Designing with VHDL and FPGA

Instructor:

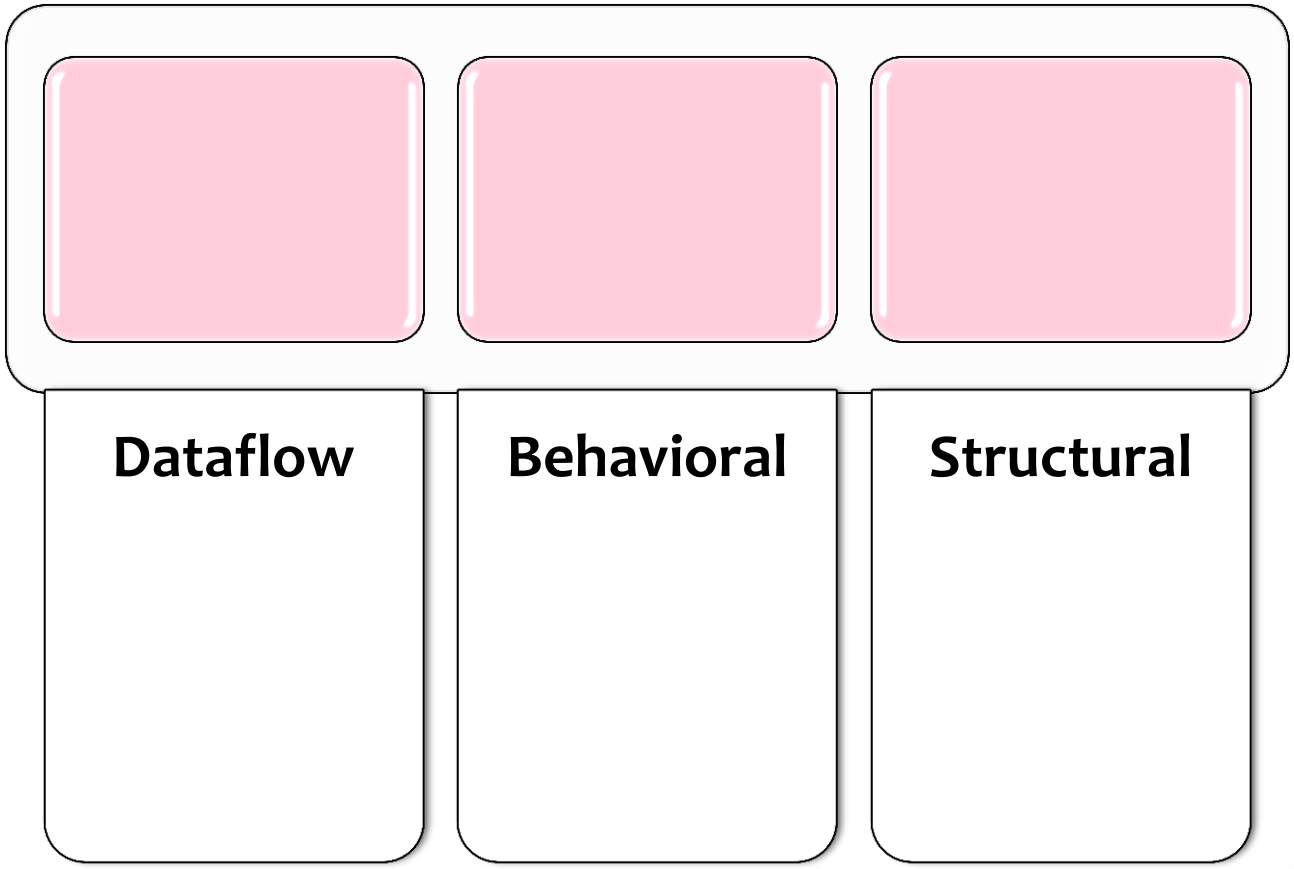
Dr. Ahmad El-Banna

LAB# 2
FALL 2016

Agenda

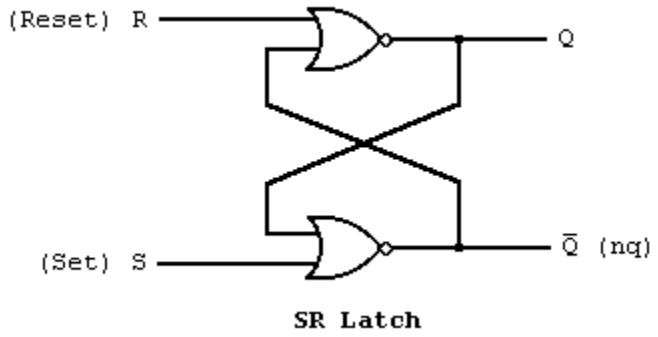
- VHDL language constructs
- Data flow and Behavioral Implementation

Remember!
3 ways to do the VHDL way



Jumping right in to a Model

- lets look at a **SR latch** model -- doing it the dataflow way..... ignore the extra junk for now –

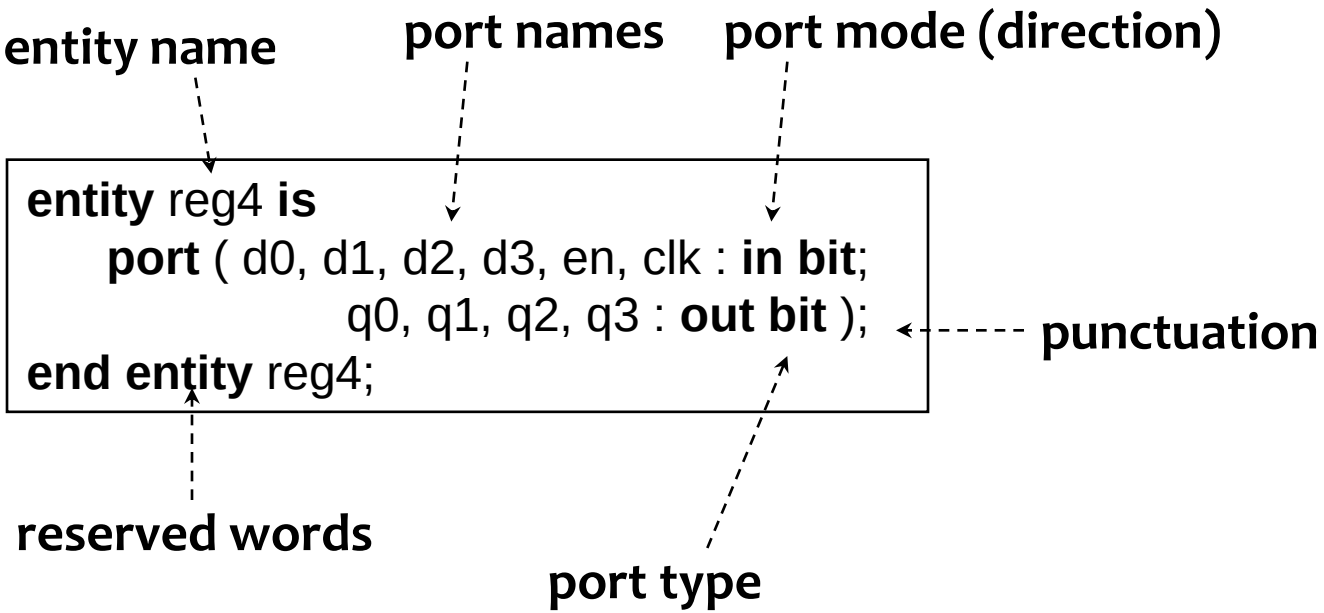


SR Latch

```
entity latch is
  port (s,r : in bit;
        q,nq : out bit);
end latch;

architecture dataflow of latch is
begin
  q<=r nor nq;
  nq<=s nor q;
end dataflow;
```

Modeling Interfaces

- *Entity* declaration
  - describes the input/output *ports* of a module



Basic Constructs

- Comments
- Objects
  - Signals
  - Variables
  - Constants
  - Alias

```
--*******************************************************
-- example to show the caveat of the out mode
--*******************************************************
architecture arch of mode_demo is
   signal ab: std_logic; -- ab is the internal signal
begin
   ab <= a and b;
   x <= ab;                    -- ab connected to the x output
   y <= not ab;
end eg_arch ;
```

```
variable variable_name, variable_name, ... : data_type
```

```
constant BUS_WIDTH: integer := 32;
constant BUS_BYTES: integer := BUS_WIDTH / 8;
```

```
signal word: std_logic_vector(15 downto 0);
alias op: std_logic_vector(6 downto 0) is word(15 downto 9);
alias reg1: std_logic_vector(2 downto 0) is word(8 downto 6);
alias reg2: std_logic_vector(2 downto 0) is word(5 downto 3);
alias reg3: std_logic_vector(2 downto 0) is word(2 downto 0);
```

Data Types in vhdl

- **IEEE1164_std_logic package** contains the basic standard logic.
- **IEEE numeric_std package** contains the arithmetic operations.

- Examples of data types :
  - integer: minimal range: $-(2^{31}-1)$ to $2^{31}-1$
  - boolean: (false, true)
  - bit: ('0', '1')
  - bit_vector: a one-dimensional array of bits

Data Types in vhdl..

IEEE std_logic_1164 package

- New data type: std_logic, std_logic_vector
- std_logic:
  - '0', '1': forcing logic ‘0' and forcing logic 1
  - 'Z': high-impedance, as in a tri-state buffer.
  - 'L' , 'H': weak logic 0 and weak logic 1, as in wired logic.
  - 'U': for uninitialized.
  - '-': don't-care.
- std_logic_vector
  - EX: signal a: std_logic_vector(7 downto 0);

Operators in vhdl

operator	description	data type of operand a	data type of operand b	data type of result
`a ** b`	exponentiation	`integer`	`integer`	`integer`
abs `a`	absolute value	`integer`		`integer`
not `a`	negation	`boolean`, `bit`, `bit_vector`		`boolean`, `bit`, `bit_vector`
`a * b`	multiplication	`integer`	`integer`	`integer`
`a / b`	division			
`a` **mod** `b`	modulo			
`a` **rem** `b`	remainder			
`+ a`	identity	`integer`		`integer`
`- a`	negation			
`a + b`	addition	`integer`	`integer`	`integer`
`a - b`	subtraction			
`a & b`	concatenation	1-D array, element	1-D array, element	1-D array

Operators in vhdl..

a **sll** b	shift left logical	bit_vector	integer	bit_vector
a **srl** b	shift right logical			
a **sla** b	shift left arithmetic			
a **srl** b	shift right arithmetic			
a **rol** b	rotate left			
a **ror** b	rotate right			
a = b	equal to	any	same as a	boolean
a /= b	not equal to			
a < b	less than	scalar or 1-D array	same as a	boolean
a <= b	less than or equal to			
a > b	greater than			
a >= b	greater than or equal to			
a **and** b	and	boolean, bit, bit_vector	same as a	boolean, bit, bit_vector
a **or** b	or			
a **xor** b	xor			
a **nand** b	nand			
a **nor** b	nor			
a **xnor** b	xnor			

Precedence of the VHDL Operators

Table 3.2 Precedence of the VHDL operators

Precedence	Operators
Highest	** abs not
	* / mod rem
	+ - (identity and negation)
	& + - (addition and subtraction)
	sll srl sla sra rol ror
	= /= < <= > >=
Lowest	and or nand nor xor xnor

Precedence of the VHDL Operators

```
a + b > c or a < d
```

The + operator will be evaluated first, and then the > and < operators, and then the **or** operator.

```
a + b + c + d
```

The `a + b` expression will be evaluated first, and then `c` is added, and then `d` is added.

expression be evaluated from right to left:

```
a + (b + (c + d))
```

Unlike the logic expression used in Boolean algebra, the **and** and **or** operators have the same precedence in VHDL, and thus we must use parentheses to specify the desired order, as in

```
(a and b) or (c and d)
```

VHDL vs. Boolean

Table 3.3 VHDL operators versus conventional Boolean algebra notations

VHDL operator	Boolean algebra notation
not	$'$
and	$\cdot$
or	$+$
xor	$\oplus$
+	$+$

$$y = a \cdot b + a' \cdot b'$$

When coded in VHDL, This expression becomes

```
y <= (a and b) or ((not a) and (not b));
```

Overloaded Operators in vhdl

There may exist multiple functions with the same name, each for a different data type.
This is known as overloading of a function or operator.

overloaded operator	data type of operand a	data type of operand b	data type of result
not a	std_logic_vector std_logic		same as a
a **and** b a **or** b a **xor** b a **nand** b a **nor** b a **xnor** b	std_logic_vector std_logic	same as a	same as a

Concatenation operator

For example, we can shift the elements of the array to the right by two positions and append two 0's to the front:

```
y <= "00" & a(7 downto 2);
```

or append the MSB to the front (known as an arithmetic shift):

```
y <= a(7) & a(7) & a(7 downto 2);
```

or rotate the elements to the right by two positions:

```
y <= a(1 downto 0) & a(7 downto 2);
```

Array aggregate

```
a <= "10100000";

a <= ('1','0','1','0','0','0','0','0');

a <= (7=>'1', 6=>'0', 0=>'0', 1=>'0', 5=>'1',
      4=>'0', 3=>'0', 2=>'0');

a <= (7|5=>'1', 6|4|3|2|1|0=>'0');

a <= (7|5=>'1', others=>'0');
```

```
a <= (others=>'0');
```

It is more compact than

```
a <= "00000000";
```

Designing with VHDL and FPGA

Instructor:

Dr. Ahmad El-Banna

LAB# 3
FALL 2016

Agenda

- Type conversions
- Functions in std package
- Process, Signals and Variables

Types Conversion

function	data type of operand a	data type of result
to_bit(a)	std_logic	bit
to_stdulogic(a)	bit	std_logic
to_bit_vector(a)	std_logic_vector	bit_vector
to_stdlogicvector(a)	bit_vector	std_logic_vector

Functions in std package

Table 3.7 Functions in the IEEE `numeric_std` package

Function	Description	Data type of operand `a`	Data type of operand `b`	Data type of result

`shift_left(a,b)`
`shift_right(a,b)`
`rotate_left(a,b)`
`rotate_right(a,b)`
	shift left
	shift right
	rotate left
	rotate right
		`unsigned, signed`	`natural`	same as a

`resize(a,b)`	resize array	`unsigned, signed`	`natural`	same as a

`std_match(a,b)`	compare '-'	`unsigned, signed`
		`std_logic_vector,`
		`std_logic`
			same as a	`boolean`

`to_integer(a)`	data type conversion	`unsigned, signed`		`integer`
`to_unsigned(a,b)`		`natural`	`natural`	`unsigned`
`to_signed(a,b)`		`integer`	`natural`	`signed`

Modeling the Behavior way

- *Architecture body*
  - describes an implementation of an entity
  - may be several per entity
- *Behavioral* architecture
  - describes the algorithm performed by the module
  - contains
    - *process statements*, each containing
      - *sequential statements*, including
        - *signal assignment statements* and
        - *wait statements*

VHDL Process

- Similar to *function* in C lang.
- Contains a set of sequential statements to be executed sequentially
- The whole process is a concurrent statement
- Can be interpreted as a circuit part enclosed inside of a black box
- May or may not be able to be mapped to physical hardware

- Two types of process
  - A process with a sensitivity list
  - A process with wait statement

A process with a sensitivity list

- A process is like a circuit part, which can be
  - active (known *activated*)
  - inactive (known as *suspended*).
- A process is activated when a signal in the sensitivity list changes its value
- Its statements will be executed sequentially until the end of the process

- Syntax

```
process(sensitivity_list)
  declarations;
begin
  sequential statement;
  sequential statement;
  . . .
end process;
```

example

- E.g, 3-input and circuit

```
signal a,b,c,y: std_logic;
process(a,b,c)
begin
   y <= a and b and c;
end process;
```

- How to interpret this:

```
process(a)
begin
   y <= a and b and c;
end process;
```

- For a combinational circuit, all input should be included in the sensitivity list

A process with wait statement

- Process has no sensitivity list
- Process continues the execution until a wait statement is reached and then suspended
- Forms of wait statement:
  - **wait on** signals;
  - **wait until** boolean_expression;
  - **wait for** time_expression;

E.g, 3-input and circuit

```
process
begin
    y <= a and b and c;
    wait on a, b, c;
end process;
```

- A process can has multiple wait statements
- Process with sensitivity list is preferred for synthesis

Sequential signal assignment

- Syntax

  signal_name <= value_expression;

- Syntax is identical to the simple concurrent signal assignment

```
signal Xtmp: bit;
```

- Caution:
  - Inside a process, a signal can be assigned multiple times, but only the last assignment takes effect

- E.g.,

```
process(a,b,c,d)
begin                          -- y_entry := y
   y <= a or c;                -- y_exit := a or c;
   y <= a and b;               -- y_exit := a and b;
   y <= c and d;               -- y_exit := c and d;
end process;                   -- y <= y_exit
```

- It is same as

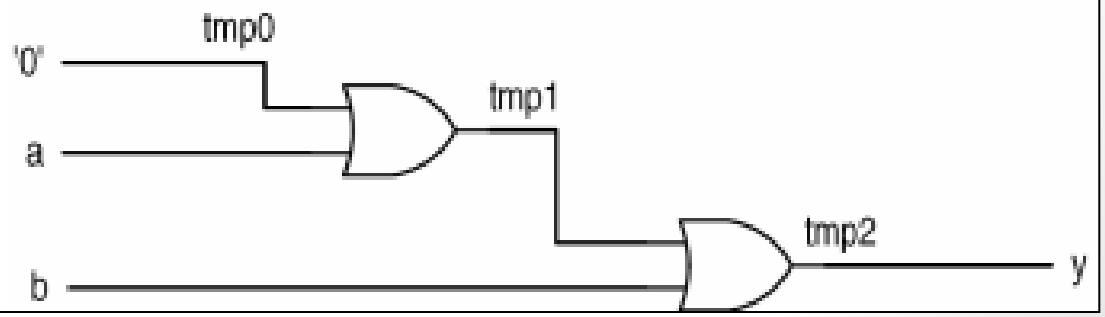
```
process(a,b,c,d)
begin
   y <= c and d;
end process;
```

Variables vs. signals

```
process(a,b,c)
  variable tmp: std_logic;
begin
   tmp := '0';
   tmp := tmp or a;
   tmp := tmp or b;
   y <= tmp;
end process;
```

- interpretation:

```
process(a,b,c)
  variable tmp0, tmp1, tmp2: std_logic;
begin
   tmp0 := '0';
   tmp1 := tmp0 or a;
   tmp2 := tmp1 or b;
   y <= tmp2;
end process;
```



Variables vs. signals ..

- What happens if signal is used?

```
process(a,b,c,tmp)
begin                    -- tmp_entry := tmp
   tmp <= '0';           -- tmp_exit := '0';
   tmp <= tmp or a;      -- tmp_exit := tmp_entry or a;
   tmp <= tmp or b;      -- tmp_exit := tmp_entry or b;
end process;             -- tmp <= tmp_exit
```

- Same as:

```
process(a,b,c,tmp)
begin
   tmp <= tmp or b;
end process;
```

Designing with VHDL and FPGA

Instructor:

Dr. Ahmad El-Banna

LAB# 4
FALL 2016

Agenda

- Sequential Statements
- if, case, for loop, … etc
- Examples

If statement

Syntax

```
if boolean_expr_1 then
   sequential_statements;
elsif boolean_expr_2 then
   sequential_statements;
elsif boolean_expr_3 then
   sequential_statements;
. . .
else
   sequential_statements;
end if;
```

E.g., 4-to-1 mux

```
architecture if_arch of mux4 is
begin
    process(a,b,c,d,s)
    begin
        if (s="00") then
            x <= a;
        elsif (s="01")then
            x <= b;
        elsif (s="10")then
            x <= c;
        else
            x <= d;
        end if;
    end process;
end if_arch;
```

input s	output x
0 0	a
0 1	b
1 0	c
1 1	d

Example 2

E.g., 2-to-2^2 binary decoder

```
architecture if_arch of decoder4 is
begin
    process(s)
    begin
        if (s="00") then
            x <= "0001";
        elsif (s="01")then
            x <= "0010";
        elsif (s="10")then
            x <= "0100";
        else
            x <= "1000";
        end if;
    end process;
end if_arch;
```

input s	output x
0 0	0001
0 1	0010
1 0	0100
1 1	1000

Example 3

E.g., 4-to-2 priority encoder

```
architecture if_arch of prio_encoder42 is
begin
    process(r)
    begin
        if (r(3)='1') then
            code <= "11";
        elsif (r(2)='1')then
            code <= "10";
        elsif (r(1)='1')then
            code <= "01";
        else
            code <= "00";
        end if;
    end process;
    active <= r(3) or r(2) or r(1) or r(0);
end if_arch;
```

input	output	
r	code	active
1 - - -	11	1
0 1 - -	10	1
0 0 1 -	01	1
0 0 0 1	00	1
0 0 0 0	00	0

Comparison to conditional signal assignment

- Two statements are the same if there is only one output signal in if statement
- If statement is more flexible
- Sequential statements can be used in then, elsif and else branches:
  - Multiple statements
  - Nested if statements

Note:

According to VHDL definition:

- Only the "then" branch is required; "elsif" and "else" branches are optional
- Signals do not need to be assigned in all branch
- When a signal is unassigned due to omission, it keeps the "previous value" (implying "memory")

example

```
sig <= value_expr_1 when boolean_expr_1 else
       value_expr_2 when boolean_expr_2 else
       value_expr_3 when boolean_expr_3 else
          . . .
       value_expr_n;
```

It can be written as

```
process(...)
   if boolean_expr_1 then
      sig <= value_expr_1;
   elsif boolean_expr_2 then
      sig <= value_expr_2;
   elsif boolean_expr_3 then
      sig <= value_expr_3;
   . . .
   else
      sig <= value_expr_n;
   end if;
end process
```

e.g., find the max of a, b, c

```
if (a > b) then
   if (a > c) then
      max <= a;   -- a>b and a>c
   else
      max <= c;   -- a>b and c>=a
   end if;
else
   if (b > c) then
      max <= b;   -- b>=a and b>c
   else
      max <= c;   -- b>=a and c>=b
   end if;
end if;
```

Case statement

Syntax

```
case case_expression is
   when choice_1 =>
      sequential statements;
   when choice_2 =>
      sequential statements;
   . . .
   when choice_n =>
      sequential statements;
end case;
```

E.g., 4-to-1 mux

```
architecture case_arch of mux4 is
begin
    process(a,b,c,d,s)
    begin
        case s is
            when "00" =>
                x <= a;
            when "01" =>
                x <= b;
            when "10" =>
                x <= c;
            when others =>
                x <= d;
        end case;
    end process;
end case_arch;
```

input s	output x
0 0	a
0 1	b
1 0	c
1 1	d

Example 2

E.g., 2-to-2^2 binary decoder

```
architecture case_arch of decoder4 is
begin
    proc1:
    process(s)
    begin
        case s is
            when "00" =>
                x <= "0001";
            when "01" =>
                x <= "0010";
            when "10" =>
                x <= "0100";
            when others =>
                x <= "1000";
        end case;
    end process;
END case_arch;
```

input s	output x
0 0	0001
0 1	0010
1 0	0100
1 1	1000

Example 3

E.g., 4-to-2 priority encoder

```
architecture case_arch of prio_encoder42 is
begin
    process(r)
    begin
        case r is
            when "1000"|"1001"|"1010"|"1011"|
                 "1100"|"1101"|"1110"|"1111" =>
                code <= "11";
            when "0100"|"0101"|"0110"|"0111" =>
                code <= "10";
            when "0010"|"0011" =>
                code <= "01";
            when others =>
                code <= "00";
        end case;
    end process;
    active <= r(3) or r(2) or r(1) or r(0);
end case_arch;
```

input	output	
r	code	active
1 - - -	11	1
0 1 - -	10	1
0 0 1 -	01	1
0 0 0 1	00	1
0 0 0 0	00	0

Case versus if choices

- Note
  - Choice values can NOT be duplicated in ***case*** statements
  - However, expressions can be duplicated in ***if*** statement.

```
begin
    process(s)                          syntax Error
    begin
     case s is
        when "000" | "001"   => y <= (7=>'1' , others
        when "001" => y <= (6=>'1' , others =>'0');
        when "010" => y <= (5=>'1' , others =>'0');
        when "011" => y <= (4=>'1' , others =>'0');
        when "100" => y <= (3=>'1' , others =>'0');
        when "101" => y <= (2=>'1' , others =>'0');
        when "110" => y <= (1=>'1' , others =>'0');
        when others => y <= (0=>'1' , others =>'0');
      end case;
```

```
process(r)
begin
  if(r(3)='1') or (r(2)='1')then
     cod<="11";

   elsif(r(2)='1')then
     cod<="10";                          syntax OK

     elsif(r(1)='1')then
     cod<="01";
```

Comparison to selected signal assignment

- Two statements are the same if there is only one output signal in case statement
- Case statement is more flexible
- Sequential statements can be used in choice branches

Note:

- According to VHDL definition:
  - Signals do not need to be assigned in all choice branch
  - When a signal is unassigned, it keeps the "previous value" (implying "memory")

Comparison to selected signal assignment ..

```
with sel_exp select
   sig <= value_expr_1 when choice_1,
          value_expr_2 when choice_2,
          value_expr_3 when choice_3,
          . . .
          value_expr_n when choice_n;
```

It can be rewritten as:

```
case sel_exp is
   when choice_1 =>
      sig <= value_expr_1;
   when choice_2 =>
      sig <= value_expr_2;
   when choice_3 =>
      sig <= value_expr_3;
   . . .
   when choice_n =>
      sig <= value_expr_n;
end case;
```

Designing with VHDL and FPGA

Instructor:

Dr. Ahmad El-Banna

LAB# 5
FALL 2016

Simple for loop statement

- VHDL provides a variety of loop constructs
- Only a restricted form of loop can be synthesized
- Syntax of simple for loop:

```
for index in loop_range loop
    sequential statements;
end loop;
```

- loop_range must be static
- Index assumes value of loop_range from left to right

Example

- E.g., bit-wide xor

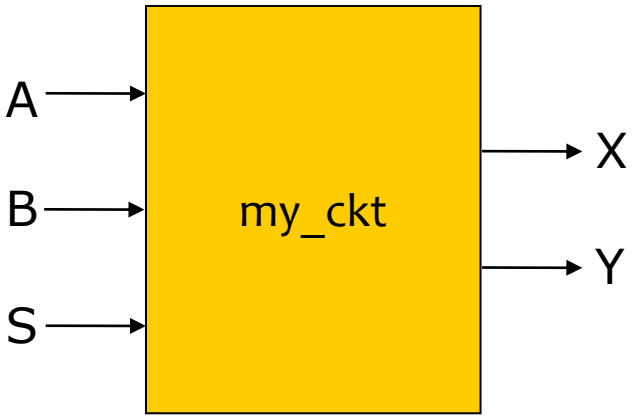
```
library ieee;
use ieee.std_logic_1164.all;

entity wide_xor is
    port(
        a, b: in std_logic_vector(3 downto 0);
        y: out std_logic_vector(3 downto 0)
    );
end wide_xor;

architecture demo_arch of wide_xor  is
    constant WIDTH: integer := 4;
begin
    process(a, b)
    begin
        for i in (WIDTH-1) downto 0 loop
            y(i) <= a(i) xor b(i);
        end loop;
    end process;
end demo_arch;
```

Example 2

- E.g., reduced-xor

```
library ieee;
use ieee.std_logic_1164.all;

entity reduced_xor_demo is
   port(
      a: in std_logic_vector(3 downto 0);
      y: out std_logic
   );
end reduced_xor_demo;

architecture demo_arch of reduced_xor_demo is
   constant WIDTH: integer := 4;
   signal tmp: std_logic_vector(WIDTH-1 downto 0);
begin
   process(a,tmp)
   begin
      tmp(0) <= a(0);    -- boundary bit
      for i in 1 to (WIDTH-1) loop
         tmp(i) <= a(i) xor tmp(i-1);
      end loop;
   end process;
   y <= tmp(WIDTH-1);
end demo_arch;
```

Example



- Example: my_ckt
  - Inputs: A, B, C
  - Outputs: X, Y
- VHDL description:

```
entity my_ckt is
port (
        A: in bit;
        B: in bit;
        S: in bit;
        X: out bit;
        Y: out bit);
end my_ckt ;
```

- Functional Spec.
  - Behavior for output X:
    - When S = 0
      X <= A
    - When S = 1
      X <= B
  - Behavior for output Y:
    - When X = 0 and S =0
      Y <= '1'
    - Else
      Y <= '0'

VHDL Architecture

- VHDL description (sequential behavior):

```
architecture arch_name of my_ckt is
begin
 p1: process (A,B,S)
 begin
  if (S='0') then
   X <= A;
  else
   X <= B;
  end if;

  if ((X = '0') and (S = '0')) then
   Y <= '1';
  else
   Y <= '0';
  end if;

 end process p1;
end;
```

Error: Signals defined as output ports can only be driven and not read

VHDL Architecture..

```
architecture behav_seq of my_ckt is

  signal Xtmp: bit;

  begin
   p1: process (A,B,S,Xtmp)
      begin
    if (S='0') then
     Xtmp <= A;
    else
     Xtmp <= B;
    end if;

    if ((Xtmp = '0') and (S = '0')) then
     Y <= '1';
    else
     Y <= '0';
    end if;

    X <= Xtmp;
   end process p1;
end;
```

Signals can only be defined in this place before the `begin` keyword

General rule: Include all signals in the sensitivity list of the process which either appear in relational comparisons or on the right side of the assignment operator inside the `process` construct.

In our example:
`Xtmp` and `S` occur in relational comparisons
`A`, `B` and `Xtmp` occur on the right side of the assignment operators

Example: VHDL Architecture…

- VHDL description (concurrent behavior):

```
architecture behav_conc of my_ckt is

  signal Xtmp: bit;

begin

  Xtmp <= A when (S='0') else
          B;
  Y <= '1' when ((Xtmp = '0') and (S = '0')) else
       '0';

  X <= Xtmp;

end ;
```

EXAMPLES

Comparator Example

Behavior Code

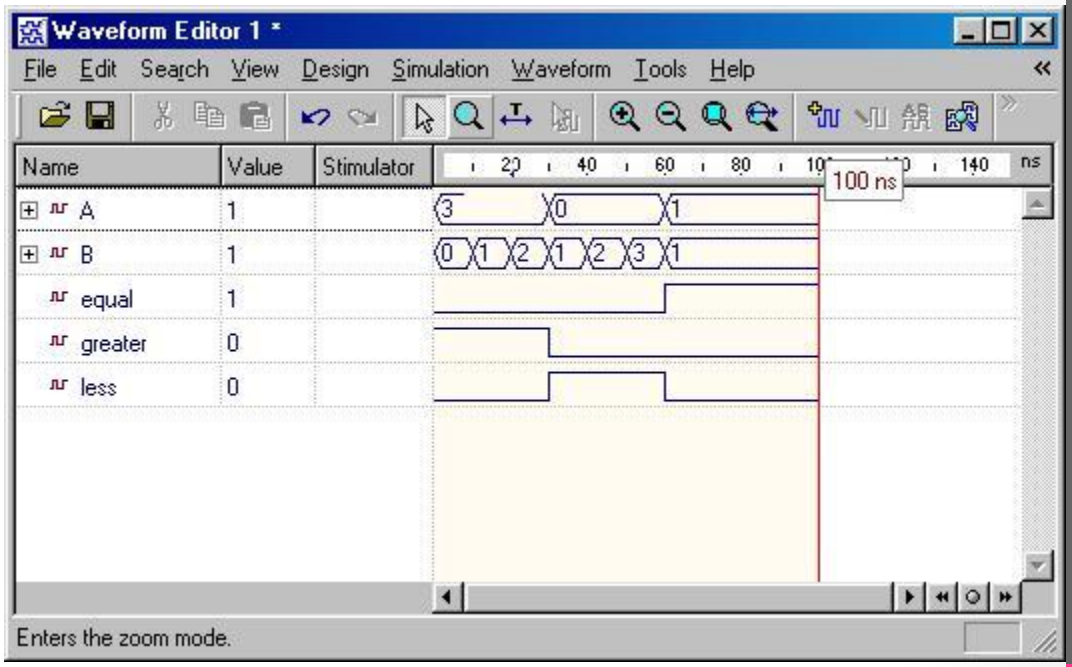
```
-------------------------------------------
-- n-bit Comparator (ESD book figure 2.5) by Weijun Zhang, 04/2001
-- this simple comparator has two n-bit inputs &  three 1-bit outputs
-------------------------------------------
 library ieee;
use ieee.std_logic_1164.all;
-------------------------------------------
entity Comparator is
generic(n: natural :=2);
port(       A:          in std_logic_vector(n-1 downto 0);
            B:          in std_logic_vector(n-1 downto 0);
            less:                   out std_logic;
            equal:                  out std_logic;
            greater:    out std_logic
);
end Comparator;
-------------------------------------------
```

Comparator Example..

Behavior Code & Simulation Waveforms

```
architecture behv of Comparator is
begin
      process(A,B)
   begin
      if (A<B) then
                  less <= '1';
                  equal <= '0';
                  greater <= '0';
      elsif (A=B) then
                  less <= '0';
                  equal <= '1';
                  greater <= '0';
      else
                  less <= '0';
                  equal <= '0';
                  greater <= '1';
               end if;
   end process;
end behv;
```

Simulation Waveforms



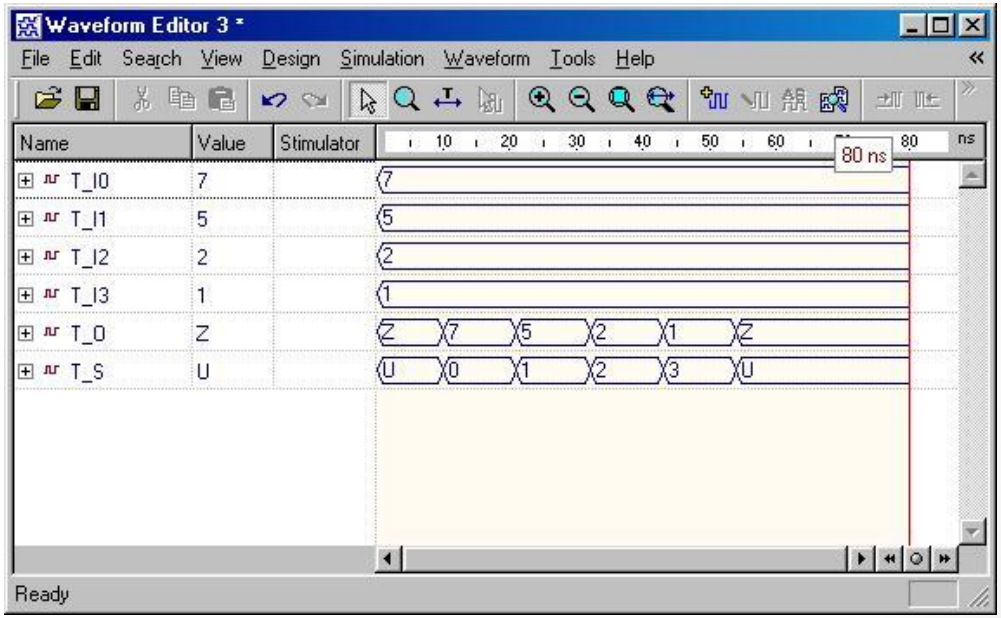
4x1 Multiplexer

```
----------------------------------------------
-- VHDL code for 4:1 multiplexor-- (ESD book figure 2.5)-- by Weijun Zhang, 04/2001
---- Multiplexor is a device to select different  inputs to outputs. we use 3 bits vector to  -- describe its I/O ports
----------------------------------------------
library ieee;
use ieee.std_logic_1164.all;

----------------------------------------------

entity Mux is
port(       I3:         in std_logic_vector(2 downto 0);
            I2:         in std_logic_vector(2 downto 0);
            I1:         in std_logic_vector(2 downto 0);
            I0:         in std_logic_vector(2 downto 0);
            S:          in std_logic_vector(1 downto 0);
            O:          out std_logic_vector(2 downto 0)
);
end Mux;
----------------------------------------------
```

4x1 Multiplexer ..

```
architecture behv1 of Mux is
begin
   process(I3,I2,I1,I0,S)
   begin
      -- use case statement
      case S is
                  when "00" =>          O <= I0;
                  when "01" =>          O <= I1;
                  when "10" =>          O <= I2;
                  when "11" =>          O <= I3;
                  when others =>  O <= "ZZZ";
               end case;
   end process;
end behv1;
----------
architecture behv2 of Mux is
begin
  -- use when.. else statement
   O <=        I0 when S="00" else
               I1 when S="01" else
               I2 when S="10" else
               I3 when S="11" else
               "ZZZ";
end behv2;
-----------------------------------------------
```

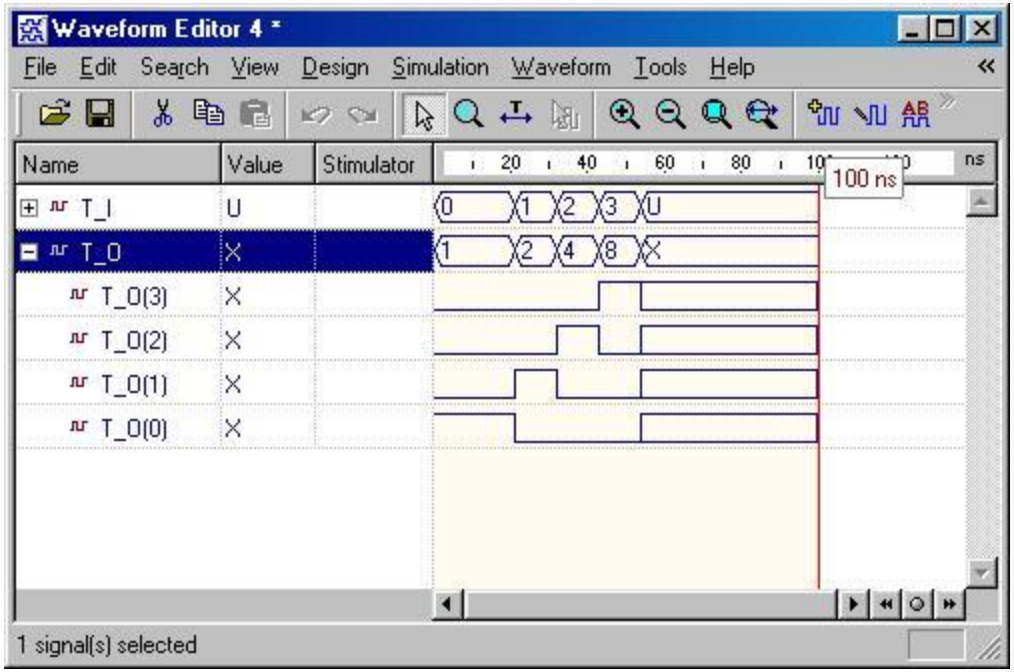
Simulation Waveforms



Decoder

```
----------------------------------------
-- 2:4 Decoder (ESD figure 2.5)-- by Weijun Zhang, 04/2001
-- decoder is a kind of inverse process-- of multiplexor
----------------------------------------
library ieee;
use ieee.std_logic_1164.all;

----------------------------------------

entity DECODER is
port(    I:          in std_logic_vector(1 downto 0);
         O:          out std_logic_vector(3 downto 0)
);
end DECODER;

----------------------------------------
```

```
architecture behv of DECODER is
begin
   -- process statement
   process (I)
   begin
        -- use case statement
      case I is
                     when "00" => O <= "0001";
                     when "01" => O <= "0010";
                     when "10" => O <= "0100";
                     when "11" => O <= "1000";
                     when others => O <= "XXXX";
                  end case;
   end process;
end behv;

architecture when_else of DECODER is
begin
   -- use when..else statement

   O <=          "0001" when I = "00" else
                 "0010" when I = "01" else
                 "0100" when I = "10" else
                 "1000" when I = "11" else
                 "XXXX";
end when_else;
-----------------------------------------
```



- For more details, refer to:
  - VHDL Tutorial: Learn by Example *by Weijun Zhang*
    - http://esd.cs.ucr.edu/labs/tutorial/
  - "**Introduction to VHDL**" presentation by Dr. Adnan Shaout, *The University of Michigan-Dearborn*
  - **The VHDL Cookbook**, *Peter J. Ashenden, 1st edition, 1990.*
- For inquires, send to:
  - ahmad.elbanna@feng.bu.edu.eg