systemVerilog 语法

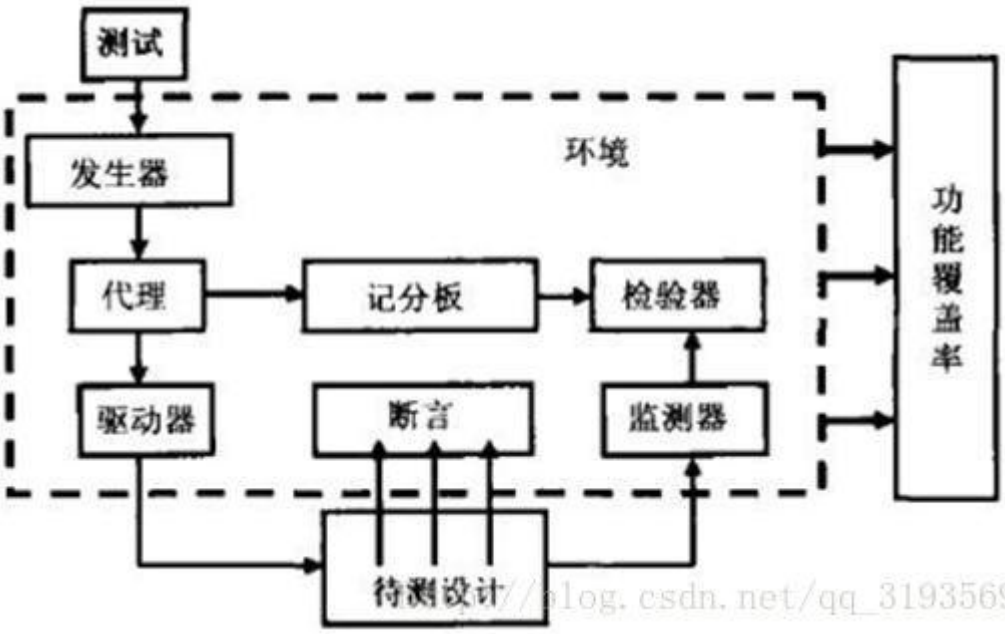
一、验证导论：

作为一个验证工程师，最重要的原则是“程序漏洞利大于弊”

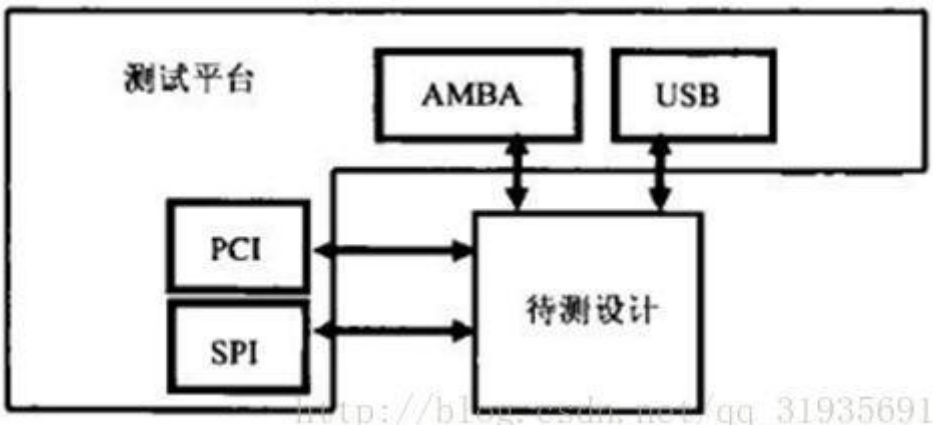
设计流程：1）阅读硬件规范，解析其中的自然语言表述。2）使用 RTL 代码之类的机器语言创建相应的逻辑。

验证流程：1）阅读硬件规范，制定验证计划。2）创建测试来检查 RTL 代码是否实现所有特性。

单一的 BFM 测试平台：



多个 BFM 测试平台：



二、数据类型：

1、Logic：

任何使用 wire 或者 reg 的信号在测试平台是都可使用 logic。（注意：对于双

向总线的信号不能用 logic，只能用 wire）

2、双状态数据类型：

有利于提高仿真器的性能并减少内存。最简单的双状态类型是 bit，无符号的。另外 4 种带符号位的双状态数据类型：byte、shortint、int、longint。 例：

```
bit b;                          // 双状态，单比特
bit [31:0] b32;                 // 双状态，32 比特无符号整数
int unsigned ui;                // 双状态，32 比特无符号整数
int i;                          // 双状态，32 比特有符号整数
byte b8;                        // 双状态，8 比特有符号整数
shortint s;                     // 双状态，16 比特有符号整数
longint l;                      // 双状态，64 比特有符号整数
integer i4;                     // 四状态，32 比特有符号整数
time t;                         // 四状态，64 比特无符号整数
real r;                         // 双状态，双精度浮点数
```


扩展：对四态信号的检查：（$isunknown） 例：对 iport 信号的检测

```
if ($isunknown(iport) == 1)
   $display ("@%0t: 4-state value detected on iport %b",
             $time,iport);
```


3、定宽数组：

例：

```
int lo_hi[0:15];            // 16 个整数 [0]...[15]
int c_style[16];            // 16 个整数 [0]...[15]

int array2 [0:7][0:3];      // 完整的声明
int array3 [8][4];          // 紧凑的声明
array2[7][3]=1;             // 设置最后一个元素
```


如果越界读取数据，则返回元素类型的缺省值，对于 logic 型的将返回 X，对于双状态的则返回 0。线网未驱动的输出 Z。

4、常量数据：

例：

```
int ascend [4] = '{0,1,2,3};            // 对 4 个元素进行初始化
int descend [5];

descend = '{4,3,2,1,0};                 // 为 5 个元素赋值
descend [0:2] = '{5,6,7};               // 为前 3 个元素赋值
ascend = '{4{8}};                       // 四个值全部为 8
descend = '{9,8,default:1};             // {9,8,1,1,1}
```

三、数组操作

1、操作数组的最常见的方式是使用 for 或 foreach 循环。Foreach 会自动遍历数组中的元素。

例：

```
initial begin
   bit [31:0] src[5],dst[5];
   for (int i=0;i<$size(src);i++)
        src[i]=i;
   foreach (dst[j])
      dst[j]=src[j] *  2; // dst 的值是 src 的两倍
   end
```


2、基本数组操作-复制和比较

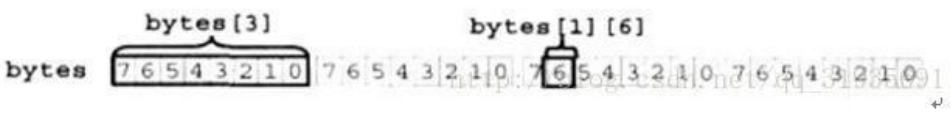
例：

```
initial begin
  bit [31:0] src[5] ='{0,1,2,3,4},
             dst[5] ='{5,4,3,2,1};

  // 两个数组的聚合比较

 if (src==dst)
       $display("src==dst");
 else
   $display("src!=dst");

 // 把 src 所有元素值复制给 dst
 dst=src;
```


3、合并数组

例：

```
bit [3:0] [7:0] bytes;              // 4 个字节组装成 32 比特
bytes=32'hCafe_Dada;
$displayh (bytes,,                  // 显示所有的 32 比特
          bytes[3],,                // 最高字节“CA”
          bytes[3][7]);             // 最高比特位“1”
```



四、动态数组

动态数组在声明的时候使用空的下标[]。使用时需调用 new[]操作符来分配空间。

例：

```
int dyn[],d2[];                    // 声明动态数组

initial begin
    dyn=new[5];                    // A:分配 5 个元素
    foreach (dyn[j]) dyn[j]=j;     // B:对元素进行初始化
    d2=dyn;                        // C:复制一个动态数组
    d2[0]=5;                       // D:修改复制值
    $display(dyn[0],d2[0]);        // E:显示数值(0 和 5)
    dyn=new[20](dyn);              // F:分配 20 个整数值并进行复制
    dyn=new[100];                  // G:分配 100 个新的整数值

                                       // 旧值不复存在
    dyn.delete();                      // H:删除所有元素
end
```


五、队列

队列的声明使用带有美元符号的下标：[]。队列元素的编号从 0 到。

例：

```
int j=1,
    q2[$]={3,4},               // 队列的常量不需要使用“'”
    q[$]={0,2,5};              // {0,2,5}

initial begin
    q.insert(1,j);             // {0,1,2,5}      在 2 之前插入 1
    q.insert(3,q2);            // {0,1,2,3,4,5}  在 q 中插入一个队列
    q.delete(1);               // {0,2,3,4,5}    删除第 1 个元素

// 下面的操作执行速度很快
q.push_front(6);               // {6,0,2,3,4,5}  在队列前面插入
j=q.pop_back;                  // {6,0,2,3,4}    j=5
q.push_back(8);                // {6,0,2,3,4,8}  在队列末尾插入
j=q.pop_front;                 // {0,2,3,4,8}    j=6
foreach (q[i])
    $display(q[i]);            //                打印整个队列
q.delete();                    // {}             删除整个队列
end
```


六、数组的排序

例：

```
int d[] =                    '{9,1,8,3,4,4};
d.reverse();        //       '{4,4,3,8,1,9}
d.sort();           //       '{1,3,4,4,8,9}
d.rsort();          //       '{9,8,4,4,3,1}
d.shuffle();        //       '{9,4,3,8,1,4}
```


七：使用 typedef 创建新的类型

例：下图的例子可以适应不同的比特位宽。

```
parameter OPSIZE=8;
typedef reg[OPSIZE-1:0] opreg_t;

opreg_t op_a,op_b;
```


八、枚举类型

由于宏的范围太大，故出现枚举类型，只能用于本模块中。 例：

```
typedef enum {INIT,DECODE,IDLE} fsmstate_e;
fsmstate_e pstate,nstate;          // 声明自定义类型变量

initial begin
   case(pstate)
      IDLE: nstate=INIT;           // 数据赋值
      INIT: nstate=DECODE;
      default: nstate=IDLE;
   endcase
   $display("Next state is %s",
            nstate.name());        // 显示状态的符号名
end
```


九、过程语句和子程序

1、过程语句：begin•••end、fork•••join。

用于循环的 continue：表示跳过本轮循环剩下的语句直接进入下一轮循环。

Break：用于终止并跳出循环。

2、任务 task、函数 function 以及 void 函数

任务消耗时间而函数不消耗。如果你想调用函数并且忽略它的返回值，可以使用 void 进行结果转换。 例：

```
function void print_state(...);
    $display("@%0t: state=%s",$time,cur_state.name());
endfunction
```


3、在子程序中去掉 begin•••end

例：

```
task multiple_lines;
    $display("First line");
    $display("Second line");
endtask : multiple_lines
```

4、带参数方向的声明

例：

```
task T3(a,b,output bit [15:0] u,v);
```

其中 a,b 是比特为 1 的 logic 的输入。

5、利用 ref 和 const 传递数组

例：

```
function void print_checksum (const ref bit [31:0] a[]);
    bit [31:0] checksum=0;
    for (int i=0;i<a.size();i++)
        checksum ^=a[i];
    $display("The array checksum is %0d",checksum);
endfunction
```

传递数组

其中 ref 参数只能用于自动存储的子程序中，在模块中定义 automatic 属性。Const ref 定义的话数组不能被子程序修改。

6、子程序的返回

例：

```
task load_array(int len,ref int array[]);
    if (len<=0) begin
        $display("Bad len");
        return;
    end

    // 任务中其余的代码
    ...
endtask
```


使用 ref 参数的形式将数组传递到函数里，例：

```
program automatic test;
    task wait_for_mem(input [31:0] addr,expect_data,
                      output success);
        while (bus.addr!==addr)
            @(bus.addr);
        success=(bus.data==expect_data);
    endtask
...
endprogram
```


十、自动存储

利用修饰符 automatic 自动存储

例：

```
program automatic test;
    task wait_for_mem(input [31:0] addr,expect_data,
                      output success);
        while (bus.addr!==addr)
            @(bus.addr);
        success=(bus.data==expect_data);
    endtask
...
endprogram
```


十一、时间值

1、 时间单位和精度

`timescale 1ns/100ps

2、 时间参数：

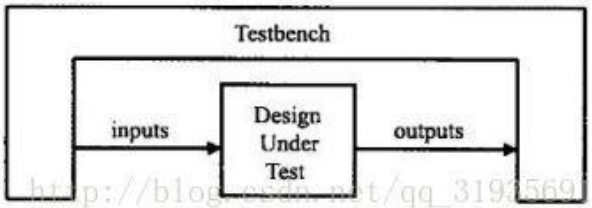
time、timeformat（-9 代表纳秒，-12 代表皮秒，3 代表时间精度（保留 3 位小数））、$realtime 。例：

```
module timing;
   timeunit 1ns;
   timeprecision 1ps;
   initial begin
         $timeformat(-9,3,"ns",8);
         #1 $display("%t",$realtime);// 1.000ns
         #2ns $display("%t",$realtime);// 3.000ns
         #0.1ns $display("%t",$realtime);// 3.100ns
         #41ps $display("%t",$realtime);// 3.141ns
   end
endmodule
```

十二、连接设计和测试平台

验证设计的几个步骤：生成输入激励，捕获输出响应，决定对错和衡量进度 。

测试平台-设计环境：



1、 使用接口将所有信号捆绑起来

例：

```
interface arb_if(input bit clk);
    logic [1:0] grant,request;
    logic rst;
endinterface
```

在 test 中使用接口：其中 arbif 越短越好 。例：

```
module test (arb_if arbif);
...
  initial begin
      // 此处省略了复位代码

      @(posedge arbif.clk);
      arbif.request<=2'b01;
      $display("@%0t: Drove req=01",$time);
      repeat (2)@(posedge arbif.clk);
      if (arbif.grant!=2'b01)
          $display("@%0t: a1: grant!=2'b01",$time);
      $finish;
  end
endmodule : test
```


在 top 中使用接口。 例：

```
module top;
  bit clk;
  always #5 clk=~clk;

  arb_if arbif(clk);        // 例 4.4
  arb a1 (arbif);           // 例 4.5
  test t1(arbif);           // 例 4.6
endmodule : top
```

2、 使用 modport 将接口中的信号分组

例：

```
interface arb_if(input bit clk);
    logic [1:0] grant,request;
    logic rst;

    modport TEST (output request,rst,
                  input grant,clk);
```


3、 使用时钟块控制同步信号的时序

一个接口可以包含多个时钟块，每个时钟块对应一个时钟域。还可以在时钟块中使用 default 语句指定时钟偏移。 例：

```
interface arb_if(input bit clk);
    logic [1:0] grant,request;
    logic rst;

    clocking cb @(posedge clk);// 声明 cb
        output request;

        input grant;
    endclocking

    modport TEST (clocking cb,// 使用 cb
                  output rst);
    modport DUT (input request,rst,output grant);
endinterface
```


4、 接口中的 logic 和 wire 对比

在接口中使用过程赋值语句驱动一个异步信号，那么该信号必须是 logic 类型的。Wire 类型变量只能被连续赋值语句驱动。 例：如何驱动接口中的 logic 和 wire 信号 ：

```
interface asynch_if();
    logic l;
    wire w;

endinterface

module test(asynch_if ifc);
    logic local_wire;
    assign ifc.w=local_wire;
```

5、 时钟延时的两种表示方法：

#10；addr=8’h42;

Repeat（2）@arbif.cb;

6、 通过时钟块驱动接口信号：

例：

```
busif.cb.request<=1;// 同步驱动
busif.cb.cmd<=cmd_buf;//同步驱动
```

7、 接口中的双向信号：

例：

```
interface master_if (input bit clk);
    wire[7:0] data;// 双向信号
    clocking cb @(posedge clk);
        inout data;
    endclocking

    modport TEST (clocking cb);
endinterface
```


8、 时钟发生器：

不能把时钟发生器放在程序块中，正确时钟发生器，例：

```
module clock_generator (output bit clk);
  initial
  always #5 clk=~clk;// 在时间 0 之后生成时钟沿
endmodule
```


十三：systemverilog 断言（SVA）

1、 立即断言：

检测设计或者测试模块中信号的正确性 。例：检测 grant 信号的正确性 。

```
bus.cb.request<=1;
repeat (2) @bus.cb;
a1: assert (bus.cb.grant==2'b01);
```

如果正确产生 grant 信号，则继续执行，若不符合期望值，则报错。报错信息如下：

```
"test.sv",7: top.t1.a1: started at 55ns failed at 55ns
offending '(bus.cb.grant==2'b1)'
```


2、定制断言行为：

如果想改变默认的消息，可以添加自己的输出信息。 Systemverilog 有 4 个输出消息的函数：info,warning,error 和 fatal 。例：

```
a1: assert (bus.cb.grant==2'b01)
else $error("Grant not asserted");
```

报错信息如下：

```
"test.sv",7: top.t1.a1: started at 55ns failed at 55ns
Offending '(bus.cb.grant==2'b1)'
Error: "test.sv",7: top.t1.a1: 55 ns 时
Grant not asserted
```

3、并发断言：

你可以认为它是一个连续运行的模块，为整个仿真过程检查信号的值。例：

```
interface arb_if(input bit clk);
    logic [1:0] grant,request;
    logic rst;

    property request_2state;
        @(posedge clk) disable iff (rst);
        $isunknown(request)==0;        //确保没有 Z 或者 X 值存在
    endproperty
    assert_request_2state: assert property (request_2state);
endinterface
```

十四、面向对象的基础（oop）

1、 oop 术语：

类（class）：包含变量和子程序的基本构建块。

对象（object）：类的实例

句柄（handle）：指向对象的指针。一个句柄可以指向很多对象。

属性（property）：存储数据的变量

方法（method）：任务或者函数中操作的程序性代码

原型（prototype）：程序的头，包括程序名、返回类型和参数列表

2、 声明和使用 handle：

例

```
Transaction tr;//声明一个句柄
Tr=new();    // 为一个 Transaction 对象分配空间
```

在需要调用类的地方定义句柄和分配空间

在声明句柄 tr 的时候，它被初始化为特殊值 null。

正确调用 new（）函数的方法：

例：

```
class Transaction;
      ...
endclass : Transaction

class Driver;
      Transaction tr;
      function new(); // Driver 的 new 函数
          tr=new(); // 调用 Transaction 的 new 函数
      endfunction
endclass : Driver
```


3、 new()和 new[]的区别

new[]操作建立一个含有多个元素的数组。new()可以使用参数设置对象的值，而 new[]只需使用一个数值来设置数组的大小。

4、 对象的解除分配

例：transaction t; // 创建一个 handle

t=new(); // 分配一个新的 transaction

t=new(); // 分配第二个，并且释放第一个 t

t=null; // 解除分配第二个

5、 使用对象的方法：

例：

```
Transaction t; // 声明一个 Transaction 句柄
t=new(); // 创建一个 Transaction 对象
t.addr=32'h42; // 设置变量的值
t.display(); // 调用一个子程序
```


6、 使用静态变量的方法

例：

```
class Transaction;
      static Config cfg; // 使用静态存储的句柄
      MODE_E mode;

      function new();
          mode=cfg.mode;
      endfunction
endclass

Config cfg;
initial begin
      cfg=new(MODE_ON);
      Transaction::cfg=cfg;
      ...
end
```

十五、类的方法

1、 类中的方法使用：

例：

```
class Transaction;
      bit [31:0] addr,crc,data[8];
      function void display();
            $display("@%0t: TR addr=%h,crc=%h",$time,addr,crc);
            $write("\tdata[0-7]=");
            foreach (data[i]) $write(data[i]);
            $display();
      endfunction
endclass

class PCI_Tran;
      bit [31:0] addr,data; // 使用真实的名字
      function void display();
            $display("@%0t: PCI: addr=%h,data=%h",$time,addr,data);
      endfunction
endclass
```


2、 类外的方法声明：

在开始处添加关键词 extern。然后在类定义后面，在方法名前加上类名和两个冒号::。 例：

```
class Transaction;
      bit [31:0] addr,crc,data[8];
      extern function void display();
endclass

function void Transaction::display();
      $display("@%0t: Transaction addr=%h,crc=%h",
            $time,addr,crc);
      $write("\tdata[0-7]=");
      foreach (data[i]) $write(data[i]);
      $display();
endfunction
```


3、 this 的使用方法：

当使用一个变量名的时候，systemverilog 将先在当前作用域内寻找，接着在上一级作用域内寻找，直到找到该变量为止。 例：

```
class Scoping;
     string oname;

     function new(string oname);
          this.oname=oname;      // 类变量 oname=局部变量 oname
     endfunction
endclass
```


4、 在一个类中使用另外一个类的方法。

通过使用指向对象的句柄，一个类内部可以包含另一个类的实例 。例：

类中调用类

```
class Transaction;
  bit [31:0] addr, crc, data[8];
  Statistics stats;
endclass

class Statistics;
 time startT, stopT;
 static int ntrans = 0;
 static time total_elapsed_time;
endclass
```


5、 注意正确放好 new 的位置 。例：

注意在循环中创建new需要正确放置好位置

```
task generator_good(int n);
     Transaction t;
     repeat (n) begin
          t=new(); // 创建一个新对象
          t.addr=$random(); // 变量初始化
          $display("Sending addr=%h",t.addr);
          transmit(t); // 将它发送到 DUT
     end
endtask
```


6、 使用 new 操作符复制一个对象

```
class Transaction;
     bit [31:0] addr,crc,data[8];
endclass

Transaction src,dst;
initial begin
     src=new; // 创建第一个对象
     dst=new src; // 使用 new 操作符进行复制
end
```


例：

使用 new 操作符复制复杂类 。例：

```
Transaction src,dst;
initial begin
    src=new();                      // 创建第一个对象
    src.stats.startT=42;            // 设置起始时间
    dst=src.copy();                 // 使用深层复制将 src 复制给 dst
    dst.stats.startT=96;            // 仅改变 dst 的 stats 值
    $display(src.stats.startT);     // "42",见图 5.8
    end
```


十六、随机化

通过随机化可以通过利用 CPU 的时间来换取人工检查的时间，提高效率，提供足够的激励。

采用受约束的随机测试法（CRT）产生测试集：使用随机的数据流为 DUT 产生输入的测试代码。改变伪随机数发生器（PRNG）的种子（seed）。

一般会在测试设计时考虑设计规范的边界处，甚至测试设计规范之外的行为。

1、 简单的随机变量的简单类

例：

```
class Packet;
   // 随机变量
   rand bit [31:0] src,dst,data[8];
   randc bit [7:0] kind;
   // src 的约束
   constraint c {src >10;
                 src <15;}
endclass
```

Randc 表示周期随机性，即所有的可能的值都赋值后随机值才可能重复

Randomize()函数在遇到约束方面的问题时返回 0.

2、 权重分布的约束

Dist 操作符允许产生权重分布。

：=操作符表示值范围内的每一个值的权重是相同的。

：/操作符表示权重要均分到值范围内的每一个值。

例：

```
rand int src,dst;
constraint c_dist {
  src dist {0:=40,[1:3]:=60};
  // src=0,weight=40/220
  // src=1,weight=60/220
  // src=2,weight=60/220
  // src=3,weight=60/220

  dst dist {0:/40,[1:3]:/60};
  // dst=0,weight=40/100
  // dst=1,weight=20/100
  // dst=2,weight=20/100
  // dst=3,weight=20/100
}
```

对变量的范围和出现次数的约束语句（dist和inside）

3、 集合成员和 inside 运算符

用 inside 运算符产生一个值的集合，还可以求反!，用$代表最大值和最小值

例：

```
rand int c; // 随机变量
int lo,hi; // 作为上限和下限的非随机变量
constraint c_range {
  c inside {[lo:hi]}; // lo < =c 并且 c<=hi
}
```

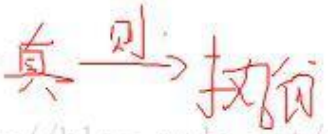

4、 条件约束

支持->和 if-else，其中->操作等价于 case 产生的效果

例：

```
class BusOp;
  ...
  constraint c_io {
    (io_space_mode) ->
      addr[31]==1'b1;
  }
```

条件 约束语句：-> 等价于if··else



5、 高效的约束办法 。

例：

```
rand bit [31:0] addr;
constraint near_page_boundry {
  addr[11:0] inside {[0:20],[4075:4095]};
}
```

inside约束例子

6、 带关系操作的约束办法

例：

```
class Impl;
  rand bit x;              // 0 或 1
  rand bit [1:0] y;        // 0,1,2 或 3
  constraint c_xy {
    (x==0) ->y==0;
  }
endclass
```


7、 带关系操作和约束的类

例：

```
class Imp2;
  rand bit x;              // 0 或 1
  rand bit [1:0] y;        // 0,1,2 或 3
  constraint c_xy {
  y > 0;
  (x==0) ->y==0;
  }
endclass
```


当 x==0 时，y==0，但 y==0 时，对 x 没有约束。

8、 使用 slove••••before 约束引导概率分布

slove••••before 约束不会改变解的个数，只会改变各个值的概率分布。 例：

```
constraint c_xy {
  (x==0) -> y==0;
  solve x before y;
}
```

9、 控制多个约束块

可以使用内建 constraint_mode（）函数打开或者关闭约束。

可以用 handle.constrain.constraint_mode（）控制一个约束块。

用 handle.constraint_mode（）控制对象的所有约束。

例：

```
class Packet;
  rand int length;
  constraint c_short {length inside {[1:32]}; }
  constraint c_long {length inside {[1000:1023]}; }
endclass

Packet p;
initial begin
  p=new();

  // 通过禁止 c_short 约束产生长包
  p.c_short.constraint_mode(0);
  assert (p.randomize());

  transmit(p);

  // 通过禁止所有的约束,然后使能短包约束来产生短包
  // then enabling only the short constraint
  p.constraint_mode(0);
  p.c_short.constraint_mode(1);
  assert (p.randomize());
  transmit(p);
end
```


10、内嵌约束（randomize() with）

通过 randomize（） with 来增加额外的约束 例：

```
class Transaction;
  rand bit [31:0] addr,data;
  constraint c1 {addr inside{[0:100],[1000:2000]};}
endclass

Transaction t;

initial begin
    t=new();

  // addr 范围：50-100,1000-1500,data <10
  assert(t.randomize() with {addr >= 50; addr <=1500;
                             data < 10;});

  driveBus(t);

  // 强制 addr 取固定值,data>10
  assert(t.randomize() with {addr==2000; data>10;});

  driveBus(t);
  end
```

11、pre_randomize 和 post_randomize 函数

构建浴缸分布的例子：

```
class Bathtub;
  int value; // 浴缸型分布的随机变量
  int WIDTH=50,DEPTH=4,seed=1;

  function void pre_randomize();
    // 计算指数曲线

    value=$dist_exponential(seed,DEPTH);
    if (value >WIDTH) value=WIDTH;

    // 把这一个点随机地放在左边或右边的曲线上
    if ($urandom_range(1))
      value=WIDTH - value;
   endfunction

endclass
```


12、常用的随机函数

(1) $random() —— 平均分布，返回 32 位有符号随机数；
(2) $urandom() —— 平均分布，返回 32 位无符号随机数；
(3) $urandom_range() —— 在指定范围内的平均分布；
(4) $dist_exponential() —— 指数衰落，如图 6.1 所示；
(5) $dist_normal() —— 钟型分布；
(6) $dist_poisson() —— 钟型分布；
(7) $dist_uniform() —— 平均分布。
$urandom_range()函数有两个参数，一个上限参数和一个可选的下限参数。

13、约束技巧和技术

1）使用变量设定上限的约束

```
class bounds;
  rand int size;
  int max_size=100;
  constraint c_size {
    size inside {[1:max_size]};
  }
endclass
```

2）带权重变量的 dist 约束

```
typedef enum {READ8,READ16,READ32} read_e;
class ReadCommands;
  rand read_e read_cmd;
  int read8_wt=1,read16_wt=1,read32_wt=1;
  constraint c_read {
    read_cmd dist {READ8 :=read8_wt,
                   READ16 :=read16_wt,
                   READ32 :=read32_wt};
  }
endclass
```


14、随机化中常见的错误

1）小心使用有符号变量（除非必要，不要在随机约束里使用有符号类型）：错误的例子如下 。例：

```
class SignedVars;
  rand byte pkt1_len,pk2_len;
  constraint total_len {
    pkt1_len +pk2_len==64;
  }
endclass
```

2）使用无符号 32 位变量随机化也将产生错误。

如：randm bit[31:0] pkt1_len.••••••••••••••错误

15、约束数组或者队列的大小

Size（）函数可以约束动态数组或队列的元素个数。Inside 约束数组大小的上限和下限 。例：

```
class dyn_size;
  rand logic [31:0] d[];
  constraint d_size {d.size() inside {[1:10]}; }
endclass
```


15、约束数组和队列的每一个元素

1）可以用 foreach 对数组的每一个元素进行约束。 例：

```
class good_sum5;
  rand uint len[];
  constraint c_len {foreach (len[i])
                    len[i] inside {[1:255]};
                    len.sum < 1024;
                    len.size() inside {[1:8]};}
endclass
```

正确的数组约束例子，注意变量必须是无符号位的

2）产生具有唯一元素值的数组 。例：

```
class UniqueSlow;
  rand bit [7:0] ua[64];
  constraint c {
    foreach (ua[i])              // 对数组的每个元素操作
      foreach (ua[j])
        if (i!=j)                // 除了元素自己
          ua[i]!=ua[j];          // 和其他元素比较
  }
endclass
```


16、使用 randcase 和$urandom_range()随机控制 。例：

```
initial begin
  int len;
  randcase
    1: len=$urandom_range(0,2); // 10% : 0,1,or 2
    8: len=$urandom_range(3,5); // 80% : 3,4,or 5
    1: len=$urandom_range(6,7); // 10% : 6 or 7
  endcase
```


$urandom_range 函数返回一个指定范围的随机数，若只用一个参数，则当做（0，最大值）对待。

使用 randcase 建立决策树 。例：

```
initial begin
  // Level 1
  randcase
    one_write_wt: do_one_write();
    one_read_wt: do_one_read();
    seq_write_wt: do_seq_write();
    seq_read_wt: do_seq_read();
  endcase
  end
// Level 2
task do_one_write;
  randcase
    mem_write_wt: do_mem_write();
    io_write_wt: do_io_write();
    cfg_write_wt: do_cfg_write();
  endcase
endtask

task do_one_read;
  randcase
    mem_read_wt: do_mem_read();
    io_read_wt: do_io_read();
    cfg_read_wt: do_cfg_read();
  endcase
endtask
```

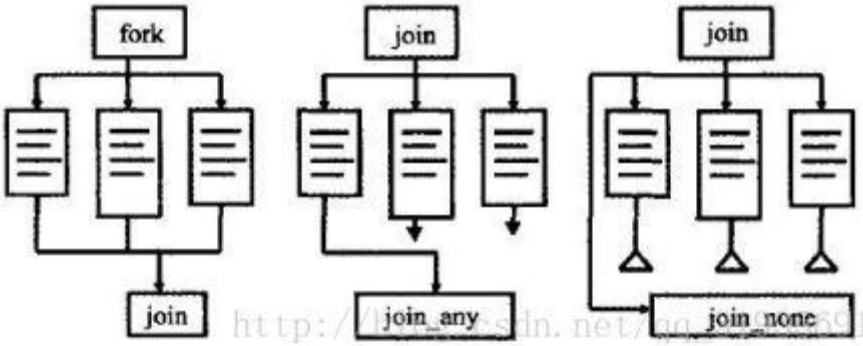
十七、线程

线程的量级比进程小，其代码和存储区可共享，而且所消耗的资源比典型的进程小的多。

1、 线程的使用

1） begin ••••end 顺序执行、fork•••join 并行执行

2） fork•••join、fork•••join_none、fork••••join_any 的区别



注意：1）fork•••join_none 块后的那个语句执行早于 fork_none 内的任何语句。

3） fork•••join_any 块对块内语句进行调度，当完成第一个语句后，父线程才继续执行，其他停顿线程也得继续执行。

2、 等待所有的衍生线程

可以通过 wait fork 语句来等待所有子线程结束 。例：

```
task run_threads;
    ...                          // 创建一些事务
    fork
        check_trans(tr1);    // 产生第一个线程
        check_trans(tr2);    // 产生第二个线程

         check_trans(tr3);    // 产生第三个线程
    join_none
    ...                          // 完成其他的工作
    // 在这里等待上述线程结束
    wait fork;
endtask
```


3、 在线程间共享变量

避免使用全局变量，这样容易导致漏洞。

4、 停止线程的方法

使用 disable 语句可以用于停止 systemverilog 中的线程 。停止单个线程：例

```
parameter TIME_OUT=1000;
task check_trans(Transaction tr);
    fork
        begin
            // 等待回应,或者达到某个最大时延
            fork : timeout_block
                begin
                    wait (bus.cb.addr==tr.addr);
                    $display("@%0t: Addr match%d",$time,tr.addr);
                end
                #TIME_OUT $display("@%0t: Error: timeout",$time);
            join_any
            disable timeout_block;
        end
    join_none
endtask
```


停止多个线程，例：

```
initial begin
    check_trans(tr0);                    // 线程 0
    // 创建一个线程来限制 disable fork 的作用范围
    fork                                 // 线程 1
        begin
            check_trans(tr1);            // 线程 2
            fork                         // 线程 3
              check_trans(tr2);          // 线程 4
            join
            // 停止线程 1—4,单独保留线程 0
            #(TIME_OUT/2) disable fork;
        end
    join
end
```


十八、线程间的通信

1、 事件

1）在事件边沿阻塞：例

```
event e1,e2;
initial begin
    $display("@%0t: 1: before trigger",$time);
    -> e1;
    @e2;
    $display("@%0t: 1: after trigger",$time);
end
initial begin
    $display("@%0t: 2: before trigger",$time);
    -> e2;
    @e1;
    $display("@%0t: 2: after trigger",$time);
end
```

其输出结果如下：

```
@0: 1: before trigger
@0: 2: before trigger
@0: 1: after trigger
```

2）等待事件的边缘触发方法

```
forever begin
    // 这里避免了零时延循环!
    @handshake;
    $display("Received next event");
    process_in_zero_time();
end
```


4） 传递事件

```
class Generator;
    event done;
    function new (event done);              // 从测试平台中传来事件
        this.done=done;
    endfunction

    task run();
        fork
            begin
                ...                         // 创建事务
                ->done;                     // 告知测试程序任务已完成
            end
        join_none
    endtask
endclass

program automatic test;
    event gen_done;
    Generator gen;
    initial begin
        gen=new(gen_done);                  // 测试程序实例化
        gen.run();                          // 运行事务处理器
        wait(gen_done.triggered());         // 等待任务结束
    end
endprogram
```


2、 旗语的使用方法

New 方法可以创建一个带当个或多个钥匙的旗语，使用 get 可以获取一个或多个钥匙，而 put 则可以返回一个或多个钥匙，如果试图获取一个旗语而不希望

被阻塞，可以使用 try_get()函数 。例：

```
program automatic test(bus_ifc.TB bus);
    semaphore sem;                  // 创建一个旗语
    initial begin
        sem=new(1);                 // 分配 1 个钥匙
        fork
            sequencer();            // 产生两个总线事务线程
            sequencer();
        join
end

task sequencer;
    repeat($urandom%10)             // 随机等待 0-9 个周期
        @bus.cb;
    sendTrans();                    // 执行总线事务
endtask

task sendTrans;
    sem.get(1);                     // 获取总线钥匙
    @bus.cb;                        // 把信号驱动到总线上
    bus.cb.addr <=t.addr;
    ...
    sem.put(1);                     // 处理完成时把钥匙返回
    endtask
endprogram
```


如果需要使用带多个钥匙的旗语，需要注意：

1）你返回的钥匙可以比你取出来的多。

2）当你的测试程序需要获取和返回多个钥匙时，务必谨慎。

3、 信箱的使用方法

两个线程之间传递信息可以通过信箱。信箱可以简单的理解为一个具有源端和收端的 fifo。

信箱是一种对象，必须调用 new 函数来进行实例化

使用 put 任务可以把数据放入信箱里，而使用 get 任务则可以移除数据。如果信箱为满，则 put 会阻塞；如果为空，则 get 会阻塞。Peek 任务可以获取对信箱里数据的拷贝而不移除它。

注意：可以在信箱中放入句柄，但不能是对象。务必在一个信箱里只放入一种类型的数据。

创建多个对象的发生器的正确方法：例

```
task generator_good(int n,mailbox mbx);
    Transaction t;
    repeat (n) begin
        t=new();                    // 创建一个新的事务
        assert(t.randomize());      // 对变量进行随机化
        $display("GEN: Sending addr=% h",t.addr);
        mbx.put(t);                 // 把事务发送给驱动器
    end
endtask
```


接收信箱的事务驱动器：例

```
task driver(mailbox mbx);
    Transaction t;
    forever begin
        mbx.get(t);                 // 获取来自信箱的事务
        $display("DRV: Received addr=% h",t.addr);
        // 驱动事务到待测设计中
    end
endtask
```


使用信箱实现对象的交换，例：

```
class Driver;
    Transaction tr;
    mailbox mbx;

    function new(mailbox mbx);
        this.mbx=mbx;
    endfunction
    task run(int count);
        repeat (count) begin
            mbx.get(tr);            // 提取下一个事务
            @(posedge bus.cb.ack);
            bus.cb.kind <=tr.kind;
            ...
        end
    endtask
endclass
```


使用信箱和事件来实现线程同步的方法：

```
program automatic mbx_evt;
mailbox mbx;  event handshake;
    class Producer;
        task run;

            for (int i=1; i<4; i++ ) begin
                $display("Producer: before put(%0d)",i);
                mbx.put(i);
                @handshake;
                $display("Producer: after put(%0d)",i);
            end
    endtask
endclass

    class Consumer;
        task run;
            int i;
            repeat (3) begin
                mbx.get(i);
                $display("Consumer: after get(%0d)",i);
                ->handshake;
            end
      endtask
    endclass:Consumer

    Producer p;
    Cosumer c;

    initial begin
      mbx=new();
        p=new();
        c=new();
        // 使生产方和消费方并发运行
        fork
            p.run();
            c.run();
        join
      end
    endprogram
```


输出结果为：

```
Producer: before put(1)

Consumer: after get(1)
Producer: after put(1)
Producer: before put(2)
Consumer: after get(2)
Producer: after put(2)
Producer: before put(3)
Consumer: after get(3)
Producer: after put(3)
```

十九、面向对象编程的高级技巧

继承：允许从一个现存的类得到一个新的类并共享其变量和子程序。原始类被称为基类或者超类，而新类因为扩展了基类的功能，被称为扩展类。 OOP 真正强大的地方在于它可以使你继承现有类。

1、事务基类。 例：

```
class Transaction;
    rand bit[31:0]src,dst,data[8];   // 随机变量
    bit[31:0]crc;                    //计算得到的 CRC 值

    virtual function void calc_crc;
        crc=src^dst^data.xor;
    endfunction

    virtual function void display(input string prefix="");
    $display("%sTr:src=%h,dst=%h,crc=%h",
             prefix,src,dst,crc);
    endfunction
endclass
```


2、扩展类：

```
class BadTr extends Transaction;
    rand bit bad_crc;

    virtual function void calc_crc;
        super.calc_crc(); // 计算正确的 CRC
        if(bad_crc)crc=~crc; // 产生错误的 CRC 位
    endfunction

    virtual function void display(input string prefix="");
        $write("%sBadTr:bad_crc=%b,",prefix,bad_crc);
        super.display();
    endfunction
endclass:BadTr
```

将类中的子程序定义成虚拟的，这样 它就可以在扩展类中重新定义，这一点适用于所有的任务和函数，除了 new 函数。

OOP 中类的变量称为属性，而任务或者函数称为方法。

3、 扩展类中带参数的构造函数（构造函数 new） 。例：

```
class Base1;
    int var;
    function new(input int var);      //带有参数的构造函数
        this.var=var;
    endfunction
endclass

class Extended extends Base1;
    function new(input int var);      //需要参数
        super.new(var);               // 必须是 new 函数的第一行
        // 构造函数的其他行为
    endfunction
endclass
```


4、 对象的复制

带 copy 虚函数的事务基类，例：

```
class Transaction;
    rand bit[31:0]src,dst,data[8];      // 变量
    bit[31:0]crc;

    virtual function Transaction copy();
        copy=new();
        copy.src=src;                   // 复制数据域
        copy.dst=dst;
        copy.data=data;
        copy.crc=crc;
    endfunction
endclass
```


带有 copy 虚函数的扩展事务类，例：

```
virtual class BaseTr;
    static int count;                          // 需要创建的实例数
    int id;                                    // 唯一的事务 id

    function new();
        id=count++;                            // 每一个对象对应一个 ID
    endfunction

    pure virtual function bit compare(input BaseTr to);
    pure virtual function BaseTr copy(input BaseTr to=null);
    pure virtual function void display(input string prefix="");
endclass:BaseTr
```


5、 抽象类和纯虚办法

使用纯虚方法的抽象类，例：

```
virtual class BaseTr;
    static int count;                          // 需要创建的实例数
    int id;                                    // 唯一的事务 id

    function new();
        id=count++;                            // 每一个对象对应一个 ID
    endfunction

    pure virtual function bit compare(input BaseTr to);
    pure virtual function BaseTr copy(input BaseTr to=null);
    pure virtual function void display(input string prefix="");
endclass:BaseTr
```


Transaction 类扩展抽象类，例：

```
class Transaction extends BaseTr;
    rand bit[31:0]src,dst,crc,data[8];

    extern virtual function bit compare(input BaseTr to);
    extern virtual function BaseTr copy(input BaseTr to=null);
    extern virtual function void copy_data
                            (input Transaction copy);
    extern virtual function void display(input string prefix="");
    extern function new();
endclass
```


Transaction 类的实体部分，例：

```
function bit Transaction::compare(input BaseTr to);
    Transaction tr;
    assert($cast(tr,to));                    // 检查 to 是否为正确类型
    return ((this.src==tr.src)&&
            (this.dst==tr.dst)&&
            (this.crc==tr.crc)&&
            (this.data==tr.data));
endfunction:compare

function BaseTr Transaction::copy(input BaseTr to=null);
    Transaction cp;
    if(to==null)cp=new();
    else     $cast(cp,to);
    copy_data(cp);
    return cp;
endfunction

function void Transaction::copy_data(Transaction copy);
    copy.src=src;          // 复制数据域
    copy.dst=dst;
    copy.data=data;
    copy.crc=crc;
endfunction

function void Transaction::display(string prefix="");
    $display("%sTransaction%0d src=%h,dst=%%x,crc=%x",
    prefix,id,src,dst,crc);
endfunction:display;

function Transaction::new();
    super.new();
endfunction:new
```


二十、功能覆盖率

功能覆盖率是用来衡量哪些设计特征已经被测试程序试过的一个指标。

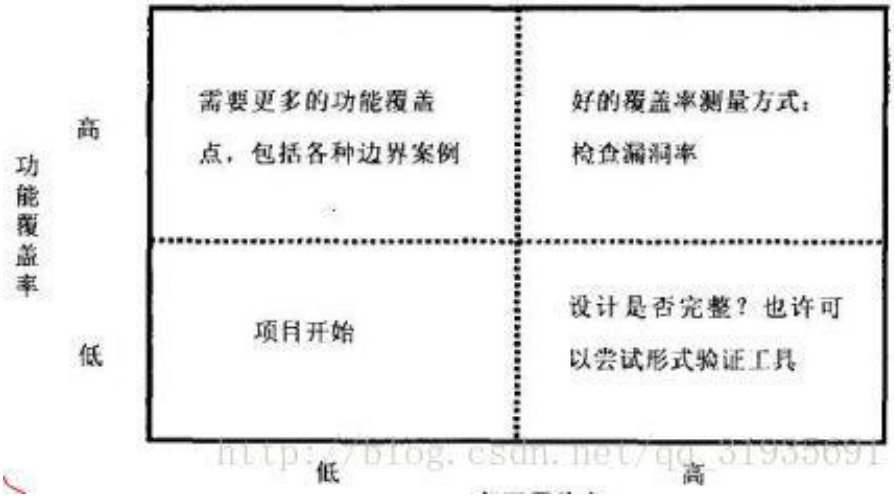
1、 覆盖率的类型：

1） 代码覆盖率：衡量多少代码已经被执行过。一般都通过调用代码覆盖率的命令即可实现。

2） 功能覆盖率：确保设计实现了所有的特性。

3）断言覆盖率：一般检查设计中的关键信号是否正确执行，特别是时序检查时需用到断言。

2、覆盖率一般出现的情况，根据下图进行相应的操作，例：



2、 功能覆盖率的实现方法：例

```
program automatic test(busifc.TB ifc);
    class Transaction;
        rand bit[31:0]data;
        rand bit[ 2:0]port;              // 八种端口(port)数据
    endclass

    covergroup CovPort;
        coverpoint tr.port;              // 测量覆盖率
    endgroup

    initial begin
        Transaction tr;
        CovPort ck;
        ck=new();                        // 实例化组
        tr=new();
        repeat(32)begin                  // 运行几个周期
            assert(tr.randomize);        // 创建一个事务
            ifc.cb.port<=tr.port;        // 并发送到接口上
            ifc.cb.data<=tr.data;
            ck.sample();                 // 收集覆盖率
            @ifc.cb;                     // 等待一个周期
        end
    end
endprogram
```


覆盖率的报告，如下图：

```
Coverpoint Coverage report
CoverageGroup:CovPort
    Coverpoint:tr.port
Summary
    Coverage:87.50
    Goal:100
    Number of Expected auto- bins:8
    Number of User Defined Bins:0
    Number of Automatically Generated Bins:7
    Number of User Defined Transitions:0

    Automatically Generated Bins

    Bin         # hits        at least
    ..........  ..........    ..........
    auto[1]         7             1
    auto[2]         7             1
    auto[3]         1             1
    auto[4]         5             1
    auto[5]         4             1
    auto[6]         2             1
    auto[7]         6             1
    ..........  ..........    ..........
```


3、 类中的功能覆盖率实现方法：例

```
class Transactor;
    Transaction tr;

  mailbox mbx_in;
  covergroup CovPort;
      coverpoint tr.port;
  endgroup

  function new(mailbox mbx_in);
      CovPort=new();                  // 实例化覆盖组
      this.mbx_in=mbx_in;
  endfunction

  task main;
      forever begin
          tr=mbx_in.get;              // 获取下一个事务
          ifc.cb.port<=tr.port;       // 发送到待测设计中
          ifc.cb.data<=tr.data;
          CovPort.sample();           // 收集覆盖率
      end
  endtask
  endclass
```

4、 使用事件触发的覆盖率，例：

```
event trans_ready;
covergroup CovPort @(trans_ready);

    coverpoint ifc.cb.port;          // 测量覆盖率
endgroup
```


5、 使用 SVA 触发的覆盖率组，例：

```
program automatic test(simple_bus sb);

    covergroup Write_cg @($root.top.m1.write_event);
        coverpoint $root.top.m1.data;
        coverpoint $root.top.m1.addr;
    endgroup
    Write_cg wcg;

    initial begin
        wcg=new();
        // 在此处添加激励
        sb.write_ena<=1;
        ...
        #10000 $finish;
    end
endprogram
```


6、 使用条件覆盖率的方法：例

```
    covergroup CoverPort;
        //当 reset==1 时不要收集覆盖率数据
        coverpoint port iff(!bus_if.reset);
    endgroup
```


7、 枚举类型的功能覆盖率，例：

```
typedef enum{INIT,DECODE,IDLE}fsmstate_e;
fsmstate_e pstate,nstate;          // 声明自有类型变量
covergroup cg_fsm;
    coverpoint pstate;
endgroup
```


二十一、高级接口

虚接口（virtual interface）是一个物理接口的句柄（handle）。

虚接口和对应的通用方法可以把设计和验证平台分隔开来，保证其不受设计改动的影响。当我们对一个设计引脚名字进行改动时，无须改动驱动这个接口的方法，而只需在例化该实物交易处理器的时候，给虚接口绑定对应连接的实体接口即可。以此来实现实物交易处理器的更大重用性。

虚接口的定义：virtual interface_type name；

虚接口可以定义为类的一成员，可以通过构建函数的参数或者过程进行初始化。

例：

```
Interface sbus;
    Logic req,grant;
    Logic[7:0] addr ,data;
Endinterface
Class sbustransaction;
    Virtual sbus vif;
    Function new(virtual sbus s);
        Vif=s;
    Endfunction

  Task request();
      Vif.req <=1’b1;
  Endatsk

  Task wait_for_bus();
      @(posedge vif.grant);
  Endtask
endclass
```

二十二、systemverilog 与 c 语言的接口

Systemverilog 引入直接编程接口(DPI),它能更加简单的连接 C,C++或者其他非 Verilog 编程语言。一旦你声明或者使用 import 语句导入一个 C 子程序，你就可以像调用 systemverilog 的子程序一样调用它。

1、例：systemverilog 代码调用 C 语言子程序 factorial

```
import "DPI-C" function int factorial(input int i);

program automatic test;
    initial begin
        for (int i=1; i<=10; i++)
            $display("%0d! =%0d", i, factorial(i));
    end
endprogram
```


2、C 语言 factorial 函数

```
int factorial(int i) {
    if (i<=1) return 1;
    else    return i* factorial(i-1);
}
```


3、参数方向

```
import "DPI- C" function int addmul ( input int a, b,
                                      output int sum);
import "DPI- C" function void stop_model();
```


4、 systemverilog 和 C 语言之间的数据类型映射

SystemVerilog	C(输入)	C(输出)
byte	char	char*
shortint	short int	short int*
int	int	int*
longint	long long int	long int*
shortreal	float	float*
real	double	double*
string	const char*	char**
string[N]	const char**	char**
bit	svBit or unsigned char	svBit* or unsigned char
logic, reg	svLogic or unsigned char	svLogic* or unsigned char*
bit[N:0]	const svBitVecVal*	svBitVecVal*
reg[N:0] logic[N:0]	const svLogicVecVal*	svLogicVecVal*
open array[]	const svOpenArrayHandle	svOpenArrayHandle
chandle	const void*	void*

5、 使用 C++模型的测试平台

```
import "DPI-C" function chandle counter7_new();
import "DPI-C" function void counter7_count(input chandle inst);
import "DPI-C" function void counter7_load(input chandle inst,
                                           input bit[6:0] i);
import "DPI-C" function void counter7_reset(input chandle inst);
import "DPI-C" function int counter7_get(input chandle inst);

// 用类封装计数器接口以隐藏 C++ 实例的句柄
class Counter7;
    chandle inst;

    function new;
        inst=counter7_new();
    endfunction

    function void count();
        counter7_count(inst);
    endfunction

    function void load(bit[6:0] val);
        counter7_load(inst, val);

    endfunction

    function void reset();
        counter7_reset(inst);
    endfunction

    function bit[6:0] get();
        return counter7_get(inst);
    endfunction
endclass : Counter7
```


使用 C++的测试平台

```
program automatic counter;
    Counter7 c1;

    initial begin
        c1=new;

        c1.reset();
        $display("SV: Post reset: counter1=%0d", c1.get());

        c1.load(126);
        if (c1.get()!=126) $display("Error in load");

        c1.count(); // count=127
        c1.count(); // count=0
        if (c1.get()!=0) $display("Error in rollover");
    end

endprogram
```


6、 导出一个 systemverilog 函数

```
module block;
    import "DPI-C" context function void c_display();
    export "DPI-C" function sv_display;      // 没有类型定义或者参数

    initial c_display();

    function void sv_display();
        $display("SV: block");
    endfunction
endmodule : block
```