VHDL

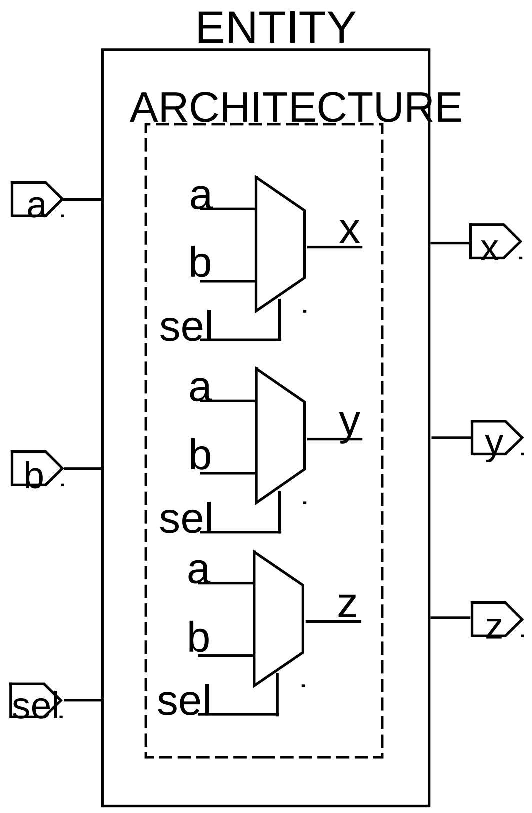
Zdeněk Matěj
72963@mail.muni.cz

VHDL

- **VHSIC (Very High Speed Integrated Circuit)**
- **Hardware**
- **Description**
- **Language**

Structure

- Construction blocks
  - Entity
    - Describes a module's interface
  - Architecture
    - Behavioral description of the block



Entity

- Describes interface of **latch** - it's inputs and outputs

```
entity latch is                -- Use double dash for a comment
port (   s,r: in bit;
         q,nq: out bit);
end latch;
```

- Input and output signals are marked with **in** and **out** (we can also use **inout**)

Architecture

- Represents a behavioral description's instance
- Generally, one entity may have multiple architectures

```
architecture dataflow of latch is  – archictecture's name is dataflow
begin
  q<=r nor nq;                    -- operator <= assigns signal
  nq<=s nor q;                    -- function „nor“ has been defined
end dataflow;
```

Component

- We use it for module's instantiation

```
entity latch is
  port (s,r: in bit;
        q,nq: out bit);
end latch;

architecture structure of latch is
component nor_gate
    port (a,b: in bit;
          c: out bit);              -- make sure you keep the ports' order:- in,in,out
  end component;
begin
  n1: nor_gate
    port map (r,nq,q);              -- component ports are assigned to given signals in the parent architecture
  n2: nor_gate
    port map (s,q,nq);
end structure;
```

Assignments

- VHDL is strongly typed and there are no automatic type casts like in C language
- Signal
  - a<=b and c;
- Variable
  - v:=a + 2;

Process

- It allows us to create a structured description of architecture

```
process (b,c)   -- process is executed whenever b or c changes
begin
    a<=b xor c;
end process;
```

Variables

- Used exclusively inside a **process** block

```
count: process (x)
  variable cnt : integer := -1;
begin
  cnt:=cnt + 1;
end process;
```

- Before first process execution, value of **cnt** will be initialized as **0**

Variables - cont.

- For data types, we can use **integer**, **bit**, etc. (see the documentation)
- However, for signal we use **STD_LOGIC** data type
- Example of vector definition: **std_logic_vector(0 to 17);**

Sequential expressions

- Can be created exclusively inside a **process**

```
count: process (x)
  variable cnt : integer :=0 ;
begin
  if (x='1' and x'last_value='0') then
    cnt:=cnt + 1;
  end if;
end process;
```

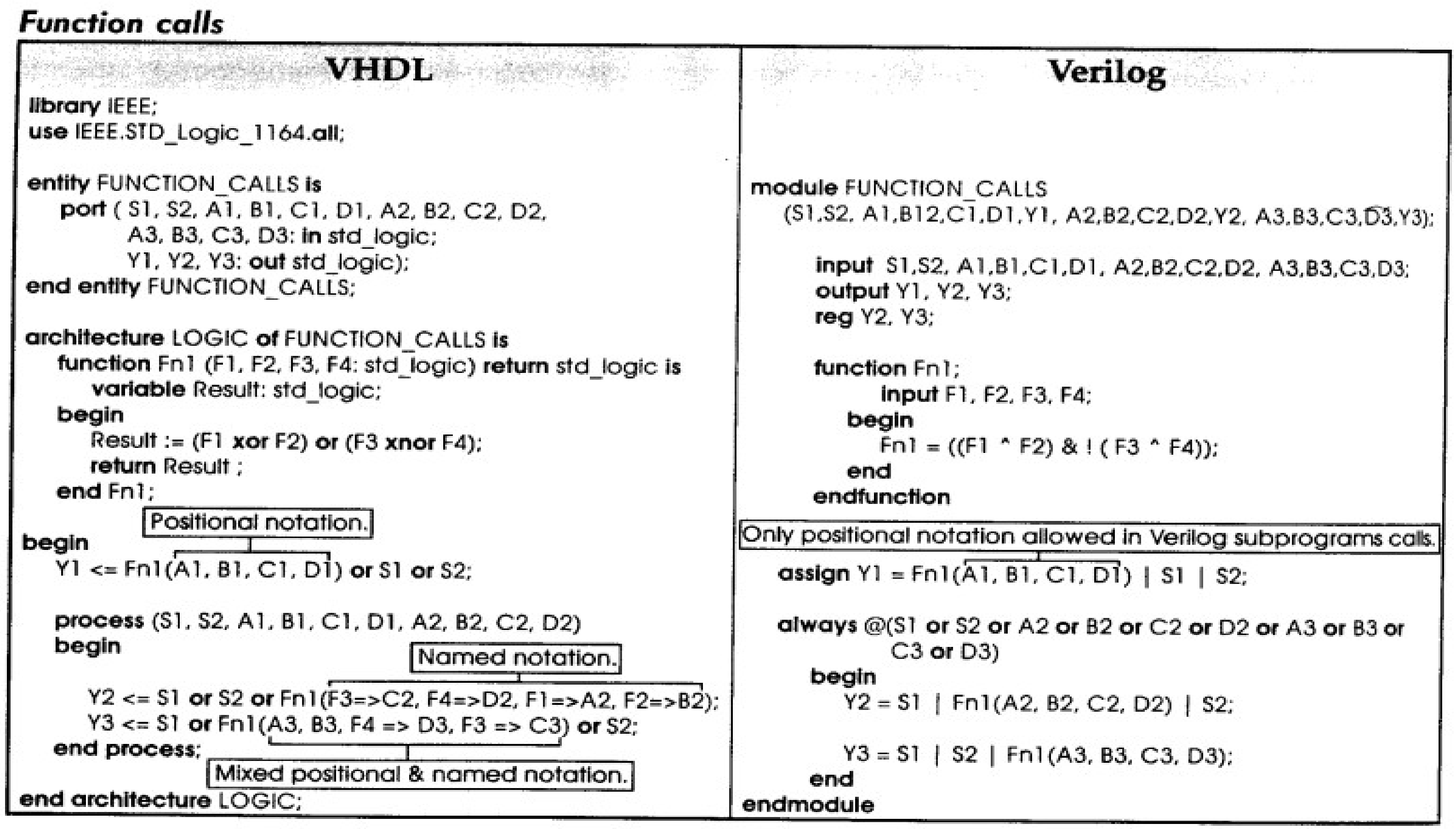
- **last_value** stands for latest value of variable **x** before invoking the process
- Another sequential constructions can be examinated in the documentation

Signals

- Signals are local for an architecture and can't be accessed from outside
- All assignments to signals are performed after the whole process body is finished

```
signal x,y,z : bit;
...
process (y)      -- change of y starts the process
begin
 x<=y;
end process;
```

Comparison of VHDL and Verilog



Function calls

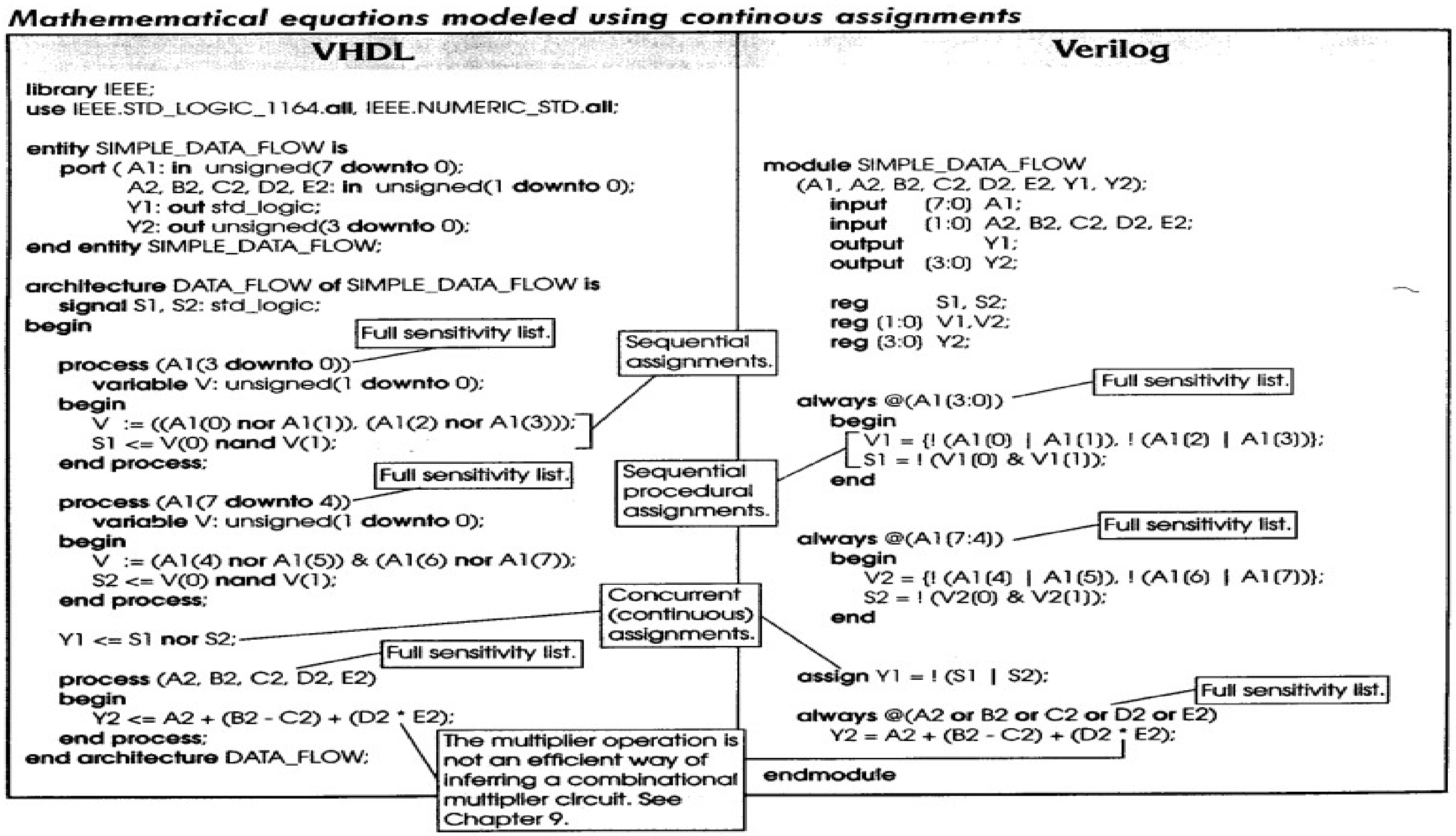
VHDL

```
library IEEE;
use IEEE.STD_Logic_1164.all;

entity FUNCTION_CALLS is
   port ( S1, S2, A1, B1, C1, D1, A2, B2, C2, D2,
          A3, B3, C3, D3: in std_logic;
          Y1, Y2, Y3: out std_logic);
end entity FUNCTION_CALLS;

architecture LOGIC of FUNCTION_CALLS is
   function Fn1 (F1, F2, F3, F4: std_logic) return std_logic is
      variable Result: std_logic;
   begin
      Result := (F1 xor F2) or (F3 xnor F4);
      return Result ;
   end Fn1;
                      Positional notation.
begin
   Y1 <= Fn1(A1, B1, C1, D1) or S1 or S2;

   process (S1, S2, A1, B1, C1, D1, A2, B2, C2, D2)
   begin                                    Named notation.

      Y2 <= S1 or S2 or Fn1(F3=>C2, F4=>D2, F1=>A2, F2=>B2);
      Y3 <= S1 or Fn1(A3, B3, F4 => D3, F3 => C3) or S2;
   end process;
                 Mixed positional & named notation.
end architecture LOGIC;
```

Verilog

```
module FUNCTION_CALLS
   (S1,S2, A1,B12,C1,D1,Y1, A2,B2,C2,D2,Y2, A3,B3,C3,D3,Y3);

      input  S1,S2, A1,B1,C1,D1, A2,B2,C2,D2, A3,B3,C3,D3;
      output Y1, Y2, Y3;
      reg Y2, Y3;

      function Fn1;
            input F1, F2, F3, F4;
         begin
            Fn1 = ((F1 ^ F2) & ! ( F3 ^ F4));
         end
      endfunction
Only positional notation allowed in Verilog subprograms calls.
   assign Y1 = Fn1(A1, B1, C1, D1) | S1 | S2;

   always @(S1 or S2 or A2 or B2 or C2 or D2 or A3 or B3 or
            C3 or D3)
      begin
         Y2 = S1 | Fn1(A2, B2, C2, D2) | S2;

         Y3 = S1 | S2 | Fn1(A3, B3, C3, D3);
      end
endmodule
```

Comparison of VHDL and Verilog



Mathemematical equations modeled using continous assignments

VHDL

```
library IEEE;
use IEEE.STD_LOGIC_1164.all, IEEE.NUMERIC_STD.all;

entity SIMPLE_DATA_FLOW is
    port ( A1: in  unsigned(7 downto 0);
           A2, B2, C2, D2, E2: in  unsigned(1 downto 0);
           Y1: out std_logic;
           Y2: out unsigned(3 downto 0));
end entity SIMPLE_DATA_FLOW;

architecture DATA_FLOW of SIMPLE_DATA_FLOW is
    signal S1, S2: std_logic;
begin

    process (A1(3 downto 0))
        variable V: unsigned(1 downto 0);
    begin
        V  := ((A1(0) nor A1(1)), (A1(2) nor A1(3)));
        S1 <= V(0) nand V(1);
    end process;

    process (A1(7 downto 4))
        variable V: unsigned(1 downto 0);
    begin
        V  := (A1(4) nor A1(5)) & (A1(6) nor A1(7));
        S2 <= V(0) nand V(1);
    end process;

    Y1 <= S1 nor S2;

    process (A2, B2, C2, D2, E2)
    begin
        Y2 <= A2 + (B2 - C2) + (D2 * E2);
    end process;
end architecture DATA_FLOW;
```

Verilog

```
module SIMPLE_DATA_FLOW
    (A1, A2, B2, C2, D2, E2, Y1, Y2);
        input    [7:0]  A1;
        input    [1:0]  A2, B2, C2, D2, E2;
        output          Y1;
        output   [3:0]  Y2;

        reg         S1, S2;
        reg [1:0]   V1,V2;
        reg [3:0]   Y2;

    always @(A1[3:0])
        begin
            V1 = {! (A1[0] | A1[1]), ! (A1[2] | A1[3])};
            S1 = ! (V1[0] & V1[1]);
        end

    always @(A1[7:4])
        begin
            V2 = {! (A1[4] | A1[5]), ! (A1[6] | A1[7])};
            S2 = ! (V2[0] & V2[1]);
        end

    assign Y1 = ! (S1 | S2);

    always @(A2 or B2 or C2 or D2 or E2)
        Y2 = A2 + (B2 - C2) + (D2 * E2);

endmodule
```

Full sensitivity list.

Sequential assignments.

Full sensitivity list.

Sequential procedural assignments.

Concurrent (continuous) assignments.

Full sensitivity list.

The multiplier operation is not an efficient way of inferring a combinational multiplier circuit. See Chapter 9.

Full sensitivity list.

Full sensitivity list.

Full sensitivity list.

Literature

- https://l202.fi.muni.cz/vyuka/pv200/materialy

- HDL chip design – Douglas J. Smith
- Altera: http://www.altera.com/support/examples/vhdl/vhdl.html
- http://esd.cs.ucr.edu/labs/tutorial/
- http://www.angelfire.com/in/rajesh52/verilogvhdl.html
- http://en.wikipedia.org/wiki/VHDL

Tasks

- Task no. 1
  - Describe 4-bit adder in VHDL, use LEDs and SWITCHes
- Task no. 2
  - Describe RS circuit in VHDL, use two KEYs and LED to indicate the current state.
- Task no. 3
  - Describe clock divider in VHDL in a separate module and use the module's instance in another module (e.g., in top module). Propagate divided clock signal to a LED.