D0 Note 3767
September 21, 2000

The D0 Run II Trigger Database

Elizabeth Gallas, Barb Angelos, Amber Boehnlein,
Carmem Silva, Julie Trumbo
Fermi National Accelerator Laboratory

Joseph Boon Kuah
University of Michigan

The D0 Run II trigger, and the trigger simulator (intended to emulate the trigger) require detailed configuration before operation. Trigger related configuration instructions are stored in the Run II Trigger Database. Details of this database are described here.

Contents

1 Introduction

This note describes the trigger database which stores all trigger specific instructions and settings for all trigger configurations used in the D0 detector through Run II. In addition, the database may also store similar or identical trigger configuration information for runs of the trigger simulator.

There is a huge amount of information that must be recorded for every run. The intention of the Trigger Database is to store specifically the trigger configuration information. Other information must be stored elsewhere.

Both the online and offline versions of the Trigger Database are tables that are actually just a subset of all the tables in the D0 Online and Offline Database, respectively. When we say in this document "the Trigger Database", in all cases, we really mean "the Trigger Database tables within the Database that stores all tables including trigger, calibration, geometry and alignment, hardware configuration, luminosity, monitoring, ... database tables." There are 2 such databases, one for D0 online and the other for D0 offline.

Currently, development of the Trigger Database is occurring within the offline D0 database environment. It includes information that will facilitate user friendliness of web based user interfaces to the trigger database information. Eventually, an online version will be developed which may be a subset of the full offline database (containing only the information neccessary for online systems).

Goals of the Trigger Database Must store all information needed to build a trigger list and the tools/filter definition files associated with it. Generate a given version of a trigger list on demand. Perform validation of all data inserted into the tables and output to the trigger list. Provide a user-friendly interface for doing the above. Serve as a testbed for the ATLAS trigger database.

Goals (continued) Physicists will be able to create their own triggers and trigger lists for simulation studies. These will be kept in the TDB itself so interesting triggers can be shared. These triggers will be tagged differently from the "official" triggers used by the Trigger Meisters.

Structure of the Trigger DB Status Members of the DZero Trigger Database group are: Level 1,2: Elizabeth Gallas, Joseph Kuah Level 3 : Barbara Angelos, Amber Boehlein, Carmen Silva Level 1,2 Tables are close to their final form. (E. Gallas) Populating the database with triggers defined by the DZero Trigger Panel. Begun to design the interface to COOR (Run Control SW) and the GUIs for users to define triggers and trigger lists.

Responsibilities: Develop and maintain a MISWEB based web page to display records in the database. MISWEB is a PERL script developed by Fermilab Computing to query an ORACLE database. Assume responsibility for developing a JAVA GUI to display and modify trigger lists. Prototype has been developed for Level 3 by C. Silva, Fermilab contract staff who is leaving in Oct. 2000 Need to focus on procedures to validate entries to the Trigger DB and extend the GUI to Level 1,2. Currently a standalone program that only runs on PCs. Must run on all machines and possibly via the Web

2 Trigger System Summary

Here are some details of the D0 Trigger System relevant to the Trigger Database.

L1, L2 and L3 - There are 3 levels of triggering used to select the events that will be recorded for offline processing.

- The L1 (Level 1) trigger, is principly a hardware based trigger which includes many FPGAs (Field Programmable Gate Arrays) to make the L1 decision. L1 decisions are based on information coming from LUM, CFT/CPS, FPS, CAL, and MUON detectors. Psuedo L1 terms are logical and/or combinations of other L1 terms that act effectively like other L1 detector terms. L1 detectors report their findings to the L1 framework (L1FW), which supports 128 unique L1 trigger bits, each of which is an AND combination of any number of L1 AND/OR TERMS.
- The L2 (Level 2) trigger includes preprocessors for the CAL, CFT, MUON, CPS and FPS detectors followed by a global preprocessor which is capable of combining L1 trigger objects from different detectors. There is a one-to-one mapping between L1 and L2 bits.
- The L3 (Level 3) trigger

blah blah - blah

3 What is stored

The Trigger Database stores all Run 2 trigger lists. Specific trigger lists are used by

- ONLINE Detector (via COOR)
- Trigger Simulator (via COORSim)
- WWW Display

There will be many VIEWS of the data in the trigger database or in a specific trigger list. These VIEWS are described in Section **??**. This section gives an overview of what is stored in the trigger database and defines some of the naming conventions used throughout this document.

FPGA versions - L1

AND/OR terms - L1

Trigger Level Bit Names -

Trigger Bit Names -

Reference Sets -

Trigger Lists -

Configuration scripts -

Configuration Dependencies -

Property constraints - L2 constraints on parameter values.

TM Terms -

TM Lists -

Prescales -

Exposure Groups - and their associated list of crates.

4 What is not stored

Here are some things that are NOT stored in the Trigger Database tables (but may be stored in other tables in the Online or Offline Database.

Run numbers - The same trigger configuration may be used to record data for many runs. The Run Summary Database is expected to store information for each run (specified by Run Number) including an identifier that will uniquely specify the trigger database configuration used to record that run.

The Trigger Database stores NO output information from any run.

Readout Crates -

Data Streams -

5 Versioning

Correct entry of any versioning information in the Trigger Database will be a part of the Trigger Certification Process that will need to be executed each time any change in the trigger occurs. This section outlines some of the difficult issues that need to be resolved in order to understand the changes in the data that will occur over time due to changes in the detector trigger and the trigger list over the course of Run 2.

This document currently focusses on L1, but it is clear that L2 and L3 will have similar issues. Decisions made along these lines will have a direct impact on the Trigger Simulation project.

Other L1-specific VIEWS of the Trigger Database include

* Lists of NEOTERMS, which are the terms stored in the Trigger Managers which are used by all L1 subsystems except Calorimeter * List all L1 AND/OR terms (Valid ONLINE and/or simulation terms)

User interfaces are being written to store and retrieve this information (Elizabeth Gallas, Joseph Boon Kuah and Carmem Silva)

Changes in the hardware/firmware/software of the detector trigger need to be recorded in the trigger database if those changes are expected to have any effect on the triggering decision. Such changes generally are reflected in an incremental change in version number associated with that component or set of components.

The reasons for such changes justifying a version increment may have the following origins:

* physics - such as replacement of an unused or spare term with a new term in a Trigger Manager (TM) * Hardware changes (at what level?) These are not needed in trigger DB if they are these included in the hardware database or the Run Summary DB * Firmware or Software change (code stored in D0CVS repository ?) o Algorithm improvements o Bug fixes

Are there other related components of the trigger system that need to be included ? Are there L1 framework or other trigger element versions to store as well - anything that might affect trigger decision outcome that is not stored elsewhere ?

There will be versioning of components at all levels of the trigger. There is some disparity between the trigger levels from this point of view. For example, at L3, the algorithm that is run online is exactly or nearly exactly the code that is 'versioned' in CVS and used in the trigger simulator. Therefore, at L3, the version of the L3 algorithms used in the trigger can be some form of the corresponding version in CVS. Knowing what code actually ran still can create difficulty: In this case, the simulation of data across many runs which use different versions of one or more algorithms may require the generation of multiple executables (and releases?) in order to study performance of that algorithm for the changing conditions. Is there anything we could do now to reduce this necessity? One question that arose during the meeting: will L3 algorithms be backward compatible (to what extent can we use L3 algorithms compatible with previous releases in the current release?).

On the other hand, for example at L1, there may be a varying level of correspondence between what can be stored in CVS, what is used by the detector trigger component and the corresponding implementation in the trigger simulator. In addition to the multiple executable problem mentioned above, the simulation of components may improve over time (bug fixes and improvements in the accuracy of simulation). These improvements in the trigger simulation must be separated from actual changes in the online trigger.

L2 must be somewhere in between the L1 and L3 examples.

Currently in the trigger database, versioning at L1 follows the tentative plan established for the muon system. Muon, like all other L1 subsystems except the Calorimeter, uses a Trigger Manager (TM) Actually, MUON uses 2 TMs which allow a total of 256 specific L1 terms to be stored. A version number and a comment are associated with all the NEOTERMs for each subdetector. If there is a change in one term, then the version number is incremented for all terms for that subdetector's NEOTERMS. This system could be expanded to specify which TM terms (NEOTERMs) were specifically affected by the particular change that prompted the increment in the NEOTERM version. The advantage to this Snapshot Model is that the history of terms in the TM could be viewed as a function of time. One alternative is storing a version for every term, but this could easily

turn into a bookkeeping nightmare.

We trust that your trigger code/equations/matrices/constants are being entered into some code repository such as CVS (or will be). My (limited) understanding of CVS is (please correct me): By tagging a CVS package, the current version of every routine in that package is recorded with that tag identifier. It is then possible to recollect that set of routines corresponding to that CVS tag number at some later date. One question is: Are CVS tag numbers and a written comment sufficient to indicate your trigger version changes ? If CVS is used, the relevant CVS package MUST

* contain only the code you use online and * be organized in a way such that others can understand how it is distributed to your online components (include documentation that indicates this). * trigger simulation code MUST be stored in a separate package to completely decouple these CVS versions.

If insufficient, what more would you like to store? From the trigger database perspective, it is important to record sufficient information. But the organization and quantity of that information must be considered carefully to reduce confusion about what is being entered (source of database corruption) and reduce volume of information entered which also is a source of transcription error and also could potentially hold up data taking. It is also desirable to have some uniformity from one subsystem to the next (for efficiency of reporting and coherence to the collaboration). The needs of clients of the database and simulator, after all, are diverse, ranging from those of the trigger hardware expert to the new graduate student trying to understand the data for an analysis.

One additional problem brought up in the meeting was a problem with CVS: it was reported that CVS can't store binaries (?), which is what is downloaded to some L1 subsystems (muon FPGA). There is a corresponding TDF (T? Download File) but it is not necessarily a complete description. This issue is yet to be resolved and we look to the experts for advice.

Individual CVS packages containing L1 trigger simulation code for the various subsystems should have dependencies on the CVS packages that contain any ONLINE code that they use (the online code should not be stored/maintained in 2 packages). Changes in the trigger simulation may result from bug fixes, improved accuracy of simulation, or reflect a real change in firmware. It has been suggested that once simulation reaches a steady state, L1 simulation code could be written to recognize run number (or uses RCP switches to relay which algorithm to run) and execute the appropriate simulation code accordingly. Along these lines, a database could be set up which keeps track of which simulation code best represents particular run ranges (including the bug fixes and improved accuracy of simulation but also reflecting changes in firmware), but while this approach may reduce the need for multiple executables it may be fraught with problems if backward/forward compatibility is not designed into the system from the start (starting now). It is not clear how often the ONLINE trigger will change or the simulator will be improved before a steady state of data taking and/or post data taking simulation is achieved. No resolution of this issue was felt to have been accomplished at the meeting.

Some attendees at the meeting expressed interest in contributing to the design of the Run Summary Database. Jeremy Simmons and Taka Yasuda should be contacted.

6 Conclusions

References

[1]