



ATLAS TDAQ/DCS ROS Prototype-RobIn HLDD

Document Version: 2.4
Document Date: September 24th, 2002
Document Status: 1st Release

1. Abstract

This document describes the high-level hardware design of the final ATLAS prototype-RobIn, as evolved from a number of previous prototypes. A dual TDAQ-interface is supported to allow investigations of both bus- and switch-based ROS implementations up to the TDR.

Keywords: Atlas, DAQ, Read-Out, ROS, RobIn

2. Institutes and Authors

Royal Holloway University of London: B. Green
NIKHEF Amsterdam: G. Kieft
University of Mannheim: A. Kugel

3. Revision Log

1. Document title: High Level Design of ATLAS Prototype-RobIn			
2. Document Reference Number			
4. Issue	5. Revision	6. Date	7. Reason for Change
1	0	25 th Feb. 2002	Birth
1	1	11 th Mar. 2002	Topics from RobIn Meeting of March 7 th added
1	2	18 th Mar. 2002	Format changed to match (more or less) SDLT template
1	3	19 th Mar. 2002	Some modifications, still very preliminary
1	4	20 th Mar. 2002	Diagrams modified, intro added
1	5	26 th Mar. 2002	Text added
1	6	27 th Mar. 2002	Uses cases and text added
1	7	27 th Mar. 2002	
1	8	12 th Apr. 2002	UR's added
1	9	15 th Apr. 2002	Modifications after Phone Conf
1	10	15 th Apr. 2002	More text added
1	11	16 th Apr. 2002	Implementation options + Requirements added
1	12	16 th Apr. 2002	UR's modified
1	13	17 th Apr. 2002	Minor modifications
1	14	21 th Apr. 2002	API section + COTS removed, component text added
1	15	22 nd Apr. 2002	Streamlined for High-Level design
1	16	22 nd Apr. 2002	More HW details added
1	17	22 nd Apr. 2002	Functional diagram added
1	18	22 nd Apr. 2002	Typos etc. corrected
1	19	24 th Apr. 2002	Some functions added
1	100	26 th Apr. 2002	Start of "official" draft. Address "review" comments Apr. 23 rd
1	101	2 nd May 2002	Comments from JP and RC included
2	0	6 th Sept. 2002	Comments from "Review Preparation Team" adopted
2	1	17 th Sept. 2002	Minor corrections
2	2	18 th Sept. 2002	"Special Issues" appendix added
2	3	23 th Sept. 2002	Some corrections of "guidelines" (DJF, MLV,...)
2	4	24 th Sept. 2002	Management data structure added

Table 1: Document Change Record

Contents

1.	Abstract	1
2.	Institutes and Authors.....	1
3.	Revision Log	2
4.	Introduction	4
4.1.	Purpose of this document	4
4.2.	Glossary, acronyms and abbreviations	4
4.3.	References	4
5.	Requirements and Constraints.....	5
6.	Guidelines from previous work.....	5
6.1.	Hardware	5
6.1.1.	MFCC.....	5
6.1.2.	FPGA/MicroEnable.....	6
6.1.3.	1960	7
6.1.4.	SHARC.....	8
6.1.5.	Summary	8
6.2.	Software	9
6.2.1.	Local Software	9
6.2.2.	Host Software	9
6.3.	Performance	10
7.	Design	11
7.1.	Block Diagram	11
7.2.	Functional Building Blocks.....	12
7.3.	Complex Functions.....	15
7.3.1.	Buffer Management.....	15
7.3.2.	Download/Configuration.....	15
7.4.	Hardware Building Blocks	16
7.4.1.	PCI Bridge.....	16
7.4.2.	Ethernet MAC/PHY	16
7.4.3.	ROL-Interface	16
7.4.4.	Core	16
7.4.5.	Buffer Memory.....	17
7.4.6.	Management Memory	17
7.4.7.	Processor Memory.....	17
7.5.	Firmware	17
8.	Cost Model	17
8.1.	Multiple ROLs.....	18
9.	Appendix A	19
9.1.	Message loss.....	19
9.2.	Response shaping	19
9.3.	Network addressing.....	19
10.	Appendix B	20
10.1.	Management memory data structures.....	20
10.1.1.	FIFO extension lists.....	20
10.1.2.	Fragment information entry.....	20
10.1.3.	Hashing.....	20
10.1.4.	Hash-entry	20
11.	Appendix C	21
11.1.	List of Tables.....	21
11.2.	List of Figures	21

4. Introduction

4.1. Purpose of this document

For the TDR we must be able to present a clear view of a possible implementation (plus options) of the RobIn component, satisfying the performance requirements. As the final implementation of the ROS is not defined yet as well, the RobIn required now has to support a variety of options, in particular bus- and switch-based ones. Therefore the prototype-RobIn – with extended functionality – will be used as an intermediate step, prior to the final pre-production RobIn.

This document provides the high-level design of the prototype-RobIn. The design is developed as a joint effort of the three institutes RHUL, NIKHEF and UniMA. The final aim of the design-team is to present a single detailed design of a prototype RobIn that will allow to study bus and switched based ROS implementations.

4.2. Glossary, acronyms and abbreviations

See [5]

4.3. References

- [1] ROBIN Summary Document: <http://atlasinfo.cern.ch/Atlas/GROUPS/DAQTRIG/ROS/documents/ROBINsummary.pdf>
- [2] ROS-URD: unreleased, see ROS web site at <http://atlas.web.cern.ch/Atlas/GROUPS/DAQTRIG/ROS/ros.htm>
- [3] ATLAS Readout Link recommendation, http://edms.cern.ch/file/332389/1/rod_rol.pdf
- [4] HOLA S-Link documentation: <http://hsi.web.cern.ch/HSI/s-link/devices/hola>
- [5] prototype-RobIn URD: <http://akugel.home.cern.ch/akugel/robIn/docs/urd.pdf>
- [6] prototype-RobIn SWID: <http://akugel.home.cern.ch/akugel/robIn/docs/swid.pdf>
- [7] RobIn Measurements Document (preliminary): <http://atlasinfo.cern.ch/Atlas/GROUPS/DAQTRIG/ROS/documents/ROSSystemTestReport.pdf>
- [8] RobIn Measurements Presentation July 2002: http://doc.cern.ch/archive/electronic/other/agenda/a02164/a02164s5t2/transparencies/Matthias_Vers5.pdf
- [9] MFCC RobIn: DAQ-2000-53: Read-Out Buffer in DAQ/EF prototype -1
- [10] MicroEnable RobIn: ROB meeting, Amsterdam 1999: <http://www-li5.ti.uni-mannheim.de/fpga/atlas/rob-in-new.pdf>
- [11] I960/UK-RobIn (Measurements): DAQ-2000-053: Read-Out Buffer in DAQ/EF prototype -1
- [12] I960/UK-RobIn (Details): DAQ-2000-013: The UK ROB-in a prototype ATLAS readout buffer input module
- [13] I960/UK-RobIn documentation (processor): <http://www.hep.ucl.ac.uk/atlas/rob-in/processor.html>
- [14] Sharc-RobIn: DAQ-2000-021: A SHARC based ROB Complex : design and measurement results
- [15] The Active Rob Complex: An SMP-PC and FPGA based solution for the Atlas Readout System. R. Bock, J. A. Bogaerts, P. Werner , A. Kugel, R. Manner, M. Muller, <http://ific.uv.es/rt2001/proceedings/proceedings.pdf> , page 199ff
- [16] DAQ-2000-10: The Use of Low-cost SMPs in the Atlas Level-2 Trigger
- [17] DAQ-2000-051: DAQ-Unit intra and inter-IOM communications summary document
- [18] ROS Workshop Feb 2002 Presentation Bus-Based RobIn: <http://doc.cern.ch/archive/electronic/other/agenda/a0281/a0281s1t21/transparencies/BusBased.pdf>
- [19] The Message Format used by DataCollection in the ATLAS TDAQ Integrated prototype: DC note, CERN 2002, <http://atlas.web.cern.ch/Atlas/GROUPS/DAQTRIG/DataFlow/DataCollection/docs/DC-022.pdf>

5. Requirements and Constraints

User Requirements and Constraints are detailed in [5].

6. Guidelines from previous work

In the previous phases a number of RobIn prototypes have been developed and used by the ROS community: MFCC-RobIn [9], I960-RobIn [12][11], FPGA-RobIn [10] and SHARC-RobIn [14]. They all provide basically the same functionality. Many measurements have been carried out in the past, however a standardized setup has only been used recently [8] which still needs to be integrated with [7]. In general the performance of the prototypes depends on the properties of the involved hardware, namely the RobIn itself, the host bus and the bus interface, but also on the software, again locally (if any) and on the host.

This section summarises the experience gained with the previous RobIn prototypes and sets guidelines for the new prototype-RobIn.

6.1. Hardware

All 4 prototypes exhibit a similar architecture: a single S-Link is attached to a programmable logic device (CPLD or FPGA), data from the S-Link is temporarily stored in a local buffer, requested data are transferred via a PCI bridge device to the host. All but the FPGA-RobIn use a local processor to manage the buffer. The general approach to use reconfigurable logic (FPGA) for the high-speed/high-rate part of the design and a (local or remote) processor for the less time critical parts seems to be all right.

In the following a block diagram of each of the prototypes is presented together with the major conclusions from [1].

6.1.1. MFCC

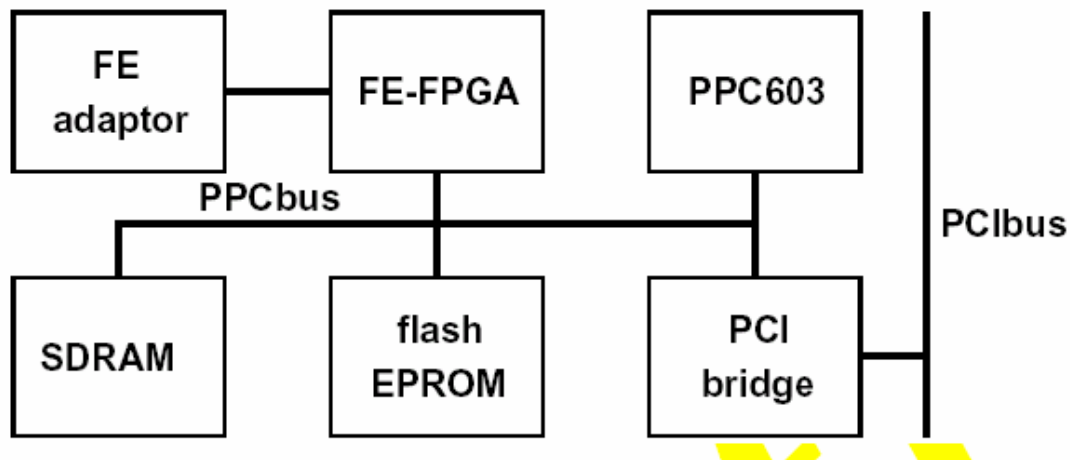


Figure 1: MFCC block diagram

Issues	
	FPGA at limits
	Multiple ROLs not possible (mechanics, performance)
	CPU performance too low
	Message passing over PCI slow
Conclusions	
	OS convenient (application start, memory allocation/management, debugging, remote access)
	CPU + FPGA good for flexibility, complementary functionality, DAQ-1 ROB monitoring code could be reused on RobIn CPU (not done yet)
Recommendations	
	Better FPGA (Xilinx) and PCI (64 bit) on next generation

6.1.2. FPGA/MicroEnable

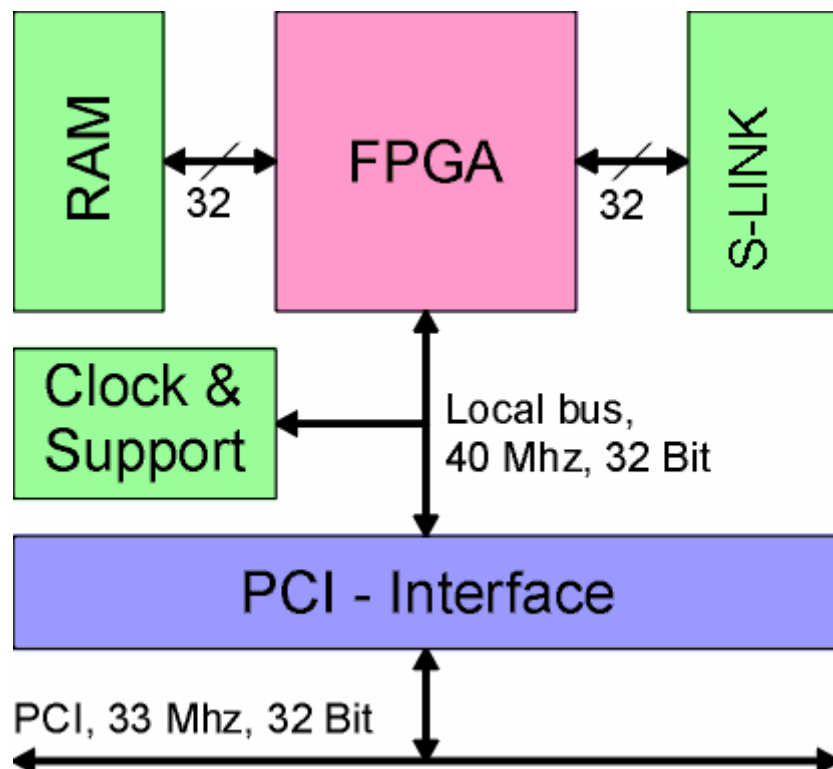


Figure 2: MicroEnable block diagram

Issues	
	Slow memory interface
	API implementation: initially large DMA overhead (improved version used in [8])
	Input bandwidth not fully reached due to memory interface (dual-port emulation with async. SRAM)
	Small buffer, simple management => buffering of few events only
Conclusions	
	Most requirements achieved with simple architecture
Recommendations	
	Larger + faster buffer and faster FPGA on next generation

6.1.3. I960

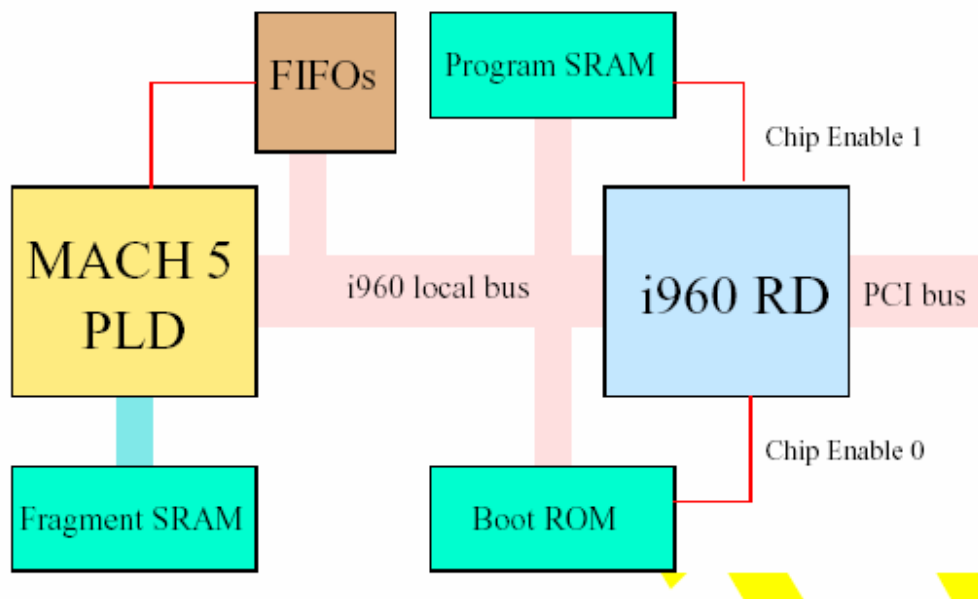


Figure 3: I960 block diagram

<u>Issues</u>	
	CPLD too small, code changes problematic
	Processor at limits
	SRAM cost + size will limit buffer size
<u>Conclusions</u>	
	CPU + FPGA convenient + flexible
	Initial requirements (100MB/s input) achieved
<u>Recommendations</u>	
	Faster processor
	FPGA replacing CPLD
	Faster buffer
	Integrated S-Slink
	GE interface for output

6.1.4. SHARC

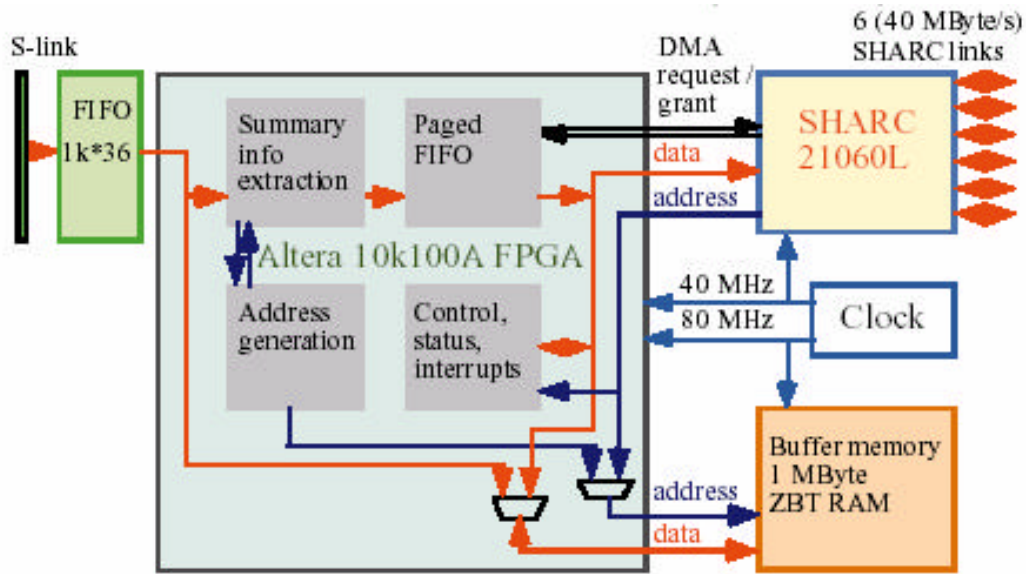


Figure 4: Sharc block diagram

Issues	
	FPGA design (no bursts) limits buffer read-out speed to 40MB/s
	PCI speed limited to 80MB/s
	Reduced performance without asynchronous I/O
Conclusions	
	Requirements achieved
	Testing facilities useful
Recommendations	
	None indicated

6.1.5. Summary

The most important recommendations are:

- Use recent FPGA technology with sufficient capacity and speed
- Use recent processor with sufficient speed
- Larger and faster buffer
- Faster PCI (64 bit)
- Don't put the buffer on the processor bus
- If using dual-port emulation, watch the memory bandwidth
- Provide local DMA capability (e.g. via host interface), avoid data moved by either processor
- Use distributed (HW/SW, word-level, page-level) paged buffer manager á la I960

6.2. Software

6.2.1. Local Software

Despite their similarity in hardware design the previous prototypes have used rather different approaches with respect to local software. The range spans from using a local OS (MFCC) to using a stand-alone application (I960, SHARC) to using no local software at all (MicroEnable). While the first two don't make a big difference for the hardware design the last one can well do and in fact there are still potential implementation options which might try to avoid the processor. As a consequence the hardware design of the prototype-RobIn shall provide full support for a processor with OS as well as the possibility to run a RobIn-application without processor.

6.2.2. Host Software

Although the host software is clearly not an issue of RobIn design the RobIn hardware has to take into account some particular facts.

- The “official” ROS software used up to summer 2002 implemented a synchronous interface to the RobIn modules which lead to a significant loss of performance. This can be seen easily when comparing the results from the improved software version (with asynchronous requests, Figure 5, from [8]). There are also clear indications for the benefits of asynchronous fragment processing in [15] and [16]. The prototype-RobIn shall not force the software to use synchronous fragment processing.

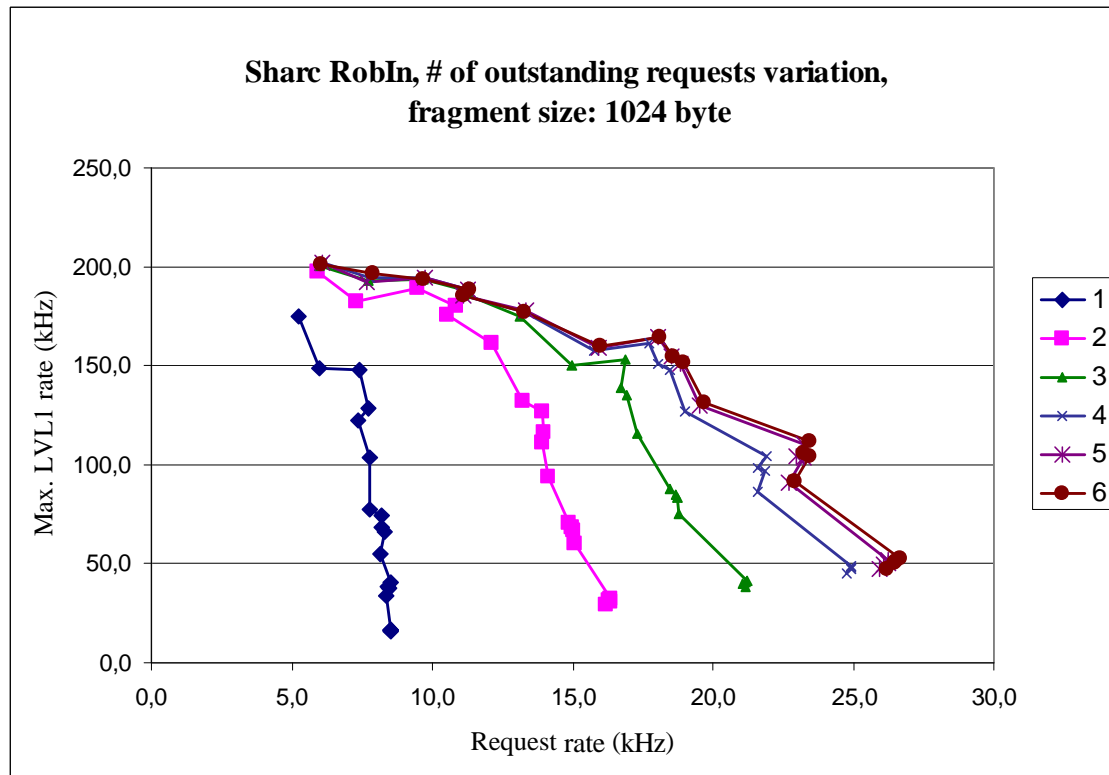


Figure 5: Effect of asynchronous fragment processing

- Another important factor is the communication mechanism between the RobIn and the host. In [17] the message passing used in the MFCC-RobIn is described and the clear conclusion is that this kind of message passing should be avoided. A simpler and more efficient approach is described in [15] and is already recommended in [18]. This technique uses the integrated DMA controller of the PCI-bridge together with a specialized host library to enhance the communication throughput. The following paragraph explains the idea as presented in the reference:

The principle is to set up an DMA with a large memory buffer of 500 kB or more before the first event fragment is requested. Using the DMA-on-demand feature the transmission is postponed by the PLX9080 and the FPGA design as long as no data transfer is in progress. To request an event fragment

from the microEnable RobIn, the ARobC library writes the event information into an FPGA register using a single cycle I/O. The FPGA localizes the data inside the microEnable memory and reactivates the delayed DMA-on-demand using the dedicated flow control signal of the PLX9080. To perceive the arrival of the requested data the host polls the DMA buffer position which has been set to zero before. After the transmission has started this position contains the length of the requested data fragment. According to this length the end position of the dataset and the start of the next is computed. To indicate that the data has completely arrived, a magic word is transmitted as last word and can be polled from the DMA buffer.

Using this scheme only the data request is a host-initiated single-cycle while the acknowledgments are performed by the PCI hardware and polling is done in the system memory. This leads to a big improvement in PCI utilization and performance. prototype-RobIn shall support the required PCI bus master capability.

- In case the host-software wishes to rely on asynchronously reported “conditions”, the RobIn shall provide a mechanism to asynchronously generate signals to the host software. In a bus-based environment the RobIn shall provide a hardware interrupt for this purpose.
- Finally one should take care of not copying data too frequently [8]. One transaction from the RobIn into host memory and one from host memory to the destination NIC – as used during the recent tests – is acceptable. Additional copies, e.g. to collect event fragments or from kernel to user memory should be avoided. No extra copy operations shall be forced by the prototype-RobIn hardware.

6.3. Performance

Figure 6 (derived from [8]) shows the performance which has been achieved with some of the previous RobIn¹ prototypes in a stand-alone ROS setup.

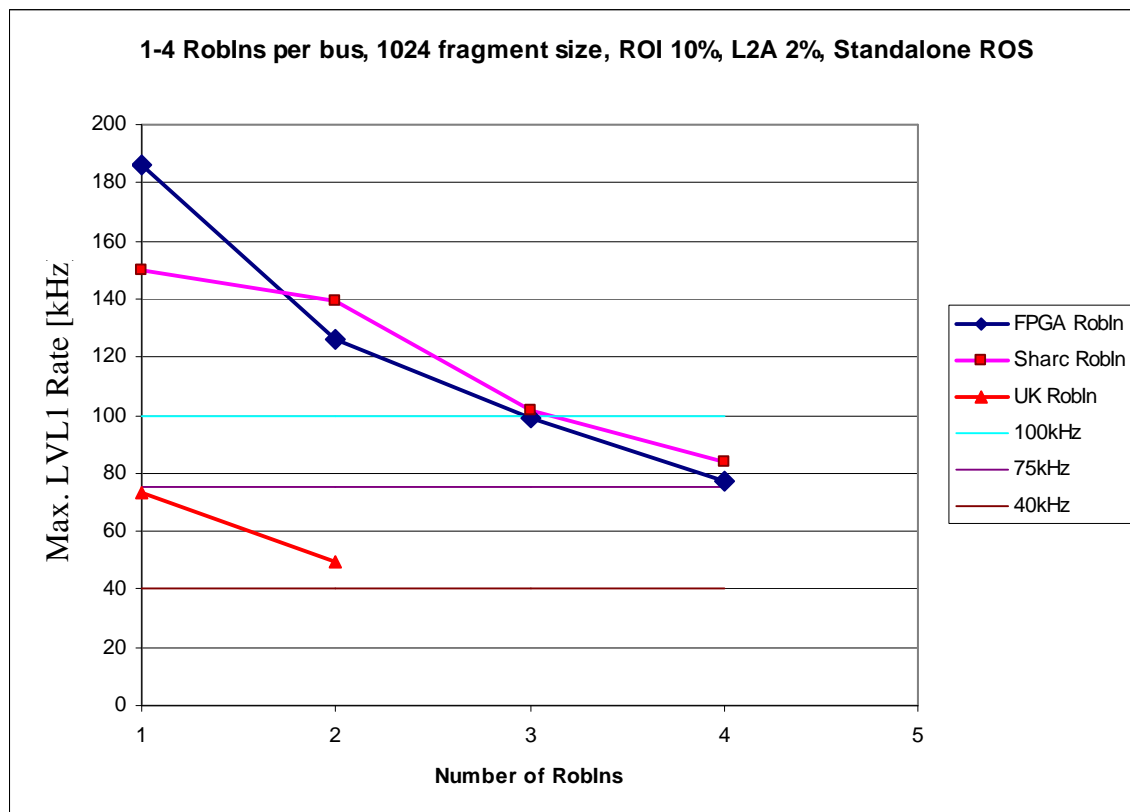


Figure 6: Basic RobIn performance

¹ The UK-RobIn (I960) suffers most from the missing asynchronous processing in the ROS-software. The hardware has demonstrated better performance in a different setup. Additionally the two others don't have real input in this setup.

At least for a limited number of boards the performance goal is probably in reach for a PCI based ROS, provided the indicated optimisations are implemented. Therefore, the design of the prototype-RobIn will in general build upon the common features of the previous prototypes while avoiding the observed bottlenecks.

7. Design

The RobIn requires a very high real-time performance, in particular short response times at high rates, and is therefore a demanding task. On the input side (from the detector) hardware assistance by a fifo and programmable hardware is essential. Hardware assistance is also required for the management of the fragment buffer, if a CPU is used.

Although the prototype will support two flavours of TDAQ-interfaces a major aim of the project is to present a design which can be turned into the final RobIn version with little modifications, in particular only by removing (and not by adding) functionality. This approach will be taken down to an as detailed level as possible.

The prototype-RobIn can be used in either a completely bus-based or a completely switch-based environment simply by changing the local firmware. Additionally it will also support a “hybrid” style, by activating both TDAQ-interface at the same time, e.g. one for control messages and the other one for data messages (e.g. NET-based RobIn housed in a PCI system).

The designs starts at the high-level with a simple block-diagram and a subsequent functional decomposition into fairly independent “boxes”. This approach will help to define later-on clean interfaces for a modular detailed design both at the hardware and software level. The final design of the RobIn software and the implementation itself will be a separate task.

7.1. Block Diagram

The design principle presented here builds upon the experience with the previous RobIn prototypes (see section 6.1). The primary functions *receive – buffer – deliver – release* [5] are mapped onto a small number of specialised building blocks: ROL-IF – CORE – MEM – TDAQ-IF (Figure 7). This corresponds to the structure described in [1]. In fact TDAQ-IF represents TWO interfaces, for bus(PCI)- and network(MAC)- oriented communication. The core has two major aspects, one dealing with high-bandwidth/high-rate requirements of the internal data-flow (DF-CORE), the other dealing with lower priority requirements (AUX-CORE).

Each of the building blocks comprises a certain functionality (Table 2) which together build up the RobIn functionality. Functionality is ultimately implemented by physical devices (like CPU, MAC, etc.).

Based upon experience it is straightforward to use an FPGA to implement the DF-CORE functionality. Also the use of a processor for the AUX-CORE is appropriate (see section 6). However it is not excluded that the FPGA may alternatively implement the AUX-CORE functionality (for special scenarios).

For the purpose of testing various ROS implementations the interfaces to NET and BUS will both comprise the full set of required functionality in a way that either of the interfaces can be completely disabled.

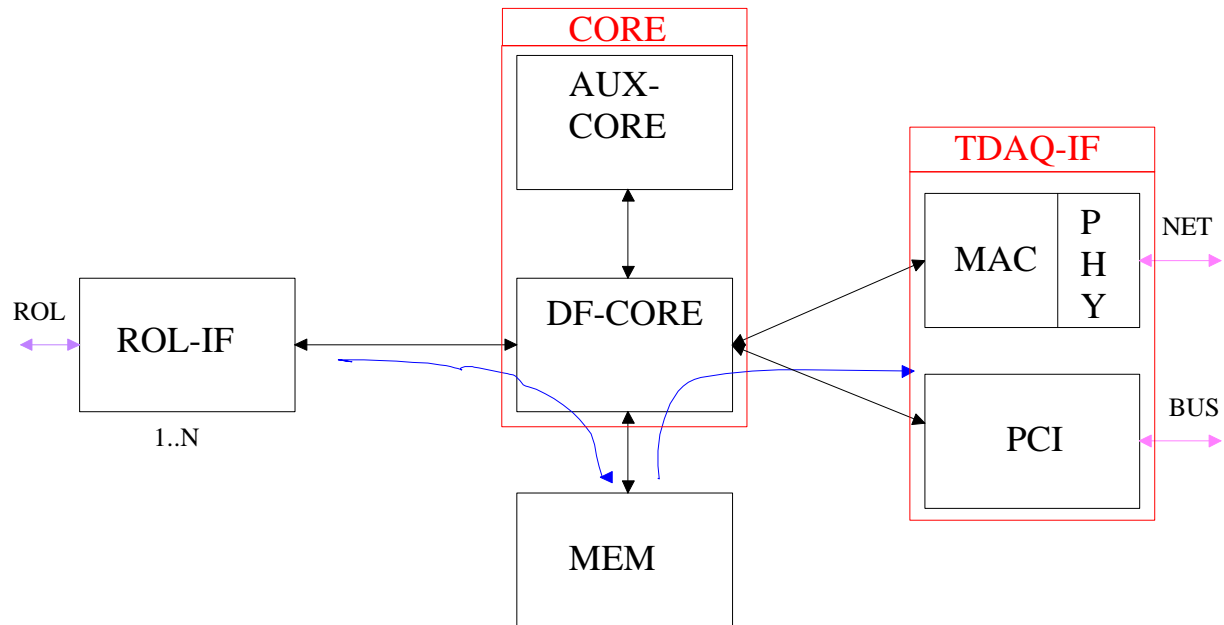


Figure 7: Basic Block Diagram

7.2. Functional Building Blocks

The basic interactions of the main functional elements are displayed² in Figure 8. The thick red lines indicate the main data-path which runs on the path: ROL-IF -> BufMgr-IN -> BUFFER -> DMA -> TDAQ-IF.

The input from the ROL is realised in the usual way via an ATLAS S-Link [2]. The functions related to input and buffering are already well known from previous RobIn work. Messaging functions are also known, e.g. from a RobIn in the context of the DAQ-1 EventManager and DC message passing.

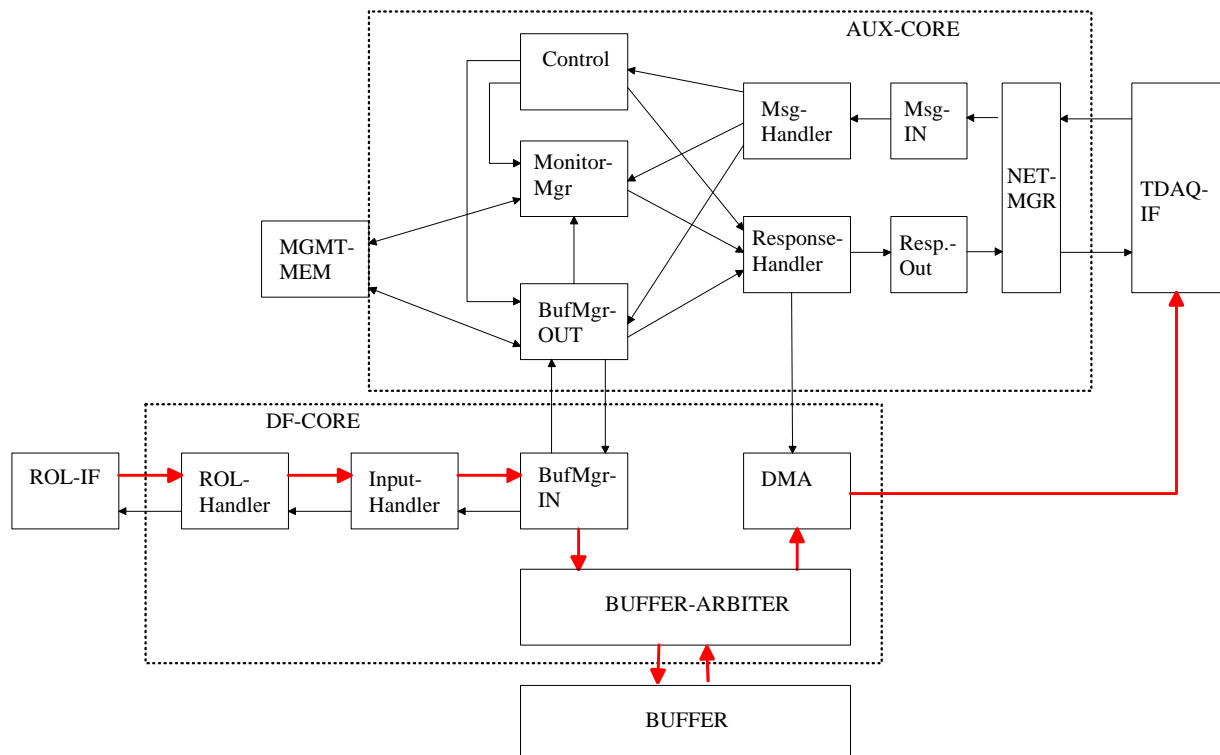


Figure 8: Basic Functional Diagram

² Boundaries for DF-CORE and AUX-CORE are only for illustration of a typical mapping.

A summary of these – plus some more – functional blocks is presented in Table 2.

Certainly the functionality needs to be further detailed, however it is assumed that the presented list is sufficient to complete the design of the new RobIn hardware, in particular taking into account the flexibility of the components which will be used.

Item #	Function	Prio.	Location(s)	Description
1	ROL handler	1	DF-CORE + ROL-IF	Attach to ROL-IF and drive protocol. Provide framed input-data stream
2	Input handler	1	DF-CORE	Take fragments from ROL handler, check header, length, etc
3	Buffer	1	MEM	Provide storage for fragment data
4	BufMgrIN	1	DF-CORE	Put data put into buffer, update lists (fifos) for paged buffer
5	BufMgrOut	1	CORE	Maintain fragment index list, etc. Provide fragment address upon request
6	MsgIN	1	AUX-CORE	Receive messages from DAQ interface (depends on BUS vs. NIC)
7	Msg handler	1	AUX-CORE	Decode and dispatch DAQ messages, e.g. fragment requests
8	Response handler	1	AUX-CORE	Format/assemble responses incl data for DAQ interface
9	Response OUT	1	AUX-CORE	Send response to DAQ interface (depends on BUS vs. NIC)
10	Control	1	AUX-CORE	Control RobIn state of operation
11	DMA	1	CORE, TDAQ-IF	Transfer data between buffer and NIC, or buffer and BUS
12	NET interface (is part of TDAQ-IF)	1	MAC (+PHY)	TDAQ interface via network
13	BUS Interface (is part of TDAQ-IF)	1	PCI	TDAQ interface via PCI bus
14	MgmtMem	1	NN (addition to AUX-CORE)	Store monitoring + status information
15	BufferArbiter	1	DF-CORE	Provide access to fragment buffer for all clients (BUFMGR, DMA, MEMMAP,...)
16	NetMgr	1	CORE	Handle higher-level of network protocol, e.g. sequence checking, ICMP, etc.
17	Configuration interface	1	AUX-CORE	Provide for software upgrade, power-on configuration, etc.
18	Local MemMap	2	DF-CORE	Let on-board CPU access buffer and peripherals
19	Bus MemMap	2	DF-CORE + PCI	Provide local address space for buffer enabling PCI-bridge to map buffer into BUS address space. Eventually also map management area(s)
20	Monitor Mgr	2	CORE	Support event and operational monitoring, maintain status information
21	SelfTest	2	CORE	Perform basic testing
22	Error handler	2	CORE	Perform error specific actions (e.g. send NACK packet on missing seq#. Drop invalid packets/fragments)
23	Emulation data generator	3	NN (CORE)	Means to fill buffer and manager with emulated fragments
24	Fragment processing	3	CORE	Placeholder for potential operations like reformatting, pre-processing, etc.

Table 2 : Functions



7.3. *Complex Functions*

7.3.1. Buffer Management

Buffer management is THE central function of the RobIn and a large amount of experience has been gained in this area in the previous prototype work. The design presented is based on the paged buffer management scheme developed by the UK group [12], with some optimisations applied. It involves mainly a buffer manager, a paged³ buffer memory and a set of fifos. The buffer management is performed at two levels, one operating at word-level (BufMgr IN) and the other operating at page-level (BufMgr OUT). BufMgrIN will always run in the FPGA. BufMgrOUT will run on the CPU but can be put on the FPGA – in a simplified version – if no CPU is available.

To provide fast access for the BufMgrIN the fifo's will be implemented in hardware, internally in the FPGA. As this approach would limit the maximum number of pages available to approximately 1k (= 1MB buffer at 1kB fragments) the hardware fifo's are extended by the BufMgrOUT via additional lists of used and free pages.

Upon initialisation all available pages are entered into the free-page fifo (FPF). For each incoming fragment the BufMgrIN retrieves a page from the FPF and moves it to the used-page fifo (UPF). Fragments longer than a page will subsequently use additional items from the FPF. Event ID and header information are copied on-the-fly to the BufMgrOUT via the UPF, as well as status and error information (e.g. last page, truncation, link-error, etc.). Once the FPF reaches a low watermark the BufMgrIN directs the input handler to stop the ROL-IF. Transmission resumes after more free pages have become available.

The BufMgrOUT reads entries from the UPF and adds it to its list of available events. List management will use a hashing algorithm derived from the I960 prototype, adapted⁴ to the new memory layout. As all relevant data are available in the UPF the BufMgrOUT never needs to access the buffer memory directly.

Upon a data request from the message handler the BufMgrOUT retrieves the pages of the particular fragment from the event list and provides the address parameters to the DMA engine, which performs the transfer to the TDAQ-IF. Similarly for release requests the relevant pages are removed from the fragment list entered into the FPF, which effectively deletes the fragment from the memory.

Both BufMgrIN and BufMgrOUT will perform error detection and handling, where appropriate. BufMgrIN will detect and flag low level error like transmission errors and incorrect L1ID sequence. BufMgrOUT will detect errors like duplicate L1ID and will handle the errors indicated by BufMgrIN.

The BufMgrOUT provides various status and error information (e.g. most recent L1ID, fragment error flags) via a set of registers to the MonitorMgr. The MonitorMgr will in turn update error counters, pending request queue and local status information.

More information on management data structures is given in section 10.1.

7.3.2. Download/Configuration

The RobIn needs to be in-situ upgradeable and needs permanent store for boot-code [5]. In all cases where a dedicated bus interface is available the implementation is simple (e.g. flash-EEPROM on local bus). For other cases the AUX-CORE must provide functionality to update the permanent memory:

1. Receive new code via TDAQ-IF and buffer into on-board memory
2. Verify code via TDAQ-IF
3. Reprogram on-board permanent memory
4. Reboot from on-board permanent memory

For factory testing and initialisation a JTAG (or equivalent) interface shall support to download boot code into the on-board permanent memory.

³ A page here is a synonym for a fixed size fraction of the buffer memory. Its size is determined by the logical management scheme, not by a physical constraint of the memory. An individual fragment might use multiple pages.

⁴ The original I960 hashing uses the lower 10 bits of the event ID. This number is likely to be increased with larger buffer and management memory.

7.4. Hardware Building Blocks

The distribution of the functions across hardware components is a bit more complicated than mapping the block-diagram, not because of the complexity of individual tasks but because of the number of implementation options. From a traditional point of view the following major components will be needed for an implementation of the basic block diagram:

- S-Link
- FPGA
- Memory
- Processor
- Network interface
- PCI bridge
- Some “glue”

For this set of components a preliminary ad-hoc selection has been made to verify the pure existence of devices. This selection is used as a starting point in section 8.

To achieve a higher degree of integration (and less components) some of these components could be merged into a single hardware object. Obvious options to achieve a higher degree of integration are:

- Merge processor and network interface into network processor
- Merge processor and BUS interface
- Use FPGA with IP-cores for any combination of Processor, PCI, Network interface

7.4.1. PCI Bridge

In a bus-based ROS environment the PCI-bridge implements the TDAQ-IF. In a switch-based ROS environment the PCI-bridge can be used for configuration, control and operational monitoring, i.e. it provides a control path independent of the data path. In the scenarios studied so far the required bandwidth per RobIn can be handled by a 32bit/33MHz implementation. However for multiple ROLs per RobIn or multiple RobIn's per PCI bus a 64Bit PCI architecture should be considered. PCI-master and DMA capabilities are required, supporting the transfer mechanism⁵ used by the microEnable prototype (see page 9).

7.4.2. Ethernet MAC/PHY

In a SWITCH-based ROS environment the ethernet MAC implements the TDAQ-IF, together with the physical interface device (PHY). In a bus-based ROS it will probably not be used. Although modelling shows that the required bandwidth is just at the limit for 100Mbit/s a GE interface will be used (more headroom, more recent technology, more flexibility). For flexibility reasons the MAC should use a GMII interface to attach to the PHY interface. Functionality corresponding to the listed UR's needs to be available.

7.4.3. ROL-Interface

The ATLAS ROL is realised with an implementation of the ATLAS S-Link, the latest specification supports 160MB/s over a single optical fibre (HOLA, [4]). No return-signals apart from XOFF to stop transmission will be used (also see UR section). The implementation might either be done using an S-Link mezzanine LDC or by using an embedded version, integrated into the RobIn PCB.

7.4.4. Core

FPGA

The FPGA is the primary candidate to implement all high-rate, high bandwidth DF-CORE functionality. In addition to an estimated minimum capacity of 100k Gates support for multiple clock-domains, embedded memory and asynchronous FIFOs is required. The final resource requirements will depend upon implemented functionality, e.g. additional IP-Cores. An SRAM-based technology (=> infinite number of reconfigurations) will be used. FPGA code will be stored in an on-board flash-memory (see configuration section).

Processor

When a dedicated processor is used, which is recommended according to section 6.1, then it is the primary candidate to implement the functions of the AUX-CORE. Depending on the ROS implementation it will provide the higher level messaging functions. In particular for switch-based architectures it will be involved in the book-

⁵ Demand-Mode DMA: the DMA-engine of the PLX chip operates under control of the FPGA.

keeping of the events in the buffer memory and handle the different requests from the network and/or PCI bus and perform monitor and self-test tasks. A processing power of 60 MIPS [13] is required per ROL. The processor will require memory for code/data at an adequate speed, which is separate from the buffer memory.

7.4.5. Buffer Memory

The buffer memory closely interacts with the buffer management functions to continuously store the event fragments arriving from the ROD-IF and to provide data requested from the TDAQ-IF. Performance requirements are given in [2][5], however the required buffer size is not well defined. Previous prototypes have provided in the order of 0.5 to 2 MB. In [2] a value of 2.5MB is indicated. Recent discussions however suggest that at least for this prototype a larger buffer would be of benefit (in particular for a switch-based ROS).

Dual-ported static memory is most suitable for this application, but these memories are expensive and not compact. Normal synchronous static memories (SSRAM) used as pseudo dual-ported memories, are more compact and cheaper. For easy buffer management and access for monitoring purposes, non-bursting memories seem more appropriate although the memories will probably be pipe-lined for performance reasons. Because the memory may alternately be read and written, memories without an additional wait state between consecutive write and read cycles (e.g. ZBT-SRAM) are preferable. SD-RAM would provide the highest capacity at the lowest price, however access and arbitration is more complicated. Device densities vary from 18Mbit for ZBT to 256Mbit for SD-RAM.

7.4.6. Management Memory

Management memory will be used by the AUX-CORE to store any kind of book-keeping, e.g. for lists of buffer pages, monitoring information, message buffers etc. To optimise overall performance the management memory should be completely separated from the buffer memory. Capacity depends mainly on the number of supported fragments and buffer pages and is estimated to be in the order of 1MB (per ROL). This memory could eventually be included in the processor's main data memory.

7.4.7. Processor Memory

The processor will require separate memory for code/data at an adequate speed. Required capacity varies depending on functionality, processor type and software implementation (e.g. use of an OS).

7.5. *Firmware*

The prototype-RobIn will involve two flavours of firmware, one being the software running on the local processor, the other one being the FPGA design. Programming languages will be C/C++ and VHDL for CPU and FPGA respectively, with appropriate development tools. Both languages provide (reasonably) good support for modular designs. Component selection shall consider that adequate development tools are available.

8. Cost Model

For an initial cost estimation a set of components has been used, assuming that the following items have the major contribution: CPU, FPGA, Buffer Memory, S-Link, PCI-Bridge, GE-NIC, PCB. There will be some more components for power, control, configuration support etc. summarised as "Misc". Current prices are for prototype quantities (approx. 10pcs), not for production quantities.

Item	Type	Estimated Unit Price (€)	Comment
CPU	Intel IOP321	75	
FPGA	Xilinx XC2V1000	300	Alt. XC2S300E: 70€
Buffer	ZBT RAM 1M*36	140	Might be less
S-Link	2.5Gbit/s optical	200	Opto TX/RX + SerDes
PCI-Bridge	PLX9656	60	
GE-NIC	LTX1000+IXF1002	120	MAC 75\$, PHY 25\$
PCB	64 Bit PCI, short	150	Volume price much lower
Assembly		200	Volume price much lower
Misc	Cap's, R's, PLD, Prom	150	Guess only
Total		1395	

Table 3 : Cost Estimate

There are additional costs not listed above for the development of the prototype, e.g. for PCB layout (if not done at an institute), initial PCB and production charges, small-quantity surcharges, and for development tools (CPU, FPGA).

For volume production one can expect a cost reduction in the order of 10 to 40% for electronic components, even more for PCB and assembly. According to this estimation the cost of the prototype will be twice the one aimed at, however with some optimisations and considering the simpler TDAQ-IF and volume prices the cost goal is within reach for the final version.

8.1. Multiple ROLs

From the use of multiple ROLs on a single RobIn one can expect some cost savings, as not all parts will get the same multiplicity. There will always be only one PCB and assembly, but at volume production these two don't contribute so much. We assume a maximum of 4 ROLs for a PCI board, because it will become very difficult to get more than 5 links (4 ROLs + 1 GE) through the back panel. An initial calculation (Table 1) shows that the cost per ROL drops most when going from 1 to 2 ROLs. Considering the increasing complexity of the design and the more demanding manufacturing process the optimum number of ROLs per RobIn appears to be 2.

Type	1 ROL	2 ROLs	3 ROLs	4 ROLs
CPU	1	1	2	2
FPGA	1	1,5	2	3
RAM	1	2	3	4
S-LINK	1	2	3	4
PCI	1	1	1	1
NIC	1	1	1	1
PCB	1	1,2	1,5	2
Assembly	1	1,2	1,5	1,8
Misc	1	1,5	2	2,5
	100%	76%	70%	70%

Table 4 : Multiple ROLs⁶

⁶ Numbers indicate cost equivalent, not necessarily component count

9. Appendix A

A number of issues related to the design of the RobIn – but not at the level of hardware – will be discussed elsewhere.

9.1. *Message loss*

Without a reliable network protocol a strategy to prevent overflow of the buffer, or at least to recover from this situation, will be needed. The strategy is described in [6].

9.2. *Response shaping*

In a switch-based scenario care has to be taken that not all RobIn send data to the same destination SFI simultaneously. The strategy is described in [6].

9.3. *Network addressing*

For certain DC messages a “third-party” addressing scheme has to be used. To date two different approaches are proposed, which will be pursued concurrently for a limited time. Refer to the proposal presented in [6].

10. Appendix B

10.1. Management memory data structures

The main purpose of the management memory is to keep the extended lists of free and used pages and the fragment lists. Additionally it will keep a number of error counters, configuration and status information, however the total size of the latter is negligible compared to the former lists.

10.1.1. FIFO extension lists

The number of buffer pages supported by the prototype-RobIn is much larger than that of the previous prototypes. As a consequence, the FPGA-embedded hardware fifo's are not large enough to keep all pages. Therefore two lists of page-id's are maintained in the management memory, which extend the size of the hardware fifo's. The size of these lists depends on the total number of buffer pages.

10.1.2. Fragment information entry

For each buffer page allocated to the incoming data stream a fragment information entry (FIE) is created by the BufMgrIn and entered into the UPF. The (preliminary) content of this structure is as follows:

Offset	Byte 3	Byte 2	Byte 1	Byte 0
0: ID	ECR	L1ID		
1: Hdr Info 1	Placeholder for additional header information, e.g. RUN number			
2: Hdr Info 2	Placeholder for more additional header information			
3: Page Info	Page number		Length within page	
4: Status	Last page indicator	Error + status bits		

Table 5: FIE format

The total size of an FIE is 20 bytes, which already accounts for the proposed RUN number and the additional header field.

10.1.3. Hashing

BufMgrOut uses a hashing algorithm to store event fragment information. Previous prototypes have used the lower 10 bit of the L1ID as a hash-key, which lead to low memory requirements. For the new prototype-RobIn a good approach – in particular for an FPGA-based BufMgrOut – is to match⁷ the hash-key size with the number of buffer pages, because this enables a simple and compact⁸ memory layout. Alternatively – and with a large amount of management memory available – the key-size can be increased as much as possible in order to achieve quick search access.

10.1.4. Hash-entry

Each entry in the fragment lists is composed of a pointer field (previous, next) and the FIE. Entries belonging to the same fragment⁹ are always added or removed sequentially.

⁷ Example: 16k pages correspond to a 14 bit hash-key.

⁸ The first level of indirection is removed, because the first list element is stored directly at the “hash” position.

⁹ Of a fragment with multiple pages.

11. Appendix C

11.1. List of Tables

Table 1: Document Change Record	2
Table 2 : Functions	14
Table 3 : Cost Estimate	18
Table 4 : Multiple ROLs	18
Table 5: FIE format	20

11.2. List of Figures

Figure 1: MFCC block diagram	5
Figure 2: MicroEnable block diagram.....	6
Figure 3: I960 block diagram	7
Figure 4: Sharc block diagram	8
Figure 5: Effect of asynchronous fragment processing	9
Figure 6: Basic RobIn performance	10
Figure 7: Basic Block Diagram	12
Figure 8: Basic Functional Diagram	12