E-bone interconnect specification Version 1.3

Table of Contents

1 Overview

E-bone specification defines the interconnection between some master cores (Intellectual Property block) to a collection of other slave cores. It targets FPGA environments for two main purposes.

- Simple control applications, under some (remote) Host computer supervision.
- Dumping large data memory down to the Host.

E-bone specifications cater for both the Control Interconnect (a bidirectional path, usually 32 bit wide) and the Fast Transmitter bus (up to 256 bit wide one way path).

E-bone Control Interconnect data path may itself be extended up to 256 bits in width to cope with demanding data readout systems.

The objectives of the E-bone specifications are summarized thereafter.

- E-bone defines a common core interface to promote design re-use.
- E-bone enforces simple yet burst oriented implementation.
- E-bone is scalable. It can be used for simple control and for complex data readout sub-systems, or both simultaneously.
- Control Interconnect data granularity is 32 bits in width, not less.
- A fast unidirectional path allows dumping data at full speed to the Host, with 64 bit addressing down to the byte location.
- All synchronous to a single system clock.
- E-bone is applicable in a single FPGA or crossing FPGA boundaries.

A typical E-bone application is for interfacing to a PCI-express root complex. Some Endpoint Interface bridges the PCI-express data link to the E-bone interconnection.

- The PCIe Endpoint Interface operates as an E-bone master.
- Often a large memory needs to be dumped as fast as possible to the root complex. A Direct Memory Access Controller (DMAC) is generally used. The Endpoint also incorporates a slave hooked on the E-bone interconnection.

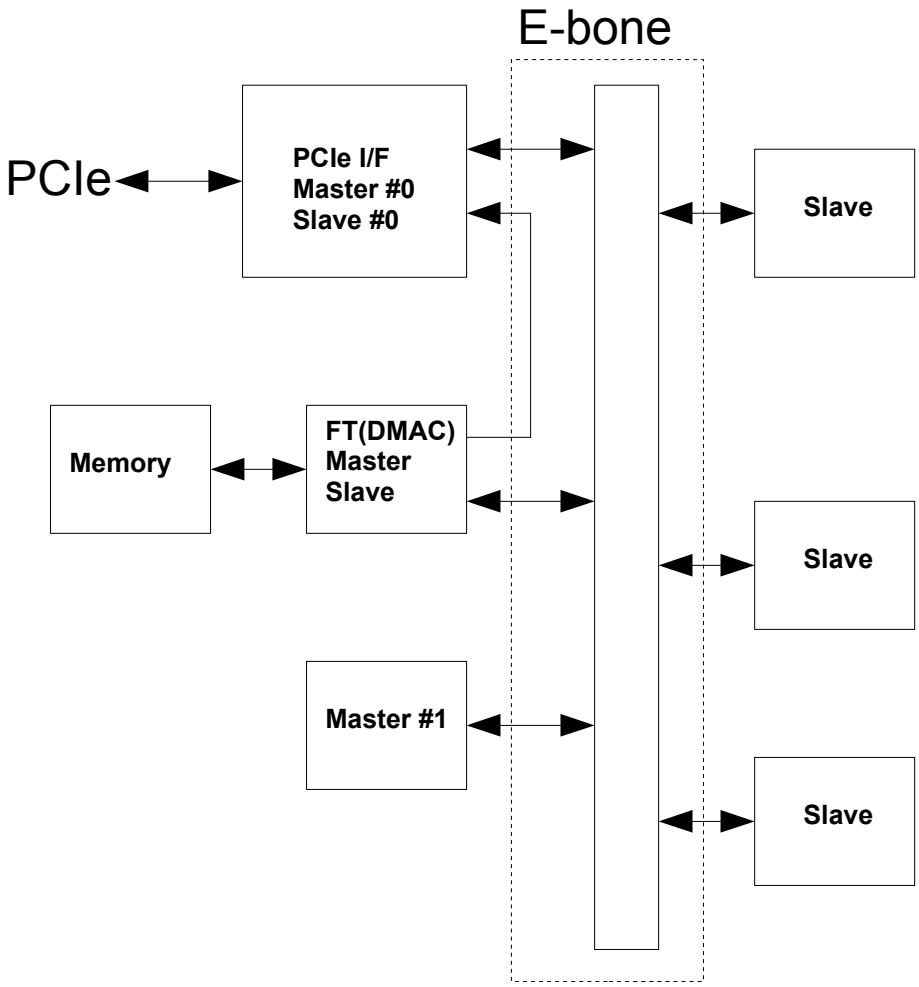


Fig. 1 : E-bone architecture example

Comments.

- PCIe Endpoint Interface is the E-bone primary master.
- The Fast Transmitter is a specific E-bone master that drives the PCIe Endpoint.
- The Fast Transmitter is controlled via the general purpose E-bone Interconnect, as any slave.
- The number of masters and slaves are application dependent. Typical E-bone Interconnect usually supports at least 2 masters (including the Fast Transmitter).
- A specific core, the E-bone Manager provides core features like arbitrating, routing and monitoring.

Chapter 2 describes the Control Interconnect.
Chapter 3 addresses the Fast Transmitter bus.
Chapter 4 deals in some more details with the E-bone implementation.
Chapter 5 addresses the usage of E-bone with extended features for data readout sub-systems.

2 Control Interconnect signals and protocol

2.1 Signal description

Regarding the naming conventions, signals dedicated to (mostly those originating from) a master start with *m#_* , where # is a number. In this chapter we a focusing on the primary master (#0). We shall refer to master #0 as the Endpoint (but this does not restrict E-bone to PCI-express). All signal names are prefixed by the *eb_* string.

Depending on the direction at the core ports the suffixes _i (input) and _o (output) are appended to the signal names. There are no bidirectional signals.

Master	Slave	Size	Description
eb_m0_clk_o	eb_clk_i	1	Clock from the Endpoint
eb_m0_rst_o	eb_rst_i	1	Clock synchronous reset from the Endpoint
eb_m0_brq_o	NC	1	Master bus request, asserted for the whole bunch transfer
eb_m0_bg_i	NC	1	Bus granted by the E-bone manager.
eb_bmx_i	eb_bmx_i	1	Bus busy (any m# but FT) asserted by the E-bone manager.
eb_m0_dat_o	eb_dat_i	32 (*)	Multiplexed Type/Offset/Data write bus
eb_m0_as_o	eb_as_i	1	Address Strobe
eb_m0_eof_o	eb_eof_i	1	End of Frame, asserted when last data
eb_m0_aef_o	eb_aef_i	1 (**)	Almost End of Frame, asserted ahead of *eb_eof*
eb_dat_i	eb_dat_o	32 (*)	Data read bus
eb_dk_i	eb_dk_o	1	Address/Data acknowledge
eb_err_i	eb_err_o	1	Error, transfer cancelled

(*) May be larger (64, 128 or 256) in some implementations, see chapter 5.

(**) To help in data pipeline management. May be ignored by most simple slaves.

2.2 E-bone burst sequence

2.2.1 Requesting E-bone mastership

Before placing any request, a master must check that E-bone bus is not busy. That means *eb_bmx* and *eb_ft_bg* (FT dedicated busy status, described in chapter 3) are not asserted. Then the master initiates a burst transfer by asserting *eb_m#_brq* to request bus mastership. The E-bone Manager asserts *eb_m#_bg* thus granting the bus to this dedicated master. The Manager also assert *eb_bmx*, the E-bone busy status signalling to all slaves.

The master must keep *eb_m#_brq* asserted for the whole burst transfer, thus locking the transaction.

When the burst is done *eb_m#_brq* must be released *for at least two clock cycles* before it can be re-asserted.

When a master is denied (not granted) the bus access *within the next clock cycle*, he must withdraw his request and wait for *eb_bmx* and *eb_ft_bg* not asserted before retrying.

Master #0 is assigned the highest priority, whereas FT is at the lowest priority.

The E-bone Manager implementation is not addressed by this specification. Usually arbitration is done through a fast asynchronous loop.

All slaves must check the *eb_bmx* signal to differentiate the plain masters from the FT bursts (see chapter 3).

2.2.2 Addressing phase

A transfer continues with a Type/Offset descriptor word presented on the write bus and *eb_as* (Address Strobe) is asserted. The descriptor is formatted as follows.

- One direction bit (read or write).
- Encoded 2-bit segment base address selectors (1, 2 or 3 corresponding to three PCIe base addresses; segment zero is a special case, actually reserved for the internal design of the EndPoint in the case specific of PCI-express).
- The E-bone address offset within the segment.

31	30	29-28	27-0
R=1/W=0	Reserved	Segment #	Offset

Then, in case of a write transfer, data follows. In case of a read transfer data are presented by the addressed slave on the read bus.

The addressing range are determined by generic constants common to all cores hooked on the E-bone. The segment addressing ranges are implementation dependent, but limited to 28 bits *words* (whatever the number of bytes per word, as it will be described in the E-bone extension).

The addressing phase ends after a slave asserts *eb_dk* and/or *eb_err*.

***eb_dk* asserted, *eb_err* not asserted: switch to data phase.**
When a slave decodes its address *and* if it can manage the data flow, it asserts *eb_dk* (Data Acknowledge). In the case of a *write* burst however the master *may* further extend the addressing phase at his convenience by keeping *eb_as* asserted. The addressing phase ends when the master releases *eb_as*. In the case of a *read* burst the master *cannot* extend the addressing phase. The master must accept data (releasing *eb_as*) as soon as the slave asserts *eb_dk*.

***eb_dk* not asserted, *eb_err* asserted: error.**
Alternatively if it receives the *eb_err* signal (*eb_dk not* asserted) , a master must cancel the on-going transfer as soon as possible. *eb_err* must remain asserted until AS is de-asserted. Asserting *eb_err* alone indicates an unrecoverable error or panic status.

Both *eb_dk* asserted and *eb_err* asserted: retry.
A slave may request the master to give up for now and later retry the current burst. Retry condition is signalled by the slave asserting both *eb_dk* and *eb_err* during the addressing phase (AS still asserted).

2.2.3 Data phase

The addressing phase may require several clocks (up to the limit imposed by the Manager) from *eb_as* to *eb_dk* assertion. It is expected that this time is used by the slave for decoding the address *and* making sure that it is ready for accepting (write burst) or delivering (read burst) data without any delay.

After *eb_dk* has been asserted during the addressing phase, it cannot be withdrawn till the end of the data transfer.

The signal *eb_dk* remains asserted for at least 2 clocks cycle: the fist one to indicate that the slave is ready, the second one to transfer the first data (otherwise *eb_err* would have been asserted in the addressing phase).

The data phase starts when *eb_as* is released.

Slaves not addressed must drive zero on their outputs.

However an addressed slave may still drive the interconnection for one clock after *eb_bmx* has been de-asserted.

E-bone data granularity is always 32 bits (or more, generally the whole data bus range) in width.

A slave may cancel the current burst by releasing *eb_dk* during the data phase. Then the master aborts without retrying. The *eb_err* signal is meaningless during the data phase.

The master asserts the *eb_eof* (End of Frame) signal to flag the last data in the burst. An early version of the same signal, *eb_aef* (Almost End of Frame) is asserted one clock ahead. The purpose of the *eb_aef* signal is for helping in data pipeline management.

In the particular case of *single* data burst the *eb_aef* signal is asserted any time during the addressing phase. It is even possible for the master to assert *eb_aef* prior to DK being asserted by the slave. In that case on of single data *eb_aef* then may be asserted for more than one clock. *eb_aef* must remain asserted until *eb_eof* is asserted.

A burst cannot cross a slave boundary address.

The master may abort a burst at any time by de-asserting *eb_m#_rq* (thus releasing *eb_bmx)* and without asserting *eb_eof*.

Next table summarizes important E-bone concepts and associated benefits.

Slaves not addressed must drive zero on their outputs	Lower interconnect resources
Data granularity is whole data bus range	Simplifies design
DK asserted cannot be withdrawn	Direct connection even for pipelined systems. Once initialized the pipeline proceed without disturbance (cannot pause/restart): simple and efficient designs.

2.3 Burst write

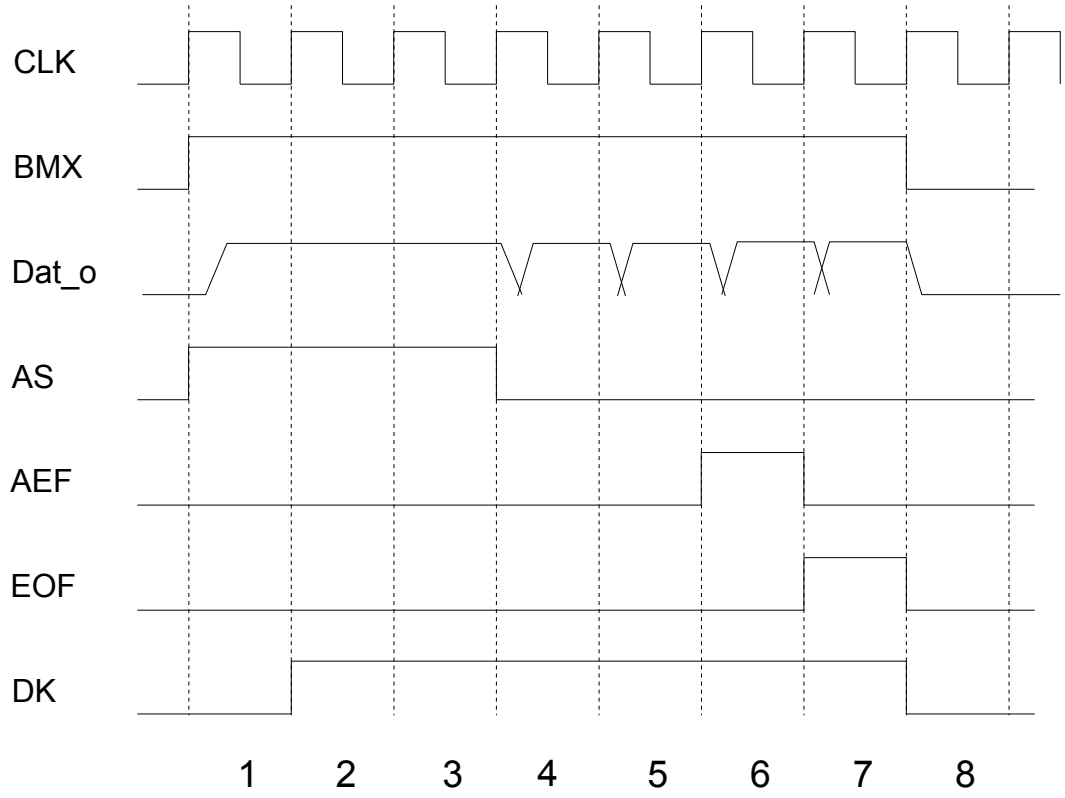


Fig. 2

0	Not shown. Master request BRQ is shortly granted BG by the Manager. So all master outputs are shortly available to the slaves. Manager concurrently asserts BMX. In case of asynchronous implementation this occurs beginning of cycle 1
1	AS is asserted. Slaves decode the Type(write)/Offset and BMX informations from the bus.
2	The addressed slave drives DK and stores the Offset LSbits for internal addressing. Slave anticipates the data burst and early asserts DK. ERR also asserted at this stage would have triggered a retry from the master.
3	Master sees DK acknowledge. But it needs an extra clock until 1st data is actually available. Therefore the master maintains AS asserted. The slave waits.
4	Master presents 1st data. AS is de-asserted. Slave stores 1st data. Since EOF is de-asserted it anticipates next data and early asserts DK.
5-6	Master presents next data. Slave stores data. A new data is available at every cycle. Slave may cancel the burst by de-asserting DK.
7	Slave stores last data. Since EOF is asserted it stops asserting DK. AEF has been asserted one clock before EOF at step 6. Master releases BRQ, so BMX gets de-asserted.
8	BMX must be de-asserted before any master can request bus mastership. In case the master would keep BMX asserted (which is not recommended), the slave must not capture any more data past the EOF signalling.

2.4 Burst read

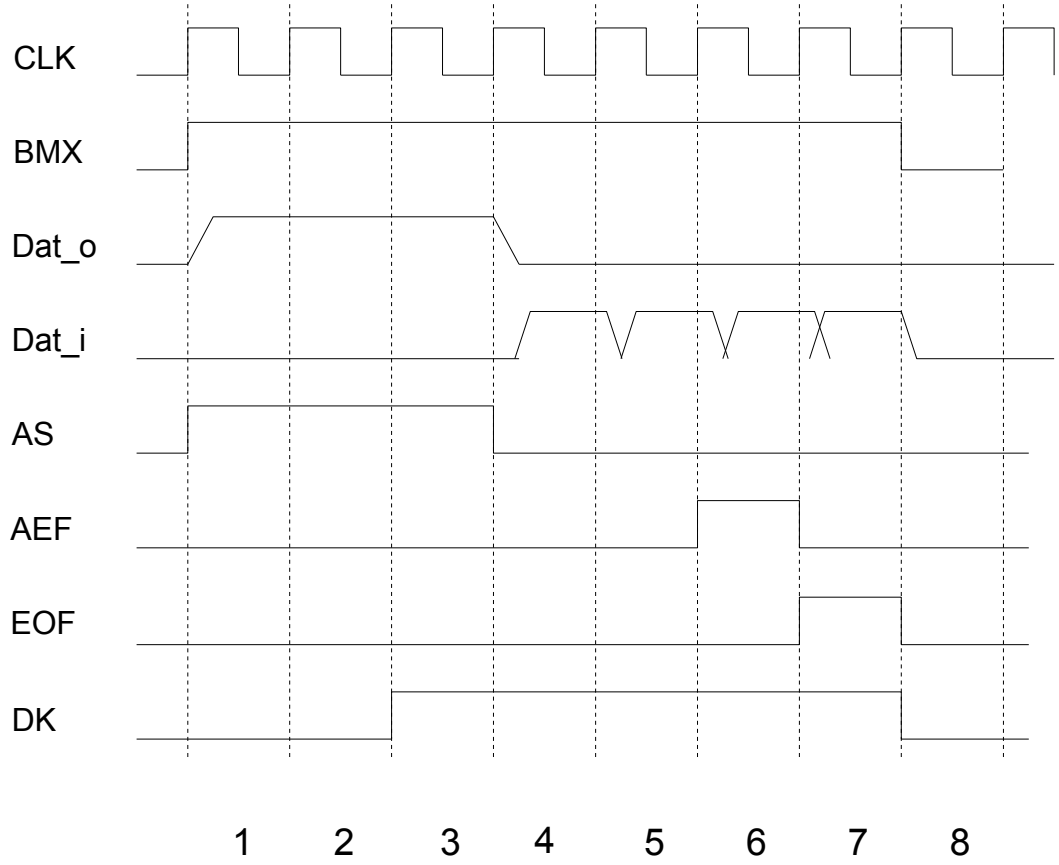


Fig. 3

1	Master BRQ/BG not shown. BMX asserted indicates a new burst. AS is asserted. Slaves decode the Type(write)/Offset and BMX informations from the bus. In this case the DK acknowledge is delayed. So master waits.
2	DK still de-asserted. So master waits.
3	The addressed slave drives DK and stores the Offset LSbits for internal addressing. Master sees DK acknowledge and switch to reading from the input data bus. ERR also asserted at this stage would have triggered a retry from the master.
4	Slave delivers 1st data. Since EOF is de-asserted it anticipates next data and early asserts DK.
5-6	Slave drives the output data bus and asserts DK. A new data is available at every cycle. Master sees DK acknowledge and capture next data. Slave may cancel the burst by de-asserting DK.
7	Slave delivers last data. Since EOF is asserted it stops asserting DK. AEF has been asserted one clock before EOF at step 6. Master releases BRQ, so BMX gets de-asserted.
8	BMX must be de-asserted before any master can request bus mastership. Slave may drive DAT_o for one cycle after BMX has been de-asserted.

Single data burst read

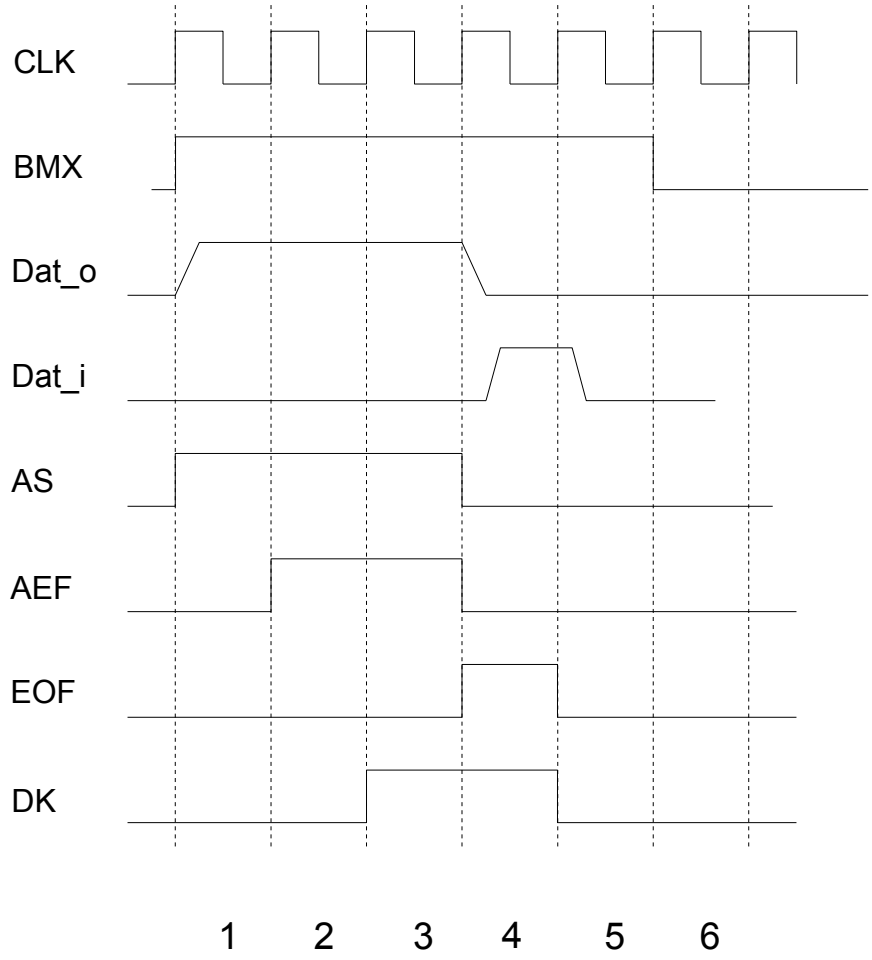


Fig. 3b

1	Master BRQ/BG not shown. BMX asserted indicates a new burst. AS is asserted. Slaves decode the Type(write)/Offset and BMX informations from the bus. In this case the DK acknowledge is delayed. So master waits.
2	AEF is asserted any time during addressing phase in the case of single data burst. DK still de-asserted. So master waits.
3	The addressed slave drives DK and stores the Offset LSbits for internal addressing. Master sees DK acknowledge and switch to reading from the input data bus. ERR also asserted at this stage would have triggered a retry from the master. AEF remains asserted until EOF is asserted.
4	Slave delivers the single data. Since EOF asserted the slave stops asserting DK.
5	Master still does not release the bus...
6	Master releases BRQ, so BMX gets de-asserted.

2.5 Broad-cast

This is a Type(write)/Offset only transfer.

Slaves recognize broad-cast when *eb_as* and *eb_eof* are asserted together during the addressing phase. All slaves are concerned.

The write bus bit 31 is cleared as required for a write transfer.

The offset field (bits 27-00) is the broad-cast message.

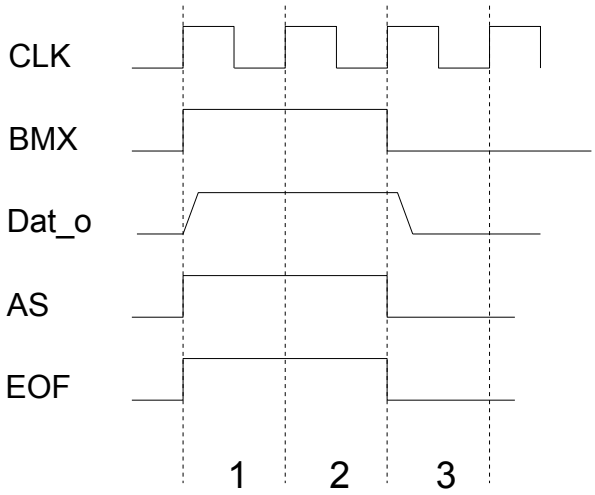


Fig. 4

The broad-cast is a fixed timing burst. All slaves must decode it without extra clock delays.

The *eb_dk* acknowledge is not actually used by the master and may or may not be generated by the slaves. The same way the *eb_err* response is ignored by the master.

The Fast Transmitter, as any E-bone master, can generate a broad-cast.

2.6 Broad-call

This is a single word read transfer.

Slaves recognize broad-cast since *eb_as* and *eb_eof* are asserted together during the addressing phase.

The write bus bit 31 is set as required for a read transfer.

Broad-call is used to implement legacy interrupts from the slaves to the Host. To this aim the Endpoint interface is required to issue E-bone broad-calls at periodic intervals. Every slave with interrupt capability is assigned a single (and exclusive) bit pattern in the answer back data word. The pattern is 16 bits in length, all zero but one bit set to one. A slave may assert its assigned pattern onto the 16 lower bits of the data bus to notify an interrupt condition.

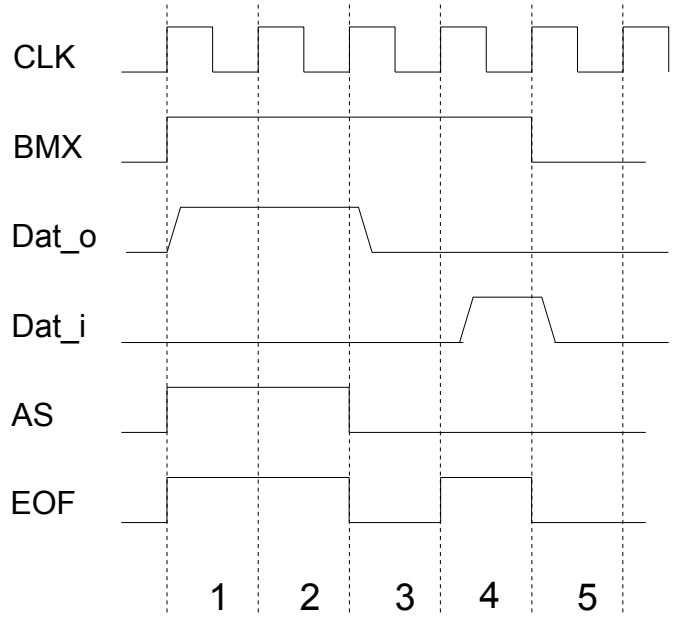


Fig.5

The broad-call is a fixed timing burst. All slaves must decode it and supply the interrupt status pattern without extra clock delays. The *eb_dk* acknowledge is not actually used by the master and may or may not be generated by the slaves. The same way *eb_err* response is ignored by the master.

The interrupt status pattern must self-clear at the completion of the broad-call, thus preventing repeated messages to the Host. Internal record must be kept of a pending interrupt until it has been actually served by the Host. It is noteworthy that the E-bone bridge manages three main interrupt requests sources. These originate

- from the periodic broad-call (addressed in this chapter);
- from the Fast Transmitter broad-cast (see chapter 3.4);
- in some case from the Endpoint internal resources, out of the scope of this specification.

3 Fast transmitter signals and protocol

3.1 Overview

The Fast Transmitter establishes a one way channel from a dedicated master (typically a DMAC) to the Endpoint Interface. It is a secondary master on the E-bone interconnect. It complies with the E-bone protocol but with some extensions and some restrictions.

- The Fast Transmitter is a write only master. It cannot initiate burst read on the E-bone.
- It profits from a dedicated data bus. Its width may be 32, 64, 128 or 256 bits.
- The first word(s) sent on the bus is a full 64 bit address to the Host (and *not* in E-bone format). When the data bus is sized to 32 bits the address most significant bit word is sent first, then the least significant bits (actually as the first data within the burst).
- Byte management is supported with specific signalling, with the burst length being announced in the beginning of the transfer. There is no *eb_aef* signalling.
- Data down-sizing is supported with specific signalling.
- The Fast Transmitter uses a dedicated line to request E-bone mastership.
- The corresponding *eb_ft_bg* grant signal is also routed to the PCIe Endpoint Interface *which is directly addressed* and must act as such, dealing with the E-bone handshaking signals.
- It is recommended to reserve Slave #0 to the dedicated Fast Transmitter receiving port in the Endpoint Interface.

3.2 Signals description

Next table lists the interface of the Fast Transmitter master port.

Transmitter	Endpoint	Size	Description
eb_ft_brq_o	NC	1	Transmitter bus request, to the E-bone Manager
eb_ft_bg_i	eb_ft_bg_i	1	FT dedicated grant from the Manager.
eb_ft_dxt_o	eb_ft_dxt_i	32, 64, 128, 256	FT dedicated data bus
eb_ft_blen_o	eb_ft_blen_o	16	Burst length in bytes
eb_ft_dsz_o	eb_ft_dsz_i	8	Data size qualifier, valid during the addressing phase is dual purpose. First validated by *eb_ft_as* it tells the number of most significant bytes to keep from the first data word: 0=all, N=keep N most significant bytes. Then validated by *eb_ft_ss* it is used for down-sizing management as follows: 0=32 bits, 1=64 bits, 2=128 bits, 3=256 bits.
eb_ft_ss_o	eb_ft_ss_i	1	Size qualifier strobe
eb_ft_as_o	eb_as_i	1	Plain E-bone, also strobe for *eb_ft_blen* and *eb_ft_dsz*.
eb_ft_eof_o	eb_eof_i	1	Plain E-bone
eb_dk_i	eb_dk_o	1	Plain E-bone
eb_err_i	eb_err_o	1	Plain E-bone
eb_bmx_i	eb_bmx_i	1	Plain E-bone, FT checks it before asserting BRQ

The usual E-bone protocol applies. The Endpoint FT interface behaves like a write only slave of the Fast Transmitter.

Data burst length is limited to the maximum payload size supported by the Endpoint, or more generally determined by the transport layer. This is system dependent.

The receiving FT slave may (or not) cross-check the assertion of the *eb_eof* signal with the *eb_ft_blen* burst length. As a master the FT may use the *eb_ft_eof* signal to generate a broad-cast.

The signal *eb_bmx* is *not* asserted thus other slaves are *not* addressed and should *not* decode the Type/Offset/Word.

Data granularity may be managed down to the single byte.

The Fast Transmitter cannot generate a broad-call since it is a write only master.

3.3 Fast transmitter burst

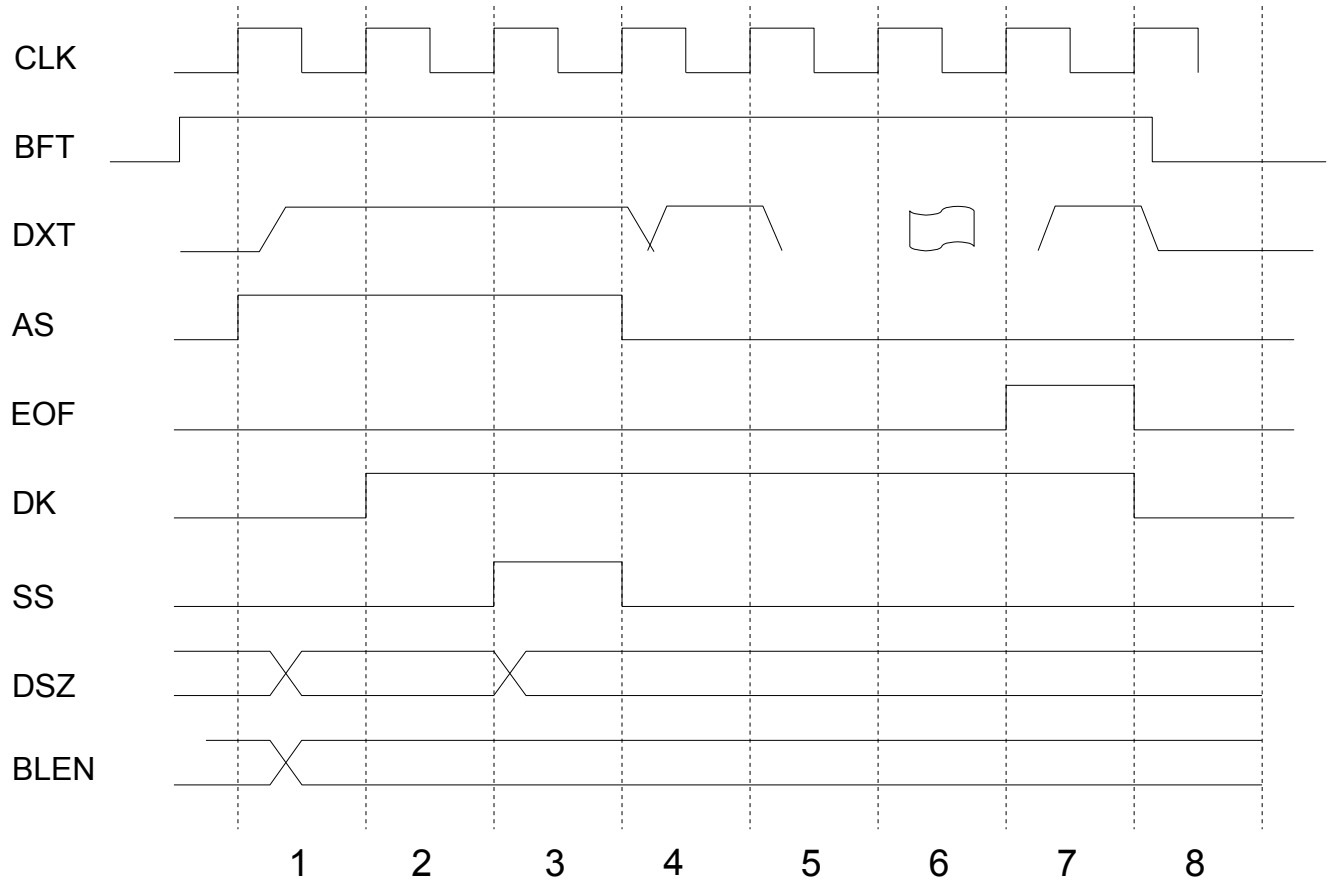


Fig. 6

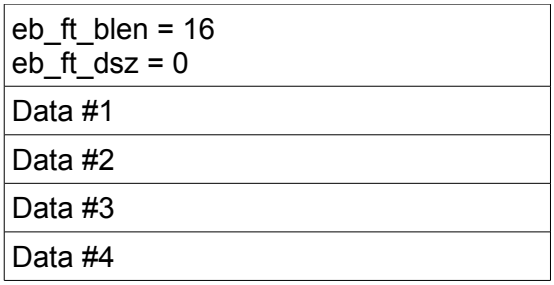
1	BRQ not shown. BFT already asserted by the E-bone Manager to grant FT bus access. AS is asserted. Endpoint knows it is addressed. Other slaves are not.
2	The Endpoint asserts DK to tell it is ready. Burst length is captured from the BLEN bus. DSZ bus tells the number of most significant bytes to be kept from the first data word. Asserting both DK and ERR would cause the FT to retry.
3	FT extends the addressing phase until it is ready to supply the data. SS is asserted. The actual data size is now valid on DSZ (0=32, 1=64, 2=128, 3=256).
4	FT delivers 1st data (in case the FT data bus is reduced to 32 bits this is the destination address LSW). AS is de-asserted. Slave starts capturing data. Slave may cancel the burst by releasing DK.
4-6	A new data is available at every cycle. Data burst length cannot exceed the PCIe payload size (expressed in FT words).
7	Endpoint gets last data. Since EOF is asserted it stops asserting DK. The BLEN information must be used to know how many bytes to keep from this last word. FT releases BRQ, so BFT gets de-asserted.
8	BFT must be de-asserted before any master can request bus mastership. In case the master would keep BFT asserted (which is not recommended), the slave must not capture any more data past the EOF signalling.

3.4 Fast Transmitter unaligned data transfer

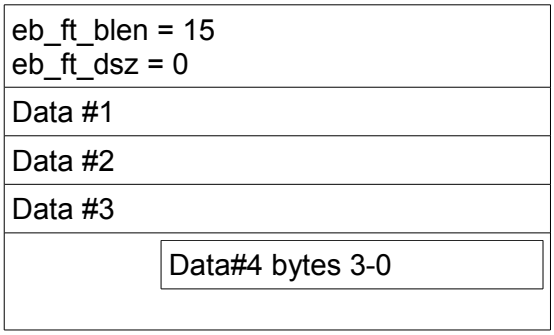
Unaligned data management examples follow. That involves the *eb_ft_blen* and the *eb_ft dsz* bus signals. Only the data source is considered. At the receiving end the destination address must be independently used for byte alignment.

Next examples are assuming a FT data width of 64 bits without down-sizing. The data word width actually fits the full FT data bus width. In all cases the *eb_ft dsz* bus therefore carries the value "01" when *eb_ft_ss* is asserted. This will not be repeated thereafter where we shall focus on the early *eb_ft dsz* bus value indicating the number of most significant bytes to keep out of the first data word.

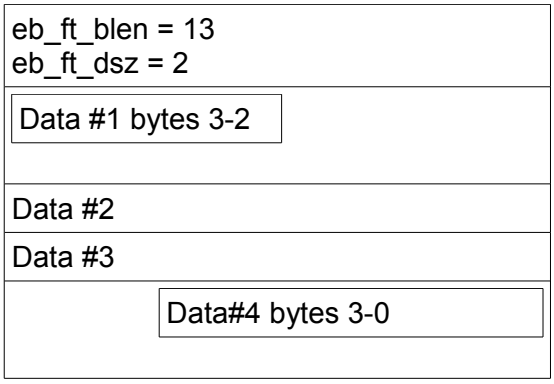
3.4.1 Source fully aligned transfer



3.4.2 Source origin aligned transfer



3.4.3 Source unaligned transfer



3.5 Fast Transmitter Interrupt signalling

The Fast Transmitter may send interrupt request to the Endpoint. It just generates a broad-cast. The Endpoint Interface decodes it since *eb_eof* is immediately asserted during the addressing phase. See Fig. 4, but *eb_ft_bg* replaces *eb_bmx*. This will usually be translated to the root complex as a slave #0 interrupt.

4 Core interconnect implementation

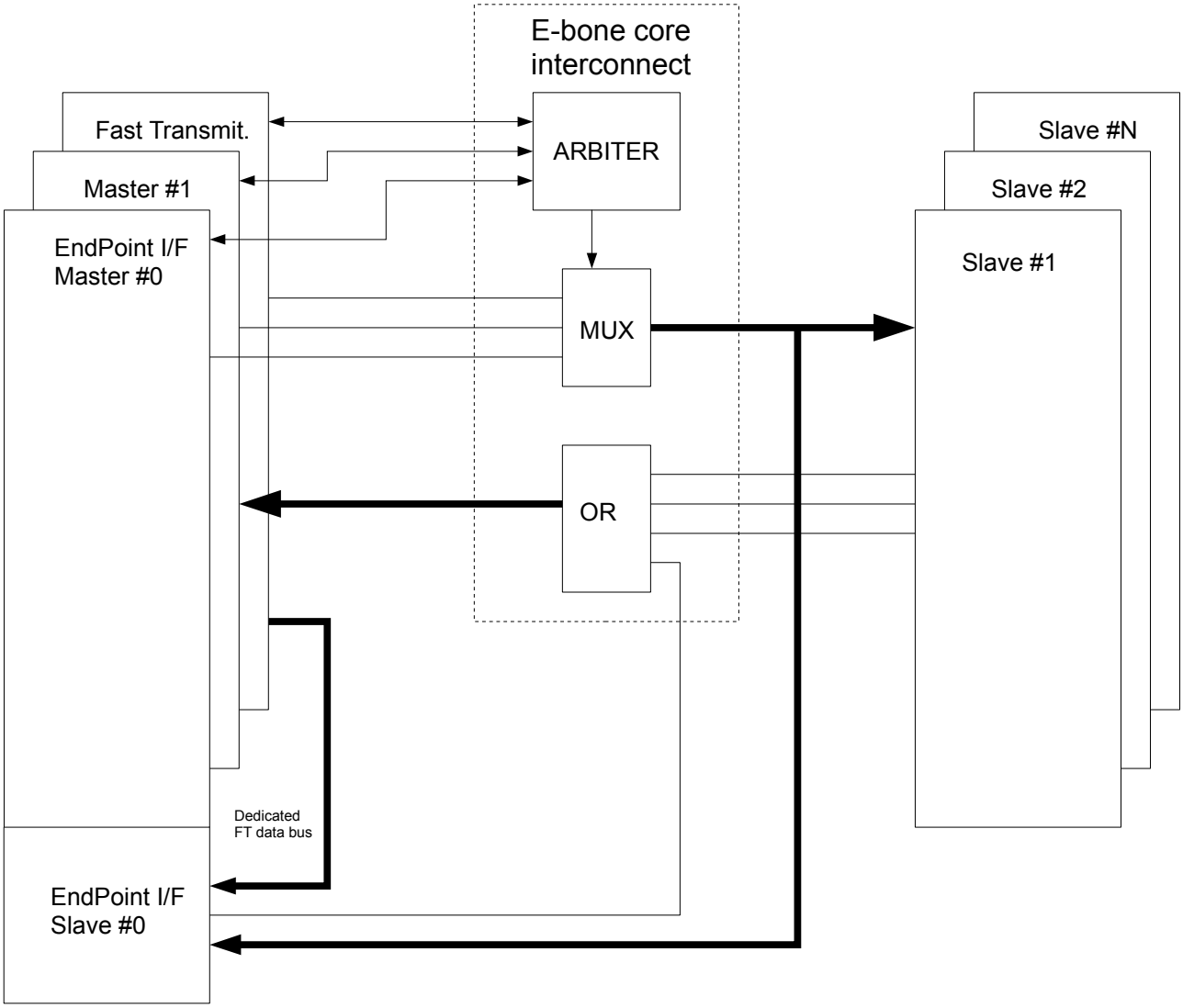


Fig. 7 : E-bone
conceptual implementation

4.1 Clock, Reset and signalling

The (master #0) Endpoint Interface generates the E-bone clock (*eb_m0_clk*) and reset (*eb_m0_rst*) signals. In case of secondary masters, their E-bone interfaces are driven by this same clock.
Reset (active high) falling edge must be clock synchronous. Reset must be active for 3 clocks minimum.
All signals are '1' when asserted, '0' when de-asserted.

4.2 E-bone manager

The E-bone manager is a required core at the basis of a complete E-bone based system. It is responsible for the following tasks.

- Arbitrate between the masters (asserting the *eb_m#_bg*, *eb_bmx* and *eb_ft_bg* signals)
- Route the elected master outputs to the slaves.
- Wire-or the all the slaves outputs to the master (slaves not addressed must drive zero on their outputs).
- Monitor the slave answer and time out the idling transfers (asserting the *eb_err* signal).
- Audit the E-bone protocol and cancel buggy transfer (asserting the *eb_err* signal). Masters are assumed reliable (protocol error free).

4.3 Generic constants

4.3.1 E-bone manager generics

Name	Type	Function
EBS_TMO_DK	Natural ≥ 2	AS to DK time out is 2**EBS_TMO_DK clock cycles

4.3.2 E-bone top level generics

Name	Type	Function
EBS_SLV_NB	natural	Number of Slaves in the system

4.3.3 E-bone Fast Transmitter generics

Usually the EndPoint imposes one size for the FT bus. Next generic is to be used on DMAC cores so they fit the EndPoint bus size.

Name	Type	Function
EBFT_DWIDTH	natural	FT data bus width (32, 64, 128 or 256)

4.3.4 E-bone slave generics

These generics must be consistently assigned to every slave at the system top level. A slave may be designed for supporting (short addressing) segment #1 only.

Name	Type	Function
EBS_AD_RNGE	natural ≤ 28	Number of address bits to decode. This parameter must be the same for all the slaves mapped in the same segment. The smallest the best in the scope of a real implementation.
EBS_AD_BASE	1, 2 or 3	Segment number to be decoded
EBS_AD_OFST	natural 2**N	Offset in the segment
EBS_AD_SIZE	natural	Size occupancy in the segment
EBS_DWIDTH	natural	Data bus width (optional)
EBS_IRQ	std16	Message interrupt bit pattern (optional) The pattern 0x01 is reserved to the FT slave port.

EBS_AD_SIZE must be a power of 2, greater or equal to 4.
EBS_AD_SIZE is required, even though the core is using a known fixed size.
EBS_AD_OFST must be a multiple of EBS_AD_SIZE.

A particular slave may use more generics, in addition to those required.

4.3.5 Endpoint Interface parameters

Application dependent. Main parameters are usually:

- PCI Device ID
- Number of lanes
- BAR mapping to E-bone segments.

A possible implementation would be as follows.
BAR0 is fixed 4KB memory reserved for the EndPoint internal resources.
Next table summarizes the PCIe to E-bone mapping.

PCI-express Address increments by 4	EndPoint i/f	E-bone segment Address increments by 1
BAR0	Internal resources	0 (reserved for future extension)
BAR1		1 (typically I/O space)
BAR2-3		2-3 (typically memory space)

4.4 Core names

It is recommended to use the following naming conventions.
E-bone EndPoint core must begin with ***ebm0_*** (master #0).
E-bone other masters cores must begin with ***ebm_*** .
Fast Transmitter core must begin with ***ebft_***
Other E-bone (slaves) core names must be prefixed by ***ebs_***.
E-bone master functional models must be prefixed by ***ebfm_***.

4.5 Multiple FPGA implementation

E-bone control interconnect implementation in a single FPGA is no problem. But it may be necessary to have the E-bone spawning over two FPGAs, thus crossing the pads boundaries. If enough pads are available the E-bone buses and controls can just be wired between the two FPGAs.

When pads must be saved, or the Endpoint clock is too fast, E-bone can be transported over fast full-duplex serial link (for example using the AURORA protocol). The E-bone protocol has been defined to ease such serial implementation. Anyway the E-bone signalling, as defined in this specification, must be re-created at both ends of the links. This ensure the E-bone cores are re-usable independently of the system implementation.

5 E-bone extensions for data readout

5.1 Motivation

Two (or more) E-bone interconnects may be used in the same system.
A typical architecture would use (fig. 8) two E-bone interconnections.

- A first E-bone (#1) for interfacing to the Endpoint, and,
- A second E-bone (#2) dedicated to the data readout backbone. The latter actually dumps data at high speed into the Fast Transmitter port of the Endpoint E-bone.
- E-bone #2 data with may be extended beyond 32 bits in width, with a mixture of slaves of different data widths (still multiple of 32 bits).

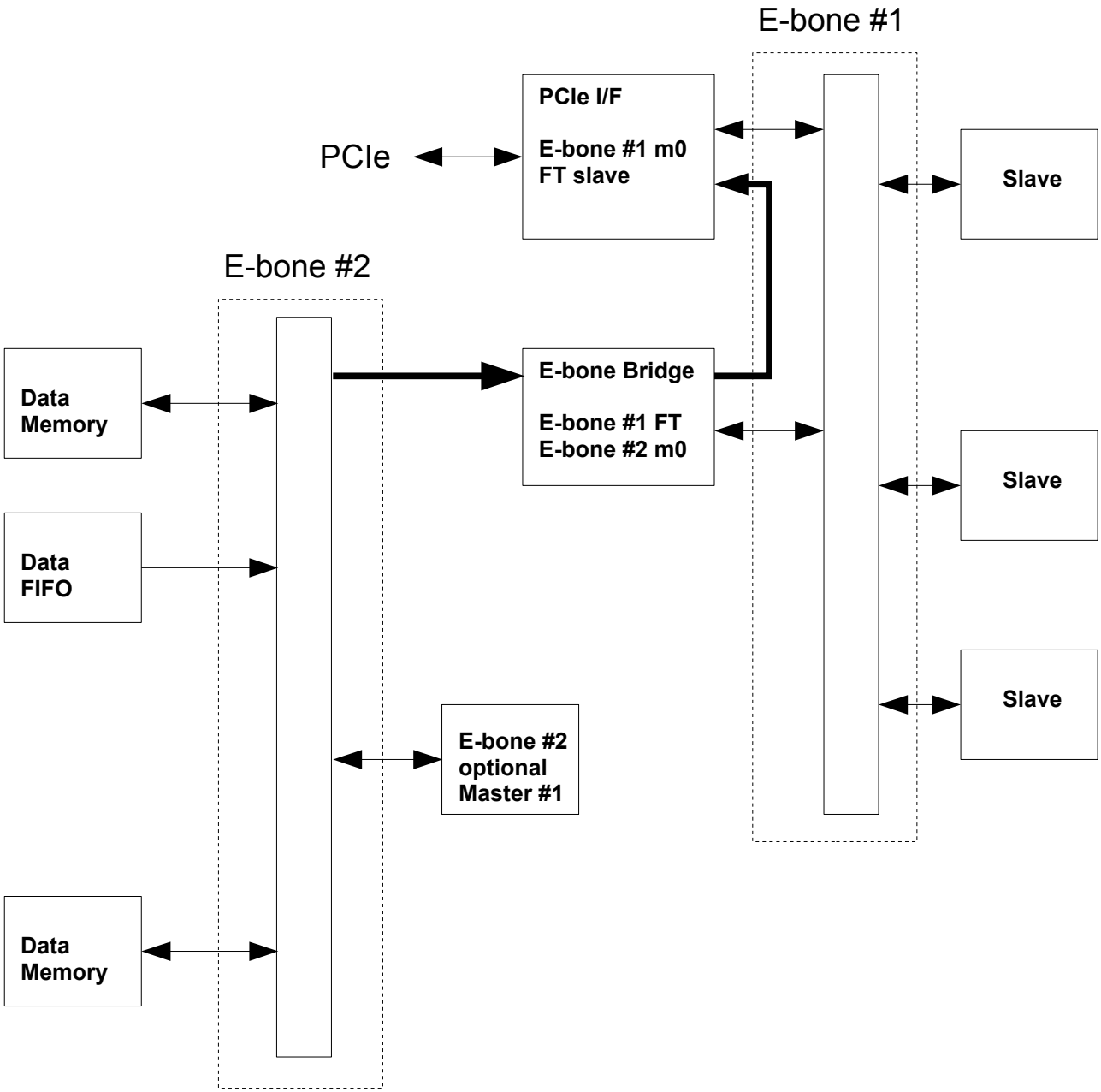


Fig. 8 : Dual E-bone architecture for data readout

The main data flow is emphasized in bold on fig. 8.
The E-bone bridge acts as

- a Fast Transmitter master on E-bone #1
- a plain master (#0) on E-bone #2, mainly reading out from data storage areas.

Nothing prevents the Bridge to write onto the E-bone #2 (not represented in fig. 8). Optionally a second master (Master #1) could manage data as required by the application.

The whole system is still controlled through a number of (32 bits) slaves connected on the first E-bone.

For best data throughput the second E-bone interconnect data width should fit the first E-bone Fast Transmitter data width (usually 64 bits or more). This is possible with the E-bone extended specification. It allows for interconnect data width larger than the basic 32 bits number.

A fast communication mechanism is often required (similar to “interrupts”) between a slave and a master. For example a (master) DMAC should know when data is actually available from the (slave) storage memory. Periodical broad-cast is not adapted to this case. A direct message passing is defined instead.

5.2 E-bone extended data width

The data bus may be extended beyond 32 bits in width.

Master	Slave	Size	Description
eb_m#_dat_o	eb_dat_i	32, 64, 128 or 256	Multiplexed Type/Offset/Data write bus
eb_dat_i	eb_dat_o	32, 64, 128 or 256	Data read bus
ebx_dsz_i	ebx_dsz_o	4	Actual data size. Generated by the slave during the addressing phase ahead and until DK is asserted (0=32, 1=64, 2=128, 3=256).

In the addressing phase, whatever the data width, the format remains unchanged. Only bits 31-0 are meaningful. Upper bits are not used.

5.3 E-bone extended messages

5.3.1 Message concepts

The message concept actually covers two independent features.

First, during any addressing phase cycle, it allows the master and the selected slave to exchange additional informations. Concurrently a channel message may be set up, that will survive the only addressing phase, until it is explicitly closed by the originating master.

Secondly, but when addressing phase, messages are point to point connections from a slave towards a master that can be dynamically managed by the masters. Once the connection has been set up (on request from a master), this master receives the messages from the specific slave the request was addressed to. The message data is 16 bits in width. An E-bone extension inter-connect (actually a cross-bar switch) routes the message as required.

The message channel exists until explicitly closed by the master. It is noteworthy that its scope extends beyond the elementary burst duration.

5.3.2 Message signals

<table>
<tr><th>Master</th><th>Slave</th><th>Size</th><th>Description</th></tr>
<tr><td>ebx_msg_set_o</td><td>ebx_msg_set_i</td><td>8</td><td>Valid only for the addressing phase.
<table><tr><td>Bits 7-4
Reserved</td><td>Bits 3-0
Master sends its unique identifier (MID) to a slave.
The slave memorizes the MID (non zero).
The message channel has been established.
The channel is closed when the slave receives a zero MID.</td></tr></table></td></tr>
<tr><td></td><td>ebx_msg_dst_o</td><td>4</td><td>The MID attached to any message from the slave allowing the E-bone extension interconnect to route to the destination master.</td></tr>
<tr><td>ebx_msg_dat_i</td><td>ebx_msg_dat_o</td><td>16</td><td>Addressing phase
Reserved

Outside addressing phase
Message data.
Actually a permanent virtual connection for the whole channel duration.
There is no validation signal associated to the message that may change at every clock edge.</td></tr>
</table>

The *ebx_msg_set_o* signal must be driven to zero by all potential masters, but the acting one. Thus the *ebx_msg_set_i*, which is shared by all slaves, just results from the OR of the master requests.

ebx_msg_dat must remain stable for the whole addressing phase. Then is is allowed to change freely.

It is noteworthy that E-bone extension can be implemented fully independently from the plain E-bone interconnect. It just adds on aside.

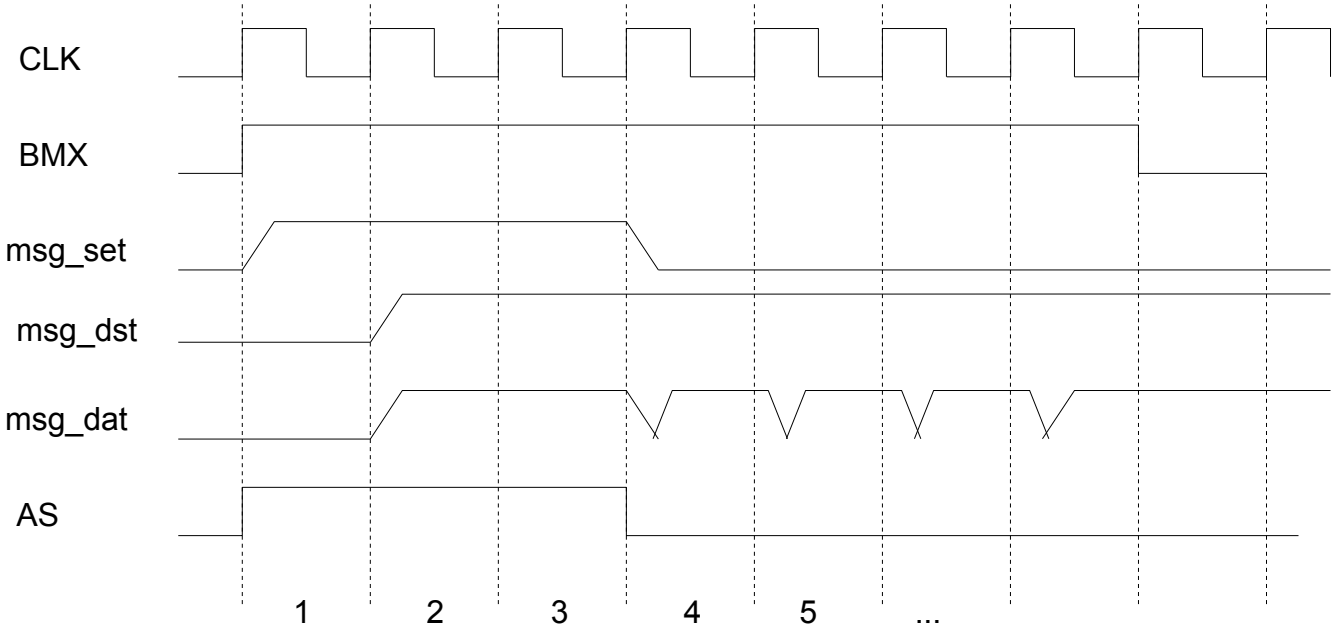


Fig. 9

1	Master tells the addressed slave its MID by setting *msg_set*. Message channel is open. Addressing extensions may be simultaneously specified.
2	Slave drives *msg_dest* with the MID, so the message can be routed back. Slave drives message data *msg_dat*, possibly to inform the master about its capabilities. *msg_dat* must remain stable for the whole addressing phase.
3	Slave continues driving message data *msg_dat.*
4	AS goes un-asserted. *msg_dat* may carry any information until the message channel is closed.
5	Message channel remains open, past the E-bone burst. *msg_dat* may carry any information until the message channel is closed.
6	(not shown on this figure) Any new channel opening request will be rejected by the slave. This is done by cancelling the addressing phase (E-bone ERR response).
7	(not shown on this figure) Master, along with some addressing phase, drives *msg_m_set* to zero, thus closing the message channel. *msg_dst* goes to zero.

5.3.3 Slave message semantic

The message semantic from a slave to a master is left open by this specification. Messages can be defined on purpose for specific system requirements. However the following recommendations apply to promote inter-operability.

Addressing phase	**Outside addressing phase**
Reserved.	Interrupt or status; for. ex. FIFO occupancy... used by a slave to inform the master about some event.

6 Annexes

6.1 E-bone Endpoint interfacing summary

Signal	M/S	Size	Description
eb_m0_clk_o	M	1	System clock
eb_m0_rst_o	M	1	System reset
eb_m0_brq_o	M	1	Master bus request
eb_m0_bg_i	M	1	Bus granted by the E-bone manager.
eb_m0_dat_o	M	32	Multiplexed Type/Offset/Data write bus
eb_m0_as_o	M	1	Master port data strobe
eb_m0_eof_o	M	1	Master port end of frame
eb_m0_aef_o	M	1	Master port almost end of frame
eb_dk_i	M	1	Master port data acknowledge
eb_err_i	M	1	Master port error
eb_dat_i	M	32	Data read bus
eb_bmx_i	M	1	Equivalent to eb_m0_bg_i in the absence of others masters
eb_ft_bg_i	S	1	Fast Transmitter granted
eb_ft_dxt_i	S	32	FT data bus
eb_ft_blen_i	S	8	FT burst length
eb_ft_ss_i	S	1	FT size strobe
eb_as_i	S	1	FT Slave port data strobe
eb_eof_i	S	1	FT Slave port end of frame
eb_dk_o	S	1	FT Slave port data acknowledge
eb_err_o	S	1	FT Slave port error

Total nets
73 for a light Endpoint without Fast Transmitter,
119 with a 32 bit wide Fast Transmitter.

6.2 E-bone slave interfacing summary

Signal	Size	Description
eb_clk_i	1	System clock
eb_rst_i	1	System reset
eb_bmx_i	1	Master (but FT) bus granted
eb_dat_i	32	Write Type/Offset/Data bus
eb_dat_o	32	Read data bus
eb_as_i	1	Data strobe
eb_eof_i	1	End of frame
eb_dk_o	1	Data acknowledge
eb_err_o	1	Error, transfer cancelled

Total 71 nets (assuming eb_aef_i not used, as for basic slaves).

Generics

Name	Type	Function
EBS_AD_RNGE	Natural ≤ 28	Number of address bits to decode
EBS_AD_BASE	1, 2 or 3	Segment number to be decoded
EBS_AD_OFST	Natural 2**N	Offset in the segment
EBS_AD_SIZE	Natural	Size occupancy in the segment
EBS_IRQ	std16	Message interrupt bit pattern (from the slave #)

7 Revision history

Rev.	Date	Comment
0.1	10/01/09	Preliminary
1.0	04/01/10	1st stable release
1.1	17/09/10	Addressing phase extended. Retry revisited. Added *eb_aef* signal. Interconnect data width extended beyond 32 bits.
1.2	11/04/11	ERR not used any more in the data phase. DK usage made crystal clear.
1.3	12/09/13	E-bone extension revisited; added messages; added FT word down-sizing; added FT burst length and FT unaligned data management.