# Components/Entities

- It is a method of describing entities that are used often.
- Components are like the sockets that will be connected on a breadboard.

- Entities are like the chips that go into the sockets.

- An entity is a real interface of the design with multiple architecture.

- Components have no real architecture parts and has only interface and connection.
- You can write your design without a component by direct instantiation as you put the chips directly on the breadboard and connect them.

# VDHL: Components Vs Entities

Reserved Words

```
component OR_3
port (A,B,C: in bit;
      Z: out bit);
end component ;
```

Reserved Words

- Declarations of Components and Entities are similar
- Components are virtual design entities

```
entity OR_3 is
port (A,B,C: in bit;
      Z: out bit);
end OR_3;
architecture MODEL of OR_3 is
begin
      Z <= A or B or C;
end MODEL;
```

# VDHL: Connection of Components

*Signal mapping or wiring:*
*By* ***Positional*** *Association Connectivity*

*Name of Component* *Reserved word*

```
--  Connection of Components
A1: AND_2 port map (A_IN, B_IN, TEMP1);
A2: AND_2 port map(A_IN, C_IN, TEMP2);
A3: AND_2 port map(B_IN, C_IN, TEMP3);
O1: OR_3  port map(TEMP1, TEMP2,
                   TEMP3, COUT);
```

*Label (identifier)*

**Component Instantiation Statements**

Every component instantiation statement creates an *instance* of a declared component

# VDHL: Connection of Components

```
--  Entity
--  Architecture
--  Declaration Part

begin
--  Connection of Components with Named Association
A1: AND_2 port map (A =>A_IN, B =>B_IN , Z =>TEMP1);
A2: AND_2 port map (A =>A_IN, Z =>TEMP2 , B =>C_IN);
A3: AND_2 port map (A =>B_IN, B =>C_IN, Z =>TEMP3);
O1: OR_3   port map (A =>TEMP1, B =>TEMP2, C =>TEMP3, Z=> C_out);
end structure;
```

***Signal Assignments***

***Position not Important***

***Name of the Architecture***

***Port map clause***

# Hierarchical Design with VHDL

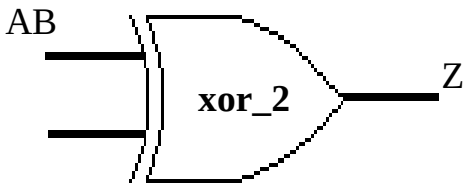
- **A hierarchical structure description is a powerful modeling construct in VHDL as it provides the mechanism to decompose the description of a large, complex digital system into smaller pieces.**

- **Structural hierarchies reflecting convenient functional & physical digital system decompositions is a good modeling practice.**

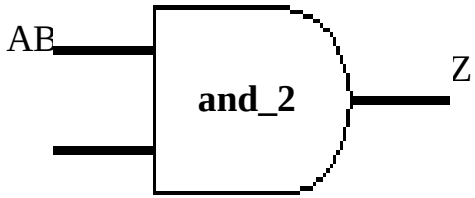
**Next few slides show how hierarchy is built in a Full Adder by using Half Adders and the Half Adder by using other components.**

# Hierarchical Design with VHDL

```
-- First Component
entity xor_2 is
port (A,B: in BIT; Z: out BIT);
end xor_2;
architecture  Data Flow of xor_2 is
begin
Z <= (not A and B) or (A and not(B) );
end Data Flow;
```

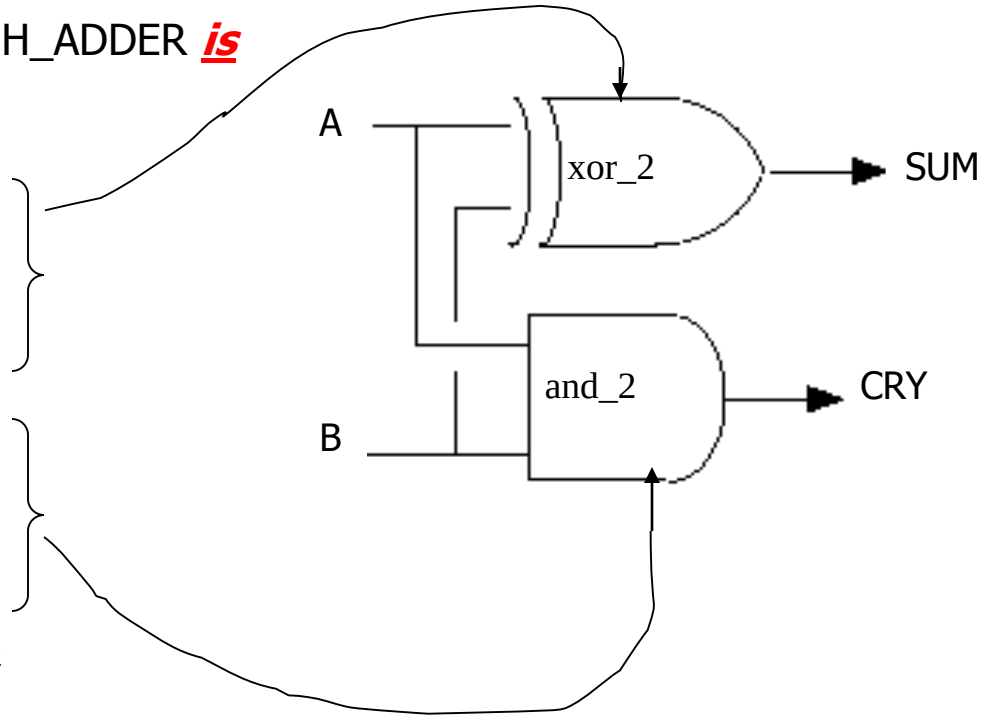



```
-- Second Component
entity and_2 is
port (A,B: in BIT; Z: out BIT);
end and_2;
architecture Data Flow of and_2 is
begin
Z <= A and B;
end Data Flow;
```



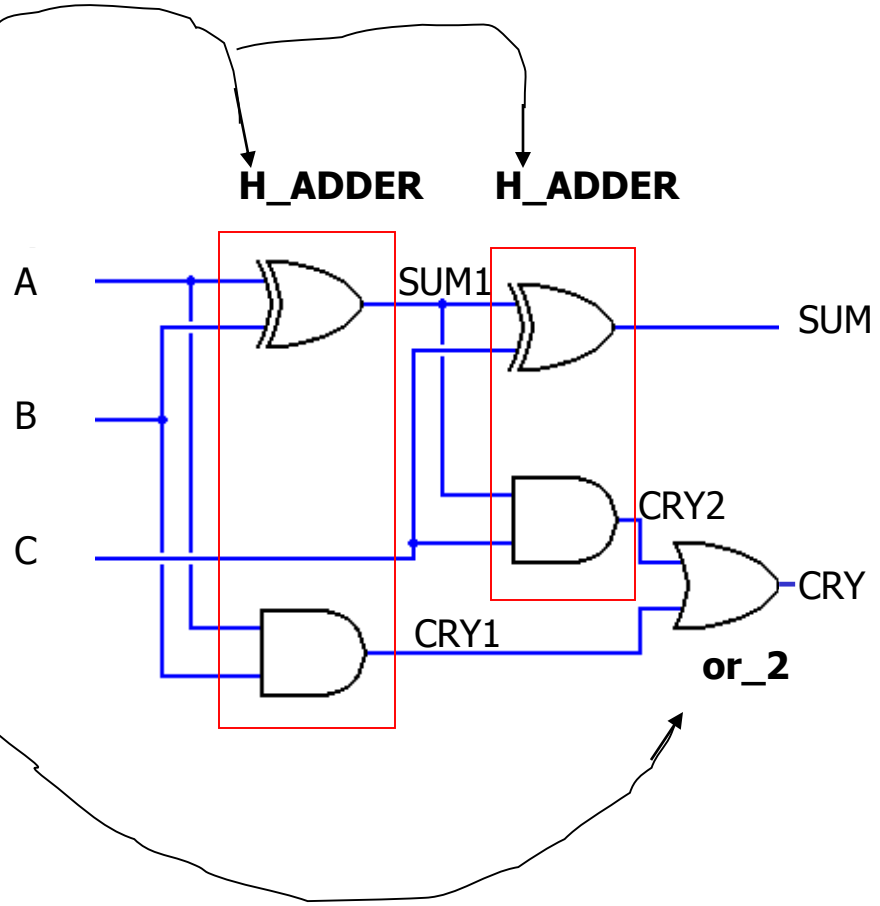
# Half Adder Design

```
-- Interface
entity H_ADDER is
port (A,B: in BIT; SUM, CRY: out BIT);
end H_ADDER;
-- Body
architecture  STRUCTURAL of  H_ADDER is

component xor_2
port (A,B : in BIT; Z: out BIT);
end component;

component and_2
port (A,B : in BIT; Z: out BIT);
end component;

begin
X1: xor_2 port map (A,B,SUM);
A1: and_2 port map (A,B, CRY);
end  STRUCTURAL;
```



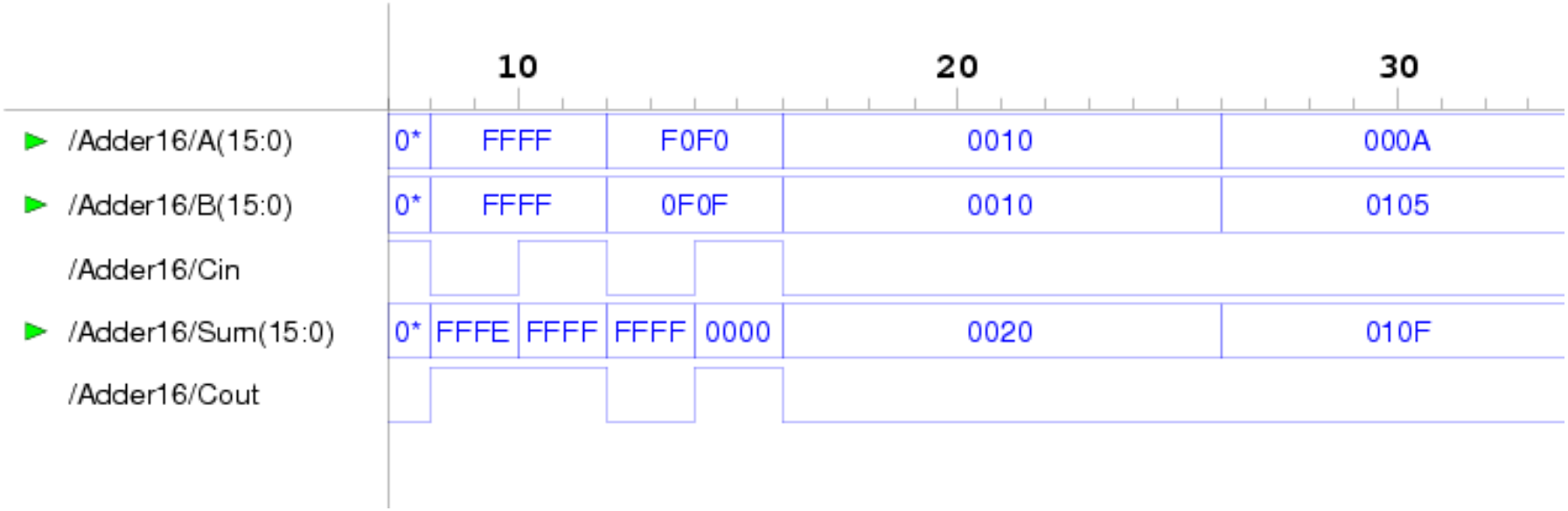
# Full Adder Design

```
-- Interface
entity FULL_ADDER is
port (A,B,C: in BIT; SUM, CRY: out BIT);
end FULL_ADDER;
-- Body
architecture STRUCTURAL of F_ADDER is

component H_Adder
port (X,Y : in BIT; Z1,Z2: out BIT);
end component;

component or_2
port (X,Y : in BIT; Z: out BIT);
end component;

signal SUM1, CRY1, CRY2: BIT;

begin
HA1: H_ADDER port map (A, B, SUM1, CRY1);
HA2: H_ADDER port map (SUM1, C, SUM, CRY2);
O_2: OR_2    port map (CRY1, CRY2, CRY);

end STRUCTURAL;
```



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

entity Adder16 is
        port (A, B: in std_logic_vector(15 downto 0);
        Cin: in std_logic; Cout: out std_logic;
        Sum: out std_logic_vector(15 downto 0));
end Adder16;
 architecture Ripple of Adder16 is
component Full_Adder
port (X, Y, Cin: in std_logic; Cout, Sum: out std_logic);
end component;
signal C: std_logic_vector(15 downto 0);
-- Before instantiating the components you must tell the VHDL compiler
--which components to use.  We use the for-use construct for this purpose

for FA0 : Full_Adder use entity WORK.Full_Adder(Concurrent);

begin
                FA0 : Full_Adder port map (A(0),B(0),Cin,C(0),Sum(0));

Stages: for i in 15 downto 1 generate
      B1 :block
                for FA : Full_Adder use entity WORK.Full_Adder(Concurrent);
                begin
      FA:  Full_Adder port map
                (A(i),B(i),C(i-1),C(i),Sum(i));
                end block ;
     end generate;
   Cout <= C(15);
end Ripple;
```

A = FFFF, B = FFFF, Cin =0 ➔ Sum = FFFE, Cout = 1;
A = FFFF, B = FFFF, Cin = 1 ➔ Sum = FFFF, Cout = 1;
A = 0F0F, B = F0F0, Cin = 0 ➔ Sum = FFFF, Cout = 0;
A = 0F0F, B = F0F0, Cin = 1 ➔ Sum = 0000, Cout = 1;




**Waveform 16-bit Adder**

# Packages

- In real life when building a pc board,usually the same components are often picked up from a storage area and placed on the board. VHDL structural description often use the same set of components. Repeating all the constructs to describe all the components is very tedious. The problem is solved by introducing  the package concept.

- A package serves as a central place for frequently used utilities, such as component declarations.

- The needed component declaration is only written ONCE in a package.

- The declaration may then be accessed by any VHDL model by simply accessing the package.

# Packages....

**Packages are the mechanism to share objects among different design units**

- **The *use* statement placed just before the architecture gives access to all the declarations in the package:**
  ***use* WORK.ASIM_LIB.*all***
  **The statement above gives access to all component declarations in ASIM_LIB located in library WORK.**

- **The *use* statement allows the package ASIM_LIB to export its declarations.**

- **A package declaration is a design unit and can be analyzed by itself.**

- **It is important to have standard naming convention for components, type and signal names. Standard naming conventions can be enforced by declaring the commonly used names within a package.**

# SOME OF THE IEEE STD PACKAGE

```
library IEEE;
        use IEEE.std_logic_1164.all;
        use IEEE.std_logic_textio.all;
        use IEEE.std_logic_arith.all;
        use IEEE.numeric_bit.all;
        use IEEE.numeric_std.all;
        use IEEE.std_logic_signed.all;
        use IEEE.std_logic_unsigned.all;
        use IEEE.math_real.all;
        use IEEE.math_complex.all;
```

The package textio provides user input/output

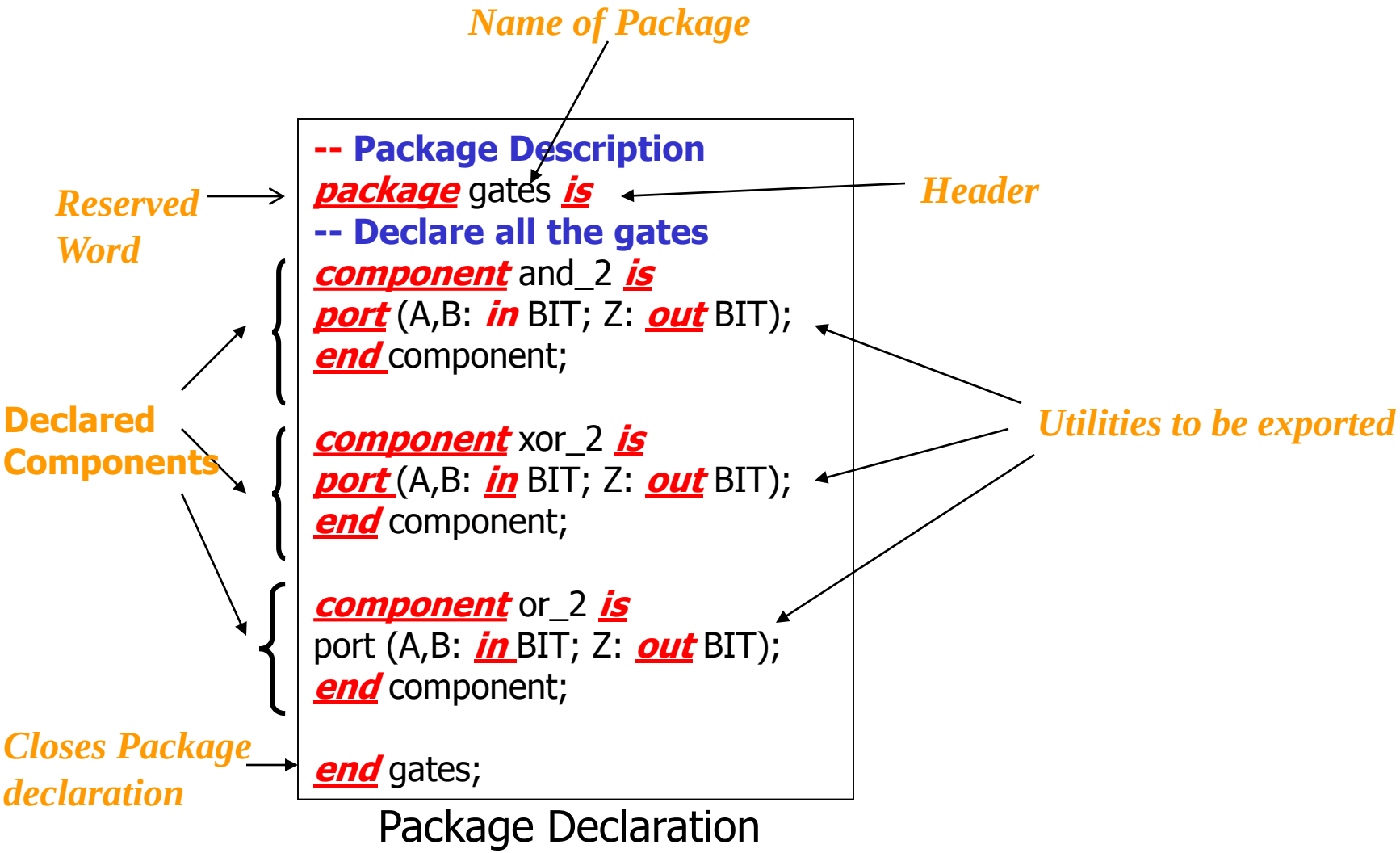
Types defined include: line, text, side, width

Functions defined include: readline, read, writeline write endline.

The package std_logic_arith provides numerical computation

The package std_logic_1164 provides enhanced signal types

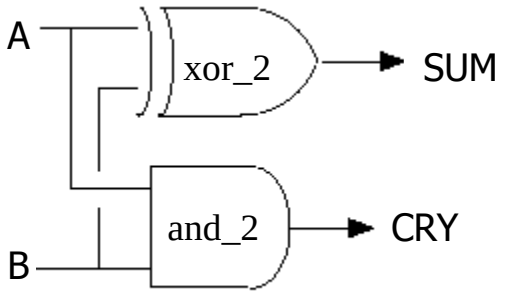
Types defined include: std_ulogic, std_ulogic_vector, std_logic, std_logic_vector




Package Declaration

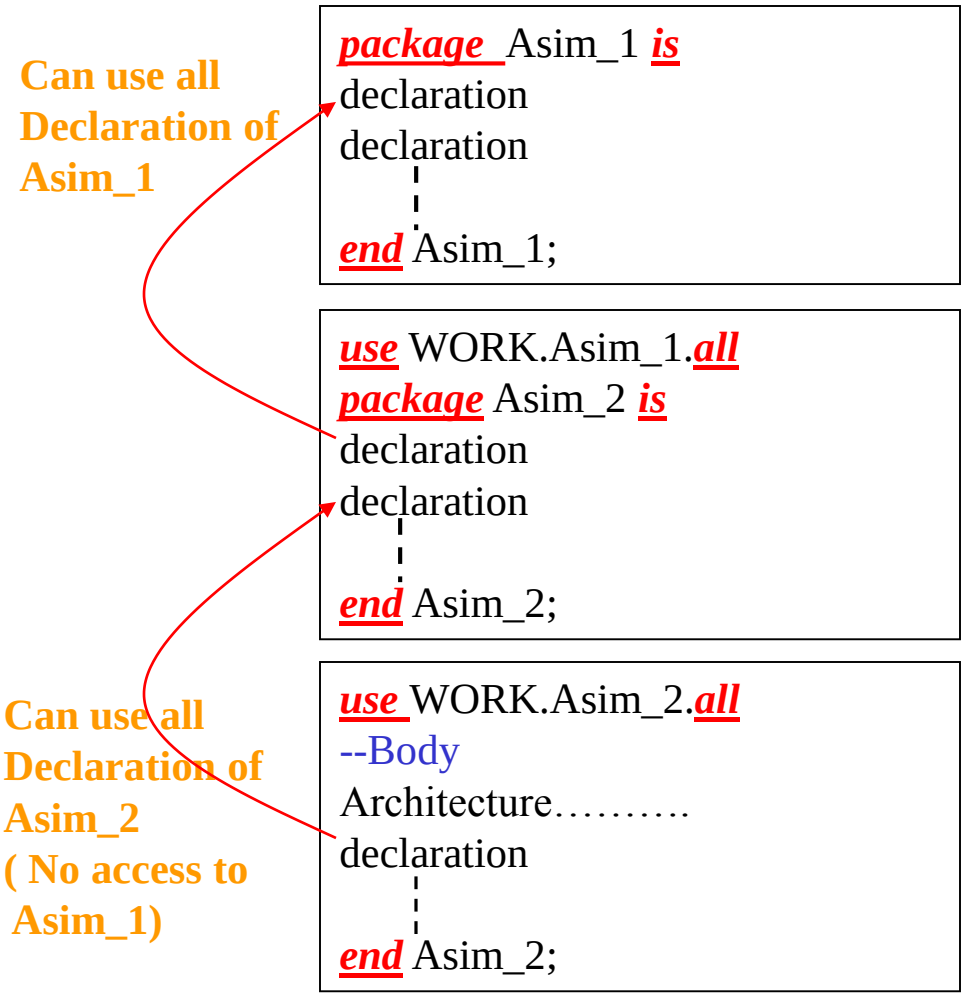
# Half Adder Design using packages

```
-- Interface
entity HALF_ADDER is
port (A,B: in BIT; SUM, CRY: out BIT);
end HALF_ADDER;

-- Body
-- Use components in Package gates
use WORK.gates.all ;

architecture structural of HALF_ADDER is
begin

X1: xor_2 port map (A,B,SUM);
A1: and_2 port map (A,B, CRY);

end structural;
```



# 2 - LEVEL PACKAGE HIERARCHY

A use clause can be placed before an entity declaration, giving the design entity and associated architecture access to the package contents.
A use clause can also be placed before a package declaration giving a package access to another package. Thus an hierarchy of packages is constructed in which the declarations of one package may be based upon declarations in other packages.



# Names and Scopes

- In VHDL everything must be declared before it can be used. A declaration defines what a name represents.

- The scoping rules define the name space:
  - Anything declared within the declaration part of an architecture may be used only within the architecture body.
  - Anything declared within a design entity declaration may be used only within the enclosing entity declaration and associated architecture.
  - Anything declared within a package declaration may be used within the enclosing package and also by the use statement in the other parts of VHDL

# Package: Selected Name Notation

```
-- You may select only an element Selected  of a Package

use WORK.gates.and_2;

Architecture structural of access is
  .
  .
  .
begin
A1: and_2 port map (.....);
          .....
          .....
OR1:
end access;
```

Gives acces only to and_2

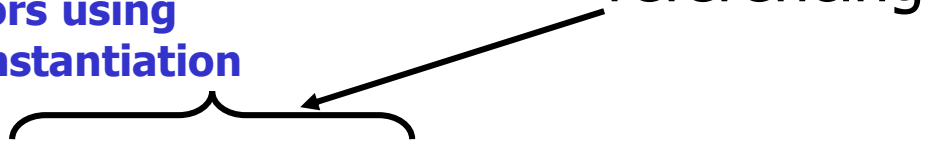
# Nested Scopes

```
-- Example of Nested Spaces
package signals is
signal      Z: bit := '0';
end signals;

use WORK.signals.all ;
entity example is
.....
.....
end example;

architecture structural of example is
signal   Z: BIT :='1';
.....
.....
end structural;
```

Z will take the value of 1

# Direct Design Entity Instantiation

```
-- Example
entity CARRY_GENERATE is
port (A_IN, B_IN, C_IN : in  BIT;  C_OUT : out  BIT);
end CARRY_GENERATE;

architecture structural of  CARRY_GENERATE is
signal temp1, temp2, temp3;
begin
-- Connect Logic Operators using
-- Direct Design Entity Instantiation

A1: entity WORK.LOGIC.and_2 port map (A_IN, B_IN, TEMP1);
A2: entity WORK.LOGIC.and_2 port map (A_IN, C_IN, TEMP2);
A3: entity WORK.LOGIC.and_2 port map (B_IN, C_IN, TEMP3);
O1: entity WORK.LOGIC.or_3  port map (TEMP1, TEMP2, TEMP3, C_OUT);

end structural;
```



# Styles of description

**Architecture**

- **Structural**

- **Behavioral**
  - **Data Flow**
  - **Algorithmic**

- **Mixed**

**We will use a full adder design to show the different architectural styles**

# Structural Modeling

**It IMPLICITLY defines the input/output functions by describing components and their interconnections. It treats the system to be described as a collection of gates and other components built on hierarchy that are interconnected to perform a certain function.**

**Structural modeling mimic actual hardware design, like a schematic diagram that has the components and their interconnections. It is by the use of defined components (cells or micros entities etc. ) over and over again and their interconnection.**

**All used component have to be defined earlier, usually in a package.**

**Structural modeling uses hierarchy to reduces modeling and the design complexity.**

**At the lowest hierarchy component are given in a behavioral model, using the basic logic operators such as *AND*, *OR* etc.**

**Within the architecture body declare:**

**All components to be used.**

**All signals that are used to interconnect the components.**

****Use labels for each instance of the component used for clear identification.**

# DATA FLOW Modeling

**This kind of modeling describes how data moves through the system.**
**The data flow model makes use of concurrent statements that are executed in parallel as soon as data arrives at the input.**
**With Concurrency , when a change occurs on the right hand side of any statement, all other statements that get affected are executed in the same time sample. This is the nature of the event driven simulation of VHDL. This is to say that, the order in which the statements are written does not matter and has no bearing on the execution of the statements. Concurrent statements are executed in parallel.**
**Concurrent design usually has no hierarchy and is a flat design.**
**Example:**

```
LEVEL1 : block
begin
Temp1 <=  A xor  not B after 2 ns;
Temp2 <=  B xor not A after 2 ns;
Temp3  <=  Temp1 or Temp2  after 5ns;
end
end Block LEVEL1;
```

# Behavioral Modeling

**It is the highest level of abstraction that describes a system in terms of what the system does, or how the output is related to the input signals.**
**Algorithmic architecture is composed of one or more concurrent processors. The statements inside each process execute sequentially .**
**It could be of many forms such as Boolean expression or Register Transfer etc.**
**example:**
**The house alarm will sound if the Alarm, is activated and one of the inside doors D1,D2 or D3 is opened.**

**Alarm sounds = Alarm_on *and* ( D1_open *or* D2_open *or* D3_open)**

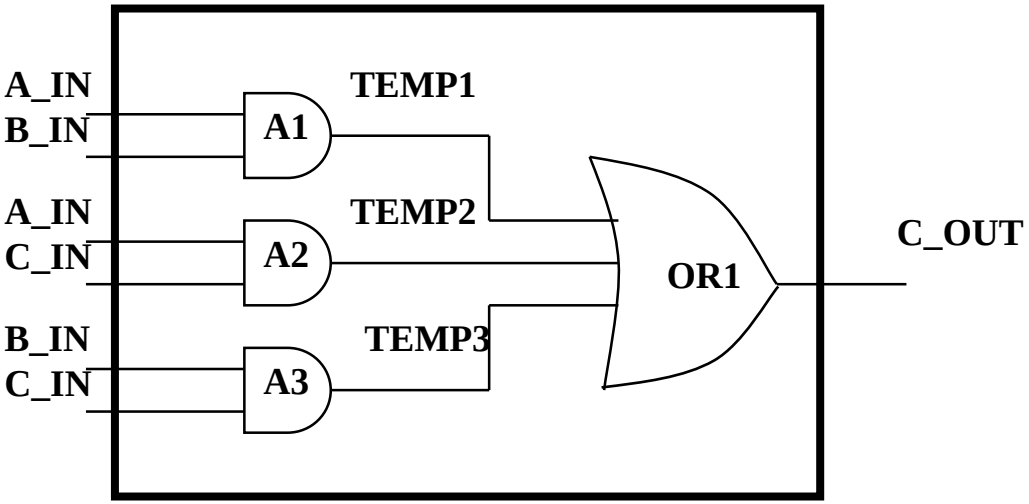
```
LEVEL1 : block
begin
Temp1 <=  A xor  not B after 2 ns;
Temp2 <=  B xor not A after 2 ns;
Temp3  <=  Temp1 or Temp2  after 5
ns;
end
end Block LEVEL1;
```

**Concurrent Constructs**

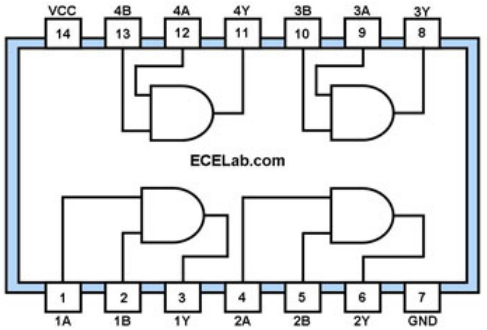
```
process (x)
 begin
 if a1 = 1 then a1 <= not x  after 5 ns;
 end process;
```

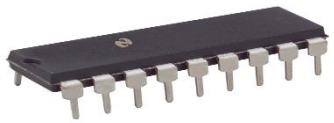
**Behavioral Construct**

## Schematic "Structural Constructs"




## Physical View



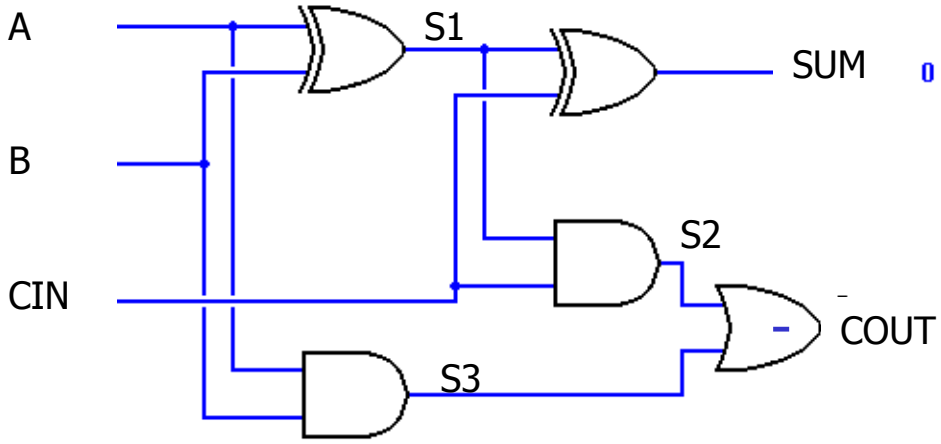
# Block

**The use of block statement is for organizational purpose only and it does not effect the simulation.**
**Each block must be assigned a label placed just before the block reserved word.**
**Example:**

```
LEVEL1 : block
begin
Temp1 <=  A xor  not B after 2 ns;
Temp2 <=  B xor not A after 2 ns;
Temp3  <=  Temp1 or Temp2  after 5 ns;
end
end Block LEVEL1;
```

# Gate Implementation - Structural

```
architecture GATE_IMPLEMENTATION of FULL_ADDER is
Block
component or_gate port (A,B : in BIT; C: out BIT);
end component;
component and_gate port (A,B : in BIT; C: out BIT);
end component;
component xor_gate port (A,B : in BIT; C: out BIT);
end component;
--  Local Signal Declaration
signal S1, S2, S3: BIT;

begin
X1: xor_gate port map (A, B, S1);
X2: xor_gate port map (S1, CIN, SUM);
A1: and_gate port map (CIN, S1, S2);
A2: and_2    port map (A, B, S3);
O1: or_gate  port map (S2, S3, COUT);
end Block;
end GATE_IMPLEMENTATION ;
```



# Data Flow Implementation

```
architecture  DATA_FLOW_IMPLEMENTATION  of FULL_ADDER is
block
signal  S1, S2, S3: BIT;

begin
        S1      <= A  xor  B;
        SUM     <= S1  xor  CIN;
        S2      <= A  and  B;
        S3      <= S1  and  CIN;
        COUT <= S2  or  S3;
end  block;

end  DATA_FLOW_IMPLEMENTATION;
```

# Algorithmic Implementation

```
architecture  ALGORITHMIC_IMPLEMENTATION of FULL_ADDER is
block
begin
        process (A,B,CIN)
        variable S: BIT_VECTOR ( 1 to 3 ) := A & B & CIN;
        variable COUNT: INTEGER range  0 to 3 :=0;
        begin
        for  i:= 1 to 3 loop
        if  S(i) = '1' then
        COUNT := COUNT +1;
        end if ;
        end loop;
        case COUNT is
        when 0 => COUT <= '0'; SUM <= '0';
        when 1 => COUT <= '0'; SUM <= '1';
        when 2 => COUT <= '1'; SUM <= '0';
        when 3 => COUT <= '1'; SUM <= '1';
end case;        end   process;
end block;       end  ALGORITHIC-IMPLEMENTATION;
```

# Functional Implementation (mixed)

```
architecture FUNCTIONAL_IMPLEMENTATION of FULL_ADDER is
use convert_pack.all      -- Contains type conversion Functions
block
port ( A,B,C: in INTEGER; S, CO: out INTEGER;
--  Type conversion between BIT and INTEGER is performed in ports
            port map ( X=> Bin_to_Int(A), Y=> Bin_to_Int(B), CIN=> Bin_to_Int(C),
                       INT_to_Bin (S) => SUM, INT_to_Bin (CO) => COUT);
            process (A,B,C)     -- Sensibility list of the process
            variable  TOTAL: INTEGER;

            begin
            TOTAL := A + B + C;
            S <= TOTAL mod 2;
            CO <= TOITAL / 2;
            end Process;
end Block;
end FUNCTIONAL_IMPLEMENTATION;
```

# Mixed Implementation

```
architecture MIXED_IMPLEMENTATION of FULL_ADDER is

 signal  WIRE: BIT;
component XOR_G
port ( X1, X2: in BIT;  XO1: out BIT);
end component;
for all: XOR_G use _ XOR_GATE(BEHAVIORAL);
                        -- Selection of Component Bodies
        begin

        XOR1: XOR_G port map (X,Y,WIRE);
        XOR2: XOR_G port map (WIRE,CIN,SUM);
        COUT <= (WIRE  and  CIN)  or  (X  and  Y);


        end  MIXED_IMPLEMENTATION ;
```

SRUCTURAL

Data Flow

- **THERE ARE VARIOUS SITES THAT YOU MAY TRY TO GET VHDl**
- **http://www.freedownloadscenter.com/Best/vhdl-tool-free.html**
- **http://www.csee.umbc.edu/help/VHDL/#free**
- **ActiveHDL**
  **http://www.aldec.com/products/active-hdl/**
  **Please visit this site for window based VHDL they have a demo that you can be downloaded The tool is called ActiveHDL.**
- **Xilinx:**
  **www.xilinx.com/ise/logic_design_prod/webpack.htm**
- **VHDL Simili**
  **http://www.symphonyeda.com/products.htm. There's a free version for students, but you can only simulate 10 waveforms at the same time. There is also a 30 day trial for the standard/professional edition which does not have this limit. It is very good and**
- **Aldec's Active-HDL EDA tool and free educational resources**
  **http://www.aldec.com/downloads**

# Environment set-up

## Please go through the Tutorial, it is more upto date

- Login to any Linux machine in ENCS ( H915)
- To write and debug and simulate VHDL/Verilog code you use the Modelsim software using the following commands:
  1. source /CMC/ENVIRONMENT/modelsim.env
  2. vsim

  ---

- To synthesis your RTL design FPGA_Advantage software using the following commands:
  1. Source /CMC/ENVIRRONMENT/fpga_advantage.env
  2. Precision
- One can login to any Linux server in encs from home ( please check with encs website or consult Ted to find out wbout the server you are eligible to remotely login)

  -in Linux OS use: ssh –Y login.encs.concordia.ca , then enter your encs username and password

# Environmental Set Up for Synopsys VHDL Analyzer/Simulator

- For information on how to set up your environment for the VHDL simulator please go to the following website:
- Remote login from windows OS:
  - Download putty or SSH secure client from internet
  - For graphical user interface you need to install a X-client software like Xming.
  - Follow the instruction in

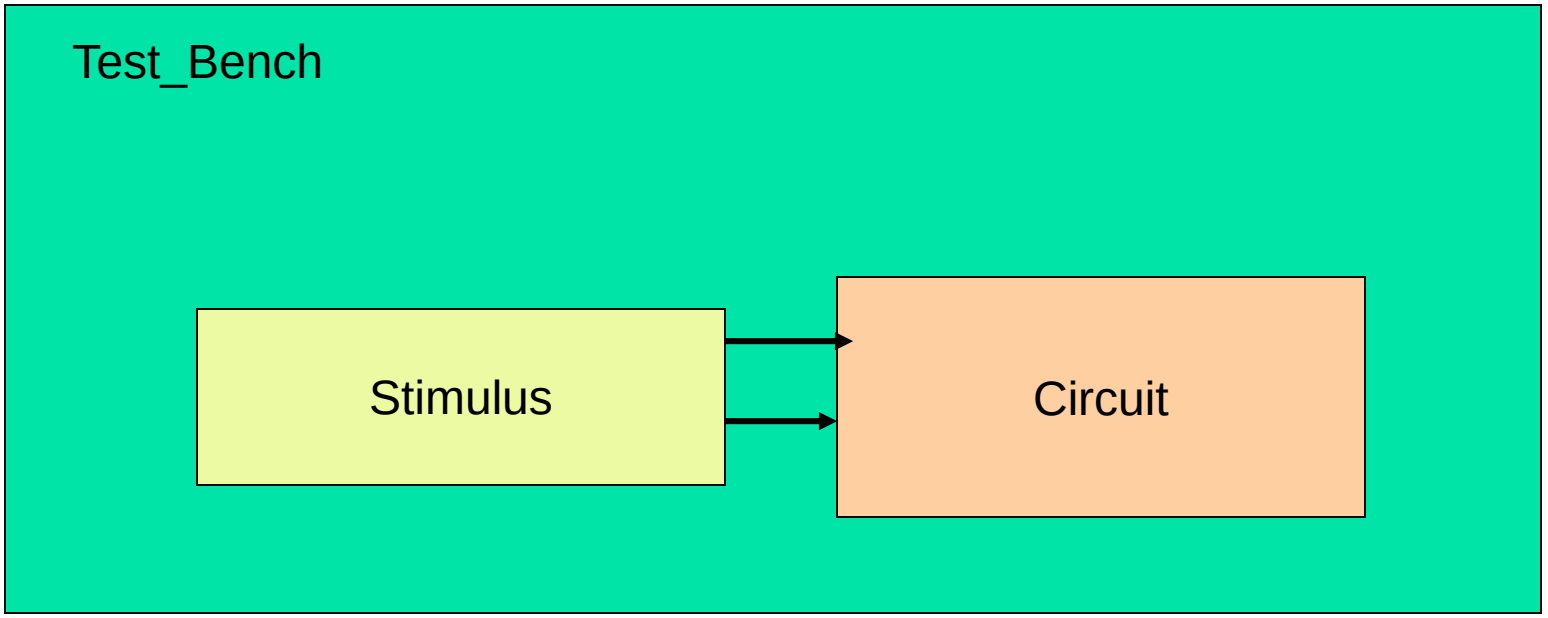
    http://www.encs.concordia.ca/helpdesk/howto/xserver.html

- Please go through the tutorials available in:

  http://www.encs.concordia.ca/helpdesk/resource/tutorial.html

# Test Bench

- To be able to test the circuit that you have designed then you have to apply some test vectors.
- This task is achieved by writing a test bench.



# Circuit : AND GATE

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity and_2 is
   Port ( a : in  bit;
          b : in  bit;
          c : out  bit);
end and_2;

architecture dataflow of and_2 is

begin

          c <= a and b;

end dataflow;
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity stim is
Port ( out_1 : out  bit; out_2 : out  bit);
end stim;
architecture Behavioral of stim is
signal a, b : bit:='0';
begin
process
begin
        --00
        out_2 <= '0';
        out_1 <= '0';
        wait for 10 ns;
        --01
        out_2 <= '1';
        out_1 <= '0';
        wait for 10 ns;
        --10
        out_2 <= '0';
        out_1 <= '1';
        wait for 10 ns;
        --11
        out_2 <= '1';
        out_1 <= '1';
        wait for 10 ns;
end process;      end Behavioral;
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity bench is
end bench;

architecture concurrent of bench is
component and_2 port (a, b : in bit; c : out bit);
end component;
component stim port (out_1, out_2 : out bit);
end component;

for  inst_stim:   stim      use entity WORK.stim(behavioral);
for  inst_and2:  and_2 use entity  WORK.and_2(dataflow);

signal x, y, z : bit:='0';
begin
inst_stim:    stim port map (x, y);
inst_and2:   and_2 port map (x, y, z);
end concurrent ;
```

# Simulation results of TestBench