Lab 1 – V1.1

Modify the sanity-check testbench for Gullfaxi to make it compliant to what is described in this document.

Generation of the packets

At time 0, the testbench shall generate a number of packets nrPkts, where nrPkts is a parameter. Every packet shall be generated at a high level of abstraction, in which packets are characterized by the following two properties only: length and output port. Both of them should be random numbers that can take any legal value, as described in the GIP design specifications. There is no need to generate the payload of the packet and save it, this will be generated randomly on-the-fly and it is not required to do any check on the payload.
The generated high-level packets shall be put into a queue of a type that can store all the required information (queue of packed structs is a good choice).
The generated packets shall be copied into a second queue, so that one queue is used to push the packets to the DUT, the other to check that the received packets are correct

GIP driver

An initial block shall be used to drive the generated packets to the DUT. This block shall have a loop that iterates until the packet queue is empty. On every iteration, it shall be checked that the DUT is ready to accept a packet. If not, the GIP driver shall wait for the DUT to be ready. A packet is then popped and driven to the DUT based on the GIP protocol rules. The GIP protocol says that in certain cycles the driver can introduce wait-cycles, i.e. cycles in which the driver does not output data and keeps the GIP valid line low. In every cycle where it is legal to have a wait-cycle, the driver shall introduce the wait-cycle with a probability equal to 1/invProbWaitCycle, where invProbWaitCycle is a parameter of the testbench.
After a packet has been pushed through the GIP port, the GIP driver shall wait a random number of cycles between minWaitForSend and maxWaitForSend to send the next packet, where minWaitForSend and maxWaitForSend are two parameters of the testbench.

GOP consumer

Three initial blocks, one per port, shall be used to receive data from the DUT. Every block shall start with grant low and shall iterate until it detects a GOP request. Once the GOP request is detected, the block shall raise the grant after a random number of cycles between minWaitForGrant and maxWaitForGrant, where minWaitForGrant and maxWaitForGrant are two generics. Once the DUT starts transmitting, grant shall be lowered and the block shall check that the length of the packet, as given on the GOP length line, and the port are as expected, by popping a new element from the packet-receive queue. If the packet is not as expected, then the block shall produce an error and terminate. The GOP consumer shall print to the console, every time it receives a packet, a string saying: “Received a packet on port <N>, length is <L>” and then shall print under it all the payload bytes one by one.

The simulation will be terminated by the GOP consumer after the last packet in the receive-queue has been checked.

Simulation control

To prevent a hanged simulation, the simulation shall generate an error when the simulation time exceeds maxCycles, where maxCycles is a parameter.

Parameter summary

- nrPkts: number of packets to be generated
- invProbWaitCycle: inverse of the probability to introduce a wait-cycle during a GIP transmission where this is legal
- minWaitForSend: minimal amount of cycles to wait between two GIP sendings
- maxWaitForSend: maximal amount of cycles to wait between two GIP sendings
- minWaitForGrant: minimal amount of cycles to wait before granting a GOP request
- maxWaitForGrant: maximal amount of cycles to wait before granting a GOP request
- maxCycles: if the simulation lasts more than maxCycles, the testbench shall be terminated with an error

Processes

Use six initial processes:

- One to generate the queues of high-level packets at time 0
- One to act as GIP driver
- Three to act as GOP consumers
- One to stop simulation if maxCycles is reached

Optional tasks

(1) The GOP consumer shall check that: (a) the GOP length signal is stable from the time in which req is asserted to when the GOP end signal is raised; (b) the number of items sent by the DUT (between the GOP start and end signals) corresponds to the length of the transaction on the GOP length line; (c) when the signal req becomes one, it stays at one until grant is given; (d) req remains zero during the packet transmission (from when start is one to when end is one). If any of these properties is not true, an error message shall be printed and the testbench shall be terminated. Hint: it is much easier to do these checks using several different initial blocks. It is not allowed to solve this task using assertions and/or sampled functions ($stable, $past, $rose, $fall).

(2) Modify the high-level packet structure so that the packets payloads are generated together with the packets and saved in the packets queues at time 0. The GIP driver shall drive the payload bytes to the DUT and the GOP consumer shall check that all the payload bytes reaching it are correct. If not, it shall display an error message and terminate the simulation.