PROTOTYPING HARDWARE FOR THE ATLAS READOUT BUFFERS

R.Cranfield (rc@hep.ucl.ac.uk), G.Crone, University College London
G.Boorman, B.Green (B.Green@rhbnc.ac.uk), Royal Holloway University of London
O.Gachelin, M.Huet (huet@hep.saclay.cea.fr), P.LeDu, M.Mur, Saclay
H.Boterenbrood, P.Jansweijer, G.Kieft, J.Vermeulen (i73@nikhef.nl), NIKHEF
A.Kugel (Andreas Kugel@ti.uni-mannheim.de), R.Rissmann, University of Mannheim

Abstract

Current design studies for the ATLAS readout assume that detector data is held in standard buffers until needed for full event building. These buffers are fed with data in parallel via a large number of high bandwidth links from the sub-detector readout electronics for events accepted at the first level, but only a subset of this data is required for level 2 trigger processing, selected from "regions of interest" identified at level 1. It is assumed that such buffers will not be available "off-the-shelf", so ATLAS is pursuing a programme of buffer design, building prototypes and exploring architectural options. This paper describes both the current mainline prototypes, which use Intel i960 processors and interface to PCI, as well as an entirely FPGA-based solution, and a SHARC DSP-based board which is being designed to explore an alternative buffering strategy.

1. ASSUMPTIONS

The basic strategy[1] adopted by ATLAS for taming the enormous data rate expected after Level-1 (hardware) triggering is to concentrate Level-2 processing on only that subset of the data issuing from "regions of interest" identified by Level-1. In this way only a fraction of the data needs to be merged from the readout lines into the Level-2 processors, and subsequently only the fraction of events selected at Level-2 needs to be sent on to the Event Filter.

This strategy implies the incorporation of special buffering into the readout lines where data can sit until either rejected by Level-2 or required by the Event Filter. The system components responsible for the whole buffering process are referred to as ROBs (Read Out Buffers) and are assumed to form the first common stage in the readout chain. A ROB has several interfaces: it must be able to accept detector data at a high rate from one or more Read Out Lines; it must communicate with the Level-2 system, servicing requests for data "fragments" from regions of interest; it must communicate with the Event Filter, servicing requests for whole event data for accepted events; and it must communicate with the readout software infrastructure, providing downloading, initialisation, monitoring and debugging facilities as required.

The ultimate structure of a ROB is not yet decided, but for hardware prototyping purposes its functionality has been split into an input component (ROB-in) which includes the data buffer, a supervisory component (ROB-controller), and a network interface component (ROB-out). This enables different groups to concentrate on the development of the different parts, as well as allowing for experimentation with different degrees of input and output multiplexing. This paper describes recent work on the development of the ROB-in component.

The basic references for this work are the ESA-style User Requirements Documents (URDs) which have been drawn up for both the ROB[2] and the ROB-in[3] on the basis of current knowledge. Final versions of these documents are planned for the end of the prototyping phase.

2. PROTOTYPES

The rationale for designing and building ROB-in prototypes is two-fold. Real modules are needed for the construction of "vertical slices" of both the Level-2 trigger and the DAQ front-end, required for proving designs for these sub-systems. At the same time the design process gives us hands-on experience of the problems intrinsic to building an appropriate hardware buffer. This is particularly important since it is by no means clear that such buffers will be available as "Commodity Off-The-Shelf" items and they may well need to be custom-made. This is notwithstanding the fact that there are now commercially-produced boards[4][5] suitable for the implementation of a ROB-in. The scale of the eventual ATLAS buffering (~2,000 Read Out Links) justifies further exploration of the cost, density and modularity of the final buffer system through development of the present prototypes and their use in multi-channel configurations.

One of the commercial boards, the **microEnable**, is described here. It is a multi-purpose design with SRAM memory, FPGA and S-Link interfacing, that evolved from development work at Mannheim University[6]. The ROB-in "design" consists of programming the FPGA such that the ROB-in requirements are satisfied. The current ROB-in implementation on this board includes a Level-2 pre-processing task, which is potentially useful for sub-detectors such as the TRT, where only a

compacted form of the data is needed for making the Level-2 decision.

Two of the custom prototypes have evolved from earlier designs that used Texas Instruments' DSPs. The UCL/RHUL **ROB-in RD** board[7] followed on from the C40-based *T2B* buffer successfully used in early small-scale trigger demonstrators, whilst the Saclay **ROBIN**[8] was developed from on an earlier C80-based design. Both these boards are intended for use in demonstrators associated with the Level-2 *Pilot Project*[9], as well as in prototype multi-ROB-in *ROB Complex*[10] configurations. The RD board is also being used in the DAQ *Prototype-1*[11] front-end demonstration.

The **CRUSH** board[12] is a recent design from NIKHEF using the SHARC DSP and a different buffer management technique and is aimed at local tests of ROB Complex configurations.

3. APPROACH

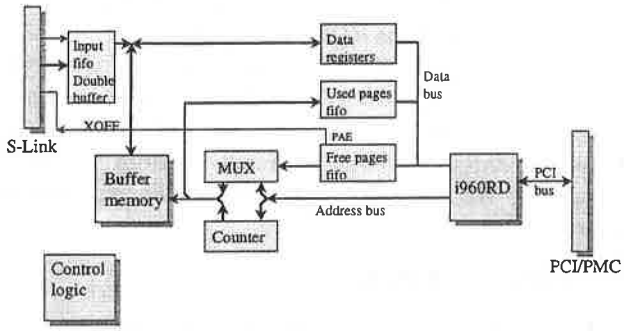

Fig 1: UK ROB-in RD

The two current mainline ROB-in prototype variants, the UK ROB-in RD and Saclay ROBIN, follow similar design approaches. Detector data enters (from one of many Read Out Drivers associated with a sub-detector) on a Read Out Line and is written into fast dual-port memory by programmable logic, under the direction of buffer manager software running on a microprocessor.

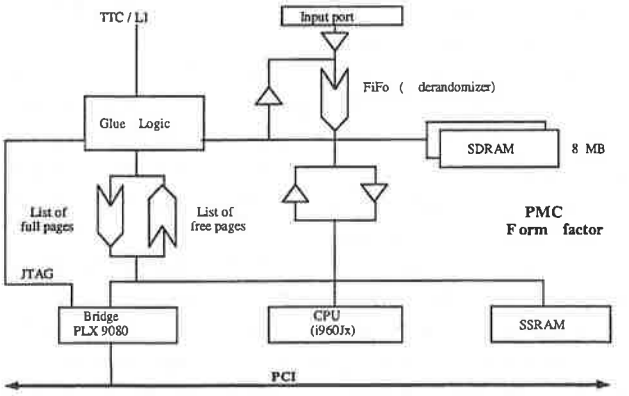

Fig 2: Saclay ROBIN

The buffer manager software receives data request messages across a PCI interface and initiates DMAs of the identified data back across the PCI interface for subsequent transmission to Level-2 and/or Event Filter.

The buffer management is performed by allocating the buffer memory in fixed-size pages and keeping track of which pages contain fragments of which events. The programmable logic needs only to be supplied with the addresses of free pages, and to inform the buffer manager how many bytes have been written into each as they are used; the event id itself can be read by the buffer manager directly from the event data in the buffer.

The new design from NIKHEF employs a simpler circular event buffer, copying data out of the buffer into buffer manager memory (internal memory of the SHARC DSP) whenever it is in danger of being overwritten. This design too uses programmable logic and dual-port memory with a microprocessor-based buffer manager and communication and output across PCI.

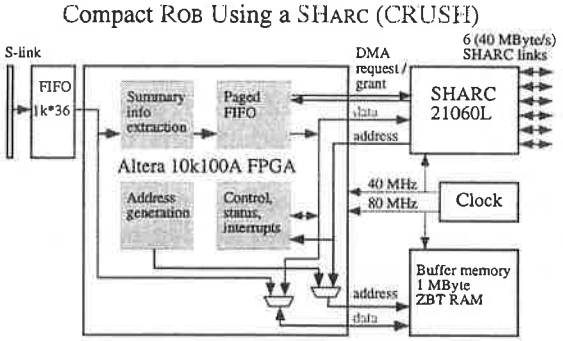

Fig 3: NIKHEF CRUSH

The microEnable-based design is completely realised in programmable logic (Xylinx FPGAs). It uses a circular buffer like the NIKHEF board, but there is no microprocessor and the buffer management is built directly into the firmware.

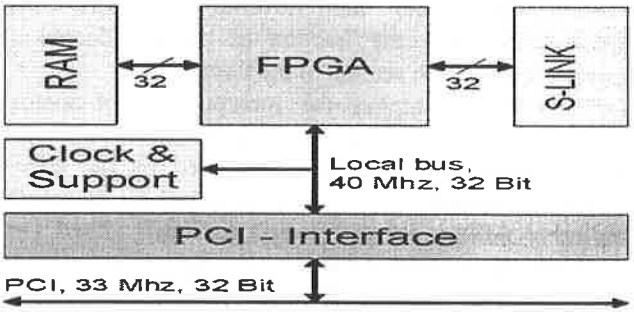

Fig 4: microEnable

4. COMPONENTS

4.1. *Data Input*

Data is assumed to enter the buffers via a dedicated link from the detector readout components. The

canonical speed for this link is 100 MB/s or 100 kHz of 1KB event fragments. The implementation of this link is still under discussion, but most of the prototyping work assumes that it will be interfaced according to the S-Link specification[13], which provides a 32-bit-wide data connection together with control and status lines.

The S-Link interface provides an error bit that can be read by the ROB-in and reported to the controlling infrastructure. It also carries an XON/XOFF protocol, which is used both for holding off data prior to initialisation and signalling when the buffer memory is nearly full. It is assumed that this handshake will be propagated back along the readout chain in some way ultimately blocking the data flow out from the sub-detector.

The detector data is fed into an input fifo which accommodates the lack of synchronisation between the clock controlling the Read Out Link and the clock controlling the buffer memory. In general it can therefore be quite short, though in the Saclay design it is also necessary to buffer data in this fifo before it can be burst into the buffer memory.

It is assumed that the beginning and end of each event fragment is marked in the data. The S-Link specification defines a special bit marking a word as a control word, and the proposed Read Out Link format[14] defines a pair of control word patterns (BOB and EOB) indicating the beginning and end of a data block respectively. Actually only the end marker is strictly necessary, enabling the ROB-in hardware to know when to record the event details and to ready itself for the next fragment.

4.2. *Buffer Memory*

The buffer memory must take in data at an average rate of 100 MB/s i.e. one 32-bit word every 40 ns. However the memory must be a lot faster than that since it must also be read both by the buffer manager software (to pick up the event ids) and by the DMA engine responsible for outputting event fragments to Level-2 and the Event Filter. The rates for these tasks are about 400 kB/s and 10 MB/s respectively.

In the ROB-in RD normal fast (10ns) asynchronous SRAM is used and advantage is taken of the difference in access rates required on the two ports with the output side being allowed up to one access in every four. Memory size is limited by the size of available SRAM chips with the current prototypes providing 1 MB total buffering.

The Saclay ROBIN meanwhile uses SDRAM. This permits a larger memory (8 MB) to be accommodated in the available board space though at the expense of increased logic complexity and with the constraint that the fastest access requires the use of burst mode. This in turn has resulted in the selection of a longer input fifo, as noted above, and the restriction of fast access from the microprocessor side to read-only. Slow read and write word access is possible, however, from either local or host CPU.

The microEnable uses standard asynchronous SRAM (0.5 - 2 MB), with dual-port capability being emulated in the FPGA by running the RAM-control task at twice the speed (40 MHz) of the I/O task.

The CRUSH design makes use of ZBT (Zero Bus Turnaround) memory, which avoids the wait states necessary with fast synchronous SRAM when alternating between reading and writing. The buffer memory runs on an 80MHz clock, allowing one write from the S-Link interface every 25 ns (resulting in a maximum input bandwidth of 160 MB/s) plus one access from the SHARC. Due to pipeline delays in the FPGA and the properties of the bus protocol of the SHARC the bandwidth to/from the SHARC is limited to 40 MB/s.

4.3. *Microprocessor*

Both the ROB-in RD and Saclay ROBIN make use of the Intel i960 family of I/O microprocessors. The Saclay ROBIN uses the more established JX variant, whilst the London-designed board incorporates the newer RD chips. These latter chips have the convenience of built-in PCI bridging, but introduce the complications of increased power dissipation and BGA mounting technology. The i960 family provides an appropriate balance between speed, power dissipation, size and complexity.

As already noted, the NIKHEF design uses instead a SHARC DSP as its microprocessor. This also provides an appropriate balance of operating parameters, as well as the possibility of interconnection of ROB-ins via its proprietary communication links.

The fourth prototype does away with a microprocessor altogether. This potentially increases the speed of buffer manager processing, though making the coding less transparent and hence arguably less versatile. The trade-off between hardware and software implementations of buffer management in the ROB-in is an important open question.

4.4. *Buffer Management*

The heart of the ROB-in design is the buffer manager. It is this that allows the input data to be written at a continuously high rate whilst also allowing for the variable amount of time that an event fragment might need to remain in the buffer during Level-2 processing or whilst waiting to be requested by the Event Filter.

In the page-managed schemes (ROB-in RD and Saclay ROBIN) all the pages can be used since the buffer manager notifies the hardware of all the free pages, and keeps track of their usage. The only inefficiency arises from the use of a fixed page size. This is no problem for those sub-detectors with fixed event fragment sizes (no zero-suppression), but in the case of

variable fragment sizes a compromise has to be established between minimising the number of pages used per event fragment and minimising the amount of wasted space within pages.

An alternative buffer management scheme has been tested in the ROB-in RD in which the overhead associated with tracking several pages per event has been minimised by linking the pages of an event directly in the buffer memory. This means that small pages can be used without penalty and enables particularly efficient memory utilisation.

Both the CRUSH and the microEnable designs incorporate a circular buffer, which simplifies the input task, since it needs only a cyclically repeating address counter. However it means that an event fragment can block the buffer if it needs to be retained for longer than the buffer cycle time. In the CRUSH design it is assumed that the SHARC microprocessor will transfer such an event fragment into the microprocessor's own heap memory. This then requires a management scheme for the fragments on the heap, but should be feasible if it is a relatively rare occurrence. In the microEnable prototype it is also planned to try a fifo-style buffer as an alternative to the circular buffer.

All the buffer management schemes need to maintain some sort of index table to associate an event ID with the address of its data fragment in buffer memory. In the i960-based designs the microprocessor reads which pages have been used by the hardware via a "used page fifo" and is then able to read the event IDs directly from the buffer. In the SHARC-based design the hardware itself notes the IDs (along with a few other key data words) and passes these to the microprocessor together with the address window of the event fragment in the circular buffer. In the FPGA-based design the FPGA generates a simple lookup table as it reads event fragments into the circular buffer.

4.5. *PCI & Format*

All four prototypes use PCI for communications other than the front-end data input. The PCI bus provides a convenient common connection standard for prototyping work, as well as constituting a real technology option for the final system, since the total bandwidth required for a operation of a single channel ROB-in is about 15 MB/s, which is well within the PCI spec.

Both the i960 modules come in PMC format, enabling them to be plugged directly onto a motherboard via the PCI bus. The possibility then exists of constructing a complete ROB from e.g. a VME single board processor as a ROB-controller, with a pair of PMC daughter boards, acting as ROB-in and ROB-out respectively.

The microEnable module and an alternative version of the ROB-in RD come in a standard PCI format suitable for plugging directly into a PCI slot in a standard PC, whilst the CRUSH module communicates via its SHARC links with a second PCI module containing a SHARC and a PCI interface.

4.6. *Error-Handling*

The ROB-ins must be sensitive to several kinds of errors in the data-stream, as well as being able to recognise errors reported by the link system itself. Event fragments might arrive without BOB or EOB markers and/or they may contain too much data. Level-2 or Event Filter requests may be received for data that has not yet arrived at the buffer and/or the data may never arrive. Data may also be corrupted in various ways. Whilst data corruption in general has to be left for further processing downstream, corruption of identifying information must be handled non-fatally at the ROB-in.

In general situations in which data is not released quickly enough from the buffers are handled by the S-Link handshake mechanism. The ROB-in RD, for example, asserts XOFF whenever the number of free buffer pages drops below a certain level, or if the input fifo fills too much. This means that extra-long event fragments, possibly caused by the absence of EOB markers, will not disrupt operation; they can be recognised by the buffer manager and immediately released.

Requests for events not yet received can be handled by flagging the corresponding entry in the index table, so that the request can be satisfied once the data arrives.

It is not yet clear whether fragments can be guaranteed to arrive in strict order of ID, but the buffer manager can easily check that they are not grossly out of sequence, or for duplicates, as part of the indexing process.

The thorniest problems concern event data and requests that never match up. This begins to be a concern once the buffer manager needs to re-use an already occupied slot in its index table. It is easy to avoid the immediate problem by linked-list extensions to the simple lookup table, but it is not yet clear what should be provided in the way of timeouts from the point of view of overall system efficiency.

4.7. *Monitoring*

It is assumed that event fragments will be requested for monitoring purposes as well as for the mainstream DAQ dataflow. Such requests can simply be handled by the standard fragment-request mechanisms. What is not yet decided is the extent to which individual ROB-ins will need to select and/or store events of different types; this could have a major impact on future ROB-in design.

5. OPERATION

The i960-based prototypes boot from ROM, but there is enough on-chip memory for the main buffer manager code to run from cache, with consequent speed improvements. Code has also been downloaded across PCI, which is useful for testing and might be convenient

for online program changes. JTAG is used extensively for downloading both programmable logic and microprocessors and also as a route for emulator-based debugging of the microprocessors.

The SHARC prototype is booted across a SHARC link by the SHARC on its associated PCI interface board, itself booted via PCI. The SHARC on-chip memory has room for the buffer manager code, index table and overflow buffer, as well as a library supporting terminal and disk I/O via the host PC.

Testing is carried out both by running special programs in place of the regular buffer manager and by connecting to auxiliary test facilities, such as the SLIDAS S-Link data source. The designs also incorporate mechanisms for injecting data directly into the front-end of the ROB-in under host control.

SOFTWARE

The buffer manager software on the microprocessors is written in C for transparency and possible portability. Earlier experience with the "T2B" buffer shows that C code can achieve about half the speed of assembler in this kind of application.

Since the buffer manager's role is essentially to service its input sources, with no significant background tasks, it has generally been implemented as a large polling loop, rather than employing interrupts. Comparative timing measurements justify this design choice. Efficiency has been gained, however, by servicing a group of inputs from each source at a time.

The Xylinx chip on the microEnable board is coded using CHDL, a C-like high-level language developed for easy programming of FPGAs.

At the moment each prototype uses its own proprietary programs, but the aim is to define a set of common APIs for future prototyping work which could eventually evolve into the production software. Work has begun on this with the first iteration of an API for the operation of a ROB within the Level-2 *Reference Software*[15] framework.

As well as the buffer manager code itself attention must also be paid to the interaction between the ROB-in and the rest of the world. Currently APIs have been defined[16] in the context of the DAQ Prototype-1 covering communication between a ROB-in and the rest of the DAQ front-end and these have now been implemented for the ROB-in RD.

6. PERFORMANCE & STATUS

The designs described here are intended primarily for prototyping experiments to be conducted over the next eighteen months, essentially the Level-2 Pilot Project and the DAQ Prototype-1. Versions now exist of all four boards, though some hardware debugging and refinement remains to be completed.

Early results from simulation environments and measurements of loop times suggest that all the prototypes are fundamentally capable of coping with the specified 100 kHz event rate, and can comfortably provide the required output bandwidth.

7. FUTURE DEVELOPMENT

The main focus of ROB-related work in the Level-2 Pilot Project will be to analyse the options for a ROB Complex, defined to be a generalised buffering system incorporating one or more front-end links and network connections. This will involve exploration of hardware possibilities and constraints as well as extrapolation of technology and performance relevant to the Level-2 trigger, preparatory to a final design programme starting in 2000. At the same time the DAQ Prototype-1 programme will explore the integration of ROB components into the full DAQ system.

As well as using the current ROB-ins in Level-2 vertical slice demonstrators it is planned to configure several test setups, each containing a number of the current prototype ROB-ins, to measure the performance of different architectural organisations for a ROB Complex. It is also planned to pursue paper designs for possible multi-ROB Complexes, which will explore possibilities for tighter coupling and closer packing of ROB components.

8. REFERENCES

1 http://atlasinfo.cern.ch/Atlas/GROUPS/DAQTRIG/LEVEL2/DOCUMENTS/level2docs
2 http://www.cern.ch/HSI/rob
3 http://www.cern.ch/HSI/rob-in
4 http://www.silicon-software.com
5 http://www.ces.ch/Products/Products
6 http://www-mp.informatik.uni-mannheim.de/groups/mass-par-1/parallelproc
7 http://www.hep.ucl.ac.uk/atlas/rob-in/
8 http://www-dapnia.cea.fr/Phys/Sei/exp/ATLAS/rob/index
9 http://atlasinfo.cern.ch/Atlas/GROUPS/DAQTRIG/L2PILOT/l2pilot
10 http://www.nikhef.nl/pub/experiments/atlas/daq/ROB
11 http://atddoc.cern.ch/Atlas/
12 http://www.nikhef.nl/pub/experiments/atlas/daq/Nikhef-studies
13 http://www.cern.ch/HIS/s-link/
14 http://www.cern.ch/HSI/atlas/format
15 http://www.cern.ch/Atlas/project/LVL2testbed/www/
16 http://atddoc.cern.ch/Atlas/Notes/063/Note063-1
http://atddoc.cern.ch/Atlas/Notes/065/Note065-1
http://atddoc.cern.ch/Atlas/Notes/071/Note071-1