SynthWorks

VHDL Training Experts

ResolutionPkg Quick Reference

1. ZeroOneHot, OneHot

```
signal SA, SB, SC, SD : std_logic;
signal ZOH, OH : Boolean ;
```

ZeroOneHot returns true when the std_logic_vector input either has a single one or is all zero.

```
ZOH <= ZeroOneHot((SA & SB & SC & SD)) ;
```

OneHot returns true when the std_logic_vector input either has a single one.

```
OH <= OneHot((SA & SB & SC & SD)) ;
```

2. Transaction Handshaking

Used for handshaking between the client side of a transaction interface and the model.

2.1 RequestTransaction

Request a transaction from the client side (TestCtrl).

```
procedure DoTransaction(
  ModelRec : inout ModelRecType ;
  DataIn     : in DataType
)_ is
begin
  ModelRec.Data <= DataIn ;
  RequestTransaction(
    Req => ModelRec.CmdReq,
    Ack => ModelRec.CmdAck ) ;
end procedure DoTransaction ;
```

2.2 WaitForTransaction

Model side control to wait for client side (TestCtrl) to request a transaction

```
ExecuteTransProc : process
begin
  WaitForTransaction(
    Clk => Clk,
    Req => ModelRec.CmdReq,
    Ack => ModelRec.CmdAck
  ) ;
  -- decode and execute transaction
```

2.3 WaitForTransaction with Timeout

Model side control to wait for client side (TestCtrl) to request a transaction with Timeout. The timeout is to handle applications where the flow needs to be disrupted by an alternate stream of transactions (such as an interrupt handler).

```
ExecuteTransProc : process
begin
  WaitForTransaction(
    Clk => Clk,
    Req => ModelRec.CmdReq,
    Ack => ModelRec.CmdAck,
    TimeOut  => InterruptReq,
    Polarity => '1'
  ) ;
```

2.4 Finish Transaction Handshaking

Used when handshaking with multiple streams of transactions.

```
Finish(Ack => ModelRec.CmdAck) ;
```

2.5 TransactionPending

Returns true when a steam of transaction has a transaction ready.

```
If TransactionPending(Rdy =>
ModelRec.CmdRdy) then
  . . .
```

3. Process to Process Synchronization

One process toggles a signal using the toggle procedure, the other process waits until the signal changes.

3.1 Toggle

Toggle a signal between 0 and 1. Signal type dcan be either bit or std_ulogic.

```
Toggle(Sync1) ;
Toggle(Sync2, 2*tperiod_Clk);
```

3.2 WaitForToggle

Wait until a signal changes. Signal type can be either bit or std_ulogic.

```
WaitForToggle(Sync1) ;
```

4. Barier Synchronization

All processes stop until all processes have reached the barrier and called WaitForBarrier.

```
WaitForBarrier(Sync1) ;
WaitForBarrier(Sync2, 25 ms) ;
```

Type of Sync1 and Sync2 may be either std_logic or integer_barrier.

```
signal sync1 : std_logic := '0' ;
Signal sync2 : integer_barrier := 1 ;
```

5. Waiting for Clock

Wait for clock periods specified in either time units or an integral number of clock cycles. Is aligned to clock when it finishes.

```
WaitForClock(Clk, 5 * Tperiod_Clk);
WaitForClock(Clk, 5) ;
```

6. Wait for Level

Wait until a signal is at a level.

```
WaitForLevel(A, '1') ; -- A='1'
WaitForLevel(Bool) ;   -- TRUE
```

7. Create Clock

Create clock with designated period and duty cycle.

```
CreateClock(
  Clk       => Clk,
  Period    => 10 ns,  -- 100 MHz
  DutyCycle => 0.5     -- Default
) ;
```

8. Create Reset

Create clock with designated period and duty cycle.

```
  CreateReset (
    Reset       => nReset,
    ResetActive => '0', -- active low
    Clk         => Clk,
    Period      => 5 * Tperiod_Clk,
    tpd         => 2 ns
  ) ;
```

9. Clock Polarity

Clock polarity is controlled by the constant CLK_ACTIVE. This will be changed to a generic in a future revision.

```
constant CLK_ACTIVE : std_logic := '1' ;
```

9.1 CheckFinish

For test completion when using alerts.

```
SB.CheckFinish(
        FinishCheckCount => 1,
        FinishEmpty => TRUE ) ;
```

If CheckCount < FinishCheckCount then signal an alert and increment the internal error count. If FinishEmpty is TRUE and Empty is FALSE then signal an alert and increment the internal error count.

9.2 GetErrorCount

Only intended for non-alert flows. If not using separate AlertLogIDs and ReportAlerts, GetErrorCount returns the current error count.

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

9.3 GetItemCount

Get number of items put into the scoreboard.

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

9.4 GetCheckCount

Get number of items checked by the scoreboard.

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

9.5 GetDropCount

Get number of items dropped by the scoreboard.

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

9.6 SetName

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

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

9.7 GetName

Get the scoreboard name

```
print("…" & SB.GetName) ;
```


SynthWorks Design Inc.
VHDL Design and Verification Training
11898 SW 128th Ave. Tigard OR 97223 (800)-505-8435
http://www.SynthWorks.com jim@synthworks.com

SynthWorks

VHDL Training Experts

10. 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.

11. 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.

11.1 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.

11.2 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 ;
```

11.3 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);

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

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

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

-- Check at least 100 items and
-- Finish Empty
SB.CheckFinish(1, 100, TRUE) ;
SB.CheckFinish(2, 100, TRUE) ;
SB.CheckFinish(3, 100, TRUE) ;

-- test completion via AlertLogPkg
ReportAlerts ;

-- Getting Error Counts (non-Alert)
TotalErrorCount :=
  SB.GetErrorCount(1) +
  SB.GetErrorCount(2) +
  SB.GetErrorCount(3);

TotalErrorCountAlt := SB.GetErrorCount ;
```

11.4 Arrays of Simple Scoreboards

The following are operations appropriate for arrays of simple scoreboards. In all examples, 4 is the index, ExpectedVal has the type ExpectedType, and ReceiveVal has the type ActualType.

```
SB.Push(4, ExpectedVal) ;
SB.Check(4, ReceiveVal) ;
SB.Pop(4, ExpectedVal) ;

if SB.Empty(4)  then   …

ItemNum := SB.Find(4, ReceiveVal);
SB.Flush(4, ItemNum) ;
```

11.5 Arrays of Tagged Scoreboards

The following are operations appropriate for arrays of tagged scoreboards. In all examples, 4 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(4, "WriteOp", ExpectedVal) ;
SB.Check(4, "WriteOp", ReceiveVal) ;
SB.Pop(4, "WriteOp", ExpectedVal) ;

if SB.Empty(4, "MyTag")  then   …  -- **
if SB.Empty(4)  then   …  -- **

ItemNum := SB.Find(4, "Red", ReceiveVal);
-- two possible alternatives
SB.Flush(4, "Red", ItemNum) ; -- **
SB.Flush(4, ItemNum) ; -- **
```


SynthWorks Design Inc.
VHDL Design and Verification Training
11898 SW 128th Ave. Tigard OR 97223 (800)-505-8435
http://www.SynthWorks.com jim@synthworks.com