A Local-Global Implementation of a Vertical Slice of the ATLAS Second Level Trigger

P. Maley*‡ J. Schlereth† J. Dawson† E. Denes‡
R. Dobinson‡ D. Francis‡ G. J. Crone§
R. Cranfield§ G. Boorman¶ B. Green¶ J. Strong¶
M. Dam‖ H. Bertelsen‖ J. R. Hansen‖ B. Rensch‖

January 27, 1998

1 Introduction and Project Aims

In this paper we report upon a project which forms a part of the ATLAS level-2 trigger Demonstrator-B programme [1]. The aim of this project is to implement a vertical section of the level-2 trigger [2] using technology available at the present day.

We have set ourselves the following principal aims:

- To implement a system conforming to "architecture-B" (defined in section 2).
- To demonstrate that the proposed protocol is functional and stable.
- To measure parameters of the system's performance which can be used as input into modelling studies of much larger systems.

*Liverpool University
†Argonne National Laboratory
‡CERN
§University College London
¶Royal Holloway, University of London
‖Neils Bohr Institute

The structure of the paper is as follows: section 2 describes the architecture-B protocol and defines terms which are referred to in the rest of the paper. Section 3 describes the hardware and software of the actual implementation we have constructed. We present the results of the measurements performed on the system in section 4 and compare them with the predictions of a simple model. Finally in section 5 we draw some conclusions from the project and comment on possible future investigations.

2 Trigger Architecture

The ATLAS trigger system is based upon the notion that within the chaos that is an LHC event there maybe one or more Regions-of-Interest (RoI) [3]. The level-1 trigger identifies RoIs which warrant further investigation and passes pointers to these regions to the level-2 trigger system. The function of the level-2 trigger is to analyse a sub-set of the data from the detector in those regions indicated to it by the level-1 and decide whether an event merits further consideration or should be rejected.

Figure 1 shows the functional model of the architecture-B level-2 trigger. The components shown in the figure and their respective functions are outlined below and a possible implementation of this architecture is shown in figure 2.

Supervisor (Sup)

The Supervisor, as its name suggests, is the overall coordinator for the level-2 system. It receives the RoI list from level-1 and passes on the information to the RoI-Distributers (RoIR message). Incoming level-2 decisions (GOutR messages) are passed onto the Level-3 trigger and also to the RoI-Distributers (T2DR messages).

RoI-Distributer (Dist)

The main function of the RoI-Distributer is to identify the Readout-buffers containing the front-end data for a given RoI and notify them (RoIRC message). It is also responsible for passing on level-2 decisions to the readout-buffers.

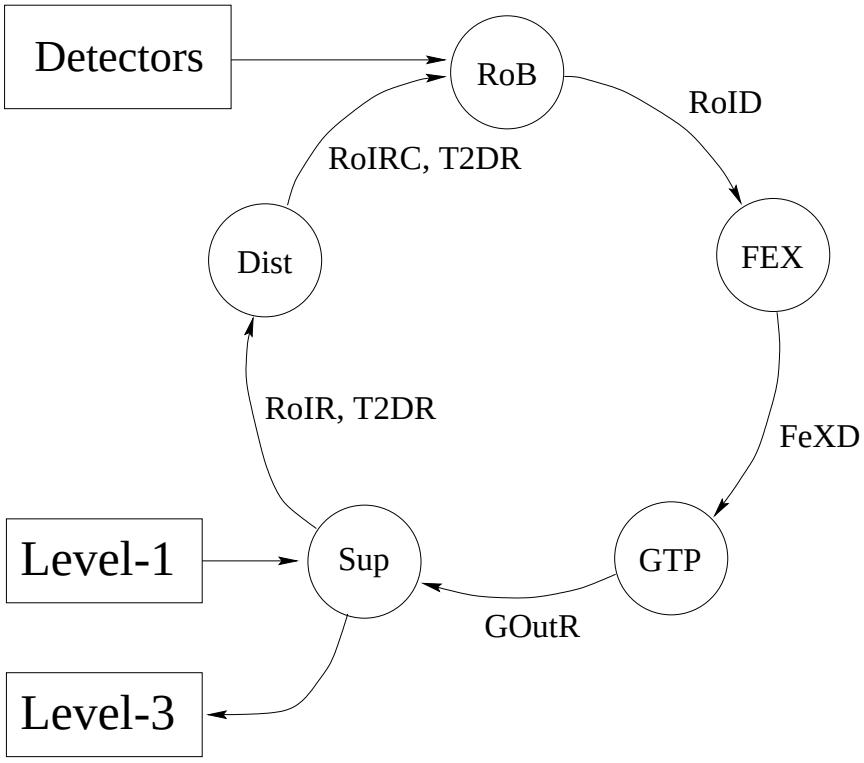

Figure 1: The architecture-B functional model.

Readout-Buffers (RoBs)

The Readout-buffers contain the buffered front-end data. The data is held there until the RoB is instructed to either send it to a level-2 processor (RoID messages) or discard it. architecture-B is a requested-push architecture [1], in the sense that data is sent from the RoB to a level-2 processor on receipt of an RoIRC message from the RoI-Distributer.

Local Processors (FeXs)

architecture-B requires that RoIs are first analysed individually in dedicated processors called Local processors or Feature extractors (hence "FeX"). All of the RoBs pertaining to a given RoI are sent to the same FeX where the data is collected together and analysed. Each RoI is analysed on a separate FeX. The results of this analysis are then sent onto a Global processor (FeXD messages).

Global Processors (GTPs)

The Global processor collects together all the analysed RoIs for a given event and makes an overall decision as to whether to reject the event or not. The decision is passed back to the level-2 Supervisor.

3 Implementation

3.1 System Overview

The system which we have built consists of several technologies:

- 5 readout buffers, implemented with C40 embedded processors.
- C40 to DS-link interfaces.
- 4 local processors, implemented with TransAlpha modules.
- 2 global processors, implemented with TransAlpha modules.
- A DS-link switching network.
- The supervisor: implemented with 3 PowerPC processors and some custom hardware.

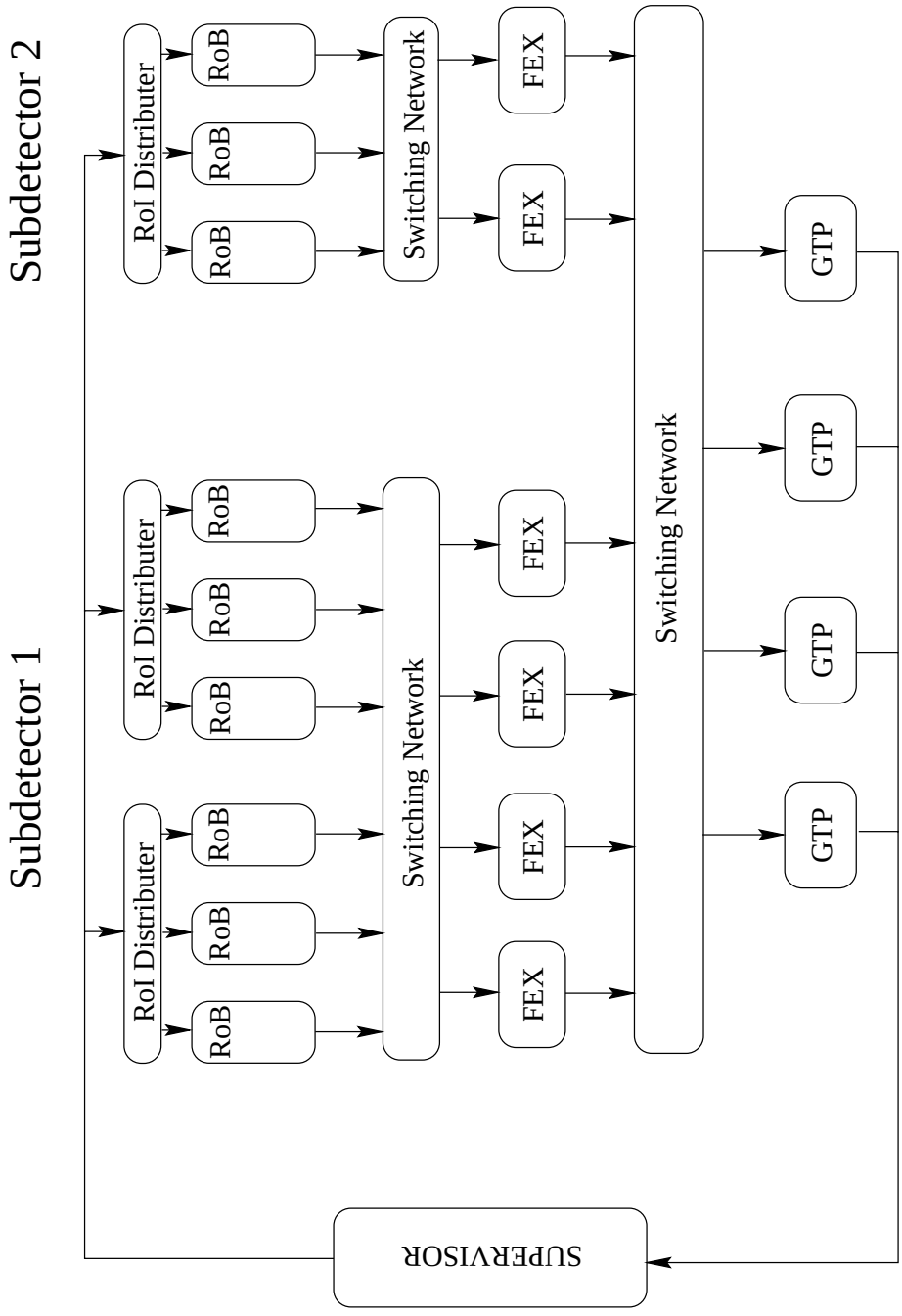

Figure 2: A possible implementation of architecture-B.

Several aspects of the system are non-optimal but are constrained by the hardware available. The hardware implementation is shown in figure 3. and the components are described in the following sections.

Since architecture-B is a requested-push protocol, the RoBs need to be informed of where to send their data, and furthermore the local processors also need to be informed of a destination GTP. There are several possible schemes that may be adopted, but for the tests described in this paper we do as follows:

- Global processor allocation is done by the supervisor on a round-robin basis.
- Allocation of local processors is performed by the RoI-Distributer according to the value of the RoI η-index (see below).

To drive the system we preload a set of events into the supervisor (or the level-1 emulator). Each event contains one or more RoI descriptors and each RoI has η and ϕ-indices. In ATLAS for a given RoI the values of both of these indices would be used by the RoI-Distributer to decide which RoBs have data for that RoI. For the purposes of the demonstrator, we use only the ϕ index for this. The η index is used instead to allocate a local processor to the RoI. Several schemes are possible for local processor allocation and we chose this one simply for convenience.

The ϕ-to-RoB mapping and FeX allocation scheme is shown in table 1, where in the ϕ table "$\bullet$" implies the RoB has data to send.

The information required in each component of the system is carried in the message headers, as defined in [4]. Apart from the data in these headers, which is required to ensure the system runs according to the architecture-B protocol, no actual data (in the sense of simulated detector data) is passed around the system. When data is sent, say, between the RoBs and the FeXs it is simply dummy. Similarly, we emulate the local and global algorithms by wait loops.

3.2 Level-2 Supervisor

3.2.1 Supervisor Hardware

The supervisor consists of commercial PowerPC processor single board computers (CES 8061) and a custom built router bus developed at ANL. Input

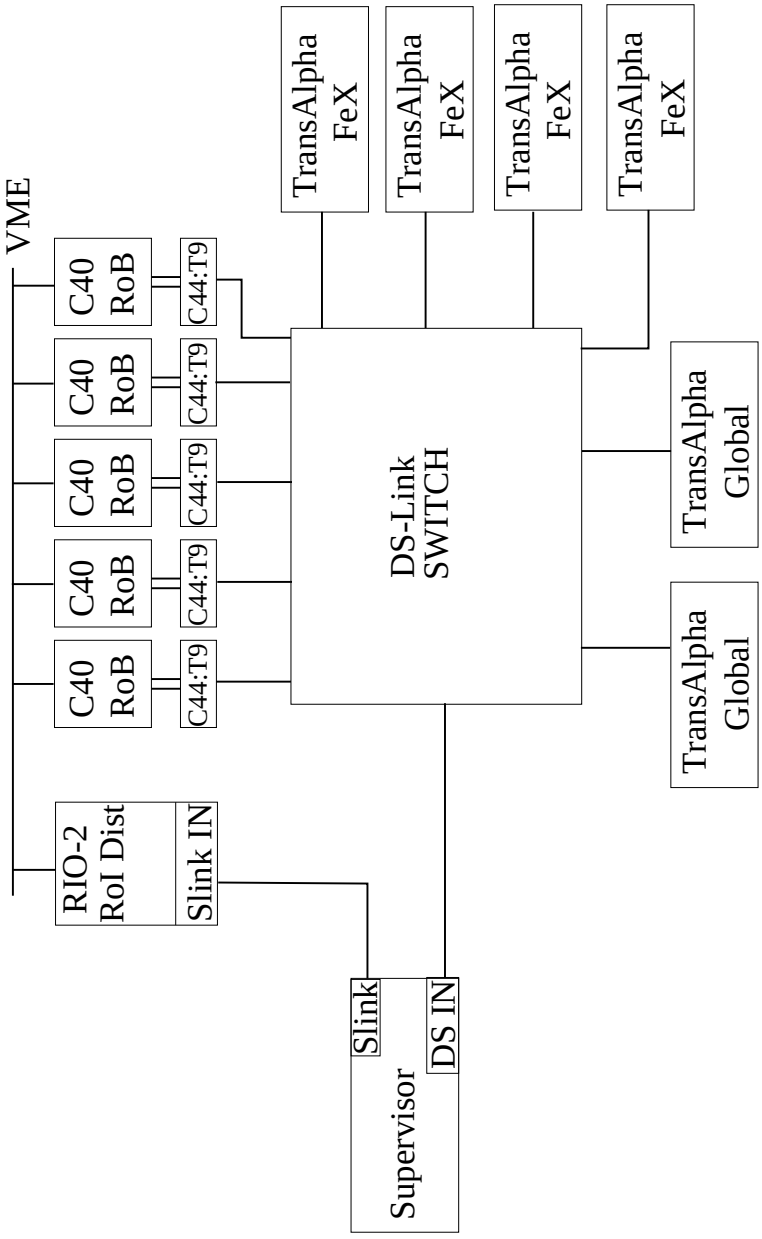

Figure 3: The hardware implementation used in the tests.

η	FeXId
0	1
1	2
2	3
3	4

ϕ	RoB Id				
	1	2	3	4	5
0	•				
1	•	•			
2	•	•	•		
3	•	•	•	•	
4	•	•	•	•	•
5		•			
6		•	•		
7		•	•	•	
8		•	•	•	•
9			•		
10			•	•	
11			•	•	•
12				•	
13				•	•
14					•

Table 1: Specification of RoB hits and FeX allocation.

from the level-1 trigger system and output to the RoI-distributer(s) is via S-link [5]. The components of the router bus are an input router, an output router, a central arbiter, a two channel bus and PCI mezzanine cards to interface the bus to the PowerPC boards. Each of the cards on the bus has a unique destination and source address. The central arbiter grants the bus to source cards requesting the bus, including the input router, on a round-robin basis.

The input router has an S-link protocol connector which can accommodate an S-link physical layer card. Data received are buffered in a 4K word deep FIFO memory. RoI pointers received from a level-1 emulator are routed to a destination address based on the event ID using a lookup table. The destination address is a PMC located on one of the PowerPC boards RoI pointers are stored in a 4K word deep FIFO memory on this card and read onto the RoI processor in fixed length blocks. A more detailed discussion of the software follows in a later section.

In order to send records to the RoI distributer, the output router is used. This card is assigned a unique destination address on the router bus and when one of the PowerPC boards is ready to send data it does so by selecting that card ID as the destination address, and writing a record to the 4K word deep output FIFO memory on the PMC interface. After the record has been transferred to the output FIFO, the card requests the bus. When the card has been granted the bus, it transfers the record to the output router's FIFO memory. As soon as this FIFO is not empty, the output router starts the S-link data transfer, flagging the first word as an S-link control word. The final word is also flagged as an S-link control word, by the change of the FIFO's state to empty.

After the level-2 system has processed the event, a dedicated PowerPC board receives the decision record from the DS-link network through a DS-link PMC interface and forwards the record to the supervisor board over the VMEbus.

For the tests we report here, the level-1 system was emulated by a PowerPC RIO 8061 board with a CERN Simple PCI to S-link (SPS) interface card. RoI pointers were generated and stored to a file. At the beginning of a run, the entire file was dumped into memory and the events were cycled through. Each RoI pointer was sent over the S-link interface with a leading and trailing control word to the input router.

3.2.2 Supervisor Software

The tasks of the supervisor are as follows:

- Read raw data records from level-1 and pack the RoI pointers into a structure.
- Build RoI records to be sent to the ROBs when the complete level-1 information has been received.
- Send the RoIR to the RoI-Distributer (and thus to the RoBs).
- Handle global decisions returned from the level-2 GTP farm.
- Forward the decisions to the RoI-Distributer (and hence to the ROBs).

In order to balance the processing load among the global processors, events are assigned to available global processors on a round-robin basis. A preset

number of events are permitted to be queued to a global processor pending the receipt of a global event decision. A list of free global processor tags is maintaned such that an item is removed form the list as a global tag is assigned when building the RoIR and retruned to the the list when a global decision has been received.

The recipt of the global decision is handled by a dedicated PowerPC with a DS-link network interface card. This implementation, which decouples the supervisor from the network, has several advantages. Since the Demonstrator-B programme is also investigating SCI networks, a common supervisor program supports both projects by changing only the network interface card and drivers. In each case, the global decisions are forwarded to the supervisor over the VMEbus. The design is also extensible to additional supervisors on the router bus which can operate in parallel, all receiving decisions from the same network card forwarded over the VMEbus.

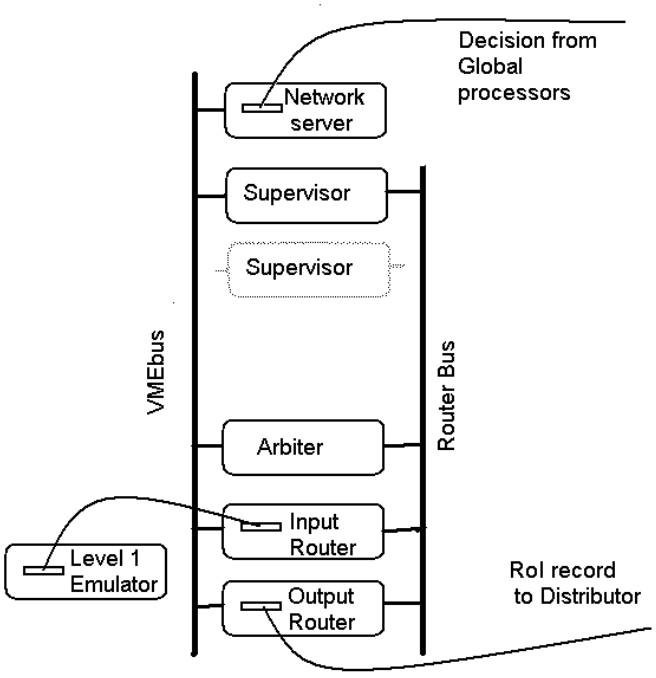

Figure 4: Block diagram of the Supervisor components.

The supervisor provides the measurements of the overall rate and round trip latency presented later. The rate measurement is straightforward but that of the latency is more difficult. Ideally, one would measure the latency as the time from which the event arrives from the level-1 to the time the decision is received by the RoBs. Due to the lack of a common clock accessible to all the processors, the latency is measured only within the supervisor. The latency is defined as the time from when the supervisor assigns a global processor to an event to the time it receives a decision from the global farm. Thus it excludes the time to read an event from the level-1 RoI source and the time to build an event from the RoI pointers.

The supervisor software was developed at Argonne Lab, USA.

3.3 RoI-Distributer

The RoI-Distributer is based on the CES RIO 8061 PowerPC board running software developed at CERN. An S-link PMC attached to the VME board handles the input from the supervisor and it communicates with the Readout buffers via the VME backplane. Software for both the S-link and the VME message passing was taken from the Prototype -1 project [6].

The function of the RoI-Distributer is to interpret the RoIR messages coming from the supervisor and pass on the information to the appropriate RoBs. Which RoBs are part of a given RoI depends upon the ϕ-index of the RoI according to a lookup table (see table 1).

The performance of the program has been measured as follows:

- Time to receive a message on the S-Link: 17 μs.
- Processing time: 5 μs.
- Chained DMA setup time: 17 μs.
- VME transfers: 11 μs per RoB.

In principle, the receipt of incoming messages from the supervisor (on S-link) and the output of messages to the RoBs (across VME) can occur concurrently. However since both must go across the PCI bus, and the S-link data is received by a series of single word reads, these two operations become effectively sequential.

3.4 Readout Buffers

3.4.1 Hardware

Each ROB emulator is implemented by a VME processor card. The cards, designed and built by RHBNC [7], contain a 60MHz TMS320C40 (C40) Digital Signal Processor from Texas Instruments, some local program memory and some memory shared between the C40 and the VMEbus. The C40 processor has 6 communication ports which are all routed to the front panel of the card.

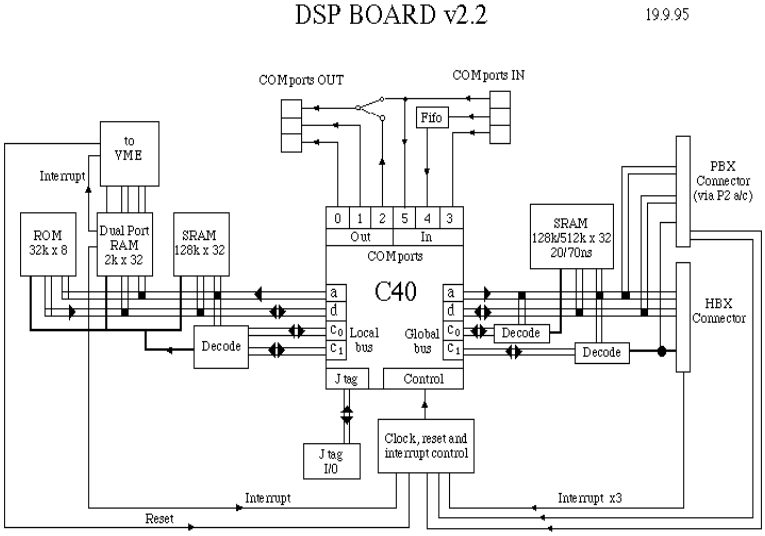

Figure 5: Block diagram of the C40-based RoB emulator board.

3.4.2 Software

The ROB emulator software, developed at UCL[1], is implemented as a simple polling loop. Messages from the RoI Distributor are input through a circular buffer in the VME shared memory. RoI data records are output through one of the communication ports using the C40's built-in DMA controller. Each time round the polling loop the program dealt with a single message if found in the shared memory.

[1]University College, London

If an RoI request message is seen (and there is space in the queue) an RoI data record and an associated DMA descriptor are prepared and added to the queue of records for output. If the queue is full the program waits for the currently active DMA transfer to finish in order to reuse its queue entry. The queue has space for ten RoID records. Decision records are received, read and then ignored.

After checking for incoming messages the software checks the status of the DMA controller and its output queue, starting a new DMA if the last one has completed and the queue is not empty.

3.5 C40 to DS interface

To connect the C40-based ROBs to the FeXs via a DS-link network a C44-T9 interface has been designed and built at NBI. Each interface consists of two processors (a 25 MHz T9000 and a 40 MHz TMS320C44) that exchange data via a dual ported memory (DPM). The C44 is responsible for reading data from the RoB to the DPM and the T9000 takes data from the DPM and outputs it into the DS-link network. For synchronisation purposes each processor can interrupt the other one. The interfaces are housed on 6U VME format boards (for power only).

The software model to drive the interface is shown in figure 6. The DPM is partitioned into four queues of fixed length buffers. Access to these queues (and also to the interrupt and acknowledge locations) is guarded by semaphores to prevent simultaneous access within and across processor boundaries. Messages are read into the DPM from the C40 link by the C40 and then written to the DS-link by the T9000. The interface is thus a store and forward node.

Two DS-links are used to connect each T9000 to the DS-link network to match up with the bandwidth of a C40-Link. In the test system we have a RoB to FeX header-to-trailer latency given by:

$$(45 + N_{\text{bytes}}/3.25)\mu s$$

The RoB to FeX throughput has been measured as a function of message size and is shown in figure 7.

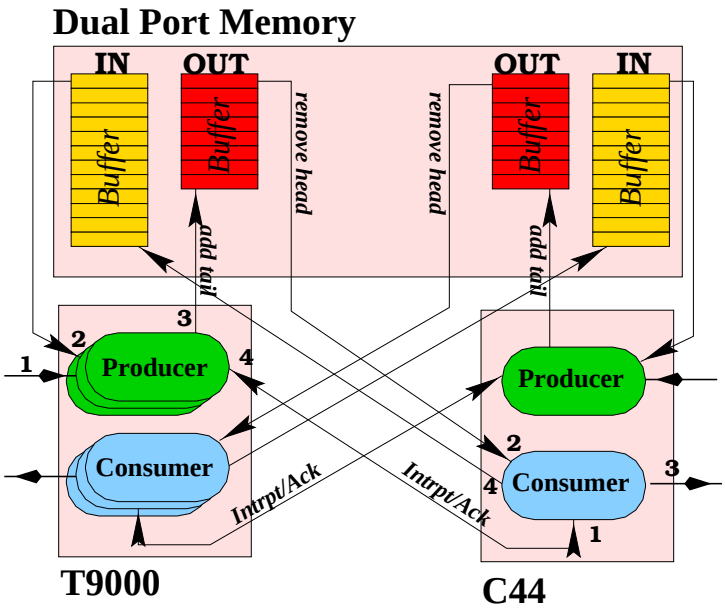

Figure 6: The C44 to DS interface.

3.6 Local-Global Network

The switching-network is the communication medium between the RoBs (after the C40 to DS interfaces); the local processors; the global processors; and from the global processors to the supervisor.

The network is implemented using IEEE 1355 DS-links [8]. These are point to point serial links operating at 100 Mbits/s. In addition to the links there are 32 port switches (C104s). These switches implement worm-hole routing and support the grouping of links.

Figure 8 shows the layout of the network. All links are point to point, though some are shown as multiplexed together for clarity. The layout of this network was determined largely by the hardware available. The T9000s (T9s) at the top of the figure form part of the RoB C40-DS interface. These are housed on VME modules as described above. The central C104 is also mounted on a VME board. The T9000s at the bottom of the figure form part of the TransAlpha modules (see section 3.7 implementing the local and global processors.

These modules are housed on motherboards in boxes with a limited num-

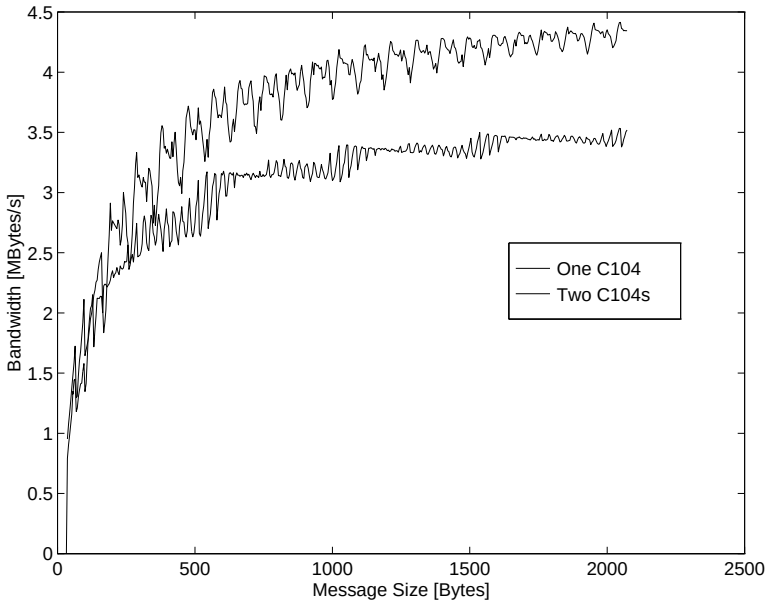

Figure 7: Bandwidth versus message size for C40 to T9000 transfers via one (top curve) and two (bottom curve) C104(s).

ber of external connections. For this reason the RoB to FeX data paths traverse two C104s and the FeX to GTP data paths traverse either one or two C104s. The result of this is that there are unequal bandwidths between different elements, e.g. the bandwidth from FeX1 to GTP1 is not the same as that from FeX1 to GTP2.

The input from the GTPs to the supervisor is also shown on the figure.

3.7 Local and Global Processors

3.7.1 Hardware

The local and global processors are implemented by a hybrid module called the TransAlpha module [9] developed by a collaboration of Liverpool university and Parsys Ltd..

The module was designed to be plugged directly into an existing transputer-based system. It has a T9000 transputer which acts as a communications coprocessor for a 233MHz DEC ALPHA. The two processors can

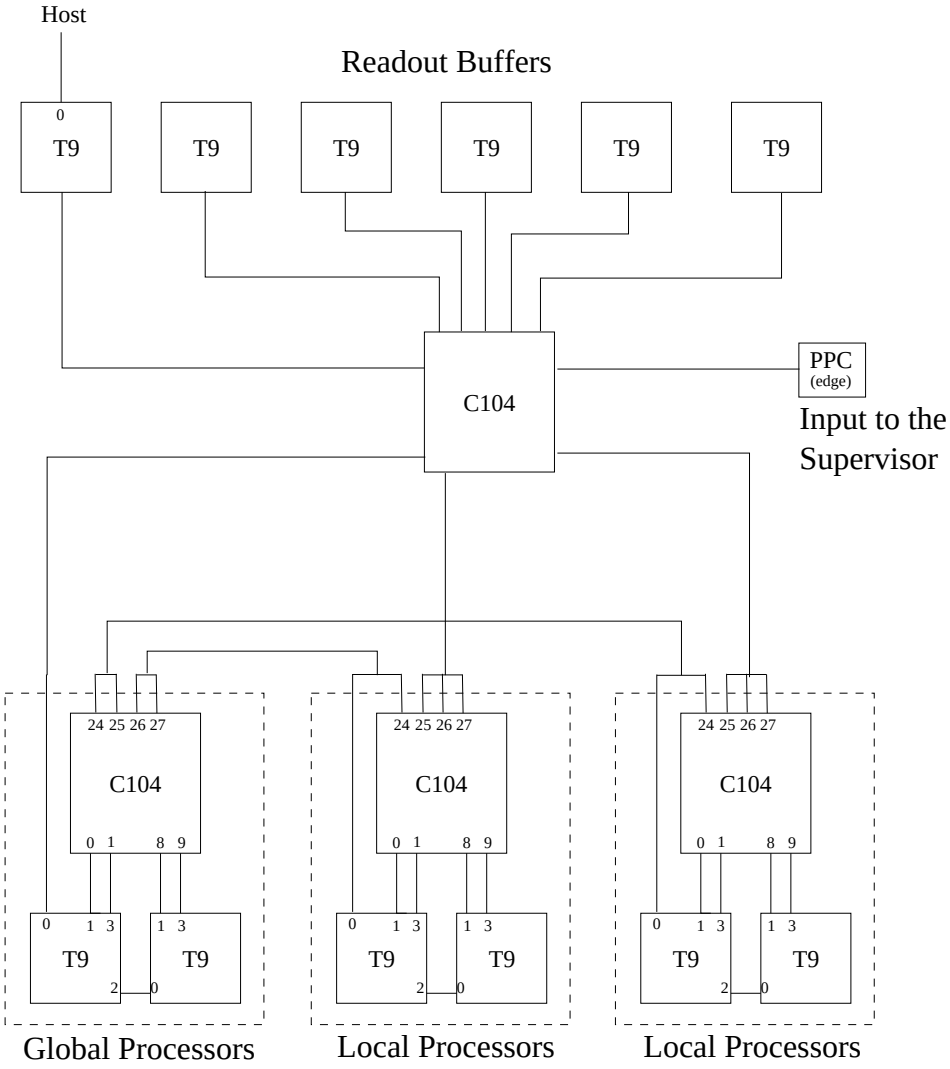

Figure 8: Schematic of the Local/Global DS-link network.

exchange data between their respective memories across a PCI bus. This bus has a PLX PCI960 bridge controller which supplies DMA and interupt services.

3.7.2 Software

The functionality of the local and global processors maps well onto a module such as the TransAlpha. The one micro second context switch time and the integrated hardware for utilising DS-links makes it ideal for collecting together the different fragments of an RoI or the RoIs of an event. The Alpha is then solely responsible for running an algorithm on the data. Its efficiency for computation being maximised, since it does not need to deal with an interrupt every time a message is received.

Figure 9 shows the mapping of the processes in the module. A number

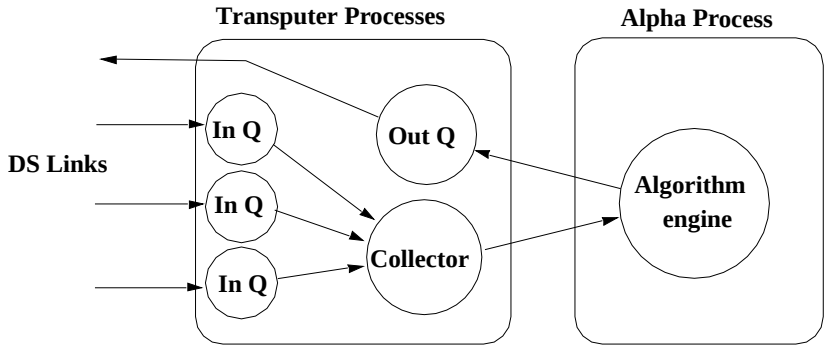

Figure 9: Mapping of processes to the module. For the these tests all processes (including the algorithm engine were run on the T9000).

of Input-Queue tasks receive data from either the buffers or local processors. The information required to collate the different fragments is extracted and passed to a collection manager task while the bulk of the data is stored ready for DMA transfer to the Alpha.

The Collector task is responsible for the book-keeping. It identifies which event fragments belong together and when a complete set of fragments has been collected a descriptor with pointers to all the fragments is queued to be used by the algorithm process on the alpha. For the purposes of the demonstrator we do not use real data and so the "algorithm" is in-fact a simple wait loop, which further-more is executed on the T9000 itself.

Details of the performance of the TransAlpha module can be found in reference [9].

3.7.3 Performance

Both the local and global processor implementations use almost identical code. We parametrise the performance of the processors by two numbers as below.

The time difference between the entry of an RoID message into a FeX and the exit of a FeXD message has been measured to be 80 μs in the case that no fragment collection is done (i.e. an RoI consisting of one single RoB). Additionally, the time required by the fragment collection process has been

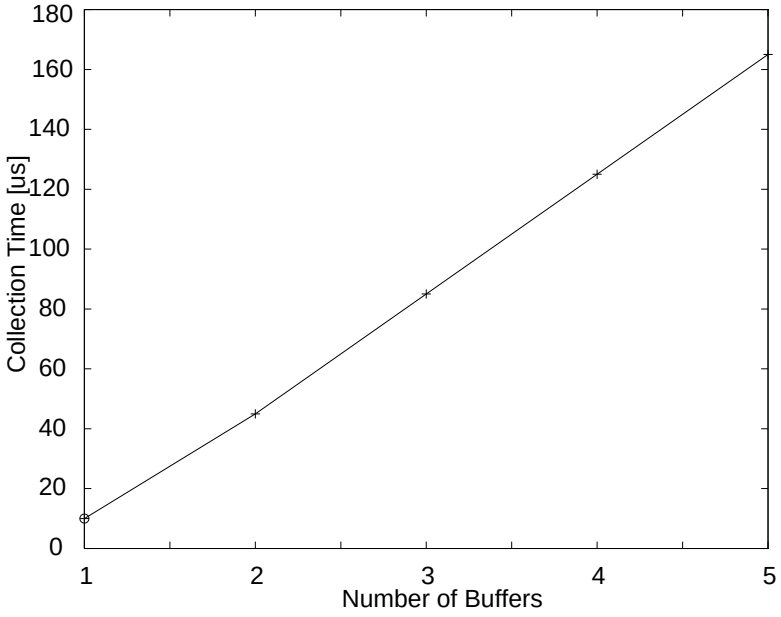

Figure 10: Variation of fragment collection time with the number of sources.

measured as a function of the number of fragments collected (e.g. the number of RoBs in an RoI). The result is shown in figure 10. The curve is essentially linear and shows that approximately 40 μs of processing time are required to unpack the message and do the book-keeping for each source.

We thus parametrise the local and global processor latency as:

$$\text{Latency} = 80 + 40 \cdot (N_{\text{fragments}} - 1) \quad \mu s$$

Where for a local processor $N_{\text{fragments}}$ is equal to the number of RoBs in an RoI and for a global processor it is equal to the number of RoIs.

3.8 Global Processors to Supervisor

In order to pass the level-2 decisions from the global processors back to the supervisor we use a DS-link PCI Mezzanine Card (PMC), designed and built by DESY Zeuthen. This card provides a DS-link interface for embedded processors, in particular for the RIO 806x boards we use here.

The software library developed to allow an application to utilize the link emulates the DS-link hardware of the T9000, implementing the same packet and message protocols. Figure 11 shows the bandwidth obtained between the PowerPC and a T9000 as a function of message size. For these tests the

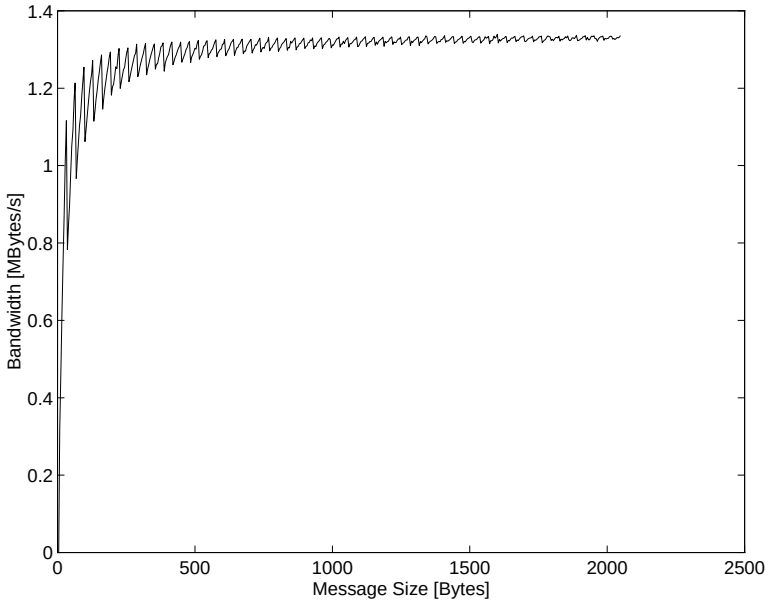

Figure 11: Performance of the DS-link PMC: T9000 to RIO2 via a single C104.

messages between the GTPs and the supervisor are always twenty bytes and the interface can run at upto 20kHz for messages of this size.

Data is passed between the PowerPC and the Rx/Tx FIFOs of the PMC by DMA transfer. In order to avoid context switching overheads the PowerPC

is not interrupted on the arrival of every packet. Instead it polls the registers of the STC101 (the chip which actually drives the DS-link). This has the dis-advantage that a heavy load is placed on the CPU. In order to prevent this interface being the system bottleneck, the supervisor has one CPU whose sole task is handling the DS-link.

The current software implementation does not allow multiple message sources to be handled concurrently and for this reason a small multiplexer task runs on one of the TransAlpha modules. This task collects the decisions from both GTPs and sends them one at a time to the supervisor. It has the effect that the latency from the second global processor to the supervisor is longer (by around 10 μs) than from the first global processor because the messages must traverse an extra link.

4 System Performance

4.1 Overview

We have performed a series of measurements of the event throughput and decision latency of the vertical slice.

The event throughput is simply the number of events per second that the level-2 system can process, while the decision latency (as used in this report) is the time between the arrival of a level-1 "yes", in the level-2 supervisor, and the arrival of a level-2 decision in the supervisor.

We have measured these parameters with a variety of different processor and RoI configurations, and also varying several system parameters such as the local processor processing time and the amount of data passed between the RoBs and the FeXs, etc.

To perform the measurements we run the system in one of two modes:

- Fixed rate.
- Free running.

In fixed rate mode, the supervisor receives level-1 triggers from a level-1 emulator program. The desired rate is set before the run is started. If the level-2 system cannot cope with the rate set then the buffers fill very quickly and the supervisor aborts. Otherwise the system runs stably. This mode is clearly closest to the manner in which the level-2 system would run within ATLAS.

In, so-called, free running mode, no level-1 emulator is used. Instead the supervisor controls itself when an event is sent in to the level-2 system. This control is done via a single setable parameter which is the number of events allowed to be inside the level-2 system concurrently (this parameter is referred to also as the "queue size"). For example, suppose this parameter is set to three, then after the supervisor has sent out three events (usually in very quick succession) then no more events may be sent out until a decision has come back to the supervisor. On the return of a decision the next event is released.

All measurements of latency and throughput are performed by the supervisor and mean values and distributions are available at the end of a run. Runs consist of between several hundred thousand and several million events.

The system was run in three basic configurations, which we refer to as:

- Single Pipeline
- Multi-Pipeline
- "Wide" Systems

An explanation of these configurations and the results obtained are presented in the following sections along with a simple model which attempts to interpret them.

As noted above, fixed-rate running is the most realistic running mode, however free running is useful for determining the key parameters of the system. In order to extract the unloaded latency and the maximum sustainable throughput of the different configurations we run the system in free-running mode. The number of events allowed concurrently in level-2 is set to one in order to measure the latency, and eight (sixteen when there are two GTPs) in order to measure the throughput. We also ran some configurations in fixed-rate mode in order to measure the latency as a function of the throughput. The results are presented in the following sections.

Unless stated otherwise, all measurements are performed with 128 bytes of data (not including the 32 byte message header) passed from each RoB to the FeXs and with only the 32 byte message headers passed between the FeXs and the GTPs.

4.2 Pipeline Results

In the Single pipeline configuration, a single RoI per event is used. This RoI corresponds to a single RoB (always the same RoB), the RoB data is sent always to the same FeX and then the FeX sends its data to a single Global processor - a classic pipeline.

4.2.1 Free Running: The Effect of Queue Size

The variation of the latency and the throughput of the system with queue size is shown in figure 12. The inverse throughput is plotted as "micro seconds per event", i.e. inverse event rate. Remember that the queue size refers to

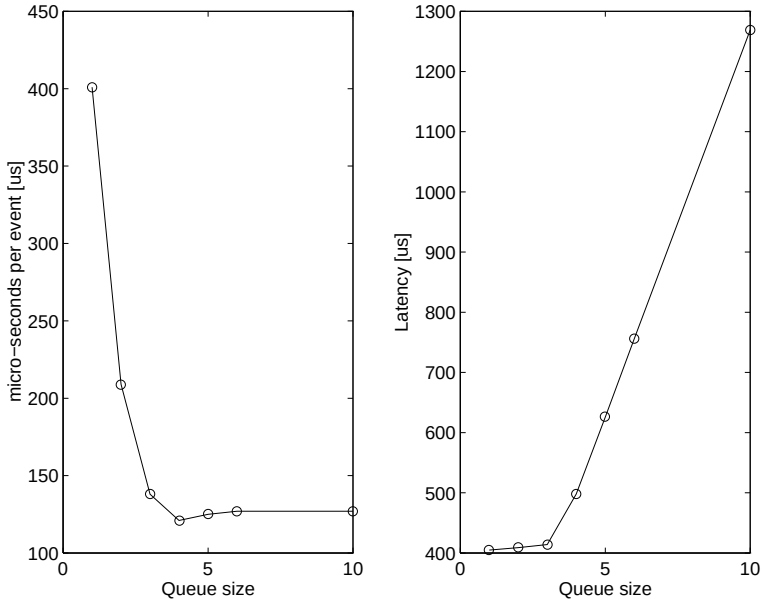

Figure 12: Variation of the number of events in level-2.

the size of the queue of available global processors held in the supervisor (see section 3.2.2), and is just the number of events which the supervisor allows to be within the level-2 system concurrently.

We have a simple model for the latency and the throughput of the pipeline configuration.

The pipeline consists of a number of components, e.g. the RoB, the FeX, a DS-link etc. An event passing around the system takes a certain time to

pass through each component and we refer to these times as the component latencies: T_i.

When only a single event is allowed within the level-2 at any given time the latency, i.e. the time to pass around the system, is just given by the sum of the individual component latencies as there are no other events to impede its progress. We term this the fundamental latency of the system. The inverse throughput (the "number of micro-seconds per event") in this case is just equal to the latency, as the supervisor only allows a new event to enter the system when the previous event has returned.

To understand what happens when more events are allowed to be concurrently within the level-2 system consider figure 13. The figure shows a pipeline system consisting of three elements with component latencies equal to $T_1 = 2$, $T_2 = 4$, and $T_3 = 3$ units of time. The evolution of the system is shown for the cases with queue sizes equal to two and three. An event is allowed into the system only if the number of events already there is less than the queue size. Otherwise an event must wait for a previous event to leave the system before it can enter. These figures show that the transient states at start-up are practically non-existent and steady running is achieved after a very few events.

Looking at the left-hand figure we see that the inverse throughput is 4.5 time units, i.e. $\frac{1}{2}(T_1 + T_2 + T_3)$, while for the right-hand figure the inverse throughput is T_2.

Similarly while the latency for a queue size of two is just equal to the sum of T_1, T_2 and T_3, for a queue size of three, the latency is now 12 units of time, i.e. equal to $3 \times T_2$.

This leads us to the following model for the latency and inverse throughput of a system with a set of component latencies, T_i:

$$\begin{aligned} \text{Latency} &= \text{Max}\left(\sum_i T_i, N_\text{Q} T_\text{Max}\right) \\ \text{Throughput}^{-1} &= \frac{1}{N_\text{Q}}\text{Latency} \end{aligned}$$

where N_Q is the number of events that the supervisor allows to be concurrently in the level-2 system (the queue size), T_Max is the largest of the set of component latencies, and $\text{Max}(a, b)$ is equal to a when $a > b$ and b when $a < b$.

As can be seen from figure 12 allowing two or three events into the system concurrently causes practically no increase in latency (an increase of 8 μs

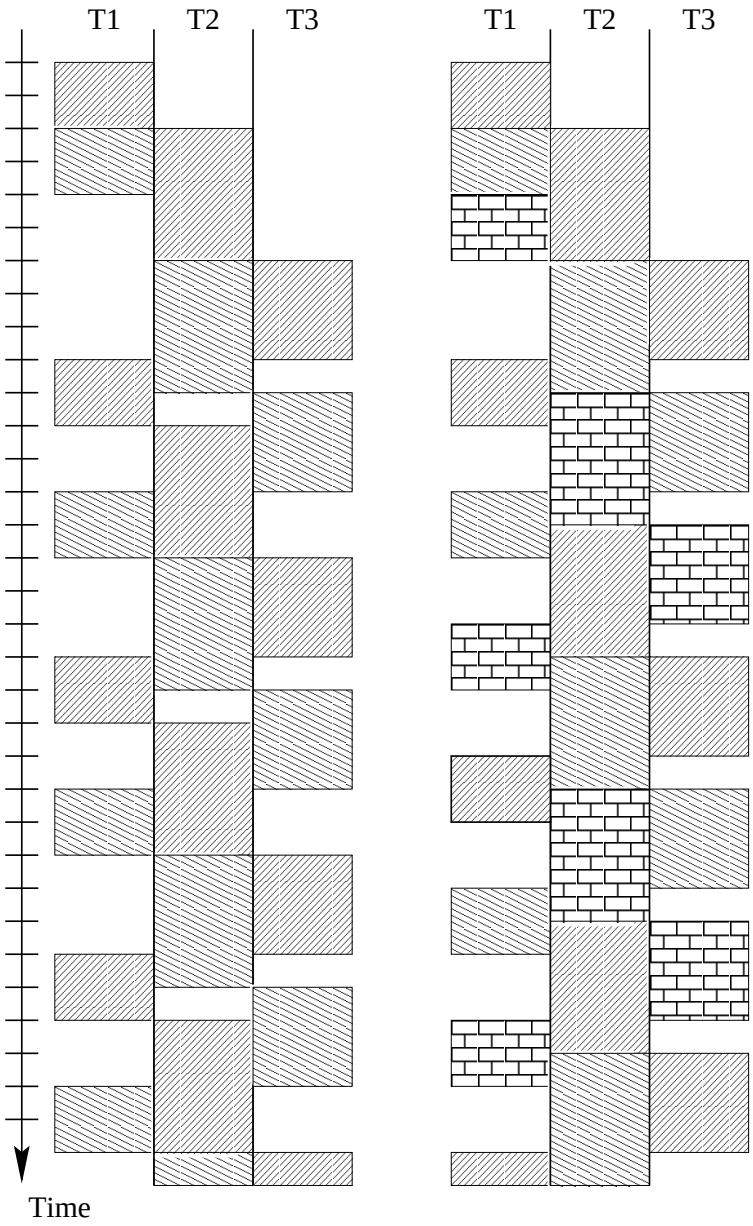

Figure 13: Timing diagrams for queue sizes of two (left figure) and three (right figure). See text for explanation.

between one and three events) whereas the simple model predicts no increase. When four events are allowed into the system, the latency becomes dominated by the events queueing at the bottle-neck component, i.e. the component with the largest component latency. From this point onwards the latency is given by: $N_{\mathrm{Q}} T_{\mathrm{Max}}$.

The throughput similarly has two distinct regimes. When less than four events are in the system it is not saturated and the inverse throughput is simply the fundamental latency divided by the queue size. When four or more events are in the system this time becomes less than the maximum component latency and events are forced to wait before passing this component. Thus the inverse throughput levels off at a value of T_{Max}.

Thus from these measurements we can determine two quantities that characterise the system:

Largest component latency	T_{Max}	127 μs
Fundamental level-2 latency	$\sum_i T_i$	405 μs

The fundamental latency is the lowest latency possible in the system, and the largest component latency determines the maximum rate at which the system may be run in fixed rate mode, i.e. as it would be run in ATLAS.

Table 2 shows the measured component latencies and their total. The total latency is in good agreement with what we measure; the exact agreement is merely fortuitous, since some of the times are estimates good only to a few micro seconds.

There is however a notable discrepancy. No single component has a latency of 127μs as we would expect from the model. We would expect the inverse throughput to be limited by the FeX and GTP components at 80 μs per event. We return to this point below, when looking at the effect of varying the RoID message size.

We have investigated the variation of two other parameters in the simple pipeline configuration: The effect of changing the amount of data passed between the RoBs and the local processors (figures 14 and 15); and the effect of increasing the processing time in the local processor (figures 16 and 17).

There are several things we learn from varying the amount of the data passed in the RoID message (RoB to FeX). The variation of both the latency and the inverse throughput vary linearly with the amount of data passed. This is exactly the behaviour we would expect for the latency, since the time

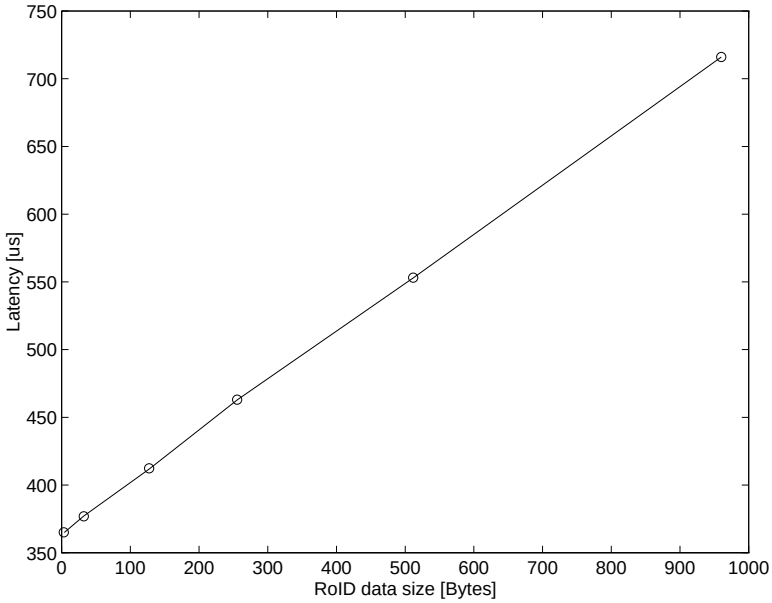

Figure 14: Variation of level-2 latency with RoID data size.

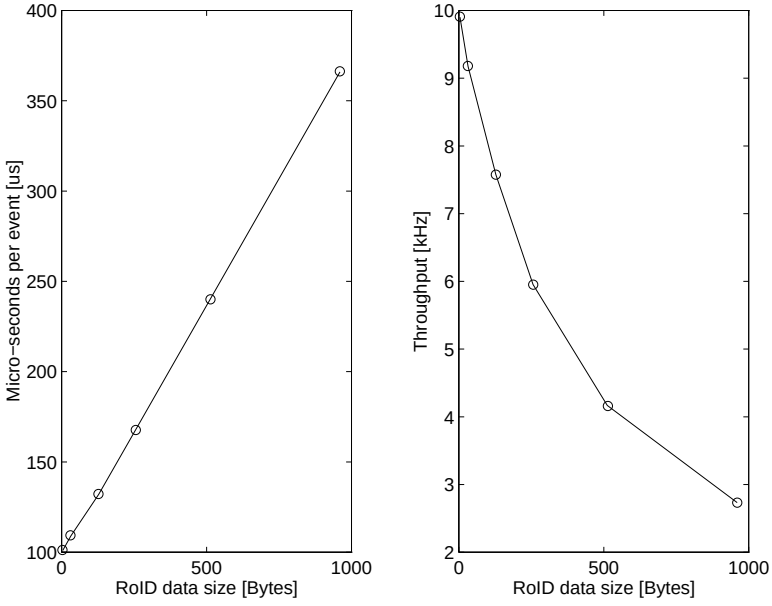

Figure 15: Variation of throughput with RoID data size.

Pipeline component	Latency [μs]
Supervisor to RoI-Distributer	17
RoI-Distributer processing	5
RoI-Distributer to RoB	28
RoB processing	50
RoB to interface	50[a]
interface to FeX	45[b]
FeX processing	80
FeX to GTP	7
GTP processing	80
GTP to Supervisor	7
Supervisor processing	36
Total	405

[a]For 128 bytes of data.
[b]For 128 bytes of data.

Table 2: Component latencies for the pipeline configuration.

to send a message across a link is given by:

$$\text{Overhead} + \text{Message size}/\text{Bandwidth}$$

Thus the time to send the message, and hence the total system latency, is linearly dependent upon the message size.

However, the fact that the inverse throughput is also linearly dependent on the message size implies that T_{Max} is linearly dependent on the message size and thus that this message transfer is somehow part of the system bottleneck. This would be expected for very large messages, but not for small ones where the transfer times are less than $50\mu s$.

If we consider the effect of reducing the amount of data transfered from 128 bytes to 4 bytes we see that the latency decreases by 45 μs (from 412 μs at 128 bytes to 365 μs at 4 bytes) and the inverse throughput decreases by 30 μs (from 132 μs at 128 bytes to 101 μs at 4 bytes). The only way that it is possible to reduce the latency by more than the inverse throughput is if *two* component latencies are simultaneously reduced. This is in fact precisely what is happening. The RoB to FeX transfer passes through the C40-DS interface which as described in section 3.5 is a store and forward node. Both

the transfer time from the RoB to the interface and from the interface to the FeX are linearly dependent on the message size and thus both are reduced.

The bandwidth of the C40 link is twice that of the DS-link and so the reduction in the RoB-to-interface latency is half that of the reduction from the interface-to-FeX. Thus of the total reduction in latency of 45 μs: 15 μs is due to to C40 link and 30 μs is due to the DS-link.

Now this 30 μs is just the reduction we see in the inverse throughput. So the implication is that the transfer from the interface to the local processor is contributing to the bottleneck component latency. Furthermore, the latency of such a transfer for RoID messages with 128 bytes of data is 45 μs, which when added to the latency of the FeX, 80 μs, is suggestively close to our bottleneck component latency of 127 μs.

The implication then, is that the reading of the data into the FeX and the internal FeX processing (at least the book-keeping in the fragment collection process) are not concurrent activities but together form a single pipeline component.

Figures 16 and 17 show the effect of increasing the processing time in the local processor. The processing time is emulated by causing the algorithm engine process (see section 3.7) to sleep for some number of microseconds. The latency measurements shown in figure 16 are consistent with (for processing times greater than zero):

$$\text{Latency} = \text{Latency(no processing)} + \text{Processing time} + \delta$$

For zero processing time the actual wait code was not present.

The extra time δ is due to the implementation of the wait. This is done by instructing the algorithm engine to do nothing until a number of clock ticks has passed. For a low-priority process on the T9000 each tick is 64 μs. The time between the issue of the wait instruction and the next tick can vary between zero and 64 μs, thus to ensure that a whole tick has passed the T9000 waits for the following tick before scheduling the process again. Thus a requested wait of n ticks will result in a delay of anywhere between n and $n + 1$ ticks, or $(n + 1/2) \cdot 64 \quad \mu s$ on average, giving a $\delta = 32 \quad \mu s$.

Looking at the inverse throughput (figure 17) we see that it similarly increases linearly with the processing time. This implies that T_{Max} is increasing linearly with the processing time, and identifies the local processor as the bottle-neck component in this configuration.

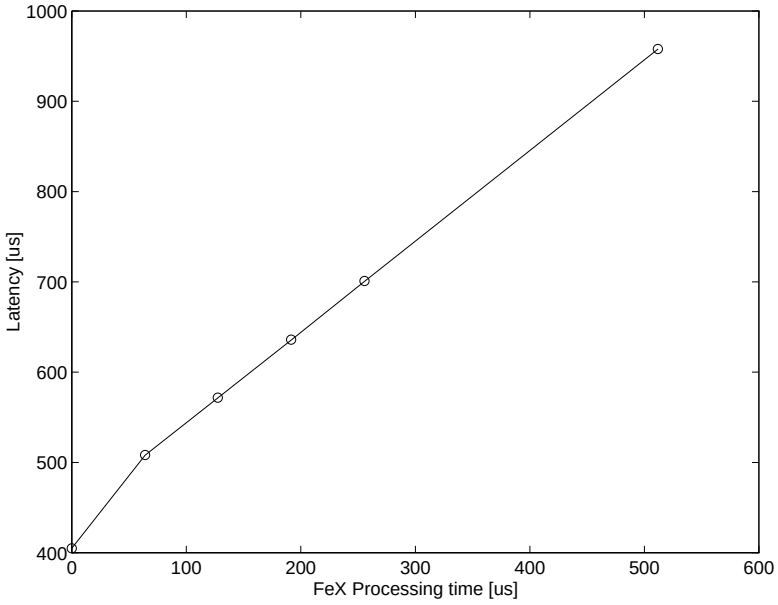

Figure 16: Variation of level-2 latency with the processing time in the FeX.

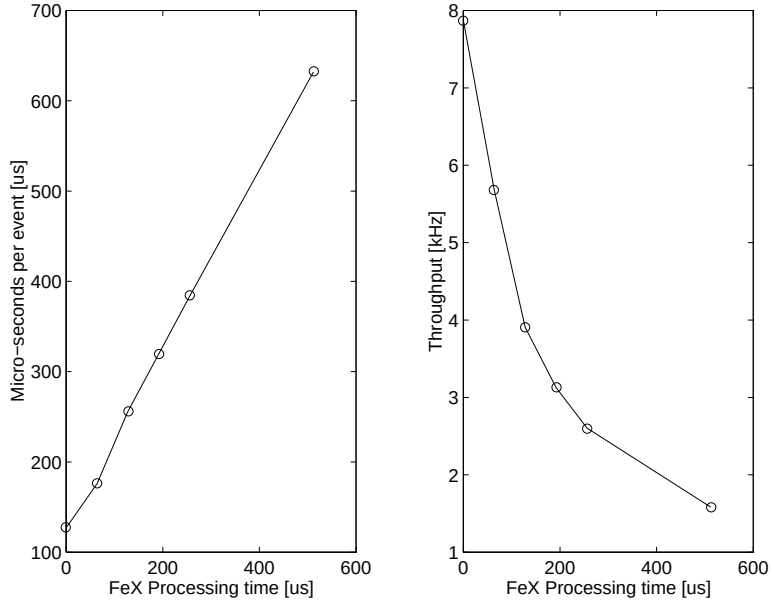

Figure 17: Variation of throughput with FeX processing time.

4.2.2 Fragment Collection in the FeX

Keeping a single local and a single global processor, but now varying the number of RoBs in the RoI, we can investigate the behaviour of the fragment collection in the local processor

The behaviour we expect is that presented in section 3.7.3, whereas that observed is show in figures 18 and 19. If we consider the latency variation, we see that it increases linearly with the number of RoBs as expected. However, instead of an increase of 40 μs per RoB the increase is closer to 45 μs per RoB.

The throughput as we have already seen in section 4.2.1 does not agree with our expected value of 80 μs per event for a single RoB. However if we modify the parametrisation of section 3.7.3 to:

$$\text{Latency} = 80 + 45 \cdot (N_{\text{fragments}} - 1) \quad \mu s$$

then for four and five RoBs we predict around 220 μs and 260 μs per event, in reasonable agreement with those measured.

If we accept the inference (section 4.2.1 that there is no concurrency in the receiving of data into the FeX and the internal processing of the data, then we see that this becomes less true as we increase the number of data sources. In principle all of the incoming RoID messages may be received concurrently, with the data packets interleaved in time on the DS-link and thus even in the event of no concurrency between the receiving and processing, we still benefit from the concurrency in the receiving of data.

Accepting that the exact dynamics of the fragment collection remains unclear, we parametrise the FeX collection time for use in the model of section 4.4.1 as:

$$\text{Latency} = \text{Max}(125, 80 + (N_{\text{RoIs}} - 1) \cdot 45)$$

4.2.3 Running with the level-1 Emulator

The level-2 system proper will of course be running in what we have termed fixed rate mode. Thus we must consider how the latency of the system varies with the throughput. For the simple pipeline configuration we have considered so far, since there are no points of contention we would expect the latency to be independent of the throughput until we reach the maximum rate, at which point the system buffers will fill up and the system will fail.

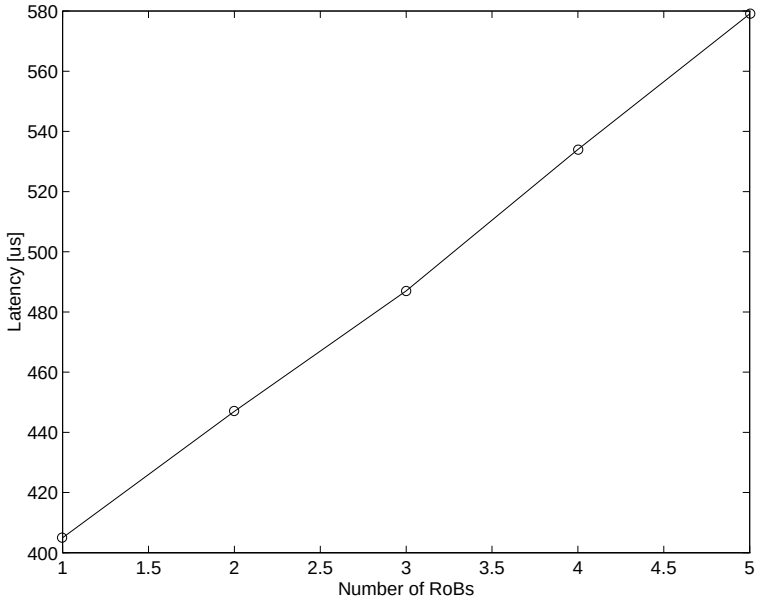

Figure 18: Latency as a function of the number of RoBs.

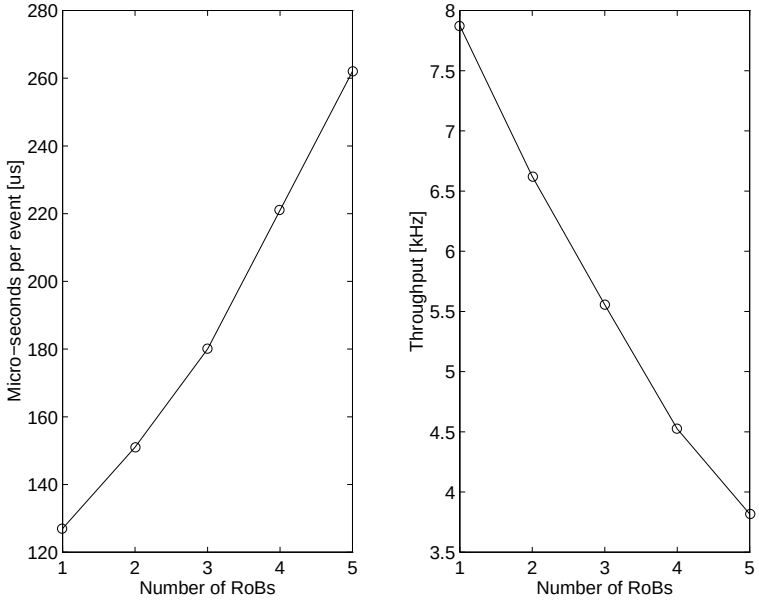

Figure 19: Throughput as a function of the number of RoBs.

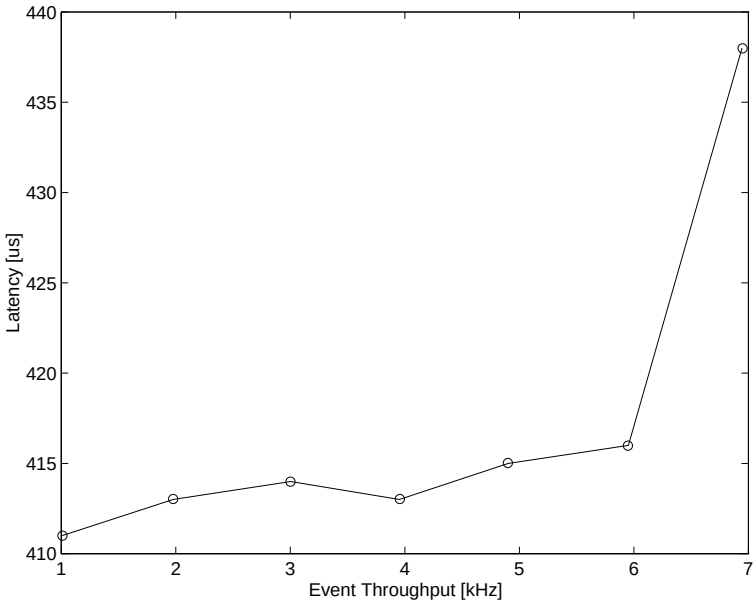

Figure 20: level-2 latency as a function of level-1 input rate.

The measured variation is shown in figure 20. It is fairly flat, increasing by about 1%, from 1kHz to 6KHz but then starts to increase rapidly.

This sharp increase at 7 kHz is unexpected but unfortunately we have not had time to investigate this behaviour in more detail. It would be interesting to investigate whether the system would run faster, since from figure 12 we expect it to be capable of running up to 7.9 kHz.

The distribution of the latency is shown in figure 21. The distributions are practically identical up to 6 kHz, but then the shoulder starts to increase at 7 kHz. The number of events in the shoulder is several orders of magnitude down on the number in the peak and so are not responsible for the latency increase. However the peak for the 7 kHz run is slightly wider than the others.

4.3 Multi-Pipeline Results

The multi-pipeline configuration has more than one RoI per event, but every event in a given run is identical, i.e has the same RoIs. Each RoI corresponds to a single RoB and each RoB sends data to a different FeX (each RoI is

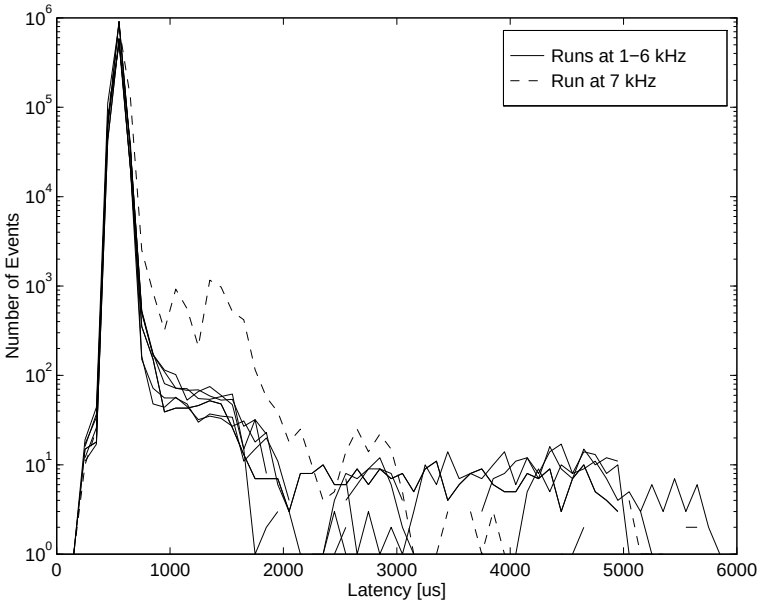

Figure 21: level-2 latency distributions at different rates.

allocated its own FeX in architecture-B). The data from the FeXs is then collected into a single GTP and the decision sent back to the supervisor. Thus we have several RoB to FeX pipelines followed by collection into a single global processor.

Running the system in this configuration with multiple RoIs allows us to investigate the fragment collection process in the global processors. We would again expect the behaviour parametrised in section 3.7 as the coding of the local and global processors is practically identical. The actual behaviour observed is shown in figures 22 and 23. Looking at the throughput variation we see that the measurements for three and four RoIs are consistent (to within 5%) with the parametrisation of section 3.7.3:

$$\begin{aligned} \text{3 RoIs:} \quad & 80 + (3-1) \cdot 40 = 160 \quad \mu s \\ \text{4 RoIs:} \quad & 80 + (4-1) \cdot 40 = 200 \quad \mu s \end{aligned}$$

The lower two points (for one and two RoIs) do not match, but neither would we expect them to because for those configurations the GTP is not the bottleneck for the system. The limiting component for those two points

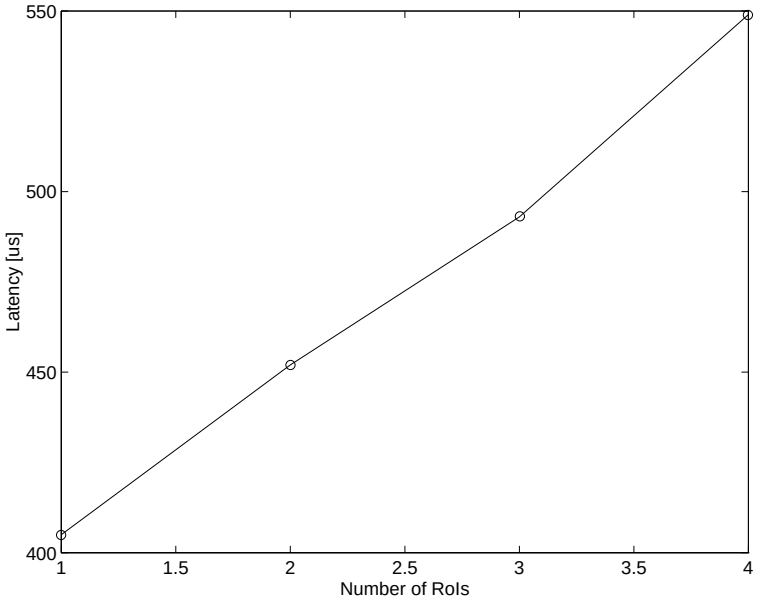

Figure 22: Variation of latency with the number of RoIs.

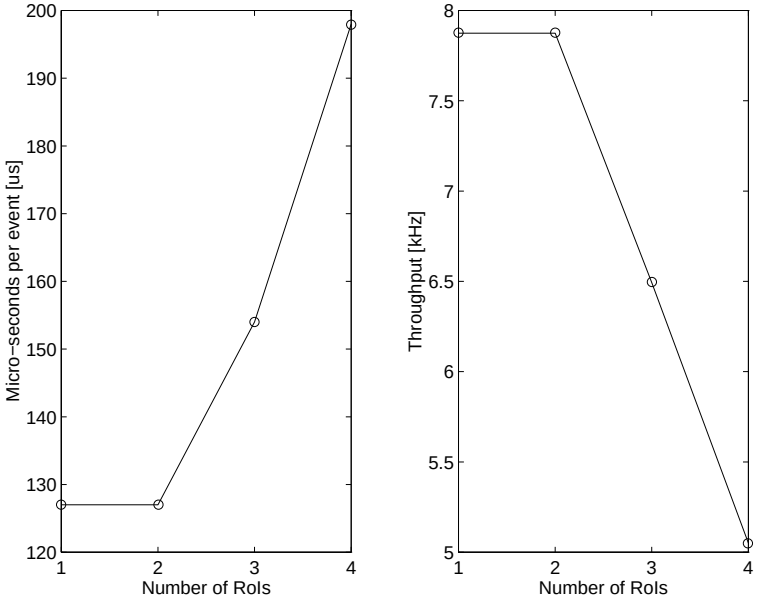

Figure 23: Variation of throughput with the number of RoIs.

is the local processor which as we saw from section 4.2.1 limits the system to 127 μs per event.

From these measurements we cannot determine the magnitude of the GTP latency for one and two RoIs, however, we will show in section 4.4.1 that the measurements point towards a GTP latency of 80 μs for a single RoI.

Since the coding of the GTP and FeX is practically identical we should try to understand why they perform differently. There are two differences: The size of the data passed between the RoBs and the FeXs is 160 bytes, or 5 packets on the DS-link whereas the FeXD messages (FeX to GTP) are always 32 bytes, or a single DS-link packet with only about 7 μs latency. Secondly the global processor used in these measurements shares the T9000 with the multiplexer task as described in section 3.8.

The presence of the multiplexer task may be related to what seems to be an inconsistency. That is, if we look at the variation of the latency with the number of RoIs, from the arguement above we would expect it to increase by 40 μs with each RoI. In fact the latency increases by 50μs with each additional RoI. A difference between the increase in latency and the increase in throughput again implies some not understood concurrency in the processor.

4.4 Wide System Results

4.4.1 Free Running

In the so-called wide system configurations we again use a single RoI per event but now this RoI can correspond to several RoBs. All RoBs are collected together in a single FeX as determined by the architecture-B protocol, but several local and global processors are available. Successive events are routed to different FeXs and GTPs in a round-robin fashion. The full list of configurations measured along with the results is given in table 3.

We perform the measurements as for the pipeline configuration, namely running in the free running mode with the supervisor limiting the number of events concurrently within the level-2 system. To determine the latency we run with a queue size of one, and to determine the throughput we run with a queue size of eight events per GTP.

For the configuration of five RoBs, four FeXs and 3 GTPs, we have performed some additional measurements running in fixed-rate mode.

Also presented in the table are the numbers we expect from our simple

Configuration			Throughput			Latency		Limiting
GTPs	FeXs	RoBs	$[\mu s]$/event		[kHz]	$[\mu s]$		component
1	1	1	127	(125)	7.9	405	(405)	FeX
1	1	2	151	(125)	6.6	447	(450)	FeX
1	1	3	180	(170)	5.6	487	(495)	FeX
1	1	4	221	(215)	4.5	534	(540)	FeX
1	1	5	262	(260)	3.8	579	(585)	FeX
1	2	1	81	(80)	12.4	408	(405)	GTP
1	2	3	95	(85)	10.5	490	(495)	FeX
1	2	5	139	(130)	7.2	587	(585)	FeX
1	3	1	81	(80)	12.4	409	(405)	GTP
1	3	3	82	(80)	12.2	495	(495)	GTP
1	3	5	102	(94)	9.8	594	(585)	Dist
1	4	1	81	(80)	12.4	408	(405)	GTP
1	4	3	82	(80)	12.2	494	(495)	GTP
1	4	5	103	(94)	9.7	588	(585)	Dist
2	1	1	126	(125)	7.9	420	(405)	FeX
2	1	3	195	(170)	5.1	507	(495)	FeX
2	1	5	287	(260)	3.5	622	(585)	FeX
2	2	1	65	(63)	15.4	455	(405)	FeX
2	2	3	94	(85)	10.6	498	(495)	FeX
2	2	5	148	(130)	6.8	610	(585)	FeX
2	3	1	60	(50)	16.7	442	(405)	Dist
2	3	3	73	(72)	13.7	505	(495)	Dist
2	3	5	99	(94)	10.1	605	(585)	Dist
2	4	1	58	(50)	17.2	451	(405)	Dist
2	4	3	69	(72)	14.5	505	(495)	Dist
2	4	5	103	(94)	9.7	601	(585)	Dist

Table 3: Summary of the wide-system measurements. The numbers in parentheses are the predictions of the model.

model extended to these configurations (see below). The final column shows which component, according to the model, is the limiting component of the system.

To extend our model to these configurations we need do two things: Allow for the fact that the latency of a component may depend on the configuration, e.g the fragment collection time in the FeX depends upon the number of RoBs in the RoI (see section 4.2.2); and take into account that we no longer have a pipeline, but that multiple FeXs and GTPs are now available.

As discussed in section 4.2.1, if we have a pipeline of components each of a given latency, then the system latency is given by the sum of the component latencies, $\sum_i T_i$, (for a queue size of one) and the throughput is determined by the largest component latency: T_{Max}.

For the local and global processors which collect event fragments together the latency depends upon the number of RoBs and RoIs respectively. Similarly for the RoI-Distributer, the latency depends upon the number of RoBs to which it must send RoIRC messages. We parameterise the dominant latencies as follows:

$$\begin{aligned} T_{\mathrm{RoIDist}} &= 50 + (N_{\mathrm{RoB}} - 1) \cdot 11 \quad \mu s \\ T_{\mathrm{RoB}} &= 50 \quad \mu s \\ T_{\mathrm{FeX}} &= \mathrm{Max}(125, 80 + (N_{\mathrm{RoB}} - 1) \cdot 45) \quad \mu s \\ T_{\mathrm{GTP}} &= 80 + (N_{\mathrm{RoI}} - 1) \cdot 40 \quad \mu s \end{aligned}$$

where for the local and global processors we use parametrisations reflecting what we have learned from the simpler configurations.

In order to extend the model we used for the pipeline configuration, consider, say, the local processors. If a local processor has a latency of $T_{\mathrm{FeX}} \mu s$ then it can process one event every $T_{\mathrm{FeX}} \mu s$. If there is more than one FeX then the set of FeXs may process an event every $T_{\mathrm{FeX}}/N_{\mathrm{FeX}} \mu s$.

Thus with the definitions given above we extend the simple model thus:

$$\mathrm{Throughput}^{-1} = \mathrm{Max}\left(T_{\mathrm{RoIDist}}, T_{\mathrm{RoB}}, \frac{T_{\mathrm{FeX}}}{N_{\mathrm{FeX}}}, \frac{T_{\mathrm{GTP}}}{N_{\mathrm{GTP}}}\right)$$

Looking through the table we see that our model certainly gets the basic trends correct. If we consider that our model has six parameters and is predicting a table of fifty-six numbers to within 10% almost everywhere, and often more precisely, we can regard it as quite successful. An indication that the GTP latency for a single RoI is indeed 80 μs (as suggested in section

4.3) can be seen from the throughputs measured for the configurations with a single GTP and multiple FeXs, where the GTP would be expected to limit the system. The model predicts this, and the measurements confirm it.

The latencies are calculated by simply summing the component latencies as done in table 2 but now with T_{FeX} depending on the number of RoBs. The latency for the RoI-Distributer is held fixed for the calculation because the stagger in the passing of messages over VME (Dist to RoBs) is concurrent with the (longer) FeX collection time and thus is not seen.

Since a single RoI is used per event the only point at which event fragments are collected together is in the Local processors. Now if all processors are equal and if they are supplied by equal network bandwidth, then the latency does not depend on which processor deals with the event. Thus the only factor affecting the event latency is the collection time in the FeX. Looking only at the results for systems with a single GTP we can see that this is the case to within 2 %.

For systems with two global processors the latencies increase somewhat. The reason for this is that as noted in section 3.6 not all processors are supplied with equal bandwidth, and in fact to get to the second global processor two C104 switches must be traversed. Additionally, events which go to the second GTP must then traverse an extra DS-link in order to pass through the multiplexor task as described in section 3.8.

It is not obvious why the differences in the latencies between configurations with one GTP and those with two GTPs are as large as they are, but apart from one measurement, it remains the case that the latency depends only upon the number of RoBs contributing to the RoI.

The model predictions for the throughput are not affected so much by this last point, as the throughput is insensitive to low latency components (such as an extra DS-link traversal), and depends only on the limiting component.

4.4.2 Fixed Rate running

For the largest system we had available, five RoBs, four FeXs and two GTPs we have perfromed some additional measurements in fixed-rate mode.

Figure 24 shows how the latency varies with throughput. There is a definite upwards tendency above 5 to 6 kHz. This is to be expect in this wide-system configuration (as opposed to the pipeline) because now we may have contention across the switch. At 8 kHz with an RoID message size of 160 bytes, each FEX is receiving data at about 1.6 MBytes/s, about half

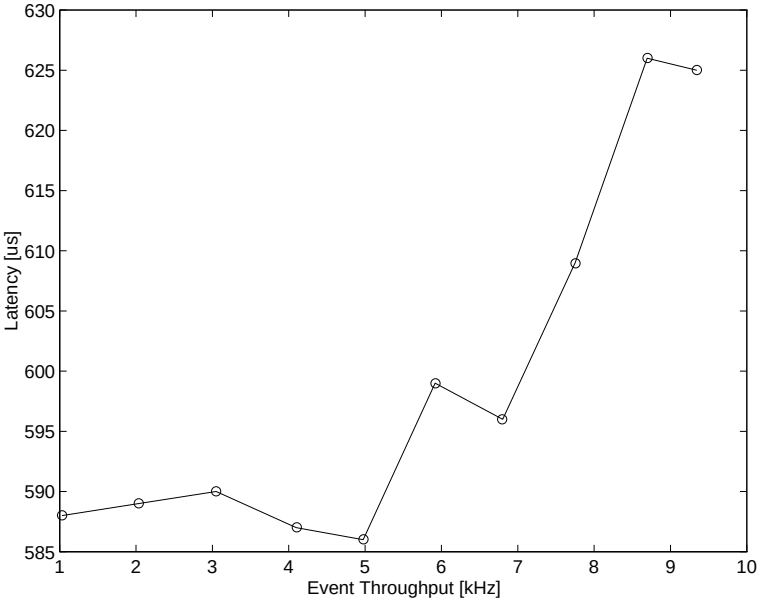

Figure 24: Latency as a function of throughput in fixed rate mode.

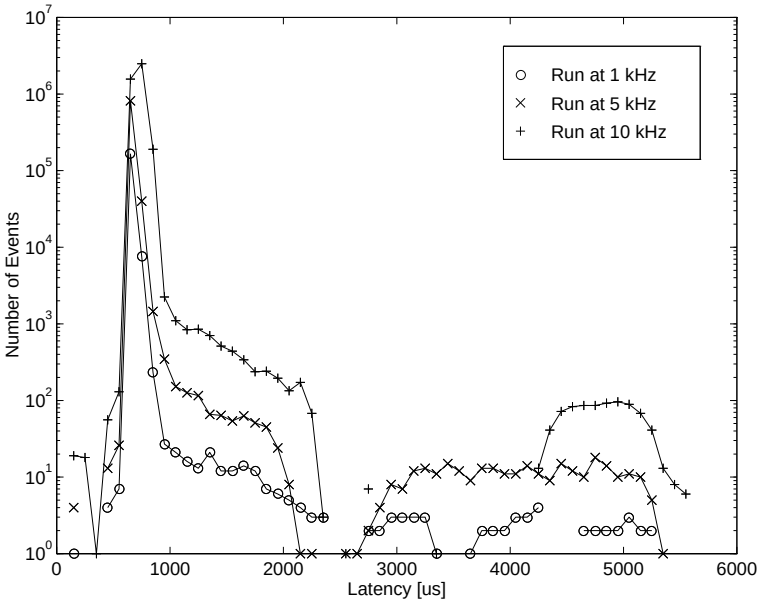

Figure 25: Latency distributions at different rates

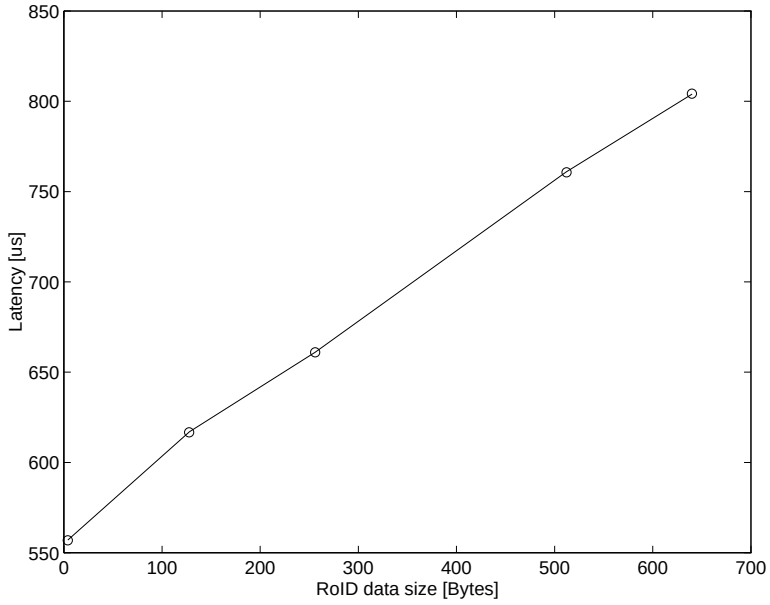

Figure 26: Latency as a function of RoID message size.

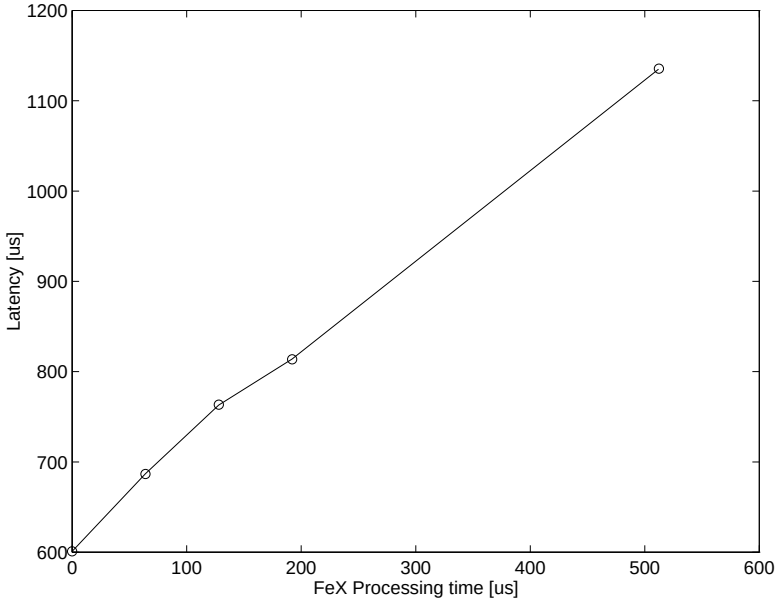

Figure 27: Latency as a function of FeX processing time.

of the bandwidth of a single link. The latency distributions, figure 25, can be seen to be fairly constant, until at 10 kHz, the main peak is becoming significantly wider.

The latency as a function of RoID message size and FeX processing time is shown in figures 26 and 27. These measurements were made at a rate of 5 kHz (4 kHz for the last point of figure 26).

4.5 Soak Test

The full system configuration (i.e. five RoBs, four FeXs and two GTPs) was left running at around 10 kHz for a total of seventeen million events. It was then stopped by the supervisor because the number of timed-out events was set at sixteen. It is likely that these events were distributed fairly evenly in time and were not due to a system failure, i.e. the system would have continued to run if this parameter had been set higher.

A reason why timeouts should occur at all can be inferred from looking at figure 28. This shows how the sampled latency, i.e. the level-2 latency averaged over a four second interval, varies with time. The two large peaks

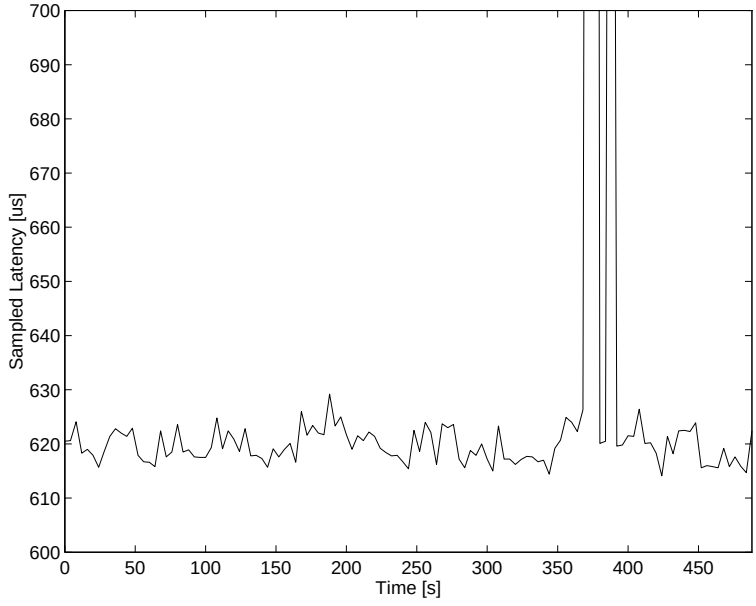

Figure 28: Variation of latency with time.

are probably due to one of two things:

- Background network activity on the networked processors (i.e. the RIO2s)
- Periodic printing of statistics from the Transputers (over Ethernet)

Either of these things can affect the accuracy of measurements made in the shorter runs, and we were careful to avoid them as much as possible.

5 Conclusions

We have constructed a vertical section of the level-2 trigger and demonstrated the architecture-B protocol to be functional. We have performed a series of measurements of the system and produced a simple model with a few parameters which fits the measurements to within about 10 %.

References

[1] Note in preparation.

[2] ATLAS Technical Proposal CERN/LHCC/94-43 p 139.

[3] ATLAS Technical Proposal CERN/LHCC/94-43 p 139.

[4] Fred Wickens *Proposed Data Formats for T2 Demonstrator Programme – Draft 1.1*

[5] See http://www.cern.ch/hsi/s-link/

[6] See http://atddoc.cern.ch/Atlas/

[7] See http://www.hep.ph.rhbnc.ac.uk/atlasT2/C40.ps

[8] See http://www.cern.ch/hsi/dshs/

[9] T.C. Carden et al. *High-performance computing nodes for real-time parallel applications* NIM A394 (1997) 211-218