ECE 448 Lecture 4
Sequential-Circuit Building Blocks
Constants & Packages
Mixing Description Styles
Reading

Required

• P. Chu, *FPGA Prototyping by VHDL Examples*
  *Chapter 4, Regular Sequential Circuit*

Recommended

• S. Brown and Z. Vranesic, *Fundamentals of Digital Logic with VHDL Design*
  *Chapter 7, Flip-Flops, Registers, Counters, and a Simple Processor*
Behavioral Design Style: Registers & Counters
VHDL Description Styles

- **dataflow**
  - Concurrent statements

- **structural**
  - Components and interconnects

- **behavioral**
  - Sequential statements
    - Registers
    - Shift registers
    - Counters
    - State machines
  - and more if you are careful

**synthesizable**
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

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

OPTIONAL
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.**
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 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
Registers
D latch

Graphical symbol

Truth table

<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>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**

```
   D  Q
--->  -->
   Clock
```

**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>0</td>
<td>Q(t)</td>
</tr>
<tr>
<td>1</td>
<td>Q(t)</td>
<td>Q(t)</td>
</tr>
</tbody>
</table>

**Timing diagram**

- **Clock**
- **D**
- **Q**

---

**ECE 448 – FPGA and ASIC Design with VHDL**

11
D latch

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

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 ;
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

D → Q
Clock →

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;
D flip-flop with synchronous reset

LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY flipflop_sr IS
  PORT ( D, Reset, 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 Reset = '1' 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);
           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 ;
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;

Diagram of the N-bit register with asynchronous reset.
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’)
U1: ENTITY work.regn(behavioral)
    GENERIC MAP (N => 4)
    PORT MAP (D => z ,
             Reset => reset ,
             Clock => clk ,
             Q => t );
Component Instantiation in VHDL-87

U1: regn GENERIC MAP (N => 4)
PORT MAP (D => z ,
    Reset => 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

```vhdl
PROCESS (Clk, Reset)
BEGIN
    IF Reset = '1' THEN
        Cout <= '0';
        V <= '0';
    ELSIF rising_edge(Clk) THEN
        IF Enf = '1' THEN
            Cout <= Cout_tmp;
            V <= V_tmp;
        END IF;
    END IF;
END PROCESS;
```
Implementing two registers in a single process

![Diagram showing two registers with labels: Cout_tmp, En, D, Q, Cout, Clk, Reset, V_tmp, En, D, Q, V, Clk, Reset.]}
Implementing two registers in a single process

PROCESS (Clk, Reset)
BEGIN
    IF Reset = '1' THEN
        Cout <= '0';
        V <= '0';
    ELSIF rising_edge(Clk) THEN
        IF EnC = '1' THEN
            Cout <= Cout_tmp;
        END IF;
        IF EnV = '1' THEN
            V <= V_tmp;
        END IF;
    END IF;
END IF;
END PROCESS;
Counters
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;
ENTITY upcount IS
  PORT ( Reset, 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 Reset = '1' THEN
        Count <= "00" ;
      ELSE
        Count <= Count + 1 ;
      END IF ;
    END IF;
  END PROCESS;
  Q <= Count;
END behavioral;
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;

ENTITY upcount_ar IS
    PORT ( Clock, Reset, Enable : IN STD_LOGIC;
           Q : OUT STD_LOGIC_VECTOR (3 DOWNTO 0)) ;
END upcount_ar ;
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, Reset )
    BEGIN
        IF Reset = '1' 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

Diagram showing a shift register with inputs for Sin, Clock, and Enable, and outputs for Q(0) to Q(3).
Shift Register With Parallel Load

Load

D(3)

Sin

Clock

Enable

D(2)

D(1)

D(0)

Q(0)

Q(1)

Q(2)

Q(3)
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 ;
ARCHITECTURE behavioral OF shift4 IS
  SIGNAL Qt : STD_LOGIC_VECTOR(3 DOWNTO 0);
BEGIN
  PROCEDURE (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 PROCEDURE;
  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 ;

\begin{center}
\begin{tikzpicture}
  \node [draw, fill=red!20] (shiftn) {\textbf{\textsc{shiftn}}};

  \node (D) [left of=shiftn] {D};
  \node (Q) [right of=shiftn] {Q};
  \node (Enable) [above of=shiftn] {Enable};
  \node (Load) [below of=shiftn] {Load};
  \node (Sin) [below of=Load] {Sin};
  \node (Clock) [below of=Sin] {Clock};

  \draw[->] (D) -- (Q);
  \draw[->] (Enable) -- (shiftn);
  \draw[->] (Load) -- (shiftn);
  \draw[->] (Sin) -- (shiftn);
  \draw[->] (Clock) -- (shiftn);
\end{tikzpicture}
\end{center}
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

End IF ;

ELSE

End IF ;

END IF ;

END PROCESS ;

Q <= Qt;

END behavioral;
Generic Component Instantiation
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 Behavior OF regne IS
BEGIN
    PROCESS (Clock)
    BEGIN
        IF (Clock'EVENT AND Clock = '1' ) THEN
            IF Enable = '1' THEN
                Q <= D ;
            END IF ;
        END IF ;
    END PROCESS ;
END Behavior ;
Circuit built of medium scale components

\[ s(0) \rightarrow w_0 \rightarrow p(0) \rightarrow y_1 \rightarrow q(1) \rightarrow z(3) \]
\[ r(0) \rightarrow w_1 \rightarrow p(1) \rightarrow y_0 \rightarrow q(0) \rightarrow z(2) \]
\[ r(1) \rightarrow p(2) \rightarrow w_2 \rightarrow y_1 \rightarrow \text{优先级} \rightarrow \text{dec2to4} \]
\[ r(2) \rightarrow p(3) \rightarrow w_3 \rightarrow y_0 \rightarrow \text{ena} \]
\[ r(3) \rightarrow r(4) \rightarrow r(5) \]
\[ s(1) \rightarrow \text{En} \rightarrow \text{Enable} \rightarrow \text{D, Q, Clock, Regne, T(0), T(1), T(2), T(3)} \]
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 ;
Structural description – example (2)

VHDL-87

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

END structural;
Constants
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;
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.
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;
library ieee;
use ieee.std_logic_1164.all;

library work;
use work.alu_pkg.all;

entity alu_comb is

.............
Mixing Description Styles
Inside of an Architecture
VHDL Description Styles

- **dataflow**
  - Concurrent statements

- **structural**
  - Components and interconnects

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

**synthesizable**
Mixed Style Modeling

architecture ARCHITECTURE_NAME of ENTITY_NAME is

• Here you can declare signals, constants, types, etc.

begin

Concurrent statements:
• Simple signal assignment
• Conditional signal assignment
• Selected signal assignment

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);
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
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;
Non-synthesizable VHDL
Delays

Delays are not synthesizable

Statements, such as

\[ \text{wait for } 5 \text{ ns} \]
\[ a \leq b \text{ after } 10 \text{ ns} \]

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

```vhdl
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)

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 PROCESS;
Dual-edge triggered register/counter (3)

```vhdl
PROCESS (clk)
BEGIN
    IF (clk'EVENT) THEN
        counter <= counter + 1;
    END IF;
END PROCESS;

PROCESS (clk)
BEGIN
    counter <= counter + 1;
END PROCESS;
```