|  |
| --- |
| NetSpeed Orion NSIP NoC C++ Model Specification and Integration Guideline  Version: ORION-NSIP-16.04  April 15, 2016 |

NoC C++ Model Specification and Integration Guideline

About This Document

This document describes the C++ NoC model of Orion and how to use it and integrate it in the customer system.

Audience

This document is intended for users of NocStudio:

* NoC Architects
* NoC Designers
* SoC Architects

Prerequisite

Before proceeding, you should generally understand:

* Basics of Network on Chip technology
* NetSpeed Streaming Protocol

Related Documents

The following documents can be used as a reference to this document.

* NetSpeed NocStudio User Manual
* NetSpeed Streaming Interface Protocol and Bridge Spec

Customer Support

For technical support about this product, please contact [support@netspeedsystems.com](mailto:support@netspeedsystems.com)

For general information about NetSpeed products refer to: [www.netspeedsystems.com](http://www.netspeedsystems.com)

Contents

[About This Document 2](#_Toc416271095)

[Audience 2](#_Toc416271096)

[Prerequisite 2](#_Toc416271097)

[Related Documents 2](#_Toc416271098)

[Customer Support 2](#_Toc416271099)

[1 C++ NoC Model Overview 5](#_Toc416271100)

[1.1 How to Include Model 5](#_Toc416271101)

[1.2 How to Configure Model 5](#_Toc416271102)

[1.3 How to Simulate Model 6](#_Toc416271103)

[Flit Construction for Injection 6](#_Toc416271104)

[1.3.1 Flit Injection 7](#_Toc416271105)

[1.3.2 Flit Ejection 9](#_Toc416271106)

[1.3.3 Advance Clock 9](#_Toc416271107)

[1.4 Summary of NoC Model APIs 10](#_Toc416271108)

[1.5 Restictions in using the C++ Models 10](#_Toc416271109)

List of Tables

[Table 1 NoC model configuration error causes and string values 6](#_Toc416271110)

[Table 2 Injection error cause and string values 8](#_Toc416271111)

[Table 3 Ejection error cause and string values 9](#_Toc416271112)

# C++ NoC Model Overview

The C++ NoC model consists of a statically linked library containing all functions of the model, and a header file containing the APIs that are used to run the model. C++ model is named nocstudio.a for Linux and nocstudio.lib for Windows. The C++ header files are named nocstudio.h.

## How to Include Model

To use the NoC model, users must include the .h header file and link the static library in the project. Then the model must be configured with the same command script file that was used to generate the NoC IP in NocStudio. Once model is configured, the model can be simulated by using the flit injection and flit ejection APIs defined in the header files. The Flit data structure is also defined in the same library. The model is clock cycle aware, therefore the clock must be advanced in the model cycle by cycle by calling the advance clock API.

An example of how to include various model files and use the NoC model APIs is provided in APITest.cpp in the NocStudio release package.

## How to Configure Model

To configure the model a command script file must be created. This file must contain the same set of commands that were used to generate the NoC IP. Alternatively the command log file generated in NocStudio can be used.

The command script must be cleaned up a bit as NoC model does not allow standard NocStudio simulator commands such as run, cont, etc. Therefore all such command should be removed from the command script. If not removed, these commands will be ignored while configuring the NoC model.

Also the command that generate NoC IP, gen\_rtl, is ignored, as the model library is disabled to generate the RTL and verification files. Currently all commands that write on the disk are disabled, including sim\_stats, gen\_image, create\_trace\_files, etc. and are ignored during the model configuration.

Once the script file is ready, the following API is used to configure the model.

Sim\* create\_sim(string filename, string \*error\_string = NULL);

The function returns NULL pointer in case of error in script processing, mapping of the NoC, or Sim creation, and writes a description of the error in error\_string if it is not NULL.

Table 1 NoC model configuration error causes and string values

|  |  |
| --- | --- |
| **Cause** | **Error message** |
| Invalid commands in script | Error: Failed processing commands from script file filename |
| Mesh cannot be created | Error: Cannot extract a NocStudio grid from console; new\_mesh not called? |
| Traffic is not mapped | Error: Cannot run sim unless traffic is mapped. |

## How to Simulate Model

A number of APIs are defined in nocstudio.h file. These APIs are used to simulate the model. During simulation, flits are injected into the model and ejected from the model at various bridge interfaces.

### Flit Construction for Injection

A base NocFlit header class will be provided in the nocstudio.h file. Its attributes are as follows:

struct NocFlit {

int src\_bridge\_id; //source bridge id

int src\_ifce; //source interface id

int dest\_bridge\_id; //destination bridge id

int dest\_ifce; //destination interface id

int qos; //QoS

bool sop; //is start of packet

bool eop; //is end of packet

string payload; //payload

};

A flit object of the above struct type must be constructed for injection into the NoC. Flit injection process must adhere to the protocol requirements of the interface. The payload length must match the interface width. Each character of the string indicates 8 bits of the interface. If interface width is not a multiple of 8, then the string length must be equal to the interface width rounded to upper multiple of 8 bits.

Streaming bridge interfaces a, b, c, d are mapped to integers 0, 1, 2, 3, respectively.

If payload string is not empty but does not match the interface width, then the payload is either truncated (if it is longer than ceil(interface width / 8)) or padded (if it is shorter than ceil(interface width / 8)) to match the flit size with the interface width.

If payload string is empty then only performance simulation is performed and no payloads are carried. This may be used to speed up the simulation when only performance modeling (not functional modeling) is being performed.

### Flit Injection

Following API is used for flit injection.

bool inject\_flit(Sim\* s, const NocFlit& flit, string\* error\_string = NULL);

bool can\_inject\_flit(Sim\* s, const NocFlit& flit, string\* error\_string = NULL);

For an injection to be successful, the flit provided must match a valid traffic hop specified in the imported script file with which the model is configured. Flit injections must adhere to following protocol.

1. SOP/EOP Logic
   1. A series of flits must be started with an SOP flit
   2. An EOP flit must come at some point after an SOP to complete the series
   3. The first injection from any interface must always be an SOP flit
2. Flits for different traffic flows cannot be interleaved when injecting into the same interface
3. Only one flit may be injected per interface per cycle

During each cycle, a flit may be injected at every input interface of all bridges of the NoC by calling the inject\_flit API. The injection API returns true if flit injection is successful and false if injection fails. Injection may fail because of protocol errors, or invalid message, or invalid fields in the flow. It may also fail if there is flow control asserted at the injection interface. In this case the same flit may be injected again in the next cycle.

*can\_inject\_flit* API is to peek an interface to determine if a flit can be injected there. An error\_string may be provided for logging of errors. If error\_string argument is not NULL then error messages are logged in the string.

Table 2 Injection error cause and string values

|  |  |
| --- | --- |
| **Cause** | **Error message** |
| Invalid src\_h | Source bridge id not valid. |
| Invalid dest\_h | Destination bridge id not valid. |
| Invalid src\_i | Source interface id must be in range 0...3. |
| Invalid dest\_i | Destination interface id must be in range 0...3. |
| Invalid src\_br | Cannot get a valid bridge from source bridge id. |
| Invalid dest\_br | Cannot get a valid bridge from destination bridge id. |
| Invalid src\_ifce | Cannot get a valid transmitting interface. |
| Invalid dest\_ifce | Cannot get a valid receiving interface. |
| Invalid src\_br (regbus) | Source bridge is Regbus Master. Currently not supported. |
| Invalid dest\_br | Cannot get a valid bridge from destination bridge id. |
| Invalid dest\_br (regbus) | Destination bridge is Regbus Master. Currently not supported. |
| Invalid tx | Cannot get a valid transmitting interface. |
| Invalid rx | Cannot get a valid receiving interface. |
| Invalid message | Cannot create a valid packet. No traffic flows match flit attributes. |
| Invalid message | No packet is being sent; need sop to start a new packet. |
| Invalid message | Current packet is still transmitting; need eop to complete. |
| Invalid message | Flits are being interleaved. QoS and destination of flits from same packet must match. |
| Invalid message | Flit offset mismatch. |
| Flow control | Can only inject one flit per interface every cycle. |
| Flow control | Cannot inject; rate limit exceeded. |
| Flow control | Cannot inject; fifo full. |
| Flow control | Flow Control. |
| Other reasons | Failed to inject flit in the NoC. / Cannot inject flit in the NoC. |

### Flit Ejection

Following API is used for flit ejection.

shared\_ptr<NocFlit> eject\_flit(Sim\* s, const int dhport, const int difce, string\* error\_string = NULL);

bool can\_eject\_flit(Sim\* s, const int dhport, const int difce, string\* error\_string = NULL);

During each cycle, a flit may be ejected at every input interface of all bridges of the NoC by calling the eject\_flit API. If a flit is inside the destination interface fifo, it is ejected and pointer to the flit returned. If there is no flit stored, eject\_flit will return NULL. If invalid arguments are provided then and error\_string is not NULL then error messages are logged in the string. error\_string also logs additional messages in case of abnormal events.

*can\_eject\_flit* API is to peek an interface to determine if a flit can be ejected from there.

Table 3 Ejection error cause and string values

|  |  |
| --- | --- |
| **Cause** | **Error message** |
| Invalid dhport | Destination bridge id not valid. |
| Invalid difce | Destination interface id must be in range 0...3. |
| Invalid dest\_br | Cannot get a valid bridge from destination bridge id. |
| Invalid dest\_ifce | Cannot get a valid receiving interface. |
| Flow control | Cannot eject; rate limit exceeded. |
| Flow control | Cannot eject; fifo empty. |
| Invalid flit received | Destination hostport in NocFlit does not match with parameter dhport. |
| Invalid flit received | Destination interface in NocFlit does not match with parameter difce. |

### Advance Clock

Following API is used to advance the internal clock of the NoC model.

void advance\_time(Sim\* s);

advance\_time advances the simulation time by one clock cycle, consuming all inputs of all blocks and producing outputs during that clock.

## Summary of NoC Model APIs

The NoC model uses four commands, which are summarized below:

1. **create\_sim**
   1. *input:* filename, length of file
   2. *output:* pointer to model simulator
   3. *about:* creates the sim pointer from a script
2. **advance\_time**
   1. *input:* simulator pointer
   2. *about:* runs the simulator for one clock cycle
3. **inject\_flit and can\_inject\_flit**
   1. *input:* simulator pointer, NocFlit pointer, payload string, payload size
   2. *output:* bool value of successful injection
   3. *about:* injects a flit
4. **eject\_flit and can\_eject\_flit**
   1. *input:* simulator pointer, NocFlit pointer, int destination hostport ID, int destination interface ID, payload buffer
   2. *output:* NocFlit pointer

## Restictions in using the C++ Models

1. Multi-noc is currently un-tested and might break the models

2670 Seely Avenue

Building 11

San Jose CA 95134

[www.netspeedsytems.com](file:///C:\Users\Ranjeet\Documents\Netspeed\NocStudio\netspeed_noc\trunk\doc\release_docs\www.netspeedsytems.com)