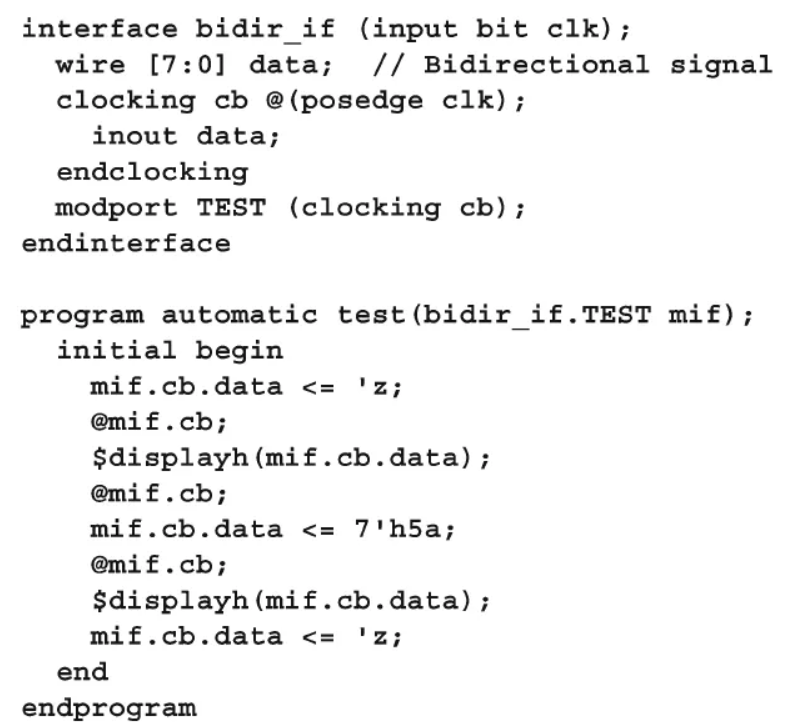
**[ SV 046 ]** 如何用interface连接双向端口？

A:

双向端口是一种多驱动，SV041提到多驱动必须要用wire。那么，可以像下面这么写：



当然还可以像RTL的双向端口连接那样，把inout拆成in、out、en三个信号。

**[ SV 047 ]** interface中的clocking语句有什么作用？

A:

同步。定义了数据与时钟的关系。

**[ SV 048 ]** interface里除了上面提到的信号定义、信号分组、信号同步，还可以写些什么？

A:

还可以写task、function、assertion等。

**[ SV 049 ]** 下面的代码中ref是什么含义？

1 module incr(ref int c, d);

2   always@(c)

3     #1 d = c++;

4 endmodule

A:

module里的ref与function/task里的ref类似，表明是引用，数据只存储一份。在module端口里ref只用于变量，不能用于信号。与inout的区别是，inout需要计算多个驱动的结果。而ref相当于覆盖，不会出现X或者Z。绿皮书P126。

**[ SV 050 ]** 下面的interface的用法是否可综合？

1 module mem(interface a, input logic clk);

2   ...

3 endmodule

A:

上面的代码有一个错误，端口申明时需要写interface的具体名字，而不是interface关键字。但sv里interface的写法是以可综合的。

**[ SV 051 ]** OOP英文全称是什么？

A:

Objective Oriented Programming，面向对象编程。

**[ SV 052 ]** class一般在哪里定义？

A:

class可以定义在module或者package内，也可以定义在module或者package外，即root空间。

**[ SV 053 ]** package是做什么的？

A:

package，包，是一个独立的编译单位（作用域），通常用来存放同一类的class。比如test case packge、agent package，env package等。类似C++中的命名空间、Python的module。

**[ SV 054 ]** 名词解释：类、对象、句柄、属性、方法？

A:

类：一类物体的抽象，是这类物体的描述。

对象：是类的实例化。

句柄：一个变量，指向对象。类似C++里的对象的指针。

属性：类的变量。

方法：类的函数。

**[ SV 055 ]** 解释下面代码6~7行的执行过程：

1 class SPI\_Driver;

2   ...

3 endclass

4 initial

5 begin

6   SPI\_Driver handle;

7   handle = new();

8   ...

9 end

A: SPI\_Driver handle是对象的声明，声明类型为SPI\_Driver，句柄为handle。但此时还没有实例化，handle为null。第7行的new()会实例化，并把handle指向这个对象。

**[ SV 056 ]** new()和new[]有什么区别？

A:

new()是实例化类的对象，而new[]是分配一个动态数组的内存。

**[ SV 057 ]** 例化一个对象的数组怎么写？对一幅10行x20列的图像建模，请补全：

1 class Pixel;

2   byte r,g,b;

3 endclass

4 initial

5 begin

6  //TODO: declare handle

7

8   //TODO: allocate object

9

10 end

 A:

Pixel image[];

image = new[10\*20];

for(int row=0; row<10; row++)

    for(int col=0; col<20; col++)begin

        image[row\*10 + col] = new();

        image[row\*10 + col].randomize();

    end

**[ SV 058 ]** 解释下面的代码7~9行：

1 class Pixel;

2   byte r, g, b;

3 endclass

4 initial

5 begin

6   Pixel p;

7   p = new();

8   p = new();

9   p = null;

10 end

 A:

第7行，实例化一个对象，p指向这个对象

第8行，实例化另一个对象，p指向这个新对象。同时由于没有句柄指向第一个对象，第一个对象被立即释放。

第9行，把句柄指向null，释放第二个对象。

**[ SV 059 ]** 把方法定义在类的外面有什么好处？

A:

把方法定义在类的外面，类里面只保留方法的声明。这样可以使类看起来**更简洁**，有利于大型类的书写和理解。

一般把class的声明放在svh的头文件里，在sv里写class方法的实现。

**[ SV 060 ]** SystemVerilog的类的属性是公有的还是私有的？

A:

sv的类为了更利于复用，而类操作的安全性需求并不是特别突出，所以sv里默认是**公有属性**。这一点与C++相反。类似的sv也不存在友元函数。

**[ SV 061 ]** 类中的static属性有什么作用，怎样读写static属性？

A:

static属性是类的所有对象共享的，类似一个全局的变量。

除了可以用“对象.静态属性”的方式来读写，还可以用“类::静态属性”的形式。

**[ SV 062 ]** 类中的static方法有什么作用，static方法里能否访问非static属性？

A:

static方法可以用来维护static属性，静态意味着调用可以早于class对象的初始化。在uvm里的type\_id::create()就是一个静态方法。static方法里只能操作static属性，不能操作非static属性，因为非static属性必须要在class对象创建并初始化之后才能用。

**[ SV 063 ]** 类中的this是指什么？

A:

this指对象本身。“this.属性”指class级别的属性，例如，this.name = name; 把局部的name赋值给class的成员变量name。

**[ SV 064 ]** 如何避免编译顺序引起的类未定义的问题？

A:

用typedef class xxx; 避免未定义，

而避免重定义可以用

`ifndef XXX

`define XXX

...

`endif

**[ SV 065 ]** 下面的代码是否存在bug？

1 task generator(int n);

2   Trans t;

3   t = new();

4   repeat(n)

5   begin

6     t.addr = $random();

7     transmit(t);

8   end

9 endtask

 A:

临时对象t发送后，紧接着又产生下一个t，极有可能会覆盖上一个对象。如果需要的话，可以用拷贝或克隆。

**[ SV 066 ]** 什么是浅拷贝和深拷贝？

A:

浅拷贝只会复制子对象的句柄（地址），并不会复制子对象的数据。而深拷贝是完整的复制。

**[ SV 067 ]** 解释composition和inheritance的区别及各自的应用场景？

A:

composition（组合）和inheritance（继承）是常见的两种复杂数据建模的方法，但两者有明显的不同。composition是包括、包含，比如，A包括了B和C的特征。而inheritance是指A继承了B或C的特征，也可以具有额外的新特征。由于不支持多继承，如果需要同时支持B和C的特征就需要用composition。具体用哪种方法需要需要结合自己数据的特点。详见绿皮书8.4节。

**[ SV 068 ]** 子类new()里的super.new()是不是必须的？

A:

不是必须的，会隐式的自动调用。

**[ SV 069 ]** 为什么把父类赋值给子类需要$cast？

A:

子类除了包括父类中所有的属性和方法，还增加了新的属性和方法把子类赋值给父类时，自动去除新增的属性即可。而父类赋值给子类时，**子类中新加的属性无法知晓，必须要特殊处理。**

**[ SV 070 ]** 子类里的方法能否在父类的方法基础上增减函数参数或修改返回值类型？

A:

不可以。绿皮书8.3.3。

**[ SV 071 ]**什么是抽象类？有什么作用？

A:

可以被扩展但不能被直接实例化

**[ SV 072 ]**什么是纯虚函数？纯虚函数是否**只能**在抽像类里定义？

A:

没有实体的方法原型

扩展类并对所有纯虚方法提供具体实现

**[ SV 073 ]**回调callback的主要适用场景用哪些？

A:

注入干扰、错误。

如注入错误、放弃transaction、延迟transaction、与其它transaction同步、把transaction放入scoreboard、收集功能覆盖率数据等。

**[ SV 074 ]参数类**的主要作用是什么？

A:

一些Class，如stack、generator，只能用于单一的数据类型（换一个数据类型就要重新写）。参数类就可以解决这个问题

**[ SV 076 ]**下面的代码有Bug吗？

1 class ClassA #(int p);

2 endclass

3

4 ClassA handleA;

A:

不带默认值的参数类实例化时，一定要指定类型或者初值。

如ClassA #(10) handleA;

**[ SV 077 ]**类可以嵌套吗？

A:

不行。

**[ SV 079 ]**“接口类只能包含纯虚函数、类型定义、参数定义”，这句话是否正确？

A:

正确

**[ SV 075 ]**用参数类实现一个通用fifo，存储的数据类型未知。

A:

parameter int SIZE = 100;

class Fifo #(type T=int);

    local T arr[SIZE];

    local int rd\_ptr = 0;

    local int wr\_ptr = 0;

    function push(input T i);

        if(is\_full())

            $display("FIFO is full");

        else

            arr[wr\_ptr++] = i;

    endfunction

    function T pop();

        if(is\_empty())

            $display("FIFO is empty");

        else

             return arr[rd\_ptr++];

    endfunction

    function bit is\_empty();

        if(wr\_ptr == rd\_ptr)return 1'b1;

        else return 1'b0;

    endfuntion

    function bit is\_full();

        if(wr\_ptr + 1'b1 == rd\_ptr)return 1'b1;

        else return 1'b0;

    endfuntion

endclass

**[ SV 078 ]**什么是接口类（interface class）？

A:

1. **不再依赖于UVM的uvm\_analysis\_port**，即在任何SV环境中都可以完成一对多的广播-收听（publish-listen）软件设计模式。
2. **不再要求所有的连接需要发生在UVM的connect阶段**，也不需要广播方或者收听方必须是uvm\_component，提高了实现的便利性，脱离了UVM的组织结构限制。
3. 对于广播的方法，**不再限定于其内容和参数个数**，因此当需要传递多个参数时，利用接口类将比TLM传送事务更加方便。

**广播-收听的设计模式可以利用接口类进一步提高灵活性。多继承的模式也能减轻代码的体积，简化类的继承层次**

**[ SV 080 ]**下面的代码有Bug吗？

1 typedef interface class IntfD;

2

3 class ClassB implements IntfD #(bit);

4 virtual function bit[1:0] funcD();

5 endclass

6

7 interface class IntfD #(type T1 = logic);

8 typedef T1[1:0] T2;

9 pure virtual function T2 funcD();

10 endclass

[ SV 081 ]在IC验证中，我们一般对哪些进行随机化？

[ SV 082 ]随机化的优势是什么？是不是意味着不再需要定向case了？

[ SV 083 ]rand和randc有什么区别？

[ SV 084 ]一个class中定义了若干个rand型变量，如果只需要随机化其中的一个？

[ SV 085 ]如何随机化一个动态数组？长度>=5且<=10，元素数值>10且<200。

 [ SV 086 ]解释下面的约束：

1 class ClassA;

2 rand bit[31:0] addr;

3 rand bit inject\_error;

4 constraint c\_addr{

5 inject\_error ->

6 addr[0]==1'b0;

7 }

8 …

1 class ClassA;

2 rand bit test\_mode;

3 rand bit inject\_error;

4 constraint c\_error{

5 inject\_error<->test\_mode;

6 }

7 …

[ SV 087 ]randomize with {...}中的约束与class中的约束是什么关系？

[ SV 088 ]如何随机化一个数组，使得元素数值不重复？

[ SV 089 ]在随机化一个序列时，如何根据上一次随机的结果来控制本次的随机？

[ SV 090 ]如何基于随机化的验证环境写定向case？

[ SV 091 ]最经典的问题：fork ... join/join\_any/join\_none的区别？

[ SV 092 ]利用fork写一段防止仿真超时的代码。

[ SV 093 ]两个并行进程之间如何交互数据？

[ SV 094 ]如何防止两个并行的进程对同一个文件同时写操作？

[ SV 095 ]如何等待另一个进程执行结束，和杀死另一个进程？

[ SV 096 ]为什么说旗语semaphore和信箱mailbox可以起到同步进程的作用？

**[ SV 097 ]**mailbox与fifo有什么区别？

A:

信箱例化 定容

[ SV 098 ]子进程中是否可以访问全局变量？

[ SV 099 ]多进程在验证环境中有哪些应用？

**[ SV 101 ]**与dpi相比，pli的缺点是什么？

**[ SV 102 ]**import "DPI-C" \expect = function int fexpect(); 中的斜杆'\'是做什么的？

A:

C函数expect映射到SV中fexpect

**[ SV 103 ]**dpi函数的方向支持ref吗？

A:

不支持

**[ SV 104 ]**如何处理sv的四值变量，比如logic的X态在c里被转成了什么？

A:

双状态变量使用svBit(实际存储空间是unsigned char)表示，双状态变量带下标使用svBitVecVal\*表示

四状态变量使用svLogic(实际存储空间是unsigned char)表示，四状态变量带下标使用svLogicVecVal\*表示

0---在C中对应0x0，1---在C中对应0x1，Z---在C中对应0x2，X---在C中对应0x3

**[ SV 105 ]**把下面的代码补充完整，reset时清零，load时加载i的值，其它情况加1：

1 #include<\_\_\_.h>

2 void counter7(svBitVecVal \* o,

3 svBitVecVal \* i,

4 const svBit reset,

5 const svBit load)

6 {

7 //TODO:

8 }

A:

sv通过DPI调用c，reset和load是一个双状态的比特信号，以svBit类型进行传递。输入i是双状态7bit变量，用svBitVecVal类型传递。

#include <svdpi.h>

void counter7(svBitVecVal \* o,

              svBitVecVal \* i,

              const svBit reset,

              const svBit load)

{static unsigned char count = 0;

     if(reset)   count = 0;

     else if(load) count = \* i;

     else count++;

     count &= 0x7F;

     \*o = count;

}

[ SV 106 ]io\_printf比c的printf有什么好处？

[ SV 107 ]下面的代码是做什么的？

1 #ifdef \_\_cplusplus

2 extern"C"{

3 #endif

4 …

5 #ifdef \_\_cplusplus

6 }

7 #endif

A:

用上这样的方式，可以让编译器把extern “C”代码块中的内容按照C语言的编译方式进行编译。为什么要这样呢？因为c++中支持函数重载，而C语言不支持

[ SV 108 ]什么是open array？

**[ SV 109 ]**sv如何调用python或perl脚本程序？

A:

$system(python," ")

**[ SV 110 ]**DPI、PLI、VPI之间是什么关系？

A:

Verilog与C之间进行程序交互，PLI(Programming Language Interface)方式给仿真带来了额外的负担，DPI(direct programming interface)能够更简洁的连接c。DPI只需import声明和使用，导入一个c子程序。