Interface and Modport Enhancements

Peter Flake

26 Feb 2010

Agenda

- Purpose
- Problems
- Modport names example
- Proposal 1
- Proposal 2
- Alternate proposal
- Summary

Purpose

- Interfaces encapsulate interconnect
- Modports encapsulate port lists
- Encapsulation reduces lines of code
- Encapsulation reduces bugs

BUT

- They are not widely used
- Use is difficult with different source code bases
- Missing flexibility

Problems with Interfaces and Modports

Using interfaces and modports with IP is difficult because:

- Rigid name matching
- No distinction between modport type and modport instance in interface
- Parameterization of modport is in the interface, not the module
- Interfaces require hierarchical names to access wires in modules
  - Needs a lot of source editing

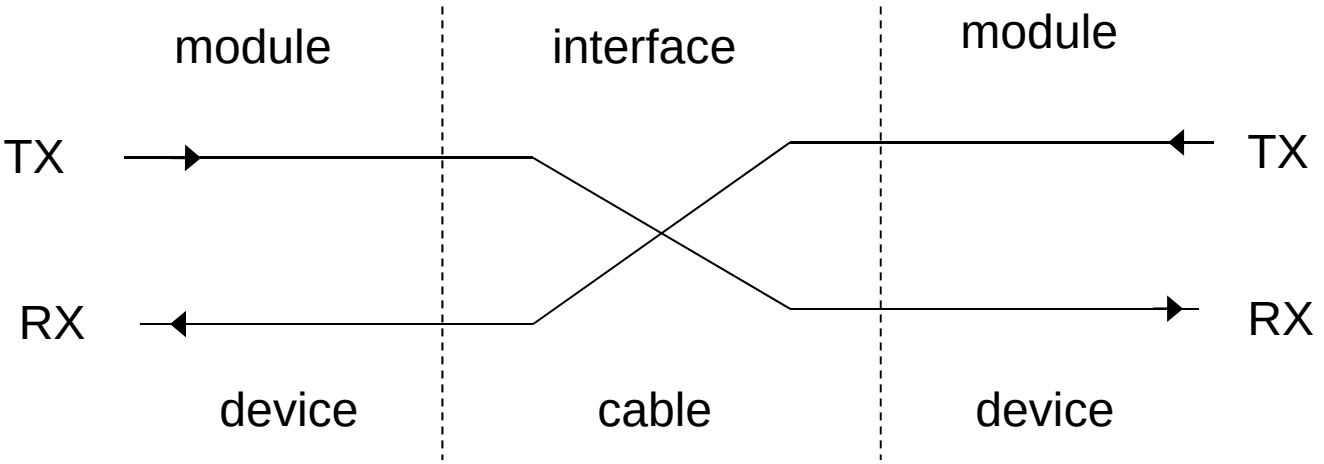
Port Declaration Modports

- Natural way to code port directions
- See example 1800-2009 Section 25.5.1

BUT

- Interface type must be specified with modport name
- A new interface version with a new type name invalidates the modport
  - Have to change more than top level module
- Modports of same (type) name connect to same interface wires
- No way to pass parameters to modport e.g. bit width

Modport Names Example



Different Modport Names


```
interface cable;
    wire L2R, R2L;
    modport left(output .TX(L2R), input .RX(R2L));
    modport right(output .TX(R2L), input .RX(L2R));
endinterface

module device(interface i); //cannot use modport here
    assign i.TX = i.RX; // needs hierarchical names
endmodule

module system;
    cable c;
    device L(c.left);
    device R(c.right);
endmodule
```

Same Modport Name?


```
interface cable;
    wire[1:0] wires;
    for (genvar j=0; j<2; j++) begin: ends
        modport plug(output .TX(wires[j], input .RX(wires[1-j]));
    end
endinterface

module device(interface i); // want cable.plug
    assign i.TX = i.RX;
endmodule

module system;
    cable c;
    device LR[0:1] (c.ends.plug); // is this legal?
endmodule
```

Proposal 1 – No New Syntax

- Allow modport as module port declaration to be in a different interface from port connection.
- Allow port connection modports to be in sub-interfaces.
  - Provides a means of instantiating modports in an interface
- Check port names, types (and directions if connection by modport) at elaboration time.
- Possibly allow connection to legacy Verilog port declarations or port connections

Example: Modports in Sub-interfaces


```
interface plug(output TX, input RX);
    modport p(output TX, input RX);
endinterface

interface cable;
    wire L2R, R2L;
    plug left(.TX(L2R), .RX(R2L)); // instance
    plug right(.TX(R2L), .RX(L2R)); // instance
endinterface

module device(plug.p i); // now specifies directions of RX and TX
    assign i.TX = i.RX; // needs hierarchical names
endmodule

module system;
    cable c;
    device L(c.left.p);
    device R(c.right.p);
endmodule
```

Proposal 2 – Minimal New Syntax

- Give a modport a local name in the module header, like a generic interface.
- Instantiate the modport in the module body
  - Use interfacename.modportname
  - Pass module parameters to interface parameters
  - Make port connections to rest of module
- Allow modport to be configured by parameters
  - Can use a generate block and conditionals
  - Can configure list of ports

Instantiating Modports


```
interface plug(output TX, input RX);
    modport p(output TX, input RX);
endinterface

module foo #(myparam) (modport a,b);
    wire TX, RX;
    plug.p a(.*); // equivalent to .TX(TX), .RX(RX)
    assign TX = RX; // no hierarchical names

    // pass parameter down to other modport
    myinterface.mymodport # (myparam) b (.*);
endmodule
```

Alternate Proposal

- Mantis 1861
- Paper “Is There a Future for SystemVerilog Interfaces?” Jonathan Bromley, Gordon Vreugdenhil, DVCon 2009

Summary: First Proposal

- Allow a port declaration modport to be in a different interface from its port connection
- Allow a port connection to use a modport in a sub-interface
- Check port names and types (and directions)
- Fixes problem of interface and modport name maintenance
- No new syntax
  - Could be regarded as a “clarification”

Summary: Second proposal

- Explicitly instantiate module modport
- Use placeholder in module header
- Needs minimal new syntax
- Avoids hierarchical wire names in module
- Allows parameterization of modports in module headers.
- Allows modport to be configured by parameters
  - Can use a generate block and conditionals
  - Can configure list of ports

Conclusion

- Use of interfaces and modports is hampered by language defects
- This is preventing the benefits of concise design descriptions being realized
- There are proposals to fix the defects
- This should be on the agenda for the next revision of the standard

Questions?

flake@elda.demon.co.uk