

A SHARC based ROB Complex : design and measurement results

H. Boterenbrood, P. Jansweijer, G.Kieft, R. Scholte¹, R. Slopsema¹, J. Vermeulen

NIKHEF, Amsterdam, the Netherlands

30 March 2000

Abstract : ROB hardware, based on and exploiting the properties of the SHARC DSP and of FPGAs, and the associated software are described. Results from performance measurements and an analysis of the results for a single ROBIn as well as for a ROB Complex with up to 4 ROBIns are presented.

1 Introduction

A ROB Complex based on two different modules, each with a SHARC DSP, is described in this document. The CRUSH (“Compact ROBIn Using a SHARC”) module functions as ROBIn, the ShaSLINK module (“Sharc with S-Link output”) functions as the interface between ROBIns and a PC : input of requests and of decisions and output of event fragments is done via PCI bus, using the PCI interface of the ShaSLINK module. Requests for data and decisions are fanned out by the ShaSLINK module to the ROBIns, while it also takes care of event fragment building.

2 The CRUSH module

The CRUSH module is a test implementation of the ATLAS ROBIn. It provides automatic hardware-controlled storage of the bulk of event fragments - entering from an ROL (Read-Out Link) - in a buffer while some relevant event information (“summary information”) is copied (a few words per event fragment, e.g. Begin-Of-Fragment, Event-ID, End-Of-Fragment). This information together with the start position of the event fragment in the buffer is written to a so-called Paged FIFO, from where the on-board processor can read it. The event fragments are written one after the other into the buffer memory with roll over to the start of the buffer memory once the end is reached. The on-board processor is responsible for preventing overwriting of data by moving data out of the buffer memory if necessary. A block scheme of the CRUSH is presented in 1, the card itself is shown in Figure 2 on page 3.

The on-board processor is an ADSP 21060L (SHARC) processor from Analog Devices with a clock frequency of 40 MHz. Its main features are its on-chip memory (512 kBytes), 6 independently operating communication links (40 MByte/s each) with hardware handshaking and 10 on-chip DMA controllers for transferring data via the links as well as between the external memory bus of the SHARC and its internal memory. Its DMA capabilities make the SHARC very well suited for handling the data and message streams present in a ROB, as multi-channel communication can take place in parallel with buffer management and data processing operations.

1. University of Twente, Enschede, the Netherlands

A useful feature of the SHARC's DMA controllers is their ability to perform so-called chained DMA operations, whereby a block of data containing setup values for the DMA controller registers (the so-called Transmission Control Block (TCB)) is loaded by the DMA controller – without intervention of the processor core – into its registers and the DMA operation is started (DMA controller “auto-initialization”); one of the DMA registers points to the next TCB which is automatically loaded at the end of the current DMA operation. The processor starts the chain by writing the address of the first TCB in the appropriate DMA controller register. It is possible to set up an end-less loop of DMA operations when the last TCB of the chain of TCBs points to the first. The TCBs itself have to be set up only once by the processor. It is also possible to start just one DMA operation in this way. The software sets up all DMA parameters beforehand and at any time afterwards can start the DMA by a single write cycle, i.e. by writing the address of the TCB into the appropriate DMA controller register.

The FPGA is an Altera 10k100A clocked with the same clock as the SHARC processor. The buffer memory is implemented with 1 MByte of ZBT RAM. This type of memory has the advantage that it can operate without wait cycles, also when a read cycle is immediately followed by a write cycle or vice versa. The buffer memory is clocked at 80 MHz, allowing an access from the SHARC and an access from the S-link FIFO every 25 ns. Interfacing to the buffer memory in the FPGA is strongly pipelined. This is not a problem for the S-link, from which up to 160 MByte/s can be transferred into the memory. However, the SHARC requires four system clock cycles to write or read data in or out of the buffer memory, so the transfer speed from buffer memory to SHARC is at maximum 40 MByte/s. This is equal to the bandwidth of one of the SHARC links.

Several functions of the FPGA are controlled by the on-board processor. For example what a 'Begin-Of-Fragment'-word should look like and which words (specified by a word count after a 'Begin-Of-Fragment'-word) have to be copied from the event fragment to the Paged FIFO, can be set through software, using on-board registers.

Compact ROBIN Using a SHARC (CRUSH)

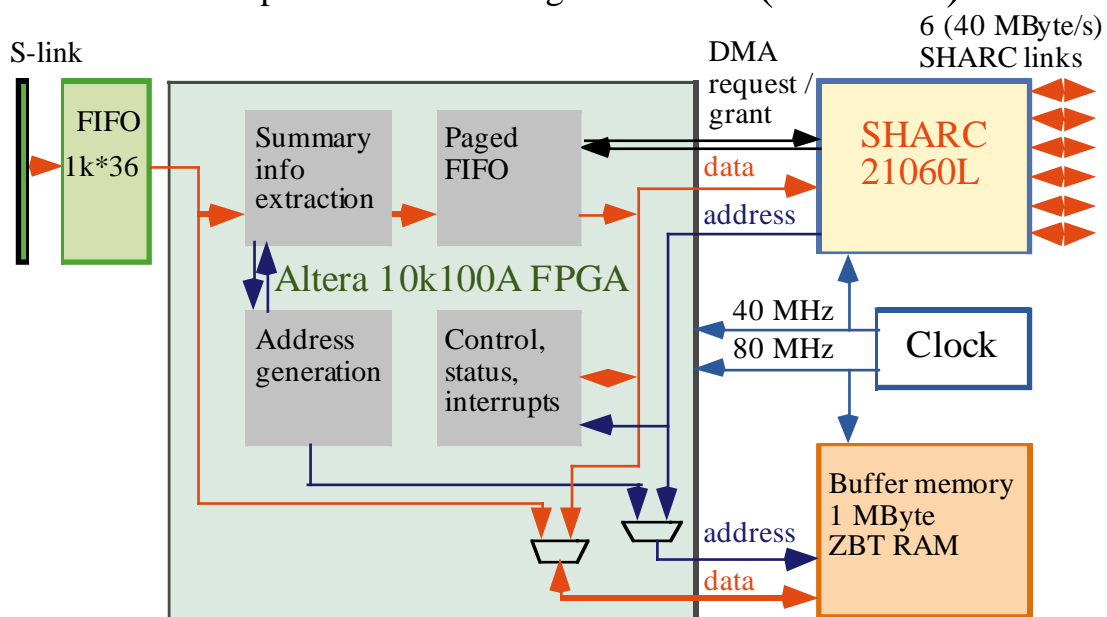


Figure 1. Block scheme of the CRUSH module

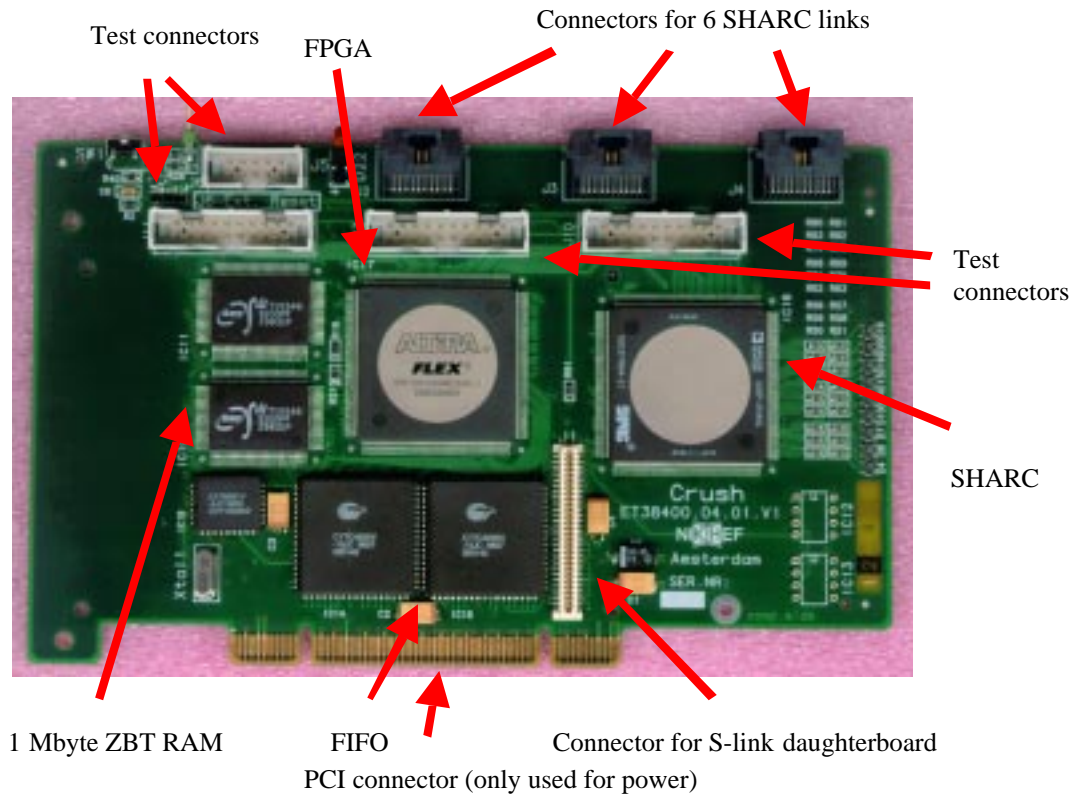


Figure 2. The CRUSH module

The CRUSH board has a PCI board form factor but the CRUSH only takes its power from the PCI-bus. Most components are 3.3 V components, the power dissipation of the complete board (without S-link daughter board) is about 2 - 3 W. The on-board SHARC processor is booted via its Link Port 4; all 6 SHARC Link Ports are available on connectors on the board's edge.

A detailed description of the design of the CRUSH module can be found in [1].

3 Software for the CRUSH module

All actions are triggered by I/O polling in one big loop, introducing potentially some latency in the actions, but avoiding any interrupt routine overhead. All input and output by the SHARC is done under DMA control. All input and output streams are buffered in SHARC on-chip memory.

3 shows a schematical view of software for the CRUSH. The polling loop is indicated by the (blue) arrows connecting the various blocks. Arrows entering from the left or pointing to the left from the CRUSH indicate data flowing into or out of the SHARC. Blocks labelled with “.... input “ indicate places in the code where the status of DMA transfers for input of data from either the paged FIFO of the FPGA (normal operation for summary information (Event Info data)) or from a SHARC link is looked at and where new DMA transfers for input of data are initiated. Event fragment data (“RoID” data for requested RoI data or “EBD” data for accepted event data to be sent to the Event Builder) is transferred with two DMA transfers : first the data is transferred from the buffer memory to a buffer in the internal memory of the SHARC (the bandwidth for these transfers is 40 MByte/s), in the second step the data is transferred via one of the SHARC links.

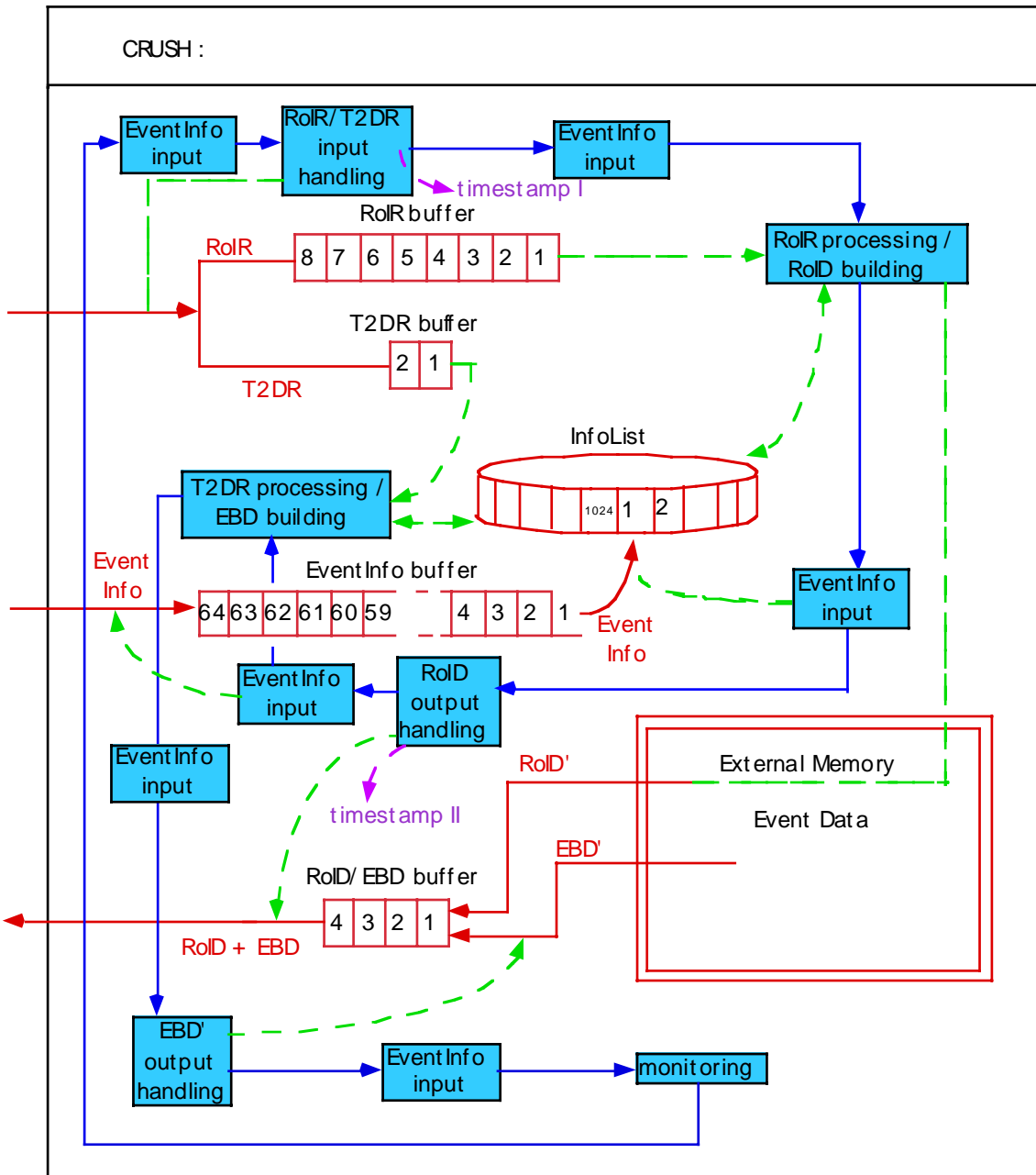


Figure 3. Schematical view of the software for the CRUSH module

The following types of processing can be distinguished :

1. “Event Info input” processing consists of copying event summary information to an ordered list according to event id, after checking whether the location in the list has not already been taken followed by checking whether the buffer memory is (almost) full,
2. RoI request and LVL2 decision block input handling (“RoIR / T2DR input handling”) consists of checking whether a RoI request or a decision record is arriving, starting a DMA for transfer of the remaining information into the “RoIR” buffer or the “T2DR” buffer, detection of the end of transfers and updating of the appropriate buffer pointers,
3. RoI Request processing (“RoIR processing / RoID building”) consists of finding the event summary information in the ordered list for the event id specified in the RoI request message, creation of a header (7 words) in the “RoID/EBD buffer” and of starting a DMA transfer from the

event buffer memory to the “RoID/EBD buffer” in the SHARC internal memory. This is done only if there is an empty slot in the “RoID/EBD buffer” and if a DMA transfer is possible,

4. LVL2 Decision Record processing (“T2DR processing / EBD building”) consists for each decision in the decision record of finding the event summary information with the corresponding event id in the ordered list. For a reject the occupied event fragment buffer space and the entry in the event summary list for the event id is marked as being free. For an accept a header (7 words) is created in the “RoID/EBD buffer”, a DMA transfer is started from the event buffer memory to the “RoID/EBD buffer” in the SHARC internal memory and the entry in the event summary list for the event id is marked as being free. This is done only if there is an empty slot in the “RoID/EBD buffer” and if a DMA transfer is possible. After handling up to 10 decisions “T2DR processing / EBD building” stops, unless all decisions in the record have been processed. In that case first a contiguous block of buffer memory - up to the first event fragment not marked as being free - is added to the free buffer memory space,
5. “EBD’ output handling” checks for the end of a DMA transfer of a fragment of an accepted event from the buffer memory to the “RoID/EBD buffer”: if the transfer is finished the buffer space used is marked as being free,
6. “RoID output handling” initiates DMA transfers for output across one of the SHARC links if there is data in the “RoID/EBD buffer” and the output SHARC link is free.

Keeping track of which parts of the buffer memory are used for storing event fragment data is done as part of the “Event Info input” processing. This includes checking whether there is enough contiguous space left (including a safety margin) in the buffer memory for storing new event data arriving via the ROL. If there is not enough space left the oldest fragment(s) need to be moved from the buffer memory to the internal memory of the SHARC. The occupied space in the buffer can then be used for storing new event fragments. Checking whether there is enough space left is implemented, but the present software cannot yet move event fragments if necessary (but with a large enough buffer memory this should be necessary only for a small fraction of the events). For more details on buffer management see the appendix.

The buffer memory is mapped twice in the memory map of the SHARC. This simplifies the handling of buffer wrap-around situations: it enables transfer of an event fragment by a single DMA operation even when it is “wrapped-around” the end of the buffer.

Currently the software uses less than 10 k words for code (consists of 48-bit words) and about 17 k words for data (32-bit), leaving close to 100 k words (32-bit) available for additional dynamically allocated event-summary and data storage. The software is written in C using the Analog Devices Development Tools version 3.3 for the SHARC processor.

4 Results with the CRUSH module

4.1 Measurement procedure

Two different test set-ups with a single CRUSH module have been used as shown in 4 and 5. They contain a mix of CRUSH and PCISHARC modules. The PCISHARC module is a board developed at NIKHEF with a SHARC processor and an AMCC PCI interface chip. One PCISHARC provides a path for booting and interaction with the CRUSH module. Another PCISHARC module provides the source and destination of the data streams required to run the simulation.

In the set-up of 4 an S-Link input stream is simulated by externally providing a stream of event summaries on one of the SHARC Link Ports (“virtual S-link”). The CRUSH reads the summaries from the Link Port in just the same way as if it reads them from its Paged FIFO. To the SHARC processor there is no difference in handling event summaries entering through the Paged FIFO or through a Link Port. There is no need to provide real S-Link input to test the software because the actual event data flow into the CRUSH’s ring buffer is completely handled by hardware and does not affect the SHARC's processing performance in any way. The advantage of such a simulated or “virtual S-Link” input is that the “virtual S-Link” input rate automatically is limited by either the maximum performance of the ROBIN software (running on the SHARC) or the maximum transfer speed out of the buffer memory (in the tests “fake” event data is read out) or the maximum throughput of the SHARC links. This maximum performance is measured as a function of event fragment size; in different words: the maximum S-Link input rate for a particular event fragment size that can be handled, is measured.

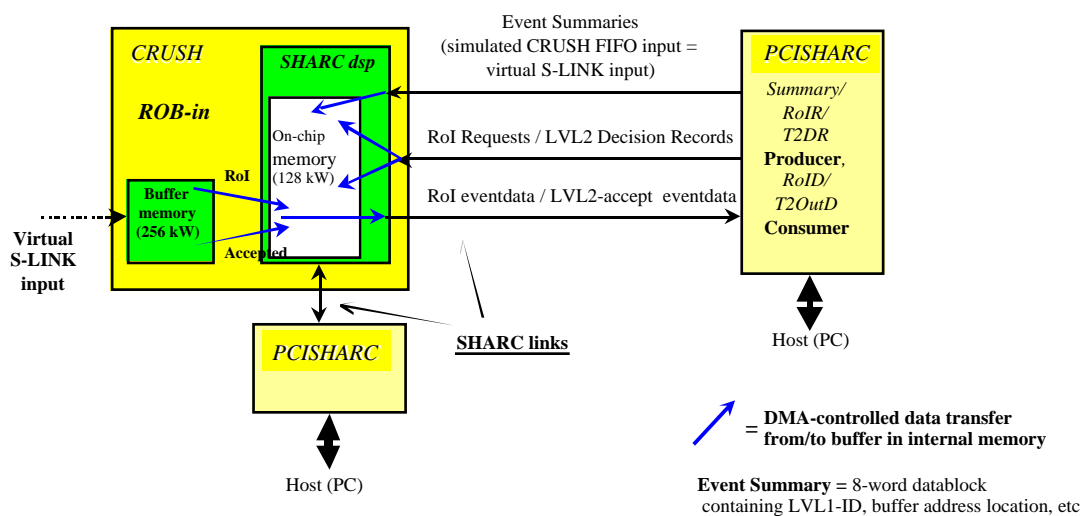


Figure 4. Test set-up with a virtual S-link

Nevertheless it was felt necessary to test at least the proper functioning of the CRUSH with real S-Link data input. To this end the set-up shown in 5 is used. It uses an S-Link input stream originating from either a MicroEnable module [2] or a SLIDAS module [3]. The MicroEnable’s programmable hardware is programmed with the so-called MicroSlate application, turning it into an event fragment S-Link output source. Both MicroEnable and CRUSH are equipped with either a Fiber-Channel S-Link interface (measured throughput ca. 56 MByte/s) or a SCSI-cable S-Link interface (measured throughput ca. 80 MByte/s). The SLIDAS module is an S-Link output source mounted directly on the CRUSH providing an S-Link input data stream of 5, 10, 20, 40, 80 or 160 MByte/s.

Since the event fragment data stream entering the CRUSH via the S-Link is in principle uncontrolled and continuous, the CRUSH itself has to trigger the PCISHARC, so that the PCISHARC can generate the RoI requests and LVL2 decisions according to the required RoI request rate (as a fraction of the event fragment input rate) and the actual event fragment stream arriving across the S-Link. To do this requires an extra communication channel between CRUSH and PCISHARC, which is labeled “RoI Request requests” in 5.

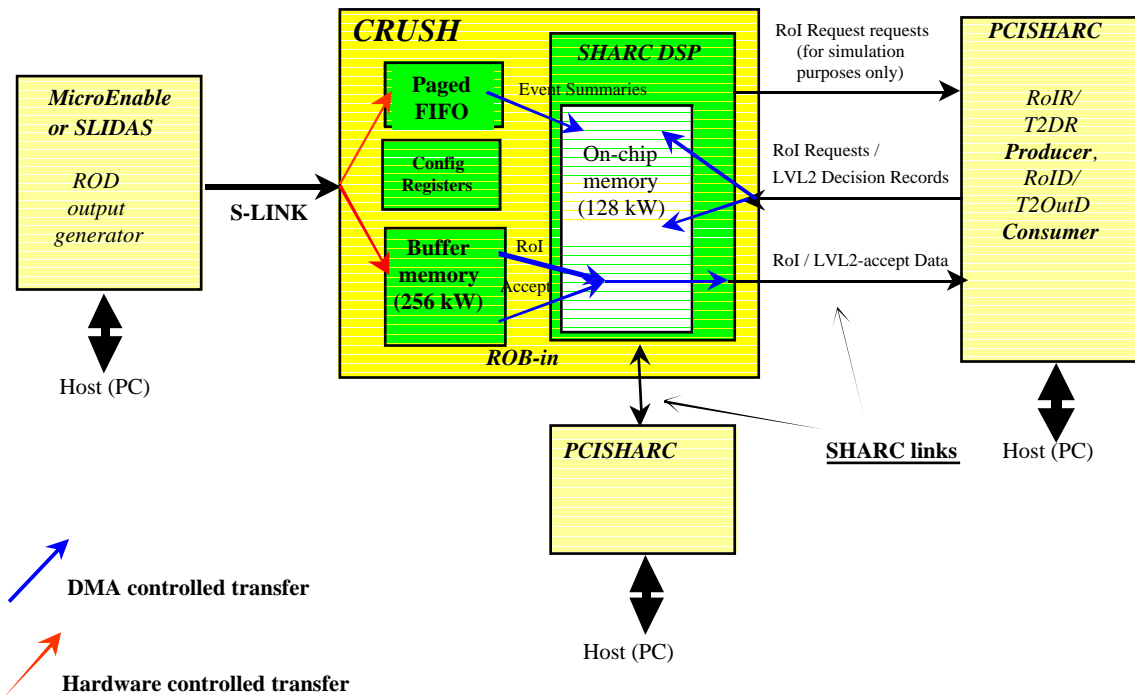


Figure 5. Test set-up with real S-link input

Measurements showed that the “virtual S-link” setup of 4 results in a roughly 10 % lower maximum event fragment rate due to necessary simulation-related overhead than equivalent measurements with the setup of figure 5. Hence measuring with the setup of 4 results in *lower* values for the maximum event fragment rate than would be observed with real S-link input of event data.

4.2 Measurement results : maximum event rate

Measurements were done for event sizes of 256, 512, 768, 1024, 1280, 1536, 2048, 3072 and 4096 Bytes, RoI request fractions of 5, 10, 14, 20, 25, 33, 50 and 100 % and accept fractions of 0, 1, 3, 5 and 10 %. The size of the decision record was kept constant at 100 decisions. The fractions are with respect to the rate of events received via the (virtual) S-link.

For each combination the maximum event fragment rate the CRUSH module can handle was determined. Table 1 and Table 2 contain an overview of the results. The maximum event rate as a function of the event size for the different accept fractions and for RoI request fractions of 10%, 25% and 100 % is shown in 6, 7 and 8 respectively.

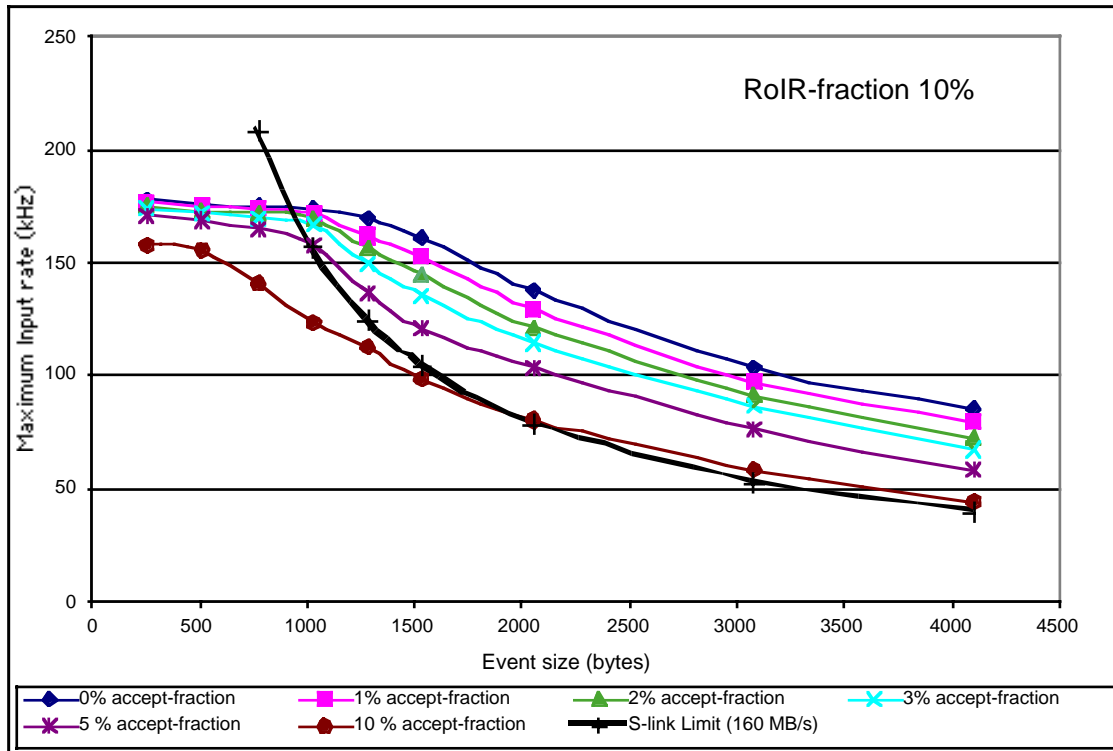


Figure 6. Maximum event rate as a function of fragment size for different accept fractions and a RoI request fraction of 10 %

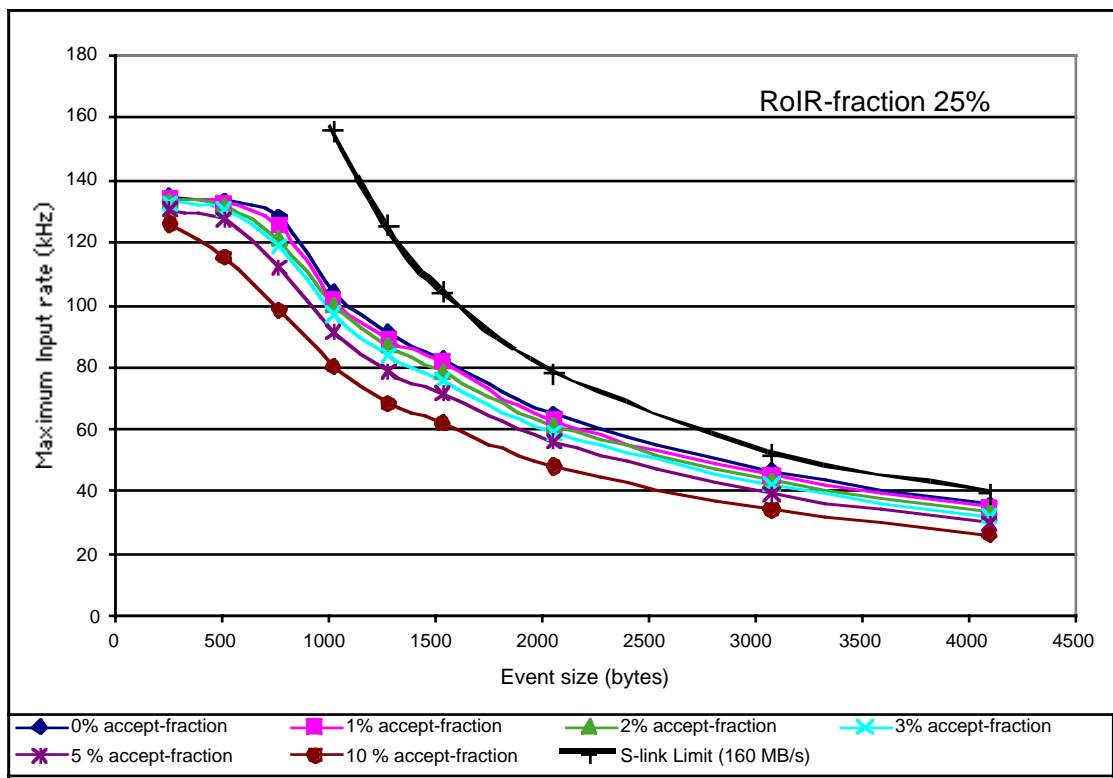


Figure 7. Maximum event rate as a function of fragment size for different accept fractions and a RoI request fraction of 25 %

5 % RoI request fraction						
Bytes	0% Acc.	1% Acc.	2% Acc.	3% Acc.	5% Acc.	10% Acc.
256	187.3	184.4	183.8	183.6	184.7	165.2
512	186.8	183.3	182.9	183.1	183.4	164.1
768	185.5	182.8	182.5	181.4	179.0	155.1
1024	185.6	182.0	181.0	180.2	174.4	137.3
1280	185.3	181.7	180.1	178.9	171.8	128.7
1536	184.5	180.8	179.3	177.6	166.2	114.6
2048	182.7	178.2	178.2	170.5	148.5	102.3
3072	178.9	167.3	147.3	132.6	109.9	76.8
4096	150.5	137.6	117.3	108.0	-	-
10 % RoI request fraction						
Bytes	0% Acc.	1% Acc.	2% Acc.	3% Acc.	5% Acc.	10% Acc.
256	177.4	176.4	174.8	173.2	170.2	158
512	176.0	174.7	173.0	171.4	168.3	155.9
768	174.7	173.2	171.6	169.7	165.1	141.0
1024	173.3	172.0	169.9	166.8	157.7	123.8
1280	169.6	162.1	157.1	149.3	137.0	112.0
1536	160.9	152.9	144.8	135.8	120.7	98.5
2048	137.9	129.1	121.8	114.6	103.3	80.6
3072	103.6	97.3	91.2	86.8	76.6	58.4
4096	85.7	79.5	71.8	66.8	58.4	44.3
14.3 % RoI request fraction						
Bytes	0% Acc.	1% Acc.	2% Acc.	3% Acc.	5% Acc.	10% Acc.
256	162.9	162.2	160.8	159.5	157.0	150.8
512	162.0	160.9	159.3	157.7	154.0	144.4
768	160.0	159.1	157.3	155.1	151.3	129.7
1024	157.9	151.3	146.3	140.5	128.5	105.9
1280	138.2	134.3	129.1	122.7	112.3	92.7
1536	128.1	122.7	117.7	112.2	102.4	84.0
2048	103.9	101.1	96.0	91.6	84.0	68.2
3072	77.7	73.9	70.1	67.2	60.6	48.4
4096	61.9	57.4	54.0	51.1	46.0	36.8
20 % RoI request fraction						
Bytes	0% Acc.	1% Acc.	2% Acc.	3% Acc.	5% Acc.	10% Acc.
256	146.6	145.9	144.8	143.8	141.6	136.5
512	145.2	144.8	143.4	141.5	138.8	129.3
768	142.8	141.7	140.2	136.9	129.0	111.8
1024	124.8	121.8	119.4	115.0	106.9	91.0
1280	109.0	107.0	103.3	100.0	93.1	79.6
1536	99.3	95.4	91.7	89.2	82.0	69.7
2048	78.9	77.5	74.7	71.7	67.2	56.6
3072	57.5	55.1	52.8	51.0	47.0	39.4
4096	43.6	42.2	40.3	38.7	35.7	29.9

Table 1: Overview of the measured maximum event rates (in kHz) for RoI request fractions up to 20 %, “Acc” refers to the accept fraction

25 % RoI request fraction						
Bytes	0% Acc.	1% Acc.	2% Acc.	3% Acc.	5% Acc.	10% Acc.
256	135	134.3	133.4	132.4	130.5	126.2
512	133.5	132.8	131.6	130.2	127.7	115.2
768	128.1	125.2	120.9	118.8	112.1	98.6
1024	103.9	101.4	100.0	96.9	91.4	80.2
1280	91.2	88.7	86.5	83.8	78.8	68.7
1536	82.7	81.4	78.6	76.1	71.5	62.1
2048	64.5	62.9	60.9	59.0	55.9	48.3
3072	46.3	45.0	43.7	42.3	39.6	34.0
4096	35.7	34.7	33.4	32.3	30.2	26.1
33 % RoI request fraction						
Bytes	0% Acc.	1% Acc.	2% Acc.	3% Acc.	5% Acc.	10% Acc.
256	119.3	118.7	118.0	117.2	115.7	112.1
512	117.7	117.1	116.1	115.2	113.1	101.4
768	103	101.6	99.3	97.4	92.7	83.5
1024	85.9	83.9	81.9	80.0	76.1	68.1
1280	73.3	71.3	69.6	67.9	64.7	57.9
1536	63.9	62.1	60.5	58.9	56.0	49.9
2048	49.6	48.9	47.6	46.4	44.1	39.2
3072	34.9	34.1	33.2	32.3	31.0	27.5
4096	26.8	26.2	25.4	24.8	23.5	20.8
50 % RoI request fraction						
Bytes	0% Acc.	1% Acc.	2% Acc.	3% Acc.	5% Acc.	10% Acc.
256	96.4	96.1	95.6	95.1	94.1	91.3
512	90.5	88.1	86.7	85	82.3	75.8
768	76.1	73.8	72.5	71.4	69.1	64.0
1024	59.0	58.0	57.0	56.0	54.1	49.8
1280	49.5	48.5	47.6	46.8	45.2	41.6
1536	43.5	42.6	41.9	41.2	39.8	36.7
2048	34.2	33.7	33.1	32.5	31.5	29.0
3072	23.8	23.7	23.2	22.8	22.0	20.1
4096	18.3	17.8	17.5	17.2	16.6	15.2
100 % RoI request fraction						
Bytes	0% Acc.	1% Acc.	2% Acc.	3% Acc.	5% Acc.	10% Acc.
256	62.10	62	61.8	61.5	61.1	59.1
512	51.6	51.2	50.8	50.2	49.2	46.8
768	38.6	38.4	37.9	37.4	36.8	35.2
1024	30.5	30.3	30.1	29.8	29.2	28.0
1280	26.8	26.6	26.3	26.1	25.6	24.4
1536	22.7	22.5	22.2	22.0	21.6	20.6
2048	17.1	17.0	16.8	16.7	16.4	15.7
3072	11.9	11.8	11.7	11.6	11.4	10.9
4096	9.2	9.1	9.0	8.9	8.7	8.3

Table 2. Overview of the measured maximum event rates (in kHz) for RoI request fractions of 25 % and higher, "Acc" refers to the accept fraction

In 9 the inverse of the maximum event rate is shown as a function of the event size for the different accept fractions and a RoI request fraction of 50 %. For small event fragment sizes the maximum event rate is independent of the size, for larger event fragments the inverse rate is seen to be approximately linear dependent on the size. This leads to the conclusion that for small event fragments processing in the SHARC limits the maximum rate, while for larger fragment sizes the bandwidth of the output link (40 MByte/s) is the limiting factor.

After some tries it was found that the following formulae represented the data best :

$$(1/\text{rate}) = \max \{ (1/\text{rate})_a , (1/\text{rate})_b \}$$

$$(1/\text{rate})_a = c_1 + c_2 \cdot R + c_3 \cdot A$$

$$(1/\text{rate})_b = c_4 + c_5 \cdot R + c_6 \cdot A + c_7 \cdot E \cdot (R+A)$$

with R : RoI request fraction, A: accept fraction and E : event fragment size.

The results were fitted with a ROOT-macro, using TMinuit, a class interfacing to the FORTRAN minimizing program Minuit. Two fits were made, one with c_7 fixed to 25 ns per byte (corresponding to the 40 MByte/s bandwidth of the output link) and one with c_7 free. The latter fit resulted in a value of 24.9 ns for this parameter. The results for the parameters (all values are in microseconds, except for c_7 which has the dimension microsecond per byte) are presented in Table 3. It was also attempted to make separate fits of the equations for $(1/\text{rate})_a$ and $(1/\text{rate})_b$, for each fit using a different set of selected data points. The resulting parameters are also presented in Table 3. In 10 the distribution of the ratio of predicted and measured rate for all measurements (430 in total) for the overall fit with c_7 free is shown . A similar distribution for the overall fit with c_7 fixed looks almost the same. In 11 the distribution of the ratio of predicted and measured rates for all measurements using the parameters obtained with the two separate fits is shown. A somewhat narrower peak is observed. The distributions are asymmetrical with a tail for values lower than 1, i.e. the largest deviations between measured and predicted rates occur for predicted rates which are too low.

Type of fit	c_1	c_2	c_3	c_4	c_5	c_6	c_7
1 fit, c_7 free	4.70	11.29	6.69	1.35	4.62	4.83	0.0249
1 fit, c_7 fixed	4.69	11.34	6.94	1.34	4.46	4.63	0.0250
2 fits, c_7 free	4.54	11.53	7.51	1.31	5.18	4.84	0.0249

Table 3. Fitted values of parameters

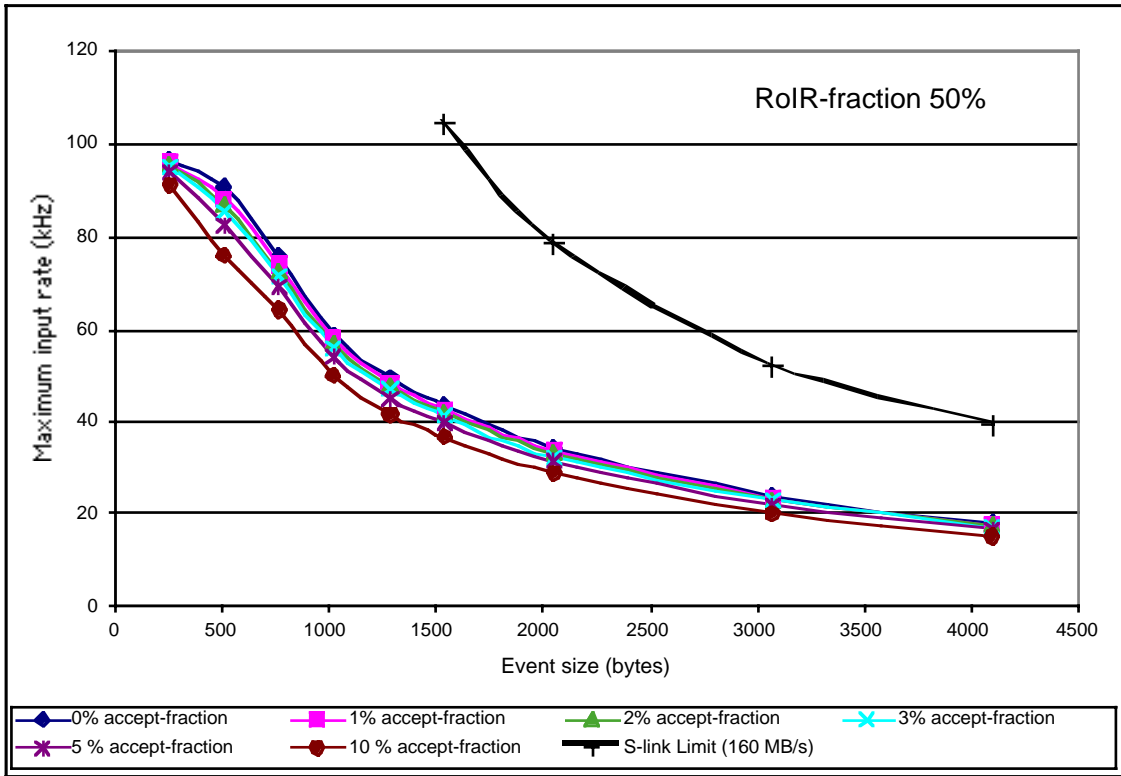


Figure 8. Maximum event rate as a function of fragment size for different accept fractions and a RoI request fraction of 50 %

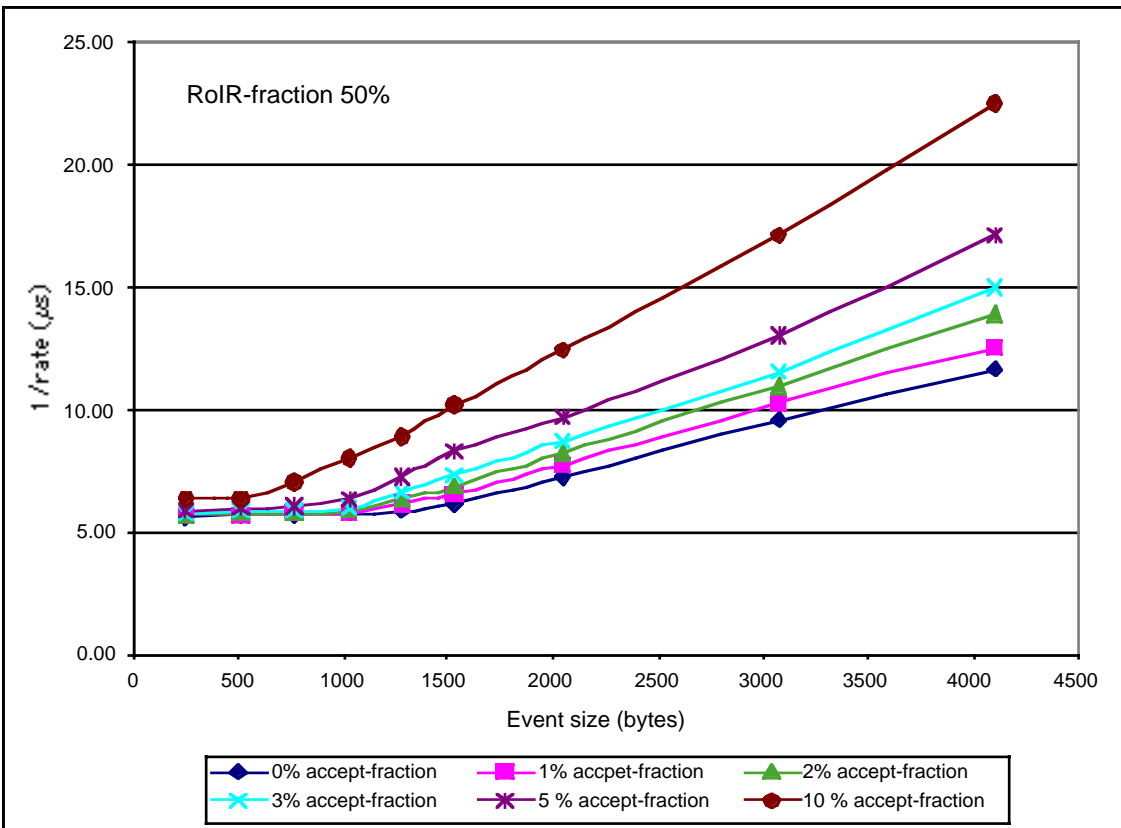


Figure 9. The inverse of the maximum event rate as function of the event size

:

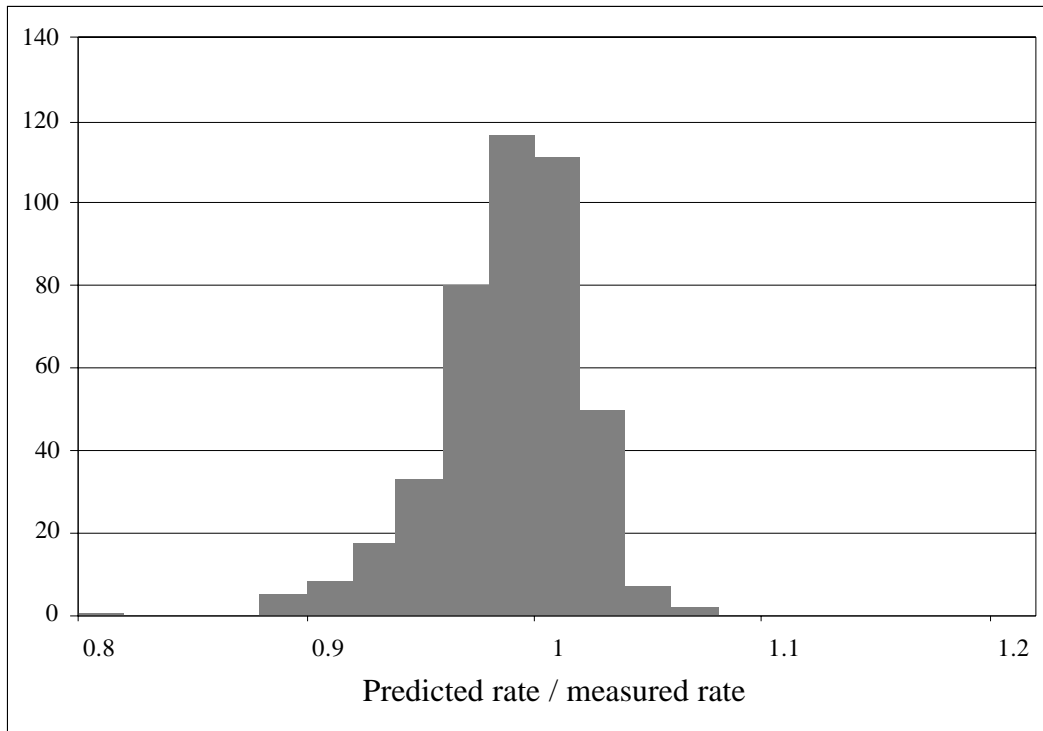


Figure 10. The ratio of predicted and measured maximum event rate for the overall fit with c_7 free

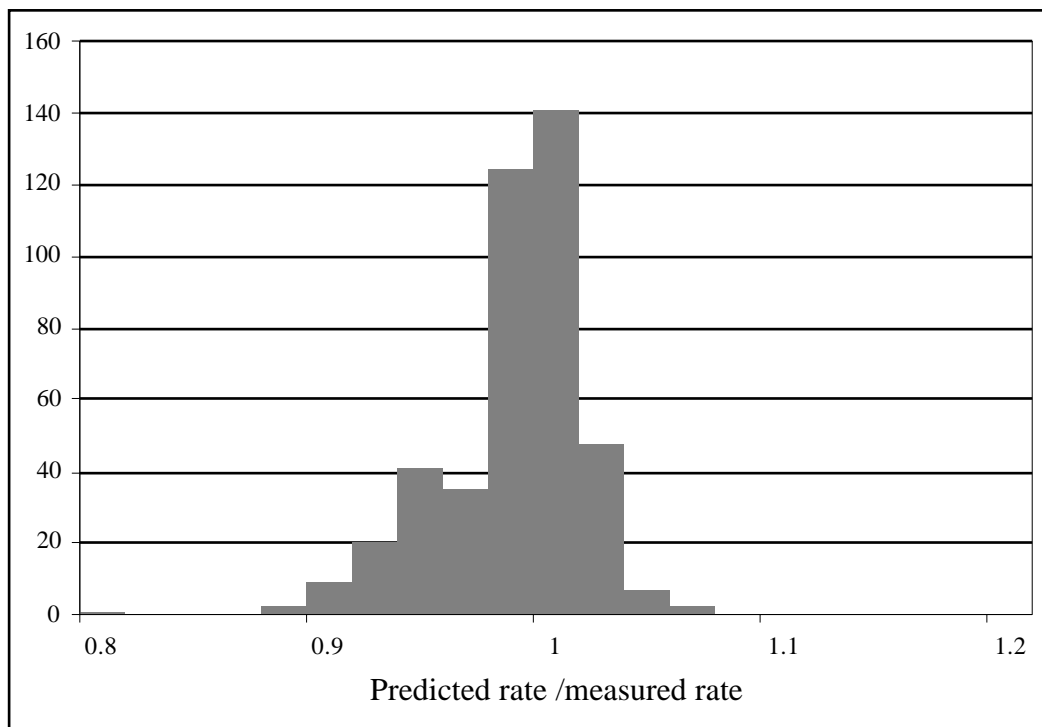


Figure 11. The ratio of predicted and measured maximum event rate for all measurement results if two separate fits are made.

4.3 Measurement results : latency

For different event sizes, RoI request fractions and acceptance fractions the latency, i.e. the time between the arrival of a RoI request and the start of the output to the link of the event data requested, has been measured. For the time measurements the internal clock of the SHARC processor on the CRUSH (clock frequency : 40 MHz) has been used. For each RoI request two timestamps were made (see 3). The first timestamp was made when the DMA transfer getting the RoI request from the input link finished, the second when a DMA transfer started to output the relevant event data via the output link. Both timestamps, together with the length, event id and ROB id, were copied into the header of the RoI data. In the PCISHARC module receiving the RoI data the timestamps were subtracted and put in a histogram. In the same way as described in the previous section the maximum event rate was also determined.

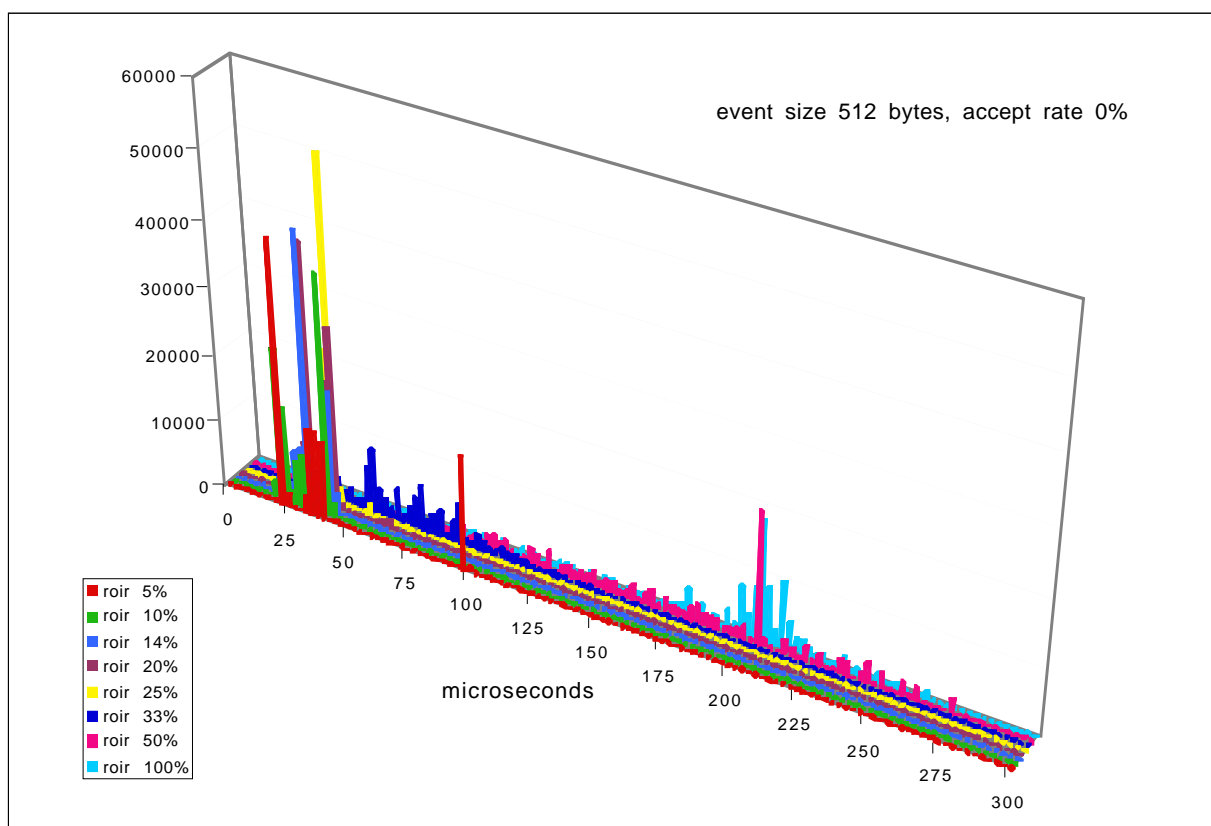


Figure 12. Latency for different RoI request fractions and an event size of 512 bytes and an accept fraction of 0 %

The latency measurements hardly have an effect on the performance of the CRUSH module as the length of all messages with event data is not changed and the time to make the timestamps is negligible. Making the histogram in the PCISHARC module takes more time. However, it was found that the maximum event rates are the same as without latency measurement, showing that the CRUSH module is the limiting factor in the system.

Measurements have been done for event sizes of 256, 512, 768, 1024, 1280, 1528, 2048, 3072 and 4096 bytes, for RoI Request fractions of 5, 10, 14.3, 20, 25, 33, 50 and 100 %, and for acceptance fractions of 0, 1, 2, 3, 5 and 10 %. Some of the results are shown in 12 and 13. The peaks are caused by the structure of the software and the size of the buffers (see 3).

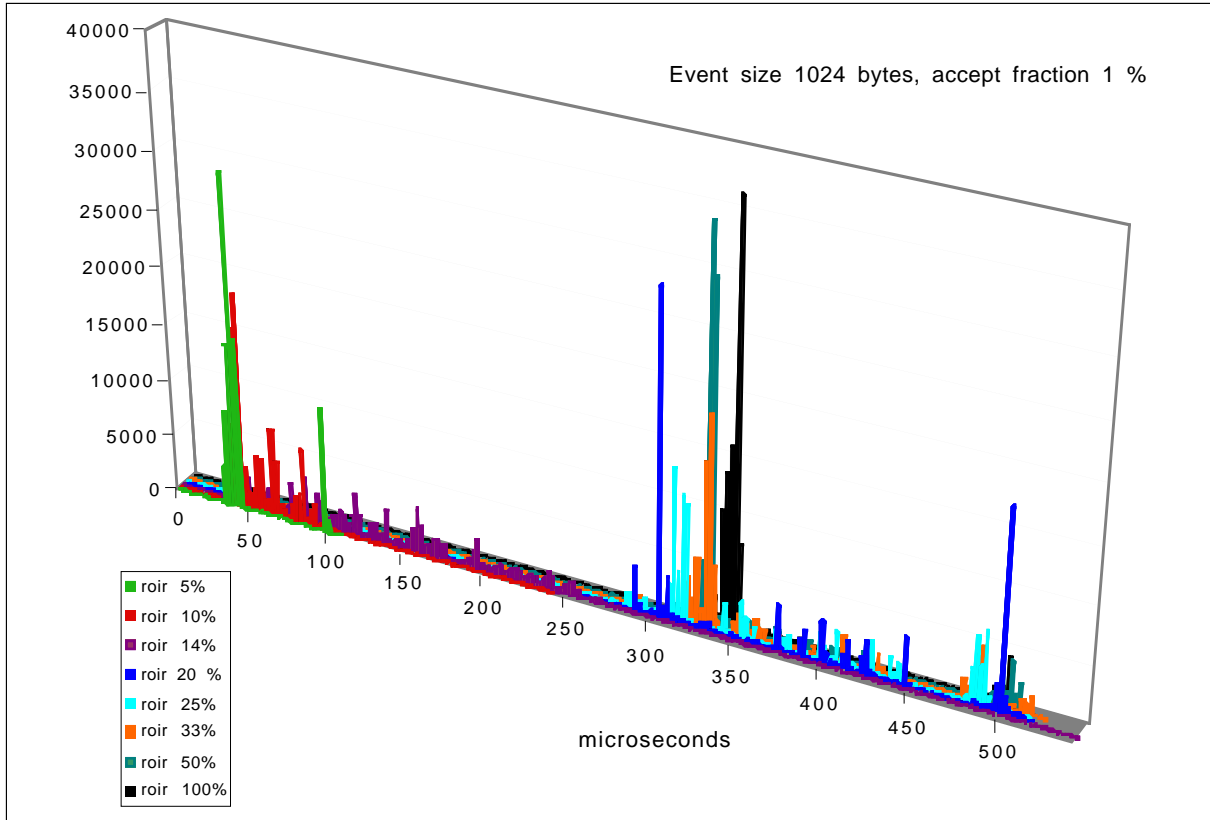


Figure 13. Latency for different RoI request fractions and an event size of 1024 bytes and an accept fraction of 1 %

5 The ShaSLINK module

The ShaSLINK (Sharc + S-link source) module is a PCI card with a SHARC processor, a PCI interface (implemented with the PLX 9054) and an S-link source interface. A block scheme is presented in 14, the module itself is shown in 15. A detailed description of the module can be found in [4].

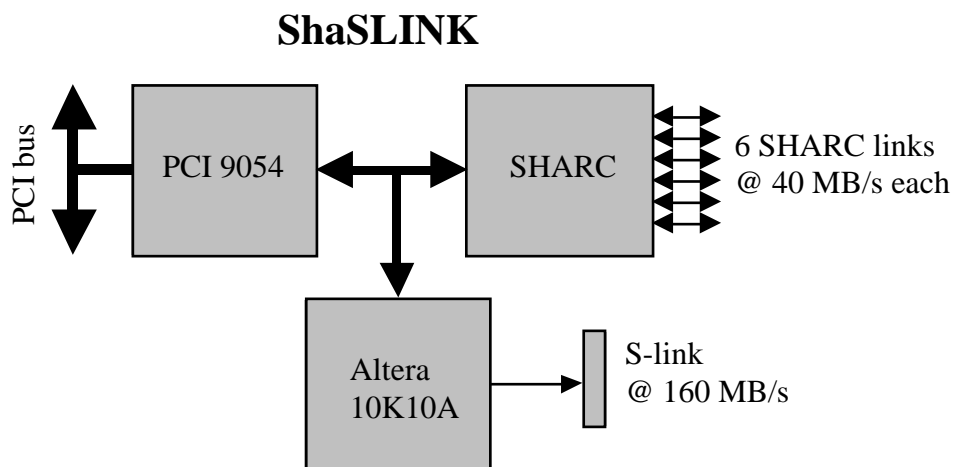


Figure 14. Block scheme of the SHaSLINK module

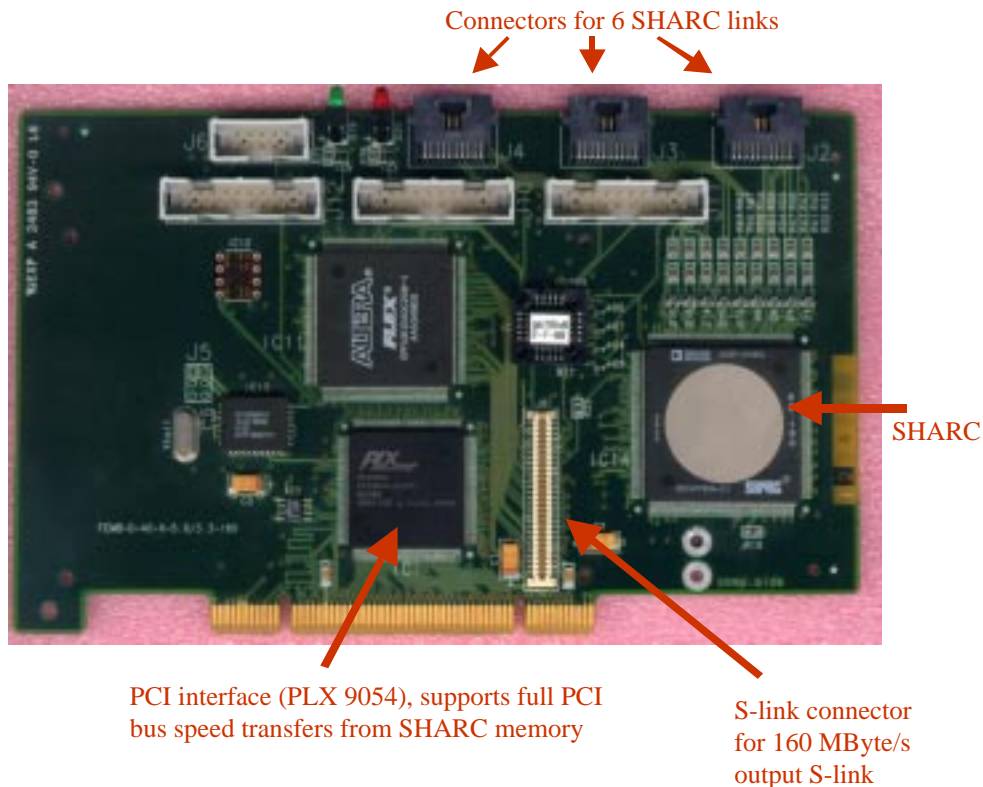


Figure 15. The ShaSLINK module

6 Configuration of a ROB Complex with CRUSH and ShaSLINK modules

A ROB Complex has been built from two CRUSH and a ShaSLINK module connected with SHARC links and a PC. Output can go either via PCI bus or S-link. In this document output via S-link is not considered. In 16 the lay-out of the ROB Complex test setup built with two CRUSH modules, two ShaSLINK modules, a PCISHARC module and two PCs is shown.

The PCISHARC module is used for distributing RoI requests and LVL2 decisions to the “ROB-Ins”. For the test measurements again “virtual S-link input” (see Section 4.1) has been used: the event summary information, during normal operation read from the Paged FIFO in the FPGA is sent to each CRUSH module via a SHARC link. One of the ShaSLINK modules (the EVT-GEN block in the figure) generates the event summary information, while the other functions as “ROB-MUX”. Each CRUSH module functions as a single ROBIN, but data transferred via one of its links can also be transferred via a second link. In this way from the point of view of the ROB-MUX 2 ROBIns are sending their data in stead of a single ROBIN. The setup therefore allows to obtain information on the behavior of a ROB complex with up to 4 ROBIns.

One of the PCs, a 200 MHz Pentium Pro machine, generates RoI requests and LVL2 accepts / rejects and waits for event fragments to arrive. The RoI request rate and the accept rate relative to the LVL1 rate can be specified. The number of requests handled by the ROB complex at any given time (i.e. passed to the ROB-MUX) is bound to a maximum. Again the maximum event fragment rate is measured.

The configurations are :

- “1” : 1 CRUSH module used, data is transferred to the ROB-MUX via 1 SHARC link,
- “1 + 1” : 2 CRUSH modules used, each transfers event data to the ROB-MUX via one SHARC link, the ROB-MUX builds events from the fragments from both input data streams,
- “2 + 1” : 2 CRUSH modules used, one transfers event data via a SHARC link and in parallel the same event data via a second SHARC link to the ROB-MUX and one transfers event data to the ROB-MUX via one SHARC link, the ROB-MUX builds events from the fragments from the three input data streams,
- “2 + 2” : 2 CRUSH modules used, both transfer event via a SHARC link and in parallel the same event data via a second SHARC link to the ROB-MUX, the ROB-MUX builds events from the fragments from the four input data streams.

In 17 and 18 results for the inverse rate as function of the fragment size and for 1 % accept fraction are presented graphically. The same behavior as for a single CRUSH module is observed : for small fragments the rate is independent of the fragment size, for larger fragments the inverse rate increases linearly with the fragment size, indicating that there is a bandwidth limitation. For the “1” and “1 + 1” configurations the limitation comes from the SHARC link(s). The maximum internal bandwidth of the SHARC of the ShaSLINK for simultaneous input via its links and output via its external bus interface is 80 MByte/s. This limits the output bandwidth for the other configurations (and for the “1 + 1” configuration as well, but in this case the limit is the same as the maximum throughput via two SHARC links). The measurement results from Table 5 were fitted with almost the same formulae as used in Section 4.2 :

$$(1/\text{rate}) = \max \{ (1/\text{rate})_a, (1/\text{rate})_b, (1/\text{rate})_c \}$$

$$(1/\text{rate})_a = c_1 + c_2 \cdot R + c_3 \cdot A$$

$$(1/\text{rate})_b = c_4 + c_5 \cdot R + c_6 \cdot A + c_7 \cdot E \cdot (R+A)$$

$$(1/\text{rate})_c = 1 / 165 \text{ (“1” configuration)} \text{ or } 1 / 120 \text{ (other configurations)}$$

The maximum event rate for small RoI request and accept fractions and small event fragments is about 165 kHz for the “1” configuration and about 120 kHz for the “1 + 1”, “2 + 1” and “2 + 2” configurations. These numbers are the same for different RoI request and accept fractions and fragment sizes. This shows that the measurement procedure rather than the performance of the CRUSH modules and ROB-MUX determines these maximum rates. The reduction of the maximum event rate for the “1 + 1”, “2 + 1” and “2 + 2” configurations can be attributed to the EVT-GEN, which has to generate and send event summary information at twice the full event rate for these configurations. For higher RoI request fractions and still for small event fragments a maximum rate is found which depends on the RoI request and accept fractions, but not on the event fragment size. This is the behavior described by the equation for $(1/\text{rate})_a$. Only measured rates lower than 165 (“1” configuration) or 120 kHz (“1 + 1”, “2 + 1” and “2 + 2” configurations) have been used in fitting it. For the larger event fragments the equation for $(1/\text{rate})_b$ was fitted to the measured rates. In view of the limited number of accept fractions it was not attempted to do a combined fit. The measurement results used for each fit were selected with the help of plots like shown in 17 and 18.

5 % RoI request fraction												
Bytes	1 % accept fraction				2% accept fraction				5 % accept fraction			
	1	1+1	2+1	2 + 2	1	1+1	2+1	2 + 2	1	1+1	2+1	2 + 2
256	165	120	120	120	165	120	120	120	164	119	119	119
512	165	120	120	120	164	119	119	119	163	119	119	119
1024	164	120	120	119	163	119	119	119	160	118	118	115
2048	162	119	118	117	161	118	118	102	133	117	86	66
3072	157	118	114	84	146	117	98	73	106	81	64	48
10 % RoI request fraction												
Bytes	1 % accept fraction				2% accept fraction				5 % accept fraction			
	1	1+1	2+1	2 + 2	1	1+1	2+1	2 + 2	1	1+1	2+1	2 + 2
256	164	119	119	120	163	119	119	119	160	119	119	112
512	163	119	119	119	162	119	119	119	159	118	118	107
1024	159	119	119	107	159	119	119	100	152	116	100	80
2048	128	116	84	67	116	110	77	61	99	82	62	49
3072	96	92	63	47	91	85	58	44	73	62	48	35
4096	74	70	49	37	69	67	45	34	58	46	35	27
14.3 % RoI request fraction												
Bytes	1 % accept fraction				2% accept fraction				5 % accept fraction			
	1	1+1	2+1	2 + 2	1	1+1	2+1	2 + 2	1	1+1	2+1	2 + 2
256	150	119	119	110	149	119	119	106	147	119	106	88
512	149	119	119	109	148	119	119	104	146	118	100	84
1024	138	119	98	78	137	118	93	74	121	98	79	62
2048	95	90	62	48	90	85	59	45	79	67	52	38
3072	71	67	46	34	68	62	43	32	57	49	37	28
4096	54	53	35	27	51	50	33	25	45	37	28	21
20 % RoI request fraction												
Bytes	1 % accept fraction				2% accept fraction				5 % accept fraction			
	1	1+1	2+1	2 + 2	1	1+1	2+1	2 + 2	1	1+1	2+1	2 + 2
256	136	119	100	82	134	119	96	77	132	114	83	70
512	134	118	97	80	133	114	94	76	128	97	81	66
1024	108	95	72	57	105	92	69	55	92	79	62	48
2048	72	66	46	35	68	63	44	34	61	52	40	30
3072	53	49	34	25	51	46	32	24	44	39	29	22
4096	41	39	26	20	38	37	25	19	35	30	22	17
25 % RoI request fraction												
Bytes	1 % accept fraction				2% accept fraction				5 % accept fraction			
	1	1+1	2+1	2 + 2	1	1+1	2+1	2 + 2	1	1+1	2+1	2 + 2
256	125	103	82	66	123	100	79	63	118	94	70	58
512	123	95	79	64	120	92	77	62	111	80	68	55
1024	91	78	58	47	88	76	56	45	80	67	53	40
2048	59	54	37	28	56	51	36	27	51	44	32	25
3072	43	40	27	20	42	38	26	20	37	32	24	18
4096	33	31	21	16	32	30	20	15	29	22	18	14

Table 4. Maximum event fragment rates for different RoI request rate fractions, accept fractions, fragment sizes as measured with the system of 16 for the configurations described in the text for RoI request fractions up to 25 %

33.3 % RoI request fraction												
Bytes	1 % accept fraction				2% accept fraction				5 % accept fraction			
	1	1+1	2+1	2 + 2	1	1+1	2+1	2 + 2	1	1+1	2+1	2 + 2
256	103	78	63	49	100	77	61	48	93	70	56	45
512	99	72	61	49	96	71	59	48	90	63	54	45
1024	71	60	43	35	69	58	43	35	64	53	41	32
2048	45	40	29	22	43	39	28	21	40	35	25	20
3072	32	30	20	15	32	29	20	15	29	26	19	14
4096	25	24	16	12	24	23	15	12	23	20	14	11
50 % RoI request fraction												
Bytes	1 % accept fraction				2% accept fraction				5 % accept fraction			
	1	1+1	2+1	2 + 2	1	1+1	2+1	2 + 2	1	1+1	2+1	2 + 2
256	70	51	42	34	69	52	42	33	65	49	39	32
512	69	49	41	33	67	48	41	33	64	45	38	31
1024	49	41	29	24	48	40	29	24	45	38	29	22
2048	30	27	19	15	30	27	19	14	28	25	18	14
3072	22	20	14	10	22	20	13	10	20	18	13	10
4096	17	16	11	8.1	17	16	11	8.0	16	14	9.5	7.7
100 % RoI request fraction												
Bytes	1 % accept fraction				2% accept fraction				5 % accept fraction			
	1	1+1	2+1	2 + 2	1	1+1	2+1	2 + 2	1	1+1	2+1	2 + 2
256	35	27	21	17	35	26	21	17	34	26	21	17
512	35	25	21	17	35	25	21	17	34	24	20	16
1024	25	21	15	12	25	21	15	12	24	20	15	12
2048	15	14	10	7.2	15	13	10	7.1	15	13	9.4	7.3
3072	11	10	7.0	5.3	11	10	6.9	5.2	11	10	6.9	5.3
4096	8.7	7.9	5.4	4.1	8.5	7.9	5.3	4.0	8.3	7.8	5.2	4.1

Table 5. Maximum event fragment rates for different RoI request rate fractions, accept fractions, fragment sizes as measured with the system of 16 for the configurations described in the text for RoI request fractions of 33.3 % and higher.

The results of the fits are presented in Table 6 (all values are in microseconds, except for c_7 which has the dimension microsecond per byte). The distribution of the ratio of predicted and measured maximum rate for all 564 measurements is presented in 19. For the “2 + 2” configuration the dependence on A was not taken into account for $(1/\text{rate})_b$ as a negative value was found for the c_6 parameter if A was used in the fit. For the “1 + 1” configuration c_1 and c_4 were found to have small negative values. The parameters in Table 6 result from a fit with c_1 and c_4 set to 0.

The maximum bandwidth available for output of data can be inferred from the value found for c_7 : it is the 40 MByte/s of a single SHARC link for the “1” configuration. For the other configurations the merging of the fragments has to be taken into account, so the maximum bandwidth is either $2 / 0.0250 = 80$ MByte/s, $3 / 0.0403 = 74$ MByte/s or $4 / 0.0512 = 78$ MByte/s. These results confirm the expectation that the maximum throughput of the ROB-MUX is limited to 80 MByte/s by the internal bandwidth of the SHARC for simultaneous data input via the links and output via the external bus interface. The traffic on the PCI bus was also studied with a PCI bus analyzer. Within bursts the data was flowing at the maximum rate. However, the bursts were interrupted regularly, which is probably due to the throughput of the ROB-MUX being smaller than the 132 MByte/s bandwidth of the PCI bus.

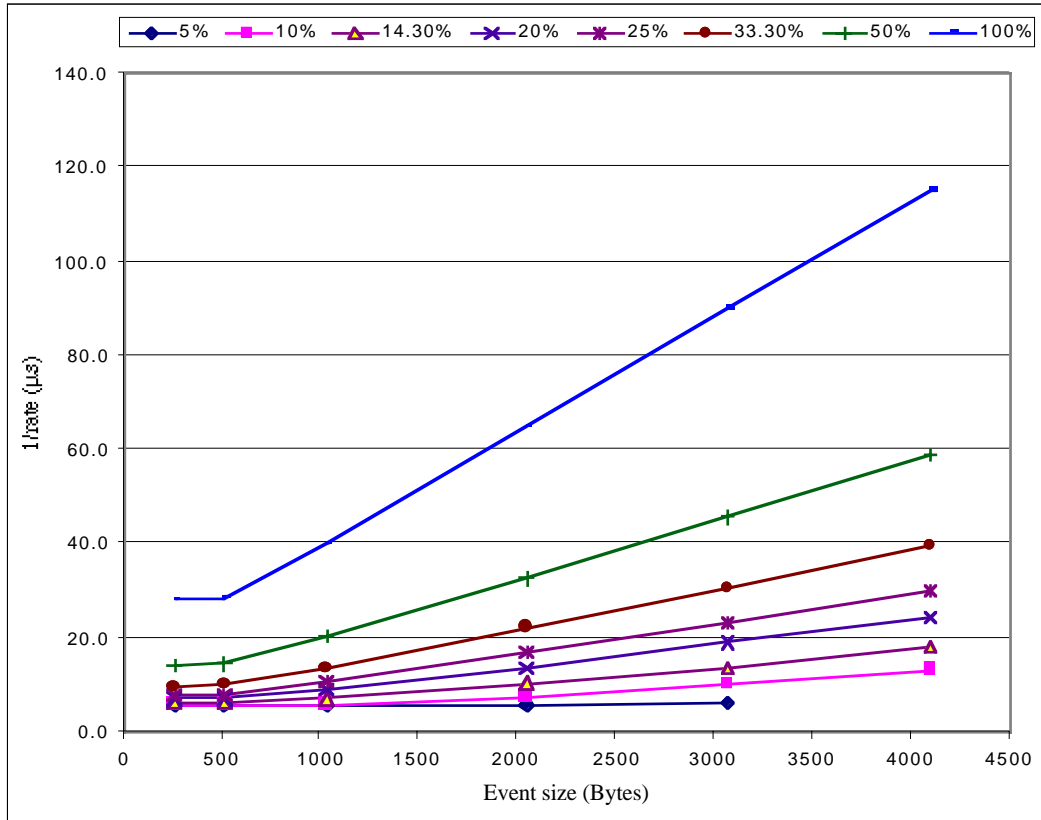


Figure 17. The inverse of the maximum event rate as function of the event size for different RoI request fractions (accept fraction is 1%) in the “1” configuration.

The parameters depend more strongly on the RoI request fraction than found for a single ROBIN. This results in a faster decrease of the maximum event rate for increasing RoI request fraction, which can be attributed to the action of the ROB-MUX and to the generation of event summary information and of RoI request and accept messages. The same measurements, but with output of event data to the ROB-MUX switched off could provide more insight in the limitations due to driving the ROBIns. However, these have not been done at the time of writing this note. The measured maximum rates hence provide lower bounds to the rates that could be obtained in the real application.

Configuration	c_1	c_2	c_3	c_4	c_5	c_6	c_7
1	0.68	27.4	23.8	0.78	13.7	9.57	0.0244
1 + 1	--	38.8	36.8	--	20.4	55.8	0.0250
2 + 1	0.38	46.1	51.2	0.69	17.7	5.49	0.0403
2 + 2	0.36	58.3	51.6	1.32	28.8	--	0.0512

Table 6. Parameters obtained from fitting the equations shown in the text to the measurement results .

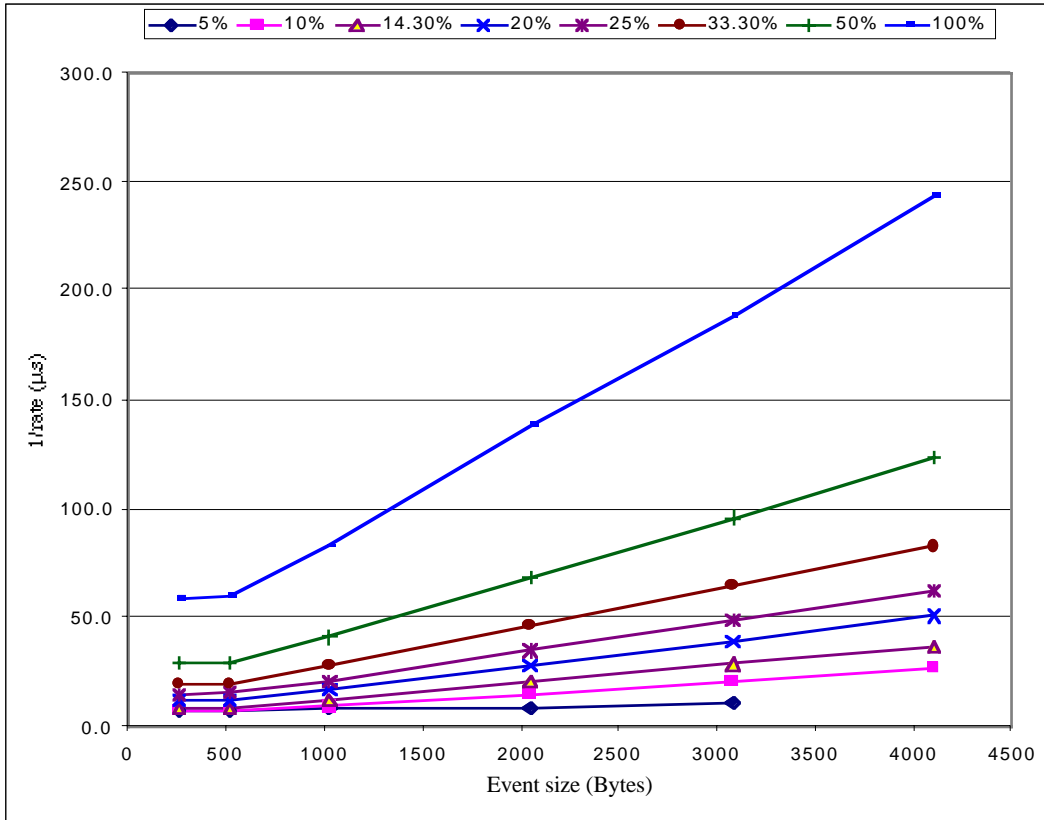


Figure 18. The inverse of the maximum event rate as function of the event size for different RoI request fractions (accept fraction is 1 %) in the “2 +2 “ configuration.

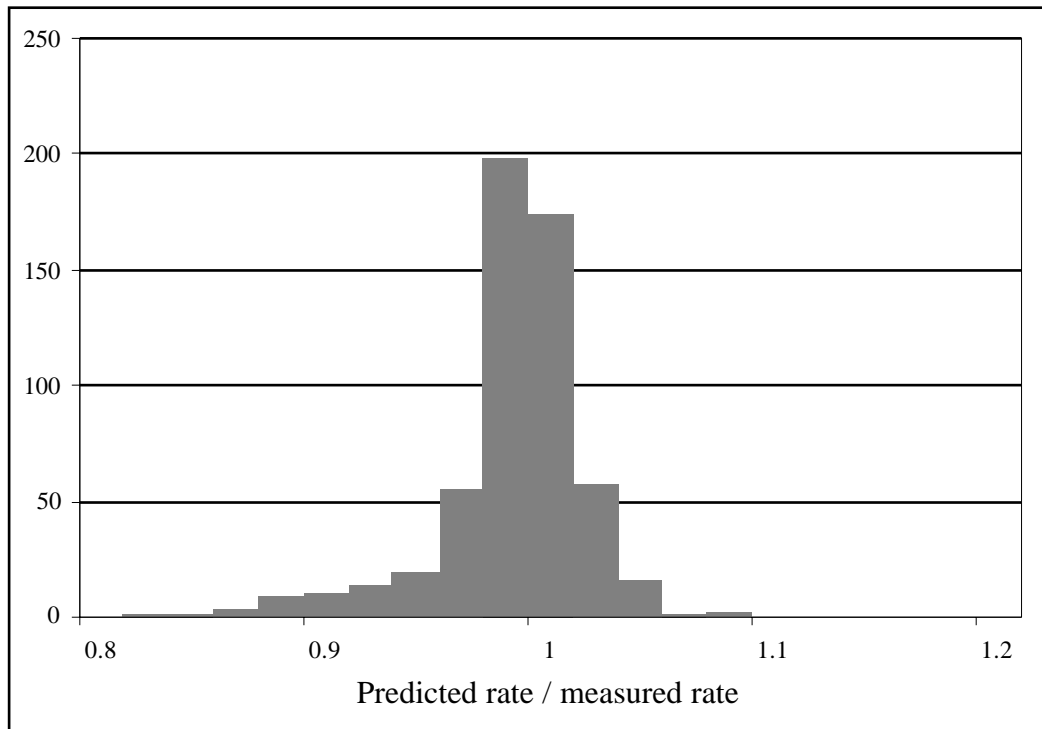


Figure 19. The ratio of predicted and measured maximum event rate for the “1”, “1+1”, “2+1” and “2+2” configurations

8 Conclusions and discussion

The measurement results show good performance of the hardware and software, satisfying the requirements obtained from modelling [5,6]. The measurement procedure may be limiting the maximum rates, in particular for the ROB Complex measurements. The maximum rates observed can be estimated with simple equations, based on parameters obtained from fitting the measurement results.

The point-to-point SHARC links are well suited for the internal (probably on-board) connections inside a ROB Complex. The automatic handshaking per 32-bits word transferred across a link is very useful and makes it straightforward to avoid overflows of buffers for requests and decisions inside the ROB Complex. As the ROB-MUX takes care of fragment building and fanning out RoI requests and decisions the message rate across the PCI bus is smaller than for ROB Complexes using the PCI bus for connecting ROBIns to a CPU of a PC or Single Board Computer. Also the load of the CPU of the PC or Single Board Computer may be smaller.

The compact design and the low power dissipation of the CRUSH would allow to build a ROB complex with about 6 ROBIns on a single 6U Eurocard, if a good way is found to connect 6 S-links to a 6U board.

The ADSP-21160 processor is the successor of the ADSP-21060. It is 2.5 times as fast, has 6 100 MByte/s links, also 512 kByte of internal memory, a smaller footprint than the ADSP-21060 and also a low power dissipation. A new design based on it (possibly in combination with a new generation of FPGAs) would not suffer from the bandwidth limitation of 80 MByte/s in the ROB-MUX, whereas the performance of this processor could be high enough to allow servicing of two S-links. An even more compact design could be possible at a lower price due to a reduction in the number of processors needed.

The S-link output of the ShaSLINK could be used as an alternative to output via PCI bus. This allows to study a scenario in which groups of ROBs are outputting event data via dedicated links (S-links in this case) connecting to processors of the LVL2 farm. These processors would then provide the required network interfacing together with processing capacity for the LVL2 trigger. RoI requests and decisions would reach the ROB complex either via PCI bus or via a SHARC link interfaced to a PC or Single Board Computer servicing many ROB Complexes. This and other scenarios are discussed in [6] and [7].

References

1. "A Compact Robin Using the SHarc (CRUSH)", September, 1998, P. P. M. Jansweijer, G. N. M. Kieft, J. C. Vermeulen, <http://www.nikhef.nl/pub/experiments/atlas/daq/CRUSH-hw.pdf>
2. Manufactured by Silicon Software, <http://www.silicon-software.de>
3. SLIDAS, "S-Link Infinite Data Source", <http://hsi.web.cern.ch/HSI/s-link/devices/slidas/>, available from INCAA computers, <http://www.incaacomputers.com>
4. ShaSLINK description, P. Jansweijer, <http://www.nikhef.nl/pub/experiments/atlas/daq/ShaSLINK.pdf>
5. "Paper modelling of the ATLAS LVL2 trigger system", J.Bystricky and J.C.Vermeulen, ATL-COM-DAQ-2000-022
6. "ROB Complex Master Working Document", R. Cranfield and J. Vermeulen, ed., ATL-COM-DAQ-2000-033
7. "Scenarios for a ROB system built with SHARC processors", Jos Vermeulen, October 1999, <http://www.nikhef.nl/pub/experiments/atlas/daq/Scenarios/Scenarios-NIKHEF.pdf>

Appendix Buffer management schemes

There are different ways for keeping track of free space in the event fragment buffer and detecting/avoiding a buffer full condition. Two schemes have been tried in the tests. No significant difference in performance was found for the two buffer management schemes.

The first scheme of buffer management of the Event Buffer and associated Event Info List (list of event summaries) is illustrated in 20.

EvtInfo_list is the 'Event Info List', an array of event fragment summaries, each holding the 8 words of event fragment information that the SHARC reads from the paged FIFO for each event fragment (plus two extra words as explained below). The list is ordered according to the lower 10 bits of the event's Event-ID (so EvtInfo_list has space for 1024 event fragment summaries). Each event fragment summary holds - among other things - the index into EvtData_buffer where the actual event data is to be found, and its length, enabling the retrieval of an event fragment based on its event identifier.

In principle there is no fixed maximum number of event fragments that can be present in EvtData_buffer at any one time; however, if at some point an event fragment arrives with an Event-ID with the 10 lower bits identical to an event fragment already present in buffers EvtData_buffer and EvtInfo_list, the information in EvtInfo_list would be overwritten and the reference to the other event fragment would be lost if this is not taken into account and appropriate action is not taken. In this case space is allocated from the heap for storing the event summary. A pointer in the summary in EvtInfo_list is set to point to the new summary. Any other summaries to be stored in the same EvtInfo_list entry list are added to the linked list thus created (as shown in 20). The current software checks for the presence of a linked list but does not yet create or scan a list (since it is assumed that this rarely occurs the presence of the check alone suffices to include the resulting impact on performance).

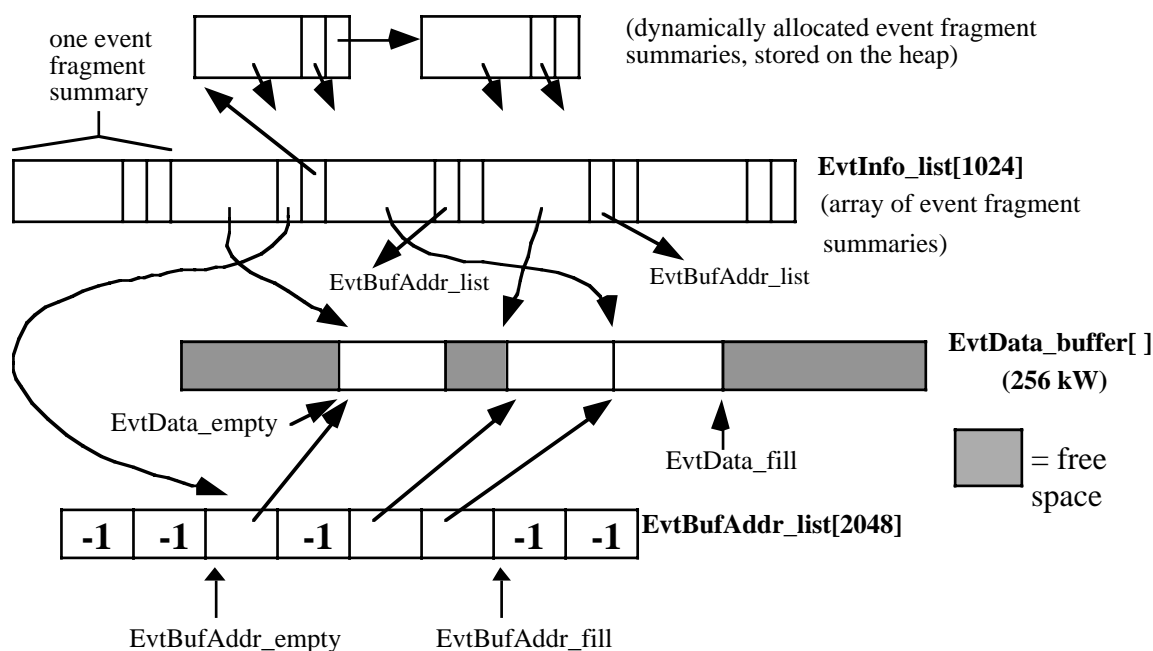


Figure 20. Event fragment buffer management scheme using a separate array for administration of free space.

Since event fragments are taken out of EvtData_buffer (“deleted”, i.e. their memory space marked as free space, or sent on to the Event Builder) in a more or less random order, it is necessary to keep track of the free space in this buffer available for new event fragments which are written sequentially into EvtData_buffer. So a “fill” and “empty” index (or pointer), called here EvtData_fill and EvtData_empty, need to be maintained.

The solution illustrated in 20 is to define an array EvtBufAddr_list, which holds for every event fragment in EvtData_buffer the index (or pointer) into this buffer, in the same order as the fragments were written into EvtData_buffer (since buffer EvtBufAddr_list can overflow as well a “fill” and “empty” index are also maintained for this array, called EvtBufAddr_fill and EvtBufAddr_empty). If the event fragment is no longer present the corresponding entry in EvtBufAddr_list, which is found through an extra entry in the info-block containing the index, is set to -1; after this the EvtBufAddr_empty index is incremented until an EvtBufAddr_list entry unequal to -1 is found or until EvtBufAddr_empty is equal to EvtBufAddr_fill (meaning the event fragment buffer is empty): this will be the new value for EvtData_empty, i.e. Evt-Data_empty is set to EvtBufAddr_list[EvtBufAddr_empty].

If space in EvtData_buffer runs low either the flow of event data from the front-end has to be stopped (by generating an XOFF on the ROL) or more contiguous free space has to be created by moving event fragments from EvtData_buffer elsewhere (i.e. to the SHARC's on-chip memory). It is possible that either EvtData_buffer runs out of space, or EvtBufAddr_list runs out of space; both cases can be handled by moving event fragments from EvtData_buffer to the heap (in the SHARC's on-chip memory) to free space in both arrays. Since occurrences of these situations should again be rather rare the current software does not implement the actual moving of event data, but the checks for buffer overflow (which have to take place for every incoming event) are done so that realistic tmeasurement results are obtained.

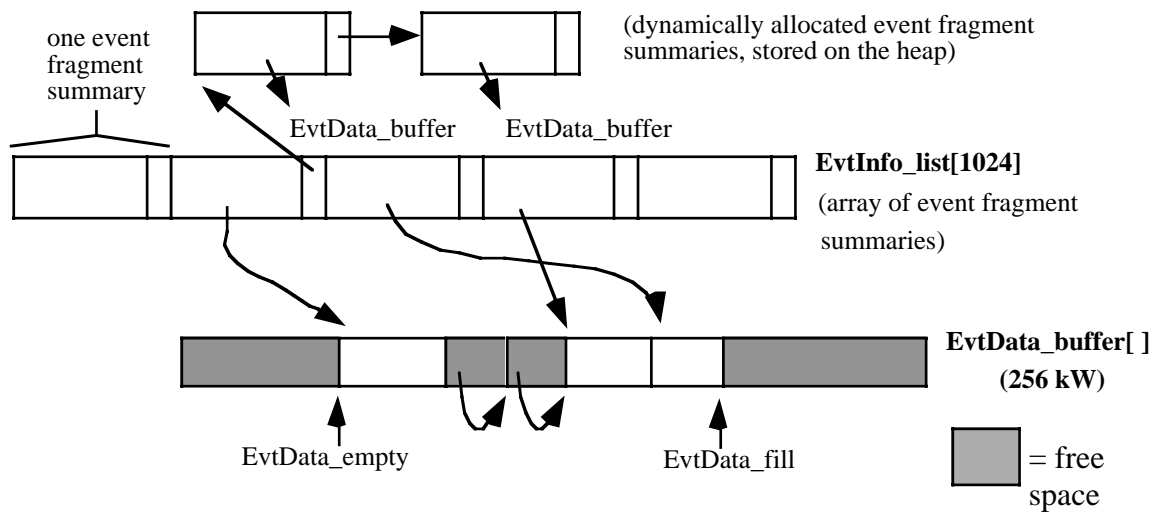


Figure 21. Event data buffer management scheme using the event data buffer for administration of free space in the buffer.

The second scheme of buffer management is illustrated in 21. This scheme has been used for the measurements discussed in this document. The SHARC in the CRUSH module is capable of reading from as well as writing to the event buffer, making it possible to do the free-space buffer management in the event buffer itself (with as consequence that the EvtBufAddr_list array and its management are no longer needed). When the buffer space of an event fragment in the event buffer is released the value -1 and the size of the event fragment is written in the first two words of the freed space. The EvtData_empty pointer should now be updated in a loop: if it points to a value -1 it should be increased with the size value following until a value not equal to -1 is encountered (or until EvtData_empty is equal to EvtData_fill, which means that the event buffer is empty).