TbUtilPkg Package User Guide

User Guide for Release 2016.11

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 Transaction Initiation

3.1 RequestTransaction

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

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

3.2 WaitForTransaction

Suspends a model until a transaction is requested via RequestTransaction.

```
  procedure WaitForTransaction (
    signal Clk  : In  std_logic ;
    signal Rdy  : In  std_logic ;
    signal Ack  : Out std_logic
```

```
  ) ;
. . .
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 ;
```

Note that time must pass between consecutive calls to WaitForTransaction. Generally all transactions consume at least a a cycle of ModelCycle. However, for directive type actions which are used only for testbench to model exchange of information (such ase setup or error counts) at least a delta cycle (via a "wait for 0 ns;") must pass.

4 Toggle Handshaking

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

4.1 WaitForToggle

WaitForToggle causes a process to block until another process signals it to wake up. The call to WaitForToggle must occur before input changes or the change will not be seen.

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

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

4.3 Usage of WaitForToggle and Toggle

In the example below, Proc2 suspends until Proc1 has executed the procedure InitializeInterface and called Toggle on the signal InterfaceReady. The "wait until rising_edge(Clk);" ensures that each process starts on the same delta cycle.

```
Proc1 : process
begin
   InitializeInterface(Init1, Init2, . . .) ;
   Toggle(InterfaceReady) ;
   wait until rising_edge(Clk) ;
   . . .

Proc2 : process
begin
  WaitForToggle(InterfaceReady) ;
  wait until rising_edge(Clk) ;
  . . .
```

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 integer ) ;
procedure WaitForBarrier (signal Sig : InOut integer; constant TimeOut : time ) ;
procedure WaitForBarrier (signal Sig : InOut std_logic ) ;
procedure WaitForBarrier (signal Sig : InOut std_logic; constant TimeOut : time );
```

5.2 Types, Resolution Functions, and initializations

Barrier synchronization requires the usage of resolution functions and either std_logic or integer_barrier (defined in TbUtilPkg as shown below).

```
function resolved_barrier ( s : integer_vector ) return integer ;
subtype  integer_barrier is resolved_barrier integer ;
```

Going further, it is recommended that std_logic signals are initialized to '0' and integer_barrier signals are initialized to 1.

```
signal TestDone : integer_barrier := 1 ;
signal TestDone_sl : std_logic := '0' ;
```

When an integer_barrier signal is initialized to 1, its value will indicate the number of processes that have not reached the barrier synchronization point. When a std_logic signal is initialized to '0', it will have the value '0' while the processes are waiting and it will transition to 'H' for the duration of the delta cycle in which all processes have reached the barrier synchronization point.

5.3 Usage

```
ControlProc : process
begin
  -- initialize test
  SetAlertLogName("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 ;
. . .
```

6 Waiting for Clock

Wait for a number of clocks specified in either time or an integer number of clock cycles. Currently the constant CLK_ACTIVE defines the active edge of clock and the STANDARD version sets it to '1' (rising edge).

```
procedure WaitForClock ( signal Clk : in std_logic ;  constant Delay : in time ) ;
procedure WaitForClock ( signal Clk : in std_logic ;
```

```
                        constant NumberOfClocks : in integer := 1) ;
```

7 Waiting For Level

Wait for a signal to become a level.

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

8 Creating a Clock

Create a clock with a specified period and specified duty cycle. Designed so that after the first clock transition that it always transitions on delta cycle 0. Designed so that the resulting clock period will match the Period parameter and that any rounding due to DutyCycle and simulator resolution will only impact the effective DutyCycle.

```
procedure CreateClock (
    signal   Clk        : inout std_logic ;
    constant Period     : time ;
    constant DutyCycle  : real := 0.5
  )  ;
```

9 Creating Reset

Create reset that is asserted tpd after the first active edge of clock and deasserts tpd after the clock edge that is Period in time from the first active edge of clock. Currently the constant CLK_ACTIVE defines the active edge of clock and the STANDARD version sets it to '1' (rising edge).

```
procedure CreateReset (
  signal   Reset       : out std_logic ;
  constant ResetActive : in  std_logic ;
  signal   Clk         : in  std_logic ;
  constant Period      :     time ;
  constant tpd         :     time
) is   . . .
```

10 Compiling TbUtilPkg 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 ;
```

11 About TbUtilPkg

TbUtilPkg was developed and is maintained by Jim Lewis of SynthWorks VHDL Training. Prior to its release to OSVVM it was used in SynthWorks' VHDL classes.

Please support our effort in supporting the OSVVM library of packages by purchasing your VHDL training from SynthWorks.

TbUtilPkg is released under the Perl Artistic 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.

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.

12 Future Work

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.

13 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.