VHDL – Dataflow and Structural Modeling and Testbenches

ENGIN 341 – Advanced Digital Design
University of Massachusetts Boston
Department of Engineering
Dr. Filip Cuckov
Overview

1. Introduction to VHDL
2. VHDL Language Constructs
3. VHDL Testbenches
4. Lab 1 – Modular Design and Testbench Simulation
5. Generic and Generate Statements
6. Dataflow MUX Design
7. More on Testbenches
1. Introduction to VDHL

- **VDHL**: VHSIC Hardware Description Language
  - **VHSIC**: Very High Speed Integrated Circuit
- Developed with support from the DoD (US Air Force) in 1981
  - By Intermetrics, Texas Instruments, and IBM
- Based on ADA
- IEEE standard in ‘87,’93,’00,’02, ’07,’08 (VHDL-1987 ... VHDL-2008)
- HDL used for EDA
  - Hardware Description Language used for Electronic Design Automation
VHDL - Defined

• Parallel programming language
  • Concurrent process execution

• Used to model, simulate, and synthesize electronic circuits
  • At various levels of abstraction
  • Supports management of hierarchies and incremental design

• A general purpose programming language
  • This class will focus on synthesizable and simulation subset

• Competitor: Verilog
  • You will eventually need to learn it
VHDL - Modular Design

• Using a Black Box (BB) approach
  • **Entity**
    • BB external inputs and outputs
  ![Entity Diagram](image)

• **Architecture**
  • The “guts” of the BB
  ![Architecture Diagram](image)
  ```vhdl
  entity two_gates is
    port(A, B, D: in bit; E: out bit);
  end two_gates;

  architecture gates of two_gates is
  signal C: bit;
  begin
    C <= A and B; -- concurrent
    E <= C or D; -- statements
  end gates;
  ```

• Many architectures can describe the internals of one entity
VHDL - Hierarchical Design

• Use several levels of modeling abstraction to design complex systems

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

architecture Dataflow of FullAdder is
begin
  Sum <= A XOR B XOR Cin;
  Cout <= ( A AND B ) OR ( B AND Cin ) OR ( A AND Cin );
end Dataflow;
```
VHDL Modeling Styles

• Structural
  • Hierarchical approach
  • Interconnecting building blocks to compose larger system

• Dataflow
  • Define outputs as a function of inputs
  • Similar to entering Boolean equations

• Behavioral
  • Functional modeling using an algorithmic approach
  • Similar to high-level language programming
Example VHDL Code – Structural Design

• Entities instantiated in other architectures are called **components**

```vhdl
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); -- C is an internal signal
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;
```

![Diagram of a structural design](image-url)
Example VHDL Code – Dataflow Design

```vhdl
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 Dataflow of Adder4 is
  signal C1, C2, C3 : bit;  -- Internal signals
begin
  S(0) <= A(0) XOR B(0) XOR Ci;
  C1   <= ( A(0) AND B(0) ) OR ( B(0) AND Ci ) OR ( A(0) AND Ci );
  S(1) <= A(1) XOR B(1) XOR C1;
  -- ... and so on ...
  -- OR we can enter 5 single expressions for S(3), S(2), S(1), S(0), and Co
end Dataflow;
```

architecture Behavioral of Adder4 is
begin
process (A, B, Ci)
begin
    if (Ci = '0') then
        if (A = "0000" AND B = "0000") then
            S <= "0000"; Co <= '0';
        elsif (A = "0001" AND B = "0000") then
            S <= "0001"; Co <= '0';
            -- .. And so on
        else
            S <= "0000"; Co <= '0';
        end if;
    elsif (Ci = '1') then
        if (A = "0000" AND B = "0000") then
            S <= "0001"; Co <= '0';
            -- .. And so on
        end if;
    else
        S <= "0000"; Co <= '0';
    end if;
end process;
end Behavioral;
2. VHDL Language Constructs

Book Appendix A

• VHDL is NOT case sensitive
• VHDL is strongly typed
• Signal names and identifiers
  • May:
    • Contain letters, numbers, and the underscore character
  • May NOT:
    • Be a reserved keyword
    • Begin or end with an underscore
  • Must:
    • Start with a letter
VHDL Built-In Data Types

- **Bit**
  - ‘0’ or ‘1’
- **Bit_vector**
  - array of bits, i.e. “101010” or “01”
- **Boolean**
  - *TRUE* or *FALSE*
- **Integer**
  - range $-2^{31}-1$ to $2^{31}-1$
- **Real**
  - floating point range $-1.0E38$ to $1.0E38$
- **Character**
  - printable characters, enclosed between ‘@’
- **String**
  - array of characters
- **Time**
  - integer with units $fs$, $ps$, $ns$, $us$, $ms$, $sec$, $min$, $hr$
- **Delay_length**
  - positive time
- **Natural**
  - integer $\geq 0$
- **Positive**
  - integer $> 0$
VHDL Built-In Operators by Precedence

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

( Take-h()me message: ( Use parentheses (whenever) p()ssible ) )
IEEE std_logic_1164 Library Data Types

type std_ulogic is ( 'U', -- Uninitialized
    'X', -- Forcing Unknown
    '0', -- Forcing 0
    '1', -- Forcing 1
    'Z', -- High Impedance
    'W', -- Weak Unknown
    'L', -- Weak 0
    'H', -- Weak 1
    '-' -- Don't care
);

subtype std_logic is resolved std_ulogic; -- More on this later
3. VHDL Testbenches

- Entity with no ports
- Testbench code contains time and delays
  - Not synthesizable nor meant to be
- Generate stimulus for UUT in Architecture body and observe outputs
4. Lab 1- Modular Design and Testbenches

• Let’s practice
5. Generic and Generate Statements

- Recall the Ripple Carry Adder
- What if we wanted to create a 64-bit Adder?
  - There is an easier way

```vhdl
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); -- C is an internal signal
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;
```
entity Adder is

    generic (N : positive := 8);

    port (  
        A, B : in std_logic_vector (N-1 downto 0);
        Cin : in std_logic;
        S : out std_logic_vector (N-1 downto 0);
        Cout : out std_logic
    );

end Adder;

Architecture Generated of Adder is
    --Component declarations
    component Full_Adder is
        port(
            A : in STD_LOGIC;
            B : in STD_LOGIC;
            Cin : in STD_LOGIC;
            S : out STD_LOGIC;
            Cout : out STD_LOGIC
        );
    end component;
    --Internal signals
    signal Cc : std_logic_vector (N downto 0); -- Carry Chain

    begin
        Cc(0) <= Cin;
        Full_Adders: for i in 0 to N-1 generate
            FAx: Full_Adder port map (A(i), B(i), Cc(i), S(i), Cout(i));
        end generate;

        Cout <= Cc(N);
    end Generated;
-- Implementation Variant 1

\[
\begin{align*}
X(0) &\leq A(0); \\
X(1) &\leq A(1) \text{ XOR } A(0); \\
X(2) &\leq A(2) \text{ XOR } (A(1) \text{ OR } A(0)); \\
X(3) &\leq A(3) \text{ XOR } (A(2) \text{ OR } A(1) \text{ OR } A(0)); \\
X(4) &\leq A(4) \text{ XOR } (A(3) \text{ OR } A(2) \text{ OR } A(1) \text{ OR } A(0));
\end{align*}
\]

-- Implementation Variant 2

```vhdl
signal ORc : std_logic_vector (4 downto 2);
signal XORc : std_logic_vector (4 downto 1);
begin
X(0) <= A(0);

XORc(1) <= A(0) XOR A(1);
X(1) <= XORc(1);

ORc(2) <= XORc(1) OR A(0);
XORc(2) <= ORc(2) XOR A(2);
X(2) <= XORc(2);

ORc(3) <= XORc(2) OR ORc(2);
XORc(3) <= ORc(3) XOR A(3);
X(3) <= XORc(3);

ORc(4) <= XORc(3) OR ORc(3);
XORc(4) <= ORc(4) XOR A(4);
X(4) <= XORc(4);
```

architecture Generated of TC is

begin

--Internal signals

signal ORc : std_logic_vector(N - 1 downto 2); -- OR Chain
signal XORc : std_logic_vector(N - 1 downto 1); -- XOR Chain

XORc(1) <= A(0) xor A(1);
X(1)    <= XORc(1);

ORc(2)  <= XORc(1) or A(0);
XORc(2)  <= ORc(2) xor A(2);
X(2)    <= XORc(2);

TCx : for i in 3 to N - 1 generate
    ORc(i) <= XORc(i - 1) or ORc(i - 1);
    XORc(i) <= ORc(i) xor A(i);
    X(i)    <= XORc(i);
end generate;

end Generated;
6. Dataflow MUX Design

-- conditional signal assignment statement

\[ F \leftarrow I_0 \text{ when } A = '0' \text{ else } I_1; \]

\[ F \leftarrow A \text{ when } E = '1' \\
\text{ else } B \text{ when } D = '1' \\
\text{ else } C; \]
With/Select for MUX Design

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

entity MUX2to1 is
  port(
    A, B : in std_logic;
    SEL : in std_logic;
    Z : out std_logic
  );
end MUX2to1;

architecture Dataflow of MUX2to1 is begin
  with SEL select Z <=
    A when '0',
    B when '1',
    '-' when others;
end Dataflow;
```

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

entity MUX2to1 is
  generic(N : positive := 4);
  port(
    A, B : in std_logic_vector(N - 1 downto 0);
    SEL : in std_logic;
    Z : out std_logic_vector(N - 1 downto 0)
  );
end MUX2to1;

architecture Dataflow of MUX2to1 is begin
  with SEL select Z <=
    A when '0',
    B when '1',
    (others => '-') when others;
end Dataflow;
```
Larger MUXs

library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity OutMUX is
  generic (N : positive := 4);
  port(
    A, B, C, D, E, F, G, H, I, J, K : in STD_LOGIC_VECTOR(N-1 downto 0);
    FNS : in STD_LOGIC_VECTOR(3 downto 0);
    Z : out STD_LOGIC_VECTOR(N-1 downto 0)
  );
end OutMUX;

architecture Dataflow of OutMUX is
begin
  with FNS select
  Z <=
    A when "0000",
    B when "0001",
    C when "0010",
    D when "0011",
    E when "0100",
    F when "0101",
    G when "0110",
    H when "0111",
    I when "1000",
    J when "1001",
    K when "1010",
    others => '-'
  when others;

  sel <= A&B;
  -- selected signal assignment statement
  with sel select
    F <= I0 when "00",
        I1 when "01",
        I2 when "10",
        I3 when "11",
        '-' when others;
end Dataflow;
7. More on Testbenches

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use ieee.numeric_std.all; --used for incrementing/converting stimulus

entity TC_tb is
end TC_tb;

architecture Testbench of TC_tb is

constant N : positive := 4; -- Set to size wished to be simulated

-- UUT
component TC is
Generic (N : positive := 4);
Port ( A : in STD_LOGIC_VECTOR (N-1 downto 0);
      Z : out STD_LOGIC_VECTOR (N-1 downto 0) );
end component;

--Stimulus signals
signal As, Zs : std_logic_vector (N-1 downto 0) := (others => '0');

begin

UUT : TC generic map(N) port map(As, Zs);

process
begin
  for i in 0 to ((2 ** N) - 1) loop
    wait for 10 ns;
    As <= std_logic_vector((unsigned(As) + 1)); -- Convert
    -- As to an unsigned integer, +1, and then convert it back
  end loop;

  wait; -- suspend process here (i.e. run only once)
end process;

end Testbench;
```
entity Adder_tb is

end Adder_tb;

architecture Testbench of Adder_tb is

constant N : positive := 2; -- Set to size wished to be simulated

-- Component declaration for UUT
component Adder is
generic(N : positive := 8);
port(
    A, B : in std_logic_vector(N - 1 downto 0);
    Cin : in std_logic;
    S : out std_logic_vector(N - 1 downto 0);
    Cout : out std_logic
);

end component;

-- Stimulus signals
signal As, Bs, Ss : std_logic_vector(N - 1 downto 0) := (others => '0');
signal Cins, Couts : std_logic := '0';

begin

UUT : Adder generic map(N) port map(As, Bs, Cins, Ss, Couts);

process
begin
    for i in 0 to ((2 ** N) - 1) loop
        for j in 0 to ((2 ** N) - 1) loop
            wait for 10 ns;
            Bs <= std_logic_vector(unsigned(Bs) + 1);
            end loop;
            As <= std_logic_vector(unsigned(As) + 1);
        end loop;
    Cins <= '1';
    for i in 0 to ((2 ** N) - 1) loop
        for j in 0 to ((2 ** N) - 1) loop
            wait for 10 ns;
            Bs <= std_logic_vector(unsigned(Bs) + 1);
            end loop;
            As <= std_logic_vector(unsigned(As) + 1);
        end loop;
        wait; -- suspend process here (i.e. run only once)
    end process;

end Testbench;
Lab 2 – ALU Design

• Let’s Practice