VHDL Essentials

VHDL Essentials

Index

VHDL Essentials

Things you should know before we start

- Boolean algebra (logic): AND, OR, etc.
- Basic combinational digital systems such as
  - Multiplexer
  - Encoder, decoder
  - Comparator
  - Tri-state buffer
  - ALU (Arithmetic Logic Unit)
- Basic sequential digital systems such as
  - D flip-flop
  - Data and shift registers
  - Counters

VHDL Essentials

VHDL vs. processor

With a processor

1) Processors are programed to follow a series of instructions executed in a sequential order
2) Given fixed computer architecture
3) Single threat in a single processor. Ex. *Perform 4 additions with a PIC*
   a) Only 1 option: execution of ADD instruction 4 times

With VHDL

1) Statements infer digital systems, not instructions in a processor
2) There’s no processor but a custom-made circuit that employs only the necessary components that execute the intended algorithm
3) Flexibility to implement solutions in parallel. Ex. *Perform 4 additions in VHDL*
   a) Op1: 1 adder 4 times
   b) Op2: 2 adders 2 times
   c) Op3: 4 adders 1 time

VHDL Essentials

Introduction to VHDL

- VHDL = VHSIC + HDL: *Very High Speed Integrated Circuit + Hardware Description Language*
- Description of hardware circuits with a language

  Hardware Description Language (HDL)
- Circuits can be described based on their functional behavior

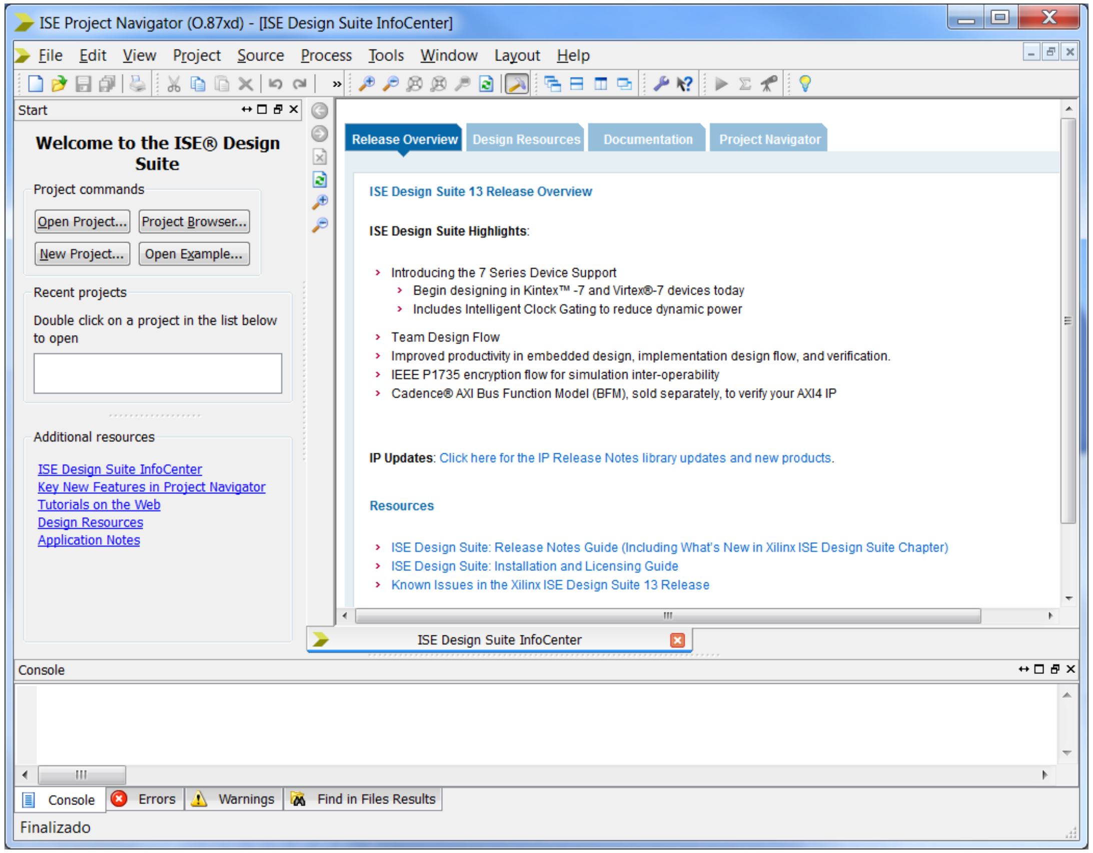
```
if A > B then
 output <= A;
end if;
```

- The VHDL synthesizer understands the user's code and infers the corresponding digital system
- VHDL designs are implemented in FPGA (Field Programmable Gate Array) devices

Introduction to VHDL

- Lab Software

  ISE Xilinx 13.4



Welcome to the ISE® Design Suite

Project commands

Recent projects

Double click on a project in the list below to open

Additional resources

ISE Design Suite InfoCenter
Key New Features in Project Navigator
Tutorials on the Web
Design Resources
Application Notes

ISE Design Suite 13 Release Overview

ISE Design Suite Highlights:

- Introducing the 7 Series Device Support
  - Begin designing in Kintex™ -7 and Virtex®-7 devices today
  - Includes Intelligent Clock Gating to reduce dynamic power
- Team Design Flow
- Improved productivity in embedded design, implementation design flow, and verification.
- IEEE P1735 encryption flow for simulation inter-operability
- Cadence® AXI Bus Function Model (BFM), sold separately, to verify your AXI4 IP

IP Updates: Click here for the IP Release Notes library updates and new products.

Resources

- ISE Design Suite: Release Notes Guide (Including What's New in Xilinx ISE Design Suite Chapter)
- ISE Design Suite: Installation and Licensing Guide
- Known Issues in the Xilinx ISE Design Suite 13 Release

Finalizado

- Full version available in the lab
- Free demo version

  ISE WebPACK can be downloaded with limited capabilities

  http://www.xilinx.com/support/download/index.htm

VHDL Essentials

Signal types

- VHDL designs have signals that can have logic or arithmetic values
- Most common signal types:
  - `Std_logic: '1'.` Bit with '0', '1' or 'Z' (high impedance) value
  - `Std_logic_vector(3 downto 0): "0011".` Vector of std_logic
  - `Signed: -2.` Integer with a sign
  - `Unsigned: 7.` Integer without a sign

- Normally signals are `std_logic`
- `Signed` and `unsigned` signals infer buses: `7 = "00000111"`
- Different signal types support different operations
  - `std_logic` signals do not support arithmetic operations
  - `signed` or `unsigned` types recommended for arithmetic operations

VHDL Essentials

Libraries and operators

- **Libraries** include definitions of signal types and operations
  - Normally our designs will have only two libraries

```
library ieee;
use ieee.std_logic_1164.all; -- for logic operations
use ieee.numeric_std.all;   -- for arithm operations
```

- **Most common operators**

Boolean: AND, OR, NOT	`out1 <= in1 OR in2;`
Arithmetic: +, -,*, / [1]	`out1 <= in1 + in2;`
Comparison: <, >=, /=	`if (in1 /= in2)...`
Concatenation (of bits): &	`out1<= in1 & in2;`

____

1 Division is not synthesizable, it can be used only in simulation

VHDL Essentials

Entity

- **Entity** defines how a design element connects to other VHDL models (its interface) and also defines its name
- **Ports** are the signals that connect entities, detailing inputs and outputs of our design

```
Entity example is
port(
      in1  : in std_logic; -- input1 port
      in2  : in std_logic; -- input2 port
      out1 : out std_logic); -- output1 port
End example;
```

Architecture

- **Architecture** describes the behavior of an entity

```
        Entity example is
        port(
                in1  : in std_logic;
                in2  : in std_logic;
                out1 : out std_logic);
        End example;

Architecture behavior of example is
-- Add here internal signals only visible within this architecture
internal_signal1 : std_logic;

Begin
 internal_signal1 <= NOT in1;
 out1 <= internal_signal1 AND in2;
End behavior;
```

VHDL Essentials

Process

- **Processes** contain the functional description of an architecture

```
out1 <= internal_signal1 AND in2; -- a simple process;
```

- A process can also be described by more than a line

```
process(A,B) --(A,B) is the sensitivity list of this process
Begin
     if (A > B) then
             Output <= A;
     else
             Output <= B;
     end if;
end process;
```

- During simulation, a process will be evaluated only when a change occurs in at least a signal of the **sensitivity list**. This shall include asynchronous signals (e.g. asynchronous reset) plus the process clock.
- Sensitivity list is only used for simulation. In synthesized circuits processes are constantly and concurrently executed.

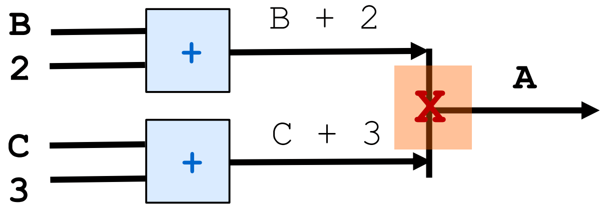
VHDL Essentials

Process

- When an architecture is defined by several processes, these will infer circuits that will run concurrently. The order of the processes is not relevant

```
-- Ex1. Processes 1 and 2 run in parallel
C = B + 2; -- process 1
A = D + E; -- process 2
-- Ex2. This code cannot be synthesized
A = B + 2; -- process 1
A = C + 3; -- process 3
```

- In Ex2, while in C language this results `A = C + 3`, in VHDL this code cannot be synthesized because signal A is assigned two different values: B+2 and C+3



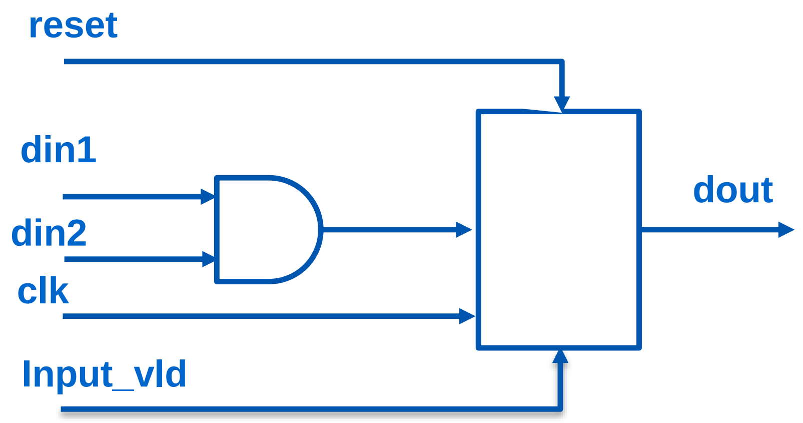
```
>> Error: Multiple sources for the same signal
```

VHDL Essentials

Synchronous Process

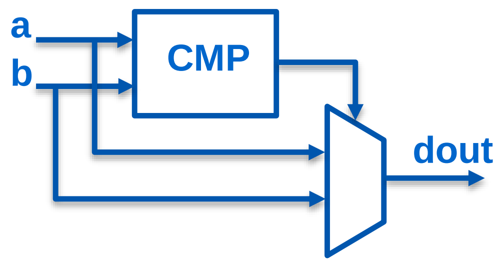
- Process output get updated just on the rising (falling) clock edge:
  - Thus, a memory element (biestable) is inferred at the output.
  - Process may have “if without else” statements → the biestable will remember the last value
  - Sensitivity list will include just async signals. Usually just reset and clk.

```
process(reset, clk)
begin
    if (reset = '1') then
        dout <= (others=>'0');
    elsif (clk'event and clk='1') then
        if (input_vld = '1') then
            dout <= din1 and din2;
        end if;
    end if;
end process;
```

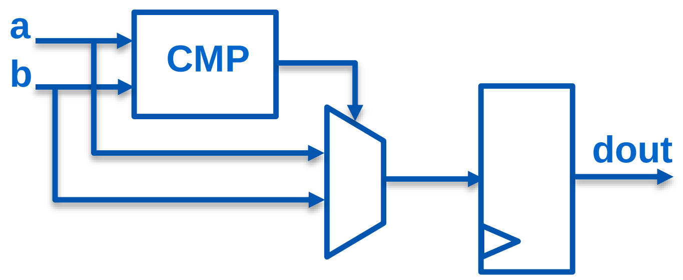


Sync vs Async Process

```
process(a, b)
begin
    if (a > b) then
        dout <= a;
    else
        dout <= b;
    end if;
end process;
```



```
process(clk)
begin
    if (clk'event and clk='1') then
        if (a > b) then
            dout <= a;
        else
            dout <= b;
        end if;
    end if;
end process;
```



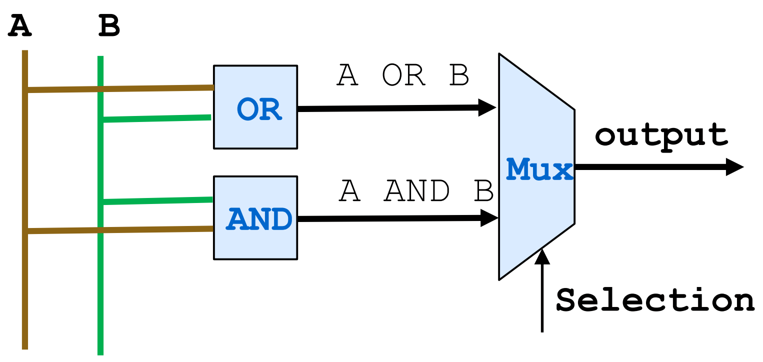
VHDL Essentials

Decision and loop statements

- If - then - elsif - else

```
 If (selection = ‘1’) then
   output <= A OR B;
else
   output <= A AND B;
end if;
```

- Both OR and AND operators are inferred



```
if (condition1 = ‘1’) then
   C <= 0;
elsif (condition2 = ‘1’) then
   C <=A;
else
   C <= B;
end if;
```

VHDL Essentials

Decision and loop statements

- **Case** is useful for branching, where is more convenient than if/else

```
case option is
  when "00" =>
        out <= a;
  when "01"
        out <= b;
  when others -- all remainder cases: "10" and "11"
        out <='0';
End case;
```

- Remember to add always `when others` to complete the definition of the case statement
- Although there is a **for** loop statement in VHDL, its function is quite different from the usual 'for' loops of C-like programming languages. Its use in VHDL is limited and out of the scope of this subject.

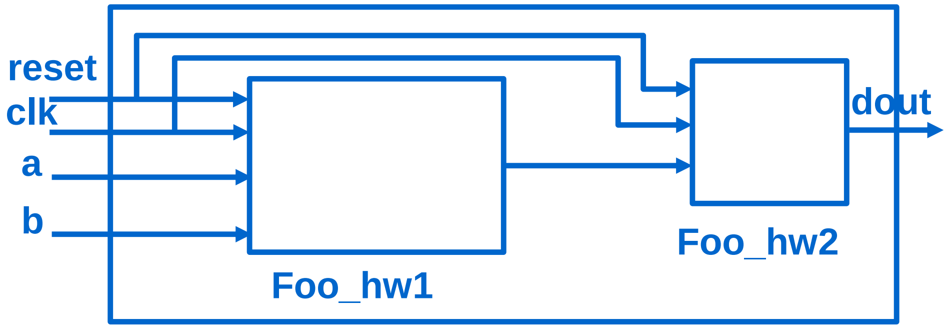
VHDL Essentials

Summary example

```
-- Bit multiplexer 2to1
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;    -- library (for logic operation)
ENTITY mux2to1 IS               -- entity circuit declaration
 PORT  (in_a:   IN  std_logic;  -- ports
        in_b:   IN  std_logic;
        sel:    IN  std_logic;
        output: OUT std_logic);
END mux2to1;
ARCHITECTURE Behavioral OF mux2to1 IS --architecture declaration
BEGIN
   PROCESS (in_a, in_b, sel)    -- sensitivity list (async signals)
      BEGIN
        IF (sel='1') THEN
          output<=in_a;
        ELSE
          output<=in_b;
        END IF;
   END PROCESS;                 -- end of process
END Behavioral;                 -- end of behavioral architecture
```

VHDL Essentials

Hierarchical design: behavioral

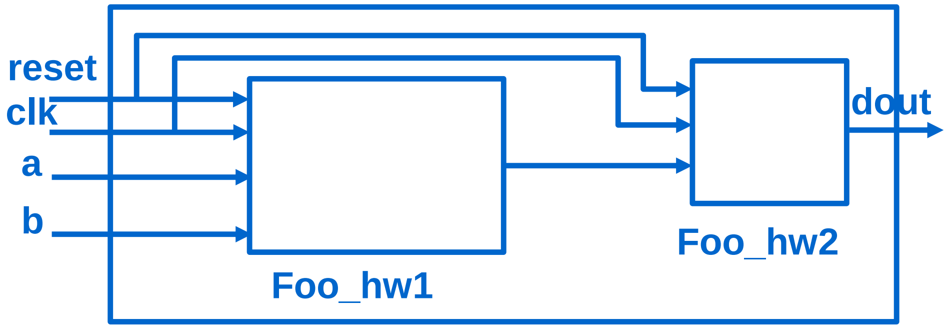


```vhdl
entity foo_behab is
    Port ( clk : in  STD_LOGIC;
           reset : in  STD_LOGIC;
           a : in  STD_LOGIC_VECTOR (7 downto 0);
           b : in  STD_LOGIC_VECTOR (7 downto 0);
           dout : out  STD_LOGIC);
end foo_behab;
```

```vhdl
architecture Behavioral of foo_behab is
  signal aux_signal: std_logic_vector (7 downto 0);
begin

  process (reset, clk)
  begin
    if (reset='1') then
      aux_signal <= (others=>'0');
    elsif (clk'event and clk='1') then
       if (a>b) then
         aux_signal <= a and b;
       else
         aux_signal <= a or b;
       end if;
    end if;
  end process;

  process (reset, clk)
  begin
    if (reset='1') then
      dout <= '0';
    elsif (clk'event and clk='1') then
      dout <= aux_signal(7) xor aux_signal(6)
          xor aux_signal(5) xor aux_signal(4)
          xor aux_signal(3) xor aux_signal(2)
          xor aux_signal(1) xor aux_signal(0);
    end if;
  end process;

end Behavioral;
```

VHDL Essentials

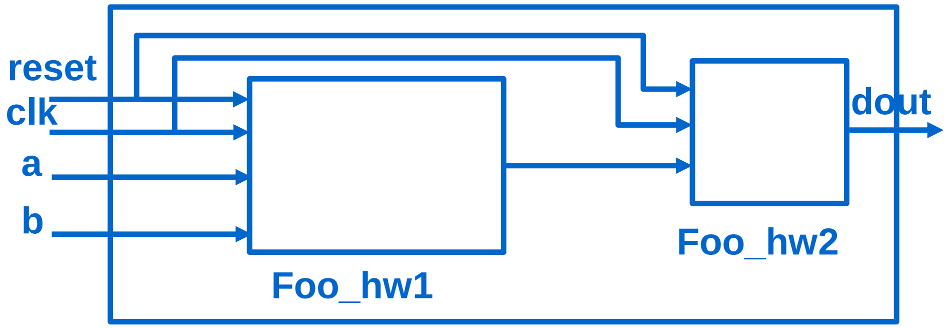
Hierarchical design: behavioral



```vhdl
entity foo_behab is
    Port ( clk : in  STD_LOGIC;
           reset : in  STD_LOGIC;
           a : in  STD_LOGIC_VECTOR (7 downto 0);
           b : in  STD_LOGIC_VECTOR (7 downto 0);
           dout : out  STD_LOGIC);
end foo_behab;
```

```vhdl
architecture Behavioral of foo_behab is
  signal aux_signal: std_logic_vector (7 downto 0);
begin

  process (reset, clk)
  begin
    if (reset='1') then
      aux_signal <= (others=>'0');
    elsif (clk'event and clk='1') then
       if (a>b) then
         aux_signal <= a and b;
       else
         aux_signal <= a or b;
       end if;
    end if;
  end process;

  process (reset, clk)
  begin
    if (reset='1') then
      dout <= '0';
    elsif (clk'event and clk='1') then
      dout <= aux_signal(7) xor aux_signal(6)
          xor aux_signal(5) xor aux_signal(4)
          xor aux_signal(3) xor aux_signal(2)
          xor aux_signal(1) xor aux_signal(0);
    end if;
  end process;

end Behavioral;
```

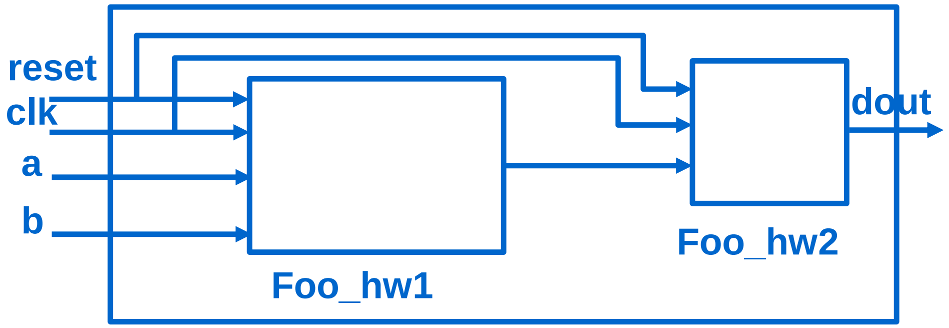
Hierarchical design: behavioral



```vhdl
entity foo_behab is
    Port ( clk : in  STD_LOGIC;
           reset : in  STD_LOGIC;
           a : in  STD_LOGIC_VECTOR (7 downto 0);
           b : in  STD_LOGIC_VECTOR (7 downto 0);
           dout : out  STD_LOGIC);
end foo_behab;
```

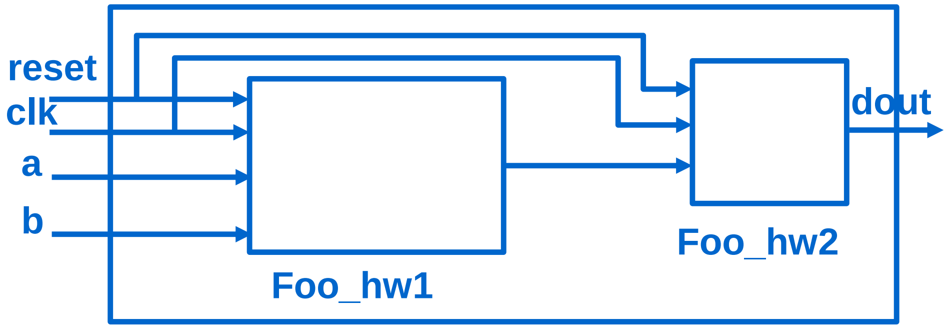
```vhdl
architecture Behavioral of foo_behab is
  signal aux_signal: std_logic_vector (7 downto 0);
begin

  process (reset, clk)
  begin
    if (reset='1') then
      aux_signal <= (others=>'0');
    elsif (clk'event and clk='1') then
       if (a>b) then
         aux_signal <= a and b;
       else
         aux_signal <= a or b;
       end if;
    end if;
  end process;

  process (reset, clk)
  begin
    if (reset='1') then
      dout <= '0';
    elsif (clk'event and clk='1') then
      dout <= aux_signal(7) xor aux_signal(6)
          xor aux_signal(5) xor aux_signal(4)
          xor aux_signal(3) xor aux_signal(2)
          xor aux_signal(1) xor aux_signal(0);
    end if;
  end process;

end Behavioral;
```

Hierarchical design: structural (i)



```
entity foo_hw1 is
  Port (  clk : in   STD_LOGIC;
        reset : in   STD_LOGIC;
             a: in   STD_LOGIC_VECTOR (7 downto 0);
            b : in   STD_LOGIC_VECTOR (7 downto 0);
         dout : out   STD_LOGIC_VECTOR (7 downto 0));
end foo_hw1;

architecture Behavioral of foo_hw1 is

begin

  process(reset, clk)
  begin
    if (reset='1') then
      dout <= (others=>'0');
    elsif (clk'event and clk='1') then
      if (a>b) then
        dout <= a and b;
      else
        dout <= a or b;
      end if;
    end if;
  end process;

end Behavioral;
```

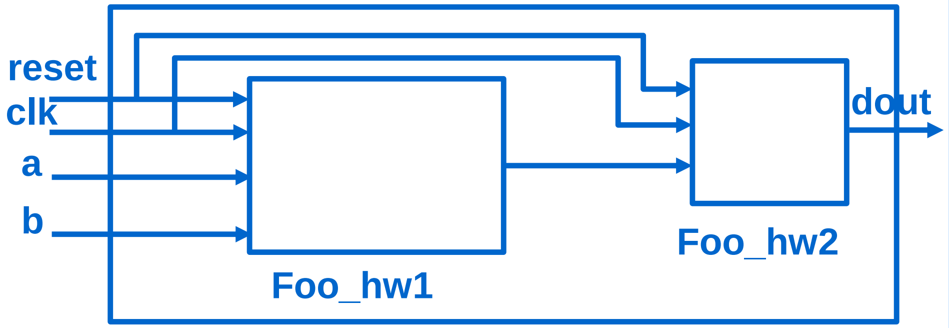
Hierarchical design: structural (ii)



```
entity foo_hw2 is
  Port (clk : in  STD_LOGIC;
      reset : in  STD_LOGIC;
          a : in  STD_LOGIC_VECTOR (7 downto 0);
       dout : out  STD_LOGIC);
end foo_hw2;

architecture Behavioral of foo_hw2 is

begin

  process (reset, clk)
    begin
      if (reset='1') then
        dout <= '0';
      elsif (clk'event and clk='1') then
        dout <= a(7) xor a(6) xor a(5) xor a(4)
           xor a(3) xor a(2) xor a(1) xor a(0);
      end if;
  end process;

end Behavioral;
```

VHDL Essentials

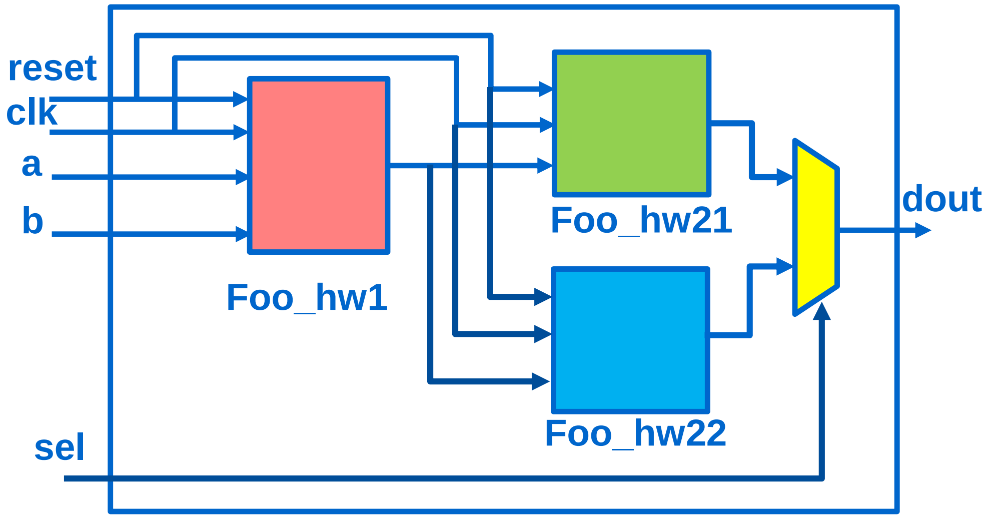
Hierarchical design: structural (iii)



```
architecture Behavioral of foo_struct is
  component foo_hw1 is
    Port ( clk : in  STD_LOGIC;
         reset : in  STD_LOGIC;
             a : in  STD_LOGIC_VECTOR (7 downto 0);
             b : in  STD_LOGIC_VECTOR (7 downto 0);
          dout : out  STD_LOGIC_VECTOR (7 downto 0));
  end component;

  component foo_hw2 is
    Port ( clk : in  STD_LOGIC;
         reset : in  STD_LOGIC;
             a : in  STD_LOGIC_VECTOR (7 downto 0);
          dout : out  STD_LOGIC);
  end component;
  signal aux_signal: std_logic_vector (7 downto 0);
begin
  comp1: foo_hw1 port map(
                    clk => clk,
                  reset => reset,
                      a => a,
                      b => b,
                   dout => aux_signal);

  comp2: foo_hw2 port map(
                    clk => clk,
                  reset => reset,
                      a => aux_signal,
                   dout => dout);
end Behavioral;
```

```
entity foo_behab is
    Port ( clk : in  STD_LOGIC;
           reset : in  STD_LOGIC;
           a : in  STD_LOGIC_VECTOR (7 downto 0);
           b : in  STD_LOGIC_VECTOR (7 downto 0);
           dout : out  STD_LOGIC);
end foo_behab;
```

Hierarchical design: mixed style



```
architecture Behavioral of foo_struct is
   ...

begin
  comp1: foo_hw1 port map(clk => clk,
                          reset => reset,
                          a => a,
                          b => b,
                          dout => aux_signal);

  comp2: foo_hw2 port map(clk => clk,
                          reset => reset,
                          a => aux_signal,
                          dout => aux1);

  comp3: foo_hw2 port map(clk => clk,
                          reset => reset,
                          a => aux_signal,
                          dout => aux2);

  process (aux1, aux2)
  begin
    if (sel='1') then
      dout <= aux2;
    else
      dout <= aux1;
    end if;
  end process;

end Behavioral;
```

```
entity foo_behab is
    Port ( clk : in  STD_LOGIC;
           reset : in  STD_LOGIC;
           a : in  STD_LOGIC_VECTOR (7 downto 0);
           b : in  STD_LOGIC_VECTOR (7 downto 0);
           sel : in  STD_LOGIC;
           dout : out  STD_LOGIC);
end foo_behab;
```