

# LECTURE 6

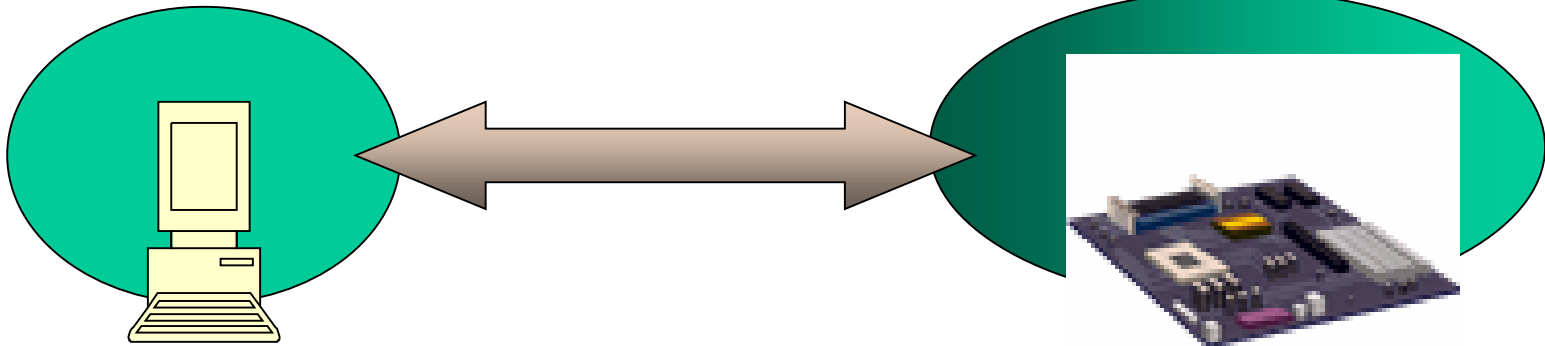
**In this lecture we will introduce:**

- **The VHDL Language and its benefits.**
- **The VHDL entity**
- **Concurrent and Sequential constructs**
- **Structural design.**
- **Hierarchy**
- **Packages**
- **Various architectures**
- **Examples**

# C-Based Hardware Design

PC with C++ Application

Implements Designs to Hardware



- Design and simulate at system level using C-based programming language such as Handel-C
- Need libraries that provide interface drivers including audio and video packages
- Need FPGA prototyping board, with variety of interfaces

- **Handel-C Language Reference Manual**

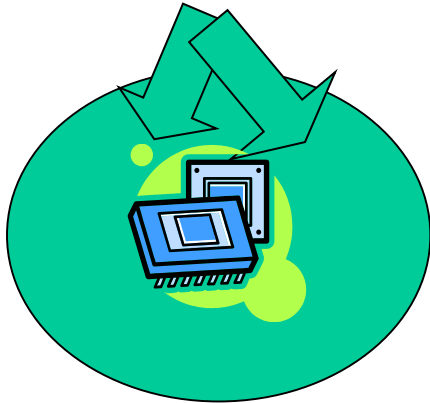
- [web.pa.msu.edu/hep/d0/l2/Handel-C/Handel C.PDF](http://web.pa.msu.edu/hep/d0/l2/Handel-C/Handel C.PDF)

- Chapter 9 is a reference for the complete Handel-C language syntax.

- **Overview 5 1.4 Basic Concepts.** This section deals with some of the basics behind the Handel-C

# Design Kit Output Targets

Target a variety  
of FPGA's



Or convert Handel-C to:

VHDL

Verilog

EDIF

System C

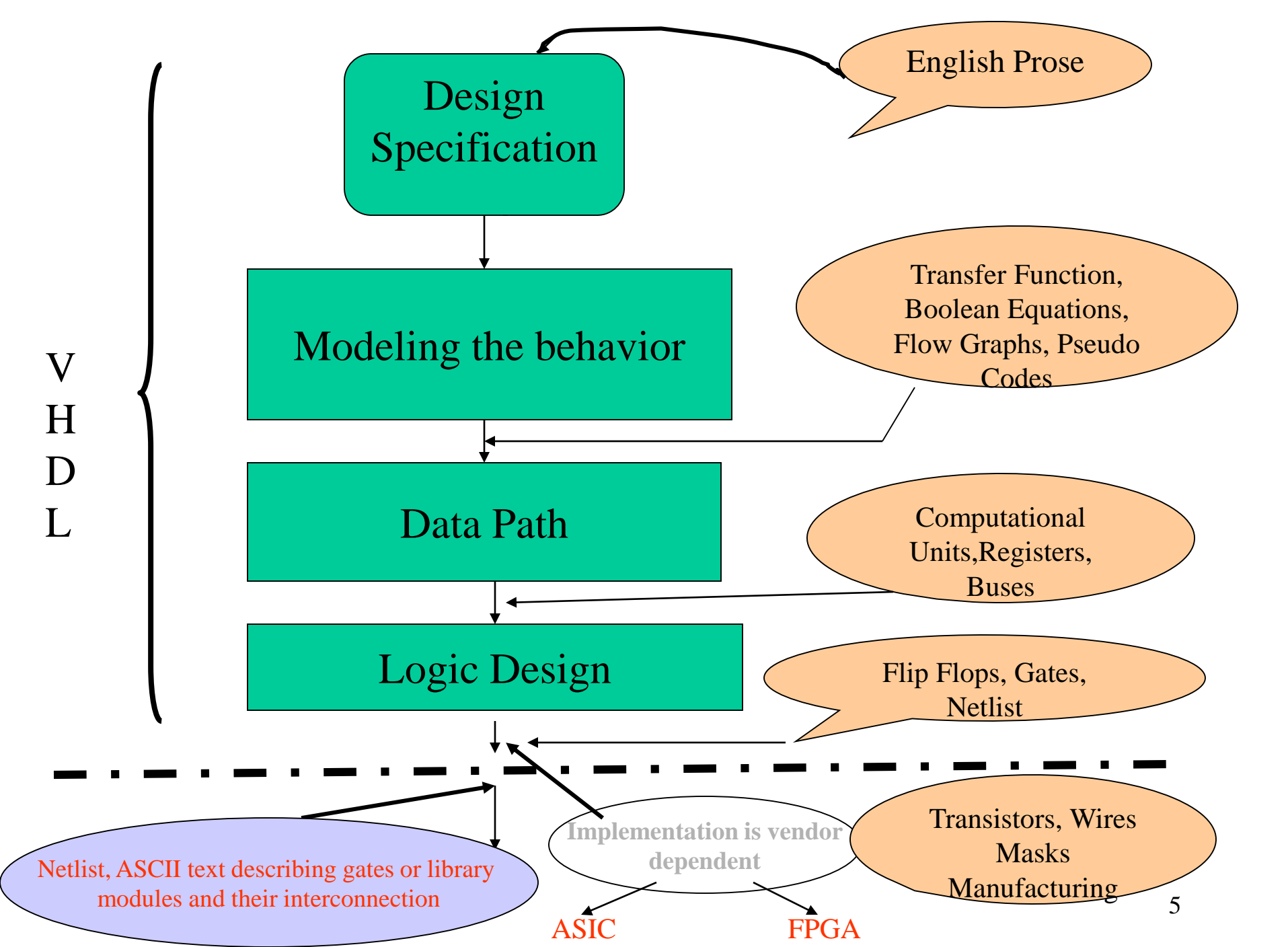
The output of the design kit can be down loaded to a variety of FPGAs or if you require some modification it can convert the Handel-C to other Forms such as VHDL....

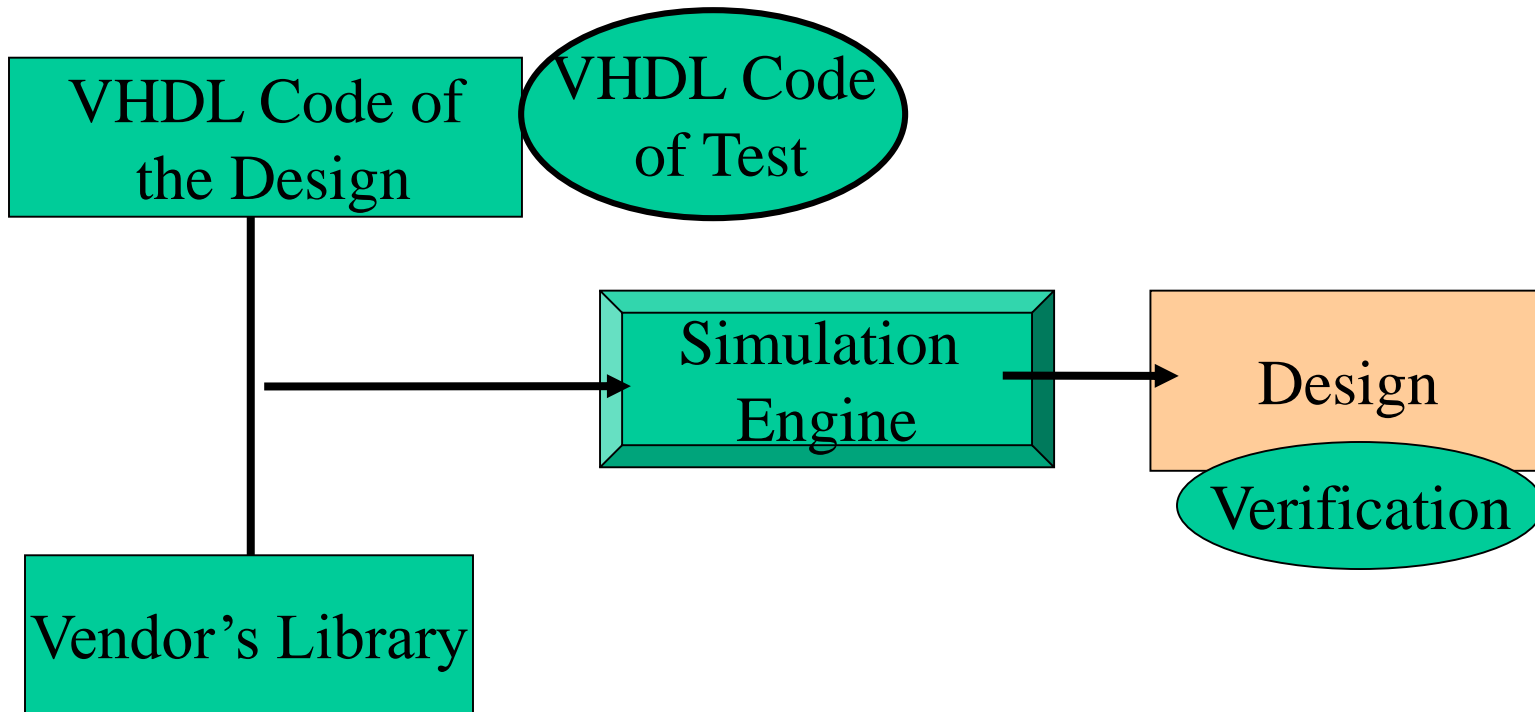
**C-to-hardware compiler (HLL synthesis), It exists but has many performance drawbacks**

**Yes, there are C to FPGA compilers. That is not a good way to go for the design presently**

# Mentor Graphics, Handel-C

- Handel-C Synthesis Methodology - Mentor Graphics  
[www.mentor.com/products/fpga/handel-c](http://www.mentor.com/products/fpga/handel-c)Cached
- DK Design Suite - Handel-C, synthesis with DK Design Suite offers a software flow for algorithm development, optimization and acceleration in embedded systems.

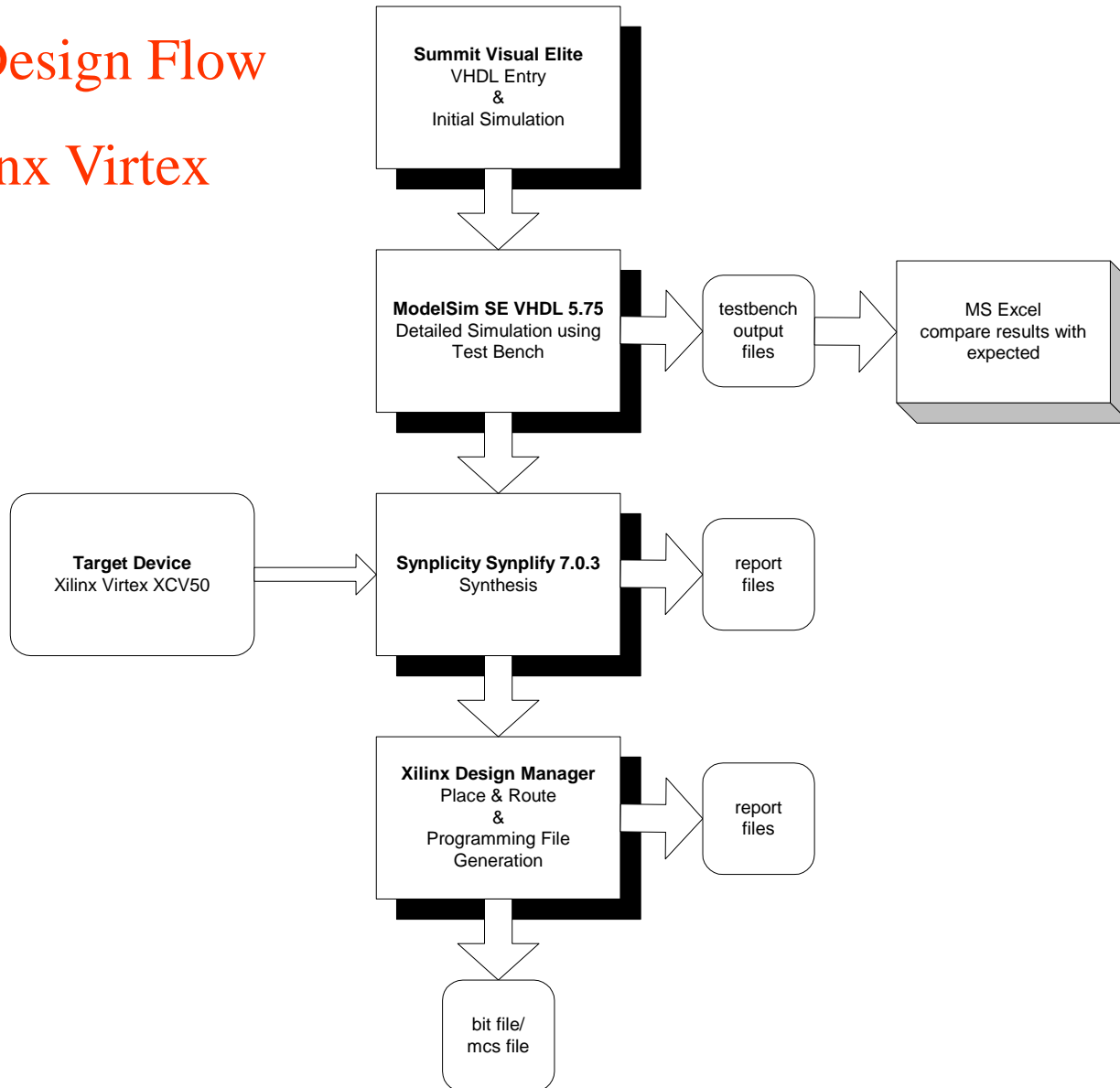




**Synthesis is the use of software packages to automatically verify and translate the VHDL code into a targeted device, using embedded optimising methods and meeting all the design constraints.**

# FPGA Design Flow

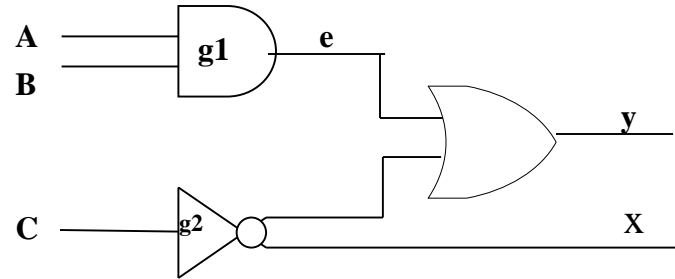
## For Xilinx Virtex XCV50



# Verilog Example

// Description of a simple circuit .

```
module circuit_1 (A,B, C, x,y);  
input A,B,C;  
wire e;  
output x,y;  
and g1(e,A,B);  
not g2 (x,C);  
or g3(y,x,e);  
endmodule;
```





## //CMOS inverter

**module** inverter (OUT, IN);

**input** IN;

**output** OUT;

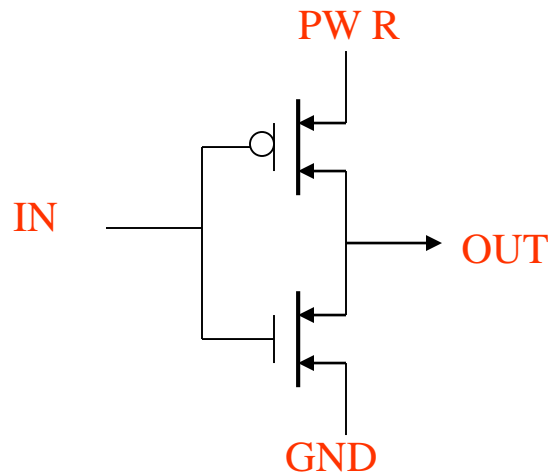
**supply1** PWR;

**supply0** GND;

**pmos** ( OUT, PWR, IN); // (Drain, Source, Gate)

**nmos** (OUT, GND, IN); // (Drain, Source, Gate)

**end module**

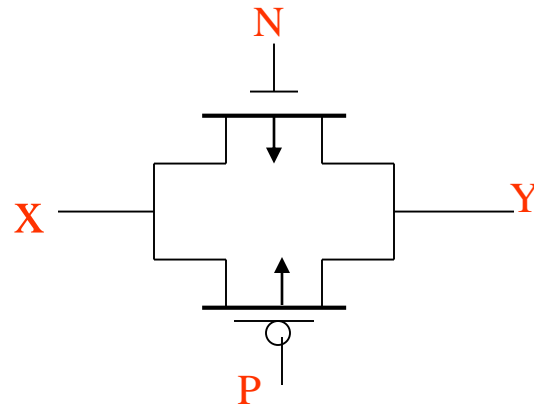


For transmission gate the keyword **cmos** is used.

`cmos (output, input, ncontrol, pcontrol); // general description.`

For example for the transmission gate shown in the Figure below

`cmos (Y,X,N,P);`



# The VHDL Language

- **Introduced in 1985, standardized in 1987 modified in 1993.**
- **It is used mainly as a specification and modeling language for digital systems .**
- **It is used as an intermediate form of design entry for many different tools**
- **It is a simulation and verification language.**
- **It is a test-synthesis language**

# The VHDL Language

- **VHDL is supported by DoD and most manufacturers.**
- **Technology Portable**
- **It is not yet standardized for synthesis.**
- **It has major application in Rapid prototyping**

# The VHDL Entity

- **General Components that performs specific function**
- **It can represent the whole system to be designed or its boards, chips, logic gates etc.**
- **It consists of 2 parts:**

**The interface**

**The Architecture**

# VHDL DESIGN UNITS

- **Entity Declaration**

Gives the interface view of the unit.

Implementation Independent

- **Architecture**

Describes the implementation(s) of the entity

- **Package Declaration**

Contains global information common to many design units.

- **Configuration**

Relates the design references to the designs saved in the library

# BASIC CONSTRUCT

-Interface

entity OR\_2 is

--Input/output ports

port

(A, B : in BIT;

Z : out BIT);

end OR\_2 ;

--Body

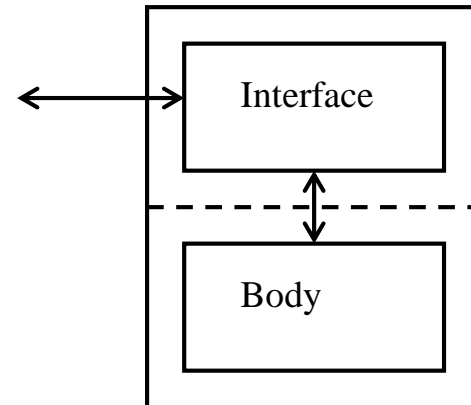
architecture DATA\_FLOW of OR\_2 is

begin

Z <= A or B; -- a construct statement implementing the OR gate

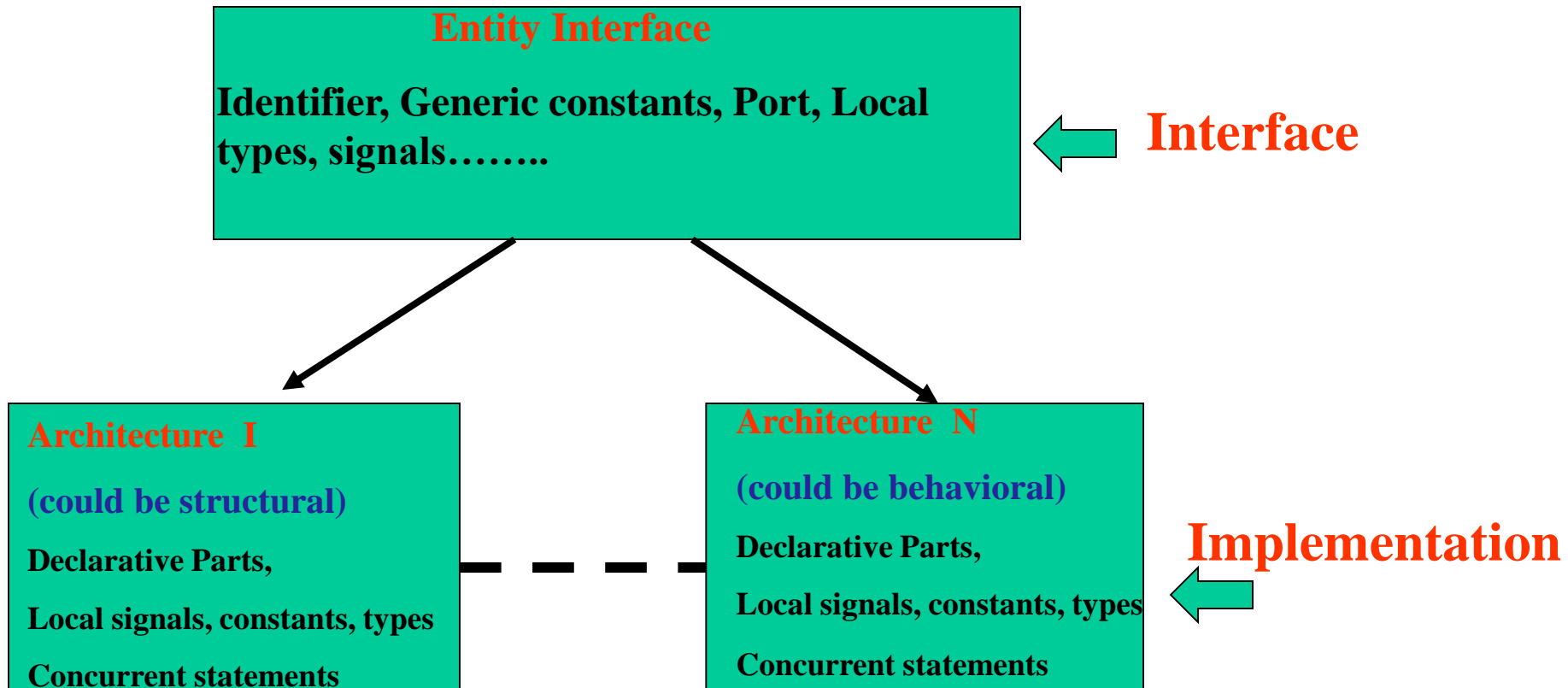
end DATA\_FLOW;

Interface is responsible for defining the black box's name, input and output



Body is responsible for describing the function that transforms the inputs to the outputs

# Entity Organization





## Difference Between two Architectures

Architecture DATA\_FLOW of  
Half\_Adder is  
Begin  
S<= A XOR B;  
end DATA\_FLOW;

Architecture Algorithmic of Half\_Adder is  
Process(A,B)  
begin  
if A=B then S=0; else  
                  S=1;  
end if;  
end process;  
end;

**Tutorials at**

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

# VHDL reserved keywords

- access
- after
- alias
- all
- and
- architecture
- array
- assert
- attribute
- begin
- abs
- block
- body
- buffer
- bus
- case
- component
- configuration
- constant
- disconnect
- downto
- else
- elsif
- end
- entity
- exit
- file
- for
- function
- generate
- generic
- guarded
- if
- in
- inout
- is
- label
- library
- linkage
- loop
- map
- mod
- new
- next
- nor
- not
- null
- of
- on
- open
- or
- others
- out
- package
- port
- procedure
- process
- range
- record
- register
- rem
- report
- return
- select
- severity
- signal
- subtype
- then
- to
- transport
- type
- units
- until
- use
- variable
- wait
- when
- with
- xor

# Additional reserved keywords in VHDL-93

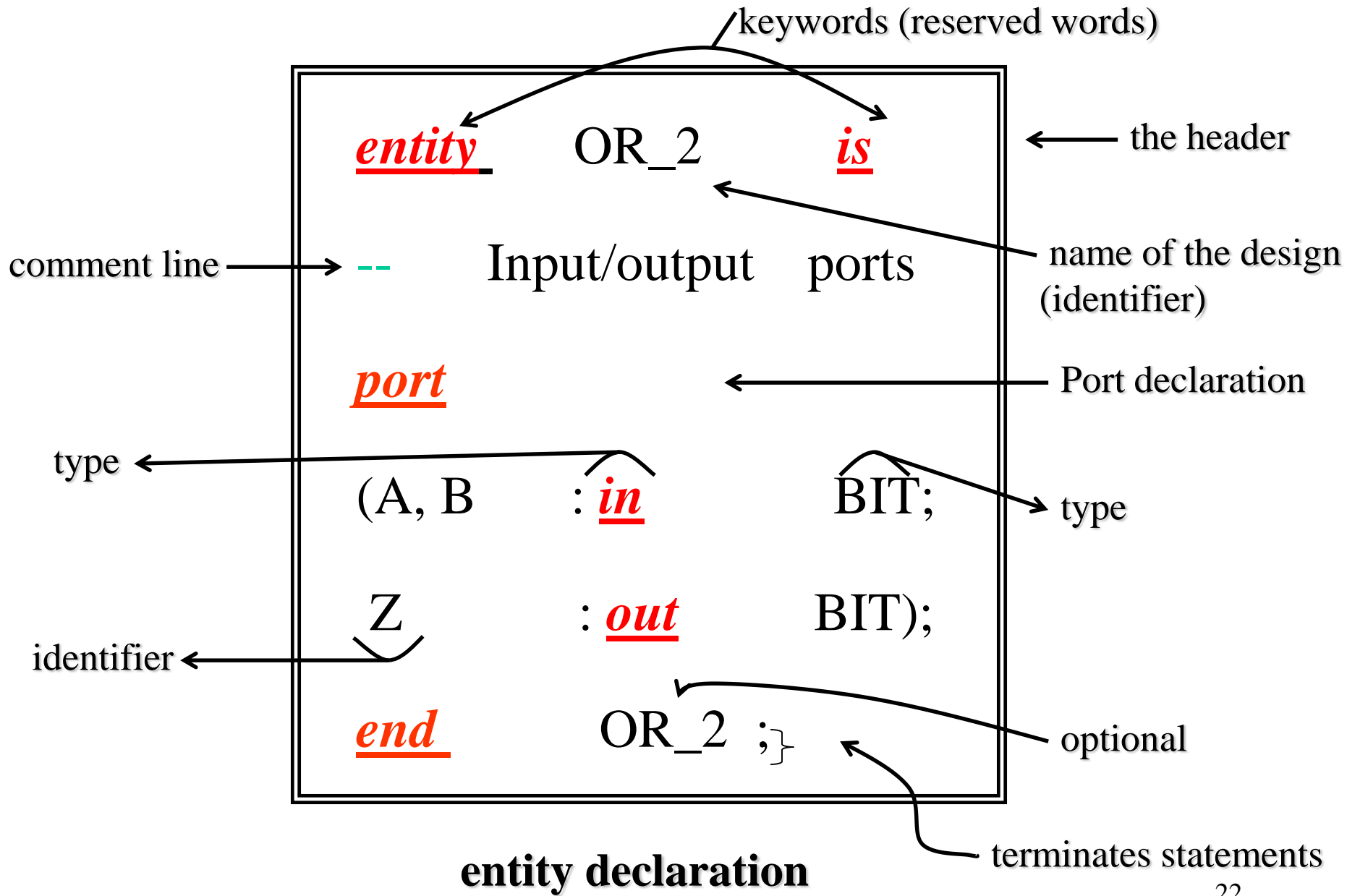
- impure
- group
- inertia
- postponed
- pure
- literal
- reject
- rol
- ror
- shared
- sla
- sl
- sra
- srl
- unaffected
- xnor

**\*\*ALL RESERVE WORDS ARE CASE INSENSITIVE\*\***

# --List of reserved operators

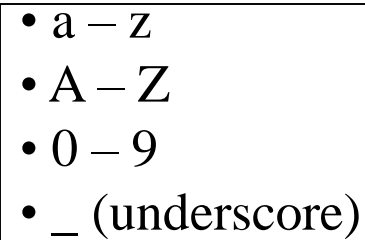
=	Equality operator
/=	Inequality operator
:=	The assignment operator for variables
<	The “less than” operator
<=	{ “less than or equal to” when used in an expression on scalar types & array The assignment operator
>	The “greater than” operator
>=	The “greater than or equal to” operator
+	The addition operator
-	The subtraction operator
*	The multiplication operator
/	The division operator
**	The exponentiation operator
&	The concatenation operator

# The Interface (connects the entity to its environment)



# Identifiers

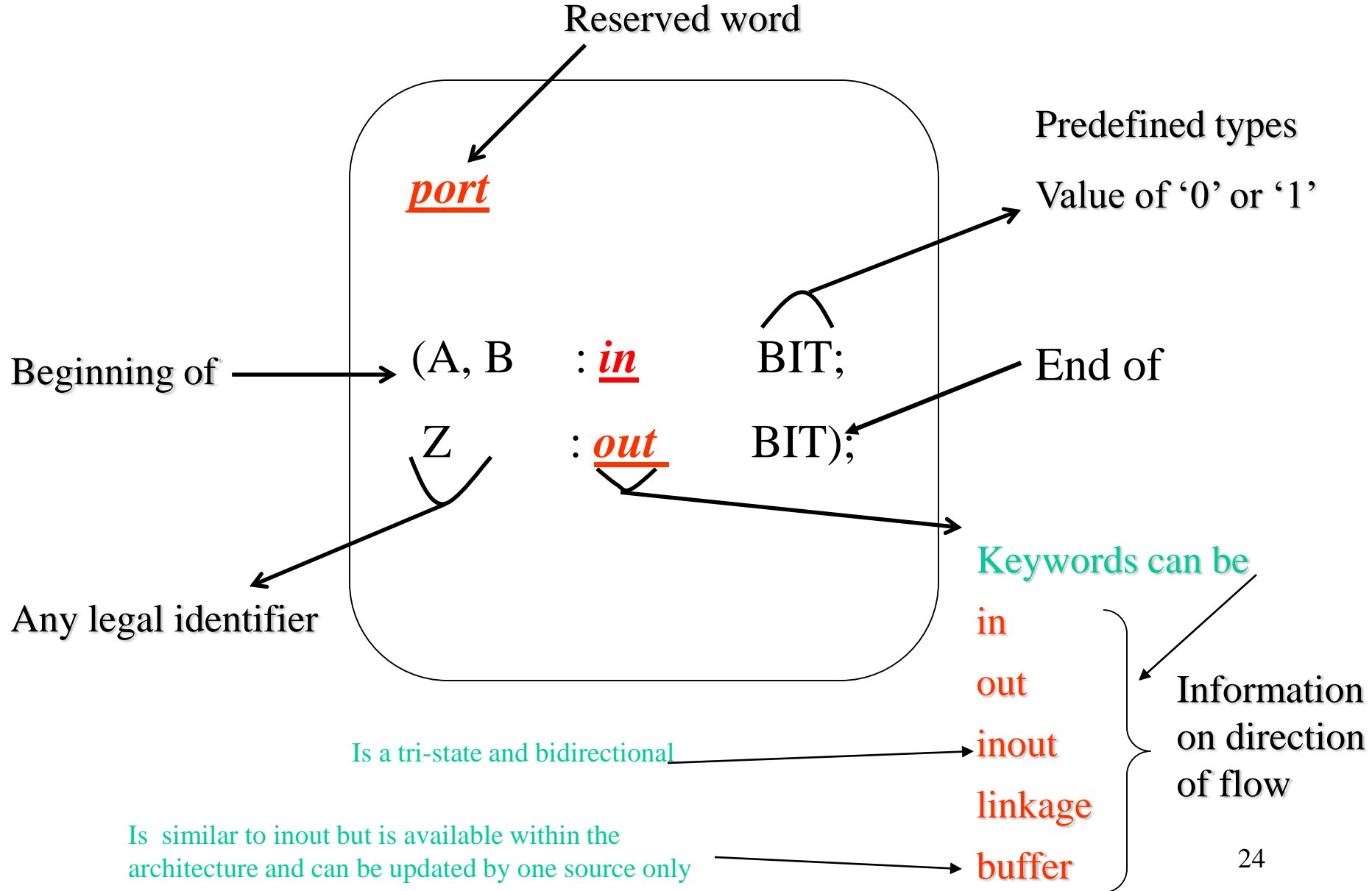
- Case insensitive
- Characters can only be:
- First character must be a letter
- Last character must not be an underscore
- No adjacent underscores

- 
- a – z
  - A – Z
  - 0 – 9
  - \_ (underscore)

# Extended identifiers

- Any length
- Must be delimited by \ \ leading & trailing backslashes
- All graphic characters
- Within backslashes any characters in any order can appear (exception is backslash which has to appear in pair)
- Is case sensitive
- An extended identifier is different from any keyword or basic identifier

# Port declaration (provides communication channels between the entity and its environment)





# --Fundamental Data Types

<b>Data Type</b>	<b>Values</b>	<b>Example</b>
Bit	'1', '0'	Q <= '1';
Bit_vector	(array of bits)	BYTE <= "00010101";
Boolean	True, False	flag <= True;
Integer	-20, 0, 1, 5090...	ACC <= ACC + 2;
Real	200.0, -2.0E2	C1 = V2 * 5.3;
Time	10 us, 7 ns, 150 ps	output <= '0' after 2 ns;
Character	'c', 'z', '5', '#', etc.	DataOut <= 'Y';
String	(Array of characters)	ADD <= "MEM" ;

# The Body defines input out put relations

Name of the architecture  
any legal identifier

Association of  
architecture

header

architecture DATA\_FLOW of OR\_2 is

header

begin

declaration  
part of objects to  
be used within the  
block

Z <= A or B ;

Statement  
Part

end DATA\_FLOW ;

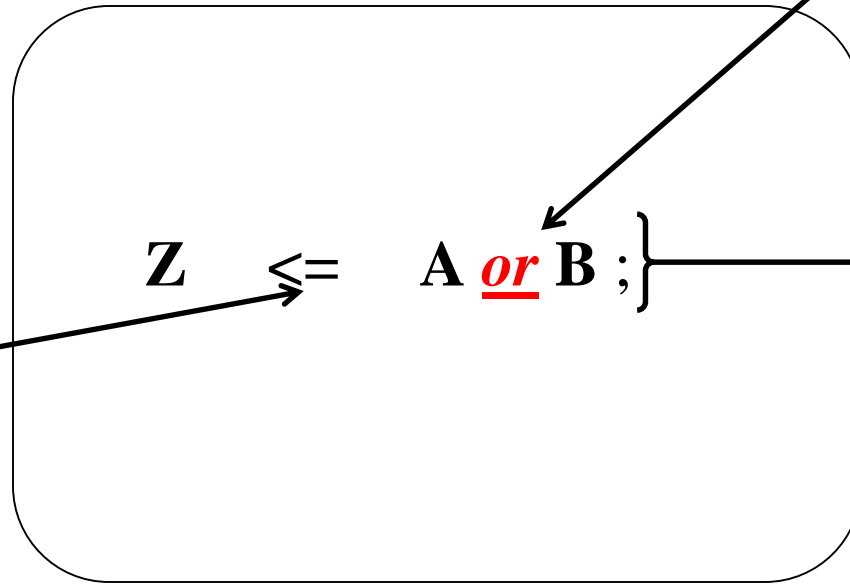
Closes  
architecture

Body declaration (architecture)

optional

# The operators

Logical operator



End of assignment

Assignment operator

Signal assignment statement

Other operators:

*and*

*or*

*xor*

*xnor*

*nand*

*nor*

*not*

\*\*\* “Anytime the input signal A and or B changes value the signal assignment statement executes and computes a new value for the output signal.” This is called “Signal Transformation.”

# Concurrency

```
-- Interface ..... 1
entity XOR_2 is ..... 2
Port ..... 3
(A,B : in BIT; Z : out BIT); ..... 4
end XOR_2; ..... 5
-- Body ..... ..6
architecture DATA_FLOW of XOR_2 is ...7
signal Sig 1, Sig 2: BIT; ..... 8
begin .....9
{ Sig 1 <= A and not B; ..... ..10
  Sig 2 <= B and not A; ..... ..11
  Z <= Sig1 or Sig 2; ..... ..12
}
end DATA_FLOW; .....13
```

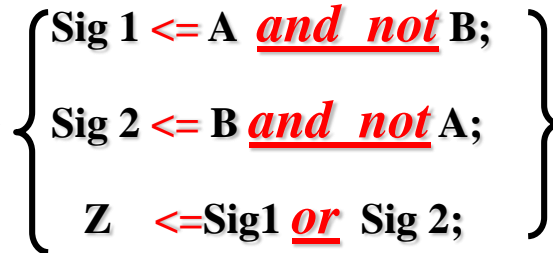
Reserved  
word



Signal  
Declaration



Concurrent  
assignment  
statement



# Modeling method

**Structural** (A description of the entity by components instantiation where the structure is explicit)  
such as gates and their interconnection

**Behavioral** { *Algorithmic* (A description of the entity by sequential statements representing behavior but no structural information)  
*Like adding two binary numbers 0001 + 1010*

*Data Flow* (A description of the entity by the use of concurrent statements to represent behavior implying structure)  
*Like logic equation  $Z = A \text{ xor } B$*

**Mixed** any mixture of behavioral and structural

## **Behavioral / Structural**

**Behavioral is easier to think about as it is similar to writing software code.**

**It is based on the functionality of the blocks**

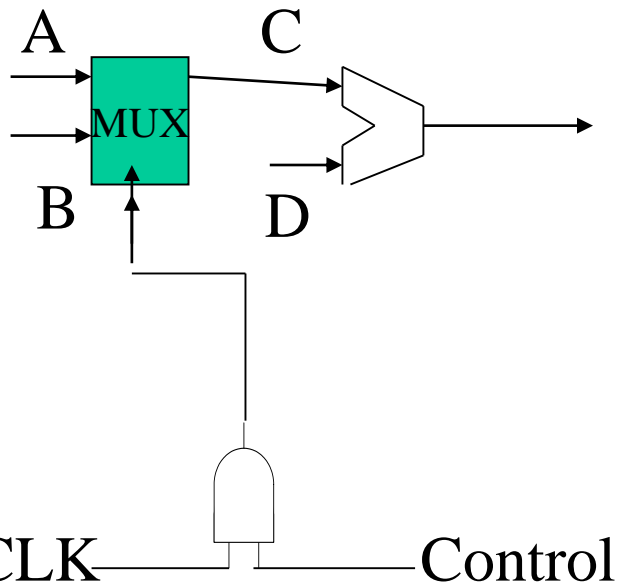
**It executes sequentially so it takes more time.**

**Structural is based on interconnecting tested working components.**

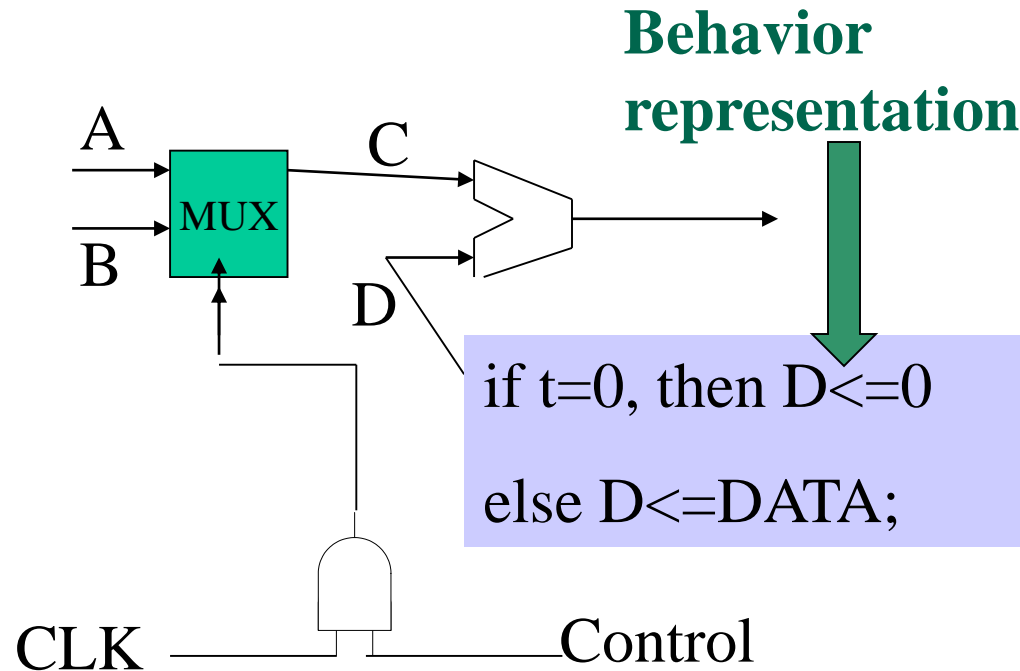
**Data flow the assignment is based on logic expressions**

**The difference is really in the process and signal assignment/variable assignment/scheduling and the delta function**

# Structural Synthesis



# behavioral Synthesis



# Configuration Statement

- It is used to bind the entity used with the architecture that is desired.
- Example:

*for all* : OR\_2 *use entity* OR\_2 (data\_Flow)



## **Libraries (Predefined)**

### **STD**

Provides declarations for predefined constructs in VHDL.

### **WORK**

The working library into which design units are presently being analyzed are stored. (ie. design entities).

## Libraries

The design entities can be stored in libraries

Libraries and their storage and implementation are achieved outside VHDL. VHDL is only the language that facilitates the usage of the libraries and its contents ie., the design entities.

- Any VHDL entity that can be analyzed is a **COMPLETE DESIGN ENTITY**
- \* analysis means checking the **syntax** and **symantic** of a design entity statically.
- \* simulate means checking the behaviour of the modelled entity dynamically.

\* There are two pre-defined libraries in VHDL:

**STD** The standard IEEE library that holds many predefined types such as BIT. Many of these types are used almost like a reserved word because they are already predefined in the STD library.

**WORK** This is the working library, where we store our currently analysed design entities

# Structural Modeling

Structural modeling is the description of set of interconnected components that are previously defined, compiled and verified.

## Real Life Design and Implementation

- 1) Design the board
- 2) Design the chips
- 3) Place sockets on the board
- 4) Put the chips in the socket

## That is exactly how VHDL operates

- 1) Design an entity that is the board
- 2) Design the entities that are the chips
- 3) You have components that are the sockets
- 4) Design entities are put in the socket

A VHDL STRUCTURAL Model interconnects the instances of chip sockets holding the chips.

## --Interface

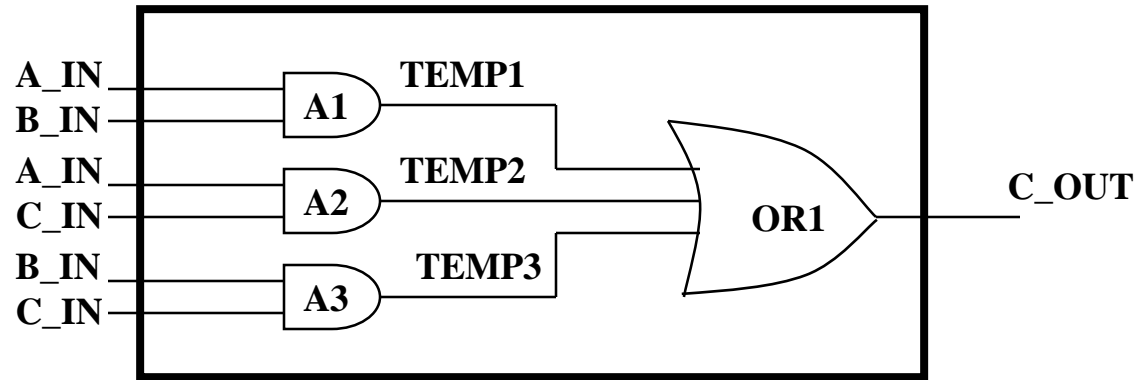
**entity** CARRY **is**

**port**

(A\_IN, B\_IN, C\_IN : **in** BIT;

C\_OUT : **out** BIT);

**end** CARRY;



## --Body

**architecture** STRUCTURAL **of** CARRY **is**

### -Declaration of components

**component** AND\_2 **port** (A, B : in BIT ; Z : out BIT); **end\_component** ;

**component** OR\_3 **port** (A, B, C : in BIT ; Z : out BIT); **end\_component** ;

### --Declare Signals

**signal** TEMP1, TEMP2, TEMP3 : BIT ;

**begin**

### -Connect Logic Operators to Describe Schematic

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) ;

O3: OR\_3 **port map** (TEMP1, TEMP2, TEMP3, C\_OUT) ;

**end** STRUCTURAL ;

# DATA\_FLOW CONSTRUCTS

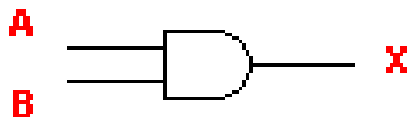
```
entity FULL_ADDER is  
port (A_IN,B_IN,C_IN : in BIT;  
SUM, CARRY : out BIT);  
end FULL_ADDER;
```

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

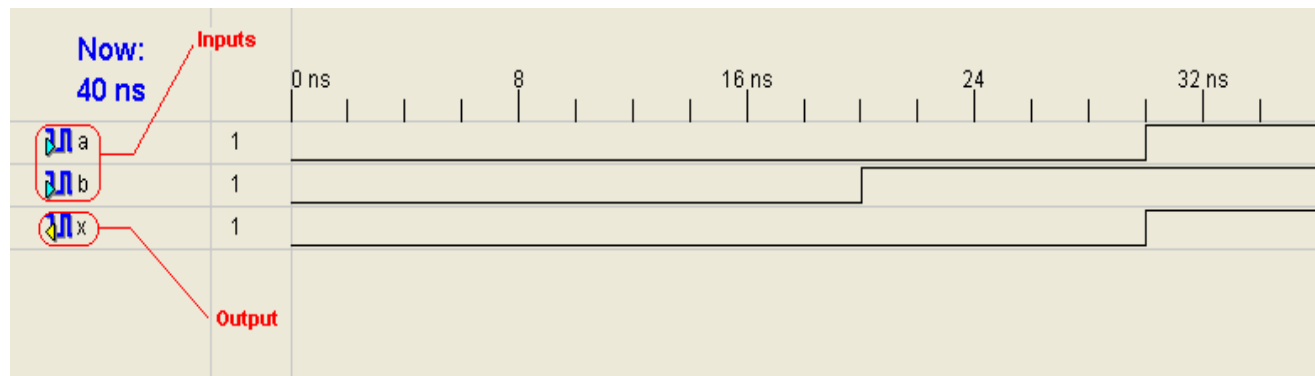
```
begin  
S1    <= A_IN xor B_IN;  
SUM   <= S1   xor C_IN;  
S2    <= S1   and C_IN;  
S3    <= A_IN and B_IN;  
CARRY <= S2   or  S3;
```

```
end DATA_FLOW;
```

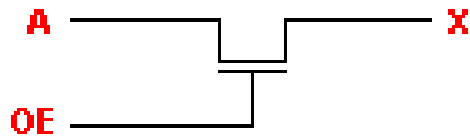
# AND Gate simulation



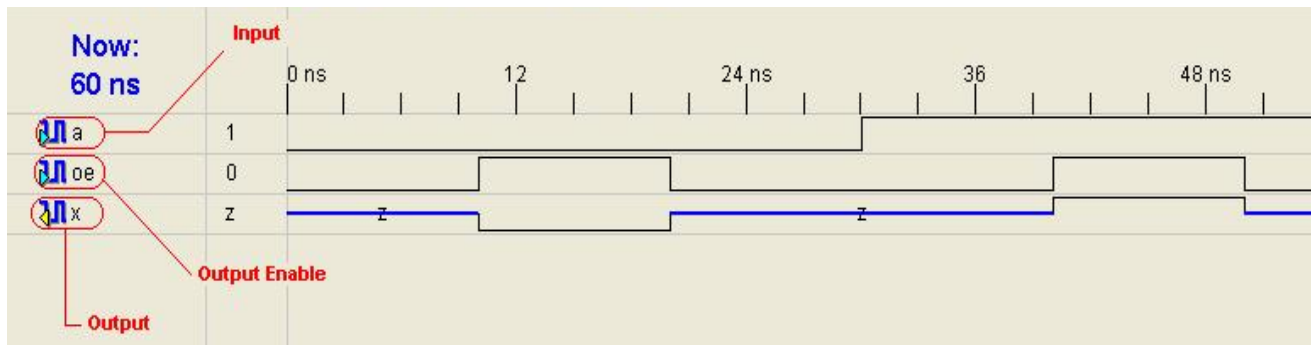
```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3 use IEEE.STD_LOGIC_ARITH.ALL;
4 use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6
7 entity AND_2 is
8 port (A,B:in std_logic;X:out std_logic);
9 end AND_2;
10
11 architecture DATA_FLOW of AND_2 is
12
13 begin
14 x<= A and B;
15 end DATA_FLOW;
```



# Passgate simulation



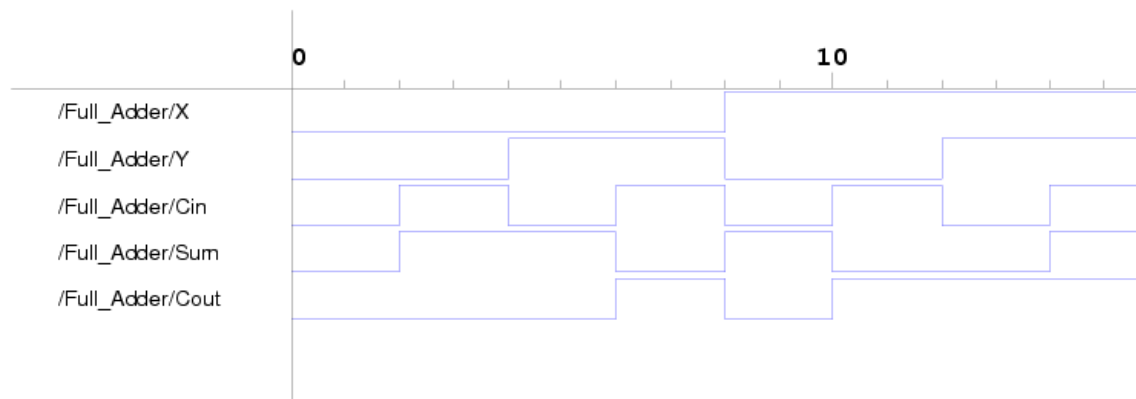
```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  entity BUFF is
7  port (A,OE:in std_logic; X:out std_logic);
8  end BUFF;
9
10 architecture DATA_FLOW of BUFF is
11
12 begin
13   X<=A when OE='1' else 'Z';
14
15 end DATA_FLOW;
16
17
```



```

library ieee;
use ieee.std_logic_1164.all;
entity Full_Adder is
    -- generic (TS : TIME := 0.11 ns; TC : TIME := 0.1 ns);
    port (X, Y, Cin: in std_logic; Cout, Sum: out std_logic);
end Full_Adder;
architecture Concurrent of Full_Adder is
begin
    Sum <= X xor Y xor Cin after 0.11 ns ;
    Cout <= (X and Y) or (X and Cin) or (Y and Cin) after 0.11 ns;
end Concurrent;

```





# What Synthesis Programs do ?

Synthesis programs are large packages that contain many algorithms for processing the VHDL Code, which generally include :

Check the Syntax and Semantics of the Code

Deduce the logic and state elements

Optimize the technology independent functions (Boolean and State optimization)

Map the optimized structure to the target technology (Place and Route)

Evaluate the Structure performance ( Timing, Power and Area)

Perform technology-dependent to obtain better final result.

It is important to note that there is a different synthesis paths

with different synthesis packages from different companies,

And that not everything that can be simulated can be synthesized.

You always have to refer to the synthesis packages to see the sequential construct or the module that you have selected is it synthesizable or not.

# Points to watch for

- The way the code is written will greatly affect the size and speed of the synthesized circuit.
- For test bench, you may write unsynthesizable structures to test your circuit.
- Always use hierarchy, regularity, modularity and locality in your code.
- Insert comments to describe the variables and your construct.
- Write: date, author, name of the entity in the first line of your entity.