L0trg Monitoring Documents

Zilong Chang, Eleanor Judd

May 2, 2019

1. Introduction

The L0trg monitoring code is used to monitor the STAR Level-0 trigger system layer by layer while the experiment is taking data. The code is usually run a few minutes after a Run ends. The inputs are the trigger data files, which are named run<runnumber>.<filenumber>.dat, and the configuration information stored in the STAR online database. The code will generate a PDF file for each run and move it to a public directory, so people can access the file from the Level-0 Trigger Monitoring link on the STAR online webpage. On each page of the PDF file the left column shows the monitoring plots for the data and the right column shows the comparison plots between data and simulations. The comparison plots are only filled if the simulations don't agree with the data. Therefore, a blank plot in the right column indicates that the STAR Level-0 trigger system is running as expected.

The monitoring task is divided into four branches, which correspond to the three main hardware branches that feed into the Trigger Control Unit (TCU) and a comparison task. The EMC branch monitors data from the BEMC and EEMC. The VTX branch monitors data from the vertex detectors: BBC, ZDC, EPD and VPD. The TOF branch monitors data from the TOF, MTD and Roman Pot detectors. The Monitor branch makes plots of the RHIC bunch-ID information, and also makes plots showing comparisons between individual detectors. The Monitor branch is unique because it does not include any simulations.

The simulations are made for a specific layer by using the same inputs as in the real data. The algorithm for that layer is applied and then the simulated outputs are compared to the outputs in the real data. The algorithm is the same algorithm used in the STAR trigger system, which can be provided by the STAR trigger group.

The command to run the monitoring code is executed in an automatically scheduled manner on the STAR online machines.

2. Monitoring Code Location

All the software is supposed to be stored in the STAR CVS system for safe-keeping and to enable multiple people to work on different branches at the same time. The software can be split into two types. The online-specific framework is only used when STAR is actually taking data so it only needs to match the current hardware configuration and there is no need to maintain backwards compatibility with previous years. However, individual algorithms can be used many times in different parts of the Level-0 system for many years and all of the routines that simulate those algorithms can be used offline during data analysis for many years after the data was actually taken. It is therefore necessary for those simulation routines to be stored in such a way that backwards compatibility can be maintained.

The STAR CVS system is split into online and offline sections. Modules in the online branch do **not** need to maintain backwards compatibility with previous configurations. Modules in the offline branch **do** need to maintain backward compatibility so there are sometimes multiple, dated versions of a piece of software, with code to choose which version to use. These modules cannot compile against code in the online section because then backwards compatibility cannot be guaranteed, however modules in the online section are allowed to compile again code in the offline section.

As a result of these considerations the L0trg monitoring code is currently stored in multiple parts of the STAR CVS system. The main location is in the online/L0Trig module. The online-specific source code and header files are kept here, in the src sub-directory. Two of the header files are softlinks to trigger-specific include files that are used by many other online software modules and are therefore stored in the general online/RTS/trg/include CVS module. All of the scripts are kept in the bin sub-directory. The individual routines that simulate specific trigger algorithms are all implemented in separate source and header files. Ultimately they should be stored in CVS too, in the offline part

of the system. That scheme has not yet been sorted out, so currently the files just exist in a checked-out version of the online/L0Trig module in the ~eleanor directory.

3. A Description of the Monitoring Code

The L0trg framework code does not need to maintain backwards compatibility with previous years. However, for historical reasons the code does currently have several parts that are dated to indicate in which RHIC running period they were used.

The source file main.cc contains the main program to run the code. It takes a trigger data file name as the input parameter. It creates the filename of the output root file using the Process ID of main. It calls routines to initialize each of the 4 code branches, load all DSM and QT register data from the online database, load the slew-correction tables from the archived files, then loop over all the events and processes the data. This code never changes however it uses the functionality defined in the DSMSimulator_YEAR.hh header file, which does occasionally change.

The header file DSMSimulator_YEAR.hh defines the functions for reading the trigger data files, loading the registers and slew-correction tables, making the data monitoring and comparison plots, and saving them into a root file. These functions are implemented in DSMSimulator_YEAR.cc. The member function *init()* initiates the output root file and calls the *init()* function of each branch. The member function *eventLoop()* loops over all the events, and for each event byte-swaps the date and then calls the member function *run()*. The function *run()* calls the *operator()* function of each branch to simulate the logic and fill the histograms. The histograms are saved into the output root file by calling the member function *finish()*. Aside from the YEAR value itself the difference between consecutive DSMSimulator_YEAR files is typically due to differences in the number and names of branches and crates. There are also difference in the code for hard-wiring the values of certain algorithm registers because the L0trg software does not correctly deal with the case where different daughter cards on the same QT board use different register values (e.g. in the case of the ZDC QT algorithm).

The class DSMSimulator_YEAR is derived from the class DSMSimulator which is defined in the DSMSimulator.hh header file. This class definition never changes.

The format of the trigger data files is defined in trgDataDefs.h. The trigger crate configuration numbers are defined in trgConfNum.h. Both trgDataDefs.h and trgConfNum.h are managed by the STAR trigger group. They can be accessed from CVS in online/RTS/trg/include

In order to use the trigger data, it is necessary to byte-swap it when it is read in from the input file because of a difference in endianness between the writing and reading operating systems. The rules for byte-swapping the data are defined in the trgUtil header file. The data for each DSM crate has to be byte-swapped using a function customized for that specific crate because each crate has a different data block definition in trgDataDefs.h. However, the data from all QT crates are byte-swapped by calling one generic function because those crates share a common data block definition. As a result, trgUtil changes whenever there is any addition/subtraction of individual DSM boards, or if there is an addition/subtraction of a complete QT crate. The byte-swapping routines are called from the *swapTriggerDataBlk* function, which is itself called from *eventLoop()* just before the call to *run()*.

The final step, before actually filling histograms and simulating data, is to unpack the data from the byte-swapped block into the local data block. The local data block consists of Board structures nested within Crate structures. The DSM and QT boards share a common definition of the Board structure, which is defined in Board.hh. Each board has a name that is up to six characters long, 32 input channels, 32 registers, an unsigned integer bit mask, and an unsigned integer output. This header file never changes. Each Crate structure contains an array of Board structures, all the unpacking functions and functions to access individual Boards. The data for each DSM crate has to be unpacked using dedicated code because of the crate-specific structures in trgDataDefs.h. However, the data from all QT crates are unpacked by calling one generic function: *decodeQT*. Currently *decodeQT* only unpacks the ADC value for each channel not the TDC value. The unpacking routines are called from the *readCrates* function, which is itself called from the *run()* function. The functions to access individual Boards use the VME base address of each board so those addresses are listed at the beginning of Crate.hh. Crate.hh therefore changes whenever there is any addition or subtraction of DSM or QT boards from the system.

After unpacking the data *run()* calls the *operator()* function for each branch to simulate the algorithms and fill the histograms. The simulation functions are called in the correct order by the branch-specific code: DSMEmcBranch, etc… The branch header file defines the branch as a public instance of the DSMSimulator_YEAR class. It defines the functions for simulating each layer and filling the histograms for that layer as well as pointers to every Board in every Crate in that branch. This header file changes whenever a board is added/subtracted from a crate or if a whole new layer of processing is added to the hardware. All of the functions are implemented in the associated source code, which is where algorithms are assigned to individual boards. The source code changes: whenever a board is added/subtracted from a crate; whenever an existing board is assigned to a new algorithm or whenever the histogram definitions change.

Each trigger algorithm is defined in <algorithm_name>.hh. Its implementation is in <algorithm_name>.cc. The algorithm takes a Board as an input. The algorithm simulates the output of the Board by using the data from the input channels and registers. The algorithm name is the same name used in the trigger DSM algorithm document.

The data from one Run is separated into 10000-event files. A script is used to run the L0trg code on all the files, combine all the separate root output files into one big root file and then use Root macros to complete the processing. The script actually splits the input files into groups of 32 and spawns one copy of the executable for each group so when a long Run is being processed there will be multiple copies of the executable running in parallel. The PlotDSMtoPDF.C root macro is used to plot the histograms in the combined root file to a PDF file. The GetRunLogSummary.C root macro is used to get some run information from the STAR online database that will be displayed on the L0trg monitoring webpage.

4. Compiling and Testing the Monitoring Code

The Makefile in the src directory has all the rules to generate the executable dsm.exe. The “clean” option will remove old copies of the executable as well as old object files and output from test runs. To compile the executable:

```
cvs co online/L0Trig
cvs co online/RTS/trg/include
cd online/L0Trig/src
ln –s ../../RTS/trg/include/trgDataDefs.h trgDataDefs.h
ln –s ../../RTS/trg/include/trgConfNum.h trgConfNum.h
make clean
make
```

The data files are stored in /trg/data/trgdata. To run the executable on one specific input file from one Run:

```
dsm.exe /trg/data/trgdata/run<runnumber>.<filenumber>.dat.
```

The output will be a root file named “dsm,<bignumber>.root. Run the Root macro PlotDSMtoPDF.C to generate the PDF file:

```
root –b –q PlotDSMtoPDF.C'("dsm.<bignumber>.root")'
```

Finally, to run a test of spawning multiple copies of the executable, looping over all data files from one run and extracting Run information from the database use the processonerun script in the bin directory:

```
cd ../bin
ln –s ..src/dsm.exe dsm.exe
ln -s ../src/PlotDSMtoPDF.C PlotDSMtoPDF.C
ln –s ../src/GetRunLogSummary.C GetRunLogSummary.C
processonerun <runnumber> &
```

All of the output files will be automatically moved into the bin/test sub-directory.

5. Running the Code in an Automatically Scheduled Manner

The L0trg code will automatically process every STAR Run after all the data files have been transferred to the /trg/data/trgdata directory. The necessary scripts are in the bin sub-directory. Automatic scheduling is achieved by using the cron system installed on the STAR online machines. The cron process can be run on some limited selection of STAR online machines. Please consult with the STAR Software group for information. For the year 2018, the code was run on the online node onl11.

The script updateL0trgYEAR.cron contains the rule to run the code. To run this script, type:

```
crontab updateL0trgYEAR.cron
```

The cron script wakes up every 10 minutes and calls the shell script updateL0trgYEAR.csh. This script is very similar to the processonerun script. The major differences are that updateL0trgYEAR.csh includes both a lock-file mechanism and a scheme for keeping track of which runs have been processed. The lock-file mechanism is used to ensure that processing of one run has finished before the next one starts. This is useful when the onl cluster is functioning slowly and the L0trg monitoring process falls behind. It prevents the onl computer from being swamped with multiple copies of the L0trg code running simultaneously. The last things that the script does when it has finished processing a run is to save that run number in a text file, and delete the lock file. The next time the cron process wakes up it will search through the list of available Runs and process that first one it finds with a number that is greater than the last one. The list of available Runs is based on looking at the "done" files in the /trg/data/trgdata directory. A "done" file is created for each Run after the transfer of its data files has been completed. Occasionally something will go wrong with that transfer (a network issue?) and there will be some files on disk but no "done" file. In this case the script will skip over that Run. The script also moves the generated PDF files to the directory /onlineweb/www/L0trg, in order to be accessible from the STAR online webpage. Access privileges to write to this directory need to be granted. To gain the access, please consult the STAR Software group.

The command "crontab –l" will allow the user to check that the cron process is still scheduled. "crontab –r" can be used to delete the cron process. If possible it is best to delete the cron process when no spawned processes are running in order to avoid problems with the locking mechanism.

The cron process produces a log file in the bin sub-directory which contains all print statements from the scipt and the running executables. The log fie name contains the date. A new log file will be created when the first Run of each day is processed, and all log messages from that day will be appended to that same log file. Those log files should be checked daily using "grep –i" for the words "break" and "exit" which can indicate that a problem occurred. They should also be checked for the word "read", which will select the line that reports how many register values were read from the database. If the number is zero, then a database access problem occurred and the simulation results will be meaningless. The old log files can be deleted after they have been checked. This is necessary to preserve disk space because they can become large.

Finally, the locking mechanism can sometimes cause problems. If the cron process is deleted or crashes while a Run is being processed, then an unwanted lock file can be left behind on disk. When the cron process is re-started it will never start processing new runs until the unwanted lock file is removed. If this happens the phrase "A lock file /scratch/$USER/updateL0trg.lock exits – exit" will show up in the log file for every single Run. In this case the safest course of action is to delete the cron process, remove the lock file and then restart the cron process cleanly:

```
crontab –r
crontab –l
rm <lockfile_name>
crontab updateL0trgYEAR.cron
crontab –l
```

6 Remarks

To add a new histogram:

- add the definition to the branch header file DSM<name>Branch.hh

- add the instantiation and code to fill it to the branch code DSM<name>Branch.cc

To replace a current algorithm:

- create a pair of .hh and .cc files that contain all the definitions and implementation of the new algorithm.
- replace the call to the old algorithm in the DSM<name>Branch.cc file with the call to the new algorithm
- modify the histogram definitions and filling code appropriately
- add the algorithm into the Makefile (if the old algorithm is not needed, you can remove the old algorithm in the Makefile)
- use “make” to compile the new code.

To add a new board to an existing crate:

- check out the updated version of trgDataDefs.h
- if the board is a DSM board then modify trgUtil to byte-swap the data from this new board.
- add the base address to Crate.hh
- add the definition of the Board name to the branch header file DSM<name>Branch.hh
- add the Board name as a member of the branch class at the beginning of the branch code DSM<name>Branch.cc
- add a call to the appropriate algorithm for this new Board to the function (run) to simulate each Layer
- modify the histogram definitions and filling code appropriately

To monitor ADC and TAC values, the distributions of these values are plotted in a 1-D histogram. For the comparison plot a 2-D histogram of the simulated values vs. the monitor is made. Since some of the ADC and TAC values can extend to a very large value for example from 0 to 4096, to save the storage space, only 256 bins for monitoring these values are sufficient.

To monitor the on/off bits (taking value 0 or 1), a 2-D histogram is recommended. Group all the on/off bits from one board, plot them on the x-axis, and then plot their values, 0 or 1, on the y-axis. For the comparison plots, keep the x-axis the same, and use three values, -1, 0, 1, on the y-axis to show the difference between the simulated bits and the bits in the data.

For each year's L0trg monitoring it is recommended to create two separate directories, one that has the currently running code and one has the new code for test purposes. Once the new code passes all tests, you can move the new code from the test directory to the monitoring directory.