VHDL Model Efficiency

- Need an entity that can be plugged into any model hierarchy for signal monitoring
- Initial goal will be simple – at end of simulation, report:
  - Signals that are never assigned (still a 'U')
  - Signals that are never assigned a '1' value
  - Signals that are never assigned a '0' value
- For first condition ('U' signals), only have to examine signal list at end of simulation
- For other two conditions, not sufficient to examine final signal values
  - If a signal value is a '1', it may or may not have been assigned a '0' during the simulation
  - If a signal value is a '0', it may or may not have been assigned a '1' during simulation

monitor VHDL Entity

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

entity monitor is
 generic (
  FName : String := "./monitor.log";
  N       : integer := 1
  );
 port (
     a  : in std_logic_vector(N-1 downto 0);
    log : in std_logic
  );
end monitor;
```

Output file

List of signals to monitor

When transitions from '0' to '1', print signal info to log file.

Need for monitor to be efficient because signal list may be very large (thousands to 10's of thousands of signals)

First Try – call this architecture 'behv'

Write a single process whose sensitivity list contains the 'a' vector. Process triggered any time a change occurs on any bit in the 'a' vector.

Declare two boolean variable arrays within the process called v_one, v_zero.

When process triggers, loop through signal list

if value of signal is '1', set corresponding entry in v_one variable to TRUE.

If value of signal is '0', set corresponding entry in v_zero array to TRUE.


```
architecture behv of monitor is


type boolv is array(natural range <>)
   of Boolean;
FILE OutFile : text;

Begin
  process (a,log)
    variable v_one : boolv(0 to N-1);
    variable v_zero : boolv(0 to N-1);
    variable init : boolean := FALSE;
    VARIABLE LL: line;

 begin
     for i in 0 to N-1 loop
       if (a(i) = '1') then
         v_one(i) := TRUE;
       end if;
       if (a(i) = '0') then
         v_zero(i) := TRUE;
       end if;
      end loop;
```

Some code has been deleted for brevity, see zip archive for complete listing.

Arrays for recording '1', '0' assignments

Loop through 'a', recording if bit has been set to '1' or '0'.

Execution time grows by N * number of events on 'a'

There must be a better way

- The execution time of the previous approach grows by N*#of events on 'a'
- Would like to eliminate unnecessary examination of signals within 'a' that did not experience an event
- Have a separate process for each bit of vector 'a'
- Execution time would then be proportional to the number of events on 'a', and not the size of 'a'
- How do we parameterize generation of the processes??
  - Use a GENERATE block!


```
architecture gen1 of monitor is
  signal v_one: std_logic_vector (N-1 downto 0);
  signal v_zero: std_logic_vector (N-1 downto 0);
  FILE OutFile : text;

 begin
  PGEN:

  for i in 0 to N-1 generate
   process (a(i))
    begin
      if (a(i) = '1') then
       v_one(i) <= '1';
      elsif (a(i) = '0') then
       v_zero(i) <= '1';
      end if;
    end process;
  end generate PGEN;

process(log)
 variable init : boolean := FALSE;
    VARIABLE LL: line;
……….
```

Signals for keeping information, visible to all processes. Shared variables not available until VHDL93

GENERATE block creates a process for each bit of 'a'

```
architecture gen2 of monitor is
  signal v_one: std_logic_vector (N-1 downto 0);
  signal v_zero: std_logic_vector (N-1 downto 0);
  FILE OutFile : text;

Begin
  PGEN:for i in 0 to N-1 generate
     v_one(i) <= '1' when (a(i)='1') else v_one(i);
     v_zero(i) <= '1' when (a(i)='1') else v_zero(i);
  end generate PGEN;
```

This accomplishes the same result as previous slide.


```
architecture gen3 of monitor is
 type boolv is array(natural range <>) of boolean;
 shared variable  v_one: boolv (N-1 downto 0);
 shared variable v_zero: boolv (N-1 downto 0);
 FILE OutFile : text;

 procedure set_value(signal x :std_logic; ind :natural) is
 begin
      if (x = '1') then v_one(ind) := TRUE;
       elsif (x = '0') then v_zero(ind) := TRUE;
      end if;

  end;
begin
  PGEN: for i in 0 to N-1 generate
    process (a(i))
    begin
      set_value(a(i),i);
    end process;
  end generate PGEN;
```

Use shared variables, use a procedure call to set the value of the shared variables (cannot assign a variable outside of a process, procedure or function)

Which Method has best execution time?

Test for different # of signals, #number of events. Times measured on 450 Mhz Sun server

# of signals, # of events	Approaches (Execution time)			
	behv (single process, local variables)	gen1 (signals, generated processes)	Gen2 (signals, generated concurrent assign)	gen3 (shared variables, generated processes)
2000, 500	3.9 s	4.0 s	4.1 s	3.7 s
2000, 50000	10.5 s	5.8 s	5.8 s	4.2 s

Increasing number of events by 100x has small impact on GENERATE approaches, large effect on single process approach.

More Data

# of signals, # of events	Approaches (Execution time)			
	behv (single process, local variables)	gen1 (signals, generated processes)	Gen2 (signals, generated concurrent assign)	gen3 (shared variables, generated processes)
2000, 500	3.9 s	4.0 s	4.1 s	3.7 s
20000, 500	5.1 s	22.0 s	22.3 s	4.4 s

Increasing number of signals by 10x has large impact on GENERATE approaches which use signals. Why?……..

One more comparison

# of signals, # of events	Approaches (Execution time)			
	behv (single process, local variables)	gen1 (signals, generated processes)	Gen2 (signals, generated concurrent assign)	gen3 (shared variables, generated processes)
20000, 500	5.1 s	22.0 s	22.3 s	4.4 s
20000, 50000	50.3 s	22.8 s	22.6 s	5.1 s

Using large number of signals, and increasing events by 100x causes little change in GENERATE approaches

Performance Data

- Obvious from code that the single process approach would be poor for large numbers of event or signals
  - execution time affected by both # of signals and # of events.
- Why are the first two GENERATE approaches sensitive to number of signals and not number of events?
  - Most of the execution time is actually initialization time
  - Takes time to allocate and initialize data structures for such a large number of signals.
- Fastest execution time and best scaling performance was the GENERATE approach that used shared variables
  - A simple message – if you can substitute a variable for a signal do so – it will save both execution time and memory usage.

Representing Databook Timing Information

- Assume we must create VHDL models for Commercial Off-The-Shelf (COTS) parts
- How can the model represent databook timing values?
  - Speed grade information (-15, -20, -25)
  - Different temperature/voltage ranges (commercial vs military)
- Different parts from same family share similar timing parameters
  - All SRAMs have Taa (access time from address)
  - But… PLDs do not have this timing parameter
- Will use a hierarchy of packages to create a structure for representing databook timing

Package Hierarchy

- Create a base timing package that will define some parameters common to all data sheets
  - time_vector types, operating_point_type
- Create a timing package that represents the timing parameters shared by all members of a particular family
  - ie. 'ram_ce_oe_tv' package is shared by all SRAMs that have both a single chip enable and an output enable (separate IO)
- Finally, create a timing view package that contains the timing data for a particular part
- Must have some method for selecting a particular set of time values in the configuration
  - Would also like to be able to override individual timing values if desired

EIA567tv Package

- EIA567 was an attempt to create a standard VHDL package for representing component timing information
- Was never widely accepted
- A few interesting type definitions – we only used the operating point type from this package in the Cypress VHDL models

```
Type operating_point_type is
            (minimum, nominal, maximum);
```


ram_ce_oe_tv - Package for RAMs with single CE

```
PACKAGE ram_ce_oe_tv IS
  TYPE model_times IS
    RECORD
      -- read cycle
      trc    : time;  -- read cycle time
      taa    : time;  -- address to data valid
      toha   : time;  -- data hold from address change
      tace   : time;  -- ce low to data valid
      tdoe   : time;  -- oe low to data valid
      tlzoe  : time;  -- oe low to low Z
      thzoe  : time;  -- oe high to high Z
      tlzce  : time;  -- ce low to low Z
      thzce  : time;  -- ce high to high Z
      ……….
      ……….
END RECORD;
END ram_ce_oe_tv;
```

Defines timing parameters for this SRAM family

Lots more like this, all record fields not shown.

cy7b134_tv Package – contained timing information for a particular part (a dual port SRAM)

```
PACKAGE cy7b134_tv IS

TYPE speed_grade_type is (cy7b134_com_20, cy7b134_com_25,
cy7b134_com_35, cy7b134_debug);
TYPE op_array IS ARRAY (operating_point_type) of  model_times;
TYPE eds IS ARRAY (speed_grade_type) of op_array;

CONSTANT times : eds;
CONSTANT speed_grade_default: speed_grade_type;
END cy7b134_tv;
```

Package header

2D array of records indexed by operating point, speed grade.

Lookup array that contains all timing data, values specified in package body

cy7b134_tv Package body - contains timing data

```
PACKAGE BODY cy7b134_tv IS
CONSTANT speed_grade_default: speed_grade_type := cy7b134_com_25;
CONSTANT times : eds  :=
( cy7b134_com_20 =>
  (minimum => (trc => 20 ns, taa => 0 ns, toha => 3 ns, tace => 0 ns,
               tdoe => 0 ns, tlzoe => 3 ns, thzoe => 0 ns, tlzce => 3 ns,
               thzce => 0 ns,  tpu => 0 ns, tpd => 0 ns, twc => 20 ns,
               tsce => 15 ns, taw => 15 ns, tha => 2 ns, tsa => 0 ns,
               tpwe => 15 ns, tsd => 13 ns, thd => 0 ns, thzwe => 0 ns,
               tlzwe => 3 ns, twdd => 0 ns, tddd => 0 ns),
   nominal => (trc => 20 ns, taa => 20 ns, toha => 3 ns, tace => 20 ns,
               tdoe => 13 ns, tlzoe => 3 ns, thzoe => 13 ns, tlzce => 3 ns,
               thzce => 13 ns,  tpu => 0 ns, tpd => 20 ns, twc => 20 ns,
               tsce => 15 ns, taw => 15 ns, tha => 2 ns, tsa => 0 ns,
               tpwe => 15 ns, tsd => 13 ns, thd => 0 ns, thzwe => 13 ns,
               tlzwe => 3 ns, twdd => 40 ns, tddd => 30 ns),
   maximum => (trc => 20 ns, taa => 20 ns, toha => 3 ns, tace => 20 ns,
               tdoe => 13 ns, tlzoe => 3 ns, thzoe => 13 ns, tlzce => 3 ns,
               thzce => 13 ns,  tpu => 0 ns, tpd => 20 ns, twc => 20 ns,
               tsce => 15 ns, taw => 15 ns, tha => 2 ns, tsa => 0 ns,
               tpwe => 15 ns, tsd => 13 ns, thd => 0 ns, thzwe => 13 ns,
               tlzwe => 3 ns, twdd => 40 ns, tddd => 30 ns)
  ),
 cy7b134_com_25 =>
 (minimum => (trc => 25 ns, taa => 0 ns, toha => 3 ns, tace => 0 ns,
```

Etc..

cy7b134 Entity

```
ENTITY cy7b134 IS

  GENERIC (
    operating_point  : operating_point_type := nominal;
    speed_grade      : speed_grade_type:= speed_grade_default;
-- EIA 567 Boolean Generics
    mgeneration      : Boolean := TRUE;  -- report timing violations
    xgeneration      : Boolean := TRUE; -- generate X's
    -- EIA 567 Wire delay generics for inputs
    WD_a_l, WD_a_r   : wd_vector;
    WD_io_l,WD_io_r  : wd_vector;
    WD_rw_l, WD_rw_r : Time := 0 ns;
    WD_oe_l, WD_oe_r : Time := 0 ns;
    WD_ce_l, WD_ce_r : Time := 0 ns;
    -- EIA 567 load delay generics for ouputs
    LD_io_l,LD_io_r  : ld_vector;
    -- override generics
    trc              : Time :=  Time'LEFT;
    taa              : Time :=  Time'LEFT;
    toha             : Time :=  Time'LEFT;
    tace             : Time :=  Time'LEFT;

    ……
   )
```

Not all generics shown.

These used for overriding individual timings

cy7b134 Entity (continued)

```
PORT (
    a_l  : IN      Std_Logic_Vector(AddressWidth-1 DOWNTO 0);
    a_r  : IN      Std_Logic_Vector(AddressWidth-1 DOWNTO 0);
    io_l : INOUT   Std_Logic_Vector(DataWidth-1 DOWNTO 0);
    io_r : INOUT   Std_Logic_Vector(DataWidth-1 DOWNTO 0);
    rw_l : IN      Std_Logic;
    rw_r : IN      Std_Logic;
    oe_l : IN      Std_Logic;
    oe_r : IN      Std_Logic;
    ce_l : IN      Std_Logic;
    ce_r : IN      Std_Logic
  );

CONSTANT t: model_times  := (
 trc   => ChooseDelay(trc,times(speed_grade)(operating_point).trc),
 taa   => ChooseDelay(taa,times(speed_grade)(operating_point).taa),
 toha  => ChooseDelay(toha,times(speed_grade)(operating_point).toha),
 tace  => ChooseDelay(tace,times(speed_grade)(operating_point).tace),
```

Constant declared in Entity, used by architecture code for all timing data

This function returns the individual generic parameter if it is not equal to TIME'LEFT (I.e, a non-default value has been specified). This allows override of databook value.