

Linking TLM and Synthesis

3/7/2006

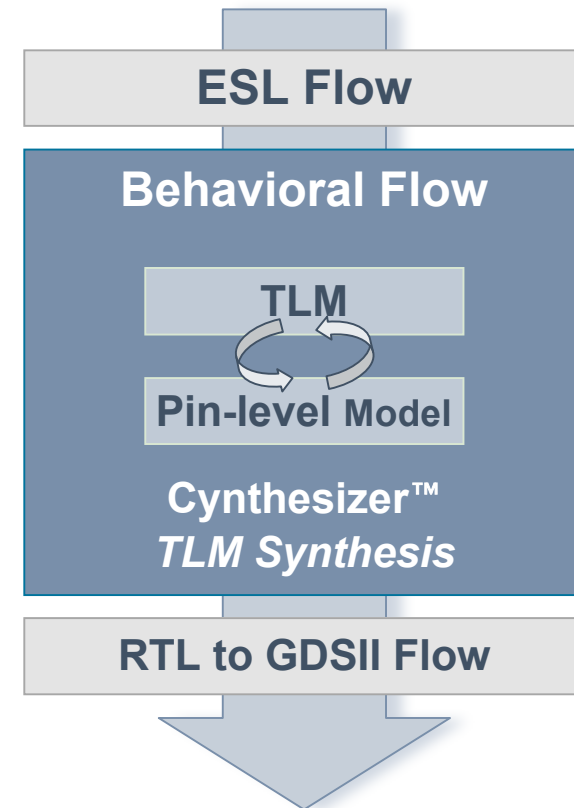
FORTE
DESIGN SYSTEMS

TLM Synthesis

- **Creates the link**
 - ESL system-design
 - Behavioral design
- **Completes the flow**
 - Integrated ESL→GDSII flow
- **Connects the team**

Single language and representation

 - System architects
 - Embedded software engineers
 - Verification engineers
 - Block-level hardware designers



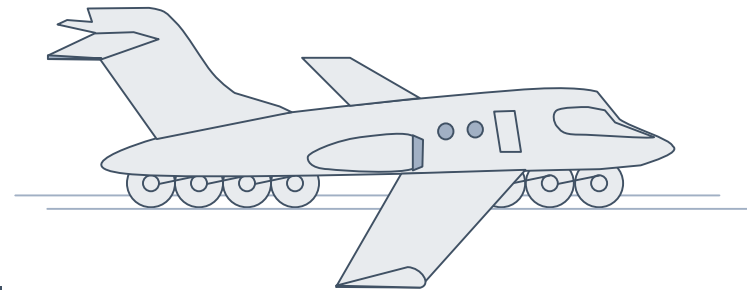
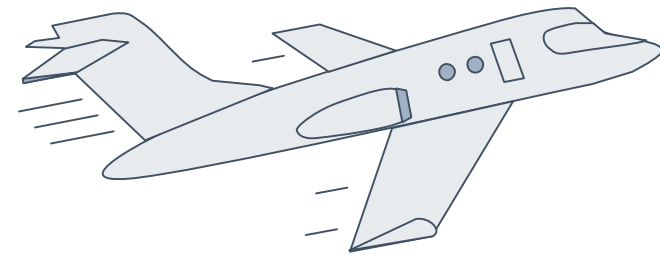
Outline

- **Overview**
 - TLM and synthesis requirements
- **Linking TLM and synthesis**
 - Point-to-point interfaces
 - Bus interfaces
- **Summary**

Compare Requirements

- **TLM requirements**
 - ***SPEED!***
 - **Appropriate detail**
 - Packet order
 - Memory map
 - Algorithm

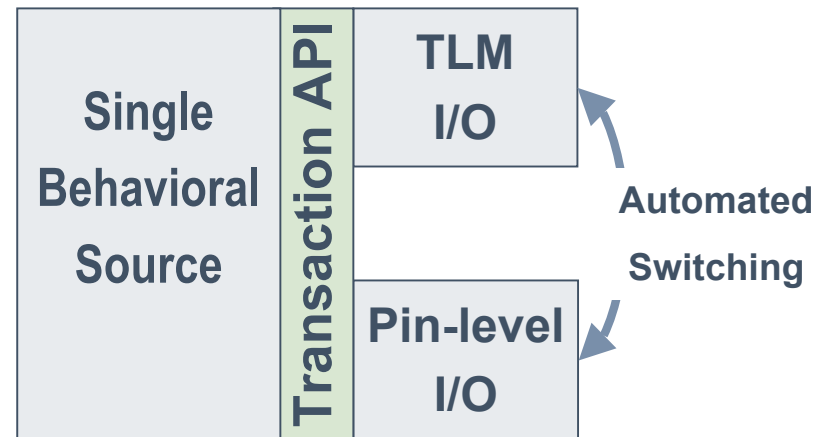
- **Synthesis requirements**
 - **Accuracy**
 - **Necessary detail**
 - Pin-accurate interface protocol
 - Memory map
 - Algorithm



Maintain common code.
Change communication
mechanism.

Maintaining Common Code

- **Model inconsistency can be costly**
 - **Redundant model development**
 - **System and software errors**
 - Caused by differences between model and implementation



- **Keeping as much code common as possible avoids these problems**

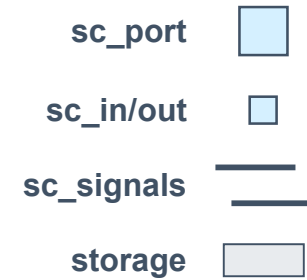
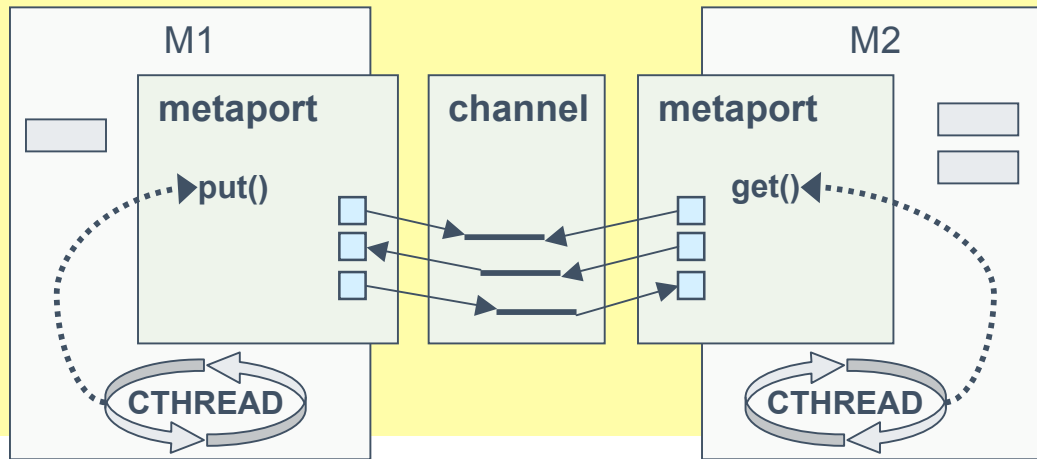
Use the same behavioral source for TLM system and software simulation as well as PIN-level synthesis and verification

Point-to-Point Interfaces

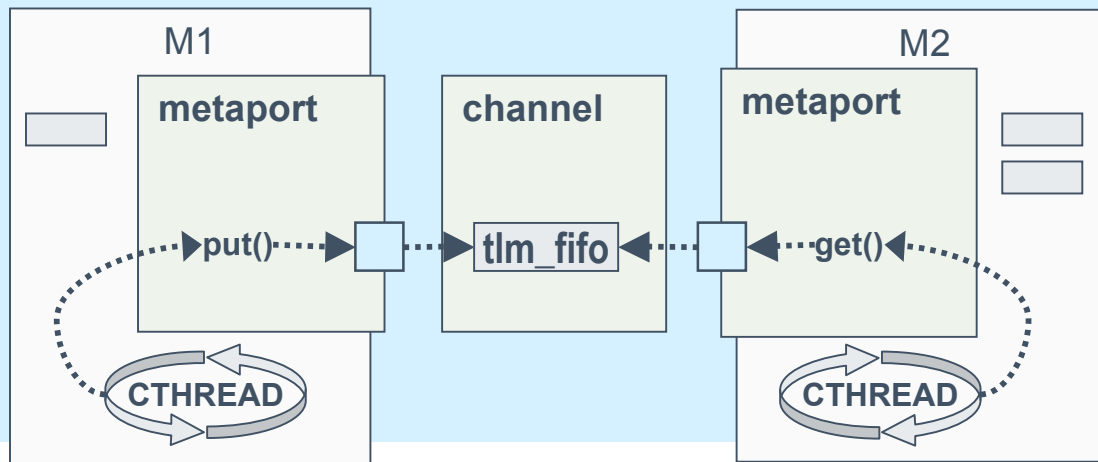
FORTE
DESIGN SYSTEMS

Modular p2p Interfaces – TLM and PIN

Pin-level



Transaction-level



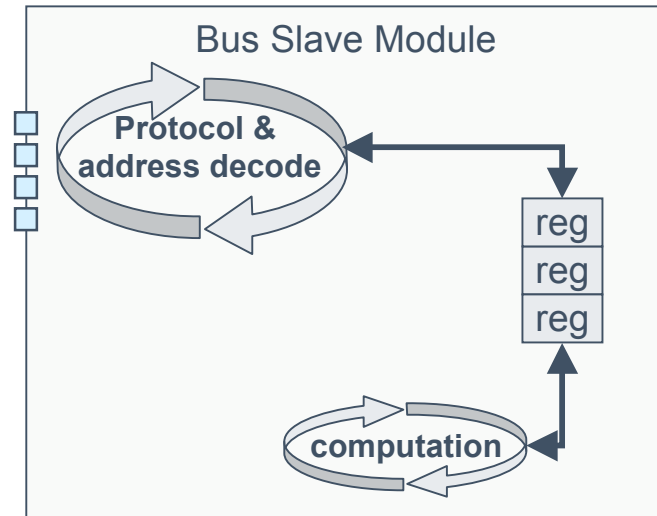
**Maintain
algorithm
and
structure.
Change
ports and
channels.**

Bus Interfaces

FORTE
DESIGN SYSTEMS

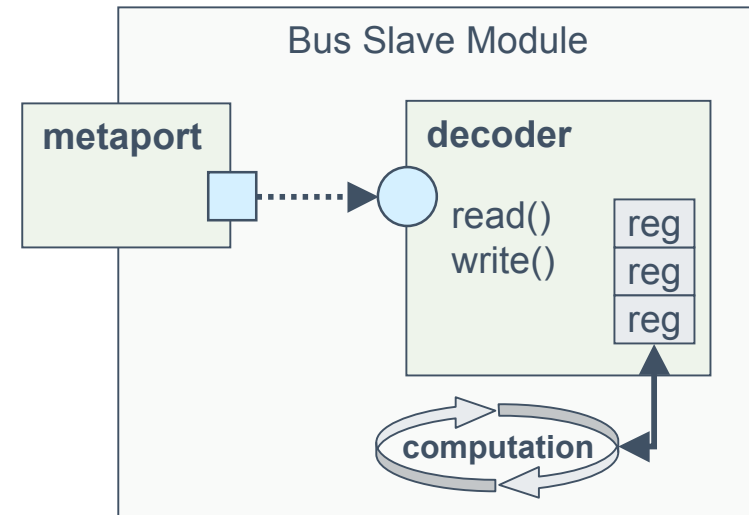
Problems with Building Synthesizable Bus Interfaces

- **Main elements of bus peripheral**
 - Bus ports
 - Bus protocol
 - Address decode
 - Registers / Memories
 - Algorithm
- **Lack of modularity**
 - Usually all elements thrown in with computation using no modular boundaries
- **Hard to satisfy reuse requirements**
 - Reuse bus protocol across projects
 - Automatically change between PIN and TLM
 - Easily produce derivative designs using different bus interfaces



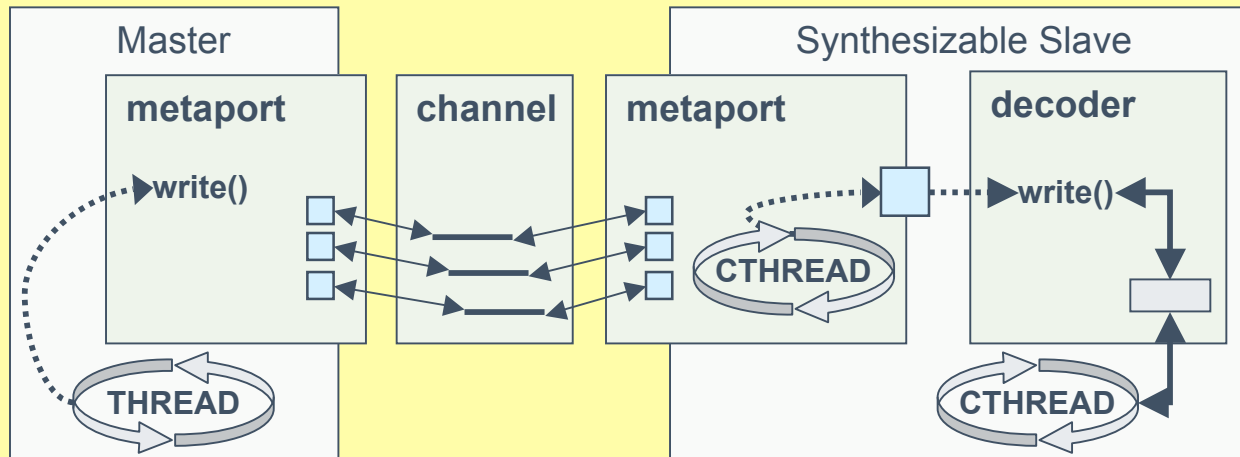
Building Modular Bus Interfaces

- **Increase modularity**
 - **Keep bus interface separate**
 - Ports and protocol in one object
 - Decode and storage in another object
- **Key Technique**
 - Put address decode and storage in “decoder” object
 - Put bus ports and protocol in metaport
 - Use `sc_port` to communicate from metaport to decoder

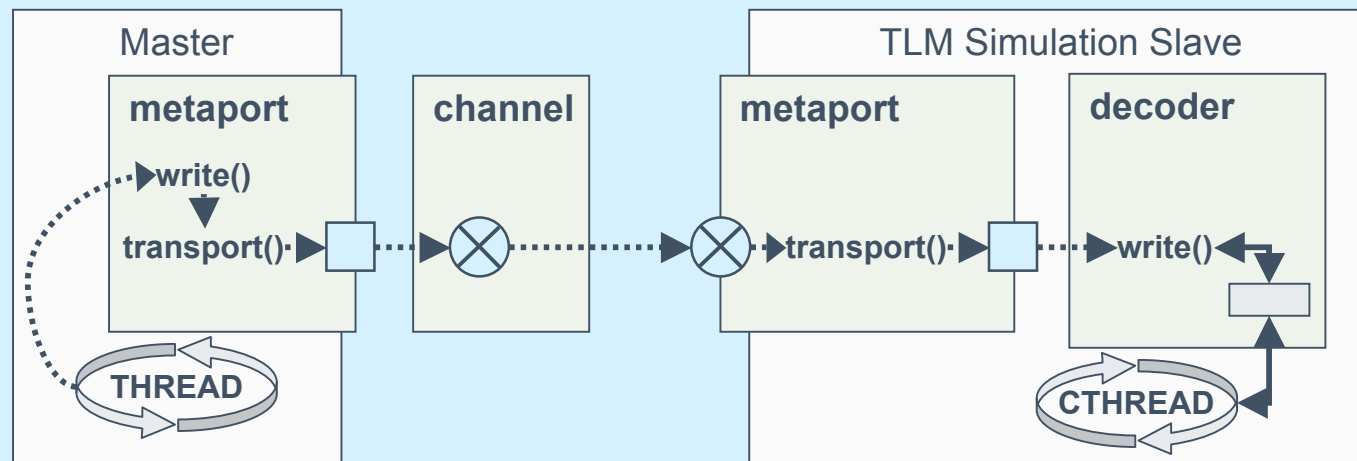


Modular Bus Interfaces – TLM and PIN

Pin-level



Transaction-level



Goal 1: Create modularity in slave. Goal 2: Avoid thread in TLM metaport.

Summary

- **Maintain same structure and binding semantic**
 - If the connection represents wires put in a channel – even if it's just an `sc_export`
- **Use modularity to maintain common code**
 - Keep algorithm and storage
 - Change communication ports and channels
 - For bus interfaces use `sc_port / sc_interface` to isolate bus details from design specifics
 - Bus protocol vs decode/storage/algorithm