Digital Systems Design

Review of Combinatorial Circuit
Building Blocks:
VHDL for Combinational Circuits

Introduction to VHDL

• Designer writes a logic circuit description in VHDL source code
• VHDL compiler translates this code into a logic circuit
• Representation of digital signals in VHDL
  – Logic signals in VHDL are represented as a data object
  – VHDL includes a data type called \textit{BIT}
  – BIT objects can assume only two values: 0 and 1
Writing simple VHDL code

- First step in writing VHDL code is to declare the input and output signals
- Done using a construct called an entity

```
entity example1 is
  port (x1, x2, x3 : in bit;
        f      : out bit);
end example1;
```

- **Name of the entity**: `example1`
- **Input and output signals (ports) defined**
  - **Mode of the port**: IN (input)
  - **Type of the port**: OUT (output)

```
x1
x2
x3
```

```
f
```
Writing simple VHDL code

- The entity specifies the inputs and outputs for a circuit, but does not describe the circuit function.
- Circuit functionality is specified using a VHDL construct called an architecture.

<table>
<thead>
<tr>
<th>Architecture name</th>
<th>Entity used by logicFunc</th>
</tr>
</thead>
<tbody>
<tr>
<td>architecture logicfunc of example1 is</td>
<td></td>
</tr>
<tr>
<td>begin</td>
<td></td>
</tr>
<tr>
<td>f &lt;= (x1 and x2) or (not x2 and x3);</td>
<td></td>
</tr>
<tr>
<td>end logicfunc;</td>
<td></td>
</tr>
</tbody>
</table>

VHDL statement that describes the circuit functionality.

Complete VHDL code example

```vhdl
entity example1 is
port (x1,x2,x3 : in bit;
    f : out bit);
end example1;

architecture logicfunc of example1 is
begin
    f <= (x1 and x2) or (not x2 and x3);
end logicfunc;
```
Boolean operators in VHDL

- VHDL has built-in support for the following operators
  - and logical AND
  - or logical OR
  - not logical NOT
  - nand, nor, xor, xnor

- Assignment operator \(<=\)
  - A variable (usually an output, mode OUT) should be assigned the result of the logic expression on the right hand side of the operator.

- VHDL does not assume any precedence of logic operators. Use parentheses in expressions to determine precedence
  - Exception to this is NOT logical operator.

- In VHDL, a logic expression is called a *simple assignment statement*. There are other types that will be introduced that are useful for more complex circuits.

Assignment statements

- VHDL provides several types of statements that can be used to assign logic values to signals
  - Simple assignment statements
    - Used for simple logic or arithmetic expressions
  - Selected signal assignments
  - Conditional signal assignments
  - Generate statements
  - If-then-else statements
  - Case statements
Selected signal assignment

• A selected signal assignment allows a signal to be assigned one of several values, based on a selection criterion
  – Keyword WITH specifies that $s$ is used for the selection criterion
  – Two WHEN clauses state that $f = w_0$ when $s = 0$ and $f = w_1$ otherwise
  – The keyword OTHERS must be used

    ```vhdl
    architecture behavior of mux2to1 is
    begin
      with s select
        f <= w0 when '0',
           w1 when others;
    end behavior;
    ```

Design using VHDL

• In VHDL, a logic signal is represented as a data object
  – We used a bit data type before that could only take on the values 0 and 1
  – Another data type, std_logic, is actually preferable because it can assume several different values
    • [0, 1, Z (high impedance), - (don't care), etc]
  – The std_logic_vector data type can be used for multi-bit values

• We must declare the library where the data type exists, and declare that we will use the data type

    ```vhdl
    library ieee;
    use ieee.std_logic_1164.all;
    ```
### 4-to-1 multiplexer VHDL code

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

entity mux4to1 is
  port ( w : in std_logic_vector(3 downto 0);
         s : in std_logic_vector(1 downto 0);
         f : out std_logic);
end mux4to1;

architecture behavior of mux4to1 is
begin
  with s select
  f <= w(0) when "00",
       w(1) when "01",
       w(2) when "10",
       w(3) when others;
end behavior;
```

### 2-to-4 binary decoder VHDL code

```
entity dec2to4 is
  port ( w : in std_logic_vector(1 downto 0);
         en : in std_logic;
         y : out std_logic_vector(0 to 3));
end dec2to4;

architecture behavior of dec2to4 is
begin
  enw <= en & w;  -- '&' is the vhdl concatenate operator
  with enw select
    y <= "1000" when "100",
        "0100" when "101",
        "0010" when "110",
        "0001" when "111",
        "0000" when others;
end behavior;
```
Conditional signal assignment

- Similar to the selected signal assignment, a **conditional signal assignment** allows a signal to be set to one of several values
  - Uses `when` and `else` keyword to define the condition and actions

```vhdl
entity mux2to1 is
  port (w0, w1, s : in std_logic;
        f : out std_logic);
end mux2to1;
architecture behavior of mux2to1 is
begin
  f <= w0 when s = '0' else w1;
end behavior;
```

Priority encoder VHDL code

```vhdl
entity priority is
  port (w : in std_logic_vector(3 downto 0);
        y : out std_logic_vector(1 downto 0);
        z : out std_logic);
end priority;
architecture behavior of priority is
begin
  y <= "11" when w(3) = '1' else
       "10" when w(2) = '1' else
       "01" when w(1) = '1' else "00";
  z <= '0' when w = "0000" else '1';
end behavior;
```
**Behavioral Versus Structural VHDL**

- The previous VHDL code examples are termed **behavioral** VHDL because they describe the behavior of a circuit without describing exactly how it is implemented in hardware.
- Another VHDL coding style is **structural**.
  - For structural VHDL, the user typically describes the structure of a design by interconnecting several simpler designs from a library
    - The library components may be user-defined or provided by the CAD tool vendor.

---

**Single Bit Full Adder and Package Definition**

```vhdl
library ieee;
use ieee.std_logic_1164.all;
entity fulladd is
  port ( cin, x, y : in std_logic;
         s, cout : out std_logic);
end fulladd;

architecture logicfunc of fulladd is
begin
  s <= x xor y xor cin;
  cout <= (x and y) or (cin and x) or (cin and y);
end logicfunc;
```
Single Bit Full Adder and Package Definition

library ieee;
use ieee.std_logic_1164.all;
package fulladd_package is
    component fulladd
        port ( cin, x, y : in std_logic;
              s, cout     : out std_logic);
    end component;
end fulladd_package;

4-bit ripple carry adder

library ieee;
use ieee.std_logic_1164.all;
use work.fulladd_package.all;
entity adder4 is
    port ( cin : in std_logic;
           x, y : in std_logic_vector(3 downto 0);
           s, cout     : out std_logic_vector(3 downto 0));
end adder4;

architecture structure of adder4 is
    signal c : std_logic_vector(1 to 3);
begin
    stage0: fulladd port map ( cin, x(0), y(0), s(0), c(1) );
    stage1: fulladd port map ( c(1), x(1), y(1), s(1), c(2) );
    stage2: fulladd port map ( c(2), x(2), y(2), s(2), c(3) );
    stage3: fulladd port map ( c(3), x(3), y(3), s(3), cout );
end structure;
Generate statements

- Whenever we write structural VHDL code, we often create *instances* of a particular *component*
  - A multi-stage ripple carry adder made from a number of single-bit full adders might be an example
- If we need to create a large number of instances of a component, a more compact form is desired
- VHDL provides a feature called the *for generate* statement
  - This statement provides a loop structure for describing regularly structured hierarchical code

4-bit ripple carry adder

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use work.fulladd_package.all;
entity adder4 is
  port ( cin : in std_logic;
         x, y : in std_logic_vector(3 downto 0);
         s : out std_logic_vector(3 downto 0);
         cout : out std_logic);
end adder4;
architecture structure of adder4 is
  signal c : std_logic_vector(0 to 4);
begn
  c(0) <= cin;
cout <= c(4);
g1: for i in 0 to 3 generate
    stages: fulladd port map (c(i), x(i), y(i), s(i), c(i+1));
  end generate;
end structure;
```
Process statement

- We have introduced several types of assignment statements
  - All have the property that the order in which they appear in VHDL code does not affect the meaning of the code
- Because of this property, these statements are called **concurrent assignment statements**
- VHDL provides a second category of statements, **sequential assignment statements**, for which the ordering of the statements may affect the meaning of the code
  - If-then-else and case statements are sequential
- VHDL requires that sequential assignment statements be placed inside another statement, the **process** statement

---

Process statement

- The process statement, or simply process, begins with the **PROCESS** keyword, followed by a parenthesized list of signals called the **sensitivity list**
  - This list includes, at most, all the (input) signals used inside the process
  - There may be no signals in the sensitivity list (i.e., the list may not exist)
  - Generally, the list includes all signals that can be used to “activate” the process
- Statements inside the process are evaluated in sequential order
  - This is true from a simulation standpoint
  - From a synthesized hardware point-of-view, multiple assignments to a single signal (variable) generally implies multiplexing of the assignments to produce a single output
- Assignments made inside the process are not visible outside the process until all statements in the process have been evaluated
  - If there are multiple assignments to the same signal inside a process, only the last one has any visible effect
2-to-1 MUX as a PROCESS

```
architecture behavior of mux2to1 is
begin
  process (w0, w1, s)
  begin
    if s = '0' then
      f <= w0;
    else
      f <= w1;
    end if;
  end process;
end behavior;
```

Sensitivity list, Whenever a list entry changes, the process is reevaluated (activated)

IF-THEN-ELSE statement to implement the MUX function

Priority encoder (IF-THEN-ELSE)

```
architecture behavior of priority is
begin
  process (w)
  begin
    if w(3) = '1' then
      y <= "11";
    elsif w(2) = '1' then
      y <= "10";
    elsif w(1) = '1' then
      y <= "01";
    else
      y <= "00";
    end if;
  end process;
  z <= '0' when w = "0000" else '1';
end behavior;
```
Priority encoder (alternative)

```
architecture behavior of priority is
begin
    process (w)
    begin
        y <= "00";
        if w(1) = '1' then y <= "01" end if;
        if w(2) = '1' then y <= "10" end if;
        if w(3) = '1' then y <= "11" end if;

        z <= '1';
        if w = "0000" then z <= '0' end if;
    end process;
end behavior;
```

Implied memory in a PROCESS

```
architecture behavior of c1 is
begin
    process (a, b)
    begin
        aeqb <= '0';
        if a = b then
            aeqb <= '1';
        end if;
    end process;
end behavior;
```

```
architecture behavior of c1 is
begin
    process (a, b)
    begin
        if a = b then
            aeqb <= '1';
        end if;
    end process;
end behavior;
```
Case statement

- A **case statement** is similar to a selected assignment statement in that the case statement has a selection signal and includes WHEN clauses for various valuations of the selection signal
  - Begins with a **CASE** keyword
  - Each **WHEN** clause specifies the statements that should be evaluated when the selection signal has a specified value
  - The case statement must include a when clause for all valuations of the selection signal
    - Use the **OTHERS** keyword

---

2-to-1 MUX with CASE

```vhdl
architecture behavior of mux2to1 is
begin
  process (w0,w1,s)
  begin
    case s is
      when '0' => f <= w0;
      when others => f <= w1;
    end case;
  end process;
end behavior;
```
2-to-4 binary decoder with CASE

architecture behavior of dec2to4 is
begin
  process (w, en)
  begin
    if en = '1' then
      case w is
        when "00" => y <= "1000";
        when "01" => y <= "0100";
        when "10" => y <= "0010";
        when others => y <= "0001";
      end case;
    else
      y <= "0000";
    end if;
  end process;
end behavior;