Implementing the DØ Level 2 Trigger for Run II

R. Moore, Michigan State University
for the DØ Collaboration

The DØ Trigger Overview

The DØ trigger has three tiers of decision making[i]. The level 1 trigger is implemented in hardware predominantly using FPGAs and produces a 10kHz trigger rate for the level 2 trigger. The level 2 system uses embedded processors with a 100μs time budget per event to reduce the trigger rate to 1kHz. This is then passed to the level 3 system which is a software trigger running on a farm of Windows NT workstations[ii].

L2 Trigger Design

The level 2 trigger system is divided into two components: pre-processors and global. The pre-processors are crates which take the level 1 data and reformat and sort it before writing it to the global crate. The global crate the uses the pre-processors' data to make a decision for the event. Processing is done in a high performance single CPU environment, rather than a parallel farm.[iii]

L2 Crate

The pre-processor and global crates both have the same basic design shown in Figure 1. The crates are standard VME64-for-Physics with a custom 128-bit "Magic Bus" (MBus) added. The data processing is done in two types of card. The Second Level Input Computer (SLIC) cards contain 5 DSPs and are used for preliminary formatting and cluster association of the data in several of the pre-processors.

The Worker and Administrator cards are 500MHz 21164 Alpha CPU boards based on the Digital PC164 motherboard. The administrator is responsible for managing all the workers in a crate. All communications to and from external sources are handled by this processor, leaving the workers free to concentrate on processing a given event.

The Magic Bus Transceiver (MBT) is responsible for data I/O. It has eight serial input channels and two output channels to transport event data, as well as a 128 bit digital output port used by the global to send the trigger decision to the framework. There is also a Serial Command Link (SCL) interface which receives and decodes control messages from the hardware framework. These messages are used to notify pre-processors of trigger decisions as well as error conditions and processing qualifiers.

The VME Buffer Driver (VBD) is the mechanism used to write data to the level 3 system. When an event is excepted the Administrator will activate the VBD which then reads out the data in the Workers over the VME bus.

The Bit3 Dual Port Memory card provides an interface for the Trigger Control Computer. This is a machine running Windows NT which will be responsible for controlling the entire trigger system. The dual port memory is primarily used by the administrator to write monitor data to and this is then read by the TCC without the need for it to perform VME bus cycles.

Figure 1: *Picture showing the composition of a level 2 crate. The processors are contained in the SLIC, Administrator and Worker cards.*

Dataflow

The data path through the crate is shown in figure 2. The data first enters the SLIC cards, if present, and is then passed to the Magic Bus Transceiver (MBT) card. This assembles data from all its inputs and once each input has sent something for a given event all the data is broadcast onto the MBus backplane. The MBus broadcast is received by the Alpha processor cards where a DMA engine places it into the main memory. From here it is processed by the CPU. In the case of the pre-processors the output is then written back to the output ports of the MBT card via MBus where it is sent to the global crate. If an event is accepted by the level 2 global crate then the VME Buffer Driver (VBD) card is activated and reads the data from the Alpha's memory over the VME bus.

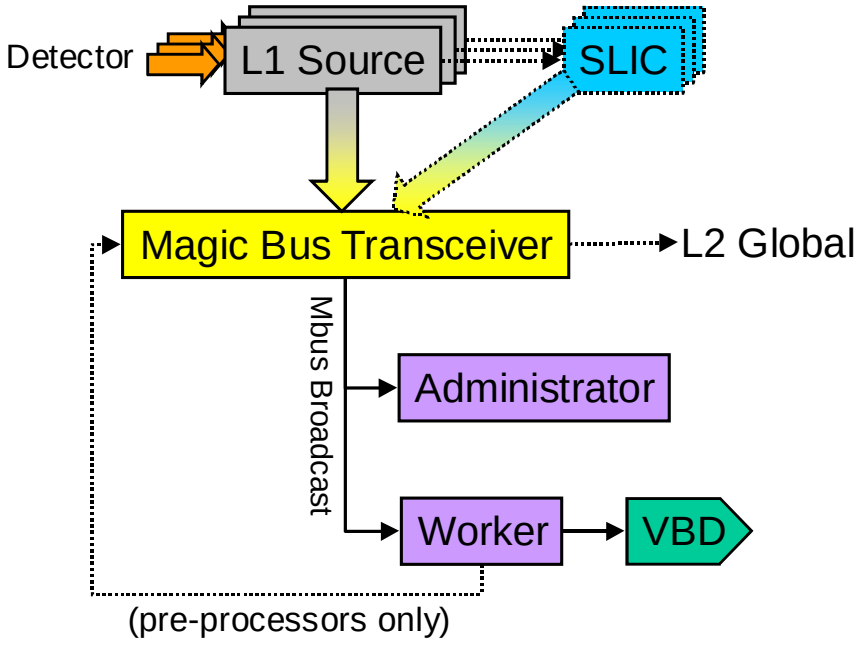

Figure 2: *Data flow path through the level 2 trigger system. The VME Buffer Driver (VBD) card is readout by the level 3 trigger.*

Alpha Processor Card

The Alpha processor cards are the mainstay of the Level 2 trigger system. The design is based on the Digital PC164 motherboard and contains of a 500MHz 21164 Alpha CPU with 128Mb of main memory and a 4Mb L3 Cache. The PCI bus contains a VME interface using the Tundra Universe II chip as well as a programmed IO and DMA engine to interface to the MBus. For monitoring purposes there is a 32 channel ECL output port as well as a PCI slot for an Ethernet card to allow network connectivity.

Software Design

The basic data movement code consists of buffer management. The event data is stored in an internal memory buffer that changes state as it moves through the processing stages. These buffers are synchronized between all the alpha cards in a crate because all the cards see every event broadcast on the MBus. The state diagram for the buffers is shown below in Figure 3. Only the administrator knows all of the buffer states: the workers only need to know a subset of the total number of states because the administrator manages all event readout.

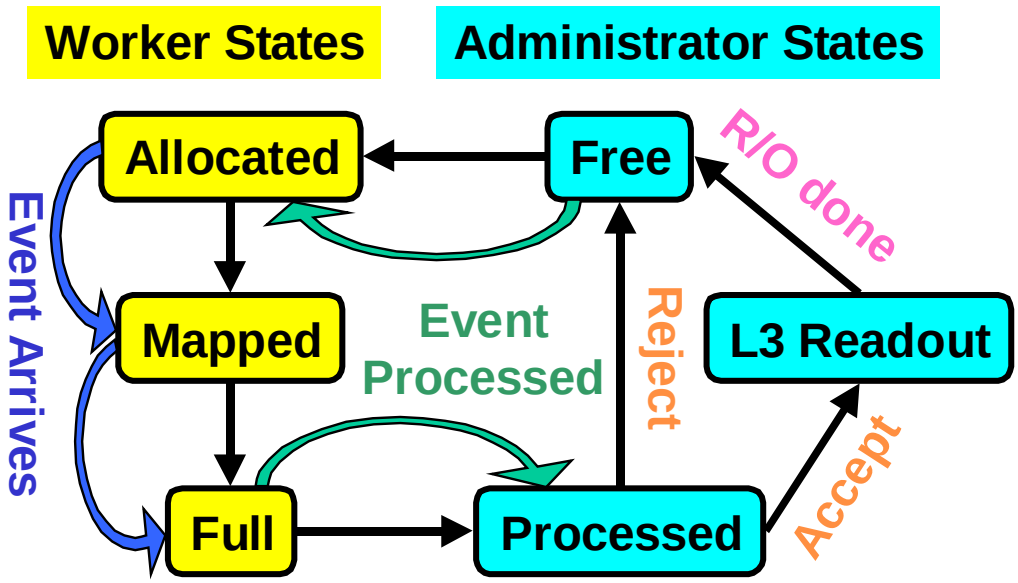

Figure 3: *State diagram for event buffers. The administrator knows about all the states the workers only know the three states on the left. The state changes are explained in the text.*

All buffers start in the free state. When the crate is initialized the administrator tells the workers to allocate 16 buffers and map another 16 onto the MBus broadcast addresses so they are ready to receive data. When an event arrives the corresponding buffer is filled in all the Alphas and an interrupt is generated. The interrupt routine changes the filled buffer's state from mapped to full and then maps an allocated buffer onto the MBus to replace the buffer removed. After a worker has finished processing an event the processed buffer is removed from the full state. The worker then sends a reply to the administrator to notify it and the message the administrator sends back tells the worker which buffer to allocate. In this way the total number of buffers in the allocated, mapped and full states should always remain constant.

Once the administrator is notified that a buffer has been processed it moves it into the processed state and awaits for a level 2 event decision. When a decision arrives the buffer is either freed if

the event was rejected or added to the level 3 readout state. Once these buffers are readout they are moved to the free state.

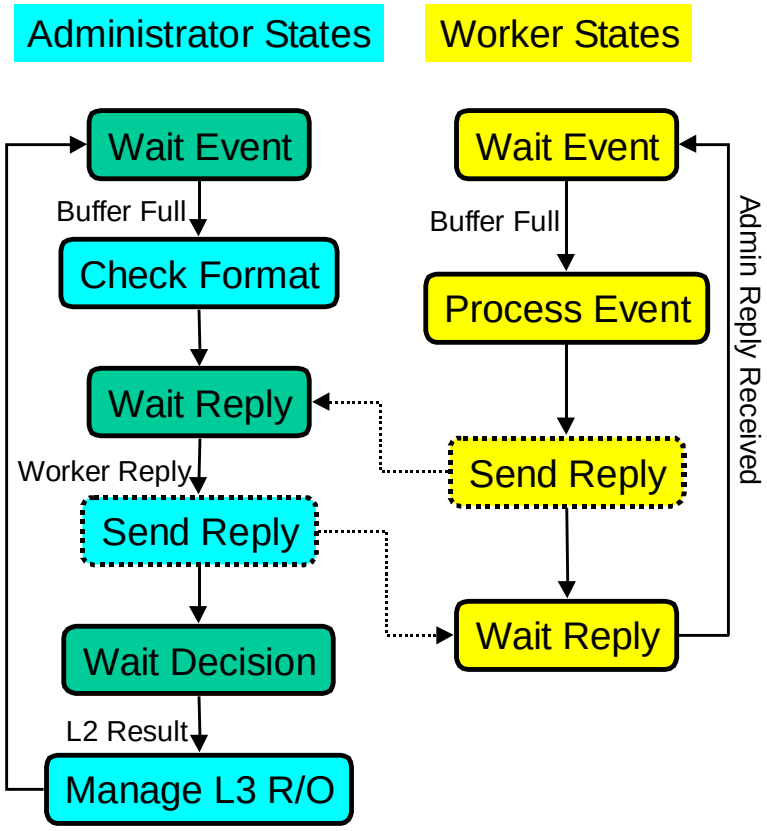

Figure 4: *State diagram of the various processing states of the administrator and worker code, showing how they are linked together.*

Figure 4 shows the processing states of the administrator and worker nodes. The Worker has a simplest event loop. When initialized it waits for an event to arrive, when it discovers and event it then processes the event and tells the administrator when it is done. It then waits until the administrator sends back a message telling it which buffers to allocate to replace the one uses and which is the next event it should process.

The administrator event loop is more complex because it must also handle read-out of the processed events. When an event arrives the administrator checks the format to ensure the data is not corrupted and then waits for the worker to respond. When a worker responds the administrator immediately sends a reply so that the worker can resume processing as quickly as possible and then checks for a decision on that (or a previous) event. In the global crate this state will actually wait for the decision for the current event to arrive from the framework. This is because the global sends the decision and so knows it will arrive back within a few hundred nanoseconds. Finally the administrator will manage the readout to level 3. This involves telling the VBD to read the next event, if the previous event has finished, and then updating the relevant buffers' states.

Software Environment

There are two software environments which will be used on the Alpha processor cards. The first is Alpha/Linux which will be used for debugging software as well as running test and diagnostic programs. The other environment is a bare, operating system free setup which may be used at

run-time due to performance issues. In the OS-free environment programs will be compiled on a DEC Unix workstation and then downloaded to the board via Ethernet for execution. The Linux environment will use the physical memory device (/dev/mem) to access the hardware interfaces. In this way the source code will be compatible between the two environments.

Programming Languages

Both C and C++ languages were evaluated for writing software for the Alphas. The compilers used were the DEC C V5.2 and DEC C++ V6.0. Using a simple FIFO routine, written in ANSI C so that it would compile under both compilers, the execution time of the code from each compiler was compared. The results showed that the C compiler produced significantly slower code without optimization but that the difference between the compilers was not noticeable with optimization enabled.

Having now decided to use the C++ compiler only, irrespective of whether C or C++ code was being compiled, further tests were performed on the programming languages themselves. Two simple prototypes of the global worker program were written: one in C++ and the other in standard C. The two executables produced by the C++ compiler with various optimization settings were then downloaded and timed on the board. Writing the output data was only implemented in the C++ prototype and so for a fair comparison the times with and without output data are given. The results are shown in Figure 5 and indicated that the C++ code is on average approximately 10% faster than the equivalent C code.

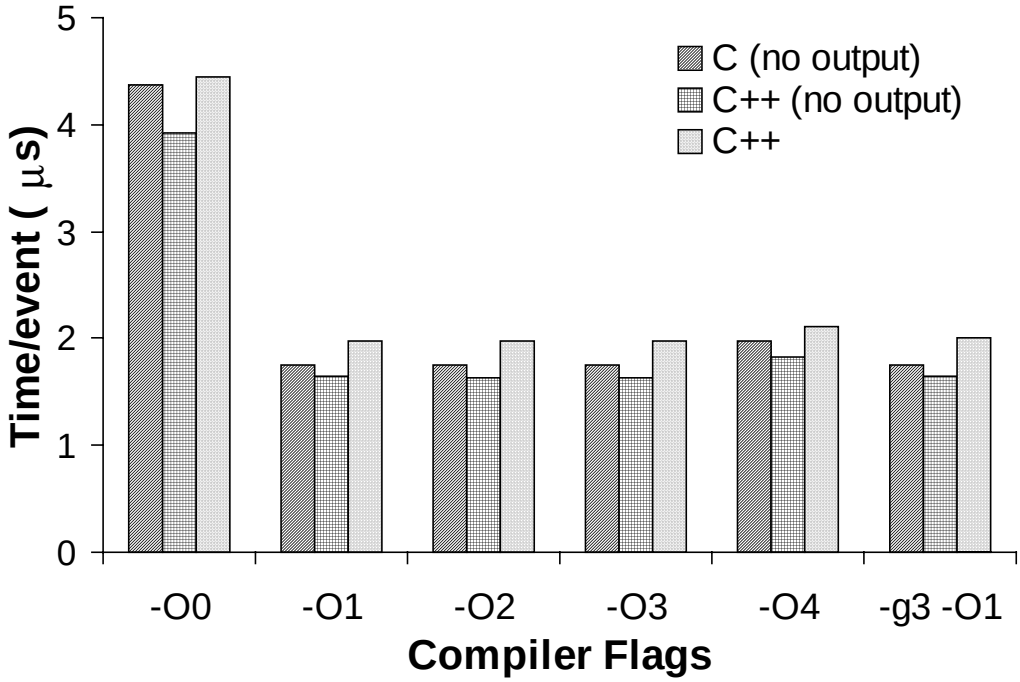

Figure 5: *Plot showing the event times for the various compiler optimizations. The lowest optimization setting is "-O0" and the highest is "-O4". The final column shows the data for optimization with debugging enabled.*

The reasons for this surprising result are several. Firstly only a subset of full C++ was allowed. Dynamic memory allocation, run-time type identification (RTTI), exceptions and the standard template library were all excluded and use of virtual functions was strictly controlled. While these restriction place C and C++ on a more even footing, use of inline functions in C++ give it a slight advantage over C.

Assembly Language

In order to achieve further performance increases assembly language was compared to C++ to see if certain critical areas of code could be optimized further. To do this the bit search algorithm, used to determine which trigger bits are set, was translated into assembly and tweaked to give optimal performance. The results of the speed trials are shown in Figure 6. These results indicate that while assembly does give a speed increase over C++ the difference is a few percent at best. The reason for this minimal increase is because the compiler is good at giving execution hints to the CPU to optimize use of the Alpha's multiple pipeline architecture.

Project Status

The first Alpha boards should arrive by mid November. The administrator software design has been completed and a prototype tested. A prototype of the global worker software already exists and several pre-processor algorithms have been tested. The full level 2 system will be tested in 1999 and will be ready to run at the start of run II in April 2000.

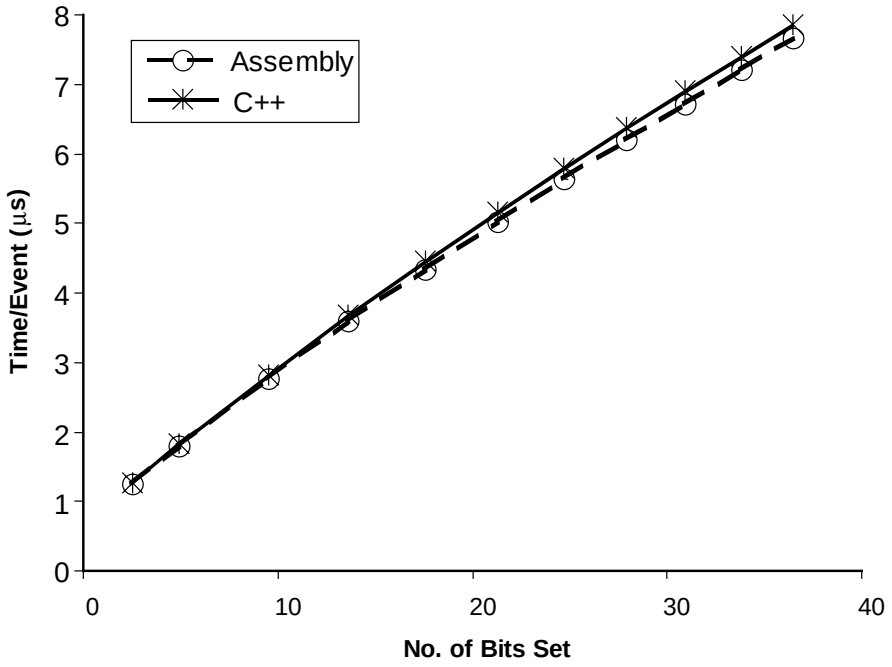

Figure 6: *Plot of the time taken to decode a 128 bit word using C++ and assembly routines as a function of the number of bits set.*

---

[i] "The D0 Trigger System in Run II", M. Fortner, CHEP '98

[ii] "The D0 Level 3 Software Trigger", G. Watts, CHEP '98

[iii] " Design of the DØ Run II Level2 Trigger", R. Martin, CHEP '98