Beyond TLM 2.0: New Virtual Platform Standards Proposals from ST and Cadence

NASCUG at DAC, June 6, 2012

Stuart Swan, Cadence Design Systems, Inc.

Jerome Cornet, STMicroelectronics

Beyond TLM 2.0: What's still needed?

- Already Being Developed in Accellera Systems Initiative:
  - Model parameter API – Accellera Systems Initiative CCI WG
  - HLS Synthesis Guidelines – Accellera Systems Initiative Synthesis WG
  - Protocol Specific Extensions – Protocol Owners (e.g. ARM, OCP)

- Still Needed:
  - Wire (Interrupt, reset, …) Modeling in VP
  - Register Proxy API
  - System Address Map API

General Principles

- There is a large existing base of TLM 2.0 models
  - Work with these models!
- Focus on external interfaces of models, and model-tool interfaces
  - Do not dictate how the internals of models are written
- Avoid “all or nothing” approaches
- Be truly open, openly invite improvements
  - Full code available with Apache 2.0 license from the start

Summary

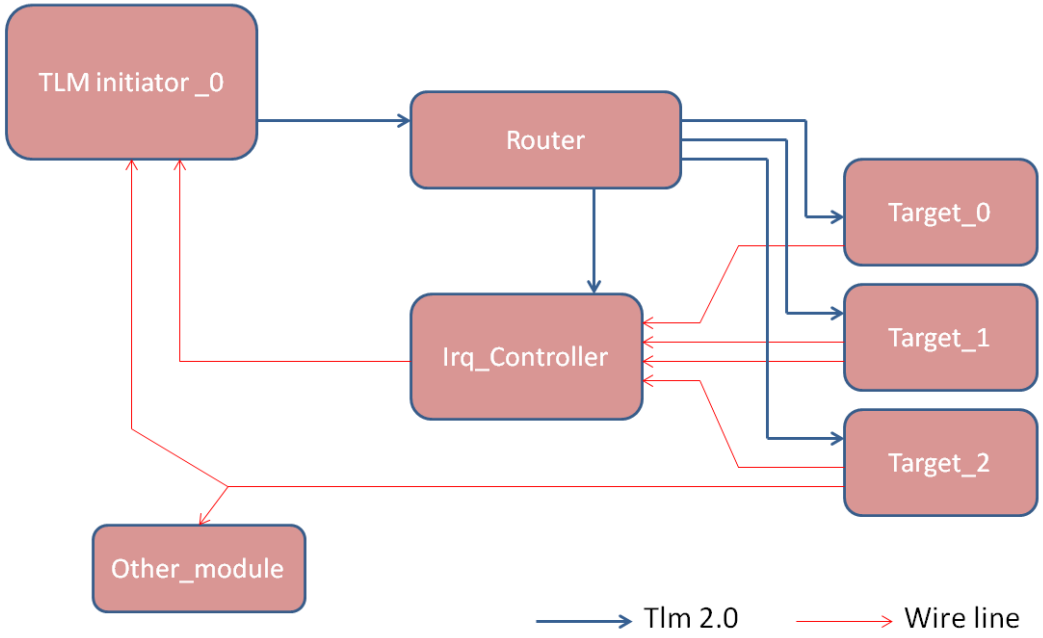
- Wire modeling
- Register Proxy API
- System Address Map API

Summary

- Wire modeling
- Register Proxy API
- System Address Map API

Virtual Platform Wire Modeling

- Problem: TLM 2.0 does not specify how wire signals are modeled
  - All VPs have interrupt, reset, etc.
  - Major problem for interoperability!



Virtual Platform Wire Modeling

- Naïve solution: use sc_signal
  - Value change not seen immediately, receiver must wait() till next delta cycle
    - Not efficient
    - In some cases, loss of faithfulness
  - Multiple drivers issue
    - In TLM multiple processes often end up driving change on a given wire line (example: Mailbox)
  - Which datatype use?
    - sc_logic?
    - sc_bit?
    - bool?
    - int?

sc_wire: Key Requirements (1/2)

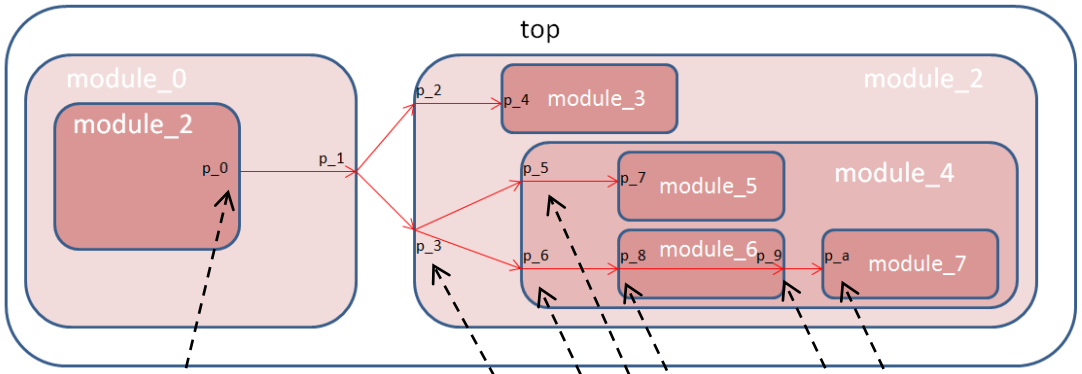
- Value changes should be propagated immediately as transactions in TLM-2
  - Solution: use the IEEE SystemC TLM-1.0 Message Passing API
- Datatype should be well defined and standardized
  - Solution: use bool datatype
- Need to easily handle groups of wires, including easy split & merge & binding of multiple bits
  - Solution: integrate with IEEE 1666-2011 sc_vector
- Provide automatic interoperability with sc_signal
- Initial values must be precisely modeled
- Unconnected ports with specific initial values must be supported

sc_wire: Key Requirements (2/2)

- Must be possible to model combinational logic
  - including resolution of proper outputs at start of simulation
  - virtual platform interrupt controllers often need this
- Must be able to model modules that have multiple sc_wire inputs & distinguish between them
- Must support feedthroughs
- Must not impose restrictions on port & module construction / binding / elaboration order
- Must enable tools to debug & trace sc_wires in system

sc_wire for VP wire signal modeling

- TLM-1 style: sc_wire models control signals as direct function calls
  - Recipient can see & respond to control signals without calling wait().
  - Intermediate recipients may transform the signal (combinational functions)



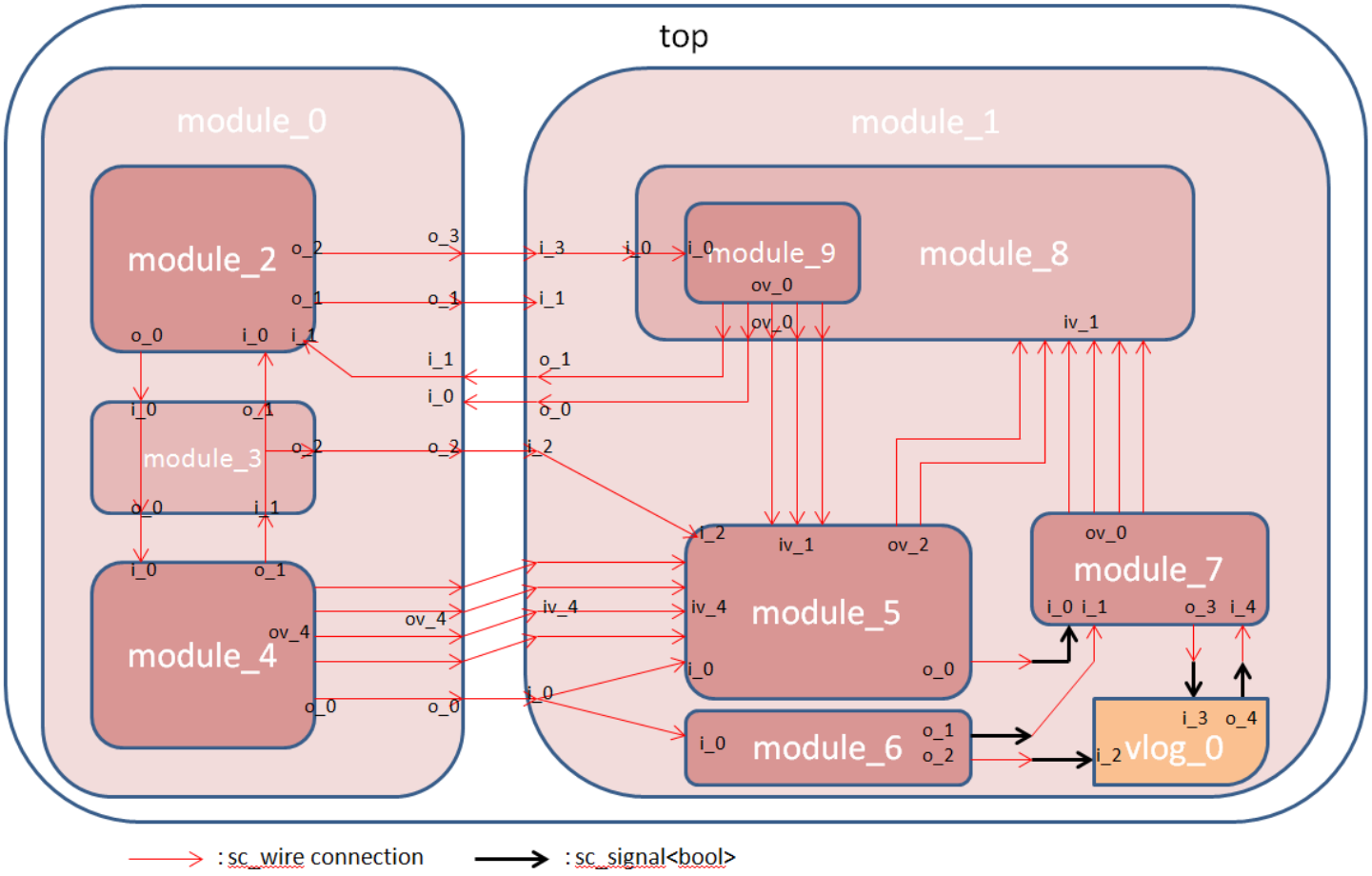

```
p_0.write(1);
```

```
virtual void write(bool v) {
  newval = v;
}
```


SYSTEMC™

A more complex sc_wire example

- This example (and others) are included in the kit



Summary

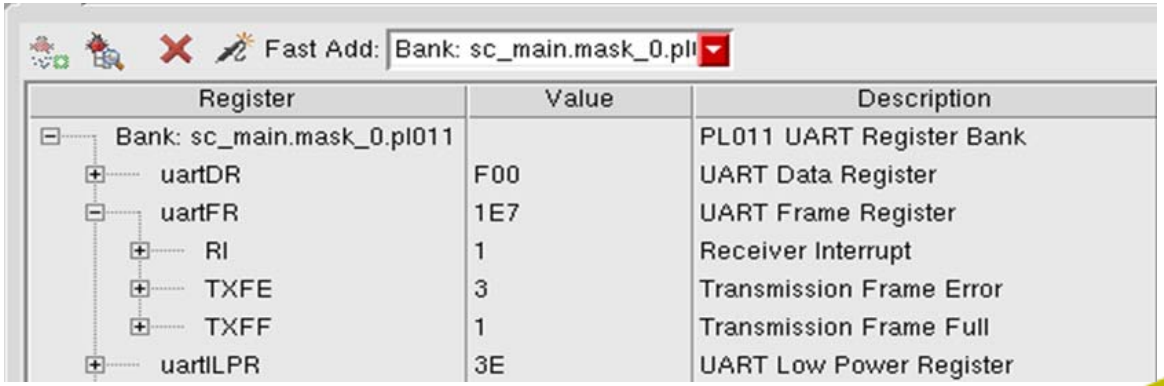
- Wire modeling
- Register Proxy API
- System Address Map API

Register Proxy API

- Reality today for SC register models:
  - Most users have pre-existing classes for modeling registers in TLM models.
  - Different types of models have different register model requirements
    - E.g. TLM2, HLS, signal level, verification models
    - Different types of register classes often used for good reasons
- Register Proxy API is a model-tool API enabling tool to automatically query registers and register banks
  - Enables registers, register banks to be visualized, debugged, traced, etc.
- Proxy API needs to work with existing user register classes
  - it is not a replacement for these classes!
- Proxy API needs to work with TLM systems that use a *mix* of different user register classes (e.g. use models from different vendors).

Register Proxy API Key Requirements

- Register Proxy API needs to allow tool to:
  - automatically find all register banks and registers
  - query names, descriptions, and bitwidths of and registers and fields
  - query for local offsets of register banks and registers
  - get and set value of registers and fields
  - get event that is notified when value of register changes during simulation
  - find TLM2 target socket associated with a register bank
  - hierarchical register banks are supported



SYSTEMC™

Register Proxy API Implementation

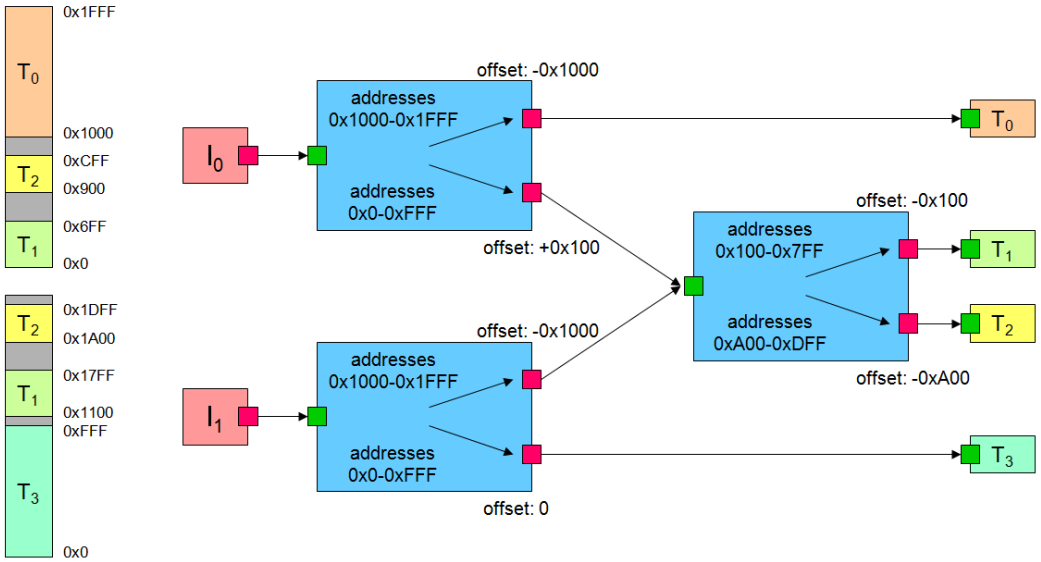
- Register Proxy API is just a few small abstract base classes
  - These classes have no state information
  - They are integrated with user register classes via multiple inheritance
  - User register classes then implement proxy API methods
- Tool can find all registers and register banks via a global registry
- Proxy API classes are invisible to model writer

Summary

- Wire modeling
- Register Proxy API
- System Address Map API

System Address Map API

- The Problem:
  - System address map is essential for visualization, debugging
  - TLM 2.0 has no API / mechanism for tool to obtain the system address map
  - System address map depends on interconnection of models
  - Each system initiator may see a different system address map



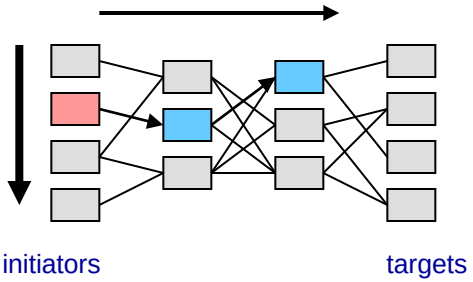
Key Requirements

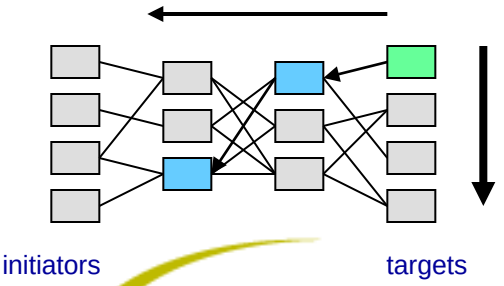
- System address map must be automatically computed based on structure of interconnect and local address maps
  - if it's not automatic, it will be source of bugs and confusion
- Any TLM 2.0 interconnect topology must be supported
  - Cascaded routers must be supported
  - Arbiters, crossbars, muxes, multi-passthrough sockets, etc.
  - Must work with existing TLM 2.0 interconnect models
- Standard must not dictate how interconnect components are written
  - Routers may have their own peculiar approaches for address decoding and transaction propagation
- Dynamic memory map reconfiguration must be supported
- Must work both for tools and models
- Must not modify TLM 2.0 header files - work with existing release

How can a tool discover the global address map of each initiator ?

- Goal: find out the memory map of each initiator
  - that is, the set of reachable targets with the address ranges selecting each of them

- Two possible approaches of the problem:
  - from each initiator, traverse the forward routing tree to find out all leaf target sockets, and compute how they can be reached according to address transformations
  - from each target, traverse the backward routing tree to find out all leaf initiator sockets, and compute how they can reach the target according to address transformations





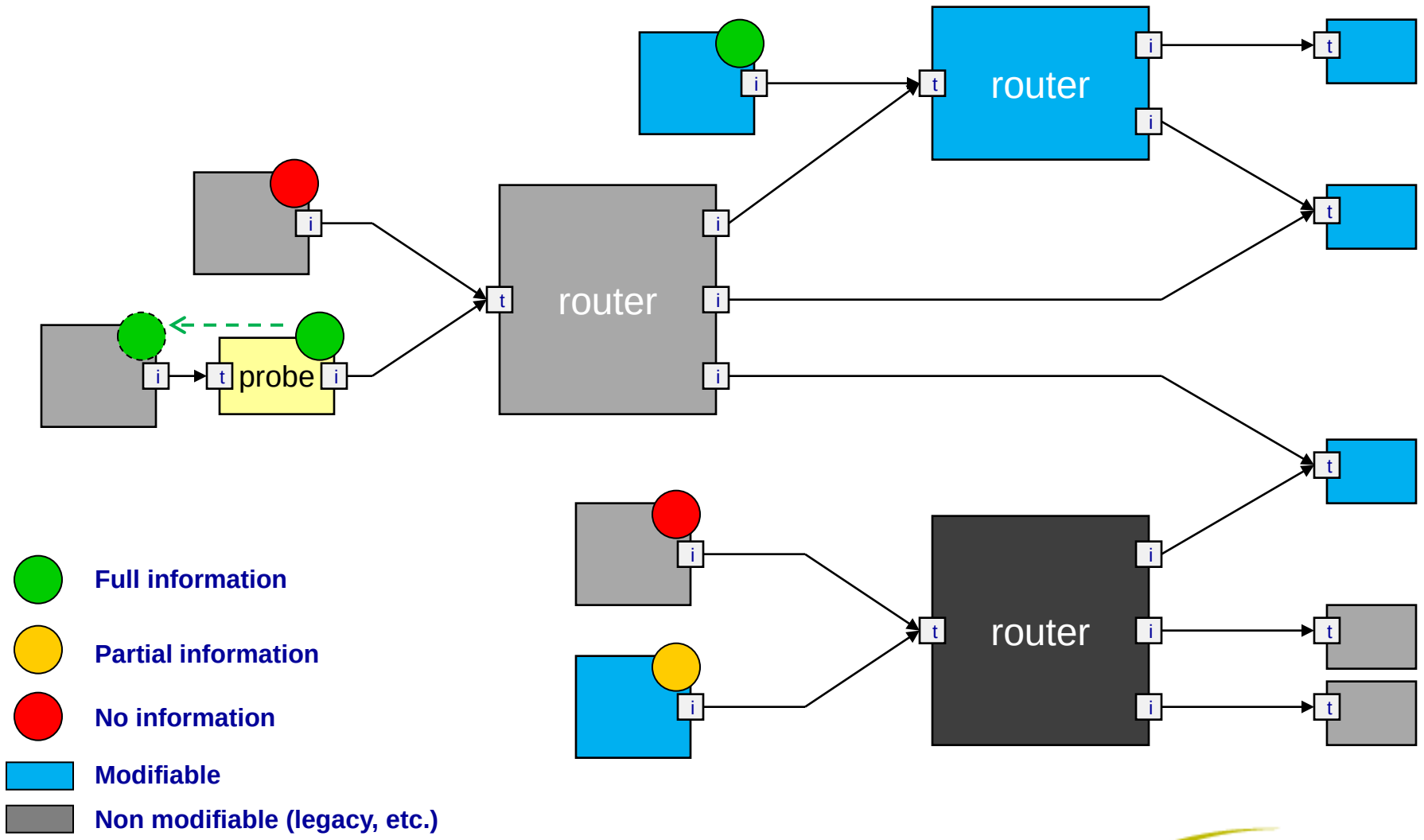
Technical details

- Traversing forward path
  - Performed using transport_dbg(), IGNORE command and ignorable extension
  - Main API to request address map information at one point in the system
  - Requires dedicated support in interconnects and targets
  - Can be used on target side to provide additional information (local address range, etc.)
- Traversing backward path
  - Performed by initiating invalidate_direct_mem_ptr() from all target sockets
  - Main way to collect information on unmodified systems
  - Requires interconnects to perform range clipping and broadcast on backward path
  - Requires “something” to collect ranges on initiator side

Combining forward and backward

- Forward path (FW) provides natural way to collect information
  - Return path of transport_dbg() call
  - Extension allows to collect more than [start address, end address]
- Backward path (BW) provides natural way to extract information from legacy models
  - Interconnects receiving invalidate_direct_mem_ptr() usually clip ranges and transform addresses like regular transactions on return of forward path
  - No other modification required in interconnect components
- Combination
  - Use FW path to request information to interconnects and targets
  - FW path implementation in modified interconnects calls both BW and FW
  - "Probes" allows to collect BW path ranges using FW

Example


router
router
router
probe
i
t
Full information
Partial information
No information
Modifiable
Non modifiable (legacy, etc.)

Address map proposal summary

- The standard proposal contains:
  - An extension carrying address map information for use on FW path
    - Main target range
    - Sub-ranges
  - APIs to collect ranges on BW path
    - Is invalidate call for introspection or for real?
    - send_collected_range()
- BW path
  - For use with existing platforms
  - Works 100% well when interconnects clip incoming address ranges and broadcast invalidate_direct_mem_ptr()
- FW path
  - To implement explicit support in interconnects and targets
  - Works 100% well when components are modified

Next Steps

- Obtaining code and examples for these proposals
  - See: http://www.accellera.org/apps/org/workgroup/tlmwg/documents.php
  - If problems, or not an Accellera member, use email contacts below
- Next steps in Accellera Systems Initiative
  - Licensing is Apache 2.0 – no formal contribution needed.
  - CCI WG
  - TLM WG
- Authors / Contacts:
  - Jerome Cornet, jerome.cornet@st.com
  - Laurent Maillet-Contoz, laurent.maillet-contoz@st.com
  - Vincent Motel, vmotel@cadence.com
  - Stuart Swan, stuart@cadence.com