Appendix C

Reference Manual

C.1 Command Line Syntax

-s <simulation_id> simulate the entry *<simulation_id>* of file `Simula.sgm`. This switch is always necessary.

-l <load> set the traffic load. This value is normalized to the torus bisection. It overrides the value set by the tag LOAD in `Simula.sgm`, it can not be used together with *-p*. *<load>* must be in the range of 0.0 to 1.0. See also LOAD in page 53.

-p <probability> set the injection probability. It overrides the value set by the tag PROBABILITY in `Simula.sgm`, it can not be used together with *-l*. *<probability>* must be in the range of 0.0 to 1.0. See also PROBABILITY in page 54.

-c <simulation_cycles> set the simulation length. It overrides the value set by the SIMULATIONCYCLES tag in `Simula.sgm`. See also SIMULATION-CYCLES in page 54.

-u <buffer_size> set the size of all buffers of each router. It overrides the value set globally by the ROUTER tag, so the buffers that have their size individually set by the BUFFER tag in `Router.sgm` are not modified. See also ROUTER in page 59 or BUFFER in page 56.

-L <packet_lenght> set the packet length. It overrides the value set by the PACKETLENGTH tag in `Simula.sgm`. See also PACKETLENGTH in page 54.

-t <pattern>[,<param1>[,<param2>]] set the traffic pattern. *<pattern>* can be any traffic pattern name. *<param1>* and *<param2>* can be added if traffic-specific parameters a needed. There may be no blank space between *<pattern>*, *<param1>* or *<param2>*. It overrides the value set by the TRAFFICPATTERN tag in `Simula.sgm`. See also TRAFFICPATTERN in page 55.

-B <long/short_packet_ratio>,<long_packet_probability> set bimodal traffic. This is, two types of packets are generated, a longer one and a shorter one. The short packet's size is set by the *-L* or the PACKETLENGTH. *<long/short_packet_ratio>* sets how many times bigger than the short packet is the long packet. *<long_packet_probability>* is the probability of generating long packets. See also TRAFFICPATTERN in page 55 and PACKETLENGTH in page 54.

-d <message_count> show the evolution of the simulation over time. When the simulation finishes, a table shows the traffic load offered to the network and the mean latency. A sample presented in the table every time the network transmits *<message_count>* messages. In addition a histogram of the distance traveled by the messages is also shown.

-b <bufferFileName> set the name of the file that will hold buffer occupation information. The buffer occupation is sampled every 100 cycles and written, in a binary format, in *<bufferFileName>*. The information stored in this file can be examined with `BView` or with `xbuffer`. See also `BView` in page 42 and `xbuffer` in page 42.

C.2 SGML Tag Reference

C.2.1 Tags in `ATCsimul.ini`

NETWORKFILE

NetworkFile id=<path>

Specifies the path where the network description file, normally `Network.sgm`, is found. The path can be absolute or relative to the current directory of the simulation.

ROUTERFILE

RouterFile id=<path>

Specifies the path where the router description file, normally `Router.sgm`, is found. The path can be absolute or relative to the current directory of the simulation.

SIMULATIONFILE

SimulationFile id=<path>

Specifies the path where the simulation description file, normally `Simula.sgm`, is found. The path can be absolute or relative to the current directory of the simulation.

C.2.2 Tags in `Simula.sgm`

LOAD

Load id=<load>

Sets the normalized traffic load to *<load>*. The injection probability is calculated as shown in[4]. It may be overridden by the the switches *-l* or *-p*. This tag can not be used together with the tag PROBABILITY. Value *<load>* must be in the range of 0.0 to 1.0. See also PROBABILITY in page 54.

MESSAGELENGTH

MessageLength id=<message_length>

Specifies the number of packets a message has.

NETWORK

Network id=<Network_id>

Sets the name of the network that will be used in the simulation. The identifier *Network_id* must be defined in `Network.sgm`

PACKETLENGTH

PacketLength id=<packet_lenght>

Sets the number of flits a packet has. It may be overridden by *-L*.In the case of bimodal traffic patterns, *packet_lenght* defines the length of the smallest packet. See TRAFFICPATTERN in page 55.

PROBABILITY

Probability id=<probability>

Sets the injection probability to *<probability>*. It may be overridden by the the switches *-l* or *-p*. This tag can not be used together with the tag LOAD. Value *<probability>* must be in the range of 0.0 to 1.0. See also LOAD in page 53.

SEED

Seed id=<seed>

Sets the seed for the random number generator.

SIMULATION

Simulation id=<simulation_id>

This is a group tag. It bundles together all the simulation parameters and associates them to *simulation_id*. This is the identifier that must follow *-s* when performing a simulation.

SIMULATIONCYCLES

SimulationCycles id=<simulation_cicles>

Sets the number of cycles that should be simulated. It may be overridden by *-c*.

STOPINJECTIONAFTER

StopInjectionAfter id=<simulation_cicles>

Sets the number of cycles after which no more packets should be injected to the network.

TRAFFICPATTERN

TrafficPattern id=MODAL type=<pattern>[,<param1>[,<param2>]]

TrafficPattern id=BIMODAL type=<pattern>[,<param1>[,<param2>]] prob=<long_packet_probability> numMsg=<long/short_packet_ratio>

Sets the traffic pattern that is injected to the network. *<pattern>* can be any traffic pattern name. *<param1>* and *<param2>* can be added if traffic-specific parameters a needed. There may be no blank space between *<pattern>*, *<param1>* or *<param2>*.

If *BIMODAL* traffic is needed. The short packet's size is set by the *-L* or the PACKETLENGTH tag. *<long/short_packet_ratio>* sets how many times bigger than the short packet is the long packet. *<long_packet_probability>* is the probability of generating long packets.

The effect of this tag may be overridden by parameters *-t* and *-B*. See also PACKETLENGTH in page 54.

C.2.3 Tags in `Network.sgm`

MIDIMEWNETWORK

MidimewNetwork id=<network_id> sizeX=<size_x> router=<router_id> [delay=<delay>]

Connect *<size_x>* routers with identifier *<router_id* using a midimew topology. The connections between the routers can have an optional delay of *<delay>*.

SQUAREMIDIMEWNETWORK

SquareMidimewNetwork id=<network_id> sizeX=<size_x> router=<router_id> [delay=<delay>]

Connect *<size_x>* routers with identifier *<router_id* using a square midimew topology. Currently *<size_x>* must be a even power of 2 (2^{2i}). The connections between the routers can have an optional delay of *<delay>*.

TorusNetwork

TorusNetwork id=<network_id> sizeX=<size_x> sizeY=<size_y> sizeZ=<size_z> router=<router_id> [delay=<delay>]

Constructs a 1D, 2D or 3D torus with *<size_x>*, *<size_y>* and *<size_z>* routers in each dimension. The connections between the routers can have an optional delay of *<delay>*.

C.2.4 Tags in `Router.sgm`

Buffer

Buffer id=<buffer_id> type=<dimension> [dataDelay=<delay>] [size=<buffer_size>] [ackgranul=<granularity>]

This component implements a simple FIFO queue for flits. It accepts flits from the component before it and serves them to the component behind. The buffer control function, also knows as flow control unction, tells when the buffer is full and can not admit more flits. This function is defined as follows:

wormhole The stop signal goes high when there is no room for another flit.

cutthrough The stop signal goes high when there is no room for another whole packet.

Connection

Connection id=<connection_id> source=<source_id> destiny=<destiny_id>

Connects two components within the router. *<source_id>* and *<destiny_id>* must be a string composed by the identifier of the component and, if necessary, a dot followed by the number of the input or output.

Consumer

Consumer id=<consumer_id>

Accepts incoming flits. It performs some checks to ensure that the flits are reaching their destination, that they arrive in proper order and not mixed with flits from other packets. In addition it performs statistical measures on the incoming flits.

Crossbar

Crossbar id=<crossbar_id> inputs=<inputs> outputs=<outputs> type=<type> [mux=<mux>] [headerDelay=<header_delay>] [dataDelay=<data_delay>]

Connects any of its inputs to any of its outputs. This component performs the spatial switching ability of the router. Depending on the *<type>* parameter, it establishes a specific protocol with the previous component, normally a routing, in order to grant or deny its requests.

<inputs> and *<outputs>* specify the number of inputs and outputs the crossbar has. The crossbar may have different number of inputs and outputs when the *<mux>* parameter is not 1. This *<mux>* parameter indicates the crossbar it has to multiplex the outputs in order to bundle various virtual channels into a physical one. *<header_delay>* and *<data_delay>* set the delays for header flits and data flits respectively. See also ROUTING in page 60.

FifoMuxed

FifoMuxed id=<buffer_id> type=<dimension> outputs=<outputs> control=<control> [dataDelay=<delay>] [size=<buffer_size>] [ackgranul=<granularity>]

This component is like a normal BUFFER but it has a demultiplexer attached to the output, this enables various virtual channels to share the space in the buffer. The number of outputs of the demultiplexer is set with the *<outputs>* parameter. The rest of the parameters are set like in the normal BUFFER tag. See also BUFFER in page 56.

INJECTOR

Injector id=<injector_id> numHeaders=<headers>

It is the component in charge of converting the messages that arrive to the node into sequences of flits organized into packets. The *<headers>* value specifies how many flits per packet are used as headers.

INPUT

Input id=<input_id> type=<dimension> { wrapper=<id_list> | [channel=<channel>] }

This tag declares each of the inputs of a CROSSBAR tag or a ROUTER tag. Each INPUT must have a different *<input_id>* from the rest of the inputs in the component. The *<dimension>* must tell to which dimension is the input connected. This can be X+, X-, Y+, Y-, Z+ or Z- if the input handles flits coming from other routers, or NODE if it is a crossbar input handling flits from the local injector.

When specifying router inputs, the *<id_list>* specifies to which component is the input connected to. In case of an input that supports multiple virtual channels, the *<id_list>* holds a list of the components that will be connected to each virtual channel.

If the input belongs to a CROSSBAR that has multiple virtual channels for each dimension, there must be a different input for each virtual channel and it must be specified using the *<channel>* parameter.

See also ROUTER in page 59 and CROSSBAR in page 57.

MPBUFFER

MPBuffer id=<buffer_id> type=<dimension> inputs=<inputs> control=<control> [dataDelay=<delay>] [size=<buffer_size>] [sizeShort=<buffer_size_short>]

MULTIPLEXORCV

MultiplexorCV id=<multiplexor_id> inputs=<inputs>

This component does a multiplexing of various virtual channels to a single physical channel. The number of channels it multiplexes is defined by *<input>*.

OUTPUT

Output id=<output_id> type=<dimension> { wrapper=<id_list> | [channel=<channel>] }

This tag declares each of the outputs of a CROSSBAR tag or a ROUTER tag. Each OUTPUT must have a different *<output_id>* from the rest of the outputs in the component. The *<dimension>* must tell to which dimension is the output connected. This can be X+, X-, Y+, Y-, Z+ or Z- if the output handles flits going to other routers, or NODE if it is a crossbar input handling flits that must go to the local consumer.

When specifying router outputs, the *<id_list>* specifies to which component is the output connected to. In case of an output that supports multiple virtual channels, the *<id_list>* holds a list of the components that will be connected to each virtual channel.

If the output belongs to a CROSSBAR that has multiple virtual channels for each dimension, and there are not multiplexed, there must be a different output for each virtual channel and it must be specified using the *<channel>* parameter.

See also ROUTER in page 59 and CROSSBAR in page 57.

ROUTER

Router id=<router_id> inputs=<inputs> outputs=<outputs> bufferSize=<buffer_size> bufferControl=<buffer_control> routingControl=<routing_control>

This tag encloses the definition of a router in terms of interconnected components. The *<router_id>* specifies the identifier of the router that will be used in the `Network.sgm` file. *<inputs>* and *<outputs>* set the number of inputs and outputs the router has, normally both values are equal. *<buffer_size>* sets a default size for the buffers in the router. <buffer_control> specifies the buffer control function, it can be WH, for worm-hole, or CT, for cut-through. And *<routing_control>* establishes the routing function of the ROUTING components.

ROUTING

Routing id=routing_id> type=<dimension> [channel=<channel>] [headerDelay=<header_delay>] [dataDelay=<data_delay>]

A ROUTING component analyzes incoming packages, calculates the output they should go and communicates with the crossbar to request a connection for it.

Routings must have information of which dimension and virtual channel there are working in. This must be stated with parameters *<dimension>* and *<channel>*. *<header_delay>* and *<data_delay>* set the delays for header flits and data flits respectively. See also CROSSBAR in page 57.

ROUTINGMUXED

RoutingMuxed id=routing_id> type=<dimension> [channel=<channel>] inputs=<inputs> [headerDelay=<header_delay>] [dataDelay=<data_delay>

This component has similar functions as the ordinary ROUTING but it can multiplex a number of virtual channel allowing them to share the output to the crossbar, thus reducing its complexity.

The only parameter that differs from the ROUTING is the number of *<inputs>* to the component. See also ROUTING in page 60.

SIMPLEROUTER

SimpleRouter id=<simple_router_id> inputs=<inputs> outputs=<outputs>

The SIMPLEROUTER component is intended to pack all of the components (Except injector and consumer) of a router into a single component. There is a slight loss of accuracy but, on the other hand, the simulation time is considerable shorter.

Within this tag there must be a list of the inputs and outputs that the component has, very much like the CROSSBAR.

C.3 Known Problems

The SGML processing routines are written on a text-line basis, therefore tags can not span more than one line of text. Similarly two tags can not be in the same line.