Dataflow Modeling

Dataflow modeling provides the means of describing combinational circuits by their function rather than by their gate structure. Dataflow modeling uses a number of operators that act on operands to produce the desired results. Verilog HDL provides about 30 operator types.

Verilog HDL Operators

Symbol	Operation
+	binary addition
-	binary subtraction
&	bit-wise AND
|	bit-wise OR
^	bit-wise XOR
~	bit-wise NOT
==	equality
>	greater than
<	less than
{}	concatenation
? :	conditional

Dataflow modeling uses continuous assignments and the keyword assign. A continuous assignment is a statement that assigns a value to a net. The datatype **net** is used in Verilog HDL to represent a physical connection between circuit elements. The value assigned to the net is specified by an expression that uses operands and operators. As an example, assuming that the variables were declared, a 2-to-1 multilexer with data inputs A nad B, select input S, and output Y is described with continuous assignment

assign Y= (A & ~S) | (B & S)

The dataflow description of a 2-to-4 line decoder is shown in HDL below. The circuit is defined with four continuous assignment statements using Boolean expressions, one for each output.

```
// Dataflow description of 2-to-4 line decoder with enable input (E)
module decoder_df (A,B,E,D);
    input A,B,E;
    output [3,:0] D;

        assign D[3] =~(~A & ~B & ~E);      Or    assign  D[3] =~(~A & ~B & ~E),
        assign D[2] =~(~A &  B & ~E);                      D[2] =~(~A &  B & ~E),
        assign D[1] =~( A & ~B & ~E);                      D[1] =~( A & ~B  & ~E),
        assign D[0] =~( A &  B & ~E);                      D[0] =~( A &  B & ~E);

endmodule
```

Conditional operator in Verilog HDL takes three operands:

Condition ? true-expression: false-expression;

This operator is equivalent to an if-else condition. HDL given below shows the description of a 2-to-1 line multiplexer using conditional operator.

```
// Dataflow description of 2-to-1 line multiplexer
module mux2x1_df (A,B,select,OUT);
    input A,B,select;
    output OUT;
        assign OUT=select ? A:B;
endmodule
```

Behavioral Modeling

Behavioral modeling represents digital circuits at a functional and algorithmic level. It is used mostly to describe sequential circuits, but can be used to describe combinational circuits. Here the behavioral modeling concept will be presented for combinational circuits.

Behavioral description use the keyword **always** followed by a list of procedural assignment statements. The target output of procedural assignment statement must be of the **reg** data type. The behavioral description of 2-to-1 line multiplexer in HDL is given below.

```
// Behavioral description of 2-to-1 line multiplexer
module mux2x1_bh (A,B,select,OUT);
        input A,B,select;
        output OUT;
        reg OUT;
        always @(select or A or B)
                if (select==1) OUT=A;
                else OUT=B;
endmodule
```

The procedural assignment statements inside the always block are executed every time there is a change in any of the variables listed after the **@** symbol. In this case, they are the input variables A, B, and select. The condition statement **if-else** provides a decision based upon the value of the select input. The **if** statement can be written whitout the equality symbol:

If (select) OUT=A;

HDL description of a 4-to-1 line multiplexer is given below. The select input is defined as a 2-bit vector and output y is declared as **reg** data. The always statement has a sequential block enclosed between the keywords **case** and **endcase**. The block is executed whenever any of the inputs listed after the @ symbol changes in value. The case statement is multiway conditional branch condition. The case expression (select) is evaluated and compared with the values in the list of statements that follow. The first value that matches the true condition is executed.

```
// Behavioral description of 4-to-1 line multiplexer
module mux4x1_bh (i0,i1,i2,i3,select,y);
        input i0,i1,i2,i3;
        input [1:0]select;

        output y;
        reg y;
        always @( i0 or  i1 or i2 or i3 or select)
                case (select)
                    2'b00: y=i0;
```

```
                2'b01: y=i1;
                2'b10: y=i2;
                2'b11: y=i3;
            endcase
endmodule
```