## Change Section 19.8 BNF – Changes in blue

```
virtual_interface_declaration ::=                                   // from Annex A.2.9
     virtual [interface] interface_identifier  list_of_virtual_interface_decl ;

list_of_virtual_interface_decl ::=                                  // from Annex A.2.3
                    variable_identifier [ = interface_instance_identifier ]
                  { , variable_identifier [ = interface_instance_identifier ] }

data_declaration ::=                                                // from Annex A.2.1.3
                [ lifetime ] variable_declaration
              | constant_declaration
              | type_declaration
              | virtual_interface_declaration

data_type ::=                                                       // from Annex A.2.2.1
                  …
                  | virtual [interface] interface_identifier
```

*Syntax 19-2—Virtual interface declaration.*

## Change Section 19.8.2 BNF – Changes in blue

### 19.8.2 Virtual interfaces modports and clocking domains

~~The **modport** construct provides direction information for module ports as well as control the use of tasks and functions within particular modules. When using a **modport**, the directions are those seen from the module in which the interface becomes a port.~~

As shown in the example above, once a virtual interface is declared, its clocking-domain can be referenced using dot-notation. However, this only works for interfaces with no modports. Typically, a device under test and its testbench exhibit **modport** direction. This common case can be handled by including the **clocking** in the corresponding **modport**, as described in Section 19.4.5. ~~The syntax for this is shown below.~~

```
~~modport_declaration ::= modport modport_item { , modport_item } ;                    // from Annex A.2.9~~

~~modport_item ::= modport_identifier ( modport_ports_declaration { , modport_ports_declaration } )~~

~~modport_ports_declaration ::=~~
            ~~modport_simple_ports_declaration~~
            ~~| modport_hierarchical_ports_declaration~~
            ~~| modport_tf_ports_declaration~~
            ~~| modport_clocking_declaration                                  Note: this is new~~

~~modport_clocking_declaration ::= clocking clocking_identifier               Note: this is new~~
```

~~*Syntax 19-3—Interface modport declaration.*~~

~~All of the **clocking** constructs used in a **modport** declaration shall be declared by the same interface as is the **modport** itself. Like all **modport** declarations, the direction of the clocking signals are those seen from the module in which the interface becomes a port. The example below shows how **modports** can be used to create~~

~~both synchronous as well as asynchronous ports. When used in conjunction with virtual interfaces, these constructs facilitate the creation of abstract synchronous models.~~

The example below shows how **modports** used in conjunction with virtual interfaces facilitate the creation of abstract synchronous models.

```
interface A_Bus( input bit clk );
    wire req, gnt;
    wire [7:0] addr, data;

    clocking sb @(posedge clk);
        input gnt;
        output req, addr;
        inout data;

        property p1; req ##[1:3] gnt; endproperty
    endclocking

    modport DUT ( input clk, req, addr,     // Device under test modport
                  output gnt,
                  inout data );

    modport STB ( clocking sb );            // synchronous testbench modport

    modport TB ( input gnt,                 // asynchronous testbench modport
                 output req, addr,
                 inout data );
endinterface
```

The above interface A_Bus can then be instantiated as shown below:

```
module dev1(A_Bus.DUT b);               // Some device: Part of the design
    ...
endmodule

module dev2(A_Bus.DUT b);               // Some device: Part of the design
    ...
endmodule

program T (A_Bus.STB b1, A_Bus.STB b2 );    // Testbench: 2 synchronous ports
    ...
endprogram

module top;
    bit clk;

    A_Bus b1( clk );
    A_Bus b2( clk );

    dev1 d1( b1 );
    dev2 d2( b2 );

    T tb( b1, b2 );
endmodule
```

And, within the testbench program, the virtual interface can refer directly to the clocking domain.

```
program T (A_Bus.STB b1, A_Bus.STB b2 );    // Testbench: 2 synchronous ports

    typedef virtual A_Bus.STB SYNCTB;

    task request( SYNCTB s );
```

```
        s.sb.req <= 1;
    endtask

    task wait_grant( SYNCTB s );
        wait( s.sb.gnt == 1 );
    endtask

    task drive(SYNCTB s, logic [7:0] adr, data );
        if( s.sb.gnt == 0 ) begin
            request(s);                 // acquire bus if needed
            wait_grant(s);
        end
        s.sb.addr = adr;
        s.sb.data = data;
        repeat(2) @s.sb;
        s.sb.req = 0;                   //release bus
    endtask

    assert property (b1.p1);            // assert property from within program

    initial begin
        drive( b1, $random, $random );
        drive( b2, $random, $random );
    end
endprogram
```

The example above shows how the clocking-domain is referenced via the virtual interface by the tasks within the program block.

## Modify Syntax 19-1 in Section 19.2 and A.2.9 as shown (add text not in purple)

```
modport_ports_declaration ::=
          modport_simple_ports_declaration
        | modport_hierarchical_ports_declaration
        | modport_tf_ports_declaration
        | modport_clocking_declaration

modport_clocking_declaration ::= clocking clocking_identifier
```

## Modify A.2.1.3 as shown (add text not in purple)

```
data_declaration ::=                                    // from Annex A.2.1.3
          [ lifetime ] variable_declaration
        | constant_declaration
        | type_declaration
        | virtual_interface_declaration
```

## Insert as Section 19.4.5

### 19.4.5 Clocking and modports

The **modport** construct can also be used to specify the direction of **clocking** blocks declared within an interface. As with other **modport** declarations, the directions of the **clocking** block are those seen from the module in which the interface becomes a port. The syntax for this is shown below.

```
modport_declaration ::= modport modport_item { , modport_item } ;          // from Annex A.2.9

modport_item ::= modport_identifier ( modport_ports_declaration { , modport_ports_declaration } )

modport_ports_declaration ::=
        modport_simple_ports_declaration
      | modport_hierarchical_ports_declaration
      | modport_tf_ports_declaration
      | modport_clocking_declaration

modport_clocking_declaration ::= clocking clocking_identifier
```

*Syntax 19-3—Interface modport declaration.*

All of the **clocking** constructs used in a **modport** declaration shall be declared by the same interface as is the **modport** itself. Like all **modport** declarations, the direction of the clocking signals are those seen from the module in which the interface becomes a port. The example below shows how **modports** can be used to create both synchronous as well as asynchronous ports. When used in conjunction with virtual interfaces (see Section 19.8.2), these constructs facilitate the creation of abstract synchronous models.

```
interface A_Bus( input bit clk );
    wire req, gnt;
    wire [7:0] addr, data;

    clocking sb @(posedge clk);
        input gnt;
        output req, addr;
        inout data;

        property p1; req ##[1:3] gnt; endproperty
    endclocking

    modport DUT ( input clk, req, addr,     // Device under test modport
                  output gnt,
                  inout data );

    modport STB ( clocking sb );            // synchronous testbench modport

    modport TB ( input gnt,                 // asynchronous testbench modport
                 output req, addr,
                 inout data );
endinterface
```

The above interface A_Bus can then be instantiated as shown below:

```
module dev1(A_Bus.DUT b);               // Some device: Part of the design
    ...
endmodule

module dev2(A_Bus.DUT b);               // Some device: Part of the design
    ...
endmodule

program T (A_Bus.STB b1, A_Bus.STB b2 );    // Testbench: 2 synchronous ports
    ...
```

```
endprogram

module top;
    bit clk;

    A_Bus b1( clk );
    A_Bus b2( clk );

    dev1 d1( b1 );
    dev2 d2( b2 );

    T tb( b1, b2 );
endmodule

program T (A_Bus.STB b1, A_Bus.STB b2 );    // Testbench: 2 synchronous ports

    assert property (b1.p1);            // assert property from within program

    initial begin
        b1.sb.req <= 1;
        wait( b1.sb.gnt == 1 );
        ...
        b1.sb.req <= 0;
        b2.sb.req <= 1;
        wait( b2.sb.gnt == 1 );
        ...
        b2.sb.req <= 0;
    end
endprogram
```

The example above shows the program block using the synchronous interface designated by the clocking-modport of interface ports b1 and b2. In addition to the procedural drives and samples of the clocking block signals, the program asserts the property p1 of one of its interfaces b1.