CS/EE 3710: Computer Design Laboratory
Lecture 3: Modeling Behavior in VHDL
(Chapter 4: Yalamanchili)
Fall Semester 1999
Chris Myers

The Process Statement

- Considered to be a single concurrent statement.
- Describes sequential execution that is dependent on one or more events, (ex. flip-flop).
- Executes in order, but in zero time.
- The order written in a process is significant.
- Think of it like a software program.

Sequential Statements

- Concurrent VHDL describes parallel systems, but it does not describe what a system does over time.
- Sequential VHDL describes operations, behavior, as a sequence of related events.
- Ideal for state machines or complex comb. logic.
- Found in processes, functions, and procedures.
- Have an order dependency, but may not imply a sequential circuit.

The Process Statement

process_name: process(sensitivity_list)
declarations
begin
  sequential statements
end process;

- Process name is optional, and is used identify it or its local variables during simulation.
- Optional sensitivity list specifies which signals changing cause the process to execute.
- Without a sensitivity list, it must contain at least one wait statement to suspend the process.
- A process must have either a sensitivity list or a wait statement, but not both.
Processes With Sensitivity Lists

* Executed during simulation whenever an event occurs on any of the signals in the sensitivity list.
* An event is defined to be any change in value.
* Used to describe circuits that respond to external stimuli, and may be combinational or sequential.
* List all inputs with asynchronous behavior.
* This includes clocks, reset signals, or inputs to blocks of combinational logic.
* CSA is actually a process with a sensitivity list of all signals on right-hand side.

Processes Without Sensitivity Lists

* Execution is somewhat different:
  * Starts execution from the `begin` statement,
  * Continues to the first occurrence of a `wait` statement,
  * Suspends until the condition in the `wait` is satisfied,
  * Continues to the next occurrence of a `wait` statement,
  * ... 
  * Continues to the `end process` statement, and repeats.

Example: Processes Without Sensitivity Lists

```vhdl
process(Rst,Clk)
begin
  if Rst='1' then
    Q<='00000000';
  elsif Clk='1' and Clk'event then
    if Load='1' then
      Q<=Data_in;
    else
      Q<=Q(1 to 7)&Q(0);
    end if;
  end if;
end process;
```

```vhdl
process
begin
  wait until Clk='1' and Clk'event;
  M_out<=data_in;
  wait until Clk='1' and Clk'event;
  M_out<=not data_in;
end process;
```

* Can describe multiple-clock circuits.
* Not supported by most synthesis tools.
* Primarily used in test benches.
* Allowed in our asynchronous synthesis tool.
Using Processes for Combinational Logic

architecture behavior of simple_mux is
begin
    process(Sel,A,B,C,D)
    begin
        if Sel="00" then
            Y<=A;
        elsif Sel="01" then
            Y<=B;
        elsif Sel="10" then
            Y<=C;
        elsif Sel="11" then
            Y<=D;
        end if;
    end process;
end simple_mux;

A Non-Combinational Process Example

process(Sel,A,B,C,D,E,F)
begin
    if Sel="000" then
        Y<=A;
    elsif Sel="001" then
        Y<=B;
    elsif Sel="010" then
        Y<=C;
    elsif Sel="011" then
        Y<=D;
    elsif Sel="100" then
        Y<=E;
    elsif Sel="101" then
        Y<=B;
    end if;
end process;

Using Processes for Combinational Logic

※ A process describes combinational logic if:
※ The sensitivity list includes all signals being read.
※ Assignment statements for the outputs cover all possible input combinations.
※ All variables used must have a value assigned to them before they are read.

Example: Full_Adder
Using Processes for Registered Logic

* Most common use of processes is to describe behavior of circuits that have memory.  
* A process describes registered logic if:  
  * Not all inputs are in the sensitivity list.  
  * if-then-elseif logic is incompletely specified.  
  * Use one or more variables in a way that it must hold a value between iterations.

```
architecture rotate1 of rotate is
signal Qreg:std_logic_vector(0 to 7);
begin
  reg:process(Rst,Clk)
  begin
      if Rst='1' then
          Qreg<="00000000";
      elsif (Clk='1' and Clk'event) then
          if (Load='1') then
              Qreg<=Qreg(1 to 7) & Qreg(0);
          end if;
      end if;
  end process;
  Q<=Qreg;
end rotate1;
```

If-Then-Else Statements

```
if first_condition then
    statements
elsif second_condition then
    statements
...
else
    statements
end if;
```

* Conditions must be of type boolean.

```
signal S: std_logic;
begin
    if S then -- error: S is not Boolean!
    Change to:
    if S='1' then -- Now it works.
* Can nest multiple levels of if-then-else stmts.
if outer_condition then
    statements
else
    if inner_condition then
        statements
    end if;
end if;
```

Case Statements

```vhdl
case control_expression is
  when test_expression1 =>
    statements
  when test_expression2 =>
    statements
  when others =>
    statements
end case;
```

※ Test expressions must be mutually exclusive, and must include all possible conditions.
※ Difference from if-then-else is no priority implied.

For Loops

```vhdl
for i in 0 to D'length-1 loop
  if D(i)="1" then
    otmp:=not otmp;
  end if;
end loop;
```
※ Index i is automatically declared.
※ Index does not need to be a numeric type.

```vhdl
architecture looper2 of my_entity is
  type stateval is Init,Clear,Send,Receive,Error;
begin
  process(a)
  begin
    loop1:for state in stateval loop
      if current_state=state then
        valid_state<=true;
      end if;
    end loop loop1;
  end process;
end architecture looper2;
```

While Loops

```vhdl
architecture while_loop of my_entity is
begin
  ...
  process(...)
  begin
    ...
    loop_name:while(condition)loop
    -- repeated statements go here
    end loop loop_name;
  end process;
  ...
end while_loop;
```

Infinite Loops

```vhdl
architecture infinite_loop of my_entity is
begin
  ...
  process(...)
  begin
    ...
    loop_name:loop
    ...
    end loop loop_name;
  end process;
  ...
end infinite_loop;
```
Simulation of Concurrent Processes

* Upon initialization all processes executed once.
* Thereafter activated by events on signals in the sensitivity list (or wait statements).
* CSAs executed when there is an event on a signal on the right-hand side of the CSA.
* A process may read or write any of the signals declared in the architecture or any of the ports.

Simulation Example: Gate-Level Full Adder

What we have learned so far:

* Processes used to model sequential behavior.
  * Several types of sequential statements:
    * if-then-else
    * case
  * for loop, while loop, and (infinite) loop
* Processes executed when there are events on signals in their sensitivity list.
* NEXT TIME: wait statements, attributes, modeling state machines using processes.