Columbia University
Department of Electrical Engineering
EE E4332. VHDL examples.

Basic example

```
library ieee;
use ieee.std_logic_1164.all;
entity full_adder is
  port(a, b, cin: in std_ulogic;
       sum, cout: out std_ulogic);
end full_adder;
architecture dataflow of full_adder is
begin
  sum <= (a xor b) xor c;
  carry <= (a and b) or (a and c) or (b and c);
end dataflow;
```

Other concurrent VHDL statements

```
library ieee;
use ieee.std_logic_1164.all;
entity multiplexer_4_1 is
  port(in0, in1, in2, in3: in std_ulogic_vector(0 to 15);
       s0, s1: in std_ulogic;
       z: out std_ulogic_vector(0 to 15));
end multiplexer_4_1;
architecture dataflow of multiplexer_4_1 is
begin
  z <= in0 when (s0 = '0' and s1 = '1') else
       in1 when (s0 = '1' and s1 = '0') else
       in2 when (s0 = '0' and s1 = '1') else
       in3 when (s0 = '1' and s1 = '1') else
       "XXXXXXXXXXXXXXXX";
end dataflow;
```

Here's another way of doing the multiplexer

```
library ieee;
use ieee.std_logic_1164.all;
entity multiplexer_4_1 is
  port(in0, in1, in2, in3: in std_ulogic_vector(0 to 15);
       s0, s1: in std_ulogic;
       z: out std_ulogic_vector(0 to 15));
end multiplexer_4_1;
architecture dataflow of multiplexer_4_1 is
signal sels: std_ulogic_vector(0 to 1);
begin
 sels <= s0 & s1;
 with sels select
   z <= in0 when "00",
        in1 when "01",
        in2 when "10",
        in3 when "11",
        "XXXXXXXXXXXXXXXX" when others;
end dataflow;
```

But maybe the selects cannot both be 1 or 0

```
library ieee;
use ieee.std_logic_1164.all;
entity multiplexer_4_1 is
  port(in0, in1: in std_ulogic_vector(0 to 15);
       s0, s1: in std_ulogic;
       z: out std_ulogic_vector(0 to 15));
end multiplexer_4_1;
architecture dataflow of multiplexer_4_1 is
signal sels: std_ulogic_vector(0 to 1);
begin
 sels <= s0 & s1;
 with sels select
   z <= in0 when "10",
        in1 when "01",
        "XXXXXXXXXXXXXXXX" when others;
 assert not(sels = "00" or sels = "11")
```

```
   report "The select signals must be orthogonal!"
   severity ERROR;
end dataflow;
```

Properties of signal assignment: delta delay and tristate

```
library ieee;
use ieee.std_logic_1164.all;
entity part1 is
  port(a, sel: in std_ulogic;
       z: out std_logic);
end part1;
architecture dataflow of part1 is
signal b, c: std_ulogic;
begin
  z <= not c when (sel = '0') else 'Z';
  c <= not b;
  b <= not a;
  z <= a when (sel = '1') else 'Z';
end dataflow;
```

4-bit adder with concurrent VHDL

```
library ieee;
use ieee.std_logic_1164.all;
entity adder is
  port(a, b: in std_ulogic_vector(0 to 3);
       cin: in std_ulogic;
       c: out std_ulogic_vector(0 to 3);
       cout: out std_ulogic);
end adder;
architecture dataflow of adder is
signal car: std_ulogic_vector(0 to 3);
begin
  car(0) <= cin;
  G1: for m in 3 downto 0 generate
    sum(m) <= a(m) xor b(m) xor car(m);
```

```
    car(m+1) <= (a(m) and b(m)) or (b(m) and car(m)) or (a(m) and car(m));
  end generate G1;
  cout <= c(4);
end dataflow;
```

4-bit adder using the std_logic_arith package

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
entity adder is
  port(a, b: in std_ulogic_vector(0 to 3);
       cin: in std_ulogic;
       c: out std_ulogic_vector(0 to 3);
       cout: out std_ulogic);
end adder;
architecture dataflow of adder is
signal temp_sum: std_ulogic_vector(0 to 4);
begin
   temp_sum <= (a + b) + cin;
   cout <= temp_sum(0);
   c <= temp_sum(1 to 4);
end adder;
```

Sequential VHDL

```
library ieee;
use ieee.std_logic_1164.all;
entity state_machine is
  port(a, clk: in std_ulogic;
       z: out std_ulogic);
end state_machine;
architecture dataflow of state_machine is
  signal p_state, n_state: std_ulogic_vector(0 to 2);

  synch: process(clk)
  begin
```

```
  if (clk = '0' and not(clk'stable)) then
    p_state <= n_state;
  end if;
end process;

comb: process(p_state, a)
begin
  case p_state is
    when "00" =>
       if (w = '1') then
          z <= '1'; n_state <= "11";
       else
          z <= '0';
       end if;
    when "01" =>
       if (a = '1') then
          z <= '0'; n_state <= "00";
       else
          z <= '1';
       end if;
    when "10" =>
       if (a = '0') then
          z <= '0';
       else
          z <= '1'; n_state <= "01";
       end if;
    when "11" =>
       z <= '0';
       if (a = '0') then
          n_state <= "10";
       else
          n_state <= "01";
       end if;
    when others =>
       n_state <= "XX";
       Z <= 'X';
  end case;
```

```
  end process;
end dataflow;
```

You can make a latch you don't intend if you aren't careful with processes

```
library ieee;
use ieee.std_logic_1164.all;
entity is_latch is
  port(a: in std_ulogic;
       b: out std_ulogic);
end is_latch;
architecture dataflow of is_latch is
begin
  process(a)
  begin
    if (a = '0') then
      b <= '1';
    end if;
  end process;
end dataflow;
```

There is a difference between signals and variables in processes

```
entity what_is_b is
end what_is_b;
architecture dataflow of what_is_b is
signal a, b: std_ulogic;
begin
 process
 begin
  a <= '1';
  if (a = '1') then
    b <= '1';
  else
    b <= '0';
  end if;
 end process;
end dataflow;'
```

```
entity what_is_b is
end what_is_b;
architecture dataflow of what_is_b is
signal b: std_ulogic;
begin
 process
 variable a: std_ulogic;
 begin
  a := '1';
  if (a = '1') then
    b <= '1';
  else
    b <= '0';
  end if;
 end process;
end dataflow;'
```

Modelling an SRAM. Use of variables.

```
library ieee;
use ieee.std_logic_1164.all;
entity sram_2168 is
  port( io: inout std_logic_vector(0 to 3);
        addr: in std_ulogic_vector(0 to 11);
        ce_n: in std_ulogic;
        we_n: in std_ulogic);
end sram_2168;
architecture dataflow of sram_2168 is
begin
  memory: process(addr, ce_n, we_n)
  type sram_array_word is std_ulogic_vector(0 to 3);
  variable sram_array: array(0 to 4096) of sram_array_word;
  begin
    if (ce_n = '0') then
      if (we_n = '1') then -- read the memory
        io <= sram_array(to_integer(addr));
      else
```

```
        sram_array(to_integer(addr)) := io;
      end if;
    else
      io <= "ZZZZ";
    end if;
  end process;
end dataflow;
```

Full structural example

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
ENTITY incrementer IS
    PORT(
        incr_out : OUT std_ulogic_vector(0 TO 15);
        incr_in : IN std_ulogic_vector(0 TO 15)
    );
END incrementer;
architecture behavior of incrementer is
  begin
    incr_out <= incr_in + '1';
  end behavior;
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
ENTITY multiplexer IS
    PORT(
        mux_out : OUT std_ulogic_vector(0 TO 15);
        in0 : IN std_ulogic_vector(0 TO 15);
        in1 : IN std_ulogic_vector(0 TO 15);
        select_in : IN std_ulogic
    );
END multiplexer;
architecture behavior of multiplexer is
  begin
    process(select_in, in0, in1)
```

```
      begin
        if (select_in = '0') then
          mux_out <= in0;
        elsif (select_in = '1') then
          mux_out <= in1;
        else
          mux_out <= "XXXXXXXXXXXXXXXX";
        end if;
     end process;
end behavior;
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY register16 IS
    PORT(
        reg_out : OUT std_ulogic_vector(0 TO 15);
        clock : IN std_ulogic;
        reg_in : IN std_ulogic_vector(0 TO 15)
    );
END register16;
architecture behavior of register16 is
begin
  SYNCH: process(clock)
  begin
    if (clock = '1' and not(clock'stable)) then
      reg_out(0 to 15) <= reg_in(0 to 15);
    end if;
  end process;
end behavior ;
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY top_level IS
    PORT(
        load_start : IN std_ulogic;
        clock : IN std_ulogic;
        start : IN std_ulogic_vector(0 TO 15)
    );
END top_level;
```

```
LIBRARY ieee, tutorial2;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
ARCHITECTURE schematic OF top_level IS
    COMPONENT incrementer
        PORT(
            incr_out : OUT std_ulogic_vector(0 TO 15);
            incr_in : IN std_ulogic_vector(0 TO 15)
        );
    END COMPONENT;

    COMPONENT register16
        PORT(
            reg_out : OUT std_ulogic_vector(0 TO 15);
            clock : IN std_ulogic;
            reg_in : IN std_ulogic_vector(0 TO 15)
        );
    END COMPONENT;

    COMPONENT multiplexer
        PORT(
            mux_out : OUT std_ulogic_vector(0 TO 15);
            in0 : IN std_ulogic_vector(0 TO 15);
            in1 : IN std_ulogic_vector(0 TO 15);
            select_in : IN std_ulogic
        );
    END COMPONENT;

    SIGNAL net12 : std_ulogic_vector(0 TO 15);
    SIGNAL net11 : std_ulogic_vector(0 TO 15);
    SIGNAL net8 : std_ulogic_vector(0 TO 15);

    FOR ALL: incrementer USE ENTITY tutorial2.incrementer(behavior);
    FOR ALL: register16 USE ENTITY tutorial2.register16(behavior);
    FOR ALL: multiplexer USE ENTITY tutorial2.multiplexer(behavior);
```

```
BEGIN


    \I3\ : incrementer
        PORT MAP(
            incr_in(0 TO 15) => net8(0 TO 15),
            incr_out(0 TO 15) => net11(0 TO 15)
        );

    \I2\ : register16
        PORT MAP(
            reg_in(0 TO 15) => net12(0 TO 15),
            clock => clock,
            reg_out(0 TO 15) => net8(0 TO 15)
        );

    \I1\ : multiplexer
        PORT MAP(
            mux_out(0 TO 15) => net12(0 TO 15),
            select_in => load_start,
            in1(0 TO 15) => start(0 TO 15),
            in0(0 TO 15) => net11(0 TO 15)
        );

END schematic;
LIBRARY ieee, tutorial2;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
ARCHITECTURE stimulus OF test IS

    COMPONENT top_level
        PORT(
            load_start : IN std_ulogic;
            clock : IN std_ulogic;
            start : IN std_ulogic_vector(0 TO 15)
        );
    END COMPONENT;
```

```
    -- Fill in values for each generic

    -- Fill in values for each signal
    SIGNAL load_start : std_ulogic := '1';
    SIGNAL clock : std_ulogic := '0';
    SIGNAL start : std_ulogic_vector(0 TO 15) := "0000000000000000";

    FOR ALL: top_level use ENTITY tutorial2.top_level(schematic);

BEGIN

    dut : top_level

        PORT MAP (load_start => load_start,
                  clock => clock,
                  start => start);

    clock <= not(clock) after 10 ns;
    load_start <= '0' after 15 ns;

END stimulus;
```