VHDL

INTRODUCTION

- The VHSIC Hardware Description Language (VHDL) is an industry standard language used to describe hardware from the abstract to concrete level.

- The language not only defines the syntax but also defines very clear simulation semantics for each language construct.

- It is strong typed language and is often verbose to write.

- Provides extensive range of modeling capabilities,it is possible to quickly assimilate a core subset of the language that is both easy and simple to understand without learning the more complex features.

Why Use VHDL?

- Quick Time-to-Market
  - Allows designers to quickly develop designs requiring tens of thousands of logic gates
  - Provides powerful high-level constructs for describing complex logic
  - Supports modular design methodology and multiple levels of hierarchy
- One language for design and simulation
- Allows creation of device-independent designs that are portable to multiple vendors. Good for ASIC Migration
- Allows user to pick any synthesis tool, vendor, or device

BASIC FEATURES OF VHDL

- CONCURRENCY.
- SUPPORTS SEQUENTIAL STATEMENTS.
- SUPPORTS FOR TEST & SIMULATION.
- STRONGLY TYPED LANGUAGE.
- SUPPORTS HIERARCHIES.
- SUPPORTS FOR VENDOR DEFINED LIBRARIES.
- SUPPORTS MULTIVALUED LOGIC.

CONCURRENCY

- VHDL is a concurrent language.
- HDL differs with Software languages with respect to Concurrency only.
- VHDL executes statements at the same time in parallel,as in Hardware.

SUPPORTS SEQUENTIAL STATEMENTS

- VHDL supports sequential statements also, it executes one statement at a time in sequence only.

- As the case with any conventional languages.

example:

```
if a=‘1’ then
      y<=‘0’;
else
      y<=‘1’;
end if ;
```

SUPPORTS FOR TEST & SIMULATION.

- To ensure that design is correct as per the specifications, the designer has to write another program known as "TEST BENCH".
- It generates a set of test vectors and sends them to the design under test(DUT).
- Also gives the responses made by the DUT against a specifications for correct results to ensure the functionality.

STRONGLY TYPED LANGUAGE

- VHDL allows LHS & RHS operators of same type.
- Different types in LHS & RHS is illegal in VHDL.
- Allows different type assignment by conversion.

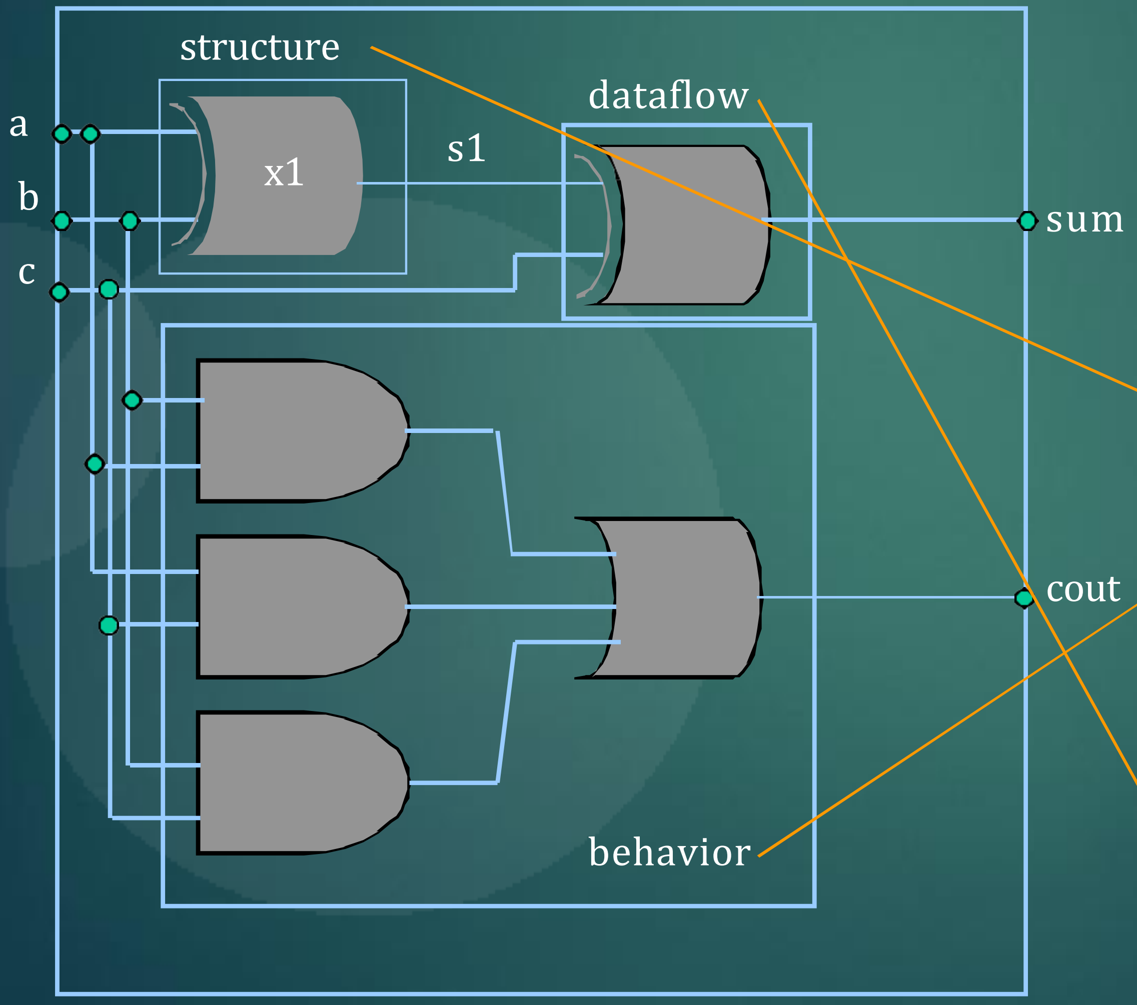
example:

```
A : in std_logic_vector(3 downto 0).
B : out std_logic_vector(3 downto 0).
C : in bit_vector(3 downto 0).
B <=A;     --perfect.
B <= C;    --type miss match,syntax error.
```

LEVELS OF ABSTRACTION

- Data Flow level
  - In this style of modeling the flow of data through the entity is expressed using concurrent signal assignment statements.
- Structural level
  - In this style of modeling the entity is described as a set of interconnected statements.
- Behavioral level.
  - This style of modeling specifies the behavior of an entity as a set of statements that are executed sequentially in the specified order.

EXAMPLE SHOWING ABSTRACTION LEVELS



```
entity full_adder is
   port(a,b,c:in bit;sum,cout:out bit);
end full_adder;
architecture fulladd_mix of full_adder is
component xor2
   port(p1,p2:in bit; pz:out bit);
end component;
signal s1:bit;
begin
  x1:xor2 port map(a,b,s1);
  process(a,b,c)
     variable  t1,t2,t3:bit;
  begin
     t1:=a and b;
     t2:=b and cin;
     t3:=a and cin;
     cout <= t1 or t2 or t3;
  end process
  sum <= s1 xor cin;
end fulladd_mix;
```

VHDL IDENTIFIERS

- Identifiers are used to name items in a VHDL model.
- A basic identifier may contain only capital ‘A’ - ’Z’ , ‘a’ - ’z’, ‘0’ - ’9’, underscore character ‘_’
- Must start with a alphabet.
- May not end with a underscore character.
- Must not include two successive underscore characters.
- Reserved word cannot be used as identifiers.
- VHDL is not case sensitive.

OBJECTS

- There are three basic object types in VHDL
  - Signal : represents interconnections that connect components and ports.
  - Variable : used for local storage within a process.
  - Constant : a fixed value.
- The object type could be a scalar or an array.

DATA TYPES IN VHDL

- Type
  - Is a name which is associated with a set of values and a set of operations.
- Major types:
  - Scalar Types
  - Composite Types

SCALAR TYPES

- Integer

  Maximum range of integer is tool dependent

  type integer is range implementation_defined

  constant loop_no : integer := 345;

  Signal my_int : integer range 0 to 255;

- Floating point

  - Can be either positive or negative.
  - exponents have to be integer.

  type real is range implementation_defined

SCALAR TYPES (Cont..)

- Physical

  Predefined type “Time” used to specify delays.

  Example :

  type TIME is range -2147483647 to 2147483647

- Enumeration

  Values are defined in ascending order.

  Example:

  type alu is (pass, add, subtract, multiply,divide)

COMPOSITE TYPES

- There are two composite types
- ARRAY :
  - Contain many elements of the same type.
  - Array can be either single or multidimensional.
  - Single dimensional array are synthesizable.
  - The synthesis of multidimensional array depends upon the synthesizer being used.
- RECORD :Contain elements of different types.

THE STD_LOGIC TYPE

- It is a data type defined in the std_logic_1164 package of IEEE library.
- It is an enumerated type and is defined as

```
type std_logic is (‘U’, ‘X’, ‘0’, ‘1’, ‘Z’, ‘W’, ‘L’, ‘H’,’-’)
```

‘u’	unspecified
‘x’	unknown
‘0’	strong zero
‘1’	strong one
‘z’	high impedance
‘w’	weak unknown
‘l’	weak zero
‘h’	weak one
‘-’	don’t care

SIGNAL ARRAY

- A set of signals may also be declared as a signal array which is a concatenated set of signals.
- This is done by defining the signal of type bit_vector or std_logic_vector.
- bit_vector and std_logic_vector are types defined in the ieee.std_logic_1164 package.
- Signal array is declared as : <type>(<range>)

  Example:

```
signal data1:bit_vector(1 downto 0)
signal data2: std_logic_vector(7 down to 0);
signal address : std_logic_vector(0 to 15);
```

SUBTYPE

- It is a type with a constraint

- Useful for range checking and for imposing additional constraints on types.

- syntax:
  - subtype subtype_nameis base_type rangerange_constraint;

- example:
  - subtype DIGITS is integer range 0 to 9;

MULTI-DIMENSIONAL ARRAYS

Syntax

```
type array_name is array (index_ range , index_range) of element_type;
```

example:

```
type memory is array (3 downto 0, 7 downto 0);
```

- For synthesizers which do not accept multidimensional arrays,one can declare two uni- dimensional arrays.

example:

```
type byte is array (7 downto 0) of std_logic;
type mem is array (3 downto 0)of byte;
```

OPERATORS

precedence	Operator class	Operators					
Low	Logical	and	or	nand	nor	Xor	xnor
	Relational	=	/=	<	<=	>	>=
	Shift	sll	srl	sla	sra	rol	ror
	Add	+	-	&			
	Sign	+	-				
	Multiply	*	/	mod	rem		
	Miscellaneous	**	abs	not			
High							

Data Flow Modeling

DATAFLOW LEVEL

- A Dataflow model specifies the functionality of the entity without explicitly specifying its structure.

- This functionality shows the flow of information through the entity, which is expressed primarily using concurrent signal assignment statements and block statements.

- The primary mechanism for modeling the dataflow behavior of an entity is using the concurrent signal assignment statement.

ENTITY

- Entity describes the design interface.
- The interconnections of the design unit with the external world are enumerated.
- The properties of these interconnections are defined.

entity declaration:

```
entity <entity_name> is
        port ( <port_name> : <mode> <type>;
       ….
           );
end <entity_name>;
```

- There are four modes for the ports in VHDL

  in, out, inout, buffer

- These modes describe the different kinds of interconnections that the port can have with the external circuitry.

  Sample program:

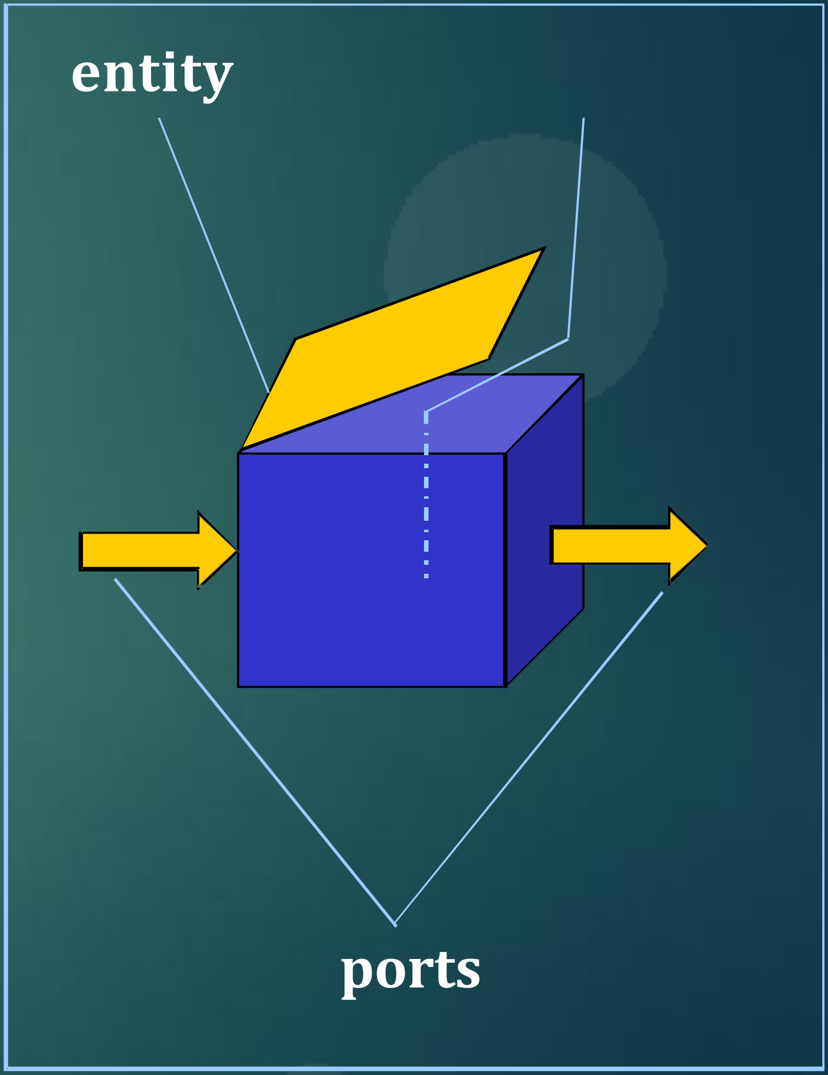
```
entity andgate is
        port ( c : out bit;
               a : in bit;
               b : in bit
        );
end andgate;
```

ARCHITECTURE

- Architecture defines the functionality of the entity.
- It forms the body of the VHDL code.
- An architecture belongs to a specific entity.
- Various constructs are used in the description of the architecture.

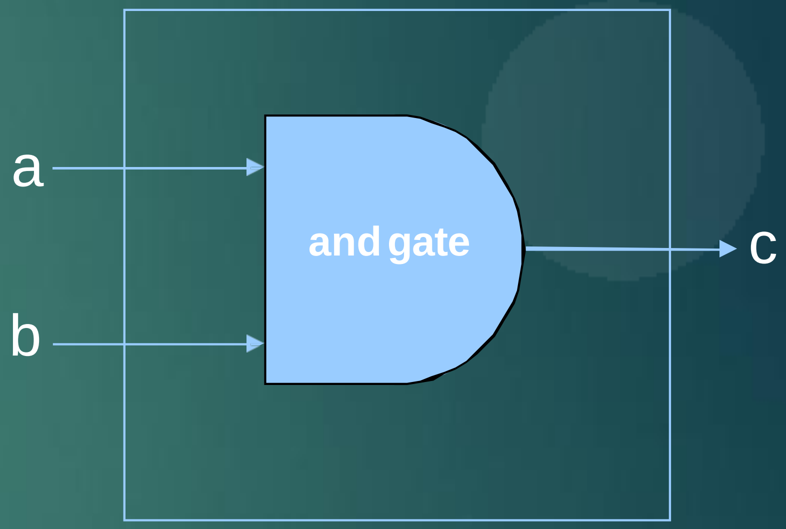
architecture declaration:

```
architecture <architecture_name> of
                <entity_name> is
                <declarations>
begin
              <vhdl statements>
end <architecture_name> ;
```



EXAMPLE OF A VHDL ARCHITECTURE

```
entity andgate is
      port (c : out bit;
                  a : in bit;
                  b : in bit
                 );
end andgate;
architecture arc_andgate of andgate is
begin
      c <= a and b;
end arc_andgate;
```



SIGNALS

- Syntax:

```
signal signal_name : type := initial_value;
```

- Equivalent to wires.

- Connect design entities together and communicate changes in values within a design.

- Computed value is assigned to signal after a specified delay called as Delta Delay.

- Signals can be declared in an entity (it can be seen by all the architectures), in an architecture (local to the architecture), in a package (globally available to the user of the package) or as a parameter of a subprogram (I.e. function or procedure).

- Signals have three properties attached to it.

- Type and Type attributes,value,Time (It has a history).

- Signal assignment operator : ‘<=‘.

- Signal assignment is concurrent outside a process & sequential within aprocess.

CONCATENATION

- This is the process of combining two signals into a single set which can be individually addressed.

- The concatenation operator is ‘&’.

- A concatenated signal’s value is written in double quotes whereas the value of a single bit signal is written in single quotes.

WITH-SELECT

- The with-select statement is used for selective signal assignment.
- It is a concurrent statement.

Syntax

```
with expression  select:
target   <= expression 1 when  choice1
                 expression 2 when  choice2
                             .
                             .
                             .
                  expression N  when  others;
```

Example:

```
entity mux2 is
      port ( i0, i1 : in bit_vector(1 downto 0);
                  y : out bit_vector(1 downto 0);
                  sel : in bit
                );
end mux2;

architecture behaviour of mux2 is
begin
      with sel select
                  y <= i0 when '0',
                  i1 when '1';
end behaviour;
```

WHEN-ELSE

syntax :

```
Signal_name<= expression1 when condition1  else
               expression2 when condition2 else
               expression3;
```

Example:

```
entity tri_state is
   port (a, enable : in std-logic;
          b : out std_logic);
end tri_state;
architecture beh of tri_state is
begin
    b <= a when enable ='1’ else
          ‘Z’;
end beh;
```

WHEN-ELSE VS. WITH-SELECT

- In the ‘with’ statement,choice is limited to the choices provided by the with‘express-ion’.
- In the ‘when’ statement each choice itself can be a separate expression.
- when statement is prioritized (since each choice can be a different expression,more than one condition can be true at the same time, thus necessitating a priority based assignment)
- with statement does not have any priority (since choices are mutually exclusive) .

DELAYS IN VHDL

- VHDL allows signal assignments to include delay specifications, in the form of an ‘after’ clause.
- The ‘after’ clause allows you to model the behavior of gate and delays.
- Delay’s are useful in simulation models to estimate delays in synthesizable design.

Two fundamental delays are

- Inertial delay.
- Transport Delay.

INERTIAL DELAY

- Inertial Delay models the delays often found in switching circuits (component delays).

- These are default delays.

- Spikes are not propagated if after clause is used.

- An input value must be stable for an specified pulse rejection limit duration before the value is allowed to propagate to the output.

INERTIAL DELAY (Cont..)

- Inertial delay is used to model component delay.
- Spike of 2ns in cmos component with delay of 10ns is normally not seen at the output.
- Problem arises if we want to model a component with delay of 10ns, but all spikes at input > 5 ns are visible output.
- Above problem can be solved by introducing reject & modeling as follows:

outp <= reject 5 ns inertial Inp after 10 ns;

TRANSPORT DELAY

- Transport delay models the behavior of a wire, in which all pulses (events) are propagated.
- Pulses are propagated irrespective of width.
- Good for interconnect delays.
- Models delays in hardware that does not exhibit any inertial delay.
- Represents pure propagation delay
- Routing delays can be modeled using

  transport delay Z<=transport a after 10

  ns;

BLOCK STATEMENTS

- Main purpose of block statement is organizational only or for partitioning thedesign.

  syntax:

```
block_label :block
                <declarations>
             begin
                <concurrentstatements>
             end blockblock_label;
```

- Introduction of a Block statement does not directly affect the execution of a simulation model.

- Block construct only separates part of the code without adding any functionality.

Behavioral Modeling

BEHAVIOR LEVEL

- The behavior of the entity is expressed using sequentially executed, procedural code, which is very similar in syntax and semantics to that of a high level programming languages such as C or Pascal.

- Process statement is the primary mechanism used to model the behavior of anentity.

- Process statement has a declarative part (before the keyword begin) and a statement part (between the keywords begin and end process).

- The statements appearing within the statement part are sequential statements and are executed sequentially.

SEQUENTIAL PROCESSING (PROCESS)

- Process defines the sequential behavior of entire or some portion of the design.

- Process is synchronized with the other concurrent statements using the sensitivity list or wait statement.

- Statements, which describe the behavior in a process, are executed sequentially.

- All processes in an architecture behave concurrently.

- Simulator takes Zero simulation time to execute all statements in a process.

- Process repeats forever, unless suspended.

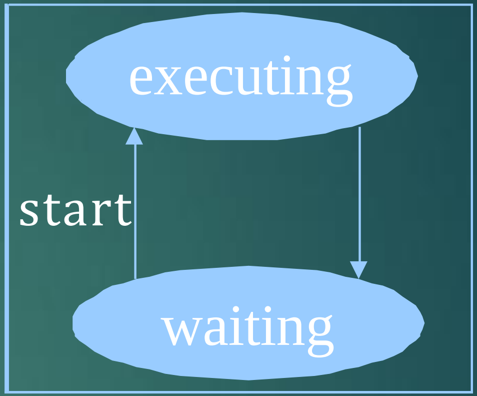
PROCESS (Cont..)

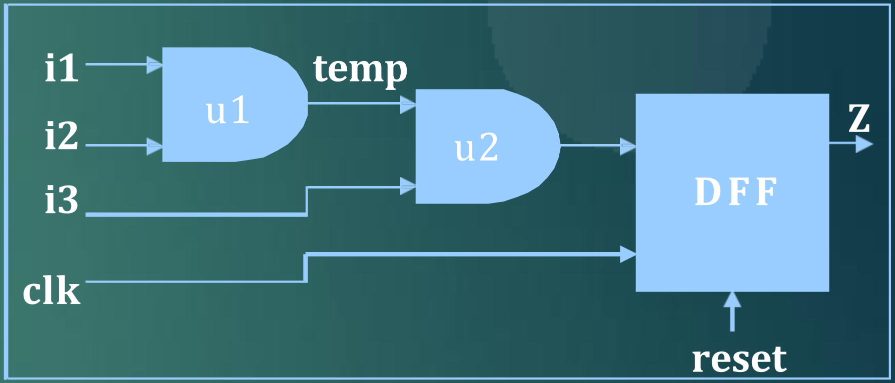
- Process can be in waiting or executing.

  syntax:

  ```
  process (sensitivitylist)
          <declarations>
   begin
   <sequential statements>;
  end process;
  ```

- Once the process has started it takes time delta ‘t’ for it to be moved back to waiting state. This means that no simulation time is taken to execute the process.





```
process(clk,reset)
begin
if reset=‘1’ then
     Z<=‘0’;
elsif clk’event and clk = ‘1’then
     Z<=(i1 and i2) and i3;
end if;
end process;
```

PROCESS TYPES

- Combinational process:
  - Aim is to generate pure combinational circuit.
  - All the inputs must be present in sensitivity list.
  - Latches could be inferred by the synthesizer to retained the old value, if an output is not assigned a value under all possible condition
  - To avoid inference of latches completely specify the values of output under all conditions and include all ‘read’ signals in the sensitivitylist.

PROCESS TYPES (Cont..)

- Clocked processes:
  - Clocked processes are synchronous and several such processes can be joined with the same clock.
  - Generates sequential and combinational logic.
  - All signals assigned within clock detection are registered(i.e. resulting flip-flop)
  - Any assignment within clock detection will generate a Flip-flop and all other combinational circuitry will be created at the ‘D’ input of the Flip-flop.

SIGNALS WITHIN PROCESS

- Process places only one driver on a signal.
- Value that the signal is up-dated with is the last value assigned to it within the process execution.
- Signals assigned to within a process are not updated with their new values until the process suspends.

SEQUENTIAL CONSTRUCTS

- The final output depends on the order of the statements, unlike concurrent statements where the order is inconsequential .

- Sequential statements are allowed only inside process.

- The process statement is the primary concurrent VHDL statement used to describe sequential behavior.

- Sequential statements can be used to generate both combina- tional logic and sequential logic.

SEQUENTIAL PROCESS TO MODEL JK FLIP-FLOP

```
process (clk)
variable state : bit := ‘0’;
begin
if clk’event and clk=‘1’ then
   if(j=‘1’ and k=‘1’) then
      state:=not state;
   elsif(j=‘0’ and k=‘1’) then
      state:=‘0’;
   elsif(j=‘1’ and k=‘0’) then
      state:=‘1’;
   end if;
   Q<=state;
   Qbar<=not state;
end if ;
end process;
```

VARIABLES

syntax :

```
variable variable_name: type := initial_value;
```

- Can be declared and used inside a process statement or in subprogram.
- Variable assignment occurs immediately.
- Variables retain their values throughout the entire simulation Sequential (inside process)
- Variable have only type and value attached to it.They don't have past history unlike signal.
- Require less memory & results in fast simulation

CONSTANTS

syntax :

```
constant constant_name : type := value;
```

- Constants are identifiers with a fixed value.
- They should not be assigned any values by the simulation process.
- Constants improve the clarity and readability of a project.
- It is used in place of the value to make the code more readable

SIGNAL vs. VARIABLE

SIGNAL

- Connects design entities together (acts as a wire).
- Signals can be declared both inside and out side of the process (sequential inside process, concurrent outside process)
- Signals have 3 properties attached
  - Type & Type Attributes.
  - Value.
  - Time.(it has a history)
- Signal is assigned it's value after a delta delay.
- Signals require more memory & showers simulation.

VARIABLE

- These are identifiers within process or subprogram.
- Variables can only be declared inside a process. These cannot be used to communicate between two concurrent statements.
- Variables have only
  - Type.
  - Value.
- Variable is assigned its value immediately.
- Variable require less memory & enables fast simulation.

SIGNAL vs. VARIABLE EXAMPLE

SIGNAL

```
process
begin
wait for 10 ns;
Sum1<=sum1+1;
Sum2<=sum1+1;
end process;
```

Time	sum1	sum2
0	0	0
10	0	0
10+1delta	1	1
20	1	1
20+1 delta	2	2
30	2	2
30+1delta	3	3

VARIABLE

```
process
begin
wait for 10 ns;
Sum1:=sum1+1;
Sum2:=sum1+1;
end process;
```

Time	sum1	sum2
0	0	0
10	1	2
10+1delta	1	2
20	2	3
20+1 delta	2	3
30	3	4
30+1delta	3	4

SEQUENTIAL STATEMENTS

- An if- elsif- else statement selects one or none of a sequence of events toexecute.
- The choice depends on one or more conditions.
- If-else corresponds to when else command in the concurrent part.
- if statements can be used to gene-rates prioritized structure.
- if statements can be nested.

```
syntax:
   if <condition1> then
       <statements>;
  elsif <condition2> then
       <statements>;
              ….
  else
       <statements>;
  end if ;
```

SEQUENTIAL STATEMENTS (Cont..)

CASE STATEMENT:

- The case statement selects,one of a number of alternative sequences of statements depending on the value of the select signals.
    - All choices must be enumerated. ‘others’ should be used for enumerating all remaining choices which must be the last choice.
  - Case statement results in a parallel structure. Syntax

```
case  expression  is
          when choice1 =>     <seq_statements>
          when choice2 =>     <seq_statements>
                     ………..
          when others => <default_instruction>
end case;
```

SEQUENTIAL STATEMENTS (Cont..)

LOOP STATEMENTS

- Used to iterate through a set of sequential statements.
- No declaration is required explicitly for Loop identifier.
- Loop identifier cannot be assigned any value within Loop.
- Identifier outside the loop with the same name as loop identifier has no effect on loop execution.

SEQUENTIAL STATEMENTS (Cont..)

WHILE LOOP :

Syntax :

```
loop_label:while condition loop
      <sequence of statements>
 end loop loop_label
```

- Statements are executed continuously as long as condition is true.
- Has a Boolean Iteration Scheme.
- Condition is evaluated before execution.

FOR LOOP :

Syntax :

```
loop_label: for loop_parameter in discrete_range loop
  <sequence of statements>

end loop loop_label;
```

SEQUENTIAL STATEMENTS (Cont..)

WAIT STATEMENT :

- The wait statement gives the designer the ability to suspend the sequential execution of a process or a sub-program.

- The wait statement specifies the clock for a process statement that is read by synthesis tools to create sequential logic such as registers andflip-flops.

- It can also be used for delaying process execution for an amount of time or to modify the sensitivity list of the process dynamically.

- Three different options available for wait are

1) wait on signal 2) waituntil Boolean_expr 3) wait for time_expr

SEQUENTIAL STATEMENTS (Cont..)

WAIT ON signal:

- Specifies a list of one or more signals that the WAIT statement will wait for events upon.if any signal list has an event occur on it, execution continues with the statement following the wait statement.

  example: WAIT ON a,b;

WAIT UNTILexpression:

- Suspends execution of the process until the expression returns a value of true.

  example: WAIT UNTIL ((x * 10) < 100);

WAIT FOR time_expression:

- Suspends execution of the process for the time specified by the time expression.

  example: WAIT FOR 10 ns;