SystemVerilog Feature Proposals

Geoff Barnes

26 FEB 2010

Two Feature Proposals

- #1 : Proposal for ‘Inheritance’ for **interfaces** and **modules**

- #2 : Proposal to support module instantiation within interfaces

Inheritance for Interfaces/Modules

- Interfaces and Modules are the primary reuse construct in Verilog.
- Do not support a way to create “specializations” or “variations” (expect via Parameters)
- Can we not “extend” them by overriding some parts, keeping some parts, and tweaking other parts?
- Named and labeled structures are “Non-anonymous” and should be easy (I guess) to override in “child” variations

- **Why? To support “templating” and increase reuse**

Module Example

- Might have similar benefit as a ‘config’, but can be instantiated

```
module mydesign ( . . .);

  XilinxRam myram (.*);

  myfsm fsm (.*);

endmodule


// replace instances
module mydesign_altera extends mydesign ;

  AlteraRam myram (.*);

  // myfsm implicitly included

endmodule
```

Module Example

```
module dummy #(foo_val = 8)(output [7:0] foo, bar);

  assign foo = foo_val;

  drive_bar : assign bar = 'hff;

endmodule

// reuse entire module except tweak param default
module dummy_foo_5 extends dummy #(foo_val = 8'd5);
endmodule


// replace an driver
module dummy_variation extends dummy_foo_6_bar_0 ;

  drive_bar : assign bar = 0;

endmodule
```

Interface Example

- Similar approach as modules

```
interface count ( input clk,reset);
   int q;
   function int reset_value; reset_value = 0; endfunction
   function int count (input int i);
       count = i + 1;
   endfunction

   always_ff @(posedge clk or negedge reset) begin : main1
      if (~reset) q <= reset_value(); else q <= count(q);
   end
endinterface

// 'Child' interface
interface count_down extends count;

   function int count (input int i);
       count = i – 1;
   endfunction

endinterface
```

Inheritance

- Many possibilities/scenarios to consider
- Maybe easiest to think of it much like using '**config'**urations. Except:
  - Can be instantiated
  - No need for a separate construct
  - Anything that has a **name** can be "configured", not just module instances.

- Interface inheritance should be exactly the same.

Two Feature Proposals

- #1 : Proposal for ‘Inheritance’ for **interfaces** and **modules**

- #2 : Proposal to support module instantiation within interfaces

Instantiating modules within interfaces

- **Why? Why not?**
- **Allows module based IP to be reused within interfaces..**