SV Important Notes

Credits : https://youtube.com/playlist?list=PL40xmtPvboRs6Ng_1Q_V-1MdJH50A6Ulz&si=ny5MRQivdkgd_GTu

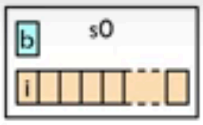
```
module dut ( input wire clk, reset );

  enum {OFF, ON} state;
  always @( posedge clk or posedge reset ) begin
      if ( reset ) begin
          state = OFF;
      end else begin
          state = (state==ON) ? OFF : ON;
      end
  end

endmodule
```

enum is synthesizable

```
module dut ( input wire clk, reset );

  string state;
  always @( posedge clk or posedge reset ) begin
      if ( reset ) begin
          state = "OFF";
      end else begin
          state = (state=="ON") ? "OFF" : "ON";
      end
  end

endmodule
```

string is not synthesizable

```
module test ();

  struct { bit b; int i; } s0;
  initial begin
      $display( "%0b", s0.b );
      $display( "%0b", s0.i );
      $display( "%0p", s0 );
      $display( "%0b", s0 );
  end

endmodule
```

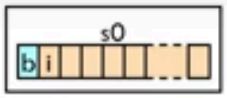


Print: 0

Print: 0

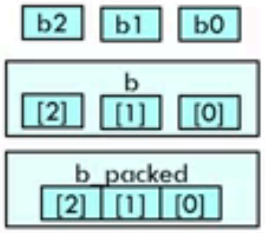
Print: '{b=0,i=0}

```
module test ();

  struct packed { bit b; int i; } s0;
  initial begin
      $display( "%0b", s0.b );
      $display( "%0d", s0.i );
      $display( "%0p", s0 );
      $display( "%0b", s0 );
  end

endmodule
```



Print: 0

```
module test ();
    bit         b0, b1, b2;
    bit         b[2:0];
    bit[2:0]    b_packed;
    byte        by0, by1, by2;
    byte        by[2:0];
    byte[2:0]   by_packed;
endmodule
```



Only single bit vector can be packed: bit, logic, reg, wire

```
module test ();
    bit     b[2:0];
    bit     b_le[2:0];
    bit     b_be[0:2];
    bit     b_cs[3];
    initial begin
        b[1] = b_le[1];
        b = b_le;
        b[2:1] = b_le[1:0];
        b = b_be;
    end
endmodule
```

Little endian style

Big endian style

C program style; same as big endian

Assignment is allowed as long as they are of the same size

Assignment with slices of array

Be careful:
b[2] = b_be[0]
b[1] = b_be[1]
b[0] = b_be[2]

```
module test (); // No automatic syntax
  function void increment();
    int a;
    a = a+1;
  endfunction
  initial begin
    increment();
    increment();
    increment();
  end
endmodule
```

a is "hardware" signal; it exists from the beginning till the end.

a = 1

a = 2

a = 3

```
module automatic test ();
  function automatic void increment();
    int a;
    a = a+1;
  endfunction
  initial begin
    increment();
    increment();
    increment();
  end
endmodule
```

a is "software" variable; it's created when the function is called and destroyed when the function call finishes.

a = 1

a = 1

a = 1

```
module test (); // Function call


  function void fSimple( int a, int b );
    $display( "a=%0d, b=%0d", a, b );
  endfunction


  function void fDefArgVal( int a, int b=1 );
    $display( "a=%0d, b=%0d", a, b );
  endfunction
```

```
  initial begin

    fSimple( 0, 1 );
    fSimple( .a(0), .b(1) );

    fDefArgVal( 0, 1 );          // a=0, b=1
    fDefArgVal( .a(0), .b(1) );// a=0, b=1
    fDefArgVal( 0 );             // a=0, b=1(default)
    fDefArgVal( .a(0) );         // a=0, b=1(default)

  end
endmodule
```

Argument values provided by position a=0, b=1

Argument values provided by name

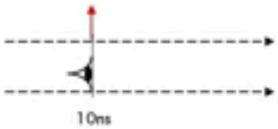
```
module test (); // fork join_any
  initial begin
    fork
      #10ns $display( "Doing A" );
      #20ns $display( "Doing B" );
    join_any
    $display( "Doing C" );
  end
endmodule
```

```
module test ( input logic reset); // Usage: timeout
  initial begin
    fork
      @( posedge reset );
      #100ns $error( "Timeout" );
    join_any
    disable fork;
    $display( "Out from fork-join_any" );
  end
endmodule
```

Reset hit first; or timeout

Disable active threads at exit

```
module test (); // trigger/wait event
  event ev;
  initial begin
    fork
      #10ns ->( ev );
      #10ns @( ev );
    join
    $display( "done" );
  end
endmodule
```



Wait for event that happens "after" me.

```
module test ();
  event ev;
  initial begin
    fork
      #10ns ->( ev );
      #10ns wait ( ev.triggered );
    join
    $display( "done" );
  end
endmodule
```

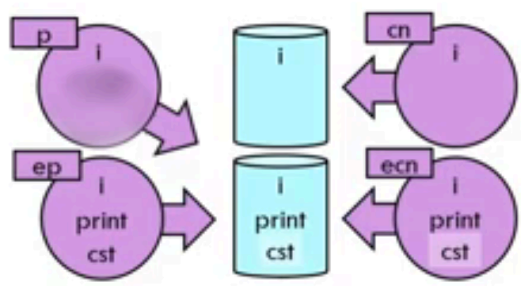
Wait for event that happens "after or same time as" me.

```
class cNumber; // static member

  int id;
  static bit debug=0;

  function new( int arg );
    id = arg;
    if ( debug ) $display( "%0d", id );
  endfunction

endclass
```

```
module test (); // Central debug example
  cNumber cn0;
  cNumber cn1;
  cNumber cn2;
  initial begin
    cNumber::debug = 1;
    cn0 = new(0);
    cn1 = new(1);
    cn2 = new(2);
  end
endmodule
```

```
class cNumber;
  rand int i;
endclass

class ecNumber extends cNumber;
  constraint cst {
    i%2 == 1;
  }
  virtual function void print();
    $display( "i=%0h", i );
  endfunction
endclass
```

```
module test ();
  cNumber     cn;
  ecNumber    ecn;
  cNumber     p;
  ecNumber    ep;

  initial begin
    cn = new(); ecn = new();
    p = ecn;
    ep = p; // same as ep = ecn;
  end
endmodule
```



$cast(ep, p):
Can use cast to check compatibility.
If it's compatible, cast will perform the assignment

```
module test();

  bit[2:0] b;

  covergroup cg;
    cp_b: coverpoint b {
      bins b_0to1 = { 0,1 };
      bins b_2to5 = { [2:5] };
      bins b_others = default;
    }
  endgroup
```

Summary for Variable cp_b

CATEGORY	EXPECTED	UNCOVERED	COVERED	PERCENT
User Defined Bins	2	0	2	100.00

User Defined Bins for cp_b

Excluded / Illegal Bins

NAME	COUNT	STATUS
b_others	1	Excluded

Covered Bins

NAME	COUNT	AT_LEAST
b_0to1	2	1
b_2to5	5	1

```
module test();

  bit[2:0] b;

  covergroup cg;
    cp_b: coverpoint b {
      bins b_0to1 = { 0,1 };
      bins b_2to5 = { [2:5] };
      illegal_bins b_illegal = { 6,7 };
    }
  endgroup
```

Summary for Variable cp_b

CATEGORY	EXPECTED	UNCOVERED	COVERED	PERCENT
User Defined Bins	2	0	2	100.00

User Defined Bins for cp_b

Excluded / Illegal Bins

NAME	COUNT	STATUS
b_illegal	1	Illegal

Covered Bins

NAME	COUNT	AT_LEAST
b_0to1	1	1
b_2to5	3	1

Coverage may still be 100% even if illegal bin is hit

If an illegal bin is hit, simulation will be stopped

```
module test();

  bit[2:0] b;

  covergroup cg;
    cp_b: coverpoint b {
      bins b_0then1 = (0=>1);
      bins b_2then3or4 = (2=>3,4);
      bins b_5then6then7 = (5=>6=>7);
    }
  endgroup
```

Summary for Variable cp_b

CATEGORY	EXPECTED	UNCOVERED	COVERED	PERCENT
User Defined Bins	3	2	1	33.33

User Defined Bins for cp_b

Uncovered Bins

NAME	COUNT	AT_LEAST	NUMBER
b_0then1	0	1	1
b_5then6then7	0	1	1

Covered Bins

NAME	COUNT	AT_LEAST
b_2then3or4	1	1

```
module test();

  bit[2:0] b;

  covergroup cg;
    cp_b: coverpoint b {
      wildcard bins b_odd = {3'b??1};

    }
  endgroup
```

Summary for Variable cp_b

CATEGORY	EXPECTED	UNCOVERED	COVERED	PERCENT
User Defined Bins	4	1	3	75.00

User Defined Bins for cp_b

Uncovered Bins

NAME	COUNT	AT_LEAST	NUMBER
b_odd_7	0	1	1

Covered Bins

NAME	COUNT	AT_LEAST
b_odd_1	2	1
b_odd_3	1	1
b_odd_5	2	1

```
covergroup cg;
    cp_a: coverpoint a { // bit[2:0] a
        bins a_0to3 = { [0:3] };
        bins a_4to7 = { [4:7] };
    }
    cp_b: coverpoint b { // bit[2:0] b
        bins b_0to3 = { [0:3] };
        bins b_4to7 = { [4:7] };
    }
    cx_ab: cross a, b;
endgroup
```

Uncovered Bins

a	b	COUNT	AT_LEAST	NUMBER
[auto[1]-auto[3]]	[auto[2] – auto[3]]	-	-	6
[auto[2]]	[auto[1]]	0	1	1
[auto[3]]	[auto[0]]	0	1	1

Covered Bins

a	b	COUNT	AT_LEAST
auto[1]	auto[0]	1	1
auto[1]	auto[1]	2	1
auto[2]	auto[0]	1	1
auto[3]	auto[1]	1	1

```
covergroup cg;
    cp_a: coverpoint a { // bit[2:0] a
        bins a_0to3 = { [0:3] };
        bins a_4to7 = { [4:7] };
    }
    cp_b: coverpoint b { // bit[2:0] b
        bins b_0to3 = { [0:3] };
        bins b_4to7 = { [4:7] };
    }
    cx_ab: cross cp_a, cp_b;
endgroup
```

Uncovered Bins

a	b	COUNT	AT_LEAST	NUMBER
a_0to3	b_4to7	0	1	1

Covered Bins

a	b	COUNT	AT_LEAST
a_0to3	b_0to3	2	1
a_4to7	b_0to3	3	1
a_4to7	b_4to7	3	1

```
covergroup cg;
    cx_ab: cross a, b {
        bins a0b0 = '{ '{0,0} };
        bins a67b67 = '{ '{6,6} , '{6,7} , '{7,6} , '{7,7} };
        bins a67b67_too = binsof( a ) intersects { [6:7] } &&
                          binsof( b ) intersects { [6:7] } ;
    }
endgroup
```

A coverpoint user defined bins will replace the auto-generated bins

A cross coverage user defined bins will not. Auto generated bins and user defined bins both exist.