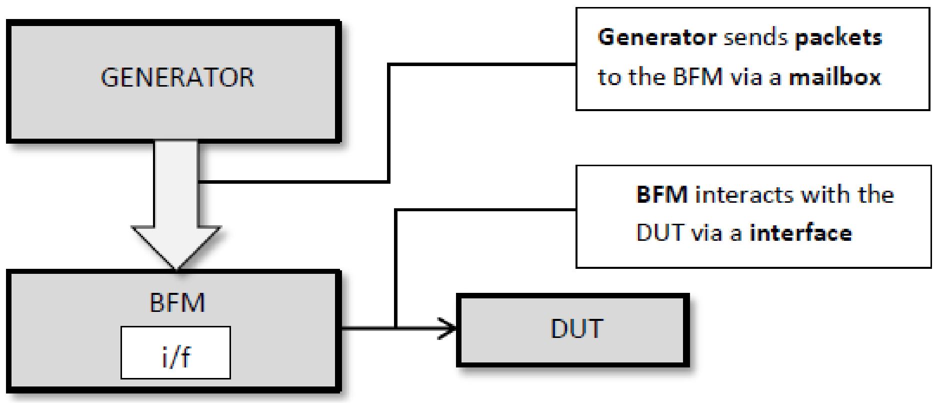
Lab 4: Bus Functional Models and generators

Goal

Define the Bus Functional Model (BFM) and the packet generator for the input channel of the DUT.
Define the BFMs and generators for the rest of the DUT's interfaces.

Overview


GENERATOR
BFM
i/f
DUT
Generator sends packets to the BFM via a mailbox
BFM interacts with the DUT via a interface

The BFM is a module that drives and samples low-level signals according to the bus protocol, using the **input_intf** interface that was previously declared. The bus protocol is described in the DUT's specification, and it is important to remember that the BFM is clock accurate, meaning that it has to work on a clock edge (positive or negative, depending on the bus it models) and that once you assign a value to a wire or bus on the interface you have to "let time pass" which means wait one clock cycle (or more, if the protocol demands it) so that the assignment can take effect.
The generator is in charge with the creation of packets and sends them via a mailbox to the BFM, which in turn will stimulate the input signals of the DUT via the **input_intf** interface.

Files

svbt_data_in_generator.sv
svbt_data_in_bfm.sv
svbt_reset_bfm.sv
svbt_channel_out_generator.sv
svbt_channel_out _bfm.sv
svbt_environment.sv
svbt_base_unit.sv

Instructions

- Complete the definition of the input **generator** class in the file **svbt_data_in_generator.sv**; make sure that the id field in the **packet** class is assigned by the generator uniquely to each packet
- Complete the input **BFM** class in the file **svbt_data_in_bfm.sv**
- Complete the **reset BFM** in the file **svbt_reset_bfm.sv** (*Note: any design needs a reset to initialize internally, so make sure every test you implement starts with a reset toggle!*)
- Instantiate the input **generator** and **BFM** in the **environment** class and start their **run()** tasks
- Add the necessary mailboxes in the base unit file
- After you make sure, by checking the waves, that the driving is correct on the input interface, use what you learned so far to implement the **BFMs** and **generators** for the output interfaces as well (OOP- you define a generic class and instantiate it three times in the env, one instance per output channel);
- Compile and run

References

SV_LRM “Processes”
SV_LRM “Procedural programming statements”
SV_LRM “Interprocess synchronization and communication”