What's Next in VHDL

By
Jim Lewis, SynthWorks VHDL Training

What's Next in VHDL

- **VHDL = Verification & Hardware Description Language**
- Verification focused
  - OO/Classes
  - Verification Data Structures
  - Randomization
  - Functional Coverage
  - Incorporate the good things from SystemVerilog, SystemC, IEEE 1850/E, Vera
- RTL: items to do, but verification is higher priority

Caution: Changes presented here are a work in progress

OO / Classes

- Classes are the foundation for both data structures and randomization of transactions
- Status: Have proposal from Peter Ashenden
- Proposal extends protected types into classes
  - Similar to other programming languages (particularly Java).
  - Adds shared variable ports
- Since classes extend protected types, can do some prototyping with the current language.

OO / Classes


```
type BoundedFIFO is protected class
  procedure put ( e : in element_type );    -- methods
  procedure get ( e : out element_type );
end protected class BoundedFIFO;
```

```
type BoundedFIFO is protected class body
  constant size : positive := 20;
  type element_array is array (0 to size-1) of element_type;

  variable elements : element_array;
  variable head, tail : natural range 0 to size-1 := 0;
  variable count : natural range 0 to size := 0;

  procedure put ( e : in element_type ) is begin
    if count = size then wait until count < size; end if;
    elements(head) := e;
    head := (head + 1) mod size;  count := count + 1;
  end procedure put;

  procedure get ( e : out element_type ) is begin   . . .

end protected class body BoundedFIFO;
```

Verification Data Structures

- Basic Requirements
  - Linked-Lists
  - FIFOs
  - Mailboxes (Put, Get)
  - Transaction Interfaces (Put, Get)
  - Scoreboards (PutValue, CheckValue, ErrCount)
  - Memories (MemInit, MemRead, MemWrite)

- The current plan is to make these class based.

A 1 Item MailBox

- An interface specifies contracts on a class (like in Java):

```
type putable is interface
  procedure put ( e : in element_type );
  procedure try_put (e : in element_type;  ok : out boolean);
end protected interface putable;

type getable is interface    . . .   -- see proposal
```

- Mailbox class implements putable and getable:

```
type mailboxPCType is protected class implements putable,
getable
  function flag up return boolean;
  procedure put ( e : in element_type );
  procedure try_put (e : in element_type;  ok : out boolean);
  procedure get ( e : out element_type );
  procedure try_get (e : out element_type; ok : out boolean);
end protected class mailbox;
```

A 1 Item MailBox

SynthWorks

- Producer port is "putable".
- Consumer port is "getable"
- Any type that implements these can be used

```
entity tlm is
end tlm ;

architecture structural of tlm is
  component producer is
    port ( shared variable data_source : inout putable );
  end component producer;

  component consumer is
    port ( shared variable data_sink : inout getable );
  end component consumer;

  shared variable MailBox : mailboxPCType;

begin

  u_producer : producer port map ( data_source => MailBox );
  u_consumer : consumer port map ( data_sink   => MailBox );

end architecture tlm;
```


Randomization

SynthWorks

- Useful when testing numerous configurable features.
  - Testing in an isolated features is straightforward
  - Testing interactions is a large verification space
    - difficult to simulate completely
    - difficult to predict corner cases
    - Randomization can reasonably coverage this space
- Use of coverage is important to identify which design features have been tested.
- Status: Have proposal from Jim Lewis
- Current proposal is based on SystemVerilog

Randomization

- Supporting two forms of randomization:
  - Class based
  - Procedural
- Class based randomization
  - Group values of a transaction together
  - Specify relationships/constraints between them
  - Randomize as a single item
- Procedural Randomization
  - Randomizing single values
  - CaseRand
  - Sequence

Class Based Randomization

- A class with constraints:

```
Type TxPacketCType is class

  Rand Variable BurstLen   : integer ;   -- Public Variables
  Rand Variable BurstDelay : integer ;

  Constraint BurstPkt is (
    BurstLen in (1 to 10) ;
    BurstDelay in (1 to 6) when BurstLen <= 3 else
    BurstDelay in (3 to 10) ;
  ) ;

End class TxPacketCType ;
```

Class Based Randomization


```
TxProc : process
  variable TxPacket : TxPacketCTType ;
  variable RV : RandomClass ;
begin
  . . .
  TxOuterLoop: loop

    TxPacket.randomize ;

    for i in 1 to TxPacket.BurstLen loop
      DataSent := RV.RandSlv(0, 255, DataSent'length);
      Scoreboard.PutExpectedData(DataSent) ;
      WriteToFifo(DataSent) ;
    end loop ;

    wait for TxPacket.BurstDelay * tperiod_Clk - tpd ;
    wait until Clk = '1' ;

  end loop TxOuterLoop ;
. . .
end process TxProc ;
```

Procedural Randomization

- Randomization within code using individual randomization calls, RandCase, or Sequence construct

```
I0 := 1; I1 := 1; I2 := 1;
for i in 1 to 3 loop

  RandCase is
    with I0 =>
      CpuWrite(CpuRec, DMA_WORD_COUNT, DmaWcIn);
      I0 := 0 ; -- modify weight

    with I1 =>
      CpuWrite(CpuRec, DMA_ADDR_HI, DmaAddrHiIn);
      I1 := 0 ; -- modify weight

    with I2 =>
      CpuWrite(CpuRec, DMA_ADDR_LO, DmaAddrLoIn);
      I2 := 0 ; -- modify weight

  end case ;

end loop ;
CpuWrite(CpuRec, DMA_CTRL, START_DMA or DmaCycle);
```

Functional Coverage

- Functional coverage supplements other forms of coverage
- Tool based or structural coverage can tell you:
  - there was a FIFO read
  - the FIFO was empty
  - however, it has no way to tell you both happened.
- Assertions via PSL can tell you this
- Functional coverage constructs provide capability to:
  - bin values of an object into separate categories
  - correlate cross coverage between items
- Status: proposal is a work in progress
  - As always, all input welcome.

Summary

- Both Standards and Vendors are heavily influenced by users
  - Make sure to voice your opinion
- Help us with the next revision …
  - Participate! Don't sit on the bench and wait and watch.
    - See http://www.eda.org/vasg
  - Ask your colleagues and vendors to participate
  - Help fund the effort
    - Joing Us

VHDL = Verification & Hardware Description Language

SynthWorks & VHDL Standards

- At SynthWorks, we are committed to see that VHDL is updated to incorporate the good features/concepts from other HDL/HVL languages such as SystemVerilog, E (specman), and Vera.
- At SynthWorks, we invest 100's of hours each year working on VHDL's standards
- Support VHDL's standards efforts by:
  - Encouraging your EDA vendor(s) to support VHDL standards,
  - Participating in VHDL standards working groups, and / or
  - Purchasing your VHDL training from SynthWorks

SynthWorks VHDL Training

Comprehensive VHDL Introduction 4 Days
http://www.synthworks.com/comprehensive_vhdl_introduction.htm
A design and verification engineer's introduction to VHDL syntax, RTL coding, and testbenches. Students get VHDL hardware experience with our FPGA based lab board.

VHDL Testbenches and Verification 4 days
http://www.synthworks.com/vhdl_testbench_verification.htm
Learn essential verification techniques including self-checking, transaction-based testing, data structures (linked-lists, scoreboards, memories), and randomization

VHDL Coding for Synthesis 4 Days
http://www.synthworks.com/vhdl_rtl_synthesis.htm
Learn VHDL RTL (FPGA and ASIC) coding styles, methodologies, design techniques, problem solving techniques, and advanced language constructs to produce better, faster, and smaller logic.

For additional courses see: http://www.synthworks.com