1 Introduction

1.1 The SIF design

The SIF design is an interface from bus X to bus W. A write on bus X appears unchanged on bus W one cycle later. A read on bus X returns data one cycle later, where the data is the address with some xoring on the biting:

{addr[15:9], addr[8]^addr[4], addr[7]^addr[5], addr[6:0]}

Both the X and W buses use the same protocol, but on the W side the read signals are not implemented.

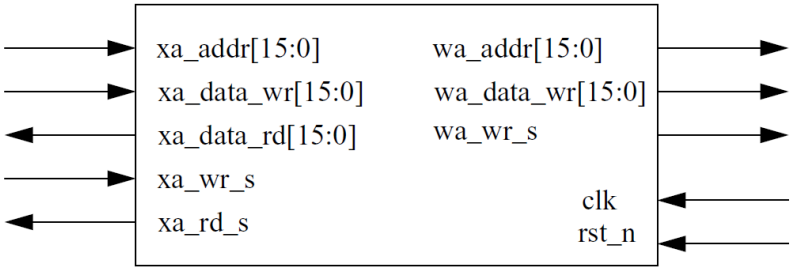

Figure 1: The SIF design

1.2 The X bus

The X bus protocol is a 2-stage pipe, with read data arriving on the second stage (Data stage).

Signal	Description	Stage
addr[15:0]	Address for read/write transaction	Address
data_wr[15:0]	Write data	Address
data_rd[15:0]	Read data	Data
wr_s	Write strobe (write enable)	Address
rd_s	Read strobe (read enable)	Address

A write transaction is a cycle with wr_s asserted. addr and data_wr must be valid in that cycle. wr_s length is one cycle, if it lasts two cycles, then those are two writes.

A read transaction is a cycle with rd_s asserted. addr must be valid in that cycle, and data_rd must be returned one cycle later.

wr_s, rd_s = 00 means Idle, and wr_s,rd_s = 11 is illegal.

2 The test environment (SystemVerilog without UVM suggestion)

2.1 Environment block diagram

The test system includes the following components:

generator - random generation of read and write transactions into a queue (fifo).

driver - read transactions from the queue and drive them to the X-bus, via an interface.

monitor - monitor transaction on the bus, for each transaction write an identical message to 2 queues. The monitor is instantiated twice, once to monitor the X bus and once to monitor the W bus.

ref_design - receive messages via the X bus monitor queue and generate expected output, for X bus (reads), for W bus (writes), and expected memory map.

comparator - compare expected and actual on a bus. The comparator is instantiated twice, once to compare the X bus and once to compare the W bus.

memory - receive the writes on the W bus.

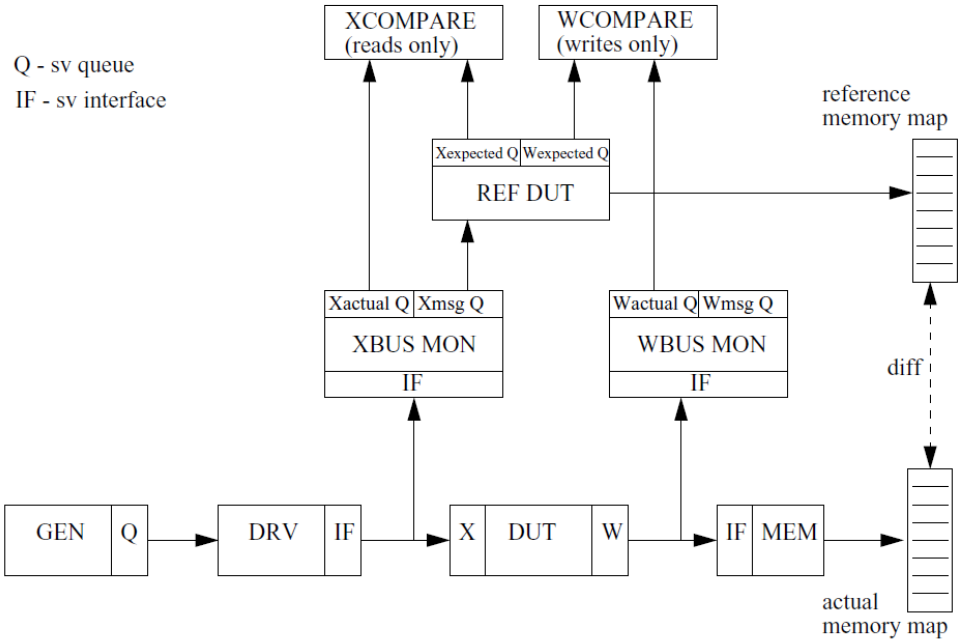

Figure 2: SystemVerilog test environment suggestion

2.2 Flow description

The generator generates transactions at time 0, and pushes them into the commands queue (fifo). The driver runs a loop on the queue and as long as it is not empty it pops a command and drives it to the X-bus (note that a field in the command also dictates how many stalls to wait before driving the next transaction).

The driver connects to the DUT via an interface.

The same interface (with different clocking block and modport) is used to connect the monitors and the slaves (memory). The interface is instantiated twice, once for the X-bus and once for the W-bus.

There are two identical monitors, one on the X-bus and one on the W-bus. The monitors write an entry into the message queue for every transaction spied on the busses, and the same message also to the actual queue.

The reference model pops a message from the message queue (whenever it is not empty), and generates an expected message into the appropriate queue. Reads are directed to the Xexpected message queue, because they do not reach the W-bus. Writes are directed to the Wexpected message queue.

The reference model also implements a reference memory model, and outputs it at the end of the simulation into a file, to be compared externally via UNIX diff, with the actual memory map.

The compare is instantiated twice. The X instance pops a message from the Xactual Q, and if it is a read pops a message from the Xexpected Q, compares them, and displays a PASS/FAIL message on the screen. The W instance does the same for writes on the W queues.

The memory is a slave on the W-bus, and is used to create the actual memory map at the end of the simulation, to be compared with the reference memory map.

2.3 Message classes

Two message classes are defined, and all the queues are using one of them.

2.3.1 X bus command

The xbus_cmd class is used by the generation and also by the generation queue. Transactions are generated via this class, and stored in a queue of type xbus_cmd. Class members are defined as follows:

```
rand bit[0:0] wr_s;
rand bit[15:0] addr;
rand bit[15:0] data_wr;
rand int stall;
```

2.3.2 X bus message

The xbus_msg_class is used by all other queues. Class members are defined as follows:

```
typedef enum { READ , WRITE } XBUS_ACCESS_TYPE;
XBUS_ACCESS_TYPE rd_wr;
bit[15:0] addr;
bit[15:0] data;
```