STAR Trigger Auxiliary Configuration Files

H. J. Crawford, E. G. Judd, J. M. Engelage, C. W. Perkins, J. M. Nelson

August 28, 2018

The configuration of most of the STAR Level-0 Trigger System is controlled through the Tier1 file. That file contains the names of auxiliary files that are used to configure large memories like look-up tables and DSM FPGA configuration memories. However, some pieces of the Level-0 VME system configured independently of the Tier1 file: the definition of which algorithm is loaded into each QT board, and the definition of the QT slew correction tables. The procedure for loading the QT algorithms and generating all of those auxiliary files is documented here.

Loading QT MCS Files

- Determine what type or generation QT board you need to load
    - Connect to the debug port of the VME CPU:
        - EITHER telnet trgserv.trg.bnl.local <port_number>
        - OR grab <cpu_name>
        - In order to get a list of which port numbers correspond to which CPU you can enter "grab xxx". Since "xxx" is not a valid CPU name the grab command will respond by printing the list of valid CPU names and their corresponding port numbers.
    - At the prompt, use the "m" command to query the FPGA version running on the mother board (i.e "m 0xYY804100,4" where "YY" is the VME address of the board to be loaded
        - If the value has the form "abcd####" then this is a 1st generation board.
        - If the value has the form "fedc####" then it is a 2nd generation board
- 1st Generation QT Boards
    - MCS files are kept in startrg.starp.bnl.gov:~trg/chris/qt_mcs_files and the program to load those files is up one directory in ~trg/chris
    - Reboot the CPU using its minimal startup script, not the STAR data taking script
        - Ctrl-x to get the CPU to reboot
        - As it starts to reboot type any character to get it to stop and wait
        - When you have a prompt enter "c" (for change)
        - Carriage-return through the entries until you get to the startup file
        - Make a note of which startup file the CPU is currently using, and then change it to /home/startrg/trg/cfg/STARTUP/qt.startup.min
        - Carriage-return until you come to the "VxWorks" prompt, and then enter "@" to tell the CPU to finish booting.
    - Change to the "/home/startrg/trg/chris" directory, load the program, change to the qt_mcs_files subdirectory and run the program, i.e.:

```
-> cd "/home/startrg/trg/chris"
value = 0 = 0x0
```

```
-> ld < prom_programmer.o
value = 33528704 = 0x1ff9b80
-> cd "qt_mcs_files"
value = 0 = 0x0
-> prom_prog(0x10000000,"qt32b_l0_v6_4.mcs")
Programming board 0x10000000
Validating chain...
Found device...
Loading ISPEN instruction...
Checking for read/write protect...
Loading XSC_UNLOCK instruction...
Erasing PROM...
Loading ISPEN instruction...
Checking for read/write protect...
Loading XSC_DATA_BTC instruction...
Loading ISC_PROGRAM instruction...
Checking device status...
PROM ready for programming
Programming prom with: qt32b_l0_v6_4.mcs    ......
PROM programmed: 65536
Verfying PROM contents   ......
PROM verified 65536
Configuring FPGA...
FPGA configured
Time: 64s
value = 0 = 0x0
->
```

- Reboot the CPU (Ctrl-x) and then read the ID register of any of the 4 daughter cards using the "m" command and check that it shows the correct version number. This register returns 0xabcdN0XY where N is the geographic ID (0:3) of the daughter card and X/Y is the major/minor version number of the code, e.g.:

```
-> m 0x109c4000,4
109c4000:  abcd0064-.
```

- If correct, reboot the CPU with Ctrl-x, stopping the process before the CPU can be loaded, change the startup script back to its original STAR data taking script (qt.startup.full) and complete the reboot sequence using "@" at the VxWorks prompt

- 2nd Generation QT Boards
  - The process for loading MCS files in 2nd generation QT boards is basically the same as for the 1st generation boards. All the files are kept in the same directories. However, a different program is loaded and the output message are also slightly different.
  - Reboot the CPU using its minimal startup script then change to the "/home/startrg/trg/chris" directory, load the 2nd generation program, change to the qt_mcs_files subdirectory and run the program, e.g:

```
-> ld < prom_programmer4.o
value = 33528704 = 0x1ff9b80
-> cd "qt_mcs_files"
value = 0 = 0x0
-> prom_prog(0x19000000,"qt32c_l0_v7_8.mcs")
************************************
To program XCF08P (QT8 2006) press 1
To program XCF16P (QT8 2015) press 2
2
Invalid Entry
Programming QT8 2015
Programming board 0x19000000
Validating chain...
Found device...
Loading ISPEN instruction...
Checking for read/write protect...
Loading XSC_UNLOCK instruction...
Erasing PROM...
Loading ISPEN instruction...
Checking for read/write protect...
Loading XSC_DATA_BTC instruction...
Loading ISC_PROGRAM instruction...
Checking device status...
PROM ready for programming
Programming prom with: qt32c_l0_v7_8.mcs    .....
PROM programmed: 65536
Verfying PROM contents   .....
PROM verified 65536
Configuring FPGA...
FPGA configured
Time: 64s
value = 0 = 0x0
```

- -> Reboot the CPU (Ctrl-x) and then read the ID register of any of the 4 daughter cards using the "m" command and check that it shows the correct version number. This register returns 0xfedcN0XY where N is the geographic ID (0:3) of the daughter card and X/Y is the major/minor version number of the code, e.g.:

```
-> m 0x199c4000,4
199c4000:  fedc0078-
```

- If correct, reboot the CPU with Ctrl-x, stopping the process before the CPU can be loaded, change the startup script back to its original STAR data taking script (qt.startup.full) and complete the reboot sequence using "@" at the VxWorks prompt

QT LUT Files

- Binary Files
  - The binary files are kept in startrg.starp.bnl.gov:~trg/cfg/Tier1/QT_LUT

- The name of the binary file is specified in the Tier1 file under the QT_DB_LUT flag, which appears once for each QT board.
- During configuration the name of the binary file is extracted from the Tier1 file and passed to the QT_LUT_Load function, which is part of QT_Config.C. The location of the file is hardwired into QT_Config.C.
- QT_LUT_Load opens the binary file and downloads the information directly to the LUTS. The function does not do any data processing because that has already been done. The binary file contains the actual LUT data.
- QT_Config.C is kept in ~trg/trg_soft_dev/trglib
- The binary files are generated by l2ana01.trg.bnl.local:~trg/online_l2/qtPed.cc.
- The qtPed function is called once for each QT crate at the end of a pedestal run.
- The input is the data from that Run and the Crate TAC offset file, which can put each channel in Offset-mode or Gain-mode (see appropriate section below for details). Note that the default setting is the basic Pedestal-mode which is used for ADC channels. Gain-mode is occasionally used for some ADC channels. Offset-mode is only used for TAC channels.
- qtPed loops over all the data and calculates a hit count, ADC sum and ADC^2 sum for every channel of every board. All ADC values greater than 0 and less than 4095 (0xfff) are included.
- Next qtPed loops over every channel on every board and calculates the mean and sigma for each channel. If mean = 0.0 then it is reset to (MIN_QT_PED = 1). The pedestal value is then set to:
    - Pedestal = ((int) (mean+0.5)) + qt_ped_offset

  where qt_ped_offset has a default value of 1, but can be reset from the "TRG RUN" window of the Run Control GUI. (NOTE: DETAILS ON THE LOCATION OF THIS WINDOW STILL NEED TO BE FILLED IN).
- Finally, qtPed performs a triple nested loop over 17 QT boards (hardwired maximum), 32 channels per board and ADC values from 0 to 4095. For each value:
    - If ADC < Pedestal, then LUT value = 0
    - Else If Offset-mode AND ADC<offset then LUT value = 0
    - Else If Pedestal-mode then, LUT value = ADC – Pedestal + qt_ped_offset
    - Else If Offset-mode then LUT value = ADC – Offset
    - Else If Gain-mode AND Gain>0 then

      LUT value = (ADC – Pedestal + qt_ped_offset) << gain
    - Else (i.e. Gain-mode AND Gain<0) then

      LUT value = (ADC - Pedestal + qt_ped_offset) >> - gain

  The LUT values are written to the binary files while the ((int) (mean+0.5)) and sigma values are written to the Pedestal Mean files.

- Pedestal Mean Files

    - The binary LUT files are difficult for us mere mortals to read so qtPed also writes out the ASCII-format Means file in l2ana01.trg.bnl.local:~trg/online_l2/ped/<crate>.mean.<run number>.
    - Each file contains the (integer) mean ADC value and the sigma that were used to create the binary LUT files for every channel of every board. NOTE that the actual pedestal values include qt_ped_offset, which is not saved in the Means file.
- Crate TAC Offset Files
    - The Crate TAC Offset files that are used in the pedestal calculations are kept on startrg.starp.bnl.gov:~trg/cal/qt/<crateId>_tac.dat.
    - During a Pedestal Run the qtPed function constructs the file name and location using sprint(badchanFileName,"/home/startrg/trg/cal/qt/%s_tac.dat",crateId) so qtPed expects the filenames to be at this location in this form.
    - For crates that contain QT boards from just one detector (e.g. EQ1, QT2, etc.) the Crate TAC Offset file is actually a soft link to the current Detector TAC Offset file in the ~staruser directory (see next section for details of those files), e.g.:

```
startrg eleanor 34 > ls -rtl e*tac*
lrwxrwxrwx. 1 staruser trg   41 Jul 30 16:55 eq1_tac.dat -> /home/startrg/staruser/eq1_tac_zeroed.dat
lrwxrwxrwx. 1 staruser trg   41 Jul 30 16:55 eq2_tac.dat -> /home/startrg/staruser/eq2_tac_zeroed.dat
lrwxrwxrwx. 1 staruser trg   41 Jul 30 16:55 eq3_tac.dat -> /home/startrg/staruser/eq3_tac_zeroed.dat
```

    - For crates that contain QT boards from multiple detectors (BBQ and MXQ) the Crate TAC Offset files are generated by merging the relevant Detector TAC Offset files:
        - create_bbq_offset will merge bbc_tac.dat, vpd_tac.dat and zdc_tac.dat to create bbq_tac.dat
        - create_mxq_offset will merge mtd_tac.dat, p2p_tac.dat, pxy_tac.dat, mvpd_tac.dat and mtd2_tac.dat to create mxq_tac.dat
    - The source code for both routines is kept in ~trg/trg_soft_dev/util/SUN/Tier1/create_<crateId>_offset.c.
    - The names and location of those Detector TAC Offset files are hardwired into the source code.
    - The executables are kept in ~trg/bin so they are available from the staruser account.
    - The executables MUST be run from the ~staruser directory by someone logged in as staruser. If they are run by someone logged in as trg then the output files cannot subsequently be over-written by staruser and the routines will fail with an error at that point.
- Detector TAC Offset Files
    - The Detector TAC Offset files are kept on startrg.starp.bnl.gov:~staruser/.
    - For those detectors like EPD, with many QT boards distributed over multiple crates, the Detector TAC Offset files are named <crateID>_tac_<date>.dat

- The smaller detector systems, which share crates, have files named <detector>_tac_<date>.dat. The current version of each file is defined by creating a soft link in the ~staruser directory from the dated <detector>_tac_<date>.dat file to the undated <detector>_tac.dat file that is read by the merging routine, e.g.:
  - startrg eleanor 43 > ls -l m*tac.dat
  - lrwxrwxrwx. 1 staruser rhstar 21 May 12 05:34 mtd2_tac.dat -> mtd2_tac_05122016.dat
  - lrwxrwxrwx. 1 staruser rhstar 20 May 12 05:34 mtd_tac.dat -> mtd_tac_05122016.dat
  - lrwxrwxrwx. 1 staruser rhstar 21 May 21 00:39 mvpd_tac.dat -> mvpd_tac_20160520.dat
- Each file contains offset and gain values for just those channels of those QT boards that are in Offset-mode (i.e. TAC channels) or Gain-mode (i.e. gain-corrected ADC channels). Any channel NOT listed in the TAC offset file will be in the default Pedestal-mode.
  - If the offset is -1 then the gain value is valid and the channel will be in Gain-mode.
  - If the offset is >=0 then the gain will be ignored and the channel will be in Offset-mode.
- The file format is ASCII so they are human-writeable.
- The individual detector groups are responsible for creating their Detector TAC Offset Files, setting the soft links and running the merging routine.
- A special TAC Offset file exists for each detector/crate, named <detector/crateID>_tac_zeroed.dat. These files contain all channels that will eventually be in either Offset-mode or Gain-mode. Each listed channel has offset = 0, which effectively turns off all offset subtraction and gain correction. These files are typically used at the beginning of a RHIC Run Period when the Offsets are being calibrated.

QT Slew Correction Files

- Crate Slew Correction Files
  - The Crate Slew Correction files are kept in /home/startrg/trg/cal/qt/<crateId>_slew_corr.txt on startrg.starp.bnl.gov.
  - These files are only usable by those subsystems that include the slew correction logic in their QT algorithm, i.e. VPD, MTD, BBC and EPD.
  - Each file contains entries for just those QT boards that actually use the slew correction tables, which is currently just QT boards servicing VPD and MTD. EPD may start using them in 2019.
  - During configuration these files are read by the function qtLoadSlewCorrections, which is part of QT_Config.C. The input file location is hardwired to "/home/startrg/trg/cal/qt/<crateId>_slew_corr,txt"
  - If qtLoadSlewCorrections fails to open the specified file, then it is assumed that this crate does not use slew corrections and the function quits without error.

    - If the Crate Slew Correction file contains slewing information for a QT board whose algorithm does not include the slew correction logic then qtLoadSlewCorrections will fail, and generate bus errors.
    - For crates that contain QT boards from just one detector (EQ1:3) the Crate Slew Correction file is actually a soft link to the current Detector Slew Correction file in the ~staruser directory (see later section for details of those files), e.g.:

```
startrg eleanor 36 > ls -rtl e*slew*
lrwxrwxrwx. 1 staruser trg 49 Jul 30 16:58 eq1_slew_corr.txt -> /home/startrg/staruser/eq1_slew_corr.06172018.txt
lrwxrwxrwx. 1 staruser trg 49 Jul 30 16:58 eq2_slew_corr.txt -> /home/startrg/staruser/eq2_slew_corr.06172018.txt
lrwxrwxrwx. 1 staruser trg 49 Jul 30 16:59 eq3_slew_corr.txt -> /home/startrg/staruser/eq3_slew_corr.06172018.txt
```

    - For crates that contain QT boards from multiple detectors (BBQ and MXQ) the Crate Slew Correction files are generated by merging the relevant Detector Slew Correction files:
        - create_bbq_slew_corr will merge bbc_slew_corr.txt and vpd_slew_corr.txt to create bbq_slew_corr.txt
        - create_mxq_slew_corr will merge mtd_slew_corr.txt, mvpd_slew_corr.txt and mtd2_slew_corr.txt to create mxq_slew_corr.txt
    - The source code for both routines is kept in ~trg/trg_soft_dev/util/SUN/Tier1/create_<crateId>_slew_corr.c.
    - The names and location of those Detector Slew Correction files are hardwired into the merging routines.
    - The executables are kept in ~trg/bin so they are available from the staruser account.
    - The executables MUST be run from the ~staruser directory by someone logged in as staruser. If they are run by someone logged in as trg then the output files cannot subsequently be over-written by staruser and the routines will fail with an error at that point.
- Crate Slew Correction Archive
    - In order to maintain a complete record of how each run is configured the slew correction files are also saved for every run. These archive files are written by the qtLoadSlewCorrections function to "/home/startrg/trg/chris/run_info/slew_corr/<crateId>_slew_corr.<runnum>.txt"
    - After each run has ended the archive files are then moved to trgscratch.starp.bnl.gov:/data/plots/slew_corr/ by the make_plots perl script, which runs on trgscratch.starp.bnl.gov.
- Detector Slew Correction Files
    - The Detector Slew Correction files are kept on startrg.starp.bnl.gov:~staruser/.
    - For those detectors like EPD, with many QT boards distributed over multiple crates, the Detector Slew Correction files are named <crateID>_slew_corr.<date>.txt

- The smaller detector systems, which share crates, have files named <detector>_slew_corr.<date>.txt. The current version of each file is defined by creating a soft link in the ~staruser directory from the dated <detector>_slew_corr.<date>.txt file to the undated <detector>_slew_corr.txt file that is read by the merging routine, e.g.:

```
startrg eleanor 44 > ls -l m*slew_corr.txt
lrwxrwxrwx. 1 staruser rhstar 27 Feb  9 19:01 mtd2_slew_corr.txt -> mtd2_slew_corr.20160209.txt
lrwxrwxrwx. 1 staruser rhstar 26 Feb  9 19:01 mtd_slew_corr.txt -> mtd_slew_corr.20160209.txt
lrwxrwxrwx. 1 staruser rhstar 27 May 21 11:25 mvpd_slew_corr.txt -> mvpd_slew_corr.20160521.txt
```

- Each file contains bin limits and slew correction values for JUST those channels of those QT boards that use the slew correction logic.
- The file format is ASCII so they are human-writeable.
- The individual detector groups are responsible for creating their Detector Slew Correction File, setting the soft links and running the merging routine.
- A special Detector Slew Correction file exists for each detector, named <detector/crateID>_slew_corr_zero.txt. These files contain all channels that will eventually use the slew correction logic. All of the slew correction values are set to 0, which effectively turns off all the correction. These files are typically used at the beginning of a RHIC Run Period when the slew corrections are being calibrated.

DSM RBT Files

- RBT files are the DSM-equivalent of the QT MCS files. They are ASCII-formatted files, which contain a header and the bit stream needed to configure the DSM Computational FPGA. **NOTE**S: When the RBT file is transferred from the Windows PC, where it was generated, to startrg.starp.bnl.gov (Linux) the transfer must be done in text mode, not binary mode, to avoid unwanted special characters being inserted at the end of each line. Also the file must be edited to reduce the header to 6 lines. The format of the RBT header was changed several times by Lattice over the years. In order to avoid complexity in the STAR software the simplest solution was to remove the unwanted header lines from the RBT files.
- These files are kept in startrg.starp.bnl.gov:~trg/cfg/Tier1/DSM.
- The name of the RBT file is specified in the Tier1 file with the DSM_FPGA_CNF flag, which appears once for each DSM board.
- During configuration the name of the RBT file is extracted from the Tier1 file ad used by the DSMinit function, which is part of DSM_Config.C, to configure the FPGA.
- DSM_Config.C is kept in ~trg/trg_soft_dev/trglib

DSM LUT Files

- General Information

- LUTS on DSM boards can be filled with a function generated during configuration, or with data read from a binary file.
- The binary files are kept in startrg.starp.bnl.gov:~trg/cfg/Tier1/DSM_LUT
- The switch between function data or file data is made using control flags, which can be specified with the DSM_LUT tag in the Tier1 file, and the binary file name, which is specified under the DSM_LUT tag.
- During configuration the control flags and the name of the binary file are extracted from the Tier1 file and examined by the DSMinit function, which is part of DSM_Config.C
  - If the control flags are present, then this indicates that the data is contained in a crate-wide binary file (used by EMC and TOF). DSMinit will open the file, read the data and call DSM_Mem_LUT to configure the LUTS.
  - If the control flags are NOT present, then one file name is specified for each of the four LUTS. If the file name is DSM_Mem_Zeros.dat then DSMinit calls DSM_Mem_All, which automatically fills the whole memory with zeros.
  - If the file name is DSM_LUT_1to1.dat then DSMinit calls DSM_Mem_Function with a function flag which automatically fills the 32-bit wide memory with two 16-bit ramps.
  - If the file name is DSM_Single_1to1.dat then DSMinit calls DSM_Mem_Function with a function flag which automatically fills the 32-bit wide memory with a single ramp.
  - For any other file names DSMinit calls DSM_Mem_File, which opens the binary file and downloads the information directly to the LUTS.
- DSM_Config.C is kept in ~trg/trg_soft_dev/trglib
- LUTS on DSM boards are typically defined as 1-to-1 maps. The exceptions are the TOF and EMC layer-0 DSM boards, which use the LUTS to mask out dead or noisy channels.

- TOF LUT Files
  - TOF LUT files are generated using the tof_lut program.
  - The source code is kept in ~trg/trg_soft_dev/util/SUN/Tier1/tof_lut.C.
  - The executable is kept in ~trg/bin so it is available from the staruser account.
  - The executable MUST be run by someone logged in as staruser. If it is run by someone logged in as trg then the output files cannot subsequently be over-written by staruser and tof_lut will fail with an error at that point.
  - There are two input files:
    - ~staruser/tof.<date>.dat contains a list of which trays should be masked out. This file is ASCII format so it is human-writable.

        - ~trg/trg_soft_dev/util/SUN/Tier1/tof_table.txt contains a map between each TOF tray and its input location in the TOF DSM boards. It is ASCII format. The filename and location are hardwired into tof_lut.C
    - The output file is always ~trg/cfg/Tier1/DSM_LUT/tof.lut.bin
- EMC LUT Files
    - EMC LUT files are generated using the conv_lut program.
    - The source code is kept in ~trg/trg_soft_dev/util/SUN/Tier1/barrel_lut.C. NOTE: the reason why the source code and executable have different names has been forgotten.
    - The executable is kept in ~trg/bin so it is available from the staruser account.
    - The executable MUST be run by someone logged in as staruser. If it is run by someone logged in as trg then the output files cannot subsequently be over-written by staruser and conv_lut will fail with an error at that point.
    - This program can generate LUT files for both BEMC and EEMC. The switch is based on the name of the input data file.
    - There are two input files:
        - ~staruser/xxx.<date>.dat contains a list of which EMC trays should be masked out. "xxx" MUST be one of "eec", "bar", "bce" or "bcw". "bar" covers both "bce" and "bcw" together. This file is ASCII format so it is human-readable.
        - ~trg/trg_soft_dev/util/SUN/Tier1/xxx_table.txt contains a map between the relevant EMC patches and their input location in the EMC DSM boards. The file names are constructed by conv_lut using the "xxx" value that was specified for the input data file. The files location in the Tier1 subdirectory is hardwired into conv_lut. The file format is ASCII.
    - The output files are always ~trg/cfg/Tier1/DSM_LUT/bce.lut.bin, bcw.lut.bin and eec.lut.bin, depending on which "xxx" was specified on input. If "bar" was specified, then both bce.lut.bin and bcw.lut.bin are generated.
- Special Cases
    - In addition, it is sometimes necessary to use an LUT that contains a 1-to-1 map with an individual noisy bit masked out. These LUT files are generated using the spec_trg_luts program.
    - The source code and executable (compiled for Linux) is kept in ~trg/trg_soft_dev/util/SUN/Tier1
    - The input arguments consist of two 32-bit words and a print flag. The 1st word defines which bits should be masked out. The 2nd word defines which bits should be forced on and the print flag turns on/off the printing of verbose messages.
    - The output file is produced in the directory where the program is run. If it is created in a non-standard directory then the file must be moved to ~trg/cfg/Tier1/DSM_LUT.

DSM Input Memory Files

- Input Memories on DSM boards can be filled with a function generated during configuration, or with data read from a binary file.
- The binary files are kept in startrg.starp.bnl.gov:~trg/cfg/Tier1/DSM_LUT
- The switch between function data or file data is made using the memory control flags (DSM_INMEM_EN and DSM_INMEM_PLAY_REC) and the binary file name, which is specified in the Tier1 file under the DSM_INMEM flag.
- During configuration the control flags and the name of the binary file are extracted from the Tier1 file and examined by the DSMinit function, which is part of DSM_Config.C
    - If the flags indicate that the Input memories are not enabled, or are enabled to record data then DSMinit calls DSM_Mem_All, which automatically fills the whole memory with zeros.
    - Otherwise (i.e. the memories are enabled to play data) if the file name is DSM_Mem_Zeros.dat then DSMinit calls DSM_Mem_All, which automatically fills the whole memory with zeros.
    - If the file name is DSM_Mem_Ones.dat then DSMinit calls DSM_Mem_All with a different flag, which automatically fills the whole memory with ones (0xffffffff).
    - If the file name is DSM_8bit_Ramps.dat then DSMinit calls DSM_Mem_Function with a function flag which automatically fills each 32-bit wide memory with four 8-bit ramps.
    - For any other file names DSMinit calls DSM_Mem_File, which opens the binary file and downloads the information directly to the Input Memories. The function does not do any data processing because that has already been done. The binary file contains the actual data. DSM_Mem_File is hardwired to look for these binary files in ~trg/cfg/Tier1/DSM_LUT
- DSM_Config.C is kept in ~trg/trg_soft_dev/trglib

DSM Output Memory Files

- Output Memories on DSM boards can be filled with zeroes or with data read from a binary file.
- The binary files are kept in startrg.starp.bnl.gov:~trg/cfg/Tier1/DSM_LUT
- The switch between zeroes or file data is made using the memory control flags (DSM_OUTMEM_EN and DSM_OUTMEM_PLAY_REC)
- During configuration the control flags and the name of the binary file are extracted from the Tier1 file and examined by the DSMinit function, which is part of DSM_Config.C
    - If the flags indicate that the output memories are not enabled, or are enabled to record data then DSMinit calls DSM_Mem_All, which automatically fills the whole memory with zeros.

    - Otherwise, DSMinit calls DSM_Mem_File, which opens the binary file and downloads the information directly to the Output Memories. The function does not do any data processing because that has already been done. The binary file contains the actual data. DSM_Mem_File is hardwired to look for the binary files in ~trg/cfg/Tier1/DSM_LUT
- DSM_Config.C is kept in ~trg/trg_soft_dev/trglib

Checking DSM Memories

- Sometimes it is desirable to check exactly what values were downloaded to a particular DSM memory block. This can be done using the readDSM program.
- The source code and executable (compiled for VME) are kept in ~trg/trg_soft_dev/trglib
- The arguments are the starting VME address of the memory block and an output file name:
    - int readDSM(int memAdd, char* fname)
- The output file is produced in /home/startrg/trg/dsm_test so this code will only work on VME processors that have access to both the trglib and dsm_test directories.
- The code will read 65536 4-byte words from the specified starting VME address and write the data to the output file in binary (not ASCII) format.