Hardware Trigger Simulation at CDF

S. Rolli[1]*, J. D. Lewis*[2] *H. Ray*[3]*, J. Nachtman*[4]*,M. Worcester*[4]*,*

[1] Tufts University, Medford, MA, USA
[2] Fermilab, Batavia, IL ,USA
[3] University of Michigan, Ann Arbor,MI ,USA
[4] UCLA, Los Angeles, CA, USA

Abstract

The CDF experiment at Fermilab has a three level trigger system, two of which (Level-1/Level-2) are hardware trigger systems. In this paper some of the software simulation/emulation packages for Level-1/Level-2 are described, in particular their implementation in the CDF RUN II offline software framework and their use as the engine for the online consumer monitor.

Keywords: trigger

1 Introduction

The trigger plays an important role in the CDF hadron collider experiment because the collision rate (8 MHz) is much higher than the rate at which data can be stored on tape (50-75 Hz). The CDF trigger system has a three levels architecture with each level providing a rate reduction sufficient to allow for processing in the next level with minimum deadtime. Level-1 uses custom designed hardware to find physics objects based on a subset of the detector information and makes a decision based on a simple counting of these objects (e.g. one 12 GeV electron or two 1.5 GeV muons). The Level-2 trigger uses custom hardware to do limited event reconstruction which can be processed in programmable processors. The Level-3 trigger is performed after full detector read-out by offline reconstruction of the raw data by a farm of processors.

The development of a comprehensive and accurate trigger simulation is crucial for insuring the success of CDF in run II. Indeed the primary use of the trigger simulation is for :

- trigger and datasets planning;
- physics analysis;
- online monitoring.

Moreover the simulation of the trigger elements must be sufficiently flexible to allow for the specification of cuts or pattern lookups as specified in a trigger table. Toward this end, the trigger simulation can be divided into two distinct stages: the first one will concern the performing of the low-level trigger functions and creation of the trigger elements, e.g. jets, XFT tracks, muon primitives. These elements can be compared directly to results from the trigger hardware found in the raw data. The functionality of this phase is broken down into units similar to the hardware. In the second stage one will implement of the trigger table: the decision phase associates the results of the various parts of the Level 1 systems with the Level 1 triggers and executes algorithms for the various functions performed by the Level 2 processors to create Level 2 trigger decisions. In this paper we will be primarily concerned with what happens in stage one.

Because CDF will have an all-digital trigger system, it should be possible to reproduce identically the results of the trigger hardware in the simulation and to perform bit-by-bit comparisons. In order to accomplish this goal, the scope of each portion of the trigger simulation and its input and output data must be completely specified. The philosophy we have adopted is a modular approach.

The simulation of each subsystem is a separate AC++ module in the CDF AC++ Framework[1], and the code for the trigger simulation resides in a CVS repository. This approach allows for:

- fast implementation of existing code for various subsystems;
- independent work by software developers and maintainers;
- independence of the data generation stage from the decision stage.

Moreover, each portion of the simulation should be able to run as a separate module in order to maximize flexibility.

2 Data Flow and code design

The flow of detector data through the trigger simulation and the precedence for simulation modules will essentially replicate the data flow in the trigger system itself. A schematic diagram of the trigger system is represented in Figure 1. The Level-1 hardware consists of three parallel synchronous processing streams, which feed the inputs of the single Global L1 decision unit. One stream finds calorimeter-based objects (L1CAL), another finds muons (MUON to PRIM-L1 $\rightarrow$ MUON) and the third one finds tracks in the central tracking chamber (XFT-XTRP-L1 TRACK). Since the muons and electrons triggers require the presence of a track pointing at the corresponding outer detector element, the tracks must be sent to the calorimeter and muon streams as well as the track only system.

The Level-2 trigger consists of several asynchronous subsystems which provide input data to programmable Level 2 processors in the Global Level-2 crate, which evaluate if any of the Level-2 triggers are satisfied. The system is composed by a cluster finder (L2CAL), a central Shower Maximum trigger (XCES) and a Silicon Vertex Tracker(SVT)[2]. In essence each box of the figure can be thought of as an AC++ module (or set of modules) and each link as a data structure (data banks and, optionally, data objects[3]).

All detector data input and output for the simulation of each subsystem must be via banks. At the lowest level, the input banks are the detector raw-data banks. The output of the simulation of a trigger subsystem is a bank identical to the raw-data bank for that system. In the online readout, some set of trigger banks that record low-level functionality are not used directly in the trigger decisions and will not be read for all Level 2 triggers, but only when a diagnostic readout list is specified. The diagnostic readout list will be included for some specified fraction of events and those events may be used for special online monitoring of the performances of the trigger. Data and simulated banks are distinguished by a description string.

2.1 The calorimeter code

At Level 1 the calorimeter triggers are divided into 2 types: object triggers (electrons, photons and jets) and global triggers (ET and missing energy). The object triggers are formed by applying thresholds to individual towers while thresholds for the global triggers are applied after summing energies from all towers. The digitized calorimeter data are summed into trigger tower energies and weighted by sinθ on the front-end cards to produce E_T. Data are processed in the 6 L1CAL crates with final global sums and trigger summaries made in the Global L1 crate. The data are then forwarded to the L2 cluster finder after a L2 accept.

At Level 2 the clustering algorithms loops over all of the trigger towers, first comparing the transverse energies with different thresholds for the seeds. If a seed is found the information on η, ϕ, and energy is stored in an array of information for a cluster. The seed is masked off, and four surrounding towers are flagged for the shoulder threshold passes. This continues until there are no more towers to be included in a given cluster. For each tower a word is created, containing the

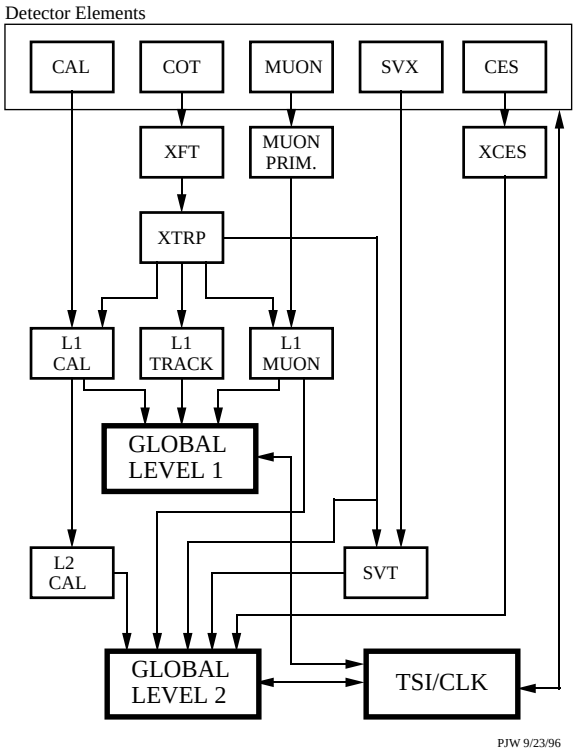

Figure 1: The data flow in the CDF trigger simulation

result of the clustering algorithm. (each tower can be *seed* or *cluster member* for a given pass). A schematic representation of the data flow in the Calorimeter code is shown in Figure 2.

The following classes have been designed and coded (package CalTrigger):

CalTriggerDataMaker : public AppModule is the driver module.

CalTriggerData : public StorableObject is the main object of the calorimeter simulation, it contains the array of trigger towers and quantities resulting from the L1 simulation and L2 clustering and isolation algorithms.

Class TriggerTower contains generic trigger tower information.

Class TriggerTowerTypeX : public TriggerTower contains specific trigger tower information (PMT sums).

Class TriggerCellKey maps the transformation of the physical towers into trigger towers.

In the class CalTriggerDataMaker there are several methods implementing the various steps of the trigger. The most important are:

void FillTower() where TriggerTower::fillTriggerTower() is called and the actual implementation of this method is found in the TriggerTowerType class.

fillTriggerTower takes the individual photomultipliers information for each detector and sums them based on the *type* of the tower.

void CorrectTransEnergy() calls the correctTransEnergy method of class TriggerTower, which uses an array of $\sin\theta$ values for each trigger tower index. The energy in each trigger tower

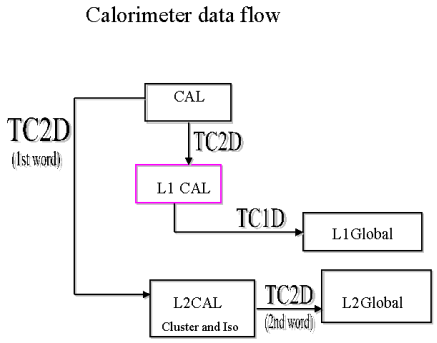

Figure 2: Calorimeter data flow

is multiplied by this sinθ, and then integerized. The resulting energies are stored in the data members of TriggerTower.

TriggerTower* towerMake() creates the elements of the 24x24 trigger tower array, after selecting the correct TriggerTowerType.

bool DoClustering() loops over all of the trigger towers, first comparing the transverse energies with thresholds for seeds, based on the physics looked at. The thresholds will be read from a database. If a seed is found the information on η, ϕ, and the energy is stored. The seed is masked off, and four surrounding towers are flagged for shoulder threshold passes. This continues until there are no more towers to be included in a given cluster. For each tower a *result* word is created, to be stored in the TC2D_StorableBank.

bool DoIsolationSums() performs isolation sums on the clusters.

Several L1 simulation methods .

In Figure 3 a class diagram for the Calorimeter code is shown.

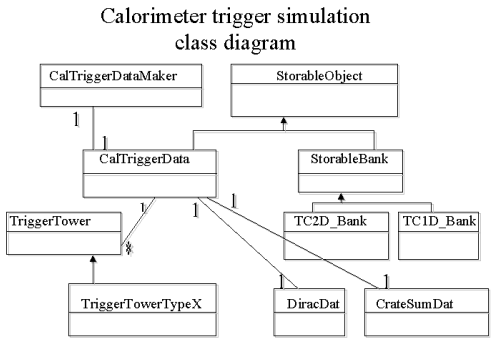

Figure 3: Calorimeter code class diagram

2.2 The trigger objects

The package TriggerObjects contains the output of the various simulation modules (communication between modules, in the AC++ Framework[1] happens through StorableBanks and/or StorableObjects[3]). For the calorimeter simulation, TC1D_StorableBank and TC2D_StorableBank have been implemented (see Fig 2) and we invite the interested reader to look at Ref. [3] for further explanations about the CDF event data model. Here we only notice that we have preferred to keep CalTriggerData (which is a StorableObject) in the CalTrigger package.

3 Online Trigger Monitoring

Monitoring of the trigger is accomplished in the context of the Online Monitoring framework described in Ref. [4], which allows events with a normal or extended readout list to be retrieved from the consumer-server in real time so that any problems in the trigger can be quickly discovered and repaired. The framework provides a template for input/output of data, database access, and classes for the monitors, and allows remote viewing of histograms and database archiving of run-by-run monitoring for later analysis. An important feature of the trigger monitoring program is its ability to run a simulation of the hardware trigger and compare with the results of the trigger. This has been implemented for the Level 1 calorimeter trigger simulation discussed in Section 2.1.

The trigger monitoring executable, TrigMon, is made up of several modules written in the AC++ framework. The input module can accept data from several sources: a file of real or simulated data, randomly-generated input or the consumer-server. Within TrigMon, the CalTriggerDataMaker module operates on the input data, producing TC1D (Trigger Calorimeter Level 1 Data) StorableBanks which contain the results of the CalTrigger simulation and a summary of the calorimeter trigger data. The TC1D_Monitor module then operates on the TC1D banks. The operation of the detector is monitored by comparing the various trigger quantities by channels or events. A bitwise comparison is done between the TC1D banks produced by CalTrigger from the calorimeter data, and the TC1D banks produced by the hardware trigger, allowing a check of the correct trigger decision based on simulation and the result from the hardware.

The output of the TrigMon program is histograms of trigger quantities and a comparison with the trigger simulation, and warning messages announcing possible problems. The histograms are output to shared memory for display. Access to the shared memory is accomplished through a ROOT-based program which allows a connection from a local or remote machine via a web browser [4]. In addition, files are written to disk at regular intervals, and summaries can be written to a database. An example histogram is shown in Figure 4, the distribution in $\eta - \phi$ space of the trigger towers exceeding an energy threshold of 5 GeV, using the calorimeter trigger data from the CalTrigger simulation.

4 Conclusions

In this paper we have reported on the ongoing effort to design, code and implement the hardware simulation software at CDF for the upcoming run II. Examples have been shown relative to the trigger calorimeter simulation, as well as the use of the simulation as the engine for the online consumer monitor process (TRIGMON).

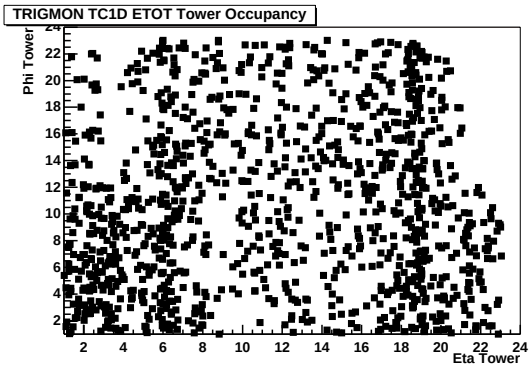

Figure 4: Calorimeter trigger towers in $\eta - \phi$ space, based on trigger simulation.

References

1 E. Sexton-Kennedy ”The Physical Design of the CDF Simulation and Reconstruction Software”, CHEP 2000, Padova, in this proceeding.

2 S. Donati, “The CDF Silicon Vertex Tracker”,CHEP 2000, Padova, in this proceeding.

3 R. Kennedy et al. “The CDF Run II Event Data Model”, CHEP 2000, Padova, in this proceeding.

4 K. Maeshima, *et al*,“Online Monitoring and Module Maintenance for CDF in the Upcoming Fermilab Tevatron Run II”, CHEP 2000 2000, Padova, in this proceeding.