Dataflow modeling style

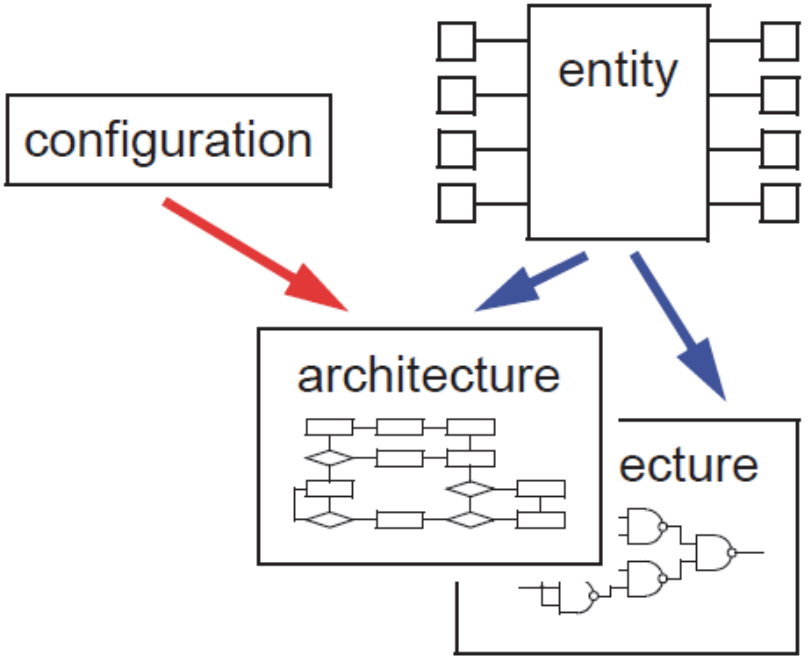
IAS0600 Digital Systems Design with VHDL

Natalia Cherezova, Peeter Ellervee

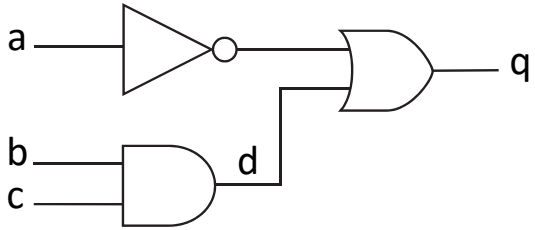
Modeling styles

- Dataflow
  - Concurrent signal assignments
- Structural
  - Interconnected components
- Behavioral
  - Sequential statements (in a process)

- Architectures may combine different styles to describe the needed behavior and/or structure of the system



Boolean equation




```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity example is
  Port ( a, b, c : in STD_LOGIC;
         q : out STD_LOGIC);
end example;

architecture bool of example is
  signal d : STD_LOGIC;
begin
  d <= b AND c;
  q <= (NOT a) OR d;
end bool;
```

- Supported operators
  - AND, OR, XOR, NOT, NAND, NOR, XNOR
- The only operator with precedence over the others is NOT

```
q <= (NOT a) OR d;    -- optional brackets
q <= a OR (b AND c); -- brackets are required
```

```
q <= (NOT a) OR d;
d <= b AND c;
```

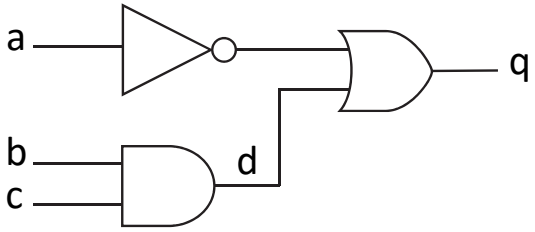
```
q <= (NOT a) OR (b AND c);
```

Conditional signal assignment

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity example is
  Port ( a, b, c : in STD_LOGIC;
         q : out STD_LOGIC);
end example;

architecture cond of example is
begin
  q <= '1' when a = '0' else
       '1' when b = '1' and c = '1' else
       '0';
end cond;
```



a	b	c	q
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

- The WHEN … ELSE statement

```
<signal> <= <value> when <condition> else
            <value> when <condition> else
            ...;
```

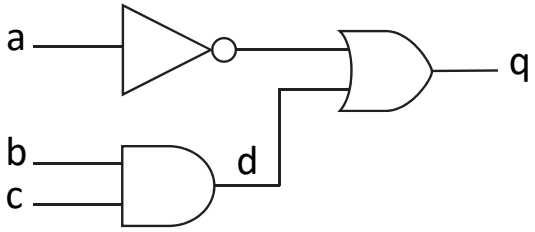
- The order of conditions is important
- Output should get the value in all the possible cases

Selected signal assignment

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity example is
  Port ( a, b, c : in STD_LOGIC;
         q : out STD_LOGIC);
end example;

architecture select of example is
begin
  with (a & b & c) select
    q <= '1' when "000" | "001",
         '1' when "010" | "011",
         '1' when "111",
         '0' when others;
end select;
```



a	b	c	q
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

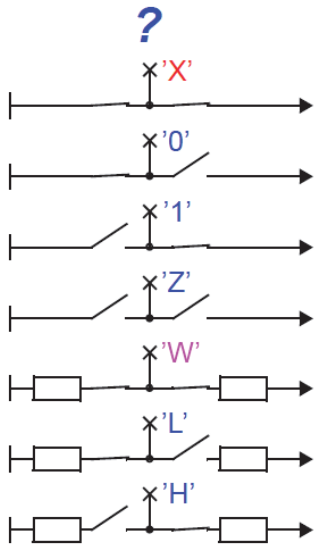
- The WITH…SELECT statement

```
with <identifier> select
  <signal> <= <value> when <identifier values>,
              <value> when <identifier values>,
              ...;
```

- Multiple identifier values can be grouped together using either I (means or) or TO (defines range)
- All identifier values should be covered, for which the keyword OTHERS is very useful

Data type STD_LOGIC

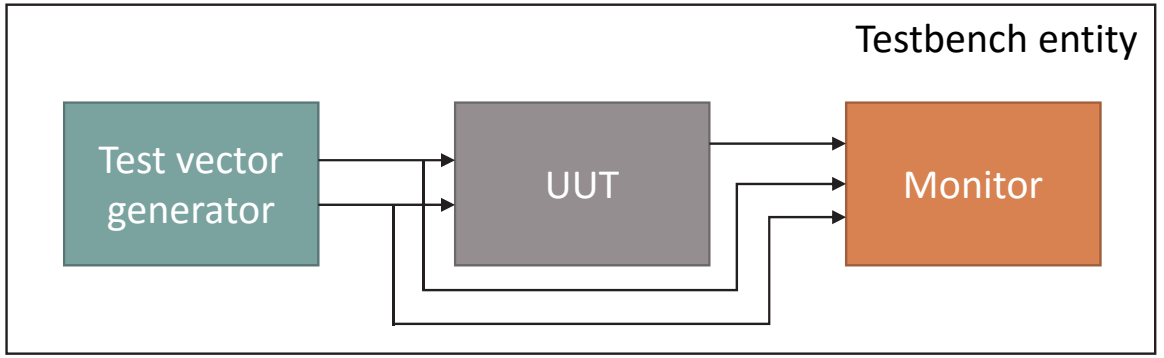
• ’U’	uninitialized	model behavior
• ’X’	forcing unknown	model behavior
• ’0’	forcing 0	logic level (”transistor”)
• ’1’	forcing 1	logic level (”transistor”)
• ’Z’	high impedance	disconnect
• ’W’	weak unknown	model behavior
• ’L’	weak 0	logic level (”resistor”)
• ’H’	weak 1	logic level (”resistor”)
• ’-’	don’t care	match all



- IEEE.std_logic_1164 – std_ulogic, std_logic ja std_logic_vector
  - std_ulogic – base type; std_logic – multiple signal sources and resolution function
  - IEEE.std_logic_arith & other packages define also arithmetic operations

Testbench automation

- During the simulation and testing, design inputs are driven by test vectors, and outputs are monitored and checked
- How to automate?
  - Output verification through assertion statements
  - Automated test vector generation for inputs



Assertions

- The ASSERT statement

```
assert <condition that should be TRUE>
  report <message issued when the condition is FALSE>
    severity <assertion type>;
```

- 4 types of assertions (severity levels):
  - note
  - warning
  - error
  - failure (simulation halts)

Input generation

- Loop through all possible input values

```
for <iterator> in <start> to <end> loop
  <loop_body>
end loop;
```

- No need to declare an iterator beforehand
- Define the range of the loop using integer values
- Convert integer values into corresponding std_logic_vector values
  - Uncomment `ieee.numeric_std.all` library
  - Integer → unsigned → std_logic_vector

```
x <= std_logic_vector(to_unsigned(<integer>, <number_of_bits>));
```

3 can be represented as 11 or as 0011 etc.

Testbench template

```
stimuli : process
begin
  for i in ... to ... loop
    for j in ... to ... loop
      in1_tb <= ...     -- convert integer i to std_logic_vector
      in2_tb <= ...     -- convert integer j to std_logic_vector
      wait for 10 ns;   -- required for the simulation!
      assert ...        -- compare UUT output with the result of i and j comparison
    end loop;
  end loop;
  wait;          -- halt the simulation
end process;
```

Problem: result of a comparison is of type BOOLEAN, design entity output is of type STD_LOGIC, there is no library function for BOOLEAN to STD_LOGIC conversion.

Solution: write conversion function ourselves.

Functions

- Function is a block of sequential VHDL code implementing a solution for a commonly encountered problem

```
function <name> (<input_list>) return <return_value_type> is
  [declarative_part]
begin
  <statement_part>
  return <expression>;
end function;
```

- A function call is always a part of the expression
- Only one value can be returned
- No timing control allowed!

Boolean to std_logic function

```
architecture arch of testbench is
-------------------------------------------
---- put your function definition in the -------
---- declarative part of the architecture ------
-------------------------------------------
  function bool_to_std_logic (input: boolean) return std_logic is
  begin
    if input then
      return '1';
    else
      return '0';
    end if;
  end function;

begin
  ...
  assert (eq_o_tb = bool_to_std_logic(i = j))
  ...
end arch;
```