

Replace section 7.19 Abstract classes and virtual methods with the following two sections

### 7.19 Virtual methods

~~Abstract classes can also have virtual methods~~A method of a class may be identified with the keyword **virtual**. Virtual methods are a basic polymorphic construct. A virtual method shall overrides a method in all ~~the of its~~ base (parent) classes, whereas a ~~normal~~ non-virtual method shall only overrides a method in that class and its descendants. One way to view this is that there is only one implementation of a virtual method per class hierarchy, and it is always the one in the latest derived class.

Virtual methods provide prototypes for ~~subroutines~~the methods that later override them, i.e., all of the information generally found on the first line of a method declaration: the encapsulation criteria, the type and number of arguments, and the return type if it is needed. Later, when subclasses override virtual methods, they shall must follow the prototype exactly by having matching return types and matching argument names, types, and directions. It is not necessary to have matching default expressions, but the presence of a default much match. Thus, all versions of the virtual method look identical in all subclasses.:

```
class BasePacket;  
  int A = 1;  
  int B = 2;  
  function void printA;  
    $display("A is ",A);  
  endfunction : printA  
  virtual function void printB;  
    $display("B is ",B);  
  endfunction : printB  
endclass : BasePacket  
class My_Packet extends Packet;  
  int A = 3;  
  int B = 4;  
  function void printA;  
    $display("A is ",A);  
  endfunction : printA  
  virtual function void printB;  
    $display("B is ",B);  
  endfunction : printB  
endclass : BasePacket
```

```
BasePacket P1 = new;  
My_Packet P2 = new;  
initial begin  
  P1.printA; // displays 'A is 1'  
  P1.printB; // displaya 'B is 2'  
  P1 = P2; // P1 has a handle to a My_packet object  
  P1.printA; // displays 'A is 1'  
  P1.printB; // displaya 'B is 4' - latest derived method  
  P2.printA; // displays 'A is 3'  
  P2.printB; // displaya 'B is 4'
```

end

Once a method has been identified as virtual, it shall remain virtual in any subclass that overrides it. The **virtual** keyword may be used in later declarations, but is not required.

### **7.19-20 Abstract classes and prototype virtual methods**

A set of classes ~~can may~~ be created that can be viewed as all being derived from a common base class. For example, a common base class of type `BasePacket` that sets out the structure of packets but is incomplete would never be ~~instantiated~~constructed. This is characterized as an abstract class. From this abstract base class, however, a number of useful subclasses ~~could may~~ be derived, such as Ethernet packets, token ring packets, GPSS packets, and satellite packets. Each of these packets might look very similar, all needing the same set of methods, but they could vary significantly in terms of their internal details.

A base class ~~sets out the prototype for the subclasses. Because the base class is not intended to be instantiated, it can be made abstract by specifying the class to be~~may be characterized as being abstract by identifying it with the keyword **virtual**:

```
virtual class BasePacket;  
...  
endclass
```

An object of an abstract class shall not be constructed directly. It may be indirectly constructed through the chaining of constructors in an extended non-abstract subclass object.

A virtual method in an abstract class may be declared as a prototype without providing an implementation. This may be indicated with the keyword **extern** together with not providing a method body. An extended subclass may provide an implementation by overriding the virtual method.

Abstract classes may be extended to additional abstract classes, but all prototype virtual methods shall have implementations in order to be extended into a non-abstract class. By having implementations for all its methods, the class is complete and may now be constructed. Any class may be extended into an abstract class, and may provide additional or overridden virtual methods without implementations.

```
virtual class BasePacket;  
  extern virtual function integer send(bit[31:0] data); // No  
  implementation  
  endfunction  
endclass  
class EtherPacket extends BasePacket;  
  virtual function integer send(bit[31:0] data);  
  // body of the function  
  ...  
  endfunction  
endclass
```

`EtherPacket` is now a class that can have an object of its type constructedbe instantiated. In general, if an abstract class has any virtual methods, all of the methods must be overridden (and provided with a method body) for the subclass to be instantiated. If any virtual methods have no implementation, the subclass needs to be abstract.

Note – A method without a statement body is still a legal, callable method. For example, if the function `send` was declared as show below, it would have an implementation:

**virtual function integer** `send(bit[31:0] data); // Will return 'x`  
**endfunction**

An abstract class can contain methods for which there is only a prototype and no implementation (i.e., an incomplete class). An abstract class cannot be instantiated; it can only be derived. Methods of normal classes can also be declared virtual. In this case, the method must have a body. If the method does have a body, then the class can be instantiated, as can its subclasses.