RoB-in Design Notes

Gordon J. Crone

June 30, 1997

Chapter 1

Processor options

1.1 Requirements

1.1.1 Processing power

The current implementation of buffer manager software, written in assembler and running on a $\sim$15 MIP processor, just about manages the designed 100kHz event rate. The equivalent program in C ran at about half the speed, therefore to allow us to use C instead of assembler we need $\sim$30 MIPs. However this is a minimal implementation without the monitoring and other requirements specified in the URD. In order to **comfortably** implement all functionality at the required rate, without resorting to coding in assembler, we should double this number again to give us a processor power requirement of $\sim$60MIPs.

1.1.2 Power consumption

For a PMC module total power consumption is limited to 7.5W.

Chips consuming a large amount of power require heat-sinks which are a problem in terms of the physical size requirements.

1.1.3 Physical size

A PMC card is not very big!!

1.1.4 DMA

After some discussion we have decided that DMA capability is not required in the processor. The only place where DMA is desirable is in the PCI bridge.

1.2 The contenders

- TMS320Cxx (see http://www.ti.com/sc/docs/dsps/products.html) (C40/C44/C31/C32), DSP from Texas Instruments (see http://www.ti.com)
- ADSP-2106x SHARC, DSP from Analog Devices (see http://www.analog.com)
- Am29040, RISC microprocessor from Advanced Micro Devices (see http://www.amd.com)
- i960 (see http://www.intel.com/design/i960/) , RISC microprocessor from Intel (see http://www.intel.com) .
- Pentium, microprocessor from Intel (see http://www.intel.com)
- PowerPC, Motorola (see http://www.mot.com) /IBM microprocessor
- IDT79R3051, Integrated Device Technology (see http://www.idt.com) microprocessor

1.2.1 Quick comparison

Processor	processing power (MIPS)	power consumption	physical size
C40 (60MHz)	30	1.75W	1.86“
C30 (60MHz)	30	?	0.9”
SHARC (40MHz)	40	1.75W	1.428“
Am29040 (50MHz)	68.8 *	1.65W	1.54”
i960 (RP) (33MHz)	31 *	3W	35mm
Pentium (100MHz)	?	10.1W	1.95“
Pentium (610=) (75MHz)	?	7.6W	24mm
PowerPC 602 (66MHz)	40 **	1.2W	?
PowerPC 603 (80MHz)	75 **	3.0W	? ***
PowerPC 603e (100MHz)	120 **	3.5W	?
IDT79R3051 (40MHz)	35	2.5W	1.195”

```
*    Quoted as 'VAX MIPS'
**   SPECint92
***  240 CQFP / 256 BGA
```

1.2.2 Notes

- C30, C40, SHARC and i960 all have some DMA channels.
- Pentium & Powerpc 603e look promising for speed but are they too large / power hungry?

- Cxx, SHARC & Am29040 all have in circuit emulator development / debugging environments.
- PowerPC requires PowerPC based development system (RIO2?).
- The i960RP looks interesting because of the ease of interfacing to PCI (it has 2 PCI busses in addition to its local bus).

1.2.3 Other considerations

Smaller chips may require a heat sink where a larger chip could dissipate enough heat without.

Chips packaged with a BGA may be more difficult to debug (hardware wise) because it is not obvious how you would attach logic analyser probes.

Processors such as the Cxx would be much simpler to integrate since we already have experience with the C40.

Processors such as the PowerPC would require a new debugging environment.

High clock speeds would make the design much more difficult in terms of routing of signals on the PCB.

1.3 Conclusions

- Pentium consumes far too much electrical power.
- Current models of C3x, C4x, SHARC & i960 do not appear to have quite enough raw processing power, but:
  - C30 should not be ruled out yet because writing in assembler rather than C it could cope and it has the following advantages:
    * Same family as C40 for which we have experience, development tools, hardware debugger.
    * Small size
    * Low power
  - i960RP should not be ruled out because integral PCI bus should make the design much simpler:
    * Fewer components as no separate bridge chip.
    * Less logic, no multi-master bus impementation.

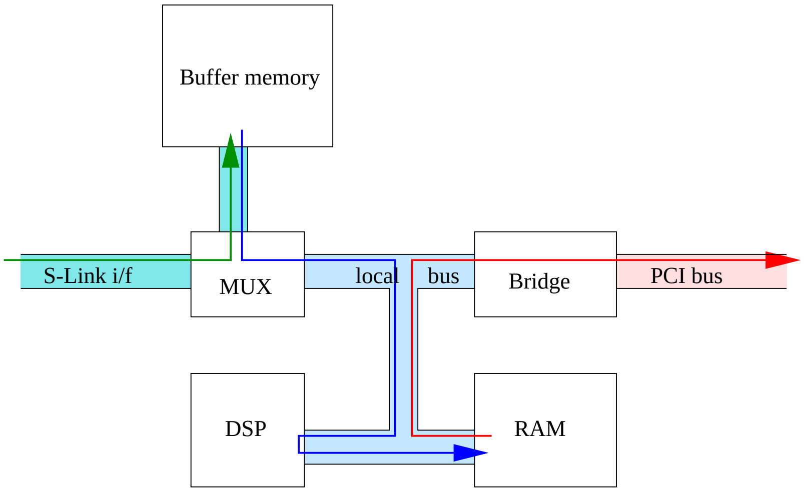
1.3.1 Our choice

We have decided to use the i960RP as this seems to be the easiest from the hardware point of view. If we find that the processing power is really not enough for the job the newly announced i960RD (a clock doubled version of the i960RP) should give us the extra we need.

Chapter 2

DMA and multiplexing of memory access

2.1 Original thoughts



Multiplexing access to buffer memory with 3:1 priority to input side would cause a large waste of bandwidth on the PCI bus if the data were copied directly from the buffer to the PCI bus. Either use intermediate buffer on the RoB-in side of the PCI bridge or change dual porting strategy.

Use DMA to copy data from main buffer to intermediate buffer, **implications for processor choice**.

Set up a chained DMA from intermediate buffer to PCI bus. Do not read from PCI until DMA has completed.

2.2 Note from discussion 17-Sep-1996

This is probably an unnecessary worry if a sensible arbitration strategy is used for the PCI bus.

- If no-one else wants the bus it doesn't matter if we use it inefficiently.
- If someone else wants the bus the arbiter should let them have it within a few ($\sim$6) cycles and should let them keep it for a few ($\sim$6) cycles. While we don't have the bus our bridge will be collecting data in an internal FIFO until we get the bus back so when we do get it back we will have data ready to go.

2.3 DMA with i960RP

The DMA controller on the i960RP allows for moving data between the i960 local bus and the PCI bus only. It does not allow copying of data from one place on the local bus to another.

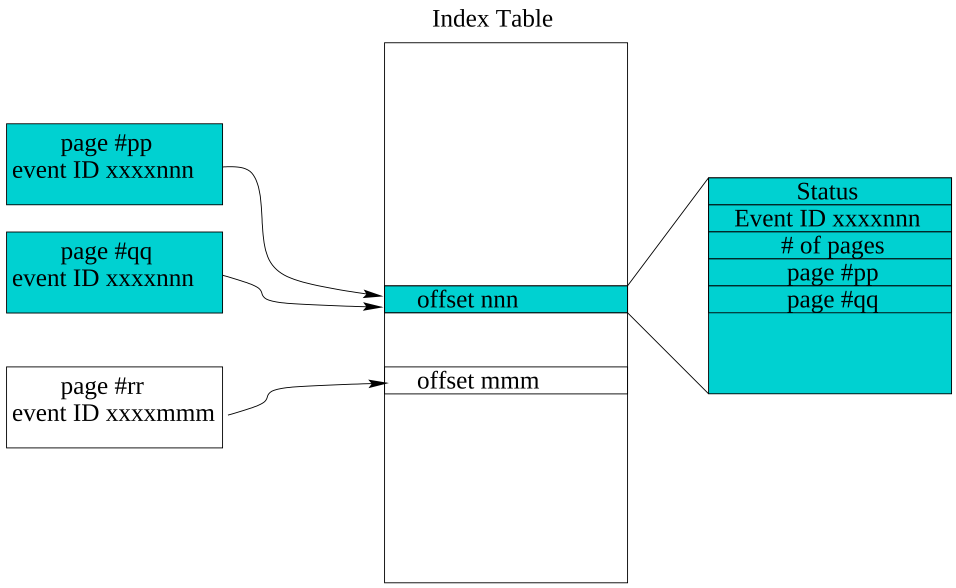
The only sensible way to send data out of the RoB-in is to use the i960RP's DMA controller, we do not have enough CPU cycles to spare to copy data under CPU control.

One particular feature of the i960RP's DMA controller is the ability to append a new descriptor to the end of an already running DMA.

Chapter 3

Event Fragment Indexing

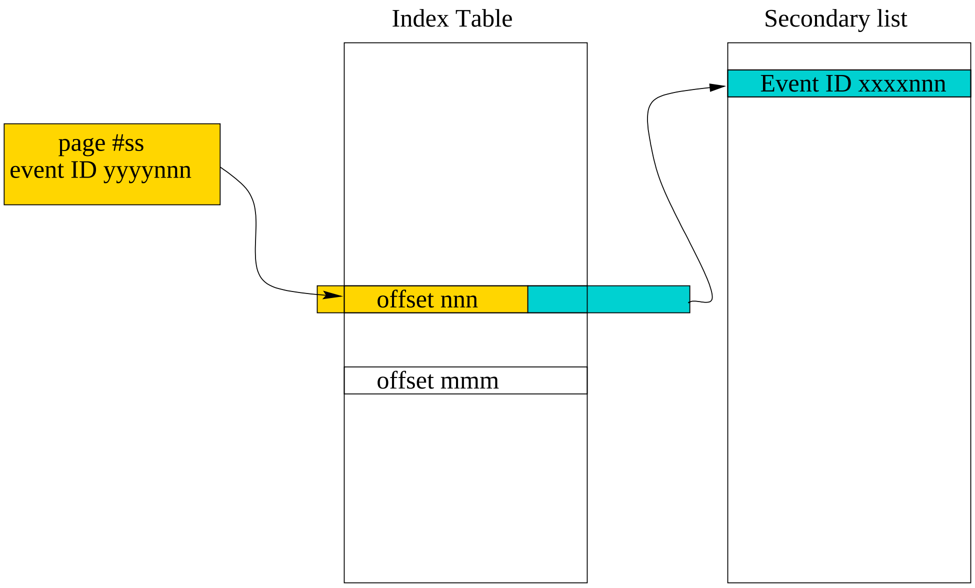
3.1 Main index table



The events are indexed by using the least significant 10 bits of the event ID as an offset into a fixed size data structure. This allows for a span of 1024 between the lowest and highest event ID's in the buffer (or $\sim$10ms @100kHz), we expect to have on average $\sim$100 events buffered at any one time.

3.2 Secondary list(s)

In the new RoB-in prototype it is proposed to supplement the main index table strategy with secondary lists for longer lived events.



- Pages are entered in appropriate place in the index table as they are read from the used page FIFO.
- Pages which would overwrite an existing entry in the index table cause the existing entry to be moved to a "forgotten list" before being inserted.
- ??Accept decisions cause the entry for the accepted event to be moved to an "accepted list".??

3.3 Releasing fragments

Fragments are released by writing their page numbers back on to the free page FIFO. If the fragment was still indexed in the index table, its entry is marked as free. If it was on the secondary list its slot number is placed on the list of spare slots.

3.3.1 Unresolved points

- What happens if a decision record rejects an event which has been requested for monitoring?

3.4 Handling of "forgotten" fragments

Deletion of "forgotten" fragments after a timeout is an inherently dangerous policy. There may well be events with significant latency in the level 2 system and in the case of sequential processing with a "reject as soon as possible" policy as suggested by Saclay, the longer an event is in the buffer the more tests it has passed!

It should be a general principle that buffer managers do not autonomously destroy data. There are at least 2 solutions to the "forgotten" fragment problem which don't violate this principle.

1. The buffer manager could ask the supervisor for clarification of the status of the fragment (has this event been rejected?).

2. The supervisor could periodically send special records saying "all current events are in the range nnn - mmm". Allowing the buffer manager to remove fragments of events ouside the current range.

Chapter 4

PCI communications

The i960RP has a comprehensive Messaging Unit (MU) to provide communictaion between the host processor and the i960 core processor. However, it is still proposed in the first instance to use a simple software fifo (circular buffer) as described in the note PCI message passing in the RoB-in context (see PCI_message_passing.html) .

4.1 i960RP Messaging Unit

The i960RP MU has the following facilities.

4.1.1 Message Registers

These can be used to pass single words only as each word written generates an interrupt whose flag must be cleared before another can be written. **This makes them unsuitable for use as the head and tail pointer registers described in the earlier note.**

4.1.2 Doorbell Registers

These can be used to generate an interupt on the i960. This may be usefull for sending a reset command but otherwise it is not intended that the i960 software use interrupts.

4.1.3 Circular queues

These appear from the i960's perspective to be very much like the software fifos proposed earlier but to the host processor they look like real hardware fifo's, words are all written to or read from the same address.

The problems with using the circular queues for passing the actual messages are:

- The host processor can only read or write one word at a time as the port address is always the same.
- The host processor cannot see the head and tail pointers so it is unable to determine how much space remains.

The circular queues are much more suited to passing identifiers (addresses or indices) of message buffers with the message bodies being stored elsewhere.

Chapter 5

Error Handling

5.1 Possible Errors

List based on the RoB-in URD error handling requirements (see "http://www.cern.ch/HSI/rob-in/URD/URD_22.html#HEADING29)

- Unknown TTC ID / orphan fragment (see section 5.2)
- Transmission error
- Fragment too big error (see section 5.3)

5.1.1 Data-flow errors

- RoI Request for non-existent event (see section 5.4)
- Reject Decision for non-existent event (see section 5.5)
- Accept Decision for non-existent event (see section 5.6)
- Level 3 Request for non-existent event (see section 5.7)

5.2 Unknown TTC ID / orphan fragment

It is not planned to implement a TTC interface on the RoB-in prototype therefore these errors do not apply.

5.3 Fragment too big

This is a genuine problem, since the index table (see "index_table.html) consists of fixed length slots a fragment which takes up more pages than allowed for in the index table will overflow its slot.

5.3.1 Action

If a page cannot be placed in its correct slot in the index table the page will be released onto the free FIFO immediately and an error will be flagged in the slot's status word.

If the offending event is selected for transmission to level2, level3 or the monitoring process the truncated event will be transmitted along with the appropriate status word.

The number of occurrences of this error will also be counted.

5.4 RoI Request for non-existent event

An RoI request is received for an event fragment which does not exist in the RoB-in. This is probably because the event ID of the fragment is corrupt either in the RoI request or in the RoB-in's index table. **This may also happen if the supervisor sends the RoI request before the RoD has sent the fragment!** Perhaps because the RoB-in has asserted XOFF.

5.4.1 Questions

- Should the RoB-in do some further checks before deciding that there is an error?
- Should the RoB-in stop fielding RoIRs while XOFF is asserted?

5.4.2 Action

The RoB-in will send an RoI data record with an empty event fragment and an error flag set in the header.

5.5 Reject Decision for non-existent event

A reject decision arrives for an event fragment which does not exist.

5.5.1 Action

The RoB-in will do nothing other than count the error.

5.6 Accept Decision for non-existent event

An accept decision arrives for an event fragment which does not exist.

5.6.1 Action

For an accept decision things are a little more complicated as a level 3 request may follow. *Should we create an entry in the accepted events list?*

5.7 Level 3 Request for non-existent event

A level 3 data request is received for an event fragment which does not exist in the RoB-in. This is probably because the event ID of the fragment is corrupt somewhere. There is no legitimate way for this situation to arise.

5.7.1 Action

The RoB-in will send a level 3 data record with an empty event fragment and an error flag set in the header.

Chapter 6

RoB-in Self Test

The ROB-IN selftest will enable data to be inserted into the ROB at the input stage to allow the ROB-IN and buffer manager to be tested. To test the hardware function the data must have the beginning and end marked with a control word and must be able to be set to produce events of any length up to about 5 pages. With pages of 256 long words, to enable events greater than 4 pages would require 11 bits. Thus there should be an 11 bit register which can be set to the required event length, and an 11 bit counter to count to the end of the event. Each event must also have an event ID at a fixed position in the data – this can be generated as the first data after the beginning of event mark. At least 1024 event IDs must be generated so that the whole indexing table of the buffer manager is used; this will require a 10 bit counter. The rest of the event data is not important; it could be a repeat of the event ID or it could be a test pattern like 3c6. The maximum data rate of this test data source will be 134 MB/s. To stop this overrunning, the counter must be stopped by the xoff signal.

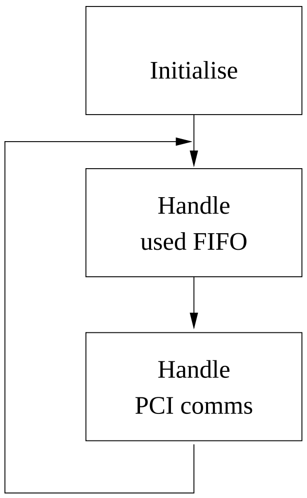
This test feature requires 32 registers and hasn't yet been added to the logic in the PLD. The currently used PLD has the largest capacity in the range, so if the test logic exceeds the available logic space the feature will be left out until a device with more space becomes available.

Chapter 7

Buffer Manager Software

This page still under construction 6-Jan-1997

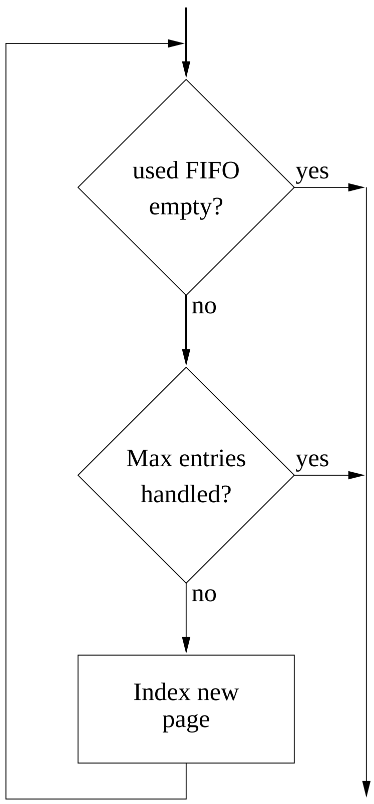
The main program is a polling loop with no interrupts required.



7.1 Used FIFO handling

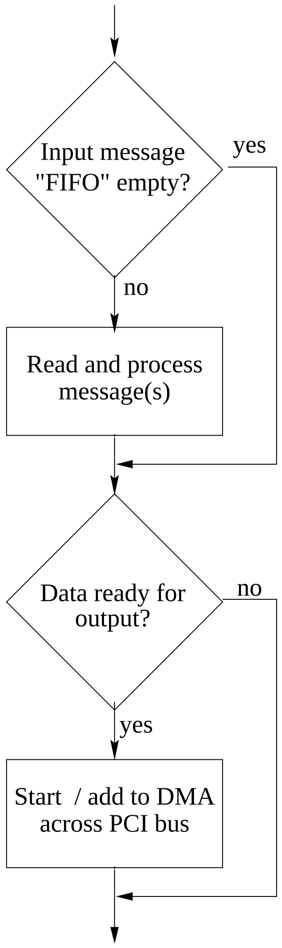
The RoB-in must handle an average input event rate of ~100kHz, since events may span more than one page the used FIFO must be handled at an average rate **higher than 100kHz**.

Since the used FIFO is long, it is only the average rate that is important and we can deal with the used FIFO entries in groups.


used FIFO empty?
yes
no
Max entries handled?
yes
no
Index new page

7.2 PCI comms handling

Data requests, decisions, control requests etc. will arrive as messages (see ../PCI_message_passing.html) through the PCI bus interface. Output data to level2, level3 and monitoring will all be sent through the PCI bus interface using a DMA. Care must be taken to avoid interference between incoming messages and outgoing data in order to avoid wasting PCI bus cycles.

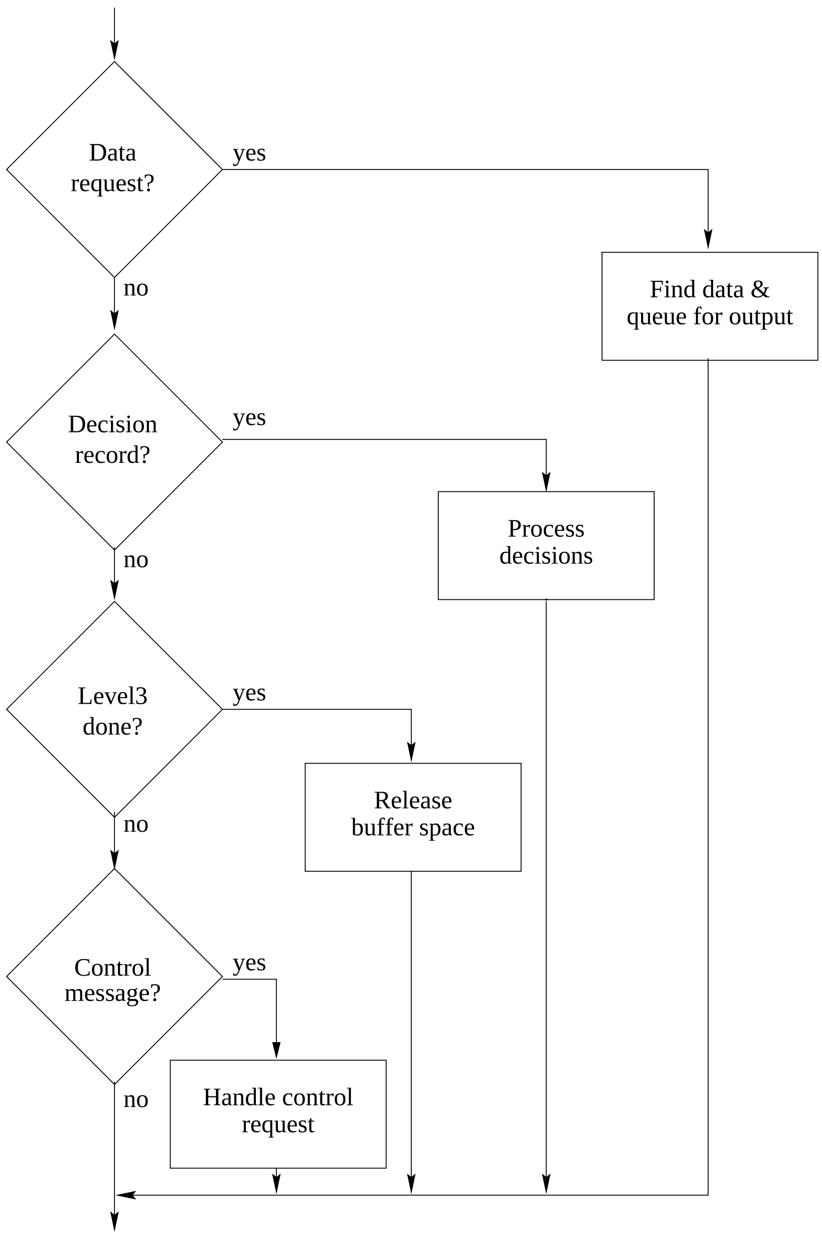


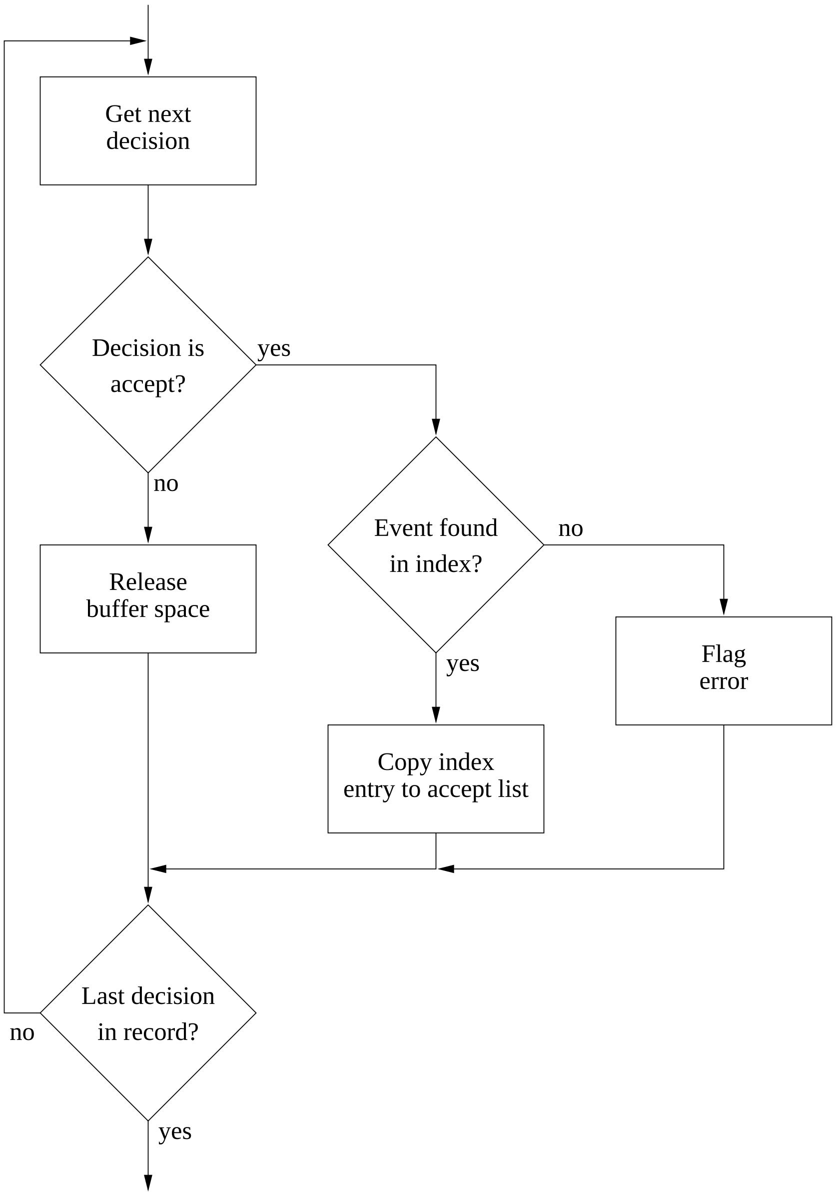
7.3 Message processing

Each message is processed according to its type.

7.3.1 Decision processing

Each decision record contains a group of up to 100 decisions.


Data request?
yes
no
Find data & queue for output
Decision record?
yes
no
Process decisions
Level3 done?
yes
no
Release buffer space
Control message?
yes
no
Handle control request


Get next decision
Decision is accept?
yes
no
Event found in index?
no
Release buffer space
yes
Flag error
Copy index entry to accept list
Last decision in record?
no
yes

Appendix A

RoB-in Glossary

ADSP-2106x "SHARC" DSP from Analog Devices Inc.

AMCC Applied Micro Circuits Coprporation – suppliers of PCI controller (bridge) devices.

AMD Advanced Micro Devices – suppliers of PLD's and microprocessors.

Buffer Manager (BM) The software that runs on the RoB-in to manage the fragment memory.

C30, C31, C32, C40, C44 DSPs from Texas Instruments (members of the TMS320Cxx family).

CMC Common Mezzanine Card, a standard for mezzanine cards specifying size, connectors etc.

Doorbell register A register which causes an interrupt when written to.

DSP Digital Signal Processor

Fragment buffer The paged managed buffer that contains the event fragments.

Free FIFO A FIFO containing a list of free pages, written to by the buffer manager, read by the input hardware.

Index table A table sorted by event ID which the buffer manager uses to keep track of event fragments.

PCI bus The Peripheral Component Interconnect local bus by Intel.

PMC PCI bus Mezzanine Card conforming to CMC spec.

PLD Programmable Logic Device.

PLX PLX Technology Inc. – suppliers of PCI bridge chips etc.

RoB Readout Buffer

RoB-in Readout Buffer input module.

SHARC A DSP from Analog Devices Inc. using *Super Harvard ARChitecture.*

Used FIFO A FIFO containing a list of addresses at which data have been stored, written to by the input hardware, read by the buffer manager.

Appendix B

PCI message passing in the RoB-in context

G.J.Crone 11/6/1996

B.1 Introduction

This note describes a simple message passing protocol based on software FIFO-s using mailbox registers in the PCI bridge chip as the read and write indices. Accesses to mailbox registers are effectively local and do not require mastering of the bus on the other side of the bridge, however since in most (if not all) cases there are two bridges between a device on a PMC and a device on the motherboard, access to the mailboxes from one side still requires mastering of the PCI bus.

For each channel one block of shared RAM and two mailboxes are required. The RAM block is organised as a circular buffer with the indices of the last words read and written stored in the mailboxes. Since the indices are likely to require 16bits or less, a pair of mailbox registers could be used for more than one channel (in the same direction). Separate mailboxes are used for the read pointer (RP) and the write pointer (WP) to avoid having to implement any locking. Only the sender writes to the WP mailbox and only the receiver writes to the RP mailbox.

As a general principle, a PMC card should not access any resources on the motherboard unless invited to do so. This implies that the mailboxes used should be in the RoB-in's bridge chip.

The most likely candidates for the bridge on the RoB-in are the PLX9060xx and the AMCC. The AMCC chip has eight 32bit mailbox registers, four writable from each side of the bridge. The PLX chip has four bidirectional 32bit mailbox registers.

```
Message direction --->
```

```
              Motherboard                           |   PMC
        +-----------------------+                   |
        | message(n)    word(0) |                   |
        | message(n)    word(1) |                   | +----+
        |            .          |  readonly <-- | RP | <-- writeonly
        |            .          |                   | +----+
        |            .          | writeonly --> | WP | --> readonly
        | message(n)    word(m) |                   | +----+
RP -->  | message(n+1)  word(0) |                   |
        |            .          |                   |
        |            .          |                   |
        |            .          |                   |
WP -->  | message(n+1)  word(m) |                   |
                        Message direction <---
              Motherboard                           |   PMC
        +-----------------------+                   |
        | message(n)    word(0) |                   |
        | message(n)    word(1) |                   | +----+
        |            .          | writeonly --> | RP | --> readonly
        |            .          |                   | +----+
        |            .          |  readonly <-- | WP | <-- writeonly
        | message(n)    word(m) |                   | +----+
RP -->  | message(n+1)  word(0) |                   |
        |            .          |                   |
        |            .          |                   |
        |            .          |                   |
WP -->  | message(n+1)  word(m) |                   |
Sender can write when
   (WP > RP)  AND (((WP + message\_length) modulo buffer\_size) < RP)
  OR (WP < RP) AND (WP + message\_length < RP)
Assuming message\_length < buffer\_size of course!
Receiver reads whenever WP <> RP
```

B.2 Sending messages

The sender can write a message, as long as there is room in the buffer, and store the index of the last word written in the write mailbox. In the case of the motherboard sending to the PMC this requires two accesses to the PCI bus, one to read RP and one to write WP (although it may only be necessary to read RP if the value last read would indicate insufficient space). In the case of the PMC sending to the motherboard access to the mailboxes is local but the data block itself is written across the PCI bus in a block transfer.

B.3 Receiving messages

The receiver can detect whether there is anything to read by comparing the contents of RP (or its own knowledge of what it last wrote to RP) with the contents of WP in a polling loop or it could be prompted to read the mailbox by "ringing" a doorbell register. In the case of the PMC, polling the mailboxes would not involve any access to the PCI bus as they would appear on the local side of the bridge. If the motherboard processor were to poll, it would involve PCI bus access.

Having calculated how much data is in the FIFO, the receiver can read all the outstanding messages in one block transfer (or two if the wrap point in the circular buffer is spanned). Alternatively, it may transfer just a single message in one (or two) block transfers (by reading the message length of the message from within the message). Another alternative option is to use fixed length messages and to pass the index of the last message slot used in the mailbox registers.

B.4 Data flowing to and from the RoB-in

```
           RoB controller                       RoB-in
           ---------->                        <---------
RoIR   ~10kHz    ~32bytes 320kB/s  |  RoID  ~10khz    ~1kbyte 10MB/s
L3R     ~1kHz    ~16bytes  16kB/s  |  L3D    ~1khz    ~1kbyte  1MB/S
DR      ~5kHz  ~160bytes 800kB/s   |
MonR    ~1kHz?  ~16bytes  16kB/s   |  MonD    1kHz?  ~1kbyte  1MB/s
Control ~1Hz?    ~8bytes    8B/s   |
L3done ~1kHz     ~8bytes   8kB/s   |
                                   |  ErrStat  1Hz?  ~2bytes   8B/s
      ======            =======           ======            ======
      ~18kHz            1.16MB/s          ~12kHz            12MB/s
Total bandwidth  ~13MB/s
```

The RoIR/RoID, L3R/L3D and MonR/MonD records are definitely request/response pairs therefore the request could specify where the data should be written and bypass this protocol for the output (the output address could even be in the RoB-out rather than the RoB controller).

The data requests (RoIR, L3R & MonR) and decision records could all be combined on one channel. Control requests should have a dedicated channel.

B.5 PCI bus accesses

```
Motherboard             ---->       PMC
```

```
      read RP                  |     read data
      write WP                 |
      ----------------------   |     ----------------
      2 single word accesses   |     1 block transfer
     Motherboard             <----      PMC
      write RP                 |     write data
      read WP                  |
      ----------------------   |     ----------------
      2 single word accesses   |     1 block transfer
3 accesses * 18kHz  = 54kHz    |  3 accesses * 12kHz  = 36kHz
Total PCI bus access rate = 87kHz
```