STYLES OF MODELING
DSD

AJIT KUMAR

STYLES OF MODELING

- They are of 3 types:

1. Structural modeling
2. Data flow modeling
3. Behavorial modeling

1. STRUCTURAL MODELING

- An entity is modeled as a set of components connected by signals,i.e,as a netlist.
- The behavior of the entity is not explicitly apparent from it model.
- The component instantiation statement is primary mechanism used for describing such model of an entity.

a. Component declaration

b. Component instantiation

a. COMPONENT DECLARATION

- Component declaration declares the name and the interface of a component. The interface specifies the mode and the type of ports.

- SYNTAX:

  Component component_name [is]

  Port (input _name : ***IN*** data_type; Output_name : ***OUT*** data_type);

  End component [component_name];

- Eg.

  Component AND1

  Port(x,y : ***IN*** bits; z: ***OUT*** bits);

  End component AND1;

b. COMPONENT INSTANTIATION

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

- SYNTAX:

  Component_label : component_name ***PORTMAP*** (associated values);

- Eg.

  A 1 : AND1 ***PORTMAP*** (x,y,z);

Eg. Write a structural VHDL code for Half adder.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY HA IS
PORT ( a, b : IN bit;
        s, c : OUT bit);
END HA;

ARCHITECTURE HA1 OF HA IS
COMPONENT XOR1
PORT (x, y : IN BIT ;
     Z : OUT BIT);
END COMPONENT XOR1;

COMPONENT AND1
PORT (l, m : IN BIT;
N: OUT BIT);
END COMPONENT AND1;
BEGIN
    X1: XOR1 PORTMAP ( a, b, s);
    A1: AND1 PORTMAP (a, b, c);
END HA;
```

2. DATAFLOW MODELING

Shows the flow of information through the entity, which is expressed using concurrent signal assignment statements and block statements.

By using following statements:

i. 1.Operators or signal assignment statements.
ii. 2.WHEN ELSE statement
iii. 3.WITH/SELECT/WHEN
iv. 4.BLOCK statements- A. Simple block
v. B. Nested block
vi. 5. GUARDED BLOCK statement
vii. 6.GENERATE statement
viii. 7.GENERIC statement
ix. 8.UNAFFECTED value

i. Operators or signal assignment statements.

SYNTAX:

ARCHITECTURE architecture_name ***OF*** entity_name ***IS***
BEGIN
Signal assignment statements;
END architecture_name;

Eg.OR gate

ARCHITECTURE or1 ***OF*** or ***IS***
BEGIN
Z<=a OR b;
END or1;

ii. **WHEN/ELSE statement**

Conditional signal assignment statement ,also called Simple WHEN.--based on condition

SYNTAX:

Target_signal<=

[waveform_elements ***WHEN*** condition ***ELSE***]

[waveform_elements ***WHEN*** condition ***ELSE***]

……

waveform_elements [***WHEN*** condition] ;

Eg.Half Adder

Y<= “00” ***WHEN*** x= “00” ***ELSE***

“10” ***WHEN*** x= “01” ***ELSE***

“10” ***WHEN*** x= “10” ***ELSE***

“01” ***WHEN*** OTHERS;

iii. WITH/SELECT/WHEN statements

Selected signal assignment statement --based on the value of select expression.

SYNTAX:

WITH expression ***SELECT***

Target_signal<= waveform_elements ***WHEN*** choices;

waveform_elements ***WHEN*** choices;

……

waveform_elements ***WHEN*** choices;

Eg.Half Adder

WITH x ***SELECT***

y<= “00” ***WHEN*** “00”;

“10” ***WHEN*** “01”;

“10” ***WHEN*** “10”;

“01” ***WHEN*** “11”;

iv. BLOCK Statements

A block statement is a concurrent statement. It can be used for three major purpose:

1. To disable signal drivers by using GUARDS.
2. To limit scope of declarations, including signal declarations.
3. To represent a portion of a design.

2 types of block statements:

A. SIMPLE block

B. NESTED block

A. SIMPLE BLOCK

SYNTAX:

Label: ***BLOCK*** [***IS***]
[declarative part]
BEGIN
(concurrent part)
END BLOCK label;

Eg.

B1: ***BLOCK***
BEGIN
Q<= d;
END BLOCK B1;

B. NESTED BLOCK

Label1:***BLOCK*** [***IS***]

[declarative part of 1st block]

BEGIN

Label2:***BLOCK***

[declarative part of 2nd block]

BEGIN

(concurrent statements of 2nd block)

END BLOCK label2;

(more concurrent statements of 1st block)

END BLOCK label1;

Eg.

```
B1: BLOCK
BEGIN
    B2:BLOCK
    BEGIN
    Y<= a + '1';
    END BLOCK B2;
Z<=y + '1';
END BLOCK B1;
```

v. GUARDED BLOCK statements

The control enters this block only if the guarded-expressions in the block is true. i.e the block is guarded or protected.

SYNTAX:

Label: ***BLOCK***(***GUARDED*** expression)[***IS***]

[declarative part]

BEGIN

Concurrent statement

END BLOCK [label];

Eg.

B1: ***BLOCK***(clk='1')

BEGIN

Q<= ***GUARDED*** d;

END BLOCK B1;

vi. **GENERATE statements**

1. FOR GENERATION Scheme

A. Concurrent statements can be replicated a predefined number of times.

SYNTAX:

Generate_label: ***FOR*** generate_identifier ***IN*** discrete_range ***GENERATE***

[block declarations

BEGIN]

Concurrent statements

END GENERATE [Generate_label];

Eg.

G1: ***FOR*** i ***IN*** (3 ***DOWNTO*** 0) ***GENERATE***
Output(i)<='1' ***WHEN*** a(i)='1'
ELSE '0';
END GENERATE G1;

2. IF GENERATION scheme

Concurrent statements can be conditionally replicated.

SYNTAX:

Generate_label: ***IF*** expression ***GENERATE***

[block declarations

BEGIN]

Concurrent statements

END GENERATE [Generate_label];

Eg.

G1: ***IF*** a>b ***GENERATE***

Out<='1';

END GENERATE G1;

vii. **GENERIC statement**

- Static parameter can be varies using Genarics.
- Generic shows how certain types of information can be passed into entity .
- Examples of such information are rise and fall **delays** and the **size** of the interface ports.This is accomplished using Generics.
- Generics of an entity are declared along with its ports in the **entity declarations** but can be used globally of all architecture.

- SYNTAX:

GENERIC (parameter_name: parameter_type:=parameter_value);

Eg.

```
GENERIC (n: INTEGER:=8);
```

Eg.Nand gate

```
ENTITY nand_gate IS
GENERIC (m:INTEGER:=2);
PORT(a :IN BIT_VECTOR(m DOWNTO 0);
        Z:OUT BIT);
END nand_gate;
```

Note: Generics can be specified as a globally static expression in one of the following:

Entity declaration, Component declaration, Component instantiation, Configuration specification, Configuration declaration.

viii. **UNAFFECTED value**

- It is possible to assign a value of **unaffected** to a signal assignment statement.
- Such an assignment causes no changes to the driver for the target signal.

SYNTAX:

WITH identifier ***SELECT***

Signal_assignment ***WHEN*** value;

……..

UNAFFECTED WHEN OTHERS;

Eg.

WITH i ***SELECT***

Output<= “0001” ***WHEN*** apply;

“0010” ***WHEN*** waits;

“0100” ***WHEN*** reset;

UNAFFECTED WHEN OTHERS;

3. BEHAVORIAL MODELING

In this modeling style, the behavior of the entity is expressed using sequentially executed, procedural code, which is very similar in syntax and semantics to that of high-level programming language like C, Pascal.

i. 1. ENTITY DECLARATION
ii. 2. ARCHITECTURE BODY
iii. 3. PROCESS STATEMENT
iv. 4. VARIABLE ASSIGNMENT STATEMENT
v. 5. SIGNAL ASSIGNMENT STATEMENT
vi. 6. WAIT
vii. 7. IF/ THEN/ELSE
viii. 8. CASE/WHEN
ix. 9. CASE/WHEN WITH NULL
x. 10. LOOP
xi. 11.EXIT
xii. 12.NEXT

i. ENTITY DECLARATION

An entity declaration describes the external interface of the entity i.e.it gives the black-box view.

SYNTAX:

ENTITY entity_name ***IS***

[GENERIC (list of generics and their types);]

[PORT (list of interface port names and their types)

[Entity-declaration]

[***BEGIN***

Entity-statements]

END [***ENTITY***] [entity_name];

e.g.

ENTITY AOI ***IS***

PORT (a, b, c, d: IN bit; z: OUT bit);

END [***ENTITY***] [AOI];

ii. ARCHITECTURE BODY

- An architecture body describes the internal view of an entity. It describes the functionality or the structure of the entity.

SYNTAX:

ARCHITECTURE architecture-name ***OF*** entity-name ***IS***

[Architecture-item-declaration]

BEGIN

Concurrent statements; these are→

Process-statement

Block-statement

Concurrent-procedure-call-statement

Concurrent-assertion statement

Component-instantiation-statement

Generate-statement

END [ARCHITECTURE] [architecture-name];

e.g.

ARCHITECTURE AOI_sequential ***OF*** AOI ***IS***
BEGIN
PROCESS (a, b, c, d)
VARAIBLE temp1, temp: BIT;
BEGIN
Temp1:=a ***AND*** b;
Temp2:=c ***AND*** d;
Temp1:=temp1 ***OR*** temp2;
Z<=***NOT*** temp1;
END PROCESS;
END AOI_sequential;

iii. PROCESS STATEMENT

A process statement contains sequential statements that describe the functionality of a portion of an entity in sequential terms.

SYNTAX:

[Process-label] ***PROCESS*** [(sensitivity-list)] [IS]

[Process-item-declarations]

BEGIN

Sequential-statements; these are→

Variable-assignment statement ;Signal assignment statement; Wait-statement; If-statement; Case-statement; Loop-statement; Null-statement; Next-statement; Assertion-statement; Report-statement; Procedure-call-statement; Return-statement

END PROCESS [process-label];

iv. VARIABLE ASSIGNMENT STATEMENT

Variable can be declared and used inside a process statement.

SYNTAX:

VARAIBLE-object: = expression;

e.g.

PROCESS (A)

VARIABLE k: integer: =-1;

BEGIN

K: =k+1;

END PROCESS;

v. SIGNAL ASSIGNMENT STATEMENT

Signals are assigned values using a signal assignment.

SYNTAX:

SIGNAL-object<= expression [AFTER delay-value]

e.g.

counter<=counter+”0010” ***AFTER*** 6ns;

vi. WAIT STATEMENT

- Wait statement provides an alternative way to suspend the execution of a process.
- WAIT are of 3 types:
- WAIT ON
- WAIT UNTIL
- WAIT FOR

- SYNTAX:
- ***WAIT ON*** sensitivity-list;
- ***WAIT UNTIL*** Boolean-expression;
- ***WAIT FOR*** time-expression;

e.g.
WAIT ON a, b, c;
WAIT UNTIL a=b;
WAIT FOR 10ns;

Note: they may also be combined in single wait statement.

SYNTAX:
WAIT ON sensitivity-list ***UNTIL*** Boolean-expression ***FOR*** time-expression;

e.g.
WAIT ON a, b, c ***UNTIL*** a=b ***FOR*** 10ns;

vii. IF STATEMENT(IF/THEN/ELSE)

- An If statement selects a sequence of statements for execution based on the value of a condition.

SYNTAX:

IF boolean-expression ***THEN***

Sequential-statements

[ELSIF boolean-expression ***THEN***

Sequential –statements]

[***ELSE***

Sequential-statements]

END IF;

e.g.

IF a ***THEN***

Out1<=”00”;

ELSIF b ***THEN***

Out1<=”01”;

ELSE

Out1<=”10”;

END IF;

viii. *CASE STATEMENT (CASE/WHEN)*

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

SYNTAX:

CASE expression ***IS***
WHEN choices=> sequential-statements
WHEN choices=>sequential-statements
……
[***WHEN OTHERS***=>sequential –statements]
END CASE;

e.g.
CASE day ***IS***
WHEN mon=> pocket-money: =6;
WHEN tue|wed=>pocket-money: =7;
WHEN fri TO sat ->pocket-money: =8;

WHEN OTHERS=>pocket-money: =0;
END CASE;

ix. **NULL STATEMENT**

- The statement ***NULL*** is a sequential statement that does not cause any action to take place; execution continues with the next statement.

CASE/WHEN with NULL

SYNTAX:

CASE identifier ***IS***

WHEN value=> assignment;

……..

WHEN value=> ***NULL***;

END CASE;

X. LOOP STATEMENT

- A loop statement is used to iterate through a set sequential statements.

3 types of LOOP statements:

A. SIMPLE LOOP
B. FOR LOOP
C. WHILE LOOP

A. SIMPLE LOOP
SYNTAX:

[Loop-label:] iteration-scheme ***LOOP***
Sequential-statements
END LOOP [loop-label];

e.g.

```
L1: LOOP
  a<=a+'1';
  b<=a+ '1';
END LOOP L1;
```

B. FOR LOOP

SYNTAX:

[loop-label:] ***FOR*** identifier ***IN*** range ***LOOP***

Sequential statements

END LOOP [loop-label];

e.g.

FOR number ***IN*** 2 ***TO*** N ***LOOP***

Factorial: =factorial* number;

END LOOP;

C. WHILE LOOP

SYNTAX:

[loop-label:] ***WHILE*** condition ***LOOP***

Sequential statements

END LOOP [loop-label];

e.g.

WHILE A<5 ***LOOP***

X (i) <=A (i) +'1';

END LOOP;

xi. EXIT STATEMENT

- The exit statement is a sequential statement that can be used only inside a loop.
- It causes execution to jump out of the innermost loop or the loop whose label is specified.

SYNTAX:

EXIT [loop-label] [WHEN condition];

e.g.

```
L1: LOOP
        J: =j+5;
        IF J> 50 THEN
            EXIT L1;
        END IF;
END LOOP L1;
```

xii. NEXT STATEMENT

- The next statement is also a sequential statement that can be used only inside a loop.

SYNTAX:

NEXT [loop-label] [WHEN condition];

e.g.

```
L1: LOOP
     IF J< 50 THEN
     J:=j+5;
     ELSIF J=50 THEN
    NEXT;
     ELSE
   NULL;
   END IF;
   END LOOP L1;
```

Assertion Statement

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

- Syntax

Assert boolean_expression

[report string_expression];

[severity expression];