Behavioral constructs

if-else statements

- The if construct checks a specific condition and decides execution based on the result.

assignment 1;

if(condition) **assignment2;**

assignment3;

assignment4;

- After execution of **assignment1**, the condition specified is checked . If it is satisfied , **assignment2** is executed; if not it is skipped.
- In either case the execution continues through **assignment3, assignment4**, etc.
- Execution of **assignment2** alone is dependent on the condition. The rest of the sequence remains.

if-else and if-elseif-else statements

```
if (<expression>)
begin
    true_statements;
end
```

```
if (<expression>)
begin
    true_statements;
end
else
begin
    false statements;
end
```

```
if (<expression 1>)
begin
    true_statements 1;
end
          :
          :
else if (<expression n>)
begin
    true_statements n;
end
else
begin
    default_statements;
end
```

Conditional statement - Examples

```
if (!ena == 1'b1)
begin
    out = in1 & in2;
end
```

```
if (sel == 1'b0)
begin
   mux_out = in0;
end
else
begin
   mux_out = in1;
end
```

```
if (in1 == 1'b0 && in2 == 1'b0)
dec_o1 = 1'b1;

else if (in1 == 1'b0 && in2 == 1'b1)
dec_o2 = 1'b1;

else if (in1 == 1'b1 && in2 == 1'b0)
dec_o3 = 1'b1;

else
dec_o4 = 1'b1;
```

case statement

- This is a multiway decision statement that tests whether an expression matches one of a number of other alternatives.

- Doesn't treat 'x' & 'z' as don't cares.

```
case (expression)
alternative 1:begin
                end
alternative 2:begin
                end
alternative 3:begin
                end
default: begin
                end
endcase
```

case statement - Example

```
module mux4_to_1 (out, i0, i1, i2, i3, s1, s0); output out;
input i0, i1, i2, i3;
input s1, s0;
reg out;
always @(s1 or s0 or i0 or i1 or i2 or i3)
case ({s1, s0})
2'd0 : out = i0;
2'd1 : out = i1;
 2'd2 : out = i2;
2'd3 : out = i3;
default: $display("Invalid control signals");
endcase
endmodule
```

casez statements

- **casez**: same as **case** statement but treats ‘z’ as don’t cares.

- All bit positions with ‘z’ can also be represented by ‘?’

casez - Examples

```
module
    casez_ex(counter,cmd);
input[0:3] counter;
output cmd;
reg cmd;
always@(counter)
casez(counter)
4'b???1:cmd=0;
4'b??10:cmd=1;
4'b?100:cmd=2;
4'b1000:cmd=3;
default : cmd=0;
endcase
endmodule
```

casex statement

- **casex**: same as **case** / **casez** statement but treats all ‘x’ and ‘z’ as don’t cares.

casex - Example

```
module casex_ex(sel,Bitposition);
input [5:1] sel;
output [2:0] Bitposition;
reg [2:0] Bitposition;
always @ (sel)
casex(sel)
5'bxxxx1 : Bitposition=1 ;
5'bxxx1x : Bitposition=2;
5'bxx1xx  : Bitposition=3;
5'bx1xxx : Bitposition=4;
5'b1xxxx : Bitposition=5;
default : Bitposition=0;
endcase
endmodule
```

Looping statements

- Verilog supports four types of loops:

– **while** loop

– **for** loop

– **forever** loop

– **repeat** loop

- Many Verilog synthesizers supports only ‘for’ loop for synthesis:

– Loop bound must evaluate to a constant.

– Implemented by unrolling the ‘for’ loop, andreplicating the statements.

for loop construct

The for loop in Verilog is quite similar to the for loop in C
Format:

```
for (initialization; expression; inr/dcr)
   begin
      statements
   end
```

The sequence of execution as follows

1. Execute **initialization**
2. Evaluate **expression**
3. If the **expression** evaluates to the true state(1), carry out **statements.** Go to step 5.
4. If the **expression** evaluates to false state (0), exit the loop.

5. Execute **incr/dcr**. Go to step 2.

for loop - Example

```
module addfor (s,co,a,b,cin,en);
output [7:0] s;
output co;
input [7:0] a,b;
input en,cin;
reg [8:0] c;
reg co;
reg [7:0] s;
integer i;
always @ (posedge en)
begin
c[0] = cin;
for (i=0; i<=7; i=i+1)
begin
{c[i+1],s[i]} = (a[i] + b[i] + c[i]);
end
co=c[8]
end
endmodule
```

repeat construct

- The repeat construct is used to repeat specified block a specified number of times.
- The format is show below , the quantity ‘a‘ can be a number or an expression evaluated to a number.
- As soon as the repeat statement is encountered, a is evaluated, the block will be executed ‘a’ times.
- If ‘a’ evaluates to 0 or x or z, the block is not executed.

Format:

```
repeat (a)
   begin
      statements
   end
```

repeat construct - Example

```
module clkgen (en, cnt);
input en;
output cnt;
integer cnt;
always @ (en)
Begin
if (en)
cnt=0;
   repeat (105)

        #10 cnt = cnt + 1'b1;
end
endmodule
```

While loop

The ***format*** of while loop is shown below.

- The **expression** is evaluated. If it is ***true*** , the **statements** executed and **expression** evaluated and checked again.
- If the expression evaluates to ***false,*** the loop is terminated and the following statement is taken for execution. If the expression evaluates to ***true,*** execution of **statement** is repeated.
- The loop is terminated and broken only if the expression evaluates to false.

Format:

```
while (expression)
   begin
       statements
   end
```

While loop - Example

```
module while2 (b,n,en,clk);
input [7:0] n;
input clk,en;
output b;
reg [7:0] a;
reg b;
always@(posedge en)
begin
a=n;
while(|a)
begin
b=1'b1;
always@ ( posedge clk)
a=a-1'b1;
end
b=1'b0;
end
initial
b=1'b0;
endmodule
```

forever loop

Repeated execution of a block in an endless manner is best done with the **forever** loop (compare with repeat where the repetition is for a fixed number of times)

Format:

```
forever
   begin
      statements;
   end
```

forever LOOP - Example

```
module clkgen (clk);
output clk;
reg clk;
initial
begin
    clk=1'b0;
    forever #10 clk=~clk;
end
initial
#1000 $finish;

endmodule
```

Reference

1. Samir Palnitkar,"Verilog HDL: A Guide to Digital Design and Synthesis" Prentice Hall, Second Edition,2003
2. T.R.Padmanabhan and B.Bala Tripura Sundari, "Design Through Verilog HDL" Wiley Student Edition