Safety and reliability features

[1 Transport error detection and correction 3](#_Toc453060020)

[1.1 Introduction 3](#_Toc453060021)

[1.2 End to end transport protection 3](#_Toc453060022)

[1.3 Hop to hop protection 4](#_Toc453060023)

[1.4 End to end packet integrity 6](#_Toc453060024)

[1.5 End to end packet stream integrity 6](#_Toc453060025)

[2 Configuration register protection 7](#_Toc453060026)

[3 Bridge logic protection 8](#_Toc453060027)

[3.1 Duplicate and compare 8](#_Toc453060028)

[3.2 Parity on flop arrays 8](#_Toc453060029)

[4 Timeouts 9](#_Toc453060030)

[4.1 Response timeouts 9](#_Toc453060031)

[4.2 Interface timeouts 9](#_Toc453060032)

[5 Error reporting 10](#_Toc453060033)

[5.1 QUESTIONS 10](#_Toc453060034)

| Revision # | Author | Date | Notes |
| --- | --- | --- | --- |
| 0.1 | Joji Philip | 26 May 2016 | Initial Draft |

# Transport error detection and correction

For safe and reliable operation of a device, error free and fault-tolerant operation of the interconnection networks used in the device is crucial. Random faults can occur in the storage elements and wiring resources used by a system wide interconnect. Such errors must be detected and corrected when possible and all uncorrected errors must be notified to system software for intervention.

## Transport error detection and correction

Data is exchanged between agents through the NoC using a packet protocol. Different levels of transport error resilience can be configured for the NoC transport infrastructure. Packets transported over the NoC can be broadly viewed as comprising of three fields.

1. Data field: This is usually some power of two multiple of an integer number of bits. Interfaces to agents and data part of NoC links belong to this category. This part can undergo upsizing and downsizing while being transported across the NoC.
2. Sideband field: An example of this is AW command carried on sideband of the AWW channel. This field does not undergo resizing through the network.
3. Packet control fields: A packet also has signals for routing, delineation, credit return etc.

## End to end transport error checking

Any flow through the NoC can be configured to provide error checking using ECC or parity. ECC uses hamming code with additional parity bit to provide SECDED code. This code can correct single bit errors and dectect double bit errors in a block of data. Parity only allows detection of odd number of bit errors.

### Data protection: Per flit ECC

On a transmit bridge for every layer with ECC protection enabled, ECC is calculated over each data flit and sent along with the flit. At the receiving end, ECC is used to detect and correct any errors in the data flit received from NoC layer before delivering to the receive host interface.

* Granularity of data width over which ECC is computed is derived and configured by NocStudio globally on each NoC layer. Smallest possible granularity is the CELL\_SIZE configured on that layer. However if the narrowest interface communicating on that layer or narrowest NoC link on that layer is *N*\*CELL\_SIZE, then this must be the granularity over which ECC is computed. Note that narrower granularity increases area overhead for ECC but provides higher detection and correction coverage. Configured granularity is a power-of-2 multiple of cell size on a layer.

An example is regbus layer, where each interface is typically 36-bits (4-cells), but NoC links can be as narrow as 9-bits (1-cell) if downsizing is performed. In this case, ECC granularity would be 9-bits.

* User can specify a maximum granularity over which ECC is to be computed. Consider a NoC where all host interfaces are 512-bits with no downsizing in the NoC. In this case, the default ECC calculation granularity will be 512-bits. However, the user may choose to specify a smaller granularity of 64-bits for ECC computation to allow better timing performance.

In summary, global granularity selected by NocStudio for ECC computation will be the smaller value between narrowest link/interface width and user specified maximum granularity. Every cycle, Multiple ECC/Parity code words are computed in parallel, one for each ‘*granularity’* wide segment of data/sideband of the flit. Computed ECC is transported similar to data flits and will undergo upsizing and downsizing with its associated data flit. ECC generation at the transmitting end and detection and correction at the receiving end will add a cycle each to overall path latency.

### Data error detection: Per flit parity

As an alternative to ECC, user may configure parity to be transported with the data flits for detecting odd number of bit errors. Granularity of data width over which parity is calculated and transported is as specified for ECC. Parity based protection does not add latency to the path.

### Data protection: Transport of user provided ECC

Another alternative allows the use to generate ECC on the data and provide it on the interface using USER bits. In this case, NoC merely transports the ECC bits from transmitting to receiving end. Note that this option is only applicable to DATA flits which do not undergo any modification in the NoC. Command fields can be modified by the NoC and hence user provided ECC will lose its integrity.

User provided ECC should be provided per byte of data through the ‘P*er byte user bits’* interface. This is transported in the data cells and can hence undergo upsizing/downsizing in the NoC.

### Sideband protection: ECC or Parity

Similar to data, information carried in packet sideband will be protected end-to-end using ECC or parity. Sideband associated with an interface has the same width over the entire network, this field does not undergo upsizing/downsizing in the NoC. Sideband width is increased to the next multiple of ECC computation granularity using msb 0 padding. At the transmitting end, ECC is calculated on sideband segments at the selected granularity and at the receiving end, error detection and correction is performed.

## Hop to hop Error checking

If data or user side band is protected by ECC, then error check operations on these fields are only performed at the NoC endpoints. However if parity is applied to data and sideband, then parity error detection on these fields occurs at every hop of the network. Similarly, other fields of packet are covered by parity error detection at every hop of the network.

### Protection of packet control fields

These fields associated with every packet flit can undergo modifications as the packet is routed over the NoC. At the transmitter, parity is calculated over these fields and sent along with the flit. At every downstream hop, parity field is used to detect any error and may be recomputed for the next hop. A dedicated parity bit is used to protect each of these signal groups.

Packet delineation fields

|  |  |  |
| --- | --- | --- |
| Name | Width |  |
| flit\_valid | 4 | Flit valid |
| flit\_sop | 1 | Start of packet |
| flit\_eop | 1 | End of packet |
| flit\_bv | log2(DATA\_WIDTH) | This signal is present only on the router links. This indicates the number of cells valid in the EOP flit of a packet. |

Packet routing information

|  |  |  |
| --- | --- | --- |
| Name | Width |  |
| flit\_route\_info | P\_ROUTE\_INFO\_WIDTH | Routing information |
| req\_outp | 3 | Next hop output port |

Link flow-control credits

|  |  |  |
| --- | --- | --- |
| Name | Width |  |
| credit\_inc | 4 | Credit return |

### Error detection using e2e ECC/Parity

A user selectable option allows error detection (only) at each hop of the NoC, using the ECC or Parity fields carried to protect data and sideband end-to-end.

|  |  |  |
| --- | --- | --- |
| Name | Width |  |
| flit\_data | P\_DATA\_WIDTH | Packet data. Protected end-to-end using ECC or Parity. Optional per hop error check. |

|  |  |  |
| --- | --- | --- |
| Name | Width |  |
| flit\_usrsb | P\_USRSB\_WIDTH | Packet side band. Protected end-to-end using ECC or Parity. Optional per hop error check. |

### Implementation of parity check

Following diagram shows where the parity check and regeneration is implemented in the router in block



## Configuring Error checking

Error checking on a flow is specified in NoCstudio using *add\_traffic* commands.

*add\_traffic [ecc/parity] …*

This maps the flow on a NoC layer with ECC or parity based error checking. Every bridge and router on that layer is configured to perform the selected error checking operations.

*mesh\_prop errorcheck\_granularity*

This property allows the user to specify an upper limit for granularity over which ECC is to be computed on data and sideband fields. NoCstudio chooses the smaller value between this upper limit and the narrowest virtual channel data width on a the layer.

Parity results in one extra bit for every ‘*granularity’* wide data/sideband segment. ECC requires 5 to 10 extra code bits for every segment.

|  |  |
| --- | --- |
| Granularity (bits) | Code size per segment (bits) |
| 1-11 | 5 |
| 12-26 | 6 |
| 27-57 | 7 |
| 58-120 | 8 |
| 121-247 | 9 |
| 248-502 | 10 |

502 bits is the maximum supported value of granularity.

For coherent NoCs, error checking can be enabled through *add\_traffic* commands or as part of *setup\_coherency* command.

*setup\_coherency –[ecc/parity]*

## End to end packet integrity

Per flit error protection does not guarantee integrity of a complete multi-flit packet. Flits in a packet could get completely dropped without triggering the flit error detection mechanism. Additional protection is needed to check for integrity of complete packets exchanged on the NoC.

### Misrouted packets

Packets routed to an incorrect destination can be detected if the destination compares its ID with the destination ID carried in the packet sideband. Source ID from the packet can be checked out against an accept list or deny list of agents to detect if the packet arrived from an unexpected source. These checks require source, destination IDs to be carried in the packet side band and the associated area overhead.

### Bit interleaved parity

In this option, data bits of a packet are XORed diagonally and an extra flit is sent at the end with the results of the interleaved XOR. This results in addition of an extra flit for every packet, adding latency and reducing throughput. To ensure this can work with agents of different data widths, interleaved XOR operation is performed on a globally common granularity.

### Flit ID

An alternative to avoid insertion of an additional cycle is to carry a sequenced identifier with every flit of a packet. The identifier can be made up of {Source ID, Flit number}. Flit number is an 8-bit sequentially increasing count of flits in a single packet. Destination can detect an error by ensuring that the flits of a single packet are from the same source and in increasing sequence number. When flits are upsized or downsized in the network, the flit sequence numbers have to be regenerated for the outgoing packet.

## End to end packet stream integrity

Flit protection and packet protection can still not guarantee the integrity of a stream comprised of multiple packets. One or more packets in a stream can get dropped without triggering the above mechanisms. Stream level integrity is not implemented by the NoC. However, Orion/Gemini protocols allow detection of such errors through response timeouts implemented in the master and slave bridges.

## Interrupt line protection

For ASIL-D level it may be worth adding redundancy to the interrupt line so that any failure of the interrupt mechanism is protected against. Note however that the interrupt bits themselves are parity protected and the interrupt signal itself is just a OR wire with the driving state held as a level.

## Error injection

### Links

Add programmable registers on router and bridges to allow single or multibit errors to be injected on selective bits of selective links. This is needed to test the error detections mechanism in silicon.

### Interrupt

Make the interrupt status registers writable to test interrupt assertion in silicon.

# Configuration register protection

All configuration registers in the system will be covered with parity for error detection. When a register is written or at reset time, parity is computed over the data written into the register and stored along with it. Every cycle, parity is calculated over the content of the register and compared with the stored value. An error flag is asserted when a parity mismatch is detected.

Parity is computed at a byte granularity for the 32/64-bit configuration registers.

# Bridge logic protection

## Duplicate and compare

## Error log

* Parity errors will be logged as interrupt status bits
* Parity error detected on interface or protocol packet can be used to drop the transaction and insert SLVERR response. This can be done at both master and slave bridges.

## AMBA Interface parity

Parity information is provided on all external interfaces: Master on an interface, provides parity calculated over all fields of AR and AW commands and the slave provides parity for R and B. Data parity is provided per byte. Parity is valid for every beat of information on these interfaces. Parity is checked off at the receiving end of the same interface before any transformation is performed. Note that byte parity on data can be transported to the remote receiving end and checked since there is no data transformation performed in the bridges.

* AR, AW, W, R, B, CR, CD, AC and ACK are channels on ACE interfaces. Depending on the type of an ACE port, some of the channels may be absent. The exact set of signals on any of these channels also depends on the port type.
* Parity protection can be using a single bit for the entire interface or one bit per 8-bits of interface signals. Byte granular parity puts an exact dependence on the order in which the bytes are concatenated for computing parity. This sequence has to be documented so that both ends on an interface generate and check parity in a compatible manner.

*NocStudio property*: Byte granular parity/Single parity, Default: Single parity

* AR, AW channels: two optional parity signals added on the interface:
  + Address parity: Parity over AxADDR
  + Command party: Parity over all other Ax fields

*NocStudio property: address parity enb, command parity enb*

* CR, BRESP, RRESP, AC: Single bit or byte granular parity
* WDATA, CDDATA, RDATA : parity protection per byte of data

Parity is computed over {data, strobe, per byte user bits}

Since bytes are unmodified, computed parity over a byte can be sent in the cell over the entire network and delivered out to the interface on the remote receiving end.

Parity is checked off as data bytes are received on the source interface, but it is also sent through unchanged to the remote end to repeat the byte parity check on the remote receiving end.

“Valid” bits are also factored in parity generation and check.

* These interface parity checks and generation can be implemented in the interface valin and valout modules
* This also offers coverage of the ASYNC FIFO, skid stage and ratio sync buffer of the valin/valout block
* For information received on an external interface, checks are performed on the read side of valin modules
* For information sent to an external interface, parity is computed on the write side of the valout modules

### Functional specification:

Data interface R, W, CD always compute a parity bit per byte of data. Further W has a parity bit per byte of WDATA and corresponding WSTRB.

*intfparity\_addr\_per\_byte* (Default 1): This property specifies that ARADDRS, AWADDRS and ACADDRS parity are computed at per byte granularity by default. Alternatively, a single parity bit is generated for the complete address field.

*intfparity\_per\_byte* (Default 0): For control fields in each channel, by default a single parity bit is created. Optionally, the fields are arranged in a specific order, msbit padded to a multiple of 8-bits and a parity bit is generated for each byte of content.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Channel | Parity | *intfparity\_addr\_per\_byte* | *intfparity\_per\_byte* | Comments |
| AR | ARADDRPAR | 0: single parity bit  1(d): one parity bit per byte | - |  |
|  | ARPAR | - | 0(d) : single parity bit  1 : one parity bit per byte | All command bits except address |
| AW | AWADDRPAR | 0: single parity bit  1(d): one parity bit per byte | - |  |
|  | AWPAR | - | 0(d) : single parity bit  1 : one parity bit per byte | All command bits except address |
| W | WPAR | - | - | {WDATA[8\*i+:8], WSTRB[i]} |
|  | WCPAR | - | 0(d) : single parity bit  1 : one parity bit per byte | {WID, WLAST} : Not for this release |
| R | RPAR | - | - | RDATA[8\*i+:8] |
|  | RRESPPAR | - | 0(d) : single parity bit  1 : one parity bit per byte | {RRESP, RID, RLAST} |
| B | BRESPPAR | - | 0(d) : single parity bit  1 : one parity bit per byte | {BID, BRESP} |
| AC | ACADDRPAR | 0: single parity bit  1(d): one parity bit per byte | - |  |
|  | ACPAR | - | 0(d) : single parity bit  1 : one parity bit per byte | {ACSNOOP, ACPROT} |
| CR | CRESPPAR | - | 0(d) : single parity bit  1 : one parity bit per byte | CRRESP: Not for this release |
| CD | CDPAR | - | - | CDATA[8\*i+:8] |
|  | CDCPAR | - | 0(d) : single parity bit  1 : one parity bit per byte | CDLAST: Not for this release |
| RACK  WACK | RACKPAR  WACKPAR | - | 0(d) : single parity bit  1 : one parity bit per byte | Not for this release |
| All | VALIDPAR  READYPAR | - | 0(d) : single parity bit  1 : one parity bit per byte | Not for this release |

* User bits are not covered
* Internal non-standard signals on interface are not covered
* Gemini agent interfaces are not covered
* R5/R7 compatibility is not implemented
* Not supported on slave bridges with virtual AXI interface
* Not supported on interface with VDC
* Invalid cycles have no parity check. Will be eventually covered by valid/ready parity
* Invalid bytes on a valid cycle will generate and expect non X parity
* Tunnel as an agent doesn’t support parity
* Dau as an agent doesn’t support parity

Master bridges:

All except: IMG4, AHBLM, AXILITE

Slave Bridge

All except: AHB, AXILITE, APB, IMG2

Is disabled on all agent bridges for IOCB, CCC, DVM and LLC

AR Channel: AR byte parity fields (MSB to LSB: top to bottom, fields concatenated and MSB padded with 0 to nearest multiple of 8)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ACE | ACEL | AXI4 | AXI3 | AXI4L |
| ARID, P\_MST\_AID\_R\_WIDTH | ARID, P\_MST\_AID\_R\_WIDTH | ARID, P\_MST\_AID\_R\_WIDTH | ARID, P\_MST\_AID\_R\_WIDTH | 1’b0 |
| ARLEN, 8 | ARLEN, 8 | ARLEN, 8 | {4’d0, ARLEN} | 8’d0 |
| ARSIZE, 3 | ARSIZE, 3 | ARSIZE, 3 | ARSIZE, 3 | Log2(P\_R\_AXI\_DATA\_WIDTH/8) |
| ARBURST, 2 | ARBURST, 2 | ARBURST, 2 | ARBURST, 2 | 2’d1 |
| ARCACHE, 4 | ARCACHE, 4 | ARCACHE, 4 | ARCACHE, 4 | 4’d0 |
| ARPROT, 3 | ARPROT, 3 | ARPROT, 3 | ARPROT, 3 | ARPROT, 3 |
| ARQOS, 4 | ARQOS, 4 | ARQOS, 4 | 4’d0 | 4’d0 |
| ARLOCK, 1 | ARLOCK, 1 | ARLOCK, 1 | ARLOCK[0] | 1’b0 |
| ARREGION, 4 | ARREGION, 4 | ARREGION, 4 | ARREGION, 4 | 4’d0 |
| ARSNOOP, 4 | ARSNOOP, 4 | 4’d0 | 4’d0 | 4’d0 |
| ARDOMAIN, 2 | ARDOMAIN, 2 | 2’d0 | 2’d0 | 2’d0 |
| ARBAR, 2 | ARBAR, 2 | 2’d0 | 2’d0 | 2’d0 |

AW Channel:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ACE | ACEL | AXI4 | AXI3 | AXI4L |
| AWID, P\_MST\_AID\_W\_WIDTH | AWID, P\_MST\_AID\_W\_WIDTH | AWID, P\_MST\_AID\_W\_WIDTH | AWID, P\_MST\_AID\_W\_WIDTH | 1’b0 |
| AWLEN, 8 | AWLEN, 8 | AWLEN, 8 | {4’d0, AWLEN} | 8’d0 |
| AWSIZE, 3 | AWSIZE, 3 | AWSIZE, 3 | AWSIZE, 3 | Log2(P\_W\_AXI\_DATA\_WIDTH/8) |
| AWBURST, 2 | AWBURST, 2 | AWBURST, 2 | AWBURST, 2 | 2’d1 |
| AWCACHE, 4 | AWCACHE, 4 | AWCACHE, 4 | AWCACHE, 4 | 4’d0 |
| AWPROT, 3 | AWPROT, 3 | AWPROT, 3 | AWPROT, 3 | ARPROT, 3 |
| AWQOS, 4 | AWQOS, 4 | AWQOS, 4 | 4’d0 | 4’d0 |
| AWLOCK, 1 | AWLOCK, 1 | AWLOCK, 1 | AWLOCK[0] | 1’b0 |
| AWREGION, 4 | AWREGION, 4 | AWREGION, 4 | AWREGION, 4 | 4’d0 |
| AWSNOOP, 3 | AWSNOOP, 3 | 3’d0 | 3’d0 | 4’d0 |
| AWDOMAIN, 2 | AWDOMAIN, 2 | 2’d0 | 2’d0 | 2’d0 |
| AWBAR, 2 | AWBAR, 2 | 2’d0 | 2’d0 | 2’d0 |
| AWUNIQUE, 1 | 1’b0 | 1’d0 | 1’d0 | 1’d0 |

## Protocol packet parity protection

* Interface command from a master, undergoes modification during splitting, field overrides, address relocation etc.
* Once a protocol packet is formed it remains unchanged till it is delivered over the noc to the remote receiver
* On the receiving end protocol packet can further be modified for overrides, width conversion etc before being put on the interface
* Each protocol packet can be enabled to carry computed byte granular parity as an additional field of the system packet format. Parity is generated by the transmitter of the protocol packet and a receiver checks off parity on received protocol packet prior to any modifications
* If protocol packet check is enabled at a receiver, then all transmitters sending to it must calculate and include parity in the protocol packet
* Protocol packet parity protections offers coverage on the large VC buffer storage in the RX switch of the receiver
* Content sent on SB or DATA for AR, AW, R, AC, CR implement this protection
* Data packets WDATA, RDATA, CDDATA do not need these as they allow end to end parity protection

## Flop structure parity protection

### AID tables

When an entry is allocated into the AID table, parity is computed over the content to be stored there. Parity is stored in that entry, along with the regular content.

* When a valid entry is ready to be deallocated, its content is checked off against stored parity and any error is recorded.
* Such a parity error may be used to convert the response into a SLVERR response. However in the case of slave-bridge, the destination of the response might be corrupted as well, so the error is unrecoverable.

### Reorder buffers

* All contents written into reorder buffer will have parity computed and stored.
* When content is read from the reorder buffer, parity check is performed

## Protection for large static tables

These are likely optimized away to GND and VCC tie offs by the synthesis tools. As such they are not susceptible to single event/transient faults. So providing any error detection coverage is lower priority.

### Route lookup tables

* These tables are constant tables provided by nocstudio. Since these are fairly large, the constant nets may undergo non SEU errors.
* Nocstudio provides parity of each entry of the table, along with its content
* Logic will scan and check parity of each entry at reset time or periodically
* Decoding combo logic might have an error resulting in incorrect route lookup error

# Random musings

* Parity check on ECC info on routers
* Pass link errors through VC buffers for link ASYNC case handling
* Bridge protection. Pass through and check at host interface
* Rinfo parity at rx bridge
* clock gating of ECC pipeline stage
* Configure router input port to do ECC/Parity check.
* Allow parity/ecc/no errchk flows to operate on the same layer if completely disjoint.
* packet sideband field indicating to receving end to perform ecc/parity or no errchk
* Coverage of bridges (host to noc)
* Regbus layer coverage
* ECC/parity protection of protocol packet content AR, AW

# Timeouts

For Orion/Gemini NoCs following time out mechanisms can be implemented.

## Response timeouts

Programmable timeout is already supported in the master bridges. Timeouts will be implemented in Slave Bridge for requests outstanding to the slave device. Once a timeout is detected, a maskable interrupt is raised to alert the system software.

Following actions can be taken and can be individually enabled/disabled.

1. Send error response to master for timed out request.
2. Block the slave device. All incoming requests to the slave from the NoC are responded with error. All responses from the slave device are dropped.

## Interface timeouts

Following interface timeouts can be enabled

* Master bridge RREADY/BREADY timeout
* Slave bridge ARREADY/AWREADY/WREADY timeout

# Error reporting

On layers with error checking enabled, every router has status bits logging any parity error detected on data segment, user sideband segments, packet delineation fields, routing information and credit information on router links and within router virtual channel buffers. These status bits are sticky and raise an interrupt by default. These interrupts can be disabled by programming the interrupt masking registers for the status bits. Similarly, bridges log errors status with maskable interrupts for layers where error checking is enabled. Details of these are available in the register manual.

Add counting of ecc errors at the bridges.

## QUESTIONS

* Per flow selection of ECC or Parity protection?
* Detail of ECC flit signals and transport
* ECC generation block is user modifiable RTL
* USER provided ECC’s granularity
* Packet checksum details