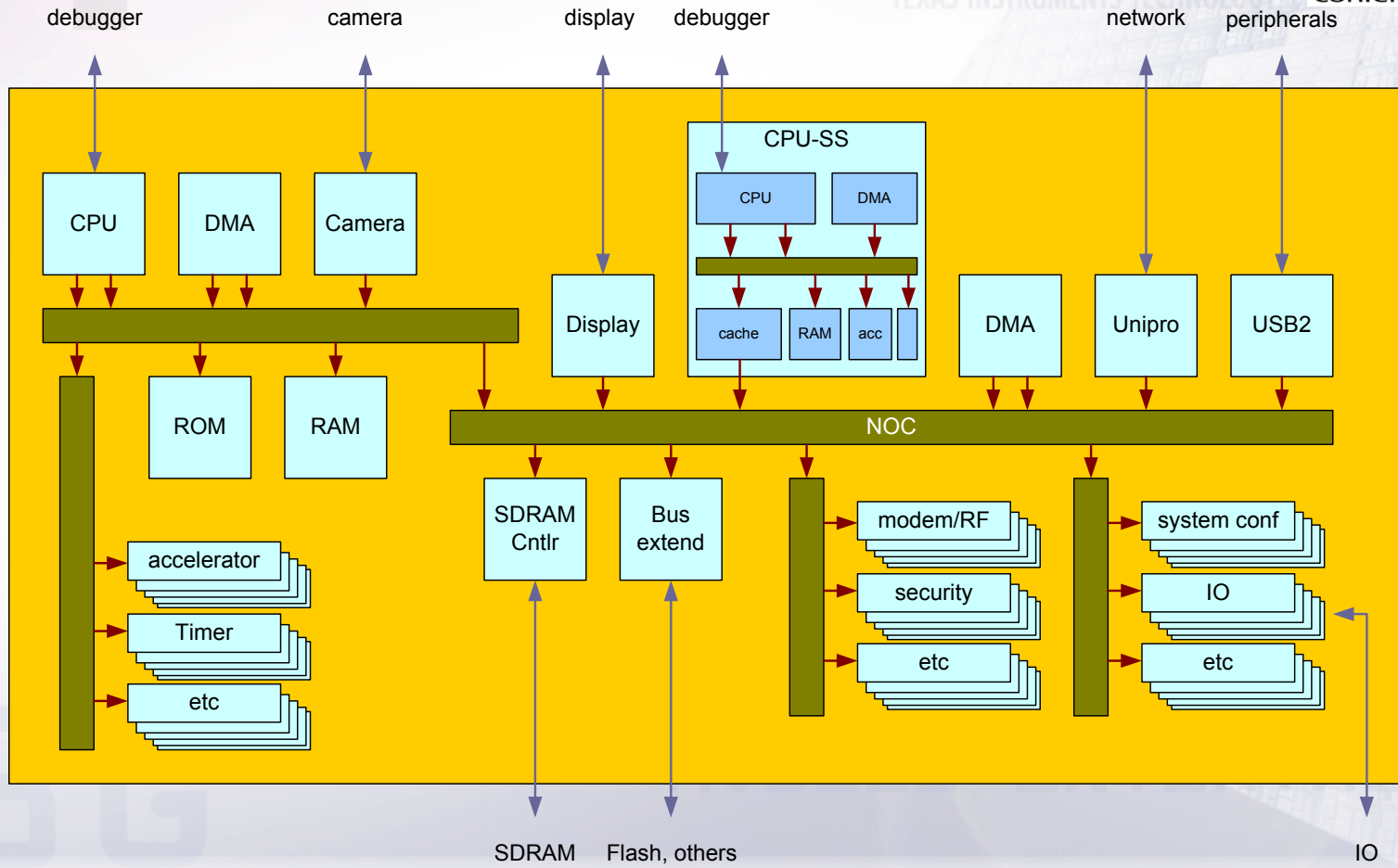


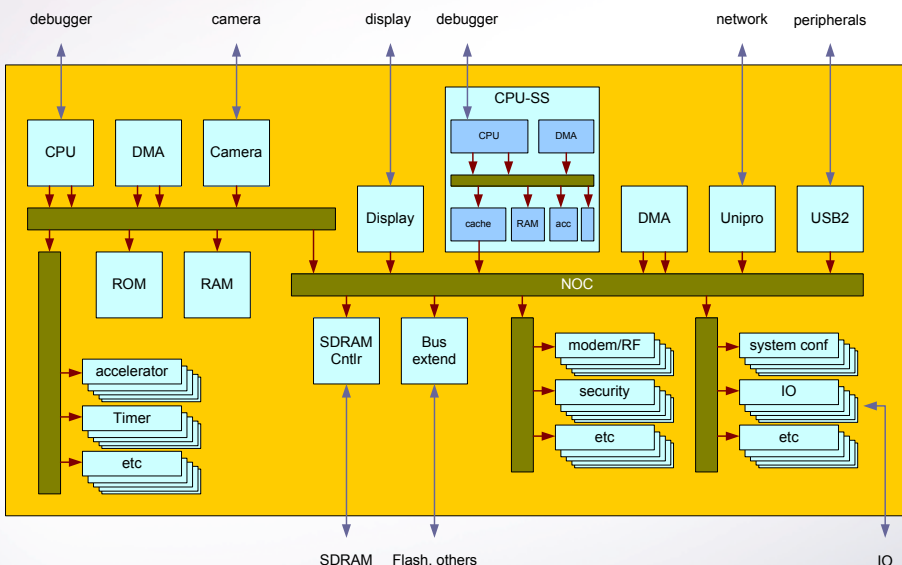
OSCI TLM-2 Technology: A Perspective from Texas Instruments and OCP-IP

James Aldis
DVCon 2008

Imaginary Example System-on-Chip



We're going to develop that product: Where does SystemC come in?



- Architecture Model
 - Earlier than HW architecture
 - Validate Performance and Power Consumption
 - Very precise but non-functional
- Software Development Model
 - Earlier than hardware
 - Better debug visibility
 - Cheaper and easier to distribute
- Support Customer Modelling
 - SOC model is just a component in a larger virtual system

Importance of Standards

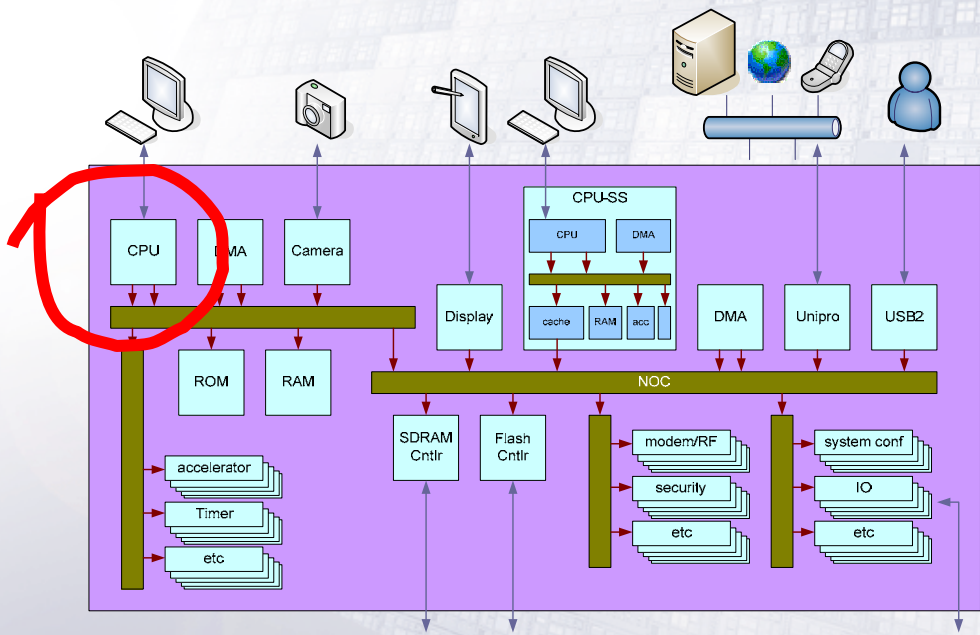


- The SoC contains >100 different components
- Many are 3rd-Party IP
- Only the supplier has the expertise to make good models
 - Even true within the corporation
- But integration of the models must be quick and easy because there are so many
- Ideally integration is automated
 - Sharing design data with RTL integration

Component Type Examples



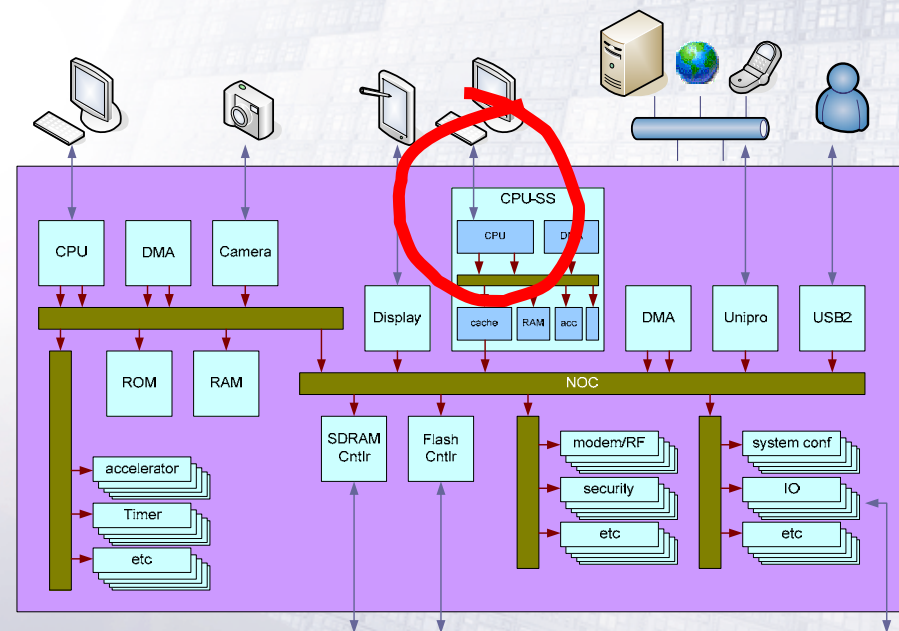
- TI CPU
 - Key strategic IP in TI
 - Key component in VP
 - Invest in optimised high-quality models
 - Bus interface is OCP
 - rich and powerful OCP variant
 - some OCP features not covered by TLM2
 - some TI-private extensions to OCP



Component Type Examples



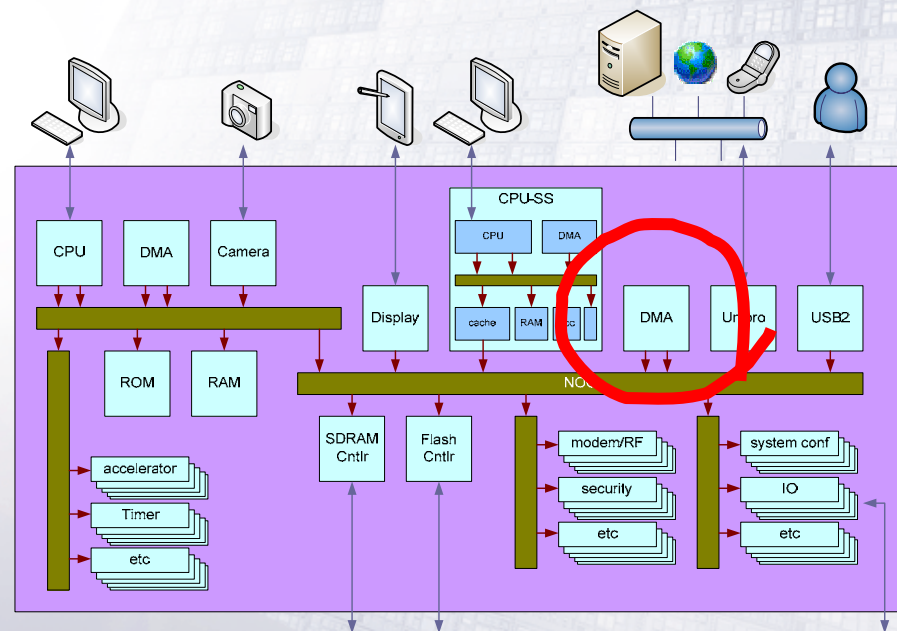
- Purchased CPU
 - Key component in VP
 - Highly complex: require optimised model
 - May be available from supplier or 4th-party
 - Bus interface may not be OCP
 - rich and powerful bus
 - several features not covered by TLM2
 - TI-private extensions can be added *outside the CPU IP itself*



Component Type Examples



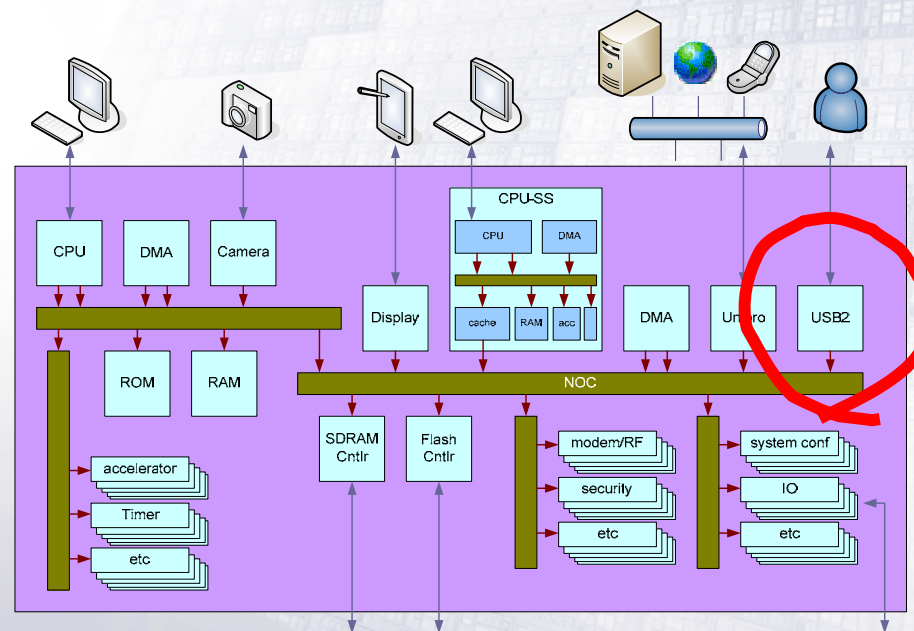
- DMA
 - TI component
 - Less critical model
 - Bus interface is OCP
 - high performance but functionally simple OCP variant
 - all OCP features covered by TLM2 generic payload



Component Type Examples



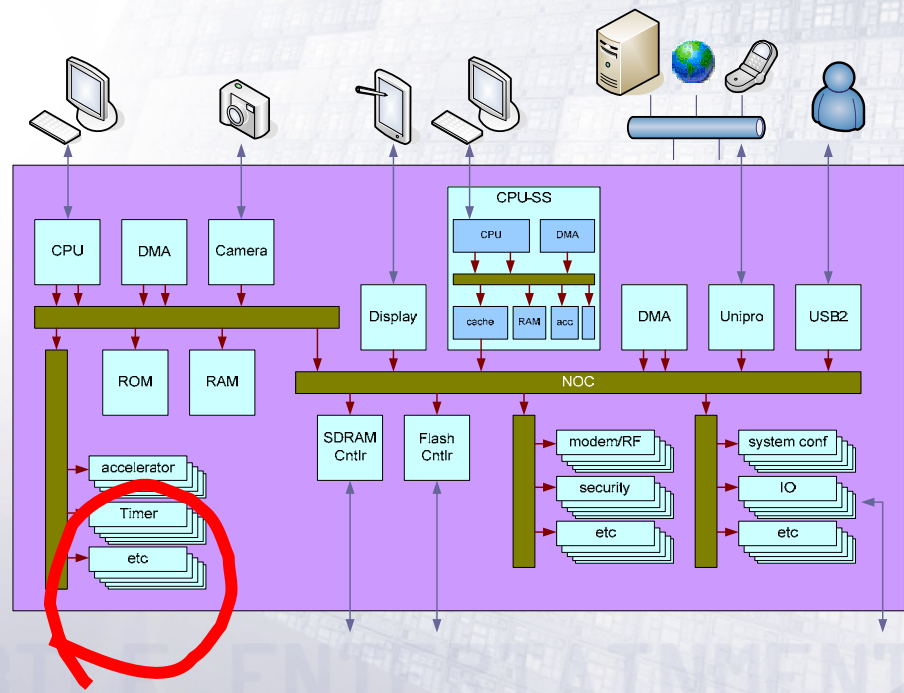
- Communications Controller
 - Could be 3rd-party IP
 - Less critical model
 - Needs to conform to external connectivity standard if there is one
 - Bus interface might/might not be OCP
 - high performance but functionally simple OCP variant
 - all OCP features covered by TLM2 generic payload
 - May have TI-specify extensions added outside purchased IP



Component Type Examples

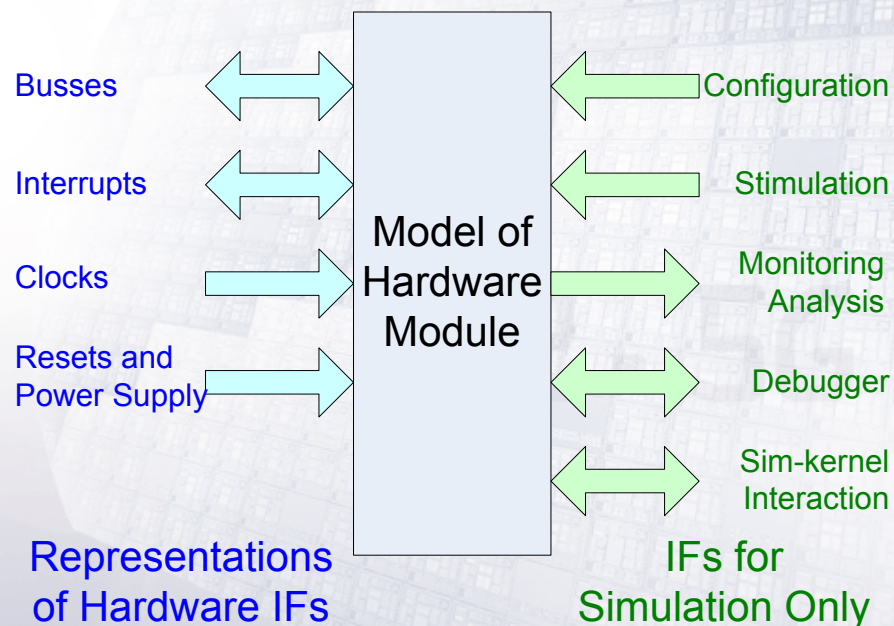


- Peripheral
 - Could be legacy IP
 - Could be created from ESL tool
 - Bus interface is OCP
 - extremely simple OCP variant
 - fully covered by TLM2



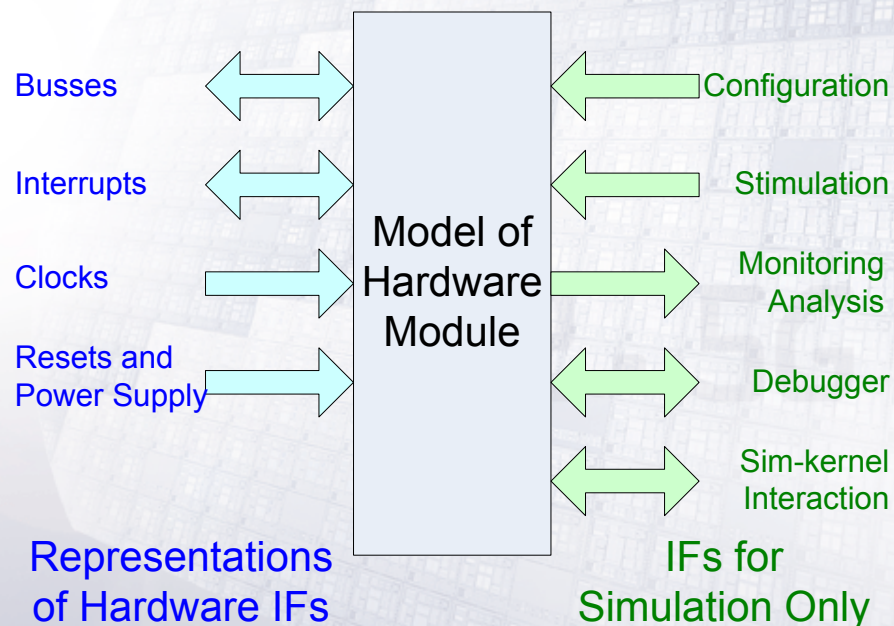
What Interfaces are there on a Component?

- SystemC resolves
 - sim-kernel-interaction
- OSCI-TLM2 resolves
 - memory-mapped busses
- Nothing else so far
- To auto-insert supplier models, we need *all*



What can OSCI provide?

- Generic technology
 - eg the SystemC language
 - eg TLM1
 - eg Configuration, monitoring, etc
- Modelling concepts
 - eg Quantum Keeper
- Interoperability interfaces for generic stuff
 - [abstracted] clocks, resets, interrupts
 - abstracted memory-mapped busses
- Not specific protocols
 - USB, GSM, WiFi, etc..
 - OCP or AXI at high level of accuracy



TLM2 Features: Generic Protocol



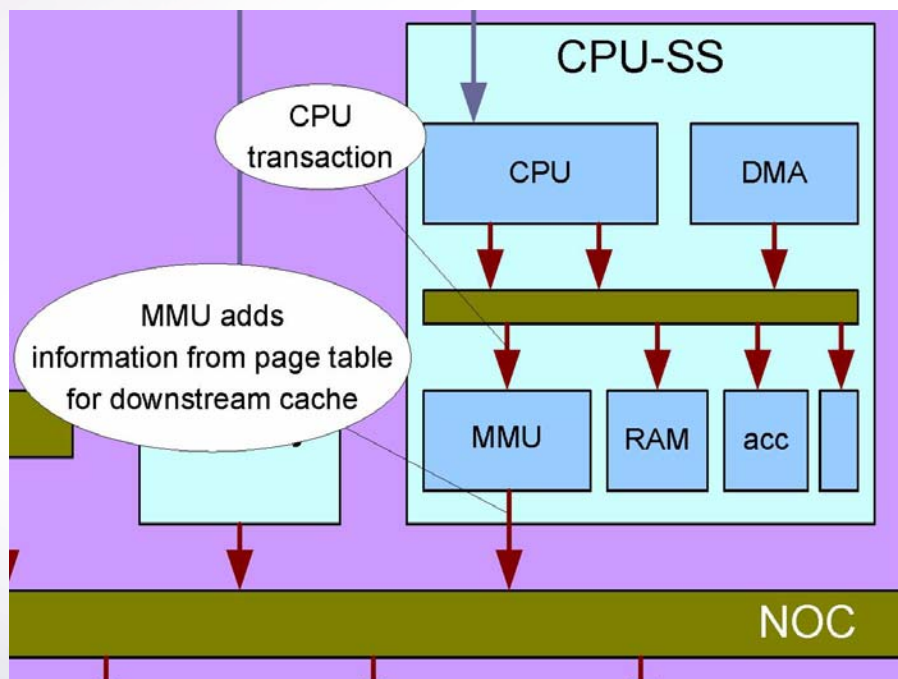
- We abstract away
 - bus width
 - in-burst data order (eg wrapping bursts)
 - codes for commands and responses used in RTL
 - all ordering and pipelining restrictions (eg tagging)
- Leaving
 - read n bytes of data from byte address A
 - write n bytes of data to byte address B
 - byte enables
 - streaming bursts (repetition of command at same address)
 - *More common for master and slave to have compatible busses than in RTL*
 - *Can even have compatible busses where in RTL protocols are different*
 - *eg OCP and AXI, or 2 different OCPs*
- Aim for simulation speed and component compatibility
 - single structure for request and response
 - almost never deep-copy a transaction

TLM2 Features: Extensions



- Some components use ‘exotic’ bus features, eg
 - Locking
 - Lazy synchronisation
 - Extended addressing modes
 - Data cacheability signalling
- Some of these are insufficiently generic for OSCI to be able to define a standard way of modelling them
- All of them are infrequently used
 - would be inappropriate in the generic payload
 - preferable to have many small extensions compared with few large ones
 - better chance of partial compatibility which improves efficiency
 - less to set up in most cases when only a few are needed
- On the SoC we have very many different combinations of extensions
 - 3rd-party and home-made IPs may have different extensions functionally equivalent
 - IPs have different needs for bus extensions (CPUs, DMAs, complex peripherals, secure peripherals, simple memories, ...)
 - Many conversions and additions are required

TLM2 Features: Extensions

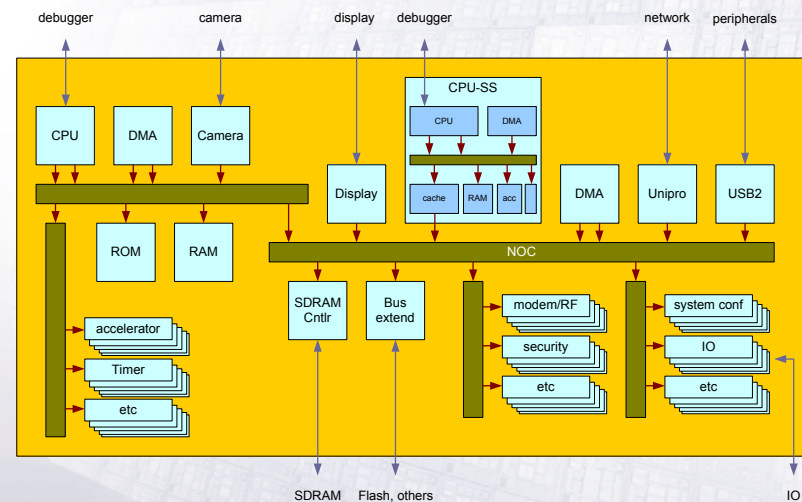


- Extensions can be added outside the IP without a deep copy of the transaction
- We can choose whether:
 - an extension is compulsory
 - linker detects whether target and initiator agree on the extensions that are present
 - initiator always provides the extension
 - eg locking, which causes data corruption if ignored
 - or optional
 - target can ignore it
 - it may not be present – target must test before reading
 - needs to have a sensible default value
 - eg rd-linked/wr-conditional, which just always fails if ignored by target
 - Zero-cost bridge between compulsory and optional is possible

TLM2 Features: DMI and Quantum Keeping



- With the initiator models being a mixture of
 - legacy
 - auto-generated
 - bought
 - home-made
- We risk major degradation of simulation efficiency
- TLM2 quantum keeper is a lightweight standard for coding style
- DMI permits this style of initiator to avoid pushing every transaction across all the NOCs
- But DMI needs extensions (see above)

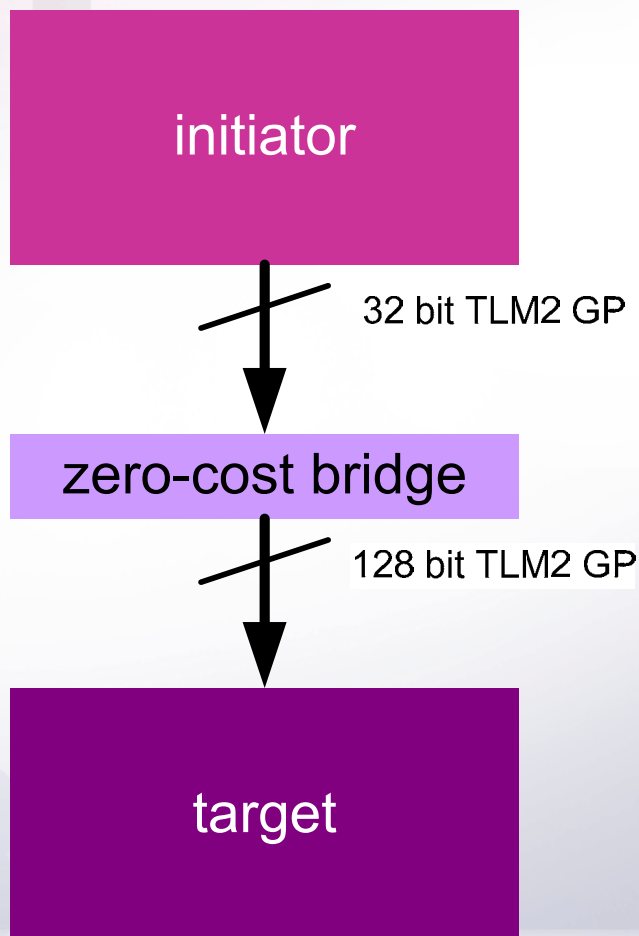


TLM2 Features: Endianness



- Data width is like a compulsory extension
 - therefore zero-cost bridge between data widths is possible
 - but only if the actual width converter in hardware has the same endianness as the simulation host
- The SoC has big and little-endian components
- Data in memory can be big or little-endian and shared between initiators
- Endian conversion can be done by software or by hardware
- All this has to be modelled correctly
 - Meaning: if the hardware gets it “wrong”, then the VP must get it “wrong” identically, otherwise we can’t test the software that fixes it
 - This means that TLM2 can not ignore endianness
 - Which is a great pity
- *Endianness can not be abstracted away*

TLM2 Features: Width Conversion

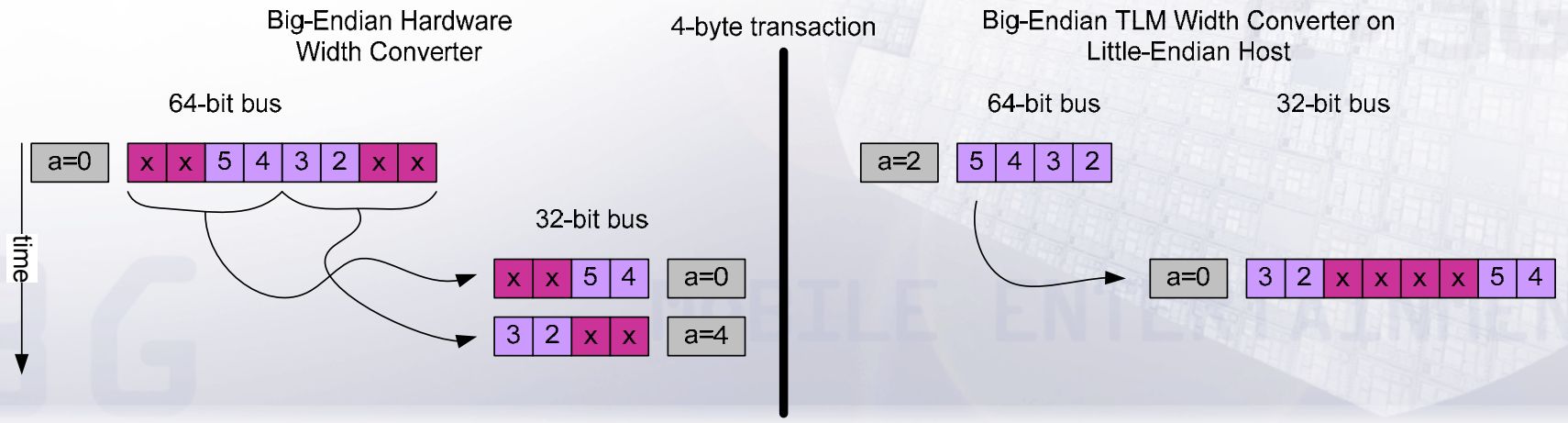


- Every TLM2 GP socket has a width
- Only equal widths can be bound
- User is forced to check when there's a mismatch
- In hardware, there are width conversion gates
- If the hardware width conversion is 'host-endian' we don't need to change the TLM transaction at all



TLM2 Features: Endianness

- So what happens if the component is not host-endian?
- Some coding effort and simulation performance penalty
- Basically, the data array in the GP is a representation of a sequence of states of the hardware data bus
- Exact mapping determined by address alignment and host endianness but *not* by TLM socket width
- To model a big-endian CPU on a little-endian host efficiently, use *arithmetic mode*
 - local storage in the CPU model is host-endian for easy arithmetic
 - bus accesses do not need byte-level data reordering; only address modification



OCP Background

- OCP is a memory-mapped bus protocol
 - For use within ICs
 - Motivation behind OCP is to improve the reusability of digital IC components through
 - protocol standardisation
 - enabling automation
- OCP is owned by the OCP-IP, which
 - publishes the specification
 - provides access to a very rich ecosystem of tools associated with the protocol
 - OCP-IP provides some SystemC interfaces and code, rather like OSCI-TLM



OCP-IP's SystemC Code

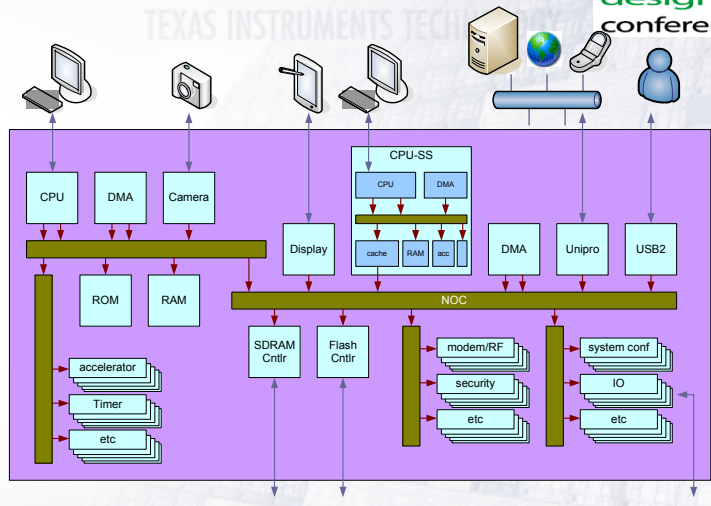
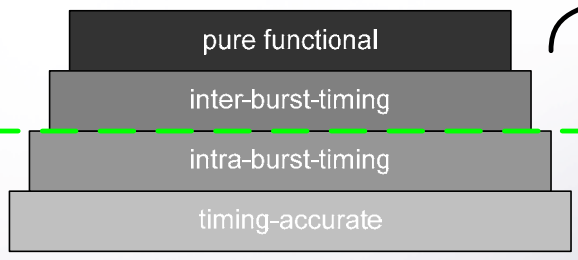
- Has existed for several years
 - `sc_interface/sc_port/sc_channel` for modelling OCP interfaces
- Three abstraction levels
 - TL1: can be fully cycle accurate
 - TL2: timing approximate but full implementation of OCP
 - TL3: more generic, yet more approximate
- Compare that with OSCI TLM2
 - untimed (blocking interface)
 - loosely-timed
 - approximately-timed
- OSCI is doing more approximate; OCP-IP is doing more accurate
 - Only one point of overlap: TL3 ~ approximately-timed



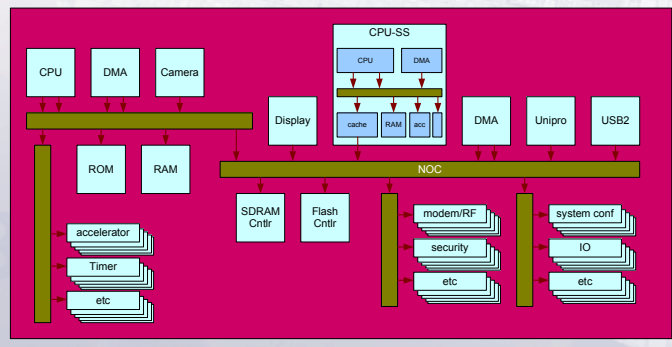
Layered Bus Modelling Protocol



virtual platform: OSCI-TLM2



architecture: OCP-IP



Layered Bus Modelling Technology



pure functional

- generic purely functional payload
- pure-functional extensions
- a very few OCP-specific extensions needed
- almost everything is compatible

inter-burst-timing

- simple 4-point timing protocol
- extensions for timing-specific extras
 - eg transaction tagging, wrapping bursts
- a very few OCP-specific extensions needed
- almost everything is compatible but timing correctness not guaranteed

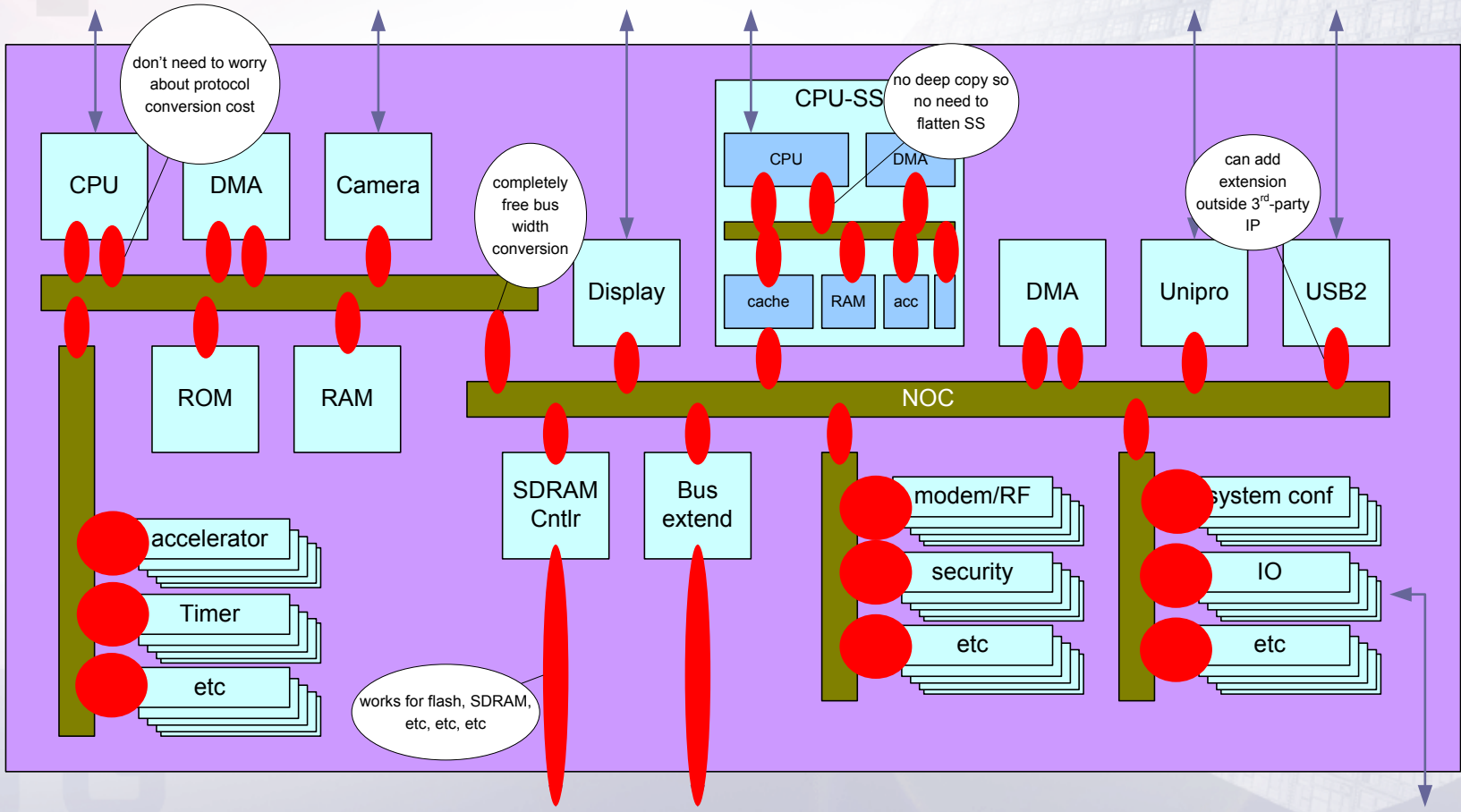
intra-burst-timing

- OCP-specific timing points and richer transaction timing protocol

timing-accurate

- generic technology for modelling cycle-synchronisation, combinatorial dependencies, determinism
- OCP-specific timing points and richest transaction timing protocol

Where Can I use TLM2?



Conclusions

- OSCI TLM2 provides interoperability for virtual-platform models of SoC components
- It meets the needs of highly complex SoCs with
 - multiple component sources
 - hierarchical and cascaded busses/networks
 - mixed bus protocols
- While remaining efficient and generic
- The majority of commercial IP will be able to use it without any extension or modification
- Not yet ready for higher levels of timing accuracy
- No libraries of standard extensions available
- Entirely in line with the wishes of bus protocol suppliers eg OCP

