VHDL
Introduction

- VHDL
  - VHSIC Hardware Description Language
- VHSIC
  - Very High Speed Integrated Circuit
VHDL Syntax

- Not case sensitive
- Semicolon (;) is used to indicate termination of a statement
- Two dashes (--) are used to indicate the start of a comment
Data Objects

- **Signals**
  - Logic signal or wire

- **Constants**
  CONSTANT width: INTEGER := 4;

- **Variables**
  - Only in processes or subprograms
  - To hold computation results or for index variables in loops
Data Object Names

- Made up of alphabetic, numeric, and/or underscore
- The first character must be a letter
- The last character cannot be an underscore
- Two underscores in succession are not allowed
- Uppercase and lowercase are equivalent
Data Object Values and Numbers

- Single bit: ‘0’ or ‘1’
- Multibit
  - “1001” (also used to denote a binary number)
- Integers can alternatively specified in decimal
Signal Data Objects

- Signals can be declared in
  - entity declaration
  - declarative section of an architecture
  - declarative section of a package

- Associated with a type
  SIGNAL signal_name: type_name ;
Signal Types

- BIT (possible values: ‘0’, ‘1’) and BIT VECTOR
  SIGNAL x1, x2: BIT;
  SIGNAL c: BIT VECTOR (1 TO 4);
  SIGNAL byte: BIT VECTOR (7 DOWNTO 0);
  - Include the following two statements to use either type
    LIBRARY ieee;
    USE ieee.std_logic_1164.all;
  SIGNAL x1, x2: STD_LOGIC;
  SIGNAL c: STD_LOGIC VECTOR (1 TO 4);
  - STD_LOGIC VECTOR can be used in conjunction with the std_logic_signed or std_logic_unsigned package for signed or unsigned signals
Signal Types

- SIGNED and UNSIGNED
  - defined in `std_logic_arith` package
- INTEGER
  - from $-(2^{31}-1)$ to $(2^{31}-1)$
  SIGNAL x: INTEGER RANGE -127 TO 127;
- BOOLEAN (possible values: TRUE, FALSE)
  SIGNAL flag: BOOLEAN;
- ENUMERATION
  TYPE state_type IS (stateA, stateBm stateC);
  SIGNAL y: state_type;
- ARRAY
  TYPE BIT_VECTOR IS ARRAY (NATURAL RANGE <> ) of BIT;
  TYPE byte IS ARRAY (7 DOWNTO 0) of STD_LOGIC;
  SIGNAL x: byte;
Strongly Typed

- VHDL is a strongly typed language
- Operators overloading is supported in which a function or procedure can be defined differently for different argument lists
Operators

From highest precedence to lowest precedence

- Miscellaneous: **, abs, not
- Multiplying: *, /, mod, rem
- Sign: +, -
- Adding: +, -, &
- Relational: =, /=, <, <=, >, >=
- Logical: AND, OR, NAND, NOR, XOR, XNOR
Design Unit

- Entity declaration
  - Specifies input and output signals and “generic” parameters
- Architecture
  - Gives circuit details
Entity Declaration

- Describe I/O and parameterized values
- Port declaration
  - Name
  - Mode
    - IN
    - OUT
    - INOUT
    - BUFFER: can appear in both the left and right sides of assignment statements
Example

LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY eqcomp4 IS PORT ( 
    a, b: IN STD_LOGIC_VECTOR (3 DOWNTO 0);
    equals: OUT STD_LOGIC);
END eqcomp4;
Architecture

- Always associated with an entity declaration
- Description styles
  - Behavioral
  - Dataflow
  - Structural
Behavioral Description

LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY eqcomp4 IS PORT ( 
    a, b: IN STD_LOGIC_VECTOR (3 DOWNTO 0); 
    equals: OUT STD_LOGIC); 
END eqcomp4;

ARCHITECTURE behavioral OF eqcomp4 IS 
BEGIN 
    comp: PROCESS (a, b) -- sensitivity list 
    BEGIN 
      IF a = b THEN equals <= '1'; 
      ELSE equals <= '0'; -- sequential assignment 
      END IF 
    END PROCESS comp; 
END behavioral;
Dataflow Descriptions

ARCHITECTURE dataflow OF eqcomp4 IS
BEGIN
  equals <= '1' WHEN ( a = b ) ELSE '0';
END dataflow;
-- No process
-- Concurrent assignment
Structural Descriptions

USE work.gatespkg.all;
ARCHITECTURE struct OF eqcomp4 IS
    SIGNAL x : STD_LOGIC_VECTOR (0 TO 3);
BEGIN
    u0: xnor2 PORT MAP (a(0), b(0), x(0)); -- component
        -- instantiation
    u1: xnor2 PORT MAP (a(1), b(1), x(1));
    u2: xnor2 PORT MAP (a(2), b(2), x(2));
    u3: xnor2 PORT MAP (a(3), b(3), x(3));
    u4: and4 PORT MAP (x(0), x(1), x(2), x(3), equals);
END struct;
Packages

■ Consist of
  - Declaration section (visible to package users): declaring exportable subprograms, constants, types, components, etc.
  - Package body (optional): defining subprogram implementations along with any internally-used constants and types, etc.

■ Made visible through “USE”
  USE library_name.package_name.item;
  USE work.std_arith.all;

■ Some vendors provide a default work library
Example: Package Declaration

PACKAGE cs4120 IS
   CONSTANT maxint: INTEGER := 16#fff#
   TYPE arith_mode_type IS (SIGNED, UNSIGNED);
   FUNCTION minimum (CONSTANT a,b: IN INTEGER) RETURN INTEGER;
END cs4120;
Example: Package Body

PACKAGE BODY cs4120 IS
  FUNCTION minimum (CONSTANT a,b: INTEGER) RETURN INTEGER IS
    VARIABLE c: INTEGER; -- local variable
    BEGIN
      IF a < b THEN
        c := a; -- a is min
      ELSE
        c := b; -- b is min
      END IF;
      RETURN c; -- return min value
    END;
  END cs4120;
Example: Full Adder

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 (x AND Cin) or (y AND Cin) ;
END LogicFunc ;
Component Instantiation

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY adder 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 adder;

ARCHITECTURE Structure OF adder IS
  SIGNAL C : STD_LOGIC_VECTOR(1 TO 3);
  COMPONENT fulladd
    PORT ( Cin, x, y: IN STD_LOGIC;
        s, Cout: OUT STD_LOGIC) ;
  END COMPONENT;
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 ( x => X(3), y => Y(3), Cin => C(3), s => S(3), Cout => Cout ) ;
END Structure ;
Declaring a Component in a Package

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;
Using a Component Defined in a Package

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE work.fulladd_package.all;

ENTITY adder 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 adder;

ARCHITECTURE Structure OF adder 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;
Concurrent Statements

- To assign a value to a signal in an architecture body
- outside of a process
- Four types
  - Simple
  - Selected (WITH-SELECT-WHEN)
  - Conditional (WHEN-ELSE)
  - Generate
Simple Signal Assignment

- General form
  
  signal_name <= expression;

- Example: 4-bit adder

  LIBRARY ieee;
  USE ieee.std_logic_1164.all;
  USE ieee.std_logic_signed.all;

  ENTITY adder 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 adder;

  ARCHITECTURE Behavior OF adder IS
    SIGNAL Sum : STD_LOGIC_VECTOR(4 DOWNTO 0);
  BEGIN
    Sum <= ('0' & X) + Y + Cin;
    S <= Sum(3 DOWNTO 0);
    Cout <= Sum(4);
  END Behavior;

CS 4120
Selected Signal Assignment

ENTITY mux IS PORT(
  a, b, c, d: IN BIT_VECTOR(3 DOWNTO 0);
  s: IN BIT_VECTOR(1 DOWNTO 0);
  x: OUT BIT_VECTOR(3 DOWNTO 0))
END mux;

ARCHITECTURE archmux OF mux IS
BEGIN
  WITH s SELET C
  x <= a WHEN "00",
       b WHEN "01",
       c WHEN "10",
       d WHEN OTHERS;
END archmux;

The conditions after each WHEN clauses must be mutually exclusive.
Conditional Signal Assignment

ENTITY mux IS PORT (
    a, b, c, d: IN BIT_VECTOR(3 DOWNTO 0);
    s:     IN BIT_VECTOR(1 DOWNTO 0);
    x:     OUT BIT_VECTOR(3 DOWNTO 0));
END mux;

ARCHITECTURE archmux OF mux IS
BEGIN
    x <= a WHEN ( s = "00" ) ELSE
        b WHEN ( s = "01" ) ELSE
        c WHEN ( s = "10" ) ELSE
        d;
END archmux;

The conditions after each WHEN clauses need not be mutually exclusive.
Example: Priority Encoder

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY priority IS
  PORT ( req1, req2, req3: IN STD_LOGIC ;
        f: OU STD_LOGIC_VECTOR(1 DOWNTO 0) ) ;
END priority ;

ARCHITECTURE Behavior OF priority IS
BEGIN
  f <= "01" WHEN req1 = '1' ELSE
       "10" WHEN req2 = '1' ELSE
       "11" WHEN req3 = '1' ELSE
       "00" ;
END Behavior;
Generate Statement

- Repeating either a logic equation or a component instantiation

- Two variants
  - FOR GENERATE
    
    ```
    generate_label:
    FOR index_variable IN range GENERATE
    statement;
    {statements;}
    END GENERATE;
    ```
  
  - IF GENERATE (seldom needed)
    
    ```
    generate_label:
    IF expression GENERATE
    statement;
    {statements;}
    END GENERATE;
    ```
Component Instantiation with FOR GENERATE

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE work.fulladd_package.all;

ENTITY adder 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 adder;

ARCHITECTURE Structure OF adder IS
  SIGNAL C: STD_LOGIC_VECTOR(0 TO 4);
BEGIN
  C(0) <= Cin;
  Generate_label:
  FOR i IN 0 TO 3 GENERATE
    bit: fulladd PORT MAP ( C(i), X(i), Y(i), S(i), C(i+1) );
  END GENERATE;
  Cout <= C(4);
END Structure;

Defining an Entity with GENERIC

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE work.fulladd_package.all;

ENTITY addern IS
    GENERIC (n: INTEGER := 4);
    PORT (Cin : IN STD_LOGIC;
        X, Y : IN STD_LOGIC_VECTOR(n-1 DOWNTO 0);
        S : OUT STD_LOGIC_VECTOR(n-1 DOWNTO 0);
        Cout: OUT STD_LOGIC);
END addern;

ARCHITECTURE Structure OF addern IS
    SIGNAL C : STD_LOGIC_VECTOR(0 TO n);
BEGIN
    C(0) <= Cin;
    Generate_label:
    FOR i IN 0 TO n-1 GENERATE
        stage: fulladd PORT MAP ( C(i), X(i), Y(i), S(i), C(i+1) );
    END GENERATE;
    Cout <= C(4);
END Structure;
Sequential Statements

- In a process, function or procedure
- IF-THEN-ELSE
- CASE-WHEN
- FOR-LOOP and WHILE-LOOP
IF-THEN-ELSE

...  
IF (condition1) THEN  
  x <= value1;  
ELSIF (condition2) THEN  
  x <= value2;  
ELSE  
  x <= value3;  
END IF;
CASE-WHEN

... CASE address IS
  WHEN "001" => decode <= X"11";
  WHEN "111" => decode <= X"42";
  WHEN "010" => decode <= X"44";
  WHEN "101" => decode <= X"88";
  WHEN OTHERS => decode <= X"00";
END CASE;
LOOP

- Two types
  - FOR-LOOP
    [loop_label:]
    FOR var_name IN range LOOP
      statement;
    {statement;}
    END LOOP [loop_label];
  - WHILE-LOOP
    [loop_label:]
    WHILE boolean_exp LOOP
      statement;
    {statement;}
    END LOOP [loop_label];
Using PROCESS

... PROCESS (sel, x1, x2) BEGIN
  IF sel = '0' THEN
    f <= x1;
  ELSE
    f <= x2;
  END IF;
END PROCESS;

... PROCESS (sel, x1, x2) BEGIN
  f <= x1;
  IF sel = '1' THEN
    f <= x2;
  END IF;
END PROCESS;
Using VARIABLE

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY Numbits IS
  PORT (X: IN STD_LOGIC_VECTOR(1 TO 3);
       Count: OUT INTEGER RANGE 0 TO 3);
END Numbits;

ARCHITECTURE Behavior OF Numbits IS
BEGIN
  PROCESS (X) -- count the number of bits in X equal to 1
  VARIABLE TMP : INTEGER;
  BEGIN
    TMP := 0;
    FOR i IN 1 TO 3 LOOP
      IF X(i) = '1' THEN
        TMP := TMP + 1;
      END IF;
    END LOOP;
    COUNT <= TMP;
  END PROCESS;
END Behavior;
n-input NAND: Using VARIABLE

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY NANDn IS
  GENERIC (n : INTEGER := 4);
  PORT (X : IN STD_LOGIC_VECTOR(1 TO n);
    f: OUT STD_LOGIC);
END NANDn;

ARCHITECTURE Behavior OF NANDn IS
BEGIN
  PROCESS (X)
  VARIABLE Tmp : STD_LOGIC;
  BEGIN
    Tmp := X(1);
    AND_bits: FOR i IN 2 TO n LOOP
      Tmp := Tmp AND X(i);
    END LOOP AND_bits;
    f <= NOT Tmp;
  END PROCESS;
END Behavior;
What is Wrong with using SIGNAL?

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY NANDn IS
    GENERIC ( n : INTEGER := 4 );
    PORT ( X : IN STD_LOGIC_VECTOR(1 TO n);  
           f: OUT STD_LOGIC ) ;
END NANDn ;

ARCHITECTURE Behavior OF NANDn IS
BEGIN
    PROCESS ( X )
    SIGNAL Tmp : STD_LOGIC;
    BEGIN
        Tmp <= X(1);
        AND_bits: FOR i IN 2 TO n LOOP
            Tmp <= Tmp AND X(i);
        END LOOP AND_bits;
        f <= NOT Tmp;
    END PROCESS;
END Behavior;
A Correct Way

LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY NANDn IS
    GENERIC ( n : INTEGER := 4 ) ;
    PORT ( X : IN STD_LOGIC_VECTOR(1 TO n ) ;
        f: OUT STD_LOGIC ) ;
END NANDn ;

ARCHITECTURE Behavior OF NANDn IS
    SIGNAL Tmp : STD_LOGIC_VECTOR(1 TO n ) ;
BEGIN
    Tmp <= ( OTHERS => '1' ) ;
    f <= '0' WHEN X = Tmp ELSE '1' ;
END Behavior ;
Sequential Circuits

- Combinational circuits can be described using either concurrent or sequential statements.
- Sequential circuits can be described only with sequential statements.
- Examples of sequential circuits to be discussed
  - D latch
  - D flip-flop
  - Register
  - Shift register
  - Counter
  - Finite state machine
D Latch

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY latch IS
    PORT (D, clk : IN STD_LOGIC;
          Q: OUT STD_LOGIC);
END latch;

ARCHITECTURE Behavior OF latch IS
BEGIN
    PROCESS (D, clk)
    BEGIN
        IF clk = '1' THEN
            Q <= D;
        END IF;
    END PROCESS;
END Behavior;
D Flip-Flop

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY flipflop IS
    PORT (D, Clock: IN STD_LOGIC;
          Q: OUT STD_LOGIC);
END flipflop;

ARCHITECTURE Behavior OF flipflop IS
BEGIN
    PROCESS (Clock)
    BEGIN
        IF Clock'EVENT AND Clock = '1' THEN
            Q <= D;
        END IF;
    END PROCESS;
END Behavior;
WAIT UNTIL

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY flipflop IS
    PORT (D, Clock : IN STD_LOGIC;
          Q: OUT STD_LOGIC);
END flipflop;

ARCHITECTURE Behavior OF flipflop IS
BEGIN
    PROCESS
    BEGIN
        WAIT UNTIL Clock = '1';
        Q <= D;
    END PROCESS;
END Behavior;
D Flip-Flop with Asynchronous Reset

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY flipflop IS
  PORT (D, Resetn, Clock : IN STD_LOGIC;
  Q: OUT STD_LOGIC );
END flipflop;

ARCHITECTURE Behavior OF flipflop IS
BEGIN
  PROCESS ( Resetn, Clock )
  BEGIN
    IF Resetn = '0' THEN
      Q <= '0' ;
    ELSIF Clock'EVENT AND Clock = '1' THEN
      Q <= D ;
    END IF ;
  END PROCESS ;
END Behavior ;
D Flip-Flop with Synchronous Reset

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY flipflop IS
  PORT ( D, Resetn, Clock: IN STD_LOGIC;
         Q: OUT STD_LOGIC ) ;
END flipflop;

ARCHITECTURE Behavior OF flipflop IS
BEGIN
  PROCESS
  BEGIN
    WAIT UNTIL Clock = '1' ;
    IF Resetn = '0' THEN
      Q <= '0' ;
    ELSE
      Q <= D ;
    END IF ;
  END PROCESS ;
END Behavior ;
Instantiating a D Flip-Flop

LIBRARY ieee;
USE ieee.std_logic_1164.all;
LIBRARY altera;
USE altera.maxplus2.all;

ENTITY flipflop IS
    PORT (D, Clock: IN STD_LOGIC;
          Resetn, Presetn: IN STD_LOGIC;
          Q: OUT STD_LOGIC);
END flipflop;

ARCHITECTURE Behavior OF flipflop IS
BEGIN
    Dff_instance: Dff PORT MAP (D, Clock, Resetn, Presetn, Q);
END Behavior;
4-bit Register

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY reg4 IS
  PORT (D: IN STD_LOGIC_VECTOR(3 DOWNTO 0);
         Resetn, Clock: IN STD_LOGIC;
         Q: OUT STD_LOGIC_VECTOR(3 DOWNTO 0))
END reg4;

ARCHITECTURE Behavior OF reg4 IS
BEGIN
  PROCESS (Resetn, Clock)
  BEGIN
    IF Resetn = '0' THEN
      Q <= "0000";
    ELSIF Clock'EVENT AND Clock = '1' THEN
      Q <= D;
    END IF;
  END PROCESS;
END Behavior;

CS 4120
n-bit Register

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY regn IS
  GENERIC ( n : INTEGER := 4 );
  PORT ( D: IN STD_LOGIC_VECTOR(n-1 DOWNTO 0);
         Resetn, Clock: IN STD_LOGIC;
         Q: OUT STD_LOGIC_VECTOR(n-1 DOWNTO 0) )
END regn ;

ARCHITECTURE Behavior OF regn IS
BEGIN
  PROCESS ( Resetn, Clock )
  BEGIN
    IF Resetn = '0' THEN
      Q <= (OTHERS => '0');
    ELSIF Clock'EVENT AND Clock = '1' THEN
      Q <= D;
    END IF;
  END PROCESS;
END Behavior ;
n-bit Register with Enable

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY regne IS
    GENERIC ( n : INTEGER := 4 );
    PORT ( D : IN STD_LOGIC_VECTOR(n-1 DOWNTO 0);
        Resetn: IN STD_LOGIC;
        E, Clock: IN STD_LOGIC;
        Q: OUT STD_LOGIC_VECTOR(n-1 DOWNTO 0) );
END regne;

ARCHITECTURE Behavior OF regne IS
BEGIN
    PROCESS ( Resetn, Clock )
    BEGIN
        IF Resetn = '0' THEN
            Q <= (OTHERS => '0');
        ELSIF Clock'EVENT AND Clock = '1' THEN
            IF E = '1' THEN
                Q <= D ;
            END IF;
        END IF;
    END PROCESS ;
END Behavior ;
4-bit Shifter Register

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY shift4 IS
  PORT (w, Clock: IN STD_LOGIC;
        Q: OUT STD_LOGIC_VECTOR(1 TO 4)));
END shift4;

ARCHITECTURE Behavior OF shift4 IS
  SIGNAL Sreg : STD_LOGIC_VECTOR(1 TO 4);
BEGIN
  PROCESS (Clock)
  BEGIN
    IF Clock'EVENT AND Clock = '1' THEN
      Sreg(4) <= w;
      Sreg(3) <= Sreg(4);
      Sreg(2) <= Sreg(3);
      Sreg(1) <= Sreg(2);
    END IF;
  END PROCESS;
  Q <= Sreg;
END Behavior;
4-bit Shifter Register using VARIABLE

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY shift4 IS
  PORT (w, Clock: IN STD_LOGIC;
        Q: OUT STD_LOGIC_VECTOR(1 TO 4));
END shift4;

ARCHITECTURE Behavior OF shift4 IS
BEGIN
  PROCESS (Clock)
  VARIABLE Sreg : STD_LOGIC_VECTOR(1 TO 4);
  BEGIN
    IF Clock'EVENT AND Clock = '1' THEN
      Sreg(4) := w;
      Sreg(3) := Sreg(4);
      Sreg(2) := Sreg(3);
      Sreg(1) := Sreg(2);
    END IF;
    Q <= Sreg;
  END PROCESS;
END Behavior;
4-bit Counter

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;

ENTITY count4 IS
  PORT ( Resetn: IN STD_LOGIC;
         E, Clock: IN STD_LOGIC;
         Q: OUT STD_LOGIC_VECTOR (3 DOWNTO 0) );
END count4;

ARCHITECTURE Behavior OF count4 IS
  SIGNAL Count : STD_LOGIC_VECTOR (3 DOWNTO 0);
BEGIN
  PROCESS ( Clock, Resetn )
  BEGIN
    IF Resetn = '0' THEN
      Count <= "0000";
    ELSIF (Clock'EVENT AND Clock = '1') THEN
      IF E = '1' THEN
        Count <= Count + 1;
      END IF;
    END IF;
  END PROCESS;
  Q <= Count;
END Behavior;
Accumulator Circuit

[Diagram of an accumulator circuit with labels for Resetn, Clock, and X, showing connections for E, Resetn, Counter, and Register with a Sum block and Result output.]
Component Declarations

LIBRARY ieee;
USE ieee.std_logic_1164.all;

PACKAGE components IS

  COMPONENT addern -- n-bit adder
    GENERIC ( n : INTEGER := 4 ) ;
    PORT ( Cin: IN STD_LOGIC ;
       X, Y: IN STD_LOGIC_VECTOR (n-1 DOWNTO 0) ;
       S: OUT STD_LOGIC_VECTOR (n-1 DOWNTO 0) ;
       Cout: OUT STD_LOGIC ) ;
  END COMPONENT ;

  COMPONENT regne -- n-bit register with enable
    GENERIC ( n : INTEGER := 4 ) ;
    PORT ( D: IN STD_LOGIC_VECTOR (n-1 DOWNTO 0) ;
       Resetn: IN STD_LOGIC ;
       E, Clock: IN STD_LOGIC ;
       Q: OUT STD_LOGIC_VECTOR (n-1 DOWNTO 0) ) ;
  END COMPONENT ;

CS 4120
Component Declarations (Cont’d)

COMPONENT count4  -- 4-bit counter with enable
    PORT (Resetn: IN STD_LOGIC ;
             E, Clock: IN STD_LOGIC ;
             Q: OUT STD_LOGIC_VECTOR (3 DOWNTO 0) ) ;
    END COMPONENT ;

COMPONENT NANDn  -- n-bit NAND gate
    GENERIC ( n : INTEGER := 4 ) ;
    PORT ( X: IN STD_LOGIC_VECTOR(1 TO n) ;
             f: OUT STD_LOGIC ) ;
    END COMPONENT ;

END components ;
ENTITY declaration

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE work.components.all;

ENTITY accum IS
  GENERIC ( k : INTEGER := 8 ) ;
  PORT ( Resetn, Clock : IN STD_LOGIC ;
         X : IN STD_LOGIC_VECTOR(k-1 DOWNTO 0) ;
         Result : BUFFER STD_LOGIC_VECTOR(k-1 DOWNTO 0) ) ;
END accum ;
Architecture Body

ARCHITECTURE Structure OF accum IS
  SIGNAL Sum : STD_LOGIC_VECTOR(k-1 DOWNTO 0) ;
  SIGNAL C : STD_LOGIC_VECTOR(3 DOWNTO 0) ;
  SIGNAL Zero_bit, Cout, Stop : STD_LOGIC ;
BEGIN
  Zero_bit <= '0' ;
  adder: adder
    GENERIC MAP ( n => k )
    PORT MAP ( Zero_bit, X, Result, Sum, Cout ) ;
  reg: regne
    GENERIC MAP ( n => k )
    PORT MAP ( Sum, Resetn, Stop, Clock, Result ) ;
  Counter: count4
    PORT MAP ( Clock, Resetn, Stop, C ) ;
  NANDgate: NANDn
    PORT MAP ( C, Stop ) ;
END Structure ;
Moore-type FSM

![Moore-type FSM Diagram]

- **Reset**
- **A/z = 0**
  - **w = 0**
  - **w = 1**
- **B/z = 0**
  - **w = 0**
  - **w = 1**
- **C/z = 1**
  - **w = 0**
  - **w = 1**
Entity Declaration

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY moore IS
    PORT ( Clock : IN STD_LOGIC;
           w : IN STD_LOGIC;
           Resetn : IN STD_LOGIC;
           z : OUT STD_LOGIC ) ;
END moore ;
Architecture Body

ARCHITECTURE Behavior OF moore IS
  TYPE State_type IS (A, B, C) ;
  SIGNAL y : State_type ;
BEGIN
  PROCESS ( Resetn, Clock )
  BEGIN
    IF Resetn = '0' THEN
      y <= A ;
    ELSIF (Clock'EVENT AND Clock = '1') THEN
      CASE y IS
        WHEN A =>
          IF w = '0' THEN
            y <= A ;
          ELSE
            y <= B ;
          END IF ;
      END CASE ;
  END PROCESS ;
END ;
Architecture Body (Cont’d)

WHEN B =>
    IF w = '0' THEN
        y <= A ;
    ELSE
        y <= C ;
    END IF ;
WHEN C =>
    IF w = '0' THEN
        y <= A ;
    ELSE
        y <= C ;
    END IF ;
    END CASE ;
    END IF ;
END PROCESS ;

z <= '1' WHEN y = C ELSE '0' ;
END Behavior ;
Alternative: using Two Processes

ARCHITECTURE Behavior OF moore IS
  TYPE State_type IS (A, B, C) ;
  SIGNAL y_present, y_next : State_type ;
BEGIN
  PROCESS ( w, y_present )
  BEGIN
    CASE y_present IS
      WHEN A =>
        WHEN '0' =>
          IF w = '0' THEN
            y_next <= A ;
          ELSE
            y_next <= B ;
          END IF ;
      WHEN B =>
        WHEN '0' =>
          IF w = '0' THEN
            y_next <= A ;
          ELSE
            y_next <= C ;
          END IF ;
      WHEN C =>
        WHEN '0' =>
          IF w = '0' THEN
            y_next <= A ;
          ELSE
            y_next <= C ;
          END IF ;
    END CASE ;
  END PROCESS ;
Alternative: using Two Processes (Cont’d)

PROCESS ( Clock, Resetn )
BEGIN
  IF Resetn = '0' THEN
    y_present <= A ;
    ELSIF (Clock'EVENT AND Clock = '1') THEN
      y_present <= y_next ;
  END IF ;
END PROCESS ;

  z <= '1' WHEN y_present = C ELSE '0' ;
END Behavior ;
Manual State Assignment

ARCHITECTURE Behavior OF moore IS
  TYPE State_type IS (A, B, C) ;
  ATTRIBUTE ENUM_ENCODING : STRING ;
  ATTRIBUTE ENUM_ENCODING OF State_type :
    TYPE IS "00 01 11" ;
  SIGNAL y_present, y_next : State_type ;
Mealy-type FSM
Entity Declaration

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY mealy IS
  PORT ( Clock, Resetn : IN STD_LOGIC;
         w : IN  STD_LOGIC;
         z : OUT STD_LOGIC )
  ;
END mealy ;
ARCHITECTURE Behavior OF mealy IS
  TYPE State_type IS (A, B) ;
  SIGNAL y : State_type ;
BEGIN
  PROCESS ( Resetn, Clock )
  BEGIN
    IF Resetn = '0' THEN
      y <= A ;
    ELSIF (Clock'EVENT AND Clock = '1') THEN
      CASE y IS
        WHEN A =>
          IF w = '0' THEN y <= A ;
          ELSE y <= B ;
          END IF ;
        WHEN B =>
          IF w = '0' THEN y <= A ;
          ELSE y <= B ;
          END IF ;
      END CASE ;
    END IF ;
  END PROCESS ;
Architecture Body (Cont’d)

PROCESS ( y, w )
BEGIN
  CASE y IS
    WHEN A =>
      z <= '0' ;
    WHEN B =>
      z <= w ;
    END CASE ;
  END PROCESS ;
END Behavior ;
Function

FUNCTION bl2bit ( a: BOOLEAN )
RETURN BIT IS
BEGIN
  IF a THEN RETURN '1';
  ELSE RETURN '0';
  END IF;
END bl2bit;
Using Function

ENTITY full_add IS PORT (  
a, b, carry_in: IN BIT;  
sum, carry_out: OUT BIT );
END full_add;

ARCHITECTURE fall_add OF full_add IS
  FUNCTION majority( a, b, c: BIT ) RETURN BIT IS
    BEGIN
      RETURN ( ( a AND b ) OR ( a AND c ) OR ( b AND c ) );
    END majority;
    BEGIN
      sum <= a XOR b XOR carry_in;
      carry_out <= majority( a, b, carry_in );
    END;
Procedure

PROCEDURE dff ( SIGNAL d: IN BIT_VECTOR;
    SIGNAL clk, rst: IN BIT;
    SIGNAL q: OUT
    BIT_VECTOR ) is

BEGIN
    IF rst = ‘1’ THEN q <= ( OTHERS => ‘0’ );
    ELSIF clk‘EVENT AND clk = ‘1’ THEN q <= d;
    END IF;
END PROCEDURE;
Defining Procedure in Package

PACKAGE myflop IS
    PROCEDURE dff ( SIGNAL d: IN BIT_VECTOR;
        SIGNAL clk, rst: IN BIT;
        SIGNAL q: OUT BIT_VECTOR );
END myflop;

PACKAGE BODY myflop IS
    PROCEDURE dff ( SIGNAL d: IN BIT_VECTOR;
        SIGNAL clk, rst: IN BIT;
        SIGNAL q: OUT BIT_VECTOR ) IS
    BEGIN
        IF rst = ‘1’ THEN q <= ( OTHERS => ‘0’ );
        ELSIF clk’EVENT AND clk = ‘1’ THEN q <= d;
        END IF;
    END PROCEDURE;
END myflop;
Using Procedure

ENTITY flop8 IS PORT (  
    clk, rst: IN BIT;  
    data_in: IN BIT_VECTOR( 7 DOWNTO 0 );  
    data: OUT BIT_VECTOR( 7 DOWNTO 0 ) );  
END flop8;

USE work.myflop.all;  
ARCHITECTURE archflop8 OF flop8 IS  
BEGIN  
    dff( data_in, clk, rst, data );  
END archflop8;