Required reading

• P. Chu, *RTL Hardware Design using VHDL*

Chapter 7, Combinational Circuit Design: Practice
Chapter 5.1, VHDL Process
Chapter 8, Sequential Circuit Design: Principle (except subchapter 8.6)
Slides for Chapter 8, available at http://academic.csuohio.edu/chu_p/rtl/rtl_hardware.html

Fixed Logical Shift Right in VHDL

SIGNAL A : STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL C : STD_LOGIC_VECTOR(3 DOWNTO 0);

\[
\begin{align*}
A & \rightarrow A(3,A(2),A(1),A(0)) \\
C & \leftarrow '0' \& A(3 \text{ downto } 1)
\end{align*}
\]

Fixed Arithmetic Shift Right in VHDL

SIGNAL A : STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL C : STD_LOGIC_VECTOR(3 DOWNTO 0);

\[
\begin{align*}
A & \rightarrow A(3,A(2),A(1),A(0)) \\
C & \leftarrow '0' \& A(3 \text{ downto } 1)
\end{align*}
\]
**Fixed Arithmetic Shift Right in VHDL**

SIGNAL A : STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL C : STD_LOGIC_VECTOR(3 DOWNTO 0);

\[
A(3) \quad A(2) \quad A(1) \quad A(0) \\
\text{C } \Leftarrow \text{A(3)} \& \text{A(3 downto 1)};
\]

---

**Fixed Logical Shift Left in VHDL**

SIGNAL A : STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL C : STD_LOGIC_VECTOR(3 DOWNTO 0);

\[
A(3) \quad A(2) \quad A(1) \quad A(0) \\
\text{C } \Leftarrow \text{A(2 downto 0) } \& \text{'}0';
\]

---

**Fixed Rotation Left in VHDL**

SIGNAL A : STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL C : STD_LOGIC_VECTOR(3 DOWNTO 0);

\[
A(3) \quad A(2) \quad A(1) \quad A(0) \\
\text{C } \Leftarrow \text{A(2 downto 0) } \& \text{A(3)};
\]

---

**Variable Rotators**
8-bit Variable Rotator Left

To be covered during the next class

2-to-1 Multiplexer

(a) Graphical symbol
(b) Truth table

VHDL:

f <= w0 WHEN s = '0' ELSE w1;
or
f <= w1 WHEN s = '1' ELSE w0;

Cascade of two multiplexers

VHDL:

f <= w1 WHEN s1 = '1' ELSE w2 WHEN s2 = '1' ELSE w3;
Decoders

### 4-to-1 Multiplexer

(a) Graphic symbol

```
\begin{array}{|c|c|}
\hline
s & f \\
\hline
00 & 0 \\
01 & 1 \\
10 & w_2 \\
11 & w_3 \\
\hline
\end{array}
```

(b) Truth table

```
\begin{array}{|c|c|c|}
\hline
s & s_1 & s_0 & r \\
\hline
00 & 0 & 0 & w_0 \\
01 & 0 & 1 & w_1 \\
10 & 1 & 0 & w_2 \\
11 & 1 & 1 & w_3 \\
\hline
\end{array}
```

WITH \( s \) SELECT
\( f \) := w_0 WHEN "00",
\( w_1 \) WHEN "01",
\( w_2 \) WHEN "10",
\( w_3 \) WHEN OTHERS;

---

### 2-to-4 Decoder

(a) Truth table

```
\begin{array}{|c|c|c|c|c|c|}
\hline
E & w & w_1 & y_0 & y_1 & y_2 & y_3 \\
\hline
1 & 0 & 0 & 1 & 0 & 0 & 0 \\
1 & 1 & 0 & 1 & 0 & 1 & 1 \\
1 & 1 & 1 & 0 & 1 & 1 & 0 \\
0 & x & x & 0 & x & x & x \\
\hline
\end{array}
```

(b) Graphical symbol

```
\begin{array}{|c|c|c|c|}
\hline
w & w_1 & y_0 & y_1 \\
\hline
0 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 \\
1 & 0 & 0 & 1 \\
1 & 1 & 1 & 0 \\
\hline
\end{array}
```

WITH \( E \) \& \( w \);

Eaw \( \leftarrow \) En \& w;

WITH Eaw SELECT
\( y \) := "0000" WHEN "100",
"0010" WHEN "101",
"0100" WHEN "110",
"1000" WHEN "111",
"0000" WHEN OTHERS;
VHDL code for a 2-to-4 Decoder entity

LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY dec2to4 IS
PORT (w : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
   En : IN STD_LOGIC;
   y : OUT STD_LOGIC_VECTOR(3 DOWNTO 0));
END dec2to4;
ARCHITECTURE dataflow OF dec2to4 IS
SIGNAL Enw : STD_LOGIC_VECTOR(2 DOWNTO 0);
BEGIN
   Enw <= En & w;
   WITH Enw SELECT
   y <= "0001" WHEN "100",
     "0010" WHEN "101",
     "0100" WHEN "110",
     "1000" WHEN "111",
     "0000" WHEN OTHERS;
END dataflow;

Priority Encoder

w3 w2 w1 w0 y1 y0 z
0 0 0 0 0 1 -
0 0 1 - 1 - 1
0 1 - - - 1 1
1 - - - - 0 0

vHDL code for a Priority Encoder entity

LIBRARY ieee;
USE ieee.std_logic_1164.all;
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 dataflow 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 dataflow;
Adders

VHDL code for an Adder mod 2\(^{16}\)

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

ENTITY adder16 IS
PORT (X : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
Y : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
S : OUT STD_LOGIC_VECTOR(15 DOWNTO 0));
END adder16;

ARCHITECTURE dataflow OF adder16 IS
BEGIN
S <= std_logic_vector(unsigned(X) + unsigned(Y));
END dataflow;

Signed and Unsigned Types

Behave exactly like \texttt{STD\_LOGIC\_VECTOR}
plus, they determine whether a given vector should be treated as a signed or unsigned number.

Require
\texttt{USE ieee.numeric_std.all;}

16-bit Unsigned Adder

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

ENTITY adder16 IS
PORT (Cin : IN STD\_LOGIC;
X : IN STD\_LOGIC\_VECTOR(15 DOWNTO 0);
Y : IN STD\_LOGIC\_VECTOR(15 DOWNTO 0);
S : OUT STD\_LOGIC\_VECTOR(15 DOWNTO 0);
Cout : OUT STD\_LOGIC);
END adder16;

Addition of Unsigned Numbers (1)
Addition of Unsigned Numbers (3)

ARCHITECTURE dataflow OF adder16 IS
    signal Sum: unsigned(16 DOWNTO 0); BEGIN
    Sum <= unsigned('0' & X) + unsigned(Y) + unsigned('0' & Cin) ;
    S <= std_logic_vector(Sum(15 downto 0));
    Cout <= Sum(16) ; END dataflow ;

Unsigned vs. Signed Multiplication

<table>
<thead>
<tr>
<th>Unsigned</th>
<th>Signed</th>
</tr>
</thead>
<tbody>
<tr>
<td>1111 15</td>
<td>1111 -1</td>
</tr>
<tr>
<td>x 1111 15</td>
<td>x 1111 x -1</td>
</tr>
<tr>
<td>11100001 225</td>
<td>00000001 1</td>
</tr>
</tbody>
</table>

8x8-bit Unsigned Multiplier

```
  a  b  *
  c  U
```

8x8-bit Signed Multiplier

```
  a  b  *
  c  S
```

8x8-bit Unsigned and Signed Multiplier

```
  a  b  *
  cu  cs
```
Multiplication of signed and unsigned numbers

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

entity multiply is
  port(
    a : in STD_LOGIC_VECTOR(7 downto 0);
    b : in STD_LOGIC_VECTOR(7 downto 0);
    cu : out STD_LOGIC_VECTOR(15 downto 0);
    cs : out STD_LOGIC_VECTOR(15 downto 0)
  );
end multiply;

architecture dataflow of multiply is
begin
  -- signed multiplication
  cs <= STD_LOGIC_VECTOR(SIGNED(a)*SIGNED(b));
  -- unsigned multiplication
  cu <= STD_LOGIC_VECTOR(UNSIGNED(a)*UNSIGNED(b));
end dataflow;
```

ROM 8x16 example (1)

```vhdl
ARCHITECTURE dataflow OF rom IS
SIGNAL temp: INTEGER RANGE 0 TO 7;
TYPE vector_array IS ARRAY (0 to 7) OF STD_LOGIC_VECTOR(15 DOWNTO 0);
CONSTANT memory : vector_array :=
  ( X"800A", X"D459", X"A870", X"7853", X"650D", X"642F", X"F742", X"F548" );
BEGIN
  temp <= to_integer(unsigned(Addr));
  Dout <= memory(temp);
END dataflow;
```

ROM 8x16 example (2)

```vhdl
ARCHITECTURE dataflow OF rom IS
SIGNAL temp: INTEGER RANGE 0 TO 7;
TYPE vector_array IS ARRAY (0 to 7) OF STD_LOGIC_VECTOR(15 DOWNTO 0);
CONSTANT memory : vector_array :=
  ( X"800A", X"D459", X"A870", X"7853", X"650D", X"642F", X"F742", X"F548" );
BEGIN
  Dout <= memory(to_integer(unsigned(Addr)));
END dataflow;
```

ROM 8x16 example (3)

```vhdl
ARCHITECTURE dataflow OF rom IS
SIGNAL temp: INTEGER RANGE 0 TO 7;
TYPE vector_array IS ARRAY (0 to 7) OF STD_LOGIC_VECTOR(15 DOWNTO 0);
CONSTANT memory : vector_array :=
  ( X"800A", X"D459", X"A870", X"7853", X"650D", X"642F", X"F742", X"F548" );
BEGIN
  Dout <= memory(to_integer(unsigned(Addr)));
END dataflow;
```

ROM 8x16 example (4)

```vhdl
ARCHITECTURE dataflow OF rom IS
SIGNAL temp: INTEGER RANGE 0 TO 7;
TYPE vector_array IS ARRAY (0 to 7) OF STD_LOGIC_VECTOR(15 DOWNTO 0);
CONSTANT memory : vector_array :=
  ( X"800A", X"D459", X"A870", X"7853", X"650D", X"642F", X"F742", X"F548" );
BEGIN
  Dout <= memory(to_integer(unsigned(Addr)));
END dataflow;
```
Tri-state Buffer entity (1)

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY tri_state IS
PORT ( e: IN STD_LOGIC;
       x: IN STD_LOGIC;
       f: OUT STD_LOGIC
    );
END tri_state;

Tri-state Buffer entity (2)

ARCHITECTURE dataflow OF tri_state IS
BEGIN
f <= x WHEN (e = '1') ELSE 'Z';
f <= not x WHEN (e = '0') ELSE 'Z';
END dataflow;
MLU Example

MLU: Entity Declaration

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY mlu IS
PORT(
  NEG_A : IN STD_LOGIC;
  NEG_B : IN STD_LOGIC;
  NEG_Y : IN STD_LOGIC;
  A :           IN STD_LOGIC;
  B :           IN STD_LOGIC;
  L1 :         IN STD_LOGIC;
  L0 :         IN STD_LOGIC;
  Y :          OUT STD_LOGIC
);
END mlu;

MLU: Architecture Declarative Section

ARCHITECTURE mlu_dataflow OF mlu IS

SIGNAL  A1 :  STD_LOGIC;
SIGNAL  B1 :  STD_LOGIC;
SIGNAL  Y1 : STD_LOGIC;
SIGNAL  MUX_0 : STD_LOGIC;
SIGNAL  MUX_1 : STD_LOGIC;
SIGNAL  MUX_2 : STD_LOGIC;
SIGNAL  MUX_3 : STD_LOGIC;
SIGNAL  L : STD_LOGIC_VECTOR(1 DOWNTO 0);

BEGIN
A1<= NOT A  WHEN (NEG_A='1') ELSE A;
B1<= NOT B  WHEN (NEG_B='1') ELSE B;
Y <= NOT Y1 WHEN (NEG_Y='1') ELSE Y1;
MUX_0 <= A1 AND  B1;
MUX_1 <= A1 OR  B1;
MUX_2 <= A1 XOR B1;
MUX_3 <= A1 XNOR B1;
L <= L1 & L0;
with (L) select
  Y1 <= MUX_0  WHEN "00",
       MUX_1  WHEN "01",
       MUX_2  WHEN "10",
       MUX_3  WHEN OTHERS;
END mlu_dataflow;

MLU - Architecture Body

BEGIN
A1<= NOT A  WHEN (NEG_A='1') ELSE A;
B1<= NOT B  WHEN (NEG_B='1') ELSE B;
Y <= NOT Y1 WHEN (NEG_Y='1') ELSE Y1;
MUX_0 <= A1 AND  B1;
MUX_1 <= A1 OR  B1;
MUX_2 <= A1 XOR B1;
MUX_3 <= A1 XNOR B1;
L <= L1 & L0;
with (L) select
  Y1 <= MUX_0  WHEN "00",
       MUX_1  WHEN "01",
       MUX_2  WHEN "10",
       MUX_3  WHEN OTHERS;
END mlu_dataflow;

Combinational Logic Synthesis for Beginners
Simple rules for beginners

For combinational logic, use only concurrent statements

- concurrent signal assignment (=)
- conditional concurrent signal assignment (when-else)
- selected concurrent signal assignment (with-select-when)

Simple rules for beginners

For circuits composed of
- simple logic operations (logic gates)
- simple arithmetic operations (addition, subtraction, multiplication)
- shifts/rotations by a constant

use

- concurrent signal assignment (=)

Example: VHDL code for a 4-to-1 MUX

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY mux4to1 IS
PORT ( w0, w1, w2, w3 : IN STD_LOGIC;
       s : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
       f : OUT STD_LOGIC )
END mux4to1;

ARCHITECTURE dataflow OF mux4to1 IS
BEGIN
  WITH s SELECT
  f <= w0 WHEN "00",
       w1 WHEN "01",
       w2 WHEN "10",
       w3 WHEN OTHERS;
END dataflow;

when-else vs. with-select-when (1)

"when-else" should be used when:
1) there is only one condition (and thus, only one else), as in the 2-to-1 MUX
2) conditions are independent of each other (e.g., they test values of different signals)
3) conditions reflect priority (as in priority encoder); one with the highest priority need to be tested first.

when-else vs. with-select-when (2)

"with-select-when" should be used when there is
1) more than one condition
2) conditions are closely related to each other (e.g., represent different ranges of values of the same signal)
3) all conditions have the same priority (as in the 4-to-1 MUX).
Left vs. right side of the assignment

- Left side  <=  Right side
  <=  when-else
  with-select <=

- Internal signals (defined in a given architecture)
- Ports of the mode
  - out
  - inout

Expressions including:
- Internal signals (defined in a given architecture)
- Ports of the mode
  - in
  - inout

Behavioral Design Style:
Registers & Counters

VHDL Description Styles

- dataflow
- structural
- behavioral

Concurrent statements
Components and interconnects
and more if you are careful

Processes in VHDL
- Processes Describe Sequential Behavior
- Processes in VHDL Are Very Powerful Statements
  - Allow to define an arbitrary behavior that may be difficult to represent by a real circuit
  - Not every process can be synthesized
- Use Processes with Caution in the Code to Be Synthesized
- Use Processes Freely in Testbenches

Anatomy of a Process

```vhdl
OPTIONAL
[label:] PROCESS [-(sensitivity list)]
  [declaration part]
BEGIN
  statement part
END PROCESS [label];
```

PROCESS with a SENSITIVITY LIST

```vhdl
label: process (sensitivity list)
  declaration part
begin
  statement part
end process;
```

- List of signals to which the process is sensitive.
- Whenever there is an event on any of the signals in the sensitivity list, the process fires.
- Every time the process fires, it will run in its entirety.
- WAIT statements are NOT ALLOWED in a processes with SENSITIVITY LIST.
Component Equivalent of a Process

- All signals which appear on the left of signal assignment statement (<=) are outputs e.g. y, z
- All signals which appear on the sensitivity list are inputs e.g. clk
- All signals which appear on the right of signal assignment statement (<=) or in logic expressions are inputs e.g. w, a, b, c
- Note that not all inputs need to be included on the sensitivity list

```
priority: PROCESS (clk)
BEGIN
  IF w(3) = '1' THEN
    y <= "11" ;
  ELSIF w(2) = '1' THEN
    y <= "10" ;
  ELSIF w(1) = c THEN
    y <= a and b;
  ELSE
    z <= "00" ;
  END IF ;
END PROCESS ;
```

D latch

Graphical symbol

<table>
<thead>
<tr>
<th>Clock</th>
<th>D</th>
<th>Q(t+1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Truth table

Timing diagram

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

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

ARCHITECTURE behavioral OF latch IS
BEGIN
  PROCESS ( D, Clock )
  BEGIN
    IF Clock = '1' THEN
      Q <= D ;
    END IF ;
    END PROCESS;
END behavioral;
```

D flip-flop

Graphical symbol

<table>
<thead>
<tr>
<th>Clock</th>
<th>D</th>
<th>Q(t+1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Truth table

Timing diagram

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

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

ARCHITECTURE behavioral2 OF flipflop IS
BEGIN
  PROCESS ( Clock )
  BEGIN
    IF rising_edge(Clock) THEN
      Q <= D ;
    END IF ;
    END PROCESS;
END behavioral2;
```
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 behavioral OF flipflop IS
BEGIN
PROCESS ( Clock )
BEGIN
IF Clock'EVENT AND Clock = '1' THEN
Q <= D ;
END IF ;
END PROCESS ;
END behavioral ;
END flipflop ;

D flip-flop with asynchronous reset

LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
ENTITY flipflop_ar IS
PORT ( D, Reset, Clock : IN STD_LOGIC ;
Q : OUT STD_LOGIC ) ;
END flipflop_ar ;
ARCHITECTURE behavioral OF flipflop_ar IS
BEGIN
PROCESS ( Reset, Clock )
BEGIN
IF Reset = '1' THEN
Q <= '0' ;
ELSIF rising_edge(Clock) THEN
Q <= D ;
END IF ;
END PROCESS ;
END behavioral ;
END flipflop_ar ;

Asynchronous vs. Synchronous

• In the IF loop, asynchronous items are
  • Before the rising_edge(Clock) statement
• In the IF loop, synchronous items are
  • After the rising_edge(Clock) statement

8-bit register with asynchronous reset

LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
ENTITY reg8 IS
PORT ( D : IN STD_LOGIC VECTOR(7 DOWNTO 0) ;
Reset, Clock : IN STD_LOGIC ;
Q : OUT STD_LOGIC VECTOR(7 DOWNTO 0) ) ;
END reg8 ;
ARCHITECTURE behavioral OF reg8 IS
BEGIN
PROCESS ( Reset, Clock )
BEGIN
IF Reset = '1' THEN
Q <= "00000000" ;
ELSIF rising_edge(Clock) THEN
Q <= D ;
END IF ;
END PROCESS ;
END behavioral ;
END reg8 ;

N-bit register with asynchronous reset

LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
ENTITY regn IS
GENERIC ( N : INTEGER := 16 ) ;
PORT ( D : IN STD_LOGIC VECTOR( N-1 DOWNTO 0) ;
Reset, Clock : IN STD_LOGIC ;
Q : OUT STD_LOGIC VECTOR( N-1 DOWNTO 0) ) ;
END regn ;
ARCHITECTURE behavioral OF regn IS
BEGIN
PROCESS ( Reset, Clock )
BEGIN
IF Reset = '1' THEN
Q <= (OTHERS => '0') ;
ELSIF rising_edge(Clock) THEN
Q <= D ;
END IF ;
END PROCESS ;
END behavioral ;
END regn ;
A word on generics

- Generics are typically integer values
- In this class, the entity inputs and outputs should be std_logic or std_logic_vector
- But the generics can be integer
- Generics are given a default value
  - GENERIC ( N : INTEGER := 16 );
  - This value can be overwritten when entity is instantiated as a component
- Generics are very useful when instantiating an often-used component
  - Need a 64-bit register in one place, and 16-bit register in another
  - Can use the same generic code, just configure them differently

Use of OTHERS

OTHERS stand for any index value that has not been previously mentioned.

Q <= "00000001" can be written as Q <= (0 => '1', OTHERS => '0')
Q <= "10000001" can be written as Q <= (7 => '1', 0 => '0', OTHERS => '0')
or Q <= (7 => '1', 0 => '0', OTHERS => '0')
Q <= "00011110" can be written as Q <= (4 downto 1 => '1', OTHERS => '0')

Component Instantiation in VHDL-93

U1: ENTITY work.regn(behavioral)
  GENERIC MAP (N => 4)
  PORT MAP (D => z, Resetn => reset, Clock => clk, Q => t);

Component Instantiation in VHDL-87

U1: regn GENERIC MAP (N => 4)
  PORT MAP (D => z, Resetn => reset, Clock => clk, Q => t);

N-bit register with enable

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY regne IS
  GENERIC ( N : INTEGER := 8 ) ;
  PORT ( D : IN STD_LOGIC_VECTOR(N-1 DOWNTO 0) ;
    Enable, Clock : IN STD_LOGIC;
    Q : OUT STD_LOGIC_VECTOR(N-1 DOWNTO 0) ) ;
END regne ;

ARCHITECTURE behavioral OF regne IS
BEGIN
  PROCESS (Clock)
  BEGIN
    IF rising_edge(Clock) THEN
      IF Enable = '1' THEN
        Q <= D ;
      END IF ;
      END IF ;
      END PROCESS ;
      END behavioral ;

Implementing two registers in a single process
Implementing two registers in a single process

PROCESS (Clk, Reset)
BEGIN
IF Reset = '1' THEN
Cout <= '0';
V <= '0';
ELSIFE rising_edge(Clk) THEN
IF EnC = '1' THEN
Cout <= Cout_tmp;
V <= V_tmp;
END IF;
END IF;
END IF;
END PROCESS;

Implementing two registers in a single process

PROCESS (Clk, Reset)
BEGIN
IF Reset = '1' THEN
Cout <= '0';
V <= '0';
ELSIFE rising_edge(Clk) THEN
IF EnC = '1' THEN
Cout <= Cout_tmp;
END IF;
END IF;
END PROCESS;

2-bit up-counter with synchronous reset

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;
ENTITY upcount IS
PORT ( Clear, Clock : IN STD_LOGIC;
Q : OUT STD_LOGIC_VECTOR(1 DOWNTO 0)) ;
END upcount;
ARCHITECTURE behavioral OF upcount IS
BEGIN
upcount: PROCESS ( Clock )
BEGIN
IF rising_edge(Clock) THEN
IF Clear = '1' THEN
Count <= "00" ;
ELSE
Count <= Count + 1;
END IF;
END IF;
END PROCESS;
Q <= std_logic_vector(Count);
END behavioral;

4-bit up-counter with asynchronous reset (1)

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;
ENTITY upcount_ar IS
PORT ( Clock, Resetn, Enable : IN STD_LOGIC;
Q : OUT STD_LOGIC_VECTOR (3 DOWNTO 0)) ;
END upcount_ar;
ARCHITECTURE behavioral OF upcount_ar IS
BEGIN
upcount_ar: PROCESS ( Clock )
BEGIN
IF rising_edge(Clock) THEN
IF Resetn = '1' THEN
Count <= "0000" ;
ELSE
Count <= Count + 1;
END IF;
END IF;
END PROCESS;
Q <= std_logic_vector(Count);
END behavioral;
4-bit up-counter with asynchronous reset (2)

ARCHITECTURE behavioral OF upcount ar IS
SIGNAL Count : UNSIGNED (3 DOWNTO 0) ;
BEGIN
PROCESS ( Clock, Resetn )
BEGIN
IF Resetn = '0' THEN
Count <= "0000" ;
ELSIF rising_edge(Clock) THEN
IF Enable = '1' THEN
Count <= Count + 1 ;
END IF ;
END IF ;
END PROCESS ;
Q <= std_logic_vector(Count) ;
END behavioral ;

4-bit shift register with parallel load (1)

LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
ENTITY shift4 IS
PORT ( D : IN STD_LOGIC_VECTOR(3 DOWNTO 0) ;
Enable : IN STD_LOGIC ;
Load : IN STD_LOGIC ;
Sin : IN STD_LOGIC ;
Clock : IN STD_LOGIC ;
Q : OUT STD_LOGIC_VECTOR(3 DOWNTO 0) ) ;
END shift4 ;

4-bit shift register with parallel load (2)

ARCHITECTURE behavioral OF shift4 IS
SIGNAL Q : STD_LOGIC_VECTOR(3 DOWNTO 0) ;
BEGIN
PROCESS (Clock)
BEGIN
IF rising_edge(Clock) THEN
IF Enable = '1' THEN
IF Load = '1' THEN
Q <= D ;
ELSE
Q <= Q & Q(3 downto 1);
END IF;
END IF;
END PROCESS ;
END behavioral ;
N-bit shift register with parallel load (1)

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY shiftn IS
    GENERIC ( N : INTEGER := 8 ) ;
    PORT ( D : IN STD_LOGIC_VECTOR( N-1 DOWNTO 0) ;
            Enable, Load, Sin, Clock : IN STD_LOGIC ;
            Q : OUT STD_LOGIC_VECTOR(N-1 DOWNTO 0) ) ;
END shiftn ;

ARCHITECTURE behavioral OF shiftn IS
    SIGNAL Qt: STD_LOGIC_VECTOR(N-1 DOWNTO 0);
    BEGIN
        PROCESS (Clock)
        BEGIN
            IF rising_edge(Clock) THEN
                IF Enable = '1' THEN
                    Qt <= D ;
                ELSE
                    Qt <= Sin & Qt(N-1 downto 1);
                END IF;
            END IF;
        END PROCESS ;
        Q <= Qt;
    END behavioral;

N-bit shift register with parallel load (2)

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY regn IS
    GENERIC ( N : INTEGER := 8 ) ;
    PORT ( D : IN STD_LOGIC_VECTOR(N-1 DOWNTO 0) ;
            Enable, Clock : IN STD_LOGIC ;
            Q : OUT STD_LOGIC_VECTOR(N-1 DOWNTO 0) ) ;
END regn ;

ARCHITECTURE Behavior OF regn IS
    BEGIN
        PROCESS (Clock)
        BEGIN
            IF ( rising_edge(Clock) ) THEN
                IF Enable = '1' THEN
                    Q <= D ;
                END IF;
            END IF;
        END PROCESS ;
    END Behavior ;

Generic Component Instantiation

Circuit built of medium scale components

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY priority_resolver IS
    PORT ( r : IN STD_LOGIC_VECTOR(5 DOWNTO 0) ;
            s : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
            clk : IN STD_LOGIC;
            en : IN STD_LOGIC;
            t : OUT STD_LOGIC_VECTOR(3 DOWNTO 0) ) ;
END priority_resolver ;

ARCHITECTURE structural OF priority Resolver IS
    SIGNAL p : STD_LOGIC_VECTOR(3 DOWNTO 0) ;
    SIGNAL q : STD_LOGIC_VECTOR(1 DOWNTO 0) ;
    SIGNAL z : STD_LOGIC_VECTOR(3 DOWNTO 0) ;
    SIGNAL ena : STD_LOGIC ;
    BEGIN
        ENd priority_resolver ;
Structural description – example (2)

VHDL-93

BEGIN
u1: ENTITY work.mux2to1(dataflow)
  PORT MAP (w0 => r(0),
  w1 => r(1),
  s => s(0),
  f => p(0));
  p(1) <= r(2);
p(2) <= r(3);
u2: ENTITY work.mux2to1(dataflow)
  PORT MAP (w0 => r(4),
  w1 => r(5),
  s => s(1),
  f => p(3));
u3: ENTITY work.priority(dataflow)
  PORT MAP (w => p,
  y => q,
  z => ena);
END structural;

LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY
priority_resolver
IS
  PORT (r : IN STD_LOGIC_VECTOR(5 DOWNTO 0);
  s : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
  clk : IN STD_LOGIC;
  t : OUT STD_LOGIC_VECTOR(3 DOWNTO 0));
END
priority_resolver;
ARCHITECTURE structural OF priority_resolver IS
  SIGNAL  p : STD_LOGIC_VECTOR (3 DOWNTO 0) ;
  SIGNAL  q : STD_LOGIC_VECTOR (1 DOWNTO 0) ;
  SIGNAL  z : STD_LOGIC_VECTOR (3 DOWNTO 0) ;
  SIGNAL  ena : STD_LOGIC ;
END structural;
COMPONENT mux2to1
  PORT (w0, w1, s : IN STD_LOGIC;
  f : OUT STD_LOGIC ) ;
END COMPONENT ;
COMPONENT priority
  PORT (w : IN STD_LOGIC_VECTOR(3 DOWNTO 0) ;
  y : OUT STD_LOGIC_VECTOR(1 DOWNTO 0) ;
  z : OUT STD_LOGIC ) ;
END COMPONENT ;
COMPONENT dec2to4
  PORT (w : IN STD_LOGIC_VECTOR(1 DOWNTO 0) ;
  En : IN STD_LOGIC ;
  y : OUT STD_LOGIC_VECTOR(3 DOWNTO 0) ) ;
END COMPONENT ;

COMPONENT regn
  GENERIC ( N : INTEGER := 8 ) ;
  PORT ( D : IN STD_LOGIC_VECTOR(N-1 DOWNTO 0) ;
  Enable, Clock : IN STD_LOGIC ;
  Q : OUT STD_LOGIC_VECTOR(N-1 DOWNTO 0) ) ;
END COMPONENT ;
**Structural description – example (5) VHDL-87**

```vhdl
uf: regne
    GENERIC MAP (N => 4)
    PORT MAP (D => z, Enable => En, Clock => Clk, Q => t);
END structural;
```

**Constants**

**Syntax:**

```
CONSTANT   name : type := value;
```

**Examples:**

```
CONSTANT init_value : STD_LOGIC_VECTOR(3 downto 0) := "0100";
CONSTANT ANDA_EXT : STD_LOGIC_VECTOR(7 downto 0) := "X'B4";
CONSTANT counter_width : INTEGER := 16;
CONSTANT buffer_address : INTEGER := 16#FFFE#;
CONSTANT clk_period : TIME := 20 ns;
CONSTANT strobe_period : TIME := 333.333 ms;
```

**Example of package**

```vhdl
library ieee;
use ieee.std_logic_1164.all;
package alu_pkg is
    constant OPCODE_NOR : std_logic_vector(2 downto 0) := "000";
    constant OPCODE_NAND : std_logic_vector(2 downto 0) := "001";
    constant OPCODE_XOR : std_logic_vector(2 downto 0) := "010";
    constant OPCODE_UADD : std_logic_vector(2 downto 0) := "011";
    constant OPCODE_SADD : std_logic_vector(2 downto 0) := "100";
    constant OPCODE_SSUB : std_logic_vector(2 downto 0) := "101";
    constant OPCODE_UMUL : std_logic_vector(2 downto 0) := "110";
    constant OPCODE_SMUL : std_logic_vector(2 downto 0) := "111";
end alu_pkg;
```

**Using objects from a package**

```vhdl
library ieee;
use ieee.std_logic_1164.all;
library work;
use work.alu_pkg.all;
entity alu_comb is
```

---

**Constants - features**

Constants can be declared in a PACKAGE, ARCHITECTURE, ENTITY

**When declared in a PACKAGE, the constant is truly global**, for the package can be used in several entities.

When declared in an ARCHITECTURE, the constant is local, i.e., it is visible only within this architecture.

When declared in an ENTITY declaration, the constant can be used in all architectures associated with this entity.
Mixing Description Styles Inside of an Architecture

### VHDL Description Styles

![VHDL Description Styles Diagram]

- **dataflow**
- **structural**
- **behavioral**

- **Sequential statements**
  - Registers
  - Shift registers
  - Counters
  - State machines

### Mixed Style Modeling

```
architecture ARCHITECTURE_NAME of ENTITY_NAME is
    
    • Here you can declare signals, constants, functions, procedures...
    • Component declarations

begin
    Concurrent statements:
    • Concurrent simple signal assignment
    • Conditional signal assignment
    • Selected signal assignment
    • Generate statement
    • Component instantiation statement

    • Process statement
    • inside process you can use only sequential statements

end ARCHITECTURE_NAME;
```

### PRNG Example (1)

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use work.prng_pkg.all;

ENTITY PRNG IS
    PORT(
        Coeff : in  std_logic_vector(4 downto 0);
        Load_Coeff : in  std_logic;
        Seed : in  std_logic_vector(4 downto 0);
        Init_Run : in  std_logic;
        Clk : in  std_logic;
        Current_State : out std_logic_vector(4 downto 0));
END PRNG;

ARCHITECTURE mixed OF PRNG is

    signal Ands : std_logic_vector(4 downto 0);
    signal Sin : std_logic;
    signal Coeff_Q : std_logic_vector(4 downto 0);
    signal Shift5_Q : std_logic_vector(4 downto 0);

BEGIN
    -- Data Flow
    Sin <= Ands(0) XOR Ands(1) XOR Ands(2) XOR Ands(3) XOR Ands(4);
    Current_State <= Shift5_Q;
    Ands <= Coeff_Q AND Shift5_Q;

    -- Behavioral
    Coeff_Reg: PROCESS(Clk)
    BEGIN
        IF rising_edge(Clk) THEN
            IF Load_Coeff = '1' THEN
                Coeff_Q <= Coeff;
            END IF;
        END IF;
    END PROCESS;

    -- Structural
    Shift5_Reg : ENTITY work.Shift5(behavioral) PORT MAP ( D => Seed,
            Load => Init_Run,
            Sin => Sin,
            Clock => Clk,
            Q => Shift5_Q);
END mixed;
```

### PRNG Example (2)

```vhdl
BEGIN
    -- Data Flow
    Sin <= Ands(0) XOR Ands(1) XOR Ands(2) XOR Ands(3) XOR Ands(4);
    Current_State <= Shift5_Q;
    Ands <= Coeff_Q AND Shift5_Q;

    -- Behavioral
    Coeff_Reg: PROCESS(Clk)
    BEGIN
        IF rising_edge(Clk) THEN
            IF Load_Coeff = '1' THEN
                Coeff_Q <= Coeff;
            END IF;
        END IF;
    END PROCESS;

    -- Structural
    Shift5_Reg : ENTITY work.Shift5(behavioral) PORT MAP ( D => Seed,
            Load => Init_Run,
            Sin => Sin,
            Clock => Clk,
            Q => Shift5_Q);
END mixed;
```
**For Beginners**

Use processes with very simple structure only to describe
- registers
- shift registers
- counters
- state machines.

Use examples discussed in class as a template.

Create **generic** entities for registers, shift registers, and counters, and instantiate the corresponding components in a higher level circuit using GENERIC MAP PORT MAP.

Supplement sequential components with combinational logic described using concurrent statements.

**For Intermediates**

1. Use Processes with IF and CASE statements only. Do not use LOOPS or VARIABLES.
2. Sensitivity list of the PROCESS should include **only** signals that can by themselves change the outputs of the sequential circuit (typically, clock and asynchronous set or reset)
3. Do not use PROCESSes without sensitivity list (they can be synthesizable, but make simulation inefficient)

**For Intermediates (2)**

Given a single signal, the assignments to this signal should only be made within a single process block in order to avoid possible conflicts in assigning values to this signal.

```vhdl
Process 1: PROCESS (a, b)
BEGIN
  y <= a AND b;
END PROCESS;

Process 2: PROCESS (a, b)
BEGIN
  y <= a OR b;
END PROCESS;
```