TAMPERE UNIVERSITY OF TECHNOLOGY

FACULTY OF COMPUTING AND ELECTRICAL ENGINEERING

DEPARTMENT OF COMPUTER SYSTEMS

Heterogenerous IP Block Interconnection (HIBI) version 3

Reference Manual

Author:
Erno Salminen,
Timo Hämäläinen

Updated:
15th November 2011

Contents

List of Figures

List of Tables

1 Introduction

This data sheet presents the third version of *Heterogeneous IP Block Interconnection* (HIBI). HIBI is intended for integrating coarse-grain components such as intellectual property blocks that have size of thousands of gates, see [10] for examples. Topology, arbitration and data transfers are presented first. After that, data buffering and the structure of wrapper component are discussed. Finally, the developed runtime configuration is presented followed by comparison to the previous version of HIBI.

HIBI is a communication network designed for System-on-Chips. It can be used both in FPGA and ASIC designs (field-programmable gate-array, application-specific integrated circuit). Fig. 1 shows an example SoC at conceptual level. There are many different types of IP blocks (intellectual property), namely CPU (central processing unit) for executing software, memories and IP blocks that are either fixed function accelerators or interfaces to external components. All these are connected using an on-chip network.

1.1 Main points

The major design choices for HIBI were

- IP-block granularity for functional units
- Application independent interface to allow re-use of processors and IP-blocks
- Communication and computation separated
- Communication network used in all transfers, no ad-hoc wires between IPs
- support local clock domains for IP granularity

A parameterizable HW component, called HIBI wrapper, is used to construct modular, hierarchical bus structures with distributed arbitration and multiple clock domains

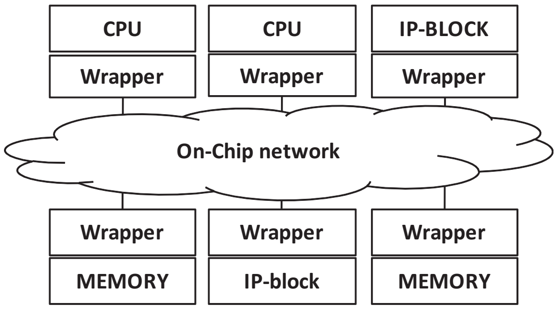


Figure 1: Conceptual structure of system-on-chip

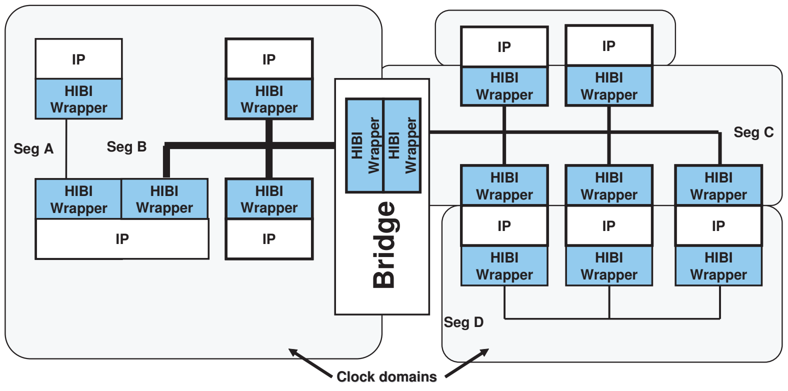


Figure 2: Example of a hierarchical HIBI network with multiple clock domains and bus segments

as shown in Fig 2 (explained later in detail). This simplifies design and allows reuse since the same wrapper can always be utilized. Configuration takes place both at synthesis time (e.g. data width and buffer sizes) and on runtime (arbitration parameters).

In addition, since we are targeting also FPGAs, there are some additional constraints

- keep the number of wires low - to avoid exhausting routing resources
- avoid global connections - to avoid long combinatorial routing delays
- avoid 3-state wires - to simplify testing and synthesis (most FPGAs allow three-state logic onlu in I/O pins)

1.2 Versions

The development of HIBI [5–7,9] started in 1997 in Tampere University of Technology. Currently, there are 3 versions of HIBI, denoted as v1-v.3. However, certain basics have remained unchanged. Hence, in the remainder the version number is omitted unless, it is necessary.

In version 2, the biggest changes were removing tri-state logic and increasing modularity and configurability.

For version 3, address decoder logic was modified to simplify usage. Furthermore, the tx and rx state machines were re-factored, which also necessitated minor change in bus timing. These latter FSM changes do not affect the IP, though.

2 HIBI topology

The topology in HIBI is not fixed, but configurable by the designer. HIBI network consists of wrappers, bus segments, and bridges. These are the basic building blocks from which the whole network is constructed and configured. All wrappers in the system are instantiated from the same parameterizable HDL (HW description language) entity and bridges are constructed by connecting two wrappers together. If the connected segments use different data widths, the bridges are responsible for the data width adaptation.

All wrappers can act both as a *master* and a *slave*. Masters can initiate transfers and slaves can only respond. In many buses, most units operate in on mode only and only few in both modes. In the most simple case, there is only segment and the topology is hence single shared bus. However, HIBI network can have multiple segments which form a hierarchical bus structure. Segments are connected together using bridges. Bridges increase latency but, on the other hand, hierarchical structure allows multiple parallel transactions. Bridge are simply constructed from 2 wrappers.

For the IP, the wrapper offers FIFO-based (first in, first out) interface, as depicted in Fig. In network side, all signals inside a segment are shared between wrappers and no dedicated point-to-point signals are used. Arbitration decides which wrapper (or bridge) controls the segment and the utilized arbitration algorithms distributed to wrappers without any central controller.

2.1 Example of hierarchical topology

Bus performance can be scaled up by using bridges. Segments having only simple peripheral devices can have a slow and narrow bus while the main processing parts have higher capacity buses.

Fig. 2 depicts an irregular HIBI network. The example has a point-to-point link (*SegA*), hierarchical bus (*SegB* and *SegC*), and multibus topology (*SegC* and *SegD*). Furthermore, *SegB* is wider than other segments and thus offers greater bandwidth. In the multibus configuration, each IP must decide which bus to use while sending. Note that *SegA* could be implemented without wrappers since there is no need for arbitration.

The example shows four clock domains. Agents in *SegA* and *SegB* are inside one domain and HIBI wrappers on *SegC* are in one domain. However, two IPs in the top right corner use different clock than the wrappers of *SegC*. The IPs in the bottom right corner and all wrappers in *SegD* are in one domain. The number of clock domains is not otherwise restricted but all wrappers in one bus segment must use the same clock. Handshaking between the clock domains is done in the IP-wrapper interface or inside the bridge [2, 3]. This allows the construction of GALS systems. The example shows only one bridge but HIBI does not restrict either the number of bridges or hierarchy levels in contrast to many bus architectures.

2.2 Switching

Transfers inside a bus segment are circuit-switched and use a common clock due to (current) implementation of the distributed arbitration. However, HIBI bridges utilize switching principle that resembles packet-switching so that bus segments are not circuit-switched together. Instead, the data is stored inside the bridge until it gets an access to the other segment. The data is forwarded to next segment as soon as possible like in wormhole routing. However, no guarantees are given for the minimum length of continuous transfer. If the bridge cannot buffer all the data, the transfer is interrupted and the source segment is free for other transfers. The interrupted wrapper will continue the transfer on its next turn. It is also possible that a bridge buffers parts from multiple transfers.

3 Data transfer operations

In HIBI, all transfers are bursts. In practice, there is always 1 address word followed by n data words. The max. n is wrapper-specific arbitration parameters. HIBI v2. used multiplexed address and data lines, but HIBI v.3 allows transmitting them in parallel. Due to multiplexed addr/data lines, it is beneficial to send many data into single address. This is quite different from "traditional" memory accesses, with address and data at the same time. Hence, the destination IP should keep track of received data count, e.g. TUT's SDRAM controller can do this to avoid excess transmitting addr + data pairs

The transfers are pipelined with arbitration, and hence the next transfer can start immediately when the previous ends. The protocol on the bus side is optimized so that there no wait cycles are allowed during a transfer. This means that is sender runs out of data or the receiver does not accept it fast enough, the transfer is interrupted. On the next arbitration turn, the wrapper it continues automatically. Note that IP may transfer data at pace it wishes. IP has only to ensure that there is space in TX FIFO while writing and that RX FIFO is not empty while reading.

In order to increase bus utilization, HIBI uses so called split-transactions in read operation. It means that single read operation is split into two phases: request and response. The bus segment is released while the addressed IP handles the read request and prepares its response. The other wrappers may use bus during that period and this increases the overall performance, although a single read becomes a little slower due additional arbitration round.

Write operation

- Includes destination address

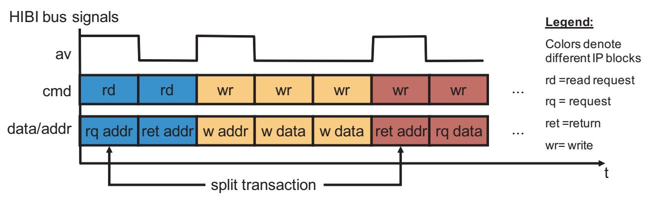


Figure 3: Example of read and write operations.

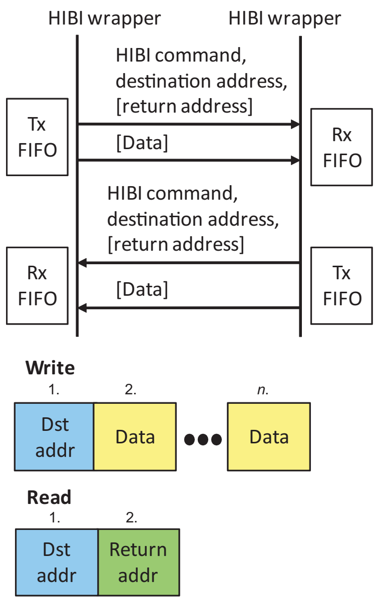


Figure 4: Basic transactions are write and read.

- Data is sent in words (=HIBI bus width)
- Several words can follow: all will be sent to the same destination address

Read operation

- Includes exactly two words: destination address and return address (where to put the data)
- Data is received in words
- Several words can be received (all to same return address)
  - No handshaking: data is transmitted/received when bus, sender, or receiver are available
  - No acknowledgements or flow control

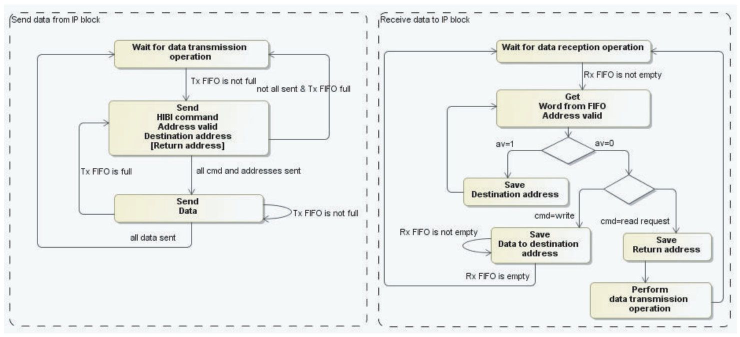

Send data from IP block
Wait for data transmission operation
Tx FIFO is not full
not all sent & Tx FIFO full
Send HIBI command Address valid Destination address [Return address]
Tx FIFO is full
all cmd and addresses sent
Send Data
Tx FIFO is not full
all data sent
Receive data to IP block
Wait for data reception operation
Rx FIFO is not empty
Get Word from FIFO Address valid
av=1
av=0
Save Destination address
cmd=write
cmd=read request
Rx FIFO is not empty
Save Data to destination address
Save Return address
Rx FIFO is empty
Perform data transmission operation

Figure 5: Logical steps that IP does during transaction.

Figs. 3 and 4 depict the two basic transfers: sending the read request, write, and the response to read. IP can send multiple read requests before the previous ones have completed. It is the responsibility of the requestor to keep track which response belongs to which request. This can be implemented with appropriate use of return addresses. The reader does not get data any faster but the advantage is that the shared medium is available for other agents in the middle of the transmission process and consequently the achieved total throughput increases. In packet-switched networks the split-transactions are commonly used and also in modern bus protocols, such as AMBA

Since there is exactly one path between each source and destination, all data is guaranteed to arrive in-order and hence no reordering buffers are needed at the receiver. Data can be sent with different relative priorities. High priority data, such as control messages, bypass the normal data transfers inside the wrappers and bridges resulting in smaller latency. This does not change the timing of bus reservations, but it selects what is transferred first.

3.1 HIBI Basic Transaction Motivation

HIBI was motivated by streaming applications where continuous flow of data is transmitted between IPs. Destinations are merely ports than random accessed memory locations. Hence, HIBI is not natively a processor memory bus but can be used for it as well.

HIBI does not implement end-to-end flow control but the IPs must do not explicitly. The FIFO buffers and rx and tx side may get full if the receiver does not eject data fast enough, and this will throttle the transmitter as well. The wrappers takes care of retransmission at the link level. (HIBI v.1 dropped data if the receiving buffer got full but usage of v.1 is not recommended anymore).

Fig. 5 shows the steps that IP needs to take when communicating using HIBI. On the left, IP sends data when the TX FIFO is not full. It must assign data, address valid (strobe), command, and write enable signals at the same time. When receiving data, IP first checks is the incoming value address or data word. This is done by examining the address valid signal. One word is removed from the FIFO on every clock cycle when receiver assigns read enable signal. Next, IP must check is the operation write or read. In case of write, it stores the incoming data to location defined by the address. In case of read, the second word denotes the return address. It is the address, where the read data word must be transmitted.

4 Addressing

All IP-blocks have unique address and register space defined at design time and every transfer starts with single destination address. Source identification not included in basic transfer and hence

a) Use data payload to define source, e.g. first world in a data packet

b) Use unique address inside IP block for each source (IP knows from the destination address the sender)

Every wrappers has a set of addresses and they set with a VHDL generic (automatic by Kactus). Wrappers may have varying address space sizes, e.g. simple UART has only 2 addresses whereas memory has 16K addresses. Incoming Addresses go through the receiving wrapper to the receiving IP and it can identify the incoming data by its address. For example, the uppermost bits define which IP is addressed and the lowermost define the register of that IP.

There are wo ways to set addresses 1. manually

2. A generator script in Kactus tool does this automatically according to system specification

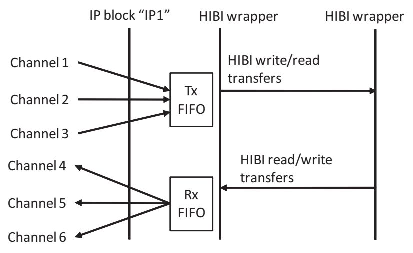
IP may write arbitrarily long bursts to wrapper. Perhaps only one address in the beginning followed by arbitrary number of data words. Moreover, IP writes data in arbitrary pace to wrapper. There can be any number of idle cycles between data words. Therefore, the bursts sent by the IP do not necessarily have the same length in the bus (between wrapper). For example, wrapper may split long IP-transfer into multiple bus transfers if the arbitration algorithms gives ownership to another wrapper in the middle. Each part of the transfer starts with the same address as previous. On the other hand, a wrapper may send many short IP-transfers consecutively at one turn.

These properties have two consequences:

1. Bursts from multiple source IP will be interleaved

2. Destination may get different number of addresses than sender.

Note that the destination IP does not know the sender unless it is separately encoded into data or address



Channel #	Global address	Destination IP-block	Source IP-block	Meaning
1	0xA01	IP3	IP1	"Processed data out to IP3"
2	0xB01	IP2	IP1	"Control output to IP2"
3	0xB02	IP2	IP1	"Status output to IPX"
4	0xC01	IP1	IP2	"Raw data input from IP2"
5	0xC02	IP1	IP3	"Status messages from IP3"
6	0xC03	IP1	IPX	"Control input from IPX"

Figure 6: Relation between addresses and channels.

4.1 HIBI destination addresses and channels

In HIBI v.2, all transfers are bursts, i.e. address is transmitted only in the beginning of the transfer and it is followed by one or more data words. The maximum burst length is wrapper-specific. HIBI uses mainly two-level addressing scheme: the upper bits of the address identify the target terminal (e.g. $destination_0$) whereas the lower bits define the additional identifier. This identifier can be used either as an address to local memory, to select the correct reception channel on DMA, to identify the source of the data, or to select requested service. Certain packet-switched networks (at least those implemented in this work) allow only one address per terminal. In that case, the second level address must increase the header length.

HIBI destination addresses are

1. internal registers
2. ports (to/from IPs internal logic)
3. IPs memory locations transparent to outside

Burst transfers use channels (or ports) and IP block must perform addressing (increment) internally since all data is sent to one address. If IP's memory is transparent, the address seen outside includes also IP-block address (e.g. in address 0xB100, oxB000 defines the target IP and 0x100 internal memory)

HIBI transfers can be abstracted as channels at IP-block side (but not formally specified how). Easiest way to separate channels is to use unique HIBI addresses. It is IP/System level design issue is to give meaning to the channels. For example, accelerator receives data from CPU0 via channel 0 and from CPU1 via channel 1 and so on. Basic HIBI transactions are used to handle possible flow control and handshaking in addition to transfers. Fig. 6 shows an example with 6 channels (addressing style of HIBI v.2) .

Note that all incoming channels 4-6 have the same 4 upper bits in their addresses. In other words, the example uses a convention that the base address of IP1 is 0xC00 and therefore its uppermost address is implcitly 0xCFF. The channels can be easily distinguished from the lowest address bits. In HIBI v.3 the addressing defined using two parameters: start and end address. Designer can use the same addresses as in HIBI v.2 based systems, but this scheme allows more freedom is address definitions, which especially beneficial in hierarchical systems

4.2 Implementing flow control

Flow control and handshaking must be implemented in IP-blocks. In practise leads to IP-block specific methods which must be carefully specified at design time. Minimum issues to be agreed

1. Sender identification (e.g. unique channel address ties Ip block and purpose together)
2. Transfer size
3. Size unit in addressing(bytes/words)
4. Are byte enables utilized
5. Messages for non-posted transactions (Acknowledgements to write/read)

4.3 Example: Overlapping and breaking transfers

It was noted that the transfers may split due to arbitration. Example in Fig. 7 clarifies the phenomenon. Let us assume that IP 1 and IP 2 send data to IP 3. We notice that IP 1 gets the first turn in the bus its two first data words arrive to IP 3. However, after that IP 3 gets two consecutive words from IP 2, then from IP 1 and so on. Note that in realistic case, the arbitration happens less frequently but the example highlights the issue.

As a conclusion

1. Data is transferred in order through FIFO
2. If tx is interrupted in bus, wrapper re-sends address and continues tx of rest of data to destination
3. Sender tx FIFO can not be cleared once written
4. Receiver can identify to which channel data is coming based on address

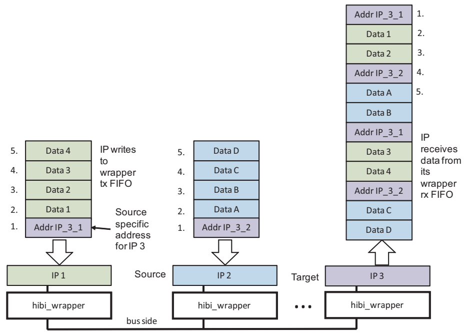


Figure 7: The transfers may get intereleaved due to arbitration.

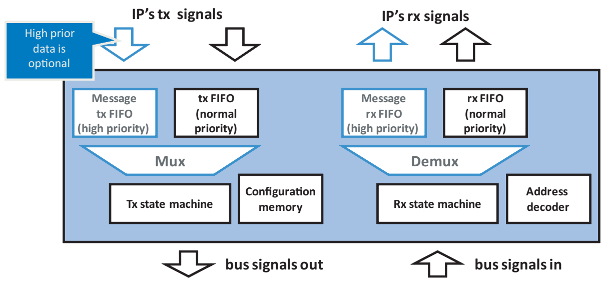

IP's tx signals
IP's rx signals
High prior data is optional
Message tx FIFO (high priority)
tx FIFO (normal priority)
Message rx FIFO (high priority)
rx FIFO (normal priority)
Mux
Demux
Tx state machine
Configuration memory
Rx state machine
Address decoder
bus signals out
bus signals in

Figure 8: Structure of HIBI v.2 wrapper and configuration memory

5 Wrapper structure

HIBI network is constructed using parameterizable builgin blocks called wrappers. The wrappers take care of arbitration, link-level transmission, data buffering, and optional clock-domain crossing. All signals on both sides of the wrapper are unidirectional. For example, there are separate multibit signals data_in and data_out. Let us first consider the bus side, i.e. the signals between wrappers.

The structure of the HIBI v.2 wrapper is depicted in Fig 8. The modular wrapper structure can be tuned to better meet the application requirements by using different versions of the internal units or leaving out properties that are not needed in a particular application.

On IP side, there can be separate interfaces for every data priority or they can be multiplexed into one interface. Furthermore, the power control signals can be routed out of the wrapper if the IP block can utilize them.

The main parts are buffers for transferring and receiving data and the corresponding controllers. The transfer controller takes care of distributed arbitration. The configuration memory stores the arbitration parameters. Relative data priority is implemented by adding extra FIFOs. A (de)multiplexer is placed between the FIFOs and the cor-

Table 1: The signals at bus side, i.e. between the wrappers, in v.2 and v.3

Signal	Width	Dir.	Meaning
data	generic	i+o	Data and address are multiplexed into single set of wires
av	1	i+o	Address valid. Notifies when address is transmitted
cmd	3	i+o	Command: read or write, data or conficuration etc.
full	1	i+o	Target wrapper is full and acannot accept the data. Current transfer will be repeated later
lock	1	i+o	Bus is reserved

responding controller so that the controller operates only on a single FIFO interface. The separate (de)multiplexer allows adding FIFOs to support priorities in excess of two without changing the control. Currently, transmit multiplexer uses pre-emptive scheduling.

HIBI v.2 has multiplexed address and data lines whereas HIBI v.1 uses separate address and data lines. Multiplexing decreases implementation area because signal lines are removed and less buffering capacity is needed for the addresses. This causes overhead in control logic but that is less than the saving in buffering. Having fewer wires allows wider spacing between wires and hence lower coupling capacitance. On the other hand, the saved wiring area can be used for wider data transfers to increase the available bandwidth. The HIBI protocol does not require any specific control signals, but message-passing is utilized when needed. HIBI v.1 assumes strictly non-blocking transfers and omits handshake signals to minimize transfer latency but one handshake signal *Full* was added to HIBI v.2 to avoid FIFO overflow at the receiver. As a result, blocking models of computation can be used in system design and, in addition, the depths of FIFOs can be considerably smaller than in HIBI v.1.

5.1 Bus-side signals

All outputs from wrappers are “ORed” together and OR-gates’ outputs are connected to all wrappers’ inputs. This scheme avoids the tri-state logic that was used in HIBI v.1. Table 1 lists the bus side signals and Fig. 9 illustrates the connection between wrapper and OR-gates. The cycle-accurate bus timing is omitted from this used guide for brevity. All bus side outputs come directly from register except the handshaking signal full.

The number of data bits can be freely chosen. This is beneficial, for example, when error correcting or detecting codes are added to data and the resulting total data width is not equal to any power of two. Active master asserts *Lock* signal when it reserves the bus. Handshaking is done with the $Full$ signal. When $Full$ is asserted, the data word

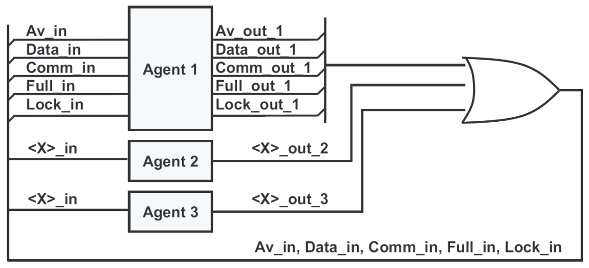


Figure 9: Structure of HIBI v.2 wrapper and configuration memory

on the bus must be retransmitted by the wrapper. To improve modularity, all signals are shared by all wrappers within a segment and no point-to-point signaling is required. Consequently, the interface of a wrapper does not depend on the number of agents and the wrapper can be reused more easily. An OR network was selected for bus signal resolution.

The HIBI implementation pays special attention on minimizing the transfer latency by removing empty cycles from the arbitration process by pipelining. Empty cycles are here defined as cycles when at least one wrapper has data to send but the bus segment is not reserved. An optimized protocol allows lower frequency, and hence lower power, for certain performance level than inefficient protocol. Empty cycles appear also when bus utilization is low as distributed round-robin arbitration takes one cycle per agent. If only one agent is transmitting, it has to wait a whole round-robin cycle between transfers. In such cases, the priority-based arbitration is useful.

5.2 IP-side signals

The signals at IP interface are mostly the same signals as in the bus side. Interface signals are connected to FIFO buffers inside the wrapper and all output signals of the wrapper come from registers.

Most signals are driven by both IP and wrapper

- Command
- Address / Address valid
- Data
  - May have high (message) and low (data) priotities (depends on wrapper type)
  - Priority is defined by transmissting IP-block (source)

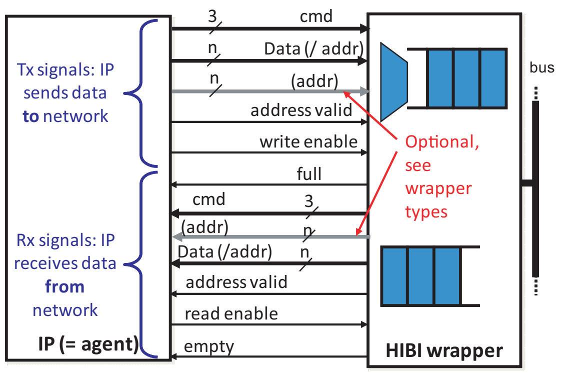


Figure 10: The signals between IP and wrapper

Table 2: The signals at wrapper's IP interface

Signal	Width	Dir.	Meaning
rst_n	1	i	Active low reset
clk	1	i	Clock, active on rising edge. Same for all wrappers inside one segment
data	generic	i+o	Data and address are multiplexed into single set of wires
av	1	i+o	Address valid. Notifies when address is transmitted
cmd	3	i+o	Command: read or write, data or concifuration etc.
re	1	i	Read enable. Wrapper can remove the first data from FIFO
we	1	i	Write enable. Adds the data from IP to TX FIFO
full	1	o	TX FIFO is full
empty	1	o	RX FIFO is empty
one_p	1	o	TX FIFO has one place left, i.e. almost full
one_d	1	o	RX FIFO has one data left, i.e. almost empty

On the other hand, the FIFO access control signals depend on the direction. Both control signals Write enable and Read enable and driven by wrapper. The status signals are driven by wrapper. There are always at least two status signals FIFO full and FIFO empty. In addition, the FIFO buffers developed for HIBI offer two others: One data left at FIFO and One place left at FIFO, which may simplify the logic IP.

The address signals at IP side offer few choices that described next.

Fig 10 depicts the signals between IP and wrapper and Table 2 list their details.

5.3 Variants of IP interface

There are 4 variants of the IP interface depending on how to handle

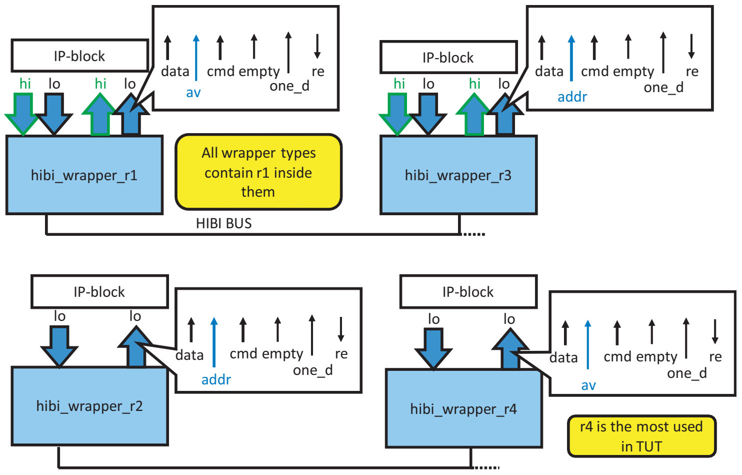

IP-block
hi lo hi lo
data av cmd empty one_d re
hibi_wrapper_r1
All wrapper types contain r1 inside them
hibi_wrapper_r3
data addr cmd empty one_d re
HIBI BUS
hibi_wrapper_r2
hibi_wrapper_r4
r4 is the most used in TUT

Figure 11: There are 4 variants of IP interface. There are two selectable features, namely separations of hi/lo-prior data and separate/multiplexed addressing.

a) high/low priority data: one or two interfaces

b) address and data: separate interfaces or one multiplexed

The different wrapper are denoted with postfix $\_r < x >$

r1: a) 2 interfaces hi+lo; b) muxed a/d

r2: a) 1 interface hi/lo; b) separate a+d

r3: a) 2 interfaces hi+lo; b) separate a+d

r4: a) 1 interface hi/lo; b) muxed a/d

Since these options affect only the IP side, different wrapper types can co-exist in the same system, and the wrappers' bus side interface is always the same. Furthermore, the addresses work directly between wrapper types. However, hi-priority data cannot bypass lo-prior data in wrapper types r2 and r4. However, all data is always transmitted

For example, Nios subsystems utilize commonly r4 but SDRAM utilizes r3. This is because SDRAM ctrl distinguishes DMA configuration and memory data traffic with priority of incoming data. It also prevents dead-lock. Fig 11 depicts variants of wrapper's IP side signals. Interface type r1 is the "native" interface that is used inside all other variants.

5.4 Signal naming in VHDL

The side and direction are marked into signal name in HIBI wrapper VHDL, for example

1. agent_data_in, agent_data_out,

2. bus_data_in, bus_data_out

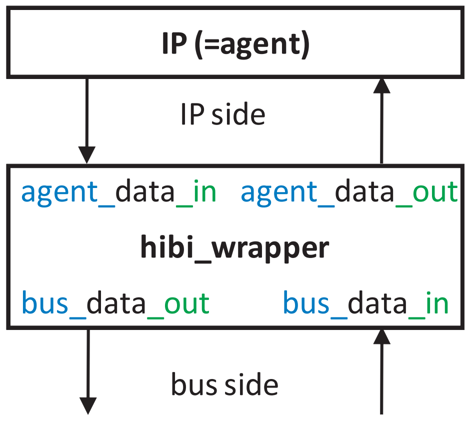


Figure 12: The naming convention of ports

Fig. 12 clarifies the naming scheme.

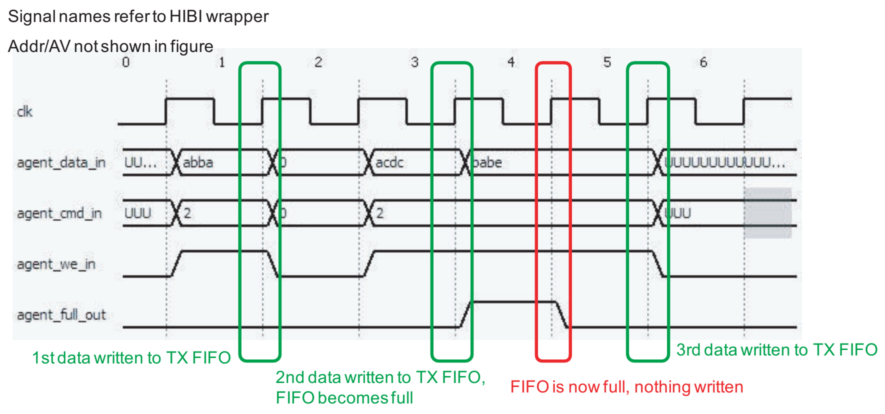
5.5 Cycle-accurate timing

For brevity, only the IP side timing is explained. It is actually very simple. The timing when transmitting is depicted in Fig 1) IP checks that tx FIFO is not full 2) IP sets data, command, addr/av, and write_enable=1 for one clk cycle

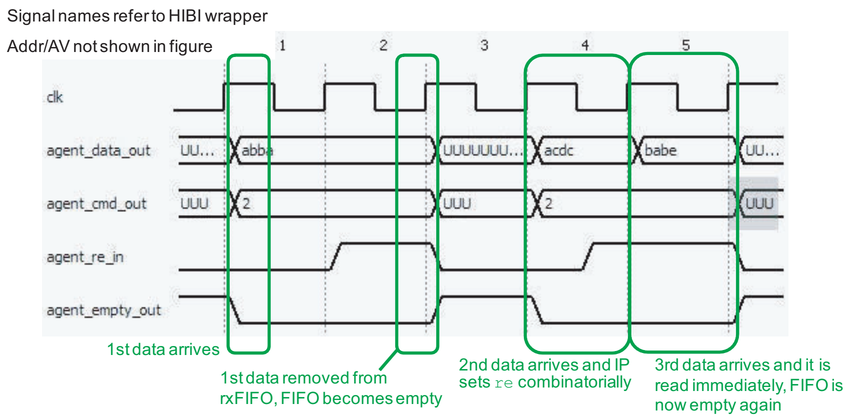
The timing when receiving is depicted in Fig 1) IP checks that rx FIFO is not empty 2) IP captures data, command, and addr/av 3) IP sets read_enable=1 for one clk cycle

Notes on signal timing

1. Very easy to write/read on every other cycle
2. Almost as easy to write/read on every cycle. Needs a bit more care with checking empty and full
3. IP may keep we=1 and re=1 continuously and just change/store data according to full/empty
4. Signal FIFO full comes from register. It goes high on the next cycle after the write, if at all. In the Tx example, writing value 0xacdc filled the FIFO
5. Setting we=1 when FIFO is full has no effect
6. Setting re=1 when FIFO is empty has no effect
7. Received data, addr/av and command appear to interface, if FIFO was empty before. IP can use them directly. They are "removed" only when read enable is activated o Checking empty==0 ensures validity


Signal names refer to HIBI wrapper
Addr/AV not shown in figure
0 1 2 3 4 5 6
clk
agent_data_in
UU... abba 0 acdc babe UUUUUUUUUUUU...
agent_cmd_in
UUU 2 0 2 UUU
agent_we_in
agent_full_out
1st data written to TX FIFO
2nd data written to TX FIFO, FIFO becomes full
FIFO is now full, nothing written
3rd data written to TX FIFO

(a) IP sends.


Signal names refer to HIBI wrapper
Addr/AV not shown in figure
1 2 3 4 5
clk
agent_data_out
UU... abba UUUUUUU... acdc babe UU...
agent_cmd_out
UUU 2 UUU 2 UUU
agent_re_in
agent_empty_out
1st data arrives
1st data removed from rxFIFO, FIFO becomes empty
2nd data arrives and IP sets re combinatorially
3rd data arrives and it is read immediately, FIFO is now empty again

(b) IP receives data

Figure 13: Examples of timing at IP interface.

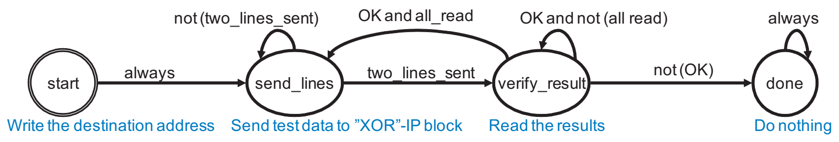


Figure 14: Example FSM of an IP

8. Data and command values are undefined when FIFO is empty. Most likely the old values remain

A Simple example VHDL code can be found in SVN /release_1/lib/hw_lib/ips/computation/image_xor/tb/tb_image_xo
It shows how to send address and data.

Fig. 14 shows the simple example FSM of the IP.

Sometimes the output registers of the IP may cause unexpected behavior for novices.

Even if FIFO appears “not full”, IP cannot necessarily write new data. That happens if it was already writing and there was only one place left at the FIFO. Hence, remember to check if IP is already writing!

The following code snippet should clarify correct writing

Example code of IP’s sending control

```
if (we_r ='1' and one_p_in='1') or full_in ='0' then
 we_r   <= '0'; //FIFO is becoming or already full
else
 we_r   <= '1'; // There is room in FIFO
 data_r <= new_value;
end if;
```

HIBI wrapper shows the data as soon as it comes from the bus. Same data might get used (counted) twice, if IP only checks the empty signal. Remember to check if IP is already reading! The following code snippet should clarify correct reading

Example code of IP’s reception handling

```
if (re_r = '1' and one_d_in = '1') or empty_in = '1' then
 re_r <= '0'; // Stop reading
else
 re_r <= '1'; // Start or continue reading
end if;

if re_r = '1' then
 if hibi_av_in = '0' then
  // handle the incoming address
 else
  // handle the incoming data
 end if;
end if;
```

Common pitfalls

- Not noticing that tx FIFO fills while writing. Consequence: Some data are lost (not written to FIFO)
- Write enable remains 1 for one cycle too long. Undefined data written to FIFO, or the same data is written twice o In both of above, the likely cause is not acocunting to output register of the IP
- Not noticing that rx FIFO goes empty while reading. Data consumed by IP is undefined
- Read enable remains 1 for one cycle too long. Next data is accidentally read away from the FIFO unless FIFO was empty
- Not noticing that rx data changes only after the clock edge when re=1. IP uses the same data twice

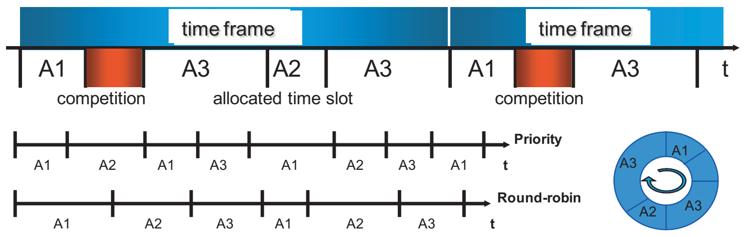


Figure 15: Example timing in 3 arvitration policies.

6 Arbitration

A distinct feature in HIBI is that arbitration is distributed to wrappers, meaning that they can decide the correct time to access the bus by themselves. Therefore, no central arbiter is required. In practice, Bus is "offered" to one wrapper on each cycle. The wrapper reserves the bus using signal lock if has data to send.

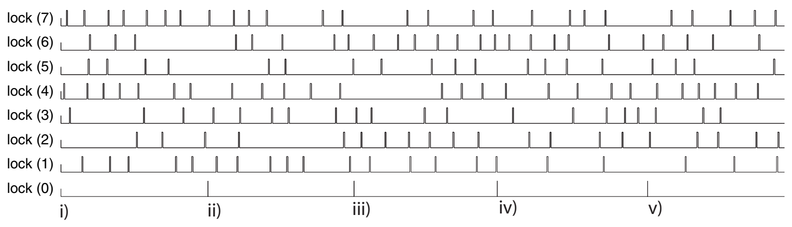
Multiple policies are supported

1. Fixed priority, Round-robin
2. Dynamically adaptive arbitration (DAA)
3. Time-division multiple access (TDMA)
4. Random
5. Combination of above

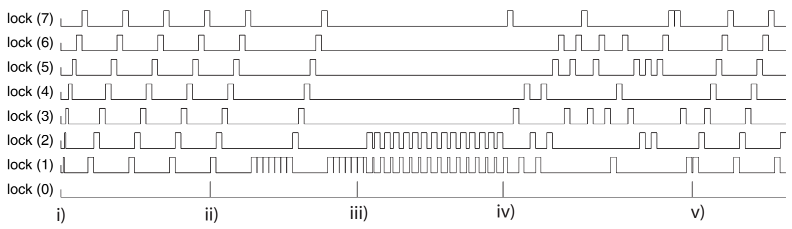
A scheme called Dynamically Adaptive Arbitration (DAA) was presented in [4]. In most cases, designers should use round-robin or DAA. If there is minor performance bottleneck, one can easily configure the arbitration parameters.

Fig. 15 shows an example of different policies. A two-level arbitration scheme, a combination of time division multiple access (TDMA) and competition, is used in HIBI. In TDMA, time is divided into repeating time frames. Inside frames, agents are provided time slots when they are guaranteed an access to the communication channel. This way the throughput of each wrapper can be guaranteed. The worst-case response time for a bus access through TDMA is the interval of the adjacent time slots. TDMA in HIBI supports two flavors for handling the slots when there is no data send: keeping them or releasing the bus for competition.

Competition is based either on round-robin or non-pre-emptive priority arbitration. The second level mechanism is used to arbitrate the unassigned or unused time slots. If the agent does not have anything to send in the beginning of its time slot, the time slot can be given away to allow maximal bus utilization. Priority arbitration as a second level method attempts to guarantee a small latency for high priority agents whereas



(a) Low contention (send probability 4% per agent).



(b) High contention (send probability 30% per agent).

Figure 16: Various arbitration schemes for 8-agent single bus and uniform random traffic. The differences become evident on highly utilized bus.

round-robin provides a fair arbitration scheme. When the bus is freed and priority scheme is utilized, the agent with the highest priority can reserve the bus on the first cycle. If the bus has been idle for two cycles, the agent with the second highest priority may reserve it and so on. The maximum transfer length is restricted with runtime configurable parameter *max_send*. For round-robin, the maximum wait time for accessing the bus is obtained by summing all *max_send* values. For priority-based arbitration, the maximum wait time can be defined only for the two highest priorities. This means that the low-priority agents may suffer starvation and system may end up in deadlock. Therefore, using only priority arbitration is not recommended.

6.1 Detailed timing example

Fig. 16 shows the differences in various arbitration policies and two traffic loads (low and high contention). HIBI is configured as single bus with 8 agents. Agent 0 performs dynamic reconfiguration (time instants $i - v$) and other agents generate uniformly distributed random traffic. The reconfiguration changes the arbitration policy at runtime. The exact configuration procedure is explained in more detail later The utilized arbitration policies are

i) round-robin

ii) combination of priority and round-robin

iii) priority

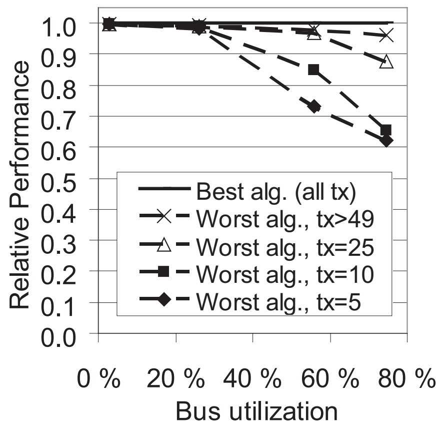


Figure 17: Relative performance of arbitration algorithms in MPEG-4 encoding [4]

iv) random

v) round-robin (again).

Round-robin offers fair arbitration (each agent has its share) whereas priority favors the highest priority agents and leads to starvation of others. Their combination switches between them at user-defined intervals. Arbitration policy does not play a major role when bus is lightly loaded, as illustrated in Fig. 16(a). The differences are clear with higher load, Fig. 16(b).

6.2 Performance implications

Various arbitration methods of HIBI were compared in [4]. The test case was MPEG-4 encoding on MPSoC. HIBI has 6 arbitrated components: 4 CPUs, SDRAM, and performance monitor; all operating at $50MHz$ frequency. The maximum transfer length was varied from 5 words (denoted as $tx = 5$) to non-limited. Transfer length has major impact but all lengths of 50 words or over (tx>49) resulted in equal performance. The bus frequency was set to $1, 2, 5$, or 50 MHz in order to achieve varying bus utilization ($75\%, 56\%, 26\%$, and 3%, respectively) with single application. The best and worse algorithms vary case by case but DAA performed well in general.

Fig. 17 plots the relative encoding performance between the worst and best algorithms. The curves denote different transfer lengths, and 1.0 is the best algorithm for each case. Tx lengths over 49 are joined for clarity because they yield practically the same results. With short transfers, the worst algorithm at 1 MHz HIBI (75% utilization) offers only $0.62x$ the performance of the best, at 2 MHz $0.73x$, at 5 MHz $0.98x$, and at 50 MHz there are no differences.

7 Commands

Source IP sets the command and most commands are forwarded to the receiving IP. The most common commands are:

- Write data - regular send operation, so called posted write
- Read request - split-transaction, the requested data is returned later with regular write command

The other, less common commands are

- Idle - IPs never use this command, but this appears on the bus when no-one sends anything
- High priority - bypasses normal data in the wrappers, otherwise just like regular operation, can be added to many commands
- Write and read config - access the configuration memories inside the wrappers. Not forwarded to the IP at the receiving end
- Multicast - send the same data to multiple targets (only in HIBI v.2)
- Non-posted write - Receveir IP must provide some response (ACK or NACK) (v.3 only)
- Linked read + conditional write - to perform read-modify-write (v.3 only)
- Exclusive access - reserve the whole path to the destination, read, write, and remove the lock (v.3 only)

HIBI v.3 has 5 command bits and v.2 had only 3 bits,see Tables 3 and 4.

8 Buffering and signaling

The model of computation used in HIBI design approach assumes bounded first-in-first-out (FIFO) buffers between processes. A simple FIFO interface can be adapted to other interfaces such as the OCP (Open Core Protocol) [8]. Consequently, IP components use only OCP protocol and are isolated from the actual network implementation. Ideally, network can be chosen freely without affecting the IPs. However, not all features of HIBI, such as relative data priorities or dynamic reconfiguration, can be used with OCP directly but only the basic transfers.

To avoid excess buffering or retransfers, the received data must be read from the FIFO as soon as possible, for example by using a direct memory access controller. As a result, the receiver buffer space is not dictated by the *amount* of transferred data, but the *latency* of reading data from the wrapper. This scheme resembles wormhole routing, but the links are not reserved if the receiver is stalled.

Table 3: The command codes in HIBI v.3

Cmd	Code [4:0]	Code [decimal]	Meaning
idle	0 0000	0	Appears on the bus when it is free
<reserved>	0 0001	1	not used, most unused codes hidden from the table
wr data	0 0010	2	Regular write
wr data hi-prior	0 0011	3	- “ - w/ high priority
rd data	0 0100	4	Request of the split-transaction
rd data hi-prior	0 0101	5	- “ - w/ high priority
rd data linked	0 0110	6	
rd d. linked hi-p	0 0111	7	- “ - w/ high priority
wr data non-post	0 1000	8	Write that expects response
wr d. non-post hi-p	0 1001	9	- “ - w/ high priority
wr conditional	0 1010	10	Write that follows rd linked
wr cond. hi-p	0 1011	11	- “ - w/ high priority
excl. lock	0 1101	13	Locks the path to the destination
excl. wr	0 1111	15	Exclusive write, must follow excl.lock
excl. rd	1 0001	17	Exclusive read request, must follow excl.lock
excl. release	1 0011	19	Removed the lock from the path
wr config	1 0101	21	
rd config	1 0111	23	
<reserved>	1 1xxx	24-31	not used

Table 4: The command codes in HIBI v.2

Cmd	Code [2:0]	Meaning
idle	000	Appears on the bus when it is free
wr config data	001	Updates config mem inside the wrapper
wr data	010	Regular write
wr data hi-prior	011	High-priority data bypasses the regualr one
rd data	100	Request of the split-transaction
rd config data	101	Requests a value from wrapper’s config mem
multicast data	110	Sends to all wrappers whose uppemost addr bits match
multicast config	111	Same as above for high-priority data

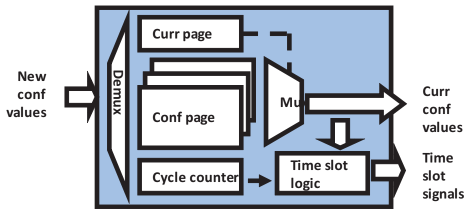


Figure 18: Structure of the wrapper's configuration memory

9 Configuration

HIBI is both modular and configurable. At design time: structural and functional settings are made, whereas at run-time, one can modify data transfer properties (arbitration types, wrapper specific QoS settings).

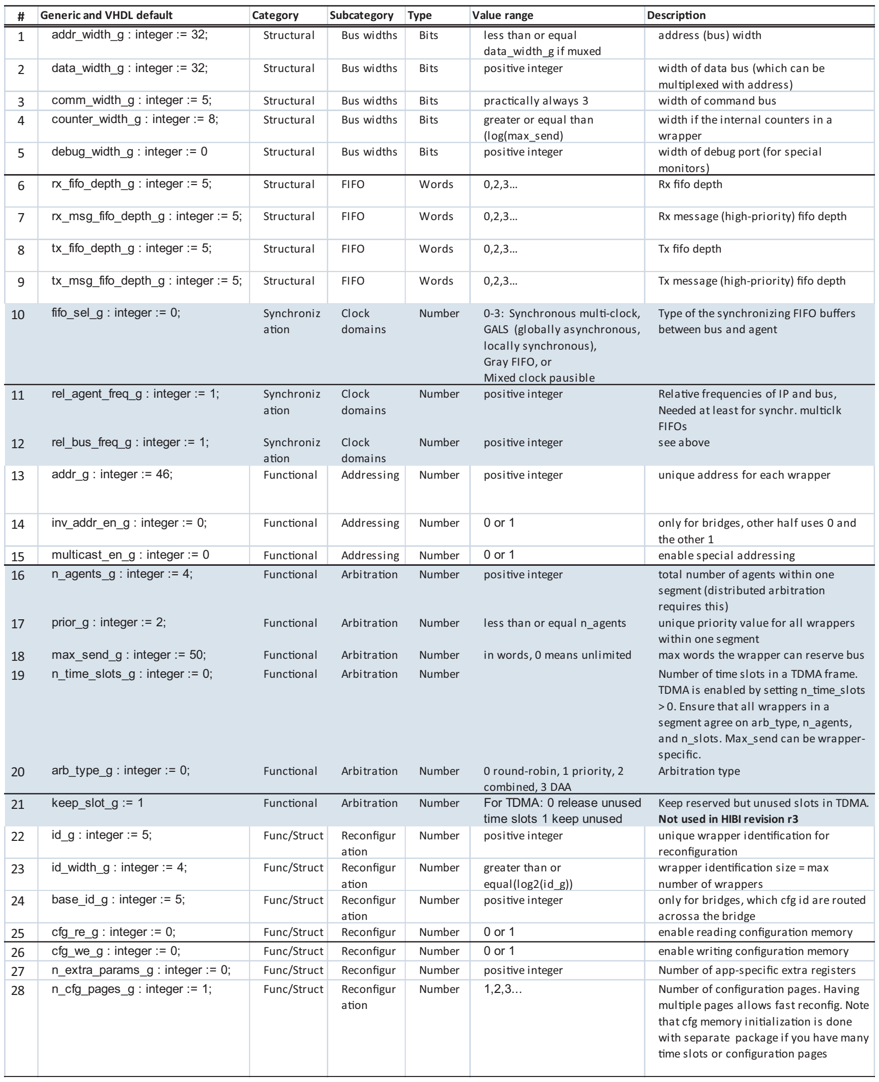
Fig. 18 shows the structure of the configuration memory.

9.1 Generic parameters in VHDL

HIBI has a large set of generic parameters. They are categorized as follows

1. Stuctural
    - Widths of interface ports: data, command, debug port
    - Widths of internal signals: address, wrapper identifier field, counters
    - Sizes of tx and rx FIFOs, both lo and hi priorities
    - Use 0, 2, 3 etc.
    - Run-time configuration: number of cfg pages, num of app-specific extra registers
2. Synchronization
    - Type of the synchronizing FIFO buffers
    - Relative frequencies of IP and bus
3. Functional
    - Identifier, own address
    - For bridges: base identifier, inverted address space
    - Arbitration: type, priority, how many words to at one turn, number of agents in the same segment
    - For TDMA: number of time slots, how to handle unused slots (keep/give away)

Table 5: Properties of HIBI v.1 and v.2.



#	Generic and VHDL default	Category	Subcategory	Type	Value range	Description
1	addr_width_g : integer := 32;	Structural	Bus widths	Bits	less than or equal data_width_g if muxed	address (bus) width
2	data_width_g : integer := 32;	Structural	Bus widths	Bits	positive integer	width of data bus (which can be multiplexed with address)
3	comm_width_g : integer := 5;	Structural	Bus widths	Bits	practically always 3	width of command bus
4	counter_width_g : integer := 8;	Structural	Bus widths	Bits	greater or equal than (log(max_send)	width if the internal counters in a wrapper
5	debug_width_g : integer := 0	Structural	Bus widths	Bits	positive integer	width of debug port (for special monitors)
6	rx_fifo_depth_g : integer := 5;	Structural	FIFO	Words	0,2,3...	Rx fifo depth
7	rx_msg_fifo_depth_g : integer := 5;	Structural	FIFO	Words	0,2,3...	Rx message (high-priority) fifo depth
8	tx_fifo_depth_g : integer := 5;	Structural	FIFO	Words	0,2,3...	Tx fifo depth
9	tx_msg_fifo_depth_g : integer := 5;	Structural	FIFO	Words	0,2,3...	Tx message (high-priority) fifo depth
10	fifo_sel_g : integer := 0;	Synchronization	Clock domains	Number	0-3: Synchronous multi-clock, GALS (globally asynchronous, locally synchronous), Gray FIFO, or Mixed clock pausible	Type of the synchronizing FIFO buffers between bus and agent
11	rel_agent_freq_g : integer := 1;	Synchronization	Clock domains	Number	positive integer	Relative frequencies of IP and bus, Needed at least for synchr. multiclk FIFOs
12	rel_bus_freq_g : integer := 1;	Synchronization	Clock domains	Number	positive integer	see above
13	addr_g : integer := 46;	Functional	Addressing	Number	positive integer	unique address for each wrapper
14	inv_addr_en_g : integer := 0;	Functional	Addressing	Number	0 or 1	only for bridges, other half uses 0 and the other 1
15	multicast_en_g : integer := 0	Functional	Addressing	Number	0 or 1	enable special addressing
16	n_agents_g : integer := 4;	Functional	Arbitration	Number	positive integer	total number of agents within one segment (distributed arbitration requires this)
17	prior_g : integer := 2;	Functional	Arbitration	Number	less than or equal n_agents	unique priority value for all wrappers within one segment
18	max_send_g : integer := 50;	Functional	Arbitration	Number	in words, 0 means unlimited	max words the wrapper can reserve bus
19	n_time_slots_g : integer := 0;	Functional	Arbitration	Number		Number of time slots in a TDMA frame. TDMA is enabled by setting n_time_slots > 0. Ensure that all wrappers in a segment agree on arb_type, n_agents, and n_slots. Max_send can be wrapper-specific.
20	arb_type_g : integer := 0;	Functional	Arbitration	Number	0 round-robin, 1 priority, 2 combined, 3 DAA	Arbitration type
21	keep_slot_g := 1	Functional	Arbitration	Number	For TDMA: 0 release unused time slots 1 keep unused	Keep reserved but unused slots in TDMA. **Not used in HIBI revision r3**
22	id_g : integer := 5;	Func/Struct	Reconfiguration	Number	positive integer	unique wrapper identification for reconfiguration
23	id_width_g : integer := 4;	Func/Struct	Reconfiguration	Number	greater than or equal(log2(id_g))	wrapper identification size = max number of wrappers
24	base_id_g : integer := 5;	Func/Struct	Reconfiguration	Number	positive integer	only for bridges, which cfg id are routed acrossa the bridge
25	cfg_re_g : integer := 0;	Func/Struct	Reconfigur	Number	0 or 1	enable reading configuration memory
26	cfg_we_g : integer := 0;	Func/Struct	Reconfigur	Number	0 or 1	enable writing configuration memory
27	n_extra_params_g : integer := 0;	Func/Struct	Reconfigur	Number	positive integer	Number of app-specific extra registers
28	n_cfg_pages_g : integer := 1;	Func/Struct	Reconfiguration	Number	1,2,3...	Number of configuration pages. Having multiple pages allows fast reconfig. Note that cfg memory initialization is done with separate package if you have many time slots or configuration pages

- Enable/disable multicast functionality
- Enable/disable runtime configuration functionality (affects structure=area as well)

Table 5 lists all the generics. Certain parameters are system-wide settings, for example the width of the command. Some are segment-wide, for example bus clock, data width, and number of wrappers in that segment. The rest are instance-specific, for example buffer sizes and priorities.

9.2 Clocking

HIBI can support may clock domains. The border is either between IP and wrapper, or in the middle of a bridge. There are five options:

1. Fully synchronous
2. Synchronous multi-clk: Clock frequencies are integer-multiples of each other. Clocks are in the same phase. Easy to use with FPGA's PLLs
3. GALS: No assumptions about relations (phase, speed) between clocks. Has longer synch. latency than synch.multiclock.
4. Gray FIFO: FIFO depth limited to power of two ($= 2^n$)
5. Mixed clock pausible

The method must be decide at synthesis time.

9.3 Runtime reconfiguration

Wrapper has config memory that stores all information for distributed arbitration. It can be synthesized in many ways:

- Permanent: ROM, 1 page
- Partial run-time configurable: ROM with several pages
- Full run-time configurable: RAM, with pages
- Kactus supports currently 1-page ROM

HIBI allows the runtime configuration of all arbitration parameters to maximize performance. This is achieved so that one of the agents (e.g. system controller CPU) writes the new configuration values to all wrappers. The configuration values are sent through the regular data lines. During the normal operation, i.e. when the configuration is not changed, the controller CPU can perform its computation tasks. In the best case, other PEs can continue their transfers even if HIBI is being configured. However, some operations, such as swapping priorities of two wrappers, necessitate disabling other transfers momentarily.

The structure of the configuration memory is illustrated at the bottom of Fig 8. It includes multiple configuration pages for storing the parameter values, a register storing the number of currently active page, clock cycle counter, and logic that checks the start and end of times of the time slots. The receive controller takes care of writing new configuration values whereas the configuration values and time slot signals are fed to the transfer controller. Configuration values can be written to non-active pages before they are used to minimize the risk of conflict when the configuration is performed.

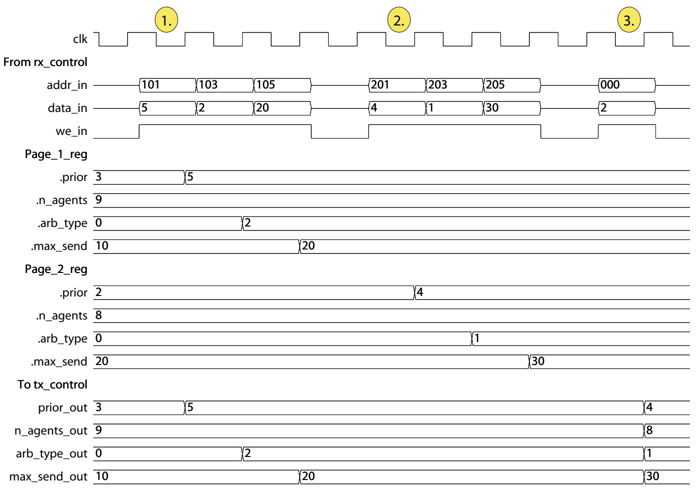


Figure 19: Example of runtime configuration

For very regular traffic, the TDMA slots can be set to minimize the latency, i.e. slot starts shortly after the availability of data. For TDMA, each wrapper has an internal cycle counter to decide correct times to access the bus. For this reason, wrappers in one bus segment must be synchronized. When data is produced with varying time intervals or quantities, the time slots cannot be optimally located. By runtime reconfiguration, the cycle counters can be reset to an arbitrary clock cycle value within the time frame to keep time slots in the correct place with respect to data availability. Also the length and owner of the slots can be changed. The resynchronization can be triggered explicitly from software or automatically by a specific monitor unit, which monitors how effectively time slots are used and starts the reconfiguration if needed [1]. Roughly 10 % improvement in HIBI v.1 throughput in video encoding due to dynamic reconfiguration was reported in [7]. Larger gains are expected when several applications are executed on a single platform. Reconfiguration was used in [4] to speed-up the exploration on FPGA. It allowed notably less synthesis runs, each of which took several hours.

As a new feature in HIBI v.2, the second-level arbitration method can be changed at runtime between priority and round-robin or both of them can be disabled. When the second-level arbitration is disabled, only the basic TDMA is used and the slot owner reserves the bus always for the whole allocated time slot. Similarly, only the second-level arbitration is utilized when no time slots are allocated.

In HIBI v.2, three methods are used to improve the configuration procedure. First, by making use of the bus nature, each common parameter can be broadcast to all wrappers. Second, enabling the reading of configuration values simplifies the procedure as the whole configuration does not have to be stored in the configuring agent. In contrast, the configuring agent can read the old parameter values to help determining the new ones. Third, additional storage capacity for multiple parameter pages has been added to enable rapid change of all parameters. When a configuration page changes, all the parameters are updated immediately with one bus operation. It is possible to store a specific configuration for every application (phase) in its own configuration page to enable fast configuration switching.

Runtime reconfiguration is illustrated in Fig 19 for 2-page configuration memory. Signals coming from receive controller to configuration memory (*addr_in, data_in, we_in*) are shown on top. In the middle are the registers *.prior, .n_agents, .arb_type, .max_send* for both configuration pages (all parameter registers are not shown for clarity). On the bottom, are the signals from memory to transfer controller (*prior_out, n_agents_out, arb_type_out, max_send_out*). In the example, the first digit of the address defines the page and two last digits define the parameter number.

1. The parameter registers for priority (*.prior*), arbitration type (*.arb_type*), and maximum send amount (*.max_send*) on current page (page 1) are configured to values 5, 2, and 20, respectively.

2. Parameters on the inactive page are updated: priority is set to 4, arbitration type is changed from round-robin (0) to priority (1), and max_send is increased to 30.

3. Page 2 is activated by writing value 2 to address 0x000. When the page is changed, all outputs to transfer controller change immediately. Since the number of agents (*n_agents*) changes to value 8, the wrapper with priority 9 cannot access the bus anymore. This way arbitration latency can be decreased if some agent is known to be idle.

10 Performance and resource usage

10.1 HIBI wrapper structure

The resource usage of the HIBI comes mainly from it's wrappers. HIBI version 3 has three types of them which include R1, R3 and R4. Figure **??** shows how a R3 wrapper is constructed of multiplexors and a R1 wrapper which has four separate FIFOs itself.

10.2 Resource usage

The resource usage for invidual HIBI wrappers was acquired from a SoC that was synthesized to a Arria II GX FPGA on a Arria II GX development board. The SoC had two HIBI components with both attached to a R3 HIBI wrapper. The size of the fifos on these wrappers was set to 4 words which means $4 \cdot 32b = 128b$ on each fifo.

Table 6 shows the combinatorial ALU (adaptive LUT) counts and register counts of a wrapper. Both minimum and maximum values are reported since synthesis does

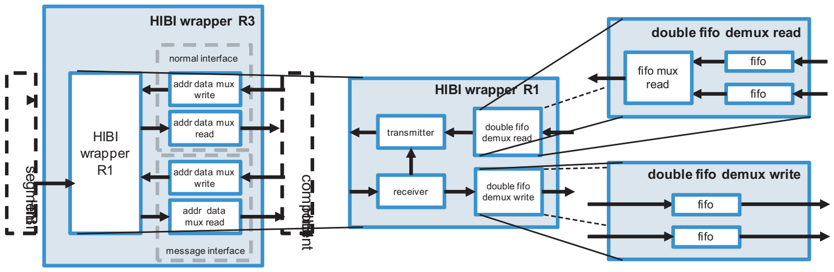


Figure 20: HIBI R3 wrapper block diagram

Table 6: Resource usage of wrapper R3, with 32b data, multiplxed address and 5b command. v.2 and v.3

Wrapper subblock	Unit	Value
HIBI wrapper r3	comb. ALUTs	724-763
	registers	1029-1168
HIBI wrapper r1	comb. ALUTs	466-533
	registers	825-935
4-word FIFO	comb. ALUTs	76-104
	registers	155-167

not always produce exactly the same results. Area can be significantly reduced if the FIFOs are implemented as onchip memories (m9k blocks in Arria II GX).

Fig. **??** shows the resource usage layout on the FPGA as seen on the Chip Planner in Quartus II. The two wrappers are highlighted in blue.

10.3 Simulated performance

The throughput was measured for a 32 bit, 200 MHz HIBI segment with two components, both of which were connected to the segment with a R3 wrapper. The sender transmitted a continous stream of 1024 words to a single address. Maximum throughput is $200MHz \cdot 32b =$ 800 MByte/s. Since the data and address are buses muxed together, the minimum time to send the stream would be 1025 cycles. Measured latency and throughput are shown in Fig.**??**. Both approach their theoretical limits as the FIFO depth increases.

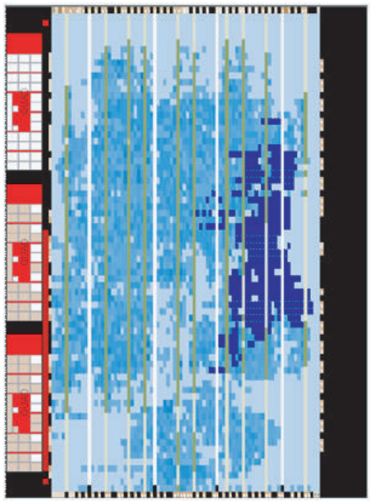


Figure 21: HIBI R3 in Quartus' chip planner tool

11 Usage examples

IP can connect directly to HIBI but CPUs should use a DMA. It allows performing transfers on the backgournd while CPU is processing.

11.1 Transmission with dual-port memory buffer and DMA controller

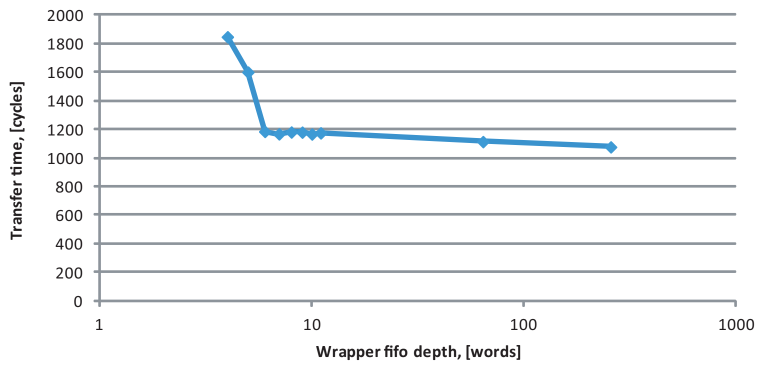
Fig. 23 shows the concept how CPU can send data using DMA.

1. CPU reserves buffer space from dual-port memory
2. CPU copies/writes data to dual-port memory
3. CPU configures DMA transfer: memory address, size of transfer, and destination IP-block's HIBI address (not local CPU address)
4. DMA reads data from dual-port memory and sends the data to the configured HIBI address

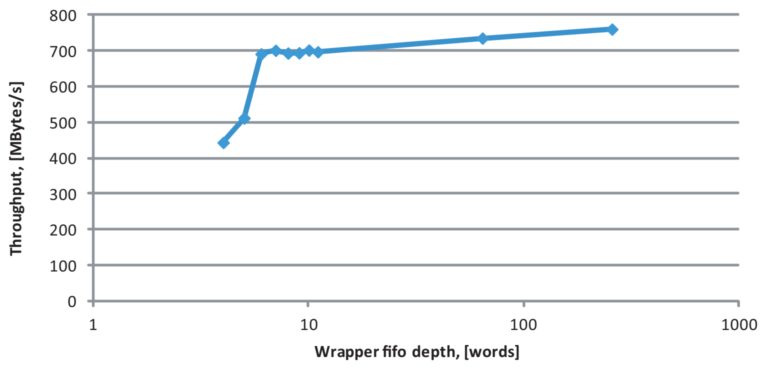
11.2 Reception with dual-port memory buffer and DMA controller

Fig. 24 shows the concept how CPU can use DMA to copy received data into the local dual-port memory.

1. CPU reserves buffer space from dual-port memory



(a) Transfer latency in cycles. Theoretical miniumum 1025 cycles (one cycle needed for address)



(b) Throuhgpput in MB/s. Theoretical max 800 MB/s

Figure 22: Performance with 1024-word transfers.

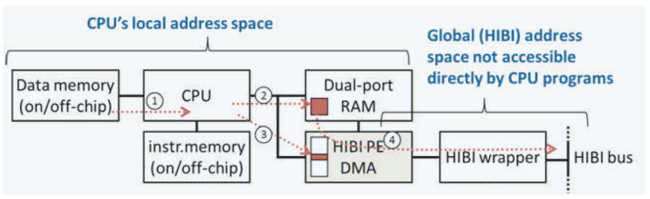

CPU's local address space
Global (HIBI) address space not accessible directly by CPU programs
Data memory (on/off-chip)
CPU
Dual-port RAM
instr.memory (on/off-chip)
HIBI PE DMA
HIBI wrapper
HIBI bus
1
2
3
4

Figure 23: Example how CPU sends using DMA.

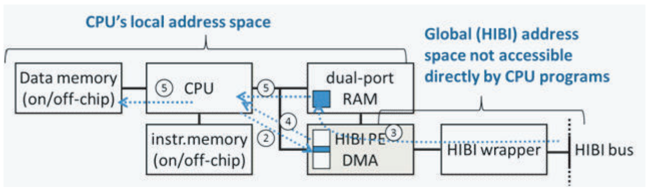

CPU's local address space
Global (HIBI) address space not accessible directly by CPU programs
Data memory (on/off-chip)
CPU
dual-port RAM
instr.memory (on/off-chip)
HIBI PE DMA
HIBI wrapper
HIBI bus
1
2
3
4
5

Figure 24: Example how CPU receives data usign DMA.

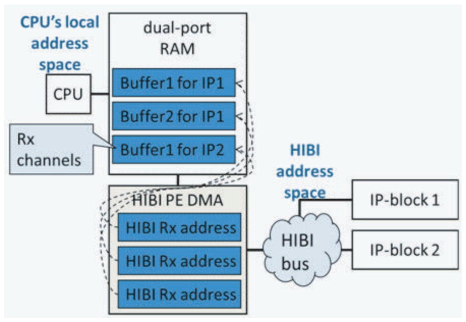


Figure 25: Example mapping between incoming address and buffer in dual-port memory.

2. CPU configures DMA: Memory address, size of transfer, and the HIBI address in which data is received
3. DMA copies the incoming data to DPRAM
4. DMA interrupts CPU when a configured number of words have been received
5. CPU knows that data is ready in memory and uses it/copies to data memory

Rx buffers are organized as channels. Fig. 25 shows how DMA translates incoming HIBI addresses into addresses in the local memory. Only memory space limits how many buffers (channels) exists at the same time. Channels have implicit meanings that must be agreed:

1. Who (what IP-block or CPU) sends data to which channel, since otherwise the sender is not known (HIBI does not send sender ID in transfers).
2. Possible explicit meaning of channel like "DCT transform Q-parameter". Then, it is not that relevant who provides data.

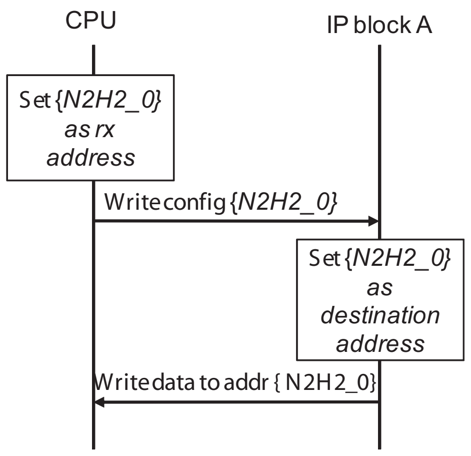

CPU
IP block A
Set {N2H2_0} as rx address
Write config {N2H2_0}
Set {N2H2_0} as destination address
Write data to addr { N2H2_0}

Figure 26: Example how CPU instructs the IP block where to put result data.

11.3 Example: use source specific addresses

Designer wished to implement following high-level sequence “HW IP-block A should send data to CPU after initialization”. The procedure to achieve this is

1. CPU Sets rx buffer address to its DMA block N2H2_0
2. CPU sends that same address to A’s IP-block specific configuration register
3. IP A knows now to where send data
4. CPU knows from where data is coming to address

It is assumed that CPU and IP A know the data amount at design time. Otherwise, it must agreed upon during initialization (that was omitted for clarity).

11.4 SW interface to DMA

There are low-level SW macros available that access the hardware registers of HIBI PE DMA (abbreaviated as HPD). They implement a driver, but can be also used from user programs.

Notes: “HPD” is HIBI PE DMA (previously called Nios-to-HIBI 2, N2H2). “Base” is the base address of HIBI PE DMA in HIBI address space. “Amount” is data amount in 32-bit words.

HIBI_TX checks that previous send operation is complete and Calls HPD_send macro. Hence, it also runs macros HPD_TX_ADDR, TX_AMOUNT, HIBI_ADDR, TX_COMM, and TX_START Releases the Tx channel.

Following example shows a data transfers between two CPUs assuming the system in Fig. 27(a). Fig. 27(b) shows the sequence diagram.

Table 7: The SW macros for accessing the DMA controller's registers

Macro	Meaning
void HPD_CHAN_CONF (int channel, int mem_addr, int rx_addr, int amount, int* base)	Configure HPD channels. After configuration, specific channel is ready to receive amount of data to rx_addr HIBI address. Received data is stored to mem_addr in HPD address space.
void HPD_SEND (int mem_addr, int amount, int haddr, int* base)	Send amount of data from mem_addr to haddr HIBI address. mem_addr is memory address in HPD address space.
void HPD_READ (int mem_addr, int amount, int haddr, int* base)	Send command to read amountof data from haddrHIBI address.
void HPD_SEND_MSG (int mem_addr, int amount, int haddr, int* base)	Send amount of data from mem_addr to haddr HIBI address as HIBI message. mem_addr is memory address in HPD address space.
int HPD_TX_DONE(int* base)	Returns status of transmit operation.
void HPD_CLEAR_IRQ(int chan, int* base)	Clears IRQ of specific channel.
int HPD_GET_IRQ_CHAN(int* base)	Return the number of the channel that caused interrupt. If interrupt hasn't occurred, return -1.

12 Summary

The most important properties of HIBI are summarized in Table. 9. HIBI network allows multiple topologies and utilizes distributed arbitration. The network is constructed by instantiating multiple wrapper components and and connecting them together. The wrapper is modular allowing good parameterization at design time and possibility to reconfigure certain parameters of the network runtime.

References

[1] T. Kangas, V. Lahtinen, K. Kuusilinna, and T. Hämäläinen, "System-on-chip communication optimization with bus monitoring," in *DDECS*, Apr. 2002, pp. 304–309.

Table 8: The SW functions for using the DMA

Function	Meaning
void HIBI_TX (uint8* pData, uint32 dataLen, uint32 destAddr, uint8 commType)	Send data over HIBI. pData is pointer to data, dataLen is length of the data in bytes, destAddr is destination HIBI address, commType is either HIBI_TRANSFER_TYPE_DATA or HIBI_TRANSFER_TYPE_MESSAGE. Differences to lower level macros are the automatic copying of memory to HIBI PE DMA-buffer and protection against simultaneous sending in different threads.
struct sN2H_ChannelInfo* N2H_ReserveChannel(int32 bufferSize, void* callbackFunc, bool handleInDsr, bool calledFromDsr, sint32 channelNum)	Reserve a channel for receiving data. bufferSize Size of the data to be received (bytes). callbackFunc: Function to call when the data arrives. Prototype: function(uint8* pData, uint32 dataLen, uint32 receivedAddr) handleInDsr: Set to false calledFromDsr: Set to false channelNum: Channel that is waiting for incoming data. The complete address will be HIBI base address + channelNum. Difference to lower level macros is that interrupt handler provided by HIBI driver, own function can be registered directly to handle data.

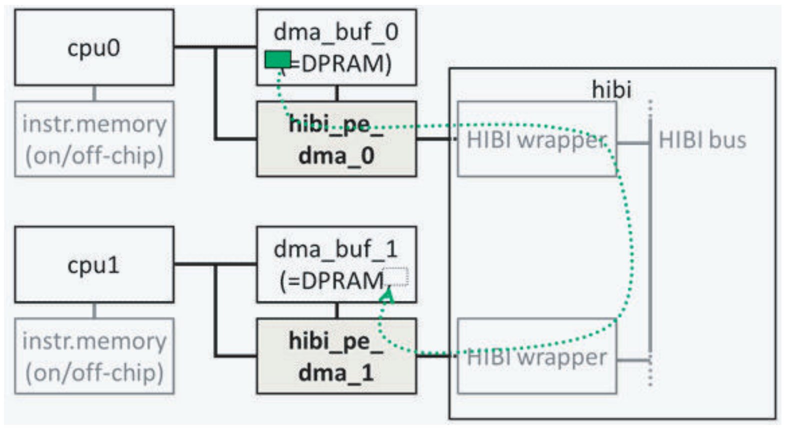
[2] A. Kulmala, T. D. Hämäläinen, and M. Hännikäinen, "Comparison of GALS and synchronous architectures with MPEG-4 video encoder on multiprocessor system-on-chip FPGA," in *Euromicro DSD*, Sep. 2006, pp. 83–86.

[3] A. Kulmala, M. Hännikäinen, and T. D. Hämäläinen, "Reliable GALS implementation of MPEG-4 encoder with mixed clock FIFO on standard FPGA," in *FPL*, Aug., pp. 495–500.

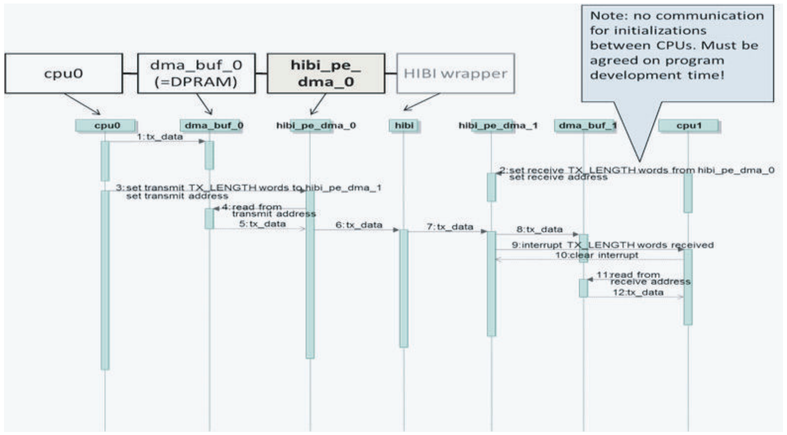
[4] A. Kulmala, E. Salminen, and T. D. Hämäläinen, "Distributed bus arbitration algorithm comparison on FPGA based MPEG-4 multiprocessor SoC," *IET Computers and Digital Techniques*, vol. 2, no. 4, pp. 314–325, Jul. 2008.

Table 9: Properties of HIBI v.3

	Property	**HIBI v.3**
Basics	Description language	VHDL, SystemC
	Topology	Hier.bus
	Interface	HIBI-specific OCP (TLM1, TLM2)
	Clocking	Synchronous wrapper GALS network
	Switching type (within segment)	Circuit-switching
	Switching type (between segments)	Wormhole (packet)switching
Configuration	Configuration	Design-time, runtime
	Design-time configurable parameters	Data width, addr width, FIFO sizes, address mutliplexing, initial configuration, addresses, clocking style Number of config pages and their type (RAM/ROM)
	Run-time configurable parameters	TDMA cycle and slots, max send, own priority Current TDMA clk cycle Utilized arbitration algorithm Change configuration page, change configuration contents (of RAM)
Transfers	Burst transfers	All transfers are bursts
	Data priority	2-level: regular and high-prior
	Commands	17 idle write: data (lo/hi-prior), config, conditional (lo/hi), non-post data (lo/hi) read request: data (lo/hi-prior), config, linked (lo/hi) exclusive access: lock, write, read, release
Signals	System signals	Clk, rst_n
	Bus signals	Data [n-1:0] (*) may contain address as well Addr valid Command [4:0] Lock Full
	IP signals	Data, addr valid, cmd, we, full, re, empty, optionally one_p + one_d
	Address lines	Multiplexed or concatenated with data
	Signal type	Unidirectional, all shared
	Signal resolution	OR-based
Flow control	Arbitration algorithms	Round-robin, DAA, TDMA, priority, random, combination
	Arbitration implementation	Distributed Pipelined
	Handshaking	1 hadnshake signal, RX buffering reserved at application level
	Qos	TDMA, round-robin/prior with limited tx length Multiple priorities for data Fast runtime configuration TDMA synchronization
Usage	Verification	HW sim, HW/SW sim FPGA prototypes
	Articles	> 10 conference, >5 journals
	Test applications (simulation)	10-CPU H.263 video encoder Synthetic test cases WLAN baseband
	Test applications (FPGA)	H.263 H.263 + WLAN (FPGA) MPEG-4, up to 35 CPUs + accelerators



(a) IP sends.


cpu0
dma_buf_0 (=DPRAM)
hibi_pe_ dma_0
HIBI wrapper
Note: no communication for initializations between CPUs. Must be agreed on program development time!
cpu0
dma_buf_0
hibi_pe_dma_0
hibi
hibi_pe_dma_1
dma_buf_1
cpu1
1:tx_data
2:set receive TX_LENGTH words from hibi_pe_dma_0 set receive address
3:set transmit TX_LENGTH words to hibi_pe_dma_1 set transmit address
4:read from transmit address
5:tx_data
6:tx_data
7:tx_data
8:tx_data
9:interrupt TX_LENGTH words received
10:clear interrupt
11:read from receive address
12:tx_data

(b) IP receives data

Figure 27: Examples of timing at IP interface.

[5] K. Kuusilinna, T. Hämäläinen, P. Liimatainen, and J. Saarinen, “Low-latency interconnection for IP-block based multimedia chips,” in *PDCN*, Dec. 1998, pp. 411–416.

[6] V. Lahtinen, “Design and analysis of interconnection architectures,” Ph.D. dissertation, Tampere University of Technology, Jun. 2004.

[7] V. Lahtinen, K. Kuusilinna, T. Kangas, and T. Hämäläinen, “Interconnection scheme for continuous-media systems-on-chip,” *Microprocessors and Microsystems*, vol. 26, no. 3, pp. 123–138, Apr. 2002.

[8] *Open Core Protocol Specification, Release 2.0*, OCP-IP Alliance, Portland, OR, 2003.

[9] E. Salminen, “On design and comparison of on-chip networks,” Ph.D. dissertation, Tampere University of Technology, 2010.

[10] E. Salminen, K. Kuusilinna, and T. Hämäläinen, “Comparison of hardware IP components for system-on-chip,” in *Intl. Symposium on Soc*, Tampere, Finland, Nov. 2004, pp. 69–73.