Texas Instruments' Feedback about OSCI TLM-2.0 draft-2

February 2008

Contents

- Overall Reaction within TI is Positive
- More Detailed Investigations Ongoing
- Basically Just a List of Miscellaneous Technical Points and Questions to OSCI
  - Should we expect sockets to be used natively?
  - Are there any OSCI simulation speed benchmarks available?
  - Capture of OSCI-TLM2 sockets+extensions in IP-XACT?
  - Debug transaction extensibility
  - Quantum keeper
    - Dynamically change quantum from a component?
    - Better example needed
  - DMI
    - Extensibility
    - 'Ignorant' initiators
    - Data conversion in the path
  - Use of extension mechanism for private data
  - Interrupt modelling and time-warping

Details

- Should we expect sockets to be used natively?
  - It isn't clear from the kit whether the socket is expected to be used 'natively' or whether a user convenience wrapper will be provided
  - Does not affect interoperability: only talking about internal interfaces in a component
- Are there any OSCI simulation speed benchmarks available?
- Capture of OSCI-TLM2 sockets+extensions in IP-XACT
  - Is OSCI working on this, alone or with SPIRIT?

- Debug transaction extensibility
  - SoCs may conditionally restrict debug access to parts of the memory map
  - For example, based on a security level attribute of the debugger
  - This seems impossible in the draft kit

Details

- **Quantum Keeper**
  - Current approach appears to be that the user chooses the quantum period
  - If it is too big, the simulation could be broken
    - A stream of interrupts from a timer or similar device will be broken if the quantum is larger than the interval
  - Would be better to have the quantum period determined automatically and dynamically by the simulation
    - Subject to user-supplied maximum
    - For example a timer knows its period and could specify a new maximum-quantum when programmed
  - Would really appreciate a better quantum-keeper example
    - In the kit it is only exercised in a unit-test

Details: DMI

- DMI requests are not extensible
  - This is a pain. Access to memories frequently depends on more than is in the kit, for example security level, cacheability, requesting-initiator, etc.
  - Could change the tlm_dmi object, but this is very bad for compatibility
  - All extensions are by definition optional, because the DMI can always be refused. There's never a functional error if extensions are not present or not needed.
- The concept of DMI appears quite limited
  - Data transformations on the transaction path will prevent DMI
    - Endianness conversion or maybe even width conversion is enough
    - Shared memory between different endianness initiators will not allow both to use DMI
    - Other more sophisticated converters clearly fail
  - Non-contiguous memories will not support DMI
  - Where a CPU does not know which of its attributes affect its memory access rights, all DMI must be disabled for it
    - For example a CPU model believes 'if I can read, I can fetch instructions' but hardware outside the CPU makes this untrue in reality
    - If we ever give a data DMI pointer to the CPU model, it may fetch an instruction and we can not stop this
- BUT: "Full" DMI is only needed for shared memory
  - Often a relatively small proportion of total number of transactions
  - In non-shared memory data-correctness is not needed
    - Therefore a 'don't care' endianness is desireable
  - Some initiators will just allocate non-shared memory inside themselves
    - Doesn't allow detection of buffer corruption bugs though

Details

- **Use of extension mechanism for private data**
  - It seems very powerful to be able to attach private data to a transaction using the (ignoreable) extension mechanism
    - Zero-cost and invisible to the rest of the system
    - For example a data-reorder buffer can store a pointer to the original transaction in an extension, for use when the response comes
  - But cascaded components of the same type can not do this
    - They would corrupt each others' data
    - Risky – some bridge might be hidden in a subsystem and my supersystem might wipe its extension
    - Because the extension mechanism is based only on TYPE
    - Is there a way to register multiple extensions of the same type? "instance-based" extensions as well as "type-based" extensions
- **Interrupt modelling and time-warping**
  - May be interesting to provide a standard TLM interrupt interface
    - Is this for TLM-3.0?
  - If so, time-warping could be used to reduce kernel activity (prolong the quantum – see issue about interrupt sequences and quantum period)
  - That is, schedule an interrupt in the future. Initiator can react to it at the correct (warped) time without waking the SystemC kernel