VHDL = VHSIC Hardware Description Language

VHSIC = Very High Speed Integrated Circuit

Hardware description, simulation, and synthesis

Describes hardware at different levels: behavioral, logic equation, structural

Top-down design methodology

Technology Independent
Gate Network

Concurrent Statements

\[ C \leq A \text{ and } B; \]
\[ E \leq C \text{ or } D; \]

If delay is not specified, “delta” delay is assumed

\[ C \leq A \text{ and } B; \]
\[ E \leq C \text{ or } D; \]

Order of concurrent statements is not important

\[ E \leq C \text{ or } D; \]
\[ C \leq A \text{ and } B; \]

This statement executes repeatedly

\[ \text{CLK} \leq \text{not} \ \text{CLK} \text{ after} \ 10 \ \text{ns}; \]

This statement causes a simulation error

\[ \text{CLK} \leq \text{not} \ \text{CLK}; \]
entity FullAdder is
  port (X, Y, Cin: in std_logic; -- Inputs
         Cout, Sum: out std_logic); -- Outputs
end FullAdder;

architecture Equations of FullAdder is
begin
  -- Concurrent Assignments
  Sum  <= X xor Y xor Cin;
  Cout <= (X and Y) or (X and Cin) or (Y and Cin);  
end Equations;
VHDL Program Structure

**entity** entity-name **is**
  **port**(interface-signal-declaration);]
**end** [entity] [entity-name];

**architecture** architecture-name **of** entity-name **is**
  [declarations]
**begin**
  architecture body
**end** [architecture] [architecture-name];
4-bit Binary Adder

**Structural Description of 4-bit Adder**

```vhdl
entity Adder4 is
  port (A, B: in std_logic_vector(3 downto 0); Ci: in std_logic; -- Inputs
       S: out std_logic_vector(3 downto 0); Co: out std_logic);  -- Outputs
end Adder4;
```
4-bit Binary Adder

![Diagram of 4-bit Binary Adder]

**Structural Description of 4-bit Adder**

```vhdl
architecture Structure of Adder4 is
component FullAdder
    port (X, Y, Cin: in std_logic; -- Inputs
          Cout, Sum: out std_logic); -- Outputs
end component;
signal C: std_logic_vector(3 downto 1);
begin
    -- instantiate four copies of the FullAdder
    FA0: FullAdder port map (A(0), B(0), Ci, C(1), S(0));
    FA1: FullAdder port map (A(1), B(1), C(1), C(2), S(1));
    FA2: FullAdder port map (A(2), B(2), C(2), C(3), S(2));
    FA3: FullAdder port map (A(3), B(3), C(3), Co, S(3));
end Structure;
```
Simulation of the 4-bit Adder

list A B Co C Ci S  -- put these signals on the output list
force A 1111  -- set the A inputs to 1111
force B 0001  -- set the B inputs to 0001
force Ci 1  -- set the Ci to 1
run 50  -- run the simulation for 50 ns

force Ci 0
force A 0101
force B 1110
run 50
[10 ns delay for each adder]

<table>
<thead>
<tr>
<th>ns</th>
<th>delta</th>
<th>a</th>
<th>b</th>
<th>co</th>
<th>c</th>
<th>ci</th>
<th>s</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>+0</td>
<td>0000</td>
<td>0000</td>
<td>0</td>
<td>000</td>
<td>0</td>
<td>0000</td>
</tr>
<tr>
<td>0</td>
<td>+1</td>
<td>1111</td>
<td>0001</td>
<td>0</td>
<td>000</td>
<td>1</td>
<td>0000</td>
</tr>
<tr>
<td>10</td>
<td>+0</td>
<td>1111</td>
<td>0001</td>
<td>0</td>
<td>001</td>
<td>1</td>
<td>1111</td>
</tr>
<tr>
<td>20</td>
<td>+0</td>
<td>1111</td>
<td>0001</td>
<td>0</td>
<td>011</td>
<td>1</td>
<td>1101</td>
</tr>
<tr>
<td>30</td>
<td>+0</td>
<td>1111</td>
<td>0001</td>
<td>0</td>
<td>111</td>
<td>1</td>
<td>1001</td>
</tr>
<tr>
<td>40</td>
<td>+0</td>
<td>1111</td>
<td>0001</td>
<td>1</td>
<td>111</td>
<td>1</td>
<td>0001</td>
</tr>
<tr>
<td>50</td>
<td>+0</td>
<td>0101</td>
<td>1110</td>
<td>1</td>
<td>111</td>
<td>0</td>
<td>0001</td>
</tr>
<tr>
<td>60</td>
<td>+0</td>
<td>0101</td>
<td>1110</td>
<td>1</td>
<td>110</td>
<td>0</td>
<td>0101</td>
</tr>
<tr>
<td>70</td>
<td>+0</td>
<td>0101</td>
<td>1110</td>
<td>1</td>
<td>100</td>
<td>0</td>
<td>0111</td>
</tr>
<tr>
<td>80</td>
<td>+0</td>
<td>0101</td>
<td>1110</td>
<td>1</td>
<td>100</td>
<td>0</td>
<td>0011</td>
</tr>
</tbody>
</table>
VHDL Processes

General form of Process
process (sensitivity-list)
begin
  sequential-statements
end process;

Process example
process (B, C, D)
begin
  A <= B; -- statement 1
  B <= C; -- statement 2
  C <= D; -- statement 3
end process;

Simulation results
<table>
<thead>
<tr>
<th>time</th>
<th>delta</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>+0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>+0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>10</td>
<td>+1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>10</td>
<td>+2</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>10</td>
<td>+3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

Concurrent Statements

<table>
<thead>
<tr>
<th>A &lt;= B</th>
<th>B &lt;= C</th>
<th>C &lt;= D</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>+0</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>+0</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>+1</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>+2</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>+3</td>
<td>1</td>
</tr>
</tbody>
</table>

Simulation Results
<table>
<thead>
<tr>
<th>time</th>
<th>delta</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>+0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>+0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>10</td>
<td>+1</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>10</td>
<td>+2</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>10</td>
<td>+3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>
D Flip-flop Model

entity DFF is
  port (D, CLK: in std_logic;
        Q: out std_logic;  QN: out std_logic := '1');
-- initialize QN to '1' since bit signals are initialized to '0' by default
end DFF;

architecture SIMPLE of DFF is
begin
  process (CLK)  -- process is executed when CLK changes
  begin
    if CLK = '1' then -- rising edge of clock
      Q  <= D after 10 ns;
      QN  <= not D after 10 ns;
    end if;
  end process;
end SIMPLE;
entity JKFF is
  port (S_N, R_N, J, K, CLK: in std_logic;
    Q: out std_logic := '0';
    Q_N: out std_logic := '1');
end JKFF;

architecture JKFF1 of JKFF is
signal Q_{int}: std_logic;
beg
process (S_N, R_N, CLK)
begin
if \( R_N = '0' \) then \( Q_{\text{int}} <= '0'; \)
    elsif \( S_N = '0' \) then \( Q_{\text{int}} <= '1'; \)
    elsif \( \text{CLK} = '0' \) and \( \text{CLK}'\text{event} \) then
        \( Q_{\text{int}} <= (J \text{ and not } Q_{\text{int}}) \) or (not \( K \) and \( Q_{\text{int}} \));
    end if;
end if;
end process;
Q <= \( Q_{\text{int}} \);
\( Q_N <= \) not \( Q_{\text{int}} \);
end JKFF1;
Equivalent Representations of a Flowchart Using Nested Ifs and Elsifs

if (C1) then S1; S2;
else if (C2) then S3; S4;
else if (C3) then S5; S6;
else S7; S8;
end if;
end if;
4-to-1 Multiplexer

MUX model using a *conditional signal assignment statement*: (Sel = A&B)

\[
F <= \begin{cases} 
I0 & \text{when } Sel = "00" \\
I1 & \text{when } Sel = "01" \\
I2 & \text{when } Sel = "10" \\
I3 & \text{else}
\end{cases}
\]

(In the above concurrent statement, Sel can also be represented as the integer equivalent of a 2-bit binary number with bits A and B.)

General form of conditional signal assignment statement:

\[
\text{signal}_\text{name} <= \text{expression1 when condition1} \\
\quad \text{else expression2 when condition2} \\
\quad \ldots \\
\quad \text{else expressionN};
\]


**Multiplexer Example**

If a MUX model is used *inside a process, a concurrent statement cannot* be used. As an alternative, the MUX can be modeled using a **case statement**:

```vhdl
case Sel is
  when 0 => F <= I0;
  when 1 => F <= I1;
  when 2 => F <= I2;
  when 3 => F <= I3;
end case;
```

The case statement has the general form:

```vhdl
case expression is
  when choice1 => sequential statements1
  when choice2 => sequential statements2
  . . .
  [when others => sequential statements]
end case;
```

```
sel <= A&B; -- concatenate A and B
with sel select
F <= I0 when "00",
  <= I1 when "01",
  <= I2 when "10",
  <= I3 when "11",
```
Compilation, Elaboration, and Simulation of VHDL Code

- VHDL Source Code
  - Compiler (Analyzer)
    - Intermediate Code
    - Working library
  - Resource Libraries
  - Elaborator
    - Simulation Data Structure
  - Simulator
    - Simulator Commands
    - Simulator Output

- Code
- Code Data
- Structure
- Synthesize
- Components & Connections
Behavioral Model of State Machine

entity SM1_2 is
  port(X, CLK: in std_logic; Z: out std_logic);
end SM1_2;

architecture Table of SM1_2 is
  signal State, Nextstate: integer := 0;
begin
  process(State,X) --Combinational Network
  begin
    case State is
      when 0 =>
        if X='0' then Z<='1'; Nextstate<=1; end if;
        if X='1' then Z<='0'; Nextstate<=2; end if;
      when 1 =>
        if X='0' then Z<='1'; Nextstate<=3; end if;
        if X='1' then Z<='0'; Nextstate<=4; end if;
      when 2 =>
        if X='0' then Z<='0'; Nextstate<=4; end if;
        if X='1' then Z<='1'; Nextstate<=4; end if;
      when 3 =>
        if X='0' then Z<='0'; Nextstate<=5; end if;
        if X='1' then Z<='1'; Nextstate<=5; end if;
      when 4 =>
        if X='0' then Z<='1'; Nextstate<=5; end if;
        if X='1' then Z<='0'; Nextstate<=6; end if;
    end case;
end process;
end Table;
Behavioral model

when 5 =>
  if X='0' then Z<='0'; Nextstate<=0; end if;
  if X='1' then Z<='1'; Nextstate<=0; end if;
when 6 =>
  if X='0' then Z<='1'; Nextstate<=0; end if;
when others => null; -- should not occur
end case;
end process;

process(CLK) -- State Register
begin
  if CLK='1' then -- rising edge of clock
    State <= Nextstate;
  end if;
end process;
end Table;
A simulator command file that can be used to test as follows:

```
wave CLK X State NextState Z
force CLK 0 0, 1 100 -repeat 200
force X 0 0, 1 350, 0 550, 1 750, 0 950, 1 1350
run 1600
```

Execution of the preceding command file produces the waveforms shown in Figure 2-14.

**Waveforms**
Sequential Machine Model Using Equations

-- The following is a description of the sequential machine of
-- Figure 1-17 in terms of its next state equations.
-- The following state assignment was used:
-- S0-->0; S1-->4; S2-->5; S3-->7; S4-->6; S5-->3; S6-->2

entity SM1_2 is
  port(X,CLK: in std_logic;
       Z: out std_logic);
end SM1_2;

architecture Equations1_4 of SM1_2 is
  signal Q1,Q2,Q3: std_logic;
begin
  process(CLK)
  begin
    if CLK='1' then -- rising edge of clock
      Q1<=not Q2 after 10 ns;
      Q2<=Q1 after 10 ns;
      Q3<=(Q1 and Q2 and Q3) or (not X and Q1 and not Q3) or
      (X and not Q1 and not Q2) after 10 ns;
    end if;
    end process;
    Z<=(not X and not Q3) or (X and Q3) after 20 ns;
  end Equations1_4;
**Structural Model of Sequential Machine**

-- The following is a STRUCTURAL VHDL description of the network of Figure 1-20.

library IEEE;
use IEEE.std_logic_1164.all;

entity SM1_2 is
  port(X,CLK: in std_logic;
      Z: out std_logic);
end SM1_2;

architecture Structure of SM1_2 is
  signal A1,A2,A3,A5,A6,D3: std_logic:='0';
  signal Q1,Q2,Q3: std_logic:='0';
  signal Q1N,Q2N,Q3N, XN: std_logic:='1';
begin
  I1: Inverter port map (X,XN);
  G1: Nand3 port map (Q1,Q2,Q3,A1);
  G2: Nand3 port map (Q1,Q3N,XN,A2);
  G3: Nand3 port map (X,Q1N,Q2N,A3);
  G4: Nand3 port map (A1,A2,A3,D3);
  FF1: DFF port map (Q2N,CLK,Q1,Q1N);
  FF2: DFF port map (Q1,CLK,Q2,Q2N);
  FF3: DFF port map (D3,CLK,Q3,Q3N);
  G5: Nand2 port map (X,Q3,A5);
  G6: Nand2 port map (XN,Q3N,A6);
  G7: Nand2 port map (A5,A6,Z);
end Structure
Behavioral Model Using a Single Process

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

entity SM1_2 is port(X, CLK: in std_logic; Z: out std_logic); end SM1_2;
architecture Table of SM1_2 is signal State, Nextstate: integer := 0;
begin
  process (no sensitivity list)
  begin
      case State is
      when 0 =>
          if X='0' then Z<='1'; Nextstate<=1; end if;
          if X='1' then Z<='0'; Nextstate<=2; end if;
      when 1 =>
          if X='0' then Z<='1'; Nextstate<=3; end if;
          if X='1' then Z<='0'; Nextstate<=4; end if;
      when 2 =>
          if X='0' then Z<='0'; Nextstate<=4; end if;
          if X='1' then Z<='1'; Nextstate<=4; end if;
      when 3 =>
          if X='0' then Z<='0'; Nextstate<=5; end if;
          if X='1' then Z<='1'; Nextstate<=5; end if;
      when 4 =>
          if X='0' then Z<='1'; Nextstate<=5; end if;
          if X='1' then Z<='0'; Nextstate<=6; end if;
      when 5 =>
          if X='0' then Z<='0'; Nextstate<=0; end if;
          if X='1' then Z<='1'; Nextstate<=0; end if;
      end case;
  end process;
end;
```
Behavioral Model Using a Single Process

when 6 =>
    if X='0' then Z<='1'; Nextstate<='0'; end if;
    when others => null; -- should not occur
end case;

wait on CLK; <-- observe here
if rising_edge(CLK) then -- rising_edge function is in BITLIB *
    State <= Nextstate;
    wait for 0 ns; -- wait for State to be updated
end if;
end process;
end table;

* Alternative:

if CLK'event and CLK = '1' then
74163 Counter

Control Signals

<table>
<thead>
<tr>
<th>ClrN</th>
<th>LdN</th>
<th>P•T</th>
<th>Q3⁺</th>
<th>Q2⁺</th>
<th>Q1⁺</th>
<th>Q0⁺</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>X</td>
<td>D3</td>
<td>D2</td>
<td>D1</td>
<td>D0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>Q3</td>
<td>Q2</td>
<td>Q1</td>
<td>Q0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>present state + 1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Next State

- (clear)
- (parallel load)
- (no change)
- (increment count)
library IEEE; -- contains int2vec and vec2int functions
use IEEE.std_logic_1164.all;

entity c74163 is
  port (LdN, ClrN, P, T, CK: in std_logic; D: in std_logic_vector(3 downto 0);
    Cout: out std_logic; Q: out std_logic_vector(3 downto 0) );
end c74163;

architecture b74163 of c74163 is
  signal Qint: std_logic_vector(3 downto 0);
begin
  process
  begin
    wait until CK = '1'; -- change state on rising edge
    if ClrN = '0' then  Qint <= "0000";
      elsif LdN = '0' then Qint <= D;
      elsif (P and T) = '1' then
        Qint <= int2vec(vec2int(Qint)+1,4);
      end if;
  end process;
  Q <= Qint;
  Cout <= Q(3) and Q(2) and Q(1) and Q(0) and T;
end b74163;
Two 74163 Counters Cascaded to Form an 8-bit Counter
library IEEE;
use IEEE.std_logic_1164.all;

entity c74163test is
  port(ClrN,LdN,P,T1,Clk: in std_logic;
       Din1, Din2: in std_logic_vector(3 downto 0);
       Qout1, Qout2: out std_logic_vector(3 downto 0);
       Carry2: out std_logic);
end c74163test;

architecture tester of c74163test is
  component c74163
    port(LdN,ClrN, P, T, CK: in std_logic;  D: in std_logic_vector(3 downto 0);
         Cout: out std_logic; Q: out std_logic_vector(3 downto 0) );
  end component;
  signal Carry1: std_logic;
begin
  ct1: c74163 port map (LdN,ClrN,P,T1,Clk,Din1,Carry1,Qout1);
  ct2: c74163 port map (LdN,ClrN,P,Carry1,Clk,Din2,Carry2,Qout2);
end tester;