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

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!

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.

There are several ways to generate multiple processes, a GENERATE block is one option.
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

<table>
<thead>
<tr>
<th># of signals, # of events</th>
<th>Approaches (Execution time)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>behv (single process, local variables)</td>
</tr>
<tr>
<td>2000, 500</td>
<td>3.9 s</td>
</tr>
<tr>
<td>20000, 50000</td>
<td>10.5 s</td>
</tr>
</tbody>
</table>

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

More Data
<table>
<thead>
<tr>
<th># of signals, # of events</th>
<th>Approaches (Execution time)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>behv (single process, local variables)</td>
</tr>
<tr>
<td>2000, 500</td>
<td>3.9 s</td>
</tr>
<tr>
<td>20000, 50000</td>
<td>5.1 s</td>
</tr>
</tbody>
</table>

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

One more comparison
<table>
<thead>
<tr>
<th># of signals, # of events</th>
<th>Approaches (Execution time)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>behv (single process, local variables)</td>
</tr>
<tr>
<td>20000, 500</td>
<td>5.1 s</td>
</tr>
<tr>
<td>200000, 50000</td>
<td>50.3 s</td>
</tr>
</tbody>
</table>

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
  - Ex: 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
            tdoes : time; -- oe low to data valid
            tll : time; -- oe low to low Z
            tllz : time; -- oe low to low Z
            thz : time; -- oe high to high Z
        END RECORD;
END ram_ce_oe_tv;

Rambus memory models

PACKAGE rambusวงศewi 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
            tdoes : time; -- oe low to data valid
            tll : time; -- oe low to low Z
            tllz : time; -- oe low to low Z
            thz : time; -- oe high to high Z
        END RECORD;
END rambusวงศewi;

rambus_memory - Package for RAMs with single CE

PACKAGE rambus_memory 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
            tdoes : time; -- oe low to data valid
            tll : time; -- oe low to low Z
            tllz : time; -- oe low to low Z
            thz : time; -- oe high to high Z
        END RECORD;
END rambus_memory;
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;
             -- EIA 567 load delay generics for outputs
             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;
             ...)
  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),
    ...)
END cy7b134 Entity (continued)
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.

Not all generics shown.
These used for overriding individual timings