BENHA UNIVERSITY
FACULTY OF ENGINEERING AT SHOUBRA

ECE-322
Electronic Circuits (B)

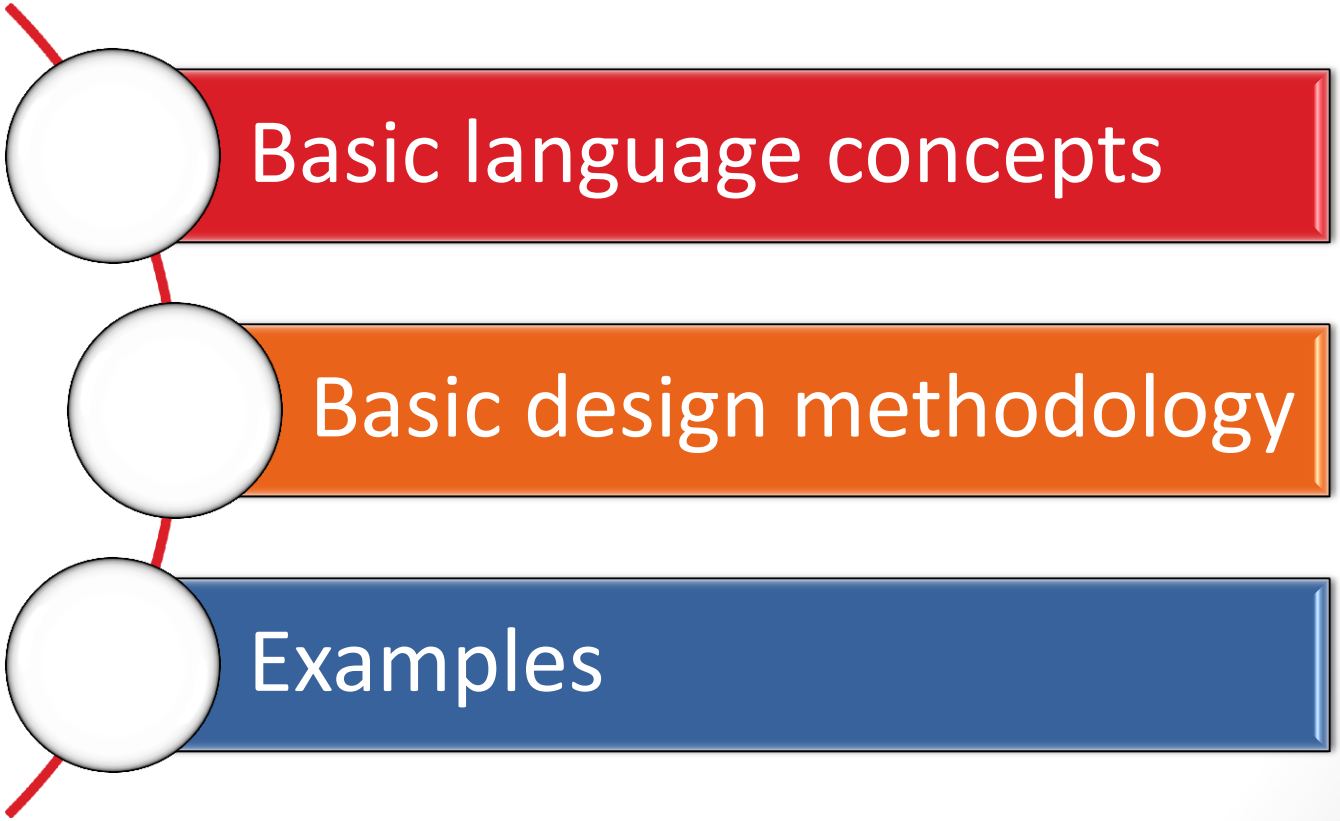
Lecture #10
VHDL Basics

Instructor:
Dr. Ahmad El-Banna

SPRING 2015

Agenda

Quick introduction to VHDL



VHDL

- Hardware description languages (HDL)
  - Language to describe hardware
  - Two popular languages
    - **VHDL**: **V**ery High Speed Integrated Circuits **H**ardware **D**escription **L**anguage
      - Developed by US DOD from 1983
      - IEEE Standard 1076-1987/1993/200x
      - Based on the ADA language
    - **Verilog**
      - IEEE Standard 1364-1995/2001/2005
      - Based on the C language
- Applications of HDL:
  - Model and document digital systems
    - Different levels of abstraction
      - Behavioral, structural, etc.
  - Verify design
  - Synthesize circuits
    - Convert from higher abstraction levels to lower abstraction levels

Modeling Digital Systems

- VHDL is for coding models of a digital system...
- Reasons for modeling
  - requirements specification
  - documentation
  - testing using simulation
  - formal verification
  - synthesis
- Goal
  - most ‘reliable’ design process, with minimum cost and time
  - avoid design errors!

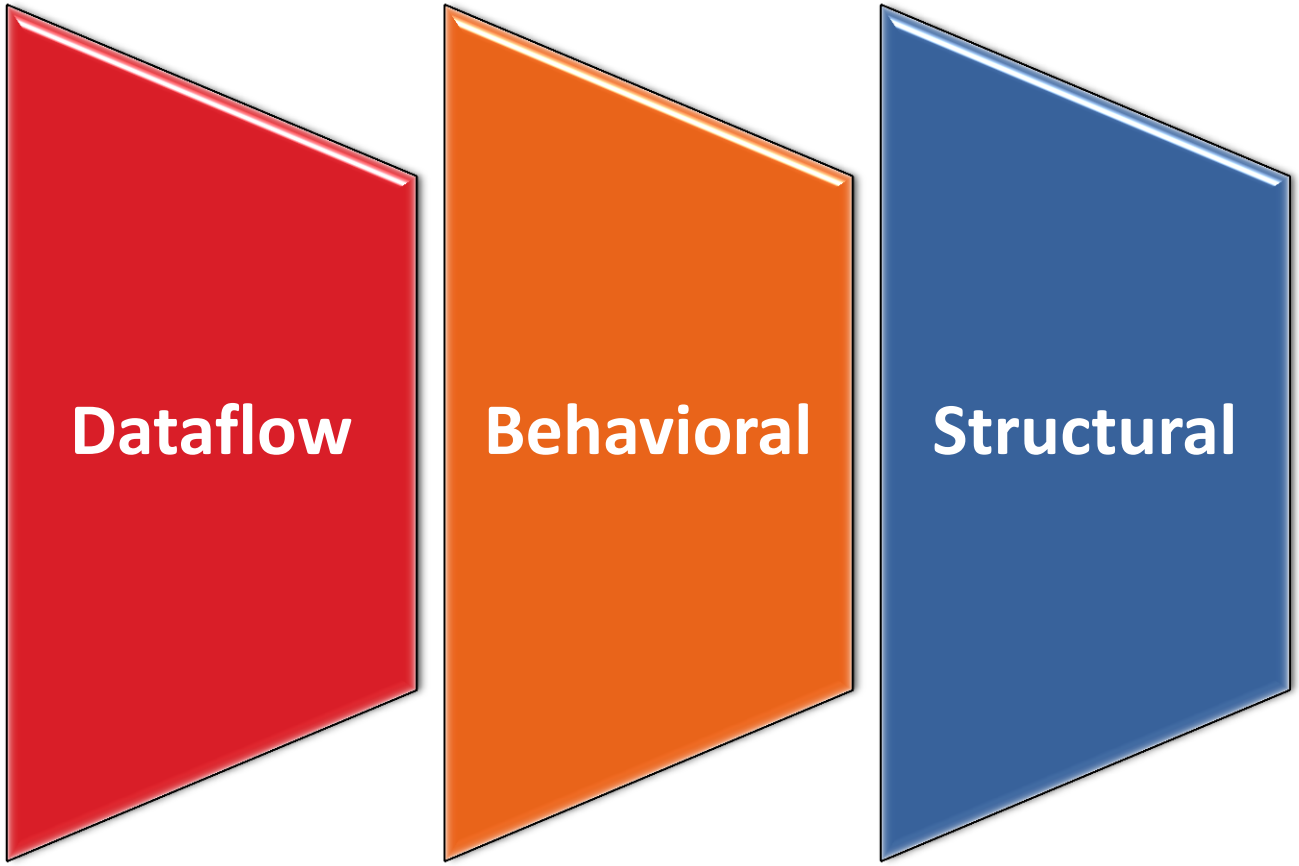
Basic VHDL Concepts

- Main Terms
  - Interfaces -- i.e. ports
  - Behavior
  - Structure
  - Test Benches
  - Analysis, simulation
  - Synthesis

- VHDL is a programming language that allows one to model and develop complex digital systems in a dynamic environment.
- Object Oriented methodology -- modules can be used and reused.
- Allows you to designate in/out ports (bits) and specify behavior or response of the system.

- But VHDL is NOT C ...
  There are some similarities, as with any programming language, but syntax and logic are quite different.

3 ways to DO IT -- the VHDL way



Modeling the Dataflow way

- uses statements that defines the actual flow of data.....

  such as,

  x <= y -- this is NOT less than equal to

  -- told you its not C

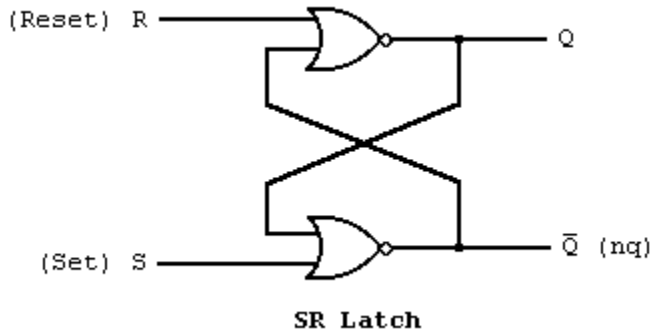
  this assigns the boolean signal x to the value of boolean signal y... i.e.

  x = y

  this will occur whenever y changes....

Jumping right in to a Model

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

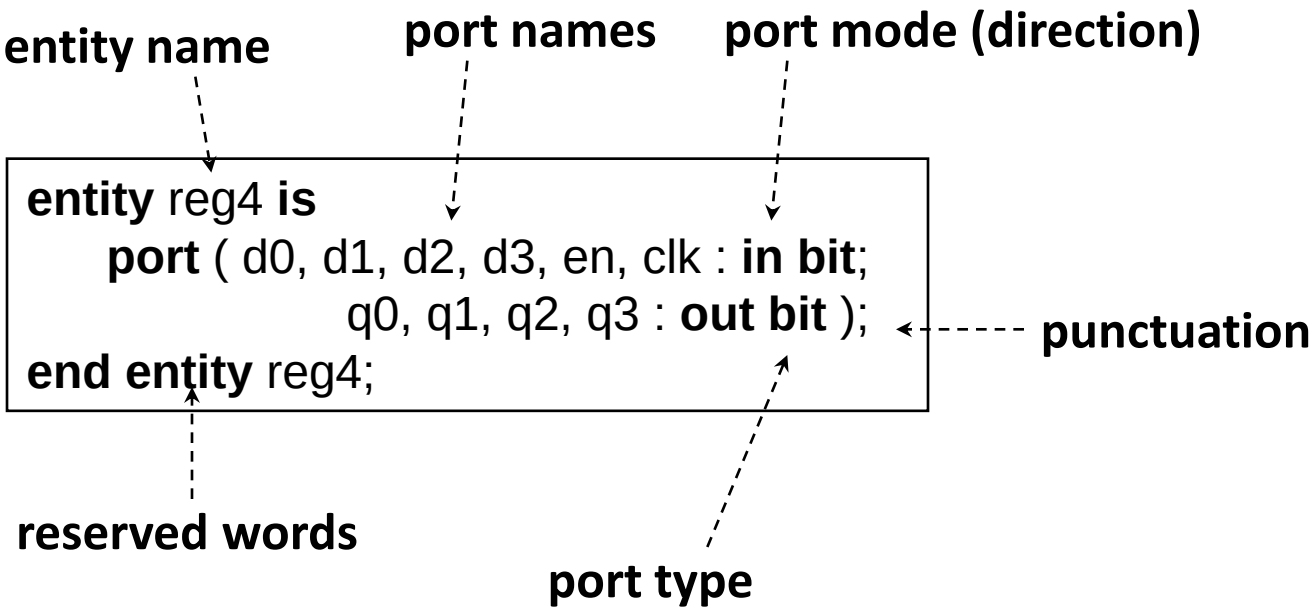


SR Latch

```
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

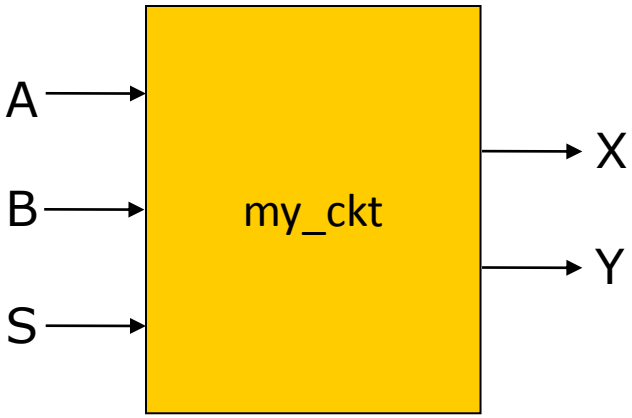
- *Entity* declaration
  - describes the input/output *ports* of a module



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*

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

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 ;
```

Test Bench your Model

- Testing a design by simulation
- Use a *test bench* model
  - a Model that uses your Model
  - apply test sequences to your inputs
  - monitors values on output signals
    - either using simulator
    - or with a process that verifies correct operation
    - or logic analyzer

Analysis

- Check for syntax and logic errors
  - syntax: grammar of the language
  - logic: how your Model responds to stimuli/changes
- Analyze each *design unit* separately
  - entity declaration
  - architecture body
  - …
  - put each design unit in a separate file -- *helps a lot.*
- Analyzed design units are placed in a *library*
  - make sure your Model is truly OOP

Simulation

- Discrete event simulation
  - time advances in discrete steps
  - when signal values change—*events* occur
- A processes is *sensitive* to events on input signals
- Initialization phase
  - each signal is given its initial value
  - simulation time set to 0
  - for each process
    - activate
    - execute until a wait statement, then suspend

Simulation Algorithm..

- Simulation cycle
  - advance simulation time to time of next transaction
  - for each transaction at this time
    - update signal value
      - event if new value is different from old value
  - for each process sensitive to any of these events, or whose “wait for …” time-out has expired
    - resume
    - execute until a wait statement, then suspend
- Simulation finishes when there are no further scheduled transactions

Comparator Example

Behavior Code

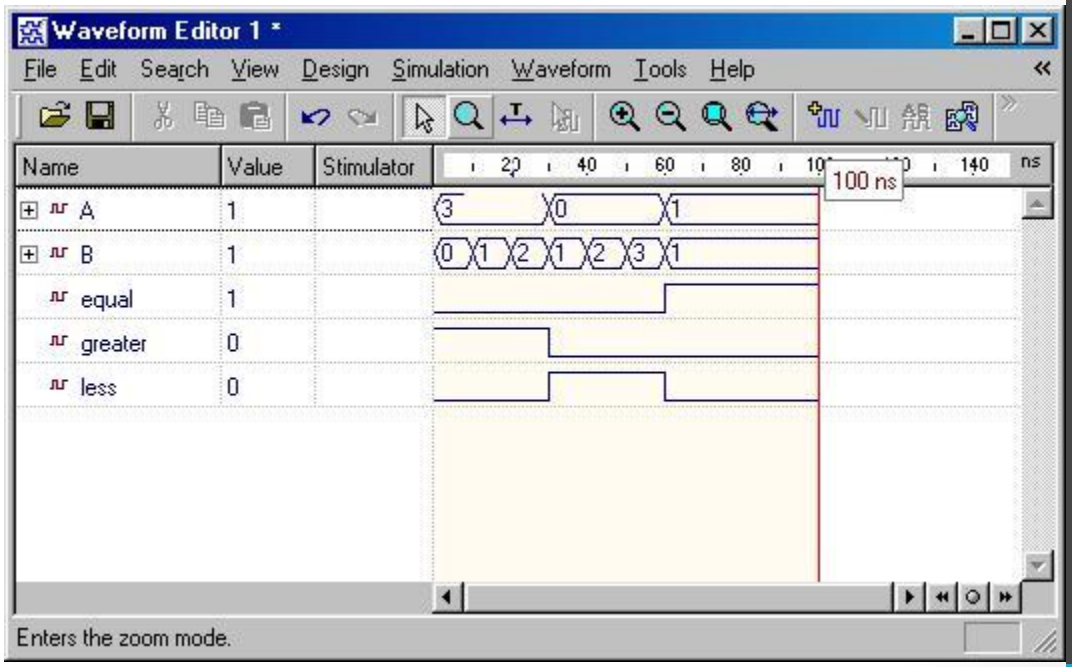
```
-------------------------------------------------
-- n-bit Comparator (ESD book figure 2.5) by Weijun Zhang, 04/2001
-- this simple comparator has two n-bit inputs &  three 1-bit outputs
-------------------------------------------------
 library ieee;
use ieee.std_logic_1164.all;
-------------------------------------------------
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



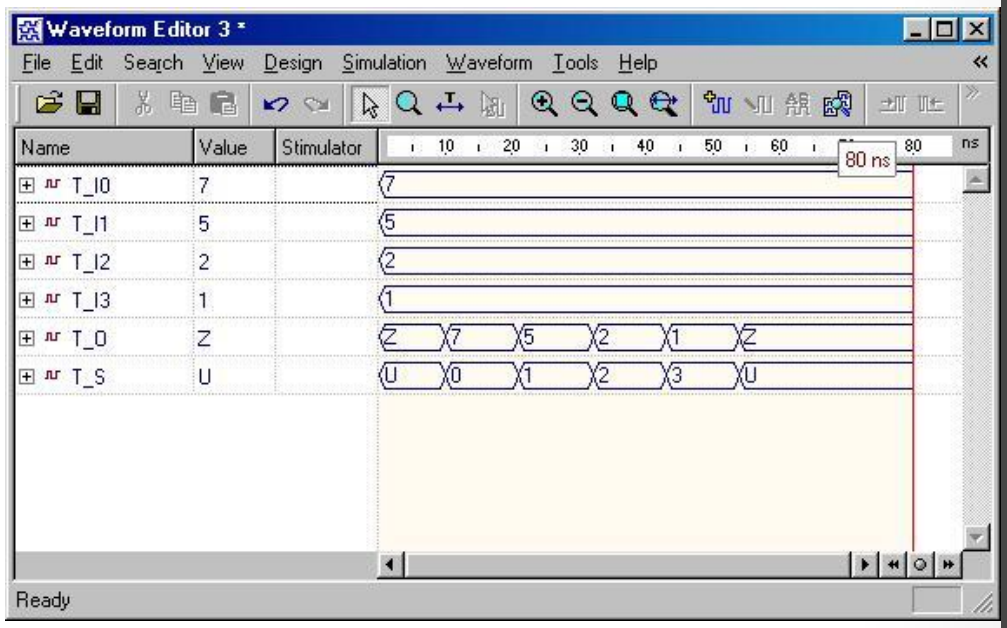
4x1 Multiplexer

```
-----------------------------------------------
-- 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;
-------------------------------------------------
```

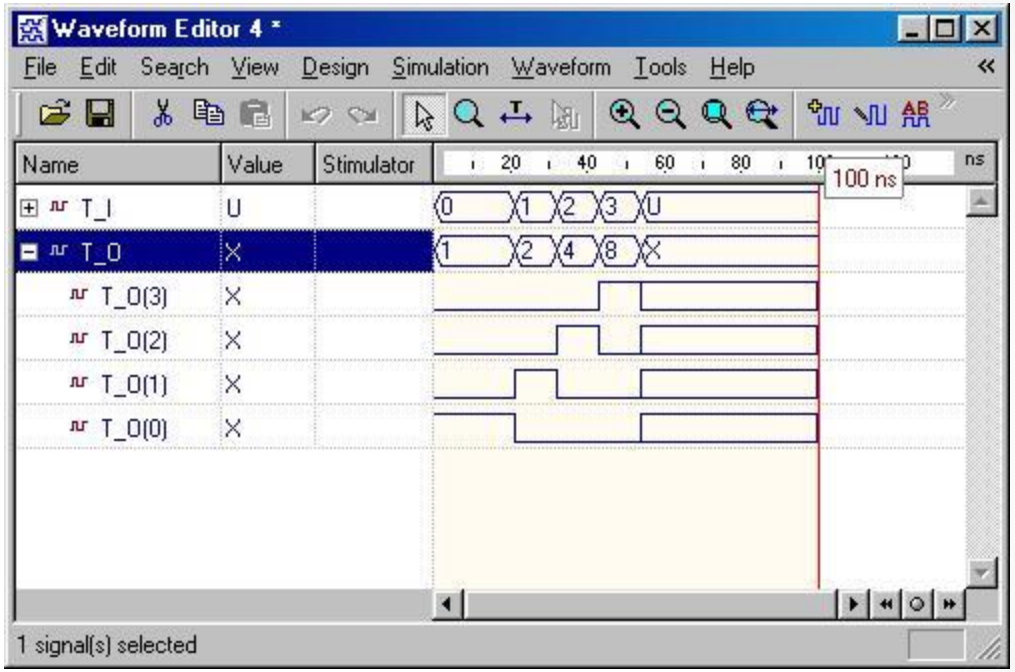
Simulation Waveforms



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;
-----------------------------------------------
```



Test Bench of the multiplexer example

```
--------------------------------------------------------------------
-- Test Bench for Multiplexer (ESD figure 2.5)-- by Weijun Zhang, 04/2001
-- four operations are tested in this example.
--------------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

entity Mux_TB is                    -- empty entity
end Mux_TB;
--------------------------------------------------------------------

architecture TB of Mux_TB is
   -- initialize the declared signals
   signal T_I3: std_logic_vector(2 downto 0):="000";
   signal T_I2: std_logic_vector(2 downto 0):="000";
   signal T_I1: std_logic_vector(2 downto 0):="000";
   signal T_I0: std_logic_vector(2 downto 0):="000";
   signal T_O: std_logic_vector(2 downto 0);
   signal T_S: std_logic_vector(1 downto 0);

   component Mux
   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 component;
```

Steps of a testbench:

- entity declaration for your testbench.
- Component Declaration for the Unit Under Test (UUT)
- declare inputs and initialize them
- declare outputs and initialize them
- Clock period definitions
- Stimulus process

Test Bench of the multiplexer example..

```
begin
   U_Mux: Mux port map (T_I3, T_I2, T_I1, T_I0, T_S, T_O);
   process

variable err_cnt: integer :=0;
   begin

            T_I3 <= "001";                    -- I0-I3 are different signals
            T_I2 <= "010";
            T_I1 <= "101";
            T_I0 <= "111";

            -- case select eqaul "00"
            wait for 10 ns;
            T_S <= "00";
            wait for 1 ns;
            assert (T_O="111") report "Error Case 0" severity error;
            if (T_O/="111") then
               err_cnt := err_cnt+1;
            end if;
            -- case select equal "01"
            wait for 10 ns;
            T_S <= "01";
            wait for 1 ns;
            assert (T_O="101") report "Error Case 1" severity error;
            if (T_O/="101") then
               err_cnt := err_cnt+1;
            end if;
```

Test Bench of the multiplexer example...

```
-- case select equal "10"
        wait for 10 ns;
        T_S <= "10";
        wait for 1 ns;
        assert (T_O="010") report "Error Case 2" severity error;
        if (T_O/="010") then
           err_cnt := err_cnt+1;
        end if;
        -- case select equal "11"
        wait for 10 ns;
        T_S <= "11";
        wait for 1 ns;
        assert (T_O="001") report "Error Case 3" severity error;

        if (T_O/="001") then
           err_cnt := err_cnt+1;
        end if;
        -- case equal "11"
        wait for 10 ns;
        T_S <= "UU";
```

```
-- summary of all the tests
                if (err_cnt=0) then
                   assert (false)
                   report "Testbench of Mux completed sucessfully!"
                   severity note;
                else
                   assert (true)
                   report "Something wrong, try again!"
                   severity error;
                end if;

                wait;

   end process;

end TB;

-----------------------------------------------------------------
configuration CFG_TB of Mux_TB is
                for TB
                end for;
end CFG_TB;
-----------------------------------------------------------------
```

- 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.*
- The lecture is available online at:
  - http://bu.edu.eg/staff/ahmad.elbanna-courses/12135
- For inquires, send to:
  - ahmad.elbanna@feng.bu.edu.eg