

Juha Arvio

PCIe to HIBI v2

DATASHEET

Table of Contents

[1. Identification 5](#_Toc304384626)

[2. Version history 6](#_Toc304384627)

[2.1 Documentation 6](#_Toc304384628)

[2.2 IP block 6](#_Toc304384629)

[3. General description 7](#_Toc304384630)

[3.1 Supported functions 7](#_Toc304384631)

[4. Detailed description 8](#_Toc304384632)

[4.1 Supported features 8](#_Toc304384633)

[4.2 Interface 10](#_Toc304384634)

[4.2.1 Ports and signals 10](#_Toc304384635)

[4.2.2 VHDL generics 12](#_Toc304384636)

[4.2.3 Register map – HIBI side 14](#_Toc304384637)

[4.3 Block description 15](#_Toc304384638)

[4.4 Address translation 15](#_Toc304384639)

[4.5 Functionality 17](#_Toc304384640)

[5. Use example 18](#_Toc304384641)

[5.1 Test setup 18](#_Toc304384642)

[5.2 Address translation: PC => HIBI 19](#_Toc304384643)

[5.3 Address translation: HIBI => PC 19](#_Toc304384644)

[5.4 Use cases 20](#_Toc304384645)

[6. Performance and resource usage 24](#_Toc304384646)

[7. Dependencies 25](#_Toc304384647)

[8. Verification 26](#_Toc304384648)

[8.1 Simulation 26](#_Toc304384649)

[8.1.1 Single blocks 26](#_Toc304384650)

[8.1.2 DUT 26](#_Toc304384651)

[8.2 Physical testing 26](#_Toc304384652)

[8.2.1 Test phases 27](#_Toc304384653)

[9. Documentation for developers 28](#_Toc304384654)

[9.1 Requirements 28](#_Toc304384655)

[9.2 Overview of the PCI bus 29](#_Toc304384656)

[9.3 PCIe transaction types 29](#_Toc304384657)

[9.4 Internal structures 34](#_Toc304384658)

[9.4.1 TLP dec 34](#_Toc304384659)

[9.4.2 TLP enc 34](#_Toc304384660)

[9.4.3 Address trans. 35](#_Toc304384661)

List of Abbreviations

fifo first in, first out

FPGA Field Programmable Gate Array

Gbps Giga bits per second

H2P HIBI to PC

HDL Hardware Design Language

HIBI Heterogeneous IP Block Interconnect

HW Hardware

ID Identification

I/O Input / Output

IP Intellectual Property

IRQ Interrupt Request

P2H PC to HIBI

PC Personal Computer

SoC System-on-Chip

SW Software

tbd to be decided

TLP Transaction Layer Protocol

TUT Tampere University of Technology

VHDL VHSIC Hardware Description Language

VHSIC Very High Speed Integrated Circuit

# Identification

Vendor: TTY

Library: communication

Name: PCIe\_to\_HIBI

Version: 0.9

Author(s): Juha Arvio

Date created: 22.09.2010

Date last modified: 22.09.2011

# Version history

## Documentation

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Version | Change description | Author |
| 22.09.2010 | 0.1 | Component documentation started | Juha Arvio |
| 08.12.2010 | 0.3 | - | Juha Arvio |
| 21.09.2011 | 0.4 | Converted documentation into a new format | Juha Arvio |
| 28.09.2011 | 0.5 | Added information | Juha Arvio |

## IP block

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Version | Change description | Author |
| 22.11.2010 | 0.1 | Component started | Juha Arvio |
| 21.09.2011 | 0.9 | Updated component | Juha Arvio |

# IP-block files

## Documentation

* doc\
  + PCIe\_to\_HIBI\_v2.docx
    - ip-block datasheet (this file)
* doc\fig\
  + PCIe\_to\_HIBI\_v2.pptx
    - Powerpoint figures used in the datasheet
  + PCIe\_to\_HIBI.mdzip, PCIe\_to\_HIBI.mdr
    - Magicdraw figures used in the datasheet
* doc\misc\
  + PCI.Express.Base.Specification.v2.0.pdf
    - specification for PCIe v2.0

## HDL files

* hdl\
  + pcie\_to\_hibi.vhd
  + pcie\_rx.vhd
  + pcie\_tx.chd
  + pcie\_dma.vhd
  + hibi\_if.vhd
  + pkt\_buf.vhd
  + buf\_ptr\_ctrl.vhd
  + req\_ctrl.vhd

## Software drivers

* drv\
  + altpciechdma.c
  + Makefile
  + Kconfig

## HDL testbench

* tb\
  + pcie\_to\_hibi\_test\_app.vhd
  + a2\_pex\_x8\_app\_if.v
  + hibiv3\_seg\_r3.vhd
  + set\_tb.do
  + tb\_wave.do

## Test systems

* ts\
  + a2gx\_pcie\_to\_hibi\

# Dependencies

Required components:

PCI-Express controller created with the Altera PCI Express Compiler using a hard IP core (required for x8 lanes), HIBI wrapper R3 (main component), HIBI wrapper R4 (tester component)

# General description

PCI-Express (PCIe) to Heterogeneous IP Block Interconnect (HIBI) is an adapter component which with the help of Altera’s PCIe controller interfaces a PCIe interconnect to a HIBI bus.



Figure - PCIe to HIBI adapter interfacing PCIe phy to HIBI phy

## Supported functions

* Transfers initiated from PC:
  + PC 🡪 HIBI (direct single word write)
  + PC 🡪 HIBI (direct single word read request)
  + PC 🡪 HIBI (dma write)
  + PC 🡪 HIBI (dma read)
* Transfers initiated from HIBI:
  + HIBI 🡪 PC (direct multi word write)
  + HIBI 🡪 PC (direct multi word read request)
  + HIBI 🡪 PC (interrupt)
* Transfers initiated from another PCIe component\*:
  + PCIe comp 🡪 HIBI (direct multi word write)
  + PCIe comp 🡪 HIBI (direct multi word read request)
* Completions initiated from PC/another PCIe component:
  + PC 🡪 HIBI (multi word read completion)

\* DMA transfers are supported by other PCIe components but they are redundant

Data flow for a single read request followed by a completion:



Figure - Read request followed by a completion

# Detailed description

PCI-Express to HIBI is an adapter component written in VHDL which interfaces a PCIe interconnect to a HIBI bus. Initially only Altera’s PCIe controllers generated with the Megawizard with an Avalon streaming (ST) interface are supported. The adapter can be roughly divided into four sub-blocks. These blocks include the transaction layer protocol (TLP) decoder and encoder connecting with the Avalon ST and interrupt request (IRQ) interface as well as the HIBI interface block which connects the component to the HIBI bus. There is also a configuration block with a local management (LM) interface which configures the Altera PCIe controller at system startup.



Figure ‑ PCIe to HIBI adapter interfacing an Atom processor to a HIBI bus trough PCIe

## Supported features

PCIe controller:

* Altera PCIe controllers (hard IP only)
* Number of physical lanes: 8
* Native endpoint (legacy endpoint and root port not supported)
* PCIe version 1.1 (2.0 might be possible)
* 128-bit Avalon ST interface
* Message based interrupts over PCIe (tbd: legacy, MSI or MSI-X)

HIBI:

* 32-bit HIBI with a R3 wrapper

Address translation:

* PC 🡪 HIBI
* No translation needed from HIBI to Atom

Key restrictions:

* Avalon ST interface has to be 128-bit wide 🡪 x8 hard IP PCIe controller only
* 32-bit HIBI
* Address spaces which are translated from PC to HIBI have to be in an acceptable range for fpga routing (max 32?)

The first two restrictions will be relaxed after the first working version of the adapter

## Interface

The interface information described here is for a PCIe to HIBI component which connects a PCIe interconnect with a x8 port to a 32-bit HIBI bus. Signal widths and other parameters will vary with other setups.



Figure ‑ PCIe to HIBI interface signals

### Ports and signals

The interface signals can be divided into the PCIe controller side and to the HIBI side. The PCIe controller side has four different signal groups which include the Avalon ST RX, Avalon ST TX, PCIe IRQ and the LM interface. The HIBI side has also four signal groups which include the HIBI RX, HIBI TX, HIBI message RX and HIBI message TX.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Signal group | Signal | Width [bits] | Dir. | Meaning |
| Reset | rst\_n | 1 | i | Active low reset |
| Clocks | clk | 1 | i | Clock, active on rising edge. |
| HIBI RX | hibi\_addr\_in | 32 | i | HIBI read address |
| hibi\_data\_in | 32 | i | HIBI read data |
| hibi\_comm\_in | 3 | i | HIBI read comm type (write, read req…) |
| hibi\_empty\_in | 1 | i | HIBI read empty |
| hibi\_re\_out | 1 | o | HIBI read enable |
| HIBI TX | hibi\_addr\_out | 32 | o | HIBI write address |
| hibi\_data\_out | 32 | o | HIBI write data |
| hibi\_comm\_out | 3 | o | HIBI write comm type |
| hibi\_full\_in | 1 | i | HIBI write full |
| hibi\_we\_out | 1 | o | HIBI write out |
| HIBI message RX | hibi\_msg\_addr\_in | 32 | i | HIBI message read address |
| hibi\_msg\_data\_in | 32 | i | HIBI message read data |
| hibi\_msg\_comm\_in | 3 | i | HIBI message read comm type |
| hibi\_msg\_empty\_in | 1 | i | HIBI message read empty |
| hibi\_msg\_re\_out | 1 | o | HIBI message read enabe |
| HIBI message TX | hibi\_msg\_addr\_out | 32 | o | HIBI message write address |
| hibi\_msg\_data\_out | 32 | o | HIBI message write data |
| hibi\_msg\_comm\_out | 3 | o | HIBI message write comm type |
| hibi\_msg\_full\_in | 1 | i | HIBI message write full |
| hibi\_msg\_we\_out | 1 | o | HIBI message write out |
| Avalon ST RX | Rx\_St\_Data\_i | 128 | i | Avalon ST receive data |
| Rx\_St\_Valid\_i | 1 | i | Avalon ST receive data valid |
| Rx\_St\_Sop\_i | 1 | i | Avalon ST receive start of packet |
| Rx\_St\_Eop\_i | 1 | i | Avalon ST receive end of packet |
| Rx\_St\_Bardec\_i | 8 | i | Avalon ST receive bar decoded |
| Rx\_St\_Be\_i | 16 | i | Avalon ST receive byte enable |
| Rx\_St\_Ready\_o | 1 | o | Avalon ST receive ready |
| Rx\_St\_Mask\_o | 1 | o | Avalon ST receive mask |
| Avalon ST TX | Tx\_St\_Sop\_o | 1 | o | Avalon ST transmit start of packet |
| Tx\_St\_Eop\_o | 1 | o | Avalon ST transmit end of packet |
| Tx\_St\_Valid\_o | 1 | o | Avalon ST transmit data valid |
| Tx\_St\_Data\_o | 128 | o | Avalon ST transmit data |
| Tx\_St\_Ready\_i | 1 | i | Avalon ST transmit ready |
| TxCred\_i | 36 | i | Avalon ST transmit credit |
| PCIe IRQ | app\_msi\_req | 1 | o | Interrupt request |
| app\_msi\_ack | 1 | i | Interrupt acknowledge |
| app\_msi\_tc | 3 | o |  |
| app\_msi\_num | 5 | o | Interrupt number |
| pex\_msi\_num | 5 | o |  |
| app\_int\_sts | 1 | o |  |
| app\_int\_ack | 1 | i |  |
| LMI | lmi\_data\_in | 32 | i | LMI read data |
| lmi\_re\_out | 1 | o | LMI read enable |
| lmi\_we\_out | 1 | o | LMI write enable |
| lmi\_ack\_in | 1 | i | LMI acknowledge |
| lmi\_addr\_out | 12 | o | LMI address |
| lmi\_data\_out | 32 | o | LMI write data |

Table 4- PCIe to HIBI ports

### VHDL generics

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Category | Generic | Type | Value range | Description |
|  | PCIE\_LANES | integer | 1,4,8 | PCIe lanes (only 8 supported currently) |
|  | HIBI\_DATA\_WIDTH | integer | 32, 64 | HIBI data and address width (only 32 supported currently) |
|  | HIBI\_DMA\_CHANS\_ADDR\_SPACE\_WIDTH | integer | 1…8 | System wide generic |
| PCIe to HIBI addr translation | PCIE\_BARS | integer | 1…6 | These have to be the same as in Megawizard |
| PCIE\_BAR\_0\_VAL | integer | 0x00000000…0xFFFFFFFF | These have to be the same as in Megawizard |
| PCIE\_BAR\_0\_WIDTH | integer | 1…31 | These have to be the same as in Megawizard |
| … |  |  |  |
| PCIE\_BAR\_N\_VAL | integer | 0x00000000…0xFFFFFFFF | These have to be the same as in Megawizard |
| PCIE\_BAR\_N\_ WIDTH | integer | 1…31 | These have to be the same as in Megawizard |
| P2H\_ADDR\_SPACES | integer | 1…32 | Total P2H address spaces |
| HIBI\_DMA\_P2H\_ADDR\_SPACES | integer | 1…32 | HIBI DMA P2H address spaces |
| P2H\_ADDR\_0\_WIDTH | integer | 1…31 |  |
| P2H\_ADDR\_0\_PCIE\_BASE | std\_logic\_vector | 0x00000000…0xFFFFFFFF | This must be within one of the BARs |
| P2H\_ADDR\_0\_HIBI\_BASE | std\_logic\_vector | 0x00000000…0xFFFFFFFF |  |
| … |  |  |  |
| P2H\_ADDR\_N\_WIDTH | integer | 1…31 |  |
| P2H\_ADDR\_ N\_PCIE\_BASE | std\_logic\_vector | 0x00000000…0xFFFFFFFF | This must be within one of the BARs |
| P2H\_ADDR\_ N\_HIBI\_BASE | std\_logic\_vector | 0x00000000…0xFFFFFFFF |  |
| HIBI to PCIe addr translation | H2P\_DIRECT\_ADDR\_BASE | integer | 0x00000000…0xFFFFFFFF |  |
| H2P\_DIRECT\_ADDR\_WIDTH | integer | 1…31 |  |
| H2P\_ADDR\_SPACES | integer | 1…128 |  |
| H2P\_ADDR\_0\_WIDTH | integer | 1…31 |  |
| H2P\_ADDR\_0\_PCIE\_BASE | integer | 0x00000000…0xFFFFFFFF |  |
| H2P\_ADDR\_0\_HIBI\_BASE | integer | 0x00000000…0xFFFFFFFF | This must be within the HIBI to PCIe direct address range |
| … |  |  |  |
| H2P\_ADDR\_N\_WIDTH | integer | 1…31 |  |
| H2P\_ADDR\_ N\_PCIE\_BASE | integer | 0x00000000…0xFFFFFFFF |  |
| H2P\_ADDR\_ N\_HIBI\_BASE | integer | 0x00000000…0xFFFFFFFF | This must be within the HIBI to PCIe direct address range |
| HIBI to PCIe DMA | H2P\_DMA\_READ\_CHANNELS | integer | 1…256 | HIBI to PCIe DMA read channels |
| H2P\_DMA\_WRITE\_CHANNELS | integer | 1…256 | HIBI to PCIe DMA write channels |
| H2P\_DMA\_RW\_AMOUNT\_WIDTH | integer | 1…24 | HIBI to PCIe DMA write/read length width |

Table 4- VHDL generics

### Register map – HIBI side

The following register map is for a PCIe to HIBI component which has access to four HIBI DMA components and has eight channels for reads initiated from the PC. The component also has 256 channels for both reads and writes originating from the HIBI side.

|  |  |  |  |
| --- | --- | --- | --- |
| offset | category | Register | Description |
| 0x00000 | IP information registers | IP information regs | reserved |
| ... | ... | ... |
| 0x0001F | IP information regs | reserved |
| 0x00020 | config | config |  |
| 0x00021 | HIBI to PC | Read request | Request a read channel |
| 0x00022 | Write request | Request a write channel |
| ... | ... | ... |
| 0x00100 | Read channel 0 configure | Configuration address for read channel 0 |
| ... | ... | ... |
| 0x001FF | Read channel 255 configure | Configuration address for read channel 255 |
| 0x00200 | Write channel 0 configure | Configuration address for write channel 0 |
| ... | ... |  |
| 0x002FF | Write channel 255 configure | Configuration address for write channel 255 |
| 0x00300 | PC to HIBI | HIBI DMA comp 0 read ack | Return address for read acknowledgement from HIBI DMA component 0 |
| 0x00301 | HIBI DMA comp 0 write ack | Return address for write acknowledgement from HIBI DMA component 0 |
| ... | ... | ... |
| 0x00307 | HIBI DMA comp 3 read ack | Return address for read acknowledgement from HIBI DMA component 3 |
| 0x00308 | HIBI DMA comp 3 write ack | Return address for write acknowledgement from HIBI DMA component 3 |
| 0x00400 | P2H read channel 0 data in | Return address for read data for P2H read channel 0 |
| ... | ... | ... |
| 0x00407 | P2H read channel 7 data in | Return address for read data for P2H read channel 7 |
| 0x00C00 | HIBI to PC | Direct access | HIBI to PC direct address access will probably not be implemented |
| ... |  |  |
| 0x1FFFFFF | Direct access |  |

Table 4- HIBI side register map

## Block description

TLP decoder:

The TLP decoder is responsible for the decoding of the TLP headers to simple internal signals that the HIBI interface uses.

## Address translation

The processor on the PC side sees PCIe devices as part of its memory space and can thus directly write to and read from them. As there is no DMA configuration done to these writes and reads the addresses for these operations are based on the actual single operations made by the processor. Figure below shows an example how the addresses are translated across different components from PC to HIBI.



Figure ‑ Address translation from PC to HIBI

The PC on on this example has two PCIe devices 0 and 1. Device 0 is the fpga that is connected to the PC with a PCIe connector. The PCIe controller on the fpga is configured to have two base address registers (BAR). These two registers are associated with the two address ranges called PCIe dev 0 BAR 0 and 1 on the figure. These address ranges can be further divided by the PCIe to HIBI adapter as can be seen with BAR 0 which is divided into two address ranges called P2H PCIe addr 0 and 1. The actual address translation is done to these P2H address ranges as can be seen on the last step in the figure.

With this automatic address translation done on adapter the PC and the HIBI addresses for the same components do not have to be same.

As the data on the PCIe interconnect is sent in packets it is not efficient to do single word transfers over it. Therefore only dma transfers are supported for the transfers from HIBI to PC. This also means that the addresses on these transfers are not inferred from the single operations done on the HIBI bus but from the dma configurations written to the PCIe to HIBI adapter. This finally leads to the conclusion that no translation is needed to be done for these transfers as can be seen from figure .



Figure ‑ Address translation from HIBI to PC



## Functionality

HIBI simple DMA

Write configuration words:

|  |  |
| --- | --- |
| word 0 | word 1 |
| Memory write begin address | byte amount |

Read configuration words:

|  |  |  |
| --- | --- | --- |
| word 0 | word 1 | word 2 |
| Memory read begin address | byte amount | HIBI return address |

# Use example - Arria II GX DDR2

This use example uses an Arria II GX board connected to a regular PC running Debian Linux. The development board has a SoC containing a Altera PCIe controller connected to a PCIe to HIBI adapter which interfaces the PC to the HIBI PHY. This HIBI PHY is also connected to a HIBI MEM DMA adapter which connects to a Altera DDR2 controller. Both Altera controllers were generated with Quartus Megawizard. All these components can be seen on the figure below along with the four different clock domains this setup has.



## DMA transfer sequence diagrams

The two following figures gives the sequence diagrams for both a DMA write and read transfer sequences. First the DMA engine is configured for the transfers, where after it fetches transfer descriptors from PC memory. After this the actual work is done and the DMA engine reads data from the PC memory and writes it to the DDR2 memory or the other way around.





## DMA transfer performance

DMA transfer performance was measured with large set of DMA write and read descriptors. Each of the descriptors instructed the DMA engine in the PCIe to HIBI adapter for a write or read transfer of 256 bytes in length. Both the write and read DMA sequences had 128 of these individual transfers. Overall write and read performance data was gathered using Quartus Signal Tap II.

Average write speed achieved was 0,862 bytes/cycle or 172 MB/s at 200MHz.

The write transfer start latency was measured at 767 cycles or 3,8us at 200MHz.

## Address translation: PC => HIBI

Cpu on PC can access HIBI address space through PCIe device 0 BARs 0 and 1. BAR 0 is divided into two ranges where the first maps to the ETH2HIBI and the second to the HW IP-block. BAR 1 maps to one page of the HIBI MEM DMA address range.



## Address translation: HIBI => PC



## Use cases

CPU on PC writes to memory on HIBI:

D:\svn\funbase\shared_lib\hw_lib\ips\communication\pcie_to_hibi\doc\PC to HIBI DMA write.wmf

1. MEMWRITE32 length 64 DW (256 bytes) to PCIe device address 0x30001000 (PCIe dev 0 bar 0 offset 0x1000)
2. PCIe controller on PC sends a packet containing a 3 DW (12 bytes) TLP header for a memory write with 32-bit addressing and a 64 DW data payload
3. Packet is sent over PCIe
4. Altera PCIe controller receives the packet and generates signals to PCIe to HIBI adapter
5. The 12-byte (padded with 4 zero bytes, fifo is 16-byte wide) header is written to the TLP header fifo on the PCIe RX unit of the adapter
6. The first 4 DWs are written to the data fifo on PCIe RX unit
7. PCIe RX unit generates internal packet signals of the header and the first DW of data for the HIBI TX unit
8. HIBI TX unit sends a request for a DMA write channel on HIBI MEM DMA
9. HIBI TX receives acknowledgment of a successful request
10. HIBI TX sends write configuration for a 256-byte write
11. HIBI TX sends first 4 bytes of the write

CPU on PC reads from memory on HIBI:

Component on HIBI makes a DMA write to CPU memory on PC:

D:\svn\funbase\shared_lib\hw_lib\ips\communication\pcie_to_hibi\doc\HIBI comp to PC HIBI DMA write.wmf

Component on HIBI makes a DMA read from CPU memory on PC:

# Performance and resource usage

# Verification

The PCIe to HIBI adapter will be verified by both simulating its operation in software with Modelsim and testing its operation physically on an Arria II GX development board connected to a PC. The focus on verification will be on the physical testing as proper software based verification would require a fully fledged PCIe testbench.

## Simulation

### Single blocks

Single VHDL entities were verified in Modelsim with signal generating script files (.do files) driving the inputs. Testbenches with actual VHDL blocks generating the input signals were not designed.

### DUT

The whole adapter was verified in Modelsim with a testbench which was based on the Altera generated chaining DMA example and its testbench. The driver in the testbench first initiated a batch of three DMA writes from the memory of the simulated PC’s memory followed by three reads from the memory on the FPGA side.

## Physical testing

Physical testing for the adapter was done on a test setup that included a desktop PC fitted with an Arria II GX FPGA board. The test setup and the relevant components on the FPGA board are depicted in figure 9-1.



Figure 9-1 Test setup

The PC ran Linux on an x64 Intel CPU and it had an x16 PCIe card slot for the Arria II GX FPGA board. The FPGA board itself had only an x8 connector so only half of the available bandwidth was used. The adapter was connected to both the Altera PCIe controller and the HIBI.

### Test phases

The testing was divided into three phases. The object of the first phase was to establish that the connection between the PC and the PCIe to HIBI adapter was working. This ensured that everything was configured correctly and the PCIe to HIBI adapter used proper protocol with the Altera PCIe controller.

In the first one the adapter was tested with a simple program running on a Linux PC. This program will use a PCIe driver that only supports single reads and writes over the PCIe interconnect.

### Single register read/write test

* Tested only the write and read of a single register on the DMA registers on the PCIe to HIBI block
* Establishes that the connection from PC to PCIE\_to\_HIBI component works (PCIE phy, Altera PCIE ctrl)
* Tested register: rdma\_desc\_addr\_r

### Complete PC to DDR2 memory to PC loop test

* Tested the PC initiated read and write DMA functionality of the adapter
* PC writes 256 write dma descriptors in PC memory

### Thorough linux driver testing

* Max number of transfers with max number of data

# Documentation for developers

## Requirements

* HIBI wrapper r3 interface
* PCIe controller support
  + Altera PCI Express hard IP
* General
  + 32/64-bit address space support
  + Byte addressing
  + Byte enable for write operations
  + Data width adapter
* **IP info regs.**

## Overview of the PCI bus

* The PCI bus has three separate address spaces, config, I/O, and memory space.
* Every PCI device responds to config commands, and it can respond to I/O accesses and/or memory accesses.
* During the boot time, the BIOS or the OS sets the base address registers (BARs) through configuration space. BARs determine address ranges in I/O or memory space that a device should respond to. Obviously, those ranges should not be duplicated anywhere else in the I/O space or memory space on the same PCI bus.
* There may be multiple PCI hostbridges on some systems.
* Multiple PCI buses can be connected through PCI-to-PCI bridges.
* Some variant of the PCI bus support hot plugging. Thanks to interesting hardware such as external PCI cages with several PCI slots that are connected to the host system over a Cardbus interface every PCI driver should be hot plug aware, every driver for a hostbridge should be written to handle multiple PCI busses. Simply use the Linux interfaces for that and you'll be safe.

[http://www.linux-mips.org/wiki/PCI\_Subsystem]

## PCIe transaction types

|  |  |  |
| --- | --- | --- |
| Address Space | Transaction Types | Basic Usage |
| Memory | Read / Write | Transfer data to/from a memory-mapped  location. |
| I/O | Read / Write | Transfer data to/from an I/O-mapped location |
| Configuration | Read / Write | Device Function configuration/setup |
| Trusted Configuration | Read / Write | Trusted Device Function configuration/setup |
| Message | Baseline (including Vendor–defined) | From event signaling mechanism to general purpose messaging |

Memory Transactions include the following types:

* Read Request/Completion
* Write Request

Memory Transactions use two different address formats:

* Short Address Format: 32-bit address
* Long Address Format: 64-bit address

Configuration Transactions are used to access configuration registers of Functions within devices.

Configuration Transactions include the following types:

* Read Request/Completion
* Write Request/Completion

**General TLP header**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| offset |  | | | | | | | | | | |  | | | | | | | | | | |  | | | | | | | | | | |  | | | | | | | |
|  | 31 | | 30 | 29 | 28 | | 27 | 26 | 25 | 24 | | 23 | | 22 | 21 | 20 | | 19 | 18 | 17 | 16 | | 15 | | 14 | | 13 | 12 | 11 | 10 | | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| 31..0 | R | Format | | | | Type | | | | | R | | TC | | | | R | | | | | TD | | EP | | RO | | NS | AT | | Length | | | | | | | | | | |
| 63..32 | TLP dependant | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 95..64 |
| 127..96 |

**Memory write request 32-bit address**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| offset |  | | | | | | | | | | |  | | | | | | | | | | |  | | | | | | | | | | | | | |  | | | | | | | | |
|  | 31 | | 30 | 29 | | 28 | 27 | 26 | 25 | 24 | | 23 | | 22 | 21 | 20 | | 19 | 18 | 17 | 16 | | 15 | | 14 | | 13 | | 12 | | 11 | 10 | | 9 | 8 | | 7 | 6 | 5 | 4 | | 3 | 2 | 1 | 0 |
| 31..0 | 0 | 1 0 | | | 0 0 0 0 0 0 | | | | | | 0 | | TC | | | | 0 0 0 0 | | | | | TD | | EP | | RO | | NS | | 0 0 | | | Length | | | | | | | | | | | | |
| 63..32 | Requester id | | | | | | | | | | | | | | | | | | | | | tag | | | | | | | | | | | | | | Last be | | | | | First be | | | | |
| 95..64 | Address[31...2] | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 0 0 | |
| 127..96 | - | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

**Completion with data**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| offset |  | | | | | | | | | | |  | | | | | | | | | | |  | | | | | | | | | | | | | |  | | | | | | | | |
|  | 31 | | 30 | 29 | 28 | | 27 | 26 | 25 | 24 | | 23 | | 22 | 21 | 20 | | 19 | 18 | 17 | 16 | | 15 | | 14 | | 13 | | 12 | | 11 | 10 | | 9 | 8 | | 7 | | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 31..0 | R | Format | | | | Type | | | | | R | | TC | | | | R | | | | | TD | | EP | | RO | | NS | | AT | | | Length | | | | | | | | | | | | |
| 63..32 | Completer id | | | | | | | | | | | | | | | | | | | | | status | | | | | | b | | Byte count | | | | | | | | | | | | | | | |
| 95..64 | Requester id | | | | | | | | | | | | | | | | | | | | | tag | | | | | | | | | | | | | | 0 | | Lower address | | | | | | | |
| 127..96 | - | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

**PCIe read packet mem**

|  |  |
| --- | --- |
| offset | Register |
| 0x00 | b31...b00: Chan 0 HIBI read address |
| 0x01 | b29: buffer full, b28...b16: Chan 0 HIBI read length, b15...b00: Chan 0 PCIe req id |
| 0x02 | b15...b08: Chan 0 HIBI data write offset, b07...b00: Chan 0 HIBI data read offset |
| 0x03 | - |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

**PCIe read packet data mem**

|  |  |
| --- | --- |
| offset | Register |
| 0x000 | Chan 0 HIBI read data 0 |
| ... |  |
| 0x07F | Chan 0 HIBI read data 127 |
| 0x080 | Chan 1 HIBI read data 0 |
| ... |  |
| 0x3FF | Chan 7 HIBI read data 127 |

* PCIe completions can be divided

## Internal structures

Incomplete and partially outdated.

### TLP dec



### TLP enc

* 

### Address trans.

