University Of Diyala
College Of Engineering
Computer Engineering Department

Lecture 3:
VHDL - Introduction

Digital system Design

Dr.Yasir Amer

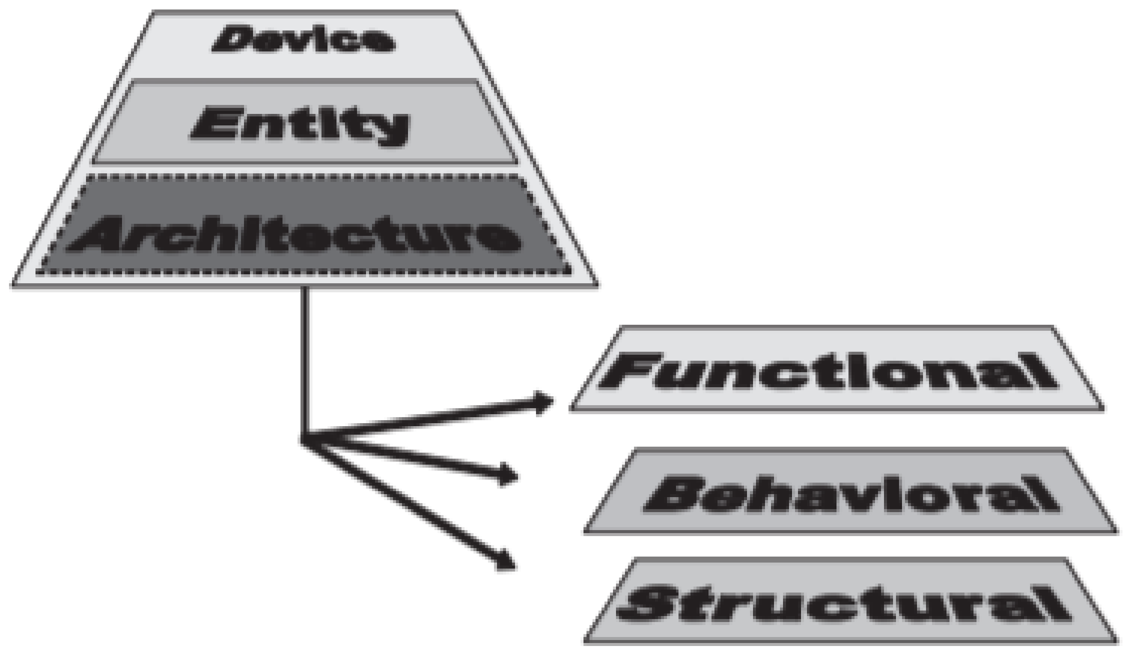
Introduction

- 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 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

VHDL Models with Different Architectures



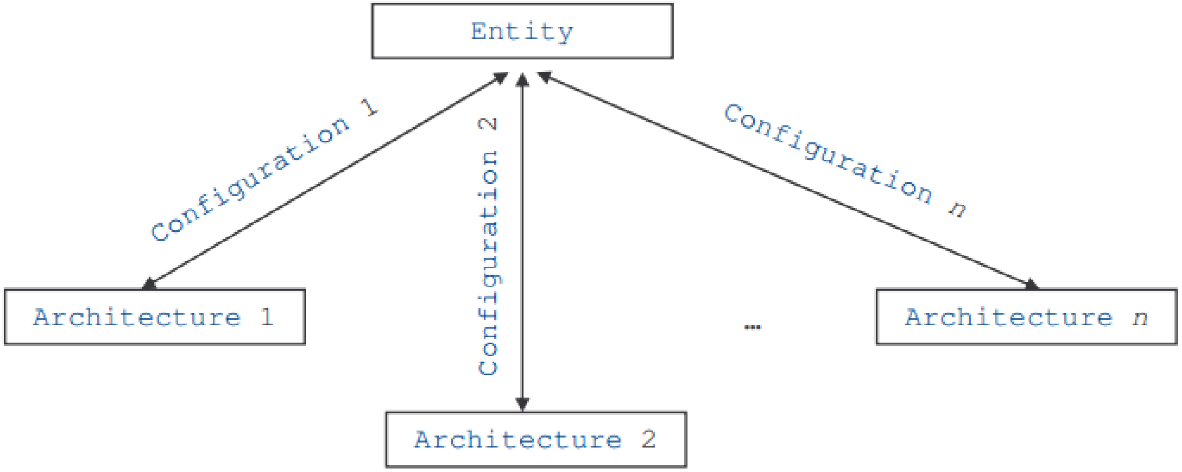
Libraries and Packages

- Libraries provide a set of packages, components, and functions that simplify the task of designing hardware
- Packages provide a collection of related data types and subprograms
- The following is an example of the use of the `ieee` library and its `std_logic_1164` package:

```
LIBRARY ieee;

USE ieee.std_logic_1164.ALL;
```

Entities, Architectures, and Configurations

- The structure of a VHDL design resembles the structure of a modern, object-oriented software design
- All VHDL designs provide an external interface and an internal implementation
- A VHDL design consists of entities, architectures, and configurations



Entities

- An entity is a specification of the design's external interface
- Entity declarations specify the following:
  1. The name of the entity
  2. A set of generic declarations specifying instance-specific parameters
  3. A set of port declarations defining the inputs and outputs of the hardware design
- Generic declarations and port declarations are optional

Entity Declarations

- Entity declarations are specified as follows:

```
ENTITY entity_name IS
    GENERIC(
            generic_1_name        : generic_1_type;
            generic_2_name        : generic_2_type;
            generic_n_name        : generic_n_type
    );
    PORT(
            port_1_name   : port_1_dir port_1_type;
            port_2_name   : port_2_dir port_2_type;
            port_n_name   : port_n_dir port_n_type
    );
END entity_name;
```

Example Entity Declaration

- The following is an example of an entity declaration for an AND gate:

```
ENTITY andgate IS
PORT (        a : IN        std_logic;
              b : IN        std_logic;
              c : OUT       std_logic );
END andgate;
```

NOTE:

In the PORT declaration, the semi-colon is used as a separator.

Ports

- Port name choices:
  - Consist of letters, digits, and/or underscores
  - Always begin with a letter
  - Case insensitive

- Port direction choices:

`IN`	Input port
`OUT`	Output port
`INOUT`	Bidirectional port
`BUFFER`	Buffered output port

NOTE:

A buffer is an output that can be "read" by the architecture of the entity.

Ports (cont.)

- IEEE standard 1164-1993 defines a package which provides a set of data types that are useful for logic synthesis
  - The external pins of a synthesizable design must use data types specified in the `std_logic_1164` package
  - IEEE recommends the use of the following data types to represent signals in a synthesizable system:

```
std_logic

std_logic_vector(<max> DOWNTO <min>)
```

Architectures

- An architecture is a specification of the design's internal implementation
- Multiple architectures can be created for a particular entity
- For example, you might wish to create several architectures for a particular entity with each architecture optimized with respect to a design goal:
  - Performance
  - Area
  - Power Consumption
  - Ease of Simulation

Architecture Declarations

- Architecture declarations are specified as follows:

```
ARCHITECTURE architecture_name OF entity_name IS
BEGIN
    -- Insert VHDL statements to assign outputs to
    -- each of the output signals defined in the
    -- entity declaration.
END architecture_name;
```

Example Architecture Declaration

- The following is an example of an architecture declaration for an AND gate:

```
ARCHITECTURE synthesis1 OF andgate IS
BEGIN
   c <= a AND b;
END synthesis1;
```

NOTE:

The keyword AND denotes the use of an AND gate.

Built-In Data Types

- VHDL supports a rich set of built-in data types as well as user-defined data types

Data Type	Characteristics
BIT	Binary, Unresolved
BIT_VECTOR	Binary, Unresolved, Array
INTEGER	Binary, Unresolved, Array
REAL	Floating Point

- Built-in data types work well for simulation but not so well for synthesis

- Built-in data types are suitable for use inside an architecture but should not be used for external pins

Logical Operators

- VHDL supports the following logical operators:

AND	NAND	NOT
OR	NOR	
XOR	XNOR	

- VHDL also supports the overloading of existing operators and the creation of new operators using functions

Other Operators

- VHDL supports the following relational operators:

  = (Equal)

  /= (Not Equal)

  < (Less Than)

  > (Greater Than)

- VHDL supports the following mathematical operators:

  + (Addition)

  - (Subtraction)

  * (Multiplication)

  / (Division)

Process Statements

- The keywords used for conditional assignments and selected assignments differ from those used within a process:

Outside Processes	Inside Processes
`WHEN..ELSE`	`IF..ELSIF..ELSE..END IF`
`WITH..SELECT..WHEN`	`CASE..WHEN..END CASE`

- A selected assignment outside a process is functionally equivalent to a case statement within a process

- Processes can be used for combinational logic but most often, processes encapsulate sequential logic

Process Statements (cont.)

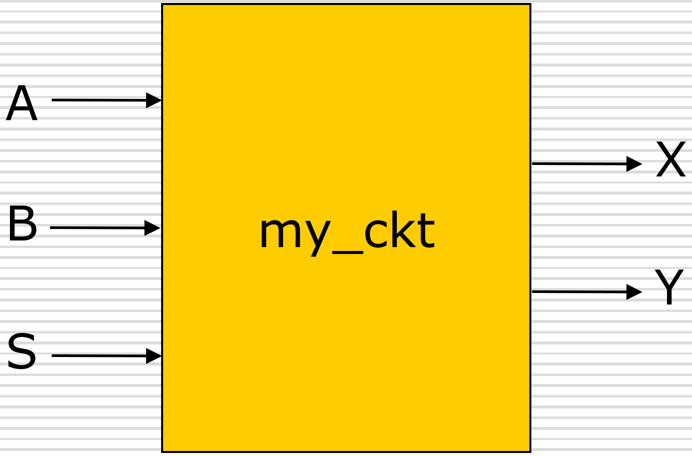
```
SIGNAL reset, clock, d, q          :std_logic;

PROCESS (reset, clock)
-- reset and clock are in the sensitivity list to
-- indicate that they are important inputs to the process
BEGIN
   -- IF keyword is only valid in a process
   IF (reset = '0') THEN
         q <= 0;
   ELSIF (clock'EVENT AND clock = '1') THEN
         q <= d;
   END IF;
END PROCESS;
```

The `EVENT` attribute is true if an edge has been detected on the corresponding signal.

NOTE:

This implements a D flip-flop with an asynchronous active-low reset signal.

Input-Output specification of circuit



- 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 ;
```

VHDL entity

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

Datatypes:
- In-built
- User-defined

name recommended
- Example:
  - Circuit name: my_ckt
  - Filename: my_ckt.vhd

S

Direction of port
3 main types:
- **in**: Input

Port names or Signal names

Note the absence of semicolon ";" at the end of the last signal and the presence at the end of the closing bracket

Built-in Datatypes

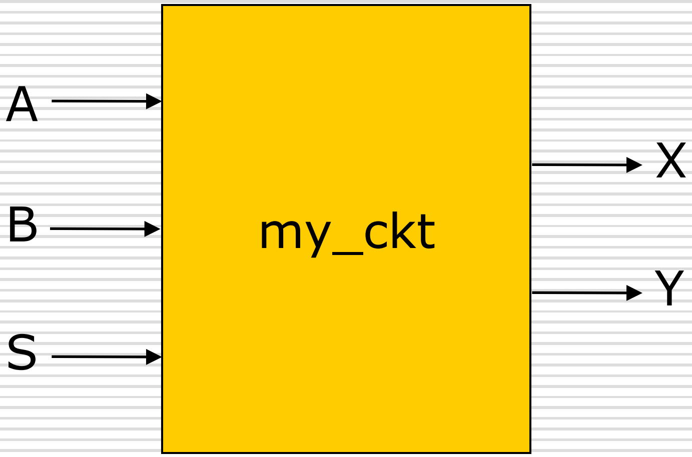
- Scalar (single valued) signal types:
  - **`bit`**
  - **`boolean`**
  - **`integer`**
  - Examples:
    - **`A: in bit;`**
    - **`G: out boolean;`**
    - **`K: out integer range -2**4 to 2**4-1;`**
- Aggregate (collection) signal types:
  - **`bit_vector`**: array of bits representing binary numbers
  - **`signed`**: array of bits representing signed binary numbers
  - Examples:
    - **`D: in bit_vector(0 to 7);`**
    - **`E: in bit_vector(7 downto 0);`**
    - **`M: in signed (4 downto 0);`**
      `--signed 5 bit_vector binary number`

User-defined datatype

- Construct datatypes arbitrarily or using built-in datatypes
- Examples:
  - `type temperature is (high, medium, low);`
  - `type byte is array(0 to 7) of bit;`

Functional specification

- Example:
  - 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')) th
         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 ;
```

Signals vs Variables

- Signals
  - Signals follow the notion of 'event scheduling'
  - An event is characterized by a (time,value) pair
  - Signal assignment example:

    `X <= Xtmp;` means

    Schedule the assignment of the value of signal **`Xtmp`** to signal **`X`** at (Current time + delta)

    where delta: infinitesimal time unit used by simulator for processing the signals

Signals vs Variables

- Variables
  - Variables do not have notion of ‘events’
  - Variables can be defined and used only inside the **process** block and some other special blocks.
  - Variable declaration and assignment example:

```
process (…)
variable K : bit;
begin
   …
   -- Assign the value of sig
immediately
   K := L;
   …
end process;
```

Variables can only be defined and used inside the **process** construct and can be defined only in this place

About VHDL

- VHDL is not case sensitive
- VHDL is a free form language. You can write the whole program on a single line.

```
-- This is a VHDL comment
entity my_exor is -- one more comment
begin
...
end my_exor;
```

my EXOR gate

```
-- This is my first VHDL program

library IEEE;
use IEEE.std_logic_1164.all;

entity my_exor is
port (ip1    : in  std_logic;
      ip2    : in  std_logic;
      op1    : out std_logic
     );
end my_exor;
```

entity declaration - describes the boundaries of the object.
It defines the names of the ports, their mode and their type.

my EXOR gate

```
library IEEE;
use IEEE.std_logic_1164.all;

entity my_exor is
port (ip1    : in   std_logic;
      ip2    : in   std_logic;
      op1    : out std_logic
     );
end my_exor;
```

entity - defines the interface.

Mode of the port :
Direction of flow.
It can be
in, out or inout

my EXOR gate

```
library IEEE;
use IEEE.std_logic_1164.all;

entity my_exor is
port (ip1   : in  std_logic;
      ip2   : in  std_logic;
      op1   : out std_logic
     );
end my_exor;
```

entity - defines the interface.

Mode of the port :
It can be
in, out or inout

std_logic is the type of the port.
Standard logic is defined by the standard IEEE 1164.
It is defined in the IEEE library.
Any node of type std_logic can take 9 different values.
‘0’ , ’1’ , ’H’ , ’L’ , ’Z’ ,
’U’ , ’X’ , ’W’ , ’-’

my EXOR gate

```
library IEEE;
use IEEE.std_logic_1164.all;

entity my_exor is
port (ip1   : in  std_logic;
      ip2   : in  std_logic;
      op1   : out std_logic
     );
end my_exor;
```

Library : Collection of design elements, type declarations, sub programs, etc.

my EXOR gate

```
library IEEE;
use IEEE.std_logic_1164.all;

entity my_exor is
port (ip1   : in  std_logic;
      ip2   : in  std_logic;
      op1   : out std_logic
     );
end my_exor;

architecture my_exor_beh of my_exor is
begin
  op1 <= (ip1 and (not ip2)) or
            (ip2 and (not ip1));
end my_exor_beh;
```

Library : Collection of design elements, type declarations,sub programs, etc.

entity - defines the interface.

std_logic is the type of the port
It is defined in the IEEE library.
Any node of type std_logic can take 9 different values.
'0' , '1' , 'H' , 'L' , 'Z' , 'U' , 'X' , 'W' , '-'

Mode of the port :
It can be
in, out or inout

The architecture describes the behaviour (function), interconnections and the relationship between different inputs and outputs of the entity.

my EXOR gate

```
library IEEE;
use IEEE.std_logic_1164.all;

entity my_exor is
port (ip1   : in  std_logic;
      ip2   : in  std_logic;
      op1   : out std_logic
     );
end my_exor;

architecture my_exor_beh of my_exor is
begin
  op1 <= (ip1 and (not ip2)) or
         (ip2 and (not ip1));
end my_exor_beh;

configuration my_exor_C of my_exor is
  for my_exor_beh
  end for;
end my_exor_C;
```

Library : Collection of design elements, type declarations, sub programs, etc.

entity - defines the interface.

std_logic is the type of the port
It is defined in the IEEE library.
Any node of type std_logic can take 9 different value.
'0' , '1' , 'H' , 'L' , 'Z' , 'U' , 'X' , 'W' , '-'

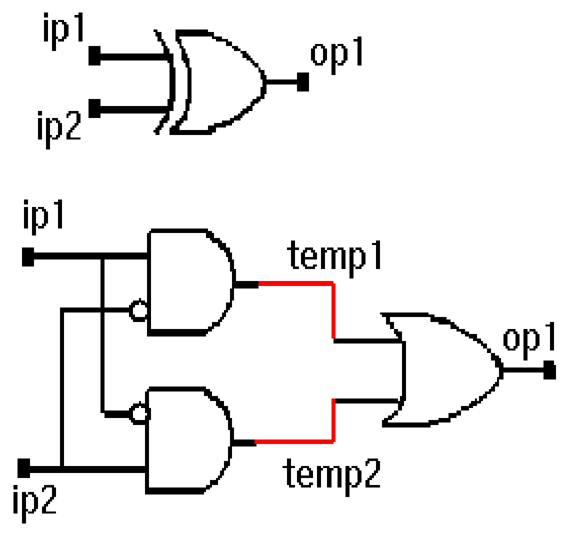
Mode of the port :
It can be
in, out or inout

The architecture describes the behaviour(function), interconnections and the relationship between different inputsand outputs.

The configuration is optional.
It defines the entity architecture bindings.
More about configurations later.

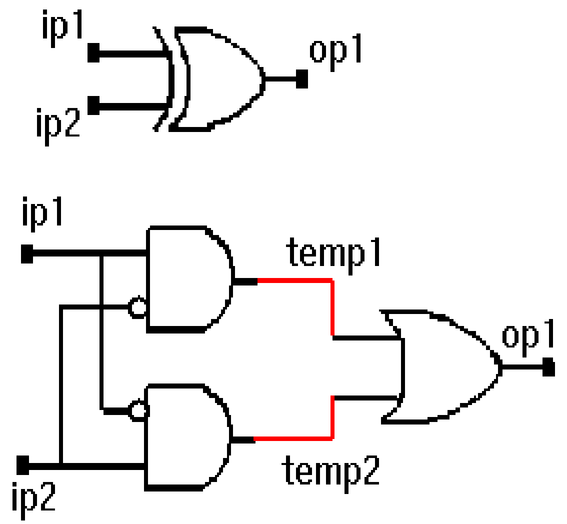
Internal connections are made using signals.
Signals are defined inside the architecture.

```
architecture my_exor_beh of my_exor is
  signal temp1 : std_logic;
  signal temp2 : std_logic;
begin
  ......
end my_exor_beh;
```



my EXOR with internal signals

```
library IEEE;
use IEEE.std_logic_1164.all;

entity my_exor is
port (ip1    : in  std_logic;
      ip2    : in  std_logic;
      op1    : out std_logic
     );
end my_exor;

architecture exor_w_sig of my_exor is
  signal temp1, temp2 : std_logic;
begin
  temp1 <= ip1 and (not ip2);
  temp2 <= ip2 and (not ip1);
  op1   <= temp1 or temp2;
end exor_w_sig;

configuration my_exor_C of my_exor is
  for exor_w_sig
  end for;
end my_exor_C;
```

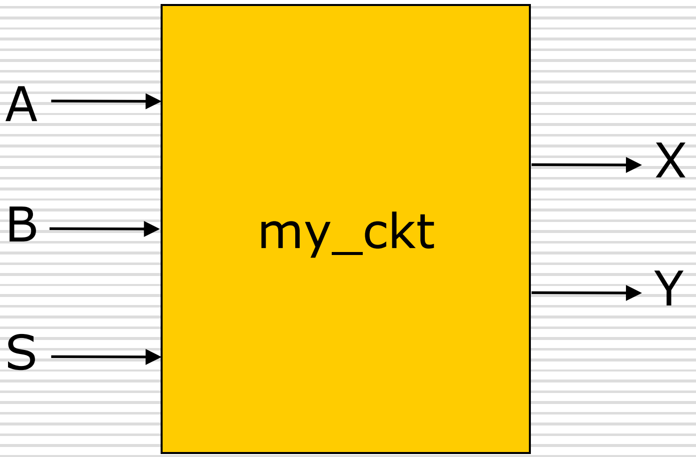


Simulation

- Simulation is modeling the output response of a circuit to given input stimuli

- For our example circuit:
  - Given the values of A, B and S
  - Determine the values of X and Y

- Many types of simulators used
  - Event driven simulator is used popularly
  - Simulation tool we shall use: ModelSim



Simulation

```
architecture behav_seq of my_ckt is
      signal Xtmp: bit;

      begin
       p1: process (A,B,S,Xtmp)
        variable XtmpVar: bit;
       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;
        XtmpVar := Xtmp;
       end process p1;
end;
```

Time 'T'	A	B	S	Xtmp	Y	XtmpVar	X
0-	U	U	U	'X'	'X'	'X'	'X'
0	0	1	0	'X'	'X'	'X'	'X'
0+d	0	1	0	0	0	0	'X'
0+2d	0	1	0	0	1	0	0
1	0	1	1	0	1	0	0
1+d	0	1	1	1	0	0	0
1+2d	0	1	1	1	0	0	1

Scheduled events executed:
Xtmp = 0
Y = 1
X = 0

Assignments executed:
XtmpVar = 0

Scheduled events list:
(empty)

d:

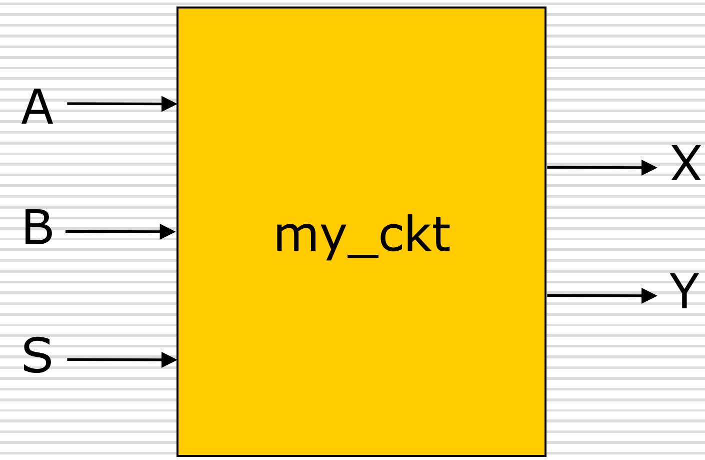
X = ('0',0+2d)

Synthesis

- Synthesis:
  Conversion of behavioral level description to structural level netlist
  - Abstract behavioral description maps to concrete logic-level implementation
  - For ex. Integers at behavioral level mapped to bits at structural level
- Structural level netlist
  - Implementation of behavioral description
  - Describes interconnection of gates
- Synthesis tool we shall use:
  Leonardo Spectrum

Structural level netlist

- Behavior of our example circuit:
  - 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



- Logic functions
  - Sbar = ~ S
  - Xbar = ~ X
  - X = A*(Sbar) + B*S
  - Y = (Xbar)*(Sbar)

Structural level netlist

```
architecture behav_conc of my_ckt is

    -- component declarations

    signal Sbar, Xbar, W1, W2: bit;

begin

    G1: not port map(Sbar,S);
    G2: and port map(W1,A,Sbar);
    G3: and port map(W2,B,S);
    G4: or  port map(X,W1,W2);

    G5: not port map(Xbar,X);
    G6: and port map(Y,Xbar,Sbar);

end ;
```

- Gate level VHDL descriptions (**and**, **or**, etc) are described separately
- Design in which other design descriptions are included is called a "hierarchical design"
- A VHDL design is included in current design using port map statement

Other VHDL resources

- Design Recipes for FPGAs
- VHDL Tutorial: Learn by Example *by Weijun Zhang*
  - http://esd.cs.ucr.edu/labs/tutorial/