Concurrent Statements

- Included within architecture definitions and “block” statements
  - Signal assignment
  - Process statement
  - Procedure/function call
  - Component instantiation
  - Generate statement
  - Assertion
  - Block statement
Block Statement

- A group of related concurrent statements
- Can be used in representing designs in a hierarchical manner
- Examples (guarded blocks)

```
BLOCK (Enable = '1')
BEGIN
  Q <= GUARDED D after 5ns;
END BLOCK;

BLOCK (Enable = '1')
BEGIN
  A <= B WHEN GUARD = '1' ELSE 'Z';
END BLOCK;
```
Example

LIBRARY ieee;
USE ieee.std_logic_1164.all;
PACKAGE bit32 IS
  TYPE tw32 IS ARRAY(31 DOWNTO 0) OF std_logic;
END bit32;

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE work.bit32.all;
ENTITY cpu IS
  PORT( clk, interrupt : IN std_logic;
        addr : OUT tw32;
        data : INOUT tw32 );
END cpu;

ARCHITECTURE cpu_blk OF cpu IS
  SIGNAL ibus, dbus : tw32;
BEGIN
  ALU : BLOCK
    SIGNAL qbus : tw32;
    BEGIN
    -- alu behavior statements
    END BLOCK ALU;
  REG8 : BLOCK
    SIGNAL zbus : tw32;
    BEGIN
    REG1: BLOCK
      SIGNAL qbus : tw32;
      BEGIN
      -- reg1 behavioral statements
      END BLOCK REG1;
      -- more REG8 statements
    END BLOCK REG8;
END cpu_blk;
Sequential Statements

- Wait statement
- Simple signal assignment
- Variable assignment
- Procedure/function call
- Conditional statement
- Loop statement
- Procedure/function statement
- Assertion
Loop Termination Statements

- NEXT [WHEN condition];
  -- end current iteration
- EXIT [WHEN condition];
  -- end current loop
Shift/Rotate Operations

- Shift/rotate left/right logical/arithmetic operators
  - SLL, SRL, SLA, SRA, ROL, ROR
Object Attributes

- Return information about a signal or data type
- Signal condition attributes (for a signal S)
  - S'DELAYED(T) - value of S delayed by T time units
  - S'_STABLE(T) - true if no event on S over last T time units
  - S'QUIET(T) - true if S quiet for T time units
  - S'LAST_VALUE - value of S prior to latest change
  - S'LAST_EVENT - time at which S last changed
  - S'LAST_ACTIVE - time at which S last active
  - S'EVEN T - true if an event has occurred on S in current cycle
  - S'A CTIVE - true if signal S is active in the current cycle
  - S'TRANSACTIONS - bit value which toggles each time signal S changes
Object Attributes (Cont’d)

- Data type bounds (attributes of data type T)
  - T'BASE - base type of T
  - T'LEFT - left bound of data type T
  - T'RIGHT - right bound
  - T'HIGH - upper bound
  - T'LOW - lower bound
Object Attributes (Cont’d)

- Enumeration data types (variable/signal \( x \) of data type \( T \))
  - \( T'\text{POS}(x) \) - position number of value of \( x \) of type \( T \)
  - \( T'\text{VAL}(x) \) - value of type \( T \) whose position number is \( x \)
  - \( T'\text{SUCC}(x) \) - value of type \( T \) whose position is \( x+1 \)
  - \( T'\text{PRED}(x) \) - value of type \( T \) whose position is \( x-1 \)
  - \( T'\text{LEFTOF}(x) \) - value of type \( T \) whose position is left of \( x \)
  - \( T'\text{RIGHTOF}(x) \) - value of type \( T \) whose position is right of \( x \)
Object Attributes (Cont’d)

- Array indexes for an array A (Nth index of A)
  - For multi-dimensional arrays, Nth index must be indicated. N may be omitted for a one-dimensional array.
  - A’LEFT(N) - left bound of index N
  - A’RIGHT(N) - right bound of index N
  - A’HIGH(N) - upper bound of index N
  - A’LOW(N) - lower bound of index N
  - A’LENGTH(N) - number of values in range of index N
  - A’RANGE(N) - range: A’LEFT(N) TO A’RIGHT(N)
  - A’REVERSE_RANGE(N) – range: A’LEFT(N) DOWNTO A’RIGHT(N)
Textio Package

- A package of functions that read and write text files
- To make it visible: USE std.textio.all;
- Data types
  - TEXT: a file of character strings
  - LINE: one string from a text file
- Example Declarations
  - FILE prog: TEXT IS IN “file_name”; -- text file “file_name”
  - VARIABLE L: LINE;
- Reading values from a file
  - READLINE(F, L); -- read one line from text file F to line L
  - READ(L, value, good); -- read one value from line L into variable value
    --(good is TRUE if successful )
- Writing values to a file
  - WRITE(F, L); --write one line to text file F from line L
Configurations

- Bind component instances to entities
- Can be also used to specify generic values for component instantiated in the configured architecture
Default Configurations

LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY counter IS
  PORT(load, clear, clk : IN std_logic;
        data_in : IN INTEGER;
        data_out : OUT INTEGER);
END counter;

ARCHITECTURE count_255 OF
  counter IS
BEGIN
  PROCESS(clk)
    VARIABLE count : INTEGER := 0;
  BEGIN
    IF clear = '1' THEN
      count := 0;
    ELSE
      IF (clk'EVENT) AND (clk = '1') AND (clk'LAST_VALUE = '0')
        THEN
          IF (count = 255) THEN
            count := 0;
          ELSE
            count := count + 1;
          END IF;
        END IF;
  END IF;
data_out <= count;
END PROCESS;
END count_255;
ARCHITECTURE count_64k
   OF counter IS
BEGIN
   PROCESS(clk)
     VARIABLE count :
       INTEGER := 0;
   BEGIN
     IF clear = '1' THEN
       count := 0;
     ELSIF load = '1' THEN
       count := data_in;
     END IF;
     ELSE
       IF (clk'EVENT) AND
           (clk = '1') AND
           (clk'LAST_VALUE = '0')
       THEN
         IF (count = 65535) THEN
           count := 0;
         ELSE
           count := count + 1;
         END IF;
       END IF;
     END IF;
   data_out <= count;
   END PROCESS;
END count_64k;
Default Configurations (Cont’d)

CONFIGURATION small_count OF counter IS
   FOR count_255
   END FOR;
END small_count;

CONFIGURATION big_count OF counter IS
   FOR count_64k
   END FOR;
END big_count;
Component Configurations

LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY inv IS
  PORT( a : IN std_logic;
        b : OUT std_logic);
END inv;

ARCHITECTURE behave OF inv IS
BEGIN
  b <= NOT(a) AFTER 5 ns;
END behave;

CONFIGURATION invcon OF inv IS
  FOR behave
  END FOR;
END invcon;
Component Configurations (Cont’d)

LIBRARY ieee; USE ieee.std_logic_1164.all;
ENTITY and3 IS
  PORT( a1, a2, a3 : IN std_logic;
       o1 : OUT std_logic);
END and3;

ARCHITECTURE behave OF and3 IS
BEGIN
  o1 <= a1 AND a2 AND a3 AFTER 5 ns;
END behave;

CONFIGURATION and3con OF and3 IS
  FOR behave
      END FOR;
END and3con;
Component Configurations (Cont’d)

LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY decode IS
  PORT( a, b, en : IN std_logic;
        q0, q1, q2, q3 : OUT std_logic);
END decode;

ARCHITECTURE structural OF decode IS
COMPONENT inv
  PORT( a : IN std_logic;
        b : OUT std_logic);
END COMPONENT;

COMPONENT and3
  PORT( a1, a2, a3 : IN std_logic;
        o1 : OUT std_logic);
END COMPONENT;

SIGNAL nota, notb : std_logic;
BEGIN
  I1 : inv
    PORT MAP(a, nota);
  I2 : inv
    PORT MAP(b, notb);
  A1 : and3
    PORT MAP(nota, en, notb, Q0);
  A2 : and3
    PORT MAP(a, en, notb, Q1);
  A3 : and3
    PORT MAP(nota, en, b, Q2);
  A4 : and3
    PORT MAP(a, en, b, Q3);
END structural;
Lower-Level Configurations

CONFIGURATION decode_llcon OF decode IS
FOR structural
   FOR I1 : inv USE CONFIGURATION work.invcon;
   END FOR;

   FOR I2 : inv USE CONFIGURATION work.invcon;
   END FOR;

   FOR ALL : and3 USE CONFIGURATION work.and3con;
   END FOR;
END FOR;
END decode_llcon;
Entity-Architecture Configurations

CONFIGURATION decode_eacon OF decode IS
  FOR structural
    FOR I1 : inv USE ENTITY work.inv(behave);
    END FOR;

    FOR OTHERS : inv USE ENTITY work.inv(behave);
    END FOR;

    FOR A1 : and3 USE ENTITY work.and3(behave);
    END FOR;

    FOR OTHERS : and3 USE ENTITY work.and3(behave);
    END FOR;

  END FOR;
END decode_eacon;
Port Maps

LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY inv IS
  PORT( x : IN std_logic;
        y : OUT std_logic);
END inv;

ARCHITECTURE behave OF inv IS
BEGIN
  y <= NOT(x) AFTER 5 ns;
END behave;

CONFIGURATION invcon OF inv IS
  FOR behave
  END FOR;
END invcon;
Port Maps (Cont’d)

CONFIGURATION decode_map_con OF decode IS
  FOR structural
    FOR I1 : inv USE ENTITY work.inv(behave)
      PORT MAP( x => a, y => b );
    END FOR;

    FOR I2 : inv USE ENTITY work.inv(behave)
      PORT MAP( x => a, y => b );
    END FOR;

    FOR ALL : and3 USE ENTITY work.and3(behave);
    END FOR;
  END FOR;
END decode_map_con;
Subprogram Overloading

LIBRARY ieee;
USE ieee.std_logic_1164.all;
PACKAGE p_shift IS
  TYPE s_int IS RANGE 0 TO 255;
  TYPE s_array IS ARRAY(0 TO 7) OF std_logic;

  FUNCTION shiftr( a : s_array) RETURN s_array;
  FUNCTION shiftr( a : s_int) RETURN s_int;
END p_shift;
Subprogram Overloading (Cont’d)

PACKAGE BODY p_shift IS
  FUNCTION shiftr( a : s_array )
    RETURN s_array IS
    VARIABLE result : s_array;
    BEGIN
      FOR i IN a'RANGE LOOP
        IF i = a'HIGH THEN
          result( i ) := '0';
        ELSE
          result( i ) := a( i + 1 );
        END IF;
      END LOOP;
    RETURN result;
  END shiftr;
END p_shift;

FUNCTION shiftr( a : s_int )
  RETURN s_int IS
  BEGIN
    RETURN ( a / 2 );
  END shiftr;
END p_shift;
Subprogram Overloading (Cont’d)

USE work.p_shift.all;
ENTITY shift_example IS
END shift_example;

ARCHITECTURE test OF
  shift_example IS
  SIGNAL int_signal : s_int;
  SIGNAL array_signal : s_array;
BEGIN
  -- picks function that works with
  -- s_int type
  int_signal <= shiftr(int_signal);
  -- picks function that works with
  -- s_array type
  array_signal <= shiftr(array_signal);
  -- produces error because no function
  -- will match
  array_signal <= shiftr(int_signal);
END test;
Overloading Operators

PACKAGE math IS
FUNCTION "+"( l,r : BIT_VECTOR)
    RETURN INTEGER;
END math;

PACKAGE BODY math IS
FUNCTION vector_to_int( S : BIT_VECTOR) RETURN INTEGER IS
    VARIABLE result : INTEGER := 0;
    VARIABLE prod : INTEGER := 1;
    BEGIN
        FOR i IN s'RANGE LOOP
            IF s(i) = '1' THEN
                result := result + prod;
            END IF;
            prod := prod * 2;
        END LOOP;
        RETURN result;
    END vector_to_int;
    FUNCTION "+"(l,r : BIT_VECTOR)
    RETURN INTEGER IS
    BEGIN
        RETURN ( vector_to_int(l) + vector_to_int(r));
    END;
END math;
Overloading Operators (Cont’d)

USE work.math.all;
ENTITY adder IS
  PORT( a, b : IN BIT_VECTOR(0 TO 7);
        c : IN INTEGER;
        dout : OUT INTEGER);
END adder;

ARCHITECTURE test OF adder IS
  SIGNAL internal : INTEGER;
BEGIN
  internal <= a + b;
  dout <= c + internal;
END test;
Comparison between Verilog and VHDL
History

- Verilog
  - Started by Gateway in 1983
  - Opened to public by Cadence in 1989
  - IEEE standard 1364 in 1995

- VHDL
  - Started by VHSIC project
  - IEEE standard 1076 in 1987
  - IEEE standard 1164 in 1993
Overview

- Equally effective
- Most EDA vendors support
- Personal preference

- Verilog is slightly better in gate/transistor level
- VHDL is slightly better in system
Data Types

- **Verilog**
  - Defined by Verilog
  - Simple and easy to use

- **VHDL**
  - Can be defined by user
  - Strongly typed (need conversion from one type to another)
Readability

- Verilog is similar to C or C++
- VHDL is similar to Ada or Pascal
High Level Constructs

- Verilog
  - Parameterized module

- VHDL
  - Package
  - Configuration
  - Generate
  - Generic
Design Reusability

- **Verilog**
  - Procedure/function local to module
  - Use ‘include compiler directive to share

- **VHDL**
  - Procedure/function can be placed in a package for sharing