Summary:

Entity Declaration:

ENTITY entity_name IS
    PORT ( signal_name, signal_name, ... : signal_mode signal_type;
           signal_name, signal_name, ... : signal_mode signal_type;
           ... signal_name, signal_name, ... : signal_mode signal_type);
END entity_name;

Signal types discussed in book (p.687, etc.)
    BIT, BIT_VECTOR, STD_LOGIC, STD_LOGIC_VECTOR, STD_ULOGIC,
    SIGNED, UNSIGNED, INTEGER, BOOLEAN, enumeration.

Signal modes: (Table A2, p. 695)
    • IN: input signal only
    • OUT: output signal only; cannot be used on the right side of the simple signal
          assignment operator <=
    • INOUT: a signal that is both an input and an output.
    • BUFFER: an output signal. However, it can be used on the right side of the
          simple signal assignment operator <=

Statement in the Architecture body can be structural or behavioral.

PORT MAP statement is the only structural type statement in VHDL, used to instantiate
a component that has been declared using a component declaration.

Simple signal assignment: signal_name <= expression;

Selected signal assignment: WITH-SLECT-WHEN statement

Conditional signal assignment: WHEN-ELSE statement

PROCESS statement (contains sequential statements)
    • Simple signal assignment statement <=
    • Variable assignment statement :=
    • IF-THEN-ELSE statement
    • CASE-WHEN statement
    • FOR-LOOP and WHILE-LOOP statements
**PROCESS statement** (contains **sequential** statements)
- Simple signal assignment statement $\leq$
- Variable assignment statement $:=
- IF-THEN-ELSE statement
- WAIT UNTIL statement
- CASE-WHEN statement
- FOR-LOOP and WHILE-LOOP statements

**Syntax:**
```
[process_label] PROCESS (sensitivity_list)
[VARIABLE declaration]
BEGIN
...
... sequential statements
...
END PROCESS [process_label];
```

- **Sensitivity list**: When there is a change in the value of any signal in the sensitivity list, the process becomes **“active”**.
- **Active**: when a process block becomes active, the statements inside the process block are **evaluated sequentially**:
  - at **“synthesis time”** (determine what circuit to build),
  - not at **“run-time”** (not like a program running inside a microprocessor).
Figure “7.37” Code for a D flip-flop

- **LamClk’EVENT**: EVENT is an “attribute” of the signal LamClk, which becomes “TRUE” when there is any change in the LamClk signal.
- When a condition in the form of “LamClk'EVENT AND LamClk = '1' ” is found in an IF statement, any signals that are assigned values inside the IF statement are implemented as the outputs of edge-triggered components (i.e., flip-flips).
LIBRARY ieee;
USE ieee.std_logic_1164.all;

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

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

Figure 7.39  D flip-flop with asynchronous reset
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY reg8 IS
  PORT ( D : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
         Resetn, Clock : IN STD_LOGIC;
         Q : OUT STD_LOGIC_VECTOR(7 DOWNTO 0) )
END reg8;

ARCHITECTURE Behavior OF reg8 IS
BEGIN
  PROCESS ( Resetn, Clock )
  BEGIN
    IF Resetn = '0' THEN
      Q <= "00000000" ;
    ELSIF Clock'EVENT AND Clock = '1' THEN
      Q <= D ;
    END IF ;
  END PROCESS ;
END Behavior ;

Figure 7.45 Code for an eight-bit register with asynchronous clear
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;
ENTITY upcount IS
  PORT ( Clock, Resetn, E : IN STD_LOGIC;
         Q : OUT STD_LOGIC_VECTOR (3 DOWNTO 0))
END upcount;

ARCHITECTURE Behavior OF upcount 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 ;
      ELSE
        Count <= Count ;
      END IF ;
    END IF ;
  END PROCESS ;
  Q <= Count ;
END Behavior ;

Figure 7.52 Code for a four-bit up-counter
Understanding PROCESS block

1. Identify all “outputs” (Z <= ….)
   - Outside of PROCESS block
   - Inside of PROCESS block
     - Inside or outside of the “scope of CLOCK ‘EVENT”

1. For each output signal, determine the type
   - Combinatorial
   - Edge-triggered flip-flop if inside the “scope” of clock’event (for IF and WAIT UNTIL)
   - Latch if “implied memory”

2. “Evaluation” vs. “assignment”
   - The statements inside the process block are evaluated sequentially:
     - at “synthesis time” (determine what circuit to build),
     - not at “run-time” (not like a program running inside a microprocessor).
   - If different values are assigned to the same output Z at different places in the PROCESS block, it will build the last circuit.
   - Remember, the bottom line is that the PROCESS statement results in some hardware. It is not a “program”.
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;

ENTITY upcountManyQs IS
  PORT ( Clock, Resetn, E : IN STD_LOGIC;
         Qa, Qb, QcBefore, QcAfter, Qd, Qe, Qf : OUT STD_LOGIC_VECTOR (3 DOWNTO 0)) ;
END upcountManyQs ;

ARCHITECTURE Behavior OF upcountManyQs IS
  SIGNAL Count : STD_LOGIC_VECTOR (3 DOWNTO 0) ;
BEGIN
  Qa <= Count ;
  PROCESS ( Clock, Resetn )
  BEGIN
    Qb <= Count ;
    IF Resetn = '0' THEN
      Count <= "0000" ;
    ELSIF (Clock'EVENT AND Clock = '1') THEN
      IF E = '1' THEN
        QcBefore <= Count;
        Count <= Count + 1 ;
        QcAfter <= Count ;
      ELSE
        Count <= Count ;
      END IF ;
    END IF ;
    Qd <= Count ;
    END IF ;
    Qe <= Count ;
    END PROCESS ;
  Qf <= Count ;
END Behavior ;
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'EVENT AND Clock = '1' ;
    Q <= D ;
  END PROCESS ;
END Behavior ;

Figure 7.38   Code for a D flip-flop using WAIT UNTIL

- A PROCESS statement without a sensitivity list can use a WAIT UNTIL statement as the first statement of the process.
- The process will not be active until the condition in the WAIT UNTIL statement is true.
- If the condition is in the form (Clock'EVENT AND Clock = '1') is used in the WAIT UNTIL statement, any signals that are assigned values inside the entire process are implemented as the outputs of flip-flips.
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY muxdff IS
    PORT ( D0, D1, Sel, Clock : IN STD_LOGIC;
           Q       : OUT STD_LOGIC );
END muxdff;

ARCHITECTURE Behavior OF muxdff IS
BEGIN
    PROCESS
    BEGIN
        WAIT UNTIL Clock'EVENT AND Clock = '1';
        IF Sel = '0' THEN
            Q <= D0;
        ELSE
            Q <= D1;
        END IF;
    END PROCESS;
END Behavior;

Figure 7.47 Code for a D flip-flop with a 2-to-1 multiplexer on the D input
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY shift4 IS
PORT ( R : IN STD_LOGIC_VECTOR(3 DOWNTO 0) ;
       L, w, Clock : IN STD_LOGIC ;
       Q : BUFFER STD_LOGIC_VECTOR(3 DOWNTO 0) ) ;
END shift4 ;

ARCHITECTURE Structure OF shift4 IS
  COMPONENT muxdff
    PORT ( D0, D1, Sel, Clock : IN STD_LOGIC ;
           Q : OUT STD_LOGIC ) ;
  END COMPONENT ;
BEGIN
  Stage3: muxdff PORT MAP ( w, R(3), L, Clock, Q(3) ) ;
  Stage2: muxdff PORT MAP ( Q(3), R(2), L, Clock, Q(2) ) ;
  Stage1: muxdff PORT MAP ( Q(2), R(1), L, Clock, Q(1) ) ;
  Stage0: muxdff PORT MAP ( Q(1), R(0), L, Clock, Q(0) ) ;
END Structure ;

Figure 7.48 Hierarchical code for a four-bit shift register
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY shift4 IS
  PORT ( R : IN STD_LOGIC_VECTOR(3 DOWNTO 0) ;
            Clock : IN STD_LOGIC ;
            L, w : IN STD_LOGIC ;
            Q   : BUFFER STD_LOGIC_VECTOR(3 DOWNTO 0) ) ;
END shift4 ;

ARCHITECTURE Behavior OF shift4 IS
BEGIN
  PROCESS
  BEGIN
    WAIT UNTIL Clock'EVENT AND Clock = '1' ;
    IF L = '1' THEN
      Q <= R ;
      ELSE
      Q(0) <= Q(1) ;
      Q(1) <= Q(2);
      Q(2) <= Q(3) ;
      Q(3) <= w ;
      END IF ;
    END PROCESS ;
  END Behavior ;

Figure 7.49 Alternative code for a shift register
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY shift4 IS
   PORT ( R : IN STD_LOGIC_VECTOR(3 DOWNTO 0); 
          Clock : IN STD_LOGIC;
          L, w : IN STD_LOGIC;
          Q : BUFFER STD_LOGIC_VECTOR(3 DOWNTO 0) );
END shift4;

ARCHITECTURE Behavior OF shift4 IS
BEGIN
   PROCESS
   BEGIN
      WAIT UNTIL Clock'EVENT AND Clock = '1';
      IF L = '1' THEN
         Q <= R ;
      ELSE
         Q(3) <= w ;
         Q(2) <= Q(3) ;
         Q(1) <= Q(2);
         Q(0) <= Q(1) ;
      END IF ;
   END PROCESS ;
END Behavior ;

Figure 7.50  Code that reverses the ordering of statements