experiment has DSTs of size about 125 KB/event. However their DSTs actually contain enough information for partial reprocessing, (which includes rerunning the calorimeter clustering and track finding and fitting.) Without this lower level information their DST size could be much smaller. For example the D0 "Thumbnail" data set which only contains very high level information that is efficiently packed is 5 KB/event. Similarly CDF produces a DST from their offline reconstruction that contains both raw and reconstructed data with an event size of about 300 KB. CDF is switching to a mode where they will write a DST with an event size of 170–190 KB. For physics analyses, CDF produces a "ntuple-type micro-DST" format that contains some raw data and averages about 30 KB/event.

Besides reducing the amount of information, some reduction can also be achieved with compression if necessary. For example, the GNU "gzip" utility can reduce the size of the FOCUS DSTs by more than 50%. The trade-off is that this would require additional CPU processing time. The event sizes quoted for D0 already include significant compression as part of their normal processing. The 220 KB CDF event size out of their L3 trigger farm can be compressed to about 160 KB.

Each L2/3 Trigger Farm worker PC and Manager PC will include large disks as part of a normal purchase. These will act as data caches to ensure good network data transfer efficiency and achieve reliability and fault tolerance of the system. For example, a 400 GB disk on each PC could buffer data input to the L2 trigger for 6.8 hours. Alternatively, if needed as an output buffer, a 400 GB disk on each PC could buffer the output from L3 for over 2 weeks. The disks on the Manager PCs will be used as database caches, for event pool caches for monitoring or trigger studies, and for use in offline processing.

11.6.1.2 Prototype, Pilot and Production Hardware

A prototype L2/3 trigger PC farm is being built using 40 dual processor worker PCs and a dual processor Manager PC in FY04. The worker PCs are an mixture of old Fermilab Computing Division Farm PCs. These consist of a regular ATX desktop case each containing two 333MHz Pentium II CPUs, or dual 500MHz Pentium III CPUs. These are sufficent for development purposes. The worker nodes are networked via a 10/100 switch to a Manager PC. The Manager PC is connected to the switch via two 1000Base-T gigabit ethernet ports. This prototype PC farm is being used for a number of development projects for the L2/3 Trigger. It will be used to try out different infrastructure software to handle and manage farm worker processing, such as FBSNG - the farm batch tools (http://www-isd.fnal.gov/fbsng) used by the Computing Division Farms group. It will be used for developing infrastructure L2/3 Trigger and DAQ specific code, and testing the monitoring and control code from the RTES project.

The prototype L2/3 Farm will also be used as one of the test beds for developing and testing software that will be needed to use the L2/3 PC farm as an offline processing resource, for example Grid-related software to enable part of the farm for Monte Carlo simulations or data processing when processor cycles are free. The prototype L2/3 farm will also be used to develop more reliable and manageable worker nodes, e.g. by investigating the effect of diskless operation or using solid state disks for the OS and system related software, or other minor hardware modifications for monitoring purposes. The software development work will be carried out by the BTeV Trigger team and members of the RTES group, together with some consultation with the Fermilab Computing Divsion, including the Farms group, the CDF/D0 Reconstruction groups and the QCD Lattice engineering group. When money becomes available (FY07), a more powerful pilot L2/3 PC farm will be built consisting of about 5% of the baseline farm (one full rack out of 24). These farms will be used for continuing software development and coding. If blade servers look to be a viable alternative, the pilot PC farm may consist of this technology for evaluation and development.

Since the L2/3 Trigger farm and DAQ infrastructure will not be purchased until relatively late in the project, simulations will have to be used for queuing studies. These studies will be carried out in the first two years as part of the development work needed to decide on the L2/3 Trigger farm design and technologies. Even without a large PC farm, individual hardware components can be tested to ensure that specifications can be met. In addition the prototype farm can be used for system queuing studies. When the pilot farm is available, these queuing and network throughput studies will continue on more up-to-date hardware, leading up to several data challenges nearer the end of the project.

11.6.2 L2/3 Software

The L2/3 trigger is a farm of commodity processors running Linux, or a similar, POSIX compliant, open software operating system. After an L1 accept all data for the event will be transferred from the Level-1 buffers to a L2/3 processor. If the event passes the L2 trigger it is then processed by L3 in the same farm node.

The distinction between L2 and L3 is that L2 uses only a subset of the data whereas L3 uses the full event data. In our baseline, the L2 trigger is solely based on the pixel data and consists of a sophisticated tracking and vertexing package, designed to reject most of the bunch crossings without heavy quark decays. However, since the data acquisition architecture delivers the entire event for all L1 accepts, we will have the possibility of creating other L2 triggers, such as an L2 Muon trigger.

The basic requirements for the L2 trigger are (i) a rejection factor of 10 on light quark crossings (ii) an acceptance higher than 90% on relevant heavy quark decays (iii) takes 5 msec or less per bunch crossing in a "12 GHz P4 CPU". We do not anticipate that memory utilization will be a critical issue.

The input data to the L2 vertex trigger consists of all L1 tracks and vertices and the raw pixel hits. The L2 algorithm performs Kalman filter track fits on the L1 tracks and refits the primary vertices. The kalman filter code is discussed in more detail in Section 11.6.2.4. It then searches for other primary vertices and detached secondary vertices. It also looks for high p_T single detached tracks, corresponding to decay modes with only one charged prong (e.g. $B^+ \to \pi^+ \pi^0$).

The goal of L3 is to achieve another factor of 2 in background rejection and to reduce the size of the event by a factor of 3. The L3 algorithm will be similar to the full offline reconstruction. We have prototype code for forward tracking, K_s reconstruction, particle ID in the RICH, and electron, photon and π^0 reconstruction in the calorimeter. The CPU and memory requirement for this L3 stage is specified in the requirements so that software developers can pay proper attention to performance issues. The L3 code must not consume more than 134 ms per event per CPU to perform all reconstruction and event formatting in a PC with 1 GB of RAM.

11.6.2.1 L2 Algorithm

The L2 pixel algorithm has basically two distinct components:

- Refinement of the L1 pixel tracks using a Kalman fit, pointing to the relevant L1 vertex that triggered the event.
- Search for primary vertices, secondary vertices and isolated detached tracks based on L2 tracks.

Given the basic feature of the pixel detector, we perform tracking/vertexing pattern recognition and fitting in three dimensions at all stages of the event reconstruction. Tracks are assumed to be straight lines in the non-bend plane (horizontal) and near perfect circles in the vertical plane. Unlike L1, track and vertex error matrices are estimated rigorously based on the known pixel resolution and the track momenta, that is, multiple scattering is always taken into account in the correlated position error calculation.

Currently only L1 tracks and the associated hits from the inner and outer triplets are used in the L2 fits. If it is found necessary to improve the L2 efficiency, extra hits on these

tracks can be searched for. In addition extra tracks can be reconstructed from the unused hits.

The Kalman filter is the track fitting engine that returns a confidence level based on hit position errors and the best track parameters either at the vertex, or at any arbitrary extrapolated (or interpolated) position. This fitting method is fairly rigorous but quite CPU intensive. The implementation can be fast because we can assume uniform magnetic field, or, if corrections are needed for long and/or low momentum tracks tracks, such corrections are small and can be treated as perturbations.

The trigger strategy is based on the fact that the primary vertex has at least 5 tracks. Thus L2 tracks are always associated with at least one primary vertex. The primary vertex that L1 trigger associated with detached tracks is used as a seed for the L2 vertex.

Vertices are constructed based on a linear approximation of the trajectories near the beam line. That is, we use the Kalman filter to extrapolate the track close to the beam line, then perform the vertex reconstruction using these track parameters entirely ignoring the magnetic field. If need be, the tracks can be extrapolated again to the fitted Z vertex position. This iterative procedure converges very fast, and we do not need to compute intersection of circles, which is a bit more complicated than lines.

The trigger decision is based on transverse momentum and detachment criteria. A secondary vertex must satisfy the following criteria: (i) tracks must have a confidence level greater than 2.5%, must be detached from the primary vertex by more than 3.5σ and must have transverse momentum > 0.5 GeV; (ii) all such detached tracks must have the same sign p_z and be pointing away from the primary vertex (to reject nearby primary vertices); (iii)the secondary vertex must have a confidence level greater than 2% and must be detached from the primary vertex by more than 3.5σ ; (iv) the secondary vertex must have an invariant mass less than 7 GeV and more than 100 MeV outside the K_s mass. An event passes L2 if it has either a detached secondary vertex or a high p_T detached track.

As the pixel planes need to be retracted while we load the Tevatron, we need to re-align the pixel detector at the beginning of each store. The pixel detector alignment will be based both on accurate position sensor information and on tracks coming from minimum bias interactions emerging from the luminous region. Again, the Kalman filter will be the workhorse of the alignment package. Pattern recognition and Kalman fits will be performed using hits with errors based on estimated alignment errors. The Kalman filter is able to report deviations of the hits with respect to the best approximation of the trajectory, leading us to a iterative algorithm. We will assume that each half-pixel plane moves as a unit; we do not plan to find alignment constants for each individual pixel at the beginning of each store.

11.6.2.2 L2 Implementation

The L2 code will be written in C++. The production code will then be optimized for speed, and will be tightly interfaced to the data acquisition software. There will be no unnecessary re-copying of the raw pixel data to the L2 internal buffer, since judicious use of pointers will be used instead ("shallow" copies).

Table 11.13: L2 trigger efficiencies

Event type	L2/L1
Light quark	7%
$B_s \to D_s K$	85%
$B^0 \to \pi^+\pi^-$	87%
$B^0 \to J/\psi K_s$	78%
$B^- \to \pi^- K_s$	72%

The L2 vertex trigger code will be tightly coupled to the pixel alignment code. It will share the same interfaces to raw data and to the Kalman filter package. In addition, the L2 code initialization will be able to access the BTeV Geometry Database upon demand. The Pixel Alignment code, running concurrently with the L2 code on the L2/3 farm, will generate new alignment data periodically (for example each time the pixel detector moves, after Tevatron injection).

We will maintain and upgrade the interface between the BTeV Monte-Carlo code and the L2 trigger. That is, all modifications of the L2 code will be first tested on Monte-Carlo data. A Monte-Carlo representing the "as built " version of the detector is therefore mandatory. The L2 code will be able to assemble and fit a given track using the exact hit list generated by the Monte-Carlo. Finally, the L2 code will also be callable from the off-line code, and can be re-run on selected raw data.

11.6.2.3 L2 Simulations

The current L2 vertex trigger code has been run on 4 different decay modes representing different decay topologies, and on light quark events. The timing results are shown in Section 11.6.1. The current code runs over 7 times faster than our requirements so we have plenty of leeway to improve the efficiency by refining the tracking and vertexing algorithms or adding extra algorithms such as a L2 muon trigger. The efficiency results are shown in Table 11.13.

11.6.2.4 L3 Algorithms

The goal of the L3 software is to fully reconstruct the heavy quark decays and characterize them. This does not necessarily mean reconstructing the entire event: for instance, if the vertex pattern and reconstructed mass is consistent with an all charged exclusive decay, the time-consuming electromagnetic shower reconstruction phase can be skipped. The L3 event reconstruction stage will be based on results from L2, raw hit data and the alignment/calibration constants from the database. The output will consist of a semi-inclusive list of 3- or 4-momentum vectors (ie. with or without particle ID), and vertex patterns. Additional information will also be made available for further off-line analysis, such a partial list of hits or ADC values associated to critical tracks. We are not planning to transfer the

entire raw data and reconstructed data to permanent storage media. However, during the commissioning period, there will be enough disk available on the L2/3 farm to store $\approx$ weeks of raw data, for L4 ("offline") subsequent re-analysis.

During the pre-conceptual R&D phase of BTeV, we wrote prototype reconstruction code for all sub-detectors (except the Muon detector). We now describe the essential features of these reconstruction codes, or refer to the detector specific section.

Forward Track Reconstruction

The Forward Silicon and the Straw detector are analyzed jointly. The goals of this reconstruction phase are to:

- improve the momentum determination
- improve the vertical track position at the vertex
- provide accurate track positions measurements at critical location for particle identification (at the entrance and mirror positions of the RICH, at the E.M. calorimeter and at the muon wall.
- reconstruct K_s and Λs

An L3 charged track can emerge from the pixel detector, or from the forward tracking system. Tracks emerging from the forward tracking system may be tracks from the $p\ \bar{p}$ collision that do not have enough hits in the pixel detector to form a reconstructible pixel track, or from decays of K_s and Λ^0, or from re-interactions. Tracks from re-interactions could be of interest because they can potentially confuse the pattern recognition within the forward tracking detector itself, or in the RICH, muon detector or the calorimeter. They will be reconstructed last, on an as needed basis.

The Forward Silicon and Straw detectors are handled jointly. There are no distinct "Silicon tracks" and "Straw tracks". This is because most of the tracks encounter first a few silicon detector, graze the hole in a straw station and finally emerge as "Straw tracks". Since there is an overlap between the Silicon and the Straw detectors, a track at any given station can have hits in both the Forward Silicon and Straw detectors.

The last tracking station, located between the electromagnetic calorimeter and the RICH detector, plays a different role from the others: its sole purpose is to pin-point the track at the RICH mirror and/or at the front of the calorimeter. Multiple scattering is such that this additional measurement point does not improve the knowledge of track direction upstream of the RICH.

Unlike the pixel detector, the forward tracking pattern recognition must recognize cases where multiple tracks go through the same cell, because the solid angle covered by such cells is much larger. Thus, it is a much more involved pattern recognition problem. Most tracks of real interest are already fairly well defined as they emerge from the pixel region. For such tracks, it is simply a matter of allocating the correct set of Silicon or Straw hits along their

path, refitting to improve the track parameters. Once that part is done, only about half of the hits remain unused. These hits are used to reconstruct K_s, Λ ("Vees") and background tracks.

A prototype for this first stage has been written. The reconstruction of the forward tracks coming from the interaction region is seeded by the L2 pixel tracks. Tracks are propagated to the next encountered station (Silicon or Straw), a list of candidates hits for each plane is built, and Kalman fits are performed for each potential combination. For the straws, we fit a set of 2 or 3 hits within one stereo view to a given track rather than individual hits to avoid unnecessary combinatorics of many possible Kalman fits. Thus, left-right ambiguities are usually lifted prior to fitting. Arbitration can be postponed until we reach the next station downstream, if not enough hits are found in a given station, for instance. Although rather CPU intensive, this procedure converges in the sense that only one set of hits remains when the track reaches the RICH (or leaves the spectrometer). In many cases, the arbitration must be postponed until the next stations are reached, because a Straw hit can be "overwritten" by an other tracks, biasing the measured track position.

The preliminary tracking efficiency versus momentum, for tracks reaching the RICH, is shown in Fig. 11.34. About half of the inefficiency is due to inaccuracies in the multiple-scattering accounting in the Kalman fits and the other half is due to pattern recognition confusion or double occupancy in the straws. The momentum resolution obtained via this full pattern recognition is in very good agreement with the fits performed in the context of BTevGeant, where all hits are always assigned to their respective tracks. Once station 6 (located in front of the RICH) is reached, the probability of accepting "ghost" tracks is quite small, about 0.5%.

The reconstruction efficiency for low momentum tracks curling in the downstream end of the magnet and leaving the spectrometer between first two station will be definitely lower than for tracks reaching the downstream end of the spectrometer. By reconstruction efficiency, we mean here the probability to obtain a set of track parameters consistent with the real one, not merely the probability to find a matching set of hits that seem to fit the track.

Unlike in the pixel detector, the magnetic field can no longer be assumed uniform and of constant direction. Exact numerical integration of the field along in the Kalman fit extrapolation routine requires too much CPU time. Therefore there will be a set of fast parametric extrapolation routines based on tabular data describing integrals of fields (or "moments").

Kalman Filter

The BTeV software suite includes prototype Kalman filter code which is currently used as the final track fitter for the simulated physics analyses of BTeVGeant output, for the L2 prototype code and for the forward track reconstruction package. It is also used by the RICH code to interpolate/extrapolate tracks into the RICH detector and by the electromagnetic calorimeter code to extrapolate tracks to the face of the electromagnetic calorimeter for track-shower matching. At present the Kalman filter code works only as a final fitter - codes

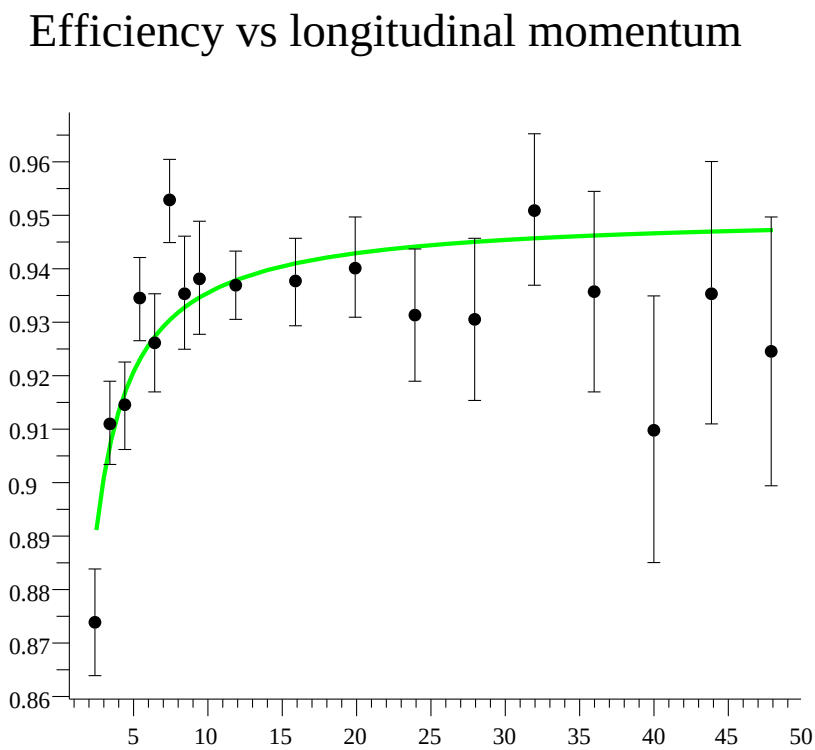

Figure 11.34: The preliminary efficiency versus the longitudinal momentum Pz (GeV/c) for the L2-seeded (pixel-seeded) L3 tracks. The fit is simply there to guide the eyes, the function is $E_\infty * (1 - k/Pz)$. The parameter k is statistically significant, indicating problems at low momentum. This is where our geometrical acceptance drops sharply. Note that the loss of "efficiency" includes particles lost through interaction in the material of the detector and due to (recoverable) problems with the current detailed description of multiple scattering sources in the analysis program.

which use it must supply it with a list of hits which are to be considered as a single track. These hit lists are provided by different codes for the different uses of the Kalman filter and those codes are described in the corresponding sections of this report, for example the previous section which described the forward tracking code.

The Kalman filter code automatically includes multiple scattering and energy loss for the material associated with each hit in its hit list. That is, it knows the amount of multiple scattering associated with the material in a single pixel station, a single forward silicon plane or a forward straw plane. Auxiliary routines, not part of the Kalman filter package, discover additional material through which the trajectory of the track passes, such as detectors with missing hits, and inert material, such as beam pipes, and pressure vessel walls. These auxiliary routines need to be improved to understand more of the details of the support structures in the detector.

The Kalman filter code can be asked to filter in several different directions. It can run

from the start of the track to the end or from the end back to the start. It can also fit in both directions to produce smoothed estimators of the trajectory at any point along the trajectory. This feature is used by the alignment code to compute residuals at each hit.

The Kalman filter code also has interfaces for adding new hits at the start or end of an existing fitted track. At present this interface is primitive and difficult to use. There is work planned to improve the interface so that it is both easy and natural to use the Kalman filter within pattern recognition codes. When that work is done many optimizations will be possible in the L2 code and in the forward tracking code.

The Kalman filter code can select from several algorithms for propagation of the track and its covariance matrix through magnetic fields. This allows fast propagation code to be used when speed is required, such as in L2, and slower, but more precise, propagation code to be used for the final fits before tracks are used in physics analyses. At present only two choices of field integrators are available, representing the extremes of fast but approximate and precise but slow. It is envisaged that several intermediate solutions will be developed.

Several other optimizations are under consideration to speed up the operation of the Kalman filter when it is used in the L2 trigger. For example, some of the matrix multiplications involve matrices with small off-diagonal elements. In certain cases these can be dropped and the matrix operations hand coded to exploit the resulting zeros.

Preliminary K_s^0 Tracking Studies

We now describe the reconstruction of the K_s^0 for which we have no (or not enough) pixel hits. From a detailed study on the track topology of these $\pi^+\pi^-$ pairs, we conclude that the largest reconstructable sample consists of tracks reaching straw station 6, for which we have 3 consecutive straw stations in the nearly field-free region beginning at $z \approx 2.75$ meters from the magnet center. Stations 4, 5 and 6 are located at $z \approx 2.9$, 3.3 and 3.8 meters from the magnet center, respectively. The following algorithm has been partly coded and is currently under study:

- Selection of "un-used" straw hits. We mark all the hits used in the above pixel-seeded L3 tracks, as "used", thereby getting rid of about 1/3 to 1/2 of the available hits.
- Reconstruction of straw hit triplets (or doublets) within a straw stack (or "view"). Despite the lack of good constraints from unknown track slopes, the multiplicity of such small, 2D tracks within a station and a stack is not overwhelmingly large.
- Reconstruction of 2D tracks between stations 4, 5 and 6, for each stereo view.
- 3D track matching of the 2D views.
- First reconstruction of a 3D K_s^0 vertex using the non-bend plane 2D vertex as a seed. The K_s^0 trajectory is constrained: it must come from the selected L2 primary vertex for which we have good detached p_t. This allows us to obtain a preliminary determination of the $\pi^+\pi^-$ and K_s^0 momenta as well as the K_s^0 mass.

- Search for confirming hits in upstream stations, followed by track and vertex refits.

Preliminary studies indicate that we can reconstruct K_s^0's that decay upstream of station 3 with about 60% efficiency. The loss of signal is mostly due to the high occupancy in the straws.

RICH Particle Identification

The RICH reconstruction code will be based on the algorithms developed to assess the performance of the detector. We note here that all charged tracks entering the RICH volume must be reconstructed so that all Cherenkov rings can be characterized based on the pion hypothesis. This is the starting point in the algorithm, as there are too many interfering hits in the 3σ ring to identify a given track independently from all the others. The output of the RICH code is a set of electron, muon, pion, kaon and proton identification probabilities for each track.

Numerous high momentum charged particles, particularly electrons, that are created in the detector material (pixel, forward silicon, beam pipe, etc.), can produce light in the RICH. Hence, we are writing a dedicated pattern recognition code aimed at reconstructing rings emitted by such high velocity particles, in order to mitigate this occupancy problem. Hits and tracks segments from Straw and Forward Silicon stations 5, 6 and 7 could also be helpful at determining such background rings.

Muon Reconstruction

The Muon reconstruction will be based on the algorithm developed for the L1 trigger, and on the knowledge of the parameters of tracks emerging from Station 7. Thus the muon code will mostly consist of track segment matching.

E.M. Reconstruction

As part of the design effort and detailed evaluation of the expected performance of the detector, a stand-alone reconstruction code has been written. As for the previous code, it is assumed that the charged particles hitting the detector are known and characterized.

Heavy Quark Filter

We require the L3 trigger to reject about 50% of the events that pass L2 in order to achieve an output rate of 2.5 kHz when running with an average of 6 interactions per bunch crossing. In order to be as efficient as possible for known decay modes of interest yet keep the trigger as open as possible to new ideas we adopt the following strategy. We conduct a search through a list of decay modes of interest, using as much information from the sub-detectors as necessary to test the relevant hypothesis. These events are selected with highest priority. We then select events with evidence of a heavy quark vertex. We expect the rate of events with reconstructable b-quark decays to reach about 1 KHz when running at full luminosity. Charm and other calibration and monitoring events will be taken and pre-scaled to keep the output rate below 2.5 kHz.

11.6.3 Risk Mitigation for L3 Processing Speed and Event Size Reduction

Two significant unknowns for the L2/3 Trigger system is the L3 Trigger processing time and the L3 output event size. These two are correlated since the faster the L3 trigger processing is done the more likely we will not need raw data and can summarize (reduce) the L3 output data into a smaller size that do not include raw data and only physics quality data. The uncertainty is caused by the lack of a full L3 trigger code and the uncertainty in the CPU performance at the time when the L2/3 Farm worker PCs are bought. The mitigation of this risk in terms of the amount of L3 processing that can be done and the event size out of L3 is described in this section.

Note that it is reasonable to assume that attention to code speed can be achieved since the traditionally offline reconstruction code for BTeV is all developed as part of the L3 Trigger project, rather than a much more open-ended offline project. We attempt to maintain this focus on performance with the specification of a definite L3 code peformance goal right from the beginning of the L3 trigger project. Additionally at the beginning of data taking the emphasis need not be on the fastest processing speed for L3. At lower luminosity, as long as the efficiency and rejection goals of the L3 Trigger are met the processing could take longer than 134 ms/event and the output could be more than 80 KB/event. At this point the code would still be developed and tweaked for optimization of the physics goals. As the luminosity increases we can be quite certain of reducing the latency for L3 to 134 ms/event/CPU while keeping the necessary efficiency and rejection. The rejection requirement for L3 is only a factor of 2 reduction compared to the L2 rate. It is envisioned that the goal of 134 ms/event for performing full "offline" reconstruction at L3 would be achieved during the running of the experiment as the code matures and the luminosity increases over time.

Mitigation for Slow L3 Processing

Although the full "offline" reconstruction code is developed and written as part of the L3 Trigger project, we would not necessarily have to run all of it actually at the L3 Trigger stage. Especially at the beginning of the experiment when reconstruction code and calibration procedures are being ironed out, some offline processing or reprocessing will need to be done. Some of this processing could be done on free cycles of the L2/3 PC farm, since we assume a duty cycle of 33% for beam, or it could be done at PC farms at collaborating universities. As the code matures and is optimized for speed, more and more of the "offline" reconstruction would actually be run at the L3 Trigger stage. Some offline processing would probably always be desirable, like splitting off different physics data streams, or producing additional interesting subsets of physics streams. The data-taking duty cycle is low enough that the L2/3 PC farm should be available a fair fraction of the time for this sort of offline processing. In addition university groups have large PC processing farms that could be used.

Nevertheless we describe a scenario to handle the possibility that the L3 processing is too slow. In all scenarios we would try to not lose interesting physics data. The results of simulations given in Table 11.9 show that the L2 Trigger can be run well within existing

CPUs. In the worse case one can consider that we only run the L2 trigger and store the output for L3 processing until the L3 Trigger can catch up.

The BTeV Level 2/3 trigger calculations are performed by a farm of 1536 CPUs, which are split into eight Highways. Each highway receives up to 6250 events per second that have passed the Level 1 trigger. Each of the 192 processors of a highway receives about 33 events per second and performs the Level 2 calculation on all of them, using about 5 milliseconds/event. Only 10the Level 2 selection. The 3.3 events are then passed to the Level 3 calculation that uses an average of 134 milliseconds. The Level 3 trigger passes about 50which comes to 1.6 events/CPU per second, or about 313 events per highway per second.

In the worse case of not running the L3 Trigger at all we would run the Level 2 software on all the events, taking 16.5on each CPU. On average, 3.3 events/sec per CPU would pass the Level 2 trigger and require processing through Level 3 on the same processor. A fraction of these events could be processed through the L3 Trigger while the rest could be written to local disk for later processing. With 200 GB of disk per CPU, even if no events were processed through L3 one could store a maximum of about 67 hours worth of data. This is assuming one keeps the event size at 250 KB/event, without any event size reduction. In practice some events would be processed through L3 while the rest are store and processed either on the L2/3 Farm when the luminosity is lower or no data is being taken. The stored events could also be processed through L3 by other computing resources, e.g. those located at the universities of collaborating institutions. Note that the data rate out of L2 is 10 times smaller than the incoming rate and can be easily handled by the DAQ network if needed.

An alternative to not doing the L3 processing, or doing the full L3 processing on a subset of the data is to do partial L3 processing on all events (crossings). The results of simulations given in Table 11.10 show that the full charged particle tracking should be able to be performed within the maximum allowed L3 processing time. This means that the partial L3 processing could consist of the full charged particle track reconstruction for which some event rejection could be done. Moreover since the raw pixel data makes up 50% of the raw event size, some data reduction can also be achieved by dropping raw tracking detector hits. Again there is enough local disk on each CPU to store this data output from partial L3 processing for a substantial time. If a rejection factor of say 1.5 (instead of the normal 2) could be achieved with partial L3 processing, the L3 output data could be stored for about 100 hours. With a data reduction factor of say 2 (instead of the normal 3), this storage time would be extended to 300 hours, or about 12 days. This should provide enough time to perform the rest of the L3 processing on the L2/3 Farm during idle cycles or even off-site on farms at collaborating institutions.

Mitigation for Larger than Expected L3 Output Event Size

The other uncertainty is whether we can obtain the specified 3 times reduction in event size to 80 KB/crossing. This specification is due to the requirement of writing output at less than 200 MB/sec to the data archival system. This requirement is for a sustained 200 MB/sec output, whereas the average duty time of data taking is assumed to be 33%.

We have already mentioned above how, even if one did not do L3 processing and kept the event size at 250 KB, the data could be stored locally on each Farm Worker node for 67 hours. So in the worse case scenario, with no L3 processing and no data reduction the sustained output data rate would be 417 MB/sec. The 67 hours of storage time should be enough buffering to output almost 50200 MB/sec. In this absolute worse case scenario one would still have enough bandwidth to output almost all crossings with interesting b-quark decays, one could easily select more drectly produced b-quark over c-quark events at the L2 trigger stage. More realistically and as mentioned above, partial L3 processing would be performed, either the full L3 for a fraction of all events out of of L2, or partial L3 processing performed over all events out of L2. Either way, the data would rate would be reduced so that more output efficiency to data archive could be achieved for shorter lived b-quarks as well as for c-quark events.

If the full L3 processing is done and only the data reduction was not done, i.e. keeping 250 KB/event, one would not lose any data at all relative to the case of 80 KB/event. This is because there is enough buffering on local disk that the instanteous output rate of 625 MB/sec, would be reduced to 208 MB/sec with the assumed 33% data taking duty cycle.

11.7 Trigger Supervision and Monitoring

11.7.1 Overview

The Trigger Supervisor and Monitor (TSM) is the system that resets, initializes, controls, and monitors status and statistics from all component modules of the BTeV trigger. The TSM system is self sufficient in that it provides complete control and monitoring capabilities for the trigger without relying on external systems for data transport. Any products that are used to construct the TSM are considered part of the TSM system. Furthermore, it encompasses a hardware platform that is used by RTES [20] to implement fault detection, fault mitigation, and error handling functions. The TSM includes fault detection and mitigation infrastructure. The RTES Collaboration is developing fault detection and mitigation software that will be used as optional components with the TSM infrastructure.

The BTeV trigger consists of a large collection of distributed processing elements operating on detector data that flows in parallel through multiple data streams. Fig. 11.35 shows a block diagram of the trigger system showing some of the parallelism (specifically the trigger highways) as well as the interconnections between the data-processing hardware, the DAQ, and the TSM. The data-processing elements range from "hardwired" logic elements to powerful data-processing computers with varying capabilities for receiving, acting upon, and generating messages that are used to implement necessary supervision and monitoring capabilities. The supervision and monitoring of so many elements is a technical and operational challenge. The TSM system is critical for maintaining adequate resources during BTeV data taking, and the system itself is complex enough so that some of its resources will be needed to maintain its own operational state. This will be accomplished with the aid of fault detection and fault mitigation tools provided by RTES.

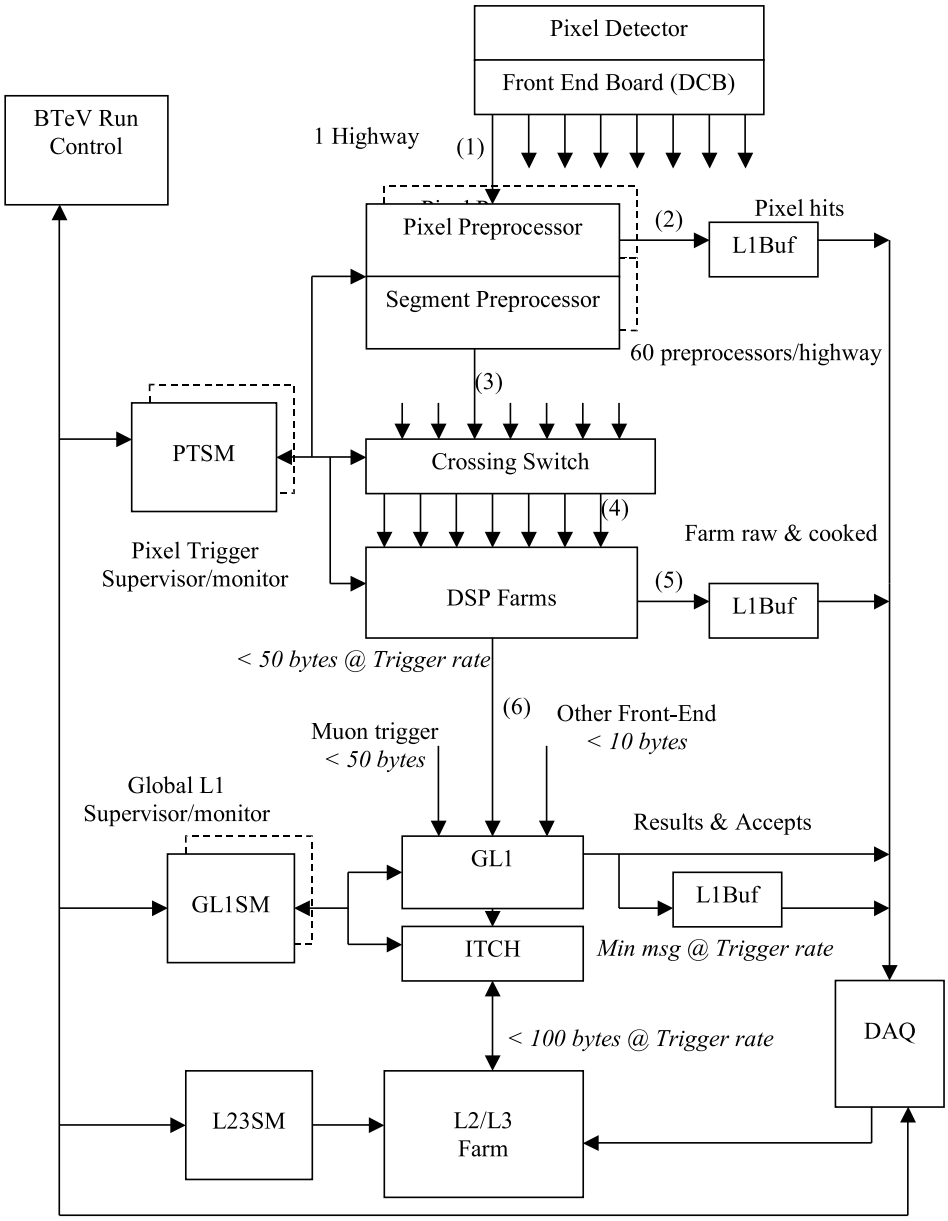

Figure 11.35: Block diagram of the BTeV trigger and interconnections between the data-processing hardware, the DAQ, and the TSM system.

11.7.1.1 Functional requirements

The TSM system provides the connectivity to send and receive messages or stream data containing commands, initialization sequences, configuration data, error messages and data pertaining to status and statistics for subsystems in the BTeV trigger. Functional descriptions of these messages are classified as supervisor functions or as monitor functions.

Supervisor functions

- initialization and configuration
- command execution and distribution to subsystems
- autonomous error handling

The TSM configures both the trigger hardware and software. The initialization is hardware dependent. The initialization of L1 trigger subsystems, for instance, includes downloadable FPGA firmware, execution code for processors, set points, and various trigger tables. Initializing the entire trigger system is a large task that may take several minutes, but we expect that a complete initialization of the system will not happen often. As described in the architecture section below (see Section 11.7.2), the TSM system has a hierarchical structure. The intermediate and lower level nodes in this structure have enough intelligence and storage resources to start up data processing and supervisory functions locally. During commissioning, these nodes will be updated frequently as changes and improvements are propagated through the system. After the trigger has been commissioned, the initialization of the processing nodes and startup of the system will take less time. During normal running, code updates to the intermediate nodes can occur in parallel with data taking. In this manner the system will be streamlined as it is integrated with the BTeV detector and electronics.

TSM messages are, for the most part, organized by the physical arrangement of the hardware. High level command messages will be translated, as needed, at each TSM intermediate node so that messages can be propagated to low-level nodes, which may have varying capabilities for supervision and monitoring. Furthermore, there will be different broadcast attributes so that commands can go to a single hardware component, a subset of components, or an entire collection of components.

Monitor functions

- error message collection and reporting
- hardware and software status messages
- status message collection and logging
- data histogramming

The TSM reads data from all programmable devices in the three trigger levels, L1, L2 and L3. This includes the programs, parameters, device configurations, status and error

messages, temperature and voltage measurements, as well as processed data at useful probe points in the data stream.

The characteristics of the messages used to monitor trigger components are the following. Since the bandwidth of the TSM network may be limited compared to the amount of monitoring data that will be available, each level of the TSM hierarchy will have the resources to perform data "compression." For example, as monitor messages propagate up from low-level hardware to the intermediate TSM nodes, the amount of data can be reduced by having the intermediate node send a single "group reply" that indicates that all of the low-level hardware components reporting to the node have replied to a request. The extent of this kind of data compression will be controlled by operational policy, where a policy is a coherent set of parameters that activate and configure "pluggable" software components. The policy or its parameters can be changed dynamically depending on running conditions. Hence, for debugging and commissioning purposes, the compression may be completely disabled so that every message is passed from the lowest levels to the highest. For normal operations, compression may be engaged at a "normal" level (determined by policy). Whereas a response to persistent error conditions, could cause the compression to be enhanced by exponential prescaling (for example, send each of the first 5 messages, then a message of the form "5 more of," then a message of the form "25 more of," then "125 more of," and so on).

Another form of "compression" will be applied to histograms that are generated for monitoring purposes. Similar to the "group reply" that is generated by an intermediate node in response to a request, individual histograms from low-level worker nodes can be combined (for example, by adding all of the individual histograms together) to reduce the volume of data sent over the TSM network.

11.7.2 Architecture

The TSM system and its network can be viewed as a pyramid structure with a host node at the top, and a message distribution network that expands through intermediate nodes down to worker nodes at the bottom. A block diagram of the TSM system is shown in Fig. 11.36. At the top of the figure is the TSM Host (TSMH) that communicates with subsystem-specific TSM hosts, which in turn communicate with TSM workers. The TSM subsystems are the following:

- PTSM - pixel trigger supervisor and monitor
- MTSM - muon trigger supervisor and monitor
- GL1SM - Global L1 trigger supervisor and monitor
- L23SM - L2/3 trigger supervisor and monitor

These subsystems implement the connection to the trigger processing elements and can be implemented as relatively powerful processor nodes, as software threads in the target trigger processor element, or as dedicated hardware connected to the trigger elements. The

flow of information will be almost exclusively vertical, with commands flowing from hosts to workers and error and status messages flowing back to the host. The TSMH is controlled by BTeV run control during data collection or detector commissioning, but the host can also perform diagnostics and housekeeping tasks in parallel when resources are available. The subsystem-specific TSM hosts are fully functional as the logical root of their TSM tree. The PTSM host, for example, can control the entire pixel trigger for debugging and commissioning purposes. The TSMH serves as the master control for subsystem-specific TSM hosts, and can also imitate run control functions during commissioning.

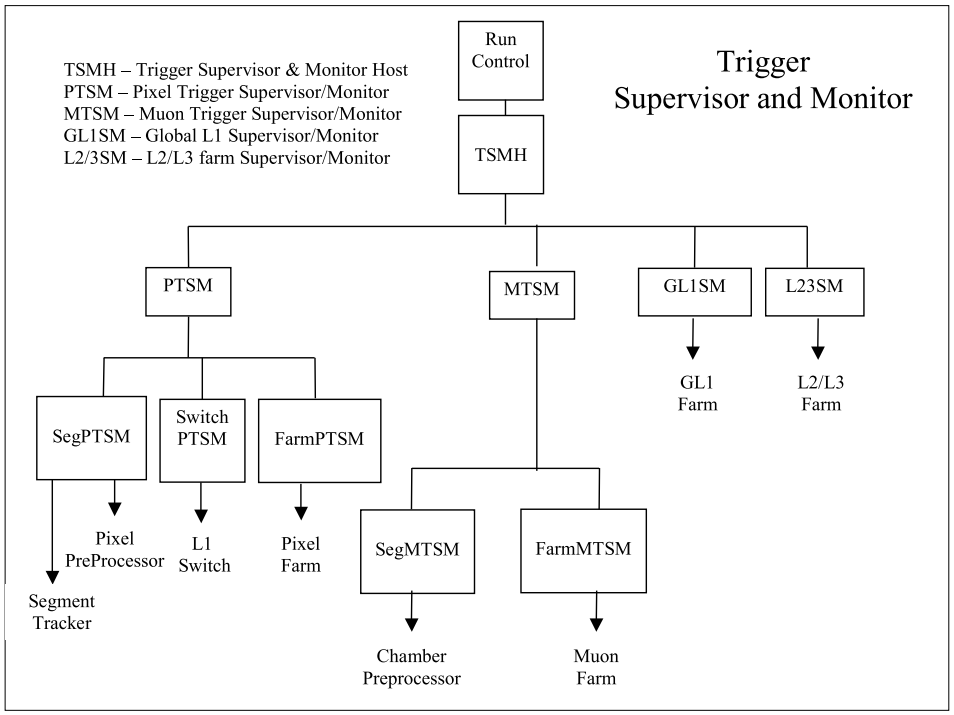

Figure 11.36: Block diagram of the Trigger Supervisor and Monitor system.

The TSM system will provide the connectivity for all of the above described messages. The system must provide extremely reliable delivery of these messages. However, the latency requirement of commands is mitigated by the fact that all data streams are tagged with the crossing number and do not need to be tightly synchronized. Furthermore, the detector data flowing through the DAQ and trigger is not synchronized across the data paths, but it is locally synchronized within the processing elements whenever necessary. This design feature allows relaxed latency requirements for the entire trigger system and DAQ at the cost of larger data buffers in the data paths. Recent decreases in the price of memory supports this design decision. The message latency requirement for the trigger is not stringent and is therefore not a significant driver for the hardware design. Most current state-of-the-art data link designs can meet the link rates and bandwidth needed. Errors will generate messages to be sent to control elements but fault mitigation will be attempted as close as possible to

the hardware reporting the error. The error messages can take some time to get to the host, but it must be reliable delivery

The determination of link bandwidths in the TSM system requires a compromise between a fairly low rate needed for normal operations consisting mostly of traffic pertaining to command and status messages, and the infrequent need for high rates to shorten the time needed to download FPGA code, trigger tables, and application code or to accommodate the error and recovery messages of a large failure. The peak link bandwidth will be determined by the bandwidth needed to start up the trigger system in a "cold start" mode, where some reasonable amount of initialization and configuration information needs to be distributed to the trigger processor elements. In a "warm start," after the system has run and halted, startup can be very quick due to reuse of much of the configuration. During normal operation, some of the TSM bandwidth will support the movement of histograms used to monitor the trigger operations.

11.7.3 TSM Software

The heterogeneous architecture of the TSM system requires a diverse collection of software elements. The TSM software will be described first in terms of the required elements that are common to all levels, then in terms of specific differences between levels. Finally, the relationship between the TSM software and the RTES project [20] is described.

TSM Similarities

Each processor of the TSM system must be capable of at least a minimal self-boot on power up, such that human intervention is not required to deliver the processor to a state where it can receive messages from a higher level. Message handling resources must be provided for down-coming, up-coming, down-going, and up-going messages. Since the TSM is a hierarchy of processing resources, command and control messages will come down from above (down-coming), and after local processing, if appropriate, will be distributed to lower level processors (down-going). Responses and errors may be received from lower levels (up-coming), and after local processing, if appropriate, will be forwarded to the next higher level (up-going). Interrupt service and task scheduling, at least at a primitive level, must be supported.

At each level, the capabilities of the TSM processors will be used in accordance with established policies. Policies correspond to configuration-defined controls that enable or disable distinct capabilities on a particular processor. For example, any TSM processor will be capable of autonomous down-going and up-going traffic as a result of local processing, which results in either locally generated command and control to the lower levels (automated recovery), or locally generated status and error reports to a higher level (self-initiated detection). However, these features will not be active unless allowed by downloaded policy configuration data. Another example concerns network bandwidth. Mid-level TSM processors will be capable of accumulating messages and providing condensed summaries (for example, all DSPs have booted successfully), as well as forwarding individual messages (DSP

N has booted, DSP N+1 has booted, etc.). The extent of data compression (as mentioned in Section 11.7.1) will be determined by policy configuration data. This capability/policy approach will allow debugging and commissioning to have access to as much or as little knowledge and control of the system as required from the top level, and it will allow normal operation to be tuned for performance by trading autonomy for detail.

Finally, each processor of the TSM must be able, as appropriate, to support code and configuration caching. As part of the self-boot process, or in response to a command from a higher level, each TSM processor will attempt to load a local copy of its executable code, and will attempt to configure itself based on a local copy of its configuration data. An important first step in this process, however, is to certify that the local copy is valid. To do this, each TSM processor will review its local storage resources, compute checksums, and compare them to checksums provided by the next higher level of the TSM system. If the local copy is valid, it is loaded without further intervention. If the local copy is not valid either by virtue of being obsolete or corrupt (i.e. the checksums do not match), the TSM processor will wait for the next higher level processor to provide a valid copy of the code and/or configuration data. Symmetrically, each TSM must, as appropriate, service requests for code and configuration coming from below, either from its own storage resources (if the checksums match), or by forwarding the request to a higher level.

TSM Differences

The TSM processor heirarchy consists of Linux processors (with local disk storage) or Linux processes with access to disk storage at the highest levels, and diskless embedded processors (DSPs) at the lowest levels, with a range of processing and storage resources in between. In particular, the lowest level consists of a local manager process, within an embedded worker processor. At each level, the resources dictate slightly different capabilities and behaviors.

At the topmost TSM level, the TSM host (TSMH) must be able to self-boot, and must be able to automatically start all services and initialize all required resources. We envision the TSM host to be a Linux process, or processor, with startup scripts that are invoked during the boot process. Message handling will include message archiving to a BTeV-standard database. Also, a graphical user interface will be native to the TSM host to display system status and to allow both global and selective command and control. However, in normal operation, the TSM host will defer to BTeV Run Control for primary start, stop, and report commands. The TSM host need not necessarily run a real-time operating system.

At the subsystem-specific TSM host level, most of the TSMH features will be available, but many will be disabled. To support debugging and commissioning, each subsystem-specific TSM host must be fully capable of acting as if it were the TSMH, for a reduced implementation of BTeV involving only that subsystem. Hence the pixel trigger can be fully managed from the PTSM host, and the muon trigger can be fully managed from the MTSM host. This includes messaging and the user interface, but may exclude message archiving. During normal operation, the subsystem-specific TSM hosts would defer to the TSMH for all command and control, acting largely as an intelligent message passing element. This

intelligence would be manifest by converting generic start, stop, and report commands into subsystem-specific equivalent commands, as well as interpreting status and error messages from lower level processors and composing higher level summary reports. The subsystem-specific TSM hosts need not necessarily run a real-time operating system.

At the regional TSM processor level, of which there may be more than one depending on networking considerations, mid-grade computers, with or without disk, will be deployed. The primary role of the regional TSM processors is to provide message handling, and as controlled by policy configuration, mid-level error detection and mitigation.

At the Farmlet manager level, an embedded processor will be used. Its resources will be severely constrained. It will be diskless, but will have flash memory to act as a storage resource. In addition to message handling between the regional TSM processors and the embedded worker processors, the Farmlet manager will also be responsible for direct, low-level control of the workers.

The lowest level of the TSM heirarchy is the local manager level, implemented as code within the embedded worker processor, along with the physics application that performs the fundamental function of the trigger. The local manager process will accept messages from the physics application (via API calls), and will forward them to the Farmlet manager. The local manager, as directed by the Farmlet manager, may stop and/or reload the physics application, or change its configuration variables. The local manager will operate under, or integrated with, a real-time kernel serving the embedded worker node.

As described above, each level of the TSM must be capable of playing an appropriate role in code and configuration caching. The topmost TSM host will be able to directly obtain code (from a BTeV standard database or from a local cache) necessary to operate without human or ancillary system intervention. The lowest level TSM process, or processor, is relieved of the obligation to provide code or configuration to lower levels, as it is at the bottom level. Furthermore, the lowest level TSM process is exempt from having a code or configuration cache; it is free to act as if the "local copy" (which does not exist) is always corrupt, and must always be downloaded from the next level up.

L23SM Subsystem

While the preceding discussion has been focused on the L1 trigger, the same perspectives apply to the L2/3 farm, with the following small differences. All TSM "processors" are processes, and each runs on a Linux computer. The "depth" of the regional TSM heirarchy may be very shallow, and there will be no layer that corresponds to the "farmlet manager" layer in the L1 trigger.

TSM software, and RTES

The TSM will most likely need to utilize external products (libraries and executables). Only those products that pass BTeV quality standards, have undergone sufficient testing, and can be supported by the staff (administrative and software development) will be considered. Any products that have value, but have not passed all the criteria for inclusion, will be treated

as add-ons or plugins, which can be easily excluded. This includes RTES components. The TSM will have a basic, limited operating mode that will allow the system to be booted, verified, and run with a minimal set of core external products. The TSM will define the abstractions or APIs and component architecture, which allow other systems or products to add new functionality to it without producing a new software release. RTES and other groups will be required to build components to this interface. The RTES group will work closely with BTeV to develop their systems to the same standards to minimize research and development time up front, and maintenance costs in the end. Areas of shared interest are most likely the message/data passing infrastructure (protocols and formats) for commands and monitoring information, database interactions, sensor reading, and APIs used by BTeV developers.

11.8 RTES

11.8.1 Overview

The BTeV trigger system encompasses approximately 5000 CPUs distributed over a three-level trigger architecture. Time critical event-filtering algorithms that run on the trigger farms are likely to suffer from a large number of failures within the software and hardware systems. If fault conditions are not given proper treatment, it is conceivable that the trigger system will experience failures at a high enough rate to have a negative impact on its effectiveness. It is also likely that an administrative staff and cast of experiment operators will not be able to service simple problems, or analyze complex problems or relationships in a timely fashion to avoid data loss. This was called out in a technical review of the trigger system as an area of significant concern.

To address this concern, we have established the BTeV Real Time Embedded System Collaboration (RTES) [20]. Funded by a $5M grant through the NSF ITR program, RTES is a group of physicists, engineers, and computer scientists working to address the problem of reliability in large-scale clusters with real-time constraints, such as the BTeV trigger. RTES is defining a software infrastructure to detect, diagnose, and recover from errors not only at the system administration level, but also at the application level. This infrastructure must be highly scalable to minimize bottlenecks or single points of failure. It has to be verifiable to make sure that it performs its functions in a timely fashion, extensible by users to acquire new detection/analysis methods, and dynamically changeable so that it can be reconfigured as the system operates. The problem is being approached by using a hierarchy of monitoring and control elements. The architecture is such that lower levels have high data rates, short reaction times, and a narrow view of system components, while higher levels have aggregated data summaries, longer reaction times, and a more global perspective of the system.

The goal of the RTES project is create fault handling that can be used by all components in the BTeV trigger and DAQ. This subsystem must be capable of accurately identifying problems and compensating for them. This includes application related activities such as changing algorithm thresholds, and overall system activities such as load balancing. As many

recovery procedures as possible must be automated. A simple example is the ability of the system to switch to a hot-spare L1 processing board when a failure is detected in a board that is actively processing data. Operators and system developers must be able to easily incorporate new procedures or policies into the system. The operators must be able to easily select error handling policies, and a detailed record of observations and actions must be kept to facilitate reproduction of analysis results and to identify long-term trends.

Creating a single subsystem for handling faults across the DAQ and the trigger can benefit the experiment by lowering new procedure integration costs, and by reducing the amount of knowledge necessary to operate and maintain the system. This is done by introducing a standard set of interfaces and protocols that reduce the number of conversions and products that must be developed and maintained. The development of standards for error handling and reporting means that the information produced or exchanged between applications can be easily processed.

Each of the universities involved in the RTES Collaboration has expertise in some aspect of the problem. In some cases, they already have toolkits that have been used to solve smaller scale problems related to real-time embedded systems and fault management. BTeV has established a prototype architecture for the trigger that is being used as a model for RTES software development. The prototype helps to establish subsystem boundaries, to give a sense of scale, and to identify required interfaces and error conditions. This prototype uses DSPs as the embedded L1 processors. Fig. 11.37 shows a block diagram of the trigger components. The farmlet shown in the figure is essentially a single-event input queue (FPGA) with three to six servers. It also contains a microcontroller that is used for configuration, supervision, and monitoring.

11.8.2 RTES Deliverables

The technologies introduced by RTES are discussed below. They are *ARMOR*s for L2/3 processors and Manager-I/O Host PCs, *VLA*s for embedded processors and specific monitoring tasks at L2/3, and *GME* for system modeling and configuration. These are the deliverables of the project. Each of these deliverables is used for different aspects of the trigger, and all of them must work together.

The deliverables form a toolkit. This includes core infrastructure (libraries and APIs) as well as methodology for organizing and creating fault tolerance components. The toolkit is designed to be expanded as the experiment comes to life. The toolkit will include many BTeV-specific fault detection/action components for both hardware and application software.

11.8.2.1 ARMORs

The University of Illinois has produced a fault management software component called Adaptive, Reconfigurable, and Mobile Objects for Reliability (ARMOR). ARMORs are multi-threaded processes composed of replaceable building blocks called *Elements*. Elements communicate by way of messages. The architecture makes this a highly flexible system that

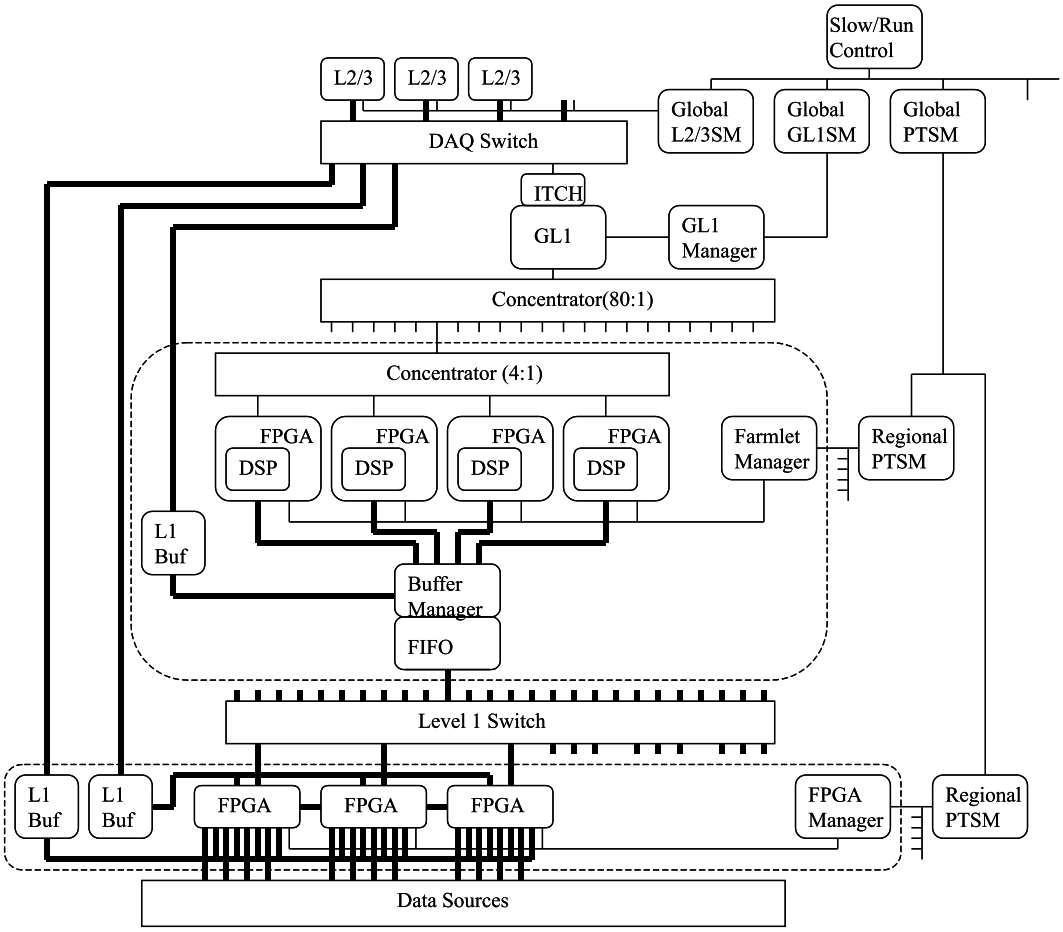

Figure 11.37: Block diagram of BTeV trigger components

includes modules such as recovery-action elements, error-analysis elements, and problem-detection elements that are developed and configured independently. ARMORs can be configured in a hierarchy across multiple processors to provide complete system coverage. Fig. 11.38 illustrates a simple armor configuration, where a node has a main ARMOR daemon watching over the node and reporting to higher-level ARMORs out on the network. Elements within these node-level ARMORs work together to make sure all nodes are operating properly. Another example of a standard ARMOR is the execution ARMOR. This ARMOR is responsible for protecting a single application. This type of protection does not require modifications to the program; it simply watches a running program. It can restart the application, generate messages for other elements to analyze, or trigger recovery actions based on returns codes from the application.

Within the trigger, ARMORs can provide error detection and recovery services to the trigger application and any other process running on L2/3 nodes. They can also watch for hardware failures. ARMORs are designed to run under an operating system (such as Linux or Windows [23]) and are not well suited for embedded systems with limited memory and

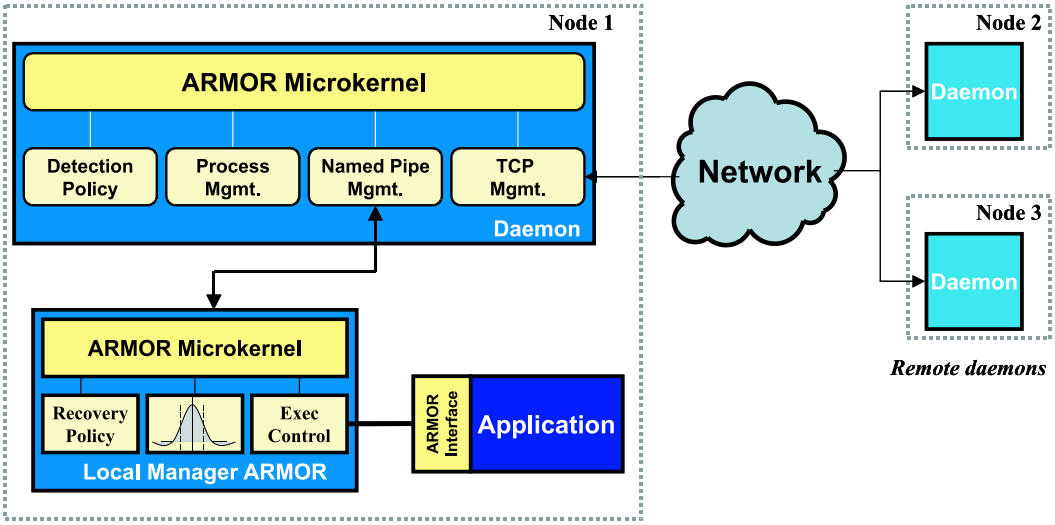

Figure 11.38: Simple ARMOR configuration

processing-time requirements, for which we are developing VLAs (see next section). Using the ARMOR API, a trigger application can report specific errors as well as other information directly to one or more elements. For example, data processing rate measurements and data quality measurements can be sent directly to ARMORs to be distributed to running elements for analysis. The BTeV online group is currently evaluating ARMORs to watch over DAQ and trigger related processes [19].

11.8.2.2 VLAs

Syracuse University and the University of Pittsburgh are developing a concept called the Very Lightweight Agent (VLA). A VLA is a software entity designed to collect various environmental and process related measures, analyze them, and perform actions in a highly constrained environment such as the L1 trigger. VLAs may be developed as standalone processes, threads, or a collection of functions that maintain the state of a subsystem within a larger application [25]. Given memory, CPU time, and network bandwidth constraints, a VLA will decide how to organize itself to provide the best possible results. In order to achieve this goal, a VLA may make use of real-time scheduling, priority queuing, and hierarchical rules to guide its decisions.

Within the context of the L1 trigger, VLAs will watch for fault conditions such as trigger algorithm crashes, link failures, the inability of processors to keep up with data rates, and processes running longer than expected. Since the L1 trigger is so restricted in terms of its resources, VLAs will rely on a higher-level control system to perform complex analysis and decision-making. It is easy to believe that running any amount of VLA code during the first part of data taking will cause the processor to fall behind. Therefore, a VLA will need to be smart enough to change its behavior as data taking progresses, to know problem priorities, and to know the best time to report status and fault conditions to the overall control system.

For the L2/3 trigger, VLAs may reside directly inside the trigger executable and perform similar function as in the L1 trigger. They will also be used to collect other hardware specific information such as CPU temperature readings and fan speeds.

An initial VLA prototype has been implemented and its interaction with an ARMOR was successfully tested. This test took place over the Internet, to verify the robustness and the distributed nature of the VLAs and ARMORs.

11.8.2.3 Modeling Tools

The ISIS group at Vanderbilt University has produced a graphical modeling tool called the Generic Modeling Environment (GME) [17]. The tool enables the user to do "Model Integrated Computing" (MIC). This tool allows a designer to model many aspects of a system by creating diagrams. The concept is similar to CASE tools in that it captures component relationships and properties. It is different in that it is not tied to a particular modeling paradigm, such as UML [24]. The tool allows a designer to create a domain specific set of rules that define modeling components. In essence the designer creates a modeling paradigm specific to a particular project. GME allows one to independently capture different aspects of a system using shapes, properties, associations, and constraints specific to the project and then combine them to form a system image [16]. Just as a compiler forms a parse tree from a programming language and then processes the information in the tree to create machine specific assembly code, GME creates a set of data structures representing the information in the models and allows "model interpreters" to generate information about the system. Typical model interpreters are C/C++ code generators and system configuration generators. Examples of aspects are hardware configuration, process dataflow, and fault handling. Hardware configuration includes physical components and their connectivity. Dataflow includes logical connectivity and executable configuration. Fault handling diagrams show system reactions to problems using hierarchical state machines. The look and feel of the GME is similar to electronic circuit design tools.

11.8.3 R&D Activities

11.8.3.1 Demonstration Project

We are in the process of creating a demonstration for the Supercomputing Conference 2003 [21], which will be held in Phoenix, AZ, in November of 2003. The demonstration will consist of a prototype of the BTeV L1 trigger hardware (approximately 10 DSPs) running a prototype of the L1 trigger software. The software will include fault handling tools (VLAs and ARMORs), and will be able to inject faults into the system and demonstrate the response of the system and operation of the fault handling technology.

This demonstration project requires that the research groups confront the issues needed to integrate their tools into a working system, and is allowing the groups to investigate the strategies and tools that have been developed so far.

11.8.3.2 Summary of R&D Activities

This section provides a brief summary of R&D activities that have been pursued by the RTES Collaboration.

ARMOR Support

A small farm of 6 Linux processors was configured to provide ARMOR support for run control, a database server, and two message services (run messages and error messages), as well as two client application nodes. This work was presented at the 13th Real Time Conference in Montreal in May 2003 [19]. Various failures and automated recovery were demonstrated. Although this system was developed for a "DAQ" application (*i.e.* run control), the exercise has been extremely valuable in coupling the ARMOR concept with trigger algorithm software. Further development, including ARMOR support for the Pixel Trigger and Muon Trigger Supervisor Monitor (PTSM and MTSM) hosts, will continue.

L1 Trigger Fault Detection and Recovery Support

The L1 trigger consists of field programmable gate arrays (FPGAs), and farmlets of DSP processors responsible for the initial analysis of incoming experimental data. A key requirement for any fault tolerance mechanism for L1 is rapid error detection so that no results are lost due to an error. Statistical integrity demands that every bunch crossing be accounted for. While error detection must be fast (at least as fast as the latency), recovery can be slower as long as the offending entity (hardware or software) is isolated from the rest of the system. Towards this end we have studied hardware and software resources provided by a TMS320C6711 DSP Starter Kit (DSK) from Texas Instruments, and the associated operating system (DSP/BIOS) from the same source [22].

The DSK development environment (Code Composer Studio) and the Vanderbilt-developed DSP hardware are both developed to work with Microsoft Windows Operating Systems [23]. One of our accomplishments was the porting of ARMOR processes to Windows 2000. A formal model for reconfigurable ARMOR processes and a distributed, ARMOR-based, software implemented fault tolerance (SIFT) environment for managing user applications were presented in [26].

Furthermore, we have designed a detection/recovery scheme to handle race conditions, which could occur when the incoming data arrival rate exceeds the processing rate of a DSP. This is a somewhat hypothetical fault condition because buffer overflow (due to mismatched data and processing rates) would be handled by the Buffer Manager FPGA, not by the DSP. However, this type of fault was easy to create, and provided a basis for exploring detection and recovery mechanisms. The detected race condition was reported to the host computer to initiate appropriate recovery actions. In order to improve our understanding of application and system behavior we explored the use of semaphores, blocked execution, software interrupts, hardware interrupts, and periodic tasks. Experiments were performed to obtain detailed knowledge of how software interrupts are handled inside DSP/BIOS, as

well as determining limitations for the scheduling of multiple interrupts. Work is underway to understand the effective use of DMA.

While these studies have been specific to the TMS320C6x family of DSPs and to DSP/BIOS in particular, we are developing an understanding of the infrastructure needs of BTeV and RTES code executing on a heterogeneous hierarchy of embedded processors, regardless of the actual processor(s) or kernel(s) used in the implementation of BTeV.

Finally, in an effort to provide fast hardware-level error detection we have explored the feasibility and potential implementation of a hardware framework for providing fault tolerance services. In this framework, error detection and recovery firmware and programmable hardware (registers, on-chip memory, and control logic) constitute a *reliability engine* implemented as an FPGA-based device, or fully integrated with the processor. The application can be instrumented (*e.g.*, using a dedicated pre-compiler) to instruct the processor about the desired level and type of runtime checking. The checking can range from full duplication of the instruction stream, to precise spot-checks of individual instructions and/or results produced by critical code sections. Several prototype timer elements have been conceived. Discussions are ongoing with regard to the implementation of these ideas in the communications support FPGA(s) of the Vanderbilt-developed DSP hardware.

L2/3 Trigger and Supervisor Host Fault Detection and Recovery Support

The L2/3 trigger consists of general-purpose processors (*e.g.*, Pentium, PowerPC), most likely running Linux. This trigger is responsible for a more sophisticated evaluation of the experimental data initially filtered by the L1 trigger.

In addition to the L2/3 trigger processors, there will be general-purpose processors responsible for system control functions. These "host" processors, or processes, include database service, time service, run control, slow control, and various supervisory elements. Some of the host processes or processors are part of the DAQ. Run Control and Slow Control are prime examples. However, to the extent that the hosts run on processors that are members of the L2/3 trigger farm, it is of considerable value to develop ARMOR support for the host applications. In the case of the Pixel Trigger and Muon Trigger Supervisor Monitor (PTSM and MTSM) hosts, ARMOR support is a specific deliverable of RTES.

The relationship between ARMORs and VLAs has been explored using LM_Sensors [18], a Linux software package for accessing low-level health monitoring hardware, commonly implemented in PCs built after 1997. It is likely that the processors of the L2/3 trigger will include such health monitoring hardware. Hence LM_Sensors are a convenient candidate for developing VLA concepts on a general-purpose processor. Using a standard messaging protocol developed by Pittsburgh for the client VLAs, a corresponding ARMOR server element has been developed. Sockets are used to abstract the relationship between the VLAs and the ARMOR. This is important, as it will allow the ARMOR recovery mechanism freedom to restart the VLA-server (ARMOR) on a different node in the event of a failure. Work is in progress to develop the managerial relationship between the ARMOR and VLA (starting, stopping, and naming VLAs), as well as both VLA failure recovery and ARMOR server failure recovery.

VLAs

The RTES collaboration has developed two VLA prototypes running on TI DSPs for L1. The first prototype was built for the TI DSP/BIOS operating system. The second prototype was developed for the hardware built by the Vanderbilt group running a proprietary kernel. A paper titled "Design of Very Lightweight Agents for Reactive Embedded Systems" was presented at the 10th IEEE International Conference on the Engineering of Computer Based Systems (ECBS).

We have also developed a scheduling algorithm, which takes advantage of the decreasing data rate (due to decreasing luminosity in the collider) during a *store*. We have tested the feasibility of this algorithm using a resource kernel on Linux with a simulated physics application and a VLA. A paper titled "A Real-Time Scheduler in a Large-Scale Embedded System" was submitted to EMSOFT.

Ongoing work with VLAs includes the following:

- Identify scenarios in the trigger systems pertinent to the design of VLAs
- Design and evaluate a task-scheduling scheme that takes advantage of the decreasing bunch-crossing rate
- Determine time requirements for the error handling routines of the VLA
- Emulate a DSP on a Linux node such that the emulation includes a running VLA, a simulated trigger algorithm, and a command processor
- Study the performance of VLAs in an environment that includes various resource limitations
- Create a simulator to study the behavior of VLAs under different load conditions and different fault-injection assumptions
- Study the interactions between at least two levels of VLAs, with the frequency of communication between VLAs and ARMORs determined by communication delays

GME

The GME modeling tool is one of three deliverables provided by the RTES Collaboration. There has been considerable progress on the development of GME for BTeV system modeling, system generation, simulation modeling, and the BTeV runtime infrastructure.

The use of GME for system modeling encompasses several aspects that are *captured* by the modeling tool. The aspects that are curently included are the following:

- The hardware infrastructure for the BTeV trigger and DAQ including both processors and interconnections between processors and their attributes, which capture performance and fault probabilities.

- Tasks are captured, along with data paths and data dependencies between tasks. The task attributes specify the function that is performed by a task and its performance attributes. For example, tasks include physics applications, VLA diagnostics, and fault-manager tasks.

- State machines capture the desired fault mitigation strategies for BTeV. For these state machines, the states correspond to failure modes of the system, while transitions between states encompass conditions (such as logic equations based on local fault sensors, regional fault conditions, global faults, or user input) and actions. Actions specify the operations to be performed when the conditions are satisfied.

- An additional aspect has been included to model the composition of ARMOR elements based upon the particular category of an ARMOR, the location of the ARMOR, and the messages that are handled by the ARMOR. This information is used to generate scripts that instantiate and activate ARMORs.

In addition to system modeling, there has been progress on system generation tools that are used in both offline and online environments. To date, the focus has been on the offline (design-time) mode. The system generation tools process the system models and generate several products:

- System configuration for bootstrapping the network.

- System configuration files needed to begin execution. This includes, but is not limited to, creating and starting tasks, establishing connections, and routing data between processors.

- Fault mitigation modules for executing the behavior specified in the mitigation models. The modules consist of synthesized C code that monitors input variables, evaluates transition conditions, and maintains the state of the system. During transitions, the user-specified behavior is executed.

- Link rerouting tables are generated to perform rapid network reconfigurations for processor/link failures. Application-specific redundant signal paths are inserted into the software configuration, and scripts to manage the swapping of data paths are also generated.

The GME modeling tool is also applied to the runtime infrastructure. The development of this infrastructure includes the following:

- Support for link swapping: An API function has been implemented that utilizes the information in the rerouting tables and redundant signal paths to perform a swap in a communication link in the event of a link failure.

- Support for mitigation operations: Additional API functions have been implemented to support fault-mitigation operations. These API functions implement facilities for dispatching a fault message, resetting a faulty communication link, resetting a faulty processor, and relocating a process.

- Support for fault detection and fault mitigation messages: Data structures have been defined for propagating fault messages. These data structures have fields for message types, the severity of a failure, and the location of fault, to name a few.

- Integration with Texas Instruments DSP/BIOS: The existing VU/DSP kernel does not have support for preemption. The core of the kernel executes a cyclic scheduler that iterates over the tasks. This model has its advantages in terms of simplicity and minimal task switching overhead, however it has limitations in meeting time deadlines of periodic intermittent tasks. To overcome this limitation the existing kernel has been extended and integrated with DSP/BIOS. DSP/BIOS offers a low-overhead context switching capability, and the task scheduling model has been augmented to incorporate classic real-time schedulers and real-time tasks.

Simulation modeling includes the creation of a detailed model of the system. The goal of the simulation is to generate accurate predictions of system behavior as well as the timing of the behavior for use in evaluating fault-mitigation designs. We are using the dynamical system modeling and simulation capabilities of Matlab®, Simulink®, Stateflow®, a toolsuite available from Mathworks Inc.to perform the simulation.

The simulation modeling begins with low-level hardware. We are developing low-level Stateflow simulation models of communication hardware protocols, along with the DMA processor that transfers data between memory and the communication link. In the model the network topology is captured by connecting communication ports. The kernel interface to the communication hardware is modeled as a communication channel. Stream structures, which implement communication conduits between software processes, are modeled on top of the channel, with multiple streams sharing a channel. The cyclic task scheduler is also modeled. Each of these models attempts to replicate the behavior of the kernel.

Finally, the application processes and interconnections are modeled with simplified behavior of the communications of each task. The next step in the simulation engine will be to add the fault mitigation simulation. We are also working to fully configure the simulation from the modeling environment, in terms of the hardware network topology, and the flow of data in the system.

11.9 Production Plan

11.9.1 L1 Pixel Trigger

The design, fabrication, and testing of the L1 pixel trigger is divided into three phases. The first phase is a prototype phase that entails a hardware design for every component

Table 11.14: Pixel Trigger Components

Item	Base Quantity	Additional	Total
Pixel/segment preprocessor subsystem			
Pixel preprocessor	960	96	1056
segment preprocessor	448	44	492
Level-1 buffer readout link	480	48	428
optical link receiver	960	96	1056
segment preprocessor interconnection	960	96	1056
switch link card	480	48	528
supervisor and link card	480	48	528
pixel trigger switch			
preprocessor to switch link card	480	48	528
switch logic board	64	7	71
switch card interconnection	8	1	9
switch to farm link card	624	60	684
switch rack interconnection data link	8	1	9
switch interface link card to PTSM	144	15	159
pixel farm			
farm processor card	418	42	460
farm daughter card	2500	250	2750
switch/farm link and buffer	418	42	460
Level-1 buffer readout link	418	42	460
GL1 result link card	418	42	460
farm supervisor and link	418	42	460

specified in the pixel trigger architecture. The second phase is a pilot system, which consists of a single trigger highway but with reduced processing and bandwidth capacity. It will represent approximately 10% to 15% of the full production system. The third phase is the production pixel trigger system. In this section we list all of the components that are part of the production system, and describe our plans for production, testing, and quality assurance.

11.9.1.1 L1 Pixel Trigger Electronic Modules and Interconnects

Electronic modules and interconnects used in the L1 pixel-trigger system will be either be commercial off-the shelf (COTS) modules or in-house designs. The in-house designs will be assembled in-house if the quantities are sufficiently small (typically, less than ten). Otherwise, they will be assembled by outside board assembly companies. Fermilab is responsible for the L1 pixel trigger hardware and a large quantity of the muon trigger hardware and will

pre-qualify board fabrication and assembly companies based on vendor interviews, site visits and/or previous experience with board and module orders.

Quality Control Procedures

All module designs will have a specifications document that includes, at a minimum, the operational requirements, circuitry description, schematics, board layout, assembly instructions and bill-of-materials. The document should be completed and reviewed before the design is complete. The specifications document can be modified at each stage through the prototype and pilot testing phases as operational information is gathered. The production specifications should not change after the production design is reviewed.

Each module design will be reviewed by BTeV engineers and physicists for safety, manufacturability and satisfaction of the requirements before it goes to production. All module designs will be required to meet applicable BTeV, Fermilab, and OSHA safety standards for electrical systems and equipment. Design reviews for pilot and production stages should include a review of the experience with the previous corresponding prototype and pilot modules.

All module fabrication contracts will include a clause specifying that the printed circuit board manufacturer will meet IPC-A-600F standards for printed circuit board fabrication. All module assembly contracts will include a clause specifying that the assembly company will meet IPC-A-610C standards for acceptability of electronic assemblies.

Incoming modules will be inspected by the responsible institution to check for compliance with fabrication and assembly standards. Prototype and pre-production modules will be 100% inspected. Pilot and production quantities may be 100% inspected or sampled at the 20% level when made possible by a history of satisfaction with previous orders from the same manufacturing company in order to meet production schedules.

Testing Procedures

Module testing will consist of a series of increasingly complicated tests. Incoming modules that pass quality control inspection will be individually power tested by having the nominal operating voltages applied and monitoring the current consumption. This will confirm module power distribution and component placement. Next will be the individual module functional test confirming the stand-alone operation of the module. There may be limitations to which functions can be tested if the module depends on a partner module. Power testing will merge with the individual module functional testing protocol as soon as the module design and assembly process are proven consistent. Interconnections with direct neighbor modules will be added and those functions tested. This process will gradually involve the support infrastructure commonly called a test stand. Also at this point, the interconnecting cables will be included and tested. An increasing scope of reasonable combinations of modules will be tested until, finally, a partial system containing at least one complete data path will be assembled to test system functionality and the inter-dependence of at least one instance of all the modules in the system. This is commonly called a vertical slice of the

system. All production modules will go through the individual module functional tests and vertical slice system testing. Additional testing steps beyond the functionality needed for the experiment will be used to confirm the testing process as well as to fully test the design and operation of the modules and sub-systems. If production continues after the vertical slice system tests become operational, production testing could be done in the vertical slice environment by including all of the preceding tests in that environment.

During production module testing, a database will be established and maintained in order to collect data that can be used to establish mean time between failure (MTBF) and men time to repair (MTTR) statistics for the modules. This database can be an early indicator of reliability problems, component manufacturing problems and/or design weaknesses. It will also help with predicting the number of spare components needed for the life of the experiment.

Each module or sub-system will have a user manual developed during design and testing to provide a reference for experiment operators and repair technicians during the life of the experiment. The user manual can be a separate document or included in the specifications document. It should be review for completeness and usefulness before system commissioning is begun.

11.9.2 L1 Muon Trigger

The L1 muon trigger will operate independently of the pixel trigger in that it generates trigger results based on the muon detector. The design of the muon trigger hardware is based on the design of the pixel trigger, and in many cases will employ identical hardware, but in smaller quantities. One specific example is the muon trigger farm; it will use the same processors, same power supplies, same packaging, etc., but approximately one tenth as many as the pixel trigger farm. In this section we list all of the components that are part of the production muon trigger system.

The plans for production, testing, and quality assurance for the muon trigger are the same as those described for the L1 pixel trigger, except that there will only be pre-production and production phases; there will not be a "pilot" phase. For all of the hardware of the muon trigger that is identical to hardware in the pixel trigger, the pixel trigger project at Fermilab will be considered as the "vendor" for this hardware. As such, all production, testing, and quality assurance will be conducted exactly as specified for the L1 pixel trigger, and will likely be manifest as simply an increase in quantity of "pixel" trigger modules. For the remaining hardware that is specific to the muon trigger, it is intended to follow the L1 pixel trigger plans, employing the same manufacturers and Fermi-qualified contractors, and executing the same procedures and protocols. Due to the proximity of Fermilab to the University of Illinois, it would not be a hardship to conduct much of this work at Fermilab, space permitting, so as to further leverage the similarities between the pixel and muon triggers.

Table 11.15: Muon Trigger Components

Item	Base Quantity	Additional	Total
Muon preprocessor subsystem			
Muon preprocessor	48	5	53
optical link receiver	96	10	106
Level-1 buffer readout link	48	5	53
Muon preprocessor interconnection	3	1	4
supervisor and link card	48	5	53
switch link card	48	5	53
muon trigger switch†	1		1
muon farm ‡	1		1
muon trigger supervisor and monitor§	1		1

†The muon trigger switch will be a scaled down version of the pixel trigger switch.
‡The muon trigger farm will be a scaled down version of the pixel trigger farm, using 1/10 as many processors.
§The muon trigger supervisor and monitor will be a scaled down version of the PTSM, servicing 1/10 as many processors.

Table 11.16: Global Level 1 Trigger Components

Item	Base Quantity	Additional	Total
Global Level 1			
GL1 farm processor card	48	4	52
GL1 farm daughter card	192	20	212
pixel, muon, front-end electronics interface	48	5	53
Level-1 buffer link for GL1 processed data	48	4	52
GL1 result link card	48	5	53
GL1 supervisor and monitor link	48	4	52

11.9.3 Global Level 1 Trigger

Global Level 1 (GL1) receives results from trigger calculations that are performed by the L1 pixel and L1 muon triggers. The results are processed by GL1 hardware in order to produce an L1 trigger decision for each bunch crossing. Additional hardware inputs are provided for possible future triggers and/or for diagnostic and calibration signals, however, there are limitations on how these inputs can be used. GL1 hardware consists of processing nodes that are based on the design of the processing nodes used in the L1 pixel trigger. Here we list components that are needed for GL1.

Table 11.17: Major L2/3 Hardware Components

Item	Quantity
Farm worker CPUs	1536
L2/3 host PCs	72
L2/3 storage	1536

The plans for production, testing, and quality assurance for the GL1 are the same as those described for the L1 pixel trigger.

11.9.4 L2/3 Hardware

The L2/3 hardware consists of commodity PCs. These are operated as a collection of PC farms in the trigger system. The design, procurement and testing of the L2/3 hardware will proceed in three stages. In the first stage during FY04, a prototype PC farm will be constructed from already existing hardware from old Computing Division PC farm nodes. The second stage consists of building a pilot farm in FY07 that has all the components configured to act as a trigger system and will represent about 5% of the full production system, this represents one complete rack out of the 24 racks of L2/3 Farm Workers. In the third stage the production farm will be constructed and then integrated into the complete BTeV trigger system. In this production stage, 20% of the L2/3 Farm will be purchased and built in FY08, so that 2 complete Highways will be ready by the end of FY08. As funds become available in FY09 the hardware for the other 6 Highways will be procured and built and integrated in a modular manner. The procurement and quality assurance for the L2/3 hardware is described in the rest of this section.

11.9.4.1 Components of the L2/3 Trigger Hardware

The major cost components for the L2/3 hardware are listed in Table 11.17, and are expected to be commodity items. The Farm worker nodes, host nodes, and storage units will all be subjected to the same procurement and quality assurance testing procedures. This consists of an evaluation stage to evaluate and qualify particular computer node products and vendors, followed by quality assurance testing after the nodes have been received.

Quality Control and Testing Procedures

The major purchases for the L2/3 Trigger PC farm will go through a process similar to the Computing Division's purchase of the fixed-target, CDF, and D0 PC farms. The products and vendors will go through a thorough evaluation stage after a detailed specifications and requirements document for the PC worker nodes has been produced and sent to vendors. This will enable vendors to perform the evaluation tests themselves. Sample nodes from prospective vendors will then be subjected to evaluation tests. The actual set of tests will

be developed as we approach the time of purchase, when more is finalized about the L2/3 trigger. We will also incorporate what we learn from the Computing Division's OSS PC farm evaluation and quality assurance testing. The current set of evaluation tests for the Computing Division Run 2 farms can be found at: `http://www-oss.fnal.gov/scs/farms/acquisitions.html` as well as a document summarizing the evaluation process.

Once the L2/3 farm nodes are received, they will be put through a set of tests similar to the evaluation tests, except that our tests will be run for a longer period of time. The Computing Division's OSS group found that some problems only manifest themselves after many months of continuous operation. The Computing Division's OSS quality assurance testing is similar to the evaluation testing described in the documents above except that the nodes are required to be up and tested for 30 continuous days instead of 2 weeks. The tests are still evolving as more is learned about what long-term problems can arise. The tests consist of benchmarking; stress and long-term tests of memory, CPU, disks and networking; tests of other devices like floppy disk and CDROM drives; electrical and thermal characteristics monitoring; and downtime monitoring. Moreover, vendors will be evaluated on their competence in the Linux operating system and in downtime incidents and other services provided during both the evaluation and quality assurance testing stages. We will continue to monitor the progress of the Computing Division's evaluation and quality assurance testing and start to participate in our own testing for a small pilot system. We will also develop the L2/3 system to be more reliable. For example, it has been found that most of the problems occur with hard disks, and we are investigating the possibility of using solid-state disks for system critical data.

11.9.5 L2/3 Software

The L2/3 trigger software consists of reconstruction software that uses L1 trigger output and raw data. The L2 trigger will detect the presence of secondary vertices, while L3 will reconstruct decay topologies and thereby the heavy-quark content (such as charm or beauty as distinguished from light-quark background) and the type of heavy-quark decay. Based on a set of predefined trigger tables and the reconstruction results the trigger will accept or reject a given bunch crossing. For L2, only the pixel data and possibly some of the data from forward-tracking stations will be used. For L3, data from all BTeV detector systems will be used to refine the trigger selection criteria. The reconstruction algorithms at L2 include pattern recognition and track fitting. For L3, we include RICH particle identification, muon identification, and electromagnetic-calorimeter shower reconstruction. The triggers will be configurable in that some predefined list of parameters will be downloaded onto processing nodes at the beginning of each run. For L2/3 software we will follow standard procedures to create and manage production versions of the software. The reconstruction software is not unlike the software that is typically written for offline computation, and as such it will be managed like any other BTeV offline package. We will therefore follow rules and conventions established for offline software. At this time we do not envision any problems with this approach. For instance, both the BTeV offline and online code is developed using

the same industry-standard tool to implement version control (CVS repositories). Similarly, the bug-tracking systems that we intend to use should be the same between offline and online code development. We also do not foresee and boundaries between developers from different institutions as the software is designed, implemented, and tested. Therefore, everybody working on L2/3 software is likely to be involved in developing code, testing and quality assurance. Development of code for L2 and L3 should be identical. This is because the programming language is most likely the same, and the same software infrastructure will probably be used for both L2 and L3. Therefore our plans for production, testing, and quality assurance are the same for both.

11.9.5.1 Prototyping and Preliminary Production Versions

The methodology that will be used for prototyping and preliminary production versions of software is the same as that used for any other software development for BTeV. The code must be maintainable by including documentation and by using version control during the development of the software. The code must be able to process Monte Carlo generated events, as well as real data coming from the experiment. The reconstruction programs will be able to produce their own diagnostic output in the form of histograms or n-tuples.

11.9.5.2 Production Versions of Code for the L2/3 Farm

Production versions of code must be tested and verified offline, both on real data and Monte Carlo generated data. Executables of the code must be built using a standard BTeV scripting procedure, ensuring that we can rebuild executables at a later stage in the experiment. This implies that executables cannot be built using code that comes from someone's personal computer or private directory. Each L2/3 algorithm will be documented in a manner that describes the algorithm and its predicted performance on a given set of L1 data. This implies that prior to execution on the L2/3 farm, one must have a predefined figure of merit against which one can benchmark the performance of new or different versions of algorithms. When a new version of an algorithm needs to be tested (prior to implementation on the L2/3 farm), the new algorithm will be executed on a small portion of the data acquisition system (DAQ). This is also referred to as a DAQ partition. If the algorithm is successful on a small scale, then the algorithm can be tested on a larger number of processing nodes (a larger partition). When this part of the test has been successfully completed, the algorithm can be deployed on all L2/3 processing nodes.

11.9.5.3 Maintenance, Verification, and Control of L2/3 Configuration

Configuration files and trigger tables are critical items in the trigger system, since they determine the sharing of processing and network resources between different needs for the system. As such, they will be critically reviewed, circulated, discussed within the collaboration, and finally approved by BTeV management.

Bibliography

[1] For example, HERA-B and LHC-b; see talks by E. Gerndt and O. Schneider, Beauty '99 Conference, Bled, Slovenia.

[2] E.E. Gottschalk, "BTeV Detached Vertex Trigger", *Nucl. Instrum. Meth.* **A473** (2001) 167.

[3] BTeV Proposal, A. Kulyavstev *et al*, `http://www-btev.fnal.gov/DocDB/0000/000066/002/index.html`.

[4] M. Wang, "BTeV Level 1 Vertex Trigger Algorithm", BTeV-doc-1179.

[5] `http://www.altera.com/`

[6] `http://www.mathworks.com/`

[7] M.H.L.S. Wang for the BTeV Collaboration, "The BTeV Trigger Architecture", FERMILAB-CONF-03-226-E, to be published in *Nucl. Instrum. Meth. A*.

[8] S. Cittolin *et al.*, in *Workshop on Recent Developments in High Energy Physics,* NCSR Demokritos, 9–11 April, 1998, p. 87.

[9] Associative memories (data-storage structures designed for rapid parallel search operations) are finding increasing use in industry; see for example the "content-addressable" memories made by MOSAID Technologies, `http://www.mosaid.com/networking/index.html`.

[10] See also D. Husby, "Systolic Associative Arrays for Track Finding," `http://www-ese.fnal.gov/eseproj/trigger/asmem/asmem.pdf`, and "Associative Memory for Track Finding," `http://www-ese.fnal.gov/eseproj/trigger/asmem/`.

[11] See for example M. H. Schub *et al.*, Nucl. Instr. Meth. **A376**, 49 (1996); W. Sippach, G. Benenson, and B. Knapp, IEEE Trans. Nucl. Sci. **NS-27**, 578 (1980); R. G. Cooper, IEEE Trans. Comp. **C-26**, 1123 (1977).

[12] D. Husby *et al.*, *Nucl. Instrum. Meth.* **A383** (1996) 193.

[13] W. Selove, in *Proceedings of the Workshop on B Physics at Hadron Accelerators*, P. McBride and C. S. Mishra, *eds.*, Fermilab-CONF-93/267 (1993), p. 617.

[14] D. M. Kaplan and V. Papavassiliou, in **CP Violation**, X.-H. Guo, M. Sevior, and A. W.Thomas, eds. (World Scientific, Singapore, 2000), p. 116; D. M. Kaplan, in *Proc. Symposium on Flavor Changing Neutral Currents: Present and Future Studies*, Santa Monica, CA, 19–21 Feb 1997, D. B. Cline, ed. (World Scientific, Singapore, 1997), p. 81.

[15] `http://www-btev.fnal.gov/`

[16] T. Elrad, R. Filman, and A. Bader, "Aspect-Oriented Programming," *Communications of the ACM*, vol. 44, no. 10, pp. 29-32, Oct 2001.

[17] "GME 2000, The Generic Modeling Environment," `http://www.isis.vanderbilt.edu/Projects/gme/`

[18] `http://www2.lm-sensors.nu/~lm78/docs.html`

[19] L. Picolli, "Evaluation of RTES components in a large scale DAQ," 13th IEEE-NPSS Real Time Conference, May 2003, Montral, Canada. `http://www-btev.fnal.gov/cgi-bin/DocDB/ShowDocument?docid=1792`

[20] `http://www-btev.fnal.gov/public/hep/detector/rtes/`

[21] `http://www.sc-conference.org/sc2003/`

[22] `http://dspvillage.ti.com/docs/dspvillagehome.jhtml`

[23] `http://www.microsoft.com/homepage/default.htm`

[24] G. Booch, J. Rumbaugh, and I. Jacobson, *The Unified Modeling Language User Guide*, Reading, Massachusetts, Addison Wesley, (1998).

[25] S. Tamhankar, J. Oh, and D. Mosse, "Design of Very Lightweight Agents for Reactive Embedded Systems," S. Tamhankar, J. Oh, Syracuse University, and D. Mosse, University of Pittsburgh, 10th IEEE *International Conference on the Engineering of Computer Based Systems*, April 2003, Huntsville, AL. `http://www-btev.fnal.gov/cgi-bin/DocDB/ShowDocument?docid=1864`

[26] K. Whisnant, Z. Kalbarczyk, and R. Iyer, "A Foundation for Adaptive Fault Tolerance in Software," in Proc. of Conference on Engineering of Computer Based Systems (ECBS) 2003.

[27] `http://www.c2.com/cgi/wiki`