Lecture 4: Modeling Behavior in VHDL (cont)

(Chapter 4: Yalamanchili)

Fall Semester 1999

Chris Myers

Signal and Variable Assignments

- Within a process, signals and variables have fundamentally different behavior.
- Variables are assigned new values immediately.
- Signal assignments are scheduled and do not occur until the current process (or subprogram) suspends.
- When you describe complex logic, you must carefully decide which to use.

Signals and Variables: Which To Use?

architecture version1 of circuit is
signal Q1,Q2:std_logic;
begin
  process(Clk)
  begin
    if rising_edge(Clk) then
      Q1<=A and B;
      Q2<=Q1 and C;
    end if;
  end process;
end version1;

architecture version2 of circuit is
variable Q1,Q2:std_logic;
begin
  process(Clk)
  begin
    if rising_edge(Clk) then
      Q1:=A and B;
      Q2:=Q1 and C;
    end if;
  end process;
end version2;
The Wait Statement

* wait for time expression;
  * wait for 20 ns;
  * wait on signal;
  * wait on clk, reset, status;
* wait until condition;
  * wait until clk='1' and clk'event;
* wait;

Example: D Flip-Flop

library IEEE;
use IEEE.std_logic_1164.all;
entity df f is
  port(D,Clk:in std_logic;
       Q,Qbar:out std_logic);
end df f;
ar chitectur e behavioral of df f is
begin
  output:process
  begin
    wait until (Clk'event and Clk='1');
    Q <= D after 5 ns;
    Qbar <= not D after 5 ns;
  end process output;
end behavioral;

Another Example: Asynchronous Communication

library IEEE;
use IEEE.std_logic_1164.all;
entity handshake is
  port(input_data : in std_logic_vector(31 downto 0));
end handshake;
ar chitectur e behavioral of handshake is
signal transmit_data : std_logic_vector(31 downto 0);
signal RQ, ACK : std_logic;
begin
  4-phase handshake
Another Example: Asynchronous Communication

```vhdl
producer: process
begin
  wait on input_data; -- DIFFERENT THAN BOOK
  transmit_data <= input_data;
  RQ <= '1';
  wait until ACK='1';
  RQ <= '0';
  wait until ACK='0';
end process producer;

consumer: process
variable receive_data: std_logic_vector(31 downto 0);
begin
  wait until RQ='1';
  receive_data:=transmit_data;
  ACK <= '1';
  wait until RQ='0';
  ACK <= '0';
end process consumer;
end behavioral;
```

Predefined Attributes

- Allow you to extract information about an object.
- Allow you to assign information to objects.
- Five kinds: function, value, range, signal, type.

Function Attributes

- **'Event** - true if signal had an event in sim. delta.
  ```vhdl
  if Clk='1' and Clk'event then --Clock edge.
  ```
- **'Active** - true if a transaction (scheduled event) occurred on this signal.
  ```vhdl
  Q<=D after 10 ns;
  A:=Q'active -- A gets a value of True
  E:=Q'event -- E gets a value of False
  ```
- **'Last_event** - time elapsed since previous event.
  ```vhdl
  Q<=D after 5 ns;
  wait for 10 ns;
  T:=Q'last_event; -- T gets a value of 5 ns
  ```
- **'Last_value** - signal value before previous event.
  ```vhdl
  Q<=1;
  wait for 10 ns;
  Q<=0;
  wait for 10 ns;
  V:=Q'last_value; -- V gets a value of '1'
  ```
- **'Last_active** - time elapsed since last transaction.
  ```vhdl
  Q<=D after 30 ns;
  wait for 10 ns;
  T:=Q'last_active; -- T gets a value of 10 ns
  ```
Value Kind Attributes

* `Left` - returns left-most bound of given type.

```vhdl
type bits1 is array(0 to 4) of bit;
type bits2 is array(4 downto 0) of bit;
variable L1:integer:=bits1'left; --L1=0
variable L2:integer:=bits2'left; --L2=4
```

* `Right` - returns right-most bound of given type.

```vhdl
variable R1:integer:=bits1'right; --R1=4
variable R2:integer:=bits2'right; --R2=0
```

* `High` - returns upper bound of given type.

```vhdl
variable H1:integer:=bits1'high; --H1=4
variable H2:integer:=bits2'high; --H2=4
```

Example: Width-independent Rotate

```vhdl
architecture behavior of rotater is
begin
  reg:process(Rst,Clk) begin
    if Rst='1' then
      Qreg<=(others=>'0');
    elsif rising_edge(Clk) then
      Qreg<=Data(Data'left+1 to Data'right)
      &Data(Data'left);
    end if;
  end process;
end behavior;
```

Range Kind Attributes

* `Range` - returns range value for an array.

```vhdl
for i in D'range loop
```

* `Reverse_range` - returns reverse of range value.

```vhdl
for i in D'reverse_range loop
```
library IEEE;
use IEEE.std_logic_1164.all;

entity state_machine is
  port (reset, clk, x : in std_logic; z : out std_logic);
end state_machine;

architecture behavioral of state_machine is
  type statetype is (state0, state1);
  signal state, nextstate : statetype := state0;
begin
  comb_process : process (state, x)
  begin
    case state is
      when state0 =>
        if x = '0' then
          next_state := state1;
          z := '1';
        else
          next_state := state0;
          z := '0';
        end if;
      when state1 =>
        if x = '1' then
          next_state := state0;
          z := '0';
        else
          next_state := state1;
          z := '1';
        end if;
    end case;
  end process comb_process;

  clk_process : process
  begin
    wait until (clk'event and clk = '1');
    if reset = '1' then
      state <= statetype'left;
    else
      state <= next_state;
    end if;
  end process clk_process;
end behavioral;

Using Processes to Model State Machines (clk part)

Using Processes to Model State Machines (comb part)

Another Example: Odd Parity Generator (comb part)
Another Example: Odd Parity Generator (clk part)

Common Syntax Errors

- Do not forget a semicolon at end of statement.
- Use `elsif`, not `elseif`.
- Use `end if`, not `endif`.
- Use 10 ns, not 10ns.
- Use half_adder, not half-adder.
- x"00000000" may be type mismatch, should use to_stdlogicvector(x"00000000").

Common Run-Time Errors

- Signals only get new value at next sim. cycle.
- Make sure only one source for signal unless you meant to have multiple sources.
- A process must have a sensitivity list or a wait statement (and not both).
- Remember all processes are executed once during initialization.

What we have learned so far

- Signals take on values only after next simulation cycle, variables take on values immediately.
- There are several types of Wait statements that can be used instead of sensitivity lists.
- Attributes can be used to get additional information about signals.
- Processes can be used to easily model state machines.
- NEXT TIME: Modeling structure in VHDL.