28/7/19

Unit-1

Introduction to VHDL

→ VHDL is an acronym for VHSIC hardware Description language.

→ It is a hardware description language that can be used to model a digital system at many levels of abstractions ranging from that of simple gate to completed digital electronic system, or anything in b/w.

→ The VHDL language is an integrated amalgamation of -

Sequential language +
concurrent language +
netlist language +
timing specifications +
waveform generation language = VHDL

→ (i) The language is used to model, ~~test &~~ simulate & test the digital system.

→ (ii) VHDL allows the description of concurrent language system. VHDL is a dataflow language rather than procedural language.

Scanned by CamScanner

(iii) It is a multipurpose language. Write once & use in many projects.

(iv) VHDL is portable which means project can be ported from VLSI etc.

(v) VHDL is a full type system.

(vi) It supports both synchronous & asynchronous model.

(vii) Test beches can be written.

Hardware Abstraction (Entity)

Internal view :- It specifies device's functionality & structures.

External view :- It specifies the interface through which it communicates with the other models in the environment.

Hardware Abstraction of a digital system is called an entity. One entity can be component of another entity.

Scanned by CamScanner

Design Units :-

VHDL provides 5 different types of primary constructs which are also called design units. These are :-

1) Entity declaration
2) Architecture body
3) Configuration declaration
4) Package declaration
5) Package body

An entity is modelled using an entity declaration and an at least one architecture body.

Brief notes :-

1) Entity declaration describes external view of the ~~body~~ entity.

Ex :- the input & output signal names.

2) Architecture body contains the internal description of the entity.

Ex :- As a set of interconnected components that represents the structure of the entity. or As a set of concurrent or sequential statements that represents the behaviour of the entity.

Scanned by CamScanner

DATE
PAGE

3) Configuration declaration is used to create a configuration for an entity.
It specifies the binding of one architecture body from the many architecture bodies, that may be associated with the ~~body~~ entity.
It also specifies no. of bindings

4) Package Declaration encapsulates (SUMMARIZE) a set of related declarations such as type declarations, subtype & subprogram declarations that can be shared across design units.

5) A package body contains the definitions of subprograms declared in package declaration.

note:- VHDL system has an analyzer and a simulator.

Analyzer compiles one or more design units contained in a single file and puts them in the design library after syntax validation.

The Simulator simulates an entity in VHDL in the following steps
1) Elaboration
2) Initialization
3) Simulation

Scanned by CamScanner

Architecture Modelling Styles :-

Architecture block defines how the entity operates.

① Structural Style of Modelling :-

In this style model is described as a set of interconnected components

Syntax :-

```
architecture identifier of entity_name is
    <declaration> ;
begin
    < statements>;
end architecture identifier;
```

Ex

```
architecture HA_STRUCTURE of HALF_ADDER is
    component XOR2
        port ( X, Y : In BIT ; Z : Out Bit);
    end component
    component AND2
        port ( A, B : In BIT ; N : Out Bit);
    end component ;
begin
    X1 : XOR2 port map (A, B, SUM);
    A1 : AND2 port map (A, B, CARRY);
end HA_STRUCTURE
```

component declaration

component instantiation statements

These are concurrent statements that means this order is not important

Scanned by CamScanner

② Dataflow Style of Modeling :-

In this modeling cycle, the flow of data through the entity is expressed primarily using concurrent signal assignment statements.

Ex :-

```
architecture HA_CONCURRENT of HALF_ADDER

begin
    SUM <= A XOR B after 8ns;
    CARRY <= B XOR B after 4ns;

end HA_CONCURRENT;
```

concurrent signal assignment

→ delay events

The symbol <= implies an assignment of value to the signal.

The signal which is assigned this value is called Target Signal.

Both statements here will run simultaneously.

Scanned by CamScanner

③ Behaviour Style of Modeling :-

This style of modeling specifies the behaviour / functionality of the entity.

Statements are executed in a sequential manner.

Here, statements use `<=' sequential signal assignments.

Ex

```
entity LS_DFF is
    port (Q: out BIT; D, CLK: in BIT);
end LS_DFF
architecture LS_DFF_BEH of LS_DFF is
begin
    process (D, CLK)
    begin
        if (CLK = '1') then
            Q <= D;
        end if
    end process;
end LS_DFF_BEH;
```

④ MIX Style of Modeling :-

All three styles can be mixed to form a single architectures.

Scanned by CamScanner

Data Objects :-

A data objects holds a value of a specified type. It is created by an object declaration. It has 4 classes.

① CONSTANT : An object of constant class can hold a single value of a given type.

It can be declared in all the declarative VHDL statements
→ sequential → Concurrent.

Syntax :- constant constant_name : constant_type := value;

This value is assigned to the constant before simulation starts and the value can't be changed during the course of the simulation.

Eg :- constant RISE_TIME : TIME := 10ns;
constant BUS_WIDTH : INTEGER := 8;
constant NO_OF_INPUTS : INTEGER := 3;

Scanned by CamScanner

(2) VARIABLES :- An object of variable class can hold a single value of a given type. In this case, different values can be assigned to the variables at different times using a variable assignment statement.

→ locally stores temporary data.
→ used only inside a sequential statement.
→ process, function & procedures.
→ visibility only inside the subprogram.
→ variable assignment : ":="

Syntax :- Variable_name : Variable_type;

Ex Variable CTRL_STATUS : BIT_VECTOR (10 downto 0);

Variable SUM : INTEGER range 0 to 100 := 10;
Variable FOUND, DONE : BOOLEAN;

(3) SIGNAL : An object belonging to the signal class holds a list of values which include the current values of the signal and a set of possible future value of the signal.

→ future values can be assigned to a signal using a signal assignment statements.

Scanned by CamScanner

→ Represents interconnection wires b/w ports

→ It may declared in declaration part of:

- packages
- architectores
- Entities
- Blocks

→ Signal assignment : <=

Syntax:- signal Signalname : Signal_type;

eg:- signal CLOCK : BIT;
signal DATA_BUS : BIT_VECTOR (0 to 7);
~~signal GATEREALTY~~
signal GATE_DELAY : TIME := 10 ns;

④ FILE:- An object belonging to a file class contains a sequence of values.
Values can be read or written to the file using read procedures & write procedures.

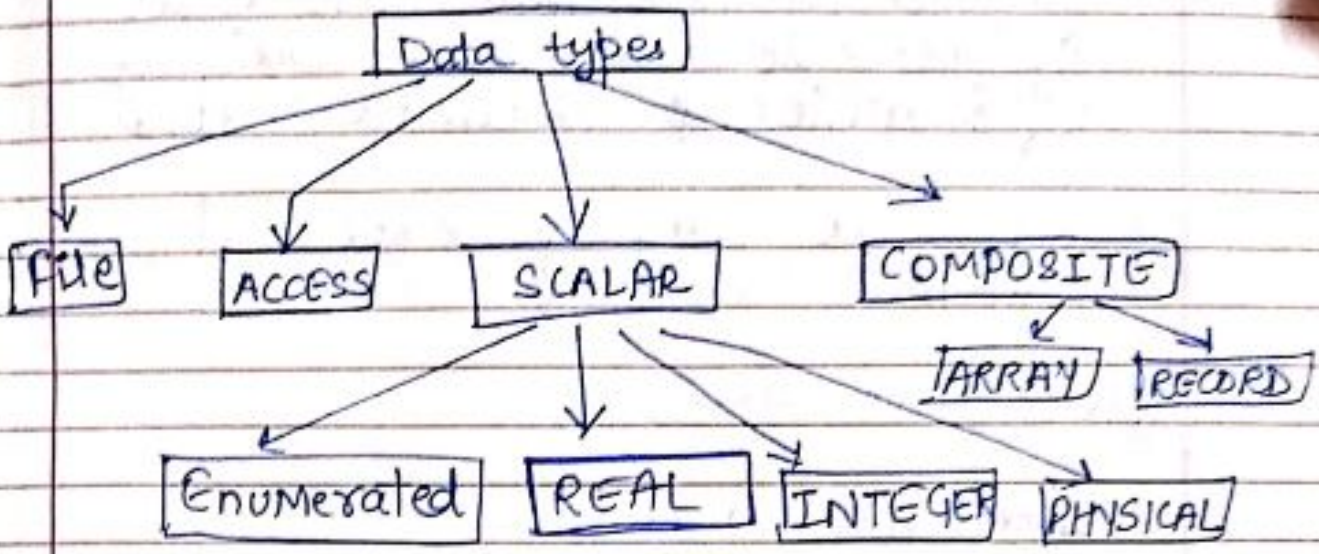
Syntax:-
File File_name : file_type name [openmode] is string-expression]

eg:-
FILE PAT1, PAT2 : STD_LOGIC_FILE;

Scanned by CamScanner

DATA TYPES

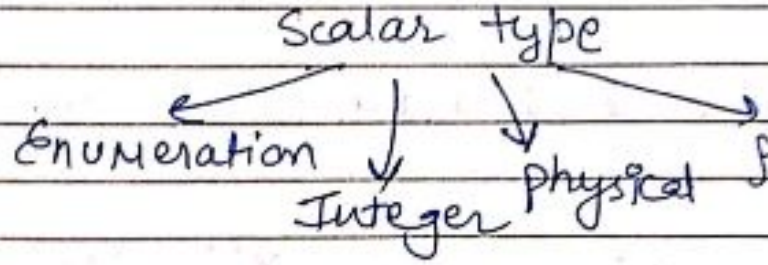
It represents a collection of values that the object can have and the set of operations that are allowed on it.



(1) SCALAR TYPE:

The value ~~appears~~ belonging to these types appear in sequential order.

- Ex :- BIT is a scalar type.



→ Integer, floating point & physical types are called numeric types.

Scanned by CamScanner

Enumerated & Integer are called discrete types.

a) Enumeration types

→ An enumeration type declaration defines a type that has a set of user-defined values consisting of identifier & character literals.

→ It is a very powerful tool for abstract modeling

literals (mistakes)

Identifier → name
character → 'X', '1' & '0'

'X' → An unknown value

'0' → false value

'1' → True value

'Z' → A tristate

Eg: Type colorIs (red, yellow, blue, green, orange);

type Four_Value is ('X' '1' '0' 'Z')

Scanned by CamScanner

(B) Integer types

→ An integer type defines a type whose set of values falls within a specified integer range.

→ Range :- $-(2^{31}-1)$ to $+(2^{31}-1)$

Ex:-
```
architecture Test of test_value is
begin
    process(x)
    variable a: INTEGER;
    variable b: int_type;
begin
    a := 1;      -- ok
    a := -1;     -- ok
    a := 1.0;    -- error
end process;
end test;
```

(C) Floating point types / Real types :-

→ It has a set of values in a given range of real numbers.

Ex
```
type TTL_VOL is range -5.5 to -1.4;
type REAL_DATA is range 0.0 to 31.9;
```

→ Range :- -1.0E38 to +1.0E38

→ Floating point literals differs from integer literals by presence of dot (.) character. Thus, 0 is an integer literal and 0.0 is an floating pt. literal.

Scanned by CamScanner

DATE
PAGE

⑥ Physical type :-

i) A physical type contains values that represent measurement of some physical quantities like time, length, voltage or current.

ii) It provides for a base unit & successive unit i.e., the smallest unit represent one base unit & largest is determined by range specified in declarations.

iii) Range :- -10 STEP to +10 STEP

iv) physical literals are written as an integer or a floating point literal followed by the unit name.

eg:- type STEP_TYPE is range -10 to +10 units
units

```
STEP ;                    --base unit
STEP 2 = 2 STEP;          --derived unit
STEP 5 = 5 STEP;          --derived units
end units ;
```


Scanned by CamScanner

② Composite type :-

→ Composite types represents a collection of values.

→ These are composed of elements of a single type.

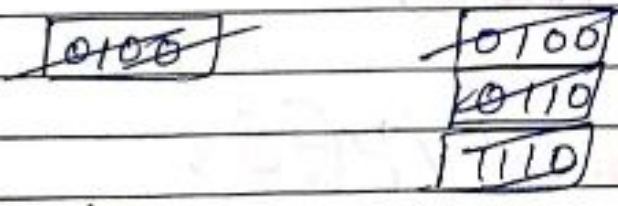
It has two types

Array type Record type

a) Array type

→ Arrays are collections of objects of the same types.

→ They can be 1D, 2D, 1D x 1D. They can also be of higher dimensions by then they are generally not synthesizable.



Syntax :-

type type-name is array (specification) of data-types;

Eg type Data is array (7 down to 0) of STD_LOGIC;

not string literal ("____") bit string

Scanned by CamScanner

b) Record type :-

→ An object of record type is composed of elements of same or different types.

→ It is analogous to record data type in pascal and the struct declaration in C.

→ Syntax :- type name is ~~record~~
record
identifier: subtype_indicati
identifier: subtype_indicatio
end record;

Eg:-
type PIN_TYPE is range 0 to
type MODULE is
record
size : INTEGER range 2 to 10;
NO_OUTPUTS: PIN_TYPE;
end record;

(3) ACCESS TYPES :-

→ In access type, the values belonging to pointers to a dynamically located object of some other type.

Scanned by CamScanner

It provides :-

→ Analogous to pointer in other language
→ Allows for dynamic allocation of storage.
→ useful for implement, queries, FIFO etc.

eg :- type PTR is access MODULE;
type FIFO is array (0 to 63, 0 to 7) of BIT;

(4) FILE TYPES :-

→ It provides access to objects that contains a sequence of values of given type.
→ It represents files in host environment.
→ It provide a mechanism by which a VHDL design communicate with the host environment.

Syntax :- type file_type_name is file of type name ;

eg :- type VECTORS is file of BIT_VECTOR;
type NAMES is ~~file~~ file of STRING ;

Scanned by CamScanner

Some predefined data types :-

① BIT :- the BIT data type can only have the value 0 or 1.

② BIT_VECTOR :- It is the vector version of BIT type consisting of two or more bits.

③ STD_LOGIC :- The STD_LOGIC data type can have value X, 0, 1, Z and these values are not synthesizable.

④ STD_LOGIC_VECTOR :- The vector version of the STD_LOGIC ~~type~~ data type. Each bits that make up the vector can have the values X, 0, 1, Z.

OPERATORS :-

① LOGICAL :- the 7 logical operators are and, or, nand, nor, xnor, not, xor.
→ These operators are defined for the predefined type BIT & BOOLEAN.
→ They are also defined for 1-D arrays of BIT & BOOLEAN.

Scanned by CamScanner

→ during evaluation BIT values '0' & '1' treated as False & True values of Boolean.

eg Y <= A nand B ; - - illegal

Y <= not (A and B); - - legal.

② Relational

→ these operator relates b/w two variables, these are :-

=, /=, <, <=, >, >=

→ The result of all relational operators is always the predefined type BOOLEAN.

eg:- BIT_VECTOR ('0','1','1') < BIT_VECTOR('1','0',

→ The (=) equality & the (/=) inequality operators are predefined for any type except file types.

Table: operator	operand	Result type
=	Any type	Boolean
/=	"	"
&		
|	"	"

Scanned by CamScanner

(3) Shift Operator :-

The shift operator shifts the bits in left or right.

There are, srl, sll, sra, sla, rol, ror

However, these operators never worked correctly & were removed from the language.

Each of the operator takes an array of BIT or BOOLEAN.

a) SLL (shift left logical)

let L => 11001001

(4) ADDING OPERATOR :-

These are +, −, &. The operands for the +, &, - operators must be of the same numeric type & result being of same numeric type.

These operator may also be used as an array operators.

Ex:- operand for the '&' → concatenation
'0' & '1' Result => "01"
'C' & 'A' & 'T' ⇒ Result => "CAT"

Scanned by CamScanner

(3) Multiplying Operators :-

→ These are '*', '/', mod, rem
→ '*' & '/' should be applied for same type of operands of same integer or floating point.
→ The result of physical type.

If A rem B = A - (A/B) * B
A mod B = A - B * N
- - for N integers

ex 7 mod 4. -- has value 3
(-7) rem 4 = -3
7 mod (-4) = -1
(-7) rem (-4) = -3

same sign
-7 rem 4 := -3

```
4 | 7 | 1
  -4
  ---
   3
```

Miscellaneous Operators :-

abs **

→ The absolute (abs) operator is defined for any numeric type.
→ The ** (exponentiation) operator is defined for the left operand to be integer or floating point type.

Scanned by CamScanner

& for right operand to be integer type.

→ The NOT operator has same for bit or boolean type.

Styles of Modelling

There are 4 types of modelling

① Behavioral Modelling

→ Behavioral modeling specifies the behaviour of an entity as a set of statements that are executed sequentially in the specified order.

→ Sequential code is called behavioral code.

→ It contains process statements, Seq statements, variable and signal assignment statement, wait, case, if-else & assertion statements.

→ VHDL code is inherently concurrent process, function & procedure are

Scanned by CamScanner

the only sections of code that are executed sequentially. However, as a whole, any of these blocks is still concurrent with any other statements placed outside it.

→ Behavioral style modeling required tooth table of system design.

→ It consists gate level abstraction.

(2) Data flow Modeling :-

→ In this modeling style, the flow of data through the entity is expressed primarily using concurrent signal assignment statement.

→ concurrent statement or code is called as data flow code.

→ It contains concurrent statement, conditional & selected signal assignment block statement.

→ It represent behavior and view of data flow as flowing a design from i/p to o/p.

Scanned by CamScanner

→ It requires boolean expression for design.

→ It consists of gate level abstraction or algorithm.

3) Mixed Style of Modeling :-

→ In this modeling, we mix any of the three modeling styles.

→ within an architecture body, use component instantiation statements, concurrent signal assignment & process statement (that represent behaviour).

Types of DELAY :-

1) Delta Delay :-

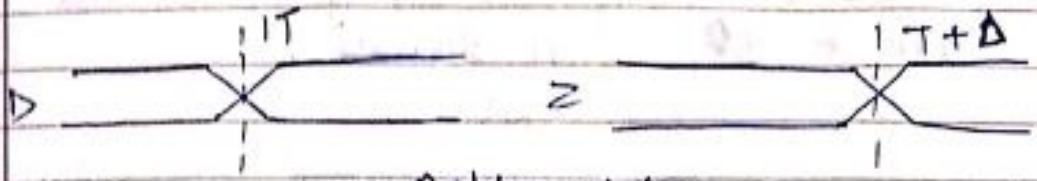
→ A delta delay is a very small delay (infinitesimally small)

→ It does not correspond to any real delay and actual simulation time does not advances.

→ This delay models hardware where a minimal amount of time is needed for a change to occur.

Scanned by CamScanner

→ In delta delay, there is no delay or default delay.



Delta delay

Signal z gets its value only at time T+Δ

2) INERTIAL DELAY :-

→ Inertial delay models the delay often found in switching ckts.

→ It represents the time for which an input value must be stable before value is allowed to propogate to the o/p.

→ It is the default delay in VHDL

→ No keyword need be explicitly specified.

→ If no pulse rejection limit is specified, the default pulse rejection limit is the inertial delay itself.

Syntax :-

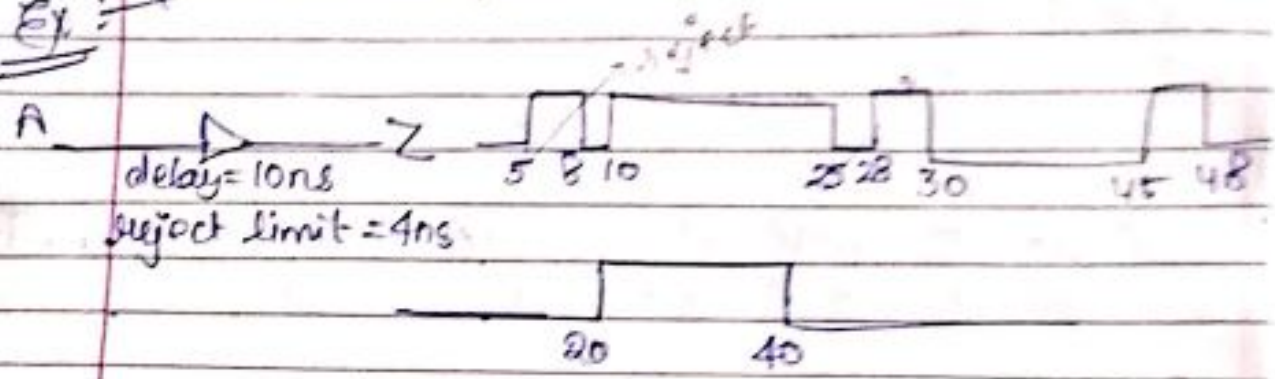
→ Signal_object <= [reject pulse_rejection_limit] inertial] expression after inertial_delay_value ;

Scanned by CamScanner

Z <= reject 4ns inertial A after 10ns;

It is used to filter any unwanted spikes & transients on signals.

Ex:-



(3) Transport Delay :-

→ Transport delay models the delay in hardware that do not exhibit any inertial delay.

→ This delay represents pure propogation delay that is any change in i/p will be transmit to the o/p no matter how small after specified delay.

→ It is not default in VHDL.

→ Use keyword transport.

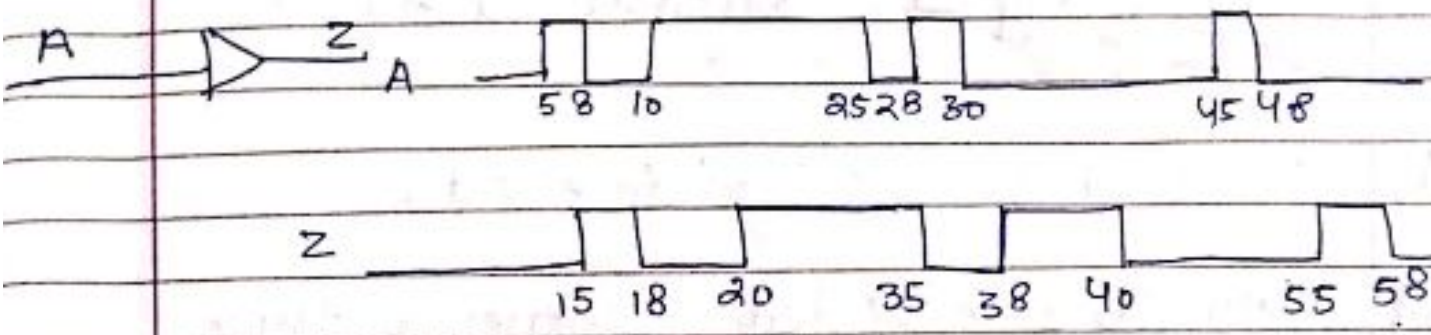
→ Use in Routing delays.

Scanned by CamScanner

Syntax :-

transport expression after transport_delay_value;

eg. : z <= transport A after 10 ns;



Concurrent & Sequential Statements

Concurrent statements	Sequential Statements
1) All statements inside an architecture block are concurrent block.	1) These statements are inside a process block.
2) They can appear outside the process block	2) Can appear only inside a process block.
3) Simple signal assignment statement.	3) Sequential variable assignment statement.
4) Ex - process, component instance, concurrent signal assignment.	4) Ex - if, for, switch-case, signal assignment.

Scanned by CamScanner

Concurrent Signal Assignment Statement :-

i) These signal assignment statements can appear outside a process block.

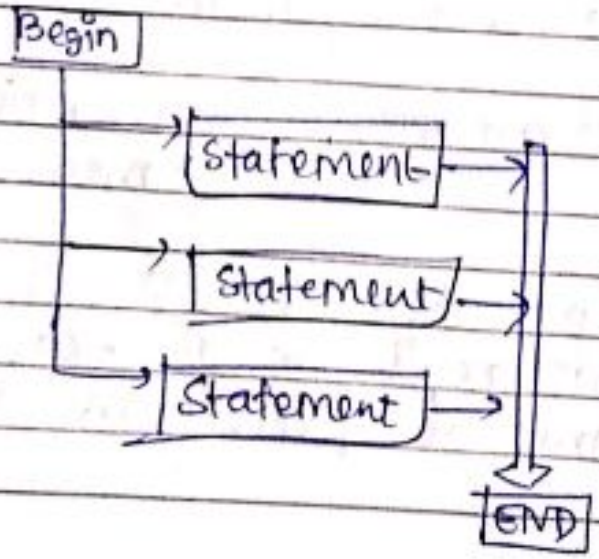
ii) These are event-triggered.

iii) They are executed whenever there is an event on a signal that appear in its expression.

iv) These are independent & asynchronous

architecture CON_SIG of FRAG is
begin

```
A <= B ;
Z <= A ;
```



Scanned by CamScanner

* Sequential Signal Assignment statement :- (SSAS)

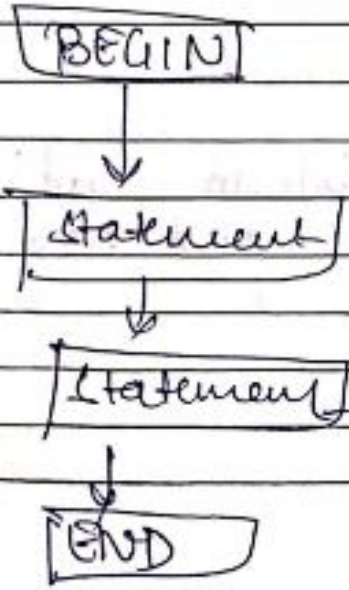
→ Appears within process block.

→ SSAS are not event triggered.

→ They are executed in sequence in relation to other sequential statement that appear with in process.

→ Seq. statements are order dependent & synchronous.

eg :-
```
architecture SEQ_SIG of FRAG is
  begin
   Process (B)
   begin
    A <= B ;        -- A, B & Z are signals
    Z <= A ;
   end SEQ_SIG ;
```



Scanned by CamScanner

Signal Drivers :-

→ A signal driver is a collection of value time pairs refer to as transaction.

→ Every concurrent statement that assigns to a signal creates a driver for that signal.

→ Only one driver is allowed for a signal unless it is a resolved signal

→ Initial value of the driver is taken from the default value of the declaration which is visible to the source process.

Q:-

```
architecture arch 4 of nand is
signal sel, A, B : std_logic;
signal Y : std_logic;
begin
    Y <= not (A and B) and sel;
    Y <= not (A or B) and not sel;
end arch 4;
```


Scanned by CamScanner

⇒ Resolution function :- [where more than 1 assignment]

→ Where there is more than one assignment to the source signal or the signal has more than one driver and a mechanism is needed to compute the effective value of the signal called RF.

→ RF consider the value of both the driver for Z & determine the effective value.

→ the value of each driver is an i/p to the RF & based on the computational performed with in the RF the value returned by this function becomes the resolved value for the signal

Conditional Signal Assignment Statement :-

(i) It is a concurrent signal assign statement

(ii) It selects different values for target signal based on the specified, possibly different conditions.

(iii) It's like an If Statement.

Scanned by CamScanner

(iv) Syntax :-

target_signal <= [waveform_elements when condition else
[" " "
. . .

waveform_element [when condition];

eg:- 4X1 MUX

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

entity MUX is
    port(A,B,C,D: in std_logic_vector(3 down to 0);
    S0,S1 : in std_logic;
    Y: out std_logic_vector(3 down to 0);
end MUX;

architecture behavioral of MUX is
    waveform_element
z <= 'A after 5ns when S0='0' and S1='0' else
     B " " " S0='0' and S1='1' else
     C " " " S0='1' and S1='0' else
     D when others;
end behavioral;
```

* - for unaffected values.

unaffected when others;

Scanned by CamScanner

Selected Signal Assignment

Statement :-

① It is also a concurrent signal assignment statement.

② It selects different values for a targetted signal based on the value of a selection expression

③ It is like a Case statement.

Syntax :-

with expression select

target signal <= waveform_element when choices;
" ";
...
waveform_element when choices;

Eg :-

So, S1 → sel : std_logic_vector (1 downto 0);
architecture behavioral of MUX16
with sel select
Z <= A when "00"
B " "01"
C " "10"
D when others;
end behavioral;

Scanned by CamScanner

Block Statement :-

① A block statement is a concurrent statement.

② It can be used for major purposes;

→ To disable signal drivers by using Guard
→ To limit scope of declarations, including signal declarations.

→ To represent a portion of a design

③ The block statement itself has no execution semantics.

④ It has 2 types :-

① Simple ② ~~Guard~~ Guarded
↳ Boolean type
↳ keyword GUARD

Syntax :-

```
block_label : block [(guard expression)] [is]
    [block_header]
    [block_declarations]
  begin
    concurrent_statement
  end block [block_label];
```


Scanned by CamScanner

ex:-

```
B1 : block [ STROBE = 'J' )
begin
    Z <= guarded not A;
end block B1;
```

* Structural Modelling :-

(1) Components declaration

→ A component declaration declares the name and the interface of a component.

→ The interface specifies the mode & the type of ports.

Syntax:-

```
component component_name is
    port (list_of_interface_ports);
end component [component-name];
```

ex:

```
component NAND2
port (A, B: in std_logic;
      Z: out std_logic;)
end component;
```


Scanned by CamScanner

Component Instantiation:

→ A component instantiation defines a sub-component of the entity in which it appears.

→ It associates the signal in the entity with the ports of that subcomponent.

→ A Format of comp. Instantiation is:

component_label : comp_name [port map (association)];

Eg:

-- component declaration:

component NAND2
port (A, B : in std_logic;
Z : out std_logic);
end component;

-- component instantiation:

N1 : NAND2 port map (S1, S2, S3);

Scanned by CamScanner

Association List :-

→ It associates signals in the entity called actuals with the ports of a component called formals.

→ An actual may be a signal & also be the keyword open to indicate a port that is not connected.

→ Two ways to perform the association :-

1) positional 2) Named

Positional Association :-

actual1, actual2, ... actualn

Each actual in component instantiation is mapped by position.

eg ⇒ port map(s1, s2, s3);

Informal

⇒ formal1 ⇒ actual1, formal2 = actual2 ....

eg

~~N1 : NAND2~~

N1 : NOR 2 port map(DB ⇒ MR, DZ = RDY, DA ⇒ S1);

Scanned by CamScanner

Generics :-

(1) They are declared in entity.

(2) Any change in the value of a generic affects all architecture with that entity.

(3) A generic declares a constant object with in_mode. The value of object must be specified at once.

(4) It declared above with its ports in the entity declaration.

(5) Generic information is static it can't be change during simulation

Syntax:-

generic (generic_interface_list);

ex:-

```
entity AND is
generic (N : NATURAL);
port (A : IN BIT_VECTOR (1 to N)
      Z : OUT BIT);
end AND;
architecture GENERIC of AND is
begin
```


Scanned by CamScanner

```
process(A)
    variable AND_OUT : BIT;
begin
process(A)
    AND_OUT = '1';
for k in 1 to N loop
AND_OUT := AND_OUT and A(k);
exit when AND_OUT = '0';
end loop;
Z <= AND_OUT;
end process
end GENERIC;
```

CONSTANT :-

① Constants are specified in ~~structure~~ architecture

② Hence any change in the value of a constant will be localised to the specific architecture only.

③ Syntax :-

```
constant constant_name : type := Value;
```


Scanned by CamScanner

Subprogram :-

A Subprogram defines a sequential algorithm that performs a certain computation.

1) Functions :- These are usually used for computing a single value.

A function executes in zero simulation time.

(default class in Constructor

→ functions are a part of a group of structures called subprograms.

2) Procedure :-

→ procedures are part of a group of structures called subprograms.
→ procedures are small sections of code that perform an operation that is reused throughout your code.
→ This serves to cleanup code as well as allow for reusability.
→ Procedures can take inputs & generate outputs. They can generally be more complicated than functions.
→ It is not required to pass any signals to a procedure.

Scanned by CamScanner

Syntax of subprogram :-

subprogram_specification is
subprogram_item_declarations
begin
subprogram_statements
end [function/procedure] [subprogram_name];

Procedure

→ Procedure allows decomposition of large behaviours into modular sections.

→ It can rather zero or more values using parameters of mode out & input.

Syntax of Procedure :-

procedure procedure_name (parameter_list)

Eg. procedure Arith (A, B: in INTEGER;
OP: in OP_CODE;
Z: out INTEGER; Zcomp out BOOLEAN);
begin
Case OP is
when ADD => Z := A+B;
" SUB => Z := A-B;
" MUL => Z := A*B;
" DIV => Z := A/B;
" LT => Zcomp := A<B;
" LE => Zcomp := A<=B;
end Case;
end Arith;

Scanned by CamScanner

Procedure Call :-

A procedure call can be either sequential or concurrent statement, based on where the actual procedure call statement is present.

Syntax :-

[label :] procedure_name (list_of_actual);

FUNCTIONS :-

Functions are small set of code that perform operation that is reused throughout the code.

- Pure function
- Impure function

→ In always use a return statement

→ wait statements cannot be used in a function.

① Pure functions

→ A pure fn is one that return the same value each time the function is called with some set of actual.

→ Keyword is pure

→ It is default function.

Scanned by CamScanner

Syntax :-

[pure/Impure] function function_name (parameter_list)
return return_type ;

eg:- pure function RISE (signal Clock : BIT)
return BOOLEAN is
begin
return Clock = '1' and clock 'EVENT;
end RISE;

② IMPURE FUNCTION :-

→ It is one that potentially returns different values each time, the function is called with some set of actual.

→ It is not default.

→ Keyword - impure.

Syntax :-

impure function_name (parameter - list)
return return_type ;

Scanned by CamScanner

eg:-

```
Impure function RANDOM(seed: REAL) return
  REAL is

Variable NUM: REAL
attribute FOREIGN of RANDOM: function
is "NUM = rand(seed)";
begin
return NUM;
end RANDOM;
```

Procedure Call:-

Subprogram Overloading :-

Sometimes it is convenient to have 2 or more subprograms with same name. In such a program the subprogram name is said to be overloaded.

eg:

```
function COUNT(orange: INTEGER)
  return INTEGER;

function COUNT(apple: INTEGER)
  return INTEGER;
```


Scanned by CamScanner

Both function aye overloaded since they have same name, COUNT.

Ex:- COUNT(20) since 20 is integer
COUNT('1') type bit.

→ A call to an overloaded subprogram is ambigous and hence an error if it is not possible to identify the exact subprogram being called using the following

a) subprogram name
b) No of actuals
c) Types and order of actuals
d) formals
e) Result type

Operator overloading :-

1) It is one of the most useful feature in the language.

2) When ~~an operator symbol~~ a standard operator symbol is made to behave differently based on the types of its operands, the operator is said to be overloaded.

Scanned by CamScanner

Eg type MVL is ('U', '0', '1', 'Z');
function "and" (L, R: MVL) return MVL;
function "or" (L, R: MVL) return MVL;
function "not" (R: MVL) return MVL;

The need for operator overloading arises from the fact that the predefined operators in the language are only defined for only for operands of certain predefined types.

Signatures :-

1) To overcome the problem of overloading the signatures are used.

2) Signatures distinguishes b/w overloaded subprograms and enumeration literals based on their parameter & result type profiles.

3) A signature may be used in an attribute name, entity declaration.

SYNTAX :-

[first_parameter_type, second_parameter_type]
[return function_return_type]

Scanned by CamScanner

Library :-

library IEEE ;
The IEEE 1164 std (multivalue logic system for VHDL model interoperability) is a technical standards published by IEEE.

It describes logic values to be used in electronic design automation for VHDL.

Packages :-

① use ieee.std_logic_1164.all ;

It contains definitions of types, subtypes and functions which extend the VHDL into a MVL.

It has logic values such as :
Z :- ~~Use IEEE~~ ie high impedance value
X :- unknown value
1 :- strong one
0 :- strong zero.
U :- uninitialised
W :- weak unknown logic
L :- weak zero
H :- weak one
- :- don't care

Scanned by CamScanner

(2) Use IEEE : std_logic_Arith.all ;
(3) Use IEEE:std_logic_signed.all ;
(4) Use ieee.std_logic_unsigned.all ;

Advanced features :-

1) Generate statements

Concurrent statements can be ~~condit~~ conditionally selected or replicated during the elaborations phase using the generate statements. There are two forms of generate statements.

a) Using the for-generation scheme

concurrent statements can be replicated on predetermined no. of times.

b) Using the if-generation scheme, concurrent statement can be conditionally elaborated.

Scanned by CamScanner

(1) For-generation scheme:

Syntax:-

generate_label: for generate_identifier in discrete_range generate
[block_declarations
begin]
concurrent_statements
end generate [generate_label];

→ the values in the discrete range must be globally static i.e., they must be computable at elaboration time.

→ These statements can also use the generate_identifier in their expression.

Ex. 4-bit Full Adder

```
library IEEE;
use ieee.std_logic_1164.all;
use ieee: std_logic_ARITH.all;

entity FULL_ADDER4 is
port (A, B : in BIT_VECTOR(3 down to 0); Cin: in BIT;
      S : out BIT_VECTOR(3 down to 0); Cout : out BIT);

end FULL_ADDER4;
```


Scanned by CamScanner

DATE
PAGE

architecture for_generate of FULL_ADDER is

component full_adder

port (PA, PB, PC : in BIT;
pCoub, PS : out BIT);

end component;

Signal Car : BIT_Vector (4 down to 0);
begin

Car(0) <= Cin;

Generate Statement → Gk : for K in down to 0 generate

FA : full_adder port map (Car(K), A(K), B(K), Car(K+1), S(K));

end generate Gk;
Cout <= Car(4);
end for_generate;

(2) IF-Generation scheme

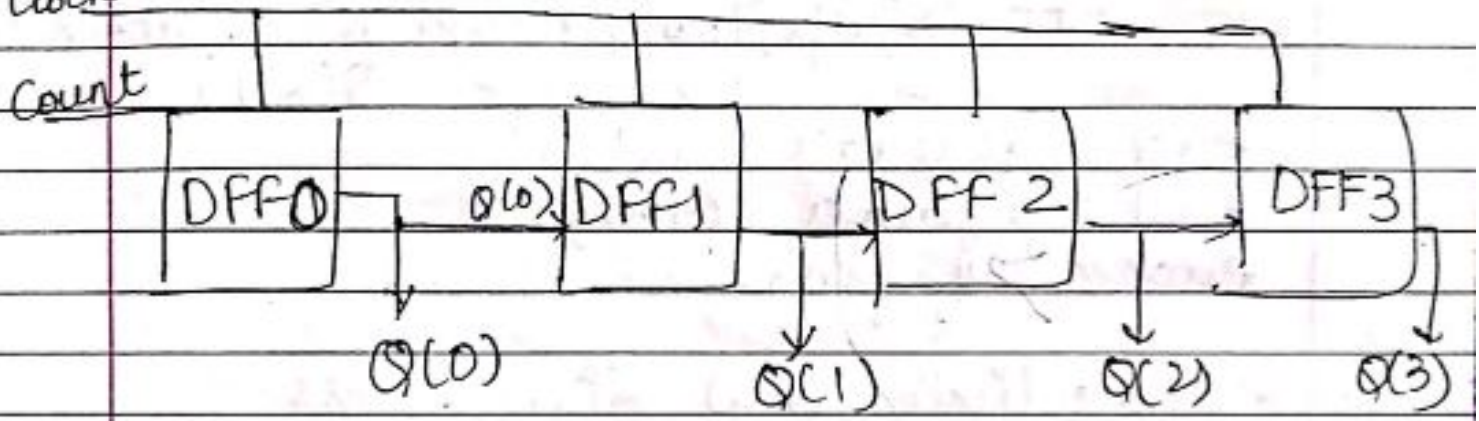
Syntax: generate_label : if expression generate
[block declarations
begin]
concurrent_statements
end generate [generate_label];

Scanned by CamScanner

→ If-generate system allow for conditional section of concurrent statement.

→ the expression must be globally static expression, that is value must be computable at ~~eve~~ elaboration time

Code for 4-bit Counter




```
entity counter4 is
    port ( count, clock : in BIT;
           Q : buffer BIT_VECTOR (0 to 3));
end counter 4;

architecture g-gen of counter 4 is
component D_flip_flop
    port (D, Clk : in BIT;
          Qu0 : out BIT);
```


Scanned by CamScanner

```
end component;
begin
    G1k : for k in 0 to 3 generate

    G1k0 : if k=0 generate
    DFF : D_flip_flop port map (count,
                                clock, Q(k));

end generate G1k0;

    G1k1_3 : if k>0 generate
    DFF : D_flip_flop port map (Q(k-1), Clock(k)
                                Q(k));
end generate G1k1_3;
end generate G1k;
end if_gen;
```

3)

4)

5)

Q) Differentiate b/w Signal ass. stt. and Variable ass. stt.

Ans:

Variable	Signal
1) A variable used in a process P must be declared inside the process	1) A signal used in a process must be declared outside or inside the process.
2) A variable is only visible in the process in which it is declared	2) A signal is visible to all process in the architecture in which it is declared.

6)

7)

8)

9)

Scanned by CamScanner

	Variable	Signal
3)	when a variable assigns to a value, it takes the value immediately.	3) When a signal is assigned to a value in a process, the signal doesn't take new value until after the process has suspended.
4)	Do not retain their value b/w executions of subprogram in which they are declared.	4) Have current & projected values.
5)	Require to reduce memory and provide fast simulation.	5) Require large complicated data structure in the simulator to maintain their data queues.
6)	Variable assignment operator ":="	6) Signal assignment operator "<="
7)	Rep's local info.	7) Rep's circuit interconnect (wires)
8)	Need less memory	8) Need more memory
9)	update immediately	9) update after delta delay.

Scanned by CamScanner

If - statement :-

→ An if statement selects a sequence of st... for execution based on the value of a condition.

→ The condition can be any exp that evaluates to a boolean va...

Syntax :-

```
if boolean_expression then
    sequential statement;
{ elsif boolean_expression then
    sequential statements }
[ else
    sequential statement ]
end if;
```

Eg:

①
```
if sum <= 100 then
    sum := sum + 10;
end if
```

②
```
if (day = 'Sunday') then
    weekend := TRUE;
elsif (day = 'Saturday') then
    weekend := TRUE;
else weekday := TRUE;
end if;
```


Scanned by CamScanner

Case Statement - [control statement]

The case statement selects one of the branches for execution based on the value of the expression.

The expression value must be of a discrete type or of a 1D array type.

Syntax :- Case expression is
when choices => seq-stmt;
when choices => seq-stmt;
when others => seq-stmt;
end case;

eg 4X1 mux

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

entity MUX is
port ( A, B, C, D : in std_logic;
       S : in std_logic_vector (0 to 1);
       Z : out std_logic);
end MUX;

architecture MUX_beh of MUX is
constant MUX_delay : TIME := 10ns;
begin
   PMUX : process (A, B, C, D, S)
variable TEMP : STD_logic;
```


Scanned by CamScanner

```
begin
    case S is

when "00" => TEMP := A;
when "01" => TEMP := B;
when "10" => TEMP := C;
when "11" => TEMP := D;
when others => TEMP := 'X';

end case;

Z <= TEMP after MUX_DELAY;
end process PMUX;
end MUX_BEH;
```

NULL Statement :-

A null statement performs no action. It's only function is to pass control on to the next statement.

The statement

```
null;
```

Loop Statement :-

It is used to iterate through a set of sequential statements

Scanned by CamScanner

Syntax :-

① [label:] for identifier in range loop
seq-stmt;
end loop [label];

② while condition loop
sequential_stmt;
end loop;

eg Factorial := 1;
For number in 2 to n loop
factorial := factorial * number;
end loop;

Exit Statement

The exit stmt is a seq stmt that can be use only inside a loop. It causes to jump out of the innermost loop or the loop whose label is specified.

Syntax :- exit [loop_label] [when condition];

eg Sum := 1; J = 0;

Scanned by CamScanner

```
<<L3>> loop
J := J + 21;
sum := sum * 10;
if sum > 100 then
   exit L3
end if;
end loop L3;
```

NEXT STATEMENT :-

→ The next stmt is also a seq. stmt that can be used only inside a loop.

→ The keyword next is used

Syntax :-

next [loop_label] [when condition];

ex For k in 10 down to 5 loop

```
if sum < total_sum then
   sum := sum + 2;
elsif sum = total_sum then
next;
else;
null;
end if;
k := k + 1;
end loop
```


Scanned by CamScanner

ASSERTION Statement :-

1) It is useful in modelling constraints of an entity.
2) The ASSERT statement is v. useful for reporting textual strings to the designer.
3) This stmt. checks the value of a boolean expression for True or False.
4) If the value is true, the stmt does nothing but if the value is false the assert stmt outputs a user specified text string to the std output to the terminal

Syntax :-

```
assert boolean_expression
  [report string_expression]
  [severity expression];
```

Eg

```
if EVENT then
  assert NOW = 0ns or
  (NOW - CLOCK) >= HOLD TIME;
  report "HOLD TIME Too Short!!!"
  Severity failure;
    on clk := NOW;
  end if;
```


Scanned by CamScanner

Report Statement :-

A report stmt. can be used to display a statement a message. It is similar to an assertion statement, but w/o the assertion check.

Syntax :-

report String_expression
[severity expression];

Eg:

```
if EVENT then
assert NOW = 0 ns or
(NOW - check) >= HOLD_TIME;
report "Hold time too short!";
Severity failure
on clk := NOW;
end if;
```

Eg.

```
if clk /= '0' and clk /= '1' then
report "clk is neither a '0' nor a '1';
severity error;
end if;
```


Scanned by CamScanner

7/9/19

Unit-2

Implementation of following circuits :-

1) first adder 2) Subtractor 3) decoder 4) Encoder 5) mux 6) ALU 7) Barrel Shifter 8) 4X4 key board encoder 9) multiplier 10) divider 11) Hamming code Encoder and correction ckt.

Adders :-

① Half Adder using behavioral modelling code : [using Truth Table]

→

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

entity half_adder is
port ( A, B : in std_logic;
       sum, cout : out std_logic );
end half_adder;

Architecture beh of half_adder is
begin
if (A='0' and B='0') then
    sum <= '0';
    Coub <= '0';
```


Scanned by CamScanner

```
elsif (A='0' and B='1') then
    Sum <='1';
    Cout <='1';

elsif (A='1' and B='0') then
    Sum <='1';
    Cout <='0';

elsif (A='1' and B='1') then
    Sum <='0';
    Cout <='1';

end if;
end process;
end BEH;
```

[using charact. eqn]

* Dataflow code :- [parallely modelling]

```
Architecture DATA of half_adder is
begin
    sum <= A xor B;
    Cout <= A and B;
end DATA;
```

-- sum = $\bar{A}B + A\bar{B}$ Cout = AB

* Structural code :- [using ckt diagram]

```
architecture STRUC of Half_adder is
component XOR_2
port (A,B: in std_logic;
      C: out std_logic);
end xor_2;
```


Scanned by CamScanner

component AND_2
port (l, m : in std_logic;
n : out std_logic);
end component;

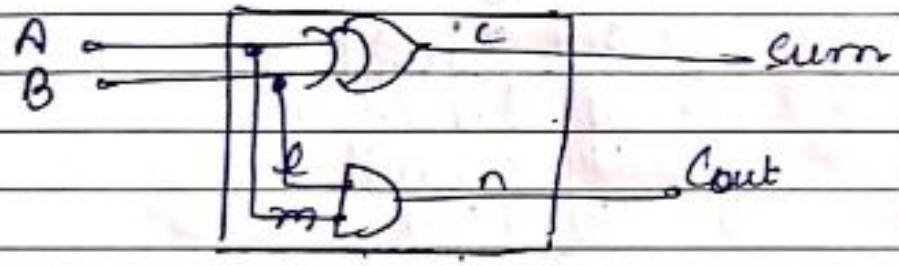
-- Component instantiation :-

Positional association (i/p, i/p, o/p)

begin
X1 : XOR_2 port map (A, B, SUM);
A1 : AND_2 port map (A => l, B => m, Cout => n);

named association

end struc;



② FULL ADDER :-

Algorithm :- Design a circuit which will take 2 input along with previous carry to produce a sum & carry.

Scanned by CamScanner

* Truth table :-

A	B	Cin	S	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

* Characteristic equation :-

Cout

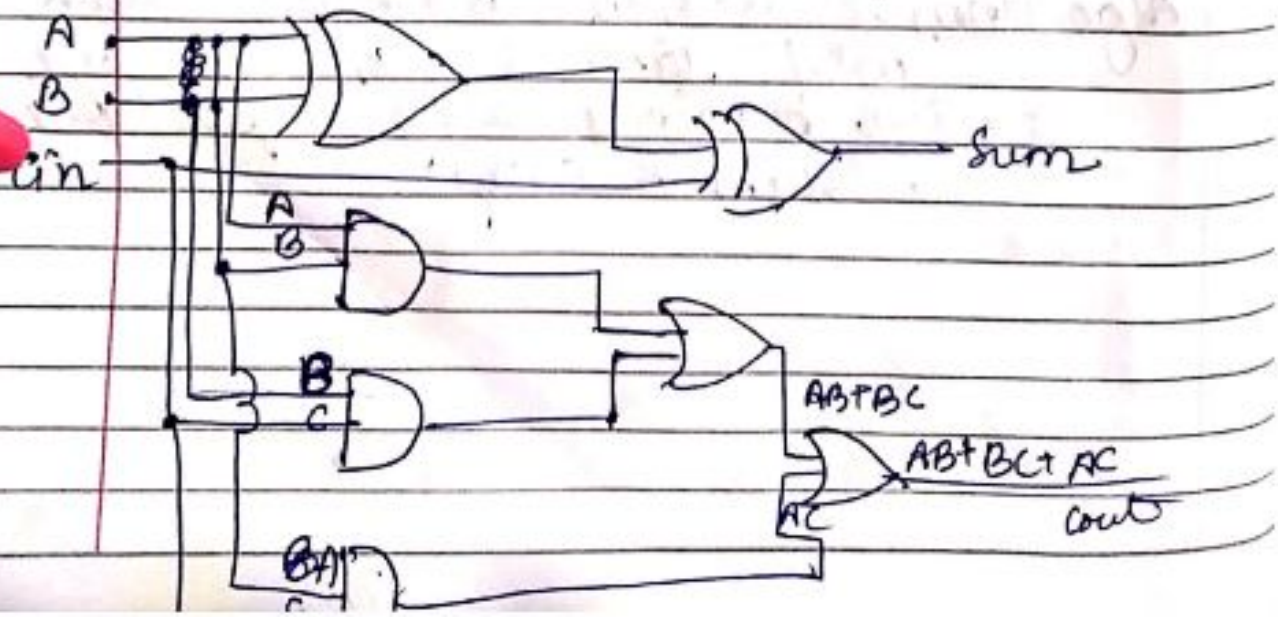
AB \ Cin	0	1
00	0	0
01	0	1
11	1	1
10	0	1

Sum

AB \ Cin	0	1
00	0	1
01	1	0
11	0	1
10	1	0

Cout = BC + AC + AB Sum = A ⊕ B ⊕ C

* Circuit diagram :-



Scanned by CamScanner

Code :- (Behavioral)

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

entity full_adder is
 port (A,B,Cin: IN std_logic;
       S,Cout : OUT std_logic);

 end FULL_ADDER;

architecture BEH of full_adder is
 begin
 process (A,B)
  begin
 if (a='0' and B='0' and C='0') then
    sum <='0';
    cout <='0';
 elsif (a='0' and B='0' and C='1') then
    sum <='1';
    cout <='0';
 elsif (a='0' and B='1' and c='0') then
    sum <='1';
    cout <='0';
    elsif (a='0' and B='1' and C='1') then
    sum <='0';
    cout <='1';
    elsif (a='1' and B='0' and C='0') then
    sum <='1';
    cout <='0';
    elsif (a='1' and B='0' and C='1') then
       sum <='0';
       cout <='1';
```


Scanned by CamScanner

elsif (a='1' and B='1' and c='1') then
sum <='1';
cout <='1';
end if;
end process;

end beh;

* Dataflow :-

~~sum <= a and b;~~
~~end data;~~

Architecture data of full-adder is
begin
sum <= A xor B xor C;
cout <=(A and B) or (B and C) or
(A and C);
end Data;

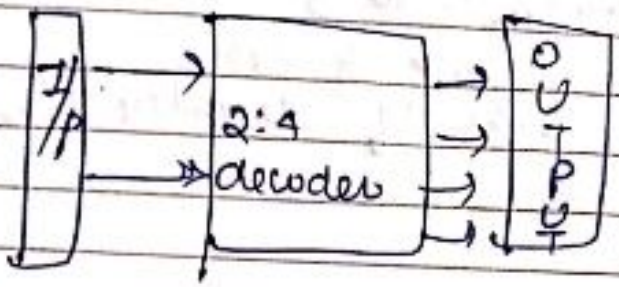
* Structural :-

Scanned by CamScanner

4. 2:4 decoder using behavioral :-

Algorithm :-

Block diagram :-



Truth Table :-

Input		Output
A	B	Y
0	0	Y_0 = 0001
0	1	Y_1 = 0010
1	0	Y_2 = 0100
1	1	Y_3 = 1000

Boolean eq^n :-

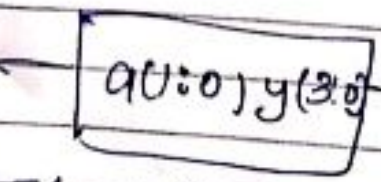
$Y[0] = A'B'$
$Y[1] = A'B$
$Y[2] = AB'$
$Y[3] = AB$

Scanned by CamScanner

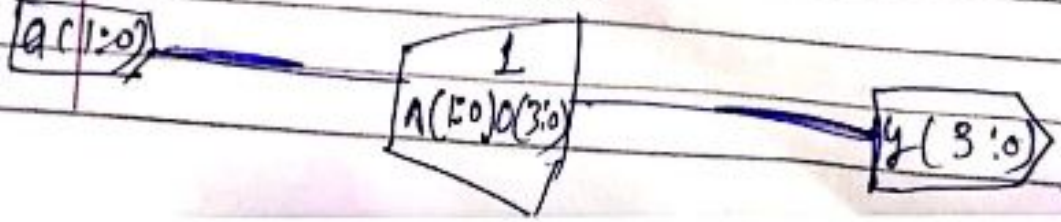
DATE
PAGE

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

entity decoder24_beh is
port (a: in std_logic_vector(1 downto 0);
      y: out "          (3 downto 0));

end decoder24_beh;

architecture beh of decoder24_beh is
begin
process (a)
begin
case a is
  when "00" => y <= "0001";
  when "01" => y <= "0010";
  when "10" => y <= "0100";
  when "11" => y <= "1000";
  when others => y <= "0000";
end case;
end process;
end beh;
```

→ Entity declaration box.



→ RTL view



Scanned by CamScanner

(Y0, Y1, Y2, Y3: out std_logic);

6

DATE
PAGE

* Using dataflow code;

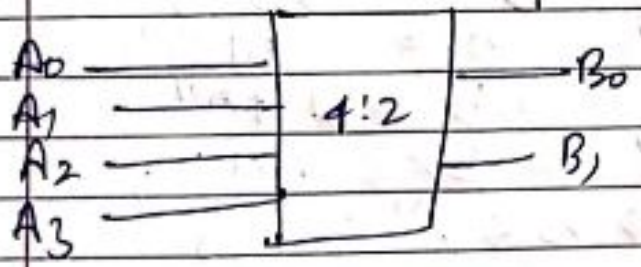
architecture data of decoder24 is

begin

Y0 <= ((not A) and (not B));
Y1 <= ((not A) and B);
Y2 <= (A and (not B));
Y3 <= (A and B);

end data;

Encoder 4:2 :-



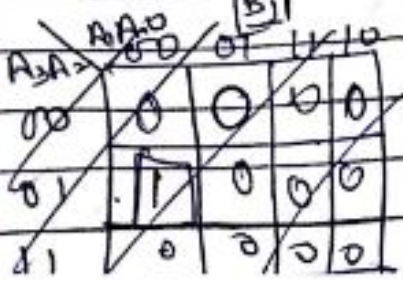
$B_0 = \overline{A_1}A_0(\overline{A_2}\,\overline{A_3} + A_3\overline{A_2})$

Truth table :-

I/P				O/P	
A_3	A_2	A_1	A_0	B_1	B_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

$B_1 = A_2 + A_1$

$B_0 = A_1 + A_3$.

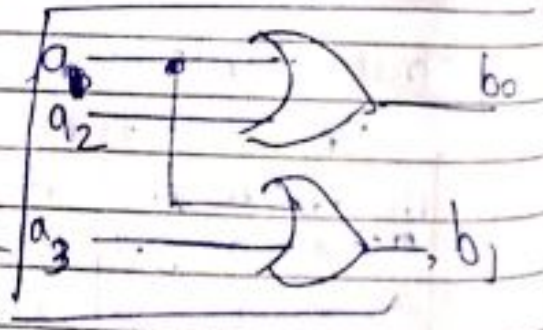


$B_1 = \overline{A_3}A_2\overline{A_1}\overline{A_0} + A_3A_2\overline{A_1}$

Scanned by CamScanner

61 chrome page.

```
library ieee;                    use ieee.all;
use ieee.std_logic_
entity encoder is
port ( a : in std_logic_vector(3 downto
                                0);
       b: out std_logic_vector(1 downto 0));
end encoder;

architecture beh of encoder is
begin
process (a)
begin
case a is
                 -- A3A2A1A0
when "1000" => b<= "00";
when "0100" => b<= "01";
when "0010" => b<= "10";
when "0001" => b<= "11";
when others => b<="ZZ";
end case;
end process;
end beh;
```



* Using logic gates :-

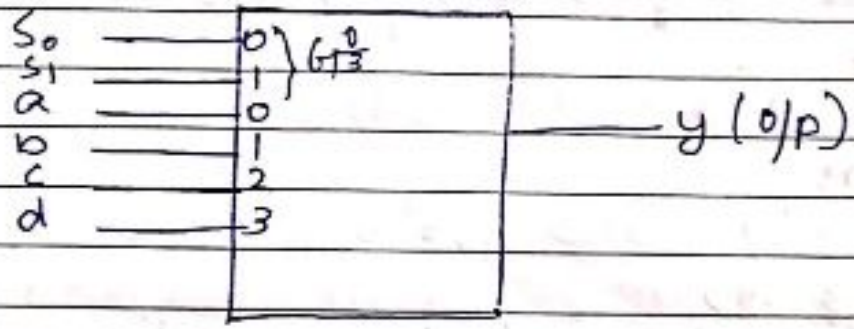
```
architecture — of encoder is
begin
b(0)<= a(1) or a(2)
b(1) <= a(1) or a(3)

end —;
```


Scanned by CamScanner

4:1 MUX :-

A mux can be used to switch one of many i/p to a single o/p. Typically mux are used to allow large, complex pieces of hardware to be reused.

The IEEE symbol of 4:1 mux is,



where G is a select symbol, 0/3 is not a fraction but it means 0 - 3. ∴ the binary i/p on the top 2 inputs is used to select one of the inputs 0 - 3.

* Truth Table :-

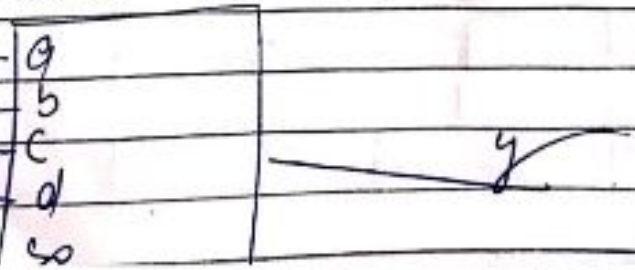
i/p	S_1	S_0	o/p
A	0	0	A
B	0	1	B
C	1	0	C
D	1	1	D

Scanned by CamScanner

* using beh. modelling :-

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

entity mux is
port ( a,b,c,d,s0,s1 : in std_logic;
       y: out std_logic);

end mux;

architecture beh of mux is
begin
process(a,b,c,d,s0,s1)
 begin
if (s1='0' and s0='0') then
  y<= a;
elsif (s1='0' and s0='1') then
   y<= b;
elsif (s1='1' and s0='0') then
   y<= c;
elsif (s1='1' and s0='1') then
   y<= d;
else
   y<= z;
end if;
 end process;
 end beh;
```

* Entity declaration box

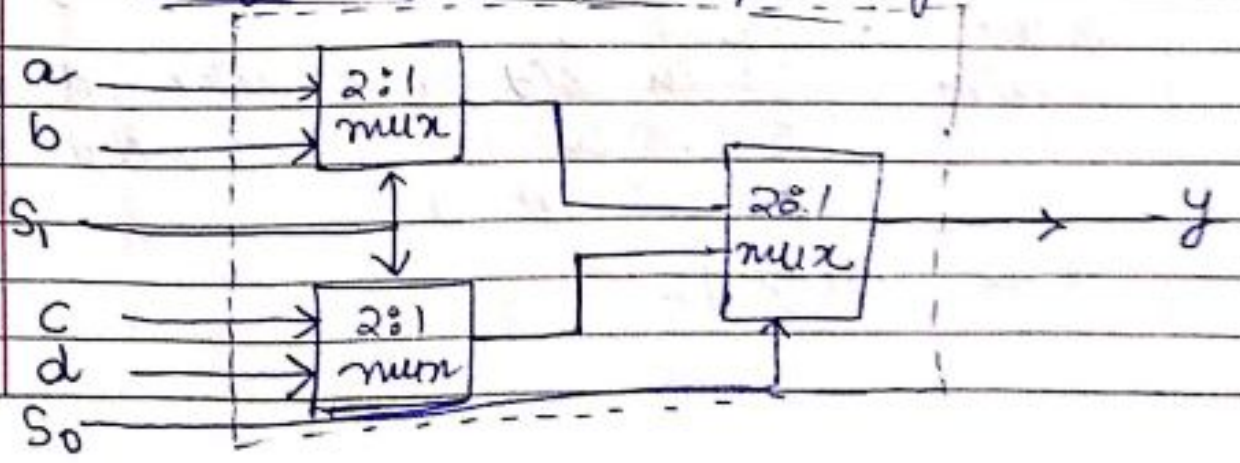


Scanned by CamScanner

* Using case statement;

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

entity mux is
port ( a,b,c,d : in std_logic;
       sel : in std_logic (1 down to 0);
       y : out std_logic );

end mux;

architecture casebeh of mux is
begin
process (a,b,c,d, sel)
begin
  case sel is
     when "00" => y<=a;
     when "01" => y<=b;
     when "10" => y<=c;
     when "11" => y<=d;
  end case;
end process;
end casebeh;
```

* Using Structural modelling :-



Scanned by CamScanner

architecture struc of mux is

component mux21 is
port (a : in std_logic;
b : in std_logic;
s : in std_logic;
y : out std_logic);

end component;

signal temp1, temp2 : std_logic;
begin

To test0 : mux21 port map (a=>a, b=>b, s=>s1, y=>temp
T1 test1 : mux21 port map (a=>c, b=>d, s=>s, y=>
T2 test2 : mux21 port map (a=>temp1, b=> temp2,
s=> s0, y=>y);

end ~~behav~~ struc;

8X1 MUX :-

library ieee;
use ieee.std_logic_1164.all;
entity mux81 is
port (i : in std_logic_vector(7 downto0);
Sel : in (2 downto0);
Zout : out);
end mux81;

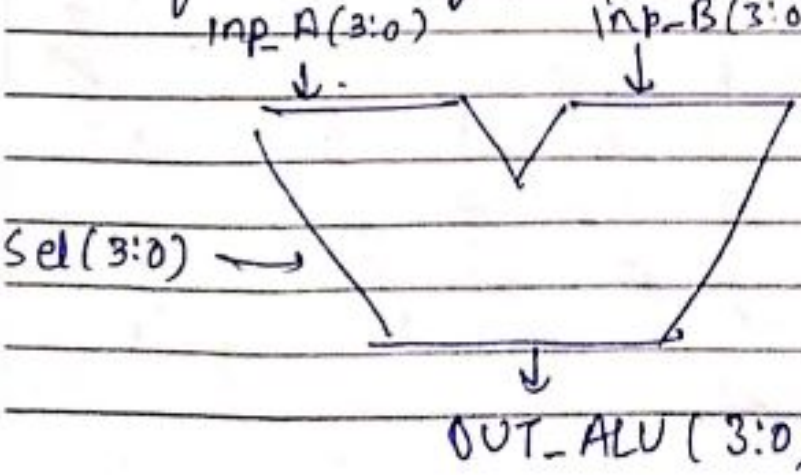
Scanned by CamScanner

```
architecture Bimux of muxBI is
begin
zout <= I(0) when sel="000" else
        I(1) when sel="001" else
        I(2) when sel="010" else
        I(3) when sel="011" else
        I(4) when sel="100" else
        I(5) when sel="101" else
        I(6) when sel="110" else
        I(7); ~~when sel="111" else~~
end BI_mux;
```

ALU :-

ALU's comprise the combinational logic that implements logic operations such as AND, OR, NOT gate and arithmetic operations such as Adder, Subtractor.

functionally, operation of typical ALU is,



Scanned by CamScanner

Selection input			operation performed
S_2	S_1	S_0	
0	0	0	A + B
0	0	1	A - B
0	1	0	A - 1
0	1	1	A + 1
1	0	0	A and B
1	0	1	A or B
1	1	0	not A
1	1	1	A xor B

* Behavioral code :

note : std_logic_arith is not an official ieee supported package file & should not be used in digital designs, instead of this ~~use~~ numeric_std should be used

A signal that is defined as type signed means that the tools interpret this signal to be either +ve or -ve.

A signal that is defined as type unsigned means the signal will be only +ve.

Scanned by CamScanner

(4-bit)

* VHDL code for ALU is

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

entity ALU is

port ( a:   in signed (3 downto 0);
       b:   in signed (3 downto 0);
       sel: in std_logic_vector (2 downto 0);
       outp: out signed (3 down to 0));

end ALU;

architecture beh of alu is
begin
    process (a, b, sel)
        begin
        case sel is
        when "000" => outp <= a+b;
        when "001" => outp <= a-b;
        when "010" => outp <= a-1;
        when "011" => outp <= a+1;
        when "100" => outp <= a and b;
        when "101" => outp <= a or b;
        when "110" => outp <= not a;
        when "111" => outp <= a xor b;
        when Others => NULL;
        end case;
    end process;
end beh;
```

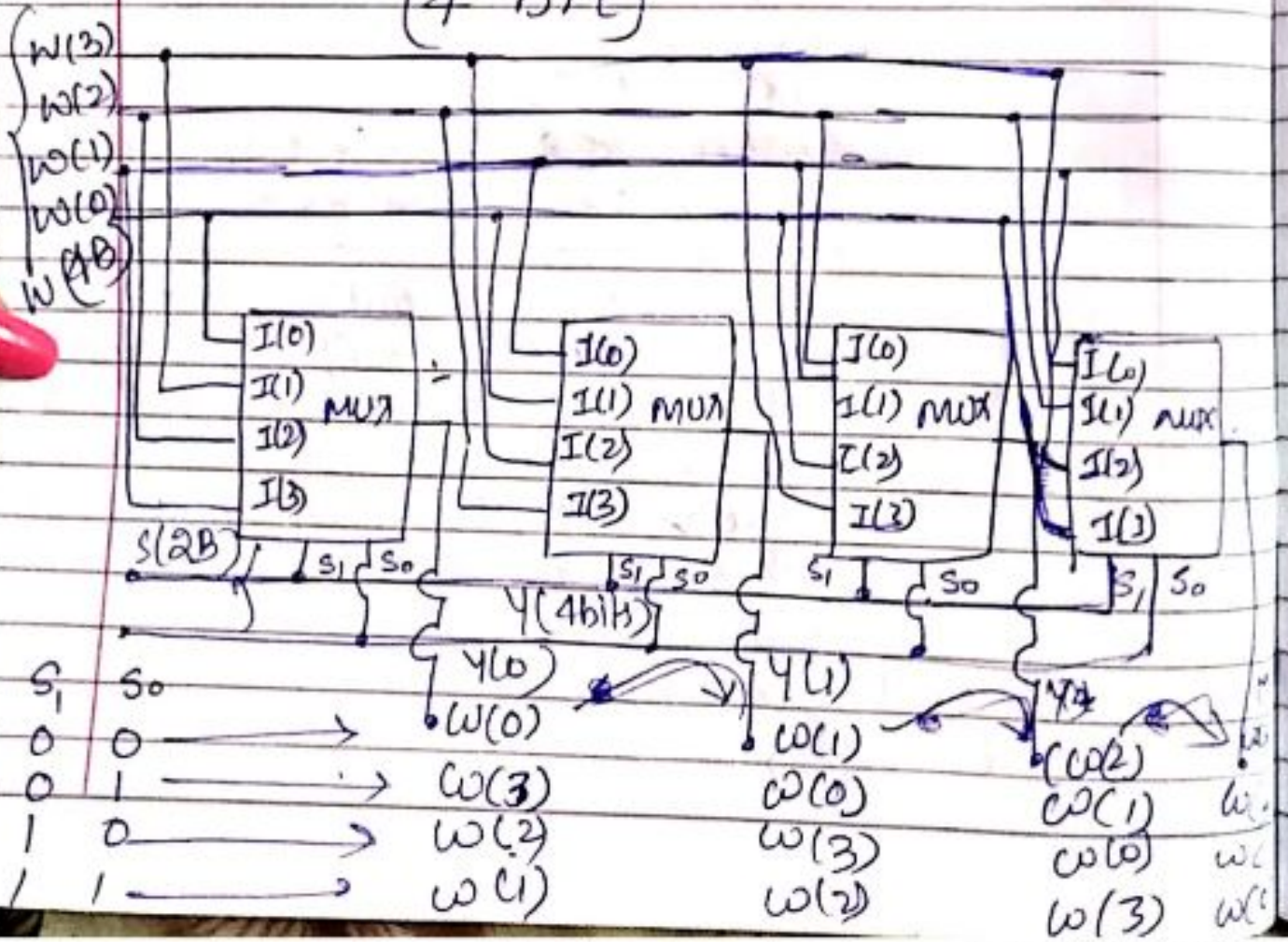

Scanned by CamScanner

Barrel Shifter:

A Barrel shifter shifts (actually rotates) the i/p data by the specified number of bits in a combinational manner.
It takes parallel data input and give shifted o/p either in left or right direction by a specific shift amount.
When shift by i/p is "0000", it will pass i/p data at the o/p w/o shifting.

for specifying shifting direction, shift_lft pin is used. When it is '0' the block will perform left shift opeyation & when it is '1', it will perform right opeyation.

(4-bit)



Scanned by CamScanner

~~unsigned~~ at the place of unsigned ~~std~~ std_logic_vector can also be used

* Code :-

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

entity bs4b is

port ( w : in unsigned (3 downto 0);
       s : in unsigned (1 downto 0);
       y : out unsigned (3 downto 0));

end bs4b ;

architecture beh of bs4b is
begin
  process(s,w)
  begin
    case s is
      when "00" => y <= w ;
      when "01" => y <= w ror 1;
      when "10" => y <= w ror 2;
      when others => y <= w ror 3;

    end case;
  end process;
    end beh;
```


Scanned by CamScanner

4X4 Bit Multiplier

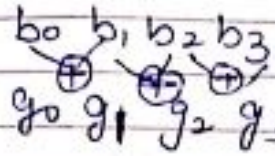
$A_3\ A_2\ A_1\ A_0$
$\times\ B_3\ B_2\ B_1\ B_0$
Result (5 downto 0)

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

entity BM44 is
port ( N1, N2 : in std_logic_vector(3 downto
                                    0);
       Result : out std_logic_vector(7 downto
end entity BM44;

architecture beh of BM44 is
begin
Result <= std_logic_vector (unsigned (N1) *
                            unsigned (N2));
end architecture beh;
```


Scanned by CamScanner

Converter :-

1) Binary to Gray



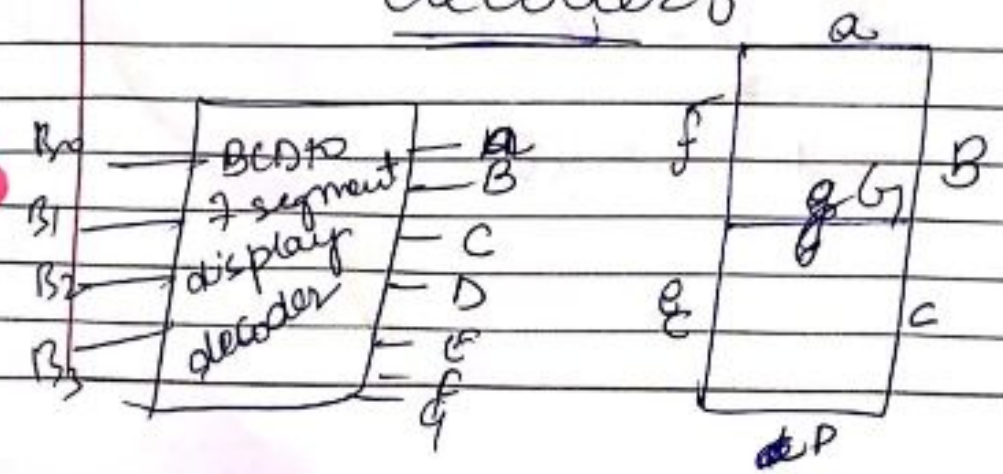

```
library ieee;
use ieee.std_logic_1164.all;

entity bin_gray is
port (
        b0, b1, b2, b3 : in std_logic;
        g0, g1, g2, g3 : out std_logic);

end bin_gray;

architecture dataflow of bin_gray is

begin

g0 <= b0;
g1 <= b0 xor b1;
g2 <= b1 xor b2;
g3 <= b2 xor b3;

end dataflow;
```


Scanned by CamScanner

2) Gray to Binary :-

```
library ieee;
use ieee.std_logic_1164.all;
entity Gray_binary is
port ( G0, G1, G2, G3 : in std_logic;
       B0, B1, B2, B3 : out std_logic);
end Gray_binary;
architecture beh of Gray_binary is
begin
B0 <= G0;
B1 <= B0 XOR G1;
B2 <= B1 XOR G2;
B3 <= B2 XOR G1;
end beh;
```

7 Segment display decoder :-



Scanned by CamScanner