Motivation for VHDL

• Technology trends
  − 1 billion transistor chip running at 20 GHz in 2007
• Need for Hardware Description Languages
  − Systems become more complex
  − Design at the gate and flip-flop level becomes very tedious and time consuming
• HDLs allow
  − Design and debugging at a higher level before conversion to the gate and flip-flop level
  − Tools for synthesis do the conversion
• VHDL, Verilog
• VHDL – VHSIC Hardware Description Language
Facts About VHDL

- Developed originally by DARPA
  - for specifying digital systems
- International IEEE standard (IEEE 1076-1993)
- Hardware Description, Simulation, Synthesis
- Provides a mechanism for digital design and reusable design documentation
- Support different description levels
  - Structural (specifying interconnections of the gates),
  - Dataflow (specifying logic equations), and
  - Behavioral (specifying behavior)
- Top-down, Technology Independent

2.1 VHDL Description of Combinational Networks – The Basics

Concurrent Statements

\[
C \leftarrow A \text{ and } B \text{ after } 5 \text{ ns}; \\
E \leftarrow C \text{ or } D \text{ after } 5 \text{ ns};
\]

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

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

Order of concurrent statements is not important

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

This statement executes repeatedly

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

This statement causes a simulation error

\[
\text{CLK} \leftarrow \text{not CLK};
\]
2.1 VHDL Description of Combinational Networks – Entity Architecture Pair

Full Adder Example

```
entity FullAdder is
  port (X, Y, Cin: in bit; -- Inputs
       Cout, Sum: out bit); -- Outputs
end FullAdder;

architecture Equations of FullAdder is
begin -- Concurrent Assignments
  Sum <= X xor Y xor Cin after 10 ns;
  Cout <= (X and Y) or (X and Cin) or (Y and Cin) after 10 ns;
end Equations;
```

2.1 VHDL Description of Combinational Networks – Hierarchy of VHDL Models

```
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];
```
Chapter 2  CPE/EE 422/522

2.1 VHDL Description of Combinational Networks – Adder4

entity Adder4 is
  port (A, B: in bit_vector(3 downto 0); Ci: in bit;         -- Inputs
    S: out bit_vector(3 downto 0); Co: out bit);        -- Outputs
end Adder4;

architecture Structure of Adder4 is
component FullAdder
  port (X, Y, Cin: in bit;                        -- Inputs
    Cout, Sum: out bit);                       -- Outputs
end component;
signal C: bit_vector(3 downto 1);
begnin -- 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;
2.1 VHDL Description of Combinational Networks - Providing Stimuli

list A B 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

<table>
<thead>
<tr>
<th>ns</th>
<th>delta</th>
<th>a</th>
<th>b</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>0</td>
<td>0000</td>
</tr>
<tr>
<td>0</td>
<td>+1</td>
<td>1111</td>
<td>0001</td>
<td>0</td>
<td>0</td>
<td>1000</td>
</tr>
<tr>
<td>10</td>
<td>+0</td>
<td>1111</td>
<td>0001</td>
<td>0</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>1</td>
<td>1101</td>
</tr>
<tr>
<td>30</td>
<td>+0</td>
<td>1111</td>
<td>0001</td>
<td>1</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>1</td>
<td>0001</td>
</tr>
<tr>
<td>50</td>
<td>+0</td>
<td>0101</td>
<td>1110</td>
<td>1</td>
<td>1</td>
<td>0001</td>
</tr>
<tr>
<td>60</td>
<td>+0</td>
<td>0101</td>
<td>1110</td>
<td>1</td>
<td>1</td>
<td>0011</td>
</tr>
<tr>
<td>70</td>
<td>+0</td>
<td>0101</td>
<td>1110</td>
<td>1</td>
<td>1</td>
<td>0111</td>
</tr>
<tr>
<td>80</td>
<td>+0</td>
<td>0101</td>
<td>1110</td>
<td>1</td>
<td>1</td>
<td>0011</td>
</tr>
</tbody>
</table>

2.1 VHDL Description of Combinational Networks – Testbench

entity ADDER is
end ADDER;

architecture TESTBENCH of ADDER is
begin
  -- signals
  A <= "1111" after 0 ns, "0101" after 50 ns;
  B <= "0001" after 0 ns, "1110" after 50 ns;
  Ci <= '1' after 0 ns, '0' after 50 ns;

  -- component being tested
  ctrl: ADDER port map (A,B,Ci,S,Co);
end TESTBENCH;
2.1 VHDL Description of Combinational Networks - Altera Full Adder Simulation

2.1 VHDL Description of Combinational Networks - Altera Adder4 Simulation
2.1 VHDL Description of Combinational Networks - Behavioral Adder4

-- Behavioral Description of 4 Bit Adder

-- Libraries
library IEEE;
use IEEE.std_logic_1164.all; -- to allow STD_LOGIC_VECTORS
use IEEE.std_logic_arith.all; -- to allow arithmetic operations
use IEEE.std_logic_unsigned.all; -- to allow integer conversions

entity ADDER4 is
  port(A,B: in STD_LOGIC_VECTOR (3 downto 0)); -- Inputs
  CI : in STD_LOGIC; -- Input
  S : out STD_LOGIC_VECTOR (3 downto 0); -- Outputs
  CO: out STD_LOGIC; -- Output
end ADDER4;

architecture BEHAVIORAL of ADDER4 is
  -- temporary signal vectors which are 5 bits to keep up with CO signal
  AA,BB,SS : STD_LOGIC_VECTOR (4 downto 0) := "00000";

begin
  -- converting from 4 bit inputs to 5 bit ones to allow full
  -- 5 bit arithmetic
  AA <= '0'&A; -- appending a leading 0 at most significant
  BB <= '0'&B; -- bit position of A and B

  -- main equation for 4 bit adder (describes desired behavior)
  SS <= AA + BB + CI;

  -- relating this internal output to the port output signals
  S <= SS(3 downto 0); -- 4 bit Sum
  CO <= SS(4); -- carry out
end BEHAVIORAL;
2.1 VHDL Description of Combinational Networks - Behavioral Adder4 Simulation

<table>
<thead>
<tr>
<th>ps delta</th>
<th>a</th>
<th>b</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>
</tr>
<tr>
<td>0</td>
<td>+1</td>
<td>1111</td>
<td>0001</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>+3</td>
<td>1111</td>
<td>0001</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>+4</td>
<td>1111</td>
<td>0001</td>
<td>1</td>
</tr>
<tr>
<td>50000</td>
<td>+0</td>
<td>0101</td>
<td>1110</td>
<td>1</td>
</tr>
<tr>
<td>50000</td>
<td>+2</td>
<td>0101</td>
<td>1110</td>
<td>1</td>
</tr>
<tr>
<td>50000</td>
<td>+3</td>
<td>0101</td>
<td>1110</td>
<td>1</td>
</tr>
</tbody>
</table>

• Whenever one of the signals in the sensitivity list changes, the sequential statements are executed in sequence one time

2.2 Modeling Flip-Flops Using VHDL Processes - The Process Statement

General form of process

```
process(sensitivity-list)
begin
sequential-statements
end process;
```

- Whenever one of the signals in the sensitivity list changes, the sequential statements are executed in sequence one time
A, B, C, D are integers A=1, B=2, C=3, D=0 D changes to 4 at time 10

\[
\begin{align*}
A &\leq B; \quad \text{-- statement 1} \\
B &\leq C; \quad \text{-- statement 2} \\
C &\leq D; \quad \text{-- statement 3}
\end{align*}
\]

\[
\begin{align*}
\text{process} &\quad \text{(B, C, D)} \\
&\quad \text{begin} \\
&\quad \quad A \leq B; \quad \text{-- statement 1} \\
&\quad \quad B \leq C; \quad \text{-- statement 2} \\
&\quad \quad C \leq D; \quad \text{-- statement 3} \\
&\quad \text{end process;}
\end{align*}
\]
2.2 Modeling Flip-Flops Using VHDL
Processes - Modeling a D Latch

```vhdl
entity DLATCH is
  port (D, Q, QN: in bit;
       Q: out bit; QN: out bit := 'L');
end DLATCH;

architecture SIMPLE of DLATCH is
begin
  process (D, Q) -- process is executed when either D or Q changes
  begin
    if Q = '1' then -- pass D through when Q=1
      Q <= D after 10 ns;
      QN <= not D after 10 ns;
    end if;
  end process;
end SIMPLE;
```

2.2 Modeling Flip-Flops Using VHDL
Processes - D Flip-Flop versus D Latch
2.2 Modeling Flip-Flops Using VHDL Processes - Altera DFF Simulation

Altera’s Simulation of D Flip-Flop

2.2 Modeling Flip-Flops Using VHDL Processes - Altera D Latch Simulation

Altera’s Simulation of D Latch
Chapter 2

Building a Shift Register with D Flip-flop Building Blocks

-- A Structural Description of a 4 Bit Shift Register

-- Libraries
library IEEE;
USE IEEE.std_logic_1164.all; --to allow STD_LOGIC
USE IEEE.std_logic_arith.all;--to allow arithmetic operation
USE IEEE.std_logic_1164.all; --to allow integer conversions

entity SHIFTS is
port(D, CLK ; in STD_LOGIC; -- Inputs
Q : out STD_LOGIC_VECTOR (3 downto 0)); -- Outputs
end SHIFTS;

architecture STRUCT of SHIFTS is
signal Q0 : STD_LOGIC_VECTOR (3 downto 0) := "0000";
component DFF is
port (D, CLK in STD_LOGIC;
Q : out STD_LOGIC; QN : out STD_LOGIC := '1');
end component;
begin
C0 : DFF port map (D,CLK,0'0),Q0(0));
C1 : DFF port map (0'0,CLK,0'1),Q0(1));
C2 : DFF port map (0'1,CLK,0'2),Q0(2));
C3 : DFF port map (0'2,CLK,0'3),Q0(3));
end STRUCT;
Testing the Shift Register

```vhdl
-- Libraries
library IEEE;
use ieee.std_logic_1164.all; -- to allow STD_LOGIC_VECTORS
use ieee.std_logic_arith.all; -- to allow arithmetic operations
use ieee.std_logic_unsigned.all; -- to allow integer conversions

entity SHIFT48 is
end SHIFT48;

architecture TESTBENCH of SHIFT48 is
component SHIFT48 is
    port(CLK, DI : in STD_LOGIC; -- Inputs
         Q : out STD_LOGIC_VECTOR (3 downto 0)); -- Outputs
end component;

signal Q : STD_LOGIC_VECTOR(3 downto 0) := "0000";
signal CLA, DI : STD_LOGIC := '0';
begin
    -- signals
    CLA <= not CLA after 50 ns;
    DI <= '1' after 0 ns, '0' after 100 ns, '1' after 200 ns, '0' after 300 ns, '1' after 400 ns;
    -- component being tested
    CTL: SHIFT48 port map (CLK,DI,Q);
end TESTBENCH;
```

A Behavioral Shift Register Model

```vhdl
-- A Behavioral Description of 4 Bit Shift Register

-- Libraries
library IEEE;
use ieee.std_logic_1164.all; -- to allow STD_LOGIC_VECTORS
use ieee.std_logic_arith.all; -- to allow arithmetic operations
use ieee.std_logic_unsigned.all; -- to allow integer conversions

entity SHIFT48 is
    port(CLK, DI : in STD_LOGIC; -- Inputs
         Q : out STD_LOGIC_VECTOR (3 downto 0)); -- Outputs
end SHIFT48;

architecture BEHAVIORAL of SHIFT48 is
begin
    process (CLK)
    begin
        if (CLK = '1') then
            0(0) <= DI;
            0(1) <= 0(0);
            0(2) <= 0(1);
            0(3) <= 0(2);
        end if;
    end process;
end BEHAVIORAL;
```
Another Behavioral Shift Register Model

-- Another Behavioral Description of a 4 Bit Shift Register

-- Libraries
Library IEEE;
USE IEEE.std_logic_1164.all; --to allow STD_LOGIC_vectors
USE IEEE.std_logic_arith.all;--to allow arithmetic operations
USE IEEE.std_logic_signed.all; --to allow integer conversions

entity SHIFT4B is
  port(CLK, DI : in STD_LOGIC); -- Inputs
  Q : out STD_LOGIC_VECTOR (3 downto 0)); -- Outputs
end SHIFT4B;

architecture BEHAVIORAL of SHIFT4B is
begin
process (CLK)
begin
  if (CLK = '1') then
    Q <= Q(2 downto 0) & DI;
  end if;
end process;
end BEHAVIORAL;

Shift Register Simulation Results

<table>
<thead>
<tr>
<th>clk</th>
<th>ps</th>
<th>delta</th>
<th>di</th>
<th>o</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>5000</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>5000</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>10000</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>15000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>15000</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>20000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>25000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>25000</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>30000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>30000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>35000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>35000</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>40000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>45000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>45000</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>50000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>50000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>55000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>55000</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>60000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>65000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>65000</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>70000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>75000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>80000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Electrical and Computer Engineering
2.2 Modeling Flip-Flops Using VHDL

Processes - Modeling a JK Flip-Flop

entity JKFF is
port (SN, RN, J, K, CLK: in bit; Q: inout bit; QN: out bit := '1'); -- inputs
end JKFF;

architecture JKFF1 of JKFF is
begin
process (SN, RN, CLK) -- see Note 2
begin
if RN = '0' then Q <= '0' after 10 ns; -- RN=0 will clear the FF
elsif SN = '0' then Q <= '1' after 10 ns; -- SN=0 will set the FF
elsif CLK = '0' and CLK'event then
Q <= (J and not Q) or (not K and Q) after 10 ns; -- see Note 3
end if;
end process;
QN <= not Q; -- see Note 5
end JKFF1;

Processes - Nested If-then-else

if (C1) then S1; S2;
else if (C2) then S3; S4;
else if (C3) then S5; S6;
else S7; S8;
end if;
end if;

if (C1) then S1; S2;
elsif (C2) then S3; S4;
elsif (C3) then S5; S6;
else S7; S8;
end if;
2.3 VHDL Models for a Multiplexer

The case statement has the general form:

```vhdl
case Sel is
  when 0 => F <= 10;
  when 1 => F <= 11;
  when 2 => F <= 12;
  when 3 => F <= 13;
  others => Y <= A(15);
end case;
```

If Statements

```vhdl
architecture RTL1 of SELECTOR is
begin
  p0 : process (A, SEL)
  begin
    if (SEL = "0000") then
      Y <= A(0);
    elsif (SEL = "0001") then
      Y <= A(1);
    elsif (SEL = "0010") then
      Y <= A(2);
    elsif (SEL = "0011") then
      Y <= A(3);
    elsif (SEL = "0100") then
      Y <= A(4);
    elsif (SEL = "0101") then
      Y <= A(5);
    elsif (SEL = "0110") then
      Y <= A(6);
    elsif (SEL = "0111") then
      Y <= A(7);
    elsif (SEL = "1000") then
      Y <= A(8);
    elsif (SEL = "1001") then
      Y <= A(9);
    elsif (SEL = "1010") then
      Y <= A(10);
    elsif (SEL = "1011") then
      Y <= A(11);
    elsif (SEL = "1100") then
      Y <= A(12);
    elsif (SEL = "1101") then
      Y <= A(13);
    elsif (SEL = "1110") then
      Y <= A(14);
    else
      Y <= A(15);
    end if;
  end process;
end RTL1;
```
2.3 VHDL Models for a Multiplexer - Conditional Concurrent Statement

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity SELECTOR is
  port (
    A : in std_logic_vector(15 downto 0);
    SEL : in std_logic_vector(3 downto 0);
    Y : out std_logic);
end SELECTOR;

architecture RTL3 of SELECTOR is
begin
  with SEL select
    Y <= A(0)  when "0000",
       A(1)  when "0001",
       A(2)  when "0010",
       A(3)  when "0011",
       A(4)  when "0100",
       A(5)  when "0101",
       A(6)  when "0110",
       A(7)  when "0111",
       A(8)  when "1000",
       A(9)  when "1001",
       A(10) when "1010",
       A(11) when "1011",
       A(12) when "1100",
       A(13) when "1101",
       A(14) when "1110",
       A(15) when others;
end RTL3;

2.3 VHDL Models for a Multiplexer - Case Statement

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity SELECTOR is
  port (
    A : in std_logic_vector(15 downto 0);
    SEL : in std_logic_vector(3 downto 0);
    Y : out std_logic);
end SELECTOR;

architecture RTL2 of SELECTOR is
begin
  p1 : process (A, SEL)
  begin
    case SEL is
      when "0000" => Y <= A(0);
      when "0001" => Y <= A(1);
      when "0010" => Y <= A(2);
      when "0011" => Y <= A(3);
      when "0100" => Y <= A(4);
      when "0101" => Y <= A(5);
      when "0110" => Y <= A(6);
      when "0111" => Y <= A(7);
      when "1000" => Y <= A(8);
      when "1001" => Y <= A(9);
      when "1010" => Y <= A(10);
      when "1011" => Y <= A(11);
      when "1100" => Y <= A(12);
      when "1101" => Y <= A(13);
      when "1110" => Y <= A(14);
      when others => Y <= A(15);
    end case;
  end process;
end RTL2;
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
entity SELECTOR is
  port(
    A : in std_logic_vector(15 downto 0);
    SEL : in std_logic_vector( 3 downto 0);
    Y : out std_logic);
end SELECTOR;

architecture RTL4 of SELECTOR is
begin
  Y <= A(conv_integer(SEL));
end RTL4;

2.3 VHDL Models for a Multiplexer - Register Transfer Level

- Compiler (Analyzer) – checks the VHDL source code
  - does it conforms with VHDL syntax and semantic rules
  - are references to libraries correct
- Intermediate form used by a simulator or by a synthesizer
- Elaboration
  - create ports, allocate memory storage, create interconnections, ...
  - establish mechanism for executing of VHDL processes

2.4 Compilation and Simulation of VHDL
2.4 Compilation and Simulation of VHDL - Transport Delay

- Transport delay must be explicitly specified
  - i.e. keyword “TRANSPORT” must be used
- Signal will assume its new value after specified delay

```vhdl
-- TRANSPORT delay example
Output <= TRANSPORT NOT Input AFTER 10 ns;
```

2.4 Compilation and Simulation of VHDL - Inertial Delay

- Provides for specification propagation delay and input pulse width, i.e. ‘inertia’ of output:

```vhdl
target <= [REJECT time_expression] INERTIAL waveform;
```

- Inertial delay is default and REJECT is optional:

```vhdl
Output <= NOT Input AFTER 10 ns;
-- Propagation delay and minimum pulse width are 10ns
```
2.4 Compilation and Simulation of VHDL - Modeling Propagation Delay

- Example of gate with 'inertia' smaller than propagation delay
  - e.g. Inverter with propagation delay of 10ns which suppresses pulses shorter than 5ns

```vhdl
Output <= REJECT 5ns INERTIAL NOT Input AFTER 10ns;
```

- Note: the REJECT feature is new to VHDL 1076-1993

---

**2.4 Compilation and Simulation of VHDL - Simulation Example**

**Signal Drivers**

```vhdl
entity simulation_example is
end simulation_example;
architecture test1 of simulation_example is
  signal A,B: bit;
begin
  P1: process(B)
  begin
    A <= '1';
    A <= transport '0' after 5 ns;
  end process P1;
  P2: process(A)
  begin
    if A = '1' then B <= not B after 10 ns; end if;
  end process P2;
end test1;
```
2.5 Modeling Sequential Circuits

Mealy Machine for 8421 BCD to 8421 BCD + 3 bit serial converter

How to model this in VHDL?

Two processes:
- the first represents the combinational network;
- the second represents the state register.
2.5 Modeling Sequential Circuits - Dataflow VHDL Model

-- 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 bit;
  Z: out bit);
end SM1_2;

architecture Equations1_4 of SM1_2 is
signal Q1,Q2,Q3: bit;
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;

Package bit_pack is a part of library BITLIB –
includes gates, flip-flops, counters
(See Appendix B for details)
2.12 VHDL Model for A 74163 Counter – Truth Table

Control Signals          Next State
ClRN  LDN  P•T  Q3*  Q2*  Q1*  Q0*
---   ---   ---   ---   ---   ---   ---
0     X     X     0     0     0     0  (clear)
1     0     X     D3    D2    D1    D0  (parallel load)
1     1     0     Q3    Q2    Q1    Q0  (no change)
1     1     1     present state + 1  (increment count)

Generate a Cout in state 15 if T=1
Cout = Q_3 Q_2 Q_1 Q_0 T

2.12 VHDL Model for a 74163 Counter - VHDL

-- 74163 FULLY SYNCHRONOUS COUNTER
library BITLIB;               -- contains int2vec and vec2int functions
use BITLIB.bit_pack.all;
entity c74163 is
  port(CLKN, ClRN, P, T, CK: in bit; D: in bit_vector(3 downto 0); 
    Cout: out bit; Q: inout bit_vector(3 downto 0));
end c74163;
architecture b74163 of c74163 is
begin
  Cout <= Q(3) and Q(2) and Q(1) and Q(0) and T;
  process
    wait until CK = '1'; -- change state on rising edge
    if ClRN = '0' then Q <= "0000";
    elsif LDN = '0' then Q <= D;
    elsif (P and T) = '1' then
      Q <= int2vec(vec2int(Q)+1,4);
    end if;
  end process;
end b74163;
2.12 VHDL Model for a 74163 Counter - Cascade Counters (Block Diagram)

```vhdl
library BITLIB;
use BITLIB.bit_pack.all;

entity c74163test is
  port(CIn, DIn1, DIn2, P, T, Clk: in bit;
       Qout1, Qout2: in bit_vector(3 downto 0);
       Cout: out bit);
  end c74163test;

architecture tester of c74163test is
  component c74163
    port (DIn, Clk, P, T, Clk: in bit;
          Qout: out bit_vector(3 downto 0);)
  end component;
  signal Carry1: bit;
  signal Count: integer;
  signal temp: bit_vector(7 downto 0);
begin
  c74163 port map (DIn, Clk, P, T, Clk, DIn1, Carry1, Qout1);
  c74163 port map (DIn, Clk, P, Carry1, Clk, DIn2, Carry2, Qout2);
  temp <= Qout2 & Qout1;
  Count <= vec2uint(temp);
end tester;
```

Electrical and Computer Engineering
2.5 Modeling a Sequential Machine - Wait Statements

• ... an alternative to a sensitivity list
  – Note: a process cannot have both wait statement(s) and a sensitivity list
• Generic form of a process with wait statement(s)

process
begin
  sequential-statements
  wait statement
  sequential-statements
  wait-statement
  ...
end process;

How do wait statements work?
• Execute sequential statements until a wait statement is encountered.
• Wait until the specified condition is satisfied.
• Then execute the next set of sequential statements until the next wait statement is encountered.
• ...
• When the end of the process is reached start over again at the beginning.

2.5 Modeling a Sequential Machine - Forms of Wait Statements

wait on sensitivity-list;
wait for time-expression;
wait until boolean-expression;

• Wait on
  – ______ one of the ___________ changes
• Wait for
  – waits until the ______ by the time expression has ______
  – What is this:
    wait for 0 ns;
• Wait until
  – the boolean expression is evaluated whenever one of ______ changes, and the process continues execution when the expression evaluates to ______
2.5 Modeling a Sequential Machine - Using Wait Statements

```vhdl
wait on CLK, X;
if rising_edge(CLK) then
    State <= Nextstate;
    wait for 0 ns;
end if;
end process;
```

2.6 Variables, Signals, Constants - Variables

- What they are for:
  Local storage in __________, __________, and __________

- Declaring variables
  ```vhdl
  variable list_of_variable_names : type_name
  [ := initial value ];
  ```

- Variables must be declared within the process in which they are used and are local to the process
  - Note: exception to this is SHARED variables
  - Variables in processes are __________, they retain their value from one activation to the next
2.6 Variables, Signals, Constants - Signals

- Signals must be declared ________ a process
- Declaration form

```vhdl
signal list_of_signal_names : type_name
[ := initial value ];
```

- Declared in an architecture can be used anywhere within that architecture
- Declared in the _____________

2.6 Variables, Signals, Constants - Constants

- Declaration form

```vhdl
constant constant_name : type_name := constant_value;
```

```vhdl
constant delay1 : time := 5 ns;
```

- Constants declared at the start of an architecture can be used anywhere within that architecture
- Constants declared within a process are local to that process
2.6 Variables, Signals, Constants - Variables versus Signals

- Variable assignment statement
  variable_name := expression;
  - expression is evaluated and the variable is
    _______________ updated
    (no delay, not even delta delay)

- Signal assignment statement
  signal_name <= expression [after delay];
  - expression is evaluated and the signal is
    _______________; if no delay is specified the signal is
    scheduled to be updated after a delta delay

Process Using Variables

``` VHDL
entity dummy is
end dummy;
architecture var of dummy is
signal trigger, sum: integer:=0;
begin
process
variable var1: integer:=1;
variable var2: integer:=2;
variable var3: integer:=3;
begin
  wait on trigger;
  var1 := var2 + var3;
  var2 := var1;
  var3 := var2;
  sum := var1 + var2 + var3;
end process;
end var;
Sum = ?
```

Process Using Signals

``` VHDL
entity dummy is
end dummy;
architecture sig of dummy is
signal trigger, sum: integer:=0;
signal sig1: integer:=1;
signal sig2: integer:=2;
signal sig3: integer:=3;
begin
process
  wait on trigger;
  sig1 <= sig2 + sig3;
  sig2 <= sig1;
  sig3 <= sig2;
  sum <= sig1 + sig2 + sig3;
end process;
end sig;
Sum = ?
```
2.6 Variables, Signals, Constants - Predefined VHDL Types

- Variables, signals, and constants can have any one of the ______________ VHDL types or they can have a __________ type.
- Predefined Types:
  - bit – {'0', '1'}
  - boolean – {FALSE, TRUE}
  - integer – [2^31 - 1.. 2^31 – 1]
  - real – floating point number in range –1.0E38 to +1.0E38
  - character – legal VHDL characters including lower-case letters, digits, special characters, ...
  - time – an integer with units fs, ps, ns, us, ms, sec, min, or hr

2.6 Variables, Signals, Constants - User Defined Type

- Common user-defined type is ______________

```vhdl
type state_type is (S0, S1, S2, S3, S4, S5);
signal state : state_type := S1;
```

- If no initialization, the ________________ is the leftmost element in the enumeration list (S0 in this example).

- VHDL is **strongly typed** => signals and variables of different types ________ be mixed in the same assignment statement, and no ________________ is performed.
2.7 Arrays - Syntax

• Example

```vhdl
type SHORT_WORD is array (15 downto 0) of bit;
signal DATA_WORD : SHORT_WORD;
variable ALT_WORD : SHORT_WORD := "0101010101010101";
constant ONE_WORD : SHORT_WORD := (others => '1');

• ALT_WORD(0) – rightmost bit
• ALT_WORD(5 downto 0) – low order 6 bits

• General form

```vhdl

type arrayTypeName is array index_range of element_type;
signal arrayName : arrayTypeName [:=InitialValues];
```

2.7 Arrays – Syntax continued

• Multidimensional arrays

```vhdl
type matrix4x3 is array (1 to 4, 1 to 3) of integer;
variable matrixA: matrix4x3 :=
((1,2,3), (4,5,6), (7,8,9), (10,11,12));

• matrixA(3, 2) = ?

• Unconstrained array type

```vhdl

type intvec is array (natural range<>) of integer;
type matrix is array (natural range<>,natural range<>) of integer;

• range must be specified when the ________________

```vhdl

```
2.6 Variables, Signals, Constants - Predefined Unconstrained Array Types

type bit_vector is array (natural range <> ) of bit;
type string is array (positive range <> ) of character;
constant string1: string(1 to 29) := "This string is 29 characters."

constant A : bit_vector(0 to 5) := "10101";
-- ('1', '0', '1', '0', '1');

• Subtypes

• include a subset of the values specified by the type

  subtype SHORT_WORD is : bit_vector(15 to 0);

• POSITIVE, NATURAL – predefined subtypes of type integer

2.7 Arrays - Sequential Machine Model Using State Table

entity SML_2 is
  port (X, CLK: in bit;
        Z: out bit);
end SML_2;

architecture Table of SML_2 is
  type StateTable is array (integer range <> , bit range <> ) of integer;
type OutTable is array (integer range <> , bit range <> ) of bit;
signal State, NextState: integer := 6;
constant ST: StateTable (0 to 6, '0' to '1') :=
  ((1,2), (1,4), (4,4), (5,5), (5,8), (9,0), (0,0));
constant OT: OutTable (0 to 6, '0' to '1') :=
  (('1', '0'), ('1', '0'), ('0', '0'), ('1', '0'), ('1', '0'), ('0', '0'), ('1', '0'));
begin
  NextState <= ST(State, X); -- read next state from state table
  Z <= OT(State, X); -- read output from output table
process(CLK)
begin
  if CLK = '1' then -- rising edge of CLK
    NextState <= NextState;
  end if;
end process;
e end Table;
2.8 VHDL Operators – Precedence Relations

- Class 7 has the highest precedence (applied first), followed by class 6, then class 5, etc

1. Binary logical operators: **and or nand nor xor xnor**
2. Relational: = /= < <= > >=
3. Shift: **sll srl sla sra rol ror**
4. Adding: + - & (concatenation)
5. Unary sign: + -
6. Multiplying: * / **mod rem**
7. Miscellaneous: not abs **

Examples

In the following expression, A, B, C, and D are bit vectors:
(A & not B or C ror 2 and D) = "11010"

The operators would be applied in the order:
not, &, or, ror, and, =

If A = "110", B = "111", C = "011000", and D = "110111", the computation would proceed as follows:
not B = "000"  (bit-by-bit complement)
A & not B = "110000"  (concatenation)
C ror 2 = "000110"  (rotate right 2 places)
(A & not B) or (C ror 2) = "110110"  (bit-by-bit or)
(A & not B or C ror 2) and D = "110010"  (bit-by-bit and)
[(A & not B or C ror 2 and D) = "110010"] = TRUE
   (the parentheses force the equality test to be done last and the result is TRUE)
2.8 VHDL Operators – Shift Operator Examples

The shift operators can be applied to any bit_vector or boolean_vector. In the following examples, A is a bit_vector equal to "10010101":

- A slt 2 is "01010100" (shift left logical, filled with '0')
- A srl 3 is "00010010" (shift right logical, filled with '0')
- A sia 3 is "10101111" (shift left arithmetic, filled with right bit)
- A sra 2 is "11100101" (shift right arithmetic, filled with left bit)
- A rol 3 is "10101100" (rotate left)
- A ror 5 is "10101100" (rotate right)

2.9 VHDL Functions - Introduction

- Functions execute a sequential algorithm and return a single value to calling program

```vhdl
function rotate_right (reg: bit_vector) return bit_vector is
begin
  return reg ror 1;
end rotate_right;

B <= rotate_right(A);
```

- General form

```vhdl
function function-name (formal-parameter-list) return return-type is
  [declarations]
begin
  sequential statements -- must include return return-value;
end function-name;
```
2.9 VHDL Functions – For Loops

General form of a for loop:

```vhdl
{loop-label:} for loop-index in range loop
    sequential statements
end loop [loop-label];
```

**For Loop Example:**

```vhdl
-- compare two 8-character strings and return TRUE if equal
function comp_string(string1, string2: string(1 to 8))
    return boolean is
end function comp_string;
```

```vhdl
variable B: boolean;
begin
    loopex: for j in 1 to 8 loop
        B := string1(j) = string2(j);
        exit when B=FALSE;
    end loop loopex;
    return B;
end comp_string;
```

---

2.9 VHDL Functions - Add Function

```vhdl
-- This function adds 2 4-bit vectors and a carry.
-- It returns a 5-bit sum
function add4 (A,B: bit_vector(3 downto 0); carry: bit)
    return bit_vector is
variable cout: bit;
variable cin: bit := carry;
variable Sum: bit_vector(4 downto 0):="00000";
begin
    loop1: for i in 0 to 3 loop
        cout := (A(i) and B(i)) or (A(i) and cin) or (B(i) and cin);
        Sum(i) := A(i) xor B(i) xor cin;
        cin := cout;
    end loop loop1;
    Sum(4):= cout;
    return Sum;
end add4;
```

Example function call:

```vhdl
Sum1 <= add4(A1, B1, cin);
```
2.10 VHDL Procedures - Introduction

- Procedures can return any number of values using output parameters

- General form

```vhd
procedure procedure_name (formal-parameter-list) is
    [declarations]
begin
    Sequential-statements
end procedure_name;
```

Procedure for Adding Bit_Vectors

```vhd
-- This procedure adds two n-bit bit_vectors and a carry and -- returns an n-bit sum and a carry. Add1 and Add2 are assumed -- to be of the same length and dimensioned n-1 downto 0.

procedure Addvec
    (Add1,Add2: in bit_vector;
     Cin: in bit;
     signal Sum: out bit_vector;
     signal Cout: out bit;
     n: in positive) is
    variable C: bit;
begin
    C := Cin;
    for i in 0 to n-1 loop
        Sum(i) <= Add1(i) xor Add2(i) xor C;
        C := (Add1(i) and Add2(i)) or (Add1(i) and C) or (Add2(i) and C);
    end loop;
    Cout <= C;
end Addvec;
```

Example procedure call:

```
Addvec(A1, B1, Cin, Sum1, Cout, 4);
```
2.10 VHDL Procedures – Parameters for Subprogram Calls

<table>
<thead>
<tr>
<th>Mode</th>
<th>Class</th>
<th>Procedure Call</th>
<th>Function Call</th>
</tr>
</thead>
<tbody>
<tr>
<td>in&lt;sup&gt;1&lt;/sup&gt;</td>
<td>constant&lt;sup&gt;2&lt;/sup&gt;</td>
<td>expression</td>
<td>expression</td>
</tr>
<tr>
<td></td>
<td>signal</td>
<td>signal</td>
<td>signal</td>
</tr>
<tr>
<td>out/inout</td>
<td>variable</td>
<td>variable</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>signal</td>
<td>signal</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>variable&lt;sup&gt;3&lt;/sup&gt;</td>
<td>variable</td>
<td>n/a</td>
</tr>
</tbody>
</table>

<sup>1</sup> default mode for functions  
<sup>2</sup> default for in mode  
<sup>3</sup> default for out/inout mode

2.11 Packages and Libraries

- Provide a convenient way of referencing frequently used functions and components
- Package header

```vhdl
package package-name is
    package declarations
end [package][package-name];
```

- Package body [optional]

```vhdl
package body package-name is
    package body declarations
end [package body][package name];
```
2.11 Packages and Libraries – Library BITLIB – bit_pack package

```vhdl
package bit_pack is
  function add4 (reg1, reg2: bit_vector[3 downto 0]; carry: bit) return bit_vector;
  function falling_edge(signal clock: bit) return Boolean;
  function rising_edge(signal clock: bit) return Boolean;
  function vec2Cnt(vec1: bit_vector) return Integer;
  function int2Vec(int1, NBits: integer) return bit_vector;
  procedure Add2; in bit_vector;
  signal Sum: out bit_vector;
  signal Cout: out bit;
  n: in natural;
  component jk
    generic(Delay: time := 10 ns);
    port(SN, RN, JK, CLK: in bit; Q, QN: inout bit);
  end component;
  component dt
    generic(Delay: time := 10 ns);
    port(D, CLK: in bit; Q: out bit; QN: out bit := '1');
  end component;
end bit_pack;
```

```vhdl
package body bit_pack is
  function add4 (reg1, reg2: bit_vector[3 downto 0]; carry: bit) return bit_vector is
    variable cout: bit:= '0';
    variable cin: bit:= '0';
    variable retval: bit_vector(4 downto 0):= "00000";
  begin
    for i in 0 to 3 loop
      cout := ((reg1(ip) and reg2(ip)) or (reg1(ip) and cin)) or (reg2(ip) and cin);
      retval(i):= reg1(ip) xor reg2(ip) xor cin;
      cin := cout;
    end loop;
    return retval;
  end add4;
  function falling_edge(signal clock: bit) return Boolean is
  begin
    return clock and not clock;
  end falling_edge;
end bit_pack;
```

2.11 Packages and Libraries –
Library BITLIB – bit_pack package

Components in Library BITLIB include:

--- 3 input AND gate
entity And3 is
generic(DELAY:time);
port (A1, A2, A3: in bit; Z: out bit);
end And3;
architecture concurrent of And3 is
begin
Z <= A1 and A2 and A3 after DELAY;
end;

--- D Flip-flop
entity DFF is
generic(DELAY:time);
port (D, CLK: in bit;
Q, QN: out bit: QN = '1' when D = '1');
--- initialize QN to '1' since bit signals are initialized to '0' by default
end DFF;
architecture SIMPLE of DFF is
begin
process(CLK)
begin
if CLK = '1' then -- rising edge of clock
Q <= D after DELAY;
QN <= not D after DELAY;
end if;
end process;
end SIMPLE;

Design of a Keypad Scanner
Design of a Keypad Scanner