Technical Design Report for the Level 2 Global Processor

Daniel Edmunds, Steven Gross, Philippe Laurens
James Linnemann, Roger Moore
Michigan State University

D0Note #3402

Version 1.04

February 25, 1998

1 Organization of this document

We first give a functional description of the Level 2 Global Processor. Then we evaluate the engineering performance of the system. Finally we discuss issues of control, monitoring, testing, downloading of code and scripts. The Level 2 web pages[1] provide more detailed documentation on many of the topics summarized here.

The discussion in this document is aimed at a description of L2 Global. The L2 Global Processor is fed data by several Preprocessors. Because the preprocessors will use the same hardware and software, the description is also useful as an overview of data movement in preprocessors.

This report describes our current understanding of the system, and demonstrates that we have workable technical solutions for all the data movement, hardware, and, online software issues which are likely to affect design of the hardware components of the system. The filtering software and the framework to steer it are not explored in detail in this document.

1.1 Conventions

In this document, B indicates a Byte, while b indicates a bit. L1, L2, and L3 indicate the Level 1, Level 2, and Level 3 trigger. HWFW refers to the HardWare FrameWork of L1 or L2. Bandwidth is normally quoted in MB/s. It is worth noting that 100 MB/s is the same as 100B/μs. Data acquisition is abbreviated by DAQ.

2 Requirements

The fundamental design requirements[2, 3, 4] of the L2 Global Processor are to handle a 10 KHz input rate, provide a rejection factor of approximately 10, with signal efficiency no lower than Run I L3, introduce at most 5% deadtime, and provide information to the L3 trigger to assist the software filter decisions. The system should be reliable, well understood, and not require excessive resources to develop or operate. VME Buffer/Driver (VBD) cards perform the readout to L3. L2 Global must announce its decisions to the L2HWFW[5] in the order in which it receives events from the L1 system[6]. The decisions are in the form of a 128 bit mask, deciding pass or fail for any of the 128 L1 decision bits passed by L1.

3 Overview of L2

3.1 Architecture

The L2 trigger, like L1, consists of a hardware framework and a separate set of processors. The framework notifies L3 and DAQ about pass/fail decisions on events, and reads out scalers. The L1HWFW also sends information to the L2 processors to assist in their decision making, generates relevant trigger Qualifiers (such as *Unbiased Sample*), and the L2HWFW receives results from the L2 Global Processor. The L1HWFW does the combinatorial logic to make the final L1 decision, but the final L2 trigger decision is performed in the L2 Global Processor.

The architecture for the L2 processor shown in Figure 1 can be thought of as a stochastic pipeline. The 10 KHz input rate gives each stage a nominal budget of 100 μs. Simulations give low deadtimes for processing times in the 50-75 μs range for each stage (processing plus output formatting combined). The requirement is stricter for earlier, parallel stages of processing, since the later stages may be delayed if any of the stages are slow. Preprocessors handle (in parallel) data specific to particular detectors and pass lists of objects found to the Global Processor. It is critical that the preprocessors not be restricted to event-synchronous operation, lest the processing time distribution in the preprocessors become a long-tailed "worst of n" distribution. Further, deadtime can increase significantly if the decision time varies significantly from event to event. Tails

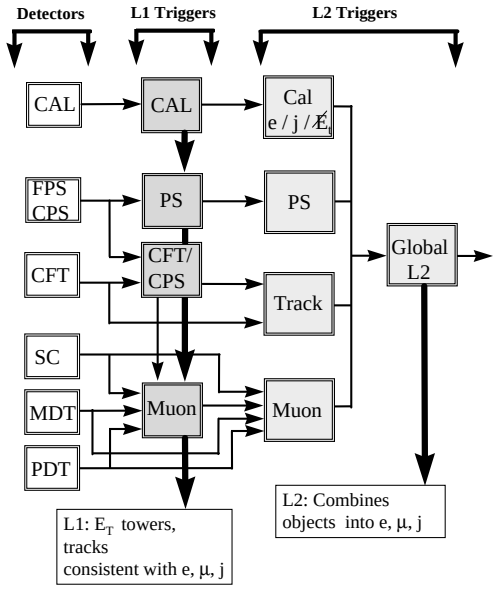

Figure 1: L1 and L2 trigger elements. The horizontal arrows denote information flow.

longer than exponential in the decision time distribution decrease the extent to which the stochastic pipeline imitates a true pipeline. To enhance performance, the processors allow new events to flow into buffers while processing takes place.

3.2 L2 Global Processor

The Global Processor makes trigger decisions based on the objects found by the preprocessors. It may require higher quality of objects found by the preprocessors, make matches between objects found in different preprocessors, or calculate kinematic variables from multiple objects. Such decisions are tailored separately for each L1 trigger bit which fired: each L1 bit which passed L1 is either confirmed or rejected, resulting in a 128-bit L2 trigger bit mask. The event passes L2 if any L2 bit passes. The decision is reported to the L2 hardware framework. The L2 decision directs final DAQ, steers data to L3 nodes, and guides L3 triggering.

The Global Processor will be implemented by a fast Alpha processor on a VME card; the hardware is the same as that used by CDF for its Global Processor, and DØ and CDF have collaborated on the specification of the system. The difficulty of the task facing the Global Processor depends on the amount of data, and the complexity of processing needed. We currently estimate .9KB of input data. The architecture guarantees that the most complex processing is localized in the preprocessors, so processing of order 100 K instructions (100 instructions per byte) seems adequate. Run I L3 required roughly 2M instructions per event, but this figure includes work done by preprocessors in the Run II L2 system. The Digital Alpha 21164 chip running at 500 MHz can perform up to 4 instruction per clock cycle, giving a potential of 200K instructions in the nominal time budget. The proposed processor passes these plausibility checks, and simulations with sample code confirm that the processing power an Alpha gives is adequate. More processors can be added if need be.

3.3 Overview of L2 Global Operation

The L2 trigger occupies a 64-bit[1] VME for Physics crate 9 U high equipped with an auxiliary Magic Bus [7] backplane. Table 1 shows the cards in the crate, and Figure 2 shows how we use the hardware. We will use this basic architecture (Worker and Administrator Alpha, with Magic Bus Transceiver (MBT) for input, VBD for output, and Multiport Memories (MPM) for other I/O) to handle data movement in all the L2 preprocessor crates as well as in the Global crate.

The Magic bus (MBus) is a 128 bit (16 Byte) wide data bus used for communication between modules in the L2 Trigger crate. The MBus is mounted on the P3 backplane of the VME crate. The bus has 32 bit addresses; among these are 8 bits of addresses which broadcast to all MBus modules. The bus can perform a cycle every 50 ns[2], giving a nominal throughput of 320MB/s. Since MBus is a synchronous bus, this rate can be sustained. MBus has been designed to be faster than the internal PCI bus of the Alpha processors, so that MBus will not limit data input speed. The attraction of the auxiliary bus it to allow data input at higher bandwidth than VME can support. The VME bus is used for readout to L3 and for general control functions[3]. The existence of the two buses allows flexibility in implementing other communication needs.

The Magic Bus Transceiver (MBT) cards accept the preprocessor inputs. Data are broadcast via MBus to both an Administrator node to handle housekeeping tasks and a Worker node to make the actual trigger decision. The multiport memory is used as a link to the trigger control computer (TCC). TCC handles run control, downloads run-specific information, and collects monitoring information via the multiport memory, which can also serve as a VME crate controller. The MBT card also sends decisions and monitoring information to the L2HWFW.

We find the coding more straightforward with this division of responsibility between Administrator and Worker, and it allows the most straightforward expansion of the number of Worker nodes if required for L2

[1] The 64 bit data width only applies during block transfers, and is not used for any functions in the L2 Global crate.

[2] To be verified during test of first prototypes

[3] The Universe chip used on Alpha VME cards does not support the VSB32 (VMX) bus, so this is not feasible mechanism for communication.

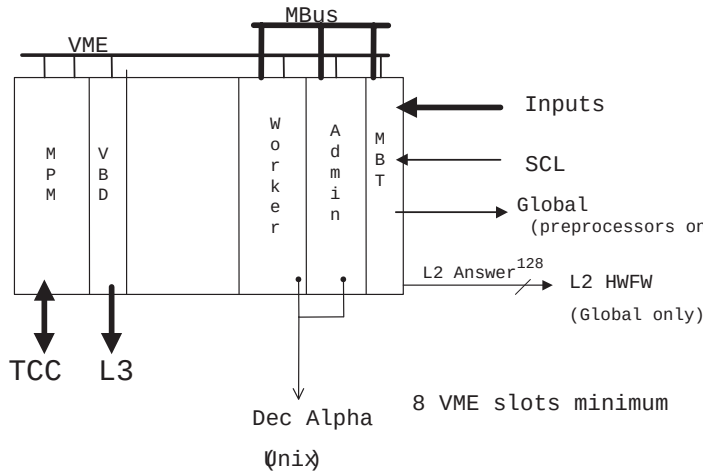

Figure 2: L2 Global Processor

Slot	Card
1	Bit3 Multiport Memory
2	Optical link for Bit3 MPM
3	VBD (readout to L3)
4	Spare
5-12	Spare Processor Slots (4)
13-14	L2 Global Shadow Node ??
15-16	L2 Global Worker Node
17-18	L2 Global Administrator Node
19	Magic Bus Transceiver Card 1
20	Magic Bus Transceiver Card 2
21	Spare

Table 1: L2 Global Processor Crate

ID	Source
0	L1 Serial Command Link
1	L1 Hardware Framework
2	Calorimeter e/γ Preprocessor
3	Calorimeter Jet Preprocessor
4	Calorimeter Etmiss
5	Central Fiber Tracker (CFT)
6	μ Central Preprocessor
7	μ Forward Preprocessor
8	Central Preshower (CPS)?
9	Forward Preshower (FPS)
10	Silicon Tracking Trigger (STT)?
11-15	Spare

Table 2: L2 Global Data Sources

Global performance. It is also easily adapted for the L2 Calorimeter preprocessor, which currently plans 3 Workers.

The Administrator manages the event Buffers, coordinates L3 readout and sends monitoring information to TCC. The Administrator looks at the data only to check data integrity.

The Worker analyzes events and reports the answer (a mask of 128 pass/fail bits) to the Administrator and the L2HWFW, which uses the information to control readout of the main event data to L3. When an event has been marked as *Collect Status*, statistics, errors, and processing time buffers are collected and written via VME to the dualport memory (DPM) accessible to the TCC, which will serve the data to monitoring consumers.

All handshakes between the Administrator and Worker take place on the MBus to avoid long latencies caused by collisions with L3 readout on the VME bus. The "Fred" register ports on the Alpha boards present the current processing state and the current number of events in the Buffers for scaling or logic analyzer viewing.

A similar system of Administrator, Worker, and MBT[4] cards will be used in all L2 preprocessors.

4 Inputs of L2 Global

After a L1 accept decision on an event, data flow to the L2 preprocessors. The data may be from the L1 trigger, or a subset of the full digitized data eventually sent to L3. The L1HWFW sends the trigger conditions requiring evaluation and whether the event has been marked for any kind of special processing. The data sources for Global Processor are shown in Table 2. The CPS and STT preprocessors are currently under consideration; the architecture allows them to be added easily at any time.

All Sources needed for a run must provide data to L2 Global for every event of the run.

Details of the data sent to L2 Global from each of the data Sources is found in Appendix B. The estimated size is 900B/event.

4.1 Input Buffering

The front end crates of the DAQ system can hold up to 16 DAQ events awaiting L2 decisions. To match this, the L2 system also can hold L2 data for up to 16 events. As is evident in Figure 3, there are many locations within the L2 system where L2 data may reside during processing. The simplest control system results when *any* of these sites is capable of holding all 16 events: any data producer is always guaranteed that it can send its data at any time. Thus, we have chosen to place a 16-event FIFO at every place where data is received in the system. The figure above shows an input FIFO 16 events deep for each of the data sources sending information to L2 Global. Further, after the

[4] MBT functionality used by L2 Global is detailed in the following two sections. Additional aspects of the MBT card, including functionality needed only by L2 preprocessors, are contained in Appendix A.

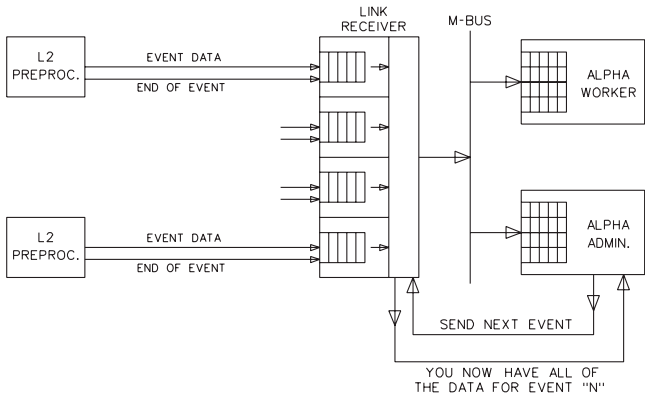

Figure 3: Buffering in L2 Data Transfers

data is broadcast to the Worker and Administrator, it is received a set of FIFOs, 16 events deep for each of the data sources. In this case, the L2 system need not generate a BUSY for the L2HWFW, since if there were 16 events spread through the L2 system, the DAQ system (which can generate its own BUSY) already has 16 events. Howver, one must monitor all the L2 buffers in order to understand the operation of the system.

There are no plans to cause L2 Busy even if all 16 buffers are occupied at any location in the L2 system. Nor are there plans to generate an Error condition if a 17th event arrives at a particular buffer, either in L2 Global, or in any L2 Preprocessor [5]. Either situation is already prevented by hardware on both Front End DAQ systems and by an up-down counter of outstanding L2 decisions in the L2HWFW. There are no good recovery mechanisms from this situation, and the design philosophy is to not detect errors which are in any case irrecoverable.

4.2 Data Transport from Preprocessors

The physical layer of the transport is 160 Mb/s Cypress Hotlinks[8] serial bus. The serial data transport embeds an 8-bit byte in a 10-bit frame, so each link is capable of 16 MB/s. The specifications for transport are found in [10]. Because of the decision to place 16 buffers at the end of each data path, transfers proceed without flow control. The end of event marker is sent by the Cypress control path, not by special values of data words.

The data format[11] includes a header specifying the format, length of header and objects, and an event number in the header and trailer, allowing for verification of the transfer. Objects are fixed length for a given preprocessor, and an integral number of 32b (4B) longwords. The data sources must pad the transfer after the trailer to form a number of longwords divisible by 4, since the Magic Bus is 16B wide.

[5] including the SLIC DSP cards used in some preprocessors

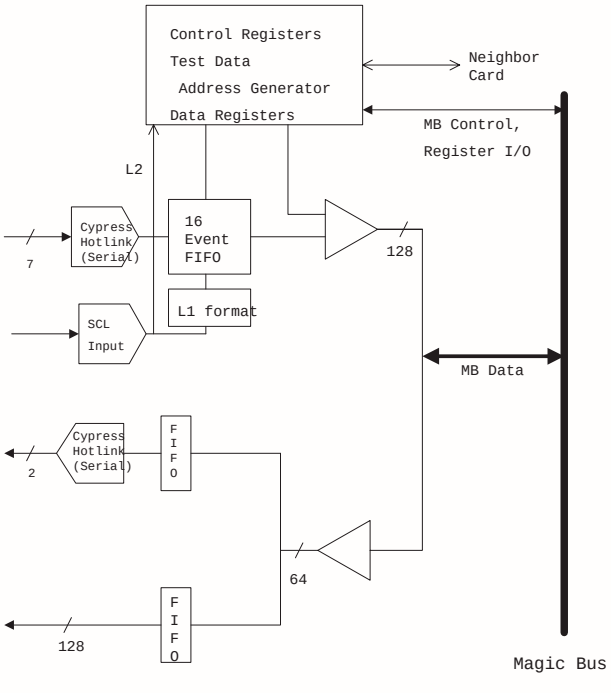

Figure 4: Magic Bus Transceiver (MBT) Card

4.3 The MBT Card

The Magic Bus Transceiver (MBT) Card[12] is shown in Figure 4. The MBT is the main I/O building block for the L2 Global and the L2 Calorimeter preprocessor. The main function of the MBT card is to receive up to 7 inputs and broadcast them to the Alpha modules via Magic Bus. Thus, the nominal input capacity of a system with two MBT cards is 224 MB/s.

In addition, the MBT card receives the Serial Command Link (SCL). MBT separates the SCL information by source. MBT selects information from the L1HWFW and makes it appear as the 8th data source on the card. MBT also selects the L2 accept/reject SCL messages buffers them, so that Administrator can read them as needed.

MBT also contains two other major functions. MBT has two Hotlinks transmitters, used by preprocessors to send results to the L2 Global. Finally, MBT has a wide I/O path, 128 bits of data and a few control lines, to send the L2 Global's answers back to the L2 HWFW. In addition, MBT supports various control, testing, and monitoring functions.

4.3.1 MBT Input Ports

The input ports receive data on the Cypress Hotlinks cable. The data are fed into FIFO memories[6]. The end

[6] The FIFO's are currently forseen to be 64KB/channel. This would support the largest currently known data source, up to 400 tracks, 8B each, 16 event's worth. It would also support raw 16 events of raw tracking data. The largest VRB FIFO's are also 64KB, sized to hold 24 events of maximum SVXII data.

of an input event is marked in the FIFO by a control signal on the Hotlink. The FIFO's can hold 16 events[7]. The control section knows which FIFO's are active, and notes when a full event is available in each input FIFO. Data are not transferred out on the MBus until a full event is available. The Source ID (which is part of the MBus broadcast address) for each FIFO is selectable. In the simplest case, FIFOs on a card can each be assigned to a different Source ID. However, if there are many MBT cards, several FIFO's (on a single card, or even on different cards) can be all assigned the same Source ID. The control section will allow MBus arbitration after each FIFO has sent its event information, and will notify its neighbor card when all FIFO's have finished an event, eventually generating a "DONE" MBus signal when all FIFO's have sent their event.

A GO control message (from Administrator) enables event transmission, but actual transmission only starts when all neighbor cards have an event ready to send. The MBT card at the highest MBus slot number (the "Pilot" card) receives the GO signal and coordinates activities of its neighbor "Assistant" cards.

The L1 HWFW information and the preprocessor information appears on the Hotlink receivers fully formatted, so no formatting and no byte swapping is done by MBT for the standard inputs. The data sources must provide bytes to the MBT so that the agreed data format appears in the correct byte order as seen by the Alpha processors[11], as the MBT performs no byte swapping.

4.3.2 Information from the L1 HWFW

The L1 HWFW builds a standard L2 header and trailer and sends to L2 Global the following information, which is not included in the information on the Serial Command Link:

- L1 trigger decision mask (16B)
- L1 trigger accept number (4 B)

L2 must make a decision for each bit set in the L1 mask.

The L1 trigger accept number is attached to any error messages generated during processing.

4.3.3 Serial Command Link (SCL) and Qualifiers

4.3.4 L1 Accept Messages

The SCL sends notification of a L1 accepted event to every geographic section which needs to read out to L3. In the case of L2 preprocessors, the receipt of the L1 Accept SCL message means that the preprocessor MUST send at least a header block to L2Global for this event, and MUST prepare at least a header block for eventual L3 readout in case the event passes L2. When L2 Global receives the L1 Accept SCL message, this means that L2 LGlobal must perform a decision cycle, and prepare at least a header block for L3 readout if the event passes (or is *Unbiased Sample* as described below).

The MBT must select and transform information from the Serial Command Link[13] information to put it in the correct format[11] before directing it to its input FIFO. The L1 accept messages have header/trailer added and are placed in a source FIFO, appearing to the rest of the system as a standard L2 Global data source. The relevant L1 information carried by SCL is the 3B crossing number at the time of the L1 accept, and the 2B set of L1 Qualifiers.

4.3.5 L1 Qualifiers

Of these L1 Qualifiers, the ones of greatest interest to L2 Global are *Unbiased Sample (UBS)*, *Forced Write*, *L2Global Needed*, and *Collect Status*.

The obligation to read out to L3 is announced by receipt of the L1 Accept SCL message, but a separate L1 *Needed* Qualifier will be provided to the L2 preprocessors (*L2 Pre_xyz Needed*, for preprocessor xyz), and possibly a *L2 Global Needed* may be defined. These Qualifiers offer the preprocessor the option to skip execution of its algorithm for an event without its *Needed* Qualifier and produce a minimal header-only output to L2Global (and preparing a minimal L3 output). This Qualifier would be attached to a L1 bit whenever a L2 Global script using the L1 bit requires the input of the preprocessor. For example, an electron requirement needs the calorimeter EM preprocessor, the L2CFT preprocessor, and possibly the L2PS preprocessor. This is basically a potential performance enhancement; if the preprocessor does not incur deadtime by running every event, the Qualifier could be safely ignored.

On a *Unbiased Sample* event, L2 Global will send the event to L3 independent of whether any of the L2 bits actually passed by marking as passed any L1 bits which were passed. This condition will be marked in the L2G event header. The actual L2 decision mask will also be recorded, as well as the nominal "decision" sent to the L2 HWFW. Such events will occur at a rate determined by the trigger programming, for an independently adjustable fraction[17] of events passing each L1 bit. A secondary effect of the *Unbiased Sample* Qualifier in L2 Global is that additional information is written to L3 to assist in debugging the event. In particular, the inputs to L2 Global are read out (normally they are not), and the outputs from L2 Global are expanded to include more information to allow detailed checking of the processing. The normal output consists of the good candidates found by L2 Global which are associated

Each chip, non-zero suppressed, produces 258B, and the longest readout "ladder" has 9 chips, or 2322B.

[7] For test purposes, the FIFOs can be filled with up to 16 events worth of test data, which can be read out at full speed.

with trigger bits which pass L2 Global. The additional output may consist of good candidates associated with failed bits, and failed candidates, together with their association with L2 bits (passing or failing).

The overall rate is expected to be roughly 1% of the 10 Hz L3 output bandwidth, about .1 Hz, or 10^{-5} of L1 accepts. *Unbiased Sample* events will be steered in L3 to a recording stream destined for the online verification programs. In L2 preprocessors, receipt of a *Unbiased Sample* Qualifier will result in readout to L3 of the preprocessors' inputs, and possibly additional output information beyond that sent to L2 Global. The preprocessor's standard output to L2 Global is not unchanged by the *Unbiased Sample* bit.

The *Forced Write* Qualifier is intended to provide a mechanism for special runs meant to study new L2 triggers. *Forced Write* produces detailed output for study off-line. *Forced Write* Qualifiers are generated at *every* firing of a L1 trigger bit which was marked with this Qualifier in the trigger configuration file. This mechanism is different than the mechanism for producing *Unbiased Sample* events, as UBS events are generated on only a (programmable) fraction of the firings of a given L1 trigger bit.

The *Forced Write* Qualifier in L2 has exactly the same effect as the *Unbiased Sample* Qualifier. However, the *Forced Write* Qualifier is recorded in the L2 Global header as a separate bit from the *Unbiased Sample* bit, because their behavior in L3 will differ: *Forced Write* events flow not to the verification stream, but to the regular recording stream of the special run which defined the *Forced Write* triggers.

After receipt of an event with a *Collect Status* Qualifier, L2 Global (and all preprocessors) will capture their scalers and other monitoring information and place it where TCC can retrieve it for serving with monitoring. The scalers will be captured after processing of the event is completed, so that scaler information should be exactly matched between the L1 and L2 HWFW, the preprocessors, and L2 Global. The monitoring blocks will be tagged with the L1 crossing number of the event with the *Collect Status* Qualifier, so TCC can assemble a consistent set of statistics. *Collect Status* Qualifiers will be generated approximately once every 5 seconds, or .2 Hz.

4.3.6 L2 SCL Information

The SCL also carries information on L2 decisions. This information is captured and placed in a set of registers readable from Magic Bus, for use by the Administrator in verifying that the decision was the same as the one sent to the L2 HWFW. Since this information includes a crossing number timestamp, L2 Global can calculate the total latency (elapsed time) since the L1 accept decision. This information from SCL for L2 decisions[8] is summarized in Table 3.

current crossing number (timestamp)	3B
L2 accept/reject information	1B
L1 trigger number	3B
L3 transfer number	2B
total	9B

Table 3: L2 SCL Decision Information

The SCL also is the means of notifying the L2 HWFW of synchronization errors (L2ERROR). Administrator does this by writing to a register on MBT when Administrator or Worker finds an event with mismatched pieces. Administrator keeps count of these occurrences. The SCL hub responds by broadcasting *SCL Initialize*, to which MBT responds by raising L1 Busy. Since any SCL node can provoke *SCL Initialize*, Administrator must be notified by MBT. Administrator polls an MBT register rather than being notified by an interrupt. Administrator announces that all data buffers have been cleared by writing to an MBT register to rescind the L1 Busy.

L2BUSY is raised by Administrator if the readout buffers for L3 are full; it is also announced and cancelled by a write to an MBT buffer connected to SCL.

5 Outputs from L2 Global

5.1 Reporting to L2 Framework

The answer will be reported to the L2 HWFW via the MBT parallel I/O port, sending 128 bits of result. L2 Global will verify that the answer was received correctly and was interpreted as pertaining to the correct event by listening to the resulting L2 reply message on SCL.

5.1.1 MBT Parallel I/O

The MBT I/O function consists of a I/O control register and 128 bits of data register. In send mode, data is sent in one or two MBus cycles[9], and the data are latched and sent when the control register is written to during a final MBus cycle. MBT sends the 128 bits of information out on a parallel cable, along a few bits of strobe and control. A special bit of the MBT I/O control register sends 128 bits of zero, without the need first send any data. This speeds up L2 Global's normal answer, that the event failed.

[8] If register reads are 8B wide, one might choose to remove the upper B of the timestamp cuts off the latency measurement at 8.5ms instead of 2.2 sec. Removing the lower B coarsens the resolution from 132ns to 33.4μs. There is enough flexibility in MBT to steer a course between these two extremes; only 1b is needed for the L2 accept/reject information.

[9] Both 64-bit and 128-bit paths are under consideration.

It then sends this information out on a parallel cable, along with control/strobe information. The control information includes a special bit which sends all zeros, speeding up L2 Global's normal answer, that the event failed.

In receive mode, the I/O function reads the current state of the 128 b of registers[10].

5.2 Outputs to L3

The current estimate of the information to be sent to L3 from L2 is 300 B. Details are given in Appendix C. L2 Global sends[11] only candidates associated with L2 bits (scripts) which pass, reducing the data volume to this small amount. The difficulty is that these passed candidates must be extracted from the full processing results, and a sufficient data structure written to associate L2 bits to the candidates which caused them to pass. Since C pointers cannot be written to a flat file directly, any pointers used in the L2 Global internal data structure must be turned into candidate numbers of the (fewer) candidates written to L3. Current thoughts aim at a numbering scheme which uses array indices. Only candidates from scripts which passed are written, and the scripts keep lists of the candidates they used.

Attempting to answer (within some level of precision) the question "did this jet pass L2" is one element driving the requirement for making an association of candidates to L2 trigger bits. Using L2's candidates as a starting place for L3 is another. The Worker output area can be rather compact, since finding the passed candidates will require copying data in any case.

Unbiased Sample events will require further postprocessing. Even if the whole of the internal tool storage is dumped, pointers must be flattened (turned into array indices).

VBD's speed depends on the number of distinct areas it is asked to read out, even if the number of words at a given location is zero. We will probably build the output event in a buffer area large enough to hold Unbiased Sample events with maximal information.

The output header for L3 will include the following status bits:

- *Unbiased Sample*
- *Forced Write*
- L2 Global Ran
- Passed in Distress
- Data format error

[10] The registers could be read twice to be sure of stability of control levels.

[11] The current plan assumes favorable results in timing studies. If the time to postprocess data is too large, a much larger data volume may have to be sent. The limit is an average of 5-10KB/event.

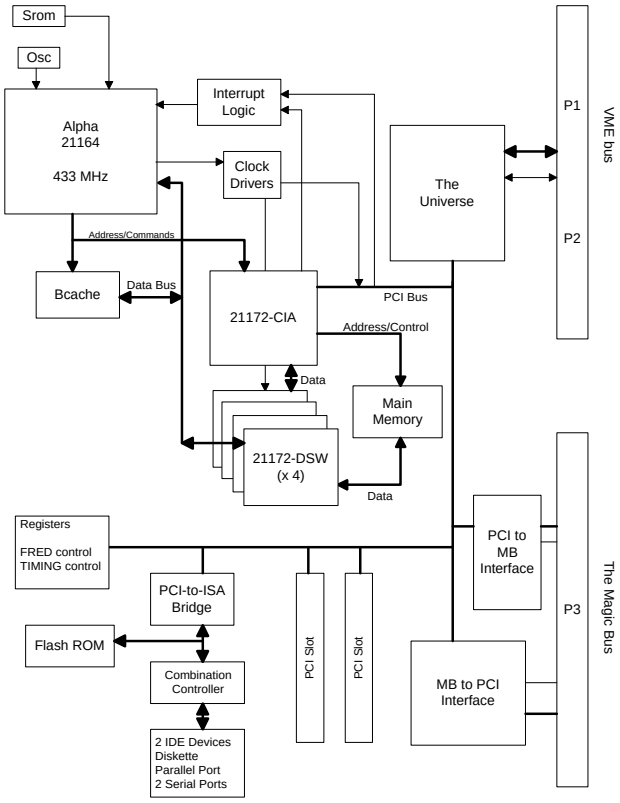

Figure 5: The Alpha in VME Card.

On Unbiased Sample events, the extra information passed, in addition to the L2 decision mask reported to L2HWFW (L2 mask = L1 pass), is

- L2 decision mask (16B, real answers)

6 Processors

6.1 Alpha in VME

The Alpha card [14] being developed is shown in Figure 5. The design is based on the layout of the PC164 card sold by Digital Semiconductor[15]. The processor can execute 2-4 instructions per cycle. The board includes Digital's 21172 PCI interface, which provides 30ns/cycle, 64-bit input path to memory, for a maximum rate of 267 MB/s. The card under development adds several elements to the internal PCI bus of the core workstation design. A 64-bit VME interface[12], the Tundra "Universe" chip [16], allows communication via VME bus. The "MB to PCI Interface", a block transfer receiver for fielding MBus broadcast writes, is the main data path into the processor. The "Fred" ("Timing Control") Registers are fast I/O port used for control or monitoring. The "PCI to MB Interface" is a bidirectional programmed I/O interface between PCI

[12] Currently, only 32 bits are expected to be implemented on the Alpha card.

and MBus. The programmed I/O is used for communication between Administrator and Worker, reading and writing test data on the MBus, output via MBus, reading MBT's data registers, and controlling the MBT card.

The "Srom" is the serial ROM including the initialization code for the Alpha 21164 processor chip. The "Flash Rom" is readable once the Srom code has run, and includes the code to download user code via a PCI Ethernet card a rudimentary local debugger, and the server for a remote debugger. The "Bcache" (Backing Cache) is the 2MB L3 Cache from which we intend to execute most code. The 21172 chipset is the interface between the Alpha chip, main memory, and the PCI bus.

A developer kit is available at very modest cost. It provides the CAD layout files for the PC164 card on which the VME Alpha is based. [13]

6.1.1 Buy vs Build and Upgrade Path

The key reason that a specialized processor was used instead of a standard workstation is the need for fast I/O from many sources. The availability of the development kit makes it possible to both leverage the design effort of the Alpha motherboard from Digital and to add I/O directly to PCI. The fast input collection from multiple sources is handled by the Magic Bus. We have consulted with Digital during the system design process and they concur that a method of collecting the many inputs was not available commercially.

Using PCI as the input bus means that the input design work is portable in principle to another processor which supports 64-bit PCI (in Digital's flavor). Using faster versions of the processor should be relatively straightforward so long as Digital's policies on producing motherboards and development kits remain the same. The 500 MHz version of the chip used in the prototypes is pin-compatible with the 400 MHz version available when design work began; 600 MHz versions exist but have not been examined in detail. The feasibility of upgrading from a 21164 to a next-generation 21264 processor before the start of Run II is less clear, as the entire chipset and architecture will change.

6.1.2 VME Interface

The VME interface allows I/O to and from Alpha memory by mapping pieces ("windows") of VME address space to Alpha memory space. I/O performance has not yet been measured. Estimates for uncoupled[14] cycles are 20MB/s, so that the Universe chip would not constrain VBD readout speed.

6.1.3 MBus Block Transfer

The Block Transfer Engine receives broadcast data from MBus and sends it to predefined addresses in Alpha Memory via PCI. There are a total of 256 possible broadcast addresses. One key function of the block transfer engine is the "Mapper", which is simply a register loaded with a base Alpha main memory address for each MBus broadcast address. Thus the Mapper associates each MBus broadcast address with a separate window of Alpha memory. The Mapper registers increment automatically as successive data words arrive from a MBus broadcast address. Each MBus (16 B) broadcast transfer is treated as a PCI block transfer, with data being sent to successive Alpha memory addresses at 8B per PCI cycle. When the MBus broadcast address changes, a new PCI block transfer begins, starting from a new Alpha memory address.

PCI on the Alpha has a nominal bandwidth of 267MB/s. Preliminary performance measurements indicate that block transfers from MBus actually take place at 20-100 MB/s, depending on loading and transfer length. The 20-25 MB/s figure applies for 16 B header/trailers without data; the 100 MB/s figure is reached for data sources with 64B or more of user data without memory contention. Thus the aggregate input rate is currently estimated at 50-75 MB/s, or 15-20 μs to load a L2 Global event, overlapped with processing. Bus loading will be further discussed in section 7.

6.1.4 MBus Programmed I/O

Programmed MBus I/O makes communication with a MBus address (typically in an MBT card or another Alpha card) available by reading or writing a PCI address. This is rather analogous to the use of VME Programmed I/O via the Tundra Universe chip. The range of MBus addresses a given Alpha card will respond to is set up during initialization of the executable[15]. This address range distinguishes different Alpha cards from each other, again in analogy with the setup of the VME address mapping in the Universe chip. The mapped address range includes the MBus address of the inboxes used for messages between Alpha nodes, as described in section 9.2 below.

[13] The first two additions to the PC164 base design, VME I/O and MBus block input, are designed, tested, and added to the board layout, while the MBus programmed I/O is designed and being added to the first prototype layout. The final addition, register I/O, is under design, and intended for a second round prototype.

[14] During uncoupled cycles (also know as posted write), the CPU can go on to other things. This is fastest, but CPU must eventually check for completion. In coupled cycles, CPU instruction cycle coupled to PCI bus cycle Coupled to VME Bus cycle. CPU stalls until operation completes on target bus. This is slowest, but gives simple software.

[15] Setting up address windows in executable downloaded code allows all Alphas to share the same boot-up ROM code. Switches to distinguish Alpha cards can help this process, though the differences among the executables may be sufficient to separate the MBus address spaces. VME address spaces will probably be separated by the same mechanism as MBus address spaces.

There is only speculation at this point as to the Programmed I/O performance; it will be measured when the first prototype arrives. Guessing that it should be no slower than VME in favorable modes, 25MB/s, that would be roughly than 1 μs for a (wide) register read or write such as sending a short message to another node, perhaps 2-4 μs to send a result to L2 HWFW or read, poll. and read back its result[16].

6.1.5 Fred I/O port

The Fred Port provides 64b of I/O directly connected to PCI, so that latency should be short. Planned uses include

Outputs:
5b Alpha input Buffer count,
5b state code
4b L3 output Buffer count
Inputs:
2 lines from VBD for Administrator

A latency of 50 ns is adequate to use as a gate for scalers in the L2HWFW, which clock at the crossing rate, every 132 ns. The Fred Port is not yet designed, so details are still lacking.

6.1.6 Timeouts

Event processing, and most data flow functions, should terminate within a given (generous) time. If it is advantagous, such time limits can be enforced by hardware timers generating an interrupt which causes reinitialization of the system on the principle that such a long time limit could only be exceeded if the system is hung. Thus, the useful scale of timeouts is of order of seconds, fast enough that self-clearing can be initiated before a human operator would be able to intervene, and fast enough to minimize deadtime generation during physics data taking.

During commissioning, it is likely more useful to allow the system to hang, so that more detailed diagnosis can be undertaken by experts. Even during physics data taking, the case is somewhat equivocal, as the self-clearing action should be well enough understood that it is more likely successful than operator intervention, noting that an unsuccessful operator intervention is likely to generate much more serious data loss. As a timeout is certain to lose data, it should be logged, and generation of timeouts should be held well below the level at which the resulting data loss is a noticable effect on physics luminosity. Further, timeouts only make sense if they normally DO succeed in reviving the system; otherwise they only serve to decrease the information available for a diagnosis.

[16] Calorimeter and other preprocessors use programmed MBus I/O to send results to Alpha. Since it is programmed I/O, the send time must be charged to event processing. At 25 MB/s, the calorimeter preprocessor would spend only 1-2 μs; a CFT preprocessor sending 400 B/event could take 16 μs/event.

If it is decided to arbitrarily pass events likely to take a long time to process, it is desirable to make the decision predictively, by looking at length of candidate lists, rather than reactively, by timing out moderately long events. Again, a timeout at a scale of a few seconds might have a chance of catching only events unlikely to ever terminate. We do not yet have a feel for whether passing such events and restarting on the next event typically would be a successful recovery, or whether more violent measures would be needed, and we are unlikely to understand the real typical problems before commissioning.

The known timing facilities on the Alpha board are committed to serving several functions, namely state timing, error message timestamping, and communication with the debugger. It is not yet clear whether a multi-second timeout could be managed with the existing timer (plus some management software effectively counting the time since the beginning of the event), or whether an additional hardware watchdog timer should be mounted on the Alpha board, perhaps in PCI space.

6.2 Memory and Cache layout

As we proceed to implementation, we will carefully lay out a map of the memory locations of VME and MBus control space, Boot Rom code, downloaded code, input and output buffers, communication inboxes, download areas, scalers, user code working storage and tables, and the relation of VME and MBus windows into memory with these items. We will attempt to write-protect code. Cache is mapped periodically to main memory, so lock code into memory, one must control its placement so data memory does not map to the same cache locations as code, and so that data areas used for an event do not contend with each other for cache.

6.3 Programming Tools

The software component of the developer kit consists of C header files and libraries; they allow one to compile, link, and download, code from an Alpha running NT or Digital Unix. I/O supported is minimal: a console, and an Ethernet controller supporting a debugger and code download via a TFTP file read. The debugger is the standard Dec Unix debugger with extensions to debug the target Alpha node from a workstation. There is no debugger for target Alpha code available on NT[17] .

[17] We have investigated using VxWorks instead of the developer kit. Tight event code can't use true operating system support in any case, so the issues are the quality of the support environment, availability, and services for occasional I/O such as initialization, event dumping, and monitoring. The VxWorks development environment does not appear to be greatly superior to that offered by the developer kit. VxWorks is not planned to be made available for the PC164 card in any case. That leaves us to live with the system of TCC (under NT) and the MPM to perform most of this occasional I/O. We are also considering other online environments, for example embedded C++ from Greenhills Systems,

Bus	Size kB	Load MB/s	Max MB/s	Used	Time μs	Wait μs
VME	.6	.6	10	6%	60	1.8
MBus	1.8	18	80	23%	23	.25
Cache	1.8	18	533	3%	3	
Input	.2	2	16	13%	13	

Table 4: L2 Global Bus Loading

Bus	Size kB	Load MB/s	Max MB/s	Used	Time μs	Wait μs
VME	.3	.3	10	3%	30	.45
MBus	3	30	80	38%	38	.70
Cache	3	30	533	6%	6	
Input	.34	3.4	16	21%	21	
1 Out	.1	1	16	6%	6	
L2Out	.3	3	16	19%	19	1.8

Table 5: L2 Calorimeter Preprocessor Bus Loading

This environment is sufficient, but austere: there is no operating system. Programs calling C routines MALLOC and FREE will link properly, but there is no virtual memory, MALLOC merely increments the number of words used, and FREE makes no attempt to actually de-allocate memory. In this environment, dynamic allocation is indistinguisable from a memory leak. We plan to use only static memory allocation. Because of this issue, any use of C++ will be approached with extreme caution.

7 Bus Loading and Latency

Table 4 shows the expected bus loading, capacity, and transfer time for various of the L2 Global crate buses.

Table 4 summarizes the bus loading situation for Global, based on data volumes per event purposely chosen to be twice those expected based on present best estimates. The Bus column shows which bus is being concentrated on: VME (only used for L3 readout), MBus for DMA input into the Alpha memory via the combination of the MBus Block Transfer and PCI bus on the Alpha, Cache for transfer to Alpha Cache from main memory, and Worst Input for the Hot Links inputs. The Size column gives the size of the average event (or fragment, for Input). The Load column multiplies the Size by its repitition rate, 10 KHz for input, and 1KHz for L3 output. The Max gives the current estimate of the channel Capacity, while the Used column shows the ratio of the Load to Capacity. The Time column gives the time to effect a transfer for one event, Size/Capacity.

Most of these transfer times are actually overlapped with calculation, so they appear as latencies but need not be charged to processing time of an event in steady state with events stacked up in the input queue. An exception to this is Cache bus. If the event processing eventually requires a large fraction of the input data, then, depending on the success of lookahead cache management, much of the data input time will be charged to the algorithm execution. More insidiously, any data produced will have to be written out to memory, so one must consider carefully the tradeoffs between remembering results and recalculating them, and one must be wary indeed of Global producing vast quantities of intermediate results. There is adequate capacity available to write the expected actual outputs. We intend to measure the Cache bandwidth more carefully; the current figure is quite conservative: a 256b path cycled at the memory speed of 60 ns gives a bandwidth of 533 MB/s.

Finally, The Wait column looks at expected latency in gaining bus mastership, which depends on how many subdivisions exist in the transfer offering an opportunity for arbitration:

$$Wait = P(busy) \times (Time/2)/Nblocks$$

The probability of finding the bus busy is just the fraction capacity used. The VME transfer is a single block, while the MBus input transfers are broken into roughly 10 blocks. Thus, in spite of the lower expected VME bus occupancy, the higher transfer speed and smaller blocks make MBus much more favorable for implementing interprocessor communication during the event cycle. This evaluation indicates why the VME bus is used only for L3 output. It is worth doing the algebra to examine dependence of the expected wait time (W) on size (S), rate on the (R_{bus}), capacity (C) and blocks (N):

$$W = R_{bus}(S/C)^2/(2N)$$

An even more interesting quantity is the fraction of the nominal processing time budget occupied by each bus wait, as a function of the input rate to the processor, $W \times R_{in}$

$$\%(ProcessingBudget) = (R_{bus} \times R_{in})\ (S/C)^2/(2N)$$

The quadratic dependence is a two-edged sword. If problems are under control, the situation is probably very comfortable, but anything going wrong goes wrong quickly. Thus our factor of 2 event size safety factor gives a factor of $\sqrt{2}$ rate safety factor.

Because the same machinery is forseen for preprocessors, Table 5 repeats the calculation for a well-studied example, the calorimeter preprocessor.

In this case the output sizes to L2 and L3 have been estimated generously, but the input to Cal Preprocessor, which is well know, is held to its actual value. Since

which would give better online support and allow more flexibility of development platform.

the input data flow is much heavier than the output data flow, VME is much less heavily loaded than in Global. One might even be tempted to use VME for interprocessor communication if it were not for a preference for identical software. Further, there are the penalties in throughput caused by long tails in processing time distribution: though the VME bus is not busy often, it stays busy a significant fraction of the time budget.

The heaviest bus loading is the use of MBus for input. Luckily, the input data volume is a well known fixed-length transfer of calorimeter input data.

Considering the slower output path to L2 Global, we note an important circumstance. The "1 Out" line describes the situation in the most heavily loaded single output to L2 Global. The "L2Out" line describes the situation for all 3 outputs, considered serially. Output will be split among several MBT's, for electrons, jets and Etmiss. But output is slow: the MBus programmed I/O is not expected to be much faster[18], if at all, than the Cypress Hotlink output section of MBT. Since the L2 output is via programmed I/O, it is charged directly to the event processing time. Further, since each processor must occupy MBus to send its data, and the simplest synchronization is for to wait until all processors have completed work, the output times for an event wind up adding to each other unless the algorithms are sufficiently different in speed that one algorithm writes results while another is still calculating.

From these considerations, it is important that the CFT preprocessor control its data volume lest output time leave no time for calculation. The simplest strategy is to send low p_T tracks only when L2 Global needs them for this event, a selection which can be controlled by a L1 Qualifier.

Given the relatively heavy loading of MBus, one might become concerned about interprocessor communication. However, Table 4 performs the expected waiting time calculation based solely on the input bandwidth of MBus. This is because communication between Worker and Administrator takes place asynchronously with respect to the input, but synchronously with end-of-event processing, where MBus is generally in a quiet state[19], so that end of event processing does not interfere seriously with L2 output I/O. However, output does interfere with the issuing of the GO signal to the MBT card during interrupt processing at the end of the receipt of the input event, and the expected cost of such interference should be added to the average processing time of an event, since on average one event arrives for each one processed. The $1.8\mu s$ penalty is not serious for the calorimeter preprocessor, but the heavier loading of the CFT MBus is $3.1\mu s$ at the expected event size, and the penalty grows quadratically with event size.

The cache is more heavily loaded by data input in calorimeter than in Global. However, the figure in Table 5 is an upper limit, as it represents an algorithm which accesses the full 3KB of data, both electromagnetic and total Et's for each trigger tower. Even Etmiss would access only half the data.

8 Queuing Simulations

Extensive queuing simulations[18] of the L2 system have been performed, using the package RESQ[19], and where possible checking the results with analytical or numerical calculations[20].

The first studies considered a farm architecture and found it entirely unsuitable for a system in which the front end readout design was frozen with a requirement that L2 decisions be returned in the order of the L1 decisions. A 16-node farm operating at the nominal 10KHz rate, with a realistic long-tailed (sum of two exponentials) time distribution produced a deadtime of 2.1% without this "serialization" requirement, but 69.3% when "serialization" was imposed. Maintaining the mean processing time while shortening tails to a single exponential only reduced the deadtime to 49%; "serialization" and a farm architecture are incompatible without more buffering than the 16 events available in the DØ front ends. The basic problem is that a slow event prevents processors which have already finished from getting a new event. The typical processing time of the system is in some sense governed by a distribution more like the slowest of the events in the system, rather than the average event. The worst-of-n distribution is very long-tailed. To recover the non-serialized deadtime with a serialized farm, the mean processing time per event would have to be reduced by a factor of 3-4.

After these results, we turned our developed the present preprocessor-global architecture. A typical set of parameters for the simulation is shown in Table 6. The Hyperexponential[20] distribution referred to here is a sum of two exponentials with the relative weight and mean adjusted to give the specified overall mean time, with a ratio of rms to mean of 2.0, as compared to the ratio of 1.0 for a simple exponential. This is a reasonable representation of the typical event processing time distribution found in Run I Level 3.

At 10KHz input rate, these parameters give a deadtime of 3.7% when 16 event buffers are supplied everywhere in the L2 system but the DAQ Front Ends

[18] There is a strong case here for 128b wide path rather than 64 if it can buy a factor of 2. This could possibly be matched by raising the Cypress Hotlinks to 320Mb/s.

[19] MBus may not be quiet if there are multiple worker nodes finishing about the same time, so one is writing to L2 while the other is communicating with Administrator after its L2 output.

[20] The parameters in a sum of two exponential are the two means τ_1, τ_2 and the q, the fraction of the weight placed on the exponential with the smaller mean. There are only two constraints, the mean time μ, and $C = rms/\mu = 2.0$. RESQ chooses the parameters according to the conventions $A = \sqrt{1 - 2/(1+C)^2}$, $q = (1+A)/2$, $\tau_i = \mu/(1 \pm A)$.

Table 6: Input parameters for the basic RESQ model of L2. The distribution type is Hyper–exponential, Exponential, Gaussian, and Fixed.

Preprocessor	Operation	Time(μs)	Distribution
CAL	Input	50	F
(EM)	Process	50	H
	Output	10	F
(JT)	Process	50	H
	Output	10	F
MU	Input	15	F
	Input	15	G
	Process	50	F
	Output	5	F
	Output	15	G
TK	Input	8	F
	Process	40	H
	Output	10	F
GLB	Process	50	H
	Output	15	F

maintain the constraint of at most 16 events anywhere in the L2 system. This reasonable deadtime is achieved with total (processing+output) times of 65 μs for preprocessors and for Global.

If the number of buffers between Global and the preprocessors were reduced to 1, the deadtime rises to 11%. Introducing even a single buffer in front of L2 Global helps the system a great deal, dropping deadtime to 6%. However, the relative importance of having buffers in front of L2 Global, as opposed to in front of preprocessors, depends on which processor is closer to limiting the rate. The decision to place 16 event buffers at all buffering points in the system avoids such guesses at the relative performance of pieces of the system, and results in a much more robust design.

We studied the effects of "busy" events by introducing correlations among the processing times in preprocessors and the global processor. The added fluctuation causes mild increases in deadtime.

We found that it was quite important to allow the various preprocessors to transport data and move on to the next event without waiting for other preprocessors to finish. With the same processing times, adding a lockstep requirement across all preprocessors increased deadtime from 3% to 35%.

A more recent study considered the results keeping multiple workers in event lockstep in the Cal preprocessor. A subsystem of 2 Missing Et processors operating in 50 μs with 7μs Output time (both times deterministic), with a jet processor with a hyperexponential 50 μs mean and an em processor with 20 μs hyperexponential mean gives 3.4% deadtime when no event synchronization is imposed, rising to 4.7% with synchronization imposed. The penalty for lockstep is more severe when the em and jet processors are comparable in processing time: the deadtime rises to above 10%. A more complex Administrator model allowing the processors to leave lockstep may indeed be required, even allowing for the likely relative timing performance of the Cal preprocessors. The need for allowing events to flow independently among processors is more urgent as the number of preprocessors with similar processing times increases and as the processing times of the processors increase.

We studied the effect of using a *Needed* Qualifier on the allowable mean processing time budget of a preprocessor. Naively, one might hope that the allowed budget for a fixed deadtime scaled as $1/f$ where f is the fraction of time the preprocessor must run. In fact, the allow budget is closer to $1/\sqrt{f}$, a useful but more modest improvement. A preprocessor running 1/3 of the time might acquire a budget of 1.7 as long.

Reducing the number of DAQ from 16 to 15 globally throughout the system produces no measurable change in the deadtime. This reduction of the Front End buffers by 1 is a good simulation of the effect of introducing another stage in the L2 processing pipeline. This is planned for the L2 Muon system, where a SLIC processor does the main calculation, and the event is handed off to a Worker in a standard crate for output formatting. The result should be a good estimate, provided that the postprocessing time is negligible and introduces no deadtime of its own. This assumption would need to be tested more carefully in a more complex situation such as the proposed STT trigger, where there is possibly considerable work to do in merging the STT and L2CFT track lists.

The results for a scenario with 2 Global Workers handling alternating events are shown in Figure 6. Since this is a N=2 serializing farm, a full factor of 2 gain in allowable time budget is not to be expected. The actual expansion of the allowable L2 time budget is only a factor of 1.2-1.3 in the usefully low-deadtime region, though it rises to 1.4 or so in the regime where L2 Global is the dominant contributor to deadtime.

A simplified description of expectations for parallelization of parts of events, is discussed further in section 9.12.1. Simulation results examined the case of a moderately-overloaded Global worker with 11.6% deadtime due to a 85μs hyperexponential mean with an additional 5 μs fixed processing time. Adding a second node to alternate events reduced the deadtime to 8.6%, a marginal improvement. The same deadtime was obtained if two nodes split the event 70% of all events, and on those events were perfectly balanced, each taking 45μs Hyperexponential mean, with the fixed pro-

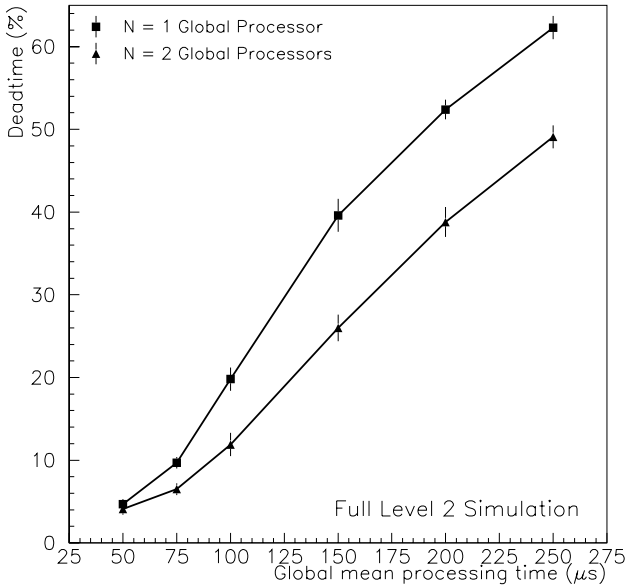

Figure 6: Comparison of N=1 and 2 L2G Workers

cessing time in a single node of 10 μs to combine the results and do the fixed-time processing. This scenario assumes a 10% overhead for splitting and recombining the event. Better deadtimes (7%) are obtained if the overhead drops, with modest dependence on the fraction of work, provided over half the processing was shared equally. To lower deadtime below 6% level, essentially all events would have to be shared with minimal overhead. This strong sharing of workload seems quite optimistic, however, and would likely be labor-intensive to manage at the trigger configuration level.

The upshot is that simple schemes of workload sharing do not seem to be easy escape routes for failing to meet the time budget. More promising is placing a serialization burden on Administrator, and complicating the Buffer management.

9 Online Software Structure

Here we give an overview of Global's online software, and some of the lower layers supporting the functionality forseen.

9.1 Data Movement

Data movement consists of keeping track of events as they arrive, processing those events after they have arrived, announcing results to the L2HWFW, and writing events to L3 if required. The input events arrive by DMA while event processing proceeds. To avoid deadtime generated by waiting until the end of processing an event, the input event flow is handled by an interrupt routine. The main event loop consists of verifying integrity of the data, computing the L2 Global decision, and announcing the decision. L3 output is less time-critical, so is handled by polling as event decisions are reached.

Administrator concentrates on managing the input and output data flow, and does more data integrity checking than Worker. Worker concentrates on performing the calculations needed to actually arrive at the triggering decision. The key piece of software for Worker is the L2 Script Runner (named in analogy with the L3 Script Runner). This interprets a downloaded L2 trigger script by calling L2 filter tools with appropriate parameters for each L2 bit needing a decision. Results are stored and candidates associated with passing L2 bits are written out for the benefit of the L3 trigger. To coordinate their activities, Administrator and Worker must exchange information on every processed event.

9.2 Message Passing

The mechanism for message passing between Administrator and Worker is for the sender to do a MBus write into a pre-assigned input area (inbox). The receiver of a message must poll the inbox periodically to notice receipt of a message, and then clear the input area so that it can recognize receipt of a new message. The input area is specified in terms of a programmed-IO MBus address, so that individual nodes may map this location to a preferred address. The Administrator will have inboxes for each Worker node for normal event processing messages. The Worker will have an inbox for normal event processing messages from Administrator.

Initialization messages from Administrator to Worker may be passed via a second inbox in each Worker. We will probably chose to implement notification of these messages by interrupts rather than polling the inbox periodically during the event loop. A cleaner event loop results by using an interrupt mechanism. Further, interrupts get Worker's attention even if Worker is confused.

9.3 I/O Library

There will be an I/O library consisting of routines to read, poll write, and possibly cause interrupts from VME and MBus. The message passing will be based on this layer.

The (Fred and MBus) I/O registers could have some of their complexity hidden by a higher-level software layer. For example, one might like to update only certain bits of the output register, so one might send a mask and a value, with the software providing memory of the previous values of bits outside the mask, even though the raw hardware might write all output bits on each update. Similarly, if certain bit fields are configured as input bits or output bits by the cabling, this knowledge could be localized in the I/O routines.

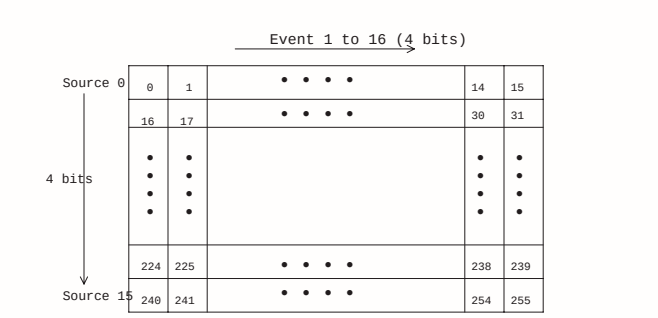

Figure 7: Mapping of Broadcast Addresses to Events and Sources

9.4 Buffer Management

The Mapper is part of the block transfer engine on the Alpha card. It maintains a target address for the next transfer from each of 256 MBus broadcast addresses. Figure 7 shows how we intend to use the 256 broadcast addresses as 16 Events X 16 data Sources. We currently do not plan to reserve any of the data Sources for interprocessor communication.

The MBT cards assign an MBus broadcast address for each of their Input FIFO's. The MBus broadcast address is assigned by combining two half-bytes. The upper half-byte is the Source ID, while the lower half-byte is a 4-bit event number taken from a counter. Initialization assigns the Source ID, and clears the event number. Each GO increments the event number part of each broadcast source address, so that on any given event, only 16 (at maximum) broadcast addresses are used in the Mapper.

A Buffer number lies roughly in the range of 0-50: 16 input working Buffers, 16 pre-assigned input Buffers, 8 L3 Buffers, and a few "spares" as needed. The set of objects indexed by a Buffer number is large: any control and status information for the Buffers, a set of input Buffers for each source, working storage for all event processing for the event, and output storage for the event.

The Mapper is initialized with a base Alpha Main Memory address for each of the broadcast addresses. This base address is the beginning of the input piece of an event the Buffer for that input source. As data arrives from MBus, the data are sent via PCI to the corresponding Alpha Main Memory address, and the address of the source increments.

The Mapper's current addresses can be read over PCI, so that transfer counts can be deduced for each source on a given event.

At any time, the 256 slots of the Mapper are pointing at (the 16 input pieces of) 16 different event Buffers. This means that the Mapper must know about where it will put data of a 16 events ahead of their arrival. After a new event has arrived and the FIFOs between the Input section and the PCI Mapper have been drained

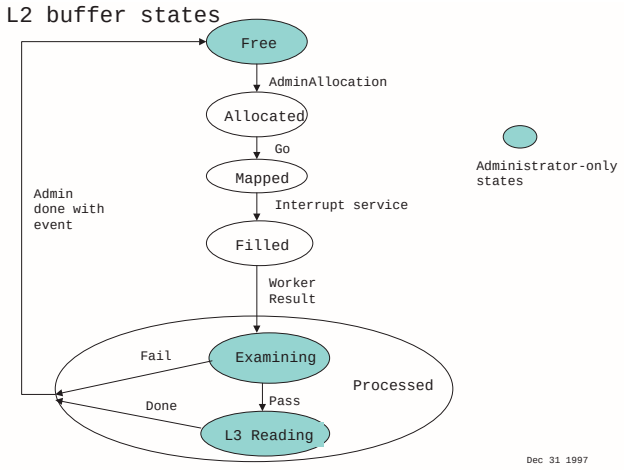

Figure 8: State Diagram for L2 Global Buffers

by sending all data to memory (on all Alpha cards), an interrupt is generated so that the Alphas may acknowledge receipt of this event and prepare for the next event. The Buffer with the new event is placed on the Filled list, and the location of the oldest Allocated Buffer is placed in the Mapper event slot just used, which will be the destination 16 events in the future. Administrator Sends a GO to the Pilot MBT card and the cycle resumes.

Figure 8 indicates the cycle of states of a Buffer during processing. The shaded areas indicate states only distinguishable in Administrator. Only the Administrator has the Buffer free list. A Buffer goes directly from Processed to Allocated as far as Worker is concerned. The lists of events in each state are in the form of a FIFO (queue).

In the initial condition, 16 Buffers are in the Mapped state, and 16 are the Allocated State[21]. From the point of view of a Worker, two kinds of changes are possible:

- A *new event* arrives, moving the oldest Mapped Buffer to Filled, and the oldest Allocated to Mapped. The number of Mapped Buffers remains at 16. The sum of Allocated and Filled remains at 16.

- An *event finishes* processing. The oldest Filled event moves to Processed. But to keep the sum of Filled and Allocated at 16, one Buffer must be added to Allocated; this is the Buffer named by Administrator in response to the Worker's Result message[22].

[21] or 17 to avoid race conditions

[22] To avoid concerns about race conditions, one could require a final handshake from Worker to verify that the Allocation is complete before Administrator proceeds to inform the L2HWFW of the event decision. Another strategy would be to have Administrator directly manage the Allocated data structure in Worker's memory via MBus messages, and remove the Allocated Buffer from the Administrator response to Worker's Answer for the event

The Administrator does additional work on Processed events. All events are examined to verify that they are not made up of incompatible pieces, and that the result and event number matches that reported by the L2HWFW. If the event is to be written to L3, it enters another queue and waits until L3 has read the event before the Buffer is returned to the Free list.

Buffer management needs a FIFO (queue) data structure, so routines to support this data structure will be supplied.

9.5 State Machine Services

As an Alpha makes a transition from one state to another, it can call a State Machine routine to help make the transition. This routine reads the clock and enters the time in a circular buffer as the input time to the new state and the exit time to the old state, as was done in the Run I L3 "Software Logic Analyzer". Keeping a circular buffer of times allows calculating not just a mean time in state (*e.g.* time spent in processing), but its distribution, which can be an important diagnostic. The routines also post the new state to the Fred port, where it can be used as a scaler gate to easily calculate mean times in state, or be visible to a logic analyzer.

Similar routines send new Buffer occupancies to the Fred port whenever Buffer counts in the Alpha change.

9.6 Timing Services

A lower level beneath the State Machine Services are the timing services. Circular timing buffers can also be used for timing intervals other than those marking entry and exit of a set if exclusive processing states. Two examples which come to mind are the L3 latency (the interval between when a buffer arrives at the L3 readout queue until the VBD has finished writing the event), and, perhaps the time spent in the Event Interrupt routine. Since these processes are to some extent overlapped with other processing, one might choose to time them separately. Alternatively, they could be displayed as extra "state" bits, with the understanding that the time in these "states" should not be summed with the time in other states.

9.7 Run and Event Number

The sole purpose of run and event numbers in L2 is to tag Error messages. The numbers are not stamped on events by L2.

A run number is downloaded with the run initialization to the L2 Global and L2 preprocessors. Only a single run number is known to Administrator, in spite of the possibility that two or more data taking runs are taking place simultaneously.

A L1 accept event number is acquired from the L1HWFW data. This number will be stamped on the event by L1HWFW should the event pass L2 [23].

9.8 Error Messages

Error messages (are known as Errors, generically, though only high-severity are sent to the alarm system) will be generated by calls to the off-line ERRMSG utility with its bottom I/O layer replaced to write into a message buffer in each node. Errors include a Run number and a L2 input event number[24]. Each node adds the node name and time of day to the message. Administrator, when handling monitoring information, concatenates all Error buffers and places them in the MPM where TCC will retrieve them. TCC may further timestamp the batch of messages.

9.9 Global Administrator

The Administrator is party to all communications involving Alpha's. This allows much more straightforward debugging.

9.9.1 Messages between Administrator and Worker

L3 readout through the VBD[9] takes place via the VME bus, coordinated by the Administrator. Since events pass L2 at 1 KHz, and the VBD does not allow re-arbitration during readout, it was judged unwise to use VME for messages between Worker and Administrator which must take place once per event, at 10 KHz.

The messages between Worker and Administrator take place on Magic Bus. They communicate once per event.

Worker announces the Result of processing to Administrator when it has finished processing an event. This message contains:

- 3B L1 crossing number for this event
- 1B containing whether event passed, whether the event pieces matched, whether *Collect Status* event, and other status codes
- Buffer number (1B) for this event

If the event passed, the Administrator will read further information from the Worker, including the 128 bit (16 B) mask of L2 filtering results, and the word counts needed to read the event.

Administrator's response contains:

[23] L2 preprocessors do not have this information. Instead, each node generates a L2 Input event number. This event number is reset by a *Clear Scalers* command. This number will be smoothly incrementing, but synchronization with L1 Accept number is unlikely, as all preprocessors need not see all events. The SCL message could provide a 3B L1 crossing number to partially tag the event for matching with Global events.

[24] and possibly a 3B L1 crossing number encoded in Hex

- the Buffer Number (1B) to Allocate
- 3B L1 crossing number for this event?

9.9.2 Messages from TCC to Administrator

Messages from TCC to the L2 system come via VME. The list includes:

- halt/resume (sit and wait for debugging/ continue)
- reset: download exe, read constants, do "startup" with TCC, Worker(s)
- download script/parameters. Tell Worker to read scripts and parameters from MPM.
- begin run *Nrun*: Admin, Workers clear scalers and monitoring buffers; Readout required or forbidden
- event dump
- remove or add a shadow node?
- collect begin/end of run scalers?
- copy monitor information to MPM
- download an event?
- perform self-test on (stored) event n?

The cleanest implementation results if TCC generates an interrupt fielded by Administrator, which looks in a known location in MPM to find what kind of message TCC has sent. Administrator will reply with an indication of whether the message was successfully executed. Administrator may need to pass these messages along to Worker by a similar interrupt mechanism.

Event dumping must be coordinated through Administrator and Worker, with the actual I/O taking place via TCC and the MPM. The first priority is capture of the buffer control structure and the input buffers. If things are in a not-too-damaged state, output and working buffers may also be capture, or even a full memory dump with a view to off-line debugging, rather than a simpler attempt at playback.

Administrator will do the primary handling of all these messages except *reset*. After determining the required action, it will inform Worker by writing into Worker's inbox and provoking an interrupt via VME to force Worker to look at the message.

9.9.3 Input Event Interrupt Routine in Administrator

The main interrupt routine in Administrator handles the arrival of a new event. So that data can flow with minimal impact on other processing, data is written to memory by DMA, and the end of the event data triggers triggers an interrupt routine, which promptly places the system in a state to receive another event. After the interrupt is handled, normal processing resumes. The following steps take place during the handling of this event interrupt.

- on fifo_empty for all Alpha cards participating, interrupt occurs
- record the length of the transfers + padding by reading Block Transfer Address registers (Perhaps only during commissioning; perhaps in Administrator but not in Worker)
- Oldest Allocated Buffer location set in Mapper slot
- after an appropriate wait, or if necessary read the Worker(s) memory to verify they have reassigned the mapper, send GO to the MBT Pilot card for the next input event

Other less time-critical functions listed above in the (such as communication with TCC) will be also implemented as interrupts as a convenience of implementation. The interrupts are listed below in section 9.10.

9.9.4 Event Processing in Administrator

Event processing:

- wait for next event [25]
- verify all inputs refer to same event (3B L1 crossing number)
- verify correct transfer length by computing trailer position
- select Buffer from free list to be Allocated

Administrator's answer response could take up to 5 μs, as 2 MBus writes and one MBus read are required.

- get answer from Worker
- verify same event as expected, and Worker event pieces matched
- if Passed event, get full 16B L2 Answer and L3 R/O lengths
- save Buffer number from Worker answer
- send decision to L2 HWFW
- Respond to Worker with Buffer Allocation
- If *Collect Status* event, capture scalers and messages.
- If passed event, prepare Admin readout (if any)

[25] Admin could poll to detect *SCL Initialize*; will do via VME interrupt instead.

- If *UBS* or *Forced Write* event, prepare special Admin readout
- Wait for decision from L2HWFW
- check decision against that returned by the L2HWFW
- If failed event, mark Buffer as Free
- If L3 Output Buffers all full, raise L2 Busy and wait for present readout to complete.
- If L3 write, put on L3 output list

9.9.5 Event-Asynchronous VME tasks for Administrator

The term "Asynchronous" is used a bit oddly here, to indicate that the VME processing of an event is not synchronized with the rest of the event's processing. It may be initiated at the end of processing an event, or even the event in question, but since the VME transfers take place while other events are being processed, the end of VME processing might not be noted until the end of a later event. Since VME processing is buffered, and does not require absolutely optimal use of the VME bus to reach its bandwidth goals, this rather inefficient use of VME bandwidth is likely to be sufficient. The .3KB output size envisioned would take 30 μs per transfer, and termination would be checked for every 100 mus on average, more than adequate give the 3% bus load. A perhaps more realistic 1KB output block still uses the bus 10% of the time, with the rate of checking perhaps lowering the effective bus bandwidth to give 20% effective occupancy, which the 8 output buffers should control.

These are all tasks requiring use of VME. Thus, they have to be coordinated with L3 readout. This series of items can be checked after event processing and the answer response are complete, and possibly in other waiting periods.

- check to see whether L3 readout of an event has finished
- if so, put Buffer on free list
- if monitoring transfer is pending, set VME windows and move monitoring data to MPM[26].
- If events in the L3 queue, start readout of new event, which requires adjusting the VME window of the Worker and Administrator.

[26] Could poll for TCC messages here, but simpler to handle them by interrupts.

9.9.6 Handling of *SCL Initialize*

SCL Initialize is a request to re-initialize buffer handling. Administrator is notified of the request by an interrupt generated on the MBT card[27] All current Filled and Processed Buffer must be dropped in both Worker and Administrator. Notification of Worker takes place via the same mechanism as that chosen for TCC messages, with a handshake back verifying completion of the task in Worker. When the Buffer management is reset, L1 Busy is lowered by Administrator.

9.10 Interrupt Usage

The following is the list of interrupts used by Administrator:

- Timer: Used by debugger What is minimum frequency to maintain contact? Can the frequency be reset by a TCC message asking for a routine to be called? Used to prevent surprise rollover of precision timer Apparently, 2 separate interrupts: how to distinguish
- Reset: Provoked by Front Panel Button or TCC: forget everything and reload
- TCC Message: The full collection of begin/end run and miscellaneous messages
- *SCL Initialize*
- New Event/Fifo Empty

The corresponding list for Worker is

- Timer
- Reset
- Administrator Message: Includes relayed TCC messages and *SCL Initialize*.
- New Event/Fifo Empty

9.11 Global Worker

Global Worker is in one sense a simplified version of Administrator, relieved of Buffer management, communication duties associated with L2 input, L3 output, and direct communication with TCC. It performs a subset of the data integrity checks done by Administrator. Of course, the real purpose is to perform actual analysis of the data, which Administrator does not. The data analysis is performed under control of the L2 Script runner.

The list of messages for Global Worker can be derived from the list of messages for Global Administrator, as Administrator is party to all messages in L2.

[27] Polling MBT registers to find the *SCL Initialize* request seems more susceptible to deadlocks.

The Input Interrupt Routine of Worker is a subset of that of Administrator. The main task is to place the Oldest Allocated Buffer location in the Mapper slot just used. During commissioning, Worker may also do more checking, and may have to write a message to Administrator verifying that it has finished the assignment.

The event processing list for Global Worker is much simplified from that of Global Administrator, and there are no "asynchronous" tasks for Worker[28].

Event processing:

- wait for next event[29]
- verify all inputs refer to same event (3B L1 crossing number)
- run L2 Script Runner
- If *UBS* or *Forced Write* event, prepare special readout
- if Passed event, prepare full 16B L2 Answer and L3 R/O lengths
- If *Collect Status* event, capture scalers and messages and prepare full scaler block using script.
- send Answer to Administrator and wait for reply
- save Allocated Buffer number from Administrator reply

L2 Script Runner supervises the calls to L2 tools which implement the scripts required by any L1 bits which are passed, and records the results in a L2 bit mask. It also does record keeping sufficient to calculate the pass fraction for each bit, each step of the script of each bit, and the global pass fraction for L2 overall, and for each of the filtering tools. To run swiftly enough, this record keeping will likely consist at most of a single counter increment for each script, recording which tool failed, or if the script passes, recording that fact. Combining these scalers with detailed knowledge of the script, the tries and passes for each step along the way can be deduced when a Monitoring block is requested. Timing information will likely be recorded only at the overall filtering level, though the simulation framework may record more detailed information to assist algorithm tuning.

We will not further describe L2 Script Runner here, as the details are being designed. It will be based on understanding from L3 of the previous and current run, suitably restricted to operate in the L2 time budget.

[28] Using an interrupt for notifcation of the arrival of a message relieves Worker of checking inboxes for messages.

[29] Could poll to detect *SCL Initialize* or relayed TCC messages, but more reliable for Administrator to use an interrupt.

number	compile or startup
VME Addresses	compile or startup
VME Offset	event answer
lengths	event answer
Buffer	event answer

Table 7: Information for L3 readout: Logical

9.12 How L2 Global Reads Out to L3

At the beginning of a run, the VBD must be informed of the *number* of all the locations from which it will read, and the *VME addresses* of these locations. Every event, the VBD must be informed of the *lengths* of the blocks at each of these locations. The VBD reads these *lengths* from a fixed VME address.

Since Administrator handles all interactions with the VBD, Administrator must know all three of these items. We take as a design principle that the Worker does not know when it is being read out, and that event data (for normal events) are copied only once, via VME into the VBD. Again mirroring the DAQ frontends, we reserve up to 8 Buffers for events awaiting L3 readout. This means that Administrator also has the job of redirecting the *VME offset* needed so that the *VME addresses* point to the correct event. So Administrator must know the MBus (or VME) location of the Universe Chip mapping registers so that it can reset the *VME offset* of each Worker before readout begins for an event [30].

Table 7 indicates when the information just described can be acquired.

Acquiring the first two items at compile time introduces coupling between the executables of Administrator and Worker, so it may be desirable to acquire the information during a "startup" dialog[31] . The second two items could be acquired directly or indirectly. Direct acquisition would have them be part of the Worker's Answer for the event, and sent along with the Buffer

[30] Only 1-4 VME windows available on an Alpha. To be read out using a small number of VME offsets (preferably 1, as the other windows might be used for monitor or control), an event must be fairly concentrated before it can be read out. The *VME offset* cannot be reset during readout, so the location of L3 output Buffer pieces with respect to the offset must be the same for all L3 output Buffers. This is simplest if the number of readout fragments is small, preferably 1, or 3 at most (header, normal output, UBS output) This is not a serious burden as the need to flatten pointers to array indices implies that output data structures must be rebuilt in order to be written. This has some implications for how structures are declared. One can define a massive output structure Out, containing various parts, and allocate an array of such structures indexed by Buffer number. But one cannot define a separate array of structures indexed by Buffer number for each part independently. The common-offset requirement tends to make the output structure a global quantity.

[31] We assume the VME readout location list is independent of trigger setup, so that information may be acquired at the first execution of the program ("startup") rather at the beginning of each run. This needs to be repeated whenever a new executable of any node is downloaded.

number	compile or startup
VME Addresses	compile or startup
VME Offsets[Buffer]	compile or startup
MBus Offsets[Buffer]	compile or startup
Buffer	event answer
lengths	MBus Read

Table 8: Information for L3 readout: Physical

number. A simpler message from Worker to Administrator results if just the Buffer is passed. [32]

The Buffer number could also serve as a pointer to the other pieces of information, simplifying the response format (at least for passed events). To use this pointer, the Administrator needs more information about Worker, namely the *MBus offsets* for each Buffer at which it can find the *VME Offset* and the *lengths*. Again, one could acquire this information (for each Buffer) at compile or startup time.

Two solutions emerge: one with compile-time coupling but simple code, and the other with more complex startup code but less compile-time coupling. The revised picture is shown in Table 8. Either choice is possible. Acquiring the information at startup time seems to offer better insulation of the Administrator from Worker's less stable executable, but the exposed locations likely to be quite stable and insulated from changes in user algorithm code.

The *length* of data read can be zero for some locations if a location is read only for *Unbiased Sample* events, but we plan to build Worker's L3 output data for any event in a single block.

Administrator builds the L3 transport header, while the VBD itself builds the trailer.

Once VBD has been given a GO, its status can be checked by lines sent to either the FRED port or MBT I/O lines.

9.12.1 Event Processing with Multiple Workers (Global)

Event processing changes in more complex scenarios:

Two Workers working on alternate events is straightforward: the Administrator takes turn responding to each Worker, and has to Allocate more than one Buffer at a time. Such strict alternation does not achieve a linear increase in allowable throughput, as was discussed in section 8. If Administrator includes in its reply which Buffer to analyze next (possibly a Buffer which has not been filled yet), and takes on the complexity of serializing out-of-order Worker Answers, the added computing power could be used more efficiently. The Workers will receive a number of Allocated buffers in response to an event Answer which varies, depending on the number of events that other Workers have finished. Care must be taken to verify that the Buffer management scheme still guarantees sufficient Allocated buffers even to a Worker which is slow to finish an event.

Two Workers each doing part of a Global event is more like the calorimeter preprocessor situation[33]. However, the Global Workers know whether they individually have passed any bits, so they can prepare for readout every event, as their preparation is likely short if they have no passed bits, and is necessary if they have passed a bit. This avoids having the Worker reply a second time to Administrator to indicate that they are prepared to read out if the event passes. Administrator must, of course, wait for all Workers before publishing the L2 decision.

Collection of monitoring information (see the next section) becomes more complex when multiple workers are involved. Some complexities might also occur in combining output of multiple nodes for L3 or worse for L2 if a single MBT input is shared between 2 nodes, as it would be complex to combine header information to be followed with data from each node.

Queing simulations are under way to explore how much might be gained by such escalations. Amdahl's law for the speedup factor with N=2 nodes and a fraction of parallelization f,

$$S = \frac{1}{1 - f/N},$$

warns that splitting a single event across processors is likely to provide small gains unless most of the work is split on nearly all the events. Even without queing losses, the expected speedup is only about 1.3 for two nodes which split 70% of the work 70% of all events ($f = .49$). Queing losses will most likely limit the alternating node scenario, as slow events stall both nodes[34].

9.13 Monitoring Information Collection

Monitoring information is captured after the processing of an event marked with a *Collect Status* Qualifier. The information captured typically consists of all scalers, circular timing buffers, and current Buffer occupancies in nodes and MBT's. The scalers may re-

[32] The Mapper slot number need not be passed each event, as synchronization is verified by comparing event numbers between Worker and Administrator.

[33] A preprocessor can't know the L2 decision, as L2 Global hasn't even seen the event yet. The calorimeter Workers will have to prepare for readout every event, and calorimeter Administrator will have another asynchronous task of looking for L2 answers to perform once per event. As a result, in the calorimeter preprocessor, Buffer freeing is delayed and the L3 output Buffer list may have to hold up to 16+8=24 events.

[34] A considerably more complex scenario would escape strict alternation, but require that Administrator serialize responses to the L2 HWFW (which requires answers in strictly the order of L1 accept messages), and Allocate variable numbers of Buffers to the Workers, reflecting that the number of events processed while a Worker finished an event now varies.

quire some (considerable) postprocessing to become intelligible. This postprocessing should be done by L2 Global Worker, which already owns the script information needed to postprocess. The monitoring information is fixed-length, regardless of the actual number of L2 trigger bits currently defined. The Worker copies the captured information to a public location known to Administrator before sending its event Answer, and marks the Monitoring information with the event's 3B crossing number. Administrator will move all monitoring data via VME to the MPM for TCC's perusal.

The ERRMSG Error information is of variable length. It is handled similarly, except that Administrator must first read the length of the buffer before transferring. Administrator concatenates Errors from Worker(s) and Administrator in MPM, and appends them to any messages that TCC has not already read. After transmission, the Errors length is zeroed by Administrator.

Since monitoring information is collected relatively rarely, there is no reason for publishing its MBus location. The VME location could be acquired at compile time or, for less coupling, at startup time. The VME address is only meaningful with respect to some VME Offset in Worker's Universe chip; this offset must be known to Administrator.

Monitoring information may be requested by TCC directly instead of via the *Collect Status* Qualifier. In this case, Administrator must ensure that information is transferred in a few tens of milliseconds, whatever VBD might be doing on VME.

This scheme avoids double copying of the monitoring data, as would be the case if the monitoring information were part of the Worker Answer, or a response to a specific request from Administrator. The cost is the need to know the location of the Worker's monitoring and message buffers.

MBus and VME[35] busy fractions may be monitored by using bus mastership lines as scaler gates, but this may prove to cumbersome to set up permanently.

TCC collects the monitoring information from the L2 subsystems, timestamps it, matches blocks across subsystems, and, by calling subsystem-specific routines, subtracts to present Delta-T and Difference over Run information. Higher-level monitoring processes on the Host system mask run data down to trigger bits owned by a requested run.

9.14 Coupling of Administrator and Worker

It seems desirable to have changes in Worker not require recompilation of Administrator. A second goal is to have a simple mechanism for growing code of an extra Worker node from that of a single Worker (or from a single master source). The mapping windows inherent in the MBus and VME interfaces help in this, by providing "logical" locations seen from outside, which can be moved to different "physical" locations. Table 9 lists the Worker locations which need to be known by Administrator, and vice versa.

Node	Location	Address	Binding
Admin	Event Inboxes	MBus	Compile
Admin	Begin Run Inboxes	MBus	Compile
Worker	Event Inbox	MBus	Compile
Worker	Begin Run Inbox	MBus	Compile
Worker	Set VME Offset	MBus	Compile
Worker	VME Interrupt ?	VME	Compile
Worker	L3 VME Offsets	VME	?
Worker	L3 *lengths*	MBus	?
Worker	L3 MBus Offsets ?	MBus	?
Worker	Monitoring and Errors	VME	?

Table 9: Public Locations of Worker and Administrator

The Worker Begin Run Inbox is where messages relayed by Administrator from TCC appear to Worker. The Administrator Begin Run Inboxes are where replies from Worker can appear. These same inboxes would be used in any dialog at startup time.

9.15 State Diagrams for Administrator and Worker

Figure 9 above shows a simplified state diagram for Administrator. The error states and interrupts have been omitted for clarity. In the Startup state, the executable is downloaded and establishes communication with TCC and Worker. The Idle state waits for TCC to provide information to enable data taking. When the run is fully set up, the Wait/Event state is entered, until an event fills an input Buffer. In Process Event, Administrator performs the format checks, then enters Wait/Answer until Worker replies with its analysis of the event. In Send Event, Administrator reads the Answer, verifies that Worker was working on the same event, and sends the Answer to L2HWFW. If the event is to be written, Format L3 Data is entered; if not, Wait L2HWFW is entered directly, and Administrator polls for the L2 SCL message verifying that the Answer broadcast by the L2HWFW matches the decision sent. In Manage Buffers, Administrator Allocates the next Buffer for Worker. If the event is not to be written, the Buffer is now Free; otherwise, an attempt is made to place the event in the L3 output queue. If the queue is full, Administrator waits for L3 to finish read out of an event. In Manage VME, the status of the VME bus (busy or VBD finished) is ascertained. If VME

[35] VME busy fraction is more difficult to measure than MBus busy fraction. MBus masters must drop mastership to allow arbitration, unlike in VME.

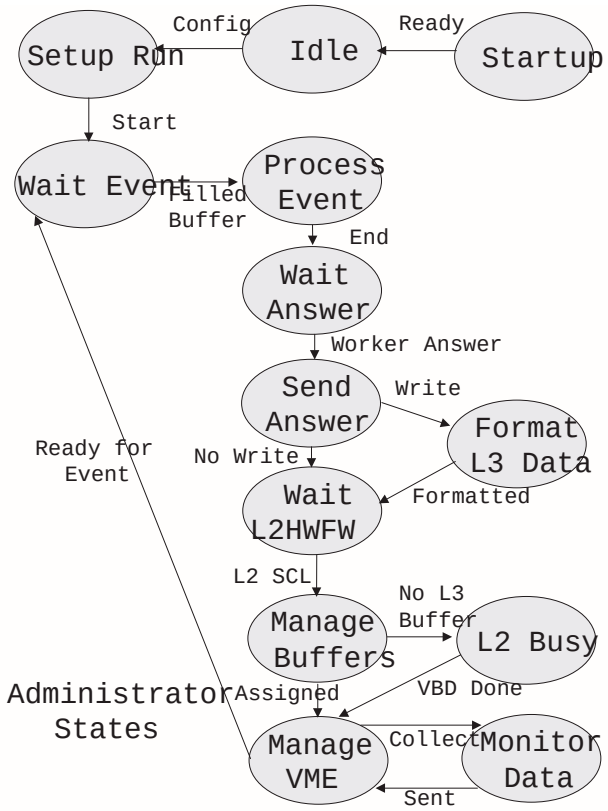

Figure 9: State Diagram for Global Administrator

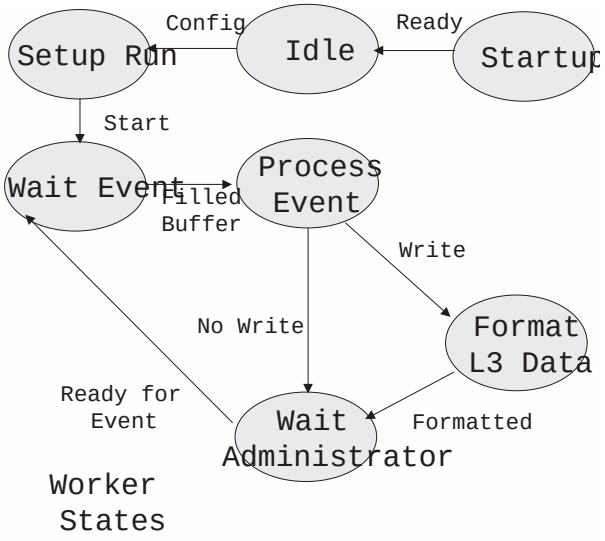

Figure 10: State Diagram for Global Worker

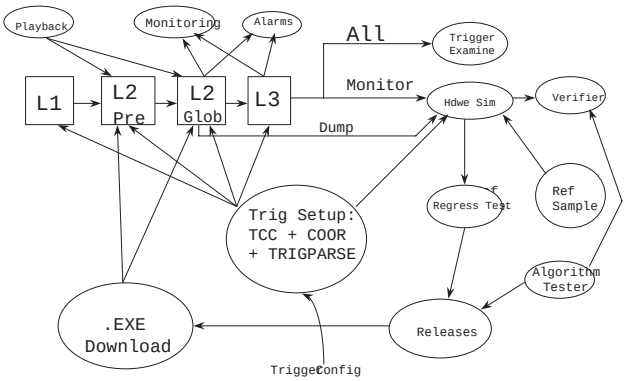

Figure 11: L2 Software Overview

bus is free, any pending Monitor data is sent to the MPM for TCC's perusal, the next event in the L3 output queue is handed to the VBD, and the cycle repeats. This is a rather linear scheme: Input and L3 Output take place during event processing, but L3 readout and buffer management takes place at a well-define point in the event cycle. If performance requires, more vigorous polling of VME status is possible.

Figure 10 shows the corresponding diagram for Worker. Again, error states and interrupt handling are excluded. Worker's state diagram is similar, but simpler, as Worker is relieved of many communication and management tasks. The basic initialization is the same for Worker as for Administrator, though the details of the actions and the communication mechanisms differ. All processing and formatting take place before the Answer is sent to Administrator, at least for the scenario where an event is fully processed in a single Worker. Worker stalls until Administrator has conveyed the decision to the L2HWFW and verified its transmission (though event input may continue in the background). When Administrator replies, Worker has the Buffer information it needs to proceed, and looks for the next event.

10 Offline Software

Figure 11 shows an overview of the offline software needed to complete the system and its verification.

The scope of the tasks are understood from Run I experience. The software design has not yet begun, but the design does not impact the online software in a major way, provided that adequate provision is made for collection of the data for verification. We regard verification and regression testing as vital to the reliable operation of an online trigger.

Downloading of the trigger configuration and creation of the executable is discussed briefly in the next section.

The online results of events which pass L2 and L3 filtering are available for histogramming online ("Trigger Examine"). These distributions perform basic sanity checks, but the design of good diagnostic plots which are not dominated by the large rejection factors, and stable against changes in trigger configuration, is not a trivial task.

These same histograms can also be part of the regression testing package to compare successive versions of the trigger algorithms between releases. This requires considerable infrastructure: a relevant data sample (which must be cajoled from the physics interest group users as well as code implementors), and automated or assisted comparison facilities.

Part of the definition of the trigger code is a self-testing package, provided by the author, which can be used as part of the suite of regression tests for a release. The development and release-test environment should also include extra checks in the code, especially in the common utilities. These can take the form of assertion tests which verify the assumed preconditions and postconditions of each module.

An important consideration is the selection of an appropriate mixture of *UBS* events, events which are written to the output stream regardless of the nominal L2 decision. Criteria[17] for setting *UBS* fractions include the time required to verify that a given trigger output rate has a bound on its fractional error. This requires a separate *UBS* fraction for each L1 bit in the L1HWFW. Simply dividing the allowed *UBS* bandwidth equally among all triggers is a robust scheme which does surprisingly well according to this criterion, though if the variation among the pass fractions of the various trigger bits does not vary too widely, a more optimal rule can allocate bandwidth so that all triggers are checked to the same fractional accuracy. All criteria require some iteration to adjust properly, as the available knob, the fraction of input triggers of a given trigger bit to declare *UBS*, depends on the vaguaries of the actual input rate, its luminosity dependence, etc.

The *UBS* events are used by verification programs which run the simulator off-line to be sure that the on-line decisions are correct. These checks can detect hardware failures, problems caused by real-time aspects of the processing, and problems caused by history-dependent bugs which are very difficult to detect in any other fashion. The verification programs also verify that data was transported successfully, by comparing the L2 input data with corresponding data elsewhere in the data stream; this detailed data is suppressed on normal events.

The simulation code should also be capable of running on dumped events; the developers' simulation environment should have a goal of reproducing most problems found online, as online debugging time is a scare resource in a system which cannot analyze beam data at the same time as code is being debugged. This implies considerable care in mimicking the only memory layout.

11 Downloading

11.1 Executables

Executables are understood to include data movement and "framework" code, algorithm code, and data such as computable geometry or lookup tables needed by algorithms. If additional information such as geometry or calibration files are needed, they are read by TFTP from TCC or a Unix host disk once the executable is downloaded.

Executables are built on an Alpha Unix workstation, from code under release control. The code is served from the disk of either TCC or an Alpha Workstation. The transfer takes place by TFTP (a subset of FTP) by ethernet, from a filename in a database keyed by the hardware ethernet ID of the ethernet card on the Alpha board. This allows the boot ROM code for all Alpha boards to be identical.

11.2 Scripts and Parameters

ASCII files control the selection of the trigger processing in L2 Global. The files contain both structural information, the script for each individual L2 bit, and the parameters of the actual cuts made in the course of applying the script. The script will most likely be implemented by calls to a series of L2 filtering tools, which are made aware of the parameters needed. These files are translated from a high-level language (with default values understood, for example) to a lower-level, but human-readable ASCII format sent by COOR to TCC. Code in TCC translates these into data structures; the same code can be used in the off-line simulation of the L2 trigger. Then the data structures are turned into byte patterns which are deposited into specific memory locations in the Bit3 MPM under TCC's control. Administrator is notified that a new download of scripts and parameters are available, and Worker and Administrator ingest the information, and verify that it was received and understood.

The need for making Script Runner fast implies a few things about the organization of L1 and L2 bits. L2 bits which refine a single L1 condition are implemented by programming several L1 bits with the same conditions. These bits should be adjacent to each other, because otherwise it slows down the process of searching through the 128 bits to find ones which L2 should process. This implies the need for access to L1 and L2 decision results by trigger name, rather than raw bit number, so that the trigger list can evolve without disturbing users wishing to select a given trigger condition.

L1 bits with no L2 script are defined as passing L2, and L2 Script Runner in L2 Global forces this to be

the case. This supports a trivial L2 which does nothing, which handy for testing, calib (where L2 has not much to offer by filtering), and automatically passing Heartbeat events without processing.

12 Control

TCC's interactions with Global at run boundaries has been previously discussed in sections 9.9.2 and 9.12.

Heartbeat events are sent periodically by the L1HWFW if no other trigger has fired recently. Their purpose is to constantly verify the readiness of the readout chain. As they are sent on a distinct reserved trigger bit, their handling can take place by the script mechanism in Worker, simply forcing a L2 Pass on the Heartbeat bit. Most likely this mechanism is the trivial one of specifying no script: L2 Global will interpret this as no selection criteria, meaning that the event passes.

13 Monitoring

13.1 Counters and Slow Monitoring

Monitoring is "slow" when it takes seconds to appear at its destination; "slow" monitoring is not necessarily monitoring a slow process.

Rates in L2 Global will come from counters held in Worker which are seized by Administrator over VME after *Collect Status* events. L2HWFW and TCC supply the absolute time scale and live fractions.

Latency is measured by comparing the crossing number when a L1 accept was sent out with the crossing number when the L2 answer came back. L2 Global will store these latency times in a circular buffer.

Circular buffers of times (latencies, or the time-in-state recorded by the state machine as it makes transitions), allow calculation of a *distribution* of latency, processing time, or other state dwell times.

Instantaneous Buffer occupancy can be siezed during slow monitoring by reading the L1 HWFW (which has a count of the events awaiting decision *anywhere* in L2), SLIC and MBT buffer occupancies, and saving the Alpha L2 and L3 buffer occupancies. These can be used to drive pac-man (pie chart) displays.

13.2 Fast Monitoring and Fraction of Time in States, and Distributions

"Fast" monitoring reflects changes essentially in real time. L2 Global has two main sources of fast monitoring.

The "Fred" registers on the Alpha boards present the current processing state and the current number of events in the buffers for scaling or logic analyzer viewing. They are used as outputs by the state machine code. One particular state of interest is the time spent processing an event. By using a decoded state bit as a scaler gate in L2HWFW, the fraction of time spent in any state may be found. By further knowing the average time between L2 events, a scale could be put on this fraction, resulting in an estimate of the average L2 processing time, independent of any internal timers in the node. However, if a distribution of the processing time is needed, for example to look for long tails, the circular buffer method mentioned above would be necessary.

Similar information will be presented by the MBT cards, and, globally for any buffer in the L2 system, from the L1HWFW. These come on 4-bit lines decoded into 16 bits to sets of L2 HWFW scalers which thus make histograms of how often n buffers were occupied at each site, or how often the system was in a given processing state. These, too can drive Pac-Man displays provided by the L3 monitoring package. By taking differences between successive monitoring periods, the averages over the previous monitoring period can be displayed instead of averages to this point in the run.

These same scaler gates can be used instead as test points for logic analyzers.

14 Commissioning

There are three phases of commissioning: standalone subsystem tests, integration tests at individual institutions, and integration with the actual hardware at DØ .

14.1 Standalone Testing

We briefly suggest some of the tests which can be performed without special test jigs, just system components and L2 VME crates.

Low bandwidth standalone testing of the MBus functions of an Alpha card is possible by writing MBus data (from the MBus programmed I/O) to an MBus broadcast address, and reading it with the MBus Block Transfer engine. With two Alpha cards, one can test the MBus Programmed Input function. A low bandwidth test of a system of Administrator and Worker(s) is possible by adding an extra Alpha to act as a MBus data source.

The MBT can be partially tested "standalone", or rather with assistance of a VME master. Some likely VME masters are a PC (such as TCC) using the Bit3/MPM, or an Alpha card, using the Universe chip. Depending on whether the MBus Programmed I/O Interface is available as a PCI card, and which MBT registers and inputs or outputs are visible from VME, some testing of MBT functionality can be done without an Alpha. For example, the MBT outputs can be used to feed MBT inputs, using data downloaded into the output registers. However, testing the MBus functionality of even a single MBT card will require an Alpha,

and testing the full functionality will require a test jig simulating the SCL input.

A crate holding only Alphas and the MBT can be self-tested. Memory holding up to 16 events (including SCL and L1HWFW information) may be downloaded, via VME, and then the system run at full speed.

14.2 Integration Testing

A preprocessor crate can feed a Global crate via the normal inputs. This requires coordinated fake data for the *e.g.* SCL and L1HWFW information.

14.3 Installation and Commissioning

At the point equipment moves to DØ a TCC is clearly needed, as well as live inputs. However, considerable work can be done with a test stand consisting of several crates with Global and a few crates containing typical preprocessor equipment. It is intended to maintain such a test stand permanently at Fermilab to facilitate debugging and testing of modified equipment and programs.

15 Desirable Software Features

In this section we consider some software features which seem desirable, but which may not be implemented because of manpower limitations. These features are desirable because of the flexibility they offer for online testing and debugging. Implementing them without impacting online performance or reliability is a concern, because of the complexity they imply.

15.1 Playback

The offline simulation will be capable of running on dumped events. Playback takes this one step further, by downloading the event into the actual hardware, or the test stand hardware, allowing debugging to be performed at more convenient times with a more accurate reflection of actual online hardware and memory layout. The technical challenge is to inject the data into Alpha memory or the MBT test data inputs.

15.2 Shadow Running

It is difficult to test algorithms online in L2, because the system is fundamentally a single processor, rather than a farm of equivalent processors. Shadow running would introduce optional duplicate Worker(s), operating on the same input data stream as the regular Worker(s).

In a simple scheme to allow online debugging, Administrator treats the Shadow Worker like a mirror of the Worker: each analyzes the same event at the same time. However, if a crash occurs, Administrator leaves the Shadow Worker in the crashed state, stops it from receiving new events, and ignores it until the Shadow Worker is restarted. Administrator dumps the event, restarts the regular Worker, and resumes normal processing without the Shadow Worker. This allows the Shadow Worker to be debugged without serious damage to data taking. Restarting the Shadow Worker requires resynchronizing buffers among Workers.

A variant of this scheme implements multiple Workers with the number of Workers deliberately chosen to be more than needed. Administrator hands out events to each Workers in turn. If one of the Workers crashes, Administrator removes it from the event distribution list until it had been debugged and restarted.

A considerably more ambitious implementation would allow Shadow Worker to perform different processing than the regular Worker, which would allow online testing of new code. The advantage of such testing is that it allows new code to be exposed to orders of magnitude more data than what is easily available for offline testing: recall the L3 output bandwidth is 1/1000 of the L2 input bandwidth. The technical challenge, which may be insuperable, is to allow the Shadow Worker to parasite off the input data stream without serious impact on the regular Worker's throughput. This is quite difficult because all Alphas listen to all data in Broadcast mode, and it is difficult to be both synchronized in Buffer management and non-interfering.

15.3 Test Events

One can imagine a self-test mode of L2 in which events with known decisions are injected into the input stream. This relies on much of the same technology as would be required for Playback, but would require in addition some code decide when to inject an event, and other code (perhaps just in Administrator?) to recognize the event as a test event and verify the result. The self-test could be performed on request from TCC or even periodically during data taking, if the mode-switching of the inputs from real to fake data was not too disruptive or dangerous.

A Further information on the MBT

For a full description, see [12]. Any of the Control or Test registers can be read by either MBus or VME.

A.1 Output

Each card has 2 output paths. These are used by Preprocessors to send data to L2 Global. Data may be sent to either of two address spaces, filled by either writing sequential addresses or writing repeatedly to the same address from MBus. The data path may be 8B (64b) wide, rather than the full 16B (128b) MBus width. End

of transmission is signalled by writing to a different register.

A.2 Control

All control registers are visible from MBus, and can be read back. The main control registers are

- initialize
- select active input channels
- set channel Source ID

  If no Source ID for a channel has been assigned, then that input channel is disabled. The SCL L1 information channel may be deselected on this card by this mechanism. If a channel is selected, then its information is required for every event. This controls the broadcast address on MBus used for each of the inputs. Setting the Source ID to the same value for all channels makes the MBT card appear as a single broadcast source.

- GO

  This gives the MBT card permission to broadcast the next event, once it has a complete event, and has gained bus mastership. The card allows arbitration between each source, even if the sources have been set up to use the same broadcast address. After the event is complete, the card signals this fact to MBus and/or its neighbor cards and cedes bus mastership.

A.3 Monitoring

The number of events in any FIFO can be read by MBUS. The same numbers appear on a connector which can be inspected by a logic analyzer, or used to gate a scaler in the L2HWFW.

A.4 Test Data

Test data can be sent to any FIFO input, or any output. Up to 16 events may be loaded for each location. At a signal, data can be made to flow at full speed.

B Inputs to L2 Global

In this Appendix, pp is used as an abbreviation for L2 Preprocessor. Some issues which are not yet resolved are set off with square brackets [ ].
Assumes packing such that:

- smallest fragment size for a variable within an object is **1 byte**
- object's total data size must be an **integer number of 4-byte words**
- multi-byte fields do not cross boundaries between 4-byte words
- each pp will preface its data with an twelve byte header
- each pp will complete transmission with a four byte trailer
- each pp will add bytes after the trailer to make a total **number of bytes divisible by 16**.
- where possible each pp object will be identified by rapidity, azimuth and E_T in that order
- objects will be sorted in descending Et order
- the least significant bit for rapidity will be 0.05, for azimuth $2\pi/160$, and for transverse momentum 0.25 GeV/c

  [what is the origin for eta? Do we try -n to n, is 0 skipped, or force all eta indices numbers positive?]

- azimuth will be reported in standard DØ coordinates

[One could either arrange so 1 header word is fixed, or so the trailer word is identical to one header word.]

Preprocessor Header	
Item	Bytes
Header Length	1
Header Format	1
Object Length	1
Preprocessor ID	1
Rotation Number (high)	1
Rotation Number (low)	1
Bunch Number	1
Status	1
Version	1
Run/Event Switches	1-2
Number of Objects	1-2

Preprocessor Trailer	
Item	Bytes
Rotation Number (high)	1
Rotation Number (low)	1
Bunch Number	1
Preprocessor ID	1

e,γ		jet
Item	Bytes	Item
η	1	η
ϕ	1	ϕ
E_T	2	E_T
Spare	1	Spare
η_{center}	1	η_{center}
ϕ_{center}	1	ϕ_{center}
Iso frac	1	Spare
EM frac	1	Spare
Total ~ 2 obj/evt $\Rightarrow$ 16 B/evt	8/evt	Total ~ 4 obj/evt $\Rightarrow$ 32 B/evt

$\not\!\!E_T$ Calpp	
Item	Bytes
$\not\!\!E_{Tx}$	2
$\not\!\!E_{Ty}$	2
Total	4 B/object
fix: 20 obj/evt $\Rightarrow$ 80 B/evt	

The $\not\!\!E_T$ data are reported by rapidity bin so that any vertex knowledge can be applied in the Global Processor. The order of the bins is from $\eta = -4$ to $+4$, each bin being .2 (one trigger tower) in width.

CFTpp	
Item	Bytes
Spare	1
$\phi(Shower\ max)$	1
p_T	2
$\phi(A\Phi)$	1
CPS Energy/Isolated/sign	1
Spare	2
Total	8/obj
~ 50 obj/evt $\Rightarrow$ 400B/evt	

The CPS and FPS data are not well defined at this point.

CPS, FPS	
Item	Bytes
η	1
ϕ	1
E	2
Spare	4
Total	8B/object
$\sim 5 + 5$ obj/evt $\Rightarrow$ 80 B/evt	

μ PP	
item	Bytes
η	1
ϕ	1
p_T	2
Algorithm	1
Quality	1
TOF	1
Direction	1
Total	8/object
~ 11 obj/evt $\Rightarrow$ 88B/evt	

TOTAL PP's $\rightarrow$ GLOBAL:

Mean variable output to the L2 Global Processor

Totals	
PP	variable Bytes/evt
L1 HWFW	16
L1 SCL	5xx
e/γ	16
jet	32
$\not\!\!E_T$	80
CFT	400
$\mu(1+2)$	88
CPS	40
FPS	40
STT?	0
Headers	160
Total:	888

C L2 Output to L3

[Headers to L3 for preprocessors, Global are not yet specified. Likely place for version # of executable.] Output of L2 Global consists of:

- Preprocessor OBJECTS
- L2 Trigger decision information for each of 128 bits
- L1 Trigger decision for each of 128 bits
- other L2 status information, including "true" decisions for *UBS* or *Forced Write* events.

The objects lists follow as does a total estimate for the L2 Global output. All rapidities are in detector coordinates unless vertex information is available. In that

case coordinates are given in physics rapidity. All E_T are uncorrected.

[why uncorrected E_T?]

[Are Zvtx objects sent, with a tag for "preferred" one?]

[do muons attempt to point back to mu candidates?]

[do leptons point to both tracks and lepton detectors?]

[why does the resolution on em and iso get larger?]

[no allowance for high-order objects, such as masses]

e,γ	
item	bytes
η	1
ϕ	1
E_T	2
EM frac	2
Iso frac	2
Preshower	1
Track info	4
E/p	1
η_{center}	1
ϕ_{center}	1
Spare	1
Total	20 B/object
var: $\sim$2 obj/evt $\Rightarrow$ 40 B/evt	

jet	
item	bytes
η	1
ϕ	1
E_T	2
η_{center}	1
ϕ_{center}	1
Spare	2
Total	8 B/object
var: $\sim$4 obj/evt $\Rightarrow$ 32 B/evt	

$\not\!\!E_T$	
item	bytes
$\not\!\!E_T$	2
$\not\!\!E_{T\phi}$	2
$\not\!\!E_{Tx}$	2
$\not\!\!E_{Ty}$	2
Total	8 B/object
fix: 1 obj/evt $\Rightarrow$ 8 B/evt	

[Are uncorrected sums also sent?]

μ	
item	bytes
η	1
ϕ	1
p_T	1
Quality	1
Track info	4
Spare	4
Total	12 B/object
var: $\sim$5 obj/evt $\Rightarrow$ 60 B/evt	

Totals	
object	variable B/evt
e/γ	40
jet	32
$\not\!\!E_T$	8
μ	60
L2 Decisions	16
Candidate Pointers	100
L1 L2 L3 evt#, flags	32
Total:	288

References

[1] The L2 trigger documentation can be found on the DØ WWW page *d0sgi0.fnal.gov/* under "Technical —DØ Upgrade —Trigger Systems" or more directly under *www.msu.edu/ d0/l2/*. A recent short overview is J. Linnemann, "The DØ Level 2 Trigger", in proceedings of Realtime 97 IEEE symposium on Nuclear and Particle Physics, Beaune, France, September 20-26, 1997, available at *http://www.pa.msu.edu/hep/d0/l2/*

[2] J. Blazey, "The DØ Trigger", in proceedings of Realtime 97 IEEE symposium on Nuclear and Particle Physics, Beaune, France, September 20-26, 1997, available at *http://www-d0.fnal.gov/ blazey/upgrade.html*

[3] The DØ Upgrade: Forward Preshower, Muon System, Level 2 Trigger, March 27, 1996; Fermilab-FN-641 web address below. See also J. Linnemann, Presentation of L2 trigger to Fermilab Program Advisory Committee, April 19, 1996 in *www.pa.msu.edu/hep/d0/l2/*.

[4] The DØ Upgrade, submission to the Fermilab PAC, April 17, 1995.

[5] The L2 hardware framework is described at *www.pa.msu.edu/hep/d0/l2/*

[6] The L1 hardware framework is described at *www.pa.msu.edu/hep/d0/l1/*

[7] *www.pa.msu.edu/hep/d0/l2/cdf.htm#MAGICBUS*

[8] Cypress HotLinks products are described on the Cypress web page. *www.cypress.com/ cypress/prodgate/prod_top.htm*

[9] Some information about the VBD can be gleaned from *d0server1.fnal.gov/www/ upgrade/d0notes/vbdvrb.htm*

[10] Global_data_input_links.ps document found on *www.pa.msu.edu/hep/ d0/ftp/l2/data_transfer/.*

[11] Global_input_format.txt on *www.pa.msu.edu/hep/ d0/ftp/l2/data_transfer/*. This format should be enhanced to specify the byte order in terms of 128-bit blocks, and be clearer that the responsibility of the sender is to land them in the Alpha in the right order.

[12] The Magic Bus Transceiver specification can be found at *www.pa.msu.edu/hep/ d0/l2/data_transfer/*

[13] The Serial Command Link and its data are described by *www.pa.msu.edu/hep/d0/ftp/scl/*

[14] *www.pa.msu.edu/hep/d0/l2/cdf.htm*

[15] The Alpha Processor and boards are described on the Digital web page. *www.digital.com/ semiconductor/alpha/alpha.html*

[16] The Universe chip is described at *www.tundra.com*

[17] R. Moore, J. Linnemann, DØ Note 3401, "How Many Events Should You Have in an Unbiased Sample to Monitor Your Trigger?"

[18] J. Wightman, M. Johnson, J. Linnemann, "Further Simulation Studies of the Level 2 Trigger for the DØ Upgrade", DØ Note 3221, Feb. 1998. See also the DØ WWW page *d0sgi0.fnal.gov/* under "Technical— DØ Upgrade —Trigger Systems". For early studies of a farm architecture, see J. Wightman, M. Johnson, J. Linnemann, "Results from a Deadtime Analysis of the Current Proposal for a Level 2 Trigger for the DØ Upgrade", DØ Note 2783, 1995.

[19] C. Sauer *et. al*, "The Research Queueing Package, Version 2," RA138, RA139 IBM Research Division, San Jose CA.

[20] A. O. Allen, "Probability, Statistics and Queueing Theory," Academic Press, 1978, and H. G. Perros, "Queueing Networks with Blocking," Oxford Press, 1994. See also J. Linnemann, "Simple Queueing Theory for Level 2 and DAQ", DØ Note 1767, 6/22/93, and J. Linnemann, "More on Queueing Theory and Upgrade L2", DØ Note 2708, 8/29/95.

Contents

List of Figures

List of Tables