TbUtilPkg Package User Guide

User Guide for Release 2024.11

By

Jim Lewis

SynthWorks VHDL Training

Jim@SynthWorks.com

http://www.SynthWorks.com

Table of Contents

1 TbUtilPkg Overview

TbUtilPkg provides testbench utilities including synchronization utilities, tests for mutual exclusion, clock creation and reset creation.

2 Testing for Mutual Exclusion

2.1 OneHot

Function OneHot returns true when exactly one of its input is a one.

```
function OneHot ( constant A : in std_logic_vector ) return boolean ;
. . .
AffirmIf(OneHot( (Sel1 & Sel2 & Sel3 & Sel4), . . .) ;
. . .
```

2.2 ZeroOneHot

Function ZeroOneHot returns true when either exactly one of its input is a one or they are all zero.

```
function ZerOneHot ( constant A : in std_logic_vector ) return boolean ;
. . .
AffirmIf(ZeroOneHot( (Sel1 & Sel2 & Sel3 & Sel4), . . .) ;
. . .
```

3 Verification Component Handshaking for Transaction Initiation

3.1 RdyType and AckType

The primary types for transaction handshaking are:

```
  subtype RdyType is resolved_max integer range  0 to integer'high ;
  subtype AckType is resolved_max integer range -1 to integer'high ;
```

3.2 RequestTransaction

Requests a transaction from the test initiation (client) to the model (TLM or VVC).

```
procedure RequestTransaction (
    signal Rdy  : Out RdyType ;
    signal Ack  : In  AckType
) ;
. . .
procedure DoTransaction( TransRec : inout TransRecType ; Parm1 : . . . ) is
begin
  TransRec.InField1 <= Parm1 ;
  . . .
  RequestTransaction(Rdy => TransRec.Rdy, Ack => TransRec.Ack) ;
  Result1 <= TransRec.OutField1 ;
  . . .
```

3.3 WaitForTransaction

Suspends a model until a transaction is requested via RequestTransaction.

```
      procedure WaitForTransaction (
        signal Clk  : In  std_logic ;
        signal Rdy  : In  RdyType;
        signal Ack  : Out AckType
      ) ;
    . . .
    entity Model is
      Port (
        -- Transaction connection
        TransRec : inout TransRecType ;

        -- DUT signaling interface
        . . .
      ) ;
    end entity model ;
    architecture behavior of Model is
    begin
      ModelBehavior : process
      begin
        WaitForTransaction (
           Clk  => ModelClk,
           Rdy  => TransRec.Rdy,
           Ack  => TransRec.Ack
        ) ;
        case ModelRec.Operation is
          when CPU_WRITE =>
             -- do CPU Write signaling
             . . .
          when CPU_READ =>
             -- do CPU Read signaling
             . . .
          . . .
          when others =>
            Alert(ModelAlertLogID, "Unrecognized operation", FAILURE) ;
            Wait for 0 ns ;
        end case ;
      end process ModelBehavior ;
```

3.4 Overloaded versions of RequestTransaction and WaitForTransaction

Overloading of RequestTransaction and WaitForTransaction also supports std_logic or bit in place of both RdyType and AckType.

4 Toggle Handshaking

Toggle type handshaking is used to block one process until another process has signaled it to continue.

4.1 WaitForToggle

WaitForToggle block until its input signal changes. The call to WaitForToggle must occur before input changes or the change will not be seen.

```
procedure WaitForToggle ( signal Sig : In integer ) ;
procedure WaitForToggle ( signal Sig : In bit ) ;
procedure WaitForToggle ( signal Sig : In std_logic ) ;
```

Internally WaitForToggle abstracts a call to "wait on".

4.2 Toggle

Toggle causes a signal to change either after a delta cycle or a specified amount of time.

```
procedure Toggle(signal Sig : InOut std_logic ; constant DelayVal : delay_length);
procedure Toggle(signal Sig : InOut std_logic ) ;
procedure Toggle(signal Sig : InOut bit ; constant DelayVal : delay_length) ;
procedure Toggle(signal Sig : InOut bit ) ;
```

Internally Toggle abstracts an assignment to Sig.

4.3 Increment

Increment adds 1 to the integer signal.

```
procedure Increment (
  signal Sig             : InOut integer ;
  constant RollOverValue : in integer := 0) ;
```

4.4 Usage of Increment and WaitForToggle

Using FIFOs to communicate requires an event. OSVVM FIFOs do not generate events. Instead, Increment and WaitForToggle are used. This is shown below.

```
TransactionDispatcher : process
Begin
   . . .
   Push(FIFO, TransRec.Data) ;
   Increment(RequestCount) ;
   . . .

DataHandler : process
```

```
    variable Data : std_logic_vector(15 downto 0) ;
  begin
    if IsEmpty(FIFO) then
      WaitForToggle(RequestCount) ;
    end if ;
    Data := Pop(FIFO) ;
    . . .
```

5 Barrier Synchronization

Barrier synchronization allows two or more processes to stop until all have arrived at a designated synchronization point.

5.1 WaitForBarrier

WaitForBarrier causes a process to stop until all processes that call WaitForBarrier with that signal have reached the same point.

```
procedure WaitForBarrier (signal Sig : InOut BarrierType) ;
procedure WaitForBarrier (
  signal   Sig     : InOut BarrierType;
  constant TimeOut : delay_length) ;
procedure WaitForBarrier (signal Sig : InOut std_logic ) ;
procedure WaitForBarrier (
  signal   Sig     : InOut std_logic;
  constant TimeOut : delay_length);
```

5.2 Types and resolution functions

The type BarrierType is designed to simplify barrier synchronization.

```
function resolved_barrier ( s : integer_vector ) return integer ;
subtype  BarrierType is resolved_barrier integer range
    integer'low+1 to integer'high ;
```

The range on BarrierType is only needed to work around bugs in Xilinx XSim.

5.3 Declarations

Usage of integer barriers requires usage of resolved_barrier. Usage of BarrierType is recommended.

```
signal TestDone : BarrierType ;                    -- Preferred
signal Sync1    : resolved_barrier integer := 1 ;  -- Supported
```

With std_logic, initialization to '0' is preferred, but not required. Do not initialize to 'H'.

```
signal Sync2 : std_logic := '0' ; -- Initialization preferred
```

5.4 Usage

```
ControlProc : process
begin
```

```
    -- initialize test
    SetTestName("Uart1_Rx") ;
    . . .
    WaitForBarrier(TestDone, 5 ms) ; -- control process uses timeout
    AlertIf(now >= 5 ms, "Test finished due to Time Out") ;
    ReportAlerts ;
    Std.env.stop ;
  End process ControlProc ;

  CpuProc : process
  begin
     InitDut(. . . )_;
     Toggle(CpuReady) ;
     -- run numerous Cpu test transactions
     . . .
     WaitForBarrier(TestDone) ;
     wait ;
  end process CpuProc ;

  UartTxProc : process
  Begin
    WaitForToggle(CpuReady) ;
     -- run numerous Uart Transmit test transactions
     . . .
     WaitForBarrier(TestDone) ;
     wait ;
  end process UartTxProc ;
  . . .
```

5.5 Predefined OSVVM Barriers

TbUtilPkg provides the following predefined barrier signals.

```
type PredefinedBarrierType is record
  ResetStarted   : BarrierType ;
  ResetDone      : BarrierType ;
  TestInit       : BarrierType ;
  TestDone       : BarrierType ;
  VcInit         : BarrierType ;
end record PredefinedBarrierType ;
signal Barrier : PredefinedBarrierType ;

alias OsvvmResetStarted  : BarrierType is Barrier.ResetStarted ;
alias OsvvmResetDone     : BarrierType is Barrier.ResetDone ;
alias OsvvmTestInit      : BarrierType is Barrier.TestInit ;
alias OsvvmTestDone      : BarrierType is Barrier.TestDone ;
alias TestDone           : BarrierType is Barrier.TestDone ;
alias OsvvmVcInit        : BarrierType is Barrier.VcInit ;
```

6 Finding Clock Changing

EdgeRose checks if the signal rose during this time step. EdgeFell checks if the signal fell during this time step. EdgeActive checks if the signal changed to the value specified by A in this time step.

```
function  EdgeRose ( signal C : in std_logic ) return boolean ;
function  EdgeFell ( signal C : in std_logic ) return boolean ;
function  EdgeActive ( signal C : in std_logic; A : std_logic ) return boolean ;
function  ChangingEdge ( signal C : in std_logic; A : std_logic ) return boolean ;
alias changing_edge is ChangingEdge [std_logic, std_logic return boolean] ;
```

EdgeRose is similar to rising_edge except EdgeRose checks if clock changed on a time step (by checking C'last_event=0 sec), where as, rising_edge checks if a clock changed during this delta cycle (by checking C'event). EdgeRose does the following check.

```
to_x01(C)='1' and to_x01(C'last_value)='0' and C'last_event= 0 sec
```

7 Waiting for Clock

Wait for the number of clocks specified in either time or an integer number of clock cycles. Active edge of clock is defined by the ClkActive parameter (which is set to '1').

```
procedure WaitForClock (
  signal Clk              : in std_logic ;
  constant Delay          : in delay_length ;
  constant ClkActive      : in std_logic := CLK_ACTIVE) ;
procedure WaitForClock (
  signal Clk              : in std_logic ;
  constant NumberOfClocks : in natural := 1 ;
  constant ClkActive      : in std_logic := CLK_ACTIVE) ;
procedure WaitForClock (
  signal Clk            : in std_logic ;
  signal Enable         : in boolean ;
  constant ClkActive    : in std_logic := CLK_ACTIVE) ;
procedure WaitForClock (
  signal Clk            : in std_logic ;
  signal Enable         : in std_logic ;
  constant EnableActive : std_logic := '1' ;
  constant ClkActive    : in std_logic := CLK_ACTIVE) ;
```

8 Waiting For Level

Wait for a signal to become a level.

```
procedure WaitForLevel ( signal A : boolean ) ;
procedure WaitForLevel ( signal A : std_logic ; Level : std_logic := '1' ) ;

procedure WaitForLevel (
  signal   A       : in boolean;
  constant TimeOut : delay_length);
procedure WaitForLevel (
  signal   A       : in std_logic;
  constant TimeOut : delay_length;
  constant Level   : std_logic := '1');

procedure WaitForLevel (
  signal   A              : in boolean;
  constant TimeOut        : delay_length;
  variable TimeOutReached : out boolean);
procedure WaitForLevel (
  signal   A              : in std_logic;
  constant TimeOut        : delay_length;
  variable TimeOutReached : out boolean;
  constant Level          : std_logic := '1');
```

9 CreateClock and CreateReset Moved to ClockResetPkg

CreateClock and CreateReset were moved to ClockResetPkg.

10 About TbUtilPkg

TbUtilPkg is part of the OSVVM Utility library.

TbUtilPkg.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.

11 Compiling and Using OSVVM Utility Library

Reference all packages in the OSVVM Utility library by using the context declaration:

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

Compilation order for OSVVM Utility Library is in OSVVM_release_notes.pdf. Rather than learning this, we recommend using the OSVVM compilation scripts.

OSVVM Utility library 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 osvvm.org or from our development area on GitHub (https://github.com/OSVVM/OSVVM).

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.

We also support the OSVVM user community and blogs through http://www.osvvm.org. Interested in sharing about your experience using OSVVM? Let us know, you can blog about it at osvvm.org.

12 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.