Concurrent & Sequential Stmts. (Review)

- **Concurrent statements**
  - Signal assignment
  - Conditional signal assignment (WHEN-ELSE)
  - Selected signal assignment (WITH-SELECT-WHEN)
  - Process

- **Statements inside a process are executed sequentially**
  - Variables, arrays, queues
  - Variable assignments (no delay)
  - IF-THEN-ELSE, CASE-WHEN, LOOP
  - WAIT UNTIL, WAIT FOR, WAIT ON
  - **WARNING — not everything you do in a process may be synthesizable by your synthesis tools!**
Four Multiplexers (Review)

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY multiplexer IS
PORT( A, B, Mux_Control : IN STD_LOGIC;
      Mux_Out1, Mux_Out2, Mux_Out3, Mux_Out4 : OUT STD_LOGIC);
END multiplexer;

ARCHITECTURE behavior OF multiplexer IS BEGIN
-- selected signal assignment statement
Mux_Out1 <= A WHEN Mux_Control = '0' ELSE B;
-- / with Select Statement
WITH mux_control SELECT
Mux_Out2 <= A WHEN '0',
      B WHEN '1',
      OTHERS =>
END SELECT;
PROCESS ( A, B, Mux_Control ) BEGIN
IF Mux_Control = '0' THEN
Mux_Out3 <= A;
ELSE
Mux_Out3 <= B;
END IF;
CASE Mux_Control IS
WHEN '0' =>
Mux_Out4 <= A;
WHEN '1' =>
Mux_Out4 <= B;
WHEN OTHERS =>
END CASE;
END PROCESS;
END behavior;

Four Flip-Flops (Slide 1 of 2)

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY DFFs IS
PORT( D, Clock, Reset, Enable : IN STD_LOGIC;
      Q1, Q2, Q3, Q4 : OUT STD_LOGIC);
END DFFs;

ARCHITECTURE behavior OF DFFs IS BEGIN
PROCESS
-- Positive edge triggered D flip-flop
-- If WAIT is used no sensitivity list is used
BEGIN
WAIT UNTIL ( Clock 'EVENT AND Clock = '1' );
Q1 <= D;
END PROCESS;

Unassigned Outputs => Inferred Latches

- If an output is assigned a value in some paths through an IF or CASE statement, but is unassigned in at least one path, then a latch is added (“inferred”)

- This is a common mistake
  - Make sure all outputs are assigned a value, no matter what path is taken through the code, or give the signal a default value before the IF or CASE

- If no (new) value is assigned, the synthesis tools will assume you want the old value
  - A latch (register) will be placed on that output
  - Latches add delay, different paths may take different amounts of time, etc.
(Accidentally) Inferred Latches

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY latch IS
PORT( A, B : IN STD_LOGIC;
       Output1, Output2 : OUT STD_LOGIC );
END latch;
ARCHITECTURE behavior OF latch IS
BEGIN
   PROCESS ( A, B )
   BEGIN
      IF A = '0' THEN
         Output1 <= '0';
         Output2 <= '0';
      ELSE
         IF B = '1' THEN
            Output1 <= '1';
            Output2 <= '1';
         ELSE
            Output1 <= '0';
         END IF;
      END IF;
   END PROCESS;
END behavior;

8-Bit Counter

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY Counter IS
PORT( Clock, Reset : IN STD_LOGIC;
      Max_count : IN STD_LOGIC_VECTOR( 7 DOWNTO 0 );
      Count : OUT STD_LOGIC_VECTOR( 7 DOWNTO 0 ) );
END Counter;
ARCHITECTURE behavior OF Counter IS
BEGIN
   SIGNAL internal_count: STD_LOGIC_VECTOR( 7 DOWNTO 0 );
   count <= internal_count;
   PROCESS ( Reset,Clock )
   BEGIN
      IF reset = '1' THEN
         internal_count <= "00000000";
      ELSIF ( clock EVENT AND clock = '1' ) THEN
         IF internal_count < Max_count THEN
            internal_count <= internal_count + 1;
         ELSE
            internal_count <= "00000000";
         END IF;
      END IF;
   END PROCESS;
END behavior;
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY ALU IS
PORT( Op_code : IN STD_LOGIC_VECTOR( 2 DOWNTO 0 );
A_input, B_input : IN STD_LOGIC_VECTOR( 7 DOWNTO 0 );
ALU_output : OUT STD_LOGIC_VECTOR( 7 DOWNTO 0 ));
END ALU;

ARCHITECTURE behavior OF ALU IS
BEGIN

PROCESS ( Op_code, A_input, B_input )
BEGIN

CASE Op_Code ( 2 DOWNTO 1 ) IS
-- Select Arithmetic/Logical Operation
WHEN "00" =>
temp_output <= A_input + B_input;
WHEN "01" =>
temp_output <= A_input - B_input;
WHEN "10" =>
temp_output <= A_input AND B_input;
WHEN "11" =>
temp_output <= A_input OR B_input;
WHEN OTHERS =>
temp_output <= "00000000";
END CASE;

-- Select Shift Operation: Shift bits left with zero fill using concatenation operator
-- Can also use VHDL 1076-1993 shift operator such as SLL
IF Op_Code( 0 ) = '1' THEN
ALU_output <= temp_output( 6 DOWNTO 0 ) & '0';
ELSE
ALU_output <= temp_output;
END IF;
END PROCESS;

END behavior;

CASE Op_code ( 2 DOWNTO 1 ) IS
WHEN "00" =>
temp_output <= A_input + B_input;
WHEN "01" =>
temp_output <= A_input - B_input;
WHEN "10" =>
temp_output <= A_input AND B_input;
WHEN "11" =>
temp_output <= A_input OR B_input;
END CASE;

IF Op_Code( 0 ) = '1' THEN
ALU_output <= temp_output( 6 DOWNTO 0 ) & '0';
ELSE
ALU_output <= temp_output;
END IF;
END PROCESS;
END behavior;
State Machine Example (Slide 1 of 2)

```vhdl
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY st_mach IS
  PORT( clk, reset : IN STD_LOGIC;
        Input1, Input2 : IN STD_LOGIC;
        Output1 : OUT STD_LOGIC);
END st_mach;

ARCHITECTURE A OF st_mach IS

-- Enumerated Data Type for State

TYPE STATE_TYPE IS (state_A, state_B, state_C);
SIGNAL state: STATE_TYPE;
BEGIN
  PROCESS (reset, clk)
  BEGIN
    IF reset = '1' THEN
      -- Reset State
      state <= state_A;
    ELSIF clk 'EVENT AND clk = '1' THEN
      CASE state IS
        WHEN state_A =&gt;
          IF Input1 = '0' THEN
            state <= state_B;
          ELSE
            state <= state_C;
          END IF;
        WHEN state_B =&gt;
          state <= state_C;
        WHEN state_C =&gt;
          IF Input2 = '1' THEN
            state <= state_A;
          END IF;
        WHEN OTHERS =&gt;
          state <= state_A;
        END CASE;
      END CASE;
    END PROCESS;
  END PROCESS;
END;  
```

State Machine Example (Slide 2 of 2)

```vhdl
CASE state IS
  WHEN state_A =&gt;
    IF Input1 = '0' THEN
      state <= state_B;
    ELSE
      state <= state_C;
    END IF;
  WHEN state_B =&gt;
    state <= state_C;
  WHEN state_C =&gt;
    IF Input2 = '1' THEN
      state <= state_A;
    END IF;
  WHEN OTHERS =&gt;
    state <= state_A;
  END CASE;
END IF;
END CASE;
WITH state SELECT
  Output1 <= '0' WHEN state_A,
             '1' WHEN state_B,
             '0' WHEN state_C;
END a;
```