ECE 545
Lecture 7
Behavioral Modeling of Sequential-Circuit Building Blocks
Mixing Design Styles
Modeling of Circuits with a Regular Structure
Required reading

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

*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
Required reading

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

  Chapter 14.5 For generate statement

  Chapter 14.6 Conditional generate statement

  Chapter 9.1, *Poor design practices and their remedies*
Behavioral Design Style: Registers & Counters
VHDL Description Styles

VHDL Description Styles

Dataflow
- Concurrent statements

Structural
- Components and interconnects

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

Synthesizable

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

OPTIONAL

[label:] PROCESS [(sensitivity list)]
[declaration part]
BEGIN
  statement part
END PROCESS [label];
PROCESS with a SENSITIVITY LIST

- 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.

```verbatim
label: process (sensitivity list)
  declaration part
  begin
    statement part
  end process;
```
Component Equivalent of a Process

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 ;

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

Truth table

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

Timing diagram
D flip-flop

Graphical symbol

Truth table

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

Timing diagram

Clock

D

Q
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;
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;
D flip-flop with asynchronous reset

LIBRARY ieee;
USE ieee.std_logic_1164.all;

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

ARCHITECTURE behavioral OF flipflop_ar IS
BEGIN
  PROCESS ( Resetn, Clock )
  BEGIN
    IF Resetn = '0' THEN
      Q <= '0' ;
    ELSIF rising_edge(Clock) THEN
      Q <= D ;
    END IF;
  END PROCESS;
END behavioral;
D flip-flop with **synchronous reset**

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY flipflop_sr IS
  PORT ( D, Resetn, Clock : IN STD_LOGIC;
         Q       : OUT STD_LOGIC);
END flipflop_sr;

ARCHITECTURE behavioral OF flipflop_sr IS
BEGIN
  PROCESS(Clock)
  BEGIN
    IF rising_edge(Clock) THEN
      IF Resetn = '0' THEN
        Q <= '0';
      ELSE
        Q <= D;
      END IF;
    END IF;
  END PROCESS;
END behavioral;
```
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);
         Resetn, Clock : IN STD_LOGIC;
         Q : OUT STD_LOGIC_VECTOR(7 DOWNTO 0) );
END reg8 ;

ARCHITECTURE behavioral OF reg8 IS
BEGIN
  PROCESS ( Resetn, Clock )
  BEGIN
    IF Resetn = '0' THEN
      Q <= "00000000" ;
    ELSIF rising_edge(Clock) THEN
      Q <= D ;
    END IF ;
  END PROCESS ;
END behavioral ;
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);
          Resetn, Clock : IN STD_LOGIC;
          Q : OUT STD_LOGIC_VECTOR(N-1 DOWNTO 0) ) ;
END regn ;

ARCHITECTURE behavioral OF regn IS
BEGIN
    PROCESS ( Resetn, Clock )
    BEGIN
        IF Resetn = '0' THEN
            Q <= (OTHERS => '0');
        ELSIF rising_edge(Clock) THEN
            Q <= D ;
        END IF ;
    END PROCESS ;
END behavioral ;
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 32-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 => ‘1’, OTHERS => ‘0’)
    or       Q <= (7 | 0 => ‘1’, 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);
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 ;
Counters
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.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
    SIGNAL Count : std_logic_vector(1 DOWNTO 0);
BEGIN
    upcount: PROCESS ( Clock )
    BEGIN
        IF rising_edge(Clock) THEN
            IF Clear = '1' THEN
                Count <= "00" ;
            ELSE
                Count <= Count + 1 ;
            END IF;
        END IF;
        Q <= Count;
    END PROCESS;
END behavioral;
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;

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

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

4-bit up-counter with asynchronous reset (1)
4-bit up-counter with asynchronous reset (2)

ARCHITECTURE behavioral OF upcount_ar IS
    SIGNAL Count : STD_LOGIC_VECTOR (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 <= Count ;
END behavioral ;
Shift Registers
Shift register – internal structure
Shift Register With Parallel Load

Load

D(3)

Sin

Clock

Enable

Q(3) Q(2) Q(1) Q(0)
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 (1)

<table>
<thead>
<tr>
<th>Enable</th>
<th>Clock</th>
<th>Load</th>
<th>Sin</th>
<th>Q</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4</td>
</tr>
</tbody>
</table>

4

shift4
ARCHITECTURE behavioral OF shift4 IS
  SIGNAL Qt : STD_LOGIC_VECTOR(3 DOWNTO 0);
BEGIN
  PROCESS (Clock)
  BEGIN
    IF rising_edge(Clock) THEN
      IF Enable = '1' THEN
        IF Load = '1' THEN
          Qt <= D ;
        ELSE
          Qt <= Sin & Qt(3 downto 1);
        END IF ;
      END IF;
    END IF ;
    END IF;
    END PROCESS;
Q <= Qt;
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 : IN   STD_LOGIC ;
    Load   : IN   STD_LOGIC ;
    Sin    : IN   STD_LOGIC ;
    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
    IF Load = '1' THEN
      Qt <= D;
    ELSE
      Qt <= Sin & Qt(N-1 downto 1);
    END IF;
  END IF;
END IF;
END PROCESS;

Q <= Qt;

END behavioral;

N-bit shift register with parallel load (2)
Generic Component Instantiation
N-bit register with enable

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;
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

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);
u4: ENTITY work.dec2to4 (dataflow)
    PORT MAP (w => q,
           En => ena,
           y => z);

u5: ENTITY work.regne(behavioral)
    GENERIC MAP (N => 4)
    PORT MAP (D => z ,
              Enable => En ,
              Clock => Clk,
              Q => t );

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;
       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 ;
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 ;
BEGIN

u1: mux2to1 PORT MAP (w0 => r(0),
                     w1 => r(1),
                     s => s(0),
                     f => p(0));

p(1) <= r(2);
p(2) <= r(3);

u2: mux2to1 PORT MAP (w0 => r(4),
                     w1 => r(5),
                     s => s(1),
                     f => p(3));

u3: priority PORT MAP (w => p,
                       y => q,
                       z => ena);

u4: dec2to4 PORT MAP (w => q,
                      En => ena,
                      y => z);
Structural description – example (5)
VHDL-87

u5: regne
   GENERIC MAP (N => 4)
   PORT MAP (D => z ,
              Enable => En ,
              Clock => Clk ,
              Q => t );

END structural;
Mixing Description Styles
Inside of an Architecture
VHDL Description Styles

VHDL Description Styles

- **dataflow**
  - Concurrent statements

- **structural**
  - Components and interconnects

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

*synthesizable*
Mixed Style Modeling

```vhdl
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;
```
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);
END ARCHITECTURE;
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;
Sequential Logic Synthesis for Beginners
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.
Sequential Logic Synthesis 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.

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

Process 2: PROCESS (a, b)
BEGIN
    y <= a OR b;
END PROCESS;
Generate scheme for equations
PARITY Example
PARITY: Block Diagram
PARITY: Entity Declaration

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY parity IS
  PORT(
    parity_in : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
    parity_out : OUT STD_LOGIC
  );
END parity;
PARITY: Block Diagram
PARITY: Architecture

ARCHITECTURE parity_dataflow OF parity IS

SIGNAL xor_out: std_logic_vector (6 downto 1);

BEGIN

xor_out(1) <= parity_in(0) XOR parity_in(1);
xor_out(2) <= xor_out(1) XOR parity_in(2);
xor_out(3) <= xor_out(2) XOR parity_in(3);
xor_out(4) <= xor_out(3) XOR parity_in(4);
xor_out(5) <= xor_out(4) XOR parity_in(5);
xor_out(6) <= xor_out(5) XOR parity_in(6);
parity_out <= xor_out(6) XOR parity_in(7);

END parity_dataflow;
ARCHITECTURE parity_dataflow OF parity IS

SIGNAL xor_out: STD_LOGIC_VECTOR (6 DOWNTO 1);

BEGIN
  G2: FOR i IN 1 TO 7 GENERATE
      left_xor: IF i=1 GENERATE
          xor_out(i) <= parity_in(i-1) XOR parity_in(i);
      END GENERATE;
      middle_xor: IF (i >1) AND (i<7) GENERATE
          xor_out(i) <= xor_out(i-1) XOR parity_in(i);
      END GENERATE;
      right_xor: IF i=7 GENERATE
          parity_out <= xor_out(i-1) XOR parity_in(i);
      END GENERATE;
  END GENERATE;
END GENERATE;
END parity_dataflow;
PARITY: Block Diagram (2)
PARITY: Architecture

ARCHITECTURE parity_dataflow OF parity IS

SIGNAL xor_out: STD_LOGIC_VECTOR (7 downto 0);

BEGIN

    xor_out(0) <= parity_in(0);
    xor_out(1) <= xor_out(0) XOR parity_in(1);
    xor_out(2) <= xor_out(1) XOR parity_in(2);
    xor_out(3) <= xor_out(2) XOR parity_in(3);
    xor_out(4) <= xor_out(3) XOR parity_in(4);
    xor_out(5) <= xor_out(4) XOR parity_in(5);
    xor_out(6) <= xor_out(5) XOR parity_in(6);
    xor_out(7) <= xor_out(6) XOR parity_in(7);
    parity_out <= xor_out(7);

END parity_dataflow;
ARCHITECTURE parity_dataflow OF parity IS

SIGNAL xor_out: STD_LOGIC_VECTOR (7 DOWNTO 0);

BEGIN

    xor_out(0) <= parity_in(0);

    G2: FOR i IN 1 TO 7 GENERATE
        xor_out(i) <= xor_out(i-1) XOR parity_in(i);
    END GENERATE G2;

    parity_out <= xor_out(7);

END parity_dataflow;
For Generate Statement

For - Generate

label:
FOR identifier IN range GENERATE
{Concurrent Statements}
END GENERATE;
Conditional Generate Statement

If - Generate

label:
 IF boolean_expression GENERATE
   {Concurrent Statements}
END GENERATE;
Generate scheme for components
Example 1
Example 1
A 4-to-1 Multiplexer

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;
Straightforward code for Example 1

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY Example1 IS
  PORT ( w : IN STD_LOGIC_VECTOR(0 TO 15) ;
         s : IN STD_LOGIC_VECTOR(3 DOWNTO 0) ;
         f : OUT STD_LOGIC ) ;
END Example1 ;
ARCHITECTURE Structure OF Example1 IS

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

SIGNAL m : STD_LOGIC_VECTOR(0 TO 3) ;

BEGIN
  Mux1: mux4to1 PORT MAP ( w(0),    w(1),    w(2),    w(3),       s(1 DOWNTO 0), m(0) ) ;
  Mux2: mux4to1 PORT MAP ( w(4),    w(5),    w(6),    w(7),       s(1 DOWNTO 0), m(1) ) ;
  Mux3: mux4to1 PORT MAP ( w(8),    w(9),    w(10),  w(11),     s(1 DOWNTO 0), m(2) ) ;
  Mux4: mux4to1 PORT MAP ( w(12),  w(13),  w(14),  w(15),     s(1 DOWNTO 0), m(3) ) ;
  Mux5: mux4to1 PORT MAP ( m(0),    m(1),    m(2),    m(3),       s(3 DOWNTO 2),   f  ) ;
END Structure ;
ARCHITECTURE Structure OF Example1 IS

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

SIGNAL m : STD_LOGIC_VECTOR(0 TO 3) ;

BEGIN
    G1: FOR i IN 0 TO 3 GENERATE
        Muxes: mux4to1 PORT MAP ( .............................................. );
    END GENERATE ;
    Mux5: mux4to1 PORT MAP ( m(0), m(1), m(2), m(3), s(3 DOWNTO 2), f ) ;
END Structure ;
ARCHITECTURE Structure OF Example1 IS

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

  SIGNAL m : STD_LOGIC_VECTOR(0 TO 3) ;

BEGIN

  G1: FOR i IN 0 TO 3 GENERATE
      Muxes: mux4to1 PORT MAP ( w(4*i), w(4*i+1), w(4*i+2), w(4*i+3), s(1 DOWNTO 0), m(i) ) ;
  END GENERATE ;
  Mux5: mux4to1 PORT MAP ( m(0), m(1), m(2), m(3), s(3 DOWNTO 2), f ) ;

END Structure ;
Example 2
Example 2
A 2-to-4 binary decoder

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 ;
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY dec4to16 IS
    PORT (w : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
         En : IN STD_LOGIC;
         y  : OUT STD_LOGIC_VECTOR(15 DOWNTO 0));
END dec4to16;
ARCHITECTURE Structure OF dec4to16 IS

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 ;

SIGNAL m : STD_LOGIC_VECTOR(3 DOWNTO 0) ;

BEGIN

    Dec_r0: dec2to4 PORT MAP ( w(1 DOWNTO 0), m(0), y(3 DOWNTO 0) );
    Dec_r1: dec2to4 PORT MAP ( w(1 DOWNTO 0), m(1), y(7 DOWNTO 4) );
    Dec_r2: dec2to4 PORT MAP ( w(1 DOWNTO 0), m(2), y(11 DOWNTO 8) );
    Dec_r3: dec2to4 PORT MAP ( w(1 DOWNTO 0), m(3), y(15 DOWNTO 12) );
    Dec_left: dec2to4 PORT MAP ( w(3 DOWNTO 2), En, m ) ;
END Structure ;
ARCHITECTURE Structure OF dec4to16 IS

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 ;

SIGNAL m : STD_LOGIC_VECTOR(3 DOWNTO 0) ;

BEGIN
    G1: FOR i IN 0 TO 3 GENERATE
        Dec_ri: dec2to4 PORT MAP ( .................., .................., ..................);
    END GENERATE ;

    Dec_left: dec2to4 PORT MAP ( w(3 DOWNTO 2), En, m ) ;
END Structure ;
ARCHITECTURE Structure OF dec4to16 IS

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 ;

SIGNAL m : STD_LOGIC_VECTOR(3 DOWNTO 0) ;

BEGIN
  G1: FOR i IN 0 TO 3 GENERATE
      Dec_ri: dec2to4 PORT MAP ( w(1 DOWNTO 0), m(i), y(4*i+3 DOWNTO 4*i) ) ;
  END GENERATE ;

  Dec_left: dec2to4 PORT MAP ( w(3 DOWNTO 2), En, m ) ;
END Structure ;
Example 3
Up-or-down Free Running Counter
Up-or-down Free Running Counter
Up-or-down Free Running Counter (1)

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity up_or_down_counter is
  generic(
    WIDTH: natural:=4;
    UP: natural:=0
  );
  port(
    clk, reset: in std_logic;
    q: out std_logic_vector(WIDTH-1 downto 0)
  );
end up_or_down_counter;
Up-or-down Free Running Counter (2)

architecture mixed of up_or_down_counter is

    signal r_reg: unsigned(WIDTH-1 downto 0);
    signal r_next: unsigned(WIDTH-1 downto 0);

begin
    -- register
    process(clk,reset)
    begin
        if (reset='1') then
            r_reg <= (others=>'0');
        elsif (clk'event and clk='1') then
            r_reg <= r_next;
        end if;
    end if;
end process;
Up-or-down Free Running Counter (3)

-- next-state logic
inc_gen: -- incrementor
if UP=1 generate
   r_next <= r_reg + 1;
end generate;

dec_gen: -- decrementor
if UP/=1 generate
   r_next <= r_reg – 1;
end generate;

-- output logic
q <= std_logic_vector(r_reg);

end mixed;
Example 4
Up-and-down Free Running Counter
Up-and-down Free Running Counter
Up-and-down Free Running Counter (1)

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity up_and_down_counter is
    generic(WIDTH: natural:=4);
    port(
      clk, reset: in std_logic;
      mode: in std_logic;
      q: out std_logic_vector(WIDTH-1 downto 0)
    );
end up_and_down_counter;
Up-and-down Free Running Counter (2)

architecture arch of up_and_down_counter is

    signal r_reg: unsigned(WIDTH-1 downto 0);
    signal r_next: unsigned(WIDTH-1 downto 0);

begin
  -- register
  process(clk,reset)
  begin
    if (reset='1') then
      r_reg <= (others=>'0');
    elsif (clk'event and clk='1') then
      r_reg <= r_next;
    end if;
  end process;

end process;
Up-and-down Free Running Counter (3)

-- next-state logic

\[
\begin{align*}
\text{r\textunderscore next} &\leq \text{r\textunderscore reg} + 1 \text{ when mode='1' else} \\
&\text{r\textunderscore reg} - 1;
\end{align*}
\]

-- output logic

\[ q \leq \text{std\textunderscore logic\textunderscore vector(r\textunderscore reg)}; \]

end arch;
Example 5
Variable Rotator
Example 3: Variable rotator - Interface

A <<< B
Block diagram
VHDL code for a 16-bit 2-to-1 Multiplexer

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY mux2to1_16 IS
    PORT (w0 : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
          w1 : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
          s  : IN STD_LOGIC;
          f  : OUT STD_LOGIC_VECTOR(15 DOWNTO 0) ) ;
END mux2to1_16 ;

ARCHITECTURE dataflow OF mux2to1_16 IS
BEGIN
    f <= w0 WHEN s = '0' ELSE w1 ;
END dataflow ;
Fixed rotation

\[
a(15) \ a(14) \ a(13) \ a(12) \ a(11) \ a(10) \ a(9) \ a(8) \ a(7) \ a(6) \ a(5) \ a(4) \ a(3) \ a(2) \ a(1) \ a(0)
\]

\[\lll 3\]

\[
a(12) \ a(11) \ a(10) \ a(9) \ a(8) \ a(7) \ a(6) \ a(5) \ a(4) \ a(3) \ a(2) \ a(1) \ a(0) \ a(15) \ a(14) \ a(13)
\]

\[
y \leq a(12 \ downto \ 0) \ & \ a(15 \ downto \ 13); 
\]

\[
a(15) \ a(14) \ a(13) \ a(12) \ a(11) \ a(10) \ a(9) \ a(8) \ a(7) \ a(6) \ a(5) \ a(4) \ a(3) \ a(2) \ a(1) \ a(0)
\]

\[\lll 5\]

\[
a(10) \ a(9) \ a(8) \ a(7) \ a(6) \ a(5) \ a(4) \ a(3) \ a(2) \ a(1) \ a(0) \ a(15) \ a(14) \ a(13) \ a(12) \ a(11)
\]

\[
y \leq a(10 \ downto \ 0) \ & \ a(15 \ downto \ 11); 
\]
Fixed rotation by $L$ positions

\[
\begin{align*}
\text{y} & \leq a(15-L \text{ downto } 0) \& a(15 \text{ downto } 15-L+1) \\
\end{align*}
\]
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY fixed_rotator_left_16 IS
  GENERIC ( L : INTEGER := 1);
  PORT ( a : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
         y : OUT STD_LOGIC_VECTOR(15 DOWNTO 0) );
END fixed_rotator_left_16;

ARCHITECTURE dataflow OF fixed_rotator_left_16 IS
BEGIN
  y <= a(15-L downto 0) & a(15 downto 15-L+1);
END dataflow;
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY variable_rotator_16 is
    PORT(
        A : IN STD_LOGIC_VECTOR(15 downto 0);
        B : IN STD_LOGIC_VECTOR(3 downto 0);
        C : OUT STD_LOGIC_VECTOR(15 downto 0)
    );
END variable_rotator_16;
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ARCHITECTURE structural OF variable_rotator_16 IS

COMPONENT mux2to1_16
  PORT ( w0 : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
        w1 : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
        s  : IN STD_LOGIC;
        f  : OUT STD_LOGIC_VECTOR(15 DOWNTO 0) ) ;
END COMPONENT ;

COMPONENT fixed_rotator_left_16
  GENERIC ( L : INTEGER := 1);
  PORT ( a : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
         y : OUT STD_LOGIC_VECTOR(15 DOWNTO 0) ) ;
END COMPONENT ;
TYPE array1 IS ARRAY (0 to 4) OF STD_LOGIC_VECTOR(15 DOWNTO 0);
TYPE array2 IS ARRAY (0 to 3) OF STD_LOGIC_VECTORS(15 DOWNTO 0);
SIGNAL Al : array1;
SIGNAL Ar : array2;

BEGIN
  Al(0) <= A;
  G: FOR i IN 0 TO 3 GENERATE
    ROT_I: fixed_rotator_left_16
     GENERIC MAP (L => 2** i)
     PORT MAP ( a => Al(i) ,
                y => Ar(i));
    MUX_I: mux2to1_16 PORT MAP (w0  => Al(i),
                                w1  => Ar(i),
                                s => B(i),
                                f => Al(i+1));
  END GENERATE;
  C <= Al(4);
END variable_rotator_16;
Block diagram
Non-synthesizable VHDL
Delays

Delays are not synthesizable

Statements, such as

\texttt{wait for 5 ns}

\texttt{a <= b after 10 ns}

will not produce the required delay, and should not be used in the code intended for synthesis.
Initializations

Declarations of signals (and variables) with initialized values, such as

```
SIGNAL a : STD_LOGIC := '0';
```

cannot be synthesized, and thus should be avoided.

If present, they will be ignored by the synthesis tools.

**Use set and reset signals instead.**
Dual-edge triggered register/counter (1)

In FPGAs register/counter can change only at either rising (default) or falling edge of the clock.

Dual-edge triggered clock is not synthesizable correctly, using either of the descriptions provided below.
Dual-edge triggered register/counter (2)

```vhdl
PROCESS (clk)
BEGIN
  IF (clk'EVENT AND clk='1' ) THEN
    counter <= counter + 1;
  ELSIF (clk'EVENT AND clk='0' ) THEN
    counter <= counter + 1;
  END IF;
END IF;
END PROCESS;
```
Dual-edge triggered register/counter (3)

PROCESS (clk)
BEGIN
  IF (clk’EVENT) THEN
    counter <= counter + 1;
  END IF;
END PROCESS;

PROCESS (clk)
BEGIN
  counter <= counter + 1;
END PROCESS;
Poor Design Practices
1. Poor design practice and remedy

- Synchronous design is the most important methodology
- Poor practice in the past (to save chips)
  - Misuse of asynchronous reset
  - Misuse of gated clock
  - Misuse of derived clock
Misuse of Asynchronous Reset
(a) Block diagram

(b) Timing diagram
• Problem
  – Glitches in transition 1001 (9) => 0000 (0)
  – Glitches in aync_clr can reset the counter
  – How about timing analysis? (maximal clock rate)
• Asynchronous reset should only be used for power-on initialization
Decade (mod-10) counter

```vhdl
architecture two_seg_arch of mod10_counter is
  constant TEN: integer := 10;
  signal r_reg: unsigned(3 downto 0);
  signal r_next: unsigned(3 downto 0);

begin
  -- register
  process (clk, reset)
  begin
    if (reset='1') then
      r_reg <= (others=>'0');
    elseif (clk'event and clk='1') then
      r_reg <= r_next;
    end if;
  end process;
  -- next-state logic
  r_next <= (others=>'0') when r_reg=(TEN-1) else
    r_reg + 1;
  -- output logic
  q <= std_logic_vector(r_reg);
end two_seg_arch;
```
Misuse of Gated Clock
Figure 9.2  Disabling FF with gated clock
• Problem
  – Gated clock width can be narrow
  – Gated clock may pass glitches of en
  – Difficult to design the clock distribution network
Dedicated Clock Tree Network – H Tree
Misuse of Derived Clock
Misuse of derived clock

- Subsystems may run at different clock rate
- Poor design: use a derived slow clock for slow subsystem
• Problem
  – Multiple clock distribution network
  – How about timing analysis? (maximal clock rate)
• Better use a synchronous one-clock enable pulse
• E.g., second and minutes counter
  – Input: 1 MHz clock
  – Poor design:

(a) Design with derived clock
– Better design

(b) Design with a single synchronous clock