ATLAS Internal Note

Evaluation of current S-Link Technologies for the TileCal ROD System

J. Roldan, J. Romance

Jesus.Roldan@cern.ch, Juanba.Romance@ific.uv.es
IFIC. Instituto de Física Corpuscular
Centro Mixto, Universidad de Valencia - CSIC
Avda. Dr. Moliner 50, E-46100 Burjassot
VALENCIA, Spain

Contents

List of Figures

1 Introduction

Two S-links cards have been tested at Valencia ROD Lab[1], using different S-link hardware devices and hosts. In the source side, a PC PentiumII 266Mhz running Linux kernel 2.2.x, a PCI to S-link board is plugged. At the receiver side, a Motorola 2604 VME card under LynxOS 3.0 houses a S-link to PMC card.

The main goal is to perform a set of tests **to picture bandwidth figure of the current commercial S-link hardware devices[2]**. In the S-link community, some public software packages can be found under different platforms. However, a custom ROD S-link set of libraries and drivers have been implemented in both sides due to the specific requirements needed[3]. S-link PCI devices are built on top of the AMCC S5933 PCI chipset[4] services. The physical S-link control/status lines are mapped through incoming/outcoming mailboxes. Further information related with the AMCC hardware description can be found at [5]. Some common issues have driven the S-link software design:

- **The latency of transfers MUST be as low as possible.** *System Calls* are only allowed when the S-link devices need a *set-up*. *Polling* is used to monitor the DMA read/write cycles. As backup of last thought, the drivers released map also the S5933 chip if a privilege intervention is required.
- **The PCI S5933 chipset is user space mapped.** Current S-link software packages shows that a user space S-link library can provide the same functionalities that a driver based library version.
- The *data flow* between S5933 internal FIFOs and the system RAM MUST be performed using the **S5933 DMA capabilities**. CPU is only used to move data from/to FIFOs when the maximum bandwidth performance is NOT required.
- **The RAM buffers for S-link transactions MUST will also be user space mapped.** Therefore, no Kernel to User/User to Kernel transactions are required to access the DMA data buffers.
- The ROD management of physical devices as RAM memory and PCI chipsets MUST **live in peace** with the kernel management of non-related ROD devices
- From the point of view of S-link programmers and S-link users a similar set of functions and data structures have been implemented/used in both systems (Motorola/Lynx, PC/Linux).

2 LynxOS Software. MVME2604

Current LynxOS 3.0 distribution does not provide a PCI driver or user library package where a PCI application can be built. It is up to user, to build the necessary software to make a PCI application.

2.1 Bones

A generic PCI driver `pcidrv.c` has been implemented providing minimum PCI services. At *start-up*, the PCI driver scans the current LynxOS kernel PCI device attachments over physical PCI bus. It will manage an internal database following the next rule: if the device is already attached (it means that another driver is already linked to the device), only READ actions over the PCI Configuration Space will be allowed. Otherwise, the hardware is up to be managed by the driver completely. From the user point of view, it means that it can be mapped on the PCI (I/O, Memory spaces) and Kernel spaces.

The kernel-user I/O is performed using typical driver calls: ioctls, open, close. Read-write calls operates over the complete PCI configuration device space ($256 bytes$), which can not be mapped. The user procedure must follow a *driver attachment* of the particular device. It will performed some read, write and ioctl calls to configure the physical chipset and will alloc the device size in the PCI driver. Alloc ioctl entry returns the physical address of the PCI chipset. This parameter will be used to map the chipset on the user context, using the typical LynxOS call smem-create.

The RAM virtual device for DMA transfers has been managed using a RIO2 driver named uiodrvr provided by CES. The same strategy has been followed on the uiolib user library provided the DAQ-1 project. S-link libraries have been written on top of this two drivers. Both of them will provide some issues pointed at beginning. A complete set of routines performs the typical functions over the *user software object* as can be open, close, reset or DMA reads. A software quality checking has been carried out building a user application. *SLIDAStest* estimates the performance of S-link to PMC card, to move data between the SLIDAS and the physical memory. SLIDAS emulates a data source generator of $32bits$ at a maximum rate of $40MHz$. It means that even $160\frac{Mbytes}{sec}$ can be loaded on our S-link PMC. The figure 1 shows the released package distribution.

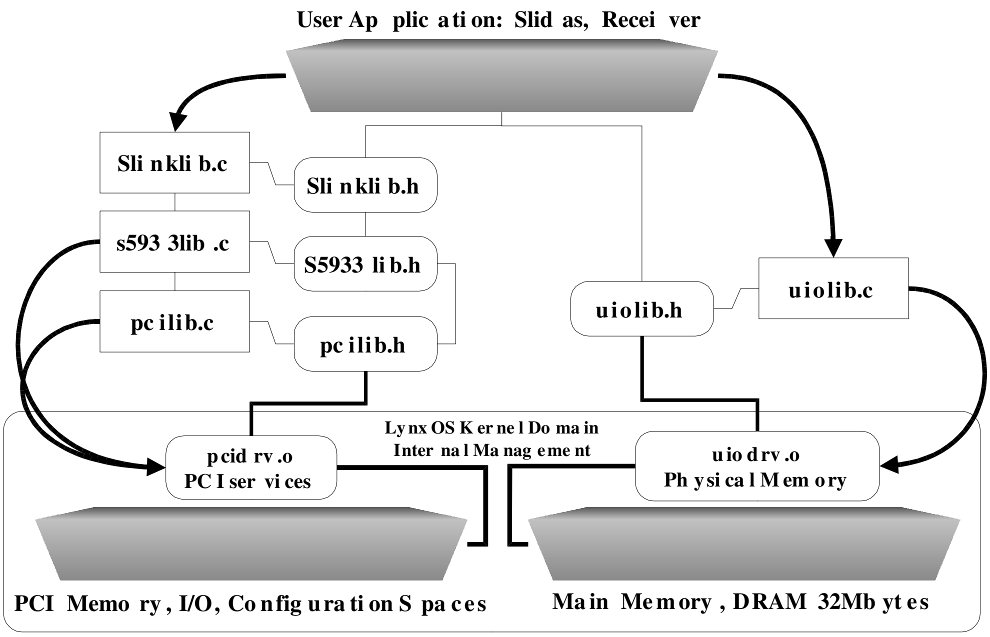

Figure 1: *S-link LynxOS object distribution.*

The current S-link to PMC hardware design, forces a continuous monitoring of the kind of S-link word present at *AddOn FIFO* (S-link FIFO), to get a continuous DMA flow. A mistmach between the expected link word on *OMB1* and the current link word reported at *IMB4* will resume the DMA.

Inside a DMA transfer, the link word is *data* type and therefore the expected word must also be *data* type, to get a continuous data flow from the AMCC S5933 FIFOs to the RAM Memory. When a *ctrl* word arrives to the external FIFO (header or trailer of current packet), DMA will resume waiting that some external mechanism (software and therefore the CPU) change the expected word. After update the values, the DMA transaction will flow again for the new packet. Our tests will try to measure this intrinsic effect of current S-link hardware on the bandwidth performance.

Below code, extracted from the DMA routine on *slinklib.c*, illustrates this event:

```
 while( dev->amcc->mwtc ){ /* -------- PMC Data Flow Monitoring ------- */
   if( mcsr->a2pfe == EMPTY ){
     if( ( imb->lderr != omb->expl_derr )||( imb->lctrl != omb->expl_ctrl ) ){
       /* Between above two "if-then" can arrive a good word */
       kk = SLINK_DMA_TIME_OUT;
       while( mcsr->a2pfe == NOT_EMPTY ){
         kk-- ;
#ifdef SPY
         if( !( kk % 1000 ) )
           printf("\n[ %sFIFO Not Empty%s ]", bd, atoff);
#endif
       }
       /* DMA Fault Report @ PMC */
       fault_index = dev->amcc->mwtc;
       fault_type  = imb->lderr;
       word_type   = imb->lctrl;

       /* Now Activate Expected Lines to let DMA flow again */
       omb->expl_derr = imb->lderr;
       omb->expl_ctrl = imb->lctrl;

       /* In parallel we can process the EMPTY event @ PMC FIFO
          using the above DMA Fault Report.
          Be Careful !!!!!!! DMA could STOP before error
          Data Processing could be done */
       fault_index = ( pci_swap32( fault_index ) / 4 );
```

2.2 Results And Conclusions

Following last thought, we have done two kinds of test. **Raw Transfers** where SLIDAS supplies raw packets whitout CONTROL words and **Control Transfers** where the packet is a normal S-link packet Header, data, data ......, data, Trailer. On both cases, the buffer size has been sweeped covering $13Kwords$. In the *ctrl sweep* the curve sampling and sweep size is constrained by the SLIDAS capabilities. However, the *raw sweep* has been software oversampled checking the *ctrl* measurement. The data plots are shown at picture 2.

- RAW DMA Transfers (DMA buffering without CTRL words) can be as high as $89\frac{Mbytes}{sec}$ with our current system, with an efficiency of 95% for $2Kwords$ buffer size. The figure shows that the buffer size required to obtain a good efficiency is not too much exigent ($\frac{1}{2}Kword$ means a 85%)

- CTRL DMA Transfers (DMA buffering with CTRL words) can reach up to $85\frac{Mbytes}{sec}$ for $8Kwords$ (the maximum Packet size provided by SLIDAS). In terms of bandwidth,

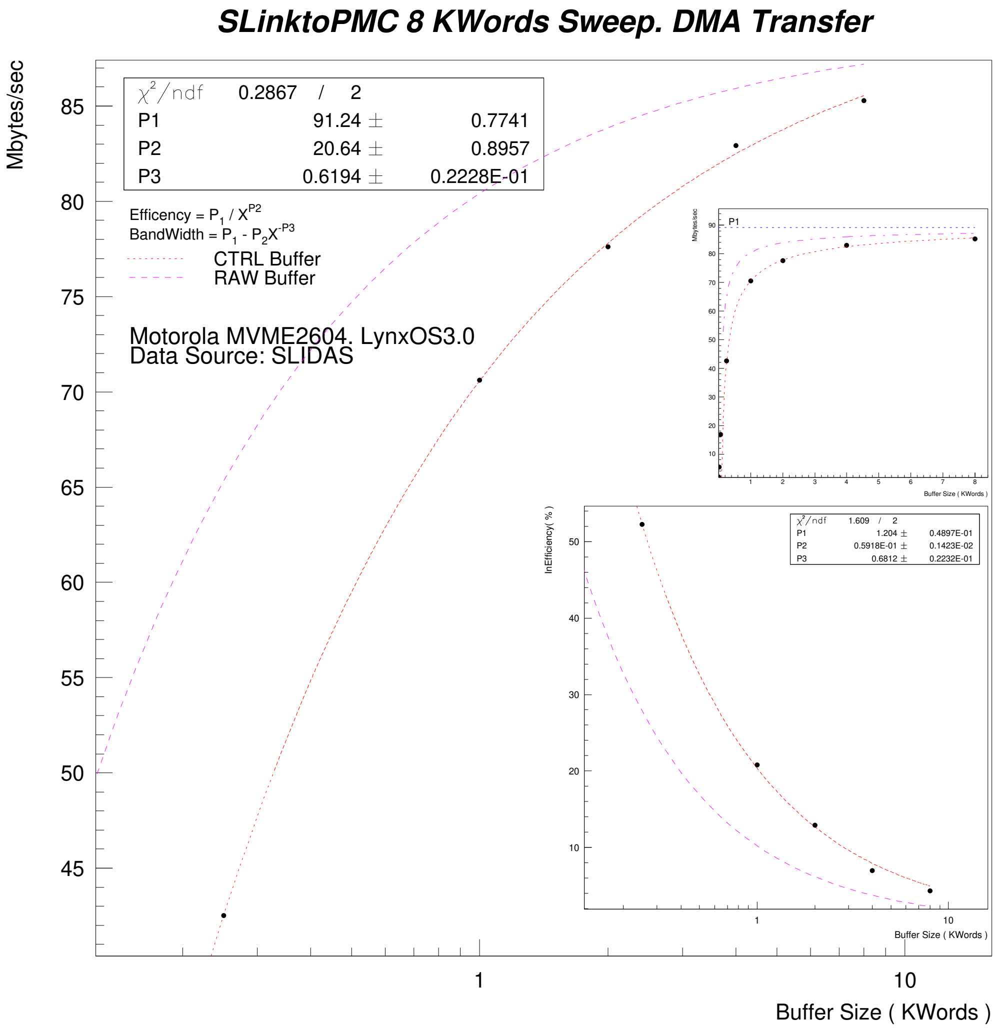

Figure 2: *PMC Bandwidth using a SLIDAS($160Mbytes/sec$) as data generator.*

the penalty coming from the software CTRL words switch, which is nicely seen in the curves, it is roughly about 10% for $1Kword$

- Both experiments trends to the same plateau (P1 parameter) for a buffer size $8Kwords$ inside a a variance lower than 4%
- A good $\frac{\chi^2}{ndf}$ has been obtained from the data collected which means a good characterization

2.3 Observations

- The calculations use software calls to measure elapsed time after M iterations for each packet size. We do not have in mind overheads coming from intermediate function calls, the procedure followed seems something like below code:

```
for( jj = 0 ; jj < STATISTICs ; jj ++ ){
   run_params.start=times( &timer );
```

```
  for( ii = 0 ; ii < loops ; ii++ ){
    if( S-link_DMA_Read( dev, ( char *)(&dma), size, PCItoPHY_M )
        != SUCCESS )
      break;
  }
  run_params.end=times( &timer );
}
```

- The S5933 chipset is designed for data widths of $32bits$, working at a maximum speed of $33MHz$. It means that the MVME2604 PCI bus is underloaded, which can extract/push data widths of $64bits$ at a full speed of $66MHz$
- Our tests are in agreement with the study performed by the ESSOS group[6]. This paper shows that a bandwidth of $88.1\frac{Mbytes}{sec}$ is obtained for the MVME2604, writing PCI bursts from the S5933 to DRAM Main memory. This measurement has been performed with a VMEmetro PCI analyzer

3 Linux Software. PentiumII

Our PC running Linux 2.2.x allocates two PCI cards that will be used in the ROD environment. A PCI to S-link board which will supply data over the FiberChannel Link to Motorola MVME 2604. The Bit3 interface bridges PCI to VME, to make slow rate tasks as monitor, control or data sampling over the VMEbus.

3.1 Bones

The ROD driver is designed having in mind the issues pointed at header page. The *Linux memory management* is the key word to allow a user space library design. Linux kernel does not provide the user function smem-create found on LynxOS, following the general devices policy used on Linux, it should not have too much sense.

Linux kernel and modules are designed to hide the physical device management, as could be the PCI space or the physical memory, to user. User applications work under the virtual memory kernel management, which properly manage the memory page faults on the devices. A nice description of memory management under Linux is found at [7].

Our Linux module, performs PCI and Memory mapping services to user using ioctls, open, close system calls. It is also implemented POSIX mmap call, which returns a virtual pointer to the memory associated to the device selected. The RAM Memory device is limited to $32pages(4096\frac{bytes}{page})$, this is the maximum number of contiguous pages granted by get_free_pages[8].

If more memory per object is needed, one can patch the kernel with bigphysarea[9] which enhances get-free-pages services. Afterwards, an update of alloc-free ioctl entries on ROD driver should be done.

The mmap driver entry makes the PCI chipset/RAM remapping job, using remap_page_range[10] kernel symbol. The trick to remove *page swapping kernel management* on system RAM, comes from the X server strategy to access physical memory. Another nice description can be found at the Linux Lab Project documentation[11]. Our modified version blocks the kernel array mem-map entries linked to the page range allocated by the get_free_pages call.

```
/* Trick from the Linux Lab Project to use remap_page_range in mmap call
   when RAM is involved. PG_Reserved bit. Same strategy as Xfree */
   for( ii = MAP_NR( tmp->k_ptr );
    ii <= MAP_NR( tmp->k_ptr + ( ( PAGE_SIZE - 1 )*alloc.pages ) );
       ii++ ){
         mem_map_reserve( ii );
   }
```

After this loop the pages are treated as PCI memory pages, and therefore are suitable to be user mapped using *mmap, remap-page-range and VMAs*.

Driver entries *write-read* are implemented to access the PCI Configuration Space which can not be mapped. Same as the LynxOS PCI driver, the Linux driver maps internally the PCI chipsets to be able to access hardware whenever required, i.e. in the /proc/rod entry or if some interrupt service is designed.

As S-link LynxOS libraries case, a similar set of data types and procedures have been implemented on top of the ROD driver services. We have check the package functionalities, **setting up a real S-link environment based on Fiberchannel**. LDC and LSC S-link/FiberChannel cards provide a maximum BW of $103\frac{Mbytes}{sec}$ over the maximum theoretical rate of $1\frac{Gbit}{sec}$ provided by the FiberChannel specification, further information about hardware is found at [12]. The figure 3 shows the object distribution.

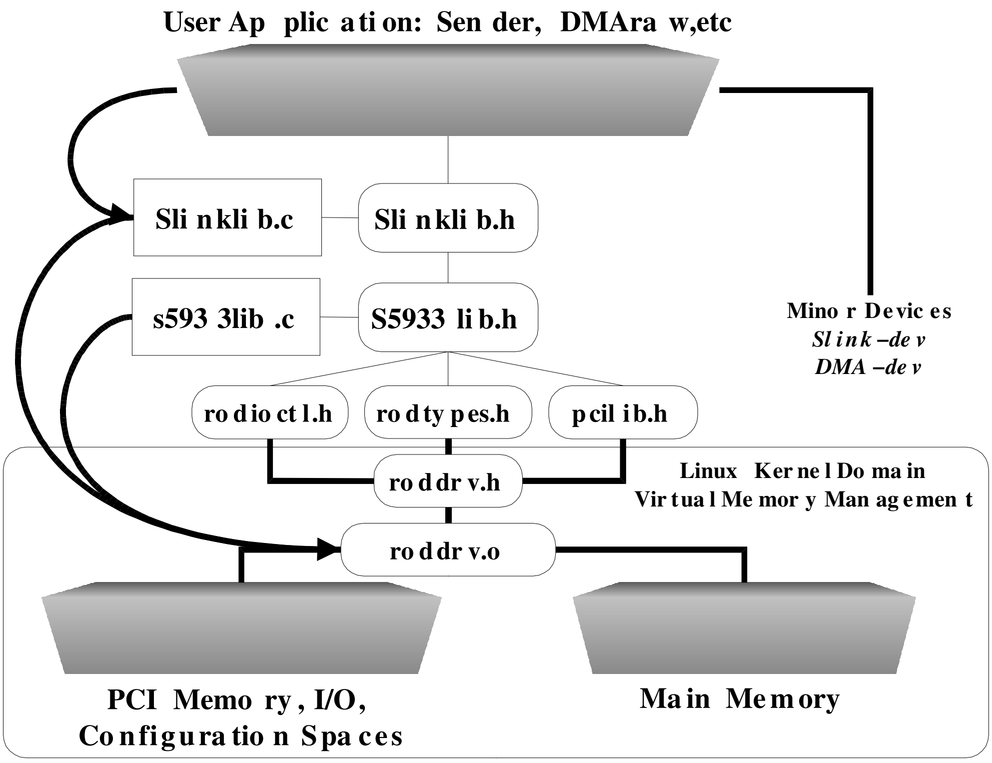

Figure 3: *S-link Linux object distribution for the TileROD Software.*

3.2 Application Code

The applications built will try to measure the PC RAM to FIFO transfers bandwidth and the Fiberchannel bandwidth. We are also interested to measure the incoming penalty from the CTRL words management. In the Linux side, the control word (S-link packet header or trailer) needs to be written by software, therefore, no DMA can be used.

A synchronization protocol between the sender and receiver software is necessary before the sweep test starts. We will use the main data link and the S-link *Return Lines* to configure the two processes. Before the sweep test starts, a special packet will be sent from the PC Memory to the PowerPC memory. It will contains the sweep parameters: the data buffer size to be transmitted, the number of S-link packets, the number of loops per sweep point and the sweep resolution. When the receiver is ready to perform the sweep, it will acknowledge its state lowering S-Link line LRL3, which will be monitored at sender side after send the synchro packet. The experimental results obtained are shown at figure 4.

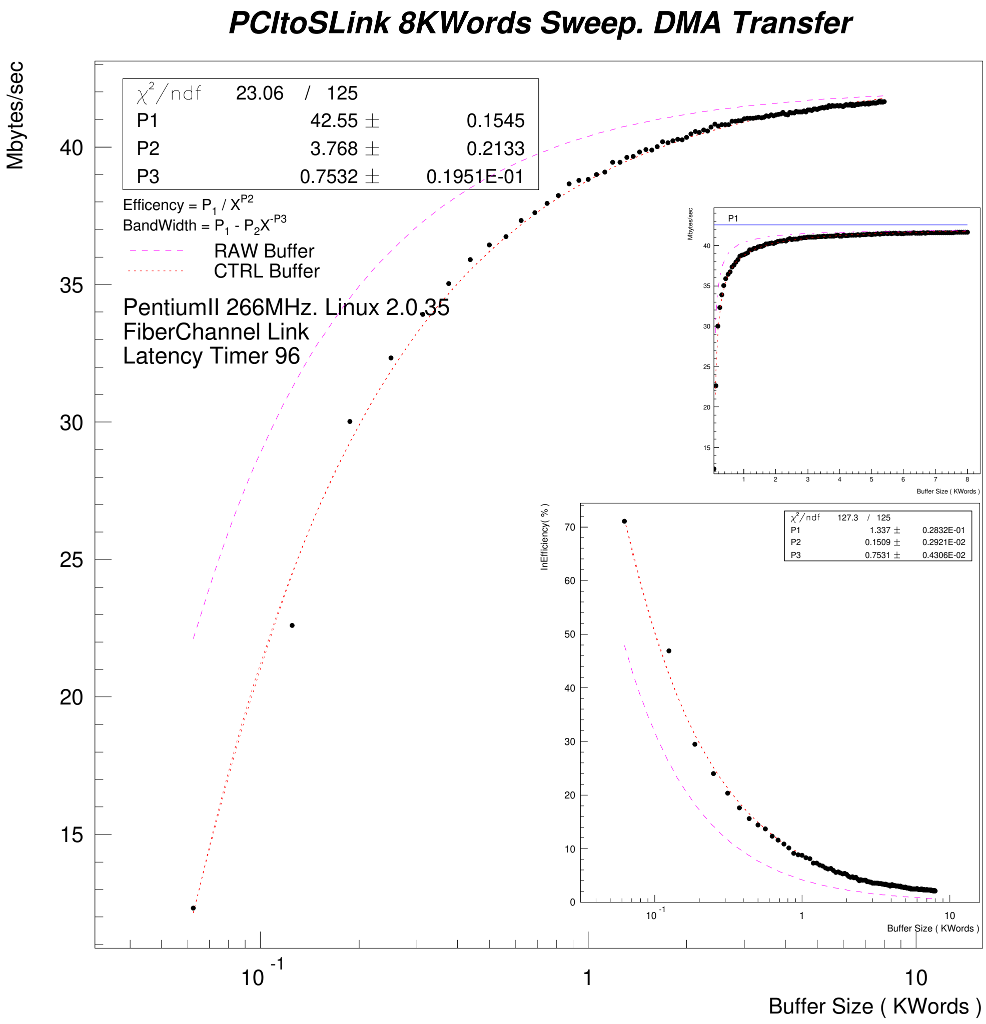

Figure 4: *Bandwidth figure using a real S-Link environment on Fiberchannel.*

3.3 Results And Conclusions

- RAW DMA Transfers (DMA buffering without CTRL words) can be as high as $42\frac{Mbytes}{sec}$ with an efficiency greater than 98% for $8Kwords$ buffer size. The figure plot shows that the size-efficiency is not too much exigent, $\frac{1}{2}Kword$ means a 90%
- CTRL DMA Transfers (DMA buffering with CTRL words) can reach up to $42,5\frac{Mbytes}{sec}$ for $8Kwords$. In terms of Bandwidth, the penalty coming from the software CTRL words management, it's roughly a 10% for $\frac{1}{2}Kword$
- Both experiments trends to the same plateau (P1 parameter), inside a 5% margin, for a buffer size $\geq 1Kwords$
- Same as SLIDAS case, a good Chi square-nfd ratio is obtained from the data collected

3.4 Observations

- The theoretical BW on the PCI to S-link card is $65\frac{Mbytes}{sec}$. Therefore, is not possible to check the FiberChannel bandwidth. Erik V der Bij has suggested us that such test should be done using a SliTest board[13] or a MicroEnable card[14] which provides a RAM Memory to/from PCI throughput of $110\frac{Mbytes}{sec}$
- The Linux load has been set to default. XServer and other *user applications* as networking services are running and our S-link application has not any special privileges. No special *kernel scheduler set-up* has been done
- RAW and CTRL DMA tests are performed for a PCI-S-link Latency timer value of 96 ($23\mu sec$ per burst). For a fixed data buffer ($8Kwords$), a latency sweep has been done covering the whole dynamic range of latency timer. We see a flat response of $35\frac{Mbytes}{sec}$ with a step response to P1 parameter from 26 latency value to the end of the sweep.
- Tests performed at Nikhef about S-link, shows that our P1 parameter is very close to the Nikhef plateau using either the Windows NT Driver or the Memory mapped LSC Linux driver provided by Steffen Luitz[16]. A PCI trace of the DMA cycles, is gathered from a PCI VMetro card analyzer. We see that an intrinsic effect is produced by the PC motherboard PCI/RAM bridge. It limits the burst cycles length to short periods on the PCI bus and therefore the bandwidth measured.

4 Summary

We have built a custom software toolkits for the LinuxOS (PC box) and LynxOS (VME PowerPC workstation) which provide a low latency environment for the ROD prototyping and Test-Beam environments using the current available commercial hardware. For ATLAS, the current *Fiberchannel/S-Link* implementation provide us enough bandwidth to download the superdrawer front-end event following LVL1A rate (worst case $632\frac{Mbits/sec}{sdrawer}$). However, costs and radiation tolerant studies should be performed to complete the overview, therefore no final decision has been taken yet for the ATLAS case.

References

[1] TileCal ROD pages. *http://ific.uv.es/tical/rod*

[2] S-link studies for the TileCal RODs. *http://ific.uv.es/tical/rod/slink*

[3] TileCal ROD, Software Reference. *http://ific.uv.es/tical/rod/CodeProduction.html*

[4] AMCC s5933 chipset. *http://www.amcc.com/Products/PCI/S5933.htm*

[5] S-link at CERN. *http://www.cern.ch/HSI/s-link*

[6] Second Generation VMEBus POWERPCs based SBCs running LynxOS. *http://www.cern.ch/ESS/OS*

[7] Chapters Seven and Thirteen of Linux Device Drivers. A.Rubini. *http://www.oreilly.com/catalog/linuxdrive*

[8] Memory Page Allocation under Linux. *http://hegel.ittc.ukans.edu/topics/linux/man-pages/man9/kmalloc.9.html*

[9] BigPhysical Area Linux Patch. *http://www.uni-paderborn.de/cs/heiss/linux/bigphysarea.html*

[10] PCI spaces *mmaping* under Linux. *http://khg.redhat.com/HyperNews/get/devices/addrxlate.html*

[11] The Linux Lab Project*http://www.llp.fu-berlin.de*

[12] Fiberchannel-Slink interfaces. *http://www.rmki.kfki.hu/detector/S-Link*

[13] S-link VME suite. *http://www.cern.ch/HSI/s-link/devices/slitest/*

[14] Microenable. *http://www.silicon-software.com*

[15] S-link studies at Nikhef. *http://www.nikhef.nl/pub/experiments/atlas/daq/S-LINK-NT-driver.html*

[16] NA48 Experiment, S-link as front-end link protocol. *http://www.cern.ch/HSI/s-link/projects/na48/na48pc/dt2sl.html*