Table of Contents

[1 Network Bist Introduction and Definitions 3](#_Toc37076654)

[1.1 BIST Coverage 3](#_Toc37076655)

[1.2 BIST Coverage 4](#_Toc37076656)

[1.2.1 FIFO Coverage for Single Beat Injectors 4](#_Toc37076657)

[1.2.2 Regbus Layer Coverage 4](#_Toc37076658)

[1.2.3 Control Signal Coverage 5](#_Toc37076659)

[1.2.4 Mesochronous Crossing Coverage 5](#_Toc37076660)

[1.2.5 Low Power Coverage 5](#_Toc37076661)

[1.3 BIST Failure Definition 5](#_Toc37076662)

[1.4 BIST Run Environment 5](#_Toc37076664)

[2 High-Level Customer Requirements 5](#_Toc37076665)

[2.1 Minimize Top-level Wires 6](#_Toc37076666)

[2.1.1 Ideal (but Not Well-Understood) Solution 6](#_Toc37076667)

[2.1.2 First Pass Solution 6](#_Toc37076668)

[2.2 Minimize Area 6](#_Toc37076669)

[2.3 Minimize Test Time 7](#_Toc37076670)

[2.4 Minimize BIST Complexity 7](#_Toc37076671)

[2.5 Maximize BIST Flexibility 7](#_Toc37076672)

[3 Sample BIST Flow 7](#_Toc37076673)

[4 NocStudio BIST Properties and RTL Parameters 9](#_Toc37076674)

[4.1.1 ILDC Parameters 9](#_Toc37076675)

[4.1.2 Data Pipeline Stage Parameters 9](#_Toc37076676)

[4.1.3 RSSB Parameters 9](#_Toc37076677)

[4.2 New Bist\_gen and Bist\_chk Parameters 10](#_Toc37076678)

[4.2.1 Common Parameters for Bist\_gen/Bist\_chk 10](#_Toc37076679)

[4.2.2 Bist\_gen Parameters 10](#_Toc37076680)

[4.2.3 Bist\_chk Parameters 11](#_Toc37076681)

[4.3 BIST\_CTRL Parameters 11](#_Toc37076682)

[5 BIST Register Definitions 12](#_Toc37076683)

[5.1 Bist\_gen Module Registers 12](#_Toc37076684)

[5.1.1 RSSB\_BIST\_PROFILE\_NUM 12](#_Toc37076687)

[5.1.2 RSSB\_BIST\_GEN\_RATE\_LIMIT 12](#_Toc37076688)

[5.2 Bist\_chk Module Registers 13](#_Toc37076689)

[5.2.1 RSSB\_BIST\_CHK\_NONFATAL\_ERR\_INT\_STATUS 13](#_Toc37076690)

[5.2.2 RSSB\_BIST\_CHK\_NONFATAL\_ERR\_INT\_MASK 13](#_Toc37076691)

[5.2.3 RSSB\_BIST\_CHK\_FATAL\_ERR\_INT\_STATUS 14](#_Toc37076692)

[5.2.4 RSSB\_BIST\_CHK\_FATAL\_ERR\_INT\_MASK 14](#_Toc37076693)

[5.2.5 FAULT\_ISOLATION (TBD) 14](#_Toc37076694)

[5.3 Bist\_cntrl Registers 14](#_Toc37076695)

[5.3.1 GLOBAL\_BIST\_CONTROL 14](#_Toc37076696)

[5.3.2 GLOBAL\_BIST\_ERR\_STATUS 15](#_Toc37076697)

[5.3.3 GLOBAL\_BIST\_ERR\_MASK 15](#_Toc37076698)

[6 BIST Packet Signal Definition 15](#_Toc37076700)

[6.1 BIST Packet Control Signals 15](#_Toc37076701)

[6.2 BIST Packet Flow Control Signals 16](#_Toc37076702)

[6.3 BIST Packet Header Payload 17](#_Toc37076703)

[6.4 BIST Packet Data Payload 17](#_Toc37076704)

[6.5 BIST Packet Acc\_sb Field 17](#_Toc37076705)

[6.6 BIST Packet RAS Fields 17](#_Toc37076706)

[7 High-Level BIST RTL Blocks and Interfaces 17](#_Toc37076707)

[7.1 Bist\_gen Definition 18](#_Toc37076708)

[7.2 Bist\_chk Definition 18](#_Toc37076709)

[7.3 Bist\_ctrl Definition 18](#_Toc37076710)

[7.4 Connectivity Diagram 18](#_Toc37076711)

[8 BIST Control Signaling 19](#_Toc37076712)

[8.1 Signalling for a BIST Run with Non-Fatal Errors 19](#_Toc37076713)

[8.2 Signalling for a BIST Run with Fatal Errors 20](#_Toc37076714)

[9 Open issues/Missing documentation 21](#_Toc37076715)

# Network Bist Introduction and Definitions

## BIST Coverage

Network BIST covers transport throughout the NoC. All the routers and links that make up the NoC on all layers, including regbus, should have the ability to be tested. BIST explicitly does not cover the protocol side of the master and slave bridges that surround the NoC or the connectivity from the protocol bridge to its switch.



In Figure 1, the light green area labelled NoC is covered by Network BIST. The blue protocol bridges are not covered by BIST, nor are the wires they use to connect to the local switch.

Figure 2 shows a more detailed picture of a single route’s layer/vc path from tx0 port of a master bridge switch to the rx0 port of a slave bridge, and the reverse trip back from the slave bridge to the master. Similar logic exists (but isn’t in the picture) for the other tx/rx ports if they exist. The green area again is covered by BIST. BIST does not cover the wires that connect the bridge to its switch.

The switch will have one bist\_gen module per tx port and one bist\_chk module per rx port. NocStudio will provide parameters to configure each bist\_gen module, including routing table information, and widths of relevant fields. It will provide a set of parameters to bist\_chk modules as well, including widths of relevant fields and the total number of flits expected from all routes to that bist\_chk module.



In the initial implementation, customers must explicitly kick off the bist\_gen blocks. This enables BIST to directly control how many BIST engines are running at once in order to satisfy test time/power requirements. As a possible improvement in the future, we could provide a hardware FSM to sequence through NoC BIST, running on a subset of the NoC in series. (See bist\_ctrl section below.)

In the initial implementation, all bridges/routes will have BIST. Customers will be able to enable BIST via a write to a config register. As a future improvement, we will add the ability to only have BIST modules on specific mission-critical bridges/routes.

## BIST Coverage

We’d like to cover all wires in the NoC and all locations--at their full widths--of all FIFOs in the NoC. NocStudio must ensure the number of flits in a packet covers all the wires when upsizing occurs. The following coverage holes exist in the initial implementation of BIST.

### FIFO Coverage for Single Beat Injectors

On paths from single beat injectors, it isn’t possible to guarantee that all locations of switch input port fifos are covered. As the injector cycles through the patterns and routes, various locations of the fifos will be covered, but no way exists to ensure that each location of the fifo gets tested. We may solve this in the future by coming up with a way for multi-flit packets to be sent on single-beat injector routes.

### Regbus Layer Coverage

The regbus layer has only credited flow control between the regbus master and its switch, whereas the normal switches use ready-valid flow control. This complicates implementing the same BIST on regbus layer as on the regular NoC layers.

A solution for this is TBD.

### Control Signal Coverage

Not all bits of the packet can be explicitly tested by a data pattern because they are used in the routing itself. Some receive coverage by default, because if incorrect the flit will be misrouted and the flit count will be incorrect. Section 6 describes each wire’s coverage in detail.

### Mesochronous Crossing Coverage

Our mesochronous crosser will not be covered by BIST because of the way its implemented (it must be written and read every cycle, with no backpressure possible). This isn’t currently an issue because nobody is using it, but may require a solution in the future.

### Low Power Coverage

BIST has to exist and run correctly in NoCs with our low power functionality enabled. Specifically, the BIST control module will need the ability to wake other power domains in order to run BIST successfully.

## BIST Failure Definition

There are two basic types of failures:

* BIST fails but can be reliably rerun (corrupted packets but no missing/misrouted ones).
* BIST fails but may not be able to rerun reliably because packets remain outstanding.

In particular, network BIST fails if any of the bist\_chk modules detects that:

1. The expected flit count does not equal the actual flit count at a bist\_chk module. This is fatal because there may be flits/packets outstanding in the network.
2. The bist\_chk module detects a potocol violation. This is fatal because there may be flits/packets outstanding in the network.
3. A flit or entire packet is corrupted, detected by pattern checking in the bist\_chk module. This is non-fatal because there are no missing packets.

We will distinguish between fatal and nonfatal BIST errors in our error-reporting. We provide no fault tolerance for NoC faults, other than ECC, which customers may not implement because of the cost.

## BIST Run Environment

Bist will run in at least two environments: manufacturing test, to screen defects, and during normal operation, for functional safety. There is no difference in how network BIST runs in these two environments. However, during functional safety runs, where normal traffic traverses the NoC concurrently with BIST traffic, it will require more time to complete BIST. The timeout value must be increased in order to avoid false failures during functional safety.

# High-Level Customer Requirements

We have two different sets of customer requirements for proposed network BIST functionality, one for BIST used in manufacturing, and one for BIST used in functional safety (FUSA) applications. We’ve analyzed both use cases and derived a unified set of requirements. (The spread sheet trunk/doc/drafts/Network BIST/BIST\_requirements.xslx contains a detailed list of requirements for each.)

The initial BIST implementation offers an inexpensive but very configurable architecture, using software to program how BIST will run both in manufacturing and the field. In future modifications, we may add in more automated control in the hardware, if the cost is justified.

## Minimize Top-level Wires

We need to be able to perform all necessary BIST communication with a minimal number of top-level wires.

### Ideal (but Not Well-Understood) Solution

Ideally, a single top-level BIST controller could kick off BIST without adding any top-level wires. The bist\_cntrl module could use the regbus layer to write to each bist\_gen/bist\_chk module using existing regbus connectivity.

After BIST completion, the bist\_cntrl logic could use CSR accesses to gather BIST fault isolation results from each individual bist\_chk module to set a global bist\_err\_interrupt signal, and provide the identity of the failing bist\_chk module without any top-level wires.

This requires a clear understanding of the desired BIST behavior and NocStudio properties to be able to describe that behavior, plus an understanding of how much cost (area and test time) customers are willing to incur for it.

### First Pass Solution

For the initial implementation, we will have a global bist\_enb\_n (active low) wire driven from a CSR in the bist\_cntrl module. It will go to every bist\_gen/bist\_chk module. Customer software will write to the bist\_cntrl CSR, trigger the start of BIST. Customer software will also be able to write to CSRs in the bist\_gen logic to refine which bist\_gen blocks are active at any one time.

To report BIST pass fail results, we will use a bist-specific interrupt signal. Each bist\_chk module drives an output called bist\_err, which resets to 1’b0. It asserts its bist\_err\_status output if it has a failure. These wires connect to a bist\_err\_status register in the RBM, with one bit per bist\_chk module. If any bits are set in an RBMs bist\_err\_status register, it asserts a bist\_err\_interrupt signal. NocStudio will OR these signals from each RBM together to create a pin on ns\_soc\_ip for the bist\_err\_interrupt.

This creates a number (proportional to the number of bridge tx/rx ports in the system) of new top-level wires but these wires are not sensitive to top-level metastability/timing issues because a CSR drives each of them, and they change state very infrequently.

## Minimize Area

Implementing NoC BIST requires a per-tx-port bist\_gen and per-rx-port bist\_chk module in each bridge switch in the NoC. The bist\_gen/bist\_chk blocks will contain control registers (CSRs) and logic (state machines) to implement their function. Since there are so many of these modules, their area must be as small as possible. We need a minimal set of CSRs and minimal state tracked to achieve BIST requirements.

A future BIST enhancement must allow for a way to specify a set of routes that have BIST coverage, to minimize area for customers who may only want BIST on certain critical specific links. The customer should not have to pay the area penalty to have BIST everywhere if they don’t need it.

The bist\_cntrl module should also minimize area, but since there is one per NoC, it has a dramatically smaller overall impact on area than the bist\_gen/bist\_chk modules.

## Minimize Test Time

Time on the tester during manufacturing test costs money, so we need to architect a way to test as quickly as possible, while staying within customer voltage/power boundaries. Until we better understand the sequencing requirements, we’ll leave this completely in the customer’s control. Their software will enable BIST testing and control which bist\_gen modules run concurrently by defining BIST profiles at NoC creation time. The default mode will be to run all bist\_gen blocks in parallel, but it will be possible to enable specific bist\_gen blocks via software writes to CSRs.

## Minimize BIST Complexity

Make it simple to kick off BIST and get results. The goal to kick off manufacturing BIST is one reg write. Getting results should be simple as well. For FUSA BIST, customers can bear more configuration complexity—for example writes to bist\_gen CSRs to turn off BIST on certain routes.

## Maximize BIST Flexibility

Customers requested the ability to individually enable/disable BIST testing on routes. Because we want to detect missing flits by a packet count, we can’t provide infinite configurability. Instead BIST profiles, where individual routes are disabled for some profiles, allows customers to define the flexibility they require. Profile 0 is always the most complete form of BIST defined for a given NoC, and is meant to satisfy the requirements of manufacturing BIST.

# Sample BIST Flow

The diagram below illustrates a sample execution flow for BIST.



All bist\_gen modules will run by default during manufacturing BIST. Software can disable BIST on some routes using profiles, if desired. Customer software can choose to decrease the timeout value that signals BIST completion when routes are disabled via profiles.

# NocStudio BIST Properties and RTL Parameters

BIST will require new NocStudio properties and parameters. In addition, the bist\_gen and bist\_chk modules will need to know some already existing parameters specific to formatting NoC flits/packets.

Specifically, NocStudio will add:

mesh\_prop enable\_bist

Allowed values: yes, no

Default value: no

If yes, enables BIST on the NoC. Instantiates BIST modules and control throughout the NoC. If no, the NoC does not support BIST.

More properties may be added later, as we receive customer feedback and refine the BIST architecture.

For all the parameters below, default/suggested initial values are supplied in parentheses after the parameter definition, to enable RTL development to begin without being dependent on NocStudio updates to set the parameters with real values.

For the initial implementation of BIST, if BIST is enabled, it is enabled on all routes in the NoC. We may add support for BIST on only some routes later.

### ILDC Parameters

The link clock cross fifos need storage space for the BIST bit if BIST is enabled on the NoC. The parameter P\_BIST\_PRESENT (Boolean) should be set on both the read and write wrappers for this module. Also, if BIST is present, the parameter P\_LINK\_CLK\_CROSS\_FIFO\_DATA\_SIZE\_LIST will need each entry to be 1 bit wider.

### Data Pipeline Stage Parameters

NocStudio must add the Boolean parameter P\_BIST\_PRESENT to all instantiations of ns\_noc\_data\_ppln modules.

### RSSB Parameters

Because BIST will use the RSSB register space, RSSB will need some new parameters:

* P\_INPUT\_PORT\_BIST\_PRESENT: vector of binary values that controls the presence of bist\_gen registers for that RSSB input port. ({P\_MAX\_NUMBER\_OF\_PORTS{1’b1}} if bist is enabled—all enabled ports get BIST.)
* P\_OUTPUT\_PORT\_BIST\_PRESENT: vector of binary value that controls the presence of bist\_chk registers for that RSSB output port. ({P\_MAX\_NUMBER\_OF\_PORTS{1’b1}} if bist is enabled—all enabled ports get BIST.)

These parameters control the instantiation of per-port BIST logic in the rtl and the existence of programmable control registers.

The following parameters control the profile register in the RSSB:

* P\_BIST\_REGS\_PRESENT: Boolean to tell the regs block to instantiate BIST registers.
* P\_BIST\_PROFILE\_WIDTH: integer value describing the number of bits required to hold a programmable profile number (The ceiling of the log base 2 of P\_BIST\_NUM\_PROFILES.) (1 bit default width, supporting two profiles.)
* P\_BIST\_PROFILE\_RESET\_VAL: The reset value of the shared profile register in the RSSB. (Default is {P\_BIST\_PROFILE\_WIDTH{1’b0}}.)

This parameter enables BIST rate limiting:

* P\_BIST\_GEN\_RATE\_LIMIT\_ENB: binary value that enables rate limiting per input port. (Default is ({P\_MAX\_NUMBER\_OF\_PORTS{1’b0}}.)
* P\_BIST\_GEN\_RATE\_LIMITER: 21 bits per input port ([671:0]) that contain the reset value for the rate limiting registers. (Default value is all 0s—rate liming disabled.)

## New Bist\_gen and Bist\_chk Parameters

### Common Parameters for Bist\_gen/Bist\_chk

NocStudio needs properties to set the following new parameters on both bist\_gen/bist\_chk modules. Suggested initial constant values in parentheses follow some of the parameter descriptions, just to enable RTL development to begin without being dependent on NocStudio updates.

* P\_BIST\_NUM\_PATTERNS: integer number of test patterns for BIST. (2)
* P\_BIST\_PATTERN\_WIDTH: integer bit width of BIST test patterns (all patterns have the same width). (2)
* P\_BIST\_PATTERNS: a concatenated list of BIST test patterns. ({2’b01, 2’b10})
* P\_BIST\_PROFILE\_WIDTH: Same parameter as programmed on the RSSB, to size the profile input wires from RSSB module. (1)

### Bist\_gen Parameters

These parameters are specific to bist\_gen:

* P\_BIST\_ENABLE\_PER\_PROFILE: concatenated list of binary values, one per profile, turning on/off bist\_gen for that profile. (Default is {1’b1, 1’b1}, both profiles are identical and have BIST everywhere.)
* P\_BIST\_NUM\_ROUTES: Integer number of entries in P\_BIST\_ROUTES for indexing through the table.
* P\_BIST\_ROUTES: a concatenated list of route\_info, lyr\_vc info, outp info, and number of flits to send for that route, one set per route. Ultimately also needs to include power domain information so that BIST can not generate packets for routes that have gated power domains along them. (Note: The parameter is shown as a table, but it’s actually a concatenated list of routes, indexed through by the sum of the widths of all the fields, because Verilog doesn’t support two-dimensional parameters.)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Strap | Entry | Profile | RouteInfo =  {first\_outp,route,exit\_outp} | Outport | VC | # of Flits |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |



Note that to match the way the bridges send to the RSSB switch the RouteInfo field above needs to hold this concatenated string: {{first\_outport}, route, exit\_outp}}Width parameters for all fields in the table above so that RTL can index through the individual fields:

parameter P\_STRAP\_WIDTH = 4;

parameter P\_BIST\_PROFILE\_WIDTH = 1;

parameter P\_ENTRY\_WIDTH = 3;

parameter P\_BIST\_ROUTE\_INFO\_WIDTH = 15; //first\_outp + route + exit\_outp

parameter P\_OPORT\_WIDTH = 4;

parameter P\_VC\_WIDTH = 3;

parameter P\_NUM\_OF\_FLITS\_WIDTH = 8;

NocStudio will calculate the correct widths for these, so no default value exists.

The RTL will calculate the width of one route by summing P\_STRAP\_WIDTH, P\_BIST\_PROFILE\_WIDTH, P\_ENTRY\_WIDTH, P\_BIST\_ROUTE\_INFO\_WIDTH, P\_VC\_WIDTH, and P\_NUM\_OF\_FLITS\_WIDTH. It will use this width to index through the table.

### Bist\_chk Parameters

These parameters are specific to bist\_chk:

* P\_BIST\_NUM\_FLITS\_PER\_PROFILE: Concatenated list of values describing the total number of BIST flits that bist\_chk module should receive for each profile. (NocStudio will calculate this.)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | Strap | Profile | Flit Count |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

* P\_STRAP\_WIDTH: integer width of the strap field.
* P\_BIST\_PROFILE\_WIDTH: integer width of the Profile field.
* P\_BIST\_FLIT\_COUNT\_WIDTH: integer width of the flit count fields in P\_BIST\_NUM\_FLITS\_PER\_PROFILE. (NocStudio will calculate this.)

## BIST\_CTRL Parameters

BIST requires a new control module instantiated by NocStudio at the top-level of the NoC. These parameters must be programmed on the new bist\_ctrl module:

* P\_BIST\_CTRL\_NUM\_RSSBS: Integer number of RSSB modules with bist\_chk modules attached, used to size the bist\_err\_status/mask registers. (NocStudio will calculate this.)
* P\_BIST\_CTRL\_NUM\_CHECKERS: integer number of bist\_chk modules in NoC. Used to size the bist\_done bus entering the bist\_cntl module.
* P\_BIST\_CTRL\_NUM\_ERR\_REGS: integer number of error regs required in bist\_ctrl to hold bits for all RSSBs with BIST. Used to instantiate the CSRs. (NocStudio will calculate this.)
* P\_BIST\_TIMEOUT\_WIDTH: Integer width of timeout counter (number of flops present for programming).
* P\_ BIST\_TIMEOUT\_RESET\_VAL: Integer number of thousands of clock cycles after which BIST should be complete. Used as the reset value for the timeout field in the BIST\_GLOBAL\_CONTROL register. This value should be programmed to some value (conservatively, much) bigger than we could ever reasonably expect to be required so as to avoid false failures. (NocStudio will calculate this.)
* P\_BIST\_DONE\_ASYNC\_MASK: vector of Boolean flags, width of P\_BIST\_CNTRL\_NUM\_CHECKERS, one bit per bist\_done signal coming into bist\_cntl. If set, the corresponding bist\_done signal is asynchronous to bist\_cntl, and synchronizers will be instantiated for that wire.

# BIST Register Definitions

All bist\_gen and bist\_chk registers will exist in reserved slices of the register space of their RSSB. To minimize flops, input port BIST registers will only physically exist in the RSSB if P\_INPUT\_PORT\_BIST\_PRESENT is true for the port in question. P\_OUTPUT\_PORT\_BIST\_PRESENT enables the existence of bist\_chk registers.

NocStudio will only instantiate the bist\_ctrl module if BIST is present in the NoC, so it will not have any area if BIST is not present.

## Bist\_gen Module Registers

The bist\_gen module reserves the slice of RSSB address space from 0x3E00 to 0x3EF8 for per-port CSRs. Network BIST can operate without any of these bist\_gen registers. NocStudio must connect the bist\_profile and bist\_rate\_limit pins on bist\_gen to all zeros if the RSSB has no registers.

### RSSB\_BIST\_PROFILE\_NUM

This CSR indicates which BIST profile is currently running. It will reset to zero, indicating manufacturing BIST—the most complete form of BIST designated for this NoC—is selected. The bist\_gen and bist\_chk blocks share the profile number output of the RSSB. Bist\_gen uses it to index into the P\_BIST\_ENABLE\_PER\_PROFILE vector to determine if it should send BIST packets. Bist\_chk uses it to index into the P\_BIST\_FLIT\_COUNT\_PER\_PROFILE vector to check for missing flits.



### RSSB\_BIST\_GEN\_RATE\_LIMIT

The existing rate limit register for the input port will limit both BIST and regular traffic. Limiting just BIST traffic requires a separate rate limit register just for BIST traffic (21 flops per bist\_gen module). The parameter P\_BIST\_GEN\_RATE\_LIMIT\_ENB will control whether or not this register exists. One register will exist per connected bridge port (a maximum of 4 registers for currently supported bridges).



## Bist\_chk Module Registers

The bist\_chk modules reserve the slice of RSSB address space from 0x3F00 to 0x3FF8.

Network BIST can’t preserve error status without these registers, so an RSSB with bist\_chk modules attached to it must have registers. These can be relocated to the neighboring bridge as a future improvement, to enable register-free RSSBs, with the rest of the bridge-related registers like the functional rate limiters.

### RSSB\_BIST\_CHK\_NONFATAL\_ERR\_INT\_STATUS

This register indicates which bist\_chk module on this RSSB detected a non-fatal error (corrupted packet). An OR of all bits in the GLOBAL\_BIST\_NONFATAL\_ERR\_STATUS register will create a non-fatal bist\_err interrupt.



### RSSB\_BIST\_CHK\_NONFATAL\_ERR\_INT\_MASK

This register allows masking of BIST nonfatal errors on a per-port basis.



### RSSB\_BIST\_CHK\_FATAL\_ERR\_INT\_STATUS

This register indicates which bist\_chk module on this RSSB detected a fatal error (protocol error or mismatching flit count). An OR of all bits in the GLOBAL\_BIST\_NONFATAL\_ERR\_STATUS register will create a nonfatal\_bist\_err interrupt.



### RSSB\_BIST\_CHK\_FATAL\_ERR\_INT\_MASK

This register allows masking of BIST fatal errors on a per-port basis.



### FAULT\_ISOLATION (TBD)

In the future we may choose to provide fault isolation information (failing pattern, etc). This depends on customer input about its value vs its cost.

## Bist\_cntrl Registers

BIST implementation requires some new configurable state registers (CSRs) in the bist\_cntrl module.

### GLOBAL\_BIST\_CONTROL

Bit 0 of this CSR globally enables BIST everywhere that BIST exists. Bit 0 resets to 0. If software sets bit 0 to 1, it turns on all bist\_gen/bist\_chk modules. This is the only register write required to kick off manufacturing BIST. If software wants to turn off BIST on some routes using LOCAL\_BIST\_ENB registers, those writes should be done before enabling bist globally.

The upper bits of the register contain the timeout in thousands of clock cycles (2^10 = 1024). NocStudio should write this to something much larger than any expected real requirement, so that customers can make it a smaller value but not a larger one. P\_BIST\_TIMEOUT\_WIDTH will define where there are flops for this field.



### GLOBAL\_BIST\_ERR\_STATUS

This register aggregates the results of a BIST run. It satisfies the customer requirement to minimize register reads to obtain BIST results. It has one bit for non-fatal errors and one bit for fatal errors, and is paired with the mask register below. NocStudio will OR together all the RSSB fatal\_bist\_err signals and provide that as an input to the bist\_ctrl module. It will do the same for the RSSB nonfatal\_bist\_err outputs.



### GLOBAL\_BIST\_ERR\_MASK

This register is used to mask off BIST error reporting for fatal or non-fatal errors. To disable the fatal/nonfatal bist error interrupts, software can set bits in this register.

### 

# BIST Packet Signal Definition

## BIST Packet Control Signals

The bist\_chk module needs the ability to distinguish BIST flits from regular flits, so BIST requires a new bit in the header. Also, this provides a way to enable customer-desired extensions to the BIST flow--for example, parity and ecc errors could be forced on BIST packets to verify that error detection/correction logic and error reporting logic works correctly. This new BIST bit is 1 for each flit that belongs to a BIST packet, and 0 for normal NoC traffic.

The new bit must be retrofitted into all the existing RTL blocks that generate/consume/pass-on the flit headers. Also, it must be included in the parity calculations/checks for NoCs that have parity enabled.

To save area, the BIST bit should not exist if BIST isn’t enabled on the NoC, so all existing RTL must be modified using generate statements that create two different forms of the headers—one for NoCs with BIST, and one for NoCs without.

With the BIST bit added, the packet control fields are:

|  |  |  |
| --- | --- | --- |
| 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\_bist | 1 | Current flit is part of a BIST pkt, not normal network traffic. |
| 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. |
| flit\_route\_info | ROUTE\_INFO\_WIDTH | Route Information for this flit. The same value for the entire packet, but modified at hops in the NoC as flits progress to their destination. |
| flit\_outp | OUTPUT\_PORT\_WIDTH | Output port for next hop, between 3 and 5 bits wide. |

The packet control fields are explicitly excluded from BIST pattern coverage because they are used to route the packet. However, if the wires carrying these bits are broken, it should result in misrouted packets and the BIST test will still fail.

## BIST Packet Flow Control Signals

BIST explicitly doesn’t cover the credit\_increment and link available signals because they control the propagation of packets through the NoC. If these wires were broken, it would likely cause a BIST failure because packets wouldn’t advance and the flit counts would be incorrect.

## BIST Packet Header Payload

NoC flits may optionally have a header field, also known as user sideband. BIST packets that include a header field will replicate a test pattern across the header. and will be self-checking—the packet will carry enough information for the checker to detect an error.

NocStudio will program a parameter with the expected concatenated list of test patterns, as well as the test pattern width. The bist\_gen block will use the width to index through the list of test patterns, sending each in turn.

To check for data corruption, the bist\_chk module will only need to verify that the expected data for the pattern specified in the LSBs is replicated across the upper bits of the payload fields (for the first flit), and across all bits of the payload fields for subsequent flits.

## BIST Packet Data Payload

The BIST packet data fields carry a replicated test pattern, like the flit\_header field.

There is no information included in the flits about which bist\_gen master sent the packet. To save area, the bist\_chk module will not do any binning of packets by master ID. Instead it detects missing flits by keeping a count of total flits received during the time bist\_enb was active and comparing it to an expected count of flits. Each bist\_chk module will get its expected total count of flits from a parameter programmed by NocStudio.

## BIST Packet Acc\_sb Field

BIST explicitly doesn’t cover this control field, used in the upsizing and downsizing of flits. The field changes as it propagates through the NoC and will not preserve any test pattern written on it.

## BIST Packet RAS Fields

If a NoC has parity or ECC enabled, the new BIST bit in the header will be covered by the control field RAS signals.

Because BIST packets are inherently self-checking, BIST can be used to detect stuck-ats in the parity or ECC wires/logic/storage. The NoC elements (routers/switches) that check and generate parity/ECC can use the flit\_is\_bist bit to disable checking and generation on BIST packets. This will allow the bist\_gen logic to replicate the test pattern across those wires as well and the bist\_chk logic to verify that the bits arrived as expected.

# High-Level BIST RTL Blocks and Interfaces

Each protocol bridge (master or slave) that can transmit NoC packets will have a bist\_gen and bist\_chk module associated with it. NocStudio will instantiate these blocks in-line with the existing links between the bridge and switch on both rx and tx interfaces, similar to the way it instantiates a pipeline stage or an ILDC module. The directory trunk/src/hw/ns\_bist/rtl holds wrappers for all the BIST modules.

The bist\_gen and bist\_chk modules below must support whatever NoC configurations exist, including:

* Header and Data Separation
* ECC/Parity Generation
* Byte Enable Generation to ensure no parts of a BIST packet are clockgated

## Bist\_gen Definition

The bist\_gen block will create and transmit the BIST packets. It will have the lion’s share of complexity and state (area cost). To see the ports, look at trunk/src/hw/ns\_bist/rtl/ns\_bist\_gen.v.

Bist\_gen will need to locally synchronize the bist\_enb signal if bist\_ctrl is in a different clock domain.

## Bist\_chk Definition

The bist\_chk block will receive and check the BIST packets. This block will be simple and low-cost (little state/area cost). To see the ports, consult trunk/src/hw/ns\_bist/rtl/ns\_bist\_chk.v.

Bist\_chk will need to locally synchronize the bist\_enb signal if bist\_ctrl is in a different clock domain.

## Bist\_ctrl Definition

A bist\_ctrl block at the top-level of the NoC will control sequencing of the different bist\_gen/bist\_chk blocks. This part of the spec is TBD, depending on customer input about its value vs its cost.

For the initial release of BIST, the bist\_ctrl module will kick off BIST based on a register write, aggregate bist\_done signals, and will have a timer that will signal the end of BIST so that the bist\_chk modules can check for missing flits.

Each RSSB will send a fatal\_bist\_err and nonfatal\_bist\_err signal to the bist\_cntrl module if any of its bist\_chk modules detects an error. NocStudio will aggregate all these signals and send a one-bit fatal\_bist\_interrupt and nonfatal\_bist\_interrupt signal to a pin on the NoC, as well as setting fatal/nonfatal BIST error bits in the Global BIST Error status register(s). The SOC can detect BIST results either by polling the error status registers or by monitoring the BIST error interrupt pins on the NoC.

To see the module definition, look at trunk/src/hw/ns\_bist/rtl/ns\_bist\_ctrl.v.

The bist\_ctrl module will need to locally synchronize bist\_ack/bist\_done signals if they come from a different clock domain.

## Connectivity Diagram

The diagram below illustrates the wire connectivity for all the BIST blocks. Note that both bist\_gen and bist\_chk have bist\_done and bist\_ack signals, which need to be connected into an OR tree and an AND tree respectively. (In the diagram bist\_gen’s signals are not shown as connected because the number of wires crossing the diagram is too dense.)



# BIST Low-Power Support

BIST must run in low-power-enabled NOCs. This means that the BIST controller needs to have the same kind of functionality we currently include in our master bridges, with the ability to wake power domains required for BIST to run. For completeness, it will also need to power down itself by supporting the sleep\_req\_n/sleep\_ack\_n signals from the NSPS (Netspeed Power Supervisor).

## Low-Power Parameters and IO

NocStudio will need to program the standard parameters for low power on the bist\_ctrl module:

parameter P\_NUM\_PD = 4;

parameter P\_SELF\_FENCE\_ENB = 1'b1;

parameter [P\_NUM\_PD-1:0] P\_PD\_AUTOWAKE\_ENB = {P\_NUM\_PD{1'b0}};

parameter [P\_PD\_ID\_WIDTH-1:0] P\_PD\_ID = {P\_PD\_ID\_WIDTH{1'b0}} ;

One new parameter will be necessary to tell the BIST controller which power domains must be running in order to successfully run BIST:

parameter [P\_NUM\_PD-1:0] P\_BIST\_PD\_REQUIRED = {P\_NUM\_PD{1'b1}};

Every power domain that has a bist\_gen or bist\_chk module in it must have its bit set to 1’b1, so that the controller can wake the domain before triggering the BIST run.

It will also need to connect the standard low-power IO to bist\_ctrl. Those signals are documented in the ns\_bist\_ctrl wrapper in hw/ns\_bist/rtl/ns\_bist\_ctrl.v. Search for the Low Power Interface comment.

## Low-Power Functionality

When the BIST controller receives a register write to trigger a new BIST run, it will check if all required power domains are enabled. It will drive the autowake signal to wake up/keep awake all required power domains throughout the BIST run. When all required power domains are active, the BIST run will start. When BIST completes, it will stop drive the autowake signal, allowing those domains to return to sleep/power-gated state.

The BIST controller will also support the sleep\_req\_n/sleep\_ack\_n interface, allowing it to go to sleep and be power-gated.

# BIST Control Signaling

This signaling handshake must support BIST modules running in multiple clock domains to reliably signal BIST start and end states. NocStudio will create an AND tree distributed through the NoC for the bist\_ack signals. The output of the AND tree will connect to bist\_cntl and signal that all modules are running BIST. NocStudio will also create an OR tree to create a global bist\_done signal that connects to bist\_cntl. Bist\_ctrl uses the OR’d signal to determing that BIST has completed so that it can terminate early without waiting for the timeout to occur.

## Signalling for a BIST Run with No Errors

The following timing diagram shows the bist\_ctrl signaling for a successful BIST run (one where no fatal or nonfatal errors occur). This system has N total bist\_gen and bist\_chk modules as both drive bist\_ack and bist\_done, and M total routers with bist\_chk modules attached to them.

When software sets the bist\_enb bit in the global control register, bist\_ctrl asserts a global bist\_enb signal. All bist\_chk and bist\_gen modules acknowledge this by asserting their bist\_ack signal. Bist\_ctrl de-asserts bist\_enb when all bist\_gen and bist\_chk modules have acked the enable signal. The bist\_gen and bist\_chk modules can then deassert their own bist\_ack signals.

At bist\_enb, all bist\_gen blocks kick off packet injection. When each has injected all packets on all routes, each asserts its bist\_done signal.

All bist\_chk blocks begin checking and counting the flits received. When each has received all the flits it expects, it asserts bist\_done, signaling completion.

This signaling handshake supports BIST modules running in multiple clock domains to reliably signal BIST start and end states.



## Signalling for a BIST Run with Non-Fatal Errors

The timing diagram below shows the bist\_ctrll signaling for a BIST run that detects only non-fatal BIST errors. No bist\_ctrl timeout occurs in this case because all modules acknowledge that BIST completed. Each RSSB (with bist\_chk modules attached) signals corrupted packet errors directly from its bist error status register. NocStudio can OR these RSSB outputs into a separate BIST non-fatal interrupt pin on the NoC.

Software can read the RSSB BIST error registers after a BIST run that detects a packet corruption error and isolate the faults to the bist\_chk module(s) in the NOC.



## Signalling for a BIST Run with Fatal Errors

The signaling will look similar to that in Figure 6 except as noted below:

* If a bist\_chk module doesn’t receive all its expected flits, it will never assert bist\_done, and the BIST controller will time-out.
* If a bist\_chk module detects protocol violations that indicate missing BIST flits, it sets its fatal error bit in the attached RSSB.
* If a bist\_chk module receives extra BIST flits, after it has asserted bist\_done, it sets its fatal error bit in its attached RSSB. These errors could assert during normal operation.

# Open issues/Missing documentation

(Error) acc\_sb not tested in this scheme. Document this. Bist\_gen/bist\_chk need to know not to set/compare. DOCUMENTED

Single flit paths are an issue. We can send single flits but it won’t cover some of the wires in the upsize logic or flops in the ivcbuf. NOT RESOLVED BUT ACKNOWLEDGED IN DOCUMENT

Number of flits in packet must be wide enough to cover all wires for upsizing and all flops in ivcbuf. DONE

Coverage numbers? Eric says NS to print out uncovered routes/fifos, will try to quantify into percentages for wires and fifos.

Disable bypass paths in routers in order to get coverage on fifos. DONE and DOCUMENTED

If any byte’s Byte-enables (for write data) is zero, that byte is not clocked. Have bist\_gen always set all all bits of byte enables. Then no bytes will be clockgated. DOCUMENTED

Header/data separation. DOCUMENTED

ECC/parity coverage DOCUMENTED

Two types of BIST errors: one where you can rerun bist one where you can’t DOCUMENTED

Categorize types of errors that can be reported, and which are fatal. (Fatal—packets outstanding or protocol violation. Non-Fatal are corrupt packets.) DOCUMENTED

If Fatal error, don’t say bist\_done (suggestion). DOCUMENTED

Mesochronous isn’t covered by this. Verification needs to know DOCUMENTED

Cover Low Power DOCUMENTED, and will be implemented.