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 declaration**

```
ENTITY example1 IS
  PORT (x1, x2, x3 : IN BIT;
        f              : OUT BIT);
END example1;
```

**Explanation**

- **Name of the entity**: `example1`
- **Input and output signals (ports)**
  - `x1`, `x2`, `x3`: IN (input) BIT
  - `f`: OUT (output) BIT

**Diagram**

- **In** (input) ports: `x1`, `x2`, `x3`
- **Out** (output) port: `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**

```
ARCHITECTURE LogicFunc OF example1 IS
BEGIN
  f <= (x1 AND x2) OR (NOT x2 AND x3);
END LogicFunc;
```

**Architecture name** ➔ **Entity used by LogicFunc**

```
ARCHITECTURE LogicFunc OF example1 IS
BEGIN
  f <= (x1 AND x2) OR (NOT x2 AND x3);
END LogicFunc;
```

**VHDL statement that describes the circuit functionality**

Complete VHDL code example

```
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;
```

**Entity** ➔ **Port declaration** ➔ **Architecture definition**

```
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;
```

**Entity declaration** ➔ **Architecture** ➔ **VHDL statement**
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
• 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 previously, for 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
    - \([0, 1, Z \text{ (high impedance)}, - \text{ (don't care)}, \text{ etc}]\)
  - 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
    SIGNAL Enw : STD_LOGIC_VECTOR(2 DOWNTO 0);
BEGIN
    Enw <= En & w; -- '&' is the VHDL concatenate operator
    WITH Enw SELECT
    y <= "1000" WHEN "00",
        "0100" WHEN "01",
        "0010" WHEN "10",
        "0001" WHEN "11",
        "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

```
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;

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 : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
       Cout : OUT STD_LOGIC ) ;
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

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);
BEGIN
  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 signals used inside the process
  - Generally the list includes all signals that can be used to "activate" the process
- Statements inside the process are evaluated in sequential order
- 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 ;
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 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 ;
END Behavior ;

ARCHITECTURE Behavior OF c1 IS
BEGIN
  PROCESS ( A, B )
  BEGIN
    IF A = B THEN
      AeqB <= '1' ;
    END IF ;
    END PROCESS ;
  END Behavior ;
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

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 ;