Introduction to HDL

The hardware description language (HDL) is used for the design of electronic circuits, describing its interface, operation and verification of its functionality without an actual hardware using simulation and synthesis.
It is a text-based language to design a circuit, instead of conventional drag and drop circuit design language.

Any hardware design language should necessarily have the following features:

1) It should have the capability to describe the hardware structure of the circuit to be designed
2) It should be able to capture the true abstract behaviour of the circuit.
3) It should also have the flexibility in the levels of abstraction at which the circuit is described.

Netlist: A textual representation of circuits schematic in terms of a number of components, identifying each component and nets that are connecting the components.
RTL: A level of description of digital circuit in which operation if spread over several clock-cycles describes circuit behaviour as a flow of data between registers.
Simulation: A process of functional verification of circuits using a model of the circuit without an actual hardware in place under the given set of its conditions, and the output is timing waveforms

VHDL is an acronym for VeryHigh Speed Integrated (VHSIC) Hardware Description Language (HDL). It is a design automation tool that is used to model digital systems at different levels of abstraction, verifying its

functionality, generating test data, and map the verified circuit to hardware.
VHDL is used in field programmable gate arrays (FPGAs), complex programmable logic devices (CPLDs) and application specific integrated circuits (ASIC) design. It is a versatile language to design digital circuits, with levels of complexity varying from simple gates to VLSI circuits.
VHDL model of a device supports external and internal views. External view specifies interface of the device with the user and internal view captures the functionality of the device.

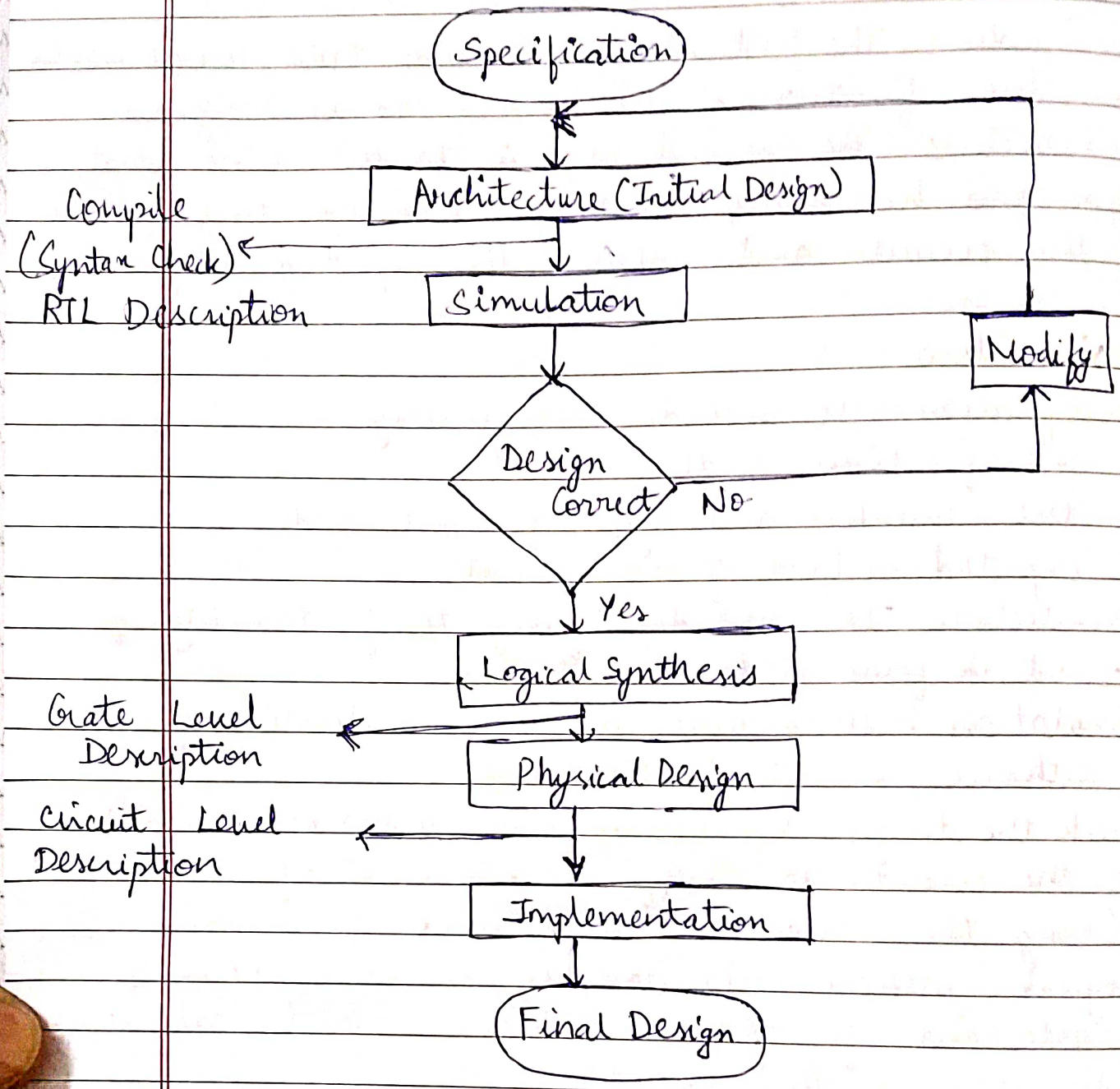
Different HDLs: AHPL, interactive design language (IDL), instruction set processor specification (ISPS), advanced boolean equation language (ABEL) and Verilog.
ABEL is used for PLD design. It allows one to describe the digital design with equations, truth tables & state diagrams. It allows us to write hardware independent programs. It allows user to choose best PLD.

Design Flow

1) The first step is to determine at what level the system has to be described. The system level description is the highest level of description, where functionality of circuit is of prime concern.
2) Next is

Design Flow

1) Specification: It includes determining
   - the function of the circuit
   - the number and types of input and
   - expected output of the circuit.
2) Architecture: It includes defining the functionality of circuit in form of its architecture.
3) Simulation: It includes functional verification of circuit without an actual hardware
4) Check the design: Simulation makes use of RTL description of the circuit to verify its correctness. If design is wrong then modifications are made to design
5) Logical Synthesis: It describes circuit in terms of gates and flip flops. Synthesis generates gate level description of the circuit
6) Physical Design: User-defined design file is mapped to the target technology and the outputs are checked.
7) Implementation: Circuit is implemented. Silicon wafers and fabrication are used to yield the final design.



Design Flow

Levels of system Description

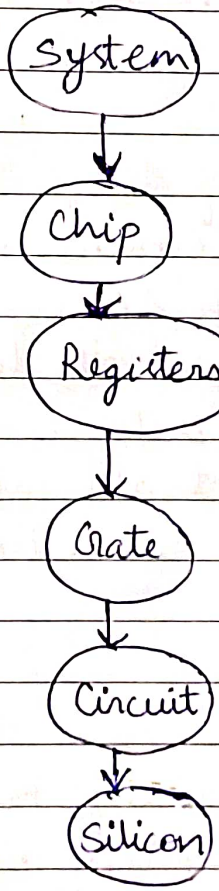
1) System Level Description : It is concerned only with the functionality of circuit
2) Chip Level Description : It is determined if the circuit is to be designed on single chip or multiple chips.
3) RTL Description : It describes a circuit in

terms of sequential registers and combinational circuitry.

4) Gate Level Description: It describes a circuit in terms of gates and flip flops.

5) Circuit Level Description: It describes circuit in terms of circuit components like resistors and capacitors

6) Silicon Level Description: Circuit is implemented in terms of silicon wafers.



Levels of System Description

EDA: Electronic Design Automation Tools

CAD tools are needed for the following tasks

1) Design Entry 2) Simulation 3) Synthesis

Design Entry:

System specifications and architecture of the design is determined using VHDL and Verilog. Information about the circuit to be designed is given in the

form of text file written in VHDL or Verilog. The code giving the system specification and the logical expression to perform the desired function is written in HDL. A language compiler is used to check syntax errors. eg. Xilinx, Aldec, Actel

Simulation: It is done to check if the circuit works perfectly. Output is given in the form of waveforms. ModelSim and Synplify are simulation tools

Synthesis: Conversion of design to gate level schematics is called synthesis. It is used to express the circuit in terms of basic components. It is used to optimise the design with reference to component delays, critical path delays or area consideration. It also checks for errors in design. Netlist, log and schematic are the outputs of synthesis.

Important Definitions

1. Simulation: It provides information regarding functionality of the system under design.

Two types of simulation are RTL simulation and functional simulation

2. Synthesis

Important Definitions

1. Simulation: It provides information regarding functionality of the system under design. Simulation is verification of VHDL code design of the circuit. Two types of simulation are:
   1) RTL simulation (pre-synthesis simulation)
   2) Functional simulation (post synthesis simulation)

2. Synthesis: It is designed as translation of a design description from one level of abstraction down to another level of abstraction. Input includes HDL description timings. Outputs include optimised net lists, estimated performance and the area of synthesised design.
   Three types of synthesis are
   a) Behaviour Synthesis: It is a process of translating algorithmic description into an RTL description. The design at RTL level includes data paths, memories and control units. It is also known as high level synthesis or architectural synthesis.
   b) RTL synthesis: It is the process of generating structural net lists for a sequential circuit from a set of register transfer functions. The task includes state minimization, state encoding, logical minimization and technology mapping
   3) Logical synthesis: It translates boolean expressions into combinational logic and is divided into 2 phases.
   (i) A front-end tool which minimises the circuit independent of technology library.
   (ii) Back end tool which maps a structural net list to an interconnection of library cells

VHDL Basics

VHDL is acronym for very high speed integrated circuit (VHSIC) hardware description language (HDL). It is used to describe the logical structure and functions of digital system. It facilitates design specification and simulation of digital systems.

Features of VHDL

1) It is hardware description language
2) It is event-driven language
3) It is platform-independent language
4) It allows both concurrent and sequential modelling
5) It provides flexibility to define data-type
6) It allows multiple architecture existing for same design
7) It supports flexible design methodologies
8) It is case sensitive language
9) It is strongly typed language
10) It allows general purpose design
11) It allows design verification at three levels.

Advantages of VHDL

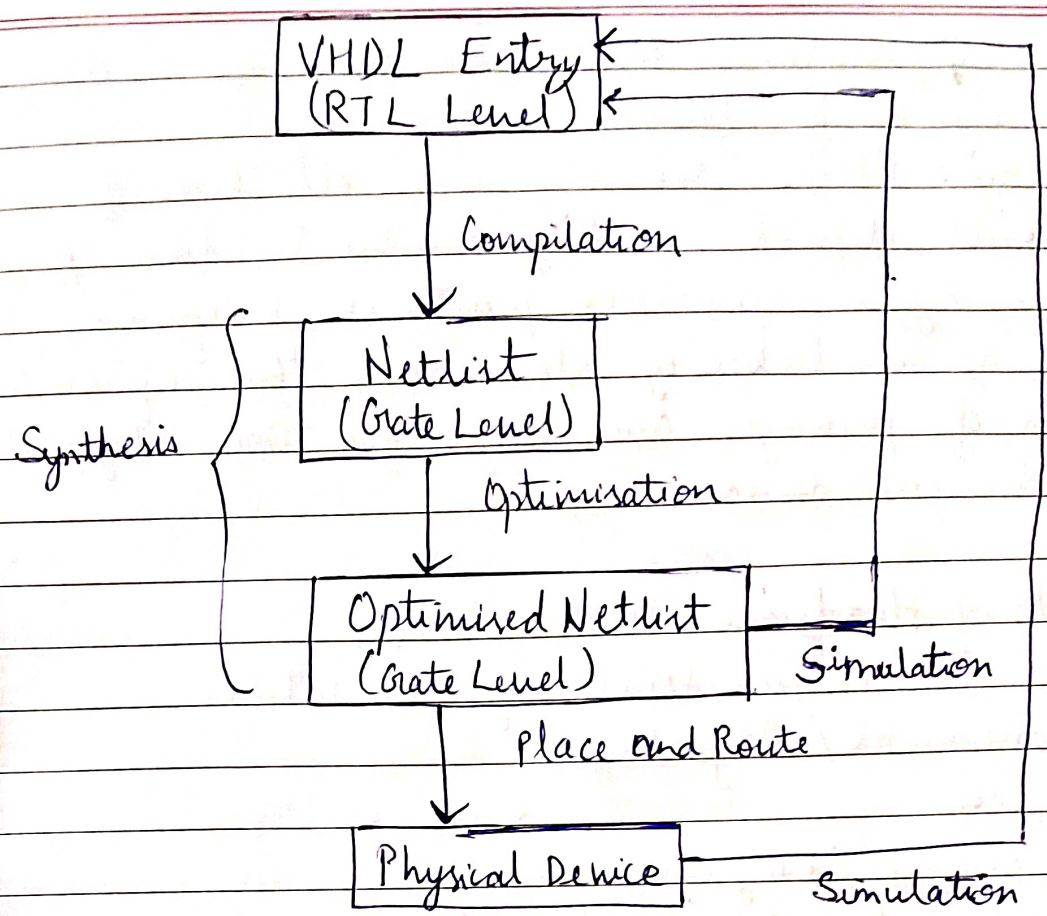
1) Early Verification of design functionality
2) Allows designer to test architecture of design
3) It is used for large projects
4) Allows design reuse
5) Increased flexibility for design changes
6) NRE cost is reduced
7) Easier design verification
8) Alternative architecture can be explored quickly

Comparison between VHDL and Verilog

1) VHDL allows both user defined and pre defined data types.
   Verilog allows only predefined data type
2) VHDL provides packages and subprograms
   Verilog doesn't provide packages and only function can be defined in form of file.
3) VHDL is strongly typed language and includes features like generics, configurations, generate statements and user defined libraries.
   Verilog doesn't have these features.
4) VHDL is suitable for large complex designs
   Verilog is suitable for primitive designs

VHDL Design Flow

1) Write VHDL code, save it with same name as it's ENTITY's name and .vdh extension
2) Compilation : It is conversion of high level VHDL language, which describes circuit at RTL into a netlist at gate level
3) Optimization is performed on gate level netlist for speed
4) Place and route (filter) will generate physical layout



VHDL Design Flow

Basic Terminologies in VHDL

1) Entity: Entity is referred to the circuit which is to be modelled. It represents the external interface of the circuit that is to be designed
2) Architecture: The architecture describes the functionality of the entity. A single entity can have multiple architectures.
3) Configuration: In case of entity with multiple architecture, a configuration is used to bind an entity with a particular architecture for simulation.
4) Subprogram: Subprograms are user defined pieces of code that are written to perform specific function and can be stored and reused when needed. VHDL supports two types of subprograms - functions and procedures
5) Package: Package is a collection of data types

and subprograms that are used in a design allowing it for reuse.

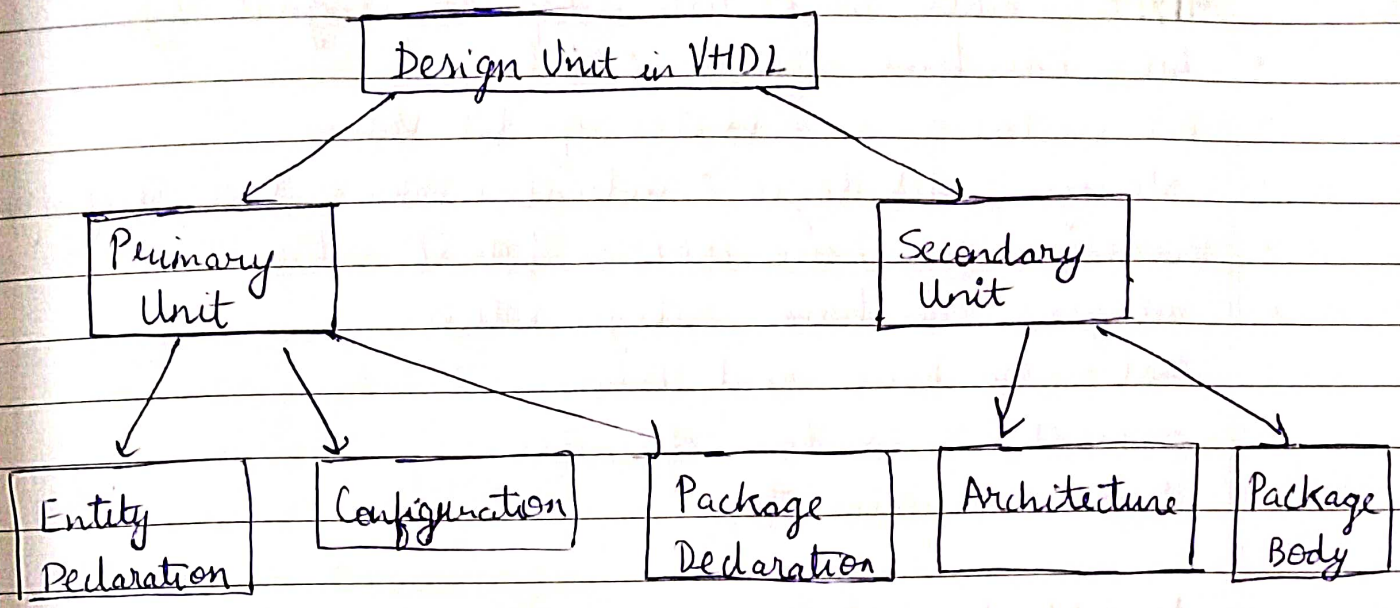
6) Library: VHDL library is a collection of all in-built defined package constructs like pre defined data types, conversion functions, language features and so on. Library stores all this information in form of packages consisting functions, type definitions and procedures.

Levels of Hardware Abstraction

There are three levels of extraction:

- Behavioural / Algorithmic
- Dataflow
- Structural / Physical

- Behavioural: It is the highest level of abstraction that describes a system in terms of what it does. It specifies relationship between input and output signals. Sequential statements are executed in sequence.
- Dataflow representation describes how a data moves through the system. This level of abstraction is close to Register Transfer Level (RTL). It makes use of concurrent statement that are executed in parallel as soon as the data arrives at input. Behavioural style is procedural while dataflow style is concurrent, event-driven approach.
- Structural level is the lowest level of abstraction It describes system as collection of gates and components that are interconnected to perform a desired function. It is physical description of the circuit. It is closer to physical realization of system.



Entity Declaration: Entity defines an external interface. Entity declaration specifies the name of entity that is being modelled and lists the set of interface ports.

General form is:

```
ENTITY  name IS
    PORT ( port_name 1, [ portname 2] : MODE type;
           port_name : MODE type;
           port_name : MODE type );
END [name];
```

MODE : is one of the reserved words to indicate signal direction:

- in - unidirectional port, indicating that the signal is an input.
- out - unidirectional port, indicating that the signal is an output
- buffer - bidirectional port, indicating that the signal is output of the entity whose value can be read inside entity's architecture.
- inout - bidirectional port, indicating that the signal can be input or output.

TYPE: built-in or user-defined signal type.

- bit - can have values 0 and 1
- bit vector - is a vector of bit values
- stdlogic, stdulogic : indicates value & strength of signal.
- Boolean - can have TRUE or FALSE values
- integer - can have integer values
- real - can have real values
- character - can have character
- time - indicates time.

Architecture : Architecture describes the functional relationship between input and output signals. Though one entity can have multiple architectures only one architecture body should be bound to an entity for execution. Architecture body consists of two parts.

- Declarative Part : includes necessary declarations such as type, signal, component and subprogram declarations.
- Statement Part: includes statements describing the functionality of the entity in terms of logical and mathematical expressions.

Syntax.

```
ARCHITECTURE architecture_name OF entity_name IS
  - declarative part
  BEGIN
      Statement 1 ;
      Statement 2 ;
END [ architecture_name ];
```

eg. Example of entity architecture for AND gate

```
ENTITY and-gate IS
    PORT ( a, b : IN BIT;
           c : OUT BIT);
END and-gate;
ARCHITECTURE simple OF and_gate IS
BEGIN
    c <= a AND b;
END simple;
```

Styles of Modelling

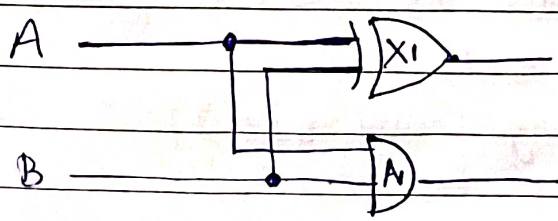
There are four styles of modelling

- Structural style
- Dataflow style
- Behavioural style
- Mixed style

1. **Structural Modelling**: It is the simplest style of modelling. It is representation of entity in terms of interconnected components. In structural modelling, architecture body of entity consists of
   * component declaration
   * component instantiation

   In component declaration each component to be used in architecture description is declared with its unique name - identifier, its number, type of input and output ports. Component instantiation is mapping between actual and formal ports of entity and components respectively. Component instantiation is port map statement.

eg.



Half Adder

```
ENTITY half_adder IS                    Entity Declaration
  PORT ( a, b : IN BIT;
         sum, carry : OUT BIT);
END half_adder;
ARCHITECTURE struct OF half_adder IS
COMPONENT and_gate IS                   Component Declaration
  PORT ( x, y : IN BIT;
         z : OUT BIT);
  END and_gate;
COMPONENT xor_gate IS
  PORT ( l, m : IN BIT;
         n : OUT BIT);
  END xor_gate;
BEGIN
X1 : xor_gate PORT MAP(a, b, sum);      } Component
A1 : and_gate PORT MAP(a, b, carry);    } Instantiation
END struct;
```

2. Dataflow Modelling

Dataflow modeling expresses the functionality of circuit as a flow of data through buses and interconnections. It is also the concurrent style of modelling. It uses logic expressions to express the flow of data from input to output through signals. It is closer to RTL level simulation

eg.

```
ENTITY half_adder IS
  PORT ( a, b : IN BIT;
         sum, carry : OUT BIT);
END half_adder
ARCHITECTURE dataflow OF half_adder IS
```

```
BEGIN
  sum <= a XOR b;
  carry <= a AND b;
END dataflow;
```

3. Behavioural Modelling: It focuses on behaviour of output with respect to changes in input. It maps the relationship between input and output. It is sequential style of modelling. It is also called algorithmic style of modelling. It is most suitable for complex designs. It is at highest level of abstraction. It is used for faster simulation of circuit.

eg.

```
ENTITY half_adder IS
  PORT (a,b : IN BIT;
        sum, carry : OUT BIT);
END half_adder;
ARCHITECTURE behave OF half_adder IS
  BEGIN
  PROCESS (a,b)
  BEGIN
  sum <= a XOR b;
  carry <= a AND b;
  END PROCESS
END behave.
```

It uses special construct PROCESS within which everything written is treated as sequential.

4. Mixed Style of Modelling

It uses all styles of modelling, namely structural, dataflow and behavioural modelling in single

architecture body.
Concurrent and sequential statements can be part of same architecture.

eg.

```
ENTITY half-adder IS
   PORT ( a, b : IN BIT;
          sum, carry : OUT BIT );
          clk : IN STD-LOGIC );
END half-adder;
ARCHITECTURE behave OF half-adder IS
  BEGIN
   carry ← a AND b;
   PROCESS (clk)
      BEGIN
        sum ← a XOR b;
   END PROCESS;
END behave;
```

Configuration: It helps in associating an entity with multiple architecture and binding an entity-architecture pair at run time. It configures the binding between entity and one of its many existing architecture models

Packages: It allows reusability and sharing of code by providing a modular storing mechanism. A package is a collection of components, functions / procedures, data types and other design units stored in design libraries
Package consists of two parts
- Package declaration
- Package body (optional)

Syntax:

```
PACKAGE name IS
    All declarations:
    Subprogram declaration;
    Type / Data Object declaration;
    Component Declaration;
    Constants / Deferred Constants;
    Attributes;
    Aliases;
USE clauses;
END name;
PACKAGE BODY name IS          Optional
    Deferred constants initialization;
    Local declarations, if any;
    Subprogram definitions;
END [name];
```

eg.

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.all;
PACKAGE new_pack IS
TYPE word_size IS STD_LOGIC_VECTOR
TYPE mem_size IS ARRAY OF word_size;
END new_pack;
```

~~Syntax to Use Package~~

Syntax to use package

```
USE Library_name.Package_name.component_name;
                OR
USE Library_name.Package_name.subprogram_name;
                OR
USE Library_name.Package_name.all;
```

Behavioural	Data flow	Structural
→ Algorithmic method of design expressed in sequential VHDL process	Data flows through a design from input to output. Data transformations are expressed as set of concurrent statements	Components of design are interconnected. It is achieved through component instantiation.
→ Consists of sequential program statements	set of concurrent statements	set of interconnected components
→ Represents function a behaviour	Represents behaviour	Represents structure explicitly.
→ No structural information is present	It implies structure	It can be heirarchical
→ Level of abstraction is gate	Level of abstraction is algorithm or gate	RTL is present
→ Truth table is design simplification	Boolean equations are design simplification	Logical diagram is design simplification

Identifier: Identifiers are user defined words that are used to name objects in VHDL models.

Rules:

a) It should contain an alpha-numeric character (A-Z, a-z, 0-9) and the underscore (_) character.

b) The first character must be a letter and last cannot be an underscore

c) An identifier cannot include two consecutive underscores.

d) An identifier is not case-sensitive

e) An identifier can be of any length.

Keywords: Some words are used by language itself indicating specific built-in constructs. These reserved words are called Keywords.
eg. function, component, generate, etc.

Data Object

VHDL is object based language and anything which is capable of holding some value is called object. The class of object determines nature of data it holds.
There are 3 types of data classes:

- Constant
- Variable
- Signal

Constant is a class that holds objects with single specific value. Once assigned it remains static and cannot be changed throughout the simulation.
CONSTANT keyword is used.

Syntax:
CONSTANT const_name : TYPE [:= value];
eg.
CONSTANT a : INTEGER := 2;
CONSTANT rise_time : TIME := 5ns;

Variable

Variables are data objects that are used for local storage and can hold a single value.
Variables are used inside subprograms or inside a process.

Syntax:
VARIABLE var_name : TYPE [:= initial value];
eg. VARIABLE a : INTEGER := 2;

Signals

Signals are data objects that are used to model

interconnections and are capable of holding a list of values. Every signal has a signal driver associated with it which determines the value and timing of changes of the signal. A signal cannot have more than one drivers.

Syntax:

SIGNAL sig-name : TYPE [:= initial value];

eg.

SIGNAL a : INTEGER := 2;

Data Types

Data type specifies the values a signal can take. Data types are classified into five data types:

1) Scalar types: It includes numeric and enumerated data types. Numeric type consists integer, floating point and physical types. Enumerated types consists bit, boolean and character.
2) Composite Types: Arrays and record types
3) Access Type: Pointers
4) File Type: Provides access to objects that contain a sequence of values of given type.
5) Other Types: They include data types provided by external libraries

1. Scalar Types

Scalar types consists of enumeration types, integer types, physical types and floating point types. Enumeration data types and integer types are called discrete types. Integer types, floating point type and physical types are called numeric types.

a) Integer Types : It covers all integer values. User can specify range by using Keyword range. We can define subtype of base type whose range is contained within bounds of base type.

eg. type num is integer;

subtype shorter is short range 0 to 31;

b) Real (Floating Point) Type

Range is -1E38 to +1E38

eg. type Real_Data is real;

type Voltage is range -12.0 to +12.0;

c) Enumerated Type

Bit, Boolean, Character and severity-level are enumerated types. Object with type severity can take one of the four values: note, warning, error or failure.

eg type Bit is ('0', '1');

type Switch_level is ('0', '1', 'x');

d) Physical Type : Values of physical type represents measurements of some quantity. Physical type defines both type and subtype of that type.

eg. type time is range -1E8 to 1E8 units;

e) User Defined Types : User can define a type by using predefined word type.

eg. type arith_op is (add, sub, mul, div);

2. Composite Type : Composite types are used to define collection of values. These include both arrays and records of values.

a) Array Type : Array consists of elements of same subtype, ~~Values of array object~~

eg. type num is integer;

type numarr is array (7 downto 0) of num;

b) Record Type:
Objects of record type consists of named elements.
eg. type DATE is
record
DAY : INTEGER range 1 to 31;
MONTH : MONTH_NAME;
YEAR : INTEGER range 0 to 4000;
end record;

3. Access Types: Access types are pointers to dynamically allocated object of some other type.
eg. type ptr is access date;

4. File Types: File Types are used to define objects representing files in host system.
eg. type file_type_name is file of type_name;
type FT is file of TM;

5. Other Types:
a) std_logic: std_logic is data type defined by IEEE standard 1164.
b) Signed: Signed is numeric type. ~~It is~~ declared in external It represents signed integer data in form of array.
c) Unsigned: It represents integer data in form of an array of std_logic and is declared in external package numeric_std.

Date
Page

Operators

It is a logical or mathematical function which takes one or two values and produces a single result.

Operators are of five types

1) Arithmetic Operators
2) Logical Operators
3) Relational Operators
4) Shift and rotate Operators
5) Miscellaneous Operators

1. Arithmetic Operators: They support basic operations like addition, multiplication and unary operations.

Addition & subtraction: It works on integer type and floating point type. Both operands are of same type.

eg. c ← a+b

Concatenation: It is used to join together two or more 1-d arrays or single elements.

eg. 'A' & 'B'

Multiplication: It works on operands of same type.

eg c ← a * b;

Division: Both operands are of same type.

c ← a / b;

Remainder: Returns remainder

eg. A REM B

Modulus: It works on integer values.

eg. A MOD B

Unary operators: + indicates positive value - indicates negative value.

2. Logical Operators: They are used to perform Boolean logic

They are defined for bit, boolean, std_logic and std_ulogic types and their 1d vectors. They are used to define Boolean logic expression or perform bit per bit operation on array of bits.
eg. AND, OR, NOT, NAND, NOR, XOR, EXNOR

3. Relational Operators: They return Boolean true or false
a) Equality eg.

```
IF a = b THEN
    c <= '0';
ELSE
    c <= '1';
END IF;
```

b) Inequality: eg.

```
IF a /= b THEN
    c <= '1';
ELSE
    c <= '0';
END IF;
```

c) Smaller than. eg.

```
IF a<b THEN
    c <= '1';
END IF;
```

d) Smaller than : eg.

```
IF a<b THEN
    c <= 'L';
END IF;
```

e) Smaller than or equal to :
eg.

```
IF a<=b THEN
    c <= '1';
END IF;
```

f) Greater than eg.

```
IF a > b THEN
  c <= '1';
END IF;
```

g) Greater than or equal to: eg.

```
IF a >= b THEN
  c <= '1';
END IF;
```

4) Shift and Rotate Operators:

a) Shift left logical (SLL): eg.

a <= 101001

b <= a SLL 2 = 10100

b) Shift right logical (SRL): eg.

a <= 101001

b <= a SRL 2 = 001010

c) Shift left arithmetic: Vacated bits on right are replicated with rightmost bit

eg. a <= 101001

b <= a SLA 2 = 100111

d) Shift right arithmetic: Vacated bits on left are replicated with the leftmost bit.

eg. a <= 101001

b <= a SRA 2 = 111010

e) Rotate Left: eg

a <= 101001

b <= a ROL 2 = 100110

f) Rotate Right: eg.

a <= 101001

b <= a ROR 2 = 011010

5) Miscellaneous Operators

a) Absolute (abs): It works with numeric data type and returns absolute value.
eg. abs(−2) = 2

b) Exponentiation: Left operand is numeric value and right operand is integer.
eg. b ← 3 ** 2 = 9

c) Logical Negation (not)

Precedence Order of Operators

Logical	Lowest precedence
Relational	
Shift	
Addition	
Unary	
Multiplying	
Miscellaneous	Highest Precedence

~~Concurrent Coding~~

Data Flow Modelling / Concurrent Coding

Dataflow modelling is concurrent style of modelling. In concurrent style of modelling all statements are executed in parallel. Concurrent signal assignment statement are event-triggered i.e. they are executed only when any event occurs on signal.

Concurrent statements by default works on delta delay, which defines an infinitesimally small delay of δt between event change and its effect on output.

The difference between concurrent signal assignment statement and ~~signal assignment~~ sequential signal assignment statement is that concurrent signal

assignment statement is event triggered statement whereas sequential signal assignment statement is executed only when the process in which they are contained is active.
eg.

```
ENTITY dataflow IS
PORT (a,b : IN INTEGER; c,d : OUT INTEGER);
END dataflow;
ARCHITECTURE ex OF dataflow IS
 BEGIN
    c <= a+b;
    d <= a*c;
 END ex;
ENTITY behavioural IS
PORT (a,b : IN INTEGER; c, d : OUT INTEGER);
END behavioural;
ARCHITECTURE ex OF behavioural IS
BEGIN
PROCESS (a, b)
BEGIN
    c <= a+b;
    d <= a*c;
END PROCESS;
END ex;
```

Concurrent Signal Assignment	Sequential Signal Assignment
→ It is used with dataflow style of modelling	It is used with sequential style of modelling
→ It is event-triggered	It is executed in sequence
→ Order of statement is not important	Order of statement is important as all statements are executed in sequence.

→ 'when' clause can be used	'when' clause is not commonly used.
→ Problem of multiple drivers arises in this style	Problem of multiple drivers is not here.

Concept of Delay in Concurrent Logic

Delta (δ) is an infinitesimal unit of time which is required for the correct execution of concurrent statement that are executed at the same time.

Statements used in Dataflow Modelling

Two types of conditional assign statements are used to check conditions in data flow style:

- WHEN - ELSE conditional statement
- WITH - WHEN - ELSE conditional assignment

1) WHEN - ELSE or Conditional Signal Assignment Statement

It is a concurrent conditional selection statement which checks for condition and assigns a value to the output if condition is true, else next condition is checked and so on.

Syntax:

OUTPUT ← Val 1 WHEN [after time expression] condition ELSE
Val 2 WHEN [after time expression] condition 2 ELSE
Val 3 WHEN [after time expression] condition 3 ELSE
.
.
.
Val n WHEN [after time expression] condition ELSE
Val [after time expression] [WHEN condition];

eg. y ⇐ a WHEN sel = "00" ELSE
b WHEN sel = "01" ELSE
c WHEN sel = "10" ELSE
d;

2) With-When-Select or select signal Assignment Statement
It is a conditional selection statement
syntax: WITH parameter SELECT
Output ⇐ Val 1 WHEN condition 1,
Val 2 WHEN condition 2,
:
:
UNAFFECTED WHEN OTHERS;

eg. WITH sel SELECT
y ⇐ a WHEN sel = "00"
b WHEN sel = "01"
c WHEN sel = "10"
d WHEN OTHERS;

3) Block Statement
Block statement is used to group self-contained logical areas. Block statements perform the following functions:
(i) to disable some signals using guard expression
(ii) to limit scope of declaration
(iii) to group logical areas in the program to improve readability.
Syntax:
BLOCK LABEL : BLOCK [guard expression] IS
Block Header
Block Declarations
BEGIN
Concurrent statements
END BLOCK [Block Label];

eg. blk 1 : BLOCK
BEGIN
blk 2 : BLOCK
BEGIN
Concurrent statements ;
END BLOCK blk 2;
Concurrent statements;
END BLOCK blk 1;

Guarded Blocks : If guard expression is included in the block then it is called guarded block. Guard expression is a Boolean expression, which if true enables the drivers of statement in the block which are guarded and if false guarded statements are not activated.

Block Header : Block header contains port and generic declarations and their mappings which associate objects outside the block to objects and generic parameters inside the block.

Concurrent Assert Statement
It is used for exception handling.
Syntax :
ASSERT condition
[REPORT "Message"] (no semicolon)
[SEVERITY expression];

Generate : It allows a section of code to be repeated a number of times, thus creating several instances of same assignment

Syntax:

label : FOR identifier IN range GENERATE
(concurrent assignment)
END GENERATE;

eg. G1 : FOR i IN x RANGE GENERATE
z(i) <= x(i) AND y(i+8);
END GENERATE;

~~Behavioral Modelling~~

Multiple Drivers

When there is more than one assignment to the same signal, the signal has more than one driver and a mechanism is needed to compute the effective value of the signal. Effective value is determined by using resolution function.

Behavioral Modelling

Behaviour of the entity is expressed using sequentially executed, procedural type code.

1. Entity Declaration.

It describes external interface of entity.

Syntax:

entity entity-name is
[generic (list of generics and their types);]
[port (list of interface port-names and their types);]
[entity item declaration]
begin
entity statements
end [entity name];

Interface ports are signals through which entity passes.

Each interface port

Each interface port can have one of the following modes:

1. in: value of input port can only be read.
2. out: value of output port can only be updated.
3. inout: value of bidirectional port can be read and updated
4. buffer: value of buffer port can be read and updated

Architecture Body

It describes the internal view of the entity.

Syntax:

```
architecture architecture-name of entity-name is
    [architecture-item-declarations]
begin
    concurrent statements; these are →
        process statement
        block statement
        concurrent procedure call
        concurrent assertion statement
        concurrent signal assignment statement
        component instantiation statement
        generate statement
end [architecture-name];
```

Process statement

It contains sequential statements that describe the functionality of a portion of entity in sequential terms.

Syntax:

```
[process-label:] process [(sensitivity list)]
    [process-item declarations]
```

- begin
  sequential statements; these are →
  variable assignment statement
  signal assignment statement
  wait statement
  if statement
  case statement
  loop statement
  null statement ;
  exit statement
  next statement
  assertion statement
  procedure call statement
  return statement
  end process [process-label];

Variable Assignment Statement

A variable is assigned a value using variable assignment statement

Syntax: variable-object := expression;

eg. $V_1 := A - V_2$;

Signal Assignment Statement

Signals are assigned values using signal assignment statement

Syntax: signal-object ⇐ expression [after delay value];

eg. a ⇐ a XOR b after 12ns;

Wait statement

It provides an alternate way to suspend the execution of a process. There are 3 basic forms of wait statement:

wait on sensitivity-list;
wait until boolean-expression;
wait for time-expression;

If Statement

It selects a sequence of statement for execution based on value of a condition.

Syntax :

if bodean- expression then
sequential statement
[else if bodean expression then
sequential statement]
[else
sequential statement]
end if;

Case statement : It selects one of the branches for execution based on the value of expression.

Syntax:

case expression is
when choices ⇒ sequential statement
when choices ⇒ sequential statement
[when others ⇒ sequential statement].
end case;

Null Statement

It is a sequential statement that does not cause any action to take place and execution continues with the next statement.

Loop statement

It is used to iterate through a set of sequential statements.

Syntax:

[loop-label:] iteration-scheme loop
sequential statement
end loop [loop label];

Exit Statement

It causes the execution to jump out of innermost loop. Syntax:

exit [loop label] [when condition];

Next statement

It skips the remaining statements in current iteration of specified loop and execution resumes with the first statement in the next iteration of this loop.

Syntax: next [loop label] [when condition];

Assertion Statement:

If the value of boolean expression is false, report message is printed along with severity level.

Syntax:

```
assert boolean expression
[ report string - expression ]
[ severity expression ];
```

Structural Modeling

Component Declaration:

It declares the name and interface of a component. The interface specifies the mode and the type of ports.

Syntax:

```
component component - name
        port (list of interface ports);
end component;
```

eg.

```
component add
        port ( a,b : in bit ; c : out bit);
end component;
```

Component Instantiation

It defines a subcomponent of the entity in which it appears. It associates the signals in the entity with the ports of that subcomponent.

Syntax:

Component-label: component-name port map (association list);

Generic

A generic declares a constant object of mode "in" and can be used in the entity declaration and its corresponding architecture bodies. The value of this constant can be specified in one of the following:

- entity declaration
- component declaration
- component instantiation
- configuration specification
- configuration declaration

eg.

```
entity NAND_GATE is
  generic (M: INTEGER := 2);
port (A: in BIT_VECTOR (M downto 1);
      Z: out BIT);
  end NAND_GATE;
```

Configuration:

Configuration is used to

1) Specify multiple views of single entity.
2) Associate a component with any one of a set of entities

OR

Configuration is used to bind

1) Architecture body to its entity declaration
2) component with an entity.

Two ways of performing this binding are:

1) by using configuration specification
2) by using configuration declaration

Configuration Specification

It is used to bind component instantiation to specific entities that are stored in design libraries. The specification appears in declaration part of architecture or block in which components are instantiated.

Syntax:

```
for list-of-comp-labels : component-name
    use binding-indication;
```

eg.

```
for X1, X2 : XOR
    use entity WORK.XOR(DUMMY);
```

Configuration Declaration

It allow binding after the architecture body has been written. It is possible to have more than one configuration declaration for an entity.

Syntax of configuration declaration:

```
configuration configuration-name of entity-name is
    block-configuration
end [configuration-name];
```

Syntax of block configuration:

```
for block-name
    component configuration
    block configuration
end for;
```

eg.

```
configuration FA-CON of FULLADDER is
    for FA-STR
        use WORK.all;
        for A1, A2, A3 : AND
        use entity CMOS;
        end for;
    end for;
end FA-CON;
```