

BaBar DAQ Controller Card Description

Version Control:	
	Uncontrolled copy. This means that this copy of the document may not be the latest version. You can download the latest version via the WWW from the BaBar Home Page-> Electronics->Common Electronics->Read-Out Module.
	Controlled copy issued to:

Author:	john.dowdell@rl.ac.uk		
File Name:	JohnD HD:ROM CC:Description:CC Description		
First Modified:	12-Jan-97	Last Modified:	26-Jun-97

CONTENTS

1. REVISION HISTORY.....	3
2. INTRODUCTION.....	4
3. HOW THE QUEUES WORK	6
3.1 ORGANISATION OF THE QUEUE MEMORY	6
3.2 THE MERGE UNIT	6
3.3 THE TRANSFER UNIT.....	6
3.4 THE DMA UNIT.....	7
4. HOW THE QUEUES ARE IMPLEMENTED.....	8
4.1 CONTENTS OF BUFFERS IN THE MUQ.....	10
4.2 CONTENTS OF THE BUFFERS IN THE TUQ.....	10
4.3 GENERATING INTERRUPTS ON THE I960 BUS	11
5. I960 BUS REGISTER MAP.....	13
5.1 SERIAL NUMBER REGISTER (SERNO).....	13
5.2 CONTROL REGISTER (CTRL).....	14
5.3 STATUS REGISTER (STATUS).....	14
5.4 QUEUE STATUS REGISTER (QSTATUS).....	16
5.5 INJECT FCTS REGISTER (INJFCTS).....	16
5.6 INJECT RUN-TIME REGISTER (INJRUNTIME).....	16
5.7 FRONT-END WRITE REGISTER (FEWRITE).....	17
5.8 FRONT-END READ REGISTER (FEREAD).....	17
5.9 PUSH MUQ REGISTER (PUSHMUQ).....	17
5.10 PUSH TUQ REGISTER (PUSHTUQ).....	18
5.11 POP DUQ REGISTER (POPDUQ).....	18
5.12 WRITE MEMORY.....	18
5.13 FCTS SIMULATOR MEMORY.....	18
5.14 QUEUE MEMORY	18
6. LATENCY THROUGH THE CC.....	18
7. TIMING SPECIFICATIONS	19
8. FLOORPLAN.....	21
9. J3 PIN-OUT.....	22
10. PC CONNECTOR PIN-OUT.....	24
10.1 PIN-OUT OF THE BUS CONNECTOR	24
10.2 PIN-OUT OF THE CONTROL CONNECTOR.....	26
11. FRONT-PANEL INDICATORS	28

1. REVISION HISTORY

Date Issued	Description of Revisions
2-Feb-97	<ul style="list-style-type: none"> • Changed to two C/DLINKs per PC (also double for EMC PC) • Corrections to the memory map
4-Feb-97	<ul style="list-style-type: none"> • Allow 64 words per buffer in the queues.
6-Feb-97	<ul style="list-style-type: none"> • Compressed the data in the queues.
6-Feb-97	<ul style="list-style-type: none"> • Added support for 2 C/D-LINKS
27-Feb-97	<ul style="list-style-type: none"> • Add more description of queue operation • Estimate latency through CC • Add timing specifications
2-Mar-97	<ul style="list-style-type: none"> • Clarified naming of the result bus
13-Mar-97	<ul style="list-style-type: none"> • Changed labelling of C/D-LINKs to A and B (from 1 and 2) • Changed base address on I960 bus to bits <31..28> only
13-Mar-97	<ul style="list-style-type: none"> • Increased number of CALPM bits to 52
23-Mar-97	<ul style="list-style-type: none"> • Time to do burst four word access increased.
02-Apr-97	<ul style="list-style-type: none"> • Registers to advance MUPTR and TUPTR added • In FEWRITE register: WRLAST becomes LENGTH (i.e. number of bytes to transfer, not the offset of the last byte to transfer).
04-Apr-97	<ul style="list-style-type: none"> • In timing diagrams: PCDATA is tri-state while PC_RD# is '1'. • Fix depiction of RXREAD/RXDATA and DONE in diagrams
26-Jun-97	<ul style="list-style-type: none"> • FULL does not stop messages going to the FEE • Format of messages in the queues is now DWORD aligned • Bits 15:8 are undefined when reading from WRMEM and SIMMEM • "SPARE" bit added from connector to i960 interface • LEDs on PC need resistor on PC

2. INTRODUCTION

This document describes the detailed design of the Controller Card (CC). It extends the 'Read-Out Module Architecture' description, which describes the functionality of the CC. The implementation described in this document is significantly different from that described in the architecture document. If there is a conflict between the two documents, believe this one.

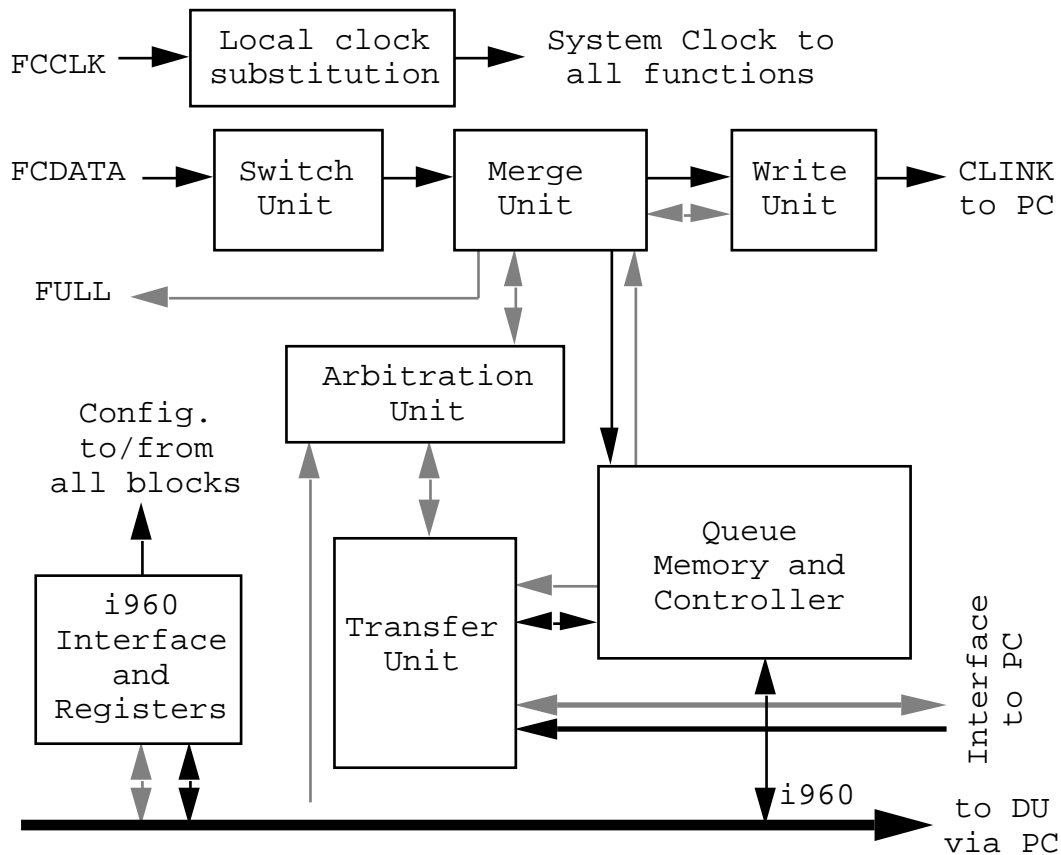


Figure 1 — Controller Card Block Diagram

The block diagram of the card is shown in Figure 1.

The CC has two functional 'units' that control the flow of data into the front-end electronics (FEE) and through to the memory in the personality card.

1. The Merge Unit (MU).
2. The Transfer Unit (TU).

The DMA Unit (DU), which controls the flow of data into the CPU memory, is implemented on the PMC card.

Data flows between these three units in logical 'queues': between the MU and TU in the MU queue (MUQ) and between the TU and DU in the TU queue (TUQ).

Other units on the CC are responsible for support functions:

1. The Switch Unit (SU) either passes FCTS messages straight through or inserts messages from a local simulator ram (SIMRAM) onto the CLINK. This SIMRAM is 4k bytes deep so a sequence length of 68 μ s can be simulated.
2. The Arbitration Unit (AU) accepts requests to locally insert messages from the TU and from the processor bus (either dataless or register read messages) and routes them to the MU for safe insertion.
3. The Write Unit (WU) inserts register write messages onto the CLINK when instructed via the bus. The WU is situated after the MU so that the write can be inserted without colliding with a message already on the CLINK and FCTS (and local) messages that are received while the write is in progress can be written into the MUQ and marked as unsent.
4. The queue controller manages the queue pointers and generates full/empty flags for the MU, TU and DU.

The CC now accommodates two CLINKs and DLINKs on the PC. There are therefore two sets of link status signals from PC to CC. The CC uses the OR of the two CLINK ready signals to determine if the CLINK is ready to receive data. The CC assumes that if either or both of the links needs to be disabled then this is done on the PC, and the link status signals gated to a safe state (all links ready and locked, no error).

3. HOW THE QUEUES WORK

3.1 Organisation of the queue memory

The queue memory is divided into thirty-two equal length containers, each consists of sixty-four, sixteen bit words. There are three 'pointers':

- `muptr` determines into which container the MU will write the next message. It follows that if the MU is writing into the queue memory then it is writing into the container given by `muqptr`.
- `tuptr` determines which container is being or will next be processed by the TU.
- `duptr` determines which container is being or will next be processed by the DU.

Upon reset all three pointers are set to zero.

It is possible to think of these containers as on a conveyor belt with the MU filling containers at the start of the belt; the TU inspecting and refilling containers somewhere in the middle and the DU emptying containers at the end, before returning them to the MU empty.

3.2 The Merge Unit

If all messages were L1Accepts, then the number of containers between the `muptr` and `tuptr` would correspond to the number of filled buffers in the FEE. Therefore, the MU raises FULL to the FCTS when the difference between the two pointers exceeds a pre-set threshold. This should veto further L1Accepts until FULL is deasserted. However, it is still possible to write messages into the queue until all containers are filled.

If not all messages are L1Accepts then FULL could be raised before the FEE buffers are actually full. However this is not a common occurrence during data-taking and so will function as well as a true buffer model under most circumstances, while eliminating the possibility that the number of L1Accepts in the model and in the MUQ could be different.

3.3 The Transfer Unit

The TU waits until there is a message for it to process, i.e. the `muptr` no longer equals the `tuptr`.

If all messages were L1Accepts (or front-end register reads) the difference between the `duptr` and the `tuptr` would correspond to the number of filled buffers in the intermediate store, therefore the TU can not process messages if the difference between the `duptr` and `tuptr` exceeds a pre-set limit. This ensures the IS does not overflow.

How the TU processes the messages depends on what sort of message it is and if it generated data at the FEE. To determine how to handle a message the TU inspects the first word in the container, this contains the information required:

- opcode of the message
- tag or address (in the case of L1Accepts or register reads respectively)
- whether the message was sent to the FEE

We will now go through the cases.

3.3.1 L1Accept with data at the FEE

In this case the TU waits for 1 μ s then asks the AU to insert a Read Event with the tag specified in the message. By waiting 1 μ s the TU ensures that the time between L1Accept and the Read Event

being sent to the FEE is greater than $2.2\mu\text{s}$. The AU will reply with `ack` or `nak` if the MU successfully or unsuccessfully inserts the message respectively.

If the MU replies with `nak`, the Read Event was not sent successfully so the transfer from FEE to IS will not take place. In this case the 'nak' bit is set in the transfer status word in the container and the `tuptr` is advanced, passing the container on to the DMA Unit.

If the MU replies with `ack` the Read Event was sent successfully, so the TU immediately asserts the `RX_DATA` signal to the PC. This tells the PC to expect event data on the DLINKs. When the data is received (or a time-out occurs) the PC asserts the `DONE` signal to the TU and supplies the status of the transfer on `STATUS[7:0]`. The TU sets the 'ack' bit and writes the status into the status word in the container.

The TU then uses the result bus (described elsewhere) to gather the result of the transfer from the PC and write it into the next words in the container. The number of words in the transfer result is variable, the PC indicates which is the last word. If the PC attempts to write more words than the container can handle the excess words will be discarded.

When this is finished, the `tuptr` is advanced, passing the container on to the DMA Unit.

3.3.2 Register Reads

In this case the TU asks the AU to insert a message with the opcode and address specified in the message. Processing is performed exactly the same way as for a Read Event, except that the `RX_READ` signal is asserted to the PC, instead of the `RX_DATA`.

3.3.3 Clear Readout

The Clear Readout mechanism is:

- When a Clear Readout message is written into the queue (from either the FCTS or locally) the throttle bit is set. This stops further messages being sent from the FCTS to the FEE.
- When the TU finds the Clear Readout message in its input, it asks the AU to inject a Clear Readout message to the FEE (and not to insert it into the queue!). When this is complete (successfully or unsuccessfully) the throttle bit is deasserted.

When this is finished, the `tuptr` is advanced, passing the container on to the DMA Unit.

3.3.4 Messages without data at the FEE

All other sorts of messages (for example: Synch or even L1Accepts that were not sent to the FEE for some reason) are not touched by the TU. When the TU identifies an event of this type it simply increments the `tuptr`.

3.4 The DMA Unit

The DU will process the message by copying the contents of the message into some form of descriptor in CPU main memory and, if there is data in the intermediate store, perform a DMA operation to get that data into main CPU memory. If the DU does not have space in main CPU memory to put the message then it will wait until it does.

The DMA Unit is in fact the i960 processor on the PMC. The interface to it is:

- The queue memory, which is randomly accessible from the i960 bus.
- One of the possible interrupt sources: 'DUQ not empty'. If this is enabled, the DU will be interrupted when the number of containers between the `tuptr` and `duptr` goes from zero to non-zero. The DU could also poll this bit rather than use interrupts, if desired.
- By reading one of the registers on the CC, the DU can find the value of `duptr`, and hence determine which container contains the next message for it to process. Because the containers

are all of fixed length, starting from a known base address, the DU then knows the start address on the i960 bus of the container.

- When the DU has processed a message and the DMA operation (if any) has finished the DU must access (read or write) the POPDUQ register, which advances the `duptr`. As well as freeing the container for use by the MU, the fact that there is one less message between the `duptr` and `tuptr` means frees up one buffer in the IS.

In addition to these registers, it is possible to read a register to find out the values of the three pointers and the number of containers between `muptr` and `tuptr` (the occupancy of the FEE buffers) and `tuptr` and `duptr` (the occupancy of the IS).

4. HOW THE QUEUES ARE IMPLEMENTED

The queues are actually implemented in a multi-port shared memory that is split into equal length containers. Each container can contain one message from the either FCTS or locally injected and the associated data added by the MU and TU. The entire contents of the memory are mapped onto the processor bus.

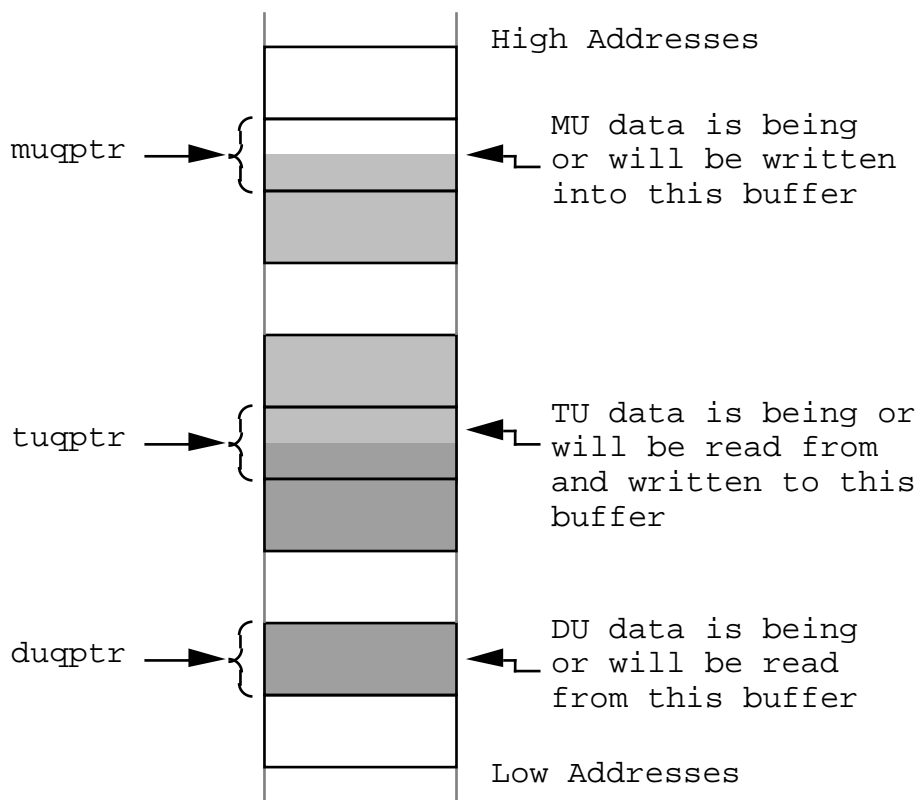


Figure 2 — Implementation of the Queues

A diagram of the arrangement of the memory is shown in Figure 2.

If no message is currently being received by the MU then the `muqptr` points to the buffer that the next message will be stored in. If a message is currently being received by the MU then the `muqptr` points to the buffer that is being written into.

In the case of an untriggered PC the sum of the number of buffers in the MUQ and TUQ is equal to the number of buffers in the intermediate store on the PC.

The `muqptr` and `tuqptr` are maintained and advanced in hardware by the respective unit. The `duqptr` is maintained in hardware but advanced by the DU (an i960 processor) by accessing a register on the bus. The contents of all three pointers can be read via the bus. The `muqptr` and `tuqptr` would be accessed for test purposes, whereas the `duqptr` would be used by the software to determine where in the queue the next buffer is.

Because the entire memory is mapped onto the processor bus and the queue pointers are visible it is possible at any time to inspect and modify the contents of any buffer for test purposes. To ensure this is useful it is possible to take the TU 'off-line' so the contents of the MUQ and TUQ remain static for long enough to be checked reliably.

The following table shows the format of data written into the queues by the MU and TU (a sixteen bit wide data path). From the i960 bus two words are read in parallel (a full 32 bit data path): word zero will be in the least significant bits, word one in the most significant and so on.

	15		8	7		0								
0	T4...T0		nclready		src[1:0]		C4...C0		daqgate		feegate		issent	
1	unused						status							
2	Creation Time [15:0]													
3	Creation Time [31:16]													
4	Creation Time [47:32]													
5	unused						Creation Time [55:48]							
6	Trigger List [15:0]													
7	Trigger List [31:16]													
8	ack		nak		0		Status word from PC [7:0]							
9	Transfer result from PC, word 0 [15:0]													
10	Transfer result from PC, word 1 [15:0]													
	...													
63	Transfer result from PC, word 54 [15:0]													

"Unused" means that those bits are not overwritten by the MU or TU.

The fields contain the following information:

- opcode (C4...C0)
- trigger tag (T4...T0)
- whether the message was destined for the MUQ (DAQ_GATE from the FCTS message)
- whether the message was destined for the front-end (FEE_GATE from the FCTS message)
- whether the message was sent to the front-end (issent, 1 = message was sent)
- whether the CLINK was ready when the message was received and about to be sent down the CLINK (clready, 1 = ready)
- the source of the message is in bits src[1,0]

The status word is made up of the following flags:

7							0
throttle	!enclink	writing	ncldreadya	ncldreadyb	FULL	PCFULL	ncldready

- whether the CLINK was throttled when the message was received (throttle, 1 = link throttled)
- whether the CLINK was disabled when the message was received (!enclink, 1 = CLINK disabled)
- whether the CLINK was being used by a register write when the message was received (writing, 1 = CLINK busy with register write)
- ncldreadya and ncldreadyb in word one indicates whether the CLINKs were ready while the header was sent down the CLINK (1 = not ready).
- whether the FULL signal was asserted when the message was received
- whether the PCFULL signal was asserted when the message was received

4.1 Contents of Buffers in the MUQ

The contents of a buffer in the MUQ will differ depending on whether the message came from the FCTS or local injection.

4.1.1 Case 1: FCTS messages

The first eight words in the buffer are filled.

Note that for a successfully transmitted FCTS message all bits in the status byte (except PC_FULL) should be zero.

4.1.2 Case 2: Locally Inserted Messages

Due to timing constraints derived from the fact that insertion must be safe into both the CLINK and MUQ, only the first word in the buffer is written, hence there is no status, time or trigger information. The first word contains the same information as an FCTS message with the exception of the source of the message (src[1,0]). This determines what the message is by where it came from:

- [0,1] dataless message
- [1,0] register read
- [1,1] transfer unit

4.2 Contents of the Buffers in the TUQ

The TU decides from the contents of the first word in the MUQ buffer what action to take.

4.2.1 LIAccept Messages

If the opcode of the message is LIAccept (regardless of the source) and `issent` is true, then the TU asks the MU via the AU to insert a Read Event message onto the CLINK with the tag specified in the buffer.

The MU indicates success or failure using two signals: `ack` and `nak`, the state of these will be stored in the TUQ buffer. If `nak` is received the TU performs no further processing. If `ack` is received the TU asserts `RX_DATA` to the PC and waits for `done` to be asserted. When `done` is received the status bits are written into the buffer in the TUQ and the transfer result is copied from the PC to the TUQ.

The words written by the MU are unchanged. Word eight shows the status of both phases of the operation: `ack`, `nak` from the MU (active high) and `status[7...0]` from the PC.

The rest of the words in the buffer are filled in sequence with the transfer result from the PC. The number of words written is variable. The transfer results are communicated from the PC using the following signals:

- `PCDATA[15...0]` Sixteen bits of data from PC to CC
- `pc_rd#` Active low tri-state enable for PCDATA (from CC to PC).
- `pc_end` Active high end of data flag (from PC to CC).

An example of an interaction is shown in the following diagram:

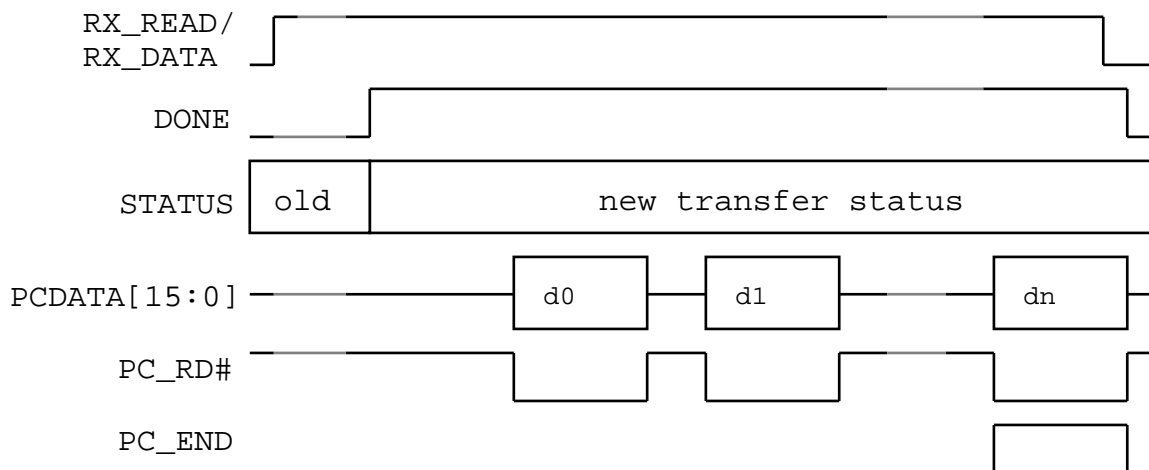


Figure 3 — Transfer Result Bus

If `pc_end` is asserted with `done` it means that there is no data to transfer from the PC.

The number of words to transfer is variable from zero to fifty-five using the `pc_end` signal. The TU does not know what is in the transfer result, it merely copies data from PC to TUQ.

4.2.2 Register Read Messages

These work in the same way as `L1Accept` messages except that `RX_READ` is asserted not `RX_DATA`, and the opcode inserted onto the `CLINK` is the opcode in the `MUQ` message. The format of data in the TUQ is the same.

4.2.3 Messages With No Front-End Data

If the message is neither of the above then the TU does nothing to it, simply pushing it onto the TUQ. The contents of the TUQ buffer will therefore be the same as for the `MUQ` buffer.

4.3 Generating Interrupts on the i960 Bus

The CC can interrupt the DU to inform it of a change in condition, for example: 'there are buffers in the TUQ awaiting processing'.

The CC has a thirty-two bit register that contains bits that describe its status. The most-significant ten of these can be used as interrupt sources. These are special bits that are known as latched bits: when the signal being latched makes a positive transition the latch bit becomes '1'. The latch bit is reset to '0' by writing '1' to the corresponding bit in the status register.

These bits (active high) are bit-wise ANDed with an 'interrupt mask' (held in the control register) and if any of the bits in the result are non-zero the interrupt signal on the i960 bus is asserted.

There are eight possible interrupt lines on the i960 bus; which is driven by the CC is determined by bits in the control register.

To find out which interrupt source caused the interrupt the status register can be read. Any or all interrupt sources can be enabled using the mask register and any or all can be reset by writing into the status register.

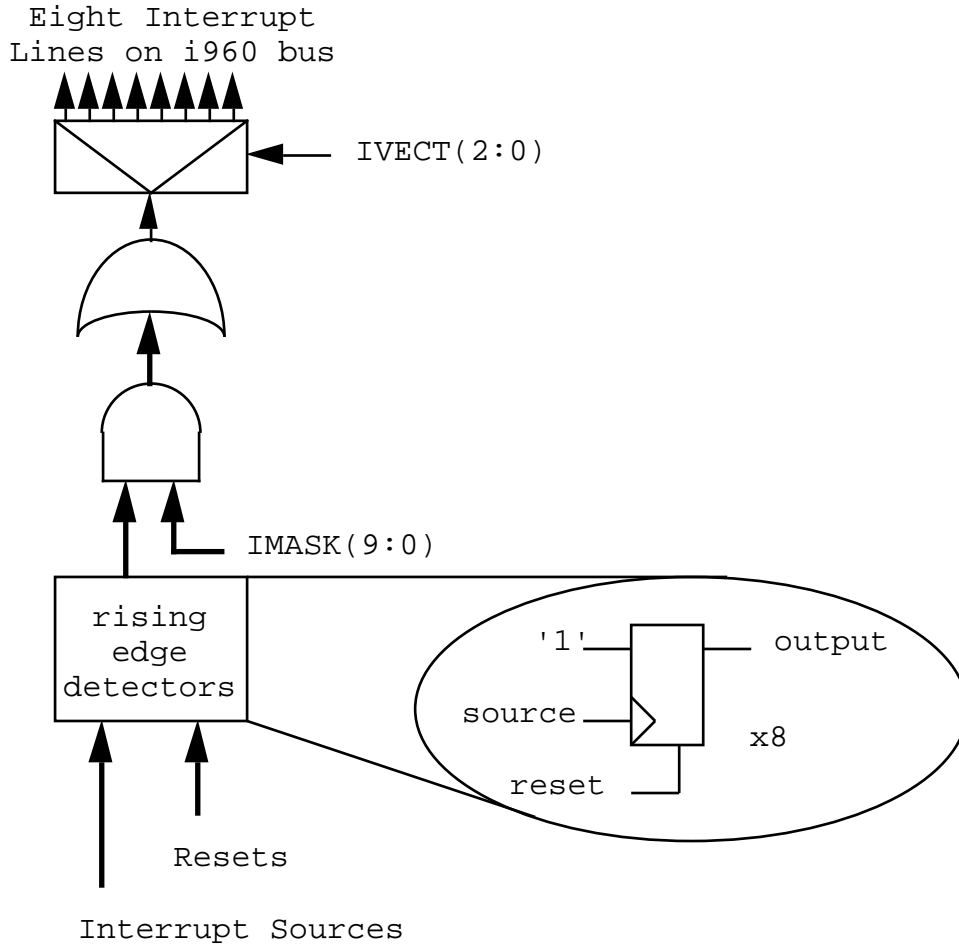


Figure 4 — Interrupt Generation

5. 1960 BUS REGISTER MAP

The following table shows the register map. Registers are located at an offset from a base address, the four most significant (31:28) of which are set by a DIP switch on the CC. The module will only respond correctly to bus accesses that assume a thirty-two bit wide bus, although the byte enables will be used to provide byte-wide addressing.

All areas of memory are accessible by infinite length burst transfers.

Offset from Base Address (hex)	Name	Description
0	SERNO	Serial number register.
4	CTRL	Control register
8	STATUS	Status register
C	QSTATUS	Number of messages in each queue and instantaneous values of the pointers
10	INJFCTS	Inject a simulated FCTS message
14	INJRUNTIME	Inject a run-time message
18	FEWRITE	Send a FEE register write message
1C	FEREAD	Inject a FEE register read message
20	PUSHMUQ	Advance the <code>muqptr</code>
24	PUSHTUQ	Advance the <code>tuqptr</code>
28	POPDUQ	Advance the <code>duqptr</code>
2C–FFF	—	Undefined
1000–1FFF	QUEUES	The queue memory
2000–3FFF	—	Undefined
8000–BFFF	WRMEM	The write payload memory
C000–FFFF	SIMMEM	The simulated FCTS message

The registers/memories are discussed in more detail in the next sections. The tables show the bit assignments, whether they are readable or writeable and what they are reset to.

5.1 Serial Number Register (SERNO)

Bit	Name	Read	Write	Reset to	Description
31:16	Unused			0	Unused, read as zero.
15:11	GA[4:0]			unaffected	Geographical address of slot, read from backplane.
10:0	SERNO[10:0]			unaffected	Serial number of the module. Set by 'program headers'.

5.2 Control Register (CTRL)

Bit	Name	Read	Write	Reset to	Description
31	ENFCDATA			0	1 = enable receiving data from FCDM
30	ENCLINK			0	1 = enable sending data down the CLINK
29	ENTU			0	1 = enable processing by the TU
28:26	IVECT			0	Which i960 interrupt to generate interrupts on
25:16	IMASK			0	Which interrupt sources to enable.
15	TEST			unaffected	Drives the test point on the front-panel.
14:13	—			unaffected	Unused.
12:8	TUQNBUF			unaffected	Number of buffers that can be filled in the TUQ before the TU stops processing the MUQ.
7:5	—			unaffected	Unused.
4:0	MUQNBUF			unaffected	Number of buffers that can be filled in the MUQ before the MU asserts FULL to the FCTS.

5.3 Status Register (STATUS)

Bit	Name	Read	Write	Reset to	Description
31	TUQNOTE_LAT			0	Latched version of TUQ not empty. Active high, write '1' to this bit to reset latch.
30	WRDONE_LAT			0	Latched version of write done. Active high, write '1' to this bit to reset latch.
29	DLNOTRDYA_LAT			0	Latched version of DLINK not ready from link A. Active high, write '1' to this bit to reset latch.
28	CLNOTLKDA_LAT			0	Latched version of CLINK not locked from link A. Active high, write '1' to this bit to reset latch.
27	CLNOTRDYA_LAT			0	Latched version of CLINK not ready from link A. Active high, write '1' to this bit to reset latch.
26	DLERRA_LAT			0	Latched version of DLINK error from link A. Active high, write '1' to this bit to reset latch.
25	DLNOTRDYB_LAT			0	Latched version of DLINK not ready from link B. Active high, write '1' to this bit to reset latch.
24	CLNOTLKDB_LAT			0	Latched version of CLINK not

			locked from link 2. Active high, write 'B' to this bit to reset latch.
23	CLNOTRDYB_LAT	0	Latched version of CLINK not ready from link B. Active high, write '1' to this bit to reset latch.
22	DLERRB_LAT	0	Latched version of DLINK error from link B. Active high, write '1' to this bit to reset latch.
21	DLNOTRDYA	unaffected	Instantaneous value of DLINK not ready A.
20	CLNOTLKDA	unaffected	Instantaneous value of CLINK not locked A.
19	CLNOTRDYA	unaffected	Instantaneous value of CLINK not ready A.
18	DLERRA	unaffected	Instantaneous value of DLINK error A.
17	DLNOTRDYB	unaffected	Instantaneous value of DLINK not ready B.
16	CLNOTLKDB	unaffected	Instantaneous value of CLINK not locked B.
15	CLNOTRDYB	unaffected	Instantaneous value of CLINK not ready B.
14	DLERRB	unaffected	Instantaneous value of DLINK error B.
13	FULL	unaffected	Instantaneous value of FULL.
12	PCFULL	unaffected	Instantaneous value of PCFULL.
11	LOCALCLK	unaffected	0 = FCTS clock driving CC 1 = local clock driving CC
10	WROK	unaffected	1 = The previous FEE register write completed OK.
9	FE_WRITING	unaffected	1 = The CLINK is currently being used to write to an FEE register.
8	SIMFCTS	0	1 = SU is currently sending simulated FCTS messages from the SIMRAM.
7	THROTTLE	unaffected	1 = The CLINK is currently throttled
6	MUQE	unaffected	1 = The MUQ has no full buffers
5	MUQBF	unaffected	1 = The number of full buffers in the MUQ equals or exceeds the number of buffers specified in the control register
4	MUQF	unaffected	1 = The MUQ is full, no more messages can be accommodated.
3	TUQE	unaffected	1 = The TUQ is empty (there are no

Bit	Name	Read	Write	Reset to	Description
2	TUQBF		unaffected	1	The number of full buffers in the TUQ equals or exceeds the number of buffers specified in the control register
1:0	—		0		Unused, read as zero.

Where ‘unaffected’ means that the value in the register keeps reflecting the state of the driving signal. The reset may affect the source of the signal of course.

5.4 Queue Status Register (QSTATUS)

Bit	Name	Read	Write	Reset to	Description
31:29	—			0	Unused, read as zero.
28:24	TUQBUIFS			0	The number of filled buffers in the TUQ
23:21	—			0	Unused, read as zero.
20:16	MUQBUIFS			0	The number of filled buffers in the MUQ
15	—			0	Unused, read as zero.
14:10	MUQPTR			0	The instantaneous value of the <code>muqptr</code>
9:5	TUQPTR			0	The instantaneous value of the <code>tuqptr</code>
4:0	DUQPTR			0	The instantaneous value of the <code>duqptr</code>

Note that the number of filled buffers is derived from the difference between the appropriate pointers, not a model.

5.5 Inject FCTS Register (INJECTS)

Bit	Name	Read	Write	Reset to	Description
31:0	—			0	Unused, read as zero.

Writing to this register causes the SU to serialise and inject the contents of the SIMMEM as if it came from the FCTS. This is only allowed if ENFCDATA is ‘0’ and LOCALCLK is ‘1’, otherwise the request is ignored. The whole contents of the SIMMEM (4k bytes) is serialised each time. The SIMMEM is 4k bytes long so combinations of messages up to 68 micro-seconds long can be produced.

5.6 Inject Run-Time Register (INJRUNTIME)

Bit	Name	Read	Write	Reset to	Description
31:12	—			0	Unused, read as zero.
11:7	TAG			unaffected	Tag in inject in the message
6:2	OPCODE			unaffected	Opcode to inject in the message
1	DAQ_GATE			unaffected	DAQ_GATE to inject in the message
0	FEE_GATE			unaffected	FEE_GATE to inject in the message

Writing to this register causes the AU to inject a run-time message (i.e. no following data). The opcode, tag and gates to use are taken from the register. Reading from this register returns its contents without injecting a new message.

Note that the same physical register is used to hold the header information for the INJRUNTIME and FERREAD, so only one of these requests can be outstanding at any given time.

5.7 Front-End Write Register (FEWRITE)

Bit	Name	Read	Write	Reset to	Description
31:28	—			0	Unused, read as zero.
27:16	LENGTH			unaffected	The number of bytes in the payload. (Note: '0' means 4k, not zero k)
15:12	—			0	Unused, read as zero
11:7	Address			unaffected	Address to send in header
6:2	Opcode			unaffected	Opcode to send in header
1	—			unaffected	unused
0	—			unaffected	unused

Writing to this register causes the WU to generate a register write message to the FEE. It firstly gains control of the CLINK, then serialises the opcode and address and finally the first LENGTH bytes from the write memory (WRMEM). Note that if LENGTH is set to zero the whole WRMEM will be sent (4k).

Reading from this register returns the contents of the register and does not start writing. The status of the operation can be determined through the STATUS register and interrupts.

5.8 Front-End Read Register (FEREAD)

Bit	Name	Read	Write	Reset to	Description
31:12	—			0	Unused, read as zero
11:7	Address			unaffected	Address to send in header
6:2	Opcode			unaffected	Opcode to send in header
1	—			unaffected	unused
0	—			unaffected	unused

Writing to this register causes the AU to insert a front-end read message as soon as the CLINK is free. The opcode and address in the message come from the contents of this register. Reading from this register returns its contents and does not cause the message to be injected. FEE_GATE is assumed to be '0' and DAQ_GATE '1'.

Note that the same physical register is used to hold the header information for the INJRUNTIME and FERREAD, so only one of these requests can be outstanding at any given time.

5.9 Push MUQ Register (PUSHMUQ)

Bit	Name	Read	Write	Reset to	Description
31:0	—			0	Unused, read as zero.

Accessing this register advances the `muqptr`, if `ENTU` is '0' (otherwise it does nothing). If the MUQ is totally full then the pointer is not advanced. This is a test feature only.

5.10 Push TUQ Register (PUSHTUQ)

Bit	Name	Read	Write	Reset to	Description
31:0	—			0	Unused, read as zero.

Accessing this register advances the `tuqptr`, if `ENTU` is '0' (otherwise it does nothing). If the TUQ buffers are full then the pointer is not advanced. This is a test feature only.

5.11 Pop DUQ Register (POPDUQ)

Bit	Name	Read	Write	Reset to	Description
31:0	—			0	Unused, read as zero.

Accessing this register advances the `duqptr`. If the DUQ is empty then the pointer is not advanced.

5.12 Write Memory

This 4k bytes of memory is directly mapped into the thirty-two bit i960 address space, but it is implemented in a single eight bit device. Each byte of the memory appears in bits seven to zero on the i960 bus, with the high order bits unused (bits 31:16 read as zero, bits 15:8 are undefined). Therefore, on the i960 with respect to the start of the memory, byte zero is at offset zero, byte one is at offset four etc.

5.13 FCTS Simulator Memory

This 4k bytes of memory is directly mapped into the thirty-two bit i960 address space, but it is implemented in a single eight bit device. Each byte of the memory appears in bits seven to zero on the i960 bus, with the high order bits unused (bits 31:16 read as zero, bits 15:8 are undefined). Therefore, on the i960 with respect to the start of the memory, byte zero is at offset zero, byte one is at offset four etc.

5.14 Queue Memory

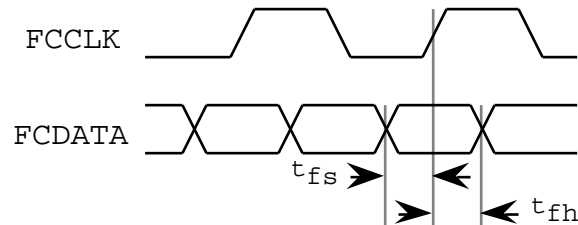
This 4k bytes of memory is directly mapped into the i960 address space. The format of data in the queues shown previously is for sixteen bit accesses from the units. From the i960 the queues are thirty-two bits wide, so two words are accessed at once.

6. LATENCY THROUGH THE CC

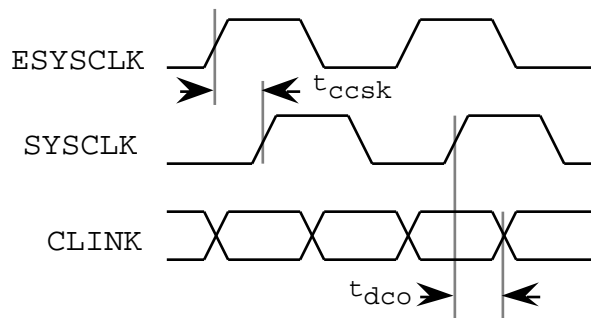
Parameter	Value
Delay from start of message to start of message to PC	300ns
Delay from start of message to assertion of FULL	300ns
Delay from start of message to MU container full	2.2μs
Time to fall through the TU if not an L1Accept	200ns
Time for TU to service L1Accept, and ...	2.0μs
... extra time per word in the transfer result	67ns
Time to do a four double-word burst access to the CC	500ns

So for L1Accepts (ignoring the PC and DU) with 40 words in the transfer result the maximum rate would be: $2.2 + 2.0 + 40 \cdot 67 = 6.9 \mu\text{s}$, i.e. 145 kHz. For dataflow messages the maximum rate (ignoring the DU) would be: $2.2 + 0.2 = 2.4 \mu\text{s}$, i.e. 400 kHz.

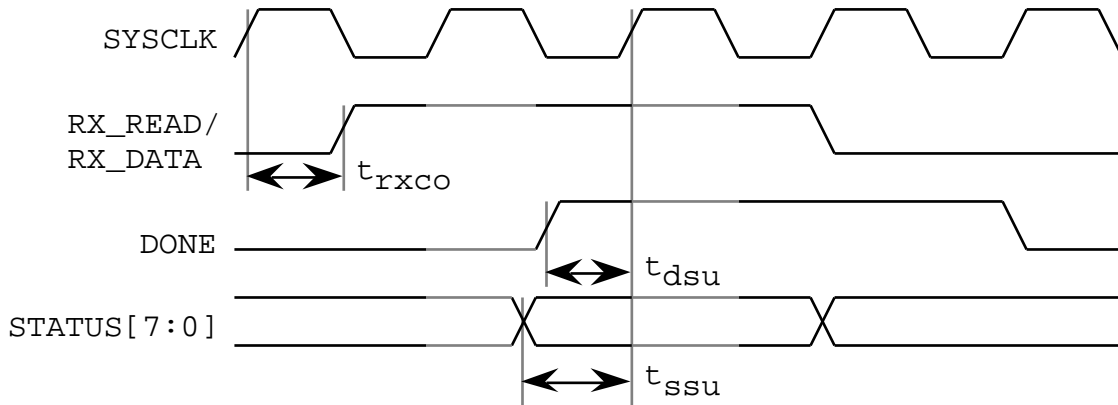
7. TIMING SPECIFICATIONS



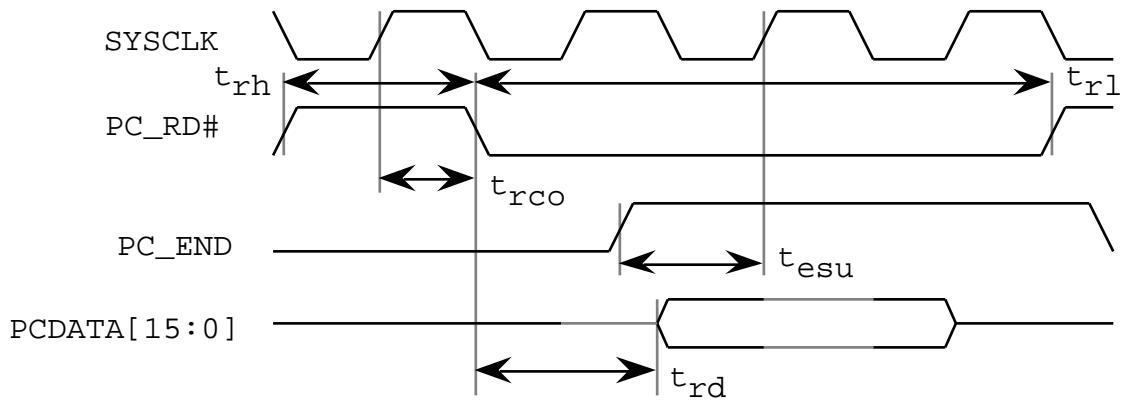
	Description	min (ps)	max (ps)
t_{fs}	FCDATA data set-up time	490	
t_{fh}	FCDATA data hold time	0	



	Description	min (ns)	max (ns)
t_{ccsk}	Skew ECL clock to TTL clock	1.7	4.2
t_{dco}	SYSCLK to CLINK data	2	8



	Description	min (ns)	max (ns)
t_{rxco}	SYSCLK to RXDATA/RXREAD	2	8
t_{dsu}	DONE to SYSCLK setup time	8	
t_{ssu}	STATUS to SYSCLK setup time	0	



	Description	min (ns)	max (ns)
t_{rco}	SYSCLK to PC_RD# asserted/deasserted	2	8
t_{rh}	PC_RD# high	1 cycle	
t_{rl}	PC_RD# low		3 cycles
t_{esu}	PC_END set-up time	8	
t_{rd}	PC_RD to data delay		5

8. FLOORPLAN

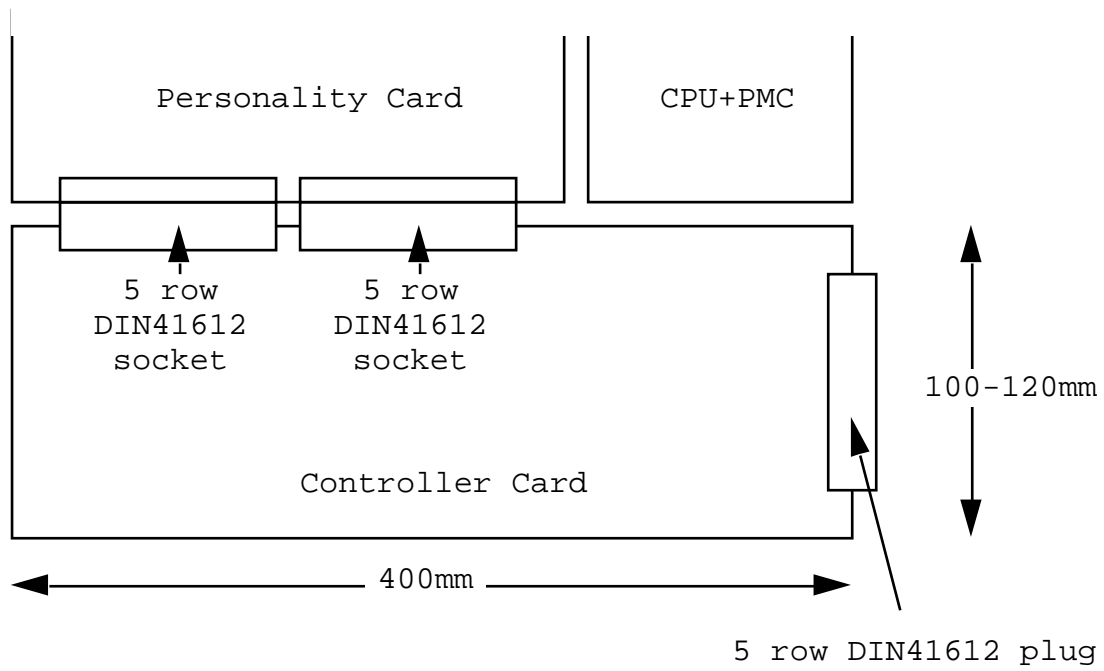


Figure 5 — CC Floorplan

Figure 5 shows the size of the controller card and the approximate position of connectors and the relationship with the other boards in the ROM.

9. J3 PIN-OUT

The J3 connector to the backplane is a five row DIN41612 plug, albeit probably with keying to ensure non-ROMs cannot be put in ROM slots and visa versa.

	A	B	C	D	E
1	GND	GND	+5V	+5V	+5V
2	CALPM0	CALPM1	GND	GND	GND
3	CALPM2	CALPM3	+5V	+5V	+5V
4	CALPM4	CALPM5	GND	GND	GND
5	CALPM6	CALPM7	+5V	+5V	+5V
6	CALPM8	CALPM9	GND	GND	GND
7	CALPM10	CALPM11	+5V	+5V	+5V
8	CALPM12	CALPM13	GND	GND	GND
9	CALPM14	CALPM15	+5V	+5V	+5V
10	CALPM16	CALPM17	GND	GND	GND
11	CALPM18	CALPM19	+5V	+5V	+5V
12	CALPM20	CALPM21	GND	GND	GND
13	CALPM22	CALPM23	GND	GA0	
14	CALPM24	CALPM25	GND	GA1	
15	CALPM26	CALPM27	GND	GA2	
16	CALPM28	CALPM29	GND	GA3	
17	CALPM30	CALPM31	GND	GA4	
18	CALPM32	CALPM33	GND	GND	
19	CALPM34	CALPM35	GND	FULL	
20	CALPM36	CALPM37	GND	GND	
21	CALPM38	CALPM39	EGND	EGND	EGND
22	CALPM40	CALPM41	EGND	FCCLK+	FCCLK-
23	CALPM42	CALPM43	EGND	EGND	EGND
24	CALPM44	CALPM45	EGND	FCDATA+	FCDATA-
25	CALPM46	CALPM47	EGND	EGND	EGND
26	CALPM48	CALPM49	-5.2V	-5.2V	-5.2V
27	CALPM50	CALPM51	EGND	EGND	EGND
28	CALPM52	CALPM53	-5.2V	-5.2V	-5.2V
29	CALPM54	CALPM55	EGND	EGND	EGND
30	CALPM56	CALPM57	-5.2V	-5.2V	-5.2V
31	CALPM58	CALPM59	GND	GND	GND
32	GND	GND	+12V	+12V	+12V

Where the signals have the following meanings:

Signal Name	Logic Level	Input/Output to/from CC	Description
+12V	+12V	power	Connected to PC for powering Finisars
+5V	+5V	power	TTL/CMOS power for the CC/PC
GND	0V	power	TTL/CMOS 0V for the CC/PC
EGND	0V	power	ECL 0V for the CC/PC
-5.2V	-5.2V	power	ECL power for the CC/PC
FCCLK±	ECL	input	Fast control clock
FCDATA±	ECL	input	Fast control data
FULL	TTL	output	Buffer full flag
GA[3:0]	TTL	input	Geographical address of slot in the backplane. Pulled up with a resistor on the CC so default is '1'. Generate a '0' by connecting to ground on the backplane.
CALPM[59:0]	TTL	input/output	Connected directly to PC for interconnect. Arranged down one side of the connector to allow a simple cable connection.

10. PC CONNECTOR PIN-OUT

There are two DIN41612 connectors between the CC and PC. One carries mainly i960 bus signals from the PMC so that if the PC is not available for testing the PMC cable can be plugged directly into one of the CC connectors.

10.1 Pin-out of the Bus Connector

	A	B	C	D	E
1	AD0	AD1	GND	INT0	GND
2	AD2	AD3	GND	GND	INT1
3	AD4	AD5	GND	INT2	GND
4	AD6	AD7	GND	GND	INT3
5	AD8	AD9	GND	INT4	GND
6	AD10	AD11	GND	GND	INT5
7	AD12	AD13	GND	INT6	GND
8	AD14	AD15	GND	GND	INT7
9	AD16	AD17	GND	NMI#	GND
10	AD18	AD19	GND	GND	HOLD
11	AD20	AD21	GND	HOLDA	GND
12	AD22	AD23	GND	GND	ALE
13	AD24	AD25	GND	ADS#	GND
14	AD26	AD27	GND	GND	RDYRCV#
15	AD28	AD29	GND	DEN#	GND
16	AD30	AD31	GND	GND	W/R#
17		GND	GND	DT/R#	GND
18		CLK	GND	GND	GND
19		GND	GND	SPARE	
20		BLAST#	GND		
21		GND	GND	-5.2V	-5.2V
22		RST#	GND	GND	GND
23	GND	GND	GND	-5.2V	-5.2V
24	WIDTH0	WIDTH1	GND	GND	GND
25	GND	GND	GND	-5.2V	-5.2V
26	BE0#	BE1#	GND	GND	GND
27	BE2#	BE3#	GND	-5.2V	-5.2V
28	GND	GND	GND	GND	GND
29	MISSCLK+	MISSCLK-	GND	-5.2V	-5.2V
30	INTCLK+	INTCLK-	GND	GND	GND
31	CMD+	CMD-	GND	+12V	+12V
32	FULL+	FULL-	GND	GND	GND

Where the signals have the following meanings:

Signal Name	Logic Level	Input/Output to/from CC	Description
AD (31:0)	TTL	input/output	Multiplexed address and data bus
INT (7:0)	TTL	output	High generates interrupt
NMI#	TTL	output	Non-Maskable interrupt (unused)
HOLD	TTL	output	Request control of bus (unused)
HOLDA	TTL	input	Control of bus granted (unused)
WIDTH (1:0)	TTL	input	Indicates width of bus cycle
BE# (3:0)	TTL	input	Byte enables for bus cycle
ALE	TTL	input	Can be used to latch address
ADS#	TTL	input	Indicates address cycle
RDYRCV#	TTL	output	Asserted when write/read completed
DEN#	TTL	input	Data on bus
W/R#	TTL	input	Indicates whether cycle is write or read
DT/R#	TTL	input	Indicates direction of data flow on bus
CLK	TTL	input	Bus clock (33MHz)
BLAST#	TTL	input	Indicates this is the last access in a burst
RST#	TTL	input	Bus reset. Initialises CC to known state.
SPARE	TTL	output	Unused connection to i960 interface.
MISSCLK±	for LED	I/O	Connect to missing clock LED.
INTCLK±	for LED	I/O	Connect to internal clock LED.
CMD±	for LED	I/O	Connect to CMD LED.
FULL±	for LED	I/O	Connect to FULL LED.
-5.2V		power	-5.2V power supply.
+12V		power	+12V power supply.

10.2 Pin-Out of the Control Connector

This connector is a five row DIN 41612 type.

	A	B	C	D	E
1	GND	GND	+5V	PCDATA0	PCDATA1
2	CALPM0	CALPM1	GND	PCDATA2	PCDATA3
3	CALPM2	CALPM3	+5V	PCDATA4	PCDATA5
4	CALPM4	CALPM5	GND	PCDATA6	PCDATA7
5	CALPM6	CALPM7	+5V	PCDATA8	PCDATA9
6	CALPM8	CALPM9	GND	PCDATA10	PCDATA11
7	CALPM10	CALPM11	+5V	PCDATA12	PCDATA13
8	CALPM12	CALPM13	GND	PCDATA14	PCDATA15
9	CALPM14	CALPM15	+5V	STATUS0	STATUS1
10	CALPM16	CALPM17	GND	STATUS2	STATUS3
11	CALPM18	CALPM19	+5V	STATUS4	STATUS5
12	CALPM20	CALPM21	GND	STATUS6	STATUS7
13	CALPM22	CALPM23	+5V	GND	WRITING
14	CALPM24	CALPM25	GND	RXREAD	GND
15	CALPM26	CALPM27	+5V	GND	RXDATA
16	CALPM28	CALPM29	GND	DONE	GND
17	CALPM30	CALPM31	+5V	GND	PC_END
18	CALPM32	CALPM33	GND	PC_RD#	GND
19	CALPM34	CALPM35	+5V	GND	PC_FULL
20	CALPM36	CALPM37	GND	CLINK	GND
21	CALPM38	CALPM39	+5V	GND	CLRDYA#
22	CALPM40	CALPM41	GND	CLRDYB#	GND
23	CALPM42	CALPM43	+5V	GND	CLKDA#
24	CALPM44	CALPM45	GND	CLKDB#	GND
25	CALPM46	CALPM47	GND	GND	DLRDYA#
26	CALPM48	CALPM49	GND	DLRDYB#	GND
27	CALPM50	CALPM51	GND	GND	DLERRA
28	CALPM52	CALPM53	GND	DLERRB	GND
29	CALPM54	CALPM55	GND	GND	SYCLK
30	CALPM56	CALPM57	GND	GND	GND
31	CALPM58	CALPM59	GND	ESYSCLK+	ESYSCLK-
32	GND	GND	GND	GND	GND

Where the signals have the following meanings:

Signal Name	Logic Level	Input/Output to/from CC	Description
CALPM (59:0)	TTL	input/output	Connected directly to J3, to allow PC interconnection.
PCDATA (15:0)	TTL	input	Carries the words to write into the TUQ
STATUS (7:0)	TTL	input	Indicates transfer status
DONE	TTL	input	Goes high when transfer is done
PC_END	TTL	input	Goes high when transferring the last word of the result
PC_RD#	TTL	output	Low enables the PCDATA bus from the PC
PC_FULL	TTL	input	High asserts FULL to the FCTS
RX_READ	TTL	output	High indicates that register read data is expected up the DLINKs
RX_DATA	TTL	output	High indicates that event data is expected up the DLINKs
CLINK	TTL	output	The CLINK to the FEEs, timing with respect to the TTL version of the SYSCLK.
WRITING	TTL	output	High indicates that the CLINK is being used to write to a front-end register.
SYSCLK	TTL	output	The 59.5MHz clock from the FCTS in TTL form.
ESYSCLK±	ECL	output	The 59.5MHz clock from the FCTS in pure ECL form.
CLRDYA# CLRDYB#	TTL	input	Active low, indicates the CLINK (A or B) is ready for data. Pulled high on CC so default is false.
CLKDA# CLKDB#	TTL	input	Active low, indicates the CLINK (A or B) is locked. Pulled high on CC so default is false.
DLRDYA# DLRDYB#	TTL	input	Active low, indicates the DLINK (A or B) is ready for data. Pulled high on CC so default is false.
DLERRA DLERRB	TTL	input	Active low, indicates there was an error from the DLINK (A or B) receiver. Pulled high on CC so default is false.

11. FRONT-PANEL INDICATORS

The front-panel of the ROM is described in detail in the Dataflow Platform Users Guide.

The controller card has only one TTL output LEMO on its front-panel. This is driven directly from a bit in the control register. This is meant to be used as an aid in debugging.

The controller card drives the following LEDs on the PC. All LED drive signals are stretched to catch short pulses.

LED Name	Colour – Purpose	Description
Missing Clock	red – error	Lit when SYSCLK is missing.
Internal Clock	yellow – warning	Lit when the CC is running from its internal clock (i.e. not the FCDM).
CMD	green – advisory	Lit when a command is received on FCADATA or simulated by the switch unit.
FULL	yellow – warning	Lit when the FULL signal is asserted to the FCDM.

For each of these signals there is a positive and negative connection on the connector to the PC: connect the anode of an LED to positive and the cathode to negative, through a suitable series resistor. The CC can drive up to 20mA, but the PC must limit the current using a resistor. All LEDs are illuminated when RST# is asserted.