Router for CFG NoC Core

Functional Specifications

Joji Philip

NoC Core Router/Switch

[1 Functional requirements 2](#_Toc536245904)

[1.1 Broad considerations 5](#_Toc536245905)

[1.2 Module usage modes 5](#_Toc536245906)

[1.3 Features 6](#_Toc536245907)

[1.4 Interface definitions 24](#_Toc536245908)

[2 Implementation details 28](#_Toc536245909)

[2.1 Block diagram 28](#_Toc536245910)

[2.2 Functional blocks 29](#_Toc536245911)

[2.3 Interfaces 29](#_Toc536245912)

[2.4 Parameter and strap configurations 29](#_Toc536245913)

# Link Protocol



The figure above shows a pair of point-to-point links between two CFG NoC routers. Each half transmits data from a router output port to the input port of the downstream router. Information is transmitted in the form of packets. Each cycle of information on the link is called a flit and a packet consists of one or more flits. The physical link supports multiple virtual channels simultaneously transmitting packets in a cycle interleaved manner. Each virtual channels transmits one complete packet at a time. But on the physical link, flits of packets active on different virtual channels can be interleaved every cycle.

All signals below are active high, unless explicitly called out.

Widths of some signals are parametrized and value depends on the configured NoC

Optional signals are explicitly indicated

## Packet control/delineation (direction TX 🡪 RX)

|  |  |  |
| --- | --- | --- |
| Signal | width | description |
| flit\_valid | NUM\_VC | One hot vector indicating the virtual channel for which flit is valid in a given cycle. |
| flit\_sop | 1 | Current flit is a start-of-packet |
| flit\_eop | 1 | Current flit is an end-of-packet. Sop and eop are asserted in the same cycle for single flit packets. |
| flit\_cell\_valid | LOG2\_NUM\_CELLS | Number of LSB cells valid in the data flit on an eop. For non-eop flits, all cells in the data are valid. |
| early\_valid | 1 | Asserted at least one cycle before a valid flit is sent on the link. When de-asserted no valid flits will be sent on the link. |
| flit\_eor | 1 | OPTIONAL: Indication of end-of-round associated with QoS. This is only valid with sop of a packet. |

## Routing information (direction TX 🡪 RX)

|  |  |  |
| --- | --- | --- |
| Signal | width | description |
| flit\_route\_info | ROUTE\_INFO\_WIDTH | Source routing information used for routing a packet to its destination. This value is constant for an entire packet. Field is modified at hops in the NoC as packet makes progress to destination. |
| flit\_output\_port | 3-5 | route\_info is used to compute the output port to which an incoming flit is switched. Value is constant for all flits of a packet, because all flits of a packet on a virtual channel follow the same route from source to destination. |

## Header fields (direction TX 🡪 RX)

|  |  |  |
| --- | --- | --- |
| Signal | width | description |
| flit\_header | HEADER\_WIDTH | Header or sideband information associated with a packet. Same for all flits of a packet. |

## Data fields (direction TX 🡪 RX)

Data fields on a link can be configured to have a fixed pipeline delay from the corresponding control and header fields.

|  |  |  |
| --- | --- | --- |
| Signal | width | description |
| flit\_data | DATA\_WIDTH | Data or payload flit.  Data is defined in terms of a basic granularity unit of cell, used for resizing the width along the links in a NoC. A cell has an integer number of bits defined based on the type of traffic being carried. Data flit itself is made up of a power-of-2 number of cells if the packet has multiple flits. Single flit packets can be an integer number of cells. |
| flit\_acc\_sb | ACC\_SB\_WIDTH | Set of bits associated with data flits but are not upsized or downsized like data. Their width remains fixed through the NoC for a given virtual channel. These bits are merged using a bit wise OR function when multiple input data flits are combined to form a wider output data flit. When a wider input data flit is downsized, this information from the input flit is replicated on to the smaller output flits. |

## Flow control (direction RX 🡪 TX)

Virtual channels have a point to point credit-based flow control on a link. At reset, for each VC on a link, the transmitting end has credits equal to the number of flits in the corresponding VC buffer on the receiving end. Each flit sent by transmitter on a VC consumes one flit credit from that VCs credit pool. On the receiving end, when a flit is popped out from the VC buffer, one flit credit is sent back for that VC to the transmitter.

|  |  |  |
| --- | --- | --- |
| Signal | width | description |
| credit\_increments | NUM\_VC | This is a multi-hot vector returning flit credits from the receiving end to the transmitting end. Each cycle when asserted is equivalent to a single flit worth of credit being returned for the corresponding virtual channel. Note that multiple VCs can return credit in a single cycle. |
| link\_available | 1 | Physical link level flow control. A flit belonging to any VC cannot be sent on the link while this signal is de-asserted. |

## Optional RAS fields

If RAS protection is enabled for a traffic, the following fields will be present in the links to carry the RAS information. (This section is subject to changes as requirements and architecture is finalized)

|  |  |  |  |
| --- | --- | --- | --- |
| Signal | width | Direction | description |
| data\_ras\_info | Variable | TX 🡪 RX | Parity/ECC on partitions of data flit |
| header\_ras\_info | Variable | TX 🡪 RX | Parity/ECC on partitions of header flit |
| acc\_sb\_ras\_info | Variable | TX 🡪 RX | Parity on partitions of merging payload flit |
| ctrl\_ras\_info | 1 | TX 🡪 RX | Parity on control/delineation fields |
| route\_info\_ras\_info | 1 | TX 🡪 RX | Parity on routing information |
| credit\_ras\_info | 1 | RX 🡪 TX | Parity on credit return signals |

## Miscellaneous

|  |  |  |
| --- | --- | --- |
| Signal | width | description |
| alt\_priority\_select | NUM\_PORTS\*NUM\_VC | OPTIONAL: Control lines to trigger alternate priority selection for virtual channels. Used for isochronous priority change for traffic which require this feature. |

# Functional requirements

This is a common packet switch module which can be used as a layer switch or packet router on a given layer. These two elements make up the core of CFG packetized NoC.

## Broad considerations

* Usage modes:

A single design should be configurable for use as a router, switch or streaming bridge

* Stamping, mirroring:

Design, parameterization, routing logic, IDs etc. has to be carefully considered with requirements for stamping and mirroring in mind.

* High speed:

The design should have options to simplify critical paths, enable optional pipelines etc. to allow the design to scale to high frequencies.

* New protocols:

Any additional features required to support new protocols like IDI, IOSF, AXI streaming must be considered.

* Low performance crossbar configuration:

It is desirable to have a configuration of the design to operate as a high radix, low frequency crossbar. This will allow deployment into low performance OCP, AHB etc. networks.

* Programmable routers:

Most configurations have traditionally been parameters, allowing synthesis optimizations. However a hardened NoC component would require a lot of these configuration options to be programmable. Stamping also requires some of these configurations to be wires. Hence a careful evaluation has to be done to classify configuration options as parameters or wire straps.

* Wire count reduction, options for lower area, reducing dynamic power consumptions are other general requirements.

## Module usage modes

### Number of ports

This module will be designed with a maximum of 32-ports. All ports are fully symmetric and each port can switch to any other port for a possible 32x32 switching. Several parameters are used to configure the module. These will be covered in more details.

#### 32-port router

A router has 4-directional links and 4-host links. Each of these can in-turn have a 1 to 4 physical ports, for a maximum of 32-ports. Any of the ports can be designated as the directional ports N, E, W, and S or host ports h0, h1, h2 and h3. In this mode, NoC router continues to be of radix 8. Switching to sub-ports within a directional or host link is statically setup through parameters.

#### N host x M layer Switch

32-ports available on the router should be flexibly split into host side and NoC side ports for switch/streaming bridge configuration. A maximum of 16-host ports and 16-NoC layer ports will be supported. Note that host ports can’t switch amongst each other, this is true for the layer ports also. This mode allows 16x16 switching.

#### N host x M layer Streaming Bridge

## Features

### Virtual channels

Each port has a maximum of 8 virtual channels. In a given cycle only a single VC is active on a physical port. Flits from different VCs can be interleave on the physical port from cycle to cycle.

#### VC Input buffers

Router has an input buffered architecture. Each virtual channel on each input port has a VC buffer. On a given input port, a single VC buffer is written in a cycle, but up to 4 VC buffers may be read and transmitted to output ports.

Input VC buffers need to support following options/configurations

* Flip-flop based VC buffer:

Each input VC has a dedicated flip-flop based VC buffer. This has to use CFG common module. Both registered and unregistered output options are required.

* RF based VC buffer: This will use an RF storage shared by all the VC enabled on that input port. RF will have a single write port and multiple read ports. Number of read ports is equal to the number of VCs enabled on the input port. Since VCs can be of different data widths, the RF will need to be sized to the largest width. Narrower VCs store data LSB aligned in their range of the RF. Even though they share a common RF, each read port/VC needs a register based prefetch stage to hide RF latency.

RF stage is bypassed to write into the prefetch stage, prefetch stage also issues reads to RF to keep itself full. RF FIFO module with prefetch stage is available in common modules and should be used.

Note: control and data fields may have to be separated into two different RFs to allow them to operate in different cycles in a pipelined manner. This will be covered in a later section.

#### Store and forward

CFG NoC is worm-hole routed, meaning that the flit at the head of a VC buffer always attempts to exit to its target output port if credit and other criterion are met. Under certain conditions, it is desirable to configure a VC buffer to have a store and forward behavior. In this case, a VC only participates for output arbitration after it has a configured amount of data or an end-of-packet. Store and forward mode can be enabled to prevent bubbles in a packet from propagating through locked VCs along the route of a packet and affecting throughput. Some example of cases where NS enables this option are: after clock crossing, after upsizing etc.

VC buffers must be implemented to support store and forward mode. Threshold occupancy for store and forward must be made programmable.

#### Shared input VC buffer

Support will be added for using a shared, dynamically allocated buffer pool for input VC buffer. Pool will have a set of shared credits which are available to any virtual channel and some dedicated entries per virtual channel. This allows independent, non-blocking forward progress for the virtual channels, while allowing a larger shared pool to handle credit loop latencies. A lower priority virtual channel could end up unfairly using up all entries in the shared pool, thus affecting through put for a higher priority virtual channel.

#### Non-Blocking input VC buffer

Logically every virtual channel buffer is organized as a FIFO structure, it imposes forced ordering on everything entering the virtual channel even if they can or require ideally to be able to leave out of order. Packets diverging out of a virtual channel to different output ports of a router encounter HOL blocking issues.

This feature proposes the option to allow each VC buffer to operate as a dynamically allocated structure maintaining logical FIFO queues for every output port reachable from that VC. Each of these output specific queues can send out a flit independently. This allows traffic diverging to different destinations/output ports on a router to be non-blocking of each other.

#### No input VC buffer

A mode is needed where an input port and VC do not have a VC buffer. Incoming flit request instead of writing the FIFO should undergo required transformation and directly participate in output arbitration. Grant from the arbitration instead of reading the FIFO and sending back credit, should be passed to the interface as a grant for the flit request. This mode is primarily intended for use in transmit switch host side interface. This transforms the credited interface to a request-grant interface. There is cycle of latency saved, but timing path is longer. Mode should be selectively deployed by NoCStudio. Would also be used when the router needs to be used as a crossbar.

When an input port is operating with no VC buffer, it is an uncredited request-grant interface. The upstream output port does not hold credits and the upstream output arbiter grants, only on receiving grant from the downstream router. Credit increment signal will be used as grant.

### Flit resizing

* Every input VC can upsize or downsize to a configurable ratio to each of its possible outputs
* All power-of-2 ratios from 1 to 16 have to be supported for upsizing and downsizing
* All logic and timing related to this has to get optimized away if there is no resizing involved.

The optimal place to do upsizing is at the input VC so that a low bandwidth input VC upsizing to a wider VC doesn’t block the wider VC till it has a full wide flit to send. However when a large packet undergoes upsizing, bandwidth of the output VC in inherently getting limited by the lower bandwidth input VC locking it. In terms of implementing upsizing, every input VC needs to instantiate an upsizer based on the maximum supported upsizing ratio from it. Flits are popped from the input VC buffer and loaded into the upsizer. Final input flit bypasses the upsizer and the entire upsized flit is ready to arbitrate for the output.

When downsizing, the optimal place to perform downsizing is at the output, to prevent the low bandwidth downsizing flow from blocking any higher bandwidth flow behind it in the input VC (Note: typically NocStudio avoids putting low bandwidth and high bandwidth flows in the same VC). However for large multi-flit packets, head-of-line blocking slows the input VC to the rate of the downsizing output VC till the packet completes. Ideally, per VC output buffers can be used to unload flits from the input VC and then downsize to the output physical link. However, input VC buffer of the downstream router can serve the same purpose if downsizing is moved to the next hop.

In terms of implementation, it is recommended to keep downsizing at the input. Head of the input VC buffer is used as the holding stage avoiding additional storage, and it is popped once the last downsized segment is sent to the output.

An alternative implementation scheme is to have a downsizing stage per output VC at the output port. Size of each of this stage has to be the widest input VC which sends to it. With the first grant from output, first downsized segment is directly sent out bypassing the holding stage. Remaining parts of the wider flit are loaded into the stage and downsized and sent on the output link in subsequent cycles. Note that multiple output VCs could be downsizing simultaneously, so some mechanism is needed to interleave their flits on the output physical link. Alternatively a single downsizer can be used for the output port, and it can occupy the output link till the entire downsizing process completes. This stage will backpressure other input requests till the downsizer is freed up.

### Routing

Routing involves identifying the output port of a packet so that an arbitration request can be raised. This computation is performed in a pipelined manner, i.e. on a given link, a router determines the output port a packet will take on the next router. Packet are routed using a source routing scheme.

**svn://sccj017606.sc.intel.com/trunk/doc/release\_docs/Archived%20Docs/NetSpeed%20Orion%20Router%20Microarchitecture%20Spec.docx**

#### Multicast routing

Refer multicast document.

**svn://sccj017606.sc.intel.com/trunk/doc/drafts/Router%20Multicast%20implementation.docx**

#### Layer, VC and route lookup

When used as streaming bridge, incoming transactions from a host interface perform a lookup to obtain the outgoing route to the destination.

Lookup key = {destination ID, QoS}

Lookup result = {route, transmit layer, output VC, destination interface};

### Arbitration request generation

A flit at the head of a VC buffer raises a request to the output port once certain conditions are met

|  |  |
| --- | --- |
| Conditions | Checks |
| Start of packet (sop) | Output VC is free + below checks |
| Non- sop | Output VC credit available |
|  | Input VC store-and-forward ready |
|  | Upsizing ready |
|  | QoS state conditions are met |

### Switching connectivity

Static parameter from NS will specify for each input port, input VC, the list of output ports that it can send transactions to. This parameters should be used to allow optimization of request path and data path logic.

### VC remapping

A flow can always remain in a single virtual channel from source to destination. VC remapping allows the flow to move from one VC to another on the routers along the path. The basic idea is to allow better sharing of existing VCs as long as deadlock and other mapping criterion are met.

Each input VC maps to one output VC on each output port (one to one mapping).

Each output VC can receive traffic from multiple input VCs (many to one mapping allowed)

**Per port IVC remap**

 {ivc3, ivc2, ivc1, ivc0}                                       // input VC

                              |

      {….p7, p6, p5, p4, p3, p2, p1, p0}                //output ports

                                           |

                                 {2-bit ovc ID}           // destination OVC ID

**Per port OVC remap**

 {ovc3, ovc2, ovc1, ovc0}                                  // output VC

                                  |

          {….p7, p6, p5, p4, p3, p2, p1, p0}   // input ports

                                               |

                                {ivc3, ivc2, ivc1, ivc0}       // 4-bit source IVCs bit map

At the input, every input VC for the target output port at its HOL, locates the output VC using the mapping parameter. This target output VC is used to mux the credit available and VC busy status.

Arbitration happens entirely in the input VC space.

At the output port, the selected input port, input VC is used to lookup the corresponding output VC and is used to generate the output valid, decrement credit and update busy status of the output VC.

### Flow control

#### VC flow control

Virtual channels use a credited flow control. Transmitting router comes out of reset with fixed preassigned credits for each downstream virtual channel, based on the depth of the corresponding VC buffer on the downstream router. Each flit transmitted on a link decrements the credit by one and each flit unloaded from the VC buffer causes a single credit to be returned upstream.

#### Link Credit

In addition to VC credits, every link can have a consolidated link credit. Upstream router checks for both link and VC credit before transmitting a packet. Link credit is optional on a link and can be deployed to allow a shared ILDC on a link.

### QoS features

In a CFG NoC, traffic flows are assigned to traffic classes. Traffic classes are isolated from each other using virtual channels of the NoC. Each traffic class can be assigned weights for differentiated bandwidth allocation. Each traffic class also has 1 of 4 possible static priority levels (0 being lowest and 3 being highest)

From router’s perspective, every VC has a static priority and packets in a stream convey the weight of the flow by appropriately marking the packets with a *qos\_state* bit. Stream of packets is made up of several *normal\_state* packets separated by *end\_of\_round\_state* packets. This information is used by router to perform weighted fairness arbitration.

### Arbitration

In a cycle, all Input VCs with connectivity to an output port can request to send a flit to the output port physical link. Output port arbitrates to select one of the input VCs. This arbitration is work conserving, in that a flit will be sent on the output link if any input VC has a flit to send. A hierarchical arbitration scheme is used and is described in detail in the document below.

**svn://sccj017606.sc.intel.com/doc/ccNoC/Router%20Arbitration.docx**

#### Isochronous priority switching

A traffic class has the ability to specify an alternate level of priority. A dedicated signal is used to indicate that a class should switch between its primary priority and alternate priority. This signal changes the priority of every virtual channel in the NoC through with that particular traffic class flows. This feature is primarily used for supporting isochronous traffic like display traffic which need the ability to elevate priority of their traffic through the fabric to prevent buffer underflow at end-points.

From routers perspective, this boils down to each input virtual channel having two priority levels and a dedicating signal which selects the priority level at which the VC requests in a given cycle.

#### Page locality enhancements

Arbitration points at multiple routers distributed in a NoC typically act as mixers of packets from different agents. This tends to remove any temporal or spatial locality that might have been be present in a stream of packets from an originating master. Preserving such locality can improve performance of slaves such as DDR SDRAM memories. These memories have performance overheads related to page open/close when requests are completely random without page locality.

Enhancements to router arbitration to preserve page locality is described in the below reference.

**svn://sccj017606.sc.intel.com/doc/ccNoC/Page%20Locality%20Enhancement.docx**

### Reset and Credit initialization

Credit counters on transmitting ends are reset to the maximum credit initialization value equal to the depth of corresponding VC buffer on the downstream router. Maximum credit value for each VC is static and is provided by NS through parameters. A router uses signal *link\_available* to indicate to its upstream router on that link, that it is out of reset and ready to receive data.



### Router mode: Routing

Routing is always performed one hop early for timing reasons. Routing has to be performed using a module which can be swapped out for different modes. Look up routing needs to have the option to make the entire table a programmable array of flops or RF.

#### Source routing

Source routing allows a flexible number of turns in the system without the cost of lookup tables at each node. However, the disadvantage is the number of routing information bits that needs to be carried around with the flits. Source routing scheme is described in the below document.

**svn://sccj017606.sc.intel.com/trunk/doc/release\_docs/Archived%20Docs/NetSpeed%20Orion%20Router%20Microarchitecture%20Spec.docx**

##### Turn router skipping

Turn routers which perform no arbitration are currently left in the NoC because they are required to compute next output port information using our source routing scheme. These routers incur area and latency cost of the VC buffers and logic, when they are really just pass through.

This features tracks the removal of all turn routers, irrespective of how many intermediate turns have been skipped between two connected router ports.

The figure below shows an example of point to point connection between two router ports with 1 to 4 intermediate turn routers which can be skipped. This effectively reduces the connection to passive wiring, with any pipeline inserted as required for timing.



Things to note:

* Odd number of turns are equivalent to a single turn. There is a change of axis with +/- direction on the new axis
* Even number of turns are equivalent to no turns. There is no change in axis but +/- direction can change on the axis
* Both pre-skip and post-skip routers themselves can be turn routers
* Pre-skip router computes a packets's next output port on the post-skip router

Proposed scheme for skipping turn routers:

* A co-ordinate encoding the skipped turns needs to be present in the source route for routes going over this link. Specific for the above example, all traffic exiting on the E port of the pre-skip router must have *Xskip* as the next co-ordinate. *Xskip* must be at least 1 more than the X co-ordinate of the pre-skip router. This co-ordinate is to ensure that all traffic arriving at the pre-skip router and destined for the post-skip router can take the E port as the next port.
* Two new parameters are required on the pre-skip router
  + Parameter [7:0] P\_TURNSKIP\_OPORT\_ENB = 8'd0: This parameter needs to be 1'b1 for every output port which has turn routers skipped in the link to the next router.
  + Parameter [8\*3-1:0] P\_TURNSKIP\_OPORT\_ID = {24{1'b0}}: This parameter renames the output port to designate it as the virtual output port from which the packet arrives into the post-skip router from the last skipped router on the link. In the above example, at the pre-skip router, output E needs to be renamed as N, W, S, and E in the 1, 2, 3 and 4 turn cases respectively.
* At the pre-skip router, when a packet is received on any input port with pre-skip output port as the next output port (East in the above example)
  + First the special skip co-ordinate (*Xskip* in this example) is dropped by left shifting the route co-ordinates
  + TURNSKIP\_OPORT\_ID is used to remap the pre-skip output port to the virtual output port to the post-skip router
  + With the above two changes, output port for the post-skip router is computed using the regular route/output computation logic
* If a host is directly connected to pre-skip router, it provides the output port even for traffic going to the skipped link. In this case OPORT\_ID remapping is required, but special co-ordinate is not required on the routing info. Hardware will not perform the step of dropping this special co-ordinate on any input port designated as a host port.

Example:

Consider the three turn skipped case in the figure above:

Packet arriving at any input port on the pre-skip router and targeting E output port toward the post skip router would have route info of the form

{*Xskip*, *Y, X* ...}

* E output is remapped as S output port
* *Xskip* is dropped and route info is changed to {*Y, X*, ... }
* Now the packet is travelling down on the Y-axis:
  + If Y coordinate of the post-skip router matches*'Y'*, then a turn is taken on the post-skip router toward the X-axis, E or W port decided by whether X coordinate of the post-skip router is greater or less than *X* respectively.
  + If Y coordinate of the post-skip router doesn't matches *'Y'*, then the packet continues down the S port of the post-skip router
  + If Y, X coordinate of the post-skip router match *Y, X*, then the packet exists to the host port

#### Lookup routing

Lookup routing allows flexible routing on the fabric but cost of the key for lookup (destination ID in a general case) is less than routing information in the source routing scheme. However routing tables are unique at each routing node and this makes stamping unfeasible.

An example lookup routing scheme.

* Route ID is generated at bridge based on a flow: {src, dst, qos, type, hash} from the exiting route lookup table. This table yields {layer, vc, route id}
* On a given router layer, a route ID uniquely identifies a route from a source to a destination.
* Nocstudio generates unique route IDs per layer using the minimum number of bits required
* At each router input, route ID is used to determine the output port on the next router
* {route id} 🡪 next output port
* Further compression of route ID may be possible. For example within a layer, there can be disjoint flows, i.e. all router on a layer may not see all flows on that layer.

Structure of the routing table:

* These lookup tables are programmed as parameters on each router
* For multiple instance of common router configs, this parameter table can be the global consolidated common table with each router also using its ID to access appropriate part of the table. (Router ID comes from pins).

**Programmable routing table has following implications**

* + Register address space has to be provisioned to program routing tables
  + After reset, the routing tables have to be programmed prior to any traffic flow through the NoC
  + Power gating would require either retention flip-flops or some mechanism for save & restore of the routing table contents.

### In-band routing information

Carrying source routing information on the sideband costs wires and flops. For lower cost networks which can tolerate latency and lowered through put, routing information can be carried in-band as the first few flits of every packet.

A serialization module can be instantiated on the TX side of a link for in-banding information at the head of every packet. A de-serialization module on RX side of the link can extract and make this information out-band again. These modules allow the routers at the end-points to always see normal out-band formats, while link traversal uses reduced wires due to in-banding of the information.

### Internal pipelining options

Every router typically has a single cycle of latency through it. This is the cycle in the input VC buffer. When output registering is enabled, latency through router is 2 cycles. Below diagram shows the important logic paths inside a router. For very high speed applications, critical routers might be optionally pipelined for performance at the cost of an additional cycle of latency through that hop.



Critical logic paths are:

1. Start : From the head of the input VC buffer

Through: request generation

Through: output arbitration

Through: output VC status update

End: output VC status and credit register

1. Start : From the head of the input VC buffer

Through: request generation

Through: output arbitration

Through: output crossbar mux

End: output register

1. Start : From the head of the input VC buffer

Through: request generation

Through: output arbitration

End: Input VC buffer read pointer

Note that fanout on the arbitration result can be significant from the loading of a very wide data crossbar mux.

#### Separate cycle for request generation and arbitration

#### Separate cycles for arbitration and output data muxing



This scheme requires the control path to have a fixed lead of a single cycle all through the fabric. Control path operation and arbitration complete in cycle 0, results are registered and data muxing happens in cycle 1. This option decouples data muxing from the critical path, yet doesn’t add an additional cycle of latency through the router, it utilizes the single cycle delay between control and data to maintain low latency, at higher frequency. Note however that this scheme doesn’t break any timing on the control paths, except reducing potential load/fan-out due to the crossbar mux selection.

Any link that has an asynchronous crossing cannot guarantee that the single cycle delay between control and data will be preserved across it. In these cases, logic has to be added on the read side of the crossing to re-establish the one cycle delay.

### Multiple payloads

The definition of multiple payloads is that each flit/packet can have more than 1 payload. Within a payload there can be more than 1 cell. Cell size is defined as the smallest granularity that you can downsized to. The number of cells is the same per all payload. There is only 1 common header or control signals for the payload(s).

#### Multiple cell size payloads

Using a single resizable payload section makes it difficult for diverse protocol packets to interoperate because of their differing cell size requirements. Partitioning transport links into multiple resizable sections enhances the ability to allow multiple protocol packets to coexist on common NoC layers.

Resizable payload sections have multiple categories with different cell sizes. A common cell sizes is 8-bits, to allow most data type fields to be handled globally across different agents and protocols. Other sections can be used to package non-data fields of transactions. An example for this would be the write strobe bit per byte of data. On links carrying multiple types of traffic, a superset of the data sections required by the protocols would exists. As the links diverge on to protocol specific regions of a NoC, only the data sections relevant for those protocol will exist on the links and others will be disabled/removed by NoCStudio. An example would be shared links which carry both write data and read response data. These links will have ‘data’ (8-bit cell) and ‘wstrb’ (1-bit cell) sections. When these links diverge to sections of the NoC carrying only read data, only ‘data’ (8-bit cell) category would exist. Section carrying only write data will need both ‘data’ and ‘wstrb’ categories.

* Each router link/VC can have multiple data sections each with specific cell sizes. Note, the cell sizes are not explicitly defined but through the parameters of number of cells and the width of the payload.
* A given link/VC can have a minimum of one data section. Width of this data section can be logical 0, implying that the link only carries header and no payload sections
* All data sections of a given input VC, resize by the same ratios
* All data sections have common end-of-packet and cell-valids. Specifically, the number of cells in each section for every flit is the same, even though the cell sizes may be different.
* All traffic through an input VC will be sized according to the number of payloads assigned.
* From an input VC, different output ports may have a different set of data sections enabled. Sections not present on an output should be appropriately generated off for synthesis optimization.
* When an input VC sends to an output with additional sections, the additional sections must be driven with zeros (this can potentially change in future)
* On a host interface, an agent can send or receive a set of data sections to/from the NoC.
* For an agent interface that does not contain data such as request, the interface data in router needs to be minimum of 1 and the VC width is also set to 1 in this case, however, the payload sections can be disabled and the payload widths can be set to 0.
* Link wires are optimized such that the payloads are concatenated together and starting from offset 0 of a given link. This allows sharing of wires and to have the minimum set of wires possible of a given link. To have this work throughout the network with byte lane gating is that, NocStudio needs to disable byte lane gating in the data pipelines and ILDC respectively if the data and wrstrb link wires are overloaded with other payloads. This is because data pipelines and ILDC assumes data is payload0 and wrstrb is payload1.
* For a given link targeting input VCs with different width, the payload’s offset from the input link are the same although the real width of the payloads differs for the respective VCs.

**NocStudio model of this feature**: Hardware in general can be configured to enable multiple payload sections as described above. Here is how NocStudio would deploy this feature in the construction of NoCs

* There will be no direct user interface for this feature using props or traffic attributes
* Every protocol bridge design will statically/internally specify the need for this feature during NocStudio integration phase
* Every protocol bridge design will specify the payload sections it requires for each of its host interfaces. All interfaces of a given bridge and type will utilize the same set of payload sections across a NoC.
* Based on mapping of the interface traffics, NocStudio will enable required payload sections in the NoC routers
* Broadly most protocol will require: 8-bit cell size data payload, 1-bit cell byte strobe, 1-bit cell byte parity
* For payloads that are disabled, the expectation is for NocStudio to program the payload width and its associated payload ras info to ‘0’.

#### Cell repacking: Per VC cell size

* Each input and output VC of a router has an associated cell size. When an input VC sends to an output VC with different cell size, each cell is changed to the output cell size by padding or truncating at the MSB
* Each VC is sized to the largest cell size of traffic it carries
* Resizing is independent of cell size and is in terms of the number of cells
* This should allow NoCStudio to map traffic requiring different cell sizes on the same layer.

Example: an input VC with 4 cells of size A, going to an output VC of cell of 8 cells of size B. In this case the upsize ratio is 2x. Each input cell is resized from A to B. This is done by NoCStudio programming the upsize ratio parameter instead of the cell size parameter.

### Merging payload

Messages often have associated with every flit of data on an interface, fields that can be merged within the fabric as the associated data flits are resized. Merging these fields doesn’t cause any loss of required information in the message. An example of this is an error sideband bit associated with every beat of data from an agents. When multiple data flits are grouped during an upsize operation, their error bits can all be ORed to create the single bit error sideband associated with the upsized data flit. When the resized message is delivered to a destination, any associated error information gets preserved.

* On the interface, merge-sb is time aligned/associated with the data fields of the flit
* Width of merge-sb remains constant through the NoC for a given traffic flow irrespective of the data field widths
* Each router VC will have a merging payload of parameterized width. Multiple VCs on a physical link can have different merging-payload widths. VCs which are linked by a traffic flow will have the same width for merging-payload globally.
* When upsizing, the merge-sb associated with each data beat will be bitwise ORed to create the merge-sb of same width associated with the upsized flit
* When downsizing, merge-sb associated with the larger flit will be repeated on the smaller flits
* Merge-sb is an optional field and it should be possible to generate off form a link/vc
* Similar to multiple data payload sections, use of merging sideband is specified by bridge design.

### Header, data separation

Currently a router flit is a single unit spanning a single clock cycle on a router link. An option is required to have two separate parts in a flit which are delayed by a fixed number of cycles. These are designated as, control/header group and data group. Header group arrives early, writes into the input VC buffer and can start the arbitration process. Data group arrives a fixed number of cycles later and gets written into its buffer and is read and muxed out by the delayed arbitration result. This delay should be configurable through a parameter. Note that this is similar to the timing improvement scheme explained in 2.3.13.2.

Note that all data path operations including re-sizing occur in a delayed stage and corresponding control path operations happen fixed cycles earlier in the pipeline.

As described earlier, an asynchronous crossing on a link will cause the pipelined delay relationship between control and data groups to be lost. This will have to be re-established using additional logic on the read side of the async crossing.

* Control and data groups will have to be separated into different input VC buffers
* Clock gating of control and data FIFOs will have to be handled separately
* Bridges will specify header-data separation for their protocol host interfaces
* Based on these specifications and traffic mapping, NocStudio will arrive at a header-data separation value for each NoC layer.
* An RSSB switch instance can have different values for HDR-DATA separation on its layers and host interfaces.
* Each input and output port of an RSSB has a parameter for its HDR-DATA separation
* All VCs of an RSSB port have the same HDR-DATA separation value
* For an RSSB switch instances, only ports (layer <-> interface) with the same value of separation will talk to each other. This restriction must be enforced by NocStudio.
  + An RSSB output port will have a parameter specifying HDR-DATA separation. Inputs talking to it will comply with this value
* RSSB re-alignment logic:
  + At an input port, on the read side of input VC buffer, re-alignment between HDR and DATA should be optionally supported
  + Re-alignment option can only be set on credited router input ports and not rdy-valid ports
  + Re-alignment cannot be supported if bypass mode is enabled on the input port
  + On an output port, DATA can be delayed by a configured amount to establish a HDR-DATA separation.
* Protocol bridges will have to implement re-alignment logic to convert the host interface header-data separation value to the value of the NoC side header-data separation.
* On ready/valid interface to RSSB, once a header is granted the data part will be implicitly granted in a pipelined manner after the specified delay value.

#### Link signals and their classification in header and data

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Signal | Direction | Width | Flit field category | Description |
| flit\_valid | Input | NUM\_VC | Header | One hot signal indicating the virtual channel to which the current cycle’s transaction belongs |
| flit\_header | Input | HEADER\_WIDTH | Header | Sideband information applicable to the entire packet. |
| flit\_output\_port | Input | 3-5 | Header | Target output port for this |
| flit\_route\_info | Input | P\_ROUTE\_INFO\_WIDTH | Header | Source routing information for the packet |
| flit\_eor | Input | 1 | Header | End-of-round indication if weighted QoS is enabled. |
| flit\_sop | Input | 1 | Header | Start of packet |
| flit\_eop | Input | 1 | Header | End of packet |
| flit\_cell\_valid | Input |  | Header | Number of valid cells in sections of payload during EOP. |
| flit\_data | Input | DATA\_WIDTH | Data | Resizable payload sections.  Parallel payload sections using different cell sizes is supported. |
| flit\_merging\_data | Input | MERGE\_SB\_WIDTH | Data | Sideband information which can be vary per flit, but an OR function is applied to combine the header fields across flits of the packet. |
| credit\_increment | Output | NUM\_VC | NA | Multi-hot vector incrementing credit for specified virtual channel |
| link\_available | Output | 1 | NA | Level signal indicating to the transmitting end that the receiving port is ready to accept data. |

#### Header-Data elastic delay

An extension of the header data separation is the case where header and data have variable delay. On a given output port, arbitration will be performed on headers and selected header will be sent out. The order of arbitration or selected input port will be recorded into a FIFO. Head of the FIFO specifies the order in which data will have to be muxed to the output port. Data is sent on the output port when data becomes available in the input data buffer of the input port at the head of the arbitration order FIFO.

An example usage of this feature is to allow write command (AW), write data (W) to maintain the arbitrary timing relationship on the AMBA interface and progress through the network without being aligned and locked on sideband plus main band.

### Example usage of multiple payload sections

IDI master and slave bridges interfaces to NoC

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Interface | Payload 0  Cell size | Payload 1 Cell size | Merging payload | Sideband/  Header | Header/Data separation |
| C2U REQ | NA | NA | NA | Entirely on header | NA |
| C2U DATA | 8-bit data | 1-bit wstrb | poison | Header | N cycles |
| C2U RSP | NA | NA | NA | Entirely on header | NA |
| U2C REQ | NA | NA | NA | Entirely on header | NA |
| U2C RSP | NA | NA | NA | Entirely on header | NA |
| U2C DATA | 8-bit | NA | Error/poison | Header | N cycles |

CMI master and slave bridges interfaces to NoC

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Interface | Payload 0  Cell size | Payload 1 Cell size | Merging payload | Sideband/  Header | Header/Data separation |
| CMI RD REQ | NA | NA | NA | Entirely on header | NA |
| CMI WR REQ WR DATA | 8-bit | 1-bit wstrb/ Metadata per beat | Poison  1-bit per 32B | Header | N cycles |
| CMI RDCPL RDCPL DATA | 8-bit | Metadata per beat | Error and poison | Header | N cycles |
| CMI WRCPL | NA | NA | NA | Entirely on header | NA |

### Output registering

Each output port has a parametrized output registering stage. This stage adds an additional cycle of latency and requires +1 credit (corresponding remote FIFO entry) for maintaining full link bandwidth. This stage needs explicit clock gating to allow toggling only when a valid flit is output.

### Link pipelining

### Multi clock domain

A single router is always a single synchronous clock domain. However domain crossing can be performed on the physical link on each router port. A link can be a clock, power or voltage domain crossing. One of the following two different domain crossing modules can be instantiated by NoCStudio on a router link. Link credit can be enabled for handling backpressure from the shared link crosser.

#### Asynchronous domain crossing using ILDC

#### Mesochronous clock crossing on the link

Note: instead of an additional storage structure on a router physical link, existing VC buffers at the input port of a router can be used for clock domain crossing. This avoids the additional cycles of latency due to the link crosser. Further, overall total storage requirement can be lower. This is because, latency of link crosser would increase the size of each input VC buffer to maintain bandwidth under increase credit loop latency and this is in addition to the storage area of the shared link crosser.

### RAS features: Parity/ECC based protection

Router port fields use parity for error detection. Document below describes the details. Some changes are needed to support end to end transport protection in router/switch modes.

When operating in switch or streaming bridge mode, ECC generation, checking and correction needs to be supported on data and sideband fields. This logic has to be optionally instantiated on a router input or output link. When functioning as an RX host port, ECC error check and correction logic needs to be instantiated on the output link. When operating as a TX host port, ECC has to be generated prior to writing into the VC buffers. In both these cases, an extra cycle can be added for the ECC logic.

In router mode, ECC logic is not needed, however parity generation and checks on other fields are necessary.

Additional registers will have to be added to allow error injection into the parity/error check logic.

Reference 4 [https://subversion.assembla.com/svn/netspeed\_noc/trunk/doc/drafts/Safety & Reliability features.docx](https://subversion.assembla.com/svn/netspeed_noc/trunk/doc/drafts/Safety%20&%20Reliability%20features.docx)

#### Internal Hop-to-Hop transport fields

Some transport link fields can change hop-to-hop. These are parity protected. A check is performed at each hop and parity is regenerated if the field is modified at the hop.

|  |  |  |
| --- | --- | --- |
| Field | Location of check | Regeneration |
| Valid | | |
| i\_flit\_valid | Check as received from input link | Generate as sent on the output link |
| Delineation | | |
| i\_flit\_sop | Read side of the input VC buffer | At the input VC control block after resizing operation |
| i\_flit\_eop |
| i\_flit\_is\_eor |
| i\_flit\_cell\_valid\* |
| Packet routing | | |
| i\_flit\_route\_info | Read side of input VC buffer | At the input VC control block after route computation |
| i\_flit\_output\_port |
| Credit and flow control | | |
| i\_credit\_increment | On the output port as received from the link | On the input port as sent to the link |
| i\_link\_available |

#### End-to-End transport fields

|  |  |  |
| --- | --- | --- |
| Field | Location of check | Generation |
| Data/payload sections | | |
| i\_flit\_payload\* | At output link | At input link |
| Merging payload | | |
| i\_flit\_acc\_sb\* |  |  |
| Sideband/header | | |
| I\_flit\_header | At output link | At input link |

|  |  |
| --- | --- |
| **RAS** | **Covered fields** |
| i\_flit\_data\_ras\_info | i\_flit\_data |
| i\_flit\_header\_ras\_info | i\_flit\_header |
| i\_flit\_acc\_sb\_ras\_info | i\_flit\_acc\_sb |
| i\_flit\_ctrl\_ras\_info | i\_flit\_valid, i\_flit\_is\_sop, i\_flit\_is\_eop, i\_flit\_cell\_valid |
| i\_flit\_route\_info\_ras\_info | i\_flit\_route\_info, i\_flit\_output\_port |
| o\_credit\_ras\_info | o\_credit\_increment |

### Rate limiters

Each input VC’s request for arbitration is gated by a leaky bucket rate limiter. Rate limiters are to be deployed at TX host interfaces in switch or streaming bridge modes to limit the injection rate from the host interface. Programmable rate limiter available in the design repository needs to be used.

### Layer switch mode: QoS weight marking

### Clock gating

All large flop structures such as input VC buffer, output register stage etc must have activity based coarse clock gating. If there is a latency penalty associated with waking a clock up, then a hysteresis counter has to be used to enter clock gating when idle.

Each output port needs to generate an early indication to wakeup clock on the downstream router. This signal is called *early\_valid. early\_valid* must be asserted at least 1 cycle before any valid flit is sent on the link. Once idle, *early\_valid* can be de-asserted to gate clock at the downstream router. *Early\_valid* is registered at the receiving router and used to wake up the write clock of the input VC buffers.

For generating *early\_valid* at each output port several options are available

* Look at the head of the VC buffer, generate early valid for HOL output port, stall arbitration if output is not registered
* Look at the write into the VC buffer from the link and assert early valid for the target ports. De-assert if HOL of none of the buffers have the target port for hysteresis number of cycles.
* Generate downstream early\_valid speculatively from input early\_valids.

Note that assertion and de-assertion of early\_valid has to be based on whether output registering is present on an output port, whether hysteresis needs to be used for de-assertion to avoid any wakeup penalty etc.

#### Advanced clock gating features

* Speculative clock wakeup and shutoff for downstream ports
* SB info valid only on SOP
* Byte lane clock gating

### Low power features

Router has a *sleep\_req, sleep\_ack* interface with NSPS. When *sleep\_req* is received, once all ports and other internal states are idle, *sleep\_ack* is returned. At the point the router can be power gated. Router de-asserts *link\_available* on all its input ports to block neighboring routers.

### Physical design guidelines

* Minimize internal wiring
* Allow replication of critical flops
* Placement guidelines

### Support for multiple instance of a common configuration

All ports are functionally equivalent and two ports with matching parameters and settings can be interchanged. Each router port has a physical port ID, however NS can orient these routers flexibly in the mesh and use physical ports as directional or host ports, by making unique logical assignments to these ports on different router instances on the mesh.

* *oport* information carried in flits is a physical port ID, switching happens using the physical port ID.
* Start port and end port values carried in the source routing information is physical port ID
* Each router has four pins designating the physical port ID for each directional port: *cfg\_north\_phy\_pid, cfg\_east\_phy\_pid, cfg\_west\_phy\_pid, cfg\_south\_phy\_pid*.
* Each router also has pins for designating the physical port ID of the neighboring router along each of its output ports.
* Routing logic uses the above mapping to compute output physical port ID for the next hop

### Low cost interconnect or crossbar

* Topology that can be considered arbitrary connection of router ports
* Higher radix routers.
* Lookup routing to allow arbitrary interconnection of ports
* Router links can support uncredited, 1 or 2 credit flow control. Uncredited link effectively operates as a request/grant interface. This allows storage and latency cost associated with VC buffers to be reduced. Long combinatorial paths can span multiple routers, but can also be broken by specifying any link to use 1 or 2 credits.
* Optionally interlocking two layers, with main layer passing arbitration results from each arbitration point to the corresponding arbitration point on the interlocked layer through queues. The locked layer does not perform independent arbitrations. An example usage is AR/AW command layer, with each AW arbitration point passing selection to W layer, allowing W layer to perform data routing in tandem with corresponding AW.

(Details to be worked out).

* + Interface at the switches/endpoint
  + How about a mix of RDATA, WDATA
  + How to handle multiple VCs

### Unused logic optimization

* Lint and synthesis tool warnings reductions
* Auto-generation of waiver precursor

## Interface definitions

Virtual channels on a router port can be interleaved on the link and use credit based flow control. Each port can be operated in this default mode or configured to operate using a request-grant protocol. This mode is primarily for use on the TX and RX host interfaces with protocol bridges. This section describes these modes of the router ports

### Transmit request-grant interface

Transmit host interfaces are in principle router input ports. However instead of a credit based flow control, a request-grant protocol is followed for flits. Further, no input VC buffers are present.

#### Normal request-grant

An incoming request specifies the output port/layer and the output virtual channel on that layer to which the packet is targeted. A normal request-grant interface appears like an input port with a single VC, in that complete packets are sent one at a time. However, the VC corresponds to the output port/layer VC. In a normal router, input VC is statically mapped to a single output VC on an output port. However pseudo input VC on a TX host interface can map to a different output VC even on the same output layer for different packets. In effect this can be considered an input VC, where the incoming packet not only specifies the output port, but also the output VC on that port.

An alternative model for this, is to consider this as an input port with multiple pseudo input VCs. These input VCs correspond to the output VCs on the different output layers this TX interface sends to. However, there is no interleaving of flits among the input VCs on the physical interface.

#### Virtualization

Protocol agents like CMI and OCP have multiple host virtual channels which can send on NoC virtual channels. Host virtual channels must follow some mapping function to NoC virtual channels depending on non-blocking progress, dependency and sharing requirements.

1. Each host VC mapping to a dedicated NoC VC (one to one).
2. Each host VC mapping to multiple NoC VC based on some transaction attribute (one to many)
3. Multiple host VCs mapping to a common shared NoC VC (many to one)

Technically, it is possible for multiple host VCs to interleave on a given TX interface. However, there are scenarios where blocking and potential deadlocks can occur. Listing a few possible complications

* If a many-to-one mapping is used, one host VC can first get access to the NoC VC and lock it. When the second host VC is interleaved on the interface, it will not be able to make progress because the NoC VC is locked. Since credit is available, the protocol side might not remove this request, thus effectively locking the interface out.
* The above scenario can be prevented if many-to-one mapping from host VCs to NoC VCs is strictly disallowed on an interface
* Note that many-to-one mapping can still be allowed from different host TX interfaces.
* Another aspect to consider when allowing interleaving on a TX host interface is the requirement to maintain states for all the host VCs simultaneously interleaving on the host interface.
* However, if host VCs cannot interleave on a given TX interface and each host VC is required to complete its current packet before another host VC can send on the interface, then the host VCs are causing head of line blocking for each other. A host VC targeting a different NoC VC is also prevented from sending on the interface, even if the currently active host VC is unable to send due to lack of host data or lack of credit from its NoC side VC.

Based on the above considerations, following bullets capture **Specification for TX host interface**

* Credit availability status of every NoC side VC is exposed to the host interfaces so that protocol side logic may factor these into selecting the appropriate host VCs
* An Interface does not allow interleaving of packets. A full packet has to be sent out to the layer, VC before a new request can be sent on the interface
* Every host VC must use a dedicated TX interface
* Requester is allowed to change a request if it hasn’t been granted
* If downsizing or any other request state needs to be preserved, then TX switch will grant on the interface and hold the state internally
* A single resizing state is maintained at a time on an interface corresponding to the ratio from the interface to the current target layer, VC
* Sharing of a TX interface across multiple host VCs is possible, if the host VCs only send single flit transactions and any transaction posted on the interface has already checked for NoC side credits allowing it to make non-blocking progress in a timely manner.

### Receive request-grant interface

RX interfaces are router output ports which use a request-grant flow control for flits sent on it. These output port do not maintain VC credit counters.

#### Normal request-grant

A single virtual channel needs to be enabled on the output port. VC0 output valid must be used as the interface request signal

* VC remapping parameter must be used to map all the appropriate NoC VCs to interface VC0
* Hardware always indicates credits available on VC0 to the NoC side input ports. VC0 VC busy status has the normal functionality, blocking other input NoC VCs from the RX interface, till packet finishes from the first granted input NoC VC.
* Router output arbitration grant will be qualified with external grant before being passed to the input VC.
* On the RX interface, request (selected NoC VC) may change while grant is not received from the interface. Once a packet has started, VC locking ensures that no other input VC can start a new packet.

#### Virtualization

RX interface can also allow multiple host VCs to utilize the interface. This mode is used by AMBA slave bridge, OCP and CMI protocol bridges.

* Router output port VCs are considered as the host VCs enabled on that RX interface.
* No credit counters are maintained on the output port. Protocol side is expected to maintain host VC credit counters.
* On the RX interface, host VC credit available status signals are provided over the credit\_incr signal of the router output port
* VC remapping parameters are appropriately programmed to map between NoC VCs and host VCs on the RX interfaces
* An additional grant input signal is provided on the router output port, when it acts as an RX interface. This grant is used to qualify the output arbitration’s grant to input.
* Host VCs can be fully interleaved on the physical RX interface. However a mode is also supported, where there is no interleaving on the RX interface. Each host VC finishes a complete packet before a different host VC can send on the RX interface. Note that this mode introduces blocking and dependencies between the Host VCs.

## Miscellaneous

* Special bypass with end to end credit:

E.g. remove RX FIFO if single layer, single VC sends to a host port. No arbitration is needed, input traffic can directly be sent to the output host interface. Credit exchange can happen directly between the last router and the host.

* Per layer format vector

Each port has a formatting bit vector for each output port it talks to. This vector is used to compress/decompress the data. Primarily for use in the layer switch mode, a host interface can send a global format, but can be compressed to different formats on different TX layers.

* QoS End-of-round pass through from interface
* Fast bypass path
* Improved fast bypass path arbitration. Still only single input can bypass to output, but it is higher priority than the normal arbitration result.
* Sideband, data unified bus
* Better BV encoding
* VC to VC interaction for IOSF
* Redundant wires for remapping bad wires
* Full bypass of router col/row for manufacturing defect on router
* Group sideband and data into a single bus

### Pitfalls

* Normalized internal data bus wires should be minimal and based on the parameters
* Reduce use of local parameters inside a generate hierarchy. If present, have option to print instance (%m) and values to log file in debug mode.
* Avoid flops implicitly optimized away by synthesis.

# Implementation details

## Block diagram

The diagram below shows a top-level block diagram, with arrows indicating the main flow of flits through the rtl:



## Functional blocks

### Input VC buffer

### Input VC processing



### Output arbitration

### Output VC status

### Output XBAR

### Clock gating and Power gating

### Link pipeline

### Clock, voltage, power domain crossing (ILDC)

### Control and status registers

## Interfaces

## Parameter and strap configurations