OSVVM

OSVVM Overview

- An accelerated approach for developing verification components that uses the transaction in terface and API defined in our transaction support library.
- Support for continuous integration (CI/CD) with JUnit XML test suite reporting.
- Co-simulation of C++ models in a hardware simulator environment
- **A rival to the verification capabilities of SystemVerilog + UVM**
- If you wanna use OSVVM. I should recoment Aldec's Rivera-PRO or Siemen's QuestaSim/ModelSim/GHDL(open source) because in Vivado you have limited usage for it (according to internet).

The Build Summary Report

It has three distinct pieces

OsvvmLibraries_RunDemoTests Build Summary Report

Build	OsvvmLibraries_RunDemoTests
Status	PASSED
PASSED	4
FAILED	0
SKIPPED	0
Elapsed Time (h:m:s)	0:00:14
Elapsed Time (seconds)	13.505
Date	2022-04-20T15:42-0700
Simulator	RivieraPRO
Version	RivieraPRO-2021.10.114.8313
OSVVM YAML Version	1.0

- Build Status
- Test Suite Summary

▼ **OsvvmLibraries_RunDemoTests Test Suite Summary**

TestSuites	Status	PASSED	FAILED	SKIPPED	Requirements passed / goal	Disabled Alerts	Elapsed Time
Axi4Full	PASSED	1	0	0	0 / 0	0	4.039
AxiStream	PASSED	1	0	0	0 / 0	0	3.436
Uart	PASSED	2	0	0	0 / 0	0	5.944

- Test Case Summary

▾ Axi4Full Test Case Summary

Test Case	Status	Checks passed / checked	Errors	Requirements passed / goal	Functional Coverage	Disabled Alerts	Elapsed Time
TbAxi4_MemoryReadWriteDemo1	PASSED	334 / 334	0	0 / 0	43.75	0	2.608

▾ AxiStream Test Case Summary

Test Case	Status	Checks passed / checked	Errors	Requirements passed / goal	Functional Coverage	Disabled Alerts	Elapsed Time
TbStream_SendGetDemo1	PASSED	340 / 340	0	0 / 0	66.67	0	0.956

▾ Uart Test Case Summary

Test Case	Status	Checks passed / checked	Errors	Requirements passed / goal	Functional Coverage	Disabled Alerts	Elapsed Time
TbUart_SendGet1	PASSED	30 / 34	0	0 / 0	-	0	1.065
TbUart_SendGet2	PASSED	22 / 26	0	0 / 0	-	0	1.193

Reports

- Test information link table
- Alert Report
- Functional Coverage Report
- Scoreboard Report
- Test Case Transcript
- HTML Simulator Transcript

OSVVM's Structured Testbench Framework

- In OSVVM, signal wiggling is replaced by **transactions**
- The top level of the testbench connects the components together and is often called a **test**

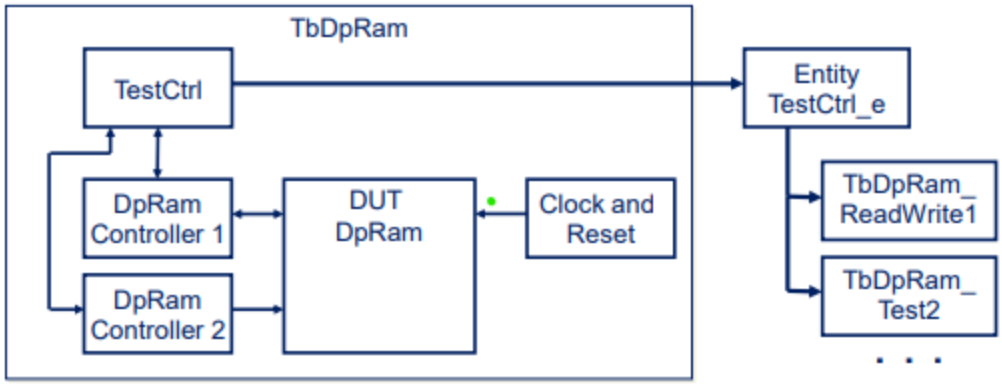

Figure 2. OSVVM Testbench Framework

harness

- Transaction procedure calls (Write, Read, ReadCheck, Send, Get, Check, …) are used to create the **test sequence of interface**

```
architecture BasicReadWrite of TestCtrl is
  . . .
begin
  ControlProc : process
  begin
    SetAlertLogName("TbDpRam_BasicReadWrite") ;
    . . .
    WaitForBarrier(TestDone, 5 ms) ;
    EndOfTestReports ;
    std.env.stop;
  end process ;
  Manager1Proc : process
  begin
    . . .
    for i in 1 to 10 loop
      Write(Manager1Rec, X"01_0000" + i, X"1000" + i ) ;
    end loop ;
    . . .
    WaitForBarrier(TestDone) ;
  end process Manager1Proc;
  Manager2Proc : process
  begin
    . . .
    for i in 1 to 10 loop
      ReadCheck(Manager2Rec, X"01_0000" + i, X"1000" + i) ;
    end loop ;
    . . .
    WaitForBarrier(TestDone) ;
  end process Manager2Proc ;
```

Figure 4. TestCtrl Architecture

- Separate tests are written in separate architectures of TestCtrl
- Using multiple architectures does not change the test harness

```
Configuration TbAxi4_MemoryReadWriteDemo1 of TbAxi4Memory is
  for TestHarness
    for TestCtrl_1 : TestCtrl
      use entity work.TestCtrl(MemoryReadWriteDemo1) ;
    end for ;
  end for ;
end TbAxi4_MemoryReadWriteDemo1 ;
```

Figure 6. Configuration for Test Architecture MemoryReadWriteDemo1

For Figure 8

- The VC receives a transaction using **WaitForTransaction**
- It decodes the transaction and generates the appropriate interface signaling
- It is often better to write the signaling code directly rather than calling subprograms, as additional abstraction can lead to code obfuscation
- The transaction interface is an **InOut** record shared between the test sequencer (TestCtrl) and the verification component (VC).

- **OSVVM provides two standard transaction interface records: `AddressBusRecType` and `StreamRecType`, suitable for most verification components.**

```
type AddressBusRecType is record
  Rdy              : bit_max ;
  Ack              : bit_max ;
  Address          : std_logic_vector_max_c ;
  AddrWidth        : integer_max ;
  DataToModel      : std_logic_vector_max_c ;
  DataFromModel    : std_logic_vector_max_c ;
  . . .
end record AddressBusRecType ;
```

Figure 9. An OSVVM Interface

- Streaming interfaces (such as AxiStream, UART, …) all do send and get transactions
  - The Transaction Interfaces: AddressBusRecType and StreamRecType.
  - The Transaction initiation procedures: Read, Write (address bus) vs Send, Get (stream).
- Running the Examples:

```
cd sim
source ../OsvvmLibraries/Scripts/StartUp.tcl
build  ../OsvvmLibraries/OsvvmLibraries.pro
build  ../OsvvmLibraries/DpRam/RunAllTests.pro
```

Figure 11. Compiling and Running DPRAM tests

OSVVM Verification Component Developer's Guide

- Writing an OSVVM VC starts by looking at the interface we want to drive

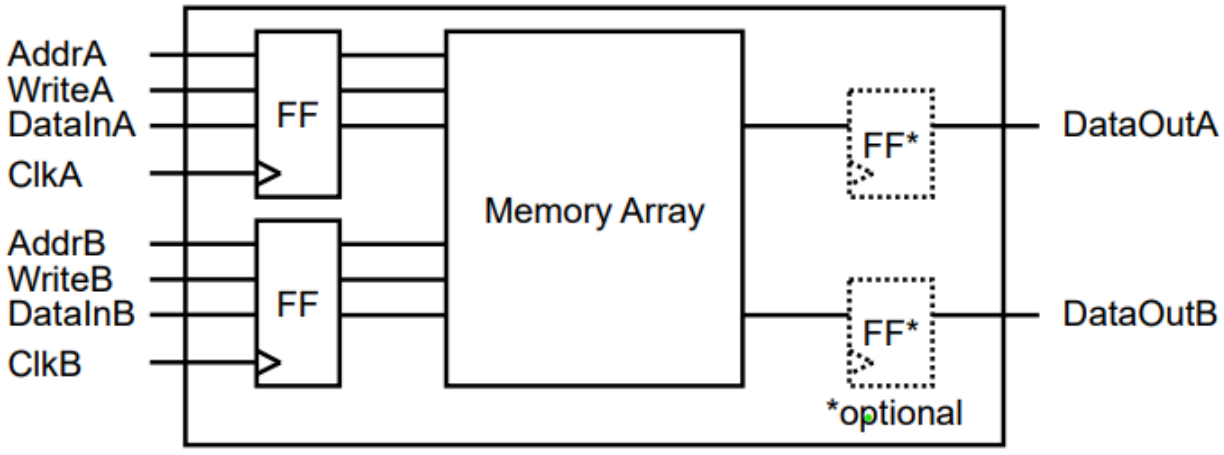

Figure 1. DpRam structure

- OSVVM Address Bus Model Independent Transactions

`Write(TRec, iAddr, iData, [MsgOn])`
`Read(TransactionRec, iAddr, oData, [StatusMsgOn])`
`ReadCheck(TransactionRec, iAddr, iData, [StatusMsgOn])`
`WriteAndRead(TransactionRec, iAddr, iData, oData, [StatusMsgOn])`

Figure 3. Address Bus Model Independent Transactions (a subset)

- **Blocking VC:** Enough for, simple sequantial interfaces like (UART)
- **Asynchronous VC:** Necessary for complex parallel interfaces

Creating a simple blocking VC

For this section getting into pdf and applying the steps is much better