AlertLog Package
User Guide

User Guide for Release 2020.05

By

Jim Lewis

SynthWorks VHDL Training

Jim@SynthWorks.com

http://www.SynthWorks.com

Table of Contents

1 Whats New: A Quick Highlight 4
2 AlertLogPkg Overview 4
3 AlertLogPkg Use Models 5
4 Simple Mode Example: Global Alert Counter 5
5 Hierarchy Mode Example: Separate Alert Counters 6
6 Package References 7
7 Name and Mode Reference 8
7.1 SetAlertLogName: Setting the Test Name 8
7.2 GetAlertLogName 8
7.3 GetAlertLogID: Creating Hierarchy 8
7.4 FindAlertLogID: Find an AlertLogID 9
7.5 PathTail - Used to Discover Instance Name of a Component 9
7.6 GetAlertLogParentID 9
8 Alert Method Reference 9
8.1 AlertType 9
8.2 Alert 9
8.3 AlertIf and AlertIfNot 10
8.4 AlertIfEqual and AlertIfNotEqual 10
8.5 AlertIfDiff 11
8.6 IncrementAlertCount 11
8.7 SetAlertEnable: Alert Enable / Disable 11
8.8 GetAlertEnable 12
8.9 SetAlertStopCount: Alert Stop Counts 12
8.10 GetAlertStopCount 12
8.11 ClearAlertStopCounts: Reset Alert Stop Counts 12
8.12 ClearAlerts: Reset Alert Counts 13
8.13 ClearAlertStopCounts: Reset Alert and Alert Stop Counts 13
8.14 SetGlobalAlertEnable: Alert Global Enable / Disable 13
8.15 GetGlobalAlertEnable 13
9 Reporting and Getting Alert Counts 13
9.1 AlertCountType 13
9.2 Reporting Alerts: ReportAlerts, ReportNonZeroAlerts 14
9.2.1 Report Summary 14
9.2.2 Report Details 14
9.2.3 ReportAlerts: Reporting Alerts 15
9.2.4 ReportNonZeroAlerts 16
9.2.5 Overloaded ReportAlerts for Reporting AlertCounts 16

9.3 GetAlertCount 16
9.4 GetEnabledAlertCount 16
9.5 GetDisabledAlertCount 17
9.6 Math on AlertCountType 17
9.7 SumAlertCount: AlertCountType to Integer Error Count 17
10 Log Method Reference 17
10.1 LogType 17
10.2 Log 17
10.3 SetLogEnable: Enable / Disable Logging 18
10.4 Reading Log Enables from a FILE 18
10.5 IsLogEnabled /GetLogEnable 18
11 Alert and Log Prefix and Suffix to Message 19
12 Affirmation Reference 19
12.1 AffirmIf / AffirmIfNot 19
12.2 AffirmIfEqual 20
12.3 AffirmIfDiff 20
12.4 GetAffirmCount 21
12.5 IncAffirmCount 21
13 Alert and Log Output Control Options 21
13.1 SetAlertLogJustify 21
13.2 OsvvmOptionsType 21
13.3 SetAlertLogOptions: Configuring Report Options 22
13.4 ReportAlertLogOptions: Print Report Options 23
13.5 Getting AlertLog Report Options 23
14 Deallocating and Re-initializing the Data structure 24
14.1 DeallocateAlertLogStruct 24
14.2 InitializeAlertLogStruct 24
15 Compiling and Referencing AlertLogPkg and Friends 24
16 Release Notes for 2020.05 24
17 About AlertLogPkg 25
18 Future Work 26
19 About the Author - Jim Lewis 26

1 Whats New: A Quick Highlight

The big enhancement in this revision is the addition of PASSED counts for both the top level error summary as well as a per hierarchy level (verification model and sublevels). For more details see the release notes at the end.

2 AlertLogPkg Overview

VHDL assert statements are a limited form of an alert and log filtering utility. In the simulator GUI, you can set an assertion level that will stop a simulation. In the simulator GUI, you can turn off some assertions from printing. However, none of this capability can be configured in VHDL, and in addition, at the end of a test, there is no way to retrieve a count of the ERROR level assertions that have occurred.

The AlertLogPkg provides Alert and Log procedures that replace VHDL assert statements and gives VHDL direct access to enabling and disabling of features, retrieving alert counts, and set stop counts (limits). All of these features can be used in either a simple global mode or a hierarchy of alerts.

Alert simplifies signaling errors (when an error occurs) and reporting errors (passed or failed when a test finishes). Alerts have the levels FAILURE, ERROR, and WARNING. Each level is counted and tracked in an internal data structure. Within the data structure, each of these can be enabled or disabled. A test can be stopped if an alert value has been signaled too many times. Stop values for each counter can be set. The default for FAILURE is 0 and ERROR and WARNING are integer'right. If all test errors are reported as an alert, at the end of the test, a report can be generated which provides pass/fail and a count of the different alert values.

Logs provide a mechanism to conditionally print information. This verbosity control allows messages (such as debug, DO254 passed test reports, or info) that are too detailed for normal testing to be printed when specifically enabled. VIA the simulator GUI, assert has this capability to a limited degree.

AssertLogPkg uses TranscriptPkg to print to either std.textio.OUTPUT, a file, or both. When the TranscriptFile is opened, alert and log print to the TranscriptFile, otherwise, they print to std.textio.OUTPUT. For more details on TranscriptPkg see the TranscriptPkg User Guide (TranscriptPkg_user_guide.pdf).

Already using another package for alerts and verbosity control? The AlertLogPkg has an extensive API that will allow you to retrieve any error information reported by the OSVVM packages and allow you to print a summary of results via another package.

3 AlertLogPkg Use Models

Alerts and Logs may be used in either simple or hierarchy model.

In simple mode, there is single global alert counter that accumulates the number of FAILURE, ERROR, and WARNING level alerts for the entire testbench. When a test completes, a summary of the total number of errors as well as the errors for each level can be produced.

In hierarchy mode, each verification model and/or source of alerts has its own set of alert counters. This allows each source of alerts and logs to be traced back to its source. Counts from lower levels propagate up to the top level counter. Each level in the hierarchy also supports separate verbosity control. When a test completes, an error report can be produced for both the top level and each level in the alert hierarchy.

4 Simple Mode Example: Global Alert Counter

By default, there is a single global alert counter. All designs that use alert or log need to reference the package AlertLogPkg.

```
use osvvm.OsvvmContext ;
architecture Test1 of tb is
```

Use Alert to flag an error, AlertIf to flag an error when a condition is true, or AlertIfNot to flag an error when a condition is false (similar to assert). Alerts can be of severity FAILURE, ERROR, or WARNING.

```
--                    message,          level
When others =>  Alert("Illegal State", FAILURE) ;
. . .
--          condition,                 message,              level
AlertIf(ActualData /= ExpectedData, "Data Miscompare …", ERROR) ;
. . .
read(Buf, A, ReadValid) ;
--             condition, message,             level
AlertIfNot( ReadValid, "read of A failed", FAILURE) ;
```

The output for an alert is as follows. Alert adds the time at which the log occurred.

```
%% Alert ERROR   Data Miscompare … at 20160 ns
```

When a test completes, use ReportAlerts to provide a summary of errors.

```
ReportAlerts ;
```

When a test passes, the following message is generated:

```
%% DONE  PASSED  t1_basic  Passed: 8  Affirmations Checked: 8  at 120180 ns
```

When a test fails, the following message is generated (on a single line):

```
%% DONE  FAILED  t1_basic  Total Error(s) = 2  Failures: 0  Errors: 1  Warnings: 1
Passed: 7  Affirmations Checked: 8 at 120180  ns
```

Similar to assert, by default, when an alert FAILURE is signaled, a test failed message (see ReportAlerts) is produced and the simulation is stopped. This action is controlled by a stop count. The following call to SetAlertStopCount, causes a simulation to stop after 20 ERROR level alerts are received.

```
SetAlertStopCount(ERROR, 20) ;
```

Alerts can be enabled by a general enable, SetGlobalAlertEnable (disables all alert handling) or an enable for each alert level, SetAlertEnable. The following call to SetAlertEnable disables WARNING level alerts.

```
SetGlobalAlertEnable(TRUE) ; -- Default
SetAlertEnable(WARNING, FALSE) ;
```

Logs are used for verbosity control. Log level values are ALWAYS, DEBUG, PASSED, and INFO.

```
Log ("A message", DEBUG) ;
```

Log formats the output as follows.

```
%% Log   DEBUG  A Message  at 15110 ns
```

Each log level is independently enabled or disabled. This allows the testbench to support debug or passed report messages and only enable them during the appropriate simulation run. The log ALWAYS is always enabled, all other logs are disabled by default. The following call to SetLogEnable enables DEBUG level logs.

```
SetLogEnable(DEBUG, TRUE) ;
```

5 Hierarchy Mode Example: Separate Alert Counters

In hierarchy mode, each verification model and/or source of alerts has its own set of alert counters. This allows each source of alerts and logs to be traced back to its source. Counts from lower levels propagate up to the top level counter. The ultimate goal of using hierarchy mode is to get a summary of errors for each model and/or source of alerts in the testbench:

```
%% DONE  FAILED  Testbench  Total Error(s) = 11  Failures: 1  Errors: 20
Warnings: 0  Passed: 15  Affirmations Checked: 25  at 10117000 ns
%%    Default            Failures: 0  Errors: 0  Warnings: 0  Passed: 0
%%    OSVVM              Failures: 0  Errors: 0  Warnings: 0  Passed: 0
%%    U_CpuModel         Failures: 0  Errors: 6  Warnings: 0  Passed: 0
%%      Data Error       Failures: 0  Errors: 4  Warnings: 0  Passed: 10
%%      Protocol Error   Failures: 1  Errors: 0  Warnings: 0  Passed: 0
%%    U_UART_TX          Failures: 0  Errors: 0  Warnings: 0  Passed: 0
%%    U_UART_RX          Failures: 0  Errors: 6  Warnings: 0  Passed: 5
```

Using hierarchy mode requires more work, but not much. Inside AlertLogPkg there is a data structure hidden inside a shared variable. Each level in a hierarchy is referenced with an AlertLogID - which is currently an integer index into the data structure. As a result, each model must get (allocate) an AlertLogID and then reference the AlertLogID when signaling alerts. Other than referencing the AlertLogID, the usage is identical.

A new AlertLogID is created by calling the function GetAlertLogID. GetAlertLogID has two parameters: Name and ParentID. Name is a string of the ALERT (that prints when the alert prints). ParentID is of type AlertLogIDType.

In the following example, CPU_ALERT_ID uses the instance name of the model as its name. Since it is a top level model, it uses ALERTLOG_BASE_ID (which is also the default) as its ParentID. DATA_ALERT_ID is an alert counter within the CPU. So it uses a string as its name and CPU_ALERT_ID as its ParentID.

```
constant CPU_ALERT_ID : AlertLogIDType :=
    --            Name (string value),         Parent AlertLogID
    GetAlertLogID(PathTail(CpuModel'PATH_NAME), ALERTLOG_ BASE_ID) ;
constant DATA_ALERT_ID : AlertLogIDType :=
    GetAlertLogID("Data Error", CPU_ALERT_ID) ;
constant PROTOCOL_ALERT_ID : AlertLogIDType :=
    GetAlertLogID("PROTOCOL Error", CPU_ALERT_ID) ;
```

The AlertLogID is specified first in calls to Alert, Log, SetAlertEnable, SetAlertStopCount, and SetLogEnable.

```
Alert(CPU_ALERT_ID, "CPU Error", ERROR) ;
AlertIf(PROTOCOL_ALERT_ID, inRdy /= '0', "during CPU Read operation", FAILURE);
AlertIfNotEqual(DATA_ALERT_ID, ReadData, ExpectedData, "Actual /= Expected Data");
--             AlertLogID,    Level,    Enable
SetAlertEnable(CPU_ALERT_ID,  WARNING,  FALSE) ;
--                AlertLogID,   Level,   Count
SetAlertStopCount(CPU_ALERT_ID, ERROR,      20) ;

Log(UartID, DEBUG, "Uart Parity Received") ;

--       AlertLogID,  Level,    Enable, DescendHierarchy
SetLogEnable(UartID,  WARNING,  FALSE,  FALSE) ;
```

Printing of Alerts and Logs include the AlertLogID.

```
%% Alert FAILURE in CPU_1, Expect data XA5A5 at 2100 ns
%% Log   ALWAYS  in UART_1,  Parity Error  at 2100 ns
```

6 Package References

Using AlertLogPkg requires the following package references:

```
library osvvm ;
use osvvm.OsvvmGlobalPkg.all ;
use osvvm.AlertLogPkg.all ;
```

Alternately use the OSVVM context clause:

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

7 Name and Mode Reference

Simple mode is the default. Hierarchical mode is initiated by creating an AlertLogID with GetAlertLogID. Once initiating hierarchical mode, it is not possible to go back to simple mode.

All AlertLogPkg alerts, logs, and affirmations support overloading for simple mode and hierarchical mode. The only difference of a hierarchical mode subprogram is that the first parameter is the AlertLogID that was obtained with AlertLogID.

Note that a simple mode subprogram may be called when in hierarchy mode. It will report using the AlertLogID, ALERTLOG_DEFAULT_ID.

7.1 SetAlertLogName: Setting the Test Name

SetAlertLogName sets the name of the current test that is printed by ReportAlerts. This is particularly recommended if a test can end due to a stop count, such as FAILURE as ReportAlerts is automatically called.

```
procedure SetAlertLogName(Name : string ) ;
. . .
SetAlertLogName("Uart1") ;
```

7.2 GetAlertLogName

GetAlertLogName returns the string value of name associated with an AlertLogID. If no AlertLogID is specified, it will return the name set bySetAlertLogName.

```
impure function GetAlertLogName(AlertLogID : AlertLogIDType:= ALERTLOG_BASE_ID)
return string ;
```

7.3 GetAlertLogID: Creating Hierarchy

Each level in a hierarchy is referenced with an AlertLogID. The function, GetAlertLogID, creates a new AlertLogID. If an AlertLogID already exists for the specified name, GetAlertLogID will return its AlertLogID. It is recommended to use the instance label as the Name. The interface for GetAlertLogID is as follows.

```
impure function GetAlertLogID(
  Name : string ;
  ParentID : AlertLogIDType := ALERTLOG_BASE_ID ;
  CreateHierarchy : Boolean := TRUE
) return AlertLogIDType ;
```

The CreateHierarchy parameter is intended to allow packages to use a unique AlertLogID for reporting Alerts without creating hierarchy in ReportAlerts. As a function, GetAlertLogID can be called while elaborating the design by using it to initialize a constant or signal:

```
Constant UartID : AlertLogIDType :=
 --                            Name,        Parent AlertLogID
       GetAlertLogID("UART_1", ALERTLOG_BASE_ID);
```

7.4 FindAlertLogID: Find an AlertLogID

The function, FindAlertLogID, finds an existing AlertLogID. If the AlertLogID is not found, ALERTLOG_ID_NOT_FOUND is returned. The interface for FindAlertLogID is as follows.

```
impure function FindAlertLogID(Name : string ; ParentID : AlertLogIDType)
    return AlertLogIDType ;
impure function FindAlertLogID(Name : string ) return AlertLogIDType ;
```

Note the single parameter FindAlertLogID is only useful when there is only one AlertLogID with a particular name (such as for top-level instance names). As a function, FindAlertLogID can be called while elaborating the design by using it to initialize a constant or signal.

```
constant UartID : AlertLogIDType := FindAlertLogID(Name => "UART_1") ;
```

Caution: only use FindAlertLogID when it is known that the ID has already been created - such as in a testbench where the testbench components have already been elaborated, as otherwise, it is appropriate to use GetAlertLogID.

7.5 PathTail - Used to Discover Instance Name of a Component

When used in conjunction with attribute PATH_NAME applied to an entity name, PathTail returns the instance name of component.

```
constant CPU_ALERT_ID : AlertLogIDType :=
    --            Name (string value),         Parent AlertLogID
    GetAlertLogID(PathTail(CpuModel'PATH_NAME), ALERTLOG_ BASE_ID) ;
```

7.6 GetAlertLogParentID

Get the AlertLogID of the parent of a specified AlertLogID.

```
impure function GetAlertLogParentID(AlertLogID : AlertLogIDType) return
AlertLogIDType ;
```

8 Alert Method Reference

Alert is intended for parameter error checking. For self-checking see affirmations.

8.1 AlertType

Alert levels can be FAILURE, ERROR, or WARNING.

```
type AlertType is (FAILURE, ERROR, WARNING) ;
```

8.2 Alert

Alert generates an alert. The following overloading is supported.

```
procedure alert(
  AlertLogID   : AlertLogIDType ;
  Message   : string ;
```

```
  Level     : AlertType := ERROR
) ;

procedure Alert( Message : string ; Level : AlertType := ERROR ) ;
```

Usage of alert:

```
. . .
Alert(UartID, "Uart Parity", ERROR) ;

Alert("Uart Parity") ;  -- ERROR by default
```

8.3 AlertIf and AlertIfNot

Alert has two conditional forms, AlertIf and AlertIfNot. The following is their overloading.

```
-- with an AlertLogID
procedure AlertIf( AlertLogID : AlertLogIDType ; condition : boolean ;
    Message : string ; Level : AlertType := ERROR ) ;
impure function  AlertIf( AlertLogID : AlertLogIDType ; condition : boolean ;
    Message : string ; Level : AlertType := ERROR ) return boolean ;
procedure AlertIfNot( AlertLogID : AlertLogIDType ; condition : boolean ;
    Message : string ; Level : AlertType := ERROR )  ;
impure function  AlertIfNot( AlertLogID : AlertLogIDType ; condition : boolean ;
    Message : string ; Level : AlertType := ERROR ) return boolean ;

-- without an AlertLogID
procedure AlertIf( condition : boolean ;
    Message : string ; Level : AlertType := ERROR ) ;
impure function  AlertIf( condition : boolean ;
    Message : string ; Level : AlertType := ERROR ) return boolean ;
procedure AlertIfNot( condition : boolean ;
    Message : string ; Level : AlertType := ERROR ) ;
impure function  AlertIfNot( condition : boolean ;
    Message : string ; Level : AlertType := ERROR ) return boolean ;
```

Usage of conditional alerts:

```
-- with an AlertLogID
AlertIf(UartID, Break='1', "Uart Break", ERROR) ;
AlertIfNot(UartID, ReadValid, "Read", FAILURE);

-- without an AlertLogID
AlertIf(Break='1', "Uart Break", ERROR) ;
AlertIfNot(ReadValid, "Read Failed", FAILURE) ;
```

The function form is convenient for use for conditional exit of a loop.

```
exit AlertIfNot(UartID, ReadValid, "in ReadCovDb while reading …", FAILURE) ;
```

8.4 AlertIfEqual and AlertIfNotEqual

Alert form AlertIfEqual and AlertIfNotEqual to check two values. In the following, AType can be std_logic, std_logic_vector, unsigned, signed, integer, real, character, string, or time.

```
-- with an AlertLogID
```

```
procedure AlertIfEqual( AlertLogID : AlertLogIDType ;  L, R : AType ;
    Message : string ; Level : AlertType := ERROR )  ;
procedure AlertIfNotEqual( AlertLogID : AlertLogIDType ;  L, R : AType ;
    Message : string ; Level : AlertType := ERROR )  ;

-- without an AlertLogID
procedure AlertIfEqual( L, R : AType ; Message : string ;
    Level : AlertType := ERROR )  ;  Message : string ;
procedure AlertIfNotEqual( L, R : AType ;
    Level : AlertType := ERROR )  ;
```

8.5 AlertIfDiff

Alert form AlertIfDiff is for comparing two files.

```
-- with an AlertLogID
procedure AlertIfDiff (AlertLogID : AlertLogIDType ; Name1, Name2 : string;
    Message : string := "" ; Level : AlertType := ERROR ) ;
procedure AlertIfDiff (AlertLogID : AlertLogIDType ; file File1, File2 : text;
    Message : string := "" ; Level : AlertType := ERROR ) ;
-- without an AlertLogID
procedure AlertIfDiff (Name1, Name2 : string; Message : string := "" ;
    Level : AlertType := ERROR ) ;
procedure AlertIfDiff (file File1, File2 : text; Message : string := "" ;
    Level : AlertType := ERROR ) ;
```

8.6 IncrementAlertCount

Intended as a silent alert. Used by CoveragePkg.

```
-- Hierarchy
procedure IncAlertCount(   -- A silent form of alert
  AlertLogID   : AlertLogIDType ;
  Level        : AlertType := ERROR
) ;
-- Global Alert Counters
procedure IncAlertCount( Level : AlertType := ERROR ) ;
```

8.7 SetAlertEnable: Alert Enable / Disable

Alerts are enabled by default. SetAlertEnable allows alert levels to be individually enabled or disabled. When used without AlertLogID, SetAlertEnable sets a value for all AlertLogIDs.

```
procedure SetAlertEnable(Level : AlertType ;  Enable : boolean) ;
. . .
--                       Level,      Enable
SetAlertEnable(WARNING, FALSE) ;
```

When an AlertLogID is used, SetAlertEnable sets a value for that AlertLogID, and if DescendHierarchy is TRUE, it's the AlertLogID's of its children.

```
procedure SetAlertEnable(AlertLogID : AlertLogIDType ;  Level : AlertType ;
Enable : boolean ; DescendHierarchy : boolean := TRUE) ;
```

8.8 GetAlertEnable

Get the value of the current alert enable for either a specific AlertLogId or for the global alert counter.

```
-- Hierarchy
impure function GetAlertEnable(AlertLogID : AlertLogIDType ;  Level : AlertType)
return boolean ;
-- Global Alert Counter
impure function GetAlertEnable(Level : AlertType) return boolean ;
```

8.9 SetAlertStopCount: Alert Stop Counts

When an alert stop count is reached, the simulation stops. When used without AlertLogID, SetAlertStopCount sets the alert stop count for the top level to the specified value if the current count is integer'right, otherwise, it sets it to the specified value plus the current count.

```
procedure SetAlertStopCount(Level : AlertType ;  Count : integer) ;
. . .
--                Level,     Count
SetAlertStopCount(ERROR,      20) ;  -- Stop if 20 errors occur
```

When used with an AlertLogID, SetAlertStopCount sets the value for the specified AlertLogID and all of its parents. At each level, the current alert stop count is set to the specified value when the current count is integer'right, otherwise, the value is set to the specified value plus the current count.

```
procedure SetAlertStopCount(AlertLogID : AlertLogIDType ;
    Level : AlertType ;  Count : integer) ;
. . .
--                AlertLogID,  Level,      Count
SetAlertStopCount(UartID,      ERROR,      20) ;
```

By default, the AlertStopCount for WARNING and ERROR are integer'right, and FAILURE is 0.

8.10 GetAlertStopCount

Get the value of the current alert stop count for either a specific AlertLogId or for the global alert counter.

```
-- Hierarchy
impure function GetAlertStopCount(
  AlertLogID : AlertLogIDType ;
  Level : AlertType
) return integer ;
-- Global Alert Stop Count
impure function GetAlertStopCount(Level : AlertType) return integer ;
```

8.11 ClearAlertStopCounts: Reset Alert Stop Counts

ClearAlerts resets alert stop counts back to their default.

```
procedure ClearAlertStopCounts ;
```

8.12 ClearAlerts: Reset Alert Counts

ClearAlerts resets alert counts to 0.

```
procedure ClearAlerts ;
```

8.13 ClearAlertStopCounts: Reset Alert and Alert Stop Counts

ClearAlerts resets alert counts to 0 and alert stop counts back to their default.

```
procedure ClearAlertCounts ;
```

8.14 SetGlobalAlertEnable: Alert Global Enable / Disable

SetGlobalAlertEnable allows Alerts to be globally enabled and disabled. The intent is to be able to disable all alerts until the system goes into reset. Alerts are enabled by default.

```
procedure SetGlobalAlertEnable (A : EnableType := TRUE) ;
impure function SetGlobalAlertEnable (A : EnableType := TRUE) return EnableType ;
```

Suppress all alerts before reset by turning alerts off during elaboration with a constant declaration and then turning them back on later.

```
InitAlerts : Process
  constant DisableAlerts : boolean := SetGlobalAlertEnable(FALSE);
begin
  wait until nReset = '1' ; --  Deassertion of reset
  SetGlobalAlertEnable(TRUE)  ;  -- enable alerts
```

For an alternate methodology, see ClearAlerts.

8.15 GetGlobalAlertEnable

Get the current value of the global alert enable

```
  impure function GetGlobalAlertEnable return boolean ;
```

9 Reporting and Getting Alert Counts

9.1 AlertCountType

Alerts are stored as a value of AlertCountType.

```
subtype AlertIndexType is AlertType range FAILURE to WARNING ;
type    AlertCountType is array (AlertIndexType) of integer ;
```

CAUTION: When working with values of AlertCountType, be sure to use named association as the type ordering may change in the future.

```
variable ExternalErrors : AlertCountType ;
. . .
ExternalErrors := (FAILURE => 0, ERROR => 6, WARNING => 0) ;
```

9.2 Reporting Alerts: ReportAlerts, ReportNonZeroAlerts

9.2.1 Report Summary

The report summary is the first line printed by ReportAlerts/ReportNonZeroAlerts

```
%% DONE  FAILED  t17_print_ctrl  Total Error(s) = 2  Failures: 0  Errors: 0
Warnings: 0   Total Disabled Error(s) = 2  Failures: 0  Errors: 2  Warnings: 0
Passed: 0  Affirmations Checked: 2  at 20160 ns
```

Note in the above message, two affirmations were done and they both resulted in an error, however, the error was disabled (see SetAlertEnable) so it reported as a Disabled Error.

Options can be enabled or disabled using SetAlertLogOptions. Let's look at the report piece by piece.

If Total Errors /= 0, then the following part of the message prints. In this case, Total Errors is nonzero since FailedOnDisabledErrors is enabled (default) and the disabled errors count as test errors. Hence, Failures, Errors, and Warnings are all 0.

```
Total Error(s) = 2  Failures: 0  Errors: 0  Warnings: 0
```

If FailedOnDisabledErrors is disabled, then the same test will PASS as shown below.

```
%% DONE  PASSED  t17_print_ctrl  Total Disabled Error(s) = 2  Passed: 0
Affirmations Checked: 2  at 20160 ns
```

Next is the Total Disabled Errors summary. It will print if the DisabledAlerts /= 0 or if PrintDisabledAlerts is enabled (default is disabled).

```
Total Disabled Error(s) = 2
```

Next is the details of the Disabled Errors. These will print if FailedOnDisabledErrors is enabled and DisabledAlerts /= 0 or if PrintDisabledAlerts is enabled.

```
Total Disabled Error(s) = 2  Failures: 0  Errors: 2  Warnings: 0
```

Next is the printing of the pass and affirmation counts. If PrintPassed is enabled (default), the passed count will be printed. If PrintAffirmations is enabled (default is disabled) or Affirmation Count /= 0 then the passed and affirmation counts will be printed.

```
Passed: 0  Affirmations Checked: 2
```

9.2.2 Report Details

When in hierarchy mode, a summary for each AlertLogID will be printed. This can include the following details. Each summary starts with "%%".

```
%%    Default   Failures: 0  Errors: 2  Warnings: 0  Disabled Failures: 0  Errors:
0  Warnings: 0  Passed: 2  Affirmations: 4
%%    OSVVM     Failures: 0  Errors: 0  Warnings: 0  Disabled Failures: 0  Errors:
0  Warnings: 0  Passed: 0  Affirmations: 0
%%    Cpu1      Failures: 0  Errors: 2  Warnings: 0  Disabled Failures: 0  Errors:
0  Warnings: 0  Passed: 2  Affirmations: 4
```

```
%%    UartTx    Failures: 0  Errors: 0  Warnings: 0  Disabled Failures: 0  Errors:
2  Warnings: 0  Passed: 2  Affirmations: 4
%%    UartRx    Failures: 0  Errors: 2  Warnings: 0  Disabled Failures: 0  Errors:
0  Warnings: 0  Passed: 2  Affirmations: 4
```

The Failures, Errors, and Warnings always prints.

Disabled Failures, Errors, and Warnings only print if PrintDisabledAlerts is enabled (default is disabled).

Passed only prints if PrintPassed is enabled (default).

Affirmations only prints if PrintAffirmations is enabled (default is disabled).

With PrintDisabledAlerts disabled (default), PrintPassed enabled (default), and PrintAffirmations disabled (default), and DisabledAlerts = 0, the report will be simplified to the following.

```
%%    Default   Failures: 0  Errors: 2  Warnings: 0  Passed: 2
%%    OSVVM     Failures: 0  Errors: 0  Warnings: 0  Passed: 0
%%    Cpu1      Failures: 0  Errors: 2  Warnings: 0  Passed: 2
%%    UartTx    Failures: 0  Errors: 0  Warnings: 0  Passed: 2
%%    UartRx    Failures: 0  Errors: 2  Warnings: 0  Passed: 2
```

9.2.3 ReportAlerts: Reporting Alerts

At test completion alerts are reported with ReportAlerts.

```
procedure ReportAlerts (
    Name            : string := "" ;
    AlertLogID      : AlertLogIDType := ALERTLOG_BASE_ID ;
    ExternalErrors  : AlertCountType := (others => 0)
) ;
. . .
ReportAlerts ;
```

ReportAlerts has 3 optional parameters: Name, AlertLogID, and ExternalErrors. Name overrides the name specified by SetAlertLogName. AlertLogID allows reporting alerts for a specific AlertLogID and its children (if any). ExternalErrors allows separately detected errors to be reported. ExternalErrors is type AlertCountType and the value (FAILURE => 0, ERROR => 5, WARNING => 1) indicates detection logic separate from AlertLogPkg saw 0 Failures, 5 Errors, and 1 Warning. See notes under AlertCountType.

```
--            Name,   AlertLogID,  ExternalErrors
ReportAlerts("Uart1", UartID,      (FAILURE => 0, ERROR => 5, WARNING => 1) ) ;
```

9.2.4 ReportNonZeroAlerts

When in hierarchy mode, a summary for each AlertLogID will be printed. This can include the following details. Each summary starts with "%%".

Within the hierarchy, if a level has no alerts set, then that level will not be printed.

```
procedure ReportNonZeroAlerts (
    Name            : string := OSVVM_STRING_INIT_PARM_DETECT ;
    AlertLogID      : AlertLogIDType := ALERTLOG_BASE_ID ;
    ExternalErrors  : AlertCountType := (others => 0)
) ;
```

9.2.5 Overloaded ReportAlerts for Reporting AlertCounts

ReportAlerts can also be used to print a passed/failed message for an AlertCount that is passed into the procedure call. This will not use any of the internal settings or information.

```
procedure ReportAlerts ( Name : String ; AlertCount : AlertCountType) ;
```

This is useful to accumulate values returned by different phases of a test that need to be reported separately.

```
ReportAlerts("Test1: Final", Phase1AlertCount + Phase2AlertCount) ;
```

9.3 GetAlertCount

GetAlertCount returns the AlertCount value at AlertLogID. GetAlertCount is overloaded to return either AlertCountType or integer.

```
impure function GetAlertCount(AlertLogID : AlertLogIDType := ALERTLOG_BASE_ID)
    return AlertCountType ;
impure function GetAlertCount(AlertLogID : AlertLogIDType := ALERTLOG_BASE_ID)
    return integer ;
. . .
TopTotalErrors  := GetAlertCount ;          -- AlertCount for Top of hierarchy
UartTotalErrors := GetAlertCount(UartID) ;  -- AlertCount for UartID
```

9.4 GetEnabledAlertCount

GetEnabledAlertCount is similar to GetAlertCount except it returns 0 for disabled alert levels. GetEnabledAlertCount is overloaded to return either AlertCountType or integer.

```
impure function GetEnabledAlertCount(AlertLogID : AlertLogIDType :=
    ALERTLOG_BASE_ID) return AlertCountType ;
impure function GetEnabledAlertCount (AlertLogID : AlertLogIDType :=
    ALERTLOG_BASE_ID) return integer ;
. . .
TopTotalErrors  := GetEnabledAlertCount ;           -- Top of hierarchy
UartTotalErrors := GetEnabledAlertCount(UartID) ;   -- UartID
```

9.5 GetDisabledAlertCount

GetDisabledAlertCount returns the count of disabled errors for either the entire design hierarchy or a particular AlertLogID. GetDisabledAlertCount is relevant since a "clean" passing design will not have any disabled alert counts.

```
impure function GetDisabledAlertCount return AlertCountType ;
impure function GetDisabledAlertCount return integer ;
impure function GetDisabledAlertCount(AlertLogID: AlertLogIDType)
    return AlertCountType ;
impure function GetDisabledAlertCount(AlertLogID: AlertLogIDType) return integer ;
```

Note that disabled errors are not added to higher levels in the hierarchy. Hence, often GetAlertCount /= GetEnabledAlertCount + GetDisabledAlertCount.

9.6 Math on AlertCountType

```
function "+" (L, R : AlertCountType) return AlertCountType ;
function "-" (L, R : AlertCountType) return AlertCountType ;
function "-" (R : AlertCountType) return AlertCountType ;
. . .
TotalAlertCount := Phase1Count + Phase2Count ;
TotalErrors := GetAlertCount - ExpectedErrors ;
NegateErrors := -ExpectedErrors ;
```

9.7 SumAlertCount: AlertCountType to Integer Error Count

SumAlertCount sums up the FAILURE, ERROR, and WARNING values into a single integer value.

```
impure function SumAlertCount(AlertCount: AlertCountType) return integer ;
. . .
ErrorCountInt := SumAlertCount(AlertCount) ;
```

10 Log Method Reference

10.1 LogType

Log levels can be ALWAYS, DEBUG, PASSED, or INFO.

```
type LogType is (ALWAYS, DEBUG, PASSED, INFO) ;
```

10.2 Log

If the log level is enabled, then the log message will print. The Enable parameter is an override of the internal settings and if true, the log message will print.

```
-- with an AlertLogID
procedure log(
  AlertLogID   : AlertLogIDType ;
  Message      : string ;
  Level        : LogType := ALWAYS ; -- Log if LogType is enabled
  Enable       : boolean := FALSE    -- also log if Enable is TRUE
) ;

-- without an AlertLogID
```

```
procedure log(
  Message   : string ;
  Level     : LogType := ALWAYS ; -- Log if LogType is enabled
  Enable    : boolean := FALSE    -- also log if Enable is TRUE
) ;
```

Usage:

```
Log(UartID, "Uart Parity Received", DEBUG) ;
. . .
Log("Received UART word", DEBUG) ;
```

10.3 SetLogEnable: Enable / Disable Logging

Excepting ALWAYS, log enables are disabled by default. SetLogEnable allows alert levels to be individually enabled. When used without AlertLogID, SetLogEnable sets a value for all AlertLogIDs.

```
procedure SetLogEnable(Level : LogType ;  Enable : boolean) ;
. . .
Log(UartID, "Uart Parity Received", DEBUG) ;
```

When an AlertLogID is used, SetLogEnable sets a value for that AlertLogID, and if Hierarchy is true, the AlertLogIDs of its children.

```
procedure SetLogEnable(AlertLogID : AlertLogIDType ;
    Level : LogType ;  Enable : boolean ; DescendHierarchy : boolean := TRUE) ;
. . .
--       AlertLogID,  Level,   Enable,  DescendHierarchy
SetLogEnable(UartID,  WARNING, FALSE,   FALSE) ;
```

10.4 Reading Log Enables from a FILE

ReadLogEnables read enables from a file.

```
procedure ReadLogEnables (FileName : string) ;
procedure ReadLogEnables (file AlertLogInitFile : text) ;
```

The preferred file format is:

```
U_CpuModel DEBUG
U_UART_TX DEBUG INFO
U_UART_RX PASSED INFO DEBUG

ReadLogEnables will also read a file of the format:

U_CpuModel
DEBUG
U_UART_TX
DEBUG
U_UART_TX
INFO
. . .
```

10.5 IsLogEnabled /GetLogEnable

IsLoggingEnabled returns true when logging is enabled for a particular AlertLogID.

```
impure function IsLogEnabled(Level : LogType) return boolean ;
impure function IsLogEnabled(AlertLogID : AlertLogIDType ; Level : LogType)
    return boolean ;
. . .
If IsLogEnabled(UartID, DEBUG) then
. . .
```

GetLogEnable is a synonym for IsLogEnabled.

```
impure function GetLogEnable(AlertLogID : AlertLogIDType ;  Level : LogType)
return boolean ;
impure function GetLogEnable(Level : LogType) return boolean ;
```

11 Alert and Log Prefix and Suffix to Message

A prefix and suffix can be added message that is printed with Alert or Log. These are set on an AlertLogID basis using the following operations.

```
  procedure SetAlertLogPrefix(AlertLogID : AlertLogIDType; Name : string ) ;
  procedure UnSetAlertLogPrefix(AlertLogID : AlertLogIDType) ;
  impure function GetAlertLogPrefix(AlertLogID : AlertLogIDType) return string ;
  procedure SetAlertLogSuffix(AlertLogID : AlertLogIDType; Name : string ) ;
  procedure UnSetAlertLogSuffix(AlertLogID : AlertLogIDType) ;
  impure function GetAlertLogSuffix(AlertLogID : AlertLogIDType) return string ;
```

12 Affirmation Reference

Affirmations are a combination of Alerts and Logs. If the Affirmation is true, then a log is generated. If an Affirmation is false, then an alert is generated.

Affirmations are intended to be used for self-checking of a test. Each call to AffirmIf is counted and reported during ReportAlerts. This provides feedback on the amount of self-checking added by a test and is used as a quality metric.

12.1 AffirmIf / AffirmIfNot

AffirmIF has two forms of overloading. The first has a separate ReceivedMessage and ExpectedMessage. When the affirmation passes, log is called with just the Received Message. When an affirmation fails, alert is called with the ExpectedMessage concatenated to the end of the ReceivedMessage.

```
-- with an AlertLogID
procedure AffirmIf(
  AlertLogID       : AlertLogIDType ;
  condition        : boolean ;
  ReceivedMessage  : string ;
  ExpectedMessage  : string ;
  Enable           : boolean := FALSE
) ;

-- without an AlertLogID
procedure AffirmIf(
  condition        : boolean ;
```

```
    ReceivedMessage  : string ;
    ExpectedMessage  : string ;
    Enable           : boolean := FALSE
  ) ;
```

The second overloading of AffirmIF has a single Message parameter. Hence, both alert and log print the same message.

```
  -- with an AlertLogID
  procedure AffirmIf(
    AlertLogID       : AlertLogIDType ;
    condition        : boolean ;
    Message          : string ;
    Enable           : boolean := FALSE
  ) ;

  -- without an AlertLogID
  procedure AffirmIf(
    condition        : boolean ;
    Message          : string ;
    Enable           : boolean := FALSE
  ) ;
```

There is also an AffirmIfNot for both of the forms above.

12.2 AffirmIfEqual

Affirmation form AffirmIfEqual checks if two values are equal. It greatly simplifies the message printing since it differentiates between the Received and Expected values. In the following, AType can be std_logic, std_logic_vector, unsigned, signed, integer, real, character, string or time.

```
  -- with an AlertLogID
  procedure AffirmIfEqual(
    AlertLogID          : AlertLogIDType ;
    Received, Expected  : AType ;
    Message             : string := "";
    Enable              : boolean := FALSE
  ) ;

  -- without an AlertLogID
  procedure AffirmIfEqual(
    Received, Expected  : AType ;
    Message             : string := "";
    Enable              : boolean := FALSE
  ) ;
```

12.3 AffirmIfDiff

Affirmation form AffirmIfDiff is for comparing two files.

```
  -- with an AlertLogID
  procedure AffirmIfDiff (
    AlertLogID     : AlertLogIDType ;
    Name1, Name2   : string;
    Message        : string := "" ;
```

```
  Level          : AlertType := ERROR
) ;
procedure AffirmIfDiff (
  AlertLogID         : AlertLogIDType ;
  file File1, File2  : text;
  Message            : string := "" ;
  Level              : AlertType := ERROR
) ;

-- without an AlertLogID
procedure AffirmIfDiff (
  Name1, Name2   : string;
  Message        : string := "" ;
  Level          : AlertType := ERROR
) ;
procedure AffirmIfDiff (
  file File1, File2  : text;
  Message            : string := "" ;
  Level              : AlertType := ERROR
) ;
```

12.4 GetAffirmCount

Returns the current affirmation check count.

```
impure function GetAffirmCount return natural ;
```

12.5 IncAffirmCount

Increments the affirmation check count. Intended to be used only in special situations, such as packages that have additional considerations when using affirmations.

```
procedure IncAffirmCount ;
```

13 Alert and Log Output Control Options

13.1 SetAlertLogJustify

SetAlertLogJustify justifies name fields of Alerts and Logs. Call after setting up the entire hierarchy if you want Alerts and Logs justified (hence optional).

```
SetAlertLogJustify(Enable : boolean := TRUE) ;
```

13.2 OsvvmOptionsType

OsvvmOptionsType defines the values for options. User values are: OPT_DEFAULT, DISABLED, FALSE, ENABLED, TRUE. The values DISABLED and FALSE are handled the same. The values ENABLED and TRUE are treated the same. The value OPT_USE_DEFAULT causes the variable to use its default value. OsvvmOptionsType is defined in OsvvmGlobalPkg.

```
type OsvvmOptionsType is (OPT_INIT_PARM_DETECT, OPT_USE_DEFAULT, DISABLED, FALSE,
ENABLED, TRUE) ;
```

13.3 SetAlertLogOptions: Configuring Report Options

The output from Alert, Log, and ReportAlerts is configurable using SetAlertLogOptions.

```
procedure SetAlertLogOptions (
  FailOnWarning         : OsvvmOptionsType := OPT_INIT_PARM_DETECT;
  FailOnDisabledErrors  : OsvvmOptionsType := OPT_INIT_PARM_DETECT;
  ReportHierarchy       : OsvvmOptionsType := OPT_INIT_PARM_DETECT;
  WriteAlertErrorCount  : AlertLogOptionsType := OPT_INIT_PARM_DETECT ; -- 2020.05
  WriteAlertLevel       : OsvvmOptionsType := OPT_INIT_PARM_DETECT;
  WriteAlertName        : OsvvmOptionsType := OPT_INIT_PARM_DETECT;
  WriteAlertTime        : OsvvmOptionsType := OPT_INIT_PARM_DETECT;
  WriteLogErrorCount    : AlertLogOptionsType := OPT_INIT_PARM_DETECT ; -- 2020.05
  WriteLogLevel         : OsvvmOptionsType := OPT_INIT_PARM_DETECT;
  WriteLogName          : OsvvmOptionsType := OPT_INIT_PARM_DETECT;
  PrintPassed           : AlertLogOptionsType := OPT_INIT_PARM_DETECT ; -- 2020.05
  PrintAffirmations     : AlertLogOptionsType := OPT_INIT_PARM_DETECT ; -- 2020.05
  PrintDisabledAlerts   : AlertLogOptionsType := OPT_INIT_PARM_DETECT ; -- 2020.05
  WriteLogTime          : OsvvmOptionsType := OPT_INIT_PARM_DETECT;
  AlertPrefix           : string := OSVVM_STRING_INIT_PARM_DETECT;
  LogPrefix             : string := OSVVM_STRING_INIT_PARM_DETECT;
  ReportPrefix          : string := OSVVM_STRING_INIT_PARM_DETECT;
  DoneName              : string := OSVVM_STRING_INIT_PARM_DETECT;
  PassName              : string := OSVVM_STRING_INIT_PARM_DETECT;
  FailName              : string := OSVVM_STRING_INIT_PARM_DETECT
) ;
```

The following options are for ReportAlerts.

PrintPassed	If PrintPassed is enabled, print number of passed logs in the report summary and for each AlertLogID	Enabled
PrintAffirmations	If PrintAffirmations is enabled, print the number of affirmations in the report summary and for each AlertLogID	Disabled
FailOnWarning	Count warnings as test errors.	Enabled
FailOnDisabledErrors	If FailOnDisabledErrors is enabled and DisabledAlerts /= 0: Test Fails DisabledAlerts are printed in the report summary. DisabledAlerts print for each AlertLogID	Enabled
PrintDisabledAlerts	If PrintDisabledAlerts is Enabled, print DisabledAlerts in the report summary and for each AlertLogID	Disabled
ReportHierarchy	When multiple AlertLogIDs exist, print an error summary for each level.	Enabled
ReportPrefix	Prefix for each line of ReportAlerts.	"%% "
DoneName	Name printed after ReportPrefix on first line of ReportAlerts.	"DONE"
PassName	Name printed when a test passes.	"PASSED".
FailName	Name printed when a test fails.	"FAILED"

The following options are for alert:

WriteAlertErrorCount	Print Error Count immediately after %% Alert	Disabled
WriteAlertLevel	Print level.	Enabled
WriteAlertName	Print AlertLogID name.	Enabled
WriteAlertTime	Alerts print time.	Enabled
AlertPrefix	Name printed at beginning of alert.	"%% Alert"

The following options are for Log:

WriteLogErrorCount	Print Error Count immediately after %% Log	Disabled
WriteLogLevel	Print level.	Enabled
WriteLogName	Print AlertLogID name.	Enabled
WriteLogTime	Logs print time.	Enabled
LogPrefix	Name printed at beginning of log.	"%% Log"

SetAlertOptions will change as AlertLogPkg evolves. Use of named association is required to ensure future compatibility.

```
SetAlertLogOptions (
    FailOnWarning           => FALSE,
    FailOnDisabledErrors    => FALSE
) ;
```

After setting a value, a string value can be reset using OSVVM_STRING_USE_DEFAULT and an OsvvmOptionsType value can be reset using OPT_USE_DEFAULT.

13.4 ReportAlertLogOptions: Print Report Options

Prints out AlertLogPkg Report Options.

13.5 Getting AlertLog Report Options

Report options can be retrieved with one of the Get functions below.

```
function GetAlertLogFailOnWarning        return AlertLogOptionsType ;
function GetAlertLogFailOnDisabledErrors return AlertLogOptionsType ;
function GetAlertLogReportHierarchy      return AlertLogOptionsType ;
function GetAlertLogHierarchyInUse       return AlertLogOptionsType ;
function GetAlertLogWriteAlertLevel      return AlertLogOptionsType ;
function GetAlertLogWriteAlertName       return AlertLogOptionsType ;
function GetAlertLogWriteAlertTime       return AlertLogOptionsType ;
function GetAlertLogWriteLogLevel        return AlertLogOptionsType ;
function GetAlertLogWriteLogName         return AlertLogOptionsType ;
function GetAlertLogWriteLogTime         return AlertLogOptionsType ;
function GetAlertLogAlertPrefix          return string ;
function GetAlertLogLogPrefix            return string ;
function GetAlertLogReportPrefix         return string ;
function GetAlertLogDoneName             return string ;
function GetAlertLogPassName             return string ;
function GetAlertLogFailName             return string ;
```

14 Deallocating and Re-initializing the Data structure

14.1 DeallocateAlertLogStruct

DeallocateAlertLogStruct deallocates all temporary storage allocated by AlertLogPkg. Also see ClearAlerts.

14.2 InitializeAlertLogStruct

InitializeAlertLogStruct is used after DeallocateAlertLogStruct to create and initialize internal storage.

15 Compiling and Referencing AlertLogPkg 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 ;
```

16 Release Notes for 2020.05

Major update to AlertLogPkg. Output formatting of ReportAlerts has changed.

Added count of Log PASSED for each level. Prints by default. Disable by using SetAlertLogOptions (PrintPassed => DISABLED).

Added total count of log PASSED for all levels. Prints in "%% DONE" line by default. Disable as per each level. However, always prints if passed count or affirmation count is > 0.

Added affirmation counts for each level. Prints by default. Disable by using SetAlertLogOptions (PrintAffirmations => DISABLED).

Total count of affirmations is disabled using SetAlertLogOptions above. However, it always prints if affirmation count > 0.

Disabled alerts are now tracked with a separate DisabledAlertCount for each level. These do not print by default. Enable printing for these by using SetAlertLogOptions (PrintDisabledAlerts => ENABLED).

A total of the DisabledAlertCounts is tracked. Prints in "%% DONE" anytime PrintDisabledAlerts is enabled or FailOnDisabledErrors is enabled. FailOnDisabledErrors is enabled by default. Disable with SetAlertLogOptions (FailOnDisabledErrors => DISABLED).

Internal to the protected type of AlertLogPkg is AffirmCount which tracks the total of FAILURE, ERROR, and WARNING and ErrorCount which tracks a single integer value for all errors. Many simulators give you the ability to trace these in your waveform windows.

Added printing of current ErrorCount in the alert and log messages. When enabled, the number immediately follows the "%% Alert" or "%% Log". Enable using SetAlertLogOptions (WriteAlertErrorCount=> ENABLED) and SetAlertLogOptions (WriteLogErrorCount=> ENABLED).

Added direct access to SetAlertLogJustify(Enable := TRUE) which makes all alert and log printing justified.

Added pragmas "synthesis translate_off" and "synthesis translate_on" in a first attempt to make the package ok for synthesis tools. It has not been tested, so you will need to try it out and report back.

Added a prefix and suffix for Alerts and Logs. They support set, unset, and get operations using SetAlertLogPrefix(ID, "String"), UnSetAlertLogPrefix(ID), GetAlertLogPrefix(ID), SetAlertLogSuffix(ID, "String"), UnSetAlertLogSuffix (ID), and GetAlertLogSuffix (ID).

Added ClearAlertStopCounts and ClearAlertCounts. ClearAlerts clears AlertCount, DisabledAlertCount, AffirmCount, PassedCount for each level as well as ErrorCount, AlertCount, PassedCount, and AffirmCount for the top level. ClearAlertStopCounts resets the stop counts back to their default values. ClearAlertCounts calls both ClearAlerts and CLearAlertStopCounts.

17 About AlertLogPkg

AlertLogPkg was developed and is maintained by Jim Lewis of SynthWorks VHDL Training. It originated as an interface layer to the BitVis Utility Library (BVUL). However, it required a default implementation and that default implementation grew into its own project.

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

AlertLogPkg is released under the Apache open source license. It is free (both to download and use - there are no license fees). You can download it from https://github.com/OSVVM/OSVVM. It will be updated from time to time. Currently there are numerous planned revisions.

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.

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

18 Future Work

AlertLogPkg.vhd 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.

19 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.