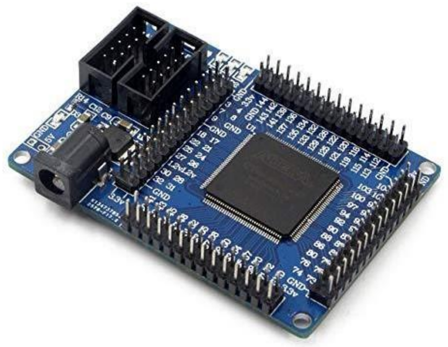
ET3233 - Digital Systems Design

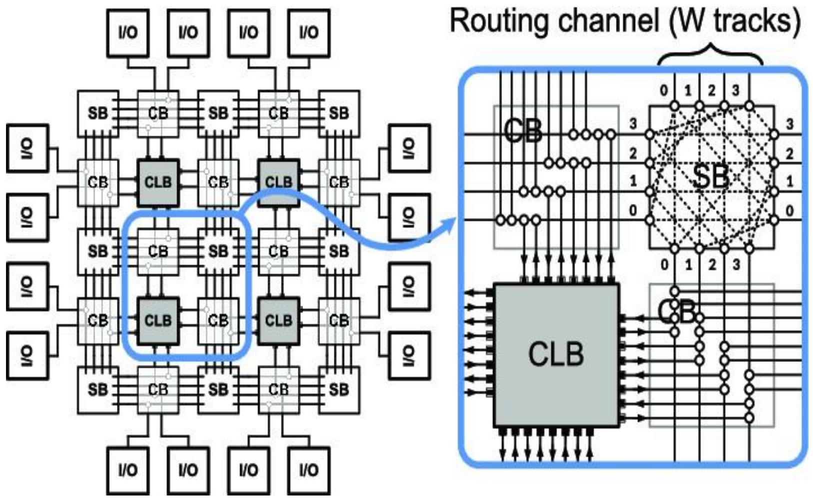
Hardware Description Languages (HDL)

Geeth Karunarathne

FPGA

- An FPGA is an **array of logic gates,** and this array can be **configured in the field**.
- It consists of a set of **C**onfigurable **L**ogic **B**locks and **S**witching **B**locks.





Hardware Description Languages (HDL)

- Required to programme FPGAs.
- There are two most popular HDLs.
  - VHDL
    - Developed by US Deprtment of Defence in 1983
    - Based on ADA language
    - Become an IEEE standard in 1987
  - Verilog
    - Gateway Design Automation in 1984
    - Similar to C language
    - Become an IEEE standard 1995

VHDL vs. Verilog

VHDL	**Verilog**
Strongly typed	Weakly typed
Easier to understand	Less code to write
More natural in use	More of a hardware modeling language
Wordy	Succinct
Non-C-like syntax	Similarities to the C language
Variables must be described by data type	A lower level of programming constructs
Widely used for FPGAs	A better grasp on hardware modeling
More difficult to learn	Simpler to learn

VHDL (Very high speed integrated circuits HDL)

- **VHDL** stands for **V**HSIC (Very High Speed Integrated Circuits) **H**ardware **D**escription **L**anguage.
- A hardware description language is inherently parallel.
  - i.e. commands, which correspond to logic gates, are executed (computed) in parallel, as soon as a new input arrives.
- A HDL program mimics the behavior of a physical, usually digital, system.
- It also allows incorporation of timing specifications (gate delays) as well as to describe a system as an interconnection of different components.
- Once the VHDL code for a design is written, it can be simulated by a software program to determine if the design works properly.

Libraries and Packages in VHDL

- A library can be considered as a place where the compiler stores information about a design project.
- A VHDL package is a file or module that contains declarations of commonly used objects, data type, component declarations, signal, procedures and functions that can be shared among different VHDL models.

Libraries and Packages in VHDL

- Declaring the use of a library and package:

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

$$\textbf{use}\ \underbrace{\text{ieee.}}_{\text{Library}}\underbrace{\text{std\_logic\_1164.}\textbf{all}}_{\text{Package}};$$

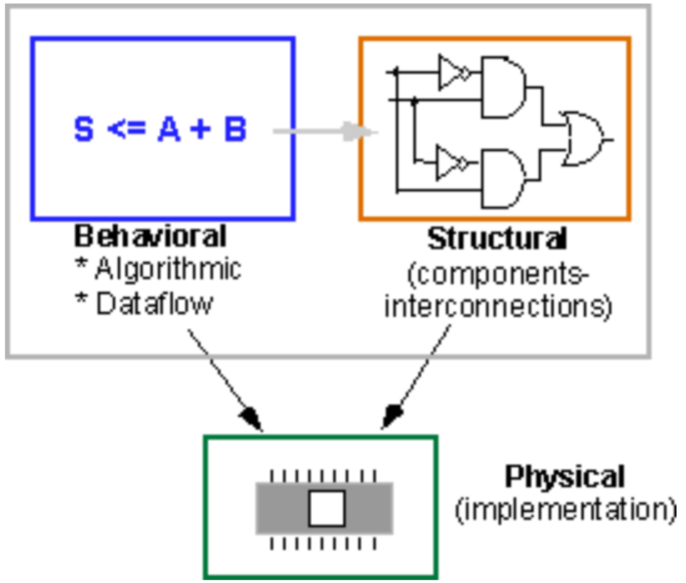
Commonly used Packages in IEEE Library

- std_logic_1164 package: defines the standard datatypes
- std_logic_arith package: provides arithmetic, conversion and comparison functions for the signed, unsigned, integer, std_ulogic, std_logic and std_logic_vector types
- std_logic_unsigned
- std_logic_misc package: defines supplemental types, subtypes, constants and functions for the std_logic_1164 package.

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

Abstraction of Digital Systems

- A digital system can be represented at different levels of abstraction.
- This keeps the description and design of complex systems manageable.



Behavioral Abstraction

- The highest level of abstraction is behavioral level.
- This defines how the system behaves in terms of interconnection between components.
- A behavioral description specifies the relationship between the input and output signals.
- This could also be a Boolean expression or an abstract description.

Behavioral Abstraction - Example

- A circuit that warns car passengers when the door is open or the seatbelt is not used whenever the car key is inserted in the ignition lock.

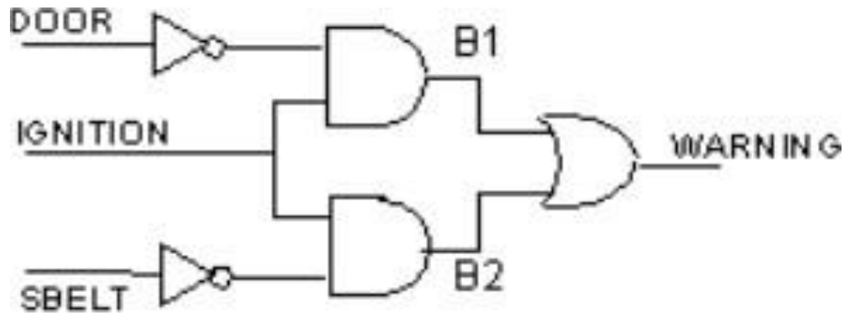
Warning = Ignition_on AND (Door_open OR Seatbelt_off)

Structural Abstraction

- Describes a system as a collection of gates and components that are interconnected to perform a desired function.
- A structural description could be compared to a schematic of interconnected logic gates.
- It is a representation that is usually closer to the physical realization of a system.

Structural Abstraction - Example

- A circuit that warns car passengers when the door is open or the seatbelt is not used whenever the car key is inserted in the ignition lock.



Abstraction of Digital Systems cntd.

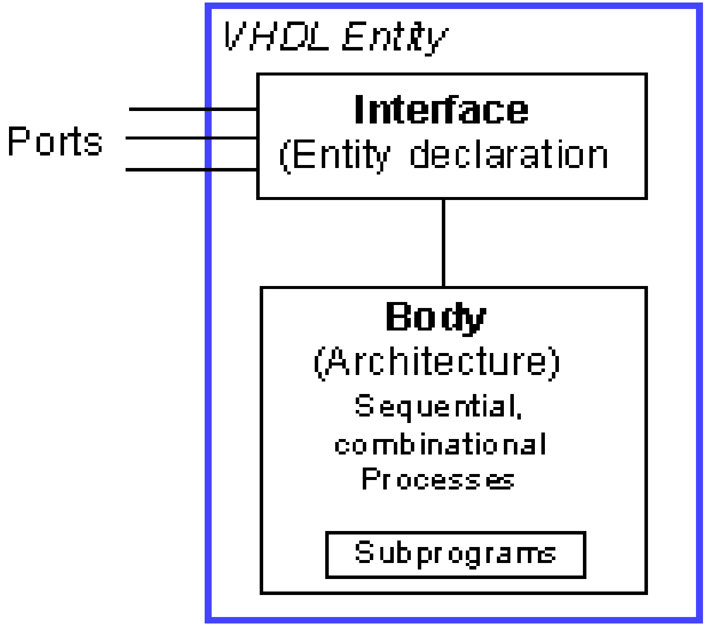
- VHDL allows one to describe a digital system at the structural or the behavioral level.
- The behavioral level can be further divided into two kinds of styles: Data flow and Algorithmic.
- The dataflow representation describes how data moves through the system.
- This is typically done in terms of data flow between registers (Register Transfer level).
- The data flow model makes use of concurrent statements that are executed in parallel as soon as data arrives at the input.

Abstraction of Digital Systems cntd.

- Sequential statements are executed in the sequence that they are specified.
- VHDL allows both concurrent and sequential signal assignments that will determine the manner in which they are executed.
- These concepts will be further explained in next lectures.

VHDL Entity

- VHDL Entity has 2 main parts
  - Interface (Entity Declaration)
    - Define the input and the output of the design block.
  - Body (Architecture)
    - Logical statements defining the behavior of the entity and how its inputs relate to outputs.



Entity Declaration

- The generic entity declaration is given below.

```
entity NAME_OF_ENTITY is [ generic generic_declarations);]
     port (signal_names: mode type;
            signal_names: mode type;
                :
            signal_names: mode type);
end [NAME_OF_ENTITY] ;
```

Entity Declaration

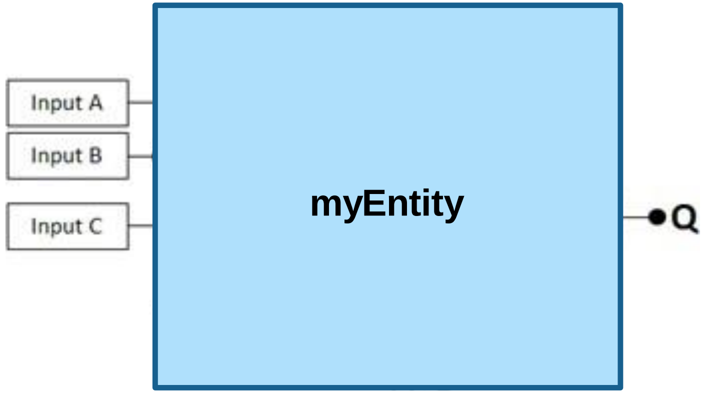
- The NAME_OF_ENTITY is a user-selected identifier.
- signal_names consists of a comma separated list of one or more user-selected identifiers that specify external interface signals.
- **mode**: is one of the reserved words to indicate the signal direction:
  - **in** – indicates that the signal is an input
  - **out** – indicates that the signal is an output of the entity whose value can only be read by other entities that use it.
  - **buffer** – indicates that the signal is an output of the entity whose value can be read inside the entity's architecture
  - **inout** – the signal can be an input or an output.

Entity Declaration

- *type*: a built-in or user-defined signal type. Examples of types are bit, bit_vector, Boolean, character, std_logic, and std_ulogic.
    - *bit* – can have the value 0 and 1
    - *bit_vector* – is a vector of bit values (e.g. bit_vector (0 to 7)
    - *std_logic, std_ulogic, std_logic_vector, std_ulogic_vector*: can have 9 values to indicate the value and strength of a signal. Std_ulogic and std_logic are preferred over the bit or bit_vector types.
    - *boolean* – can have the value TRUE and FALSE
    - *integer* – can have a range of integer values
    - *real* – can have a range of real values
    - *character* – any printing character
    - *time* – to indicate time
- **generic:** generic declarations are optional and determine the local constants used for timing and sizing (e.g. bus widths) the entity. A generic can have a default value. The syntax for a generic follows,

```
generic (
constant_name: type [:=value] ;
constant_name: type [:=value] ;
:
constant_name: type [:=value] );
```

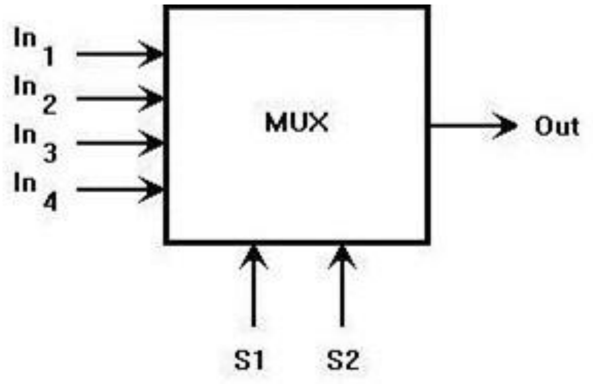
Entity Declaration



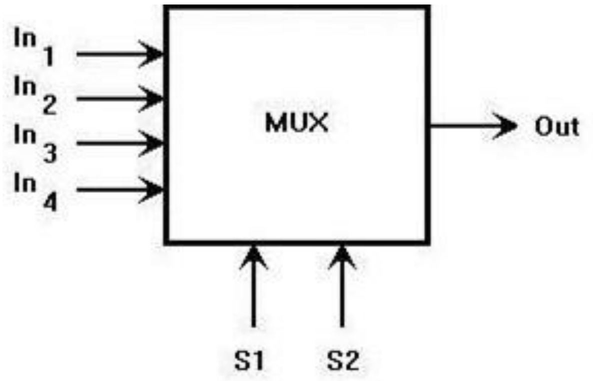
- Example:

```
entity myEntity is
        port (A, B, C: in std_logic;
                 Q: out std_logic);
end myEntity;
```

Entity Declaration - Examples



Entity Declaration - Examples



```
entity mux4_to_1 is
        port (I0,I1,I2,I3: in std_logic;
             SEL: in std_logic_vector (1 downto 0);
            OUT1: out std_logic);
end mux4_to_1;
```

$\boxed{\begin{array}{l}\texttt{SEL(0)}\\ \texttt{SEL(1)}\end{array}} \longleftarrow$ `SEL`

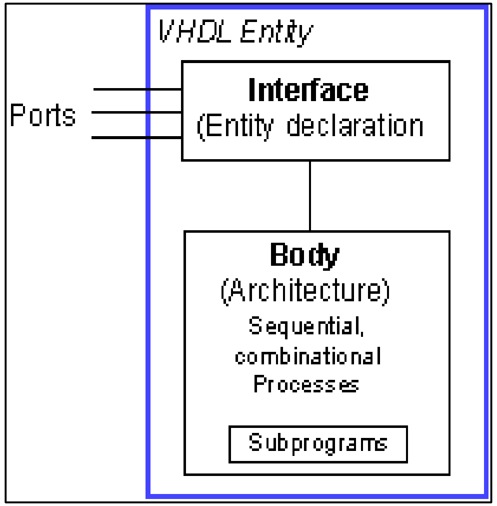
Architecture (Body)

D The generic architecture definition is given below.

```
architecture architecture_name of NAME_OF_ENTITY is
     -- Declarations
           -- components declarations
           -- signal declarations
           -- constant declarations
           -- function declarations
           -- procedure declarations
           -- type declarations
:
     begin
     -- Statements
:
     end architecture_name;
```

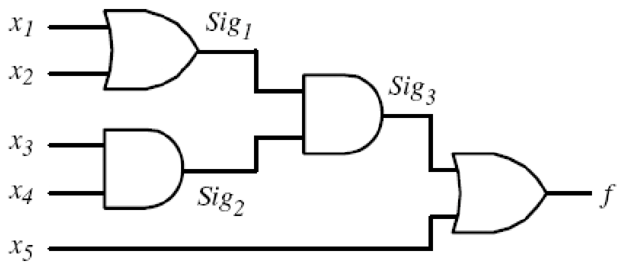
Architecture (Body)

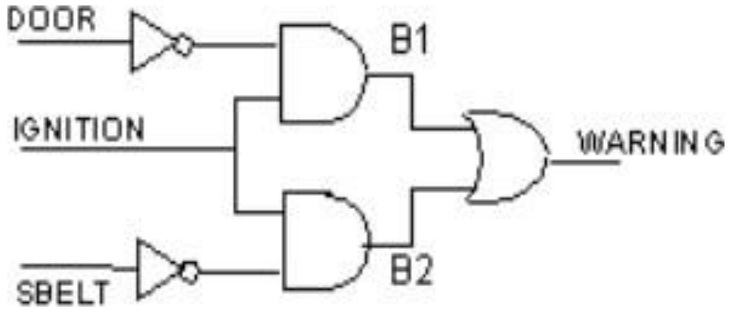
- Component declarations.
  - A predefined entity with entity declaration and architecture could be reused as a component.



Architecture (Body)

- Signal declarations.
  - Signals are the internal nets within a logic design.





  - Example:

```
signal DOOR_NOT, SBELT_NOT, B1, B2: std_logic;
```

Architecture (Body)

- Constant declarations.
  - A constant can have a single value of a given type and cannot be changed during the simulation. A constant is declared as follows,

```
constant list_of_name_of_constant: type [ := initial value] ;
```

  - where the initial value is optional. Constants can be declared at the start of an architecture and can then be used anywhere within the architecture.
  - Constants declared within a process can only be used inside that specific process.

Architecture (Body)

- Function declarations
- Procedure declarations
- Type Declarations

Commonly used from libraries.

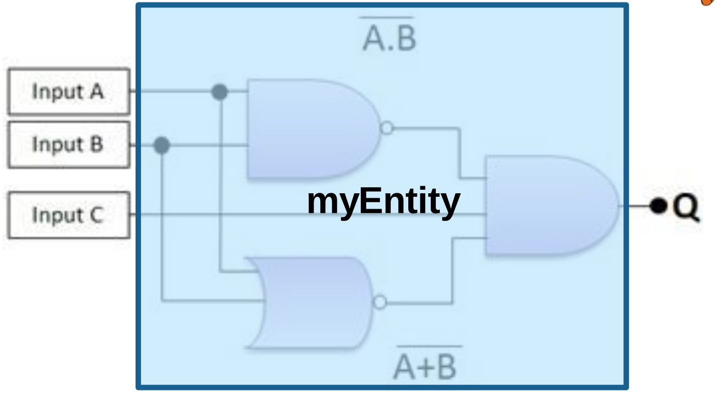
Architecture (Body)

- The statements in the body of the architecture make use of logic operators.
- Logic operators that are allowed are: **and, or, nand, nor, xor, xnor** and **not.**
- In addition, other types of operators including relational, shift, arithmetic are allowed as well.

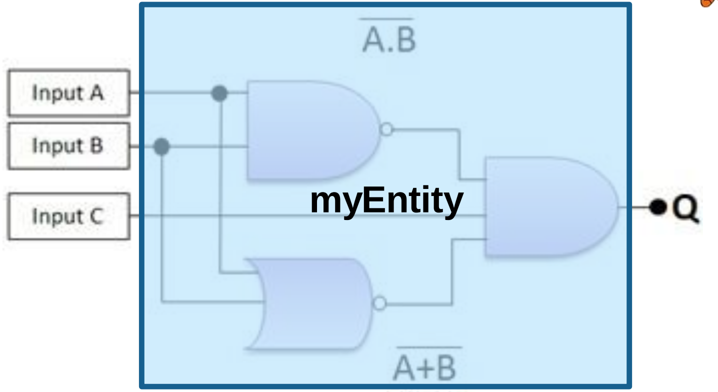
Operator Classes in VHDL

Class						
1. Logical operators	`and`	`or`	`nand`	`nor`	`xor`	`xnor`
2. Relational operators	`=`	`/=`	`<`	`<=`	`>`	`>=`
3. Shift operators	`sll`	`srl`	`sla`	`sra`	`rol`	`ror`
4.Addition operators	`+`	`=`	`&`			
5. Unary operators	`+`	`-`				
6. Multiplying op.	`*`	`/`	`mod`	`rem`		
7. Miscellaneous op.	`**`	`abs`	`not`			

Architecture (Body)



Architecture (Body)



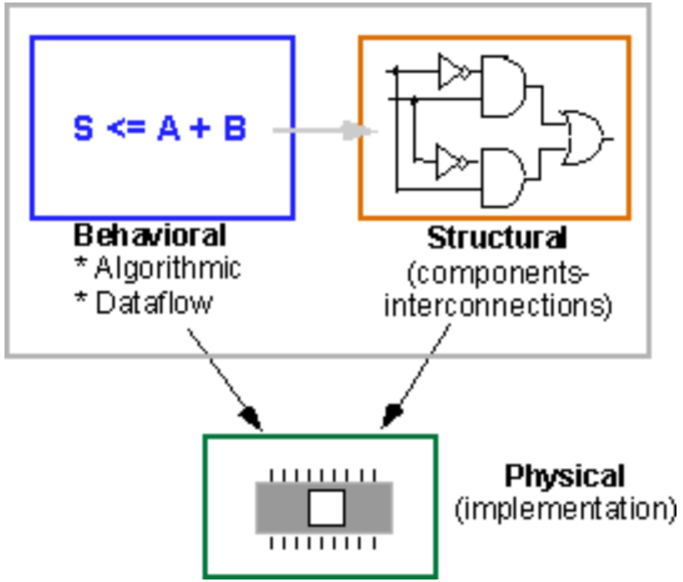
- Statements Example:

```
begin
      -- Statements
end architecture_name;
```

```
architecture archi_myEntity of myEntity is
begin
    Q<=(A nand B) or C ;
end archi_myEntity;
```

Architecture (Body)

- Coding an architecture in VHDL could be done in 2 styles.
  - Behavioral model
  - Structural model



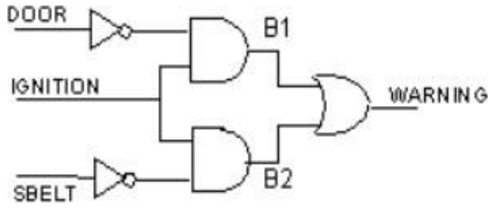
Behavioral model

- **Example:** A circuit that warns car passengers when the door is open or the seatbelt is not used whenever the car key is inserted in the ignition lock.

```
architecture behavioral of BUZZER is
begin
         WARNING <= (not DOOR and IGNITION) or (not SBELT and IGNITION);
end behavioral;
```

Structural model

Example: A circuit that warns car passengers when the door is open or the seatbelt is not used whenever the car key is inserted in the ignition lock.



```
architecture structural of BUZZER is
          -- Declarations
          component AND2
               port (in1, in2: in std_logic;
                     out1: out std_logic);
          end component;
          component OR2
               port (in1, in2: in std_logic;
                     out1: out std_logic);
          end component;
          component NOT1
               port (in1: in std_logic;
                     out1: out std_logic);
          end component;
          -- declaration of signals used to interconnect gates
          signal DOOR_NOT, SBELT_NOT, B1, B2: std_logic;
begin
          -- Component instantiations statements
          U0: NOT1 port map (DOOR, DOOR_NOT);
          U1: NOT1 port map (SBELT, SBELT_NOT);
          U2: AND2 port map (IGNITION, DOOR_NOT, B1);
          U3: AND2 port map (IGNITION, SBELT_NOT, B2);
          U4: OR2  port map (B1, B2, WARNING);

end structural;
```

Lexical Elements in VHDL

- Identifiers
- Keywords
- Numbers

Identifiers

- Identifiers are user-defined words used to name objects in VHDL models.
  - May contain only alpha-numeric characters (A to Z, a to z, 0-9) and the underscore (_) character
  - The first character must be a letter and the last one cannot be an underscore.
  - An identifier cannot include two consecutive underscores.
  - An identifier is case insensitive (ex. And2 and AND2 or and2 refer to the same object)
  - An identifier can be of any length.
- Examples of valid identifiers are: X10, x_10, My_gate1.
- Some invalid identifiers are: _X10, my_gate@input, gate-input.

Keywords

- Certain identifiers are used by the system as keywords for special use such as specific constructs.
- These keywords cannot be used as identifiers for signals or objects we define.
- Refer next slide for the complete keyword list.

Reserved words in VHDL				
abs	disconnect	is	out	sli
access	downto	label	package	sra
after	else	library	port	srl
alias	elsif	linkage	postponed	subtype
all	end	literal	procedure	then
and	entity	loop	process	to
architecture	exit	map	pure	transport
array	file	mod	range	type
assert	for	nand	record	unaffected
attribute	function	new	register	units
begin	generate	next	reject	until
block	generic	nor	return	use
body	group	not	rol	variable
buffer	guarded	null	ror	wait
bus	if	of	select	when
case	impure	on	severity	while
component	in	open	signal	with
configuration	inertial	or	shared	xnor
constant	inout	others	sla	xor

Numbers

- The default number representation is the decimal system.
- VHDL allows integer literals and real literals.
- Integer literals consist of whole numbers without a decimal point, while real literals always include a decimal point.
- Exponential notation is allowed using the letter “E” or “e”.
- For integer literals the exponent must always be positive. Examples are:
  - Integer literals: 12   10   256E3   12e+6
  - Real literals: 1.2   256.24 3.14E-2
- The number –12 is a combination of a negation operator and an integer literal.

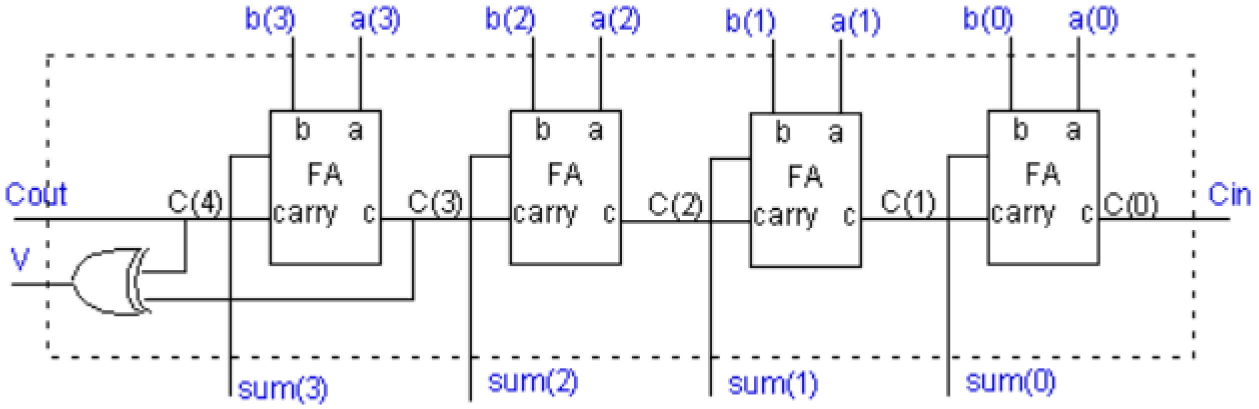
Numbers

- To express a number in a base different from the base “10”, one uses the following convention: base#number#.

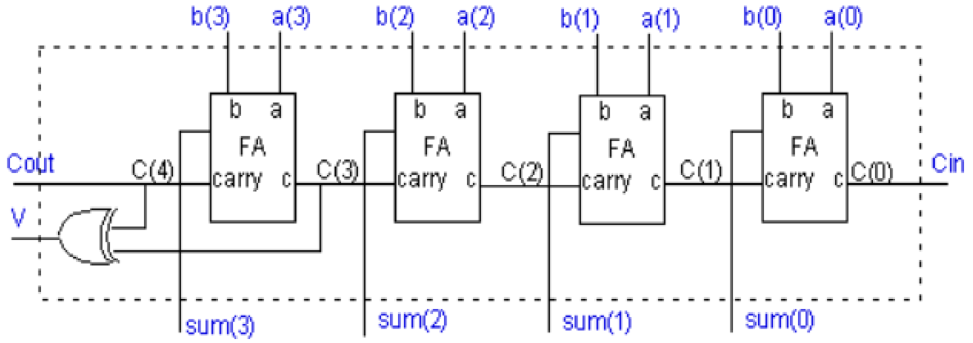
  - Example: representing the decimal number “18”
    - Base 2: 2#10010#
    - Base 16: 16#12#
    - Base 8: 8#22#
- To make the readability of large numbers easier, one can insert underscores in the numbers as long as the underscore is not used at the beginning or the end.
  - 2#1001_1101_1100_0010#
  - 215_123

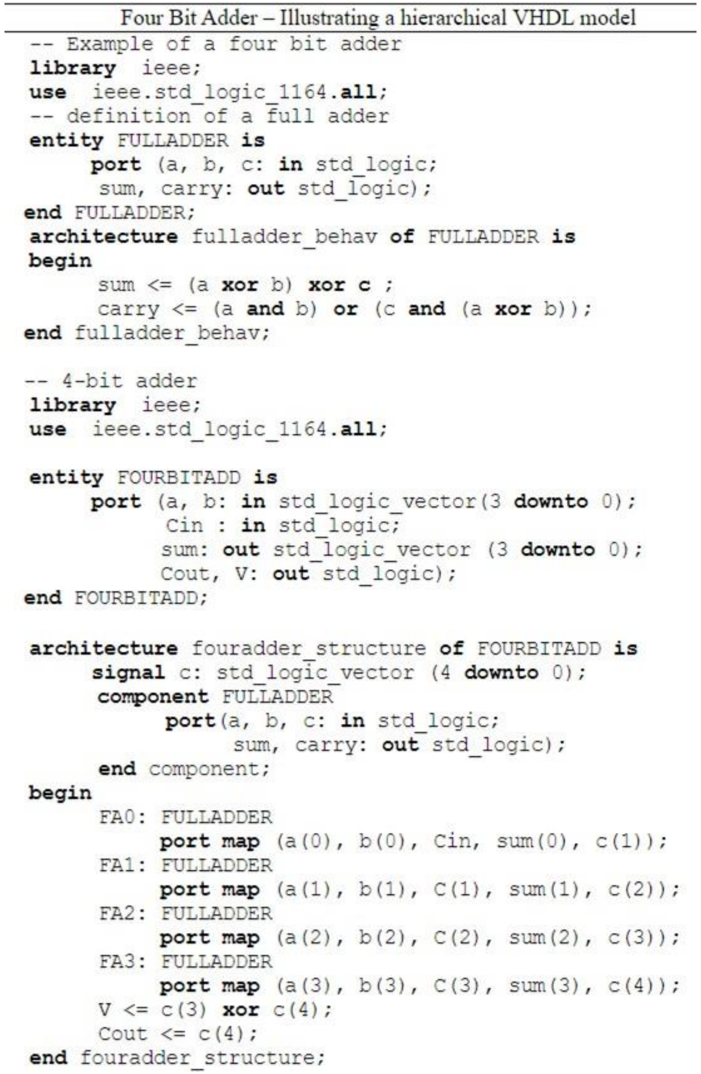
Activity

- 4-bit Adder



Activity-Answer




Four Bit Adder – Illustrating a hierarchical VHDL model

```
-- Example of a four bit adder
library  ieee;
use  ieee.std_logic_1164.all;
-- definition of a full adder
entity FULLADDER is
      port (a, b, c: in std_logic;
       sum, carry: out std_logic);
end FULLADDER;
architecture fulladder_behav of FULLADDER is
begin
       sum <= (a xor b) xor c ;
       carry <= (a and b) or (c and (a xor b));
end fulladder_behav;

-- 4-bit adder
library  ieee;
use  ieee.std_logic_1164.all;

entity FOURBITADD is
      port (a, b: in std_logic_vector(3 downto 0);
             Cin : in std_logic;
             sum: out std_logic_vector (3 downto 0);
             Cout, V: out std_logic);
end FOURBITADD;

architecture fouradder_structure of FOURBITADD is
      signal c: std_logic_vector (4 downto 0);
       component FULLADDER
             port(a, b, c: in std_logic;
                    sum, carry: out std_logic);
       end component;
begin
       FA0: FULLADDER
             port map (a(0), b(0), Cin, sum(0), c(1));
       FA1: FULLADDER
             port map (a(1), b(1), C(1), sum(1), c(2));
       FA2: FULLADDER
             port map (a(2), b(2), C(2), sum(2), c(3));
       FA3: FULLADDER
             port map (a(3), b(3), C(3), sum(3), c(4));
       V <= c(3) xor c(4);
       Cout <= c(4);
end fouradder_structure;
```

Reference

VHDL Tutorial by Jan Van der Spiegel

Department of Electrical and Systems Engineering

University of Pennsylvania

http://digitales.itam.mx/sites/default/files/u452/vhdl_tutorial.pdf

Thank You !