LHCb DAQ 2001-136

Readout Unit

FPGA version for link multipexers, DAQ and VELO trigger

LHCb Technical Note

Issue: [1]
Revision: 2.2

Reference: LHCb DAQ 2001-136
Created: 15 June 2001
Last modified: 30 October 2001

Prepared By: Hans Müller, José Toledo, Angel Guirao, Francois Bal

Summary

The FPGA-based Readout Unit (RU) was designed as entry stage to the readout networks of the LHCb data acquisition and L1-VELO topology trigger systems. The RU performs subevent building from up to 16 custom S-link inputs towards a commercial readout network via a PCI interface card. For output to custom links, as required in datalink multiplexer applications, an output S-link transmitter interface is alternatively available. Baseline readout networks for the RU are intelligent Gbit-ethernet NIC cards for the DAQ system and SCI shared memory network for the L1-VELO system. Any new protocols, like 10Gbit ethernet or Infiniband may be adopted as far as proper PCI interfaces and Linux device drivers will become available. The two baseline RU modes of operation are: 1.) link-multiplexer with N*Slink to single-Slink 2.) eventbuilder interface with quad Slink-to-PCI network interface.

Incoming event fragments belonging to the same event-tag are derandomized, buffered and assembled into single sub-events. Following a push-through scheme with intermediate buffering and subevent assembly, new subevents are retransmitted in the same order to the output network. Destination address allocation and synchronization protocols can be implemented either via the bi-directional network interface or via a dedicated link available by default for output traffic scheduling.
The subevent building process is based on the matching of equal event numbers in the headers of the low-overhead Subevent Transport Format (STF). The latter was optimized for pipelined hardware state machines and failsafe transmission. Embedded in 4 words of header and trailor, STF cointains fields for link parity, different data types, fast error tagging and redundancy for error detection. Input eventsizes of 64 times of a nominal 0.5 Kbyte block size can be transmitted within a 80 Mbyte/s bandwidth envelope. The dual-port eventbuffer can accumulate up to 1000 events of 2 Kbyte average. Programmable logic is used both in the input and output stages, allowing for flexibility in the scope of applications and their variants which are designed using high-level-language simulation and synthesis tools. Any remote re-configuration of applications is almost instantaneously possible via the networked PCI host card, the MCU, which runs a local Linux operating system, and which boots via its LAN network from a server.
All four FPGA chips are interconnected via three on-board PCI bus segments which, apart from FPGA configuration, also serve for remote access to control registers or to the data buffer. The PCI segment on the RU output is 64 bit wide and mainly dedicated to make the maximim PCI bandwidth of 1/2 Gbyte/s available for the L1-VELO application, which is based on tandem PCI master. The root segment of the PCI bus is hosted by a networked microprocessor PMC mezzanine card which runs a diskless Linux operating system. Such a Monitoring and Control Unit (MCU) with 33 MHz PCI bus mastering was built by the RU design team whilst commercial equivalents with up to 66 MHz PCI bus clock became available later. Apart from the standard ECS control tasks, the MCU is needed to initialize all PCI devices and especially specific Network Interface Cards.
The RU hardware architecture, initially targeted only for DAQ and Multiplexer applications of up to 80 Mbyte/s has been optimized in a board revision to be applicable also for the L1-VELO trigger network, which requires up to 200 Mbyte/s throughput at a 1 MHz nominal trigger rate. Also a 16 bit-wide, programmable I/O link, the “Tagnet” was added for the traffic scheduling of the DMA engines which were specifically designed for the L1-VELO topology trigger . Tagnet allows to interconnect

banks of RU's horizontally in a tagnet ring. A Tagnet master dispatches free destination address tags to the RU's in the ring which strip them from the ring, use and return them. The trigger throttling feedback signal, as required for buffer protection in the LHCb write-only architecture is generated on the RU front panel, indicating that buffers have reached a (remotely programmable) fillstate.
In DAQ or Multiplexer running conditions, trigger rates up to 100 kHz are supported within a 80 Mbyte/s throughput envelope. For the L1-VELO applications, rates in excess of 1 MHz with short blocks are supported within a throughput envelope of 200 Mbyte/s. Six RU modules have been produced as 9U, 10 layer PCB boards, using LEP crates and racks as a convenient power and cooling framework, and allowing for up to 52 RU's in a watercooled LEP rack. Four Slink mezzanines of any compatible link technology (with up to 4 link inputs each) can be inserted in the front panel space. On the rear side, the RU module carries two PMC mezzanines: the networked MCU and the output link card, i.e either NIC or Slink. One RU has also been very successfully produced in halogene-free PCB technology as a first test at CERN for halogene-free PCB production which will eventually be mandatory by European regulations. The FPGA-based RU is a completely tested and reproducible 9U module and includes co-designs like the S-link I/O card for data transmission over up to 25 m of standard network cables, S-link pattern generator cards to produce STF formatted test data and a networked processor PMC card. An PC-based RU exerciser using PCI-to-Slink cards will complete the FPGA based Readout Unit Project, and allow for error-integrity and performance testing of Readout Unit systems with any configurable data sets and tunable trigger rates.

A• Glossary of terms

- CSR .....Control and Status registers (FPGA registers for DCS control)
- DAQ .....Data Acquisition (system from RU to CPU farm)
- DCS .....Detector Control System, a network for (slow) control
- DEB .....Directory Entry Block, a directory section of SE entries in the SEB
- EBI .....Event Builder Interface (output stage for xmitting SE's)
- FEM.....Frontend Multiplexer (N-> 1 data link concentrator)
- I2C .....Simple Inter-IC-bus by Phillips, used for configuration
- LVDS ..... differential signalling standard IEEE 1596.3
- MCU.....Mezzanine Card Unit (networked CPU with PCI and control bus)
- MUX..... Multiplexer stage (N*event fragments-> 1 Subevent)
- NIC .....Network Interface Card (PCI card for the readout Network 32/64 bit)
- PCI.....Peripheral Component Interconnect (rev 2.2 Industry I/O bus)
- PMC.....PCI Mezzanine Card (IEEE P1386.1)
- RU .....Readout Unit (4*link->PCI/Slink subevent builder/multiplexer)
- RN .....Readout Network (technology for the eventbuilder network)
- SE .....Subevent (Event block made from N event fragments)
- SEB .....Sub Event Buffer (large event buffer with data and directory)
- SEM.....Sub Event Merger (input stage FPGA)
- Slink.....CERN 32 bit interface standard for high speed links
- Fastbus.....Fastbus IEEE 960 standard, crate standard of LEP experiments

Document Status Sheet

Table 1 Document Status Sheet

1. Document Title: Readout Unit FPGA version for link multipexers, DAQ and VELO trigger			
2. Document Reference Number: LHCb DAQ 2001-136			
3. Issue	**4. Revision**	**5. Date**	**6. Reason for change**
1	2.1	30.10.2001	Summary rewritten to be equal with Appendix on RU in DAQ TDR
1	2.2	6.12.2001	Major cleanup of text, references, addition of latest laboratory results

Table of Contents

1 Introduction

The DAQ [9] and VELO topology trigger-network [33] systems of the LHCb experiment share the need for Readout Units (RU). As input stage of the full event-building network, RU's receive event fragments (EF) on multiple input links from the previous stage and assemble them into larger sub-events (SE's) which get transmitted on a single output (link or network) to the next stage. Both EF's and SE's conform to the low-overhead subevent Transport Format (STF) convention (see chapt. 4.1 .) or one of it's derivatives. The subevent building operation is entirely performed in programmable FPGA logic (see chapt 11) and an associated subevent buffer (SEB) (see 5.2). The subevent building operation assembles EF's or SE's that belong to the same event identifier into new SE's which are buffered, verified and re-transmitted in the same order. Two interface possibilities for an output network exist: PCI [19] or Slink [10] (see chapt 8 .) As part of a DAQ readout hierarchy, a RU output may feed the input of another RU to perform the subevent building operation in two successive levels of buffering which may be linked via an Xon/Xoff protocol. Within such an architecture, the first layer of RU's performs frontend multiplexing, the second layer interfaces to the readout network and to its eventbuilding protocols. As part of the L1-VELO topology trigger, the RU receives trigger data from the L1 Trigger output port of the ODE electronics [32]

The sub-event assembly process is developed and simulated using tools (see chapt 11) based on the IEEE-1076 VHDL standard language[1] for unproblematic portability and maintenance. For the highest data transmission performance requirement, the 64 bit PCI is used as an output bus (see chapt 12 .) The application logic is implemented in Field Programmable System Chips (FPSC) [12] with embedded 32/64 bit PCI cores.

Three application areas have been identified in the LHCb experiment where sub event-building functionality via Readout Units is needed [6]:

1.1 Front End link Multiplexer (FEM) after Level-1 [13]

Frontend Multiplexers reduce the number of low-bandwidth output links originating from the level-1 buffers. Up to 16 input links can be connected on the input stage of one RU whose output stage is re-transmitting reassembled subevents to a single link of correspondingly higher bandwidth. The purpose is to adapt the bandwidth to the input capacity of the next stage and reduce the number of links. The bandwidth requirement for a FEM application is a factor of four less than for a RU application in the DAQ entry stage since four FEM output links feed one RU input [Figure 2].

In the FEM application, the RU is equipped with Slink receivers/transmitters (see 8.2) at both inputs and output. Subevents are built and forwarded at a moderate throughput, implying that normally only a low number of subevents are queued in the buffer. FEM's may however be used in Xon/Xoff mode where the following RU stage is the Xon/Xoff master,

1. generated via the Visual HDL tool from state machines with different FPGA configuration

asserting Xoff when its own buffer gets full. There is up to 2 Mbyte of subevent storage per RU, allowing that in normal running conditions up to 2000 subevents of 1 kbyte can be buffered on each RU or FEM layer. The trigger throttle signal is asserted at a programmable[1] buffer occupancy level when its eventbuffers tend to overflow (see 7.1).

1.1.1 Slink to Slink

On the RU, Slink (see 8.2 .) is used as a unidirectional, transmission medium which is handled by the RU as a 32 bit FiFo bus, hence there are no logical link layers to be considered.

A quad Slink Rx card (see 9.8) allows to connect up to 16 frontend links on one RU. Subevents received on all active inputs[2] are assembled and retransmitted to the Slink output which can transmit up to 80 Mbyte / s. The SLINK logic transmits subevents directly from the RU's subevent buffer to an Slink Tx output card. The subevent size (S) at the Slink inputs is limited by the input FiFo capacity to 32 kbyte. The maximum event rate (f) in this application is 100 kHz.

Within above limits, the relation $N \times f \times S \leq 80 Mbyte/s$ applies, where N is the number of input links [Figure 1].

At a nominal 40 kHz rate and 16 input links, the average eventsize may be up to 125byte per input link, producing SE's of up to 2 kbyte size at the output link.

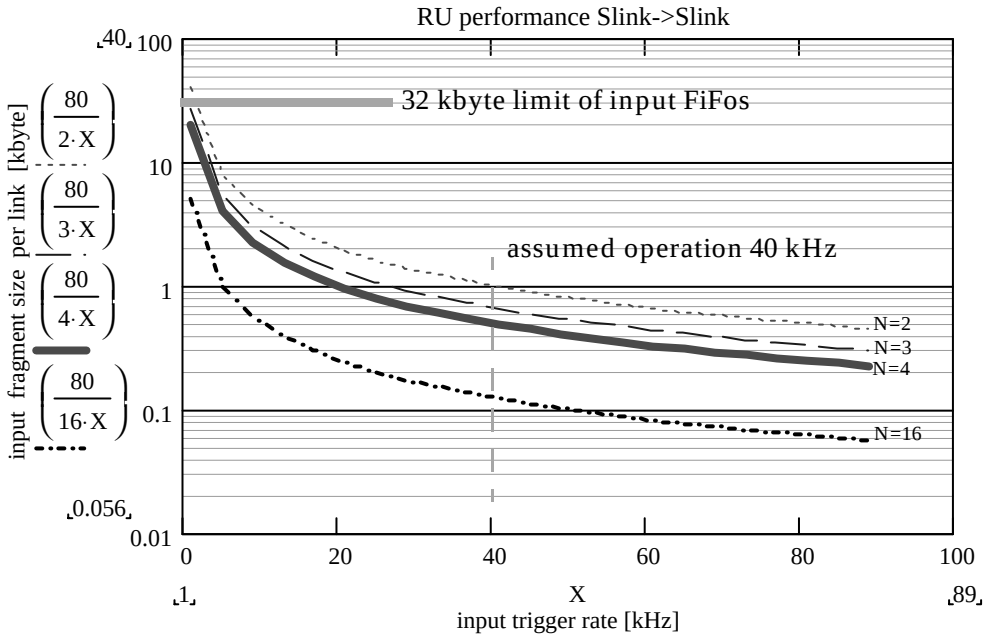


Figure 1 Multiplexer performance envelope

Fragment size versus trigger rate dependency

y: input fragment size in kbyte per link

x: input trigger rate in kHz

N: number of input links:

1. programmable via a CSR register in the PCI address space.
2. the number of active inputs is programmable via a CSR register

1.1.2 Slink to "X-link[1]"

This is like in 1.1.1 , but any "Slink compatible output link " may be implemented via the PCI bus, requiring a link transmitter card on the RU's PCI output bus. This is for example required when link technologies are used which require initialization of logical layers (example Gbit ethernet) Assuming that such a card is implemented with an a "Technology-X" chip with a 64 bit PCI bus backend, the only requirement is that the link receivers at the far end must implement the same "Technology-X" protocol. For a "X-link" payload[2] of X Mbyte/s, the relation $N \times f \times S \leq X\,Mbyte/s$ applies, where N is the number of input links. An example is GiGabit ethernet with a payload of 80 Mbyte/s, i.e. case N=4 in Figure 1 applies also here. At a nominal 40 kHz rate and 4 input links, the average eventsize is 500 byte per input link and 2 kbyte per output link.

1.2 Readout Unit (RU) entry stage for the DAQ system [9]

The readout network of the DAQ system is interfaced by commercial Network Interface Cards (NIC) on the PCI output bus on the RU [Figure 2]. The DAQ eventbuilding protocols and destination allocation are handled by a NIC card, which may autonomusly read subevents from the RU's subevent buffer.

In the standard DAQ application as defined in the technical proposal [13], RU's receive subevent fragments from multiple links and assemble them into new subevents, which are buffered and, depending on the DAQ readout protocol, either fully forwarded to the DAQ (full readout, full data volume to switch), or forwarded only after a pre-phase of fractional forwarding and a Level-2 decision. For this "phased readout", only a subset of the data volume is transmitted to the switch[3].

The simpler, full readout protocol corresponds to a "push architecture", i.e the output stage writes the full subevent data whenever subevents are available in the buffer. The assumption is that always enough buffer space is always at the destination. Flow control is implemented via an overall trigger throttling (a grouped Xon/Xoff feedback from all destination buffers to the trigger supervisor). The allocation of the destination address is part of the eventbuilding protocols and fully handled by the NIC card [23].

1. X-link standing for any Gigabit technology
2. real bandwidth of data after stripping overheads
3. The phased readout can be implemented on the RU, however is not considered any more for the LHCb DAQ system due to a higher protocol complexity.

1.3 RU as entry stage to the Level-1 Trigger network [28]

In the L1-VELO trigger application [17], RU's receive cluster fragments of a nominal size of 64 bytes from the Si-Strip detector at a nominal trigger input rate of 1 MHz. The RU builds subevents across the 64 bit PCI output bus into a NIC card which interfaces to a very low latency, shared memory network. This paradigm allows that a data source may write data to a destination address like to local memory, i.e. at very low latency[1], just by knowing a local PCI address. The latter is available to the RU output stage by picking any such free address token from the L1 toplogy scheduling network, the Tagnet. FPGA sources simply write data to a PCI address, which due to the shared memory paradigm, transmits the data through the NIC directly to (a properly mapped) CPU memory in a destination node of the trigger network. Implemented in two FPSC chips there are DMA engines [27] used via tandem PCI masterhips to reach the full PCI bandwidth[2]. Effectively payloads of up to 256 Mbyte / s on 64 bit PCI NIC cards with equal or higher performance[3] can be achieved. Network congestion is avoided by destination address allocation and traffic scheduling, implemented via the programmable 16 bit Tagnet lines [see 8.3] which "horizontally" interconnect adjacent RU's as shown in Figure 2 .

1. remote write in the order of 1 us
2. in a tandem master operation 64 bit wide transmit buffers in one transmitter are filled whilst the other one is transmitting to PCI
3. current choice is a Dolphin SCI (PMC-64/66 / D333) with 2 * 800 Mbyte/s link capability

1.4 Application areas in LHCb

The Readout Unit has been designed to cover all three LHCb applications[1] (DAQ, FEM, L1-VELO) and can be considered as a configurable, multi-purpose unit. The RU and FEM layers in the FEM/DAQ readout architecture shown in Fig[2] are based on the same module, programmed and equipped differently. .

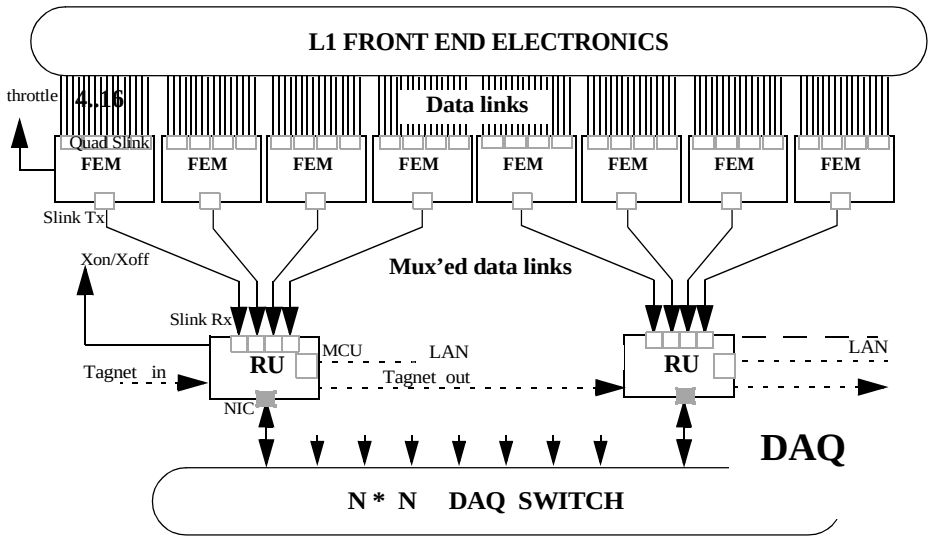


Figure 2 RU in Frontend and DAQ

ReadoutUnit in FEM and DAQ applications of LHCb. Up to 16 inputs in FEM mode, 4 inputs when used as RU. The LAN connection provides remote control. The Scheduling network link (Tagnet) is not required in these two applications

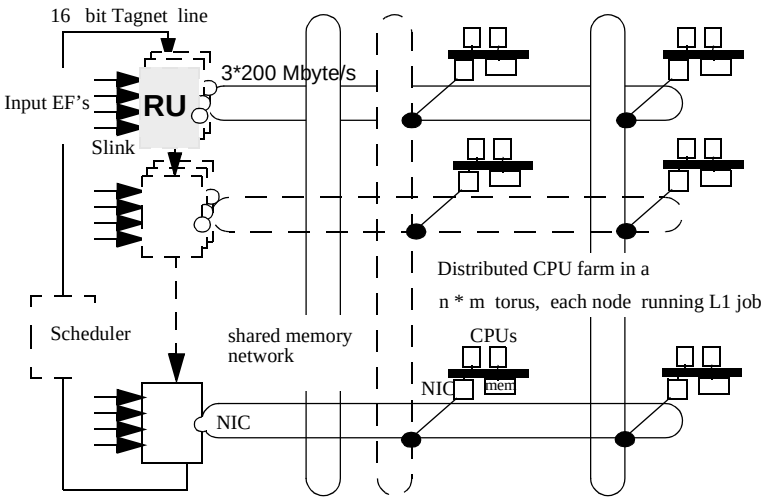


Figure 3 RU in L1-VELO trigger

Readout Units building and transmitting event fragments in the 1-MHz VELO trigger application of the LHCb Si Strip detector. A two-dimensional torus network with distributed CPUs in the intersections is used. The torus is currently implemented as horizontal and vertical SCI ringlets of 800 Mbyte/s bandwidth each. Three RU's serve one ring. The Tagnet is used to allocate the destination addresses from a dedicated RU operating as a Tagnet scheduler

It requires that on average DAQ eventflow of 4 Gbyte/s (a total average of 100 Kbyte events at 40 KHz rate) can be transferred via a stage of N * RUs. Each RU is connected at its outputs via a point-to-point network link to an input port of a N-by-N switch, where N is in the order of 100 Readout Units[2]. Taking as an example a network technology like Gbit ethernet, each RU

1. with different configuration of FPGAs and link cards
2. The revision of L1 trigger rate by a factor 2 a corresponds to a higher number of RU's

output link could transmit 40 Mbyte/s of payload to a switch port. The Number of RU's depends linearly on the required throughput of the DAQ system and on the payload capability of the readout network. It scales down with higher payload capability of the readout network and scales up with the trigger rate and the overall eventsize.

The application for the L1-Velo network is shown in Figure 3. Given a maximum payload of 256 Mbyte/s with 64 byte clusters, up to four RU's are needed to fill one GiGabyte ringlet[1] of the torus. Average clusters sizes of 32 bytes provide a safety factor of 2. The 2D system is fully scalable to larger cluster sizes by adding more RU's and ringlets to the torus. As an example, ca. 20 RU's are needed, for a total throughput of 4 Gbyte/s.

1.5 Readout unit design parameters

The RU parameters for all applications in the LHCb experiment are summarized in Table 2.

Table 2 Design parameters for RU applications in LHCb

PARAMETER	FEM	DAQ	VELO
Number of inputs	N=1..16 (32 bit Slink)	N=1..4 (32 bit Slink)	N= 1..4 (32 bit Slink)
Nominal rate	40 kHz [80 kHz]	40 kHz [80 kHz]	1 MHz
average fragment size per input @ nominal rate	1 kbyte / N	2 kbyte / N	128 byte / N
max average fragment size per input @ nominal rate	2 kbyte / N [1 kbyte / N]	4 kbyte / N [2 kbyte / N]	256 byte / N
max. instantaneous fragment size per input @ nominal rate	2 kbyte	4 kbyte	256 byte
max. fragment size per input @ lower rate	32 kbyte@2.5 kHz	32 kbyte@5 kHz	-
average output subevent size	Slink: 1 kbyte [0.5 kbyte] PCI: 2 kbyte [1 kbyte]	PCI NIC: 4 kbyte [2 kbyte]	128 byte via tandem PCI master to NIC
Throughput normal	40 Mbyte/s	40 Mbyte/s	128 Mbyte/s
Throughput maximum	80 Mbyte/s	160 Mbyte/s	256 Mbyte/s
Max. subevent buffer capacity (average eventsize)	2-10.. of 1 kbyte (up 2000 in Xoff mode)	1000 * 2 kbyte	16000 * 128 byte SE's

Parameters corresponding to a higher L1 output rate of 80 kHz for the FEM and DAQ applications are shown in square brackets. The number of link inputs N may vary and is therefore shown as a scalefactor for some parameters. The required average figures are smaller than the maximal possible figures by factors of 2 (FEM) and 4 (DAQ).

1. Considering SCI technology based on LC3 , 0.8 Gbyte/s ringles are used in today's commercial 2D systems.(http:/www.dolphinics.com)

1.6 Conceptual RU options and limitations

The SEB buffer can be equipped up to 2 Mbyte capacity, of which ca 10% are used for directory information. The data storage is of variable block size, hence in order to calculate the maximum event capacity of a RU, the average subevent size at the output must be accomodated in ca 1.8 Mbyte of memory.

The FEM requires conceptually a very low event occupancy in its SEB buffer. The defacto availability of much more SEB buffer in an FEM application would also allow using FEMs like “input FiFos” for the upstream RU’s, providing that the RU returns a Xon/Xoff the corresponding FEM’s. This is indicated in Figure 2 and can be implemented either via the RU’s throttle output or via Slink.

The RU input stage is designed to operate with Slink and STF. The Slink provides protected control words for the header and trailor words which generate markers in the input FiFos. which are important for the readout timing and delimiting. The STF format must contain a monotonically increasing event identifier in the header word, allowing the subevent-building hardware process to be performed by comparison of known event identifiers (N + 1). Empty events must send empty STF frames, such that the RU’s subevent building process does not require accept-timeouts which would lead to complicated intermediate storage. Inversely, missing empty event frames or non-monotonic event numbers are signals which generate error entries in the STF error fields.

The parameters for the DAQ application are based a the use of a 32bit @ 33 MHz NIC network card with an inherent NIC limitation to practically 80 Mbyte/s. For a potential 160 Mbyte throughput in a DAQ application, or dual network links per NIC card, a 64 bit card with at least 160 Mbyte/s capability is required.

The high throughput requirement (200 Mbyte/s) of the VELO-L1 requires a tandem PCI master operation on a PCI output bus configured to 64 bit, 66 MHz operation. For this purpose also the PCI host mezzanine card must operate at 66 MHz on the PCI bus[1].

For a Slink-to-Slink (FEM) operation the PCI host card is in principle not mandatory since no NIC card needs to be configured. The card is in this case only needed if remote control and monitoring is desired.

The Tagnet implemention is a programmable option. The 16 bit wide LVDS I/O links could also be used for any other purpose.

A JTAG chain for FPGAs and other chips is available on both a cable post connector on the RU board and via the MCU user connector P14 (optional).

The I2C link for (permanent) programming of the RU clock domains is driven from the optional P14 user connector of the MCU. Sice this is required only for factory settings, the P14 connector is not a mandatory requirement for MCU mezzanine cards.

1. the current MCU card designed at CERN so far only operates at 32 MHz, however commercial 66 MHz MCU’s like the Motorola PrPMC800 or PrPMC750 may be used.

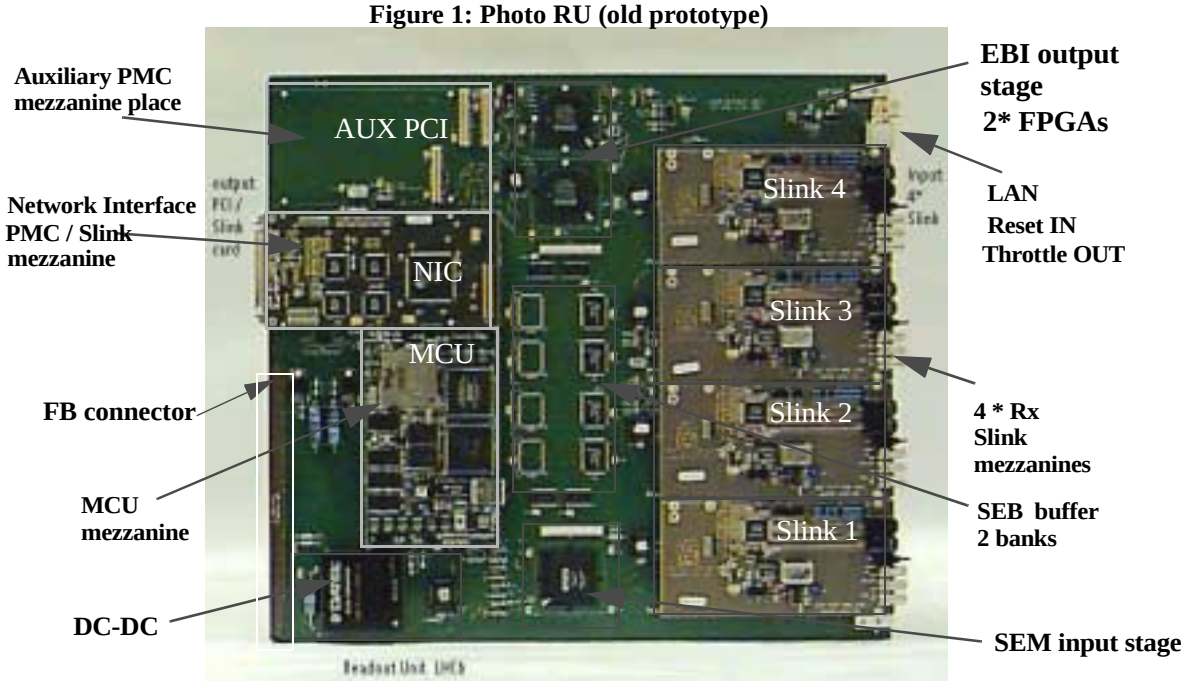
2 Readout Unit history and overview

The Readout Unit of the LHCb technical proposal [13] is a buffered, custom interface a commercial readout technology and the L1 buffers. From low-level buffer output link protocols it performs subevent building towards a readout network. Input event fragments from several links are buffered and assembled them into larger subevents (SE's) at the output. At 40 KHz with up to 1 kbyte[1] per input link the input bandwidth per input link was defined as 40 Mbyte/s. First ideas for a Readout Unit were described in [8]. The RU study and design project was started in 1999 with regular meetings [22] in the course of which a series of decisions also concering links and formats were taken and presented to the collaboration in the LHCb october DAQ workshop 1999 [5]. The global design decisions for building a RU in the CERN ED support group were:

- FPGA-based subevent building logic for speed and portability
- 9U Fastbus mechanics (for power and mechanics only)
- standard PMC mezzanine card technology for ease in upgrading external I/O technologies
- Slink with derandomizing FiFos at the link receiver inputs for link technology independence
- PCI and Slink at the output to provide two choices of output link interfacing
- an embedded processor card for networked PCI bus hosting, using the P-PMC standard

The first prototype RU module (Figure 2) was presented to the collaboration in the May 2000 LHCb workshop [4]. The crate standard chosen (chapt 10) for the RU is the Fastbus crate

Figure 1: Photo RU (old prototype)



and rack standard[2] of the LEP experiments. The RU module re-uses the existing LEP crates together with their rack-cooling infrastructure, using only the crate mechanics and power lines of the backplanes.

1. largest data producer, RICH1, 700 byte, extrapolated to 1 kbyte fragement size including overheads)
2. For example: URL http://www.wiener-d.com/fastbus.htm

The decisions taken during LHCb workshops and meetings are listed below:

- Subevent building of the Readout Unit is done in the same way as for the Multiplexers and the DAQ, using the proposed Subevent Transport Format STF [4].
- The event data transported within the STF transport frame are ignored by the RU or the Multiplexers, hence they need to be independently formatted.
- Error blocks are optionally appended in the trailor of the STF
- For error tracing, the link ID of the link from which subevents are transmitted is used as a geographical identifier of the link.
- Fast error flags are contained in the STF trailor to allow any stage (level-1,MUX, RU) to mark error conditions, or to quickly identify incoming erroneous data.
- Event-Id's in the STF header are assigned by the first data sources (i.e. level-1 electronics) and must be monontonously numbered (this adds one degree of consistency information and is faster than comparing)
- There is only one (not more and not less) data packet per event-ID sent out by any data source per trigger. This implies that also empty data packets must always be sent for STF consistency.
- The maximum event packet size per RU receiver port is about 20 times the nominal 1 Kbyte size at a proportionally reduced rate (2 KHz) (This applies for non-zero suppressed runs.)
- The RU buffers are protected by a fast throttle feedback signal (a sort of common Xon / Xoff) with programmable low- and high-water marks.
- Readout Units can be fully remotely configured and are controlled via a networked server (RU's have no connection to a backplane bus). RU's can be individually reset via an independent reset line.
- The electro-mechanical RU crate standard is the existing LEP crate standard (9U Fastbus, IEEE 960) but without using any Fastbus signals (only power and cooling)
- The "full readout" [1] protocol is deemed to be easier to implement than the "phased readout[2] of the technical proposal" (which would have to transfer much less data). It has been discarded in LHCb and hence allows that the event buffer option in the RU is much smaller.
- The trigger throttling via the DCS was discarded since the implied latencies (order of 20 ms and more) would have required to implement very large event buffers (expensive DPM)

1. A push architecture with round robin destination, flow control via common trigger throtttling
2. A combined push and pull protocol where the full event is only read after a level-2 decision

Pending are decisions on the allocation of bits in the error fields in the STF format. The type of error handling[1], error types etc. can only be implemented in the application code when this is specified. This is however independent on the RU hardware, i.e. is part of the VHDL programming domain.

Further variants of the SFT can be defined via a TYPE field. It is foreseen to derive a 2-3 word overhead STF for the Velo-Level-1 topology trigger which particularly is sensitive to overhead.

1. Error status bits and error history fields are defined in the STF, however error type definitions and corresponding actions are to be defined before being programmed into the FPSC applications

3 Final RU architecture & design criteria

The test results with the first prototye in May 2000 resulted in a list of improvements and simplifications leading to a design revision of a final RU module, as requested by the DAQ group for testing the eventbuilding protocols between intelligent[1] NIC cards [23]. The revised RU-II module was presented early 2001 [6] as a less expensive and more performant module (Figure 5) due to reduction of expensive components[2] and a revision of the bus architecture (Figure 4.). The diversification of components was further reduced, in particular a single, new type of FPSCs[3] was used with the benefit of a higher gatecount and faster 64 bit PCI handling.

The following points were considered for the design revision of RU-II during summer 2000.

A.) negative feature on Readout Unit prototype

- Too many components and connectors resulted in a expensive 12 layer PCB with highest quality requirement.
- FPGA configuration required a a dedicated bus (i960 bus from the commercial Compulab MCU)
- The choice of a commercial, networked plugin CPU card with LAN for the PCI bus initialization and slow controls turned out as a much too strong “company dependence” and also as an expensive choice. Non-standard connectors had to be integrated on the RU motherboard.
- The company had problems in porting Linux to the i960 CPU architecture, the I2C implementation did initially not work, the LAN had problems.
- The DPM buffer capacity[1] was too large due to the initial DAQ requirement to perform trigger throttling via the slow conrols return path, implying latencies of 20 ms.
- Two parallel FiFo chips per input channel were required in the prototype 64 bit input architecture (a significant cost factor).
- The FiFo flags used for indicating data availability to the FPGA at the input, resulted in the problem that the event readout from the FiFos was slow as soon as the stored part of the event was emptied, the remaining readout speed was than forced to lthe slower serial link transfer speed.

B.) Improved features on Readout Unit II

- The max. level-1 throughput requirement of 256 Mbyte / s could not be achieved with the prototype input architecture using a single, 64 bit bus for reading out all four input FiFos. The new architeture has been parallelized by using two 32-bit input stages operating in parallel, feeding a 64-bit output stage.
- A programmable 16 bit I / L link for a scheduling network likeTagnet [14] was implemented as required by the Level-1 application.
- The FPSC configuration via the Intel i960 bus was replaced (unified) via the PCI bus port of the FPGAs.
- The MCU was implemented as a standard processor PMC card, allowing for multiple vendor choices and removing entirely non-standard connector dependencies.
- Only one Flash Eprom is used per FPSC pair.
- Only a single FiFo is used per Slink input.
- The memory size is limited to 2 Mbyte[2] (enough for fast throttling with low latency)
- A diagnostic PCI connector has been added, this is defacto a standard PCI connector allowing to plug PCI tracers or PCI service cards.
- The SEB buffer is now readable and writable from both sides, i.e. in principle accessible via the PCI ports of both input and output stage.

1. up to 20 Mbyte

2. corresponding to a maximum of 2000 sub-events of 1 kbyte

As shown in Figure 4 the PCI bus interconnects all programmable FPSC chips with the PCI host and the NIC card. Two sub-ordinate PCI segments below the root PCI bus are buffered via PCI bridge chips. The PCI bus tree is implemented ortogonally to the data buses of the RU, the 64 bit PCI-1 segment however is used for both output data and configuration. The 32 bit root PCI segment contains the PCI host card and a standard 32 bit PCI connector for diagnostic purposes, allowing to insert PCI analyzer / generator cards[4]. Diagnostics of 64 bit PCI transactions is possible via a 64 bit PMC-PCI adapter card on the NIC emplacement.

1. intelligent NIC cards like the PMC697 from RAMIX contain onboard RISC processors and SDRAM
2. DPM memory, FiFo memory, Flash memory and non-standard mezzanine cards
3. Lucent OR3LP26, an FPGA with 64 bit, 66 MHz PCI core and 60-120 Kgates of user programmable logic
4. we used a TA700 from Catalyst during the development

Compared to the first RU prototype, the use of PCI as configuration bus for all FPSCs made the company-specific interface to a MCU card redundant and by using the P-PMC standard [15] for the MCU, the "special connector" area (compare Figure 1) of the previous MCU became redundant. As a consequence, the new 9U module could be routed on only 8 PCB layers. The addition of one FPSC for a revised, parallel input architecture resulted in a higher input stage performance. Extensive analog signal-integity checks (chapt 9.3) were performed, as a result of which the addition of a second PCI bridge chip for the delicate 64bit PCI signalling bus became mandatory. The photo of the (revised) RU module in shown in Figure 5.

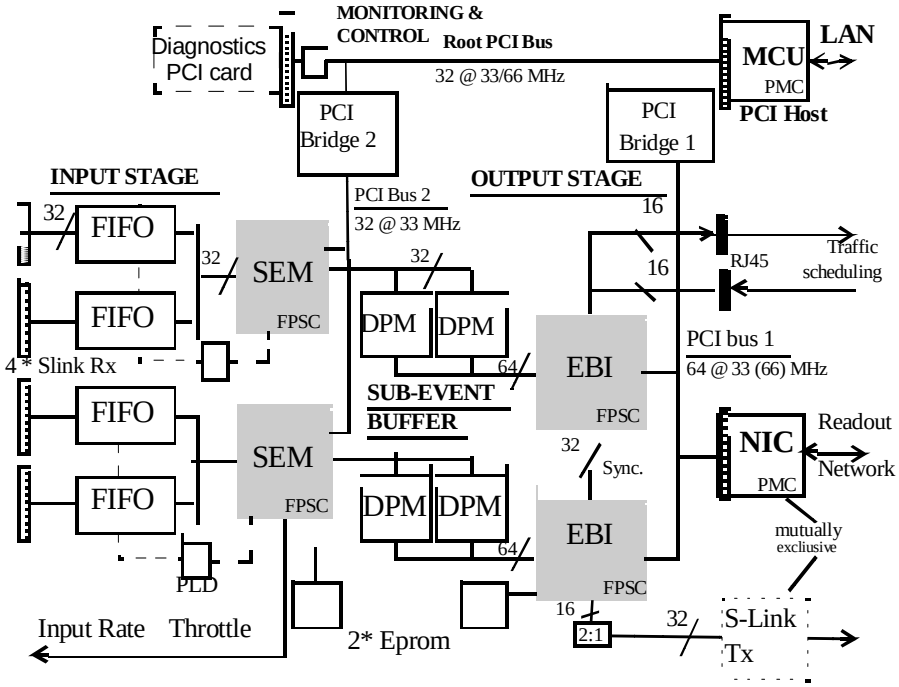


Figure 4 RU-II architecture

Two parallel Sub Event Merger (SEM) stages merge incoming fragments (by comparing event numbers) into their associated DPM buffers. The dual Event Builder Interface (EBI) logic combines the two DPM buffers to generate and transmit single sub-events to S-link or PCI. The networked Monitoring and Control Unit (MCU) is a PMC card functioning as host CPU of the root PCI segment. A decoupled, 64 bit PCI secondary bus doubles in function as the output bus of the EBI and as configuration bus of the EBI and NIC. Another decoupled 32 bit PCI bus provides access to control registers in the SEM. An auxiliary PCI connector on the root segment is available for diagnostics with standard PCI cards.

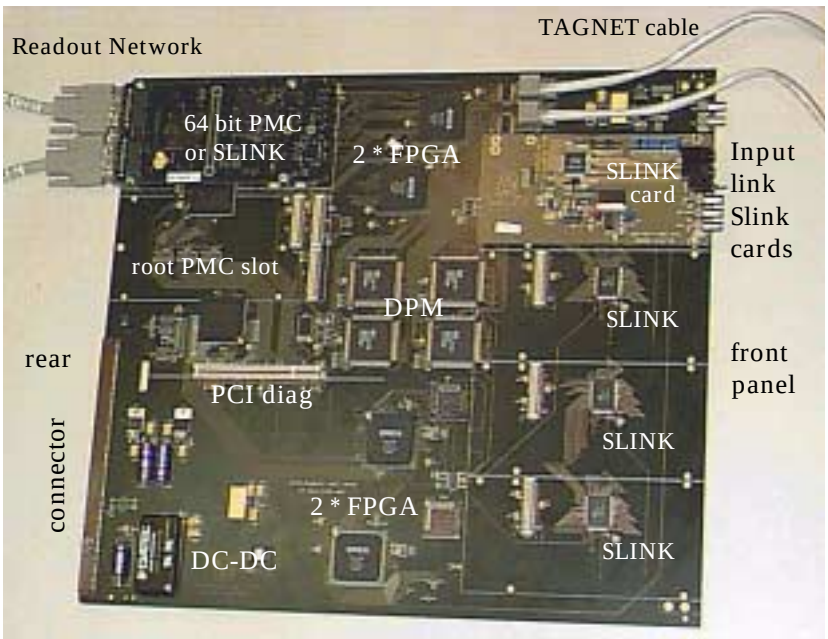


Figure 5 Readout Unit II

The revised RU-II with less components and the onboard 16 bit I / O cable option for "Tagnet" (RJ45 connectors). One Slink card is inserted in the top front emplacement, further emplacements are visible below with a single FiFo each. One NIC card is inserted in the top rear PMC / Slink slot. The root PCI slot below is for an MCU card as PCI host, below are visible a PCI bridge chip and the PCI bus, arriving at a standard PCI connector for diagnostics. Per FPGA pair (BGA chips) there is one DPM memory bank (2 * 2 chips) and one Flash Prom. The DC-DC concerter takes 5 V from the backplane and converts to 3.3 Volt.

3.1 Interface standards used on the RU

The RU interface connectors for data and/or control use common standards such as NIM, SLINK, PCI, I^2C, JTAG, LVDS, etc [11]. For the data links, we have chosen the CERN S-link [10] inhouse convention as a common FiFo-like physical layer, suitable for a variety of digital link technologies[1]. S-link provides a mezzanine-card environment which is mechanically compatible with the IEEE-1386 mezzanine standard referred to as " PMC" [16]. S-link adds to the PMC mezzanine connectors (see 9.5) a 64-pin, IEEE-1386 connector for 8, 16 or 32 bit data transfers. S-link may be used for any link technology whose physical layer can be adapted to the FiFo-like I/O paradigm. The RU's output link may be implemented either as an Slink card or an link card with PCI interface.

The choice of S-link makes the RU largely independent of specific protocols, encodings and leaves the choice of the physical link media (parallel, serial, optical or copper[2]) open. The link price for an S-link PMC is however higher than if a selected technology were directly placed on the RU motherboard. This could be subject of final revision, after a final technology choice is taken.

For the MCU[3] (Monitoring and Control Unit [14]) and for the NIC, the widely used IEEE-P1386.1 PCI mezzanine standard for the PCI bus, PMC [16], was adopted. A 32/64 bit PCI bus width 33/66 MHz option is available in the PMC slot of the NIC. The same mezzanine emplacement also proivides the S-link mezzanine connector, allowing that either a NIC card, or an Slink card can be used in this position at the rear side of the RU [Figure 5]. The MCU emplacement on a PMC position provides the user-defined I/O connector P14 of the Processor-PMC VITA standard [15] (see 8.1.1). This optional connector is used as custom interface [see 8.5] between the MCU card on the root PCI segment, providing also arbitration lines to the central root PCI arbiter on the MCU card, and driving the optional RU control buses like JTAG and I2C[4]. The Philips I2C standard is used for programming the clock chip settings in the factory or laboratory. The JTAG bus on the RU [see 8.5.2] interconnects all FPSCs, PCI bridges and PLDs and potentially allows to configure and test these these devices. Configuration and monitoring via the PCI bus is however the preferred, default solution in the RU. The onboard JTAG connector may be used via a cable connection to a JTAG master like (Altera Programmer) for configurations without using a MCU card.

The coaxial signal interface of the RU is the 50-OHM NIM for the front-panel RESET input and an open-collector TTL[5] for the trigger throttle signal output. The LAN technology is entirely integrated on the MCU card, which is plugged in the rear of the RU.

1. on the RU S-link cards we use National Semiconductors
2. see for example URL http://hmuller.home.cern.ch/hmuller/hsl.htm
3. Commercial MCU's with 64 bit, 66 MHz PCI operation exist now, at design time of the RU we required a MCU with 486 architecture which was designed and built by the RU team
4. used only for factory settings and not during normal running
5. the open collector choice allows to simply make a wired or between all RU's of a crate, to transmit a common throttle via a single line.

3.2 Data interface mezzanine cards for the RU

The following standard interface cards for links or networks may be used on the RU:

- **Single link SLINK receiver/transmitter** card, designed as I/O card (see 8.2) for the DAQ or VELO application.
- **quad datalink SLINK receiver** mezzanine card. Designed as extension to the single data link receiver card, allowing 16-fold multiplexing. The quad RCVR/MPX card (see 8.2) was designed for this purpose
- **PCI to "X-link"** transmitter card as an alternative to an Slink card, using the PCI connector of the mezzanine slot. Such a card could be designed with standard link chips which usually have a PCI backend bus.
- **optical SLINK transmitter** card for the MUX output (S-link transmitter card with Laser driver) Such cards are commercially available or could be designed as upgrade to the single Slink I/O card.
- **Network Interface PCI card (NIC)** for the DAQ readout network (PMC cards for Gbit-ethernet, ATM, SCI ..) Many such PMC cards are commercially available[1], including dual network PMC's.
- **TTCsr[2] receiver** mezzanine card [24]. There is no current LHCb application for using a TTC card on RUs, nevertheless this PMC has been used for tests

1. **Examples for Gbit ethernet PMCs:** PMC667 and 697 from RAMix Inc. PMC-3120 from SBS Technologies, ARPMC-600 from Artela Systems Inc. VMIPMC-6100 from VMIC, Dual Gbit Ethernet PMC: http://www.ccii.co.za
2. designed previously in the CERN ED group for RD12, it could be up-graded with new TTC chip

3.3 Control interfaces for the RU

All control interfaces are using standard interface connectors. The following are available on the RU.

- **monitoring & control unit** (MCU): a PMC card, using standard PCI mezzanine [15] connectors for 32 bit PCI and a (optional) user connector.
- **10/100 Mbit ethernet**: the standard RJ45 connector is part of the MCU card
- **fast throttle output**: NIM coaxial plug with open collector TTL signal for ease of oring RU's together (a modified coax cable with as many plugs as throttle outputs on one the input side)
- **reset input**: a fast RESET of the RU, or optionally[1] Xon/Xoff input for FEMs
- **JTAG interface** (4 pins on motherboard): same pinout as Altera JTAG programmer
- **TAGNET** input / output. interfaced via RJ45 network cables. The Tagnet being programmable via the FPSC, alos differnet inter-RU connections may be programmed in replacement of TAGNET.

1. RU's can be configured that the Reset Input is used as input to the FPGA logic.

4 Subevent Building

Both FEM and RU are the same modules, performing sub-event building in two successive stages (compare Figure 2). .The logical subeventbuilding process is depiced in Figure 6.

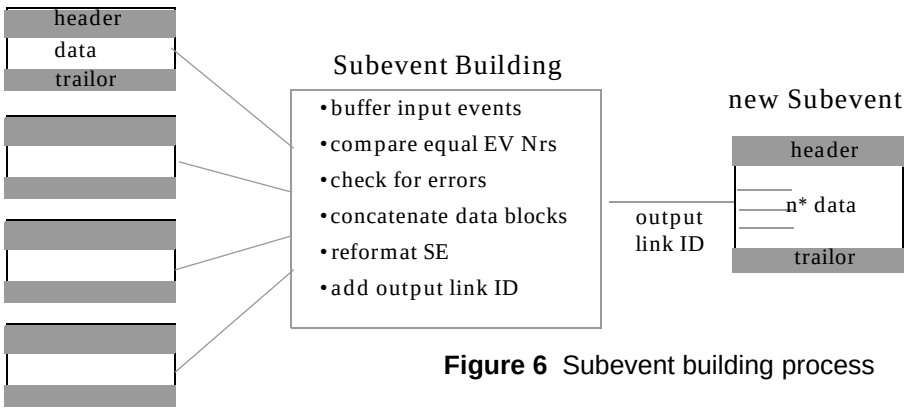


Figure 6 Subevent building process

Incoming STF-framed (see 4.1) SE's are buffered, compared for equal event numbers, checked and reformatted in STF as a new subevent, containing all input data as a concatenated block. The link ID of the output link is added before the new Subevent is made available to the output link.

An FEM outputs SE's via Slink or PCI, whilst the RU modules always output SE's via a Network Interface Card (NIC) on the PCI bus to a commercial readout network standard. The NIC card usage implies that the output stage of the RU must be able to perform PCI configuration of a NIC card via a PCI monarch for PCI device initialization via software drivers.

4.1 Subevent Transport Format (STF)

The format adopted[1] for transporting SE's (Figure 7) is optimized for pipelined sub-event building in FPGAs after the level-1 buffers.

The Subevent Transport Format (STF) consists normally of three 32-bit header and one trailer words that encapsulate incoming and outgoing data blocks. Data blocks contained within STF are transparent to the RU and must be self-consistent (i.e., have their own formatting convention).

Since data transport is via Slink, the first and last word are Slink control words with parity protection, hence their lower 4 bits SH (S-Link Header) and ST (S-Link Trailer) are used for error reporting. (see chapt. 8.2). The event identifier in the header is contained in the upper 28 bit field of the header control word. The total size of the block (32 bit words) is contained in a 20 bit field of the trailor control word, together with an 8 bit fast error flag field.

1. LHCb FE workshop January 2001

The 16 bit link ID field in the second header word identifies the outgoing link via a fixed number. This word also contains two 8 bit check bit and type fields which are to be defined for the purpose of CRC codes and data types. The default DAQ data type equals to 0, another data type is L1-VELO=1 (which implies a derived STF). The third header word contains the 16 bit OEB (offset to error block), which counts the number of following 32 bit data words in the data block (DB) .

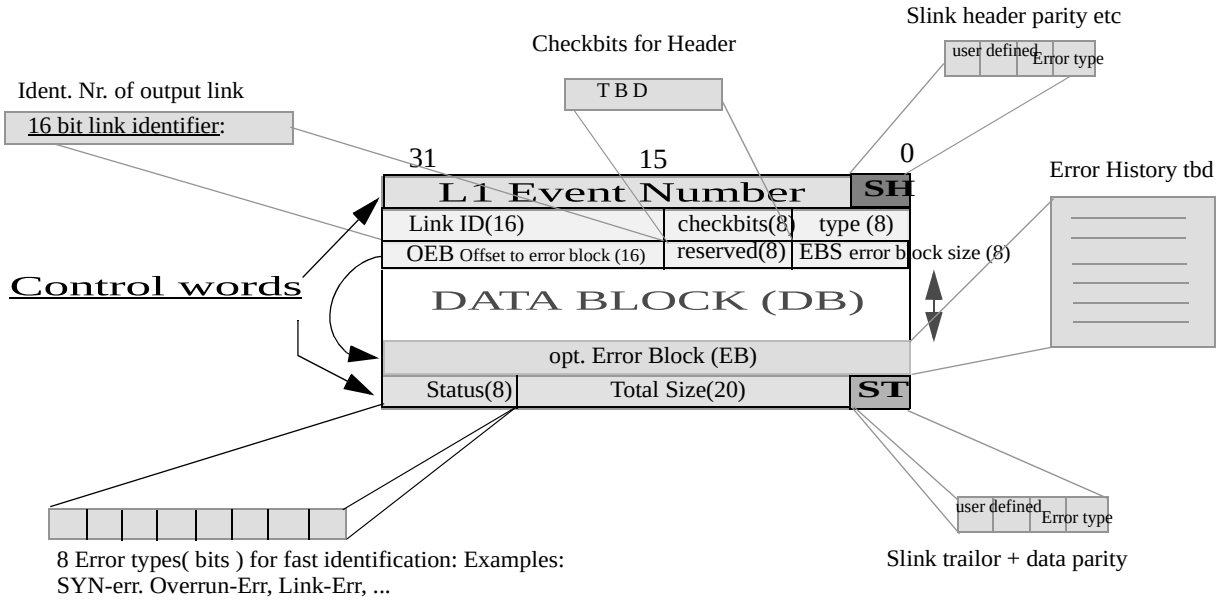


Figure 7 STF Format

The EBS is an 8 bit field counting the error block size. If there is no error block, this size is zero and the OEB points to the trailor word. There is further an 8 bit reserved field in the third header word, which must not be used until its use is defined. After the DB data block (maximum size 64 k 32 bit words[1]) follows the optional error block, which reports errors accumulated during the various stages of the subevent building. The last word contains the 20 bit TS total size field of all words in the block. With 4 fixed frame words, the total size it is calculated as

TS=4+EBS+OEB.

The essential fields for the subevent building process in the Readout Unit are:

- **L1 Event identification number:** 1st level accept number, 28 bit, monotonically increasing
- **OEB: offset to error block (or to trailor if EBS=0), equivalent to: number of contained 32 bit data words**
- **EBS:** Size of error block, i.e Number of its (32 bit) words, must be zero if no error block
- **DB = Data Block:** N * 32 bit words of data (any substructures are totally transparent to the RU)
- **EB = Error Block: optional** descriptor of error history during subevent building in successive stages
- **STATUS:** 8 bits of fast status information for 8 error classes, identified by each bit, default 0
- **TS = total size (4 + OEB + EBS):** 20 bit field containing total number of 32 bit words of the Subevent + header/trailor

1. if more should be required, the reserved field could provide a link bit to a following block.

Optional fields for the Readout Unit are:

- **SH/ST**= Slink error status and parity of the header (4 bits in total with user bits)
- **LID= Link ID**: a 16 bit field for identifying the link identifier to which the subevent is transmitted
- **CB= checkbits:** optional 8 bit checkbit field for the header word, default= 0
- **type:** 8 bit field for type of data, default DAQ=0, L1-VELO =1
- **reserved:** 8 bit field, reserved (do not use until defined)

The figure below shows an example of a subevent building process using the STF format. There are four input

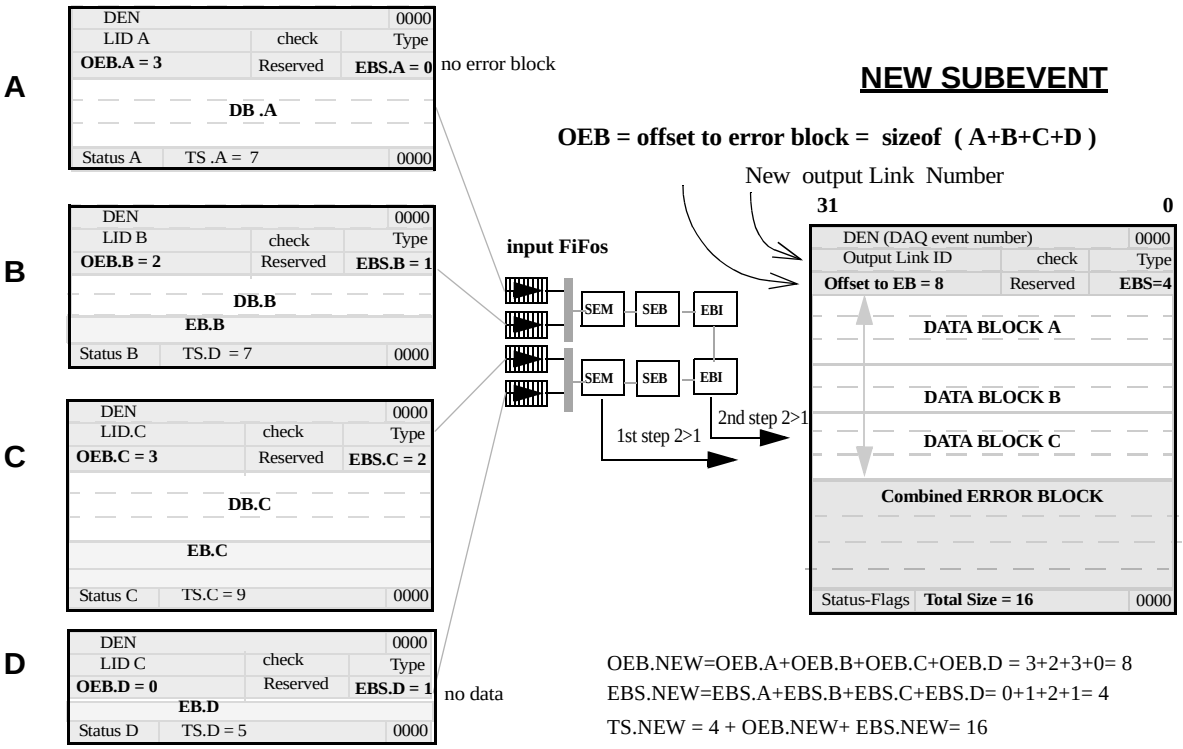


Figure 8 Detailed example Subevent building with STF

subevents A,B,C,D which get assembled in two steps, A,B and C,B are pairwise assembled by the SEM into 2 intermediate subevents residing in the SEB buffer. The EBI logic combines both into a new subevent, as shown on the right side of Figure 8 . The Total Size field (TS) in the trailors counts the number of all 32 bit words in the block. The offset to the error block (OEB) counts the number of 32 bit data words in a block. The error block size (EBS) counts the number of 32 bit words of the error block.

Subevent A consists of 3 data words and no error block, hence its OEB field contains 3 and there is no error block, EBS=0. With 3 data words, 0 error block and 4 header/trailor words, the total size TS is 3+0+4 =7.
Subevent B contains 2 data words and 1 error block word, hence OEB =2, EBS =1 and TS=7.
Subevent C contains 3 data words and 2 error block words, hence OEB=3, EBS=2 and TS=9
Subevent D contains 0 data and 1 error block , hence OEB=0, EBS=1, and TS=5

The rules for the composition of the new subevent are as follows:

The new data block size is OEB.NEW=OEB.A+OEB.B+OEB.C+OEB.D = 3+2+3+0= 8
The new error block size is EBS.NEW=EBS.A+EBS.B+EBS.C+EBS.D= 0+1+2+1= 4
The new total size is TS.NEW = 4 + OEB.NEW+ EBS.NEW= 16

5 RU hardware architecture

The horizontal view of a RU architecture (Figure 9) consists of a 32 bit input stage, a dual port buffer with 32 bit input and 64 bit output buses, and a 64 bit output stage. The vertical view consist of two dual input stages which are connected at the output stage together. This generates four logic blocks, implemented in FPSC chips which are interconnected via their PCI bus pins in a PCI bus hierarchy, of which the MCU card is host.

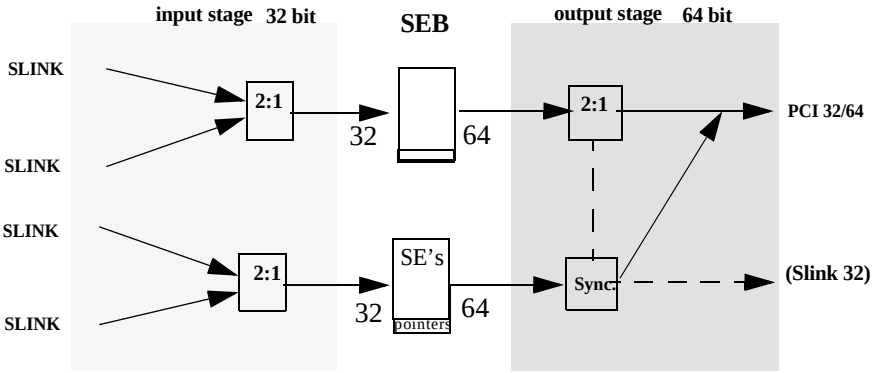


Figure 9 RU horizontal architecture

two parallel 2:1 input stages merge incoming EFs tagged with the same event number in their sub-event buffer (SEB). Further 2:1 sub-event building is implemented in the output stage combining the two SE's and transmitting complete SE either via PCI or S-Link .

Figure 10 below shows the full RU hardware architecture in more detail...

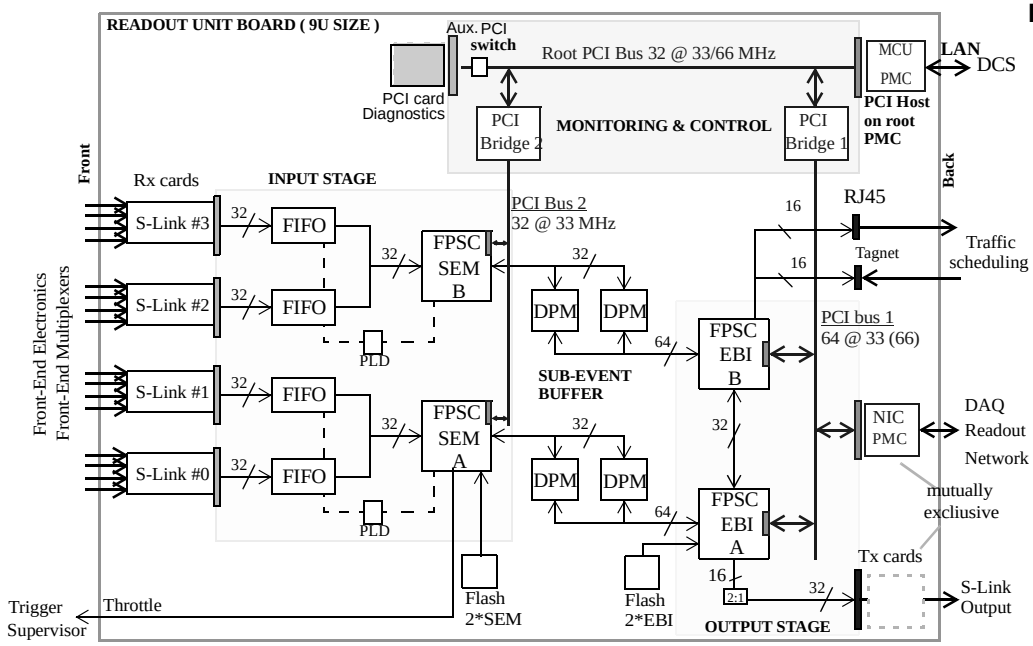


Figure 10 RU architecture

The input stage, buffer and output stage represent the horizontal slice which is important for transferring data. In a vertical view, control and monitoring paths are ortogonally implemented. The MCU, hosting the primary PCI bus is connected, via PCI bridges to the input and the outputs stages.

Two PCI bridges[1] separate the PCI bus in 3 segments which can operate independently. Only the 64 bit PCI bus segment at the output stage takes part in the event data transfer (from EBI to the Network interface card). The other two 32 bit PCI segments are used for configuration and control data.

For event synchronization, the two EBI's make use of a 32-bit bus interconnecting the two FPSC chips in the output stage. The high bandwidth PCI output bus can be used to combine subevents via successive PCI bursts into a single output buffer inside a NIC. If the NIC supports the PCI write combining [19], and if the delay between the successive bursts is within a timeout, the NIC auto-generates a single output dta block to the readout network.

5.1 Input stage (SEM = sub event merger)

The logic of one of the two identical dual Slink input channels is depicted in Fig [11]. S-Link receiver cards[2] (8.2) write incoming EFs into their associated 32 kbyte deep FIFO memories[3]. The Sub Event Merger logic (SEM) reads EFs from the two FIFOs, using a fast PLD logic to perform the time-critical synchronization of block data readout from the FiFo via a single command from the SEM logic. The SEM identifies EF pairs that share the same event number according to an input algorithm like shown in 5.1.1 . A combined SE is written into the DB section of the SEB buffer together with a pointer to its DEB directory section (see also 7) .

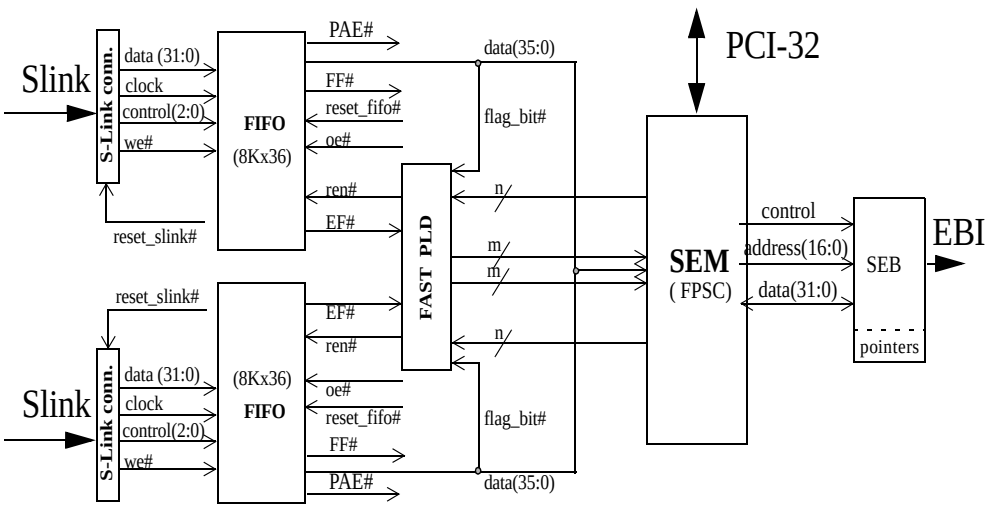


Figure 11 RU Input stage (one of two dual stages)

The FiFo flags of the input FiFos are as follows:

EF= empty flag FF = FiFo full flag PAE= almost empty programmable flag PAF= almost full programmable flag.

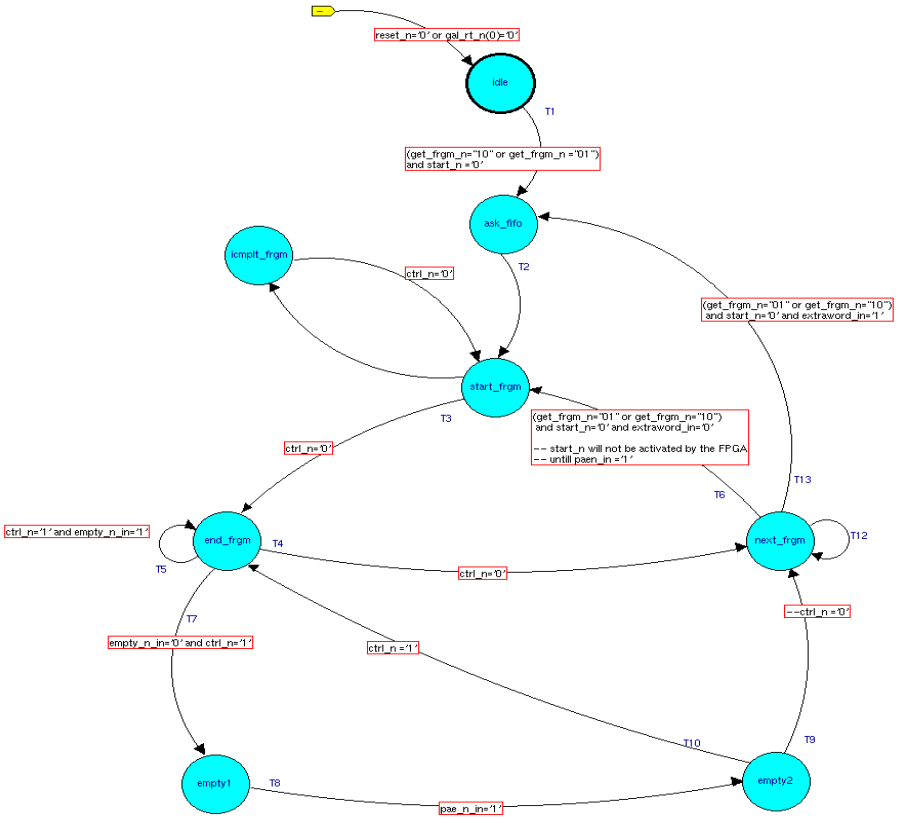
The programming of 4 offset values (7,15,31 and 1023 words) of the PAF and PAE assertion is via 2 lines FS0 and FS1 from the SEM FPSC. Their state is programmed at a reset-FiFo command from the SEM logic.

In more detail: Each S-Link input writes data directly into a separate FIFO according to the S-Link protocol (see 8.2). The value of three S-Link signals (**ldown#**, **lderr#**, **lctrl#**) are written into the 36-bit-wide FIFO together with the 32 data bits coming from the S-Link connector and the **FF#** (fifo full) flag bit coming from the FIFO. This additional information is used by the

1. Intel 21154-BC
2. 32 bit wide by default, 16 bit option via Jumper ST5
3. IDT 72V3670

PLD for frame synchronization and delimiting (**lctrl#** signal) and by the FPGA for error detection (**ldown#**, **lderr#** and **FF#** signals).The FPGA monitors the state of the **PAE#** and **EF#** signals (Programmable Almost Empty and Empty Flag, respectively) in both FIFOs. Once the FPGA determines from the **PAE#** or **EF#** flags that it can start reading a new event fragment, it reads the corresponding FIFO with the help of the fast PLD..

Figure 12 PLD state diagram



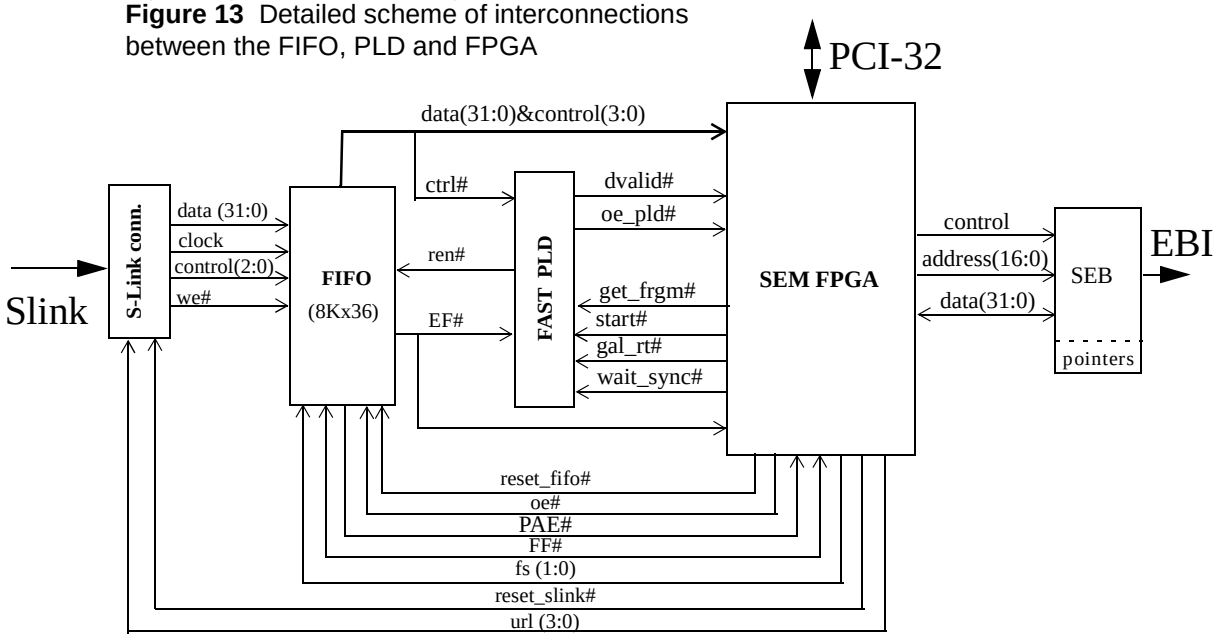
PLD state operation

1. Wait for the FPGA to assert **get_frgm#** and **start#** (***idle*** state in the diagram)
2. Read selected FIFO until a start of frame is found (states ***ask_fifo***, ***start_frgm*** and ***icmplt_frgm***)
3. Read a complete event fragment from FIFO and validate the data with **dvalid#** (state ***end_frgm***)
4. If FiFo gets empty during step 3, insert wait states (***empty1***, ***empty2***)

Go to step 2 (and thus read a new event fragment) when the FPGA needs it (state ***next_frgm***)

The PLD is needed because the FPGA is not fast enough to deassert **ren#** fifo input (read enable) as soon as an end of frame has been reached and thus an extra unwanted word (the beginning of next event fragment) is read from FIFO.

Figure 13 Detailed scheme of interconnections between the FIFO, PLD and FPGA



The programmable PLD may be used for something else than deasserting **ren#** after an end of frame or a FIFO empty condition. For this purpose, a number of lines are connected between the FPGA, the PLD and each FIFO (Figure 13):

- **ctrl#** is an S-Link dedicated signal used as flagbit for frame delimiting (i.e Slink control words in header and trailor of STF, see Figure 7)
- FIFO read control signals: **oe#** and **ren#** (output and read enable)
- FIFO flags: **EF#**, **PAE#**, **FF#** (empty, programmable almost empty ans full flags)
- PAE# level selection: **fs(1:0)**
- S-Link defined control signals (**url(3:0)** and **reset_slink#**)
- **get_frgm#**: the FPGA asks the PLD to read a new event fragment from FIFO
- **start#**: the FPGA enables the PLD operation after reset or power up
- **gal_rt#**: the FPGA resets the PLD logic
- **wait_sync#**: provides a way to read the otehr FIFO when one of them gets empty in the middle of a fragment read. Not used in the current version of the PLD logic
- **dvalid#**: the PLD validates FIFO data with this signal
- **oe_pld#**: used by the FPGA to drive **oe#** for the FIFOs

The currently implemented 8-state state machine implemented in the PLD is shown in figure Figure 12 and its operation is summarized in the textbox

5.1.1 Input algorithm

The SEM readout alogrithm is depicted in Figure 14, it is divided into three tasks (dashed boxes in the figure)

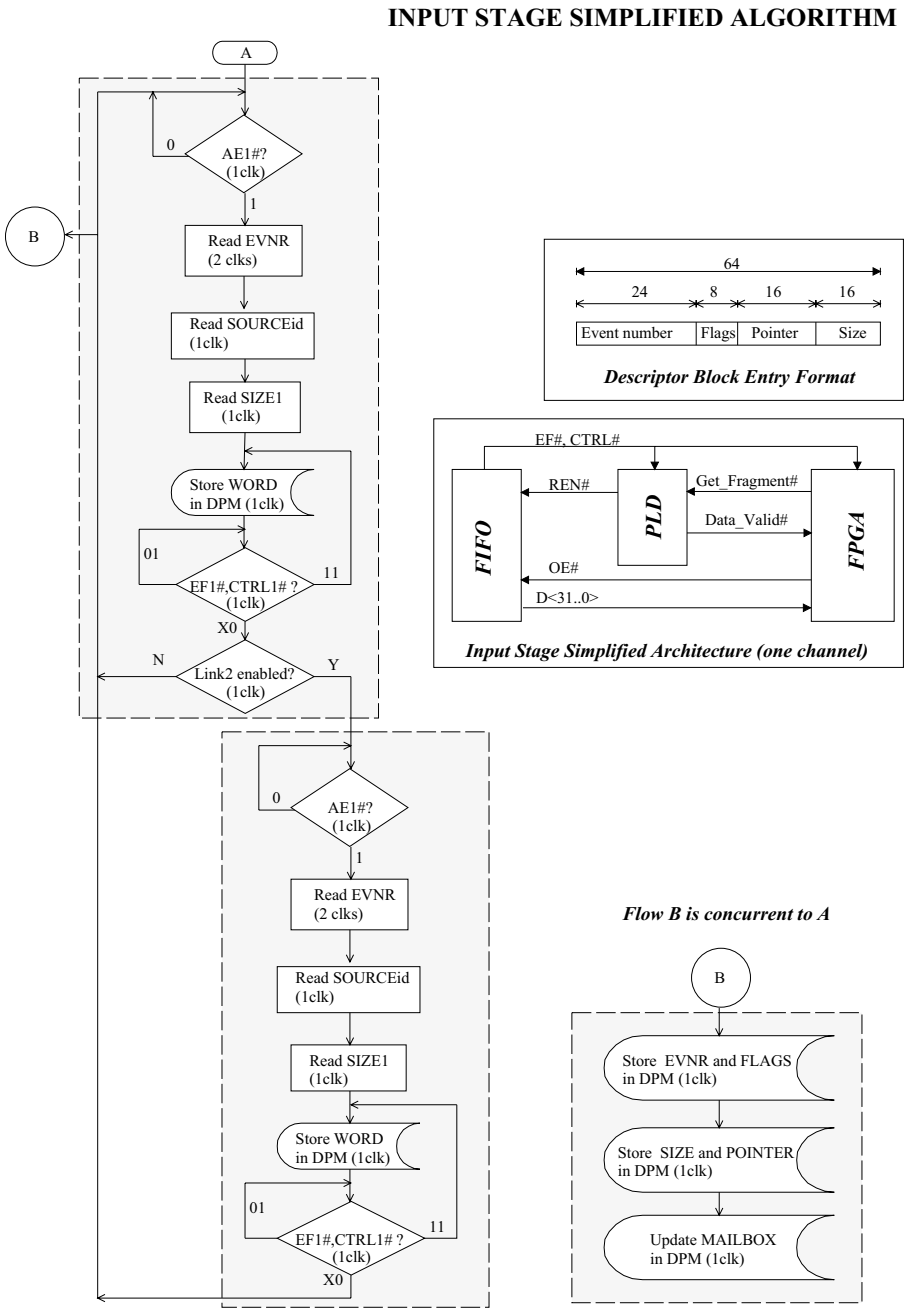


Figure 14 Input stage algorithm

1. Read one sub-event from first FIFO and store its data block in the SEB.

2. Read one sub-event from second FIFO and store its data block in the SEB so that the two data blocks are merged (2:1 merging in the input stage).

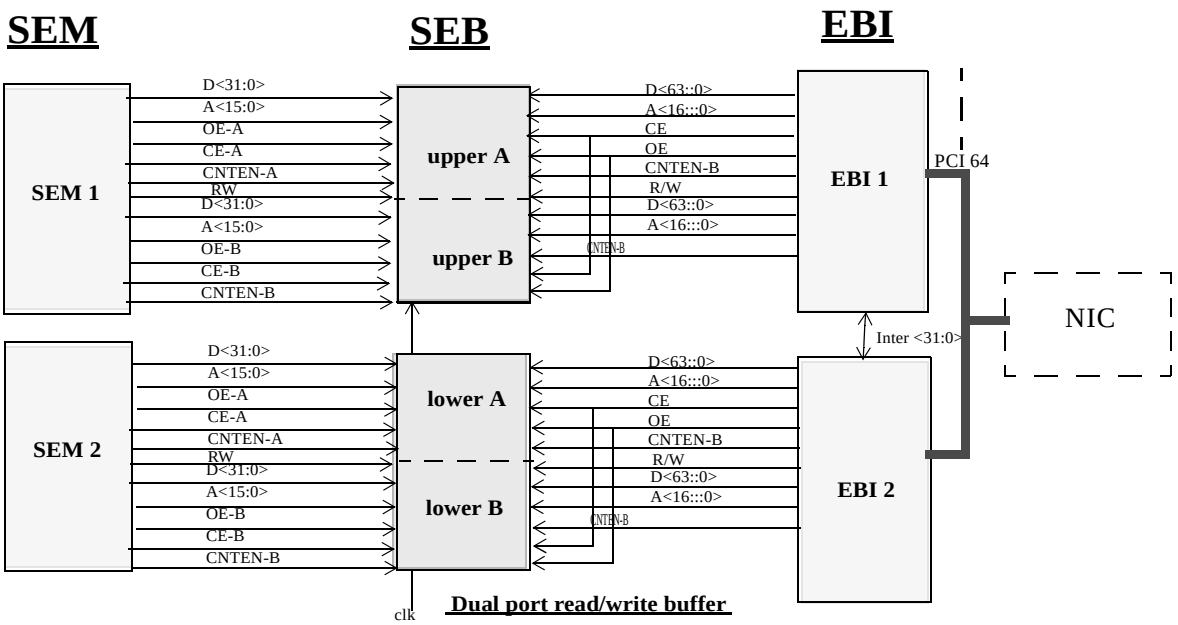
3. Write a single entry for the merged data block in the Descriptor Entry Block (DEB).

Task 3 is concurrent with task 1, resulting in zero overhead for memory management and 2:1 data merging (except for the turnaround cycle needed when switching between FIFOs).

5.2 Sub-event buffer (SEB)

The two SEB's memories are physically dual port memories between the SEM logic and the Eventbuilder Interface logic (EBI). Most of their capacity is used for storing SE data; only a small part is used as a DS pointer directory. Each SEB bank consists of two 32-bit[1]-wide memory chips, connected for 32-bit accesses from the SEM and 64-bit accesses from the EBI side (Table 4). The SEB capacity can be chosen between 0.5 to 2 Mbyte. There is read / write random access from both sides.

Figure 15 SEB dual port memory connection scheme



The photo below shows the SEB implemented in the 0.5 Mbyte option

Figure 16 Photo of SEB

There are physically 2 chips per bank, i.e in total 4 DPM chips. 3 options:

- 2 MB SEB total: IDT70V3599 128K * 32 (36) bit
- 1 MB SEM total: IDT70V3589 64K * 32 (36) bit
- 0.5 MB SEM total: IDT70V3579 32K * 32 (36) bit

The chosen device is a 36-bit wide DPM: IDT 70V35x9S in a 208 PQFP package [ref to IDT ICs], where "x" stands for 7, 8 or 9 (128 kbytes, 256 kbytes or 512 kbytes respectively). This yields a maximum 2 MByte dual-port memory. These three devices are pin compatible, but the smaller one does not include a JTAG interface. The first RU-II modules are equipped with the smallest device like shown in the photo.

1. 36 bit wide chips are used on a 32 bit data bus

The SEM logic performs 32-bit interleaved operations to memory so that two consecutive memory positions are stored in different chips. The EBI logic performs 64-bit operations reading both DPM chips simultaneously. This is a straightforward mechanism for interfacing the 32-bit input and 64-bit data output buses. A self-incrementing address counter in the SEM memory chip is loaded and enabled independently for each port via the ADS# (Address Strobe) and CNTEN# (CouNTer ENable) signals. Even if normal operation requires only write operations in the left port (input stage) and read operations from the right port (output stage), bidirectional buses and R/W# control lines are provided for special applications like debugging or test.

5.3 Output stage (EBI = Event Builder Interface)

The 64 bit output stage of the RU is shown in Figure 17. Two FPSC chips can access their

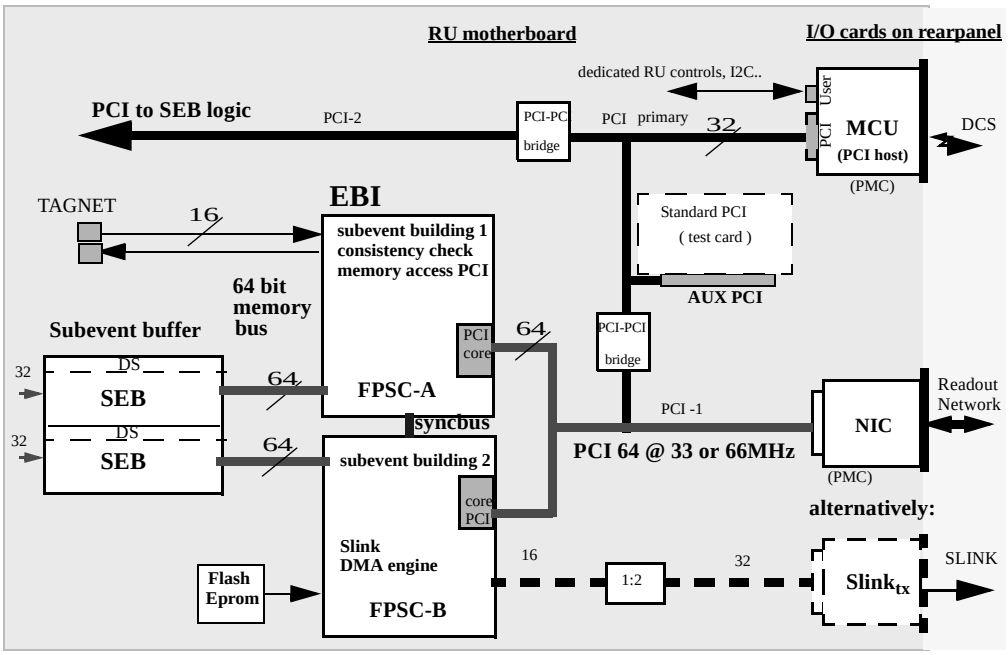


Figure 17 RU output stage

The 64 bit output stage consist of 2 SEB buffers connected to 2 EBI (event builder interface) blocks. These operate in parallel and are connected together at their outputs via the 64 bit PCI bus on which also the NIC card is connected. A 32 bit syncbus between the 2 FPGAs provides for their synchronisation. For Slink applications, the syncbus is used to combine subevents into a single Slink output.

associated SEB buffers via a 64-bit data buses in parallel. The FPSC's internal PCI cores allow to combine the PCI outputs together on the PCI bus segment 1 as required for the tandem master operation of the L1-VELO application. The PCI-64 high speed data path between FPSC and NIC is decoupled by PCI bridge from the other PCI bus segments which are mainly used for initialization and communication. The TAGnet connector 16-bit in / 16-bit out for L1 high rate traffic scheduling[1] is implemented in logic inside one FPSC. The Slink or NIC cards share (exclusively) same PMC slot. For an Slink output, the SE data are combined, using the 32 bit sync-bus interconnecting the two FPSC's and sent to the Slink connectors as a single subevent. A 16-to-32 bit bus converter is required for Slink due to not enough pins on the FPSC. This is however uncritical as the performance requirement for a link multiplexer is four times less than for a RU.

1. or any other programmable purpose

6 Output network / link options

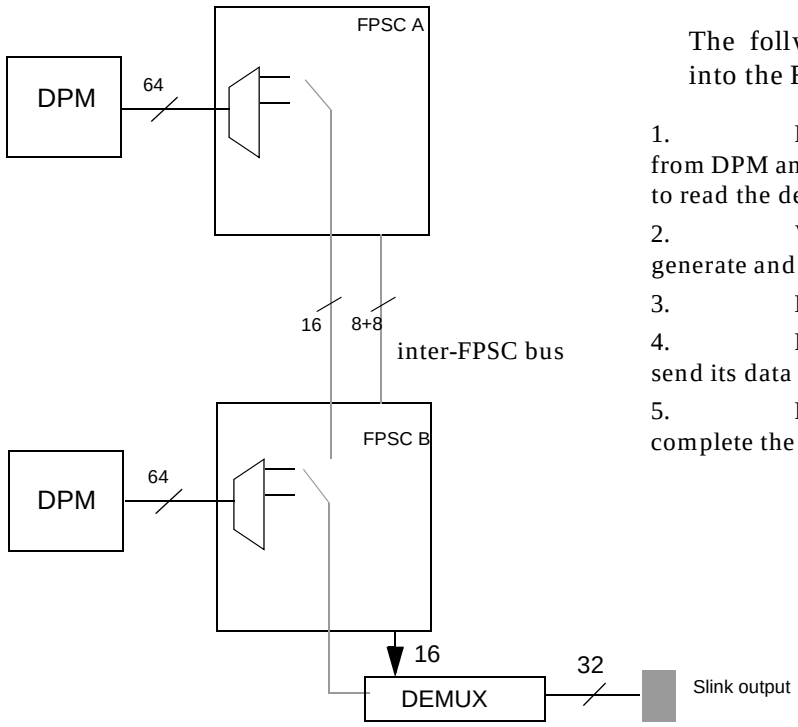
Each RU application has a different output interface and correspondingly a different output transmission scheme. The FEM uses a FiFo-like output towards Slink, the DAQ uses a NIC as PCI master combining subevents, and the VELO-L1 uses tandem DMA engines in the output FPGAs, operating as PCI masters towards a shared memory interface.

6.1 Output to Slink (FEM)

The output to Slink uses a 16-bit data path as shown in Figure 17. The FPSC-B implements the sub-event building process described in the figure caption of Figure 18

For this puropose, the dedicated 32-bit bus that interconnects the two EBI FPSCs is used for transferring 16-bit data from FPSC-A to FPSC-B, together with control signals (8 lines in each direction) between the two FPSCs required to synchronize the subevent building .

Figure 18 Slink output data path for FEM application



The follwoing procedure is programme into the FPGAs:

1. FPGA-B reads the next SE descriptor from DPM and asks FPSC-A via the inter-FPSC bu to read the descriptor and send it to FPSC-B
2. With this information, FPGA-B can generate and send the header words to S-Link
3. FPSC-B sends its data block to S-Lin
4. FPSC-B asks FPGA-A to read and send its data block
5. FPSC-B adds the trailer word to complete the frame, according to the STF

6.2 Output to a PCI Network Interface card (DAQ and Trigger)

The EBI logic in the output stage scans the directory in the SEB buffer to identify the locations of sub-events in the two SEB buffers that belong together. Using simple STF rules (see 4.1 ,) the EBI logic generates the sub-event format of the new, combined sub-event. The 64 bit PCI bus is used to generate a single, STF-framed sub-event burst for the Network Interface Card (NIC). There are several possible sub-event transfer protocols between the EBI logic and the NIC:

a. A "Push" protocol, in which the dual EBI logic reads, buffers and forwards sub-event blocks, as PCI initiator into the memory area of the NIC. Sub-event building, i.e. formatting of a single SE, is performed "on the fly". Such 'source initiated' transfers are controlled by "DMA write engines" within the EBI which only need a minimal synchronization protocol between NIC and EBI logic.

b. A "Pull" protocol where a "DMA read engine" is implemented in the NIC card for mastering of PCI read bursts. The NIC card can directly access sub-event data in the SEB buffers (via PCI memory mapping of the SEB buffers through the EBI). The assembly of fragments into single sub-events and their transmission is part of an 'intelligent' NIC card logic. This is slower, however simplifies the EBI logic to target-only operation and allows the full event-building protocol to be handled exclusively by the NIC card.

c. A third (slow) possibility exists for application development, where the MCU implements all, or part, of the sub-event building process in software.

6.2.1 DAQ readout Network interface

The preferrred DAQ readout protocol is a pull protocol to allow using a NIC card with all Eventbuilding handled inside the NIC. An alternative is to use a DMA engine engine in the EBI transmitting to the NIC buffer, however this might conflict with the eventbuilding protocols existing between the NIC logic and the readout network.

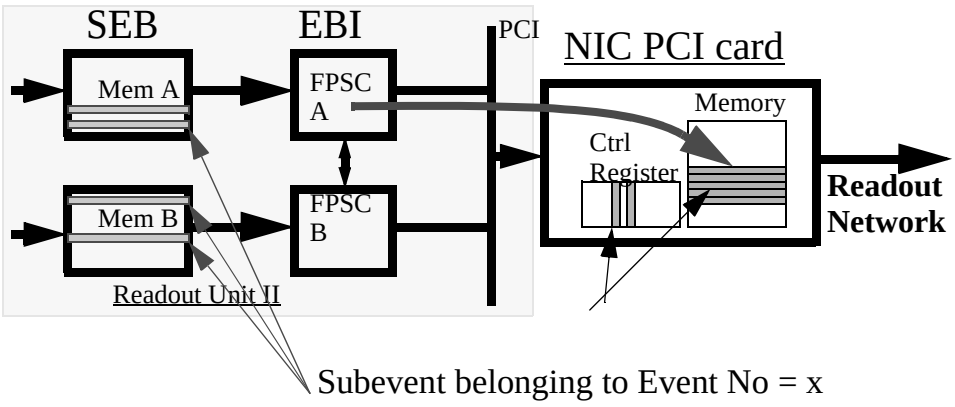


Figure 19 EBI-NIC protocol

- EBI tests (as PCI master) the NIC busy bit in the NIC control register
- If not busy, EBI writes Subevent descriptor addresses to the NIC
- EBI writes the Go bit in the NIC control register
- NIC sets busy and reads memory A +B via PCI. (Result: 1 subevent in NIC buffer)
- NIC re-creates SE header and trailor
- NIC transmits SE + resets busy bit

Figure 19 shows how a (NIC) interfaces the PCI output bus to a Readout Network. Such cards have PCI master capability and hence are capable of transferring data from the PCI bus into their internal output buffer. The access to subevents in the SEB buffer requires that a SEB memory accesss logic is implemented in the FPSCs. The event-readout from the SEB is entirely performed by the NIC card with a synchronization convention over PCI shown the Figure caption of Figure 19 . For this purpose the NIC has to implement a control register and a descriptor register.

6.2.2 High bandwidth L1-trigger output

A tandem PCI master operation with alternating DMA engines [18] in the output stage is required for the highest output performance with PCI network cards, capable of at least 200 Mbyte / s (such as SCI interface cards).

One EBI fills its internal PCI core FIFO with data while the second FPSC is flushing its FIFO to the PCI bus to the NIC card. In order to write a contiguous SE into the NIC memory, the FPSC that writes the first part of the sub-event gives a pointer to the other FPSC indicating the next free address in the NIC PCI-mapped memory (A1 in Figure 20) .

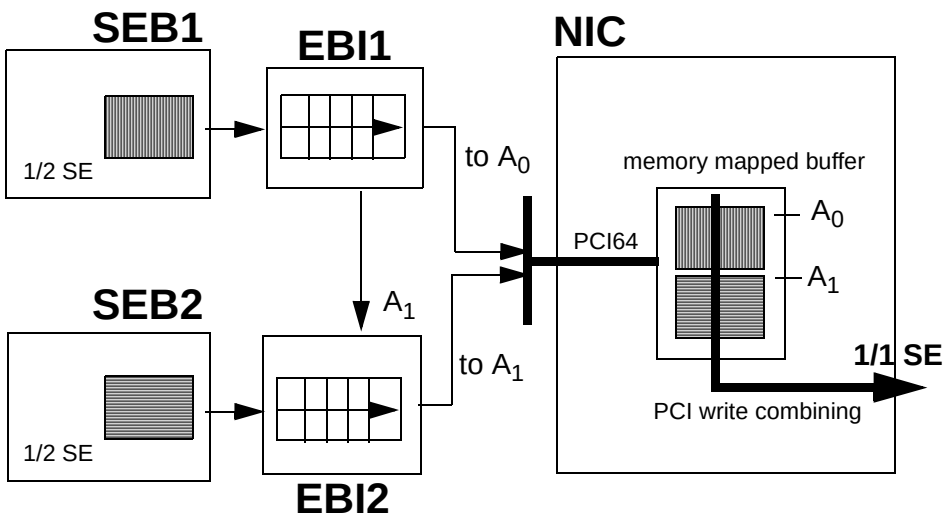


Figure 20 Tandem PCI masters

SE building using PCI's write combining feature and FPGA tandem masters on PCI. The output FiFos in each EBI are loaded with 1 / 2 SEBs. EBI1 sends the PCI address A1 to EBI2 and sends its own SE to A0. EBI2 sends immediately after to A1. PCI write combining makes a single SE burst.

The code for this type of operation has been implemented and tested by the Level-1 Topology Trigger project with SCI cards as NIC achieving up to 1.15 MHz output [17].

7 Buffer management and throttling

The push architecture in the LHCb experiment assumes that under normal running conditions, there is always enough buffer space in the destination buffers, hence also in the Readout Units. The overflow protection of all buffers is applied via a common OR (throttle), which is returned to the trigger supervisor when any buffer tends to overflow.

7.1 Throttling

The fill state of the SEB's in the RU is monitored by buffer monitoring logic in the SEM, which transmits a programmable, single-line 'throttle' signal[1] via a frontpanel coaxial connector backwards to the trigger supervisor, requesting to reduce the trigger rate. The latencies involved for the trigger throttling to have effect in less inut data determine the low and high "watermarks" for the Xon / Xoff logic (Figure 21.)

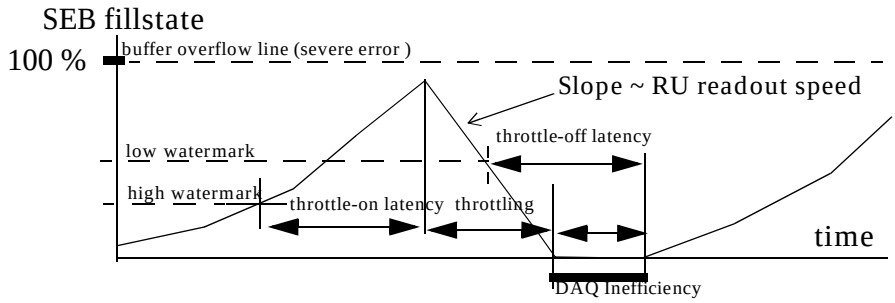


Figure 21 Throttling

When the fillstate reaches "low mark" the SEB continues to fill during the throttle latency and then drops at the speed of the readout. If the "high-mark" is set too low as in this example, it results in an empty state of the SEB (inefficiency of the DAQ system)

The throttle output is available on the front panel as TTL wired OR signal. Thus several RU throttle outputs can be connected together to produce a common throttle signal as shown in Figure 22

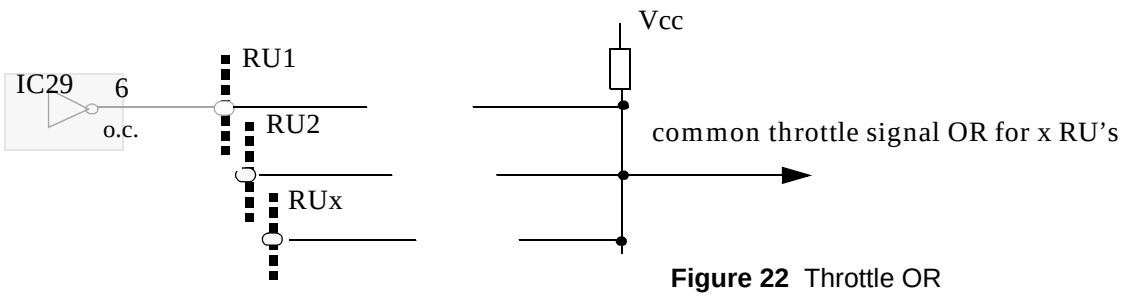


Figure 22 Throttle OR

1. open collector TTL for ease of oring all RU's in a crate via wired or.

7.2 SEB Memory management

Each DPM bank is logically divided into two blocks: a Directory Entry Block (DEB) and a Data Block (DB). There is also a special memory position used for buffer occupancy monitoring called MAILBOX. Due to the high cost and low level of integration of the dual-port memory compared to single-port DRAM memory, an efficient memory usage is required. This discards fixed-length, fixed-position buffers for storing event fragments. By means of directory entries with pointers in the DEB, non-fixed-length non-fixed-position buffers are possible in the DB with economic memory usage. An example for a directory entry is shown in Figure 23.

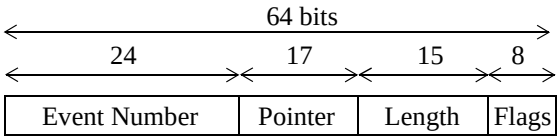


Figure 23 Example of directory entry in DEB

In this example, a directory entry consists of a single 64-bit word containing the following fields:

- **Event Number**: Despite this field is 24-bit wide, a full 28-bit event number can be stored if the directory entry address corresponds to the *n* least significant bits of the event number, where *n* is $log_2(Memory\_Size/Fragment\_Size)$ and thus depends on the application. In the first code implementation a fragment is assigned the next free entry in the DEB, with no relation to its event number in order to accommodate a descriptor entry in 64 bits. DAQ event numbers are limited to 24-bit wide. If this is considered insufficient, 128-bit descriptor entries would have to be used
- **Pointer**: Address of the start of the event fragment in the DB
- **Length**: Size of the fragment in in 32-bit words. In this example (15-bit field), a maximum 128 Kbyte fragment is allowed
- **Flags**: Includes errors reported by S-Link, by the data source and other kind of error specific to the RU. It also indicates if an Error Block from the STF trailor is stored in the DB following the fragment

Using this directory entry structure as an example, the memory looks like shown in Figure 24. The final implementation can be a different one and is obviously independent of the RU hardware.

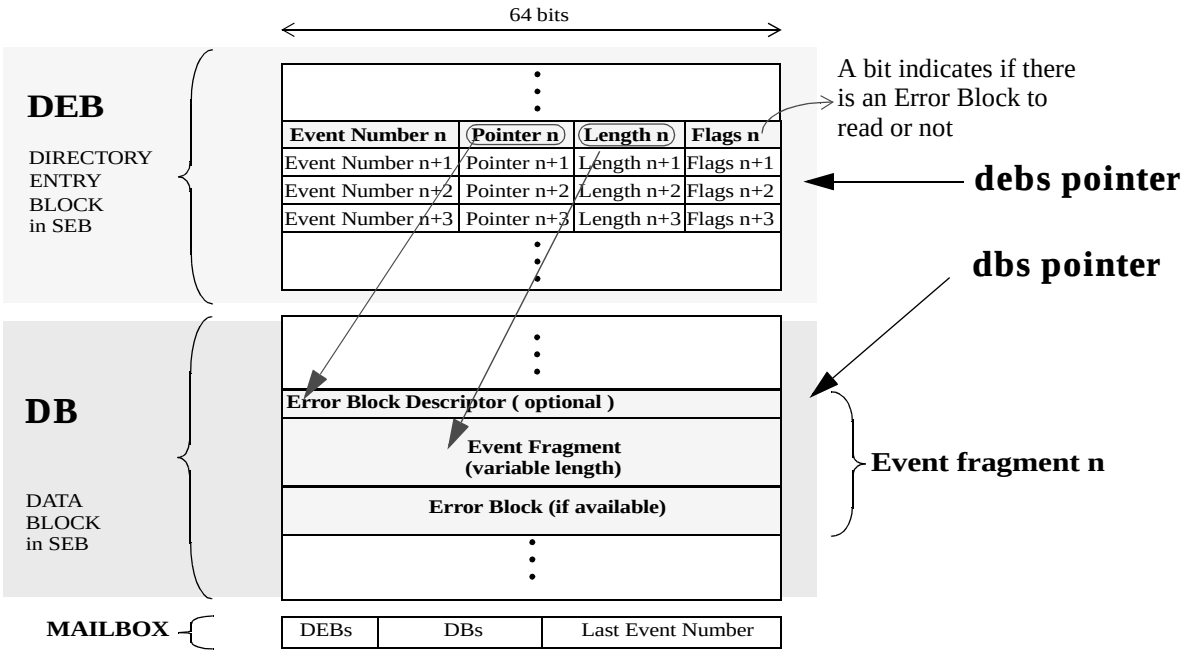


Figure 24 Event fragment storage in SEB memory

7.2.1 Buffer occupancy monitoring

The SEM logic keeps two pointers to the next free position in the DB and in the DEB (pointers **dbs** and **debs** respectively, where “s” stands for SEM). Both are pointers to circular buffers.

1,) When a new event fragment is to be written to the DB, it is stored starting in the current value of **dbs**, after storage **dbs** points to the next free position in the DB.

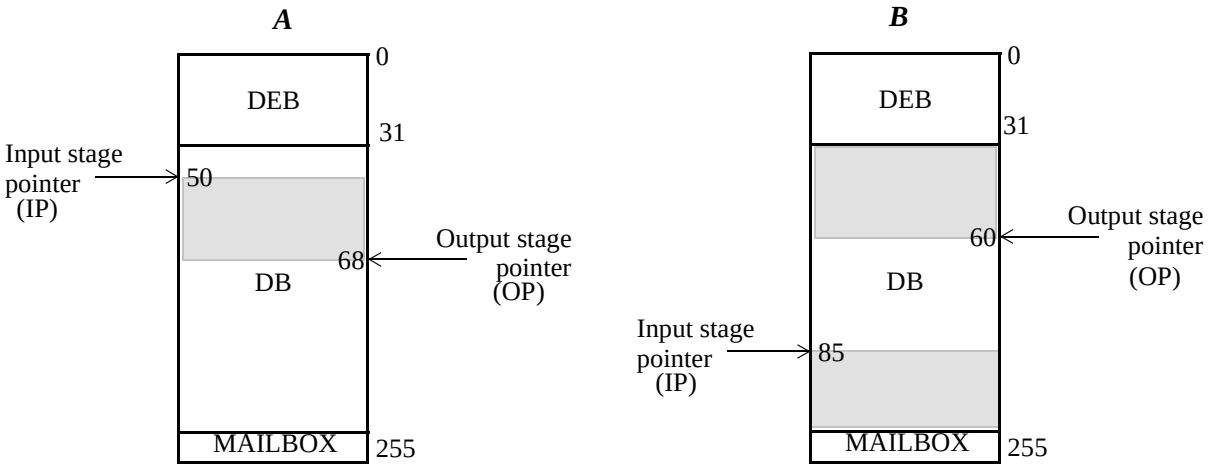
2.) The corresponding directory entry in the DEB is written to the address pointed by **debs**, which is incremented afterwards.

3.) The SEM logic writes **dbs**, **debs** and the event number in the **MAILBOX** (a dedicated memory position) and toggles a I/O line (also called MAILBOX) to warn the corresponding EBI logic in the output stage that a new subevent has been written into memory.

The EBI logic also keeps two pointers: **dbe** and **debe** for the next locations to read from DB and DEB, which are treated as circular buffers by the EBI logic. Each time the MAILBOX signal toggles, the EBI reads the **MAILBOX** and recalculates the SEB buffer occupancy. Above a certain programmable low-water threshold [see 5.3] the signal DPMFULL# is asserted to warn the SEM of this condition. If DPMFULL# or FIFO almost full conditions are detected by the SEM logic, the fast throttle signal is asserted at the RU's frontpanel connector [see chapt. Figure 59.]

In more detail, the EBI logic has to monitor the SEB occupancy in order to detect the following conditions: empty, full beyond the "low water" threshold and completely full. As both DEB and DB are circular buffers, the input stage pointers can be greater, equal o lower than the output stage pointers. An example is depicted in Figure 25, for a SEB total size of 256 postions. The DEB size is 32 and DB size is 256-32-1=223. The pointers "point" to the next position to be read (OP) and the next position to be written (IP).

Figure 25 Buffer occupancy monitoring



In case A, $OP > IP$. The free space in the DB corresponds to the shaded area and can be computed as: $OP - IP = 18$.

In case B, $IP > OP$ thus the free space in DB (shaded area) can be computed as:

$$256 - 33 - (IP - OP) = (256 + OP - IP) - 33 = 201$$

Additions and substractions are performed in two's complement. For N-bit wide addresses (thus memory size is 2^N), operands are 10 bit wide not taking into account the carry bit, the expression $(256 + OP - IP)$ when $IP > OP$ equals $OP - IP$. Then, $OP - IP$ can be computed for both cases and, if $IP > OP$ (which is detected by the carry bit resulting of $OP - IP$), 33 in this example, is substracted from the result.
In the case of the DEB, as its size is a power of two, there is no need to perform any subtraction and the free space in DEB is always $OP - IP$

The described mechanism allows to calculate the free memory in the circular buffers in an easy way. An example of VHDL implementation is shown below. As the carry bit is not accessible, two's complement operations are performed with N+1 bits and the MSB is used as the sign bit.

```
architecture simple of count_blk is

  signal dbp_int,free_dbp_int:std_logic_vector(14
downto 0);
  signal mailbox_p:std_logic_vector(8 downto 0);
  signal mailbox_d:std_logic_vector(14 downto 0);
  signal pbp_int,free_pbp_int: std_logic_vector(8
downto 0);
...
begin
...
-----------------------------------------------
substractors:process(reset_counters,clk)
  variable free_dbp1_int:std_logic_vector(15 downto
0);
  variable free_dbp2_int:std_logic_vector(14 downto
0);
begin
...
    free_pbp_int<=pbp_int-mailbox_p;
    if(free_dbp1_int(15)='1') then
      free_dbp_int<=free_dbp2_int;
    else
      free_dbp_int<=free_dbp1_int(14 downto 0);
    end if;
    free_dbp2_int:=free_dbp1_int(14 downto
0)-"1000000001"; --513
    free_dbp1_int:=('0'&dbp_int)-('0'&mailbox_d);
...
end process substractors;
-----------------------------------------------
...
end count_blk;
```

Figure 26 VHDL code for buffer fillstate monitoring

8 Data and Control buses & links

There are a variety of data and control buses used on the RU: PCI, I2C, JTAG, SLINK, TAGNET which are described in this chapter.

8.1 PCI bus on the RU

The PCI bus on the RU motherboard serves a variety of purposes

- loading of application bitfiles to individual FPSC's via the MCU (see Appendix 14.0.1)
- Control & Monitoring of RU operation via LAN from a DCS server
- Access to Control Registers inside FPSC applications
- Initialization of NIC card with standard software drivers
- main data output interface bus to the RN (Readout Network)
- Diagnostics & Development (PCI analyzer , generator, PCI utilities) (Appendix 14.0.2)

PCI [19] is the most used and best supported existing boardlevel bus. The PCI is a hierarchical system, requiring a host system on the root bus to configure the configuration registers (Table 3) of all connected devices. Bandwidth options for 32 bit@ 33MHz, 64 bit @ 33 MHz, 64 bit @ 66 MHz and 32 bit @ 66 MHz exist. All four options are supported by the PCI core of the FPSC chips used on the RU.

The PCI bus architecture of the RU is depicted in Figure 27. It is implemented orthogonal to the data flow. The clock rate of the PCI host system in the PCI root segment defines the PCI clock rate the subordinate PCI segments. It is selectable (via jumper ST6) to operate at 33 or 66 MHz[1].

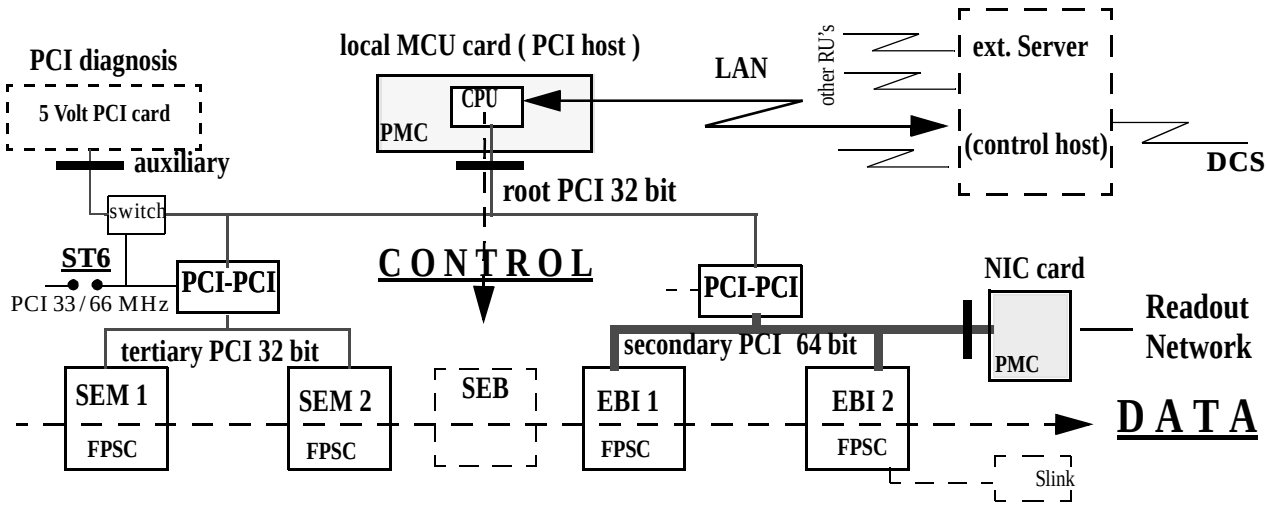


Figure 27 PCI bus hierarchy

The connector pinouts for the two PMC cards (MCU, NIC) are listen in chapter 1

The root segment and the tertiary segments are 32 bit wide, the secondary PCI segment is implemented as 64/32 bit bus, since this segment is transferring high bandwidth data between the EBI and the NIC logic. The standard PCI option corresponds to 132 Mbyte/s of theoretical bandwidth, in practise ca. 70% are reached by source initiated (write) transactions of sufficently long block length. Reading from PCI has large bandwidth penalties resulting in efficiencies as low as 5% and maximum 40% [21]. There are 4 default FPSc devices and 2 PCI bridges. Any other added device, like a NIC card, requires proper configuration before being usable as a PCI device. The standard way to initialize a PCI hierarchy is via the boot procedure of an operating system. An operating system like Linux on a processor card, plugged into the RU's root PMC slot, will automatically configure all bridges and devices.

1. The auxiliary connector is disabled for 66 MHz operation

PCI is electrically a very delicate, non-terminated bus, operating with voltage reflections on high-impedance transmission lines. Clock-skews, capacitive loading and strip-line length are highly important. The RU mainboard PCI bus layout was therefore verified for analog signal compliance (see Section 9.3, "Signal integrity simulations") with the PCI requirement, prior to PCB production.

8.1.1 PCI host cards (PPMCs) for the RU

The default host card for the PCI root segment on the RU is the MCU mezzanine [30] which includes the optional I2C, JTAG and PCI arbitration protocols on the optional user connector as defined by the standard. PPMCs which have not implemented these extra protocols may be used, since I2C is only required for factory programming of the clock chip, JTAG is an option, and the arbitration is only needed, if RU-resident PCI masters would request access to the root segment[1]. A list of commercial processor PMCs is given in Figure 28

Motorola

PrPMC600 32 bit33 MHz PCI, PPMC based on MPC8240 CPU, fast ethernet, RS232

PrPMC750 64 bit 66 MHz PCI., PPMC PowerPC based, Vita I/O connector, fast ethernet version has 1.5 width of PMC

PrPMC800, 64 bit 66 MHz PCI, PPMC high performance PowerPC, Vita I/O connector, fast ethernet, and RS232

www.powerbridge.de/daten (1)

Raytheon PowerPC 750

PPMC with 32 or 64 bit PCI connector. 33 MHz. **Linux support**

http://www.raytheon.com/c3i/c3iproducts/c3iccn/ccn01c2a.htm (2)

Artesyn Technologies PM/PMC

PowerPC™ 750 up to 458Mhz, 10BaseT-front panel or P14. 33 and 66 MHz PCI capable. PCI arbiter 33 Mhz. **Linux support**

http://www.artesyncp.com/html/pmppc.html (3)

Aeroflex

ACT-7000PPMC, networked RM7000 based PPMC with up to 66 MHz/ 64 bit PCI operation and P14 user connector

http://www.aeroflex.com/act/pdf/sd7000.pdf (4)

Transtek

PowerPC 750/7400 PMC module , 33 MHz PCI
http://www.transtech-dsp.com/powerpc/tm3.htm (5)

IC Interface

IC-405-PMCa based on 370MIPS, PCI monarch up to 6 arbiter lines, 32 bit PCI 66 MHz, ethernet and RS232, **LINUX**

http://www.interfaceconcept.com (6)

SBS Technologies

POLAMAR II, PPMC card PowerPC ,PCI 33 /66 MHz, P14 connector, RS232
http://www.sbs.com/communications/products (7)

Figure 28 some commercial PCI host cards for the RU

1. in normal data transmission mode, PCI mastership is only required on the secondary PCI bus which has its own arbiter.

PCI devices like the FPSC chips on the RU require a configuration process via the configuration registers. Exact details on this procedure are described in [19]. These configurations are automatically performed during the boot process of an operating system like Linux. Linux commands like "lspci"[1] allow viewing the details of the configuration registers of all devices. The configuration register definitions of the four FPSC chips on the RU are shown in Table 3 below.

Table 3 FPSC chip PCI configuration registers

byte 3	byte 2	byte 1	byte 0	offset
Lucent Device ID =0x11c1		**Vendor ID=0x5401**		00h
Status Register		Command Register		04h
Class Code			Revision ID	08h
BIST	Header Type	Latency Timer	cache line size	0Ch
Base Address Register 0 (i.e DPM)				10h
Base Address Register 1				14h
Base Address Register 2				18h
Base Address Register 3				1Ch
Base Address Register 4				20h
Base Address Register 5 (user registers)				24h
Cardbus CIS Pointer				28h
Subsystem ID		Subsystem Vendor ID		2Ch
Expansion ROM Base Address				30h
Reserved			Cap_Ptr	34h
Reserved				38h
Max_Lat	Min_Gnt	Interrupt Pin	Interrupt Line	3Ch
Reserved		FPGA config. Command-Status Register		40h
FPGA Configuration Data Register				44h
Scratch Register				48h
Reserved				4Ch
Reserved	HS_CSR	Next Item	Capability ID	50h
Reserved				54h Thru FFh

8.1.2 PCI access to SEB

Memory buffers like the SEB are implemented in the PCI memory space and hence their addresses and sizes are announced through the PCI base address registers during the startup of the operating system. In the Linux user mode, the recipe for getting a pointer to the FLIC memory is simply as follows:

1.open() device /dev/mem 2. mmap() system call to map into the process memory space.

With this pointer, a C routine can make use of library calls like memcpy(), memset(), memmove() etc. When using this method, no driver is required [21].

1. useful options are : "lspci -xx | more " or "lspci -vv" or "lspci -tv"

8.2 Slink receiver and transmitter interfaces

The CERN S-LINK [10] specification describes a common physical interface for 8, 16 or 32 bit wide asynchronous link technologies like Glink (serial high speed, long distance). As Slink has no knowledge of the encode / decode protocols interfaced to the SLINK connector, also cheap LVDS serializers like Channellink, as used for the RU [10] test are Slink technologies. SLINK includes error detection by block or by word, flow control (in the duplex version only), a self-test function and 5Volt or 3.3 Volt supply options. The SLINK signal convention is summarized in Figure 29 . As depicted here, the link encode / decode technology is

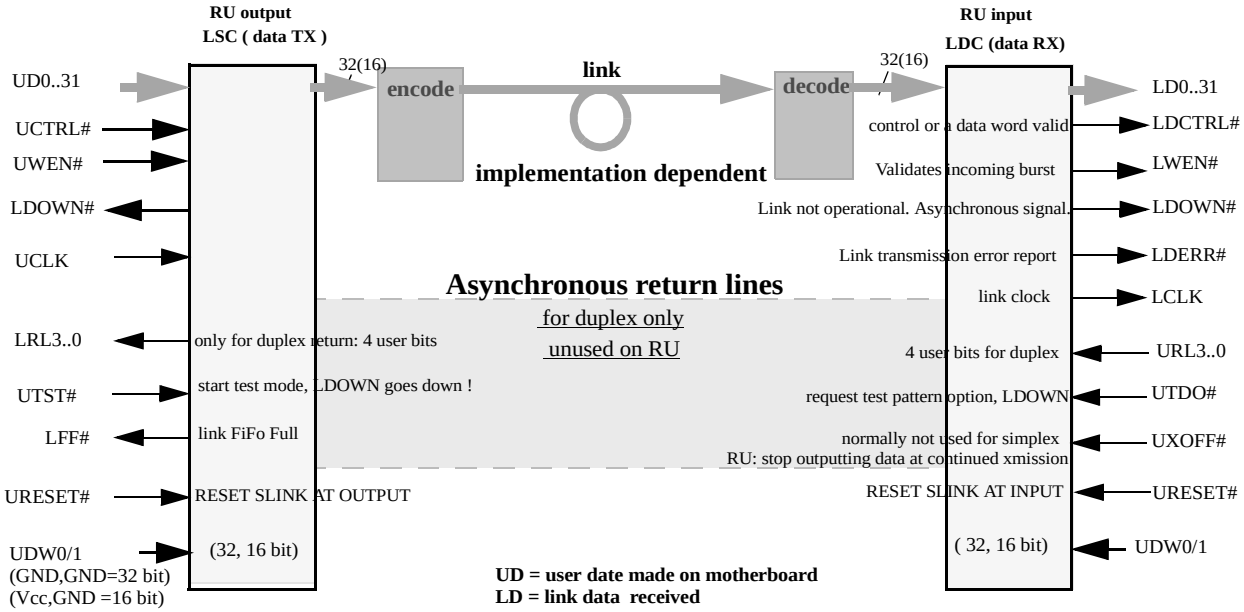


Figure 29 Slink receive and transmit signals

transparent to SLINK. The protocol can be understood as a parallel connector interface with no logical layers. The latter may have to be implemented for specific link technologies, like Gbit-ethernet, on an Slink card.

The signals sourced by an Slink carrier board are named with prefix U (user), the signals revceived by the link are prefixed L (link).

On the RU, the four SLINK input ports are by default configured as 32-bit links, i.e the bytecode UDW(O..1) is set to 0,0. The Jumper ST5 allows to set UDW0 to Vcc if 16 bit input is required. On the output of the RU, the Slink port is fixed to 32 bit.

There are consequently 16/32bit data lines on an SLINK interface, named at the transmitter side UD0..31 and on the receiver side LD0..31. The UWEN write enable line accompanies any transmitted block, consisting of control and data words. The example in Figure 30 shows the transmission of a RU data block, with control words in the header and the trailor. Control words are signalled via UCTRL=0. An LDERR signal on the receiver side would indicate

Figure 30 Slink simple example

detection of transmission errors, like parity. The corresponding STF representation is shown

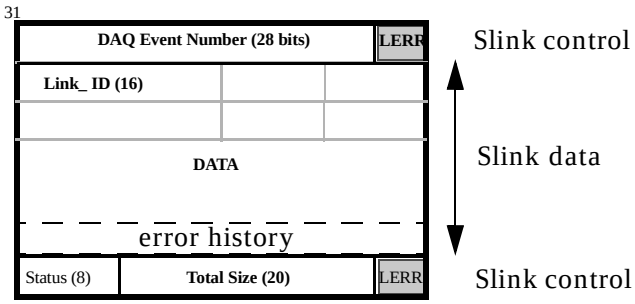


Figure 31 STF data block seen in an Slink context

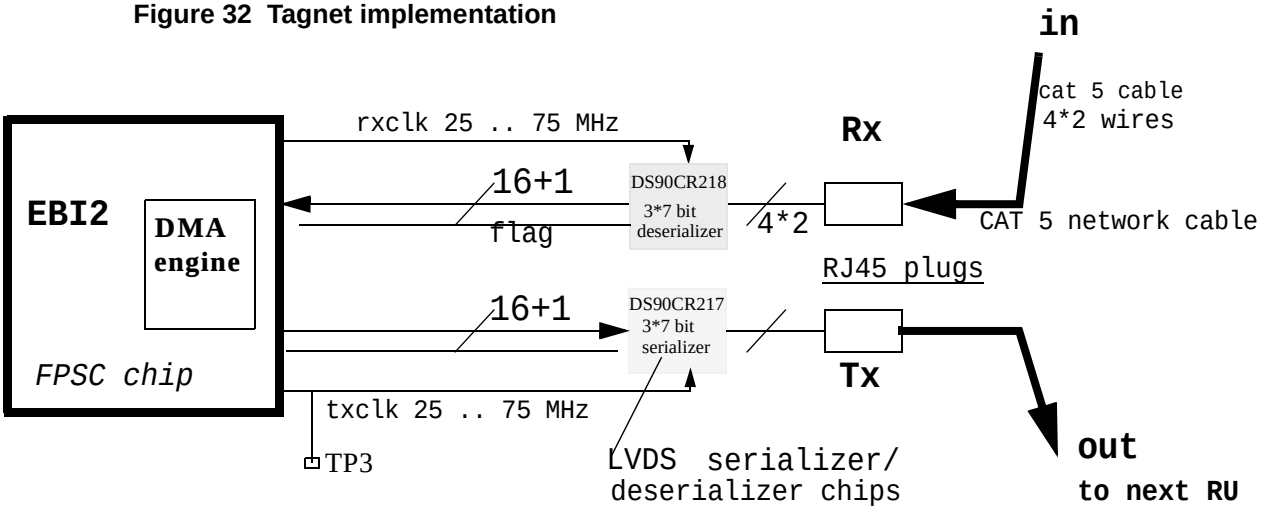
in Figure 31 .

A link clock can be transmitted via the UCLK input at the transmitter, received as LCLK at the receiver. A “link down” is signalled as LDOWN on both receiver and transmitter side by the Slink logic. The lower 4 bits in an SLINK control word are therefore reserved for reporting error codes, as defined by the specific link technology. On both receiver and transmitter side, the URESET allows to reset the local SLINK logic. The UTDO signal, in conjunction with the 4 duplex return user bits URL0..3 is foreseen on the RU for pattern tests with a local test card. (The assertion of UTDO has LDOWN as a consequence.) The UXOFF signal is unused on the RU since it is used only in duplex mode.

8.3 TAGNET line for traffic scheduling (or other purposes)

The Level-1 Velo application requires a traffic scheduling network [17] which was baptised TAGNET. This is a scalable traffic scheduling network, controlled by an external Tagnet

Figure 32 Tagnet implementation



controller which dispatches physical PCI destination addresses to the Readout Units. Free destination addresses circulate between RU's, using the Tagnet 16 bit input and output link which interconnect all RUs within a tagnet ring (see Figure 3).

Tagnet was implemented according to a preliminary definition (8.3.1) as 16+1 programmable input and 16+1 programmable output lines, connected to the FPSC chip EBI2 in the RU output stage (see Figure 32). The clock of up to 75 MHz is tranmitted across one of the four twisted pairs of a network cable. The 16 data bits are distributed over 3 twisted pairs of 7->1 LVDS serializers. The 17th bit is used as a delimiting Flag bit to avoid the need for control word delimiters. Only 3 bits of the third serializer is used. The Tagnet I/O connectors are routed to the FPGA and could hence be used for any user-defined variants of TAGNET.

For the VELO application, incoming and outgoing TAGNET links transmit the "DMA start" and the PCI destination address for a DMA engine to RU's which are interconnected in a TAGNET ring (see Section 3, "RU in L1-VELO trigger".) An accepted DMA start has as effect that a subevent is transmitted to a mapped, 64 bit PCI destination, which transports the data via the shared memory mechanism to the destination CPU in the SCI network [27] .

The TAGNET implementation on the RU was decided on a dedicated RU meeting [25] in October 2000 in Heidelberg and conseqeuently implemented using LVDS serializer / deserializer chips for transmitting 16bit data words, clocked at 40 MHz and framed by a 17th FLAG bit. Standard shielded CAT5 network cables up to a length of 5 m are used to interconnect RU's in a TAGNET ring.

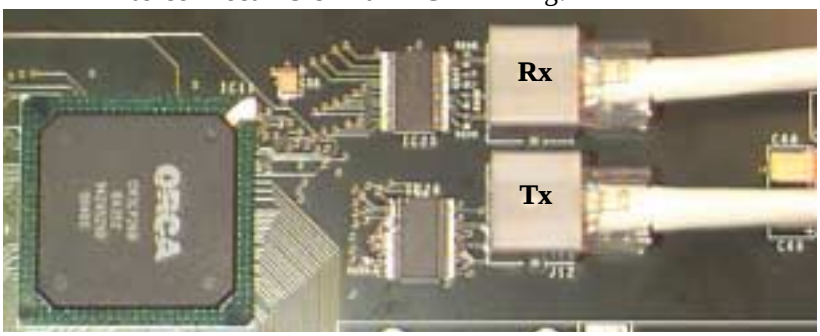


Figure 33 Photo of Tagnet plugs on RU

Standard RJ45 plugs are used. The Tx output must be connected to the Rx input of the next RU

The two, 8-pin RJ45 connectors are implemented as shwon in Figure 34, implying that non-crossed network cables with four twisted pairs must be used. There are 16+1 data bits sent over 3 pairs (Tx0,Tx1,Tx2) and one clock sent over 1 pair (CLK). The TX2 pair is used to only transmits 3 bits[1].

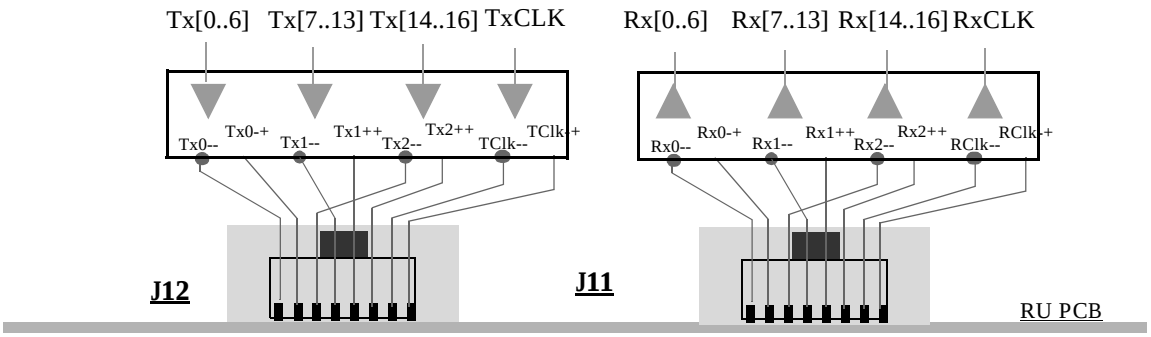


Figure 34 Tagnet I/O connectors usage

8.3.1 Tagnet format

Two possible TAGNET formats were proposed both of which include error correcting codes. Due to the vulnerability of a system to erraneous traffic scheduling information, TAGNET must be protected by self correcting Hamming code of at least a single bit. A maximum of 512 destination nodes is assumed.

The first proposal uses a single 16 bit frame as infomation unit, with 10 bit data for up to 1024 nodes. There are 2 bits for 4 commands and 4 bit for ECC.

1. The other 4 bits are also transmitted however unused and put to GND on the transmitter side.

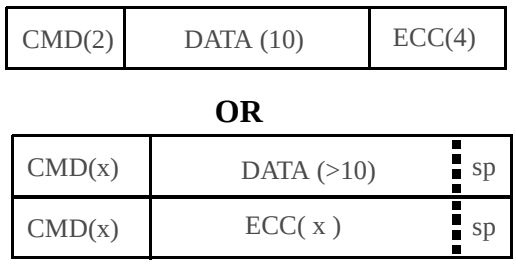
The second proposal uses double 16 bit frames as one information unit, allowing for more commands, and higher level ECC.

Tagnet 2 bit Commands:

- 00: SEND: 9 bit node-ID + 1 bit buffer-ID (= 10 data bits)
- 01: FLUSH: 10 bit event-ID
- 10: ERROR: 8 bit RU-ID (raw-Nr where error occurred) + 2 bit error-type

Figure 35 proposed Tagnet formats

16 bit frames



8.4 Auxiliary PCI bus connector on the root segment

A standard 5Volt PCI connector is used for the auiliary 32bit@33 PCI bus as shown in Figure 27. The use of this connector is foreseen for diagnostics via standard PCI cards (PCI tracers, PCI memory, Graphics cards etc). The PCI 33 MHz operation is selected via the ST6 jumper.

8.5 P14 control bus interface on root PMC slot (optional)

PCI hosting, initialization and remote control is meant to be implemented via standard PMC plugin card. The PCI bus connected to the PMC is the root PCI bus of the RU as shown in Figure 27. This emplacement includes an optional I/O connector (P14) for controls, such as programming the RU master clock chip via I2C. This interface connector P14 (Figure 50) of the root PMC has interface protocols for

- PCI bus arbitration of the two PCI bridges
- JTAG boundary scan chain
- I2C signals for the programmable clock
- Interrupt on frontpanel RU system reset
- System reset request by MCU

Some commercial PPMCs (see 8.1.1) do not have the P14 connector, or they use a different pinout and different functions. Therefore the use of the MCU card [14] designed for the RU is required when one of these functions is required for online purposes. The I2C function is normally only required to configure the RU permanently in the factory. Applications like the VELO-Level-1 do not need any of the available functions, hence a PPMC without such a connector is advisable.

8.5.1 Central arbitration for PCI root segment (option)

The central PCI arbiter for the root PCI segment must reside on the PCI host (i.e on the PMC) since usually a PCI monarch like the MCU has both internal and external PCI requesters[1] for the same root PCI bus. The request / grant signals from the 3 potential external devices on the RU are therefore routed through the I/O connector P14 to the central arbiter on the host MCU. This situation is depicted in Figure 36. The pinout of the 3 signal pairs for the two PCI bridges

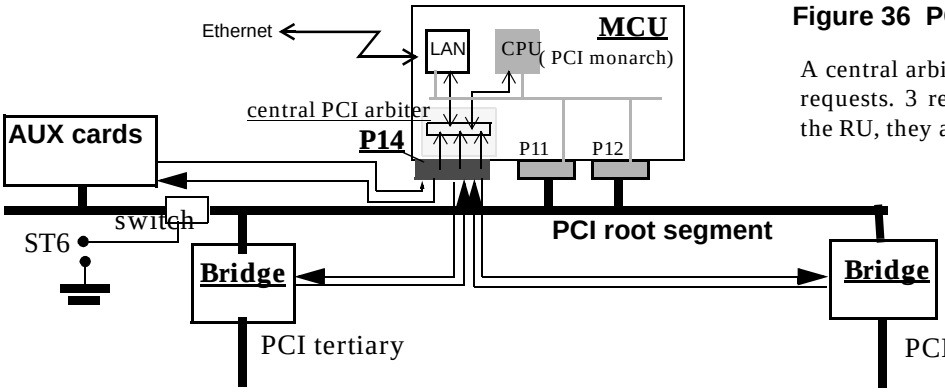


Figure 36 PCI root arbitration

A central arbiter needs to combine all requests. 3 requests may come from the RU, they are rooted through P12

and the auxiliary (diagnostic) card are shown in Table 4. The P14 connector to the arbiter is not required in normal RU applications where PCI bus mastership is only required on the secondary PCI segment (data output to NIC)

8.5.2 JTAG implementation (option)

The optional JTAG chain in the RU is controlled by the PMC I/O connector of the MCU, allowing remote configuration and test of the motherboard. The MCU card [14] designed for the PMC slot uses a special software [26] to acts as a JTAG master, allowing in this way a motherboard electrical test; device re-programability and read-back, for debug purposes.

1. like the ethernet controller

There are 8 JTAG devices in a JTAG chain as shown in Figure 37. The JTAG master is either

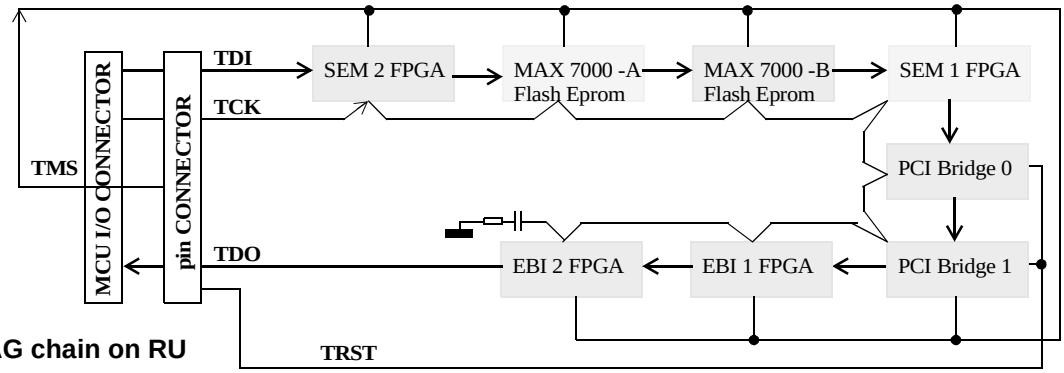


Figure 37 JTAG chain on RU

implemeted via the I/O connector of the root PMC or via JTAG connectors pins on the RU board. TMS and TCK are bused to all devices. TRST an optional signal, is not generated by the MCU. Thus, it is only available from the RU on-board JTAG connector and distributed to the PCI bridges.

The pinout of the 10 pin JTAG pin-post connector is shown in Figure 38

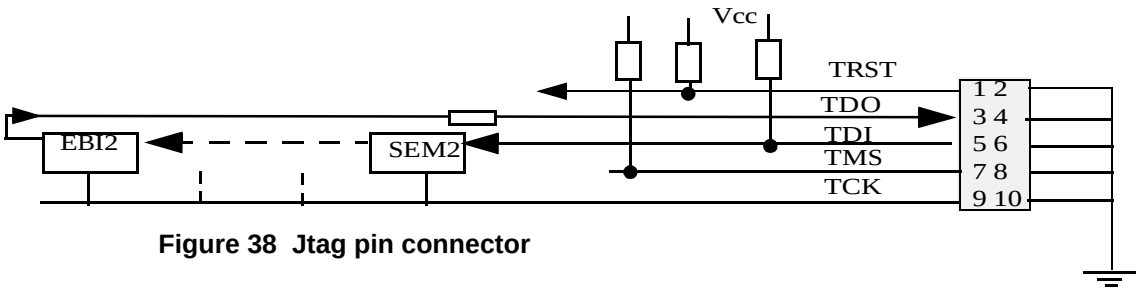


Figure 38 Jtag pin connector

In absence of an MCU card, the JTAG chain may be driven by an external master, like the JTAG Technologies tools[1], available at CERN."

1. we adopted the JTAG pinout as used by the pixel project compatible with Altera programmers

8.5.3 I2C interface for RU programmable clock domains (SEM), (EBI), (PCI)

The programming of the clock domains SEM, EBI and PCI is performed via the I2C interface available of the root PMCs I/O connector P14 (Table 9).

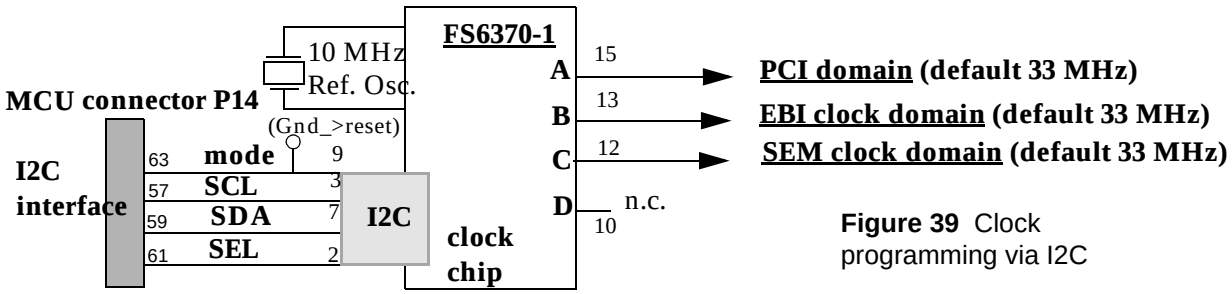
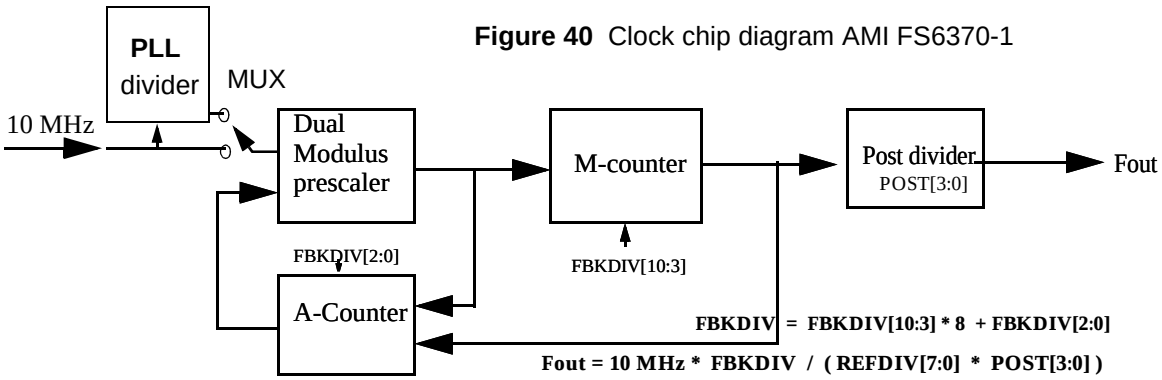


Figure 39 Clock programming via I2C

The AMI FS6370-01 programmable clock chip is driving the three clock domains of the RU as shown in Figure 39 for its outputs A,B,C. The D output is unused and may be used for any test clock generation. The reference clock frequency is a 10 MHz quartz from which all programmable frequencies can be obtained, for each domain independently. A permanent setting is programmed via I2C into the Flash register of the clock generator chip, such that the chosen settings are always selected after a system reset. The chip can be reset to the 15 MHz default by applying a short ground to the “mode” input

Figure 40 Clock chip diagram AMI FS6370-1



Each of the clock domains can be programmed to different frequencies according to the integer values stored in the FS6370 registers as shown below, where FBKDIV is the feedback divider modulus of the A and M counters (Figure 40)

Table 4 FS6370 Register Map for the ReadOut Unit

Address	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0xF	*Not used*						PDPOST_B	PDPOST_A
0xC	POST_B[3:0] (EBI)				POST_A[3:0] (SEM)			
0x5	MUX_B[1:0]	PDPLL_B	LFTC_B	CP_B	FBKDIV_B	[10:8]	M-Counter	
0x4	FKBDIV_B[7:3] M-Counter					FKBDIV_B[2:0]	A-Counter	
0x3	REFDIV_B[7:0] (EBI)							
0x2	MUX_A[1:0]	PDPLL_A	LFTC_A	CP_A	FBKDIV_A	[10:8]	M-Counter	
0x1	FBKDIV_A[7:3] M-Counter					FBKDIV_A[2:0]	A-Counter	
0x0	REFDIV_A[7:0] (SEM)							

The most convenient way to program the RU's colock domains is via a program provided by

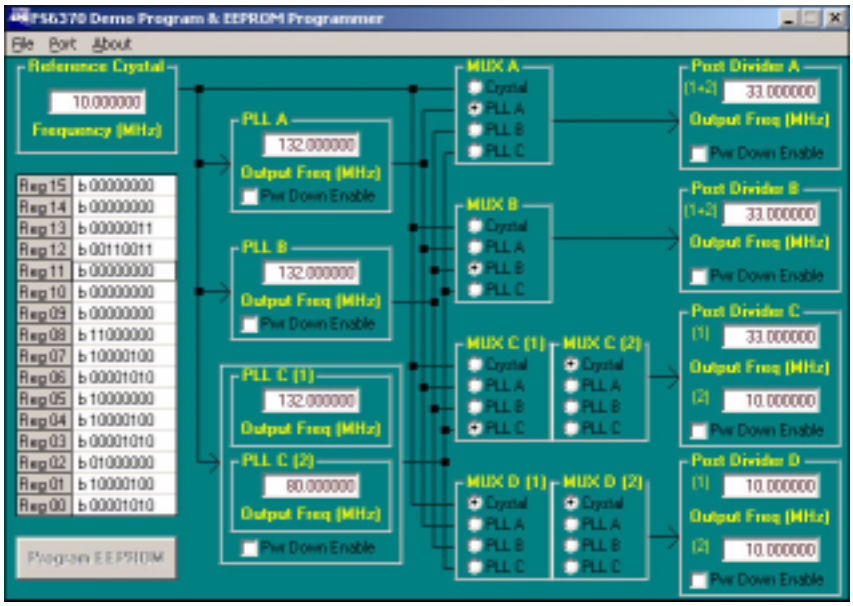


Figure 41 FS6370 congfiguration program for Windows

This program calculates the settings for the programmable clockchip registers starting from the 10 MHz reference chrystal. The default for the RU domains (33 MHz for all) is shown.

the manufacturer AMI[1].

The FS6370 demo program for Windows allows to select all desired clock settings and generates an output file for all 16 registers contained inside the clock chip. The registers have to be loadad via the I2C loader program running of the MCU.

The clocks settings of the 3 domains may be measured on buffered clock outputs on the following test pins:

- PCI.....TP2 (TP1=GND) rear part close square hole on PCB
- SEM .....TP6 (TP5=GND) between PLD sockets close to IC22
- EBI .....TP3 (TP4=GND) above DPM chips, close to IC14

8.6 Reset logic

1. http://www.amis.com/tgp/feature_sheets/fs6370.cfm

The powerup / reset logic uses a timer chip to generate at powerup a 8 ms general system reset pulse which is available on test point TP 7 as shown in Figure 42. The 2.5 Volt for the FPGAs is required to be stable before the system reset can fire. Apart from powerup there are 3 sources to generate a system reset:

- NIM Reset input on the front panel (may be used for other purposes[1])
- Pushbutton (on the RU top front, see Figure 43)
- Reset request from the MCU P14 connector (the MCU does not reset itself)

Figure 42 RU Reset logic

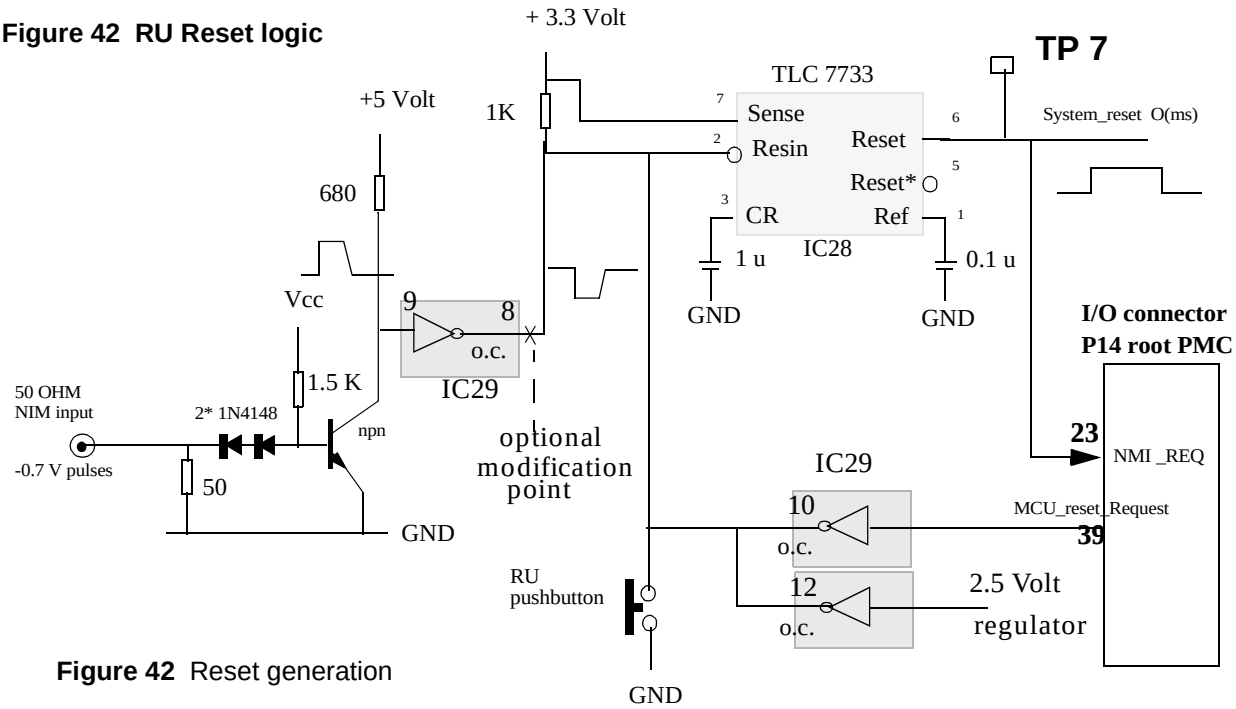
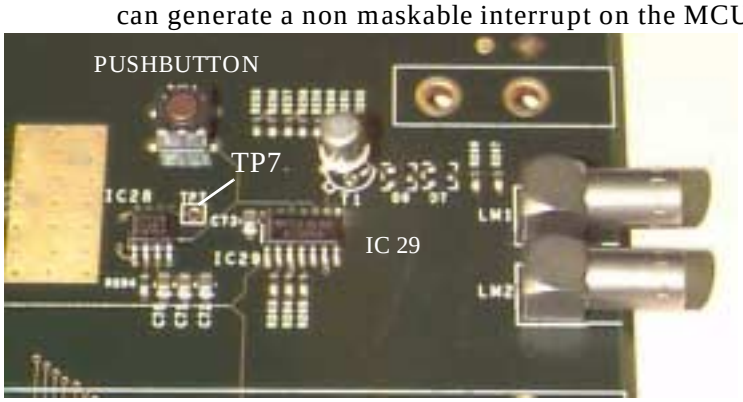


Figure 42 Reset generation

The MCU may[2] request a system reset via the P14 connector on the root PMC. A system reset can generate a non maskable interrupt on the MCU via the P14 connector.



NIM Reset In

Throttle out

Figure 43 Reset Input and pushbutton

Either a pushbutton or a NIM input pulse can be used to generate a RU system reset, including FPGA reloading . The host card will receive a NMI.

1. by unsoldering pin 8 of IC29 (see Figure 42) this pin may be reconnected to a pullup resistor to produce a positive TTL pulse when an NIM input is received.
2. the preferred method is however to use a PCI reset

8.6.1 Modified use of input NIM signal (Xon/Xoff etc)

A simple modification on the RU is required for using the NIM input as a signal for the FPGA logic. One example is the use of an Xon/Xoff input signal. In this case one of the function indicator LEDs as shown in Figure 49 must be unsoldered to give access to an unused I/O pin of the FPGA. Then pin 8 of IC29 (see Figure 42) must be disconnected from the PCB and reconnected (using a wire) to the free ORCA (previous) LED pin. As IC29 is an open collector device, a pullup resistor is needed. Note however: in case the NIM signal is of sufficient duration (order of us), the LED may be retained serving both as a pullup and as NIM signal indicator.

9 Physical RU: PCB, chips,mezzanines, connectors

The RU main card logic is implemented on a 9U IEEU-960 card as depicted in Figure 44. The use of mezzanine card standard is intentional (not neccesarily cheaper) to protect against component and standard obsolescence. The 6 mezzanine emplacements for Slink and PMCs are foreseen for 149*74 mm mezzanine cards with optional connectors in positions as shown in Figure 50.

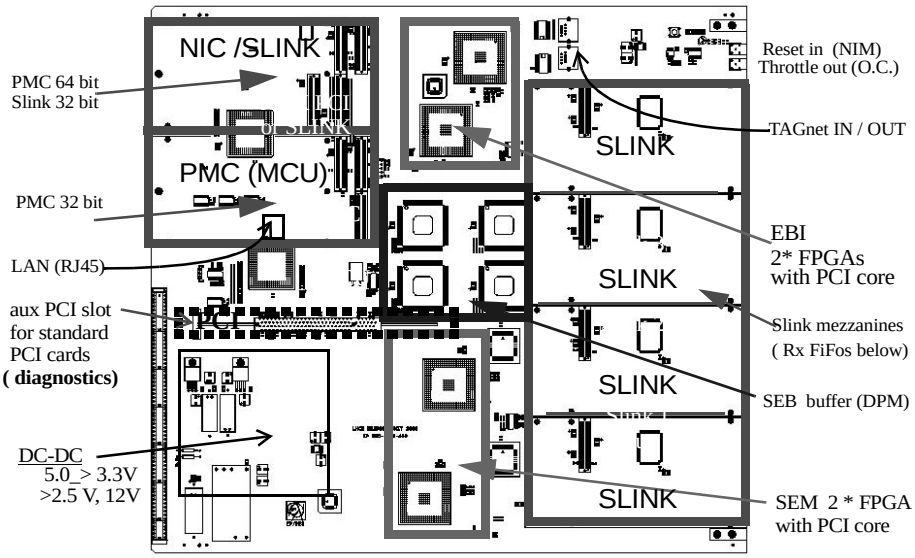


Figure 44 9U RU board arrangement

There are 6 PMC sized mezzanine card emplacements: 4 Slink inputs on the front and 2 PMC/Slink outputs at the rear. The FPSC chip based input and output logic is in the middle part, with the SEM buffer in the center. The power conversion logic is close to the FB backplane connector which is only used for power. The reset and throttle I/O logic is at the top of the front, the LAN is part of the MCU card. An auxiliary PCI slot is available for standard PCI cards.

A photo of both a NIC card and MCU mezzanine card in the rear PMC positions is shown in Figure 45.

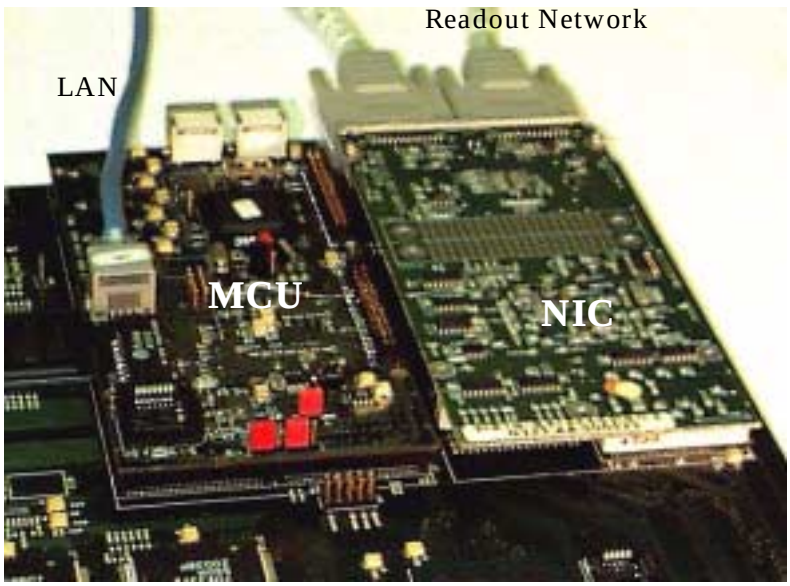


Figure 45 Photo of MCU and NIC cards on RU rear side

The MCU card was designed as a PMC specially for the Readout Unit. It's LAN cable is brought out on the read side of the RU. Due to the double width RU boards, the LAN connector fits on top of the PMC card.

The NIC card shown here is a commercial SCI network interface card as used for the VELO-L1 application.

9.1 FIFO, PLD and FPGA devices

- **PLD** is a Altera MAX 7032AE-4 in a 44-pin PLCC, the smallest device in the MAX 7000 family. With a pin-to-pin max. combinatorial delay of 4 ns, 32 macrocells and JTAG in-system reprogrammability, is well suited for the RU-II application

- **FPGA** is a Lucent Technologies' ORCA 3LP26 in a 352 BGA package. Together with ~120 kgates for user logic, it is equipped with a 64-bit 66-MHz master/target PCI interface hard core. This interface is available at boot time and is used for FPGA bitstream loading, configuration, control and monitoring from the MCU. A serial EEPROM is also provided on the RU-II board for FPGA bitstream configuration, which can be overrided by the MCU download via PCI

- **FIFO** is a IDT 72V36xx device, where "xx" can be 40, 60, 70, 80, 90, 100 or 110 depending on the required FIFO size (from 4kbytes to half megabyte). All these parts are pin compatible with one another. The preferred part is the 72V3670 (32 Kbytes) as it matches RU-II DAQ application. The device is a synchronous 36-bit FIFO in a 128pin TQFP package

9.2 Board specifications

The Readout Unit II is a 9U-size (366.7x400 mm) 2.2mm-thick[1] 8-layer board manufactured under class 5 specifications (track width is 6 mils). The stack-up structure is shown in figure 30. Copper (blue in the figure) thickness is 35 µm for top and bottm layers and half this value for inner layers, according to standard values in industry. The thicnkess of the dielectric layers (grey) has been defined in order to keep an uniform characteristic impedance among signal layers and thus avoiding signal reflections caused by impedance mismatches. One milimiter of dielectric between layers 4 and 5 increases the total board thickness to the required 2.2 mm.

Figure 46 Readout Unit II stack-up

Assuming $\varepsilon_r = 4.2$ for the dielectric, top and bottom layer impedance is 58.3 Ω whilst inner layer impedance is 55 Ω. These values correspond to unloaded lines. Propagation delays are 59.5 ps/cm for top and bottom layers and 68.3 ps/cm for inner layers.

1. ±0.2mm tolerance

9.3 Signal integrity simulations

Both pre-layout and post-layout signal integrity simulations have been carried out using Cadence SpecctraQuest SigXplorer, SpecctraQuest SigNoise and IBIS models from the different chip vendors. The results of these simulations allowed to identify lines that needed terminations, to choose and validate bus topologies and determine component placement on the board.

Special attention was paid to the PCI bus subsystem, where simulations showed the need to include a PCI bridge between the primary bus and the SEM FPGA PCI bus and helped to design the topology of the segments that had to run up to 66 MHz.

The net topology is either described based on placement assumptions (pre-layout analysis) or extracted from the PCB layout files (post-layout analysis). An example is shown in Figure 48 , where an FPGA pin (model orca_bmz_OB12) drives two DPM ICs (model 70v9279pf_70v9279z_pf_io0_mdl) Trace parameters (lenght and width) are also shown.

Once the net topology is defined, it is possible to perform signal reflection simulations, like the one shown in Figure 47 which corresponds to given example.

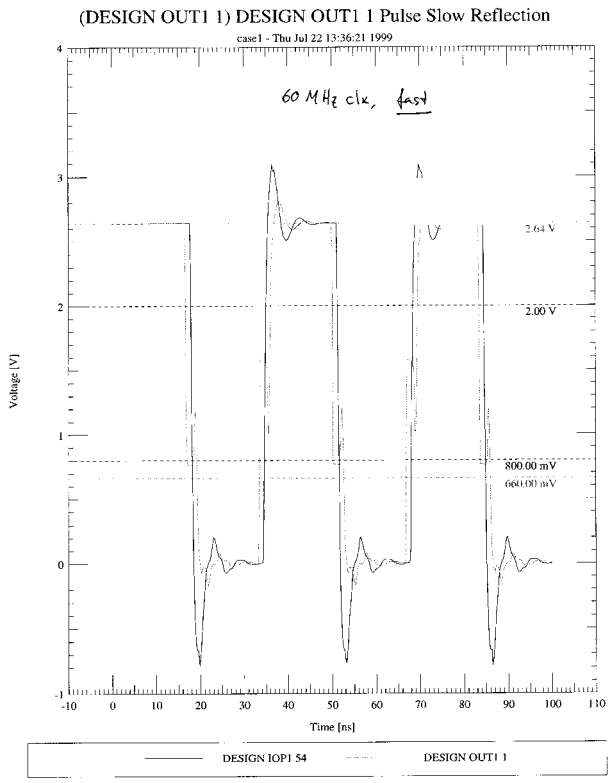
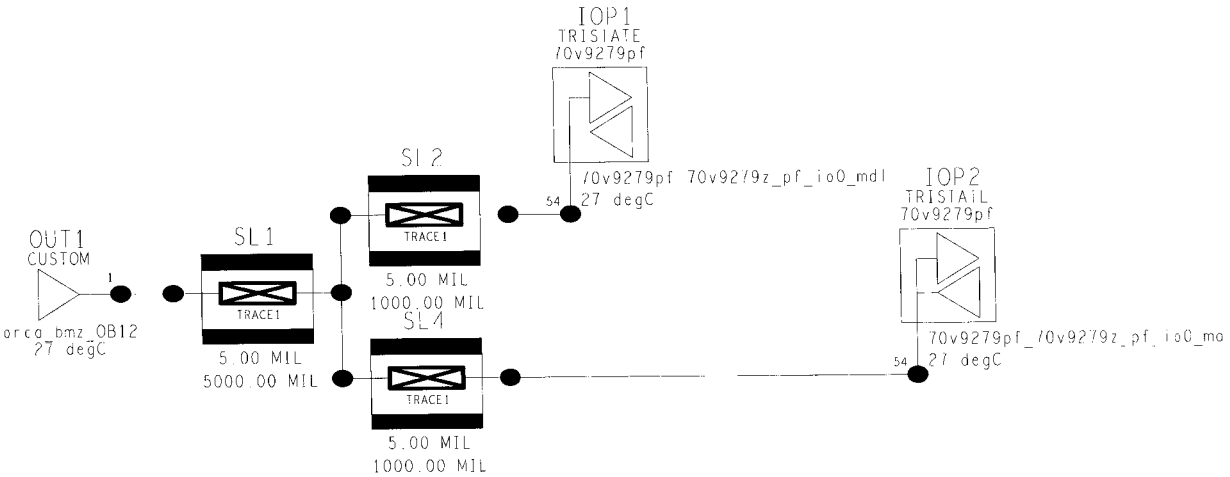


Figure 47 Definition of net topology with Specctraquest SigNoise

.

Figure 48 FPGA pin model



9.4 BGA chips

There are 4 programmable FPSC chips and two 2 PCI bridges mounted using ball grid array (BGA) technology. There are three surface mounted LEDs on two of the FPSC's to display activity and diagnostic information.

Figure 49 ORCA FPSC chip mounted on the RU in 352 pin BGA technology.

BGA improves electrical properties, allows for a very high number of pins, but makes access to individual pins impossible.

One FPSC of the input and one on the output stage are equipped with 3 LED displays. These are connected to spre FPGA pins and can be user programmed.

9.5 Mezzanine cards

There are 5 possible mezzanine connectors per emplacement for different purposes as shown in Figure 50. The connectors are 64pin, 1mm connectors[1] with SMD mounting for 10 mm mezzanine stack-hight.

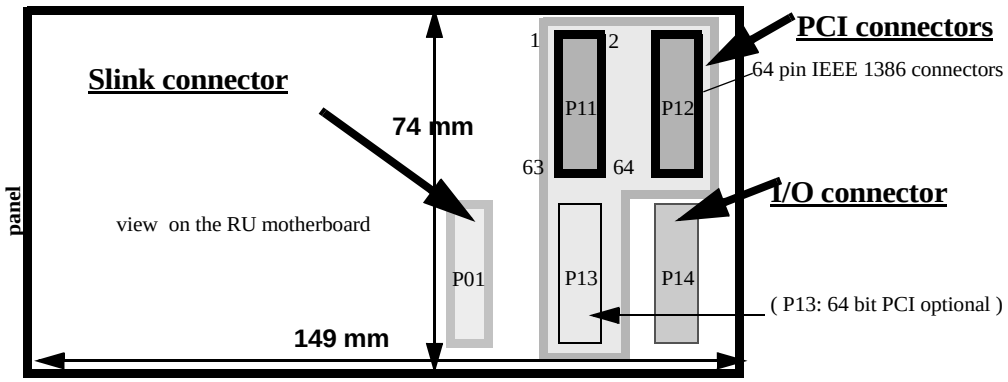


Figure 50 Mezzanine card connectors viewed on RU motherboard

64 pin connectors are used in all mezzanine emplacements. The P01 connector is used for Slink Rx /Tx cards, P11 and P12 for standard 32 bit PCI mezzanines, P13 for 64 bit PCI, and P14 for user-defined I/O

The inter-mezzanine spacing between RU's is depicted in Figure 51. PMC mezzanine cards do

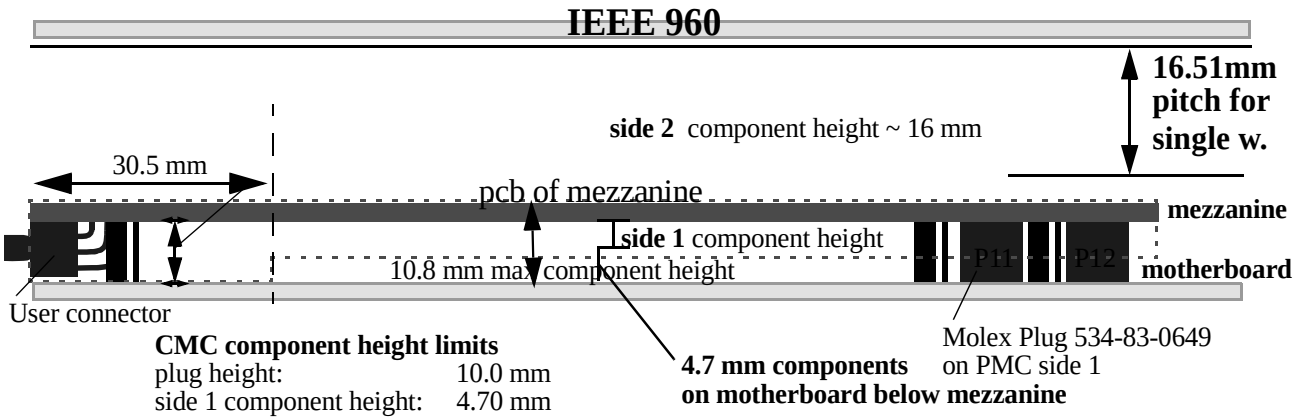


Figure 51 Slink and PMC mezzanine stack height limits for IEEE960 (Fastbus)

not fit in height in a single width IEEE960 environment of 16.51 mm pitch[2], therefore double width spacing was taken as default on the RU The advantage is that the problematic PMC side 2 component height of 2-3 mm (valid for VME single U) is much relaxed to ca 14 mm. Typical connectors like 11 mm high RJ45 network connectors for ethernet may thus be used without violation of tolerances.

1. according to IEEE P1386. The connector named "receptable" is on the RU motherboard (AMP TYCO 120521-1)
2. in VME this is ca 20 mm

9.5.1 PMC emplacement

The two rear-sde PMC emplacements (ROOT and NIC) and their relative connectors are shown in Figure 52. The root PMC carries the PCI host (MCU card) and is therefore equipped with the I/O connector P14. The pinout of the 3 PCI connectors is shown in Table 8.

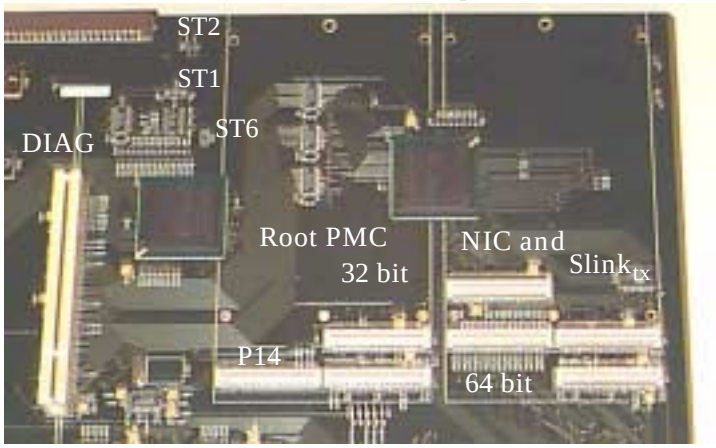


Figure 52 PMC and Slink Tx emplacements on the RU rear.

The two rearside PMC/Slink emplacements. The 32 bit Root PMC has the 32 bit connectors P11 and P12 and the user connector P14. The NIC card emplacement (right) has the 64 bit PCI connector set P11,P12, and P13 and the Slink P01.The PCI diagnostic connector is on the left side of the photo

9.6 Jumpers and Testpoints

Some hardware jumpers provide for RU configurations which are deemed to be static for the use of any particular RU application type

Table 5 Jumpers (STx) and Test Points (Tx) on the Readout Unit

Name	Type	Purpose	details
ST1	2 of 3 pin	PCI host selection	either PCI aux (A) or MCU (B) is the PCI host + arbiter
ST2	2 pin	66 MHz bridge	select PCI bridge chip version: BC = 66 MHz capable, AC =33 MHz
ST3	2 pin	SEM 1 mode	programmable Jumper selection for SEM 1
ST4	2 pin	SEM 2 mode	programmable Jumper selection for SEM 2
ST5	2 pin	Slink 16 or 32 bit	default open, closed for 16 bit Slink operation on inputs
ST6	2 pin	33 or 66 MHz PCI	if connected, the PCI bus runs at 66 MHz, AUX disabled
TP1,4,5	1 pin	GND	GND
TP2	1 pin	PCI clock domain	PCI clock as programmed by clock generator chip
TP3	1 pin	EBI clock domain	EBI clock as programmed by clock generator chip
TP6	1 pin	SEM clock domain	SEM clock as programmed by clock generator chip
TP7	1 pin	System Reset	see Figure 42

9.7 Auxiliary PCI connector

The 32 bit PCI auxiliary connector shown in Figure 53 on the root PCI segment is a standard PCI connector which may be used for standard PCI cards, in particular for PCI analyzer cards.

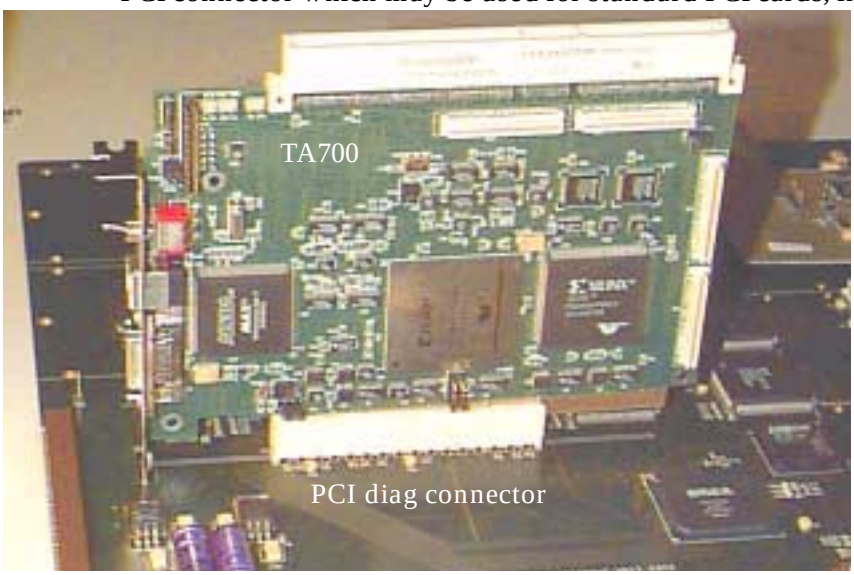


Figure 53 Diagnostic PCI card in aux PCI connector

The photo shows the auxiliary PCI connector with a PCI analyzer card for diagnostics on the root PCI segment and/or generation of PCI transactions as the PCI host. The Analyzer card is the Catalyst TA700. It can access all FPGAs, the NIC and the root PCI emplacements.

The auxiliay PCI connector is disabled (via Jumper ST6) when the root PCI is run in 66 MHz mode . For diagnosing 66 MHz or 64 bit PCI transactions on the secondary PCI segment, a 64 bit mezzanine-to-PCI adapter is to be used in the output mezzanine slot of Figure 52 .

9.8 Slink mezzanine cards

The Slink input stage mezzanine emplacement is shown in Figure 54 .. Two Slink cards have

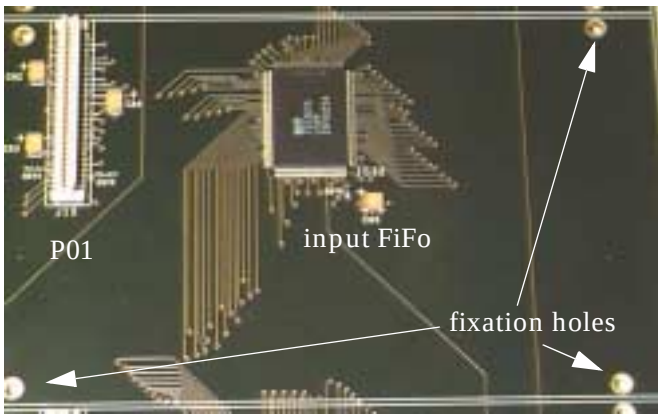


Figure 54 Slink Rx emplacement on Frontpanel side

The four Slink Rx mezzanine card emplacement have only one connector P01 (pinout see Table 10). The input FiFo of Figure 11 is a close to the connector.

There are fixation holes for the Slink mezzanine card.

been designed to be used in this place: single and quad receiver for DAQ/VELO and Multiplexer applications respectively. The single Slink card may also be configured as Slink Transmitter to be used in the NIC/Slink mezzanine slot at the RU output (see Figure 52)

Shown in Figure 55 on the left side is the single link Rx / Tx card. This board can be configured

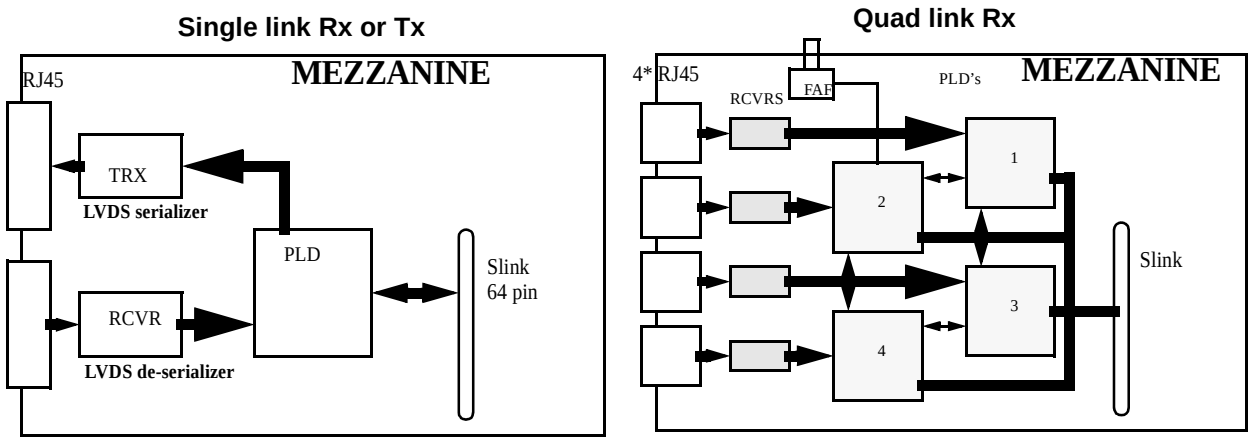
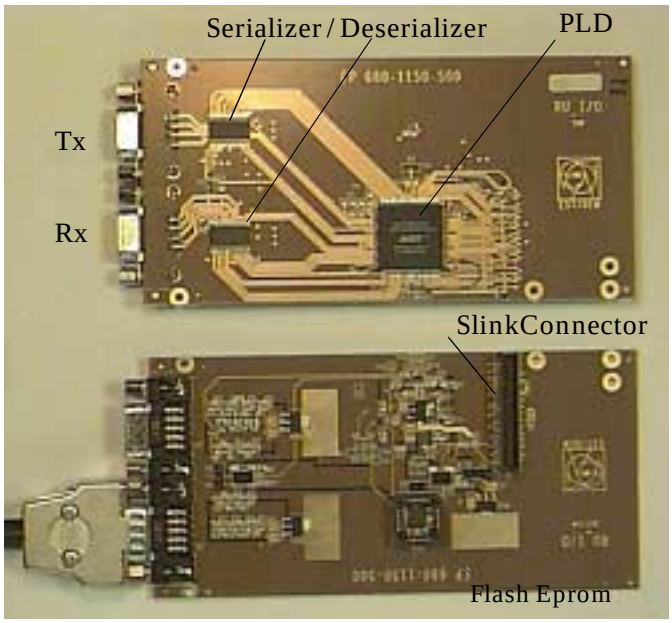


Figure 55 Two SLINK LVDScards designed for the RU

either as a transmitter (Link Source Card) or as a receiver (Link Destination Card) by jumpers and configuration PROM.

The quad receiver card (right side) contains 4 receiver channels almost identical as in the single version. The data blocs are routed consecutively via a token passing system from the PLD embedded Fifo's to the S-Link connector. A time-out circuit and a missing or bad link detection is foreseen, creating a dummy error bloc of 3 words for the failing channel. An LVDS output Fifo Almost Full signal is available on a Lemo connector as a feed back warning signal to the previous stage (required for particular applications). The photos of the corresponding cards are shown in Figure 56 .



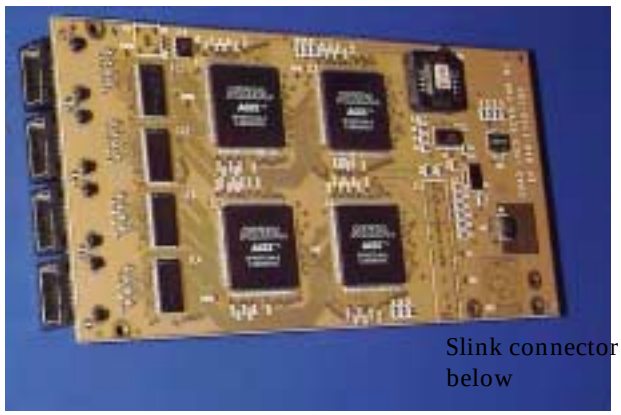


Figure 56 Photos of Slink mezzanine cards for the RU

Above: Quad Slink Receiver mezzanine card with 4 RJ45 plugs on the frontpanel.

Left: top and bottom side of configurable single link Slink Rx / Tx card

These mezzanine cards for the RU use LVDS serializer / deserializer chips in conjunction with an FPGA based interface and FiFo logic for the Slink protocol. The principle of serialization and deserialization, based on National Semiconductors “Channellink” is depicted in Figure 57 (top)

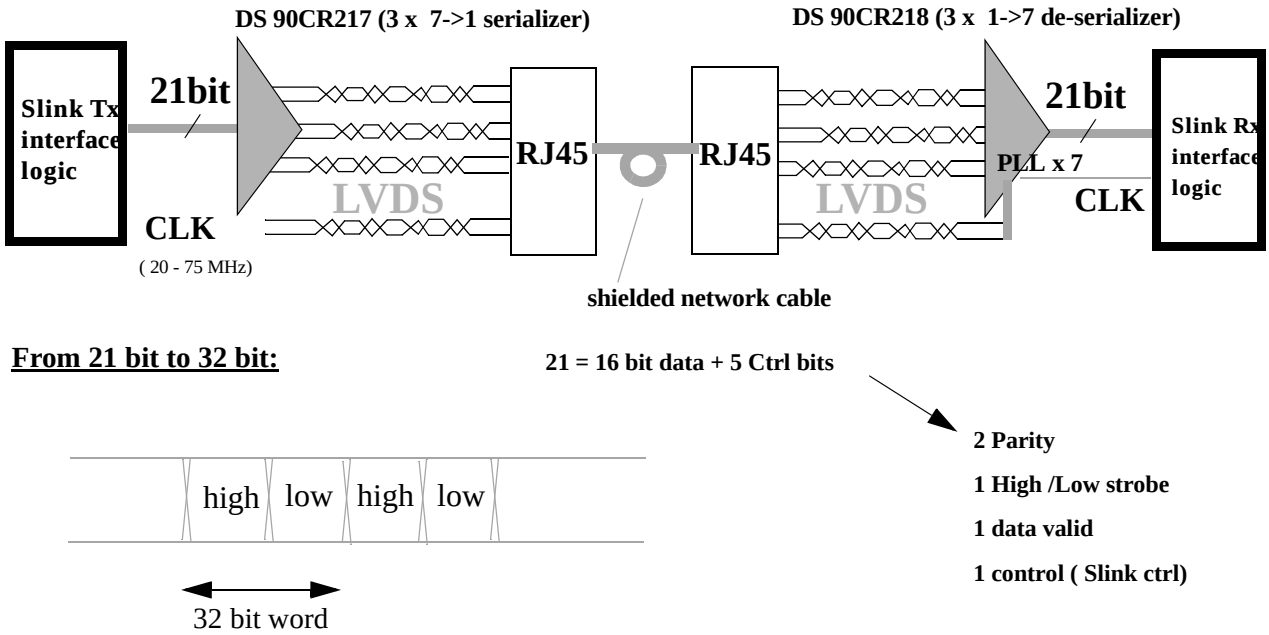


Figure 57 LVDS serialization on Slink test cards for the RU

The DS90CR217 transmitter converts 21 bits of CMOS / TTL data into three LVDS data streams. A phase-locked transmit clock is transmitted in parallel with the data streams over a fourth LVDS link. For every cycle of the transmit clock, 21 bits of input data are sampled and transmitted over 3 LVDS lines. The DS90CR218 receiver converts the three LVDS data streams back into 21 bits of CMOS / TTL data. At a transmit clock frequency of 75 MHz, 21 bits of TTL data are transmitted at a rate of 525 Mbps per LVDS data channel. Using a 75 MHz clock, the data throughput is 1.575 Gbit / s (197 Mbytes / sec). The clock frequencies defined for these applications are 20 and 40 MHz. This chipset is an ideal means to solve EMI and cable size problems associated with wide, high speed TTL interfaces.

In order to make 32 bit Slink words from a 21 bit serializer, the 21 bits are used to transmit 16 bit data plus 5 control bits (see Figure 57 bottom). With these control bits, two consecutive (high / low) transmission streams are decoded into a single Slink 32 bit word.

9.8.1 Slink card features

- 20 to 75 MHz shift clock support
- Low power consumption
- ±1V common mode range (around +1.2V)
- Up to 1.575 Gbps throughput
- Up to 197 Megabytes / sec bandwidth
- 345 mV (typical) swing LVDS devices for low EMI
- PLL requires no external components
- Compatible with TIA / EIA-644 LVDS standardPLD on the Slink cards

The PLD used is an inexpensive Altera ACEX EP1K50TC144-2 device. This IC takes care of al the logic functions i.e.:

- multiplexing / demultiplexing
- parity generation / checking
- local memory (40,960 RAM bits) configured as a dual clock synchronous FIFO
- interfacing dialog to Channel link and S-link
- test pattern generation, etc.
- token passing, channel selection etc. in quad input version

9.9 Preliminary tests on cable lengths

Some basic tests have been done with different types of cables and cable lengths i.e. category 6 and category 7 UTP & STP fitted with DB9 and MT-RJ45 connectors. Some more extensive tests will be done with the FLIC card [38] - based RU exerciser.

Results:

1) UTP cable can only be used on lengths < 2 meter and 20MHz (40 MB) transmission speed and is NOT recommended!

2) STP cat. 7 (Alcatel data cable STP600, 23 AWG), cable fitted with DB9 connectors.

- 40 MB / s max. 20 m.
- 80 MB / s max. 15 m.

3) STP cat. 6 (Daetwyler Uninet flex 4P 600 MHz), standard MT-RJ45 fully fitted cable available off the shelf in different lengths (2, 3, 5, 10 m.).

- 40 MB / s max. 25 m.
- 80 MB / s max. 20 m.

10 Crate, power cooling environment

IEEE 960 mechanics for doublewidth RUs[1] corresponds to 13 modules per bin (Figure 59.)

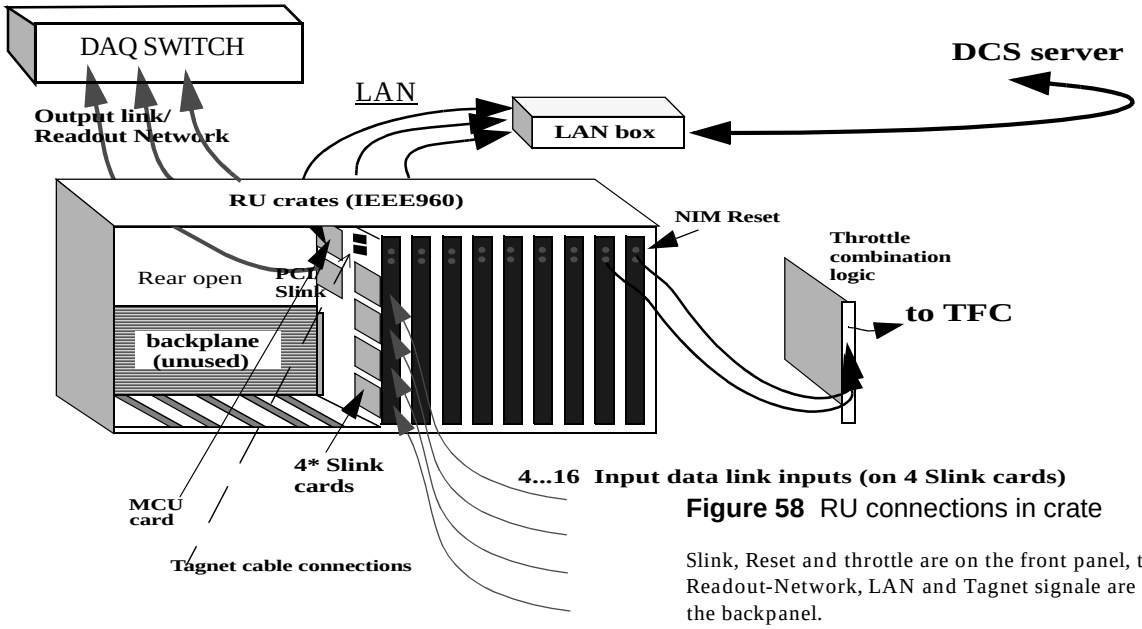


Figure 58 RU connections in crate

Slink, Reset and throttle are on the front panel, the Readout-Network, LAN and Tagnet signale are on the backpanel.

with link inputs on the frontpanel and outputs at the read side (Figure 58).

- Standard PMC or 32 bit Slink mezzanine cards can be used. Dedicated mezzanines with 16 mm height for componnets and 10 Watt power may be designed for the RU in the double width IEEE 960 mechanics.
- The power requirement per bin (13 RU modules of 65 Watt each) is ca 1 KW (Table 6)
- A RU motherboard can be equipped with: up to 4 Slink receiver cards (front panel), up to 2 PCI mezzanine cards in the rear area.
- On the rear two PMC emplacements, the upper one may can also be used for Slink
- The RESET input and Trottle output is on the front panel.
- For 100 RU modules, the DAQ application requires ca 8 crates (= 3 racks)

Figure 59 Photo of F6853 CERN crate. The upper rear half is free

Table 6 RU full crate power requirement

power modules	Maximum current	No of modules needed
+5 V	100 A	2
+15 V	25 A, 30 A	1

1. double width is required to allow for standard mezzanine cards (Slink, NIC, MCU)

11 Programmable logic in the Readout Unit

Both the SEB and EBI logic is fully contained in four field programmable system chips (FPSC), i.e FPGAs with internal ASIC cores for the PCI bus.(Figure 60)

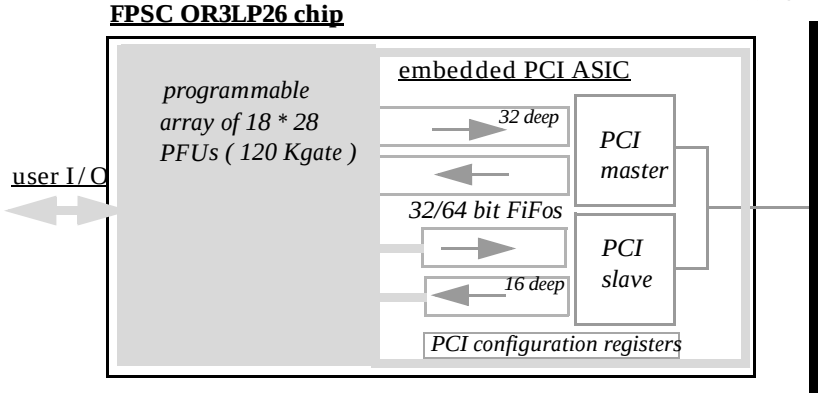


Figure 60 Lucent FPSC chip OR3LP26

Four of these special FPGAs contain the RU's SEM and EBI logic.

The integrated PCI master/slave logic is interfaced to the user logic via bi-directional FiFos.

The chosen devices are the OR3LP26[1] chips from Lucent Technologies [12]. These chips contain 504 programmable function units (PFU) and a 64-bit, 66-MHz capable PCI core which is fully compliant to the PCI 2.2 specification [19]. There are four internal FiFos interconnecting the programmable PFU logic and the PCI target/master logic. As showin in Figure 60, a total of four such FiFos (for each direction and for master and slave) are part of the internal PCI ASIC. The Fifos whicgh can be used in 32 or 64 bit mode (Figure 66) are interfaced to the programmable applications like SEM or EBI. The PCI core contains both a PCI initiator (master) and target (slave) as well as the mandatory 256 byte PCI configuration space.(Table 3)

The use of the Lucent ORCA FPSC necessitates specific tools and licencies such as the Foundry tools. The development environment from VHDL to Flash Eprom is depicted in Figure 61 .

FPSC: Orca FPSC Configuration Wizard licence: Configure the PCI core, define the BAR registers in the PCI configuration space, set PCI master capability. Generates a VHDL or a Verilog template for the design files.

VisualHDL: Design and Simulation. Design files can be either vhdl files, state diagrams, block diagrams, flow diagrams or truth tables

Synplify Synthesizer: translates to gates, flip-flops, generates an edif file, gives an aproximation of the maximum frequency of the circuit and the resources needed inside the FPGA

Orca Foundry 9.x licence: Place and Route select the position LUT's, flip-flops etc and interconnect them physically inside the FPGA

Epic: change the selections made by the automatic placer and router by hand

1. BGA chip with 352 contacts

Bitgen and Promgen: transforme a final .ncd file into a .bit file which is upload to the device itself.

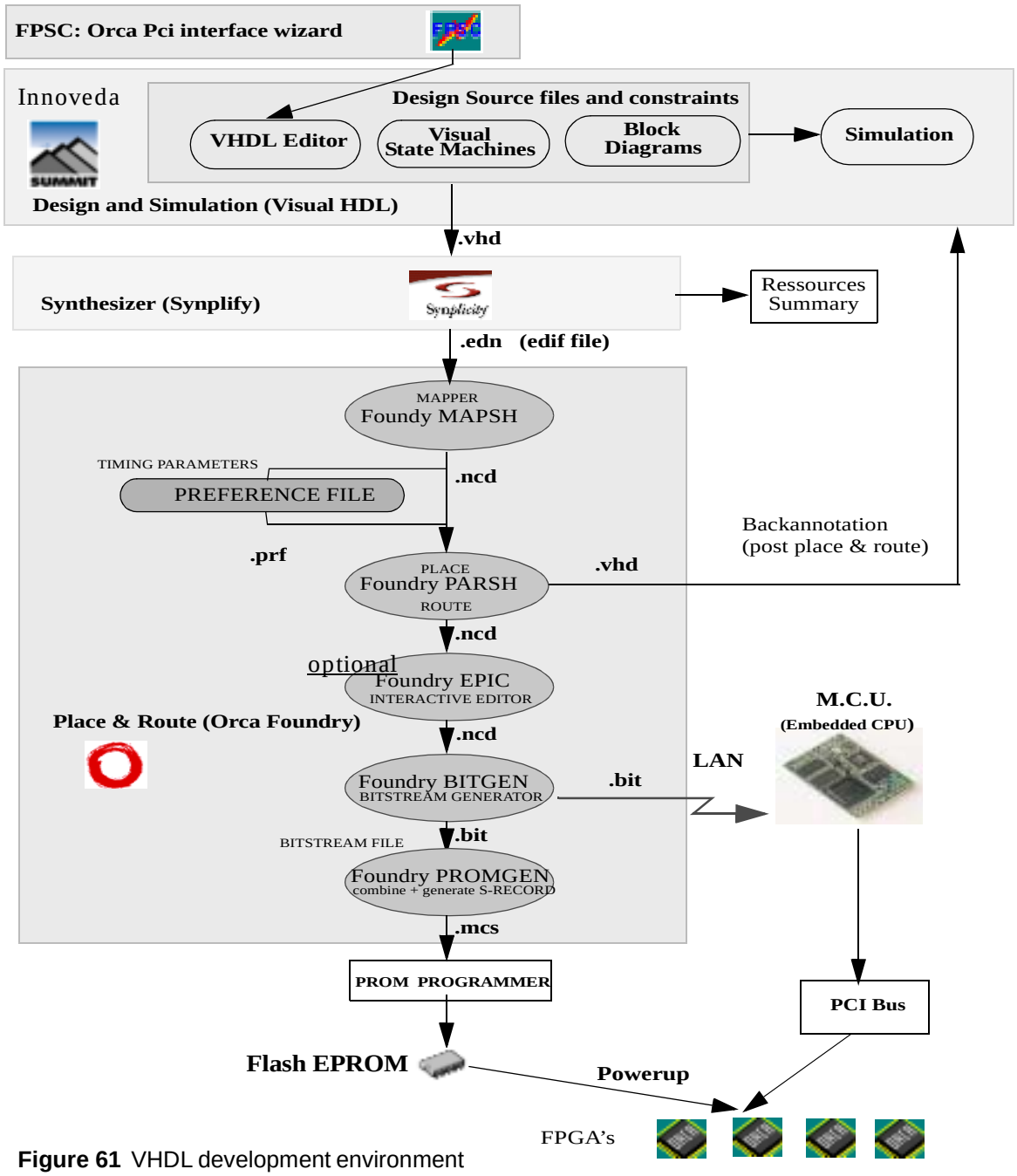


Figure 61 VHDL development environment for Lucent OTRCA FPGAs

FPGAs uploading: a.) from a FLASH EPROM at power-up. b.) online via PCI "flash" utility on PCI host system (MCU)

11.1 VHDL development

FPSC applications can be developed in the VHDL language, using compiler/simulator tools like Visual HDL. For better portability and documentation, the normal starting phase for an FPSC application in the RU project is creation of state machines as depicted in Figure 62, which is an option in the Visual HDL tool.VHDL code is automatically generated for the

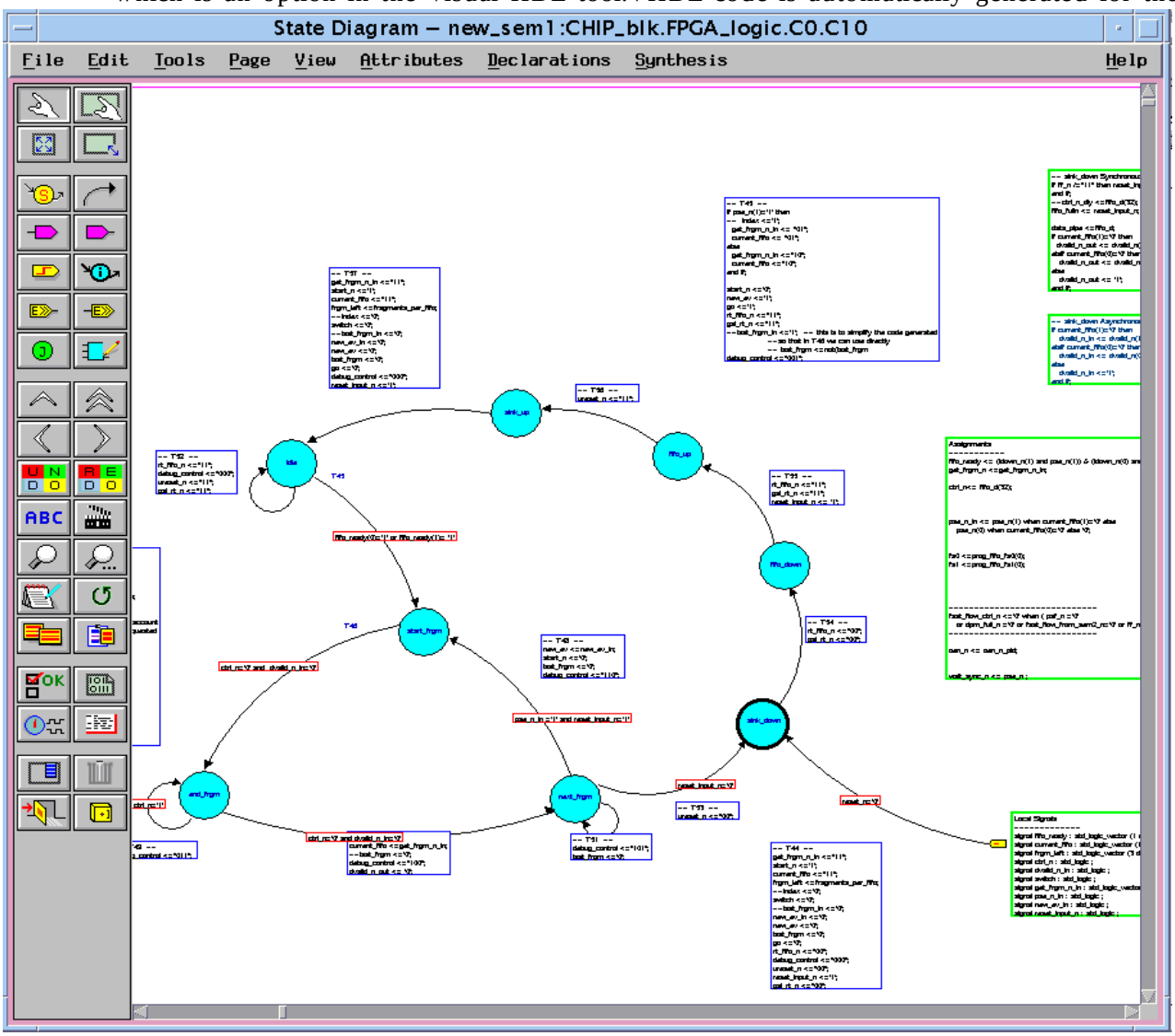


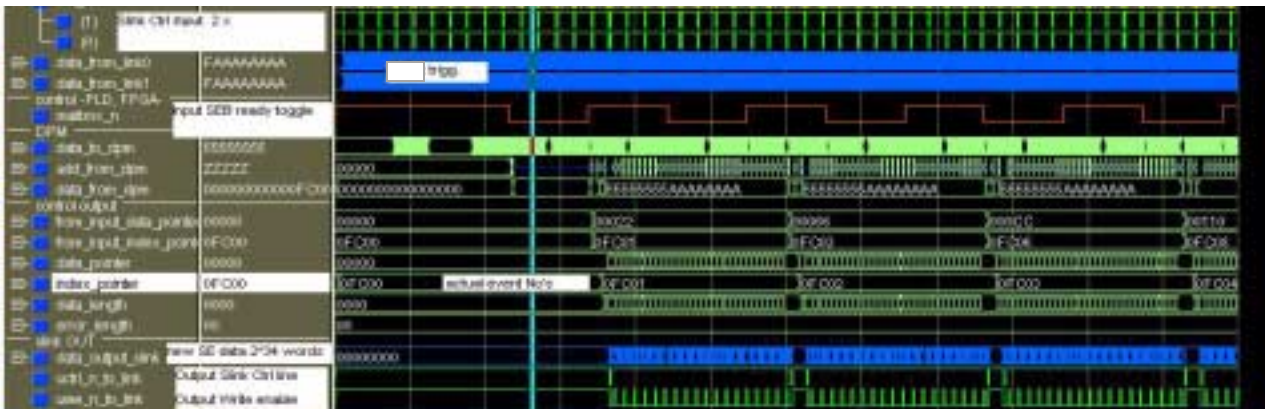
Figure 62 Creation process of an FPSC state machine on a Visual HDL screen,

subsequent synthesis process (Figure 61).

11.2 Timing simulation

The principal of timing simulation of an Slink->Slink subevent building operation is shown in Figure 63. Two Slink inputs are received, quickly filling the input FiFos. After about 10

Figure 63 Modelsim simulation of Slink-Slink



received input fragments, the SEB buffer ready signal toggles, indicating that the 2->1 subevent building process is ready. Technically this is a mailbox signal of the dual port buffer to the output stage. After the output stage latency, the output Slink transmits the new subevent, which indicated by the output Slink control line which accompanies the first and the last word of the SEB burst.

A corresponding mesaurement taken with an oscilloscope is shown in Figure 64.

Figure 64 Subevent building measurement, 2* 256 byte @ 400 kHz, FPGA clock 50% of nominal

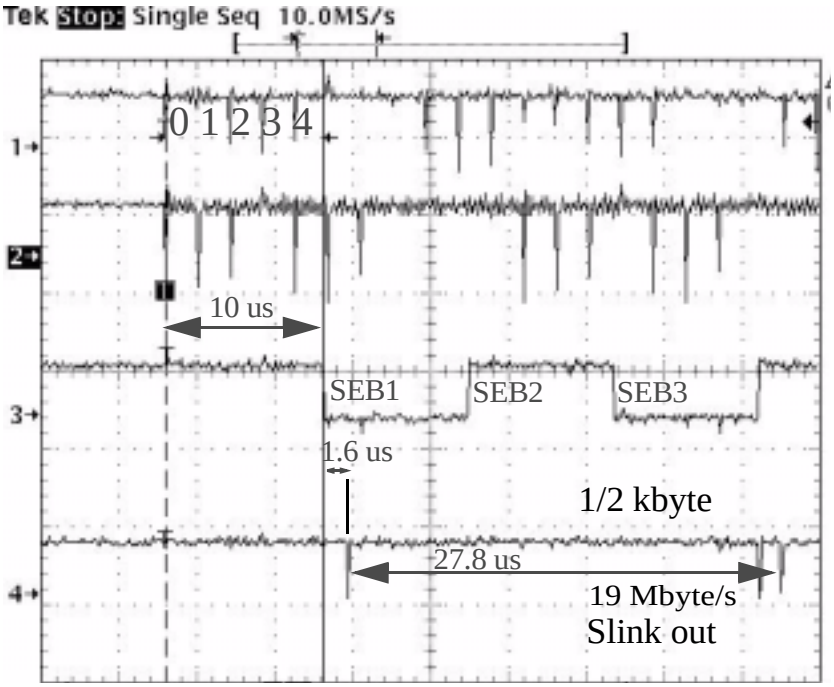
The timing is 5 us per divison.

Trace 1 shows one of the incoming Slink control inputs, with 256 byte payload every 2.5 us (400 kHz). The very first input (No 0) is used to start the Readout Unit processing (it should contain dummy data)

Trace 2 shows the corresponding input FiFo Flag signals.

Trace 3 shows the SEB mailbox signal toggling every time the SEB has a 2->1 subevent built. In this example it takes about 10 us per 1/2 kbyte subevent (in final version factor 4 faster)

Trace 4 shows the Slink output control lines. The output latency to Slink is 1.6 us. Every 27.8 us another 1/2 kbyte subevent is transmitted, corresponding to 19 Mbyte/s (in final version factor 4 faster)



This example was taken with a very low 15 MHz FPGA clock and one lost dummy state machine cycle over 3, i.e a factor 4 in the final performance will be reached with 40 MHz FPGA operation (factor 2.6 in clock) and removal of the dummy cycle (factor 1.5 in state machine). The optimized Slink->Slink performance performance (4* 19 Mbyte/s) will thus reach close to 80 Mbyte/s as in the introductory performance envelope[1] for Slink of Figure 1.

11.3 RU behavioral models

to be inserted here (Angel)

1. The output to PCI bus would give another factor 4 more BW (64 bit instead of 16 bit / clock)

12 PCI subsystem: configuration and access utilities

The PCI bus hierarchy on the RU was described in chapter 8.1 .. This chapter describes in more details the access methods from a Linux operating system, running on the on the MCU:

- Sub-event buffer access from PCI
- Control and monitoring registers

The PCI performance measurement and the FPGA configuration software are briefly summarized

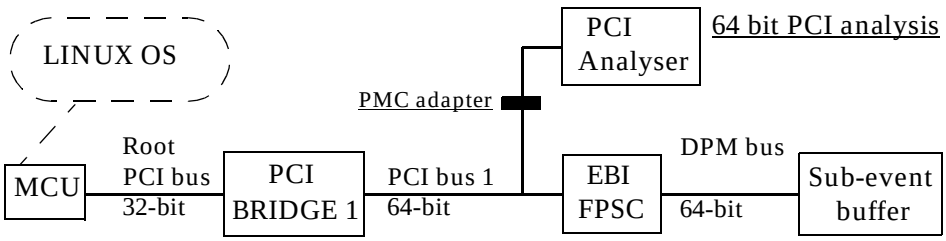
12.1 Access to the sub-event buffer from PCI

The subevent buffer (SEB) is accessible from PCI either via the input stage (SEM) or via the output stage (EBI), provided that the FPSC logic is programmed to connect its PCI configuration space to the SEB. Under Linux, there are utilities to list and edit the PCI configuration space, and to transfer test data. Using these test utilities on the MCU, a complementary way to debug errors is by inserting a PCI analyzer/tracer in the RU's diagnostic connector.

12.1.1 SEB r/w test setup

The simple test utility for Linux, ***rwpci*** (described in Section 14.0.2) can be used to provide access to the SEB from the MCU. This requires on the EBI FPSCs a target PCI interface (generated with Orca FPSC design kit) and a dual-port memory controller programmed in VHDL. Figure 65 shows the data path between the MCU and the SEB and a PCI tracer used in

Figure 65 Data path between the MCU and the Sub-event buffer



the PMC slot of PCI bus 1 for tracing 64 bit transactions.

Using simple assignments like the shaded instructions in Figure 78, the MCU accesses PCI memory space, provoking single-word (i.e. no burst) 32-bit PCI transactions on the root PCI bus. As the address is destined on PCI bus 1, the PCI bridge reacts by initiating a transaction on the 64-bit wide PCI bus 1. Both the PCI bridge and the FPSC are 64-bit capable. Nevertheless, this does not mean that the bridge will therefore initiate a 64-bit transaction on PCI bus 1. In fact, only when the EBI FPSC declares the SEB to be prefetchable[1], the

transaction address is quadword[1] aligned and at least two (memory read) or three (memory write) words are to be transferred, the bridge will perform a 64-bit transaction. Thus, single-word transactions, that are the ones generated by our test software, imply 32-bit bus accesses in PCI bus 1.

12.1.2 FPSC PCI port modes

User logic inside the FPSCs cannot know if the transaction on the PCI bus 1 is 32 or 64-bit wide. This is meant to be transparent to both ends (user logic in the FPSC and C application running on the MCU) and thus have no implication on the success of the transaction.

The EBI FPSC can only access the SEB by means of 64-bit read and write pipelined transactions. The glue logic between the PCI core and the user logic in the FPSC can be configured to either *Quad port* or *Dual port* mode. The PCI core does the data width conversions between the current PCI transaction and glue logic data widths.

- *Quad port*: Master and target interfaces have separated 32-bit read and write paths Figure 66 (left side)
- *Dual port*: Master and target share 64-bit read and write paths Figure 66 (right side).

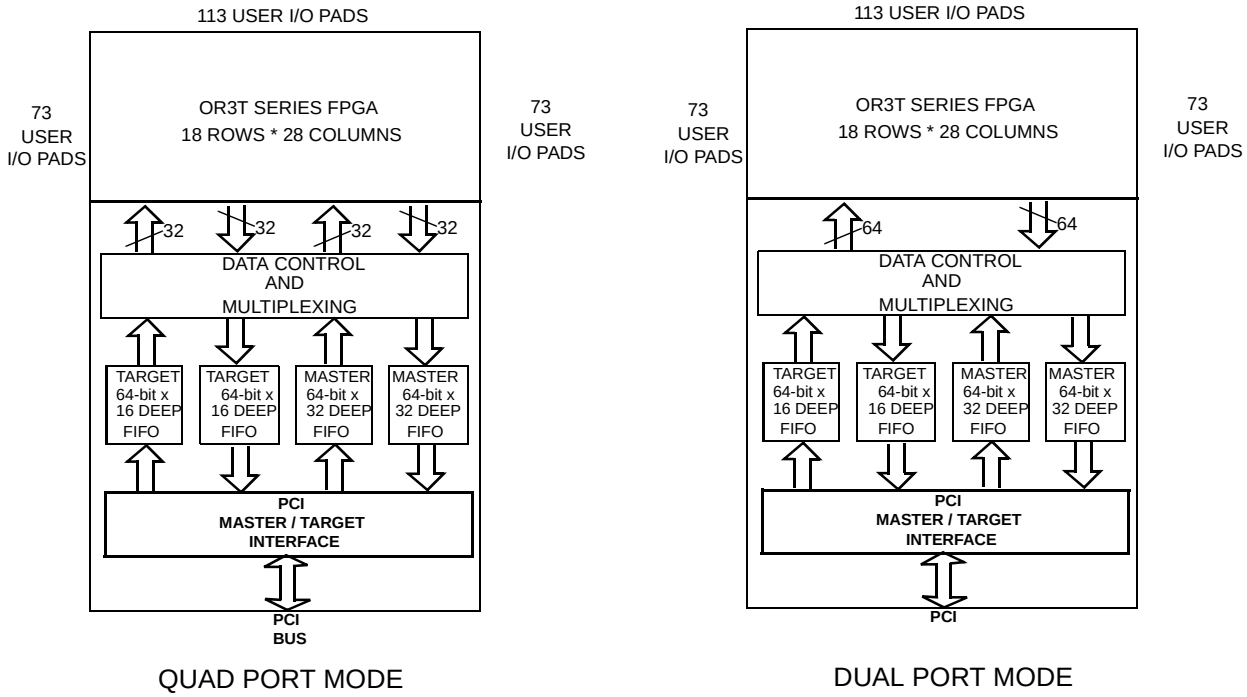


Figure 66 *Quad port* and *Dual port* modes for the PCI interface logic in the ORCA FPSC

Both modes are described in more detail in the following chapters.

1. via the FPSC design kit utility
1. “Quadword” is 64 bits. “Dualword” is 32 bits. “Word” is used as a generic term

12.1.3 Testing *Quad port* mode and non-prefetchable memory

As the SEB is declared as non-prefetchable for this test, the bridge performs single-word 32-bit transactions. This was verified with a PCI analyser[1] plugged in PCI bus 1. The *rwpci* test utility combined with the PCI analyser measurements show that read and write accesses to even memory positions are performed successfully, whilst read accesses to odd memory positions return the contents of the next memory position (i.e., read to address 5 returns data in address 6). Wrong data alignment in 32-bit to 64-bit conversion in the PCI core FIFOs is suspected.

Figure 67 PCI bus 1 activity captured with a PCI analyser

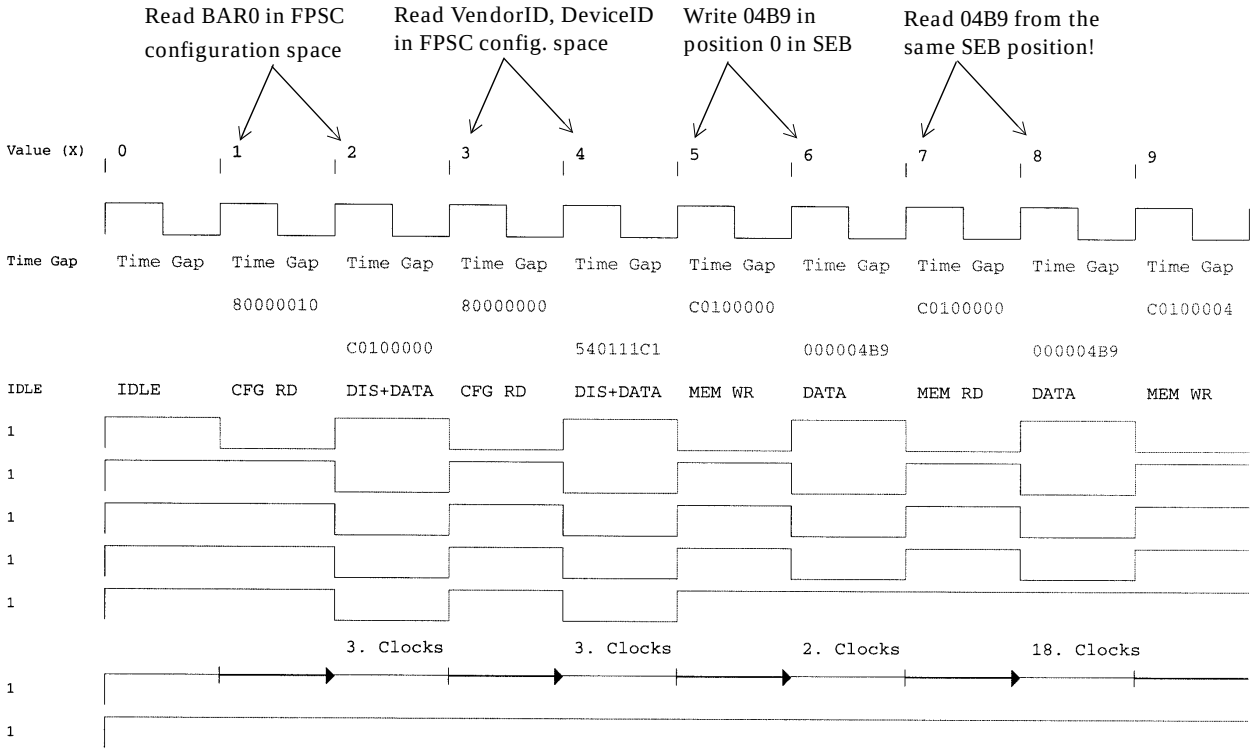


Figure shows in cycles 5-8 a successful write followed by a read to start of SEB:

- Cycle 0: IDLE. No transaction taking place in the bus
- Cycle 1: Configuration read to Ox80000010 (i.e., BAR0 which is in offset Ox10 in FPSC configuration space starting at Ox80000000)
- Cycle 2: Read value: OxC0100000. The transaction finishes with disconnect+data, which means that the configuration space does not support burst mode transactions
- Cycle 3: Configuration read to Ox80000000 (i.e., VendorID and DeviceID in offset Ox0 in FPSC configuration space

1. TA-700 of Catalyst

- Cycle 4: Reading DeviceID is Ox5401 (OR3LP26B) and VendorID is Ox11C1 (Lucent)
- Cycle 5: Memory write to first position in SEB
- Cycle 6: Written value is Ox0000049B
- Cycle 7: Memory read to first position in SEB
- Cycle 8: Read value is Ox0000049B
- Cycle 9: Write to next memory position: OxC0100004

12.1.4 Testing *Dual port* mode and prefetchable memory

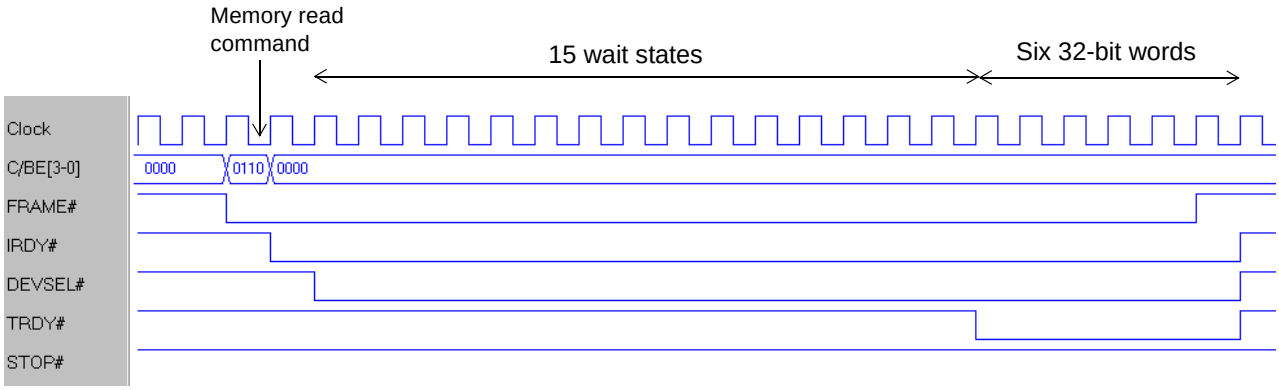
A different PCI core configuration was generated with the FPSC kit and the VHDL code was modified for a *Dual-port* interface to the PCI core. The corresponding bitstream was loaded from the MCU into the EBI FPSC using the PCI bus. Figure 68 shows a snapshot of the MCU terminal in response to a < **lspci -xvv >** Linux command. Line 7 shows that memory region 0 (where the SEB is mapped) is defined as prefetchable. This region is mapped starting at address OxC0200000.

```
01:0f.0 Signal processing controller: Lucent Microelectronics: Unknown device 5401 (rev 01)
Subsystem: Unknown device beef:dead
Control: I/O+ Mem+ BusMaster+ SpecCycle- MemWINV- VGASnoop- ParErr- Stepping- SERR- FastB2B-
Status: Cap+ 66Mhz+ UDF- FastB2B+ ParErr- DEVSEL=medium >TAbort- <TAbort- <MAbort- >SERR- <PERR
Latency: 255 min, 255 max, 64 set
Interrupt: pin A routed to IRQ 11
Region 0: Memory at c0200000 (32-bit, prefetchable)
Region 5: Memory at c0100000 (32-bit, non-prefetchable)
Capabilities: [50] #06 [0080]
00: c1 11 01 54 07 00 b0 02 01 00 80 11 00 40 00 00
10: 08 00 20 c0 00 00 00 00 00 00 00 00 00 00 00 00
20: 00 00 00 00 00 00 10 c0 00 00 00 00 ef be ad de
30: 00 00 00 00 50 00 00 00 00 00 00 00 0b 01 ff ff
```

Figure 68 MCU terminal dump

Measurements show correct behaviour for 64-bit transactions, but some word-swapping when transactions are 32-bit wide. Once again, wrong multiplexing in the PCI core FIFOs is suspected. Figure shows a single-word read operation to DPM initiated by the MCU. As the DPM is mapped in prefetchable space, the bridge initiates a burst transactions of variable length (determined by the bridge instantaneous buffer space availability).

Figure 69 Single-word DPM read from the MCU



12.2 Control and monitoring registers on the FPSC

For all the bitstreams mentioned in this chapter, five 32-bit registers have been defined in the FPSC's BAR 5 space. These user-defined registers are the way to implement configurable parameters, error counters and monitoring registers in the FPSCs accessible from the MCU via PCI (see data path in Figure 64). Figure 69 shows the result of the command ***rwpci ebi2a WREG 100 100*** (100h is written to the 256th 32-bit word -offset 400h- followed by a read for verification).

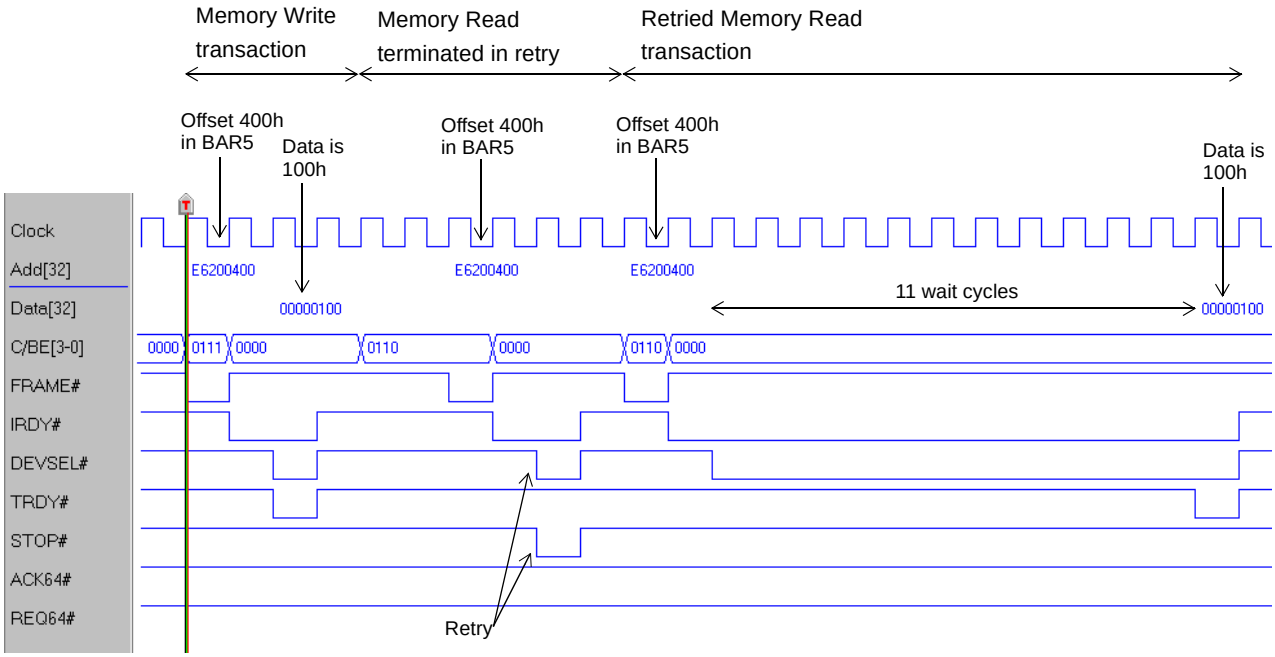
The registers are mapped in the following addresses:

Table 7 User-defined register mapping

Register	Offset (bytes)	Offset (32-bit words)
Reg 1	100h	40h
Reg 2	200h	80h
Reg 3	300h	C0h
Reg 4	400h	100h
Reg 5	500h	140h

A target retry is generated as a first response to the read, and 12 wait state cycles are inserted by the FPSC after the retry (one cycle for DEVSEL# assertion and 11 cycles due to the user logic and the embedded PCI core delays. User logic delay is in the order of one or two cycles and cannot be further reduced).

Figure 70 Access to a user defined register in the FPSC: write followed by a read



13 Readout Unit performance tests

PCI performance has been measured on three different scenarios:

d. EBI2 FPSC write to EBI1 FPSC

e. L1-VELO application emulation: EBI FPSCs write to a NIC card

f. DAQ application emulation: TA-700 (as PCI master) read from EBI FPSCs

13.1 EBI2 FPSC writes to EBI1 FPSC

This test shows the maximum performance achievable with a single FPSC in the output stage. A *Dual-Port*-mode PCI master write interface is implemented in VHDL in EBI2 FPSC, together with a set of registers[1] accessible via PCI. The MCU sets the burst length (in 64-bit words), destination address (start of DPM in EBI1 FPSC) and go/stop bit through these registers (see Figure). When enabled via the Go/Stop bit, EBI2 will continuously produce write transactions to EBI1.

This test can be carried out loading the bitstream file *hope64.bit* in EBI1 and *ebi2mw.bit* in EBI2 and running the line command ***rwpci ebi2a TESTWREBI1 length 1*** (where ***length*** is the desired number of 64-bit words). This command will configure the registers REG2 to REG4 in EBI2 and initiate the test. To stop it, run ***rwpci ebi2a TESTWREBI1 length 0***.

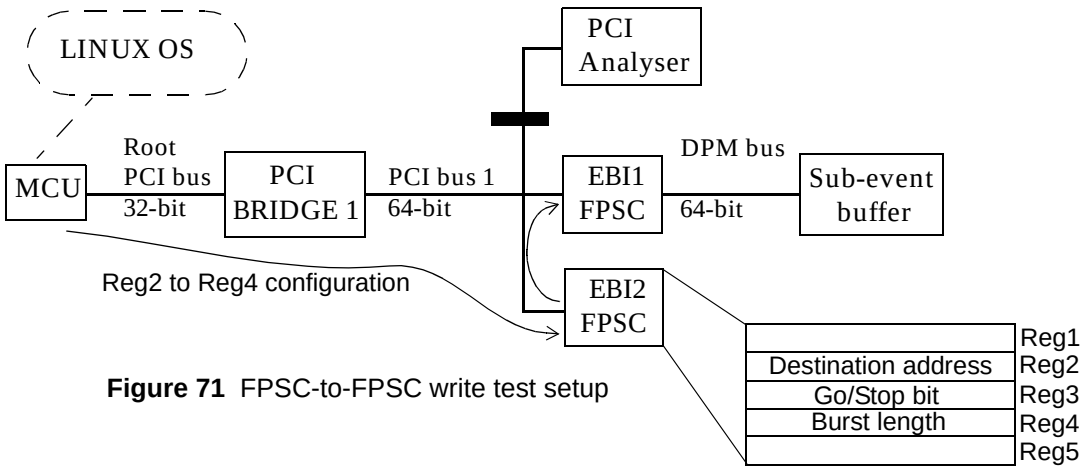


Figure 71 FPSC-to-FPSC write test setup

1. See Section 12.2 in page 73

Figure shows a 64-bit-wide 80-byte transfer between the two FPSCs, where a single wait state is inserted by the target at the beginning of the transaction (thus, the FPSCs are fast-decoding devices according to the PCI specification). Figure 73 shows that two consecutive transactions are spaced by ten clock cycles, and this is true regardless of the burst length.

Figure 72 8-word burst transaction from EBI2 to EBI1

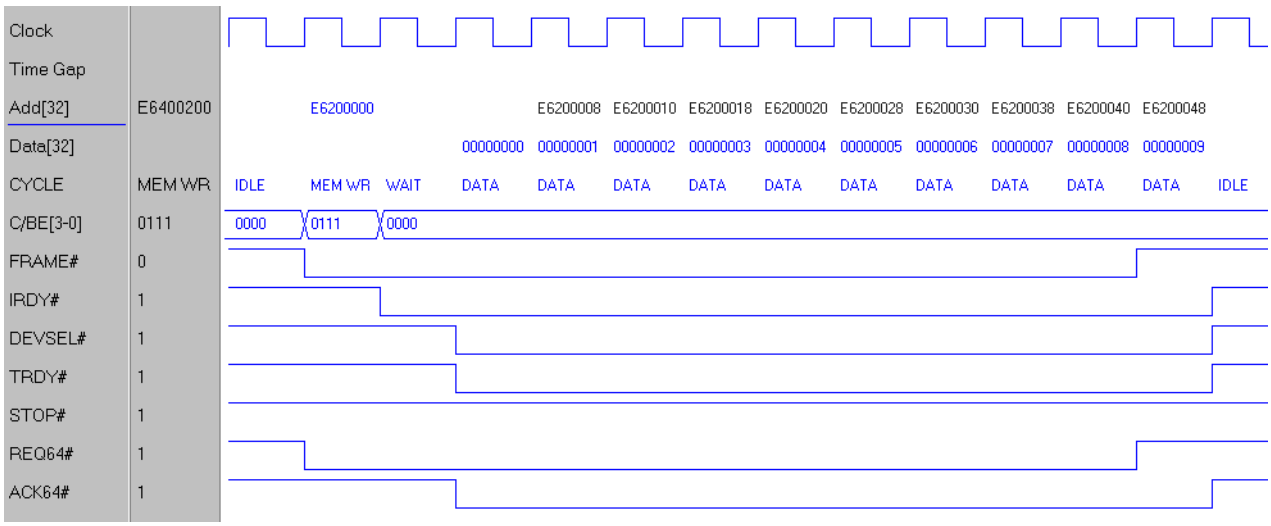
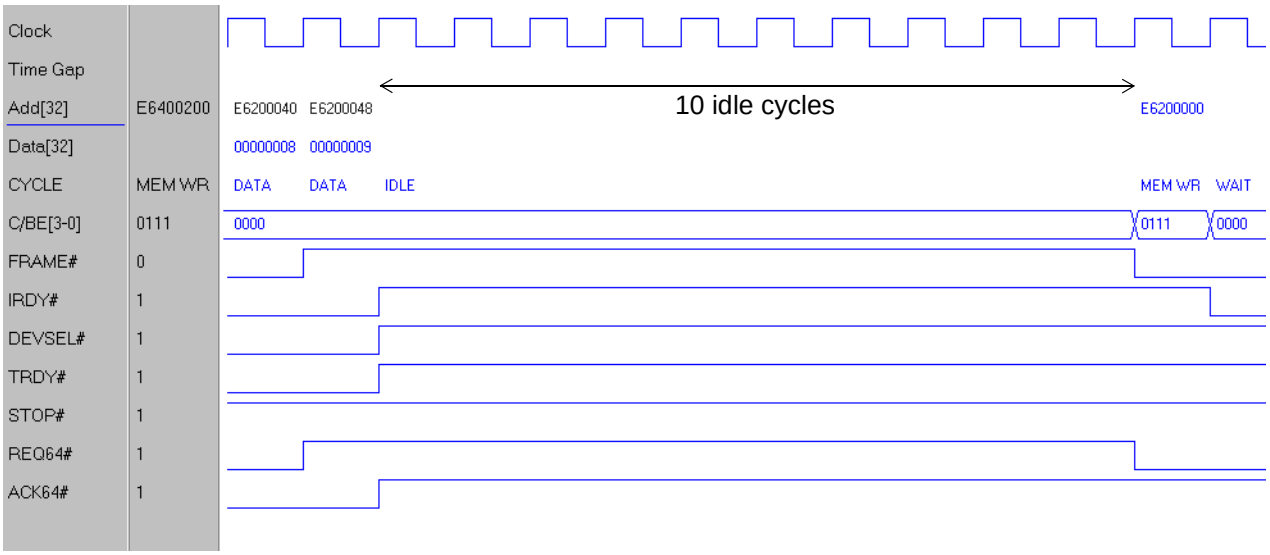


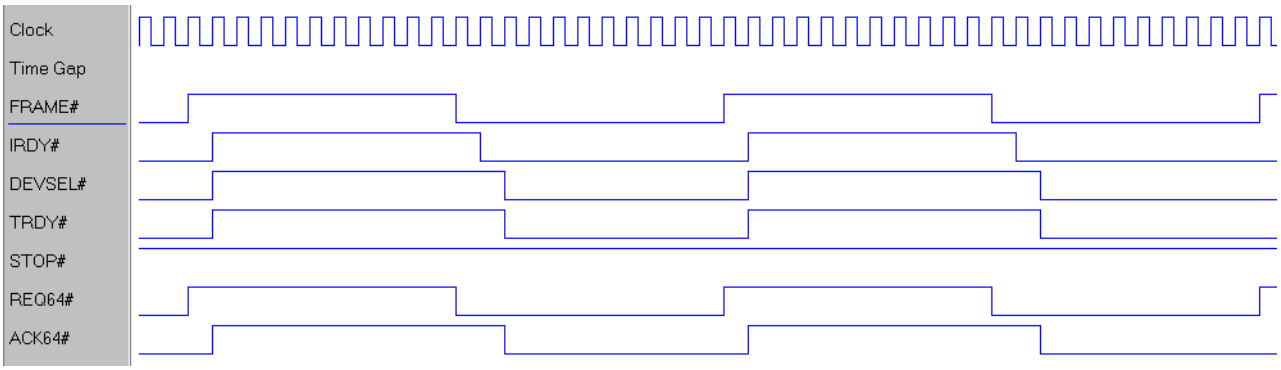
Figure 73 Ten empty clock cycles between transactions



The first empty cycle is a turnaround cycle (the current master releasing the bus after a transaction completion). The other nine are inserted by the FPSC embedded PCI core and cannot be avoided. If we take into account that for the most performance-demanding application -L1-VELO- will require in average 16-quadword bursts, the maximum achievable

performance in the PCI bus would be (counting 10 idle cycles, one wait state and one address cycle) 16/28, i.e., **57%** of the available bus bandwidth, a poor figure. This makes 220 Mbyte/s at 50 MHz clock speed or 150 Mbyte/s at 33 MHz for 64-bit transactions. This 57% duty cycle is shown in Figure .

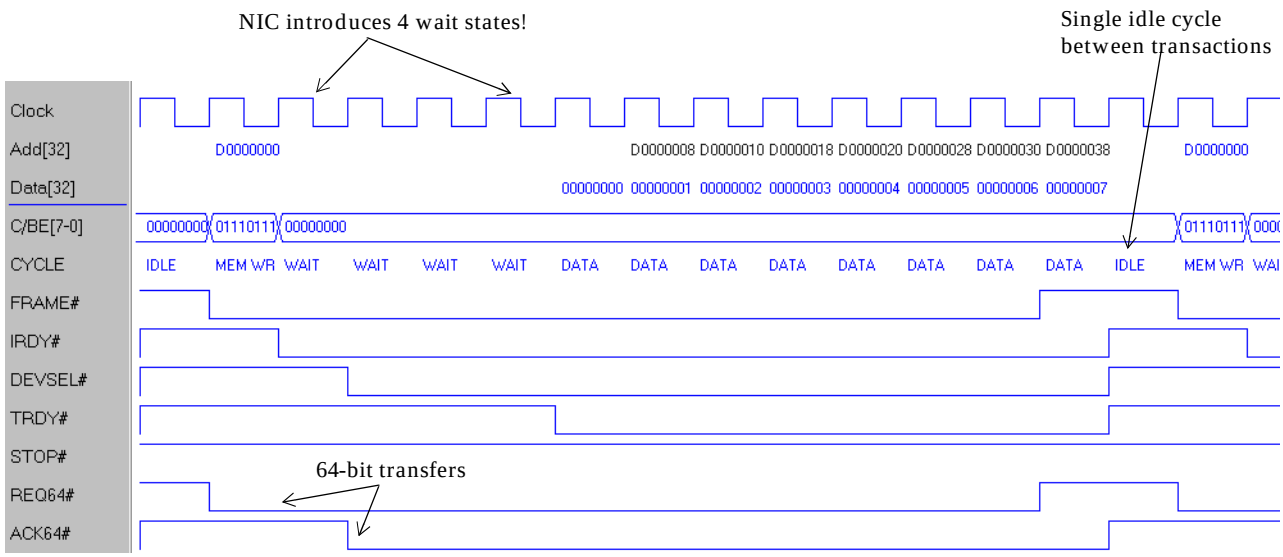
Figure 74 Less than 60% bus bandwidth available



13.2 L1-VELO application emulation: EBI FPSCs write into a NIC card

The tandem-master DMA operation [27] makes an optimum use of the PCI bandwidth, as shown in Figure 75, where a Dolphin[1] PCI-to-SCI adapter with 64-bit PCI interface has been used as NIC. This card introduces four wait states during a target write. The latest NIC cards may respond faster and thus increase the bus efficiency (for instance, the FPSC PCI interfaces responds in just one clock cycle). For the sake of comparison with the scenario described in Section 13.1, a single wait cycle is assumed for the efficiency calculations.

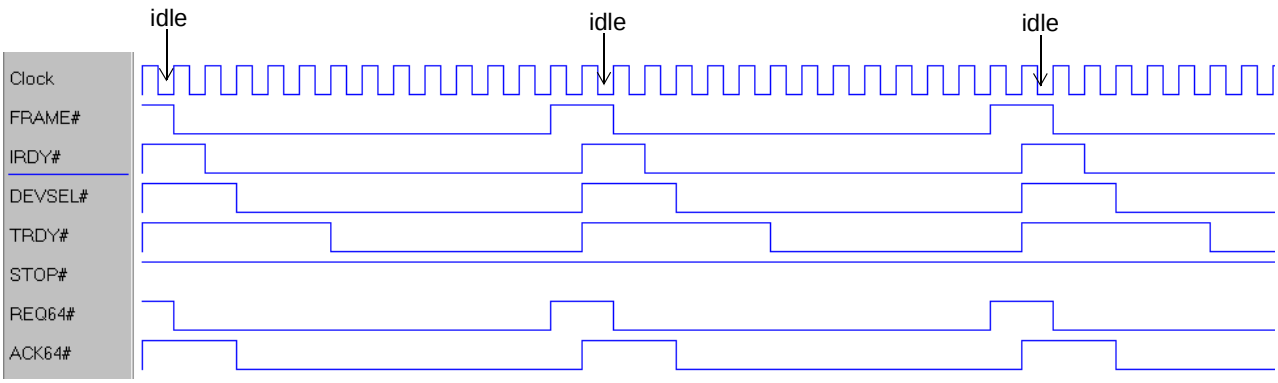
Figure 75 64-byte burst write to a Dolphin SCI NIC (LC4000 and L5B9350 chips)



1. www.dolphinics.com

A 16-quadword (128 byte) transfer would take 19 cycles (16 data, 1 address, 1 wait and 1 turnaround), resulting in 16 / 19 bus bandwidth efficiency (84%). This yields 220 Mbyte / s at 33 MHz or 336 Mbyte / s at 50 MHz. Figure 76 shows bus activity for tandem operation.

Figure 76 A single idle cycle between transactions can be achieved

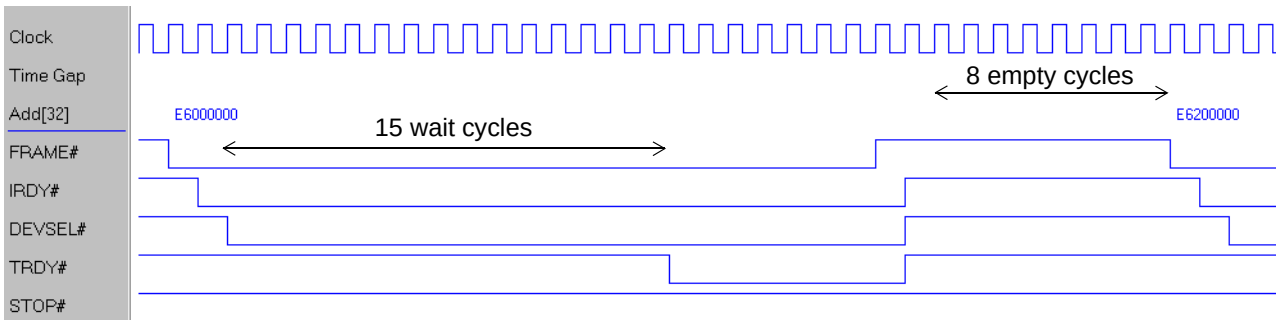


13.3 DAQ application emulation: TA-700 read EBI FPSCs

The PCI bus performance for the DAQ application has been measured using the TA-700 bus analyser as PCI master reading alternatively the two EBI FPSCs. As it can be seen in Figure 77, the FPSCs add 15 wait states before returning data, the same result as in measurement in Figure (the eleven cycles and four additional cycles due to the DPM first-word latency and user logic state machines). The TA-700 adds a turnaround cycle plus eight empty cycles between transactions and a real NIC may require a similar number of cycles (ten in the case of a FPSC, see Figure 73).

Long burst transactions (500-1000 byte) will take place in the DAQ application, reducing the impact of these 15 wait cycles. The maximum bus usage for a 500-byte sub-event fragment, taking into account the protocol overhead (1 address cycle, 1+15 wait cycles, 8 empty cycles and 1 turnaround cycle) is 83% for 32-bit transactions and 71% for 64-bit transactions. .

Figure 77 FPSC PCI read initiated by the TA-700 bus analyse



This makes 110 Mbyte / s and 187 Mbyte / s respectively at 33 MHz PCI clock operation

14 Software for test and FPSC configuration

Before attempting any transaction, the MCU needs to know the address in which the SEB is mapped (by using **pci.h** library functions to read BAR0 and BAR5 registers in the FPSC PCI configuration space). The value of these registers is masked (to have 0h in the lower 4 four bits) and used as an input to the **mmap** function to get a pointer to the SEB (see example code in Figure 78).

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/mman.h>
#include <sys/types.h>
#include <pci/pci.h>
#include <sys/stat.h>
#include <fcntl.h>
#define ORCA_BAR0 0x10

int main(int argc, char *argv[])
{
  int i,fd,dev_num, bus_num;
  u32 bar0_addr,position;
  struct pci_access *pci_acc;
  struct pci_dev *orca_config_space;
  unsigned long *ptr,value,readback;

  /* Scan the PCI bus, and store the config space in pci_acc */
  pci_acc = pci_alloc();
  pci_init(pci_acc);
  pci_scan_bus(pci_acc);

  /* Select fpsc to access */
  if(strcmp(argv[1],"ebi1")==0)
bus_num = 0x1; dev_num = 0xF; /* EBI1 */
...
...

/* Verify if an orca is there */
orca_config_space=pci_get_dev(pci_acc, bus_num,
dev_num,0);
bar0_addr = pci_read_long(orca_config_space,ORCA_BAR0)
& PCI_BASE_ADDRESS_MEM_MASK;

if(pci_read_long(orca_config_space,0) != 0x540111c1)
 {printf("Can't find orca board\n"); return(-1);}
else { printf("Orca board has been found ...\n");
fd = open("/dev/mem", O_RDWR);
ptr = (unsigned long *) mmap(
NULL, 128, PROT_READ | PROT_WRITE,
MAP_SHARED, fd, bar0_addr);
/*----Read/write memory according to user command----*/
 if (strcmp(argv[2],"W")==0)
 {
  value= (unsigned long) atol(argv[3]);
  position = (u32) atol(argv[4]);
  *(ptr+position) = value;
  printf("%x written to position ", value);
  printf("[%x]",position);
  readback = *(ptr+position);
  if (readback == value) printf("...[OK]\n");
  else printf("...error: readback %x",readback);
 }
...
...
/* free buffer before leaving */
 munmap(ptr, 128);
 return 0;
      }
}
```

Figure 78 Example of C code to access PCI-mapped memory in the FPSC

14.0.1 The *flasher* utility

Developed by NA-60 for the PCI-FLIC card[1], this utility to load a bitstream via PCI into an FPSC chips is fully functional on the Readout Unit II. The flasher utility must be called using the following convention:

flasher device bitstream_file

Where ***device*** is ***bus:device.function*** and ***bitstream_file*** is a ".bit" file created by Orca Foundry tools. Example of usage: "***flasher 1:f.0 hope7.bit"***.

1. http://hmuller.home.cern.ch/hmuller/pci-flic.htm

Three different bitstreams (*hope7.bit*, *ebi2mw.bit* and *hope64.bit*) have been used for the tests described in this chapter. Common to the three bitstreams are a read/write PCI target interface that grants access to the DPM via the BAR0 and five user-defined registers located in BAR5 offset 100h to 500h.

The files *ebi2mw.bit* and *hope64.bit* use a *dual port* interface to the PCI core, whilst *hope7.bit* uses a *quad port* interface. The file *ebi2mw.bit* includes additionally a write PCI master interface and defines BAR0 area as prefetchable. The combination of bitstreams for the different tests are shown in Table 8.

Table 8 Bitstreams used for the different tests

Test	EBI1 bitstream	EBI2 bitstream
Quad-port access to DPM	*hope7.bit*	*(not used for the test)*
Dual-port access to DPM	*ebi2mw.bit*	*(not used for the test)*
FPSC-to-FPSC write	*hope64.bit*	*ebi2mw.bit*
L1-VELO application emulation	*ebi2mw.bit*	*ebi2mw.bit*
DAQ application emulation	*hope64.bit*	*hope64.bit*

14.0.2 The *rwpci* utility

This utility must be called using the following conventions:

- ***rwpci device R n***: Reads the *n*th 32-bit DPM memory position. The current FPSC test code bitstreams (see Section) just return the lower 32 bits of the addressed 64-bit word. That is, the command *rwpci ebi1a R 3* returns the lower 32 bits of the fourth 64-bit word in DPM. The VHDL code can be easily modified to match the command function
- ***rwpci device W val n***: Writes *val* into the *n*th DPM memory position and reads back the position for verification (the used FPSC bitstreams only work with the lower 32 bits of each 64-bit memory position)
- ***rwpci device I val***: Writes *val* to the first 32 DPM positions (the used FPSC bitstreams only work with the lower 32 bits of each 64-bit memory position)
- ***rwpci device INC val***: Writes the first 32 DPM positions with a self-incrementing counter value starting with *val* (the used FPSC bitstreams only work with the lower 32 bits of each 64-bit memory position)
- ***rwpci device DUMP***: Prints in the screen the first 32 DPM memory positions (the used FPSC bitstreams only work with the lower 32 bits of each 64-bit memory position)
- ***rwpci device RREG pos***: reads the user-defined 32-bit register located in offset BAR5+*pos*
- ***rwpci device WREG val pos***: writes *val* into the user-defined 32-bit register located in offset BAR5+*pos* and verifies with a read operation

- ***rwpci device TESTWREBI1 burst_cnt go***: Configures the destination address, burst count in quadwords and go/stop bit in EBI2 FPSC to initiate a sequence of PCI write bursts to EB1 FPSC (see Section 13.1). The transactions can be stopped writing a 0 in *go* and enabled writing a 1. (Note: data are generated with a counter, do not come from the SEB in the current FPSC bitstreams. The FPSC will transfer *burst_cnt+2* quadwords per transfer)
- ***rwpci device TESTWRNIC burst_cnt go bar_nr***: Used to perform write tests to a PCI device plugged on the NIC slot (seeSection 13.2). Configures the destination address -according to the selected BAR *bar_nr* in the NIC- burst count in quadwords and go/stop bit in the selected device. The transactions can be stopped writing a 0 in *go* and enabled writing a 1. (Note: data are generated with a counter, do not come from the SEB in the current FPSC bitstreams. The FPSC will transfer *burst_cnt+2* quadwords per transfer)

The parameter ***device*** can be *sem1*, *sem2*, *ebi1*, *ebi2* (if the MCU is used) or *sem1a*, *sem2a*, *ebi1a*, *ebi2a* if the Advantech card is used, as each processor card uses a different bus numbering. The C code for the *rwpci* utility can be found below:

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/mman.h>
#include <sys/types.h>
#include <pci/pci.h>
#include <sys/stat.h>
#include <fcntl.h>

#define ORCA_BAR0 0x10
#define BAR1 0x14
#define BAR2 0x18
#define BAR3 0x1C
#define BAR4 0x20
#define ORCA_BAR5 0x24


/*.....................MAIN.....................*/

int main(int argc, char *argv[])
{

  int i,fd,fd_reg,dev_num, bus_num;
  u32 bar0_addr,bar5_addr,position,bar_aux;
  struct pci_access *pci_acc;
  struct pci_dev *orca_config_space, *orca_aux;
  unsigned long *ptr,*ptr_reg,value,readback;

  /* Scan the PCI bus, and store the config space in pci_acc */
  pci_acc = pci_alloc();
  pci_init(pci_acc);
  pci_scan_bus(pci_acc);


  /* Select fpga to access */
  if(strcmp(argv[1],"ebi1")==0) { bus_num = 0x1; dev_num = 0xF;} /* EBI1
MCU*/
   else if(strcmp(argv[1],"ebi2")==0) { bus_num = 0x1; dev_num = 0xE;}/*
EBI2 MCU*/
  else if(strcmp(argv[1],"sem1")==0) { bus_num = 0x2; dev_num = 0xF;} /*
SEM1 MCU*/
  else if(strcmp(argv[1],"sem2")==0) { bus_num = 0x2; dev_num = 0xE;} /*
SEM2 MCU*/
  else if(strcmp(argv[1],"ebi1a")==0) { bus_num = 0x2; dev_num = 0xF;}/*
EBI1 ADVANTEK*/
  else if(strcmp(argv[1],"ebi2a")==0) { bus_num = 0x2; dev_num = 0xE;}/*
EBI2 ADVANTEK*/
  else if(strcmp(argv[1],"sem1a")==0) { bus_num = 0x3; dev_num = 0xF;}
/* SEM1 ADVANTEK*/
  else if(strcmp(argv[1],"sem2a")==0) { bus_num = 0x3; dev_num = 0xE;}
/* SEM2 ADVANTEK*/
     else { printf("Bad argument, DEVICE must be 'sem1', 'sem2',
'ebi1','ebi2,'\n");
    printf("sem1a, sem2a, ebi1a or ebi2a\n"); return -1; }

  /* Verify if an orca is there */
  orca_config_space = pci_get_dev(pci_acc, bus_num, dev_num,0);
  /*printf("Bus_num = 0x%x ", bus_num);
  printf("and dev_num = 0x%x\n",dev_num);
                printf("PCI selected location is: %x\n",
pci_read_long(orca_config_space,0));*/
       bar0_addr = pci_read_long(orca_config_space,ORCA_BAR0)&
PCI_BASE_ADDRESS_MEM_MASK;
       bar5_addr = pci_read_long(orca_config_space,ORCA_BAR5)&
PCI_BASE_ADDRESS_MEM_MASK;
  /*printf("BAR0 has value: %x\n", bar0_addr);*/

  if(pci_read_long(orca_config_space,0) != 0x540111c1)
  {printf("Can't find orca board\n"); return(-1);}
  else { printf("Orca board has been found ...\n");
  /*------------Initialize memory---------------------------------*/
  fd = open("/dev/mem", O_RDWR);
    ptr = (unsigned long *) mmap(
NULL,
0x700,
PROT_READ | PROT_WRITE,
MAP_SHARED,
fd,
bar0_addr);
 fd_reg = open("/dev/mem", O_RDWR);
    ptr_reg = (unsigned long *) mmap(
NULL,
0x700,
PROT_READ | PROT_WRITE,
MAP_SHARED,
fd_reg,
bar5_addr);
```

continued next page

```
 /*----------------Read/write memory according to user command----*/
 if (strcmp(argv[2],"W")==0)
 {
  value= (unsigned long) strtoll(argv[3],(char**)NULL,16);
  position = (u32) strtoll(argv[4],(char**)NULL,16);
  *(ptr+position) = value;
  printf("%x written to position ", value);
  printf("[%x]",position);
  readback = *(ptr+position);
  if (readback == value) printf("...[OK]\n");
  else printf("...error: readback %x",readback);
 }
 else if (strcmp(argv[2],"WREG")==0)
 {
  value= (unsigned long) strtoll(argv[3],(char**)NULL,16);
  printf("%x written to register ", value);
  position = (u32) strtoll(argv[4],(char**)NULL,16);
  printf("[%x]",position);
  *(ptr_reg+position) = value;
  readback = *(ptr_reg+position);
  if (readback == value) printf("...[OK]\n");
  else printf("...error: readback %x",readback);
 }
 else if (strcmp(argv[2],"TESTWREBI1")==0)
 {
  orca_aux = pci_get_dev(pci_acc,0x2,0xF,0); /*access to ebi1a*/
                                                bar_aux                    =
pci_read_long(orca_aux,ORCA_BAR0)&PCI_BASE_ADDRESS_MEM_
MASK; /*read BAR0_ebi1a*/
  *(ptr_reg+0x80) = bar_aux; /*write BAR0_ebi1a in REG2_ebi2a*/
  value= (unsigned long) strtoll(argv[3],(char**)NULL,16);
  *(ptr_reg+0x100) = value; /*write burst length in REG4_ebi2a*/
  value= (unsigned long) strtoll(argv[4],(char**)NULL,16);
  *(ptr_reg+0xc0) = value; /*write go in REG3_ebi2a*/
  printf ("Destination address: %x...\n",*(ptr_reg+0x80));
  printf ("Burst length: %x...\n",*(ptr_reg+0x100));
  printf ("Go_write: %x...",*(ptr_reg+0xc0));
  printf("[DONE]\n");
 }
 else if (strcmp(argv[2],"TESTWRNIC")==0)
 {
  orca_aux = pci_get_dev(pci_acc,0x2,0xD,0); /*access to NIC*/
  bus_num= (int) strtoll(argv[5],(char**)NULL,16); /*which BAR?*/
  /*read BARn_NIC*/
  if (bus_num==0) {bar_aux = pci_read_long(orca_aux,ORCA_BAR0);}
  else if (bus_num==1) {bar_aux = pci_read_long(orca_aux,BAR1);}
  else if (bus_num==2) {bar_aux = pci_read_long(orca_aux,BAR2);}
  else if (bus_num==3) {bar_aux = pci_read_long(orca_aux,BAR3);}
  else if (bus_num==4) {bar_aux = pci_read_long(orca_aux,BAR4);}
                  else          if          (bus_num==5)          {bar_aux          =
pci_read_long(orca_aux,ORCA_BAR5);}
  else {printf("Bad argument, BAR is beween 0 and 5.\n"); return -1;}

      *(ptr_reg+0x80)   =   bar_aux&PCI_BASE_ADDRESS_MEM_MASK;
/*write BARn_NIC in REG2_ebia*/
  value= (unsigned long) strtoll(argv[3],(char**)NULL,16);
  *(ptr_reg+0x100) = value; /*write burst length in REG4_ebia*/
  value= (unsigned long) strtoll(argv[4],(char**)NULL,16);
  *(ptr_reg+0xc0) = value; /*write go in REG3_ebi2a*/
          printf      ("Destination      address      (selected      NIC      BAR):
%x...\n",*(ptr_reg+0x80));
  printf ("Burst length: %x...\n",*(ptr_reg+0x100));
  printf ("Go_write: %x...",*(ptr_reg+0xc0));
  printf("[DONE]\n");
 }
 else if (strcmp(argv[2],"R")==0)
 {
  position = (u32) strtoll(argv[3],(char**)NULL,16);
  printf("Reading position %x",position);
  readback = *(ptr+position);
  printf(" returns %x\n",readback);
 }
 else if (strcmp(argv[2],"RREG")==0)
 {
  position = (u32) strtoll(argv[3],(char**)NULL,16);
  printf("Reading register %x",position);
  readback = *(ptr_reg+position);
  printf(" returns %x\n",readback);
 }
  else if (strcmp(argv[2],"I")==0)
  {
    value= (unsigned long) strtoll(argv[3],(char**)NULL,16);
    printf("Resetting memory to %x...",value);
    for(i=0; i<(32); i++)
    {
      *(ptr+i) = value;
       if (i%4==0) printf("\n[%x]\t",i);
     printf("%8.8lx \t", *(ptr+i));
     if ((i+1)%4==0) printf("\n");
   }
  }
  else if (strcmp(argv[2],"INC")==0)
  {
    value= (unsigned long) strtoll(argv[3],(char**)NULL,16);
    printf("Resetting memory to %x...",value);
    for(i=0; i<(32); i++)
    {
      *(ptr+i) = value+i;
       if (i%4==0) printf("\n[%x]\t",i);
     printf("%8.8lx \t", *(ptr+i));
     if ((i+1)%4==0) printf("\n");
   }
  }
  else if (strcmp(argv[2],"DUMP")==0)
  {
    printf("Memory Dump...\n");
    for(i=0; i<(32); i++)
    {
       if (i%4==0) printf("\n[%x]\t",i);
     printf("%8.8lx \t", *(ptr+i));
     if ((i+1)%4==0) printf("\n");
   }
  }
                  else          printf          ("Command          must          be
R,RREG,W,WREG,TESTWREBI1,TESTWRNIC,INC,DUMP or I");
   /* free buffer before leaving  */
 munmap(ptr, 0x700);
 munmap(ptr_reg,0x700);
 return 0;
      }
}
```

14.1 Conclusions

The PCI subsystem has been tested at 33-MHz clock frequency (Advantech card) and 18 MHz (MCU). The tested features include:

1. FPSCs bitstream configuration from the MCU (with and without flash memories on the RU-II)
2. MCU access to the user-defined registers via PCI
3. MCU access to the DPM via PCI
4. PCI write operations between FPSCs
5. DAQ application emulation (requiring a target read / write PCI interface)
6. VELO application emulation (requiring additionally a master write PCI interface)

The results of the first three points in the above list are exportable to the input-stage's PCI subsystem. FPSC's PCI interface weak and strong points (like the single wait cycle as target write, the fifteen wait cycles as target read or the ten empty cycles between transactions as master write) have been identified and their impact on the VELO and DAQ applications studied. It has been also observed that under certain circumstances, in both the *Quad port* and the *Dual port* modes, the core swaps erroneously the lower and upper dualword in the cores' 64-bit FIFOs (see Section 12.1.3 and Section 12.1.4).

PCI bus bandwidth optimization via the tandem-FPSC operation has been demonstrated in laboratory measurements, as well as the correct operation of the output-stage PCI subsystem hardware in the Readout Unit II. A master read PCI interface should be developed in order to properly implement the VELO and DAQ application protocols between the FPSCs and the NIC.

Further work should be done on debugging the VHDL state machines for accessing the DPM, as the current test code does not work properly. Additionally, the *ebi2mw.bit* code returns wrong values as a result of MCU-initiated read operations to the user-defined registers (a problem with multiplexing).

APPENDIX

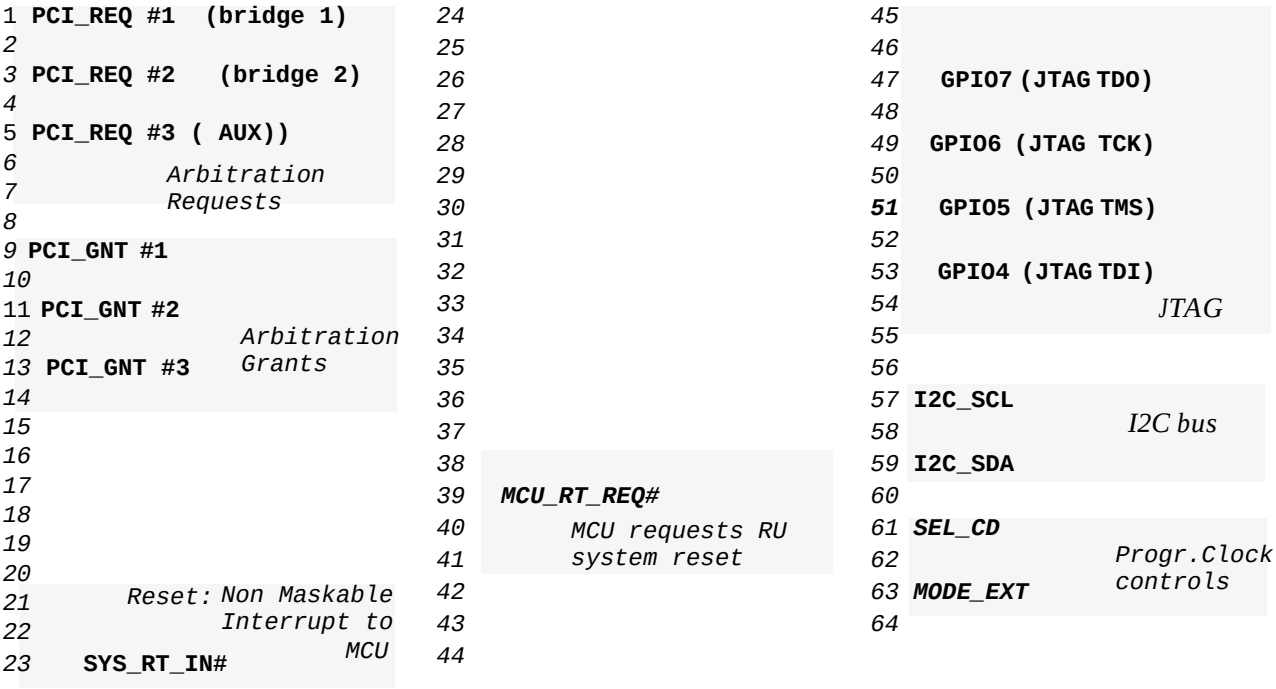
1 Mezzanine connector pinouts

The RU uses 64 pin mezzanine connectors for PCI, SLINK, and USER I/O. The naming and position of the mezzanine connectors is shown in chapter 9.5 .

1.1 User I/O connector (P14) pinout

The 64 pin I/O connector pinout of P14 which belongs optionally to the MCU card on the root PCI segment has the pinout shown in Table 9.

Table 9 User-defined I/O connector (P14) functions used by the RU



1.2 Slink connector pinout

The RU uses the Slink connector standard (see 8.2) for the input links and also for the optional output link.

Table 10 A Tx output P01

1	LRL3	LRL2	2
3	Vcc	LRL1	4
5	Vcc	LRL0	6
7	LDOWN#	GND	8
9	GND	LFF#	10
11	UCLK	GND	12
13	GND	UWEN#	14
15	URESET#	GND	16
17	UDW1	UTEST#	18
19	UCTRL#	UDW0	20
21	ud31	vCC	22
23	gnd	ud30	24
25	ud29	ud28	26
27	ud27	gnd	28
29	ud26	ud25	30
31	gnd	ud24	32
33	ud23	ud22	34
35	ud21	gnd	36
37	ud20	ud19	38
39	vCC	ud18	40
41	ud17	ud16	42
43	ud15	gnd	44
45	ud14	ud13	46
47	gnd	ud12	48
49	ud11	ud10	50
51	ud9	vCC	52
53	ud8	ud7	54
55	gnd	ud6	56
57	ud5	ud4	58
59	ud3	gnd	60
61	ud2	ud1	62
63	vCC	ud0	64

Table 10 B Rx input P01

1	LRL3	LRL2	2
3	Vcc	LRL1	4
5	LDERR#	LRL0	6
7	LDOWN#	GND	8
9	GND	UXOFF#	10
11	LCLK	GND	12
13	GND	LWEN#	14
15	URESET#	GND	16
17	UDW1	UDT0#	18
19	LCTRL#	UDW0	20
21	ld31	vCC	22
23	gnd	ld30	24
25	ld29	ld28	26
27	ld27	gnd	28
29	ld26	ld25	30
31	gnd	ld24	32
33	ld23	ld22	34
35	ld21	gnd	36
37	ld20	ld19	38
39	vCC	ld18	40
41	ld17	ld16	42
43	ld15	gnd	44
45	ld14	ld13	46
47	gnd	ld12	48
49	ld11	ld10	50
51	ld9	vCC	52
53	ld8	ld7	54
55	gnd	ld6	56
57	ld5	ld4	58
59	ld3	gnd	60
61	ld2	ld1	62
63	vCC	ld0	64

The mezzanine connector P01 pinout (see also 9.5) for Slink receiver and transmitters is shown above. The pinouts are very similar for Rx and Tx cards.

1.3 PMC connector pinout

PMC connectors (PCI): Table 11 shows the PCI pinout of a PMC on the mandatory 64 pin connectors P11 and P12 (32 bit PCI bus) and P13 for PCI-64 option. . .

mandatory P11 PCI connector

1	TCK		2	-12V
3	GND		4	INTA#
5	INTB#		6	INTC#
7	Busmode1#		8	+5V
9	INTD#		10	reserved
11	GND		12	reserved
13	CLK		14	GND
15	GND		16	GNT#
17	REQ#		18	+5V
19	V(I/O)		20	AD31
21	AD28		22	AD27
23	AD25		24	GND
25	GND		26	CBE3#
27	AD22		28	AD21
29	AD19		30	+5V
31	V(I/O)		32	AD17
33	FRAME#		34	GND
35	GND		36	IRDY#
37	DEVSEL#		38	+5V
39	GND		40	LOCK
41	SDONE#		42	SB0#
43	PAR		44	GND
45	V(I/O)		46	AD15
47	AD12		48	AD11
49	AD9		50	+5V
51	GND		52	CBE0#
53	AD6		54	AD5
55	AD4		56	GND
57	V(I/O)		58	AD3
59	AD2		60	AD1
61	AD0		62	+5V
63	GND		64	REQ64#

mandatory P12 PCI connector

1	+12		2	TRST
3	TMS		4	TD0
5	TDI		6	GND
7	GND		8	reserved
9	reserved		10	reserved
11	BUSMODE 2#		12	+3.3V
13	RST#		14	BUSMODE3#
15	+3.3V		16	BUSMODE4#
17	reserved		18	GND
19	AD30		20	AD29
21	GND		22	AD26
23	AD24		24	+3.3V
25	IDSEL		26	AD23
27	+3.3V		28	AD20
29	AD18		30	GND
31	AD16		32	CBE2#
33	GND		34	reserved
35	TRDY#		36	+3.3V
37	GND		38	STOP#
39	PERR		40	GND
41	+3.3V		42	SERR#
43	CBE1#		44	GND
45	AD14		46	AD13
47	GND		48	AD10
49	AD8		50	+3.3V
51	AD7		52	reserved
53	+3.3V		54	reserved
55	reserved		56	GND
57	reserved		58	reserved
59	GND		60	reserved
61	ACK64#		62	+3.3V
63	GND		64	reserved

optional 64 bit P13 PCI connector

1	PCI-RSVD		2	GND
3	GND		4	CE/BE[7]
5	C/BE[6]		6	CE/BE[5]
7	C/BE[4]		8	GND
9	V(I/O)		10	PAR64
11	AD63		12	AD62
13	AD61		14	GND
15	GND		16	AD60
17	AD59		18	AD58
19	AD57		20	GND
21	V[1/o]		22	AD56
23	AD55		24	AD54
25	AD53		26	GND
27	GND		28	AD52
29	AD51		30	AD50
31	AD49		32	GND
33	GND		34	AD48
35	AD47		36	AD46
37	AD45		38	GND
39	V[I/o]		40	AD44
41	AD43		42	AD42
43	AD41		44	GND
45	GND		46	AD40
47	AD39		48	AD38
49	AD37		50	GND
51	GND		52	AD36
53	AD35		54	AD34
55	AD33		56	GND
57	V[I/O]		58	AD32
59	PCI-RSVD		60	PCI-RSVD
61	PCI-RSVD		62	GND
63	GND		64	PCI-RSVD

Table 11 Pinouts of PMC PCI connectors

2 Cost

The cost estimates are based on manufacturing bills via intermediate of the facilities at CERN. No other commercial margins are included.

2.1 Development and low quantity prototypes

Table 12 shows an overview over all cards related to the Readout Unit (small Qty cost in CHF)**Note that in the total cost column, development and tooling costs (shaded columns) are not included !**

Table 12 Cost for low quantity (~ 10) electronic cards related to the Readout Unit

Card Name	Allegro layout	films + tooling	Mounting mask & Tool	PCB (FR4)[a]	Components	Component Mounting	Frontpanel + mechanics etc	Total (FR4)
Readout Unit[b c] **(8 layer 9U)**	13300	2185	1900	425 (6 pc)	2820	1200 (155 for 100+)	200	**4770**
Slink I/O[d] **(4 layer PMC)**	1850	900	1000	100	160	100	-50	~**410**
Slink testgen-erator [e]**(PMC)**	945			100	180	100	-50	~**430**
MCU card[f g] **(12 layer PMC)**	5900	1460	1480	130 (10 pc)	530	350 (100 for 100+)	-50	**1060**
Quad MUX[h] Slink **(8 layer PMC)**	3500	915		71.50 (20 pc)				**New ~600**

a. in halogene free material, 1585 CHF (5 pc)
b. RU Card reference at CERN EP 680 1150 400
c. RU printed circuit: ML8;Cl5;FR4 24/10;Cu35µm;CMS2;Ni-Au;366.7 x 400mm (9U Fastbus)
d. Slink I/O card reference at CERN: EP-680-1150-500
e. Testcard Reference EP-680-1150-250
f. MCU Card Reference at CERN EP 680 1150 450
g. MCU printed circuit: ML8;C65;FR4 16/10;Cu35µm;CMS2;Ni-Au; 149 x 74 mm (PMC)
h. Quad Mux card EP 680 1150 510, ML4;Cl4;FR4 16/10;Cu35µm;SLDM2;SLK2;Ni-Au; 149.7 x 74mm Panelised

The breakdown of the component costs for a single RU is shown in Table 13. As shown, the DPMs of the subevent

buffer and input FiFos contribute almost 50% of the component cost.

Table 13 Component prices RU in CHF , Y2000 prices, small Qantities

4 FPSCs OR3LP26B	4 DPMs	2 CPLDs	4 FiFos	SMD Caps/Reststors etc	small IC's	Connectors	total
960	620	25	750	150	190	125	**2820**

Table 14 Component prices MCU in CHF , Y2000 prices, small Qantities

1 PC-on-chip (MachZ e)	4 SDRAM	2 Flash	Ethernet chips +	SMD Caps + Reststors	small IC's + Oscillators	10 Sockets + Connectors	total
146	180	33	45	18	56	55	**533**

Table Table 14 shows the component cost details of the MCU mezzanine card. Larger quantities would significantly bring down the cost for the SDRAM and MachZ chip, which account for 60 % of the component cost.

2.2 Halogene free Readout Unit

The European Norms prescribe use of halogen-free PCB material[1] instead of FR4 epoxy.. As the first PCB at CERN, the readout Unit card CERN EP 680 1150 400 A is manufactured in halogen free material. The low quantity PCB price is a factor of 3 more expensive (1585 CHF), adding ca 1 kCHF to the price of a RU manufactured in FR4.

2.3 Large quantity system price estimation

For large quantities (> 100 pieces) there is a significant cost reduction. An expected 50% price reduction for the PCB and mounting will lower the RU cost by 1100 FS. Frontpanel and fixation costs will be reduced by 150 FS, and a conservative 30% high volume reduction on components amounts to 850 FS less. The RU price would hence fall from 4770 CHF (low Qty) to 2670 CHF (100+). By applying the same 56 % price for large (100+) quantity also to the mezzanine cards, the complete list of large quantity prices is abtained.

Large Qty Cost estimate (63 % of low quantity)

READOUT UNIT ....2.7 kCHF

MCU PMC..... .....0.59 kCHF

Slink I/O PMC..... .....0.23 kCHF

Slink test card PMC..... .....0.24 kCHF

Quand Slink MUX PMC ....0.34 kCHF

1. and plombless solder as from 2008

The total cost of a RU with PMCs in 100 + quantity is therefore (exclusive link cables)

4 link Multiplexer RU (incl 5 * Slink I/O).....3.85kCHF + MCU = 4.44 kCHF

16 link Multiplexer RU (incl. 4 * 4 Slink + Slink I/o)4.29 kCHF+MCU=4.88 kCHF

DAQ / VELO RU (incl. 4 * Slink I/0) .....3.62 kCHF+MCU = 4.21 kCHF + NIC card

3 Modifications

3.1 Modified use of input NIM signal (Xon/Xoff etc)

A simple modification on the RU is required for using the NIM input as a signal for the FPGA logic. One example is the use of an Xon/Xoff input signal. In this case one of the function indicator LEDs as shown in Figure 49 must be unsoldered to give access to an unused I/O pin of the FPGA. Then pin 8 of IC29 (see Figure 42) must be disconnected from the PCB and reconnected (using a wire) to the free ORCA (previous) LED pin. As IC29 is an open collector device, a pullup resistor is needed. Note however: in case the NIM signal is of sufficient duration (order of us), the LED may be retained serving both as a pullup and as NIM signal indicator.

References

[4] **Topics on Readout Unit**, Presentation in LHCb Frontend Electronics & DAQ workshop, May 2000 http://hmuller.home.cern.ch/hmuller/RU/daqws2000.pdf

[5] **Decisions summary for Readout Unit**, of LHCb workshop October 1999, see http://hmuller.home.cern.ch/hmuller/~HMULLER/DOCS/DAQWSsummary.PDF

[6] **Readout Unit II**, Presentation in LHCb DAQ/FE workshop, January 2001, URL http://hmuller.home.cern.ch/hmuller/RU/RU+Jan2001.PDF

[7] **Readout Unit for the LHCb experiment**, J. Toledo, H. Müller, F. Bal, B. Jost. Fifth Workshop on Electronics for LHC experiments. Snowmass, Colorado. Sept. 1999

[8] **SCI implementation study for LHCb Data Acquisition**, H.Muller LHCB Note DAQ 1998-030 URL http://hmuller.home.cern.ch/hmuller/~hmuller/docs/SCIDAQ.PDF

[9] **The LHCb Trigger and Data Acquisition System**, J.-P.Dufey, M.Frank, F.Harris, J.Harvey, B.Jost, P.Mato, H.Muller, IEEE Real Time Conference June 14-18, 1999 Santa Fe, IEEE Trans. Nucl. Sci. vol 47, no 2, pp. 86-90, Apr. 2000

[10] CERN **Slink** homepage http://hsi.web.cern.ch/HSI/s-link/

[11] For a collection of **bus and link standards** documents see URL http://ohm.bu.edu/~hazen/my_d0/std/

[12] Agere (Lucent Technologies), ORCA Field-Programmable System Chip (FPSC) URL http://www.agere.com/netcom/platform/fpsc.html

[13] **LHC-b Technical Proposal** CERN/LHCC 98-4 Chapter 13: Data Acquisition. CERN, 1998

[14] **A networked mezzanine card processor**, A.Guirao, J. Toledo, D. Domínguez, B. Bruder, H. Müller. "Proceedings 12th IEEE NPSS Real Time Conference, Valencia, pp. 81-84, June 2001.

[15] **Processor PMC card proposed standard,** VITA 32 10/30/99 Draft URL http://ohm.bu.edu/~hazen/my_d0/std/PPMC_std31.pdf

[16] **Draft Standard of physical and environmental layers for PCI mezzanine cards IEEE P1386.1** Draft 2.0 http://www.cern.ch/~hmuller/docs/PCI-SCI/pmc_draft.pdf

[17] **Evaluation of SCI as a fabric for a computer based pattern recognition trigger running at 1.12 MHz**". A.Walsch, V.Lindenstruth, M.Schulz, Proceedings 12th IEEE NPSS Real Time Conference, 2001, Valencia, p.11 ff

[18] **A Hardware/Software Triggered DMA Engine**, A.Walsch, LHCb technical note TRIG 2001-125.

[19] **PCI Specification**, Rev 2.2 , PCI Special Interest Group http://www.pcisig.com & PCI interest Group or: PCI System Architecture, Mindshare, ISBN 0-201-30974-2

[20] **LVDS** (IEEE 1596.3) specification http://www.cern.ch/~hmuller/~HMULLER/DOCS/IEEE1596-3.pdf

[21] **Performance issues in PCI reading**', Andre David, IST, Lisbon. (NA60 technical note, see URL) http://adavid.home.cern.ch/adavid/

[22] **Readout Unit papers and meetings archieve** URL http://hmuller.home.cern.ch/hmuller/RUminutes.htm

[23] **Event Building in an intelligent Network Interface Card for the LHCB DAQ**, J-P.Dufey et al. LHCb 2001-094 June 25

[24] **TTCsr receiver mezzanine card**, thesis of Jorge Ferrer in ED group URL http://hmuller.home.cern.ch/hmuller/ttcsr.htm

[25] Readout Unit Meeting at KIP Heidelberg, 9-10 Oct. 2000 Minutes http://hmuller.home.cern.ch/hmuller/RU/Minutes/Heidelberg/Heidelberg.html

[26] **MCU and Linux Programming for the Readout Unit**, B.Bruder Nov, 2000 http://hmuller.home.cern.ch/hmuller/RU/MCU/jtag.pdf see also URL http://b.home.cern.ch/b/bruderbe/www/

[27] **A Hardware/Software Triggered DMA Engine** LHCb 2001-125, internal, trigger, Walsch, A. KIP Heidelberg

[28] **Architecture of a real-time trigger compute farm running at 1.17 MHz** A. Walsch on behalf of the LHCb Collaboration Supercomputing 2001, Denver, USA, November 13-16, 2001 (to appear)

[29] **LHCb Level-1 Vertex Topology Trigger**, LHCb Tech. Note 99-31, Aug. 1999

[30] **A networked mezzanine card Linux processor**, A.Guirao et al, Proc. 12th Real Time Congress on Nucl. and Plasma Sciences, Valencia June 2001, p 81 ff.

[31] **A Readout Unit for high rate applications**, J.Toledo et al. Proc. 12th Real Time Congress on Nucl. and Plasma Sciences, Valencia June 2001, p 230 ff. to appear in Transactions on Nucl. Science

[32] **Vertex Detector Electronics: L1 Electronics Issues,** Draft, Y.Ermoline LHCb- 2001-124 VELO, IPHE 2001-013 last mod. 4 October 2001

[33] **A Large Scale SCI 2D-torus Prototype,** A.Walsch et al., in preparation

[34] **Level-1 Decision Unit** Functional Specifications, B.Jost, LHCb Technical Note Trigger CERN LHCb 2001-034

[35] Web Site for **Level 1 Decision Unit** http://www-iphe.unil.ch/~lhcb/l1du

[36] **Readout Unit / Level-1 Trigger meeting,** KIP Heidelberg October 2000, Minutes http://hmuller.home.cern.ch/hmuller/RU/Minutes/Heidelberg/Heidelberg.html

[37] **Subevent Transport Format**, WEB page http://hmuller.home.cern.ch/hmuller/STF.htm

[38] **A flexible PCI card concept for Data Acquisition**, H.Muller et al. Proc. 12th Real Time Congress on Nucl. and Plasma Sciences, Valencia June 2001, p 249ff. see also Infos on URL http://hmuller.home.cern.ch/hmuller/pci-flic.htm