Loops and Flow Control

Centurion UNIVERSITY
Shaping Lives...
Empowering Communities...

if-else-if

case/casex/casez

forever

repeat

while

do..while

foreach

Examples - Loops

```
for (int i; i < arr.size();
     j+=2, i++) begin
     arr[i] += 200;
        arr[i]--;
 end
```

```
1)x = 0;
  while (x)  begin
     $display("%d", x);
     x--;
  end
```

```
2) do
     begin
     $display("%d", x);
      x--;
     end
     while (x);
```

forever

Centurion University
Shaping Lives...
Empowering Communities...

Continuous execution, without end

Used with timing controls

Usually last statement in some block

```
initial : clock_drive
begin
   clk = 1'b0;
   forever #10 clk = ~clk;
end : clock_drive
```

repeat

Repeat a block ‘x’ times, no conditional test

repeat (*expr*) *statement*

Example

```
x = 0;
repeat (16)
  begin
    $display(“%d”, x++);
  end
```

case/casez/casex

case: 4-value exact matching

casez:

Handles z as don't care

casex:

Handles both x and z as don't care

```
casez(a)
3'b00?: $display("0 or 1"); //LINE -1
3'b0??: $display("2 or 3"); //LINE -2
default: $display("4 to 7");
```

1) If a=010 => displays "2 or 3"
2) If a=00x => displays "0 or 1"
3) If a=0zx => displays "0 or 1" ← z matches 0

User defined types

Centurion UNIVERSITY
Shaping Lives...
Empowering Communities...

Syntax:

typedef <base_data_type> <type_identifier>

eg:

typedef int inch ; *// inch becomes a new type*

inch foot = 12, yard = 36; *// 2 new variables of type 'inch'*

Enumerated types

Enumeration is a useful way of defining abstract variables.

Define an enumeration with " enum "

enum {red, green, yellow} traf_lite1, traf_lite2;

Values can be cast to integer types and auto-incremented

enum { a=5, b, c} vars; *// b=6, c=7*

A sized constant can be used to set size of the type

enum bit [3:0] { bronze=4'h3, silver, gold} medal; *// All medal members are 4-bits*

Centurion
UNIVERSITY
Shaping Lives...
Empowering Communities...

Tasks and Functions

SystemVerilog makes a number of extensions to basic Verilog syntax.

automatic tasks allocate memory dynamically at call time.

Default port direction is input

ANSI style portlists

Implied begin…end

Arguments can be ANY SV type, even structs, etc.

```
task automatic my_task( input int local_a,
                                int local_b);

if (local_a == local_b)
   begin
      my_task(local_a-1,local_b+1);
      return;   // end 'this' copy of task
   end
global_a = local_a;
global_b = local_b;

endtask
```

return keyword is supported and terminates task at that point

Full recursion is supported (automatic variables/arguments stored on stack)

- Can do concurrent calls
- Can do recursive calls

Functions

automatic functions allocate memory dynamically at call time (full recursion).

Default port direction is input (also supports output)

ANSI style portlists

```
function automatic int factorial (int n);
   if (n==0) return(1); // factorial 0 is 1
   else return(factorial(n-1)*n);
endfunction
```

Implied begin…end

Arguments and return type can be ANY SV type, even complex structs, etc.

return(value) is supported and terminates function at that point

```
function void inverta();
   a = !a
endfunction

reg a;

initial
  inverta(); // function called like a task
```

Return type of void means no return value!

Recommended style (instead of writing a task) to guarantee a task executes with 0 delay.

Tasks and Functions

SystemVerilog makes a number of extensions to basic Verilog syntax.

automatic tasks allocate memory dynamically at call time.

Default port direction is input

ANSI style portlists

Implied begin…end

Arguments can be ANY SV type, even structs, etc.

```
task automatic my_task( input int local_a,
                                int local_b);

if (local_a == local_b)
   begin
      my_task(local_a-1,local_b+1);
      return;   // end 'this' copy of task
   end
global_a = local_a;
global_b = local_b;

endtask
```

return keyword is supported and terminates task at that point

Full recursion is supported (automatic variables/arguments stored on stack)

- Can do concurrent calls
- Can do recursive calls

Functions

automatic functions allocate memory dynamically at call time (full recursion).

Default port direction is input (also supports output)

ANSI style portlists

```
function automatic int factorial (int n);
   if (n==0) return(1); // factorial 0 is 1
   else return(factorial(n-1)*n);
endfunction
```

Implied begin…end

Arguments and return type can be ANY SV type, even complex structs, etc.

return(value) is supported and terminates function at that point

```
function void inverta();
   a = !a
endfunction

reg a;

initial
  inverta(); // function called like a task
```

Return type of void means no return value!

Recommended style (instead of writing a task) to guarantee a task executes with 0 delay.

Tasks and Functions - Usage

Centurion UNIVERSITY
Shaping Lives...
Empowering Communities...

Tasks

- Tasks can enable other tasks and functions
- Tasks may execute in non-zero simulation time.
- Tasks may have zero or more arguments of type input, output and inout.

Centurion
UNIVERSITY
Shaping Lives...
Empowering Communities...

Tasks and Functions - Usage

Functions

- Function can enable other functions only. Task cannot be called from functions.
- Functions should execute in zero simulation time.
- Functions have only one return value but System Verilog also allows functions to have input, output or inout types.

Argument passing

Centurion University
Shaping Lives... Empowering Communities...

Pass by value

Each argument is copied to the subroutine area

```
function int crc( byte packet [1000:1] );
    for( int j= 1; j <= 1000; j++ ) begin
            crc ^= packet[j];
    end
    endfunction
```

- Pass by reference
  - A reference to the original argument is passed instead of copyin

```
function int crc( ref byte packet [1000:1] );
    for( int j= 1; j <= 1000; j++ ) begin
            crc ^= packet[j];
    end
endfunction
```