HyperTransport Advanced X-Bar (HTAX) Specification

Computer Architecture Group

University of Heidelberg

ver. 0.14

Heiner Litz

Table of Contents

List of Figures

List of Tables

1. Revision History

ver 0.12:

- Changed signal name from start_arbitration to release_gnt

ver 0.13:

- Clarified behaviour of tx_vc_req signal with multiple vc requests
- Clarified rx_vc_gnt in the case where rx_vc_gnt is asserted without tx_vc_req
- Clarified tx_vc_req == one hot if MFL is used

ver 0.14:

- Added burst mode capability
- Clarified behaviour of *tx_req* signals for the next packet during active transmission of current packet. *tx_req* signals may only be asserted concurrently with *tx_release_gnt*
- Updated rx/tx transaction examples
- Changed signal names by adding a preceeding *tx_* or *rx_*
- Changed protocol name of cagHT to HTOC

2. Terminology

This specification assumes that the reader is familiar with the HyperTransport 3.0 specification. Many terms and abbreviations are adopted from HyperTransport. The terms that are specific to this document are introduced in the following.

HT3.0-Core: The HyperTransport 3.0 core developed by the Computer Architecture Group (CAG).

HT fabric: This term defines the HyperTransport device chain as described in the HyperTransport 3.0 specification. The *HT fabric* ends at the HT3.0-Core.

HTOC fabric: This term defines the fabric behind the HT3.0-Core. This is the area where the user application connected to the core is located. The *HTOC fabric* defines a protocol which is very similar to the original HyperTransport protocol.

HTOC protocol: This protocol is an on-chip protocol derived from the HyperTransport protocol. It is very similar to allow for simple protocol conversion. It is enhanced to comply with the needs of an on-chip protocol.

HTOC transaction: Derived from the HyperTransport specification this term describes a complete HyperTransport message in the *HTOC fabric* which consists of a command and a data part which are made up of packets.

HTOC packet: Derived from the HyperTransport specification this term describes the physical units which are combined to form a *HTOC transaction*. Two different types of packets exist namingly command and data. *HTOC packets* are defined for two different sizes, 64 and 128 bit depending on the *HTOC fabric* implementation.

HTAX: Abbreviation for HyperTransport Advanced X-Bar. The HTAX defines a switching structure which resides in the *HTOC fabric* and allows bidirectional communication between different communication endpoints. The HTAX is a protocol agnostic switch and by definition able to switch arbitrary protocols. It is however optimized to support the HTOC protocol

Functional Unit (FU): A communication endpoint connected to the HTAX. This term includes the HT-Core.

~this~: This specific instance

3. Overview

The High Throughput Advanced X-Bar (HTAX) is a protocol agnostic, high throughput, low latency crossbar architecture. It follows a flexible and modularized approach that enables the HTAX to be used in various applications, in the area of on-chip and off-chip and node-to-node interconnection networks. The HTAX can be flexibly configured in terms of supported number of ports, virtual channels, bus width and buffer capabilities. It supports single stage and multi stage configurations which offers low latency operation for low radix switches but also the possibility to scale to a virtually infinite amount of ports. This enables it to be used in both direct connected and multi dimensional torus topologies. Interconnection networks can be applied in a wide range of applications ranging from high latency node-to-node implementations to on-chip networks that require thousands of ports and low latency. The characteristics of these applications regarding buffer sizes, topologies, latency and bandwidth differ dramatically therefore posing different sets of requirements. Due to its flexiblity the HTAX is able to satify all of them.

The HTAX consists of bidirectional ports that are interconnected by a switch matrix, as shown in figure 1. The ports provide a transmit (TX) and receive (RX) interface and can be input buffered or bufferless. The HTAX supports an arbitrary number of virtual channels for Quality of Service (QoS) and deadlock avoidance. To handle congestion both virtual output queueing (VOQ) and RECN mechanism is supported for high radix switches.

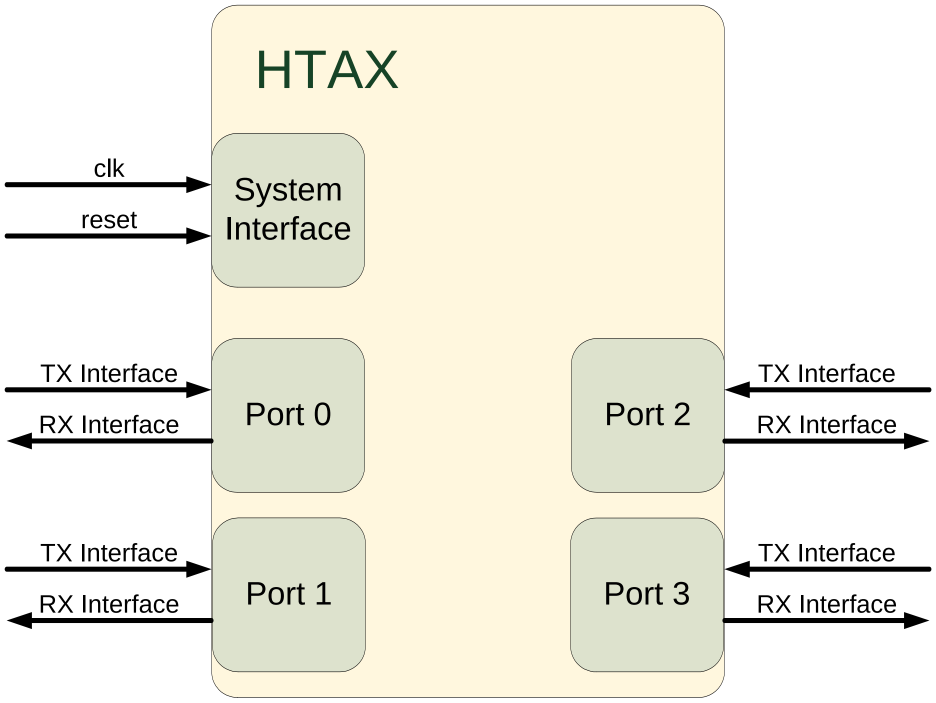


Figure 1:HTAX block diagram

4. Interfaces

The HTAX is a parametrizable design. There are various parameters which have to be defined at compile time to generate a crossbar with the desired functionality. The available parameters are described in the following:

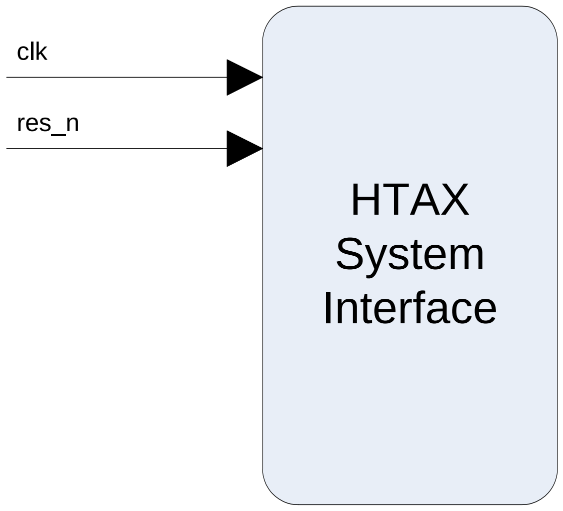
- PORTS:
  The number of ports connected to the HTAX. Each port is bidirectional and is subdivided into the TX interface and the RX interface. If a unit connected to the HTAX requires only unidirectional communication the signals have to be tied to zero. They will be removed by synthesis.

- VC:
  The amount of virtual channels.

- WIDTH
  The width of the data bus in bits.

4.1 System Interface

There may be more to follow.



4.2 TX Interface

Data packets are injected into the HTAX through the TX interface. Figure 2 shows the TX interface signaling.

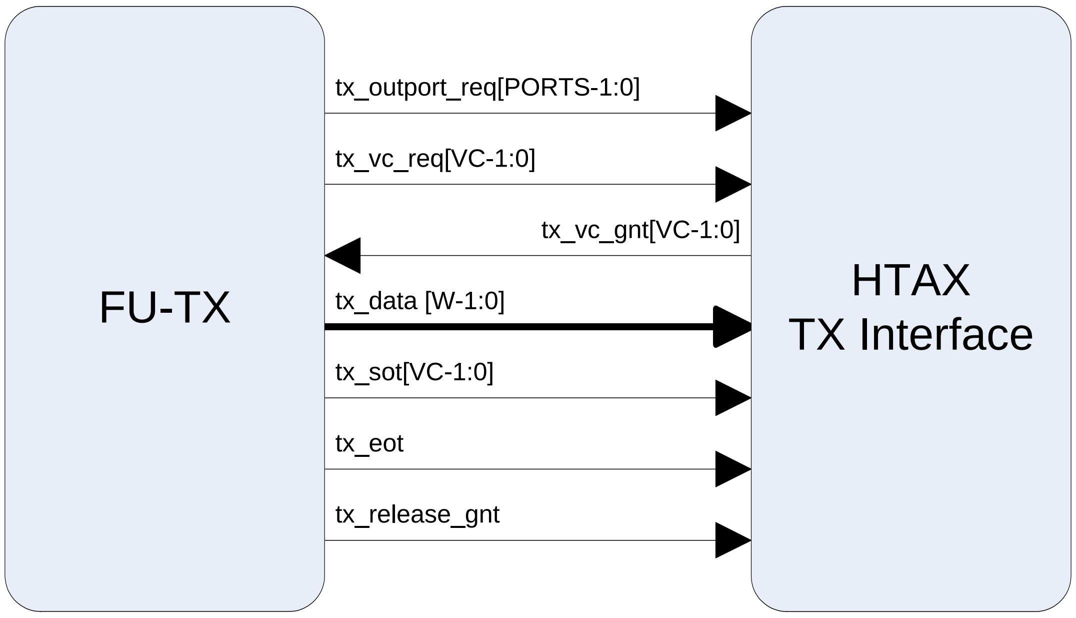


Figure 2:HTAX TX Interface

tx_outport_req[PORTS-1:0]

This signal is used by the transmitting functional unit to request an outport of the HTAX for a specific transaction. It is a one-hot encoded signal and its width depends on the number of outports connected to the HTAX. The *tx_outport_req* signal has to be asserted and deasserted simultaneously with the *tx_vc_req* signal. It is recommended that once a request is asserted it stays active until the request has been served. The transmitter may only withdraw previous requests if it takes care of a possible *tx_vc_grant* which may be asserted in the next clock cycle. The next clock cycle after *vc_grant* is asserted the *tx_outport_req* signal has to be deasserted. The FU-TX may assert *tx_vc_req* and *tx_outport_req* for the next packet concurrently to a data transfer to hide arbitration latency. The *tx_outport_req* for the next packet have to be asserted simultaneously with *release_gnt* of the previous packet.

tx_vc_req[VC-1:0]

This output signal is used by the transmitter to specify the virtual channel used for the current outport request. It is allowed to request multiple virtual channels simultaneously. The signal width depends on the number of supported virtual channels. *tx_vc_req* has to be asserted and deasserted simultaneously with the *tx_outport_req* signal. As long as an outport is requested it is mandatory to request at least a single virtual channel. It is not allowed to retrieve asserted *tx_vc_req*. If multiple *tx_vc_req* are requested and multiple *tx_vc_gnt* are granted the transmitter may decide which vc to use for the next transaction. The next clock cycle after *vc_grant* is asserted the *tx_vc_req* signal has to be deasserted. If multiple *tx_vc_req* are granted a transaction has to be started at the next clock cycle after *tx_vc_req* && *tx_vc_gnt* is active and only the *tx_vc_req* for the virtual channel that is started, defined by the *tx_sot* signal, must be deasserted. The FU-TX may assert *tx_vc_req* for the next packet concurrently to a data transfer to hide arbitration latency. The *tx_vc_req* for the next packet have to be asserted simultaneously with *release_gnt* of the previous packet.

tx_vc_gnt[VC-1:0]

This signal is provided by the HTAX to grant requested virtual channels for a requested outport. It is possible to grant several virtual channels at once. It is asserted for a single clock cycle. The requester is forced to deassert its *tx_outport_req* and *tx_vc_req* signals in the next clock cycle if it does not require to send further packets. Keeping the request signals asserted in the next clock cycle automatically leads to another outport request.

tx_data[WIDTH-1:0]

This signal bus transports the payload data. The width of the data bus is represented by the WIDTH parameter.

tx_sot[VC-1:0]

The *tx_sot* signal indicates the start of a transaction for a specific virtual channel. It is a one-hot encoded signal. A specific request has to be asserted at the same time or before *tx_sot* may be asserted. As soon as a grant for the requested outport has been received the *tx_sot* signals has to be deasserted. It has to be asserted simultaneously with the first data packet. It is recommended to assert *tx_sot* as soon as possible after *tx_vc_gnt* has been received to avoid blocking of the requested outport. It is allowed to assert *tx_sot* and data for a single requested virtual channel in prior of a received *tx_vc_gnt*. In this case the *tx_sot* and *tx_data* signals are registered by the HTAX during assertion of *tx_vc_gnt*. This minimum latenvy feature (MLF) reduces the startup latency by a single clock cycle. If the MLF is used, only a single virtual channel may be requested, respectively the concurrent *tx_vc_req* signal must be one hot encoded.

tx_eot

The end of transaction signal is asserted simultaneously with the last data packet of a transaction and determines the end of the transaction. It is asserted for a single clock cycle.

tx_release_gnt

This signal triggers the HTAX outport to perform a new arbitration cycle. Asserting *release_grant* during an active transfer allows to overlap and hide arbitration latency needed for streaming data back-to-back. It is asserted a single clock cycle in prior of *tx_eot*, *tx_release_grant* may only be asserted if a grant has been received previously.

4.3 Transmit Interface Examples

Figure 3 shows an example transaction using the TX interface. The transmitting functional unit starts a transaction in t1 by requesting two virtual channels (*tx_vc_req*) for a specific outport (*tx_outport_req*). Both requested virtual channels are acknowledged through the HTAX in t2. In t3 the transmitter begins the transaction using virtual channel number zero by asserting *tx_sot* acordingly. The next timing slots show the transmitter driving valid data on *data* and finishing the transaction by asserting the *tx_eot* signal in t7. The *start_arbitration* signal is asserted in t6 to enable the outport to start a new arbitration sequence.

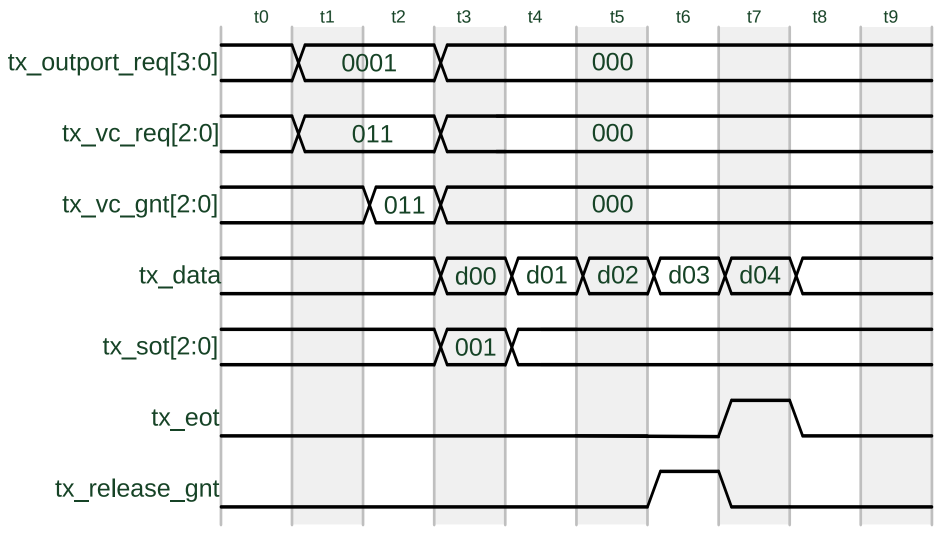


Figure 3:Single transmit request

The example shown in figure 4 shows a transmitter sending two consecutive transactions. In t1 two virtual channels for outport number zero are requested. The inport grants both virtual channels in t2 which enables the transmitter to start a transaction using virtual channel number zero in t3 by asserting *tx_sot* and driving data on the *tx_data* bus. In t3 the outport and virtual channel requests must be deasserted. The *tx_release_gnt* signal asserted in t4 allows to assert *tx_outport_req* and *tx_vc_req* for the next packet. The arbiter grants the virtual channel one in t5 which enables the second transaction.

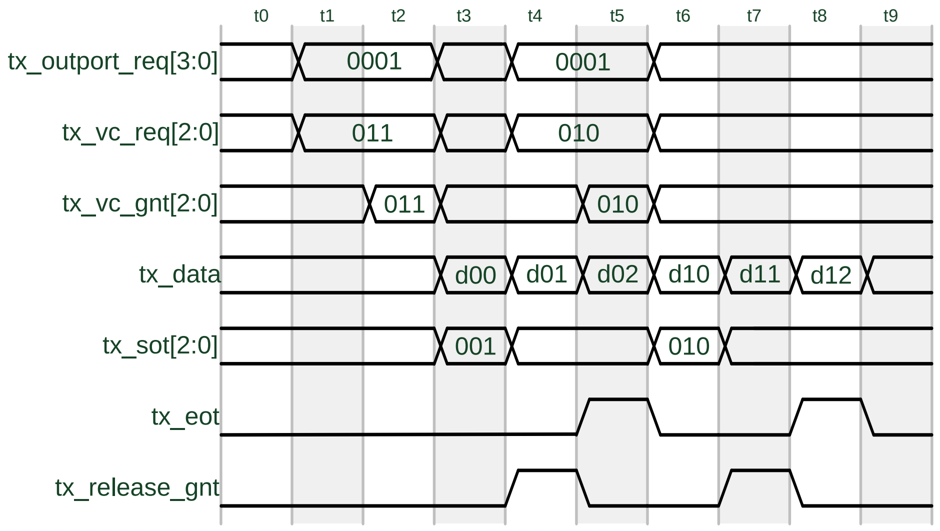


Figure 4:Multiple transmit request

Figure 5 shows a transmitter of utilizing the minimal latency feature provided by the HTAX. As there is only a single virtual channel requested in t2 the transmitter is able to assert *tx_sot* isochronously to the request. The first data packet can therefore be already transferred while *tx_outport_gnt* is received.

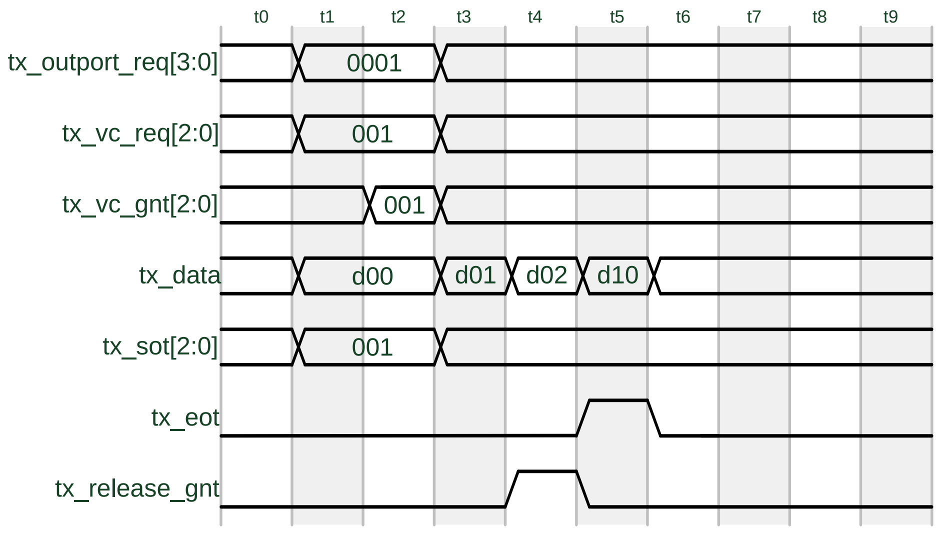


Figure 5:Transmit request utilizing the Minimal Latency Feature

4.4 RX Interface

Data packets are retrieved from the switch through the RX interface. The outport forwards packets to the receiving functional unit and contains an arbiter that hands out grants to the inports. It is a virtual channel aware multi request arbiter which allows to signal multiple packets of different virtual channels waiting for transaction to the receiver. The receiving functional unit needs to implement arbitration logic to determine which virtual channel to grant. The outport's RX interface is shown in figure 6. The receiver unit is allowed to grant several virtual channels concurrently.

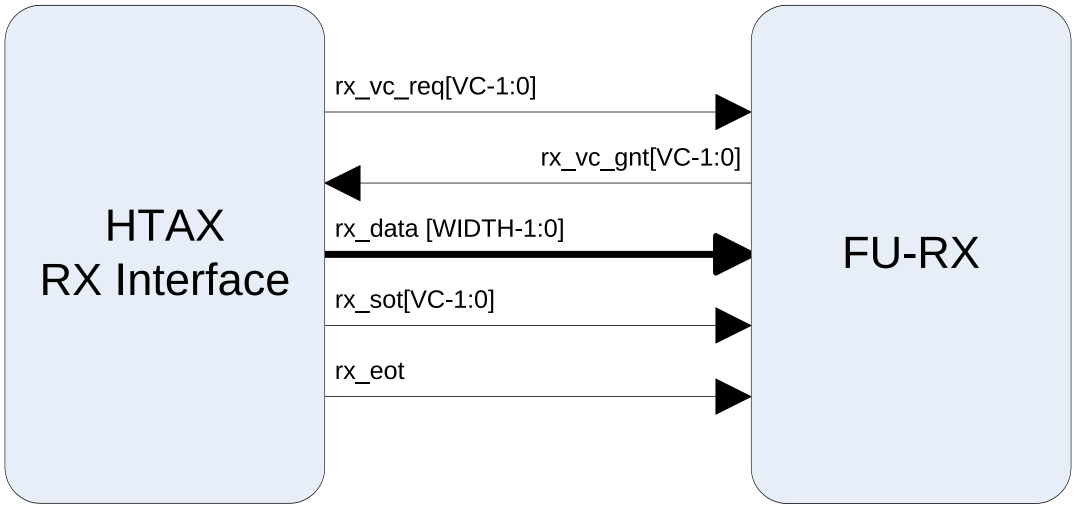


Figure 6:HTAX RX Interface

rx_vc_req[VC-1:0]

This signal determines the virtuals channels of the transactions that request the unit at the RX interface side. Its width equals the number of virtual channels supported by ~this~ HTAX. Multiple virtual channel requests at the same time are allowed.

rx_vc_gnt[VC-1:0]

This output signal acknowledges possible reception of transactions over specific virtual channels. Its width is equivalent to the number of virtual channels. Once asserted the unit is required to accommodate the complete transaction. It is allowed to assert *rx_vc_gnt* without having *rx_vc_req* of that specific virtual channel enabled. Once *rx_vc_gnt* is asserted the unit is not allowed to deassert it until a transaction starts, that is *rx_vc_req* and *rx_vc_gnt* are active for a single clock cycle. The functional unit is responsible for determining which virtual channel a current transaction is being processed on. To grant a transaction for a specific virtual channel *rx_vc_req* and *rx_vc_gnt* for the same virtual channel has to be asserted for a single clock cycle. If *rx_vc_req* and *rx_vc_gnt* for the same virtual channel are asserted for multiple cycles this may be regarded as single or multiple grants for transactions depending on whether a transaction is currently running or not.

rx_data[WIDTH-1:0]

This signal bus carries the data of the transaction. Its width is set by the parameter WIDTH.

rx_sot[VC-1:0]

This one-hot encoded signal is asserted simultaneously with the first packet of a transaction and determines that the *rx_data* bus contains the first data packet. It identifies the virtual channel which is used to transmit the transaction. It is asserted for a single clock cycle.

rx_eot

This signal is asserted simultaneously with the last packet of a transaction and determines that the transaction will be finished in the next clock cycle. It is asserted for a single clock cylce. If *rx_vc_gnt* is asserted while *rx_eot* is high and *rx_sot* is low the *rx_vc_gnt* is regarded as grant for another packet. If *rx_vc_gnt* is asserted while *rx_eot* is high and *rx_sot* is high it is not regarded as the grant for another packet.

4.5 RX Interface Examples

Figure 7 shows an outport to functional unit transaction. In t1 the HTAX signals a pending request for the specific outport. The receiver grants the requested virtual channel in t3. The transaction proceeds in t4. It contains four data packets which are framed by the *rx_sot* and *rx_eot* signals. By deasserting *tx_vc_req* in t5 the HTAX signals that no other outport requests exist.

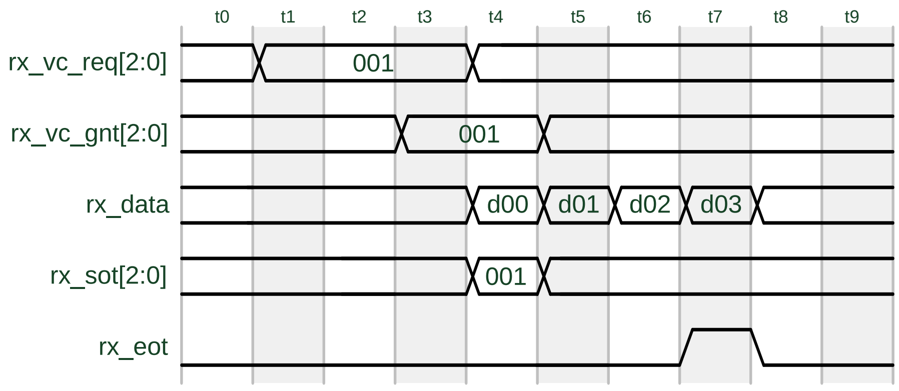


Figure 7:Single Outport to Completer Transaction

Figure 8 shows two transactions. The HTAX requests the virtual channel number zero in t1 which is granted in t2 by the receiving unit. Processing of the transaction starts in t3 by asserting the rx_sot signal and is finished in t4 signalled by *rx_eot*. As there are still pending requests the receiver unit grants another transaction in t5 which is carried out in t6 and t7. To allow back-to-back streaming of transactions the receiver unit would have to grant transactions in prior.

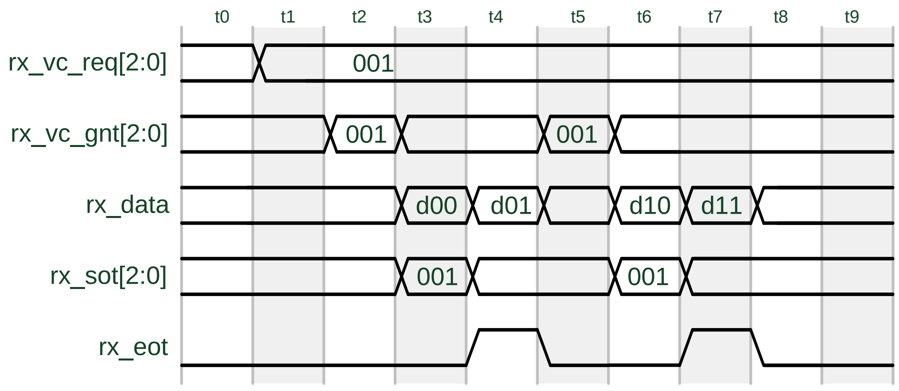


Figure 8:Latency minimized vc arbitration

The example shown in figure 9 shows back-to-back streaming of minimal sized transactions. Transactions with a single data size cannot be processed back-to-back. The receiver has sufficient buffers to accept transactions for all virtual channels and therefore grants all vc. Then the transaction can be immediately processed as soon as the *tx_vc_req* and *rx_sot* for a transaction is asserted. Note that one clock cycle latency is avoided by asserting *rx_vc_gnt* in prior.

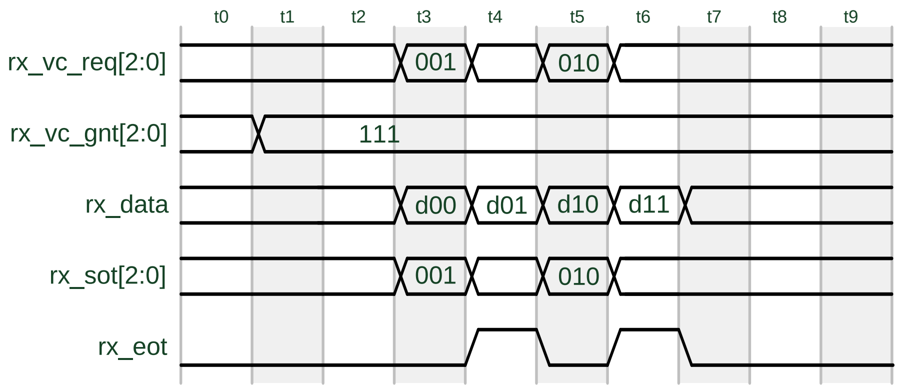


Figure 9:Back-to-Back Streaming Transactions

4.6 Burst Mode

The HTAX switch supports a burst mode. When sending minimum sized packets over the HTAX, bubbles are introduced through the delay of the arbitration mechanism. To address this issue the HTAX supports burst mode. A single grant can be used to transmit multiple packets. Each packet is framed by sot and eot signals. The grant is only released by the release_gnt signal. The FUs need to define a maximum burst size (MBS). If a grant is given by a FU the FU must be capable to receive multiple packets as defined by the MBS. The FUs may define burst capability only for small or larger messages. The definition of the MBS and the exact behaviour is out of scope of this specification and is part of the higher layer protocol (HTOC).

5. HTAX Building Blocks

The HTAX is a protocol agnostic switch and therefore contains no routing interpreter. To be able to route HTOC transactions a routing interpreter in the transmitting unit connected to an HTAX is required. As these routing interpreters are a reusable resource several routing interpreters have been defined which can be used by any unit connected to the HTAX. Routing interpreters for the following protocols have been defined.

TODO: HTAX building blocks specification

5.1 HTOC protocol

In the case of HT transactions are routed using different mechanisms:

- Address intervall routing. Non coherent HT transactions like posted writes are routed to their endpoint by interpreting the address embedded in the command.
- SrcTag lookup routing. Non coherent response transactions are routed regarding their embedded srcTag respectively ext_srcTag.
- NodeID-unitID based routing. Coherent HT transactions are routed regarding their embedded nodeID and unitID.

Mode of Operation
This chapter provides an insight into the inport and outport internals. The switch itself is simply a bunch of wires without any essential functionality connecting the various inports and outports together. The only impact of the switch on the design is the number of wires which have to be routed which hence has an impact on operation frequency. However additional pipeline stages will have to be implemented in the inports or outports and not in the switch.

5.2 Inport

The inport has the purpose of requesting the correct outport for a specific packet. Therefore it needs to interpret the command of a HyperTransport packet. HyperTransport implements two possible ways to route packets. Posted and non-posted request commands include an address field which specifies its target. These types of packets are handled by the address interpreter logic. Response packets carry no address but only a srcTag field which contains a unique number that matches the response to a certain non-posted request. Routing of responses are handled by the tag management unit.

Address Interpreter
The address interpreter unit routes posted and non-posted packets. Therefore it interprets the part of the address which is used to determine the functional unit and the base address register (BAR) field of the command. This information is used to index the Address Lookup Table (ALT) which in return provides the target functional unit. The inport can then request the corresponding outport for a specific virtual channel (posted or non-posted in this case). In the case of a non-posted request has to retrieve a srcTag before requesting an outport. In this case it requests a tag from the SrcTag Management Unit before requesting the outport.

SrcTag Management Unit
The SrcTag Management Unit has two main purposes which are the routing of response packets and the management of srcTags. Therefore it implements two data structures which are the SrcTag Lookup Table (SLT) and the Tag Available Queue (TAQ). The SLT is used to dynamically map srcTags to functional units (FUs). Each time an FU sends a non-posted request it gets assigned a srcTag by the srcTag management unit which creates a new entry in the SLT to store the srcTag/FU mapping. The SLT is then used to route incoming response packets to the appropriate FU. The SLT has a size of 32 entries, one slot for each available srcTag kept in a canonical order. The table lookup can be done in a single clock cycle as no searching is needed.

The TAQ keeps track of available and handed out tags. It is implemented with 5 bit wide FIFO that holds all available srcTags. Every new non-posted request is assigned the first item from the queue reducing the available tags by one. If the TAQ is empty all tags have been handed out and no more non-posted requests can be processed. Each time a response returns, the freed tag is inserted in the FIFO, incrementing the available tags by one.

5.3 Outport

The outport's purpose is to transfer data from the inports to the functional units. Each outport is connected to several inports however only one inport can be allowed to send data to a certain outport at a time. This necessity is enforced by the arbiter. An arbitration cycle is performed by issuing a request at the inport which then has to wait for getting assigned a grant by the arbiter residing in the outport. In detail the HTAX has to actually support two request-grant mechanisms. Besides multiple inports also multiple virtual channels from a single or several inports can request an outport. The grant of a specific inport is carried out by the outport totally transparent to the functional unit. The grant policy is first come first serve, if two inports request the same outport at a time the inport with the higher priority is favored.

Granting a specific virtual channel however is carried out by the functional unit residing behind the outport. This allows the functional unit to be in full control of the virtual channel arbitration.

Rational: Grant mechanism

The HTAX consists of inports and outports. Every outport can be accessed by all inports in the switch which leads to a shared resource problem. A possibility to solve this problem would be the use of time division multiplexing (TDM) however this method shows bad performance as it reserves a time slot for every inport even if there is no data to be send for the specific inport. A better solution is to implement a request-grant mechanism. This allows the inports to request an outport at any time however access is not guaranteed but granted by the outport. The outport therefore selects exactly one of the requesting inports. In the scope of the HTAX network the pool of possible request initiators can be subdivided into two groups. One is the physically existing inports and the other is the number of virtual channels supported by the inports. A request R(i, v) therefore has two attributes which are the inport i and the virtual channel v. In general grants are given by the outport. This paper proposes another way which is to assign grant responsibility to the requested functional unit. The several possibilities to implement this two way grant mechanism are shown in figure 10.

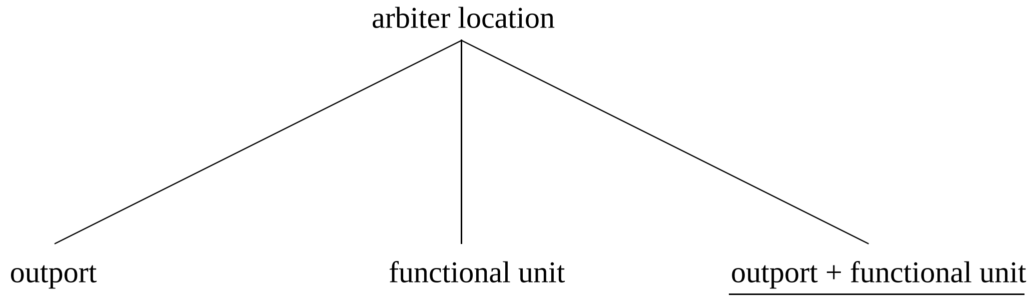


Figure 10:Arbiter location

outport

The arbiter is located in the outport and performs both arbitration of the inport and the virtual channel. In this traditional approach the arbitration sequence is totally transparent to the target. It provides the most convenient interface to the functional unit however creates a deadlock possibility in certain use cases. As an example consider a master-slave working model. A functional unit attached to the HTAX acts a a slave and is assigned new jobs by receiving posted command transactions. To execute such a job the salve needs to fetch data using a non-posted read which is then answered by a response. If the outport would grant requesting transactions solely on a first come first serve policy, new jobs could block the responses which are needed to complete previous jobs. Even if the device supports multiple outstanding jobs the possibility exists where a number of job requests block the responses.

functional unit

The cyclic dependency between posted transactions and responses discussed above can be broken off by implementing a virtual channel aware functional unit interface. Arbitration of a R(i, v) is carried out by the functional unit rather the outport. In this case the FU can select whether it is able to process new jobs by granting the posted VC or to complete a previous job by granting the response VC. The disadvantage of this approach is the required number of i + v control signals at the completer interface.

outport + functional unit

This favored solution combines the advantages of both approaches discussed above. It provides maximum flexibility and a rather convenient interface by splitting up the arbitration process into two separated sequences. The completer interface supports an additional number of v wires to signal VC requests to the functional unit. This avoids the deadlock scenario mentioned above. The eventual arbitration of a specific inport on that VC however is carried out by the outport transparently. The ability of the FU to select specific inports is lost in this implementation however the following should be considered. The HTAX provides a platform for a highly flexible and modularized architecture. Functional units are designed to be interchangeable seamlessly supporting various system architectures. The knowledge of the FU to in/outport mappings should be considered as unavailable. This in return makes selective inport granting unnecessary.

6. HT-Core-2-FU Interface

6.1 Requirements:

- Simple and convenient interface
  Minimizing design time and verification time are key factors. Modularity and easy re-usability of modules help. Encapsulation of commonly used functionality in submodules.

- Bandwidth
  Back-2-back streaming of data packets is mandatory. Hiding the arbitration latency by requesting outports during in-flight messages is needed.

- Low Latency
  Switch latency should be kept to a minimum. The more complex the design is regarding buffering, flow control, virtual channels, etc. the higher the switch latency will be.

- Efficiency
  Switch efficiency should be maximized by avoiding congestion. Key goal has to be to avoid head of line (HOL) blocking.

- Environment and traffic patterns
  The requirements shown above are in some points diametric. Architectural decisions depend on the traffic patters which will flow through the switch and the structure of the injecting and retrieving endpoints.

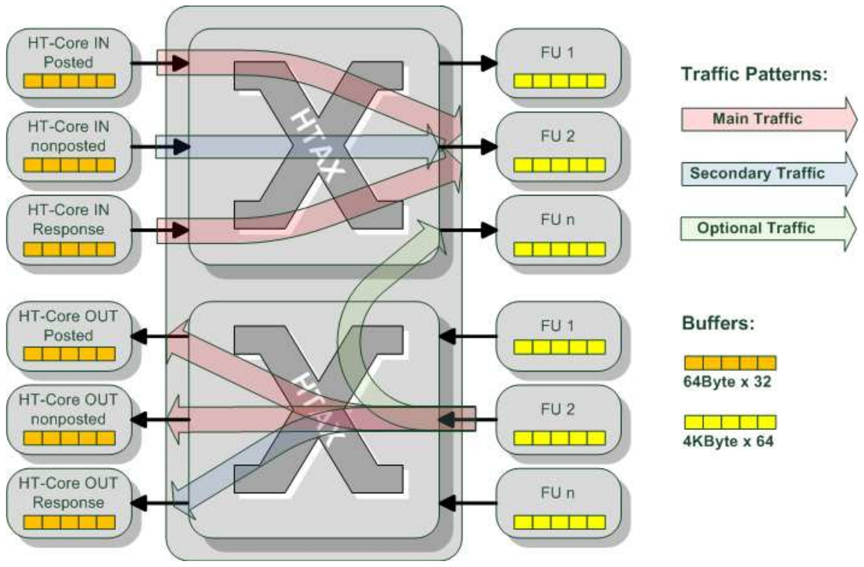


Figure 11:Toplevel Architecture

6.2 Implications

Bandwidth
To provide back-to-back streaming of data, arbitration has to be performed concurrently to a data transfer. This implicates that a data source needs to look ahead. In the case of a functional unit this is easy to implement as the FU has a specific goal, e.g. sending out 4 KB of data in several HT transactions. The HT core however is a priori unaware of the data it needs to deliver to the crossbar. The HT core has to be able to inspect the next HT transaction which is residing in its buffer while processing another transaction concurrently.

Note: The request of an outport performed by the routing interpreter is determined differently depending on the type of transaction:

- Static: The source has static information to determine the target for the transaction. The lookahead is therefore easy to implement. The content of the command (address, SrcTag) is ignored.
- Address Interval based: Regular posted and non-posted transactions are routed based on the address in their command packet. Each outport is assigned a specific address interval. The routing interpreter has to lookup the address in a table to determine the outport.
- SrcTag based: Response transactions are routed based on their SrcTag. In a HT fabric every non-posted request is assigned a SrcTag. Corresponding responses carry the same SrcTag and have to be forwarded to the initiator of the non-posted request. EXTENDED SRCTAG

Low Latency
To achieve better latency performance store and forward of packets should be avoided at any time. Virtual cut through allows to inject the transaction header into the crossbar before the complete transaction has arrived. As we cannot guarantee a continuos data flow at any time (HyperTransport may interleave transactions with other transactions) there may be bubbles in the data stream. An interrupted transaction may therefore block other transactions.

Issue: Maybe we can (have to) define that transactions may not be interrupted. As transactions always have to be CRC checked (on HT and IB side) we may be able to guarantee an uninterrupted data flow. Should we allow speculative forwarding?

Efficiency
Switch efficiency is limited by HOL blocking. A common way to alleviate this problem is to implement a large buffer in every inport of the crossbar. If one outport is blocked other transactions can bypass the congested transactions if they request another outport. A common implementation is virtual output queuing (VOQ). This buffering technique can also be extended to provide hiding of arbitration latency. Disadvantage of a buffered crossbar is the increased complexity which increases latency and the amount of additional resources needed.