Scoreboard Generic Package

User Guide

User Guide for Release 2017.05

By

Jim Lewis

SynthWorks VHDL Training

Jim@SynthWorks.com

http://www.SynthWorks.com

Table of Contents

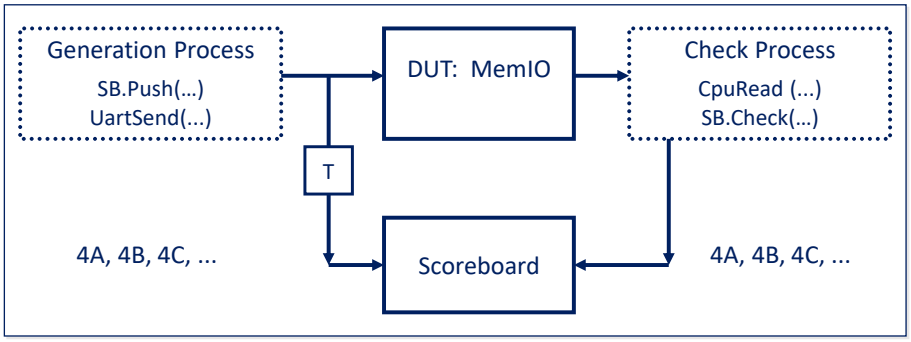
1 ScoreboardGenericPkg Overview 3
2 Using ScoreboardGenericPkg Package Generics 4
3 My Simulator Does Not Support Package Generics 5
4 Scoreboard Command Reference 6
4.1 Basic Operations 6
4.1.1 Scoreboard = Shared Variable 6
4.1.2 Push 6
4.1.3 Check 6
4.1.4 Pop 6
4.1.5 SetAlertLogID 7
4.1.6 GetAlertLogID 7
4.1.7 Find 7
4.1.8 Flush 7
4.1.9 Empty 7
4.1.10 GetItemCount 8
4.1.11 GetCheckCount 8
4.1.12 GetDropCount 8
4.1.13 SetName 8
4.1.14 GetName 8
4.1.15 Getting the Scoreboard Error Count 8
4.2 Tagged Scoreboards 9
4.3 Indexed Scoreboards 9
4.3.1 Setting Array Indices 9
4.3.2 Getting Array Indices 10
4.3.3 Arrays of Scoreboards 10
4.3.4 Arrays of Simple Scoreboards 10
4.3.5 Arrays of Tagged Scoreboards 11
5 Compiling ScoreboardGenericPkg and Friends 11
6 About ScoreboardGenericPkg 11
7 Future Work 12
8 About the Author - Jim Lewis 12

1 ScoreboardGenericPkg Overview

A scoreboard is a data structure used for self-checking in an environment where inputs are closely related to outputs, such as in data transmission (serial ports, networking, …).

Internal to a scoreboard there is a FIFO for holding values and a data checker. The use model is simple. When the testbench stimulus generation process generates a transaction, it first sends the transaction (via push) to the scoreboard and then sends the transaction to the DUT. As a result of the push operation, the scoreboard stores the transaction value in the scoreboard. When the testbench checking process receives a transaction, it sends that value to the scoreboard (via check) to be checked. Internally the scoreboard pops the top value off the FIFO and compares it to the value sent via check using AffirmIf from the AlertLogPkg.

Pictorially scoreboard operations are as shown in the following block diagram.



From a VHDL code perspective, this is as follows.

```
architecture Uart_Test1 of TestCtrl is
  shared variable SB : ScoreboardPType ;
  . . .
begin
  GenerateProc : process
  begin
    SetAlertLogName("UART_Test1") ;
    SB.SetAlertLogId("UART_SB", TB_ID) ;
    SB.Push(X"10") ;
    UartSend(UartTxRec, X"10") ;
    . . .
    SB.Push(X"FF") ;
    UartSend(UartTxRec, X"FF") ;
    TestDone <= TRUE ;
    wait ;
```

```
  end process GenerateProc ;

  CheckProc : process
    variable RcvD : std_logic_vector(7 downto 0);
  begin
    while not TestDone loop
      UartGet(UartRxRec, RcvD) ;
      SB.Check(RcvD) ;
      wait for UART_BAUD_PERIOD ;
    end loop ;
    ReportAlerts ;
  end process CheckProc ;
end architecture UART_Test1 ;
```

Note that the test generation process generates numerous values in using either directed (shown) or random methods, the checking side is a simple loop. Hence, the big advantage of using a scoreboard is that the checking side remains simple and has no need to know what the test generation side is doing.

2 Using ScoreboardGenericPkg Package Generics

The scoreboard package uses package generics to allow the test value type and receive value type to be changed. Furthermore the function to compare the test value with the receive value is passed as a subprogram generic to allow more complicated checking than a simple "=" to be done. In addition there are functions expected_to_string and actual_to_string to help printing values of ExpectedType and ActualType respectively. Hence, the interface to the package is:

```
package ScoreBoardGenericPkg is
generic(
  type ExpectedType ;
  type ActualType ;
  function Match( Actual : ActualType ;
                  Expected : ExpectedType) return boolean ;
  function expected_to_string(A: ExpectedType) return string;
  function actual_to_string  (A: ActualType)   return string
) ;
```

To use the scoreboard for your tests, you will need to create a package instance. The following example creates a scoreboard for integer. Note often you will need to reference the package that defines your types and functions that correspond to ExpectedType,

ActualType, match, expected_to_string, and actual_to_string, however, here for integer all types and functions are available in package std.standard which is implicitly visible.

```
-- reference the package that define types and functions.
-- use std.standard.all ; -- implicitly included
package ScoreBoardPkg_integer is new
  osvvm.ScoreBoardGenericPkg
  generic map (
    ExpectedType        =>  integer,
    ActualType          =>  integer,
    match               =>  std.standard."=",
    expected_to_string  =>  to_string,
    actual_to_string    =>  to_string
  ) ;
```

3 My Simulator Does Not Support Package Generics

All the simulators we test with support package generics. However, we still have you covered. Copy ScoreboardGenericPkg.vhd to ScoreboardPkg_integer_c.vhd. Comment out the generics. Uncomment the subtype and alias declarations that correspond (have the same name as) to the generics. Configure these aliases exactly as described in the section titled "Using Scoreboard Package Generics".

The top of your resulting package should look as follows:

```
-- reference the package that define types and functions.
-- use std.standard.all ; -- implicitly included
package ScoreBoardPkg_integer is
  subtype ExpectedType is integer ;
  subtype ActualType   is integer ;
  alias match is std.standard."=" [integer, integer
                                   return boolean] ;
  alias expected_to_string is
    to_string [integer return string];
  alias actual_to_string is
    to_string [integer return string] ;
```

4 Scoreboard Command Reference

ScoreboardGenericPkg also supports arrays of scoreboards (via indicies) and out of order execution (via tags). These are tersely documented in the following command reference guide.

Caution, anything not documented here is considered experimental (or has been deprecated) and may be removed in a future version. If you decide to use it and want it to stay around, be sure to let us know.

4.1 Basic Operations

4.1.1 Scoreboard = Shared Variable

A scoreboard is a shared variable.

```
shared variable SB : ScoreBoardPType ;
```

If using more than one scoreboard package instance, disambiguate the types by using a fully selected name.

```
shared variable SB_uart :
 work.ScoreBoardPkg_Uart.ScoreBoardPType;
```

4.1.2 Push

Add expected value (ExpectedType) to the scoreboard.

```
SB.Push(ExpectedVal) ;
```

4.1.3 Check

Check a received value (ActualType) with value in scoreboard. If error, increment internal error count.

```
SB.Check(ReceiveVal) ;
```

4.1.4 Pop

Use scoreboard as FIFO, get oldest value. Uses an out mode variable parameter of ExpectedType.

```
SB.Pop(ExpectedVal) ;
```

4.1.5 SetAlertLogID

Create an AlertLogID internal to the Scoreboard. The scoreboard will use this ID for reporting errors to the AlertLog data structure.

```
SB.SetAlertLogID(
  Name => "SB_UART",
  ParentID => OSVVM_ALERTLOG_ID -- use your TB_TOP_ID
) ;
```

There is an alternative form of SetAlertLogID that allows you to pass in an AlertLogID. This is useful if you want the scoreboard to report errors using the models AlertLogID.

```
signal ModelID : AlertLogIDType ;
. . .
ModelID <= GetAlertLogID ("Model Instance Name") ;
. . .
SB.SetAlertLogID(ModelID) ;
```

4.1.6 GetAlertLogID

Get the AlertLogID from the scoreboard internals.

```
SB_ID := SB.GetAlertLogID ;
```

4.1.7 Find

Return the ItemNumber of the oldest expected value that matches the received value. Find + Flush support systems that drop items before they are synchronized.

```
ItemNum := SB.Find(ReceiveVal) ;
```

4.1.8 Flush

Quietly drop all values whose item number is less than the specified item number. Find + Flush support systems that drop items before they are synchronized.

```
SB.Flush(ItemNum) ;
```

4.1.9 Empty

Check if the Scoreboard is empty.

```
if not SB.Empty then …
```

4.1.10 GetItemCount

Get number of items put into the scoreboard.

```
print("…" & to_string(SB.GetItemCount));
```

4.1.11 GetCheckCount

Get number of items checked by the scoreboard.

```
print("…" & to_string(SB.GetCheckCount));
```

4.1.12 GetDropCount

Get number of items dropped by the scoreboard.

```
print("…" & to_string(SB.GetDropCount));
```

4.1.13 SetName

Gives the scoreboard a name for reporting. Use if using a single ALertLogID for multiple items (scoreboards or other).

```
SB.SetName("Uart Scoreboard") ;
```

4.1.14 GetName

Get the scoreboard name

```
print("…" & SB.GetName) ;
```

4.1.15 Getting the Scoreboard Error Count

The scoreboard reports errors to the AlertLog data structure. Hence, the error count will be reported with ReportAlerts.

If you require a unique count for each scoreboard, be sure to give the the scoreboard a unique name when calling SetAlertLogID. Then call GetAlertCount from the AlertLogPkg. This is shown below.

```
ScoreboardErrorCount := GetAlertCount(SB.GetAlertLogID) ;
```

There is also a method, GetErrorCount, that returns this value.

```
ErrCnt := SB.GetErrorCount ;
```

In release 2016.07 and earlier releases, GetErrorCount returns the value stored in an internal error counter. In a future revision, it will return the value GetAlertCount(SB.GetAlertLogID). Hence, be sure to SetAlertLogID.

4.2 Tagged Scoreboards

Tagged Scoreboards are used for systems that allow transactions to execute out of order.

Tags are represented as string values (since most types convert to string using to_string). A tag value is specified as the first value in the calls to push, check, and pop, such as shown below. In all examples, ExpectedVal has the type ExpectedType, and ReceiveVal has the type ActualType.

```
SB.Push("WriteOp", ExpectedVal) ;
SB.Check("WriteOp", ReceiveVal) ;
SB.Pop("WriteOp", ExpectedVal) ;

if SB.Empty("MyTag")  then   …
```

For Check (and Pop), the item checked (or returned) is the oldest item with the matching tag.

```
ItemNum := SB.Find("ReadOp", ReceiveVal);
SB.Flush("ReadOp", ItemNum) ;
```

For Flush, only items matching the tag are removed. In some systems, it may be appropriate to do the Find with the tag and the flush without the tag.

4.3 Indexed Scoreboards

Indexed scoreboards emulates arrays of protected types, since the language does not support this.

Indexed scoreboards are for systems, such as a network switch that have multiple scoreboards that are most conveniently represented as an array.

4.3.1 Setting Array Indices

Use SetArrayIndex to create the array indices. The following creates an array with indices 1 to 5:

```
SB.SetArrayIndex(5) ;
```

To create array indices with a different range, such as 3 to 8, use the following.

```
SB.SetArrayIndex(3, 8) ;
```

Slicing and null arrays of scoreboards are not supported. Negative indices are supported.

4.3.2 Getting Array Indices

Use GetArrayIndex to get the indices as an integer_vector.

```
Index_IV := SB.GetArrayIndex ;
```

Use GetArrayLength to determine the number of scoreboards (effectively the length of the array).

```
Index_int := SB.GetArrayLength ;
```

4.3.3 Arrays of Scoreboards

The following operations are appropriate for any array of scoreboards. Procedures and functions not documented here are from AlertLogPkg.

```
-- Create 3 indexed scoreboards
SB.SetArrayIndex(1, 3);

-- SB_TOP_ID via AlertLogPkg
SB_TOP_ID := GetAlertLogID("Scoreboard TOP") ;
SB.SetAlertLogID(1, "SB1", SB_TOP_ID) ;
SB.SetAlertLogID(2, "SB2", SB_TOP_ID) ;
SB.SetAlertLogID(3, "SB3", SB_TOP_ID) ;

-- display PASSED logs via AlertLogPkg
SetLogEnable(SB_TOP_ID, PASSED, TRUE) ;

-- Turn off Error messages for SB1
SB1_ID := GetAlertLogID(1) ;
SetAlertEnable(SB1_ID, ERROR, FALSE) ;

-- test completion via AlertLogPkg (the easy way)
ReportAlerts ;
```

4.3.4 Arrays of Simple Scoreboards

The following are operations appropriate for arrays of simple scoreboards. In all examples, 3 is the index, ExpectedVal has the type ExpectedType, and ReceiveVal has the type ActualType.

```
SB.Push(3, ExpectedVal) ;
SB.Check(3, ReceiveVal) ;
SB.Pop(3, ExpectedVal) ;

if SB.Empty(3)  then   …
```

```
ItemNum := SB.Find(3, ReceiveVal);
SB.Flush(3, ItemNum) ;
```

4.3.5 Arrays of Tagged Scoreboards

The following are operations appropriate for arrays of tagged scoreboards. In all examples, 3 is the index, values in quotes are the tag value, ExpectedVal has the type ExpectedType, and ReceiveVal has the type ActualType. Operations where either using a tag or not is appropriate are marked with "**".

```
SB.Push(3, "WriteOp", ExpectedVal) ;
SB.Check(3, "WriteOp", ReceiveVal) ;
SB.Pop(3, "WriteOp", ExpectedVal) ;

if SB.Empty(3, "MyTag")  then   …  -- **
if SB.Empty(3)  then   …  -- **

ItemNum := SB.Find(3, "Red", ReceiveVal);
-- two possible alternatives
SB.Flush(3, "Red", ItemNum) ; -- **
SB.Flush(3, ItemNum) ; -- **
```

5 Compiling ScoreboardGenericPkg and Friends

See OSVVM_release_notes.pdf for the current compilation directions. Rather than referencing individual packages, we recommend using the context declaration:

```
library OSVVM ;
  context osvvm.OsvvmContext ;
```

6 About ScoreboardGenericPkg

ScoreboardGernericPkg was developed and is maintained by Jim Lewis of SynthWorks. It has been used for years in our Advanced VHDL Testbenches and Verification class.

We held it back from the OSVVM library waiting for current tool releases to support VHDL-2008 generic packages and resolve some of the bugs. At this point, main stream VHDL simulators seem to be ready.

Please support our effort in supporting ScoreboardGernericPkg and OSVVM by purchasing your VHDL training from SynthWorks.

ScoreboardGernericPkg is released under the Perl Artistic open source license. It is free (both to download and use - there are no license fees).

If you add features to the package, please donate them back under the same license as candidates to be added to the standard version of the package. If you need features, be sure to contact us. I blog about the packages at http://www.synthworks.com/blog. We also support the OSVVM user community and blogs through http://www.osvvm.org.

Currently the OSVVM family of packages can be downloaded from either osvvm.org or from our GitHub site: https://github.com/JimLewis/OSVVM.

Find any innovative usage for the package? Let us know, you can blog about it at osvvm.org.

7 Future Work

ScoreboardGernericPkg is a work in progress and will be updated from time to time.

Caution, undocumented items are experimental and may be removed in a future version.

8 About the Author - Jim Lewis

Jim Lewis, the founder of SynthWorks, has thirty plus years of design, teaching, and problem solving experience. In addition to working as a Principal Trainer for SynthWorks, Mr Lewis has done ASIC and FPGA design, custom model development, and consulting.

Mr. Lewis is chair of the IEEE 1076 VHDL Working Group (VASG) and is the primary developer of the Open Source VHDL Verification Methodology (OSVVM.org) packages. Neither of these activities generate revenue. Please support our volunteer efforts by buying your VHDL training from SynthWorks.

If you find bugs these packages or would like to request enhancements, you can reach me at jim@synthworks.com.