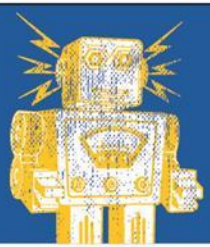


Introduction to VHDL II

Process – Sequential Statements – Concurrent Statements – Examples

Roland Höller
Email: hoeller@technikum-wien.at

The Block

- Groups concurrent statements
- Only within architectures
- Just used to name portions of code
- No "real" function associated

```
architecture rtl of logic is

  signal s_a, s_b, s_x0,
    s_x1, s_x2 : std_logic;

begin
  b_logic0 : block
  begin
    s_x0 <= s_a or s_b;
    s_x1 <= s_a xor s_b;
  end block b_logic0;

  b_logic1 : block
  begin
    s_x2 <= not(s_x0)
  end block b_logic1;
end rtl;
```

The Process

- Groups sequential statements
- Only within architectures
- Several processes run concurrently
- Process sensitivity list or wait statement
- Signals receive their value after finishing the event
- Variables receive their values immediately

```
architecture sim of logic is
  signal s_a, s_b, s_x0,
    s_x1, s_x2 : std_logic;
begin
  p_logic0 : process (s_a, s_b)
  begin
    if s_a = '0' and s_b = '0' then
      s_x0 <= '0';
    else
      s_x0 <= '1';
    end if;
    s_x1 <= s_a xor s_b;
  end process p_logic0;

  p_logic1 : process
  begin
    s_x2 <= not(s_x0);
    wait for 5 ns;
  end process p_logic1;
end sim;
```

Signals

- Signals connect the different units of a design like wires
- Communication between processes
- Keep the last value

- Signals can be declared
  - within a package
  - as port of an entity
  - within an architecture

```
architecture rtl of example is

  -- signal declaration
  signal s_result : std_logic;
  signal s_sum : integer;

begin

  -- signal assignment
  s_result <= '0';
  s_sum <= 5;

end rtl;
```

Variables

- Variables are declared and known only in processes
- Immediate assignment

- Variables are used
  - to be realized as combinatorial logic
  - to carry out an algorithm

```
p_decode : process (s_a, s_b)

  variable v_count : integer;

begin

  v_count := 4;
  s_a <= v_count;

end process p_decode;
```

Signals vs. Variables

Signals

- Assignment Operator:
  <=
- Declaration:

```
architecture rtl of fulladder
  signal s_sum : std_logic;
begin
```

- Visibility:
  in the whole architecture
- New Value:
  after suspend of process

Variables

- Assignment Operator:
  :=
- Declaration:

```
p_decode: process
  variable v_result :
std_logic;
begin
```

- Visibility:
  only in the process
- New Value:
  immediately after assignment

The IF-Statement

- Sequential statement
- “elsif” sequence
  - one or no branch
  - priority
- Avoid unintended latches!
  - Due to incomplete IF statements
  - In combinatorial processes

```
p_select : process (s_a, s_b, s_sel)
begin
  -- multiplexer
  if s_sel = '1' then
    s_out <= s_a;
  else
    s_out <= s_b;
  end if;
  -- unintended latch
  if s_sel = '1' then
    s_latch <= s_a;
  end if;
end process p_select;
```

The CASE- Statement

- Sequential statement
- Choice options must not overlap
- All choices have to be covered
- Avoid unintended latches!

```
p_select : process (s_a, s_b, s_sel)
begin
  -- multiplexer
  case s_sel is
    when "11" => s_out <= s_a;
    when "10" => s_out <= "000";
    when others => s_out <= s_b;
  end case;
end process;
```

Loops

- Sequential statement
- Loop-variable has not to be defined
- Assignments to loop variables are not allowed
- Label is optional

```
s_out <= "0000";
for_loop : for i in 3 downto 0 loop
  if s_a = i then
    s_out(i) <= '1';
  end if;
end loop for_loop;
```

```
while_loop : while true loop
  wait for 50 ns;
  clk_i <= not(clk_i);
end loop while_loop;
```

```
loop
  wait for 200 ns;
  s_run <= not(s_run);
end loop;
```

Operators

- Logical operators
  - not, and, or, xor, nand, nor, xnor, sll, srl, sla, sra, rol, ror;
  - Predefined for bit, bit_vector, boolean, std_ulogic, std_logic, std_ulogic_vector, std_logic_vector
- Relational operators (result is true or false)
  - <, >, =, <=, /=, >=;
  - Arrays are compared left-justified
- Arithmetical operators
  - +, -, /, *, **, abs, rem, mod
  - Predefined for integer, real (except mod and rem), time
  - Not defined for bit_vector, std_logic_vector, std_ulogic_vector

Operator Precedence

logical	and	or	nand	nor	xor	
relational	=	/=	<	<=	>	>=
adding	+	-	&			
unary sign	+	-				
shift	sll	srl	sla	sra	rol	ror
multiplying	*	/	mod	rem		
miscellaneous	**	abs	not			

low

high

- Use always brackets to define the sequence of calculation!

Concurrent Statements

- **"with-select" and "when" statements:**

```
data_o <= s_data when s_en = '1' else "ZZZZZZZZ";

data_o <= "000" when s_sel = "11" else
          data_i when s_cen = '0' else
          "111" when s_hub_i = "011101" else
          "010";
```

```
with s_sel select data_o <= data_i when "00",
                            "000" when "01",
                            "010" when others;
```

Syntax Errors

```
architecture rtl of test is
     signal s_u,s_x,s_c : std_logic;
     signal s_sel : std_logic_vector(2 downto 0);
     signal s_s : bit;
begin
   s_s <= '1';
   s_s <= '0';
   s_c <= s_s;
   s_u <= '1';
   s_u <= '0';
   s_x <= 'U';
   s_sel <= "000";

   process (s_sel)
   begin
     case s_sel is
       when "000" => s_x <= '1';
       when "001" => s_x <= '0';
     end case;
   end process;
end rtl;
```

- # ERROR: ./test.vhd(19): Nonresolved signal s_s already has a source (on line 18).
- **# ERROR: ./test.vhd(30): Case statement only covers 2 out of 729 cases.**
- # ERROR: ./test.vhd(36): VHDL Compiler exiting

VHDL Simulation

- VHDL code is executable
- Basic understanding of compilation and execution mechanisms for RTL coding required
- Check correctness of your RTL code using a VHDL testbench
- Verification is complex and time consuming
- Broader and deeper VHDL knowledge required

A Testbench I (Full Adder)

```
library IEEE;
use IEEE.std_logic_1164.all;
library work;

entity tb_fulladder is
end tb_fulladder;

architecture sim of tb_fulladder is

  component fulladder
    port (a_i : in std_logic;
          b_i : in std_logic;
          cy_i : in std_logic;
          cy_o : out std_logic;
          sum_o : out std_logic);
  end component;
```

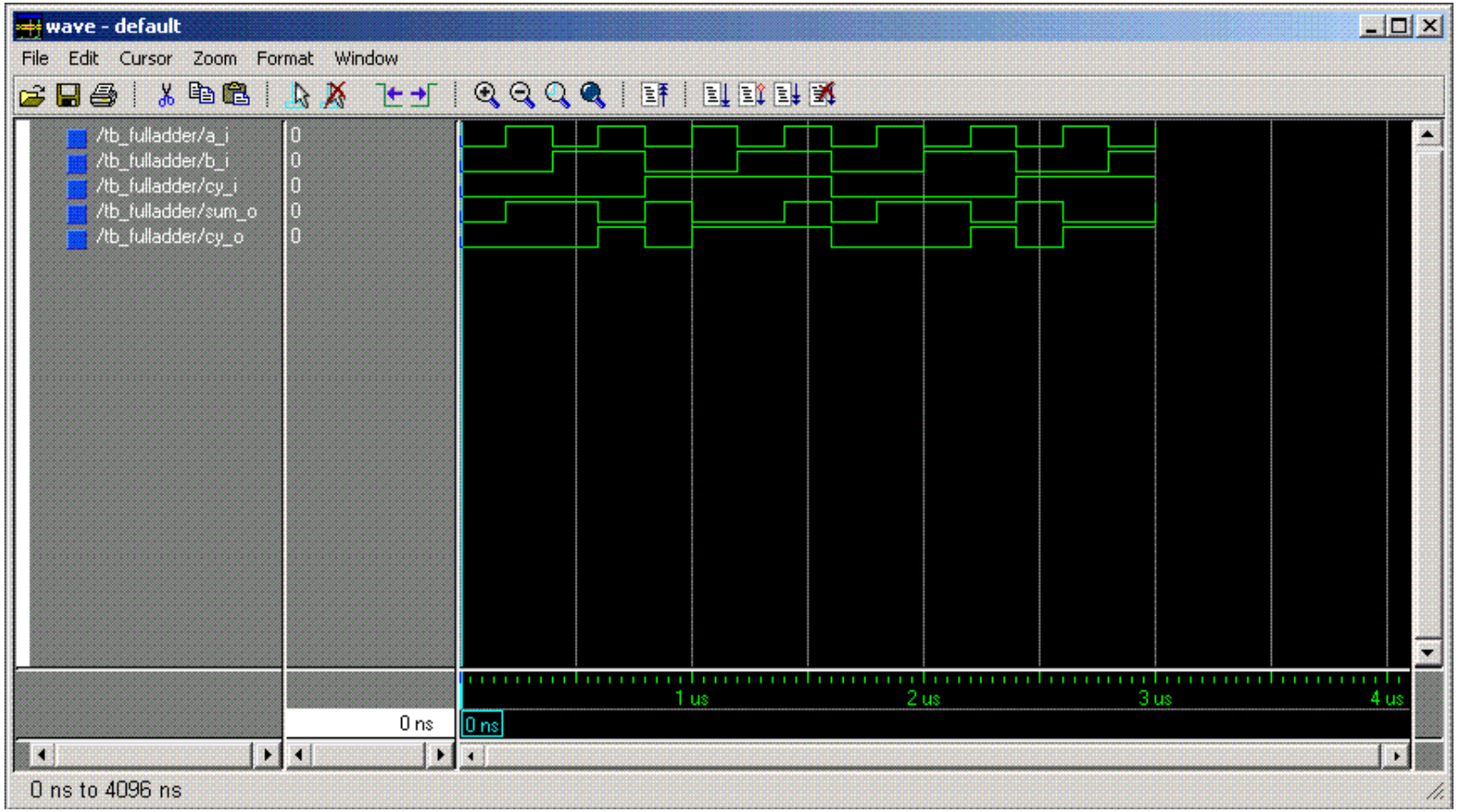
A Testbench II (Full Adder)

```
  signal a_i   : std_logic;
  signal b_i   : std_logic;
  signal cy_i  : std_logic;
  signal cy_o  : std_logic;
  signal sum_o : std_logic;

begin

  u1 : fulladder
    port map(a_i   => a_i;
             b_i   => b_i;
             cy_i  => cy_i;
             cy_o  => cy_o;
             sum_o => sum_o);
```

A Testbench III (Full Adder)

```
  p_run : process
  begin
    a_i <= '0';
    b_i <= '0';
    cy_i <= '0';
    wait for 100 ns;
    a_i <= '1';
    b_i <= '0';
    cy_i <= '0';
    wait for 100 ns;
end process p_run;
end sim;

configuration tb_fulladder_sim_cfg of tb_fulladder is
  for sim
  end for;
end tb_fulladder_sim_cfg;
```

Simulation of the Full Adder Design



Simulation

- The simulation of a VHDL model is performed in three
- steps:

1) Elaboration

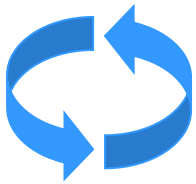
- **The design elements are created.**
- **All design objects are elaborated before simulation.**
- **Main VHDL constructs before side- or sub-components (i.e. entity before architecture).**
- **Component which is referred to in another one has to be analyzed first (e.g. package before entity).**

2) Initialization

- **Assign start values to the signals (as given in the declaration or the leftmost of the type).**

3) Execution

- **The simulation starts.**
- **One simulation cycle after another is executed.**

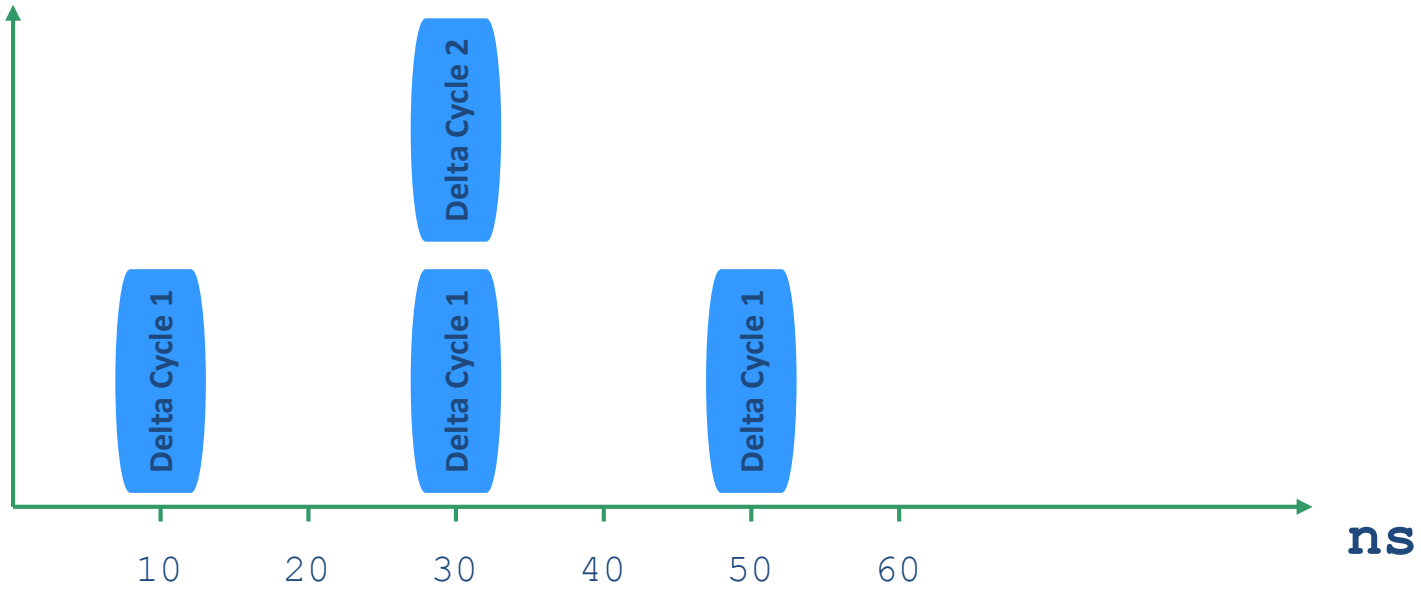


The Simulation Cycle

- The simulation cycle (delta cycle) consists of two steps:

  1) Signal update (list of signals, which changed)

  2) Process execution



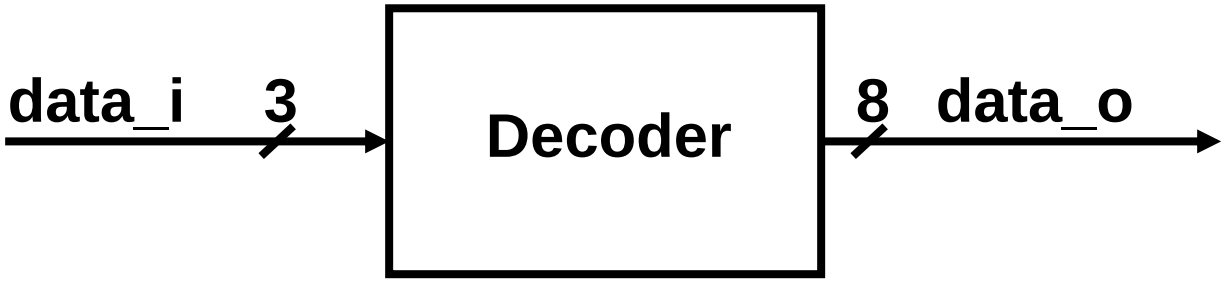
Summary

- Block, Process
- Signals, Variables
- Sequential statements
- Concurrent statements
- Operators
- Simulation with Testbench

Questions

- 1) Which portion of VHDL code contains sequential statements?
  - A The process before the begin
  - C The process after the begin
  - B The architecture
  - D The configuration
- 2) Which of the following VHDL constructs can be used stand alone?
  - A Architecture
  - C Process
  - B Entity
  - D Configuration
- 3) At which place in the VHDL code the type of a signal has to be given?
  - A At the declaration of the signal
  - C In the configuration
  - B At the first use of the signal in the code
  - D Not necessary
- 4) Which steps are necessary to simulate a VHDL model?
  - A Elaboration and Execution
  - C Execution
  - B Elaboration, Initialization, Execution
  - D Elaboration

Class Example: 3-bit Decoder



data_i	data_o
000	00000001
001	00000010
010	00000100
011	00001000
100	00010000
101	00100000
110	01000000
111	10000000