面向对象编程

概述

> *面向对象编程(OOP)使得用户可以创建更加复杂的数据类型，并将该数据类型与使用这些数据类型 的程序紧密结合在一起。用户可以在更加抽象的层次建立测试平台和系统模型。通过调用函数来是 实现电平的改变。使用事务来替代信号的翻转。使得测试平台与细节分开，增加系统的可维护性。*

考虑名词而非动词

测试平台的目的是给设计施加激励。然后检查其结果是否正确。 我们区分传统的测试平台构建的操作：创建一个事务，发送，接收，检查结果，产生报告。

但是在OOP中，我们需要重新考虑测试平台的构建以及每一个部分的功能。

- 发生器(generator)：创建事务并将他们传送给下一级
- 驱动器(driver)：与设计进行会话
- 监视器(monitor)：设计返回的事务将被监视器捕获
- 计分板(scoreboard)：将捕获的数据与预期数据进行比对。 因此测试平台被分为多个块(block)，然后定义其互相之间的通信。

编写第一个类

我们前面学习的是定义一个更加复杂的数据类型，类封装了数据，操作这些数的子程序.实例如下：

```
class Transaction;
    bit [31: 0] addr,crc, data[8];
    function void display;
        $display("Transaction: 3h", addr);
    endfunction : display
    function void calc_crc
        crc=addr^data.xor;
    endfunction : calc_crc
endclass: Transaction
```

补充：为了更加方便的取分对齐一个块的开始与结束部分，可以在块结束的最后放一个标记(label)。 在很多时候，这些标记可以方便的区分结束。例如：endtask、endfunction、endcalss

OOP术语

1）类(class):包含变量和子程序的基本构建块。
2）对象(object):类的一个实例。
3）句柄(handle):指向对象的指针。
4）属性(property):存储数据的变量
5）方法(method):任务或者函数中操作变量的程序性代码。
6）原型(prototype):程序的头，包括程序名，返回类型和参数列表。程序体则包含了执行代码。

创建新的对象

System Verilog的类，是在程序执行过程中需要的时候在被创建。所有的类在使用之前都应该被例化。 例化后就会开辟一定的内存空间。

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

在声明句柄的时候，他被初始化为了特殊格式`null`。接下来调用`new()`来创`Transaction`对象的时候，`nwe`函数为其分配空间，并变量初始化为默认值。其中二值变量初始化为0，四值变量 初始化为X。我们称作new为**构造函数**。没有返回值。 使用new对类进行例化时，可以对变量指定初始值。方式如下：

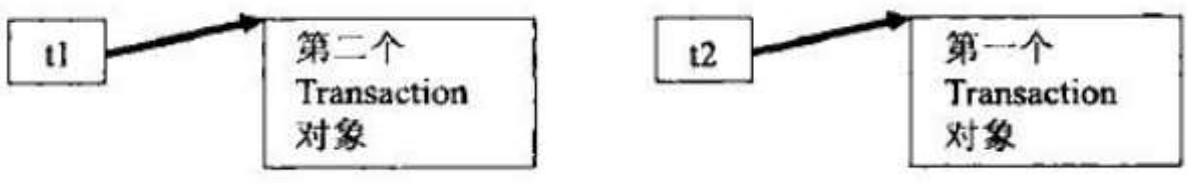
```
class Transaction;
    logic [31: 0] addr,crc, data[8];
    function new(logic [31:0] a=3,d=5);
        addr = 1;
            foreach(data[i])
                data[i]=d;
    endfunction : new
endclass: Transaction
initial begin
    Transaction tr;
    tr=new(10)//data默认为5，
```

为对象创建一个句柄

首先要对于对象与句柄进行区分。一个句柄可以指向很多个对象。

```
Transaction t1,t2;//声明两个句柄
t1=new();//为第一个Transaction对象分配地址
t2=t1;//两个句柄指向一个对象，可以理解两个指针指向同一个内存空间
t1=new();//为第二个Transaction对象分配地址，因为这个时候t2还是指向的第一个对象，所以t1指向一个新的对象
```

具体映射关系见下图



通过以上例子对与句柄与对象进行区分。对象可以理解为向系统申请的一个内存空间，申请的方式 就是通过句柄来保证的。这里句柄相当于一个对象的寻找、使用方式。一旦一个对象没有句柄指向 ， 则这个对象的空间就会被释放掉。

对象的解除分配

实例如下。当对象不在被引用，即为没有指向他的句柄。

```
Transaction t;//声明一个句柄
t=new();//为第一个Transaction对象分配地址
t=new();//这是对第一个空间进行回收，同时开辟一个新的地址空间
t=null;//解除第二个分配，释放空间
```

使用对象

假设已经分配了一个对象，接下来就是如何使用对象。可以使用`.`来引用子变量和子程序。

```
Transaction t;//声明一个句柄
t=new();//为第一个Transaction对象分配地址，创建一个对象
t.addr=32'h32;//设置变量的值
t.display();//调用一个子程序
```

静态变量与动态变量

每个对象都有自己的局部变量，这些变量不与其他任何对象共享。但若是需要存在一个某种类型的 变量需要被所有变量共享，就需要创建一个全局变量。

简单的静态变量

在System Verilog中，可以在类中创建一个静态变量，该变量将被这个类所有的实例共享。但是它 使用的范围仅仅限制与则个类。如下实例中**静态变量**`count`用来保护目前所创建的对象的数目。 他在声明的时候被初始化为0。之后每构建一个新的对象，他就被标记为一个唯一的值。同时`count` 的值加1。

```
class Transaction;
    static int count=0;//已创建的对象的数目
    int id://实例的唯一标志
    function new()
        id= count++;//设置标志，count递增
        endfunction
endclass : Transaction
Transaction t1,t2;
initial begin
    t1=new();//第一个实例,id=0,count=1
    t2=new();//第二个实例id=1, count=2
end
```

一个类应该事尽可能自己自足的，当打算创建一个全局变量的时候。首先需要考虑创建一个类的 静态变量能否解决问题，对于一个类，应用的外部变量越少越好。

通过类名访问静态变量

这里引入一个新的访问静态变量的方法。该方法不需要使用句柄，而是使用`class_name ::`即 类作用域操作符。

```
class Transaction;
    static int count=0;//创建的对象数
    ...
endclass
Transaction t;
initial begin
    Transaction t;
    run test();
    $display("%d transaction were created",t.count);//使用句柄引用静态句柄
```

```
    $display("%d transaction were created",Transaction:: count);//引用静态句柄
end
```

静态变量的初始化?

通常是在声明的时候初始化。静态变量通常在声明时初始化。你不能简单地在类的构造函数中 初始化静态变量因为每一个新的对象都会调用构造函数。你可能需要另一个静态变量来作为标 志,以标识原始变量是否已被初始化。如果需要做一个更加详细的初始化,你可以使用初始化块但 是要保证在创建第一个对象前,就已经初始化了静态变量。（待补充）

静态方法?

（待补充）

类的方法

类中的程序也称为方法。也就是在类的作用域定义的task或者function。

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

class PCI_Tran;
    bit [31:0]adx,data;//使用真实的名字
    function void display();
    $display("@%0t: PCT:addr=%h,data=%h", $time,addr,data);
    endfunction
endclass
Transaction t;
PCI Tran_pc
initial begint
    new();//创建一个 Transaction对象
    t.display();//调用 Transaction的方法
    c=new();//创建一个PCI事务
    pc.display();//调用PCI事务的方法
end
```

在类之外定义方法

SV允许在一个块外声明例子。使用关键词`extern`。实例如下

```
class Transaction;
    bit [31: 0] addr, crc, data[8];
    extern function void display ();
endclass
function void Transaction::display ();
    $display("@%0t: TR addr:=%h,crc=%h", stime,addr,crc);
    $write ("\tdata [0-7]=");
    foreach (data[i]) $write(data[i]);
    $display ();
endfunction
class PCI_Tran;
    bit [31:0]adx,data;//使用真实的名字
    extern function void display();
endclass
function void PCI_Tran::display();
```

```
    $display("@%0t: PCT:addr=%h,data=%h", $time,addr,data);
endfunction
```

作用域规则

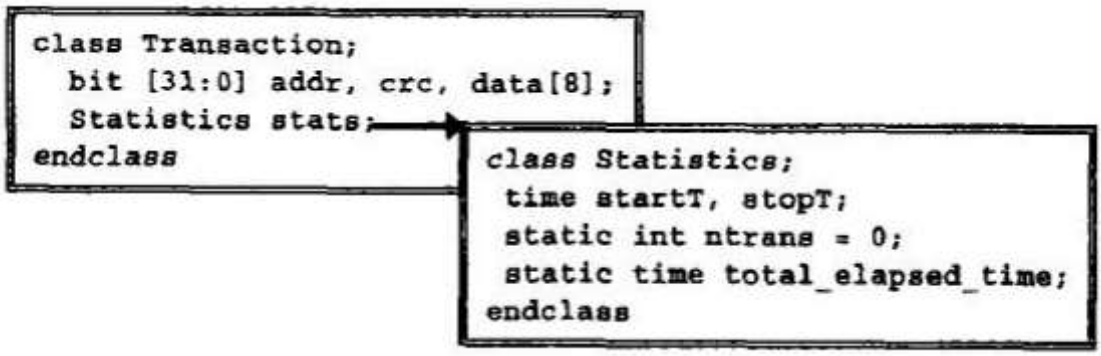
作用域是一个代码块。例如一个模块，一个程序、任务、函数、类 或者begin-end块。每个作用域都使用其作用域下定义的变量。

this

当你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。这也是 Verilog 所采用的算法。但是如果你在类的很深的底层作用域,却想明确地引用类 一级的对象呢?这种风格的代码在构造函数里最常见,因为这时候程序员使 用相同的类变量名和参数名。关键词`this`可以实现该功能。

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

在类中可以使用另外一个类




```
class Transact⊥on;
    bit [31: 0] addr, crc, data[8];
    Statistics stats;// Statistics句柄
    function new();
        stats=new();//创建 stats实例
    endfunction
    task create packet();//填充包数据
        stats.start()//传送数据包
    endtask
endclass
```

一定要记得例化对象,否则句柄stats是nu11, 调用start会失败。这最好在上层即 Transaction类 的构造函数中完成。

这其中涉及到了编译的顺序问题。有时候你需要编译一个类。而这个类 包含一个尚未定义的类。这个被包含的类的句柄就会引起错误。这时候 需要使用`typedef`来声明一个类名。

```
typedef class Statistics;
class Transaction;
    Statistics stats;
    ...
endclass
class Statistics;
```

```
    ...
endclass
```