Trigger Level 1 and 2 architecture

**V. Lindenstruth, H. Crawford, L. Greiner, E. Judd,
K. Marks, J. Whitfield**

March 10, 1995

Note: some minor parts of that document are not finalized yet. They are marked with ‘**’ at the beginning of the paragraph.

1.0 Introduction 5
1.1 Trigger L0 6
1.2 Trigger L1 7
1.3 Trigger L2 8

2.0 Trigger L0 Components 11
2.1 Trigger Control Unit (TCU) 11
2.1.1 Trigger Type FiFo (TTF) 11
2.1.2 The Trigger Token FiFo (TKF) 12
2.1.3 Trigger Response FiFo (TRF) 13
2.1.4 TPC Pile-Up Rejection 13
2.2 The Data Storage and Manipulation module DSM 14
2.2.1 DSM input buffer readout 15
2.2.2 Trigger Address Broad Cast 16
2.3 The Trigger/Clock Driver module TCD 16

3.0 Trigger L1 and L2 Components 17

4.0 Trigger Crate and Network Architecture 19
4.1 The Crate Architecture 19
4.2 The L1/L2 Network Requirements 21

5.0 L1 and L2 Software Architecture 25
5.1 The L1 and L2 Trigger Processors 27
5.2 L1 Architecture 28
5.3 L2 Architecture 29
5.3.1 EMC and TOF L2 Input 29
5.4 Initialization of the Trigger System 30

A.0 Appendix 31
A.1 Trigger Analysis Code Debugging 31
A.1.1 I/O Support on Trigger Processors 31
1.1.2 Shared Data Structures 33
1.1.3 Initialization - Configuration Files 34
1.1.4 The L1 and L2 Analysis Code Call Frame 35

1.0 Introduction

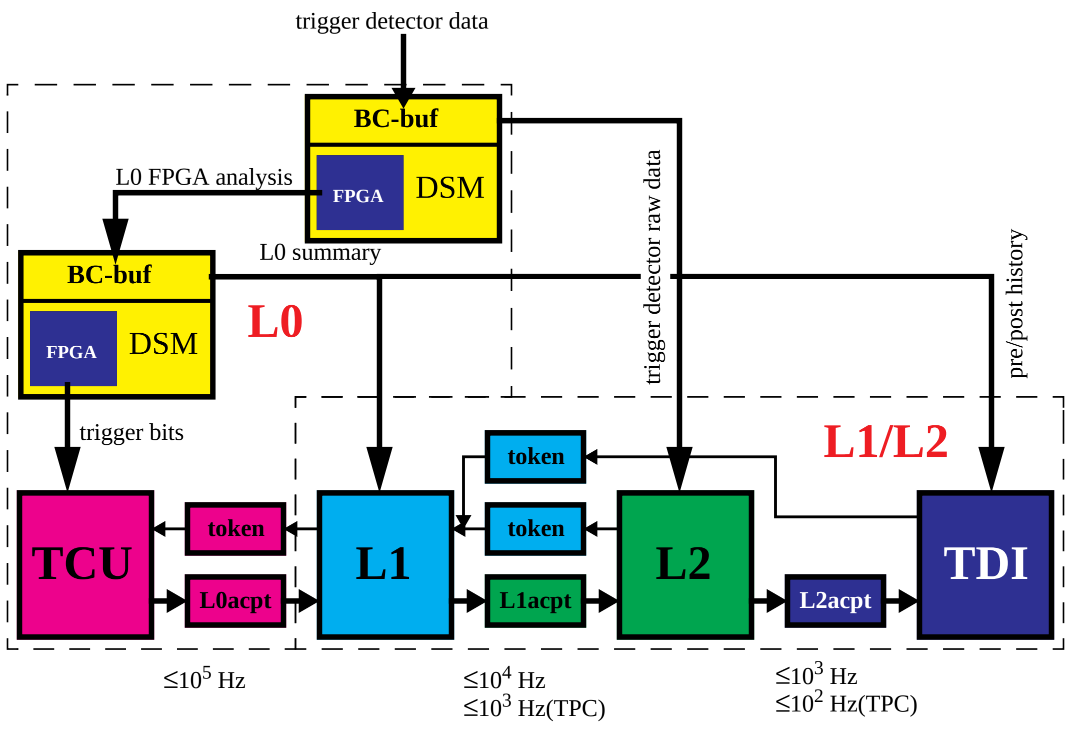
This document describes the architecture of level one and two of the STAR trigger system. These two trigger levels are two sets of veto processing units that reduce the primary trigger rate. Their principal difference is the time budget for the trigger decision and the available input data. Figure 1 sketches an overview of the STAR trigger system including L0 and the Trigger/DAQ interface. The digitized raw data of the trigger detectors is stored every bunch crossing in buffers holding 65000 bunch crossings. This input data stream is also fed into a tree structure of field programmable gate arrays (FPGA) allowing a very flexible analysis of the trigger detector data. In the actual design there is the bunch crossing buffer for 128 input bits and one FPGA producing 16 output bits implemented on a generic VME board. This *Data Storage and Manipulation/DSM* board is detailed in Krista Marks paper: *Trigger L0 Architecture - TCU and DSM Boards*. The various layers of the L0 system are formed by connecting the appropriate in- and outputs of the DSM boards to form a tree structure. The output of this L0 system drives the Trigger Control Unit (TCU) that identifies trigger conditions and fires triggers after appropriate pre scaling. The TCU is the only instance within STAR that may fire triggers. All higher levels in the trigger system can only abort an already issued trigger. If a detector wants to fire a trigger it can request a trigger by asserting the appropriate input bits to the TCU within the specified time lines. The output of the TCU is a request queue that drives the L1 veto processor farm.

There are up to 8 independent sub-detectors foreseen in STAR. Currently there are five sub-detectors defined (SVT, TPC, TOF, EMC and XTPC). STAR does not have a common dead time. Therefore the Trigger system has to be able to handle 8 individual detector dead times. It has also to be able to handle any combination of coincidence trigger requests and dead time states.

After a L0 trigger the event is sent through two layers of veto processors (level one and two) before the event is forwarded to DAQ. The major difference between L1 and L2 is the available time and input data. L1 receives the L0 summary data of the event. L2 receives both the L1 input and output and the full raw data of all trigger detectors after an L1 accept. The Trigger/DAQ interface reads also a specified pre- and post history of bunch crossings upon an L2 accept and notifies DAQ.

The STAR Trigger system is completely token driven and therefore, with appropriate buffering, can be made dead time free. Each event is accompanied by a trigger token which is unique during the lifetime of the event. For details of the trigger token concept please refer to the *Trigger/Clock distribution Tree* requirement document by Volker Lindenstruth. The TCU will issue triggers only if there are unused tokens available in its trigger token input queue (Figure 1). The L1 system will receive requests through the trigger request output queue on the TCU and return tokens into the token FiFo. Upon an L1 accept the L1 input and output data is pushed into the L2 systems request queue. A similar interface exists between L2 and the Trigger/DAQ interface. The maximum number of events per second that can be processed by L1 and L2 are indicated in Figure 1. Due to the limited number of trigger tokens and the small event size of the trigger detector there will be enough buffer space in each layer of the trigger system to accommodate any kind of trigger bursts. However the maximum length of trigger bursts that can be allowed by trigger is defined by the maximum L1 and L2 accept/abort latency which are detector parameters. The maximum trigger burst length is controlled by the number of tokens available in the trigger token FiFo on the TCU. Due to this concept there is no complex handshake required between the different layers of the trigger system.

FIGURE 1. Trigger overview



1.1 Trigger L0

Although level zero of the trigger system is not within the scope of this document its major architectural features will be outlined here to allow better understanding of the first and second trigger levels discussed in the following chapters. The L0 trigger setup is described in detail in *Trigger L0 Architecture - TCU and DSM Boards* by Krista Marks available on the World Wide Web.

The Trigger level zero (L0) hardware is pipelined and runs completely in lock step with the RHIC strobe. All Trigger detectors are digitized and readout once per bunch crossing. Therefore the L0 input event rate is $9x10^6$. The L0 output rate is limited to 10^5. This will be accomplished by setting up both the trigger conditions and the prescale factors.

The digitized result of the trigger detectors is processed by a hierarchy of FPGAs. These FPGAs are implemented on VME boards (Data Storage and Manipulation (DSM) boards). They can be configured on a run by run basis. All DSM boards are connected with each other forming a tree structure.

The output of this FPGA tree is a bit pattern identifying the various trigger conditions. There are 128 trigger bits foreseen. Other trigger detectors like TOF and EMC may input their L0 triggers at this input, too. This bit pattern is reduced using an FPGA into a summary word, which is input into the Trigger Control Unit (TCU). The TCU compares the trigger request with the availability or busy status of the STAR subdetectors and produces a 16-bit trigger word allowing up to 65k different trigger conditions. Each of these trigger conditions has an individual scale down. Depending on the scale down factor and the trigger word a trigger is generated by the TCU and forwarded to the Trigger/Clock Driver modules (please refer to the *Triger/Clock Distribution* requirement document by Volker Lindenstruth).

The Trigger/Clock Driver crate hosts the Trigger/Clock Driver VME modules (TCD). The VME P3 back-plane will be used to distribute the trigger command for every bunch crossing, which is produced by the TCU. These Driver modules serialize and forward the trigger together with the DAQ command word, the trigger token and two detector specific clock signals.

Another functionality of the DSM board is a cyclic input buffer that is filled at RHIC strobe rate with the data of the previous DSM stage or the detectors raw data and that can be readout randomly. The depth of this buffer allows the raw data of any bunch crossing to be fetched up to 7.2 ms[1] later before it is overwritten.

The output of the FPGA on the DSM board is copied into a FiFo that has two functions. First it can be used to implement a delay in units of RHIC strobes to accommodate for shorter DSM board pipelines like the VPD and VTC DSM tree. The second functionality of the FiFo is that it can be read out by a VME processor in order to debug the DSM board and to debug the FPGA algorithm (please refer to T*rigger L0 Architecture - TCU and DSM Boards* , Figure 3 by Krista Marks).

1.2 Trigger L1

The output rate of the Trigger level zero system is $\leq 10^5$ Hz. The main purpose of the L1 system is to reduce the trigger rate down to $\leq 10^4$ Hz for fast detectors like EMC and TOF and $\leq 10^3$ Hz for events in coincidence with the TPC. This is accomplished with a group of fast processors. The output of the DSM tree and correspondingly the input data available for the level one analysis is summarized in Table 1:

1. The 16-bit depth of the input buffer is driven by bit width considerations of the trigger address (refer to Figure 2)

TABLE 1. The L0 result output and data input into the trigger level one veto processor

detectors	byte count	comments
coarse pixel array	64	the combined output of the first DSM layer of the MWC and CTB - the coarse pixel array
MWC/CTB multiplicity	2	the result of the L0 multiplicity analysis of the coarse pixel array data
MWC/CTB dipole anl.	2	the result of the L0 dipole analysis of the coarse pixel array data
MWC/CTB spc. topology	2	the result of the L0 *special topology* analysis of the coarse pixel array
MWC/CTB higher moments	2	the higher moments analysis of the coarse pixel array
VPD	2	the vertex position derived from the minimum flight time on both sides of the interaction
VTC	2	veto calorimeter - measure of centrality
spare	4	32 spare L0 input bits
TOF	0	TOF does not have a Trigger L0 input
EMC	36	L0 EMC input[a]
trigger word	2	the trigger word of the given bunch crossing
trigger action word	2	the trigger action word of the given bunch crossing
spare	8	to round it up to 128 bytes
total	128	total size L1 input data set

a. 4η x 4ϕ patches (16 bit) plus 3 bits per single, two and nine tower sums and 16-bit total energy

This is essentially the L0 summary, which led to firing of the given event. The total size of this data is 128 bytes. These 128 bytes have to be transferred into one level one processor within 10 µs, which corresponds to a minimum data rate of about 13 MB/sec

The L1 analysis is based on the summary data readout of the DSM boards. It is also required that the L1 accepts are broad cast to the detectors in exactly the order of the original events. There has to be one event processed every 10 µs in the Trigger L1 pipeline. Due to that rate the L1 system is implemented next to the TCU in the CAVE.

The time needed to perform the level one functions is not exactly determined yet because it depends on the choice and performance of the processor and the complexity of the level one algorithm. However it is believed that it will be of the order of 100 µs. Therefore about 10 level one processors are required in order to build this 10 stage pipeline.

1.3 Trigger L2

The accept rate of the Trigger L1 system needs to be reduced by another order of magnitude. This is accomplished by the Trigger L2 system. Its output rate is limited to $\leq 10^3$ Hz for fast detectors and $\leq 10^2$

Hz for events in coincidence with the TPC. The L2 processors have access to the full raw data of all Trigger detectors and the results of the L1 analysis of the given event. The corresponding data volume is summarized in Table 2:

TABLE 2. The data input into the trigger level two veto processor

detector	byte count	comments
L1 input	128x (N_{pre}+N_{post}+1)	L1 input data times the number of bunch crossings of pre and post history
L1 result	128	fixed length L1 summary (working assumption)
MWC	128	multi wire chamber (2x48 channels plus DSM granularity overhead)
CTB	256	central trigger barrel (240 channels plus DSM granularity overhead)
VPD	96	vertex position detector (2x24 channels ADC and TDC)
VTC	16	4 channels veto calorimeter and 64-bit bunch crossing number
**TOF	128	128 bytes assumed as working assumption
**EMC	128	128 byte assumed as working assumption
total	1264 3568	total size of L2 input data set for N_{pre}=N_{post}=1 (EMC/TOF triggers) total size of L2 input data set for N_{pre}=N_{post}=10 (TPC triggers)

The data of the bunch crossings preceeding and following the bunch crossing causing the trigger (the pre and post history) has to be shipped in order to allow rejection of pile-up events. For slow detectors like TPC and SVT there may be up to 10 bunch crossings pre and post history. For fast detectors that may run at a higher rate than the TPC this number is limited to 1.

The data items listed in the table above have to be transferred to one level two processor within the time budget of 100 µs, because the maximum L2 input rate is 10^4 events per second. This corresponds to a maximum data rate of about 15 MB/sec assuming 9000 EMC/TOF and 1000 TPC events per second.

Every event is processed by exactly one L2 processor, based the data listed in Table 2. L2 accepts or aborts may be fired at any time. There has to be one event processed every 100 µs in the Trigger L2 pipeline. Due to that rate the L2 system can be implemented in the counting house.

As in case of the L1 architecture, the time needed to perform the level two functions is not exactly determined yet because it depends on the choice and performance of the processor and the complexity of the algorithm. However as a working assumption we assume in the following an L2 analysis time of 1 ms. Therefore about 10 level two processors are required in order to build this 10 stage pipeline.

2.0 Trigger L0 Components

The data sizes and rates quoted in Section 1.0 allow implementation of all Trigger specific hardware as VME modules. VME64 is not required but desirable in order to stay compatible with commercial VME processors. Therefore the whole Trigger system will comprise of a couple of VME crates and a VME crate interconnect.

This chapter describes the features of the Trigger specific level zero hardware components of the Trigger system. Most emphasis will be on the components of these L0 boards that interface with the L1 and L2 systems. For a detailed description of the Trigger L0 hardware refer to *Trigger L0 Architecture - TCU and DSM Boards* by Krista Marks.

The complete Trigger L0 hardware consists of three different VME 9U boards (except the Trigger detectors themselves):

- DSM (Data Storage and Manipulation module): FPGA with cyclic input buffer
- TCU (Trigger Control Unit): trigger generation, dead time logic, prescale logic
- TCD (Trigger/Clock Driver): trigger and clock distribution module

2.1 Trigger Control Unit (TCU)

This VME board is the kernel of the Trigger system. It handles the various detector dead time signals, performs the prescale functionality and issues triggers. It is the only device within the whole STAR detector that issues triggers. All functions on this board are memory mapped into the VME space allowing simple memory mapped configuration and diagnostic of the device. The handshake with the Trigger L1 and L2 system is performed by implementing elasticity FiFos. This allows to enqueue both L1 and L2 processing requests and to enqueue L1 and L2 accepts or aborts. By choosing the depth of these FiFos deep enough, FiFo full/overflow handshake logic can be avoided because even in the worst case scenario the FiFo cannot overrun. This is possible due to the limited number of outstanding trigger requests due to the 12-bit wide trigger token. The TCU module will treat a write to a full FiFo as an error, flag it in its CSR space and terminate the VME transaction with a BUS ERROR.

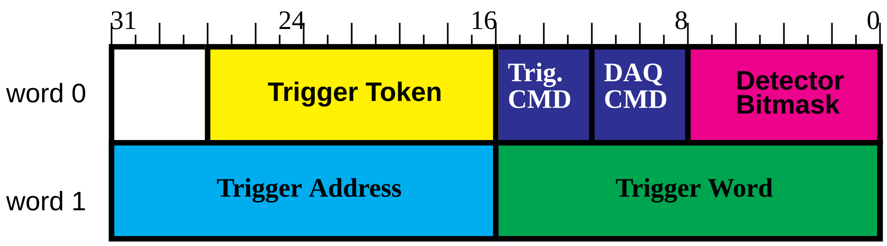
2.1.1 Trigger Type FiFo (TTF)

The Trigger Type FiFo is the L0-to-L1 request queue. If the TCU issues a trigger, it also pushes an entry into that FiFo defining the nature of that event. The level one controller CPU pulls these entries from that FiFo and submits the event for level one processing.

Figure 2 defines the data fields and their format of these L0 requests. The 4-bit wide white field is unused and reserved. It will be filled with zeroes. A trigger is completely described with this 60 bit word. It is read as two 32 bit words. The first long word defines the trigger token and the trigger command word consisting of trigger and DAQ command issued and the involved detectors. The second word defines the trigger word and the trigger address. The trigger address is the subaddress of the cyclic event buffer in the DSM boards. It is needed to fetch the data of the given event out of the cyclic event buffers on the DSM boards.

The Trigger Type FiFo has one special feature. If the level one controller tries to read it while it is empty it will respond with all data bits forced to zero. A zero trigger token, trigger command and detector bitmask is an impossible trigger condition and can be used by the L1 trigger controller CPU (L1CTL) to detect an empty trigger type FiFo. The advantage of this implementation is that the L1CTL CPU can implement trigger type FiFo reads as fixed length burst VME reads of the size of one cache line. The PowerPC, for example, has a 32 byte cache allowing up to 4 trigger types to be read in one burst. This is much more efficient than polling.

FIGURE 2. The Trigger Type FiFo (TTF)

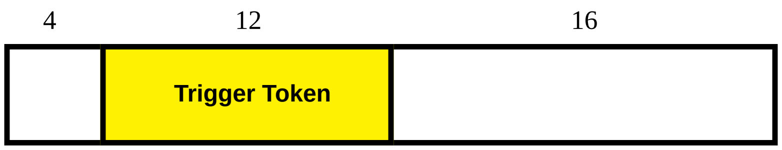


2.1.2 The Trigger Token FiFo (TKF)

The 12-bit wide Trigger Token FiFo forms the queue of available trigger tokens. The basic idea of this concept is to tag every event with a unique token to make sure that the correct subevents of the various STAR subdetectors are assembled to an event in DAQ.

The TCU reads one token every time it issues an event trigger and distributes it with that trigger. After issuing all available trigger tokens the TCU will go into a busy state and stop issuing any more triggers. This is required to ensure that each token is unique during the processing time of an event. However, this can only happen if there is no response from DAQ, because the number of available trigger tokens was chosen to be twice the maximum number of outstanding events in DAQ. A trigger token will be reused if the corresponding event is no longer active, which happens in two cases: the event is completely built and queued for taping or the event was aborted by any of the reject processors. The TCU returns the trigger token to the trigger token FiFo in case of L1/L2 aborts if it receives an appropriate abort command from either the L1 or L2 controller (refer to Section 2.1.3). In case of a L2 accept DAQ has to return the trigger token to the Trigger-DAQ interface, which will make it available again by pushing it into the trigger token FiFo, which mocks up a queue of available trigger tokens.

FIGURE 3. The trigger token FiFo (TTF)



Under no circumstances should the L1CTL CPU encounter a full trigger type FiFo when it tries to return a trigger token. This condition is a fatal hardware or software flow control error. The TCU hardware will respond with a BUS ERROR. The FiFo fill state is available through VME memory mapped CSR registers.

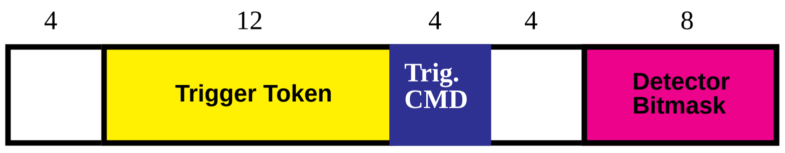
After a RESET the initialization process will clear the trigger type FiFo and then fill it with trigger tokens. This concept permits running with a smaller number of trigger tokens and therefore limiting the depth of all available pipelines. In an extreme case there could be for debugging just one trigger token forcing the whole STAR trigger and DAQ system in lock step without changing a single line of code.

2.1.3 Trigger Response FiFo (TRF)

L1/L2 accepts and aborts are posted to the TCU by writing to a memory mapped FiFo (TRF Trigger Response FiFo). The TCU will broadcast any of the requests posted if there is no other trigger request for the given bunch crossing. The TRF FiFo is 32 bits wide and has to be 8190 entries deep, because there are up to 4095 outstanding triggers, and there are up to two trigger responses per trigger (L1 accept and L2 accept or L2 abort; L1 aborts produce only one response). For debugging and maintenance purposes the TRF has associated CSR space to read its filling status and to clear it through VME.

The bit layout of the TRF is shown in Figure 4. It matches the definition of the trigger type FiFo (Figure 2). Therefore no bit shifting or masking is required by the L1CTL CPU. All white fields are unused and will be ignored by the TCU. The Trigger CMD field specifies if the given command is a L1, L2 accept or an abort. The command format is identical to the trigger command word. The four bits specified here are directly forwarded to the Trigger/Clock Drivers (TCD) allowing to fire any kind of trigger command for test purposes under software control. Under normal running conditions however only the named three of the fifteen possible trigger command words will be used. The trigger token and detector bitmask are the original values passed through the Trigger Type FiFo.

FIGURE 4. The Trigger Response FiFo (TRF)



A posted request in the trigger response FiFo will be submitted during the next bunch crossing where the TCU does not need to issue a trigger. This prioritizing scheme is required because triggers and accepts and aborts are distributed through the same trigger clock distribution tree. Triggers have to be issued immediately because there has to be a fixed timing relationship between the trigger and the bunch crossing causing the trigger. This means however that there is an undetermined latency between posting a trigger response and its actual distribution of up to 4094 bunch crossings (this would be, however, a very untypical case).[1]

2.1.4 TPC Pile-Up Rejection

It is important to be able to check for pile-up conditions during the drift time of the TPC. A TPC pile-up condition exists if there is during t_{drift} (depending on the drift velocity up to 80 µs) before and after the actual bunch crossing another trigger. The TCU will not fire a TPC trigger if there was another good event less than t_{drift} earlier. After firing a TPC event a Flip-flop will be enabled for t_{drift}. If there is an other interaction during this time this Flip-flop will be set. It is reset by the TCU synchronously with firing a TPC trigger. The L1controller will check this bit if there is a L1 accept of a TPC event. In case of a pile-up condition it may abort the event (refer to *Trigger L0 Architecture - TCU and DSM Boards*, Krista Marks).

2.2 The Data Storage and Manipulation module DSM

The two major functional components on the DSM board are the cyclic input buffer that latches all incoming data at RHIC strobe rate (data storage) and a field programmable gate array that is used to distil the relevant information out of the input data stream (data manipulation). Each DSM board has 128 input bits and 16 output bits. Usually the 128 input bits are treated as 16 8-bit wide channels and the output is treated as one 16-bit word.

The output of one DSM board can be piped into the input of another DSM board. It is therefore possible to form trees or meshes of DSM boards in order to perform the desired L0 analysis. Each input stage of this DSM board architecture has its own independent input buffer.

There are about 50 DSM boards required for just the Trigger detectors. There will be other DSM boards for the EMC and TOF L0 analysis. All DSM boards run in lock step with the RHIC strobe, which has to be distributed to all of these modules. The cyclic input buffer address counters of all DSM boards must be identical to the appropriate counter in the TCU (Trigger Address counter, refer to Figure 2), in order to allow the readout of the input data or L0 summary data of a particular event.

** The simplest way to accomplish that is to synchronously reset all address counters on all DSM boards and the trigger address counter on the TCU. There has to be a special signal for this clear because it has to be executed on all DSM boards simultaneously within one RHIC strobe (110 ns). Therefore there are two dedicated signals on the VME back plane defined (SerClock and SerData): the RHIC strobe (9 MHz) and the address counter clear signal. There will be many VME crates with DSM boards. A mechanism for distributing these two signals from crate to crate is required. The trigger/clock distribution tree seems ideal for that purpose because it is the standard distribution system for trigger and clock signals throughout STAR. Part of the functional features of this distribution tree is the distribution of the RHIC strobe and the coherent distribution of commands like counter clear. There will be one trigger/clock receiver per VME crate driving the two discussed back-plane signals.

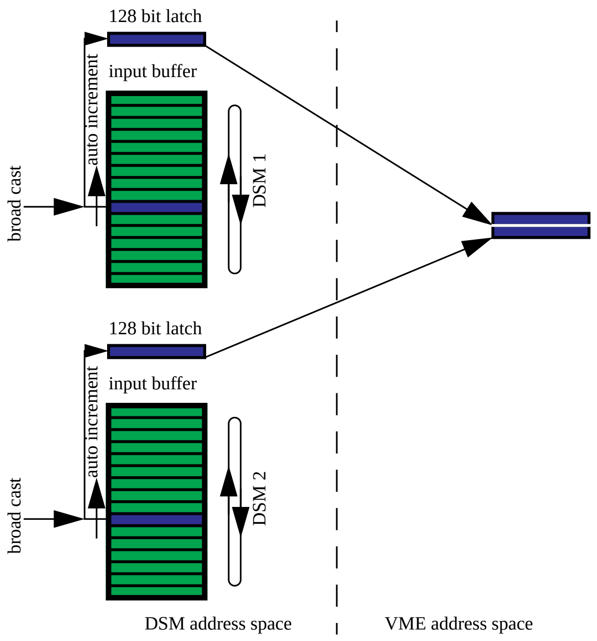
1. Note: There is a side effect to this implementation: the time when an actual clear is distributed to the front-end is undetermined and may be up to 450 µs later (this is a pathological case where one detector is always alive and requests a trigger for each bunch crossing). Therefore the L1CTL CPU cannot return the trigger token directly to the trigger response FiFo if it issues an abort to the trigger response queue. For example if the L1CTL CPU would return the trigger token simultaneously with issuing an abort for the given event there could be a race condition if the trigger system would run in a minimum bias mode issuing triggers every bunch crossing. Then the abort would stall in the trigger response FiFo and the trigger token would be associated with a new event before the old event was aborted.

   Therefore trigger tokens will be returned directly to the trigger token FiFo only if the token is returned by DAQ. The TCU will return the trigger token to the trigger token FiFo by itself if it encounters a level one or two abort command while distributing a trigger response during an empty (no trigger) bunch crossing. This implementation is completely race condition free.

2.2.1 DSM input buffer readout

If the TCU issues a trigger it associates the trigger address with the given event. This address is the appropriate subaddress of all DSM input buffers. For the L1 and L2 readout 16 bytes per DSM board have to be transferred. In order to allow larger and more efficient DMA block transfers a special but VME compatible readout scheme will be implemented. There will be a 128 bit register on each DSM board that can be preloaded with the data of one bunch crossing upon a certain command (refer to Section 2.2.2). This register will occupy exactly four 32-bit words in the VME address range. The long word ordering of these 128 bits will be big ending according to VME. Figure 5 sketches the VME read-out address mapping of the DSM board for two DSM boards. There are of course other disjunct address windows that allow access to all resources on the DSM board for configuration and maintenance.

FIGURE 5. The address mapping of the DSM board for efficient readout



It is therefore possible to program the address decode logic of all DSM boards in one crate to form a contiguous memory space that could be readout with one DMA block transfer. The fact that this memory region is spread out over several VME boards does not matter for the reading DMA engine.

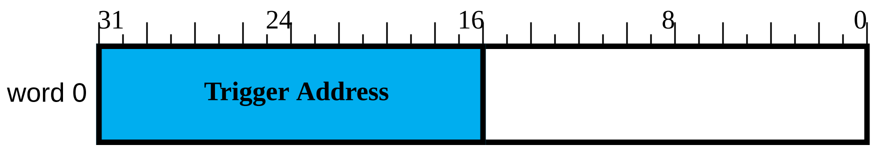
In case of the L2 readout there may have to be up to 10 pre- and post history events readout as well. In order to allow this readout with one simple chain mode DMA transaction the DSM boards perform an auto increment of the trigger address and reload their 128 bit register with the next bunch crossing if the fourth 32-bit word of this register has been read. A L2 readout consists of $N_{pre}+1+N_{post}$ chain mode DMA descriptors. All descriptors have the same VME start address. The transfer byte count is 16 times the number of DSM boards in the crate. The destination address of each chain descriptor is the destination address of the previous descriptor plus the byte count. This implementation allows to readout all DSM boards in one crate including pre- and post history most efficiently with a single DMA command.

2.2.2 Trigger Address Broad Cast

Before the DSM boards can be read out in the way described above all DSM boards have to be programmed to copy the first bunch crossing identified by the trigger address into the 128-bit register. This trigger address is identical for all DSM boards. In order to avoid performing up to 20 times the same transaction in a given VME crate a VME compatible broad cast scheme is implemented. This broad cast commands all DSM boards in the crate to copy the bunch crossing of the specified trigger address into their 128 bit register. Therefore the readout of all DSM boards in a crate consists of two VME transactions: the trigger address broad cast and the actual DMA block transfer (refer also to Figure 5).

The trigger address broad cast can be implemented very simply because all DSM boards run in lock step with the RHIC strobe. There is one slave address in the VME address space that is decoded by all DSM boards. It is a write-only register. Its format is outlined in Figure 6. If a write occurs to that register the appropriate entry of the DSM input buffer is copied into the 128-bit register. The fact that all DSM boards accept the same VME address gives it broad cast functionality. The asynchronous VME command completion signal DTACK is an open collector signal. Therefore the fact that many DSM boards will drive that signal does not matter.

FIGURE 6. The Trigger Address Broadcast (TAB) format)



In the L1 crate the coarse pixel array and the L0 summary information have to be readout simultaneously into the L1 processors. This data is, however, produced at different stages of the L0 DSM tree (refer to Figure 7) and resides therefore at different address slots inside the input buffer of the DSM board. In order to allow the readout of these modules with one trigger address broad cast there is a configuration address that permits decrementing the trigger address on a given DSM board. Therefore it is possible to configure all DSM boards in the whole L0 setup such that the data of a given event resides at exactly the same trigger address independent of where in the L0 tree the DSM board resides.

The input buffer on the DSM board allows also direct VME read/write random access for debugging and monitoring of the trigger system.

2.3 The Trigger/Clock Driver module TCD

The trigger output of the TCU has to be distributed to all (many hundred) readout boards on all detectors. There are also detector specific clock signals that need to be generated and distributed. The trigger/ clock driver modules perform that function. They receive the trigger command from the TCU and broad cast it to all the detector front-ends together with detector specific clock signals. All detector specific logic that needs to be configured will reside on these modules.

3.0 Trigger L1 and L2 Components

All L1 and L2 components are commercially available modules. The functional elements of the L1 and L2 system are summarized in the following list:

- level one controller processor (L1CTL)
- level one I/O processor (L1IO)
- level one trigger processor (L1CPU)
- level two controller processor (L2CTL)
- level two I/O processor (maybe implemented on L2CTL)
- level two trigger processor (L2CPU)
- level two interface for other trigger detectors (L2IF EMC,TOF)
- memory mapped VME-VME bridge (SCIVME)
- fiber optics interface between cave and counting house (SCISCI)
- Trigger-DAQ interface (TDIF)

The L1/L2 trigger processors will run the trigger analysis code. They are single task floating point processing units with a request input queue and a result output queue. They need only a VME slave interface because both the input and output queue reside in the on-board memory and are filled and emptied by the L1/L2 controller processor (refer to Figure 8 and Figure 9). There is no decision of which trigger processor will be used and this decision shall be delayed as far as possible because at this time the processor performance per dollar is still rapidly improving. It may turn out to be a multi CPU/DSP VME hybrid module. On the other hand the trigger analysis code is still growing, too and the decision about the trigger processor type can be done best if the processors in discussion can be compared using the latest trigger analysis code. Because the analysis code is a single thread task written in C it can be easily ported to any processor platform supporting that language.

The L1/L2 controller processors perform a management function. They send the input data to the L1/L2 trigger processors, read the results back and perform the corresponding trigger result dependent functions. There is no requirement for floating point arithmetic on the controller processors. However they have to be able to perform VME chain mode DMA transactions in order to move the data into or out of the L1/L2 trigger processors.

The L1/L2 I/O processors are the I/O servers of all trigger processors that do not have an operating system. It would be useful if these CPUs would have ethernet capability because all commercially available real-time operating systems support the ethernet transport. I/O functions may be blocking. Therefore the controller and I/O functionality was split. Especially in the L1 frame work where on average only 10 μs are available to move one L0 accept into one L1 processor potentially blocking I/O functions cannot be performed in parallel without biasing the performance of the L1 system. In case of L2 it may be possible to run both functions, the control and I/O function, on one physical processor because the time requirements are an order of magnitude relaxed there.

There are other detectors that need to contribute to the trigger veto decision. Their L0 summary data will be available to the trigger L1 veto processor system as indicated in Figure 7. There are 16 bits foreseen for each TOF and EMC. These bits are available as the L0 summary of the trigger detectors for L1 processing.

The TOF and EMC system will also perform their own L1 analysis based on the TOF and EMC raw data. If the result of that analysis shall be considered it has to be available to the L2 system. This is accomplished using shared memories in the L2 crate (L2IF). The external trigger detectors like TOF and EMC have to leave their L1 summary data in these shared memories. The L2CTL processor will copy it then into the L2 processor in order to allow the L2 decision to be derived.

The L1/L2 system will be spread out over several VME crates. Therefore a VME-VME bridge is required that allows transparent memory mapped transactions. The level one part of the trigger system will be in the cave. The level two part will be in the counting house. In order to avoid ground loops all interfaces between the detector and the counting house have to use fiber optics. Therefore a fiber optical interface is required for the data exchange between L1 and L2. It would be most elegant if this fiber optical interface would allow transparent memory mapped VME-VME transactions. In this case it would be completely transparent to the software.

All network requirements discussed are met and well exceeded by the Scalable Coherent Interface (SCI). It is a high-speed network (up to 1 GigaByte/sec) that allows memory mapped transactions between any of its nodes. The physical transport can be fiber optics as well as copper. All components required to base the Trigger L1/L2 network on SCI are available or announced.

The Trigger-DAQ interface controller is a VME processor that performs all handshake functions between Trigger and DAQ. It mocks up a shared memory with some controller functionality. The Trigger-DAQ interface is detailed in the Trigger-DAQ interface specification.

4.0 Trigger Crate and Network Architecture

This chapter discusses the crate layout of the STAR Trigger system. The requirements for the trigger network system connecting these crates in terms of connectivity and bandwidth will be derived from this architecture.

4.1 The Crate Architecture

Figure 7 shows the over all Trigger L0, L1 and L2 crate layout. The cabling of the DSM boards is indicated according to the architecture outlined in Figure 1. One important feature of this implementation is the separation of the DSM boards that are readout for level one and level two analysis. This is possible because the level one trigger processors use the DSM summary only and the level two processors use the raw data of the trigger detectors plus the level one summary. One consequence of this implementation is that the DSM boards need only one bus interface (VME) because there is no circumstance where it has to serve two data flows simultaneously. There are spare slots for DSM boards of other trigger detectors like EMC and TOF.

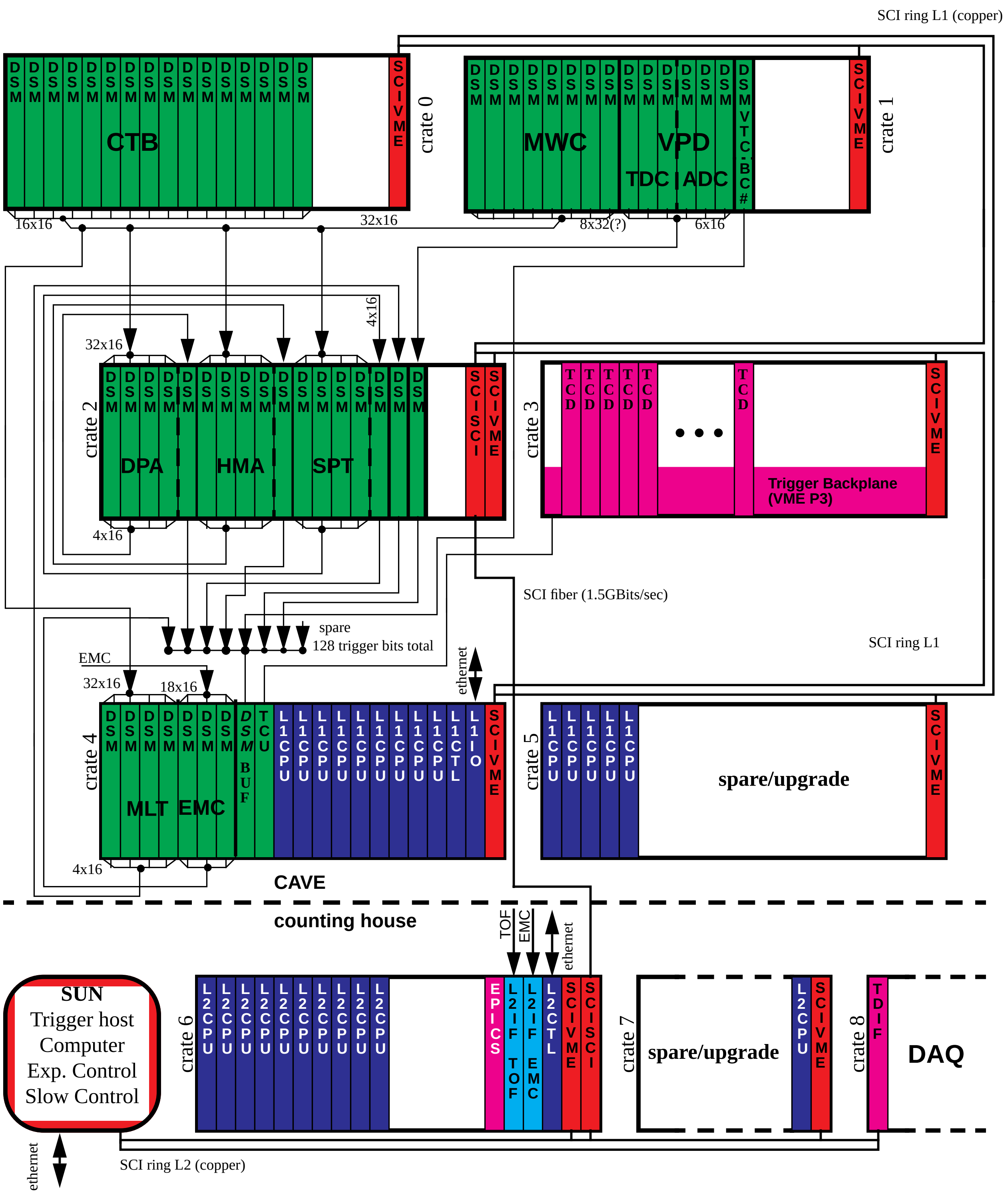
The first two crates (crate 0 and 1) are the DSM front-end. Their input buffers store the trigger detectors raw data. The only transactions in these crates are configuration transactions of the DSM boards and level two readout. There are no level one related transactions performed in these crates.

The next crate layer (crate 2) hosts DSM boards that are in the middle of the DSM tree. There are only configuration transactions required to be performed in this crate. These DSM boards are neither read out by L1 nor L2. It is however possible to configure the trigger system to readout all intermediate stages of the DSM tree for debugging purposes. In this case the total performance of the trigger throughput may be reduced.

The third crate layer (crate 4) hosts the TCU, the L1 processing units and the DSM boards that carry the data used by L1. Due to the high L0 accept rate of 10^5 the L1 processors are in the TCU crate on the detector. There is, however, another VME crate foreseen as an upgrade crate if there are more L1 processors needed than slot space is available in the TCU crate.

There are two groups of DSM boards in the TCU crate. The first group consists of three DSM boards that are logically in the middle of the DSM tree (refer to the DSM cabling indicated in Figure 7). These multiplicity analysis DSM boards (MLT) use as input the coarse pixel array, like the DiPole Analysis (DPA), the Higher Moments Analysis (HMA) and the Special Topology Analysis (STA). The coarse pixel array, however, is also required for the L1 analysis. Therefore the MLT DSM boards are in the TCU crate allowing the L1CTL CPU to read the coarse pixel array out of their input buffers.

FIGURE 7. The overall Trigger crate architecture



The fourth DSM board in the TCU crate does not function as a DSM board. Only its input buffer is used. It stores the summary of the six different L0 DSM analysis chains and possibly the summary of the L0 EMC and TOF analysis (16 bits each). The FPGA and all related hardware on the DSM board is not used on this particular board (labelled BUF in Figure 7). The advantage of this implementation is that there is no need for reading the FPGA output of any DSM board. Therefore there is no FPGA output buffer required on the DSM board. The L1 readout becomes extremely simple because it is like the L2 readout just a block transfer out of four DSM input buffers (128 bytes)[1].

The trigger/clock distribution crate (crate 3) hosts the trigger clock distribution drivers (TCD). These TCD boards receive the trigger information from the TCU and return 8 detector busy signals in case of performing a test pulse sequence (please refer to the trigger/clock distribution tree requirement document).

The L2 hardware is in the counting house. Therefore there is a fiber optical interface required connecting the L0 and L1 hardware on the detector with the L2 crates. This is indicated as independent VME modules (SCISCI). They may be, however, implemented as simple copper-fiber signal converters. The HP Gigalink chip set (HDMP 1011/1012) for example has a special SCI mode supporting this scenario. The big advantage of this implementation is that all resources in the trigger system are accessible through memory mapped transactions. Because it is possible to connect a workstation to the SCI network as indicated in Figure 7, all initialization, control and monitoring functions can be performed directly from the workstation based trigger host.

The L2 crates (crate 6 and 7) contain the L2 controller processor, the TOF and EMC L2 interface and the L2 trigger processors. There is also a spare or upgrade VME crate foreseen for the case that 16 L2 processors are not sufficient to perform the required analysis task in the given time budget.

The Trigger-DAQ interface will reside in the DAQ event-builder crate. It would be ideal if that VME processor would be connected to the SCI L2 ring allowing memory mapped transactions between the Trigger-DAQ interface and the trigger system.

4.2 The L1/L2 Network Requirements

The network requirements of the STAR Trigger system based on the crate layout sketched in Figure 7 are summarized in Table 3. All data rates are assumed in mega bytes per second. They are based on a L0 accept rate of 10^5/sec, a L1 accept rate of 10^4/sec (10^3/sec for TPC events) and a L2 accept rate of 10^3/sec (10^2/sec for TPC events). Flow control and monitoring traffic is not included and has to be added on top of these numbers. It is however only a small contribution.

The L1 readout rate inside crate 4 is twice the rate required for the principal data transmission (128 bytes times the maximum transmission time of 10 μs) because the actual data transfer will be performed by the L1CTL CPU[2]. 35 MB/sec are achievable in the VME framework.

1. Note: An independent FPGA output buffer would bare many implementation problems. For example because it is only 16 bits wide. There would have to be at least two L1 readout transactions: one for the coarse pixel array (DSM input buffers) and one for the output buffers of all other DSM boards. There would also be a lot more slot space required in the TCU crate because every DSM board forming the end of an analysis chain would have to be physically in this crate.

This problem does not arise in case of the L2 readout because the L2CTL processor will program the DMA engine on the SCI-VME interface in all crates to directly push the raw data in crate number 0 and 1 into the appropriate L2 CPU.

For the readout data rate of accepted events into the Trigger-DAQ interface the following considerations led to a data rate of about 3MB/sec: The maximum pre and post history for fast detectors like EMC and TOF is one. The maximum pre and post history is 10 for events involving TPC and SVT. Their L2 accept rate is, however, limited to 10^2 per second. Therefore the maximum readout rate for the Trigger-DAQ interface based on 900 EMC/TOF events and 100 TPC/SVT events per second using their appropriate maximum pre and post history is about 3MB/sec. Because the number of DSM boards is roughly equally split over the trigger crates 0 and 1, the corresponding readout rate there is 1.5 MB/sec. The L2 summary that has to be readout into the Trigger-DAQ interface adds no significant data rate requirement for the L2 trigger network.

TABLE 3. The network and crate data rate requirements in MB/sec of the STAR L1 and L2 trigger system

crate	data rate in crate	network data rate	comments
0	2.6+1.5	2.6+1.5	L2 readout plus readout upon L2 accept
1	2.5+1.5	2.5+1.5	L2 readout plus readout upon L2 accept
2	0	0	initialization and monitoring only
3	0	0	initialization and monitoring only
4	2x13+6.2	13+6.2	L1 readout (2x13) plus L2 readout (6.1)
5	13+6.2	13+6.2	L1 readout (6.4) plus L2 readout (6.1)
6	2.6+2.5+6.2	2.6+2.5+6.2	L2 readout of crate 0 and 1 plus L1 summary
7	2.6+2.5+6.2	2.6+2.5+6.2	L2 readout of crate 0 and 1 plus L1 summary
8	n.a.	3+1.2	including maximum pre and post history
		8.1+6.2+13	total cumulative L1 network rate
		8.1+6.2	total cumulative L2 network rate

These estimates are based on the assumption that the pre and post history data will be readout only after a L2 accept.

The data size of the TOF and EMC summary data in the dual ported memories in crate 6 is not specified yet. However if it would be 128 bytes each, which is considered a large number, it would contribute only 1.2 MB/sec each.

The SCI hardware proposed to be used for the L1 and L2 network supports a net network speed of up to 100 MB/sec. This is well above any of the requirements listed in Table 3. The two networks would not even have to be separated. For maintenance reasons it may be a good idea to decouple these two SCI

2. Note: This doubles effectively the VME load because each word has to be read from the DSM board by the DMA engine in the L1CTL VME interface and then be written to the L1CPU. If the L1CPU would perform the DMA by itself the load of the VME backplane would be just 10 MB/sec for the L1 readout. However this scenario has the burden that the L1CTL CPU has to program the DMA engine on the L1 processors. There would have to be a DMA engine implemented on the L1 processors, and they would have to be VME masters. The L1CTL software would depend highly on the hardware of the L1 processor.

ringlets with a bridge as indicated in Figure 7 by the SCI-SCI VME module. This module could also serve as interface between the copper based L1 and L2 ringlets and the fiber link between the detector and the counting house. Decupling the L1 and L2 ringlets will also greatly reduce the latency in any of the rings.

5.0 L1 and L2 Software Architecture

This chapter describes the software architecture of the Trigger L1 and L2 veto system. The major functionality is that there has to be a frame work that allows a simple implementation of a L1 and L2 analysis code. The actual analysis code is outside the context of this document. The framework has to be flexible enough to allow the implementation of this analysis code into the L1 and L2 framework. It has also to be robust enough to react gracefully on malfunctioning or even crashing L1 analysis routines, because they may change frequently and may not be fully debugged. Figure 8 shows the over all software architecture and data flow of the trigger L1 and L2 system.

The L1 and L2 software framework will be completely written in C. From first principles there is no need to constrain the language of the trigger analysis code. However maintenance and debugging considerations strongly suggest to also use C or C++.

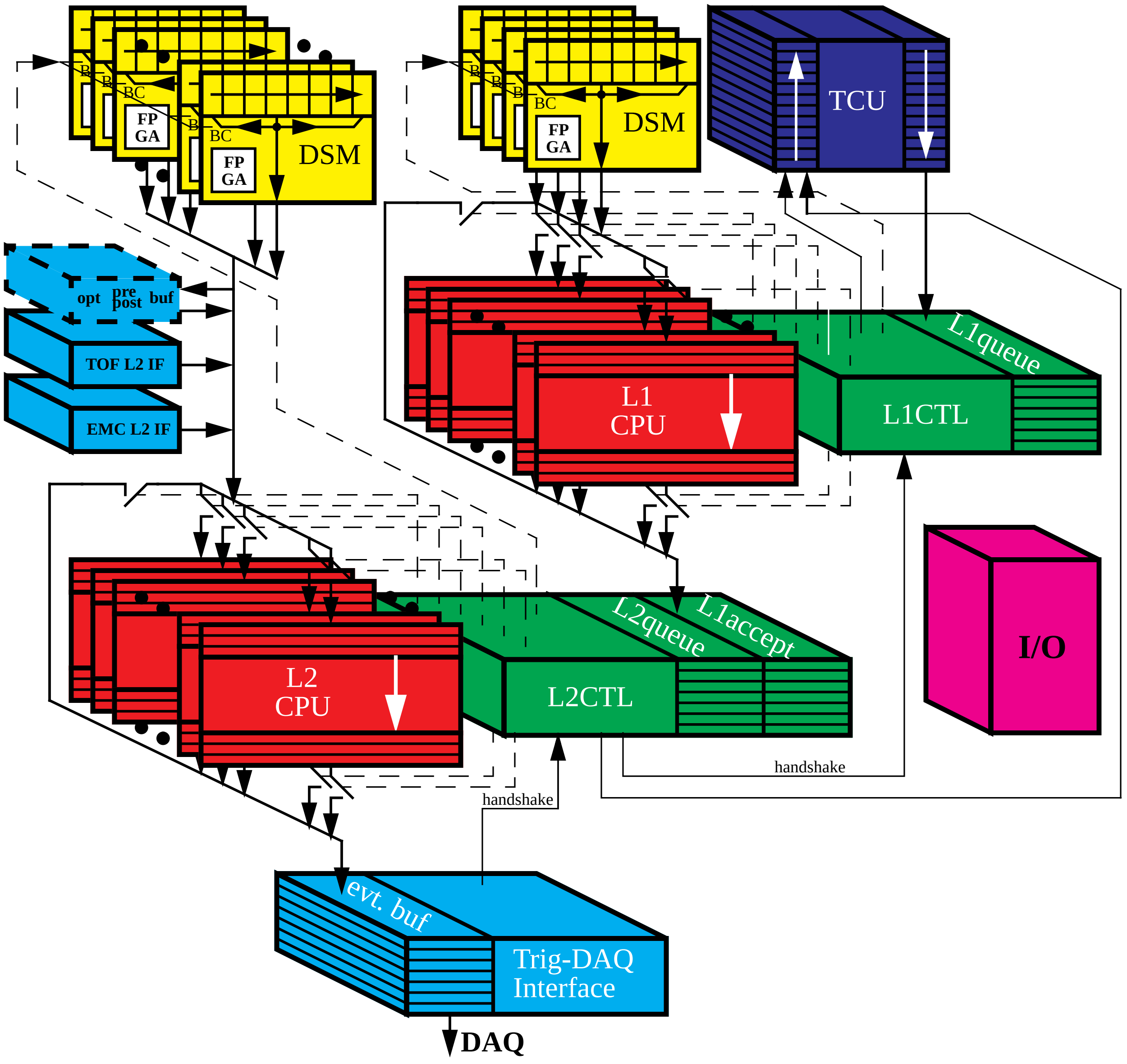
The processor architecture of the trigger processors is not defined yet. An early version of the trigger processors may even be completely different from those being finally used. Therefore it is very important that all L1 and L2 code is written in a portable way. This rules out any assembly parts. All input and output data structures have to be defined architecture independent. This is especially important taking into account that the trigger analysis code targeted for a big ending machine may have to be debugged on a little ending system. Therefore all shared data structures have to defined platform independent. Data types like `int` cannot be used because they imply the byte ordering type of the target system.

All shared data types will therefore be defined using the shared data formats defined in the approved IEEE standard 1596.5. Appropriate include files will translate these platform independent data types into the appropriate platform specific native data types or structures.

One important general feature of the L2 architecture is that one event is analyzed by exactly one processor at a time. There will not be any splitting of the analysis over many processors like in the L3 setup. This implementation is possible due to the relatively small size of data items used here. Therefore unlike in case of the L3 analysis no assumptions had to be made at all for the actual analysis code. All data will be available in the local memory to the analysis code.

Depending on the very constrained time budget in L1, it may be required to constrain a particular algorithm to one processor. This scenario would allow to keep the essential part of the analysis code in the cache avoiding time consuming instruction cache refills. In this case the L1CTL processor would have to schedule events depending on their trigger type to the L1CPU running the appropriate analysis code. However as first approach the L1CTL processor will schedule the L1CPUs the same way as in the L2 framework, strictly in a round robin mode with the trigger processor ID running fastest.

FIGURE 8. The L1 and L2 data flow and architecture



The only constraint on the analysis code is the limited analysis time. This implementation does not require the analysis code to finish in any case within the given time limit. On average, however, it has to perform the task within that time limit. This allows a very good utilization of the processing power because the code could be optimized for the typical case and not for the worst case and rare case. Therefore there are two kinds of time parameters associated with the trigger analysis: the typical analysis time t_{L1}/t_{L2} and the maximum analysis time t_{L1max}/t_{L2max}. There has to be a maximum analysis time (typically 3 times the typical time) in order to detect malfunctioning analysis code. If the trigger controller does not encounter a result after t_{max} it will halt the trigger processor, produce a core dump and reboot it. The event will be considered as accepted, however with this error condition flagged.

This implementation has as consequence that L1 and L2 accepts may occur at random times after the actual bunch crossing. In case of L1 accepts, however, the L1CTL processor will ensure that accepts are going to be sent in the order of the events.

5.1 The L1 and L2 Trigger Processors

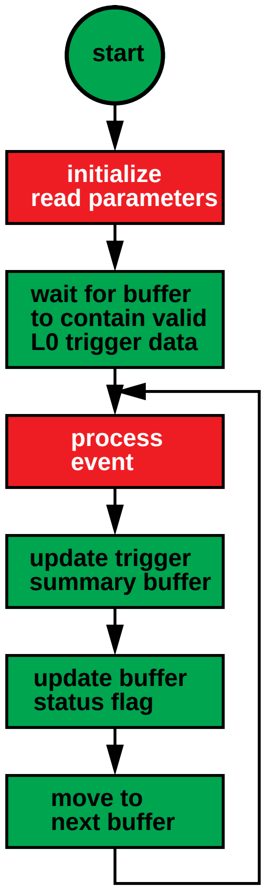
The architecture of the level one and level two processors is very similar. It is a very simple single threaded implementation as sketched in Figure 9. The actual analysis (*process event*) where most of the time is spent is linked to the trigger processor framework software, which performs all other functions indicated in Figure 9.

There is an input and output queue that allows the trigger controller processor to prepare the next event for processing or readout the summary of the previous event in case of an accept, while another event is currently being analyzed. The data will be transferred to or from the local memory of the trigger processor. If the trigger code is not completely I/O bound, which is highly improbable due to the small data size compared to typical cache sizes of several kilobytes, this will not greatly affect the performance of the trigger processor. In any case the performance burden due to the data transmission cannot exceed the fraction of the total analysis time per event needed for the data transmission, which is 10% assuming 10 analysis processors.

The depth of the in- and output queue of the trigger processor (L1CPU, L2CPU) cannot be arbitrarily deep because it adds to the latency of the event processing.

For details of the environment of the trigger processors refer to Section A.0.

FIGURE 9. The program flow of the L1 and L2 trigger processors



5.2 L1 Architecture

The L1 data flow is controlled by the L1CTL CPU. It has an internal queue that forms the level one queue. The L1CTL processor reads trigger requests from the TCU and copies the appropriate raw data into an input buffer of the next level one processor.

Upon a reject of the event the L1CTL CPU will post an appropriate command in the trigger response FiFo (refer to Section 2.1.3). If the event was accepted the computed result and the appropriate raw data including the configured pre and post history for that event type has to be copied to the level two veto system for further analysis.

The interface between the L1 and L2 system is a L1 push architecture. There is a predefined buffer with a known size that is implemented as a cyclic buffer. The first two words of each variable length event in that L1 accept queue (refer to Figure 8) will be the 64 bit event descriptor structure, which was read from the TCU in the first place and defines the detectors involved and therefore the number of pre and post events. The L1 summary data and the appropriate data for N_{pre}+N_{post}+1 bunch crossings follows. The L1CTL CPU will have a reserved memory location in its local shared memory, where the L2CTL CPU will place the latest L2 read pointer of the L1 accept queue. This simple handshake prevents the L1 controller from overrunning not yet processed events in the L1 accept queue. However the buffer space available for the L1 accept queue can be made large enough that this handshake mode will be required only in pathological cases. Under normal running conditions the L1 controller will never see a full L1 accept queue.

5.3 L2 Architecture

Unlike the L1 controller the L2 controller is passive. It does not readout the TCU or level one controller but gets requests posted in its internal local memory (L1 accept queue in Figure 8). These variable length requests contain the summary of the L1 computation and the L0 raw data including the number of pre and post history bunch crossings. The first 64 bits of these structures is the trigger type read from the trigger type FiFo (Section 2.1.1) by the L1CTL processor. It identifies the event type and its length. The L2CTL CPU will poll on the L1 accept queue in its internal memory. If there is a new event it will identify the next second level processor and setup the DMA controllers in the remote DSM crates (crate 0 and 1 in Figure 8) to move the appropriate bunch crossing directly into the local memory of that trigger processor. There will be very little DMA setup required because due to the very limited number of DMA targets it is possible to setup a DMA descriptor for every possible case during the initialization phase. The DMA request will therefore be initiated by just pointing it to the appropriate preset descriptor chain.

After a L2 trigger processor finishes the L2CTL processor will post either a L2 accept or an abort command at the trigger response FiFo (Section 2.1.3). In case of an accept the whole event including history will be copied to the Trigger-DAQ interface. The handshake between the Trigger-DAQ interface and the L2CTL CPU is similar to the handshake between L1 and L2. It is also a push architecture with a limit address specifying the upper buffer address in the cyclic buffer structure.

5.3.1 EMC and TOF L2 Input

There are two scenarios to incorporate fast external detectors like EMC and TOF in the L2 veto decision. Both use the same physical data path a shared memory as indicated in crate 6 in Figure 7.

1. supply detector specific data no later than t_{L1ext} after the bunch crossing for L2 processing.
2. supply detector specific L1 summary data no later than t_{L2ext} after the bunch crossing for the L2 decision.

Generally it is possible to feed data into the Trigger L2 system either at the end of L1 or shortly before the final L2 decision. It is required to aggree on a maximum time depending on the scenario for the external trigger detectors to supply the appropriate data.

The architectural difference is that in scenario one the data of the appropriate trigger detector has to be supplied within a very short time frame of t_{L1ext}. The advantage of this scenario is that this data is available for the L2 decision.

The second scenario gives a longer time period until the result has to be posted in the shared memory. However there is no time available to perform any analysis combined. The L2 result is restricted to a 4-bit entity. The L2CTL CPU combines this 4-bit result from the TOF, EMC and L2processor and use it as 12-bit address into a lookup table, that specifies if the event is accepted or rejected.

Which scenario will be used may depend on the trigger command word (refer to Figure 2). Therefore it is possible for example to implement TPC coincident events as scenario 1 and EMC/TOF high rate events as scenario 2.

In any case the TOF and EMC result has to be identified by the trigger token. The TOF and EMC shared memories will be organized as 4095 fixed length entries. This allows the L2CTL CPU to directly access the result of a given event by just using the trigger token as index into this memory. After processing the result the appropriate entry in the shared memories will be marked invalid by the L2CTL processor.

The time-outs (t_{L1ext}, t_{L2ext}) depend on the L1 and L2 pipeline depth. The specified numbers were derived from the assumption that there will be 10 L1 and 10 L2 processors defining a 100 μs L1 pipeline depth and a 1 ms L2 pipeline depth.

The level two trigger system is strictly non blocking. If the appropriate result is not available at the appropriate time the L1CTL processor will assume a default, that is a configuration parameter. In this case it will also increment local state counters that allow to identify these race conditions. Upon a L2 accept the event will be marked as a *non complete* L2 accept.

5.4 Initialization of the Trigger System

The initialization and configuration of the trigger system works similar to the initialization scheme used for the trigger processors. There is, however, just one read only ASCII file for the whole trigger L1/L2 frame work system including the Trigger-DAQ interface, that contains all the configuration information. The format of the file is identical to the trigger processor configuration file described in Section 1.1.3. This file will be like all other Trigger related files on the Trigger host computer and be accessible to all processors in the Trigger system.

The processors in the Trigger system that have their own operating system will mount the appropriate NFS file system and execute the appropriate configuration program during their start-up phase. For all other processors there is a configuration and bootstrap program running on the Trigger host Computer that performs all initialization steps specified in the Trigger configuration file.

A.0 Appendix

This chapter describes the implementation details of the trigger processors.

A.1 Trigger Analysis Code Debugging

Due to the limited time budget the trigger processors will not have an operating system or kernel. In-situ debugging will be restricted especially because the decision about the physical trigger processor type shall be deferred as long as possible. There may be an early version of the trigger processor that will be replaced later by a completely different processor architecture.

Given these restrictions the following two debugging mechanisms will be supplied:

- in-situ debugging using a debugger (like `gdb`) running on host platform
- off-line debugging environment on host platform

The first scenario can be implemented only after there was a decision about the trigger processor architecture. It allows, however, to debug analysis code even while the whole experiment is running. The trigger controller may be programmed to treat a given trigger processor special for debugging. In this mode it will feed it with input data but ignore the result and also not apply any analysis time constraints.

The second scenario is most important especially for software development. It may run on any platform completely independent of the actual trigger system. This debugging mode may be very simply enabled by setting the appropriate compiler flags. In this mode the management of the shared memory structures (refer to Figure 10) will be mocked up by the supplied software framework. The trigger controller functionality is part of this framework, too. It allows reading the input data from a file, which may contain actual raw data or simulated data. The appropriate analysis output is saved in this mode to another file.

A.2 I/O Support on Trigger Processors

The trigger processors will not have any I/O capabilities. During normal operation I/O has to be avoided because it may incur indeterministic latencies. However for debugging and error handling as well as initialization I/O is required. Therefore a subset of the C run-time library stdio will be supported on the trigger processors. It will be implemented as a shared memory based remote procedure call scenario. The actual I/O functions will be executed by an individual server process running on the trigger I/O processor (L1) or trigger controller (L2). The I/O functions supported are summarized in Table 4.

There are two categories of I/O functions in this table: message support and file I/O support. File I/O is supported through the buffered I/O *stdio* functions listed in Table 4. Both binary I/O functions (fread, fwrite) and string I/O (fgets, fputs) functions are supported. All other I/O functions like for example fprintf or fscanf can be emulated using string handling functions like sprintf and sscanf. There will be one I/O server process per trigger processor. Therefore potential multiple requests to the same file are being automatically taken care of by the lock management of the operating system. The I/O server process itself can be a very simple single threaded task. There will be two user identification codes (UIC): one for L1 I/O processes and one for L2 I/O processes. All I/O server processes of one kind will use the same UIC. The files and subdirectories accessible to the L1 and L2 analysis processors will be separated but part of the same NFS file system.

TABLE 4. The I/O functions supported on the L1/L2 trigger analysis processors

code	I/O function	comments
0	logmessage	generic debug/log/error message function
1	fopen	file open on I/O processor
2	fclose	file close
3	fread	raw binary read from open file
4	fwrite	raw binary write to open file
5	fgets	get one line from open file
6	fputs	write one line to open file
7	fflush	flush all buffers
8	fseek	reposition file pointer within open file
9	ftell	report file position in open file
10	feof	report end-of-file state of open file

Message support is implemented with one generic function:

```
logmessage (int severity_level, char * string)
```

This procedure will be used by all programs also on the trigger I/O processors and trigger controller processors. The way log messages are handled depends on the framework the program runs in. For example in case the code is debugged in off-line mode they will be just printed to `stderr`. During normal running conditions all logmessage requests will be posted to a global log message file together with a time stamp and an identification of the calling process. There will be also an interface allowing to forward these messages to the STAR experiment control and slow control systems. This interface, however, may implement additional filters for example to select only forwarding of error messages.

There are five severity levels supported as listed in Table 5.

TABLE 5. The severity levels supported by `logmessage`

code	level	comments
1	debug	debug messages
2	information	standard log messages
3	warning	warnings
4	error	non fatal error conditions
5	fatal error	fatal error conditions

`logmessage` does not support formatted printing with arguments like `printf`. The reason for this is the fact that the argument type of `printf` like calls depends on the argument string. Therefore these calls cannot be forwarded directly to remote platforms like in case of the RPC based I/O on the trigger processors. For example a `printf (“%i”, 123)` call issued on a big ending processor and forwarded to a little ending system would result in wrong results. Therefore the argument processing has to be performed on the client side. This, however, can be extremely simply accomplished using sprintf:

```
logmessage (severity_level,
            sprintf (buffer, string, arg. 1, arg. 2, ..., arg. n))
```

The above call of `logmessage` mocks up a `logmessage` call with arguments using `sprintf` as the translation primitive.

A.3 Shared Data Structures

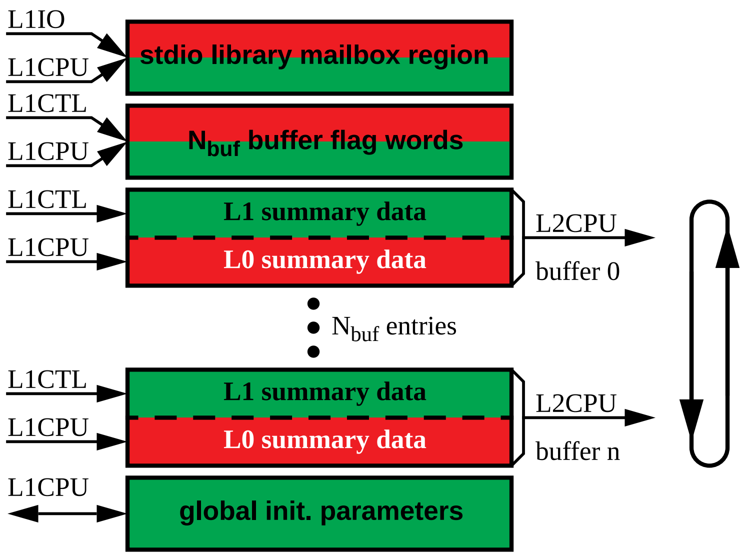
The various shared data structures required for the trigger processor to interface to the trigger controller and I/O processor are sketched in Figure 10 for the example of the level one system. There are three major groups:

- the stdio library mailbox
- the input and output buffers with their associated buffer flag fields
- the global initialization parameter buffer

The *stdio* library mailbox is used to implement *remote procedure call* based I/O functions on the trigger processors that do not have any I/O capabilities by themselves (refer to Section A.1.1).

The input and output buffers form the request and response queue. They are adjacent as shown in Figure 10 for the example of level one because this allows to read them both with one burst transaction. The input buffer part of that block is filled by the controller processor and the summary part is filled by the trigger processor upon completion of the analysis.

FIGURE 10. The trigger processor shared data structures of for example the L1 processor



The trigger controller will not perform any reformatting of the trigger data. Therefore the data posted in the L1/L2 input buffers (L0 summary data and DSM input buffers) will be a direct copy of the appropriate memory regions identifying the data format of these buffers. The format of the L1/L2 result fields, however, depends on the analysis code and is outside the scope of this document.

The state of every buffer is described by one 32-bit flag word. All flag words are adjacent in memory space allowing to burst read them all by the trigger controller processor. The definition of these flag words is defined in Table 6. The basic handshake is that the buffers are initialized and setup by the trigger controller and the various result states are posted by the trigger processor. The trigger processor will accept a buffer for processing only if the associated flag is 1. In that case it will rewrite it with two indicating that it started to take care of it. This state is useful to find out which event is currently being processed and which event caused a potential crash of the trigger processor. There may be only one buffer in state 2. States 3 through 5 define the analysis completed and return the buffer with a valid result field to the responsibility of the trigger controller.

TABLE 6. The different entries of the buffer state variable

cmd	which CPU	comments
0	trig. controller	buffer unused - state after initialization or readout of result
1	trig. controller	event posted for processing by trigger controller
2	trig. processor	event accepted for processing by trigger processor
3	trig. processor	event accepted, summary data written
4	trig. processor	event rejected, summary data written
5	trig. processor	error - trigger processor stopped

Under normal running conditions the trigger controller may write to the buffer flag only if its state is 0, 3, 4, 5. The trigger processor may correspondingly write to the buffer flag only if its state is 1 or 2.

The buffer flag words define the state of all buffers on a given trigger processor. All buffers are used in a cyclic order starting at buffer number 0. The trigger controller sets up buffers for processing and the trigger processor will process them strictly in the order received. Because the buffer flags are in the local memory of the trigger processor it may perform polling on these flags without affecting the VME bandwidth. This buffer handshake architecture does not require the trigger processor to have a VME master interface at all.

All global parameters of the trigger processor are accessible through the data field shown in Figure 10. It is split into two parts: one defines the parameters of the trigger processors shell functions like the number of buffers in the input and output queue; the second is reserved for parameters of the L1/L2 analysis code (refer to Section 1.1.3).

A.4 Initialization - Configuration Files

The L1/L2 trigger analysis code requires parameters that have to be downloaded before its execution. These parameters will contain an identifier of the actual analysis code and will be written to tape as a special event upon begin of a run. They may also be maintained in slow control data bases. Finally there has to be a mechanism to check these parameters at any time by the experiment control system. The detailed format of these parameter fields depends on the trigger analysis code and cannot be defined here. However the mechanism that initializes the global parameter buffers has to be defined in order to build the system.

The mechanism proposed for initialization of the global parameter buffers in the trigger processors is flexible. As indicated in Figure 9 the global parameter buffer is initialized by reading in the appropriate parameter. This is accomplished using the available C `stdio` functions described in Section A.1.1. An analysis code specific function will be called during the initialization phase. This initialization function that is linked together with the analysis code will read all required parameters from a specific file and setup the global parameter buffer shown in Section 10.

There are no large data bases required within the Trigger L1/L2 frame work. Therefore all parameter files will be ASCII. No special tools are required to setup a specific configuration. The configuration file name is an argument passed to the trigger processor code allowing many different configuration files to be installed and to select them on a run by run basis.

The format of these configuration files is extremely simple allowing a simple decode routine. All lines of the configuration file have the same form:

```
<command> <command dependent arguments>
```

Arguments and command words are separated by white spaces (space, TAB). Leading white spaces are ignored. Commands are case sensitive. A command word may have any length between 1 and 31 characters. Longer commands are truncated. One command word (#) is reserved for comments. Therefore comment lines have to start with a # command. There are no in-line comments supported. The type and number of arguments depends on the command. These files can also be created automatically from calibration and initialization databases with other software. However, they provide a simple mechanism to interface to the trigger software and allow the development of this software without the knowledge of the detailed architecture of the slow-control and experiment control system.

FIGURE 11. A sample configuration file

```
# This is the sample configuration file for trigger L1 analysis xyz
# The output of the analysis are: ....
buffer_size 1024
mixed_argument1 string 123 29.45
calibration_parameter 23.5 45.7 34.2 34.5
# end of configuration file
```

This format is very simple to decode. A sample configuration file read routine will be available as reference.

A.5 The L1 and L2 Analysis Code Call Frame

The supported language for analysis and initialization routines of the L1 and L2 processes is C. There are two functions that have to be written for each trigger analysis layer (the question mark stands for 1 or 2 specifying the veto processing layer):

```
• int L?_read_parameter  (void *parameter_buffer,
                          char *parameter_file_path)
• int L?_analyze         (void *parameter_buffer,
                          struct L?_data_fmt *data_buffer)
```

The function L?_read_parameter reads and interprets the parameter file specified as second argument. The first argument points to the global buffer storing these parameters for later use by the analysis routine. This pointer has to be a void pointer because the format of this buffer depends on the analysis routine and therefore cannot be specified. A non zero return value is considered as an error. The appropriate error messages are assumed to be already posted using `logmessage` (Section A.1.1). The trigger processor code will not assume normal operation if this function fails.

The analysis routine `L?_analyze` has a pointer to the parameter buffer and a pointer to the data buffer, which holds both the input data and the result update produced by this routine. The return value of this function is, according to Table 6, either 3, 4 or 5. Any other response is considered a fatal error.