Specification and Verification of Hardware

VHDL 2

Original by Magnus Björk

Computing Science
Chalmers University of Technology

Overview of the lecture

- More VHDL stuff:
  - More delta delays and simulation cycle details
  - The multivalued std_logic type
  - Arithmetic on n-bit binary numbers
  - Generic entities
  - Testbenches
- The VHDL lab
- Short Jasper Gold demo

Previous lecture

- Different abstraction levels of hardware description
  - Transistor level (not used in this course)
  - Gate level
  - Register transfer level
  - Behavioral level
  - Functional level
- Basic verification

Previous lecture (cont)

- VHDL
  - Entities: interface of (sub)circuits
  - Architectures: implementation of (sub)circuits
  - Sequential code
    - Processes, algoritms, loops, assignments
  - Concurrent code
    - Simple statements that are executed concurrently
  - Structural code
    - Entities, signals, port maps
  - Simulation time is a pair: $ms + n\delta$
  - Signal assignments are put on a queue and carried out when all processes are waiting

Simulation cycle

1. Simulation time is advanced to the time of the next scheduled event (can be a signal assignment, or a wait).
2. Signal assignments are carried out.
3. Processes resume execution if they
   - are sensitive to signals that were affected, or
   - are scheduled to wait until the current time point
4. The processes continue to run until they all reach wait statements. New events that the processes create are put in the event queue.

Delta cycles

- As long as there are more events scheduled at the current time, we add 1δ.
- We can never refer to the different δ-cycles in the futur (no such thing as **wait for** 2 ns + 5 δ).
- If there are always more delta cycles, the real time will never advance (processes without **wait**-statements, feedback loops without flip flops)
- Add the keyword **postponed** before **process** to force process to be executed when no more delta cycles are scheduled.

Our simplified model

- In reality, gates introduce delays
- If we model an and-gate with

```
o <= i1 and i2;
```

  the signal is propagated through it in one δ (no real time)
- In our model, everything happens within infinitely short time (only a number of δ) after positive clock edge. Then the circuit does nothing until the next positive clock edge.
- After synthesis, timing problems must be taken care of.

Two equivalent architectures?

```
entity and3 is
   port( i1, i2, i3 : in bit; o : out bit );
end entity and3;

architecture behavioral of and3 is
begin
   o <= i1 and i2 and i3;
end architecture behavioral;

architecture structural of and3 is
   signal s : bit;
begin
   a1 : entity work.and_gate(behavioral) port map( i1, i2, s
   a2 : entity work.and_gate(behavioral) port map( s, i3, o
end architecture structural;
```

A testbench

```
architecture arch of testbench is
  signal i1, i2, i3, o1, o2 : bit;
begin
  a1 : entity work.and3(behavioral) port map (i1, i2, i3, o1);
  a2 : entity work.and3(structural) port map (i1, i2, i3, o2);

  assert o1 = o2 report "Mismatch!";

  stimuli : process is
  begin
    i1 <= '0'; i2 <= '0'; i3 <= '0'; wait for 1 ms;
    i1 <= '1'; i2 <= '1'; i3 <= '1'; wait for 1 ms;
    i1 <= '0'; i2 <= '1'; i3 <= '1'; wait;
  end process stimuli;
end architecture arch;
```


Delta delays

- During simulation, `Mismatch!` is displayed
- And3.behavioral propagates values in 1δ, and3.structural in 2δ.
- Hence the two architectures are not completely equivalent, even though they clearly represent the same circuit.

Functional equivalence does not depend on δ delays.

- Combinational circuits: use sequential asserts, only require equivalence after **wait** statements
- Sequential circuits: only require equivalence when clock is low:
  assert clk='1' **or** o1 = o2;

std_logic

- Include the following lines before all entites:
  library ieee;
  use ieee.std_logic_1164.all;
  use ieee.std_logic_arith.all;
- Declare signals of type std_logic instead of bit
- Has the following values:

'U' uninitialized	'W' weak unknown
'X' forcing unknown	'L' weak 0
'0' forcing 0	'H' weak 1
'1' forcing 1	'-' don't care
'Z' high impedance	

std_logic vs. std_ulogic

- In some cases, one wants to have multiple drivers of a wire. Example: a data bus.
- To allow muiltiple drivers for a signal in VHDL, one mu define a *resolution function*, that tells what happens if different processes assigns different values in the sam simulation cycle.
- std_logic has a predefined resolution function
- std_ulogic has no predefined resolution function

std_logic_vector

- Can be used for arithmetics
- There are two identical types: *unsigned* and *signed*, with different operations
- Each bit can easily be accessed
- Examples in the following slides

Example: 4-bit adder

```
entity add4bit is
   port( a, b : in unsigned(3 downto 0);
         s : out unsigned(3 downto 0));
end entity add4bit;

architecture behavioral of add4bit is
begin
   s <= a + b;
end architecture behavioral;
```


Structural architecture of add4bit

```
architecture structural of add4bit is
   signal c_0, c_1, c_2, c_3, c_4 : std_logic := '0';
begin
   fa1 : entity work.full_adder(structural)
         port map (c_0, a(0), b(0), s(0), c_1);
   fa2 : entity work.full_adder(structural)
         port map (c_1, a(1), b(1), s(1), c_2);
   fa3 : entity work.full_adder(structural)
         port map (c_2, a(2), b(2), s(2), c_3);
   fa4 : entity work.full_adder(structural)
         port map (c_3, a(3), b(3), s(3), c_4);
end architecture structural;
```

This is the ripple carry adder shown in the previous lecture.

A testbench

```
architecture arch of add_tester is
   signal a, b : unsigned (3 downto 0):= "0000";
   signal s1, s2 : unsigned (3 downto 0);
begin
   a_str : entity work.add4bit(structural) port map (a, b, s1)
   a_beh : entity work.add4bit(behavioral) port map (a, b, s
   stimuli : process is
   begin
      a <= "0000"; b <= "0000"; wait for 1 us; assert s1=s2;
      a <= "0010"; b <= "0010"; wait for 1 us; assert s1=s2;
      a <= "1011"; b <= "0001"; wait for 1 us; assert s1=s2;
      a <= "1011"; b <= "0101"; wait for 1 us; assert s1=s2;
      wait;
   end process stimuli;
end architecture arch;
```

Generic Parameters

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

entity add_constant is
   generic(const : integer;
           bits : natural);
   port(i : in signed(bits-1 downto 0);
        o : out signed(bits-1 downto 0));
end entity add_constant;

architecture behav of add_constant is
begin
   o <= conv_signed(const + conv_integer(i), bits);
end architecture behav;
```


Generic Parameters

```
architecture test of test_add_constant is
   signal i, o : signed(7 downto 0);
begin
   addc : entity work.add_constant(behav)
          generic map(10, 8)
          port map(i, o);
   stimuli : process is
   begin
      i <= conv_signed(0, 8),
           conv_signed(5, 8) after 1 ms,
           conv_signed(100, 8) after 2 ms,
           conv_signed(120, 8) after 3 ms,
           conv_signed(-50, 8) after 4 ms;
      wait for 5 ms;
      assert false report "Simulation ends" severity failure;
   end process stimuli;
end architecture test;
```


More VHDL

VHDL has a lot more features, examples can be found in course literature:

- For-, while-, and until-loops
- Case statements
- Functions and procedures
- Arrays
- Enumerated types and subtypes
- Packages
- Generic structurally implemented circuits (**generate**)

The VHDL lab

- Construction and verification of a stopwatch
- Standard stopwatch with a start/stop button and a lap/reset button, and a 6 digits display
- One top level entity with two architectures: one behavioral and one RTL
- Behavioral: verify that it really implements the stopwatch (using testbenches)
- RTL: verify that it is functionally equivalent to the behavioral
- Formal verification of counter elements
- Write a short report on the verification

Guidelines

- **Guidelines for design and verification** available in the “Labs and Exams” section in the course page.
- **Hints:**
  - Don’t use structure in the behavioral model – think software
  - Use Gaisler’s two-process method (see lecture 4) in the RTL components
  - Use the guidelines on the course page

About the verification report

- Generally: Motivate why we should trust your circuit
- Use the guidelines on the course page
- Explain the choices you have made, and why
- Explain what properties you have checked
- Was some particular test especially hard? Did you find any bugs?