VALIDATION

This part involves the validation of the sub-partitioned design units for the Network on Chip as well as the top-level validation for the entire router. Each validation skeleton has essentially 4 parts:

1. Interface

This file has to be modified for each module in accordance with the sub-unit specification for interfaces given by the design team. It defines the basic inputs and outputs for each module as seen by the DUT and the bench. It is also responsible for defining the timing constraints as they relate to the system clock (using the clocking block construct provided by SystemVerilog).

1. Top-level file

This file assigns the clock, defines the interfaces, and instantiates the appropriate Design Under Test (DUT) and the testbench.

1. Bench file

This file is the one responsible for running the simulation. It stimulates the DUT with randomly generated inputs, takes the generated output signals from the DUT, generates its own outputs for the same stimuli, and compares the DUT value to its own value. Simulation continues until either an error is found or the maximum number of simulation cycles have been executed. In order to simplify use of the testbench and make its operation more understandable, the bench file is split into classes. Each class is responsible for doing a single task. The bench file consists of the following classes:

* Environment class

This class initializes the simulation parameters such as simulation duration, transaction densities for each interface, bit masks, verbosity, etc. The parameters that are initialized are specific to the unit being tested.

* Checker class

This defines whether the result passes or fails based on comparison of the DUT output and the bench-generated output.

* Tester class

This contains the golden model function, which duplicates the behavior of the DUT using non-synthesizeable SystemVerilog. The values generated by the golden model function are then checked by the checker class

* Packet class

This class defines the basic interface of communication with the module. It defines the different input and output fields specific to the DUT. It is also responsible for defining which inputs are random – a crucial component of the simulation.

* do\_cycle task

This task increments the cycle number and is responsible for passing the appropriate data to the DUT and golden model. It makes sure that clocked modules only sample their inputs at the clock edge, and that outputs are ready before the next clock edge.

1. Makefile

The Makefile is responsible for assembling the correct SystemVerilog source code, compiling it, and running it.

Each of the 4 files mentioned above must be modified appropriately for each testable sub-module.

We will continue to take a bottom-up approach to validating our router. This will be done by first implementing the design of the most basic sub-modules, implementing the corresponding testbench using the already assembled test harness, and validating their operation. This will ensure that as we assemble our higher level modules, any errors we find at the higher level of abstraction will be due to the higher level module under test, *not* the sub-modules that it instantiates. As it stands, the empty test harnesses for each of the currently-defined modules and sub-modules successfully compile with the top level files and run for 10,000 cycles.