Seamless Refinement from Transaction Level to RTL Using SystemVerilog Interfaces

Jonathan Bromley

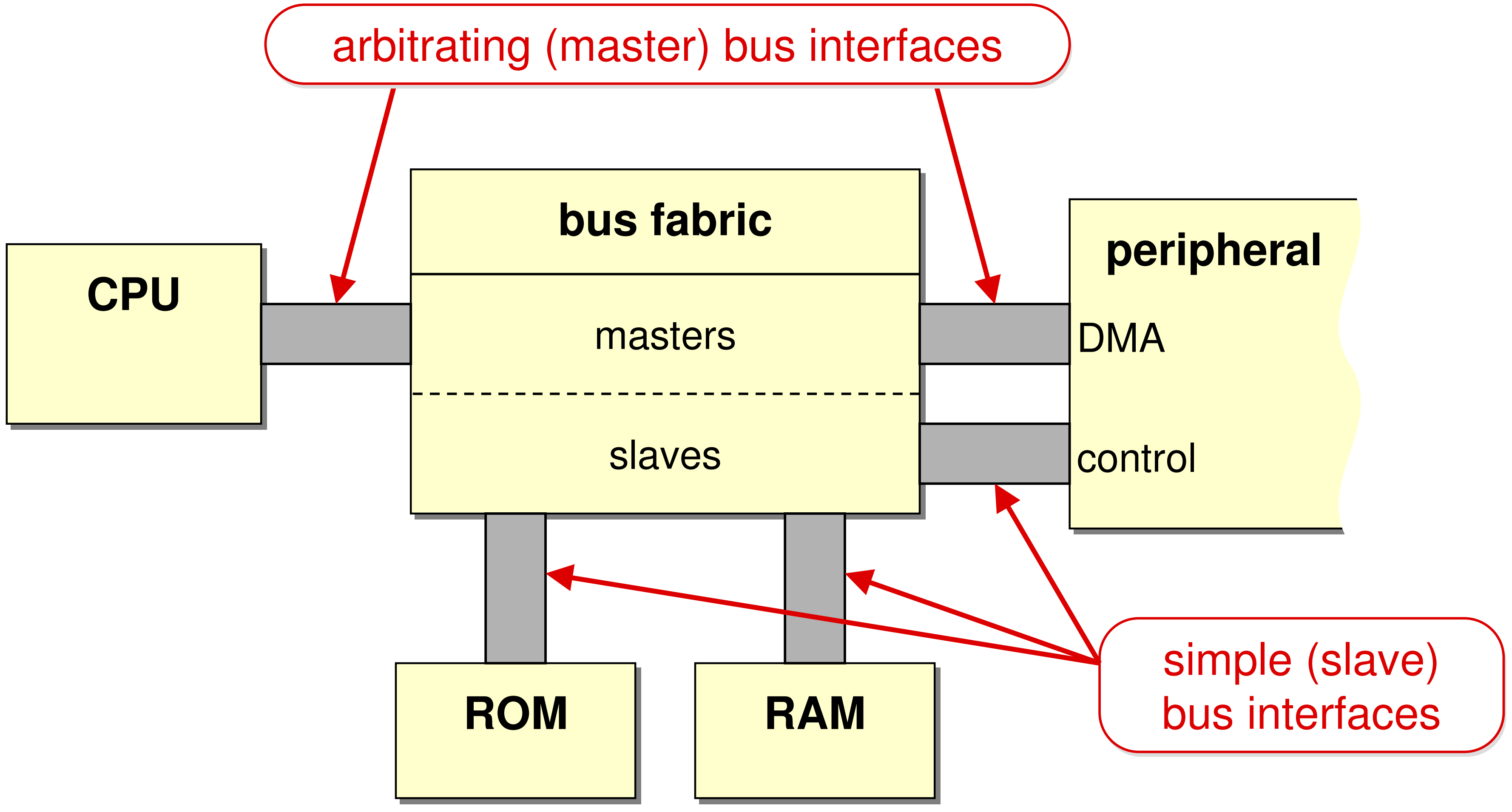
Doulos Ltd, Ringwood, UK

jonathan.bromley@doulos.com

Outline

- Introduction: refinement steps and verification
- Modeling styles and the need for partial refinement
- Use of SV interfaces in RTL and abstract modeling
- Interfaces to adapt between levels of abstraction
- Results and performance
- Summary

Example System Model


arbitrating (master) bus interfaces
bus fabric
masters
slaves
CPU
peripheral
DMA
control
ROM
RAM
simple (slave) bus interfaces

Refinement Steps and Verification

- Abstract model is attractive early in the project:
  - easier to write correct functionality than in RTL
  - simulates faster
  - ideal vehicle for debugging your verification environment
- Abstract model is progressively replaced by RTL implementation

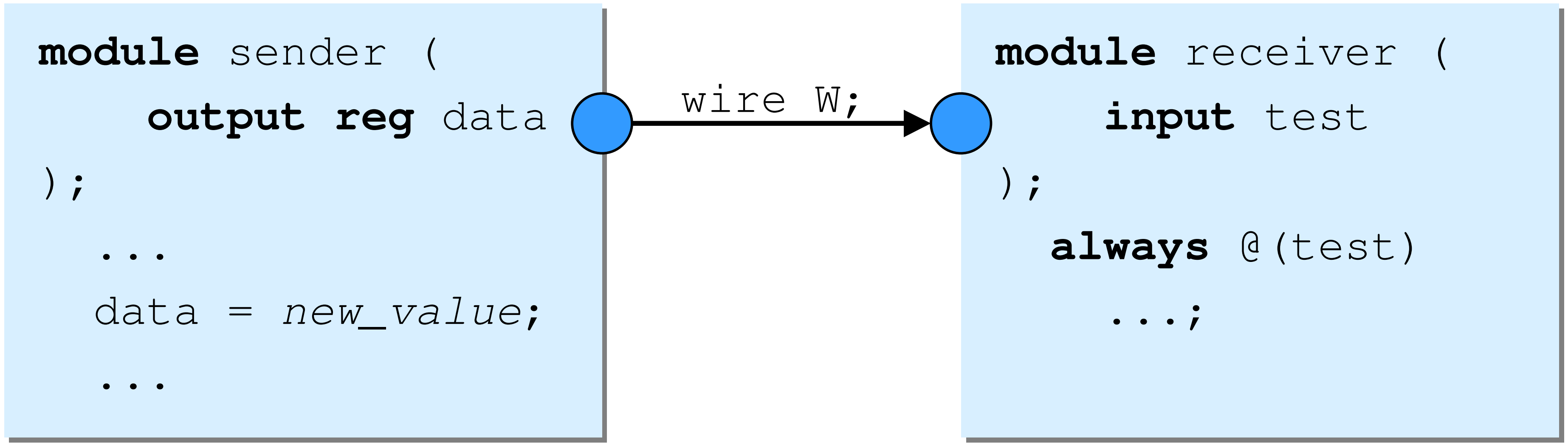
During development, abstract model is an ideal testbed for individual RTL components

Outline

- Introduction: refinement steps and verification
- Modeling styles and the need for partial refinement
- Use of SV interfaces in RTL and abstract modeling
- Interfaces to adapt between levels of abstraction
- Results and performance
- Summary

RTL Connection

- Module-to-module connections use ports and wires



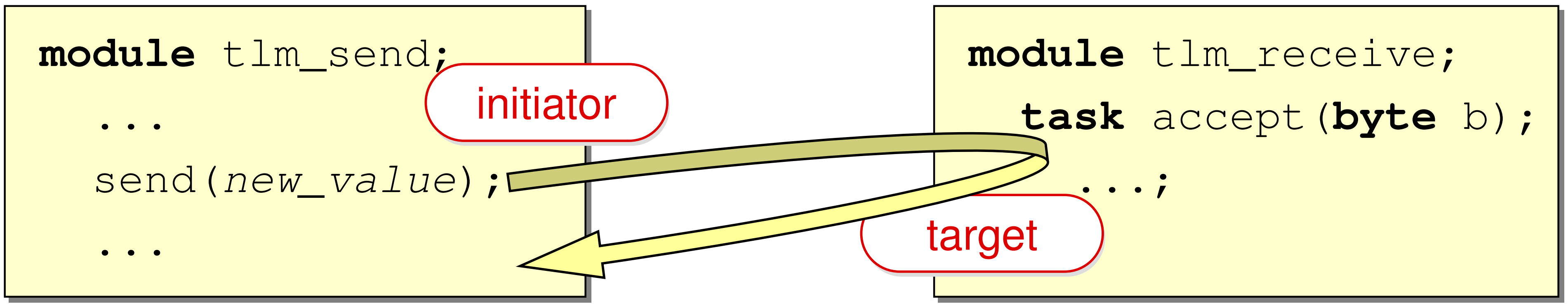
```
module sender (
    output reg data
);
  ...
  data = new_value;
  ...
```

```
module receiver (
    input test
);
  always @(test)
    ...;
```

- – Complete decoupling of connected modules
- – Slow simulation (processes sensitive to events)
- – Synthesizable

TLM Connection

- Communication by subroutine call



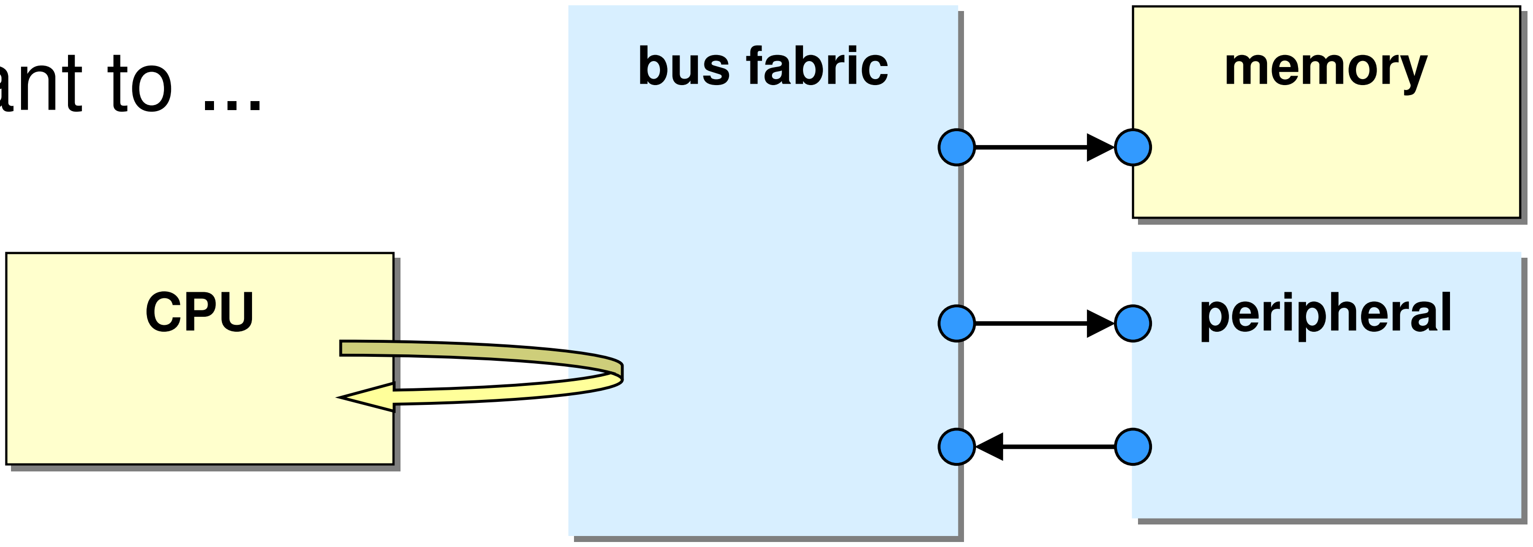
```
module tlm_send;
  ...
  send(new_value);
  ...
```

```
module tlm_receive;
  task accept(byte b);
    ...;
```

  - Much faster simulation (one call can send lots of data)
  - Not synthesizable (cross-module subroutine call)
  - Decoupling of connected modules is more difficult
  - The whole idea of connection is much less obvious

Mixed Modeling Styles

- I want to ...



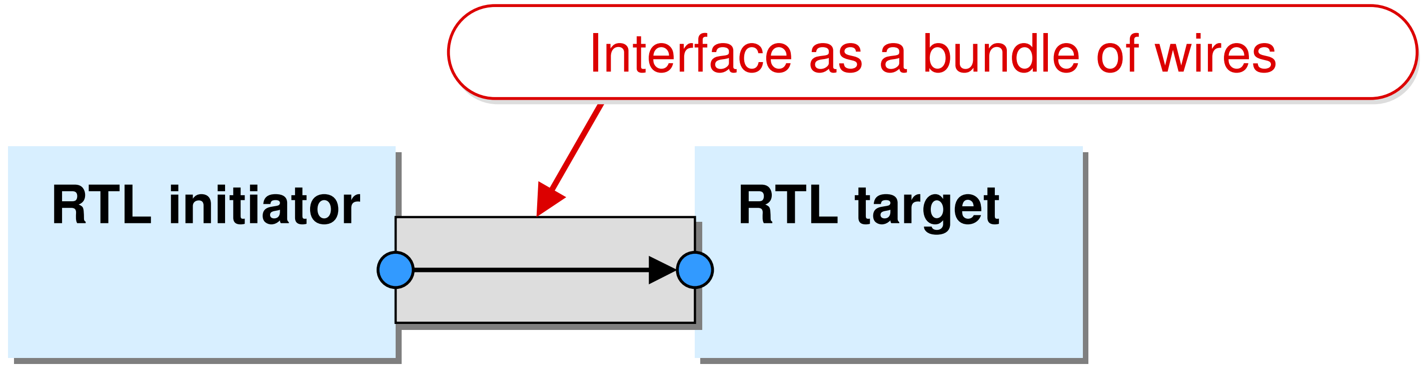
  – freely mix RTL and TLM styles in one system model
  – mix modeling styles on a single component
  – swap models without disturbing the rest of the system

Outline

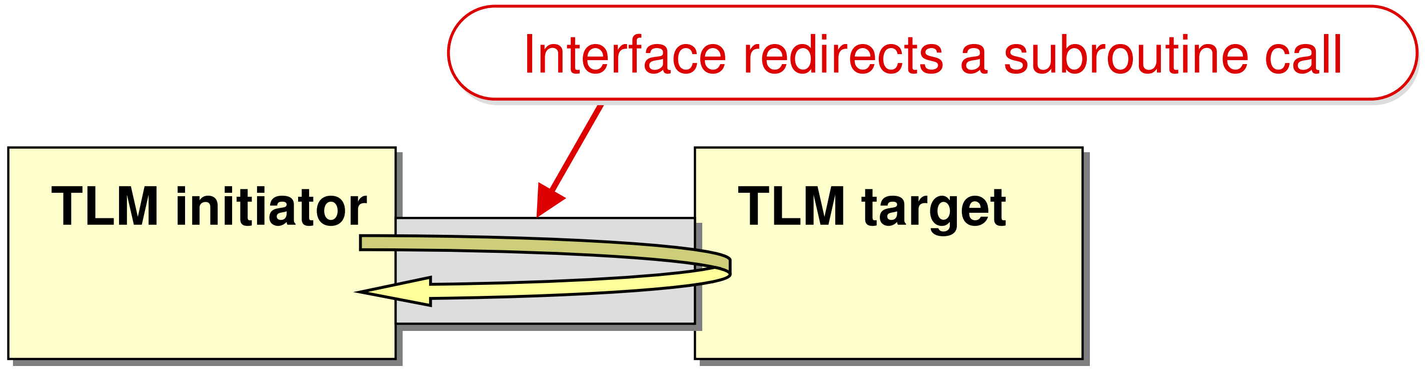
- Introduction: refinement steps and verification
- Modeling styles and the need for partial refinement
- Use of SV interfaces in RTL and abstract modeling
- Interfaces to adapt between levels of abstraction
- Results and performance
- Summary

SystemVerilog Interfaces Support RTL and TLM Connection Styles

- Well-known, synthesizable RTL usage:



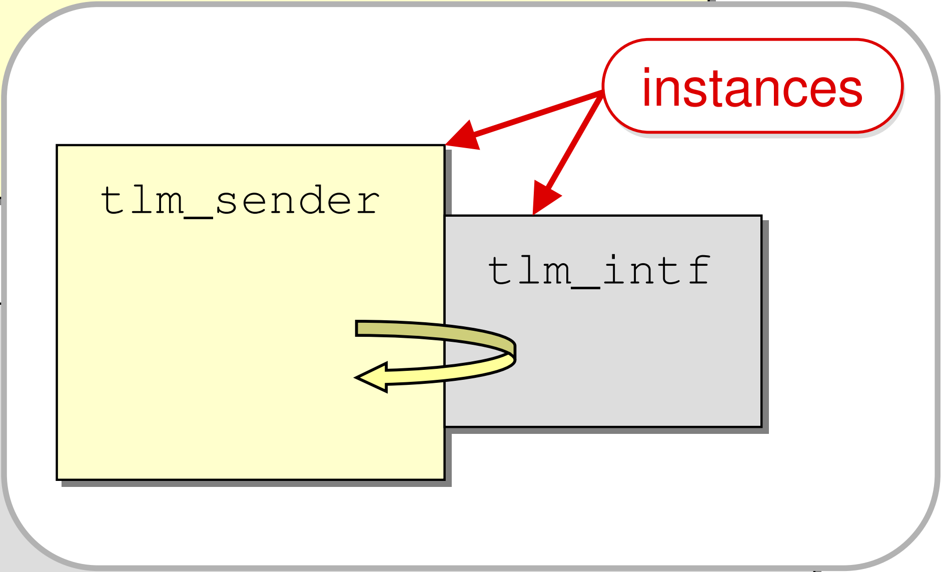
- Less familiar, but fully supported:



Subroutine Import

```
module tlm_sender ( tlm_intf.sender_mp send_port );

   ...
   send_port.put ( n );
   ...
endmodule
```

module does not know interface instance name



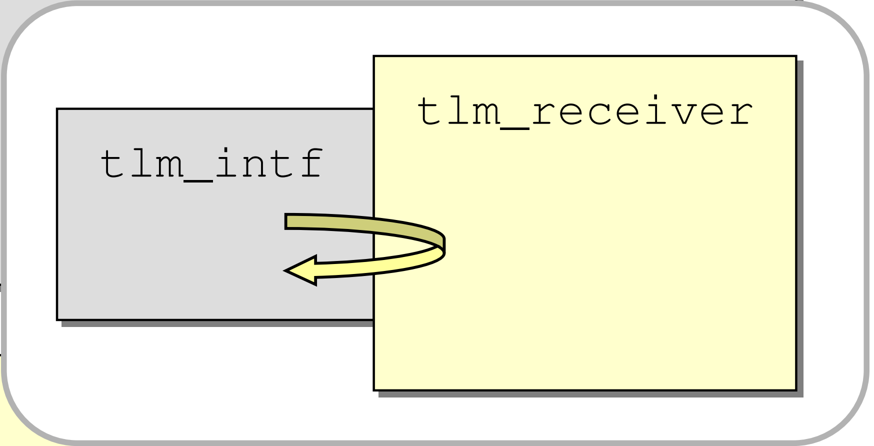
```
interface tlm_intf ();
   task put (input byte b);
     ...;
   endtask
   modport sender_mp ( import task put (input byte b) );
endinterface
```

provide task to a connected module

Subroutine Export

get task from a connected module

```
interface tlm_intf ();
  modport receiver_mp ( export task put (input byte b) );
  ...
  initial
    put ( "A" );
endinterface
```



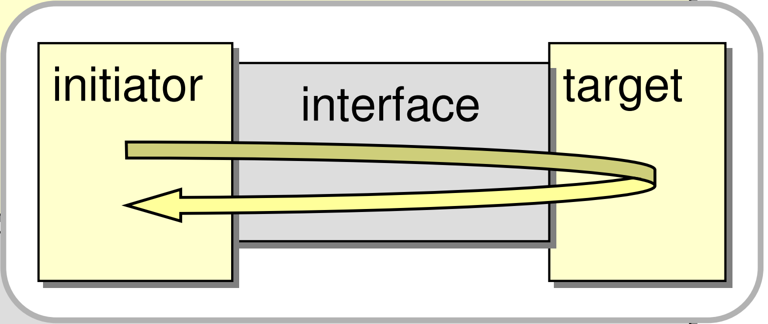
```
module tlm_receiver (
  tlm_intf.receiver_mp receive_port );

  task receive_port.put ( input byte b );
    $display("I got data=%0d", b);
  endtask

endmodule
```

module does not know interface instance name

TLM-Style Connection Through an Interface

```
module tlm_sender ( tlm_intf.sender_mp send_port );
  initial
    send_port.put ( n );
endmodule
```

call redirected to connected target



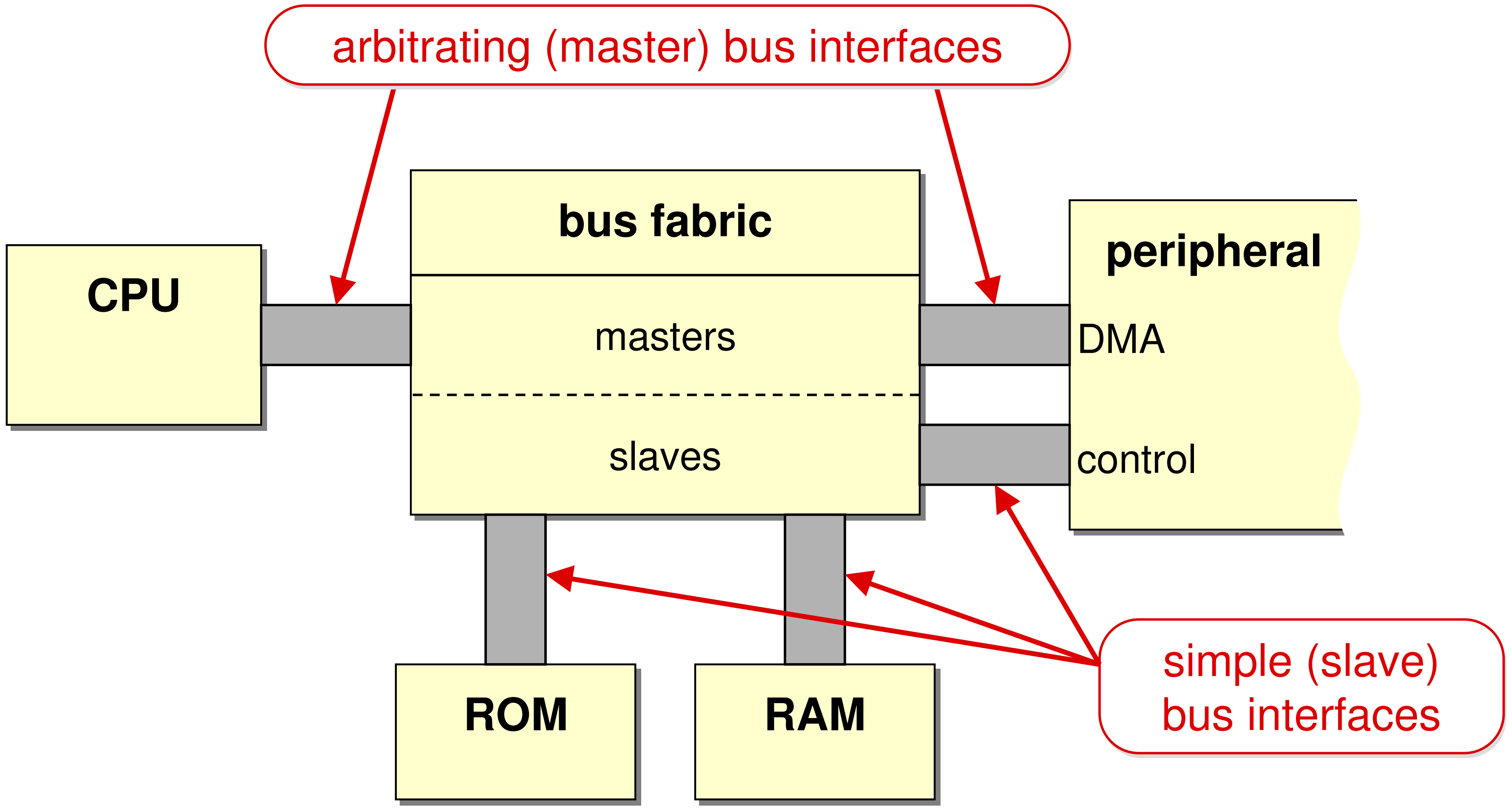
```
interface tlm_intf ();
  modport sender_mp ( import task put (input byte b) );
  modport receiver_mp ( export task put (input byte b) );
endinterface
```

interface is only a connection

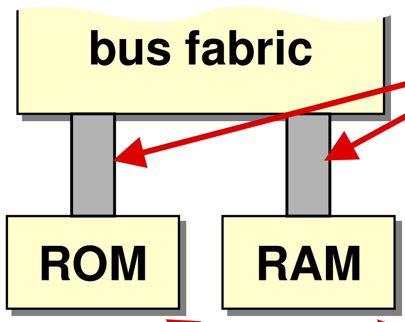
```
module tlm_receiver ( tlm_intf.receiver_mp receive_port );
  task receive_port.put ( input byte b );
    $display("I got data=%0d", b);
  endtask
endmodule
```

called by connected initiator

Example System Model


arbitrating (master) bus interfaces
bus fabric
masters
slaves
CPU
peripheral
DMA
control
ROM
RAM
simple (slave) bus interfaces

Multiple Exports Problem



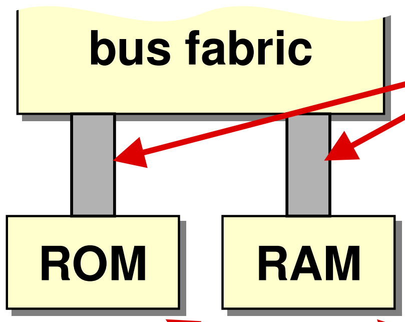
```
interface tlm_sbi ();
  modport fabric_mp ( import task write ... );
  modport slave_mp ( export task write ... );
endinterface
```

```
module tlm_ROM (tlm_sbi.slave_mp SBI);
  task SBI.write (...);
    $display("error, write to ROM %m");
  endtask
  ...
```

```
module tlm_RAM (tlm_sbi.slave_mp SBI);
  task SBI.write (input [15:0] addr, data);
    mem[addr] = data;
  ...
```

VCS error: `tlm_sbi.write` double-defined

Multiple Exports Solution



```
interface tlm_sbi ();
  modport fabric_mp ( import task write ... );
  modport slave_mp ( export task write ... );
endinterface
```

```
module tlm_ROM (interface.slave_mp SBI);
  task SBI.write (...);
    $display("error, write to ROM %m");
  endtask
  ...
```

```
module tlm_RAM (interface.slave_mp SBI);
  task SBI.write (input [15:0] addr, data);
    mem[addr] = data;
  ...
```

port has generic interface type

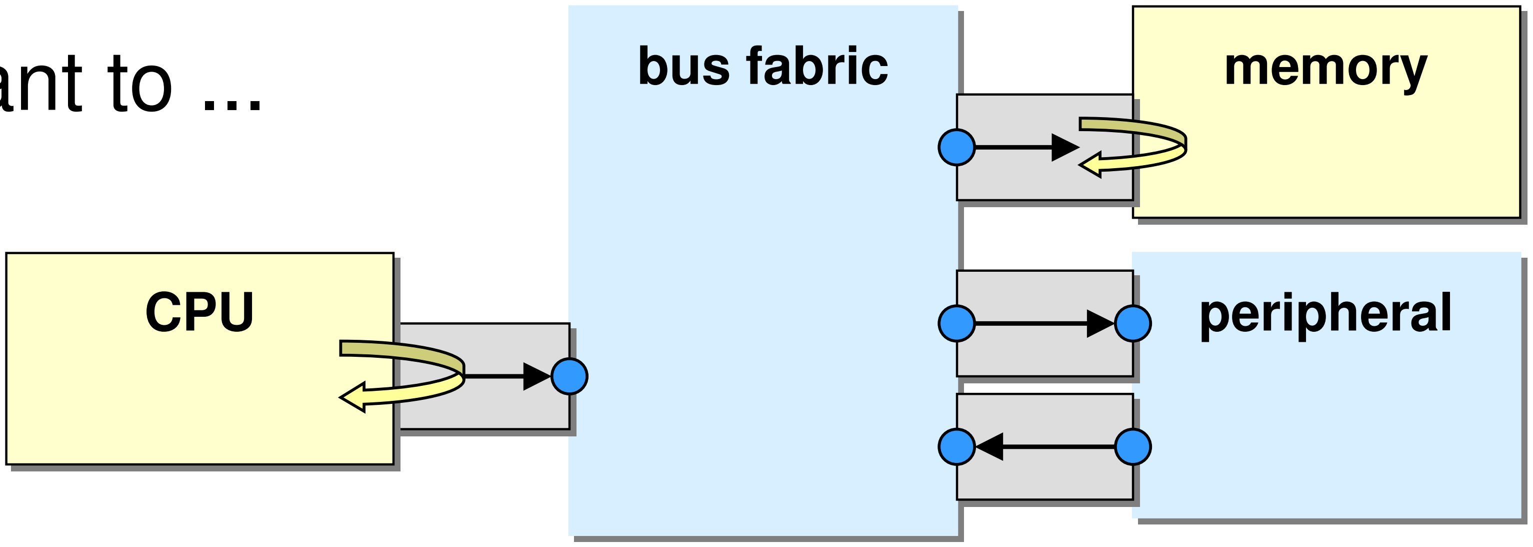
Generic port can be connected to any interface that has the correct modport

Outline

- Introduction: refinement steps and verification
- Modeling styles and the need for partial refinement
- Use of SV interfaces in RTL and abstract modeling
- Interfaces to adapt between levels of abstraction
- Results and performance
- Summary

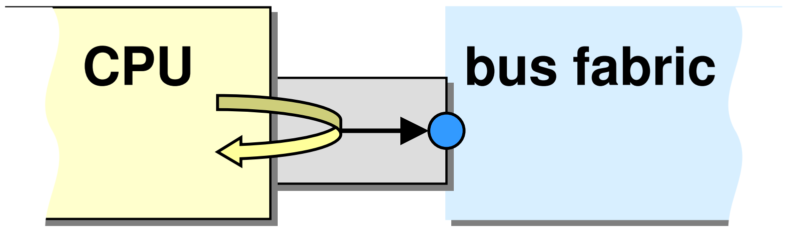
Mixed Modeling Styles

- I want to ...



  - freely mix RTL and TLM styles in one system model
  - mix modeling styles on a single component
  - swap models without disturbing the rest of the system

Abstraction Adapter Interface



- Interface provides a task that manipulates signals

```
interface tlm_to_rtl_abi (input logic clk);
  logic RE, WE;
  logic [15:0] addr, wdata, rdata;

  modport rtl_fabric (
    output rdata, input wdata, addr, RE, WE );
  task write ( input [15:0] A, D );
    @(posedge clk) addr = A; WE = 1; RE = 0; wdata = D;
    @(posedge clk) WE = 0;
  endtask

  modport tlm_master ( import task write ... );
endinterface
```

RTL-like signals and modport

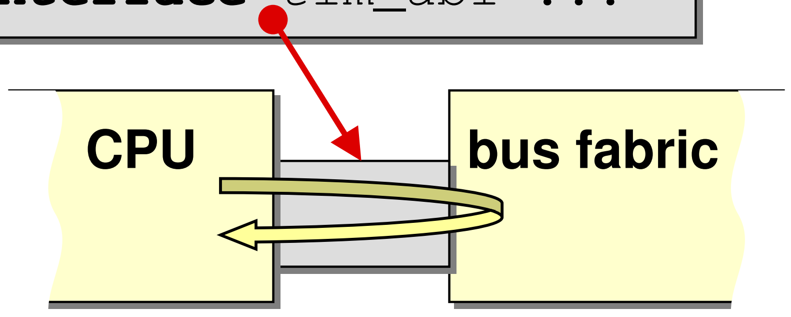
this is simply a BFM

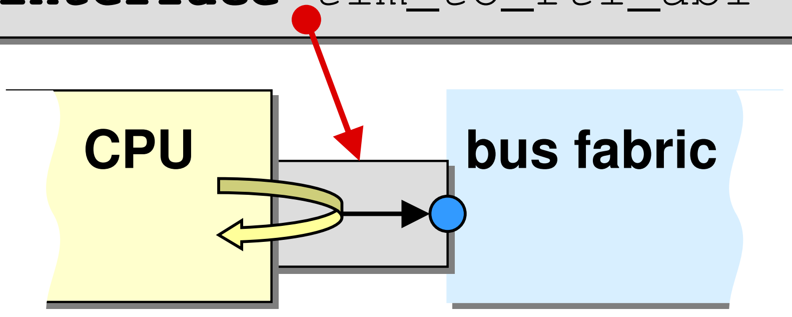
Swapping-in a Modified Model: Use an Adapter Interface

- The interface's name changes ...

```
interface tlm_abi ...
```

```
interface tlm_to_rtl_abi ...
```





- so the *unaltered* module's port list must change ...

```
module tlm_CPU (
  tlm_abi.tlm_master ABI,
  ...
```

```
module tlm_CPU (
 tlm_to_rtl_abi.tlm_master ABI,
  ...
```

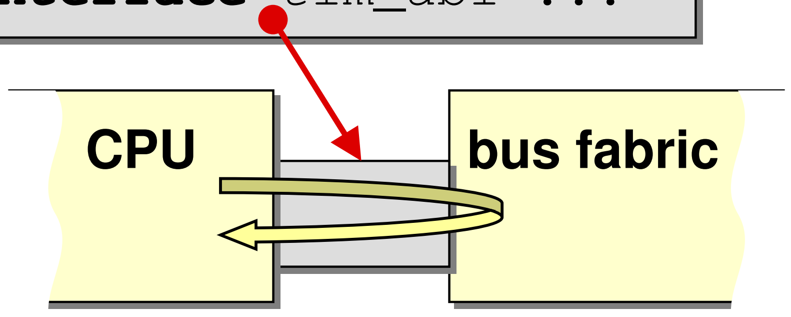
- This is unacceptable!

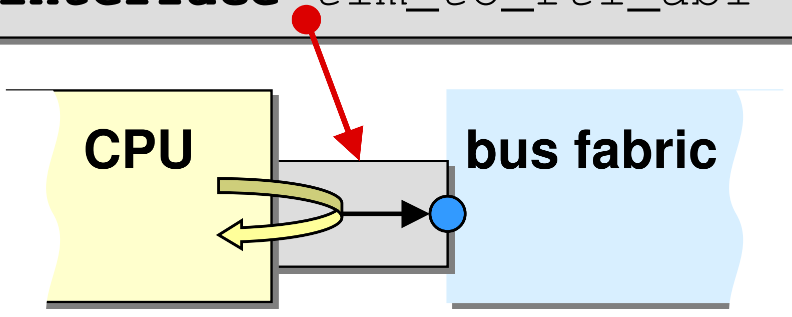
Swapping-in a Modified Model: Generic Interface Ports

- The interface's name changes ...

```
interface tlm_abi ...
```

```
interface tlm_to_rtl_abi ...
```





- and with generic interface ports ...

```
module tlm_CPU (
  interface.tlm_master ABI,
  ...
```

```
module tlm_CPU (
 interface.tlm_master ABI,
  ...
```

the unaffected module's port list can remain the same!

Outline

- Introduction: refinement steps and verification
- Modeling styles and the need for partial refinement
- Use of SV interfaces in RTL and abstract modeling
- Interfaces to adapt between levels of abstraction
- Results and performance
- Summary

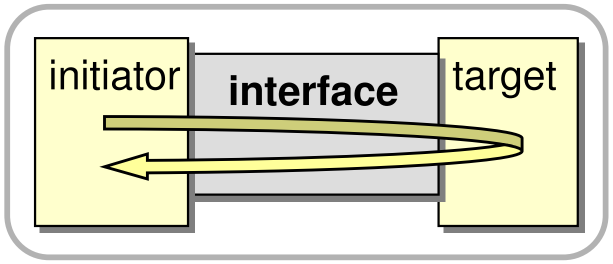
Results and Performance

- Example system modeled with various combinations of TLM and RTL:

modeling style	simulation run time (seconds)	simulated CPU instructions per second of simulation
all TLM	8	> 1 000 000
all TLM except RTL RAM	14	650 000
all TLM except RTL RAM and ROM	48	194 000
all RTL	732	12 800

Further Opportunities

- Exported/imported tasks can be implemented by DPI:



```
interface tlm_intf_SV_to_SV ();
  modport sender_mp ( import task put (...) );
  modport receiver_mp ( export task put (...) );
endinterface
```

Target is a SystemVerilog module

```
interface tlm_intf_SV_to_C ();
  modport sender_mp ( import task put (...) );
  import "DPI-C" context task put (...);
endinterface
```

Target is a function written in C

Summary

- Subroutine import/export in interfaces facilitates high-level modeling in a static RTL-like architecture
  - Less versatile than true TLM using SystemC etc
- Interface can be used as RTL-to-TLM adaptor
- All within a single-language environment
  - Existing Verilog behavioral models easily integrated
  - Familiar, static style of instantiation and connection

Thanks for your attention!

Questions?

Jonathan Bromley

Doulos Ltd, Ringwood, UK

jonathan.bromley@doulos.com