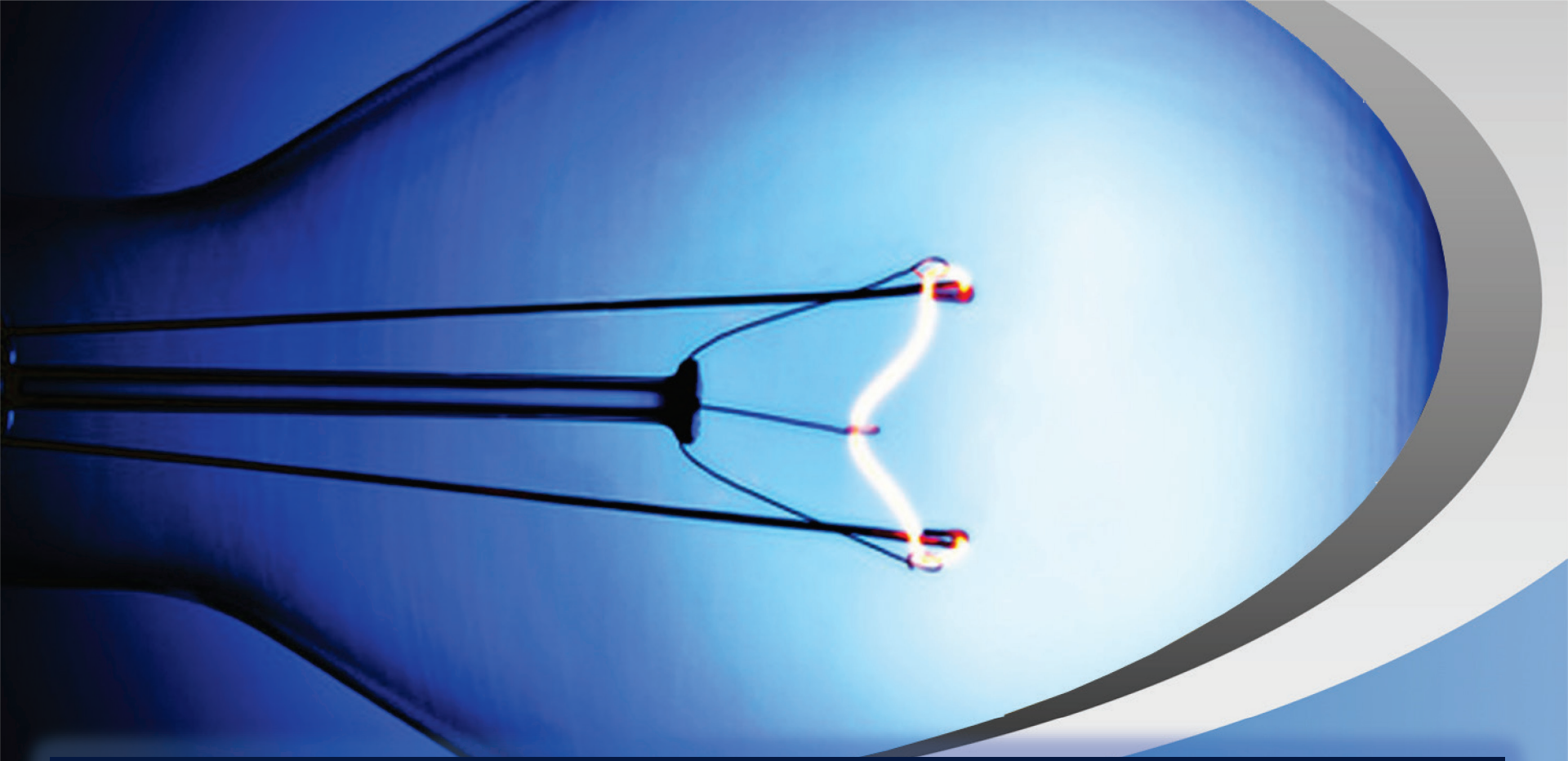


OCP: The Journey Continues

Introduction to the OCP Protocol

Drew Wingard, Sonics, Inc.

accellera
SYSTEMS INITIATIVE

Agenda

- **Welcome and Intro to the OCP protocol**
  Drew Wingard, Sonics
- Verification support for OCP
  Steve McMaster, Synopsys
- TLM 2.0 SystemC support for OCP
  Herve Alexanian, Sonics
- IP-XACT support for OCP
  Prashant Karandikar, Texas Instruments
- OCP futures and closing
  Drew Wingard, Sonics

A Quick Trip Down Memory Lane

- Since early 1997, Sonics has been a semiconductor IP supplier focused on selling interconnect networks for SoC applications
- Since Sonics' products help customers integrate IP from lots of sources (including customers!), we've always had a strong focus on IP core interfaces
- This leads us directly to the story of OCP…

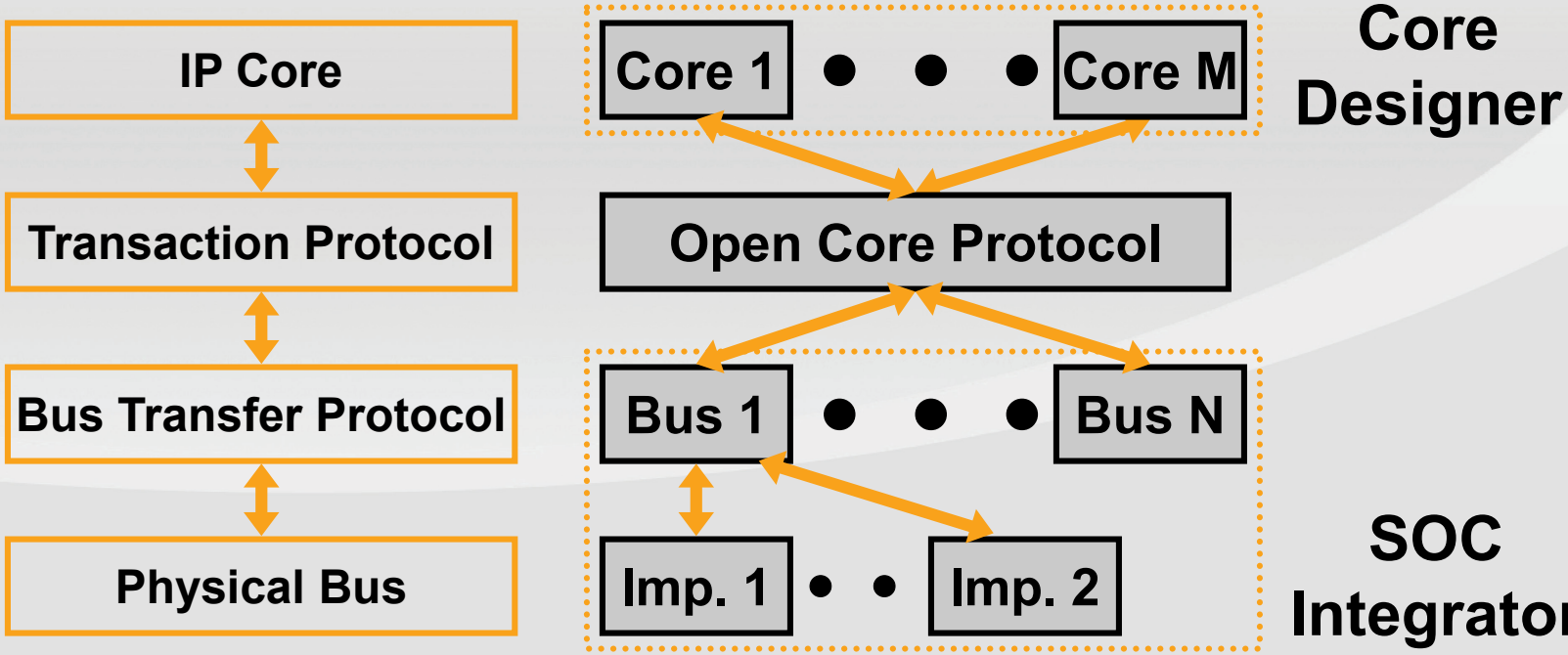
Why is an IP Interface Socket Needed?

- Thousands of cores (*M*)
- 10's (100's?) of interconnects/buses (*N*)
- O (*M x N*) bridges must be designed
- Don't forget signaling protocols, data widths, clock frequency, endian ordering, loading, control flow, etc.

Too much customization for effective reuse

Networking Approach

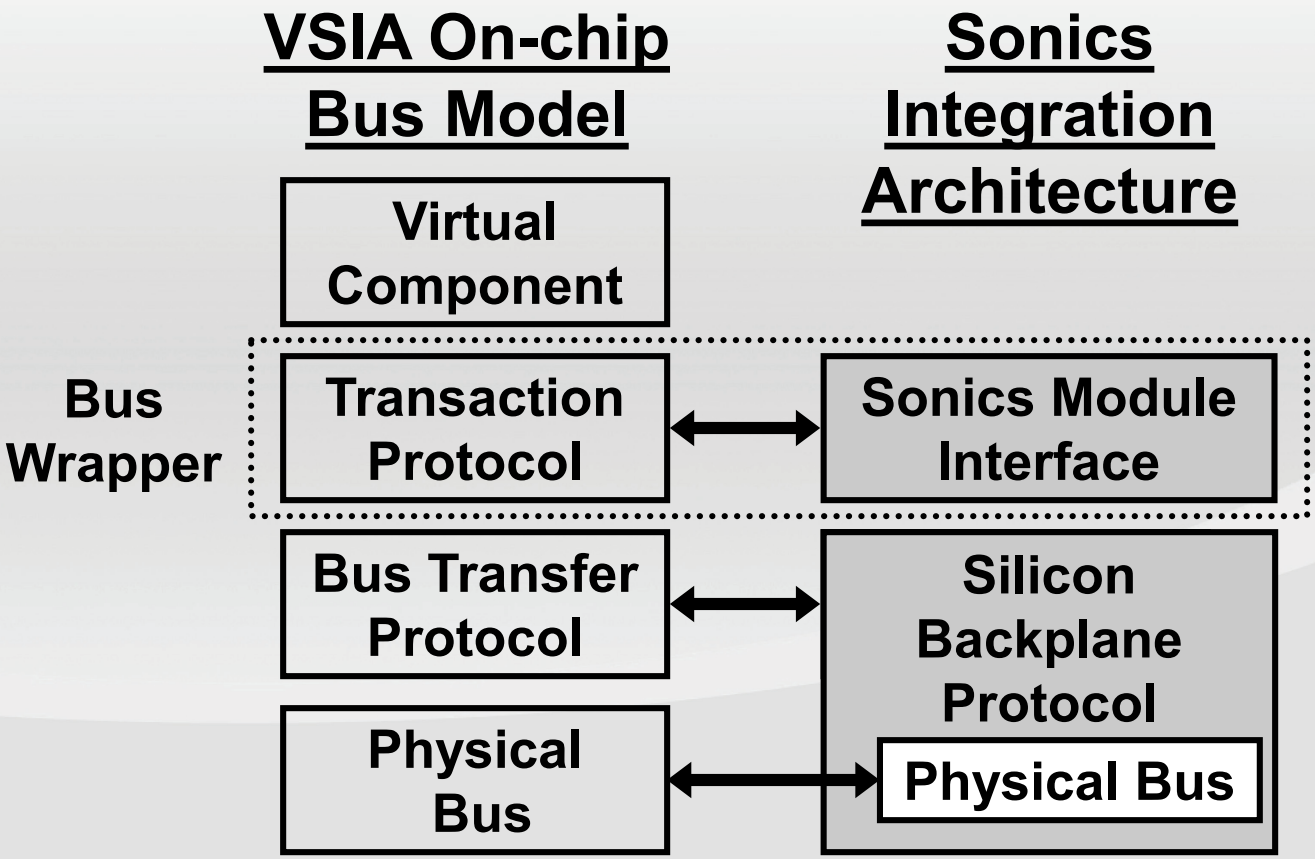
- O(*M x N*) effort becomes O(*M* + *N*) effort
- Reduced work, enhanced reuse



VSIA On-Chip Bus WG Presentation

- **Sonics Module Interface is a Virtual Component Interface specifically designed to:**
  - Isolate VCs from logical and physical bus requirements (i.e., be a bus wrapper)
  - Specify both basic and advanced functionality
    - Minimize area overhead for simple VCs
    - Improve performance for complex VCs
  - Provide structure for user-defined enhancements
  - Allow “black box” verification and testing
  - Interface should be symmetric, so VCs can also connect directly to each other (i.e., *without* an on-chip bus)

VSIA Model and Sonics Integration Architecture

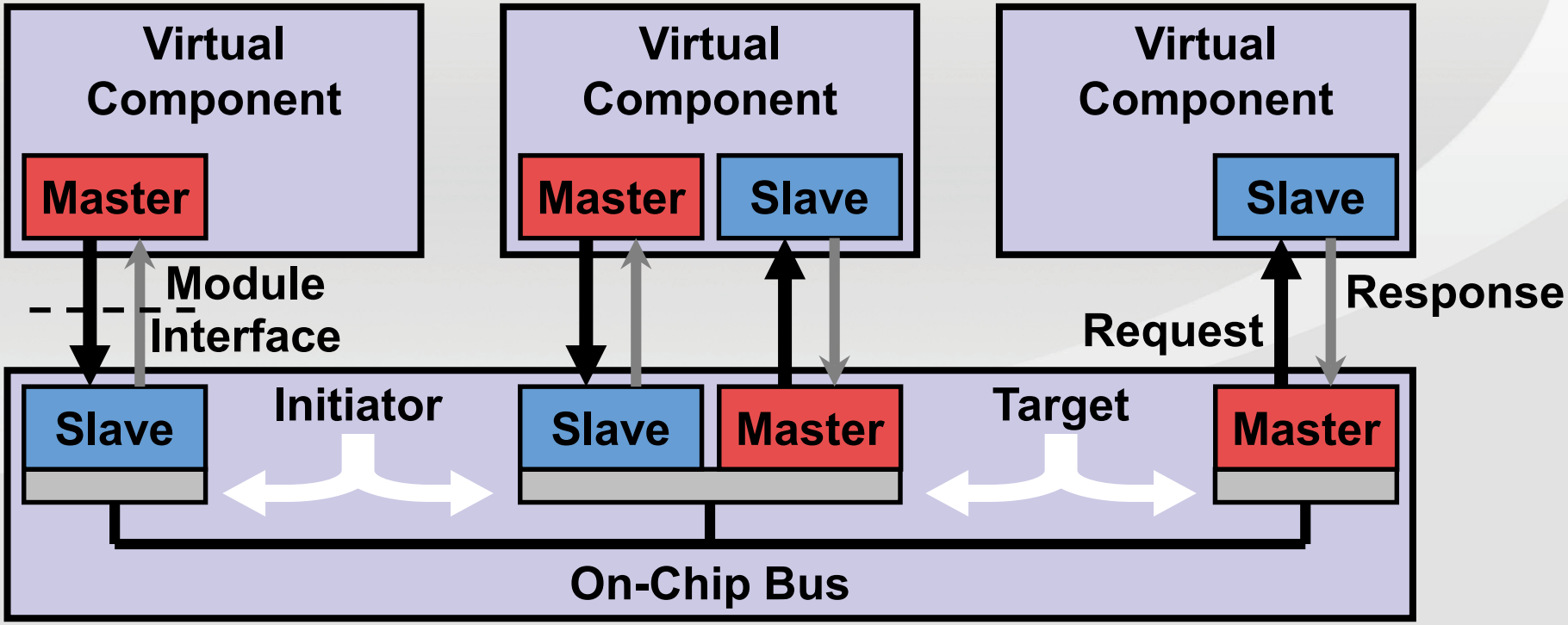


The Sonics Silicon Backplane is a proprietary communication protocol that facilitates connection of VC cores with widely varying performance requirements

Sonics @ VSIA 1997

accellera SYSTEMS INITIATIVE

Sonics Module Interface



Sonics @ VSIA 1997

accellera SYSTEMS INITIATIVE

Additions: Threads

- **A thread is a sequence of transfers that must occur *in-order* with respect to one another**
- **Transfers in different threads may occur out-of-order**
- **Threads can represent:**
  - Separate, independent streams
  - Separate operation types
  - Combinations of the above
- **Thread Identifiers are Layer 2 (Point-to-point)**
- **Additional signals to support threads**
  - Master passes ReqThreadID as tag with request (≤ 4 bits)
  - Slave returns RespThreadID with response
  - Optional ThreadBusy bit vectors for thread status

Non-blocking flow control

Testbench Example

ConnID	ThreadID	Cmd	Addr	(Length)	(Data)
0x1F	0x2	bfill32	0x1000	8	0x12345678
0x1F	0x2	bread32	0x1FFF	8	
0x10	0x1	read8	0x8		
0x10	0x1	write8	0x2008		0xFF

Transaction-based verification

- Perl-based assembler / disassembler
- Behavioral Verilog VC cores
- Protocol checker at interface

Conclusions

- **Wide adoption of any standardized VC interface depends on two technical measures**
  - Area efficiency for simple/low-performance VCs
  - Performance capability for complex/high-performance VCs
- **Sonics Module Interface defines:**
  - Small core of mandatory signals
  - Wide range of optional signals
  - Structure method for extension
  - Logical and electrical protocols
    - Necessary for validation
    - Allows true “black box” VC-based design and testing

Highly configurable

What Happened to Sonics Module Interface?

- **Re-named "Open Core Protocol" in 1999**
- **OCP-IP announced 2001**
  - Original GSC: MIPS, Nokia, Sonics, TI, UMC
  - Reached well over 170 members
  - Assets transferred to Accellera, Oct. 2013
- **Basic OCP protocol is the same as SMI**
  - OCP 2.0 added significant improvements to burst model

OCP-IP Promotion, 2001

THE COMPLETE SOCKET

Important OCP Facts

- Complete and Proven socket
- Essential for "re-use w/o rework"
  - The ONLY path to Plug and Play
- NOT tied to any one supplier
  - Bus/ Design approach, INDEPENDENT

OCP-IP Contributions

- **Open Core Protocol Specification**
  - An IP core interface used on at least 5 billion ICs so far…
  - Substantial work on scalability, optimality and non-ambiguity via explicit configurability
- **Transaction-level Modeling (TLM)**
  - Groundbreaking white paper on TLM modeling (2002)
  - Definition, implementation and distribution of SystemC channel
  - Substantial contributions to OSCI TLM 2.0 standard
- **Metadata Capture**
  - “rtl.conf” metadata to describe interface: part of specification since 1999
  - Substantial contributions to IP-XACT (pending IEEE1685-2014)
- **Also: multi-core debug, NoC benchmarking**

INTRODUCTION TO OCP

OCP Goals

- **Bus Independent**
  - Support point-to-point, bus, cross-bar, etc.
- **Scalable**
  - Choose field widths based upon IP core needs
- **Configurable**
  - Simple cores get simple interfaces
- **Synthesis/Timing Analysis Friendly**
  - Avoid problems at core boundaries
- **Encompass entire core/system interface needs**
  - Data, control, and test flows

OCP Concepts

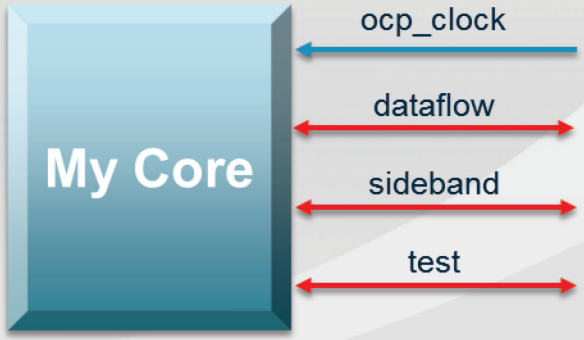
- **Point-to-point, uni-directional, synchronous**
  - Easy physical implementation
- **Master/Slave, request/response**
  - Well-defined, simple roles
- **Extensions**
  - Added functionality to support cores with more complex interface requirements
- **Configurability**
  - Match a core's requirements exactly
  - Tailor design to required features only

Protocol Elements

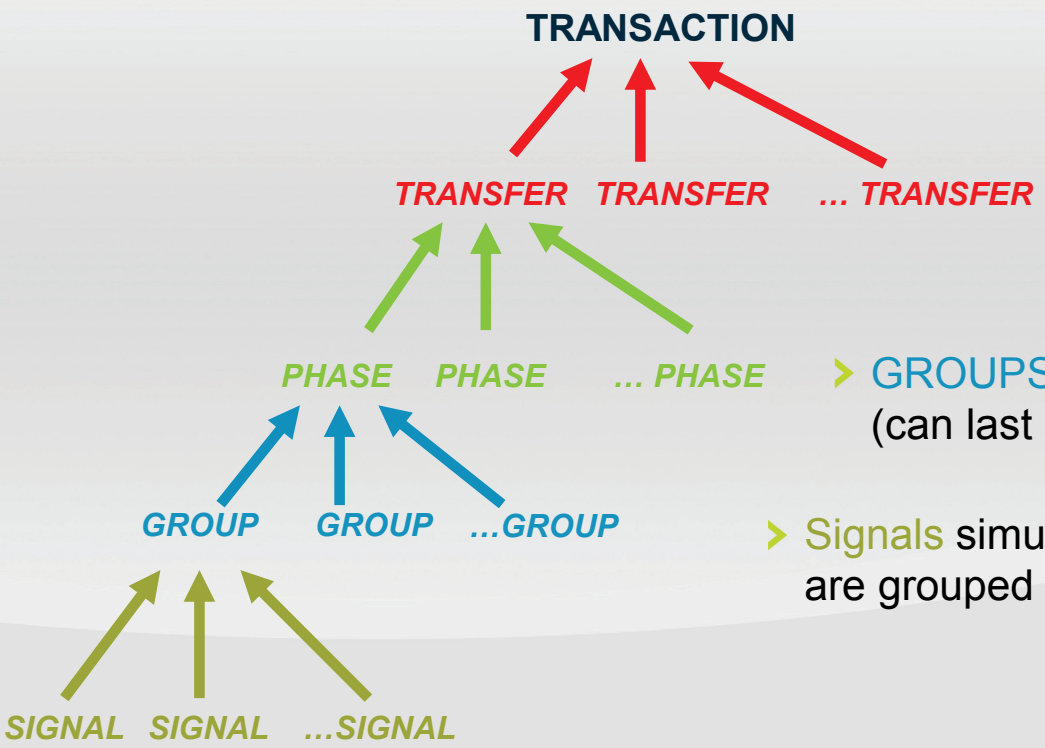
- **A signal specification**
  - Definition of set of roles (masters, slaves) that can be played in attaching to OCP
  - Definition of set of signals (fields) that may be present in the interface, including how each role relates to them
- **A transfer specification**
  - Definition of what types of information transfers (transactions) can be performed
  - Definition of how the signals are to be used for each transaction type
- **A configuration specification**
  - Specification of what combinations of signals are permitted
- **A syntax specification**
  - Specification of file syntax for describing core and its interfaces configurations
  - Specification of file syntax for describing timing of all interface pins

OCP: A Complete Socket

- **3 specs in 1: separate groups of signals**
  - Dataflow
    - Transfer of data with RD or WR
  - Sideband (control and status)
    - Misc. communication among cores
    - Interrupt, Flags, Errors …
  - Test (scan, jtag, etc.)
- **All communication is synchronous to a clock input to the core**



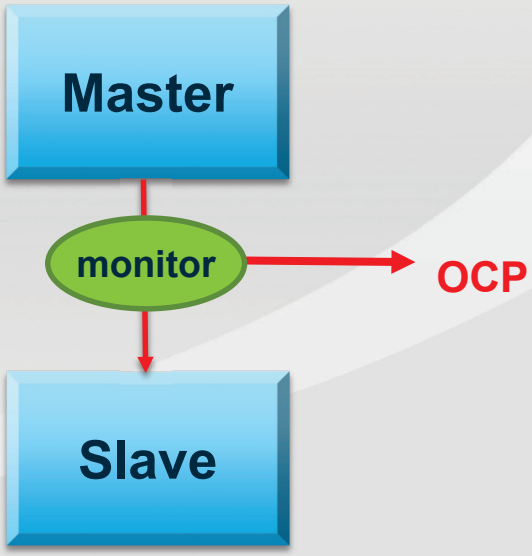
Dataflow Protocol Hierarchy



- A **TRANSACTION** is a group of transfers belonging logically together (like bursts)
- Multiple PHASES define a TRANSFER;
- GROUPS of signals define a PHASE (can last more than 1 cycle)
- Signals simultaneously active (in a cycle) are grouped in signal GROUPS

Point-to-Point Transaction Interface

- **OCP includes a set of dataflow signals that connect entities of complementary roles**
- **Three data dataflow roles:**
  - *Master*
    - Initiates transactions
  - *Slave*
    - Services transactions
  - *Monitor*
    - Watches transactions
- **By convention, when OCP bundle is shown (rather than it wires), an arrow shows direction of transfer**
  - *Signal prefix **M** = master outputs*
  - *Signal prefix **S** = slave outputs*

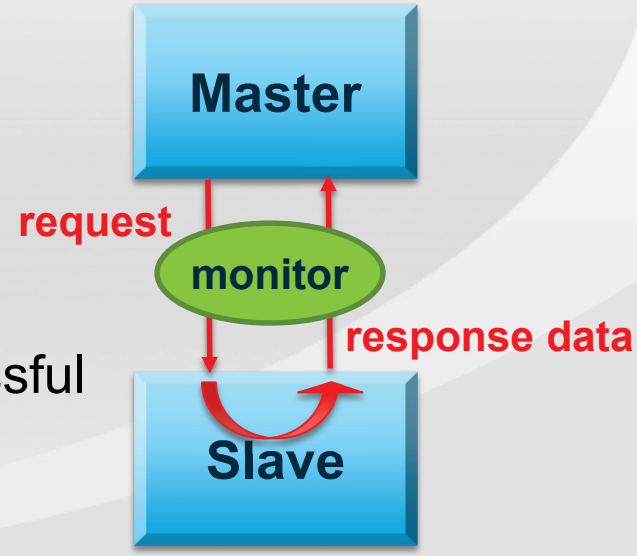


OCP Transaction Types

- **One or more *transfers* build a *transaction***
- **Bursts**
  - Reads
  - Writes
    - Posted (with or without response)
    - Non-posted
  - Broadcast
- **Semaphores**
  - Read-modify-write
    - Blocking
    - Non-blocking
- **Configuration specifies explicitly which transactions are allowed**
  - Read only or Write only
  - FIFO-like Write only (push) or Read only (pop) without address

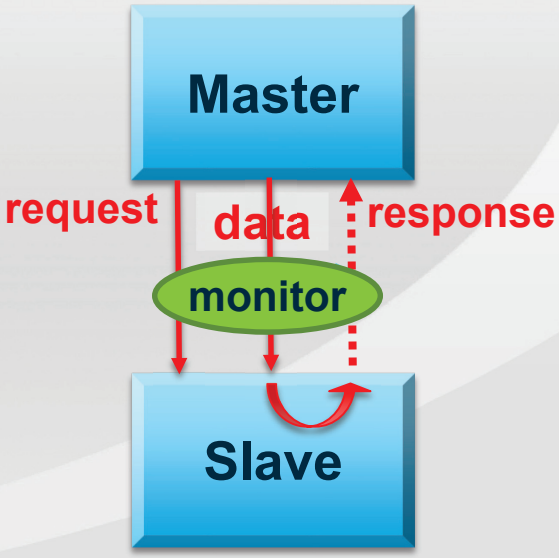
OCP Read Transfer

- **Master sends a request to slave**
  - Designates Read command
  - Specifies an address
- **Slave returns a response to master**
  - Indicates success or failure
  - Includes copy of the addressed data, if successful



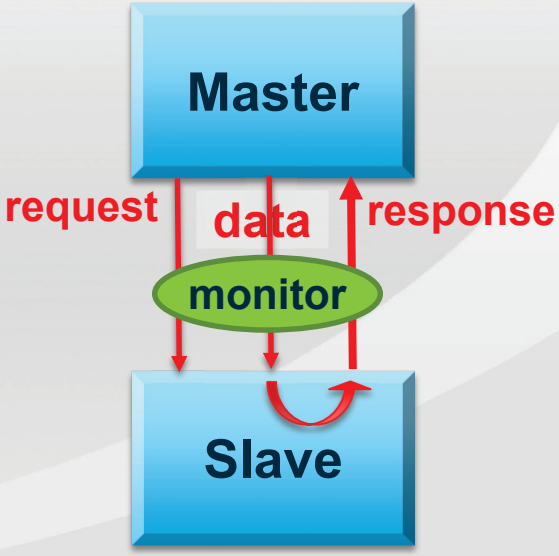
OCP Write (Posted) Transfer

- **Master sends a request to slave**
  - Designates Write command
  - Specifies an address
- **Master sends data to slave**
  - If write response is not needed, the master presumes the write is complete
  - If write response is needed, it indicates write success speculatively
  - Subject to posted write exceptions
- **Slave stores to the addressed location with the provided data**



OCP Non-Posted Write Transfer

- **Master sends a request to slave**
  - Designates Non Posted Write command
  - Specifies an address
- **Master sends data to slave**
- **Slave modifies the addressed location with the provided data**
- **Slave must return a response to master**
  - OCP must be configured for write responses
  - Fully synchronized write (response after write complete)



OCP Burst Transaction

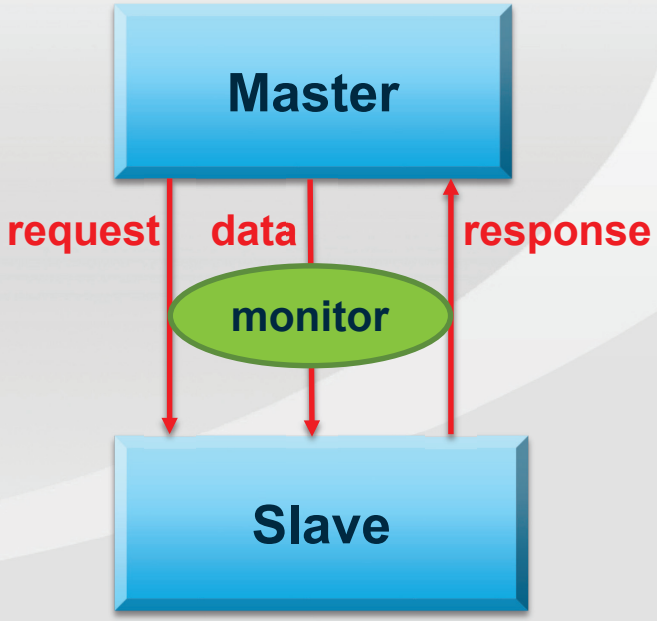
- **All reads, writes and broadcasts can be grouped in bursts**
- **Bursts have a length and an address sequence**
- **Bursts of length 1 (one OCP word) are legal**
  - Partial word transfer also possible
- **3 categories of bursts are supported**
  - SRMD – Always precise
  - MRMD – Precise
  - MRMD – Imprecise
- **Burst sequence types**
  - INCR, WRAP, STRM ….

Split Transactions (i.e., Independent Request/Response)

- **OCP Transfers are split into phases**
  - Request phase
  - Data phase
  - Response phase
- **Transfer Pipelining**
  - Determined by the number of “outstanding” transaction both master and slave can support
  - Each transfer proceeds from phase to phase in a prescribed sequence
  - One transfer doesn’t need to complete all its phases before another transfer can start
  - Permits very high performance (bandwidth) socket interface regardless of system latency
  - OCP spec. doesn’t limit the max number of outstanding transactions

Signal Groups

- **OCP avoids phase interference by supporting each phase with its own independent set of signals (including flow control)**
  - Request phase
  - Data phase signal group
  - Response phase signal group
- **Arrows only show phase initiation direction**
  - Each phase may have both inputs and outputs



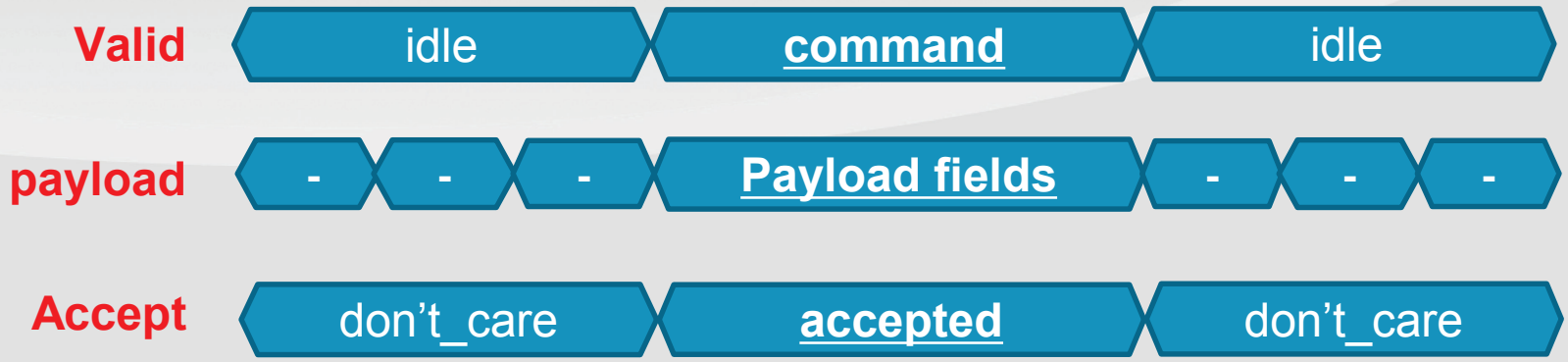
Phase Configurability

Not all phases need to be implemented:

- **Request phase must always be present**
- **Data phase is optional**
  - Present if phase valid signal is present (MDataValid)
  - If phase not present, phase payload (MData) moves with request phase
- **Response phase is optional**
  - Present if phase valid signal is present (SResp)
  - If phase not present, transactions requiring response not supported
    - Any read type commands
    - Non-posted writes & writes with responses

Signaling Protocol

- **When does a phase start and how does it end?**
  - Each OCP signal group (phase) has one signal field indicating whether the phase is valid or not
  - All other fields belonging to a phase including the accept signal which is a don't_care when the phase is not valid
  - **No retraction**: Once a phase is activated, the signal activating the phase and all other signals belonging to the same group (payload) must be held constant.



The Valid signals: MCmd, MDataValid and SResp

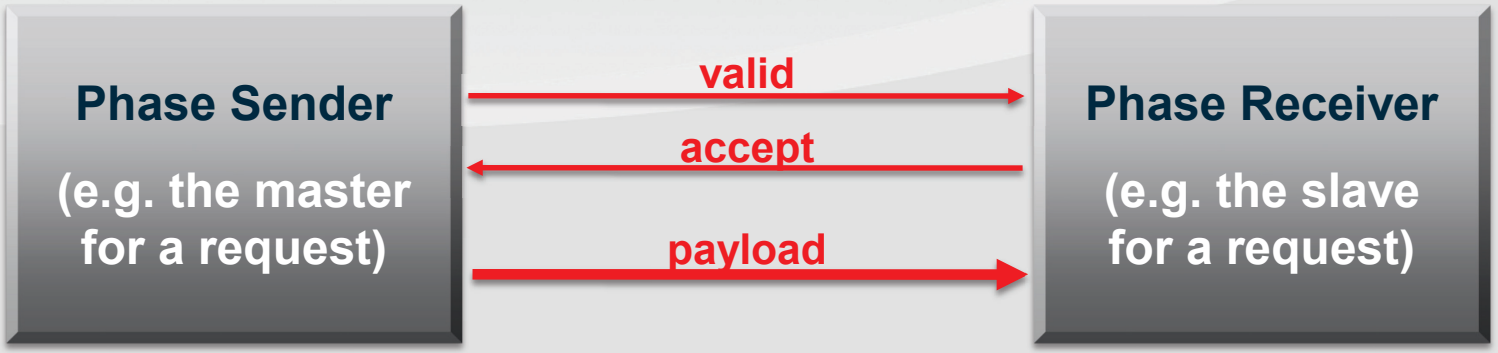
MCmd[2:0]	Command
000	IDLE
001	WRITE
010	READ
011	READEX
100	READ-LINKED
101	WRITE-NON-POST
110	WRITE-CONDITIONAL
111	BROADCAST

SResp[1:0]	Response code
00	NULL
01	DVA
10	FAIL
11	ERROR

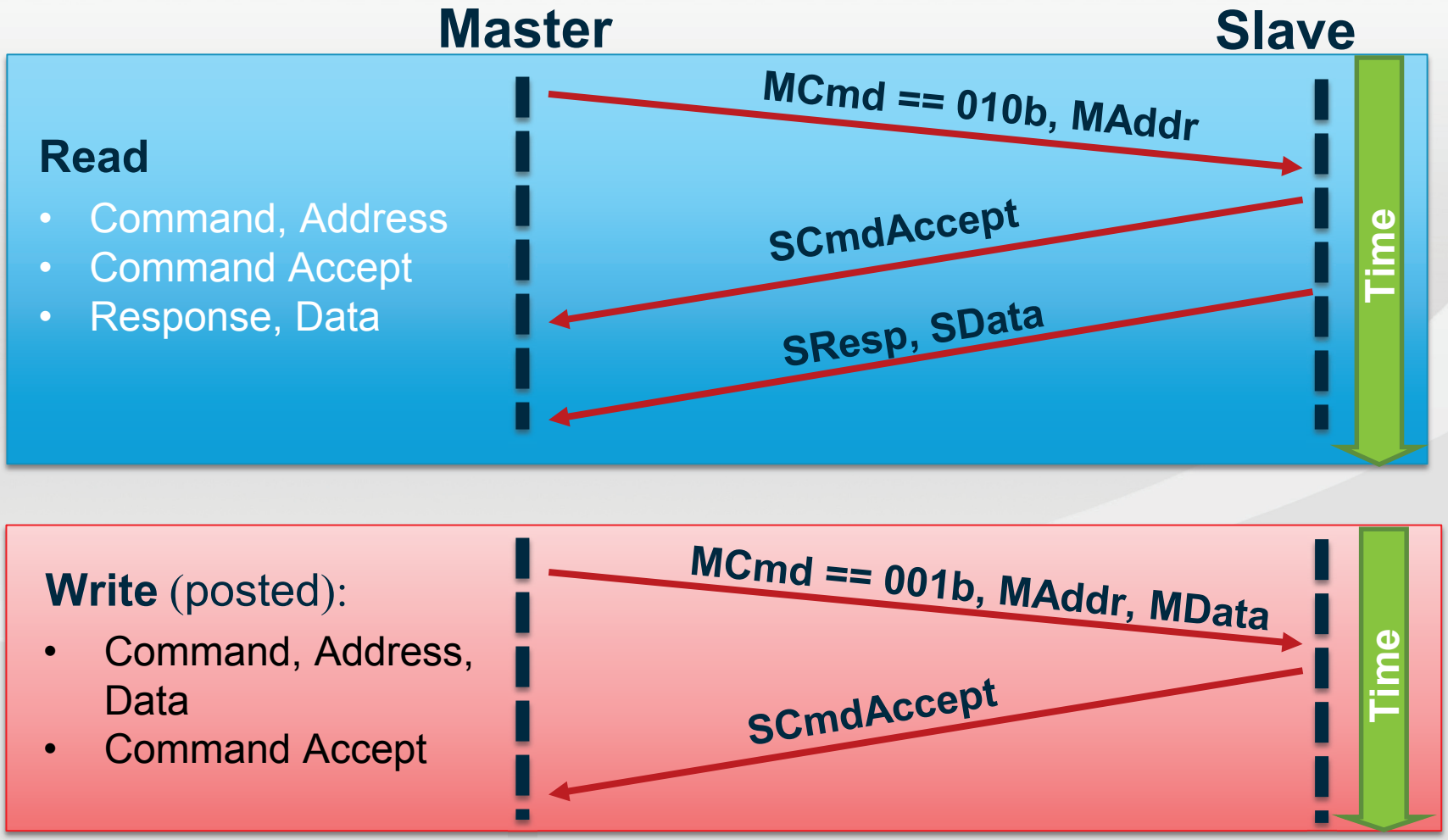
MDataValid	Write Data Phase
0	Write Data not Valid
1	Write Data Valid

The Accept signals: SCmdAccept, SDataAccept and MRespAccept

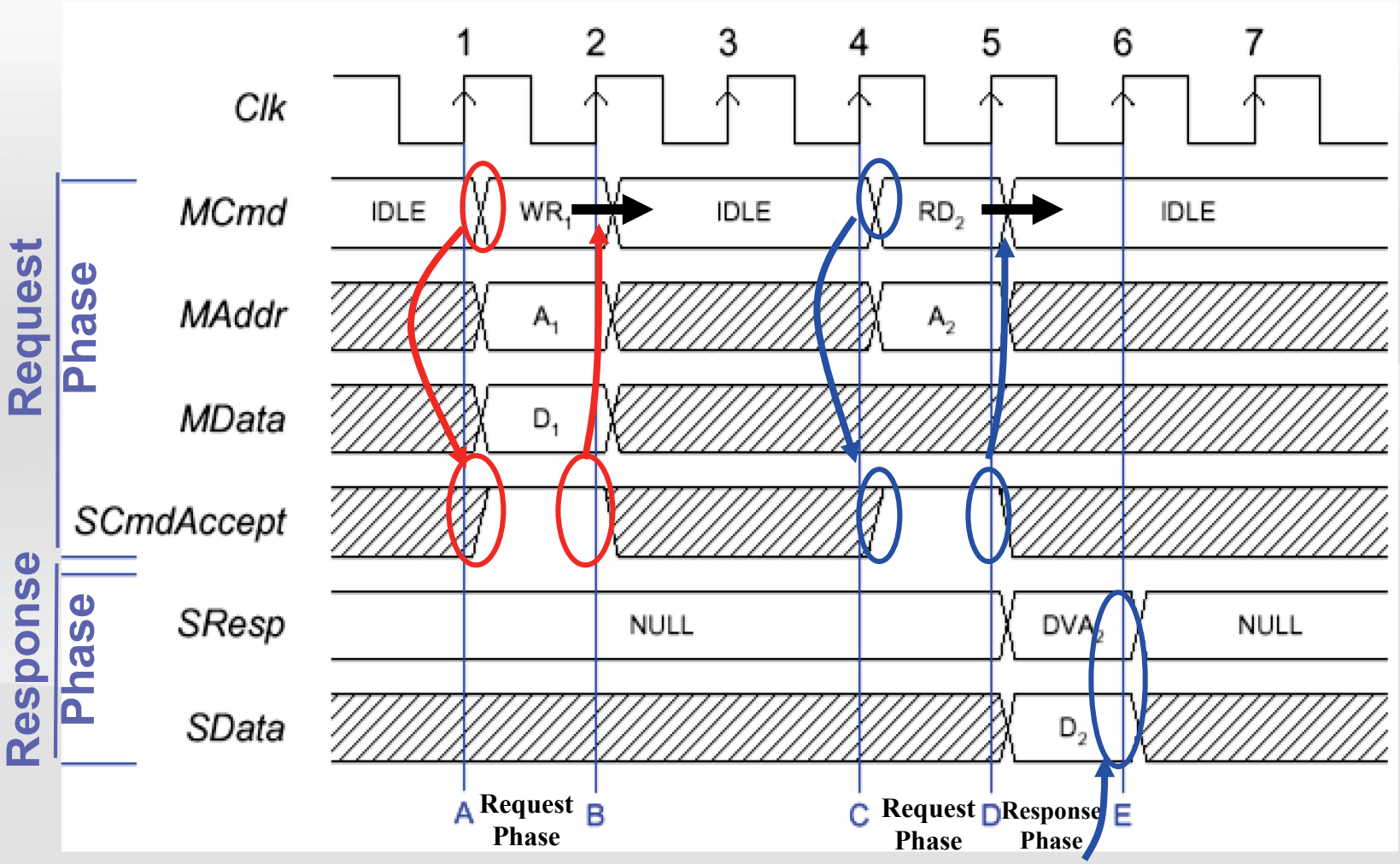
Handshake field	**Request** phase	**Data** phase	**Response** phase
Valid	MCmd	MDataValid	SResp
Accept	SCmdAccept	SDataAccept	MRespAccept
Payload	Request Group Signals	Datahandshake Group Signals	Response Group Signals



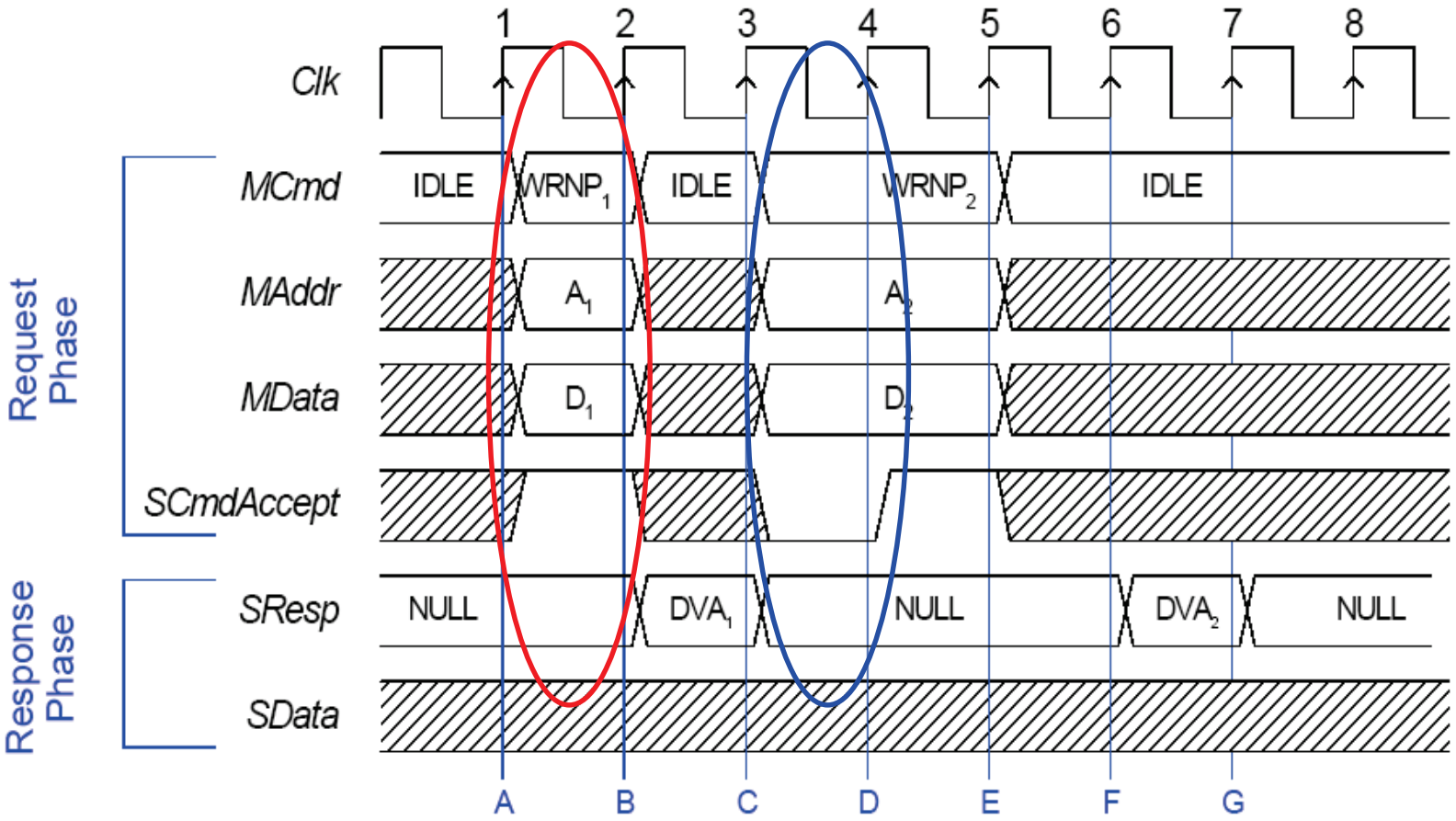
Phases of simple transfers: RD, WR



Simple Write & Read Transfer



OCP *Non-Posted* Write (WRNP)



Address and Data in OCP World

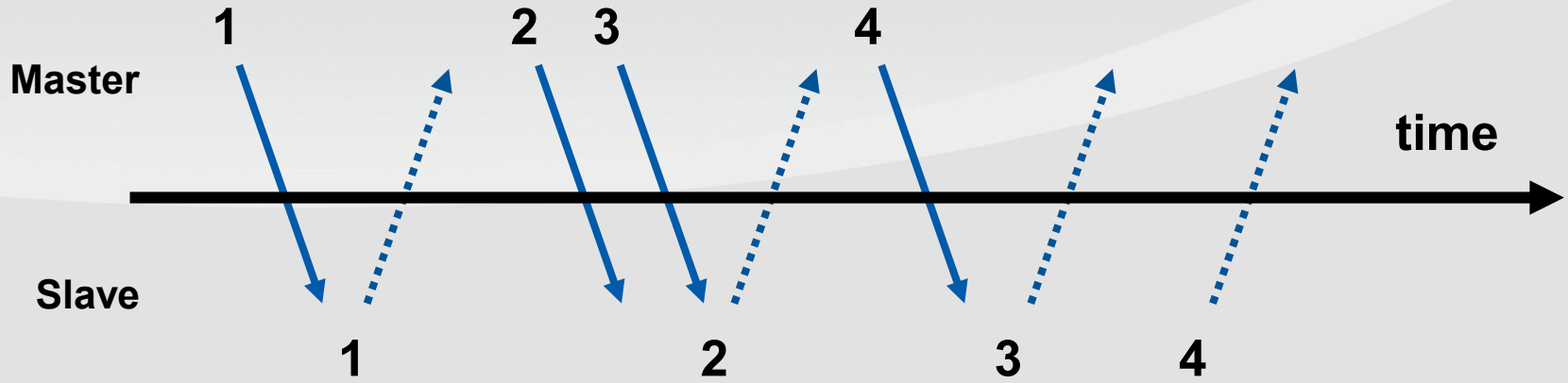
- **Addresses granularity is 8 bits (1 byte)**
- **Parameter data_wdth indicates the number of bits in an OCP word**
- **Word size is the natural transfer unit across the OCP socket**
- **Parameter addr_wdth indicates the number of address bits generated by a core.**
  - OCP addresses are byte addresses aligned to an OCP word
  - So a core can address $\frac{2^{addr\_wdth}}{data\_wdth/8}$ OCP words

Byte Enables

- **The basic OCP data transfer size is a word**
  - *Corresponds to the OCP parameter data_wdth*
- **Use one signal per byte lane for partial access**
  - MByteEn for read or both in MRMD
  - MDataByteEn for write in SRMD
- **Any combination of byte enables is legal (general byte enables)**
  - Example: byte enable = 0x35 on an 8-byte OCP
  - Example: byte enable = 0x0 is legal, too
- **If needed, use force_aligned parameter to restrict the combinations**
  - Aligned to powers-of-two

Ordering and Pipelining

- An OCP transfer is a complete request/response interaction
- Successive OCP transfers are strictly ordered on a thread
- Response return order mirrors the request order
- Successive transfer phases can be pipelined (Multiple requests can be outstanding before the first response returns, assuming the slave accepts the commands)



Without tags / threads, everything has to be processed / returned in order.

OCP Bursts Details

- **Bursts are a set of transfers:**
  - Assembled into a transaction
- **3 categories of bursts are supported**
  - SRMD
    - Single Request Multiple Data (packet) style
    - Precise length without early termination
  - Precise MRMD
    - Precise length without early termination
    - Multiple Request Multiple Data
  - Imprecise MRMD
    - Speculative length
    - Since it's imprecise, it must be MRMD style
    - Possible early termination
      - Network application detecting packet CRC error

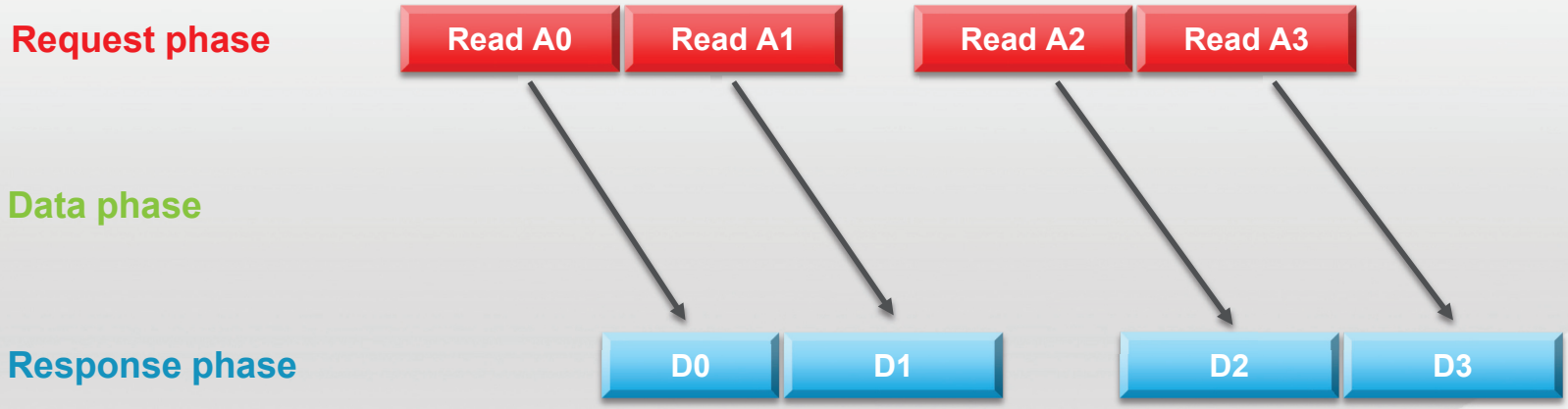
Bursts in OCP

- **MBurstSeq encodes one of the 7 burst address sequences:**
  1. Incrementing bursts for block rd/wr - packing
  2. Custom(packed) - packing
  3. Wrapping - packing
  4. Custom(not packed) - non-packing
  5. Exclusive OR - packing
  6. Streaming - no packing
  7. Unknown - no packing
  8. BLCK - packing
- **MBurstLength encodes the number of transfers in the burst**
  - Cannot change during the burst if burst is precise
  - May only be an hint when burst is imprecise
  - Need 5 bits to support 16 words burst (i.e., 5'b10000)

Bursts in OCP (cont.)

- **Optional framing information**
  - MReqLast, SRespLast, MDataLast indicate the end of a burst
  - If burst is imprecise, last transfer is indicated by MBurstLength=1
  - If burst is precise, use it to terminate a transaction without counting □ save some area
- **If an initiator may change its burst type dynamically, on a burst-by-burst basis…. additional signals are needed**
  - MRMD / SRMD (MBurstSingleReq)
  - Precise / imprecise (MBurstPrecise)
- **If a burst can be chopped by the Interconnect, Atomicity attributes (i.e., MAtomicLength) allow control of the chopping process**

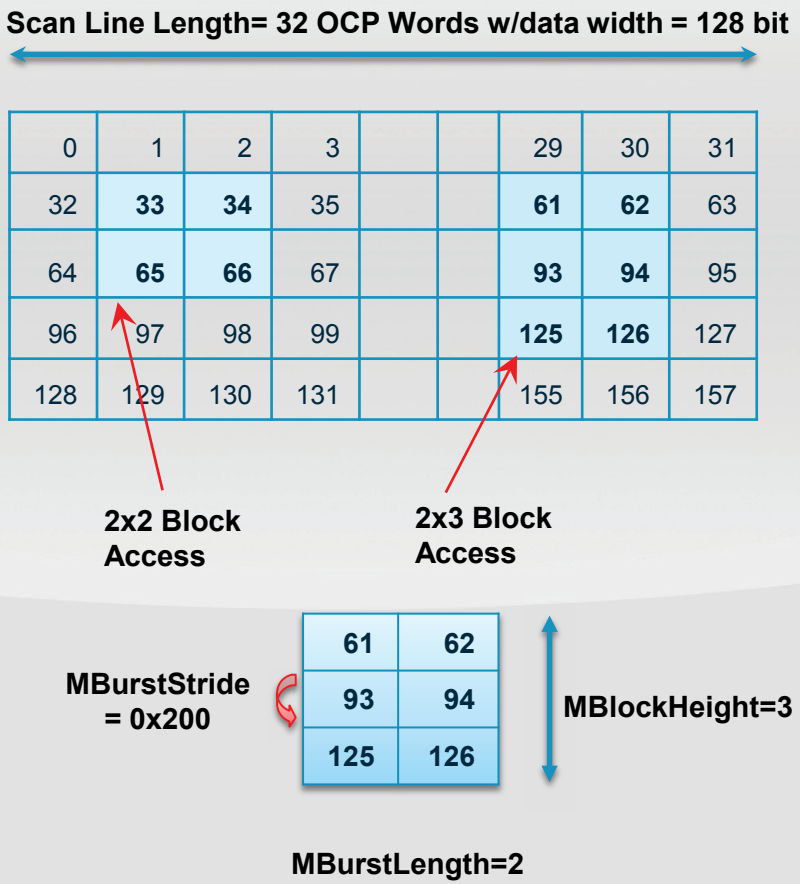
MRMD Read Burst



- Could be imprecise or precise
- Response phase can't begin before Request phase begins
- Response phase can't end before Request phase ends

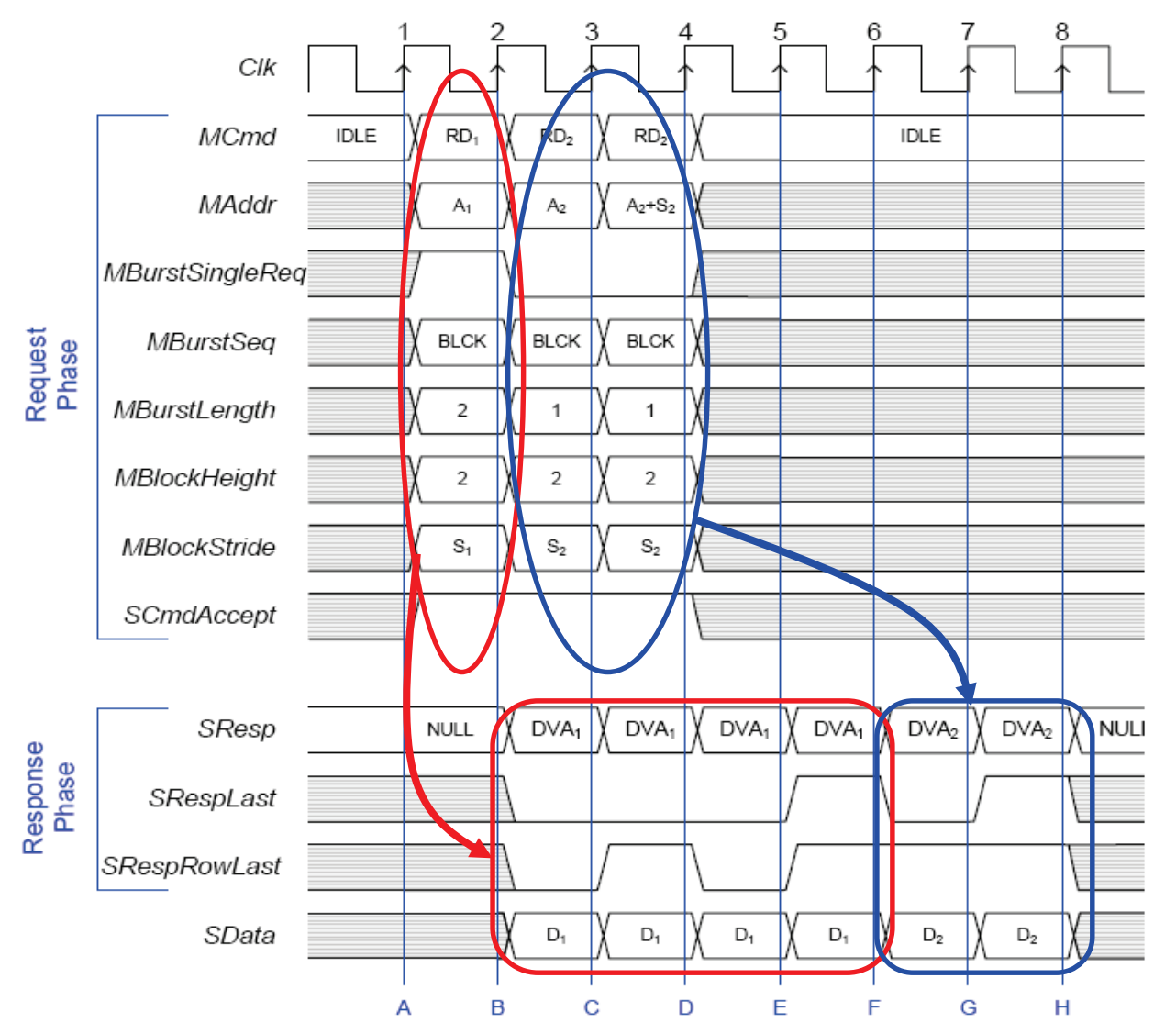
2D Burst Sequences

Multimedia Application Example



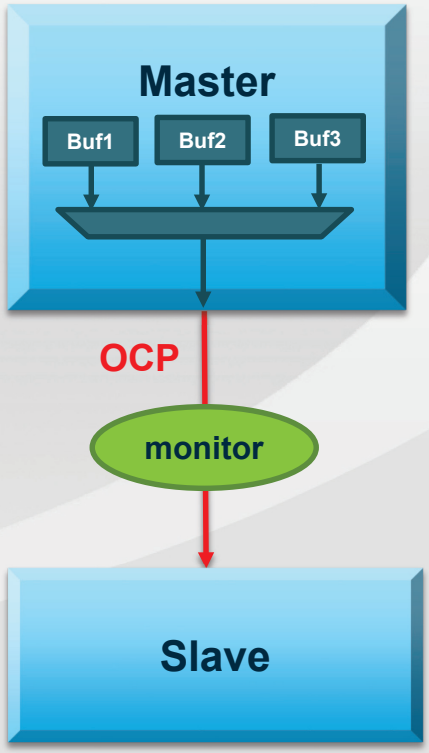
- OCP BLCK burst sequence
- Mostly useful in Digital Media Application to achieve higher DRAM efficiency
- Each transaction specifies burst length, height and stride

2D Burst Sequences

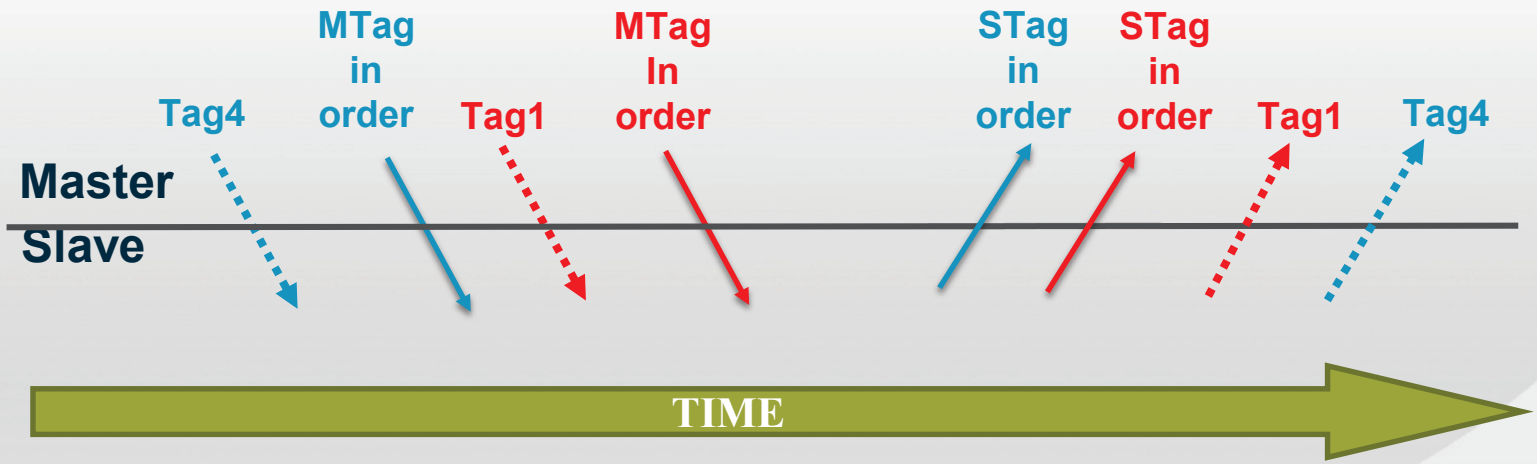


OCP Tags

- OCP Tag is similar to AXI ID
- Allow out-of-order responses within a single thread
- Shared flow control for tags within phase
- “Light weighted Thread”
- Often associated with internal buffer number
- Use MTagInOrder and STagInOrder if orders need to be maintained
- Use tag_interleave_size parameter to control interleaving of responses between different tags within a burst



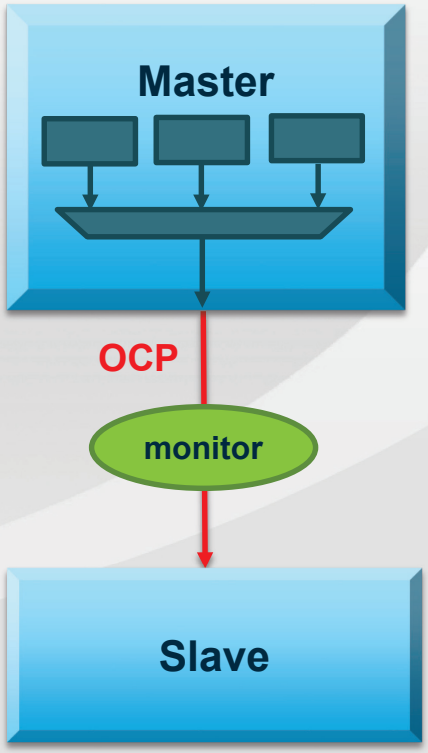
Tags and Ordering



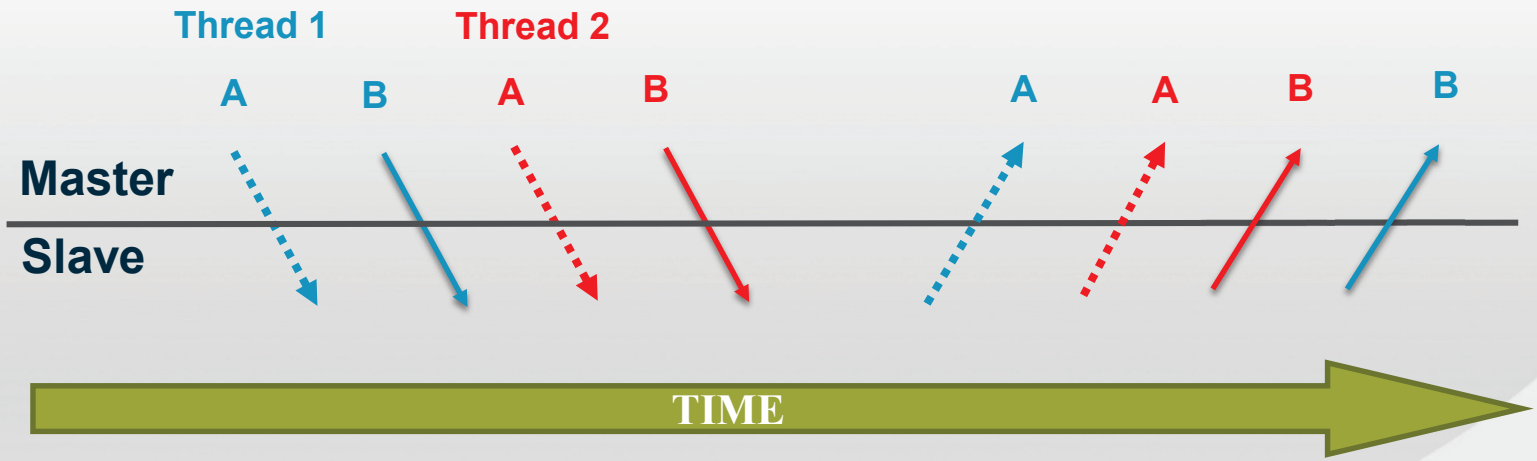
- If MTagInOrder = 1'b1, ordering is required and the xTagIDs are don't_care
- If MTagInOrder = 1'b0, use MTagID and STagID to keep track of each request/response

OCP Threads

- **Master can represent a collection of actual physical or logical masters that share a single set of OCP wires**
  - Bridge from bus
  - Sub-system (DSP with DMA)
- **Threads can be seen as multiple logical OCP connections (virtual channels) using the same physical socket**
- **From system point of view, each thread may represent a QoS level**
- **Transaction execution order is maintained within each thread**
- **Thread interactions can be minimized using *non-blocking* flow control**



Threads and Ordering



- **Strict ordering *within* a thread**

  Thread 1: A, B →1A, 1B

  Thread 2: A, B →2A, 2B

- **No ordering restriction between threads**

  1A, 1B, 2A, 2B →1A, 2A, 2B, 1B

Non-blocking Flow Control

- **A core may be busy on thread X but still able to accept requests for thread Y**
- **In order to prevent blocking, the slave can use the optional SThreadBusy signals (one bit per thread) to indicate which thread(s) are busy.**
  - A Master should not issue requests to any busy thread so the interface will NOT block
  - But it may still choose to do so → Non-blocking with hints
- **Similar arrangement exists in each non-blocking phase**
  - SDataThreadBusy for data phase
  - MThreadBusy for response phase
- **ThreadBusyExact offers the most efficient implementation**
  - The Busy indication is PRECISE
  - Both master and slave MUST obey the rule
  - Each presented request/data/response MUST be accepted
    - The most power efficient

Request, Response and Data Qualifiers (In-band Extensions)

- **MReqInfo: OCP allows requests to be qualified by additional, user-defined, data that will be routed along with the command**
  - Examples: security/protection mode, cacheability, priority
- **SRespInfo: same purpose as MReqInfo, but for response phase**
- **MDataInfo / SDataInfo**
  - Allows 2 kinds of info to be transmitted with the data, per-byte and per-word, typically for ECC/Parity
    - mdatainfo_wdth defines the overall width
    - mdatainfobyte_wdth defines the width for each byte

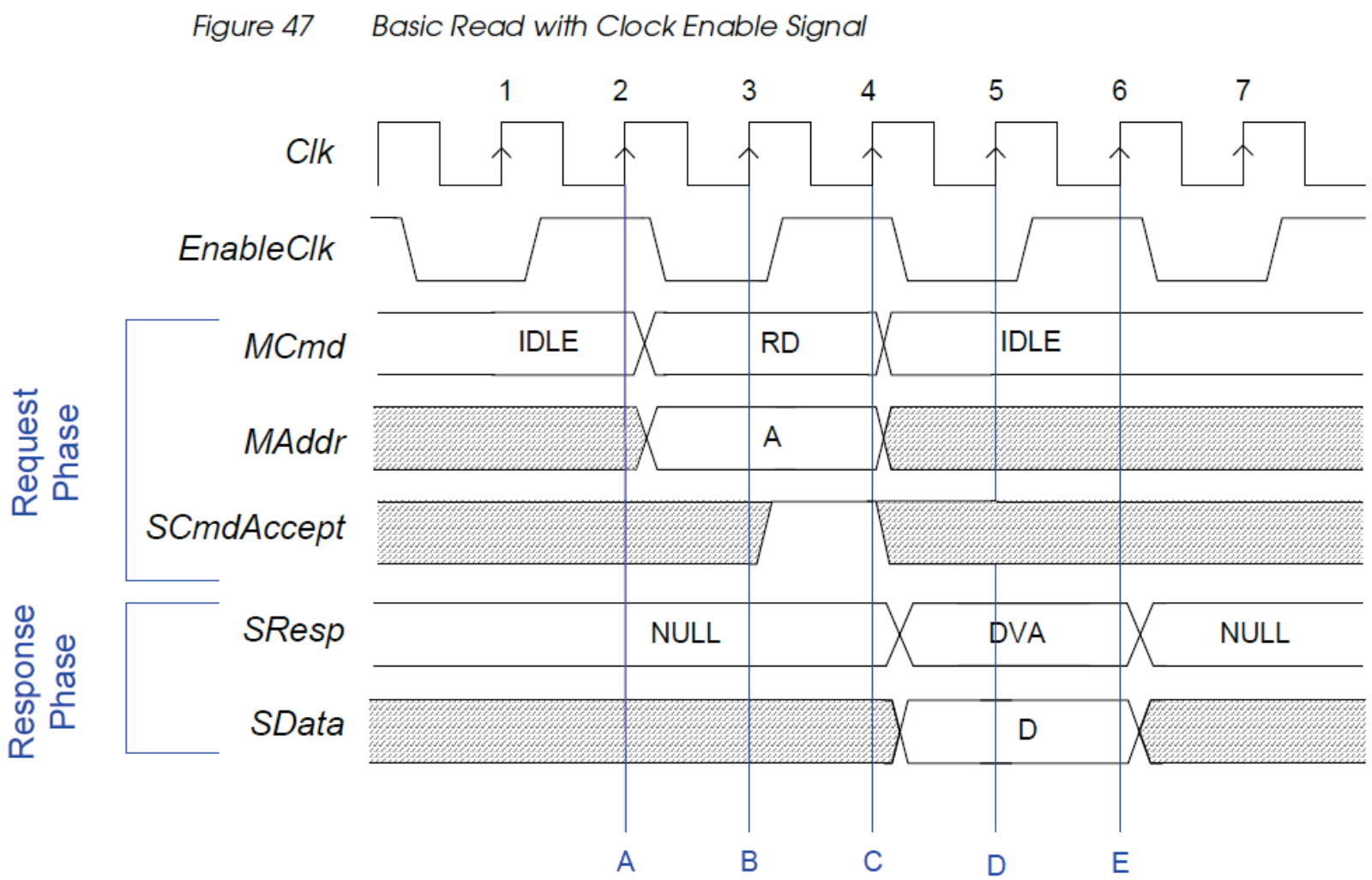
Connection Identifier

- **Connection (MConnID) is a tag that travels with a request:**
  - Usually an unique identifier for the master
  - Can be dynamically assigned by the master or assigned by the network
  - Should be delivered “as is” by the network
  - So the slave can know the master ID of a given request
- **Example uses**
  - Quality of Service determination
  - Security/protection identification
  - Debug

Sideband Signals

- **Generation and delivery synchronous to OCP clock**
- **Transit asynchronous to *data flow***
- **Signal Types**
  - Reset
  - Interrupt
  - Error
  - Core Flags
  - Connection
  - Control/Status

Divided Clocking



Reset

- **Reset de-assertion is synchronous to the OCP clock**
  - Assertion can be asynchronous
- **This is NOT normally the reset for the core but of the socket**
  - Should be seen as a protocol signal, used to restart transactions from scratch, clearing any transaction history
- **Must be asserted for at least 16 cycles**
- **OCP Reset**
  - Is mandatory!
  - Can be driven by the master (MReset_n)
  - OR can be driven by the slave (SReset_n)
  - OR can be driven by both master and slave (voting reset)

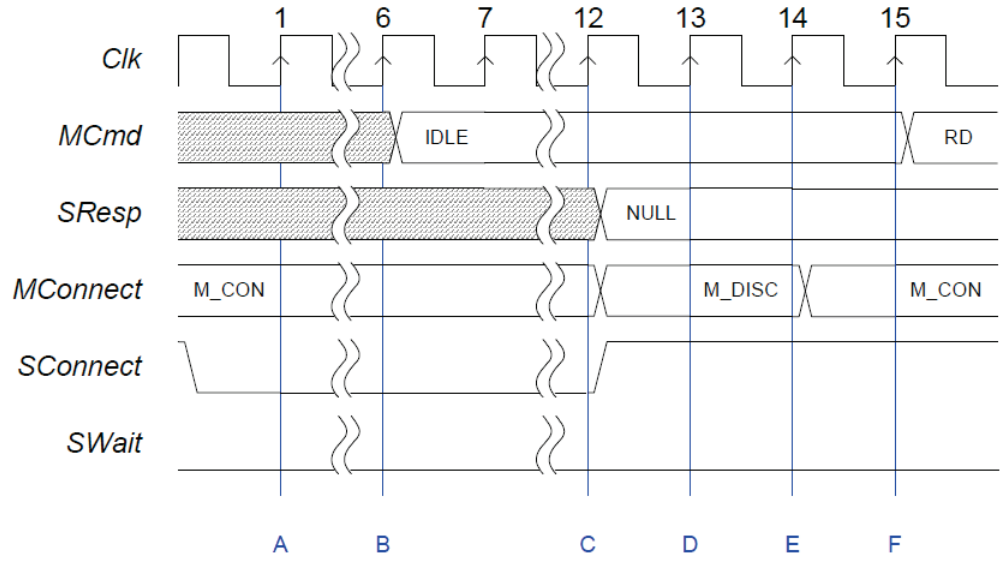
Flags

- **Master and Slave flags (MFlags and SFlags) are generic, out-of-band signals driven by master and slave, respectively**
- **Examples of use**
  - High-level flow control
  - Multiple interrupts
  - Handshaking
  - Narrow data links
- **OCP puts no restriction on their use**
- **Typically for hardware-to-hardware interactions**

Connection Protocol – Safe Power Transitions

- **MConnect/SConnect/SWait/ConnectCap**
  - Master state transition: M_OFF, M_WAIT, M_DISC and M_CON
  - Slave votes for connect/dis-connect: S_DISC and S_CON
  - Slave delayed state transition: S_OK and S_WAIT
  - Connect Cap 0 (statically connected), 1 (dynamic)

Figure 48 Slave-requested Disconnect Sequence



Test Signals

- **Mainly for hard macros...**
- **Scan**
  - With one or more chains
- **IEEE 1149 (JTAG)**
  - Standard JTAG interface
- **Clock control**
  - Used to supply a different clock for chip test

SUPPORTING TOOLING

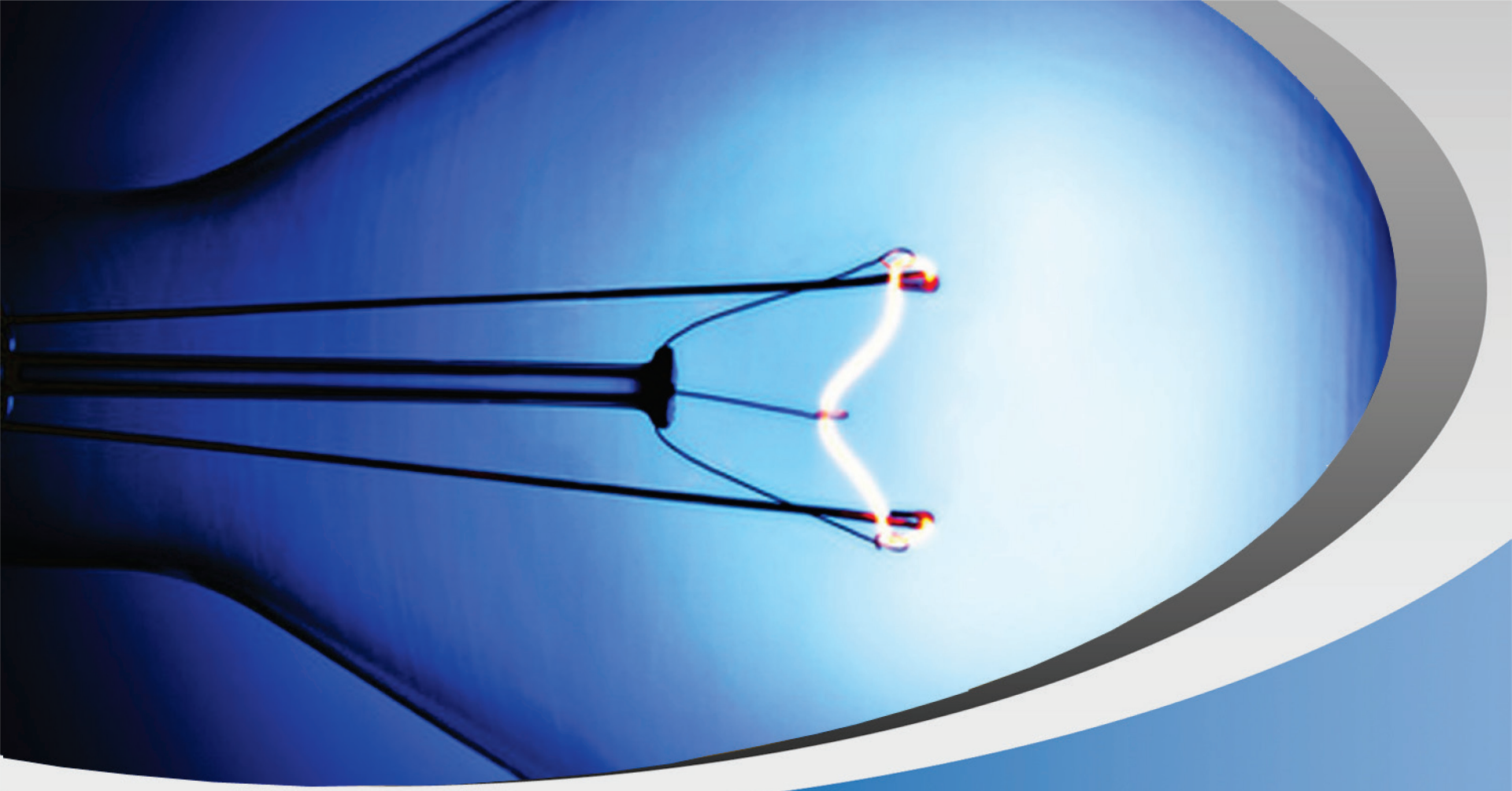
OCP Related Tools (SOLV)

- **OCP monitor/protocol checker**
  - Fully configurable monitor module useable in simulations
  - Implemented using SystemVerilog Assertions (SVA)
  - Can be attached to any OCP bundle
  - Check protocol violations and produce trace files during simulation
- **ocpdis**
  - Post processing tool to produce human readable assembly language representation of OCP activity
  - Support various formats
- **ocpperf**
  - Post processing tool to produce performance analysis of OCP activity
  - Latency, bandwidth or utilization

SOLV is a Sonics licensed product

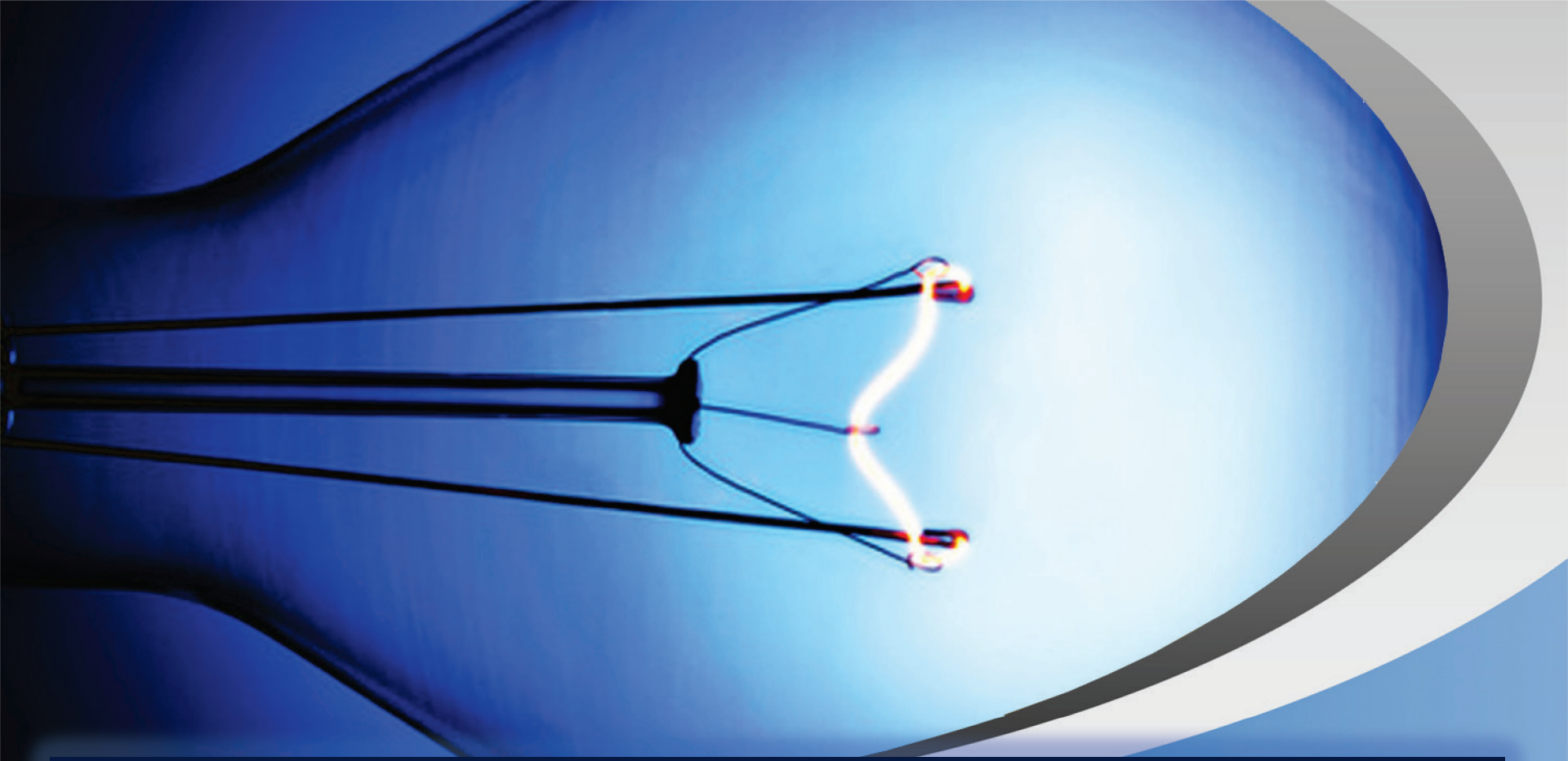
Other Commercial Providers of OCP Verification Technology

- Cadence
- Duolog
- Jasper
- Magillem
- Mentor
- Synopsys



Thank you

accellera
SYSTEMS INITIATIVE



OCP: The Journey Continues

Verification Support for OCP

Steve McMaster, Synopsys

accellera
SYSTEMS INITIATIVE

OCP is Hard to Verify / OCP is Easy to Verify !!!

- **High configurability of OCP is intended to simplify the development and verification of IP cores**
  - But on-chip network fabrics – and Verification IP – may need to support a wide variety of configurations
  - Profiles created to simplify configurability
- **Verification challenges**
  - Configuration support
  - High feature set
- **Verification support**
  - Consistent semantics (vs. separate interfaces, e.g., AXI/AHB/APB)
  - Specification provides complete verification properties
  - Explicit configuration metadata avoids ambiguity

Verification Support: Consistent Semantics

- **While Specification covers a wide range of features, most capable interfaces are a superset of less capable**
  - Features not enabled don't impact enabled features
  - Contrast this with AXI vs. AHB vs. APB
- **Generally only one configuration which fits the communication requirement**
  - May be alternatives to alter the efficiency, but basic flow remains consistent
- **Result: Designer can build a design/VIP environment via basic problem decomposition**
  - Refinements can be made without going back to the beginning

Verification Support: Specification Provides Complete Verification Properties

- **Section 3 of OCP Specification devoted to verification**
  - Protocol compliance, configuration compliance and functional coverage chapters
  - All rules/coverage points driven off configuration metadata
- **Statistics for OCP 3.0**
  - Over 100 types of protocol compliance checks
  - Over 100 sets of configuration compliance checks
  - Over 200 functional coverage points
- **Implemented by commercial VIP**

Verification Support: Explicit Configuration Metadata Avoids Ambiguity

- **Types of configuration metadata**
  - Presence/absence of signal
  - Width of signal
  - Supported capabilities of signal (e.g., burst sequences, command types)
- **Metadata explicitly supports concepts of Master, Slave and Monitor**
- **Thus, feature subsets are explicit**
  - Much less extra handholding in testbench
  - Better automatic constraining of VIP behavior
  - Easier formal verification

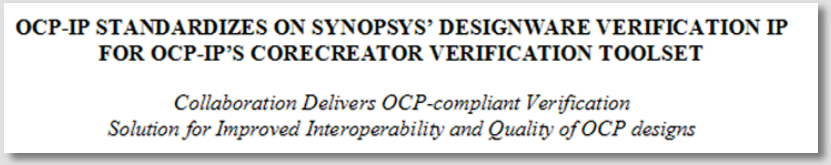
Synopsys VIP

OCP VIP EXAMPLE

Synopsys OCP VIP History

- **Synopsys OCP VIP is proven and established**
  - VIP originally released in 2005
  - Includes support for OCP 2.0 up to 3.0
    - Full range of possible configurations
  - Provided at no-charge to OCP-IP members from 2007
    - Integrated with OCP-IP's CoreCreator II
    - 2 licenses for each paid up member in good standing
    - Not carried forward to Accellera
  - Adopted by many OCP-IP members

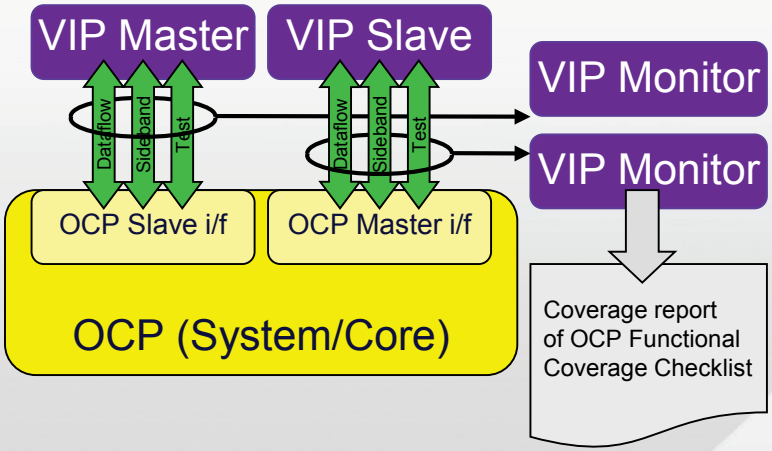
OCP International Partnership



accellera SYSTEMS INITIATIVE

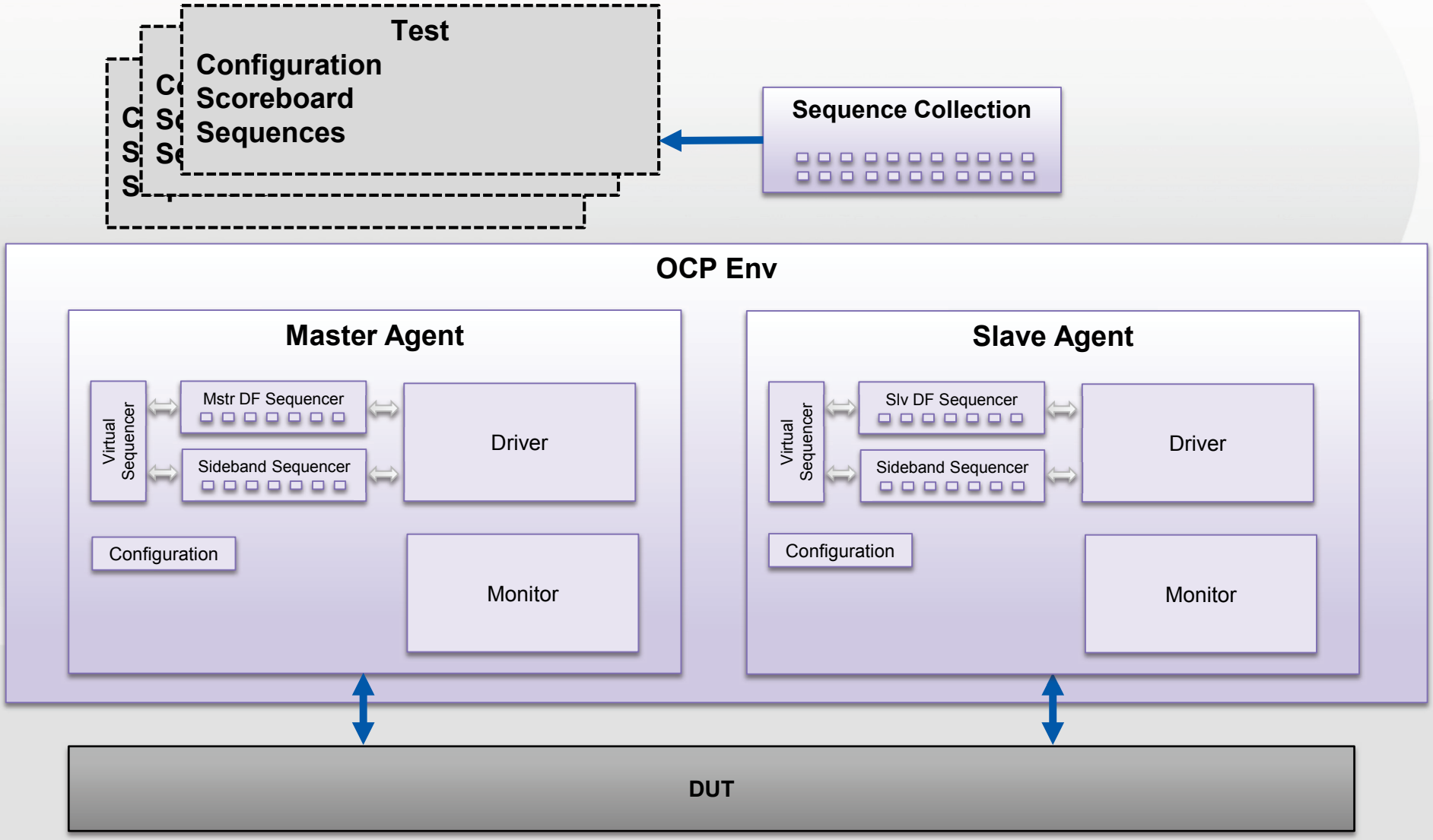
OCP Verification IP

- **OCP 2.2 dataflow & sideband transaction types**
  - ThreadBusy pipelined
  - Asynchronous reset, Enable clock
  - 2.2 Errata
  - Block transactions (2D burst)



- **OCP 3.0**
  - Reordering Responses for Transactions with Overlapping Addresses
  - WRNP and WRC Transactions Legal when writresp_enable is 0
  - Connect/Disconnect
- **Monitor observes and reports on OCP bus activity**
  - Full built-In Functional Coverage of OCP-IP-defined functional coverage groups
- **Fully configurable**
  - Signals, Width, Number of concurrent transfers, Timeouts, Burst length, Burst type, etc.

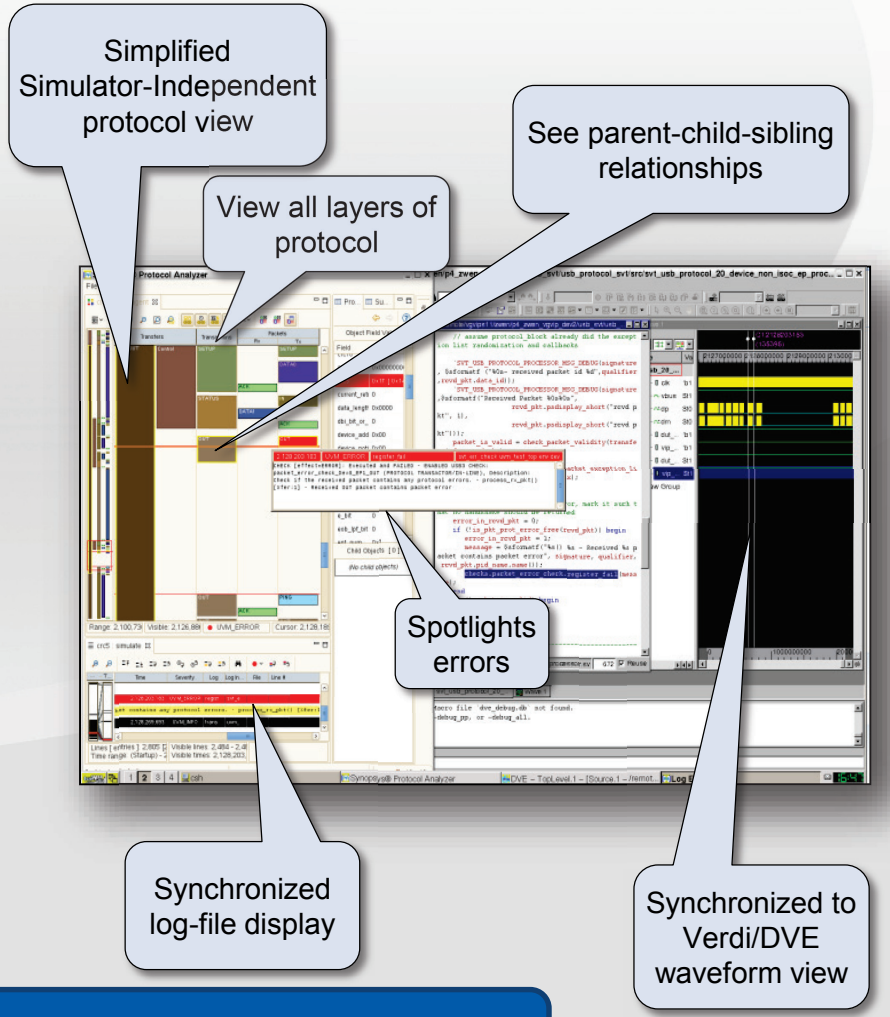
Example OCP UVM Test Environment



Protocol Aware Debug

Protocol Analyzer

- **Simplified viewing of protocol activity**
  - View all layers with parent, child, and sibling
  - Browse multiple protocols at once
- **Immediate error identification**
  - Spotlights errors on protocol-centric view
  - Provides on-demand detailed information
  - Extensive filtering to speed debug
- **Unified debug**
  - Synchronized to log files with back annotation of errors into protocol view
  - Link to VIP documentation
  - Can support customer VIP



Multi Protocol Views for SOC debug

accellera SYSTEMS INITIATIVE

Synopsys VIP Verification Plan

Full OCP-IP Spec Coverage

feature	subfeature	subfeature	subfeature	subfeature	subfeature	subfeature	subfeature	value dw_ocp.group
Functional Coverage								45.12%
	OCP Spec 2.2-1.1							45.12%
		Section 16						45.12%
			Mandatory					62.73%
				Dataflow				62.73%

subfeatur	subfeature	sub-feature	value dw_ocp.group
Transaction Coverage			71.62%
	trans_cross_trans_TypesResults		85.93%
		TransTypes	93.33%
		SResp	100.00%
		cc	64.44%
	trans_cross_trans_TypesBurstPropsByteens		73.13%
		TransTypes	93.33%
		MAtomicLength	57.81%

- **Test Plan**
  - Derived from specification and protocol-specific coverage groups
- **Automated back-annotation of coverage results**

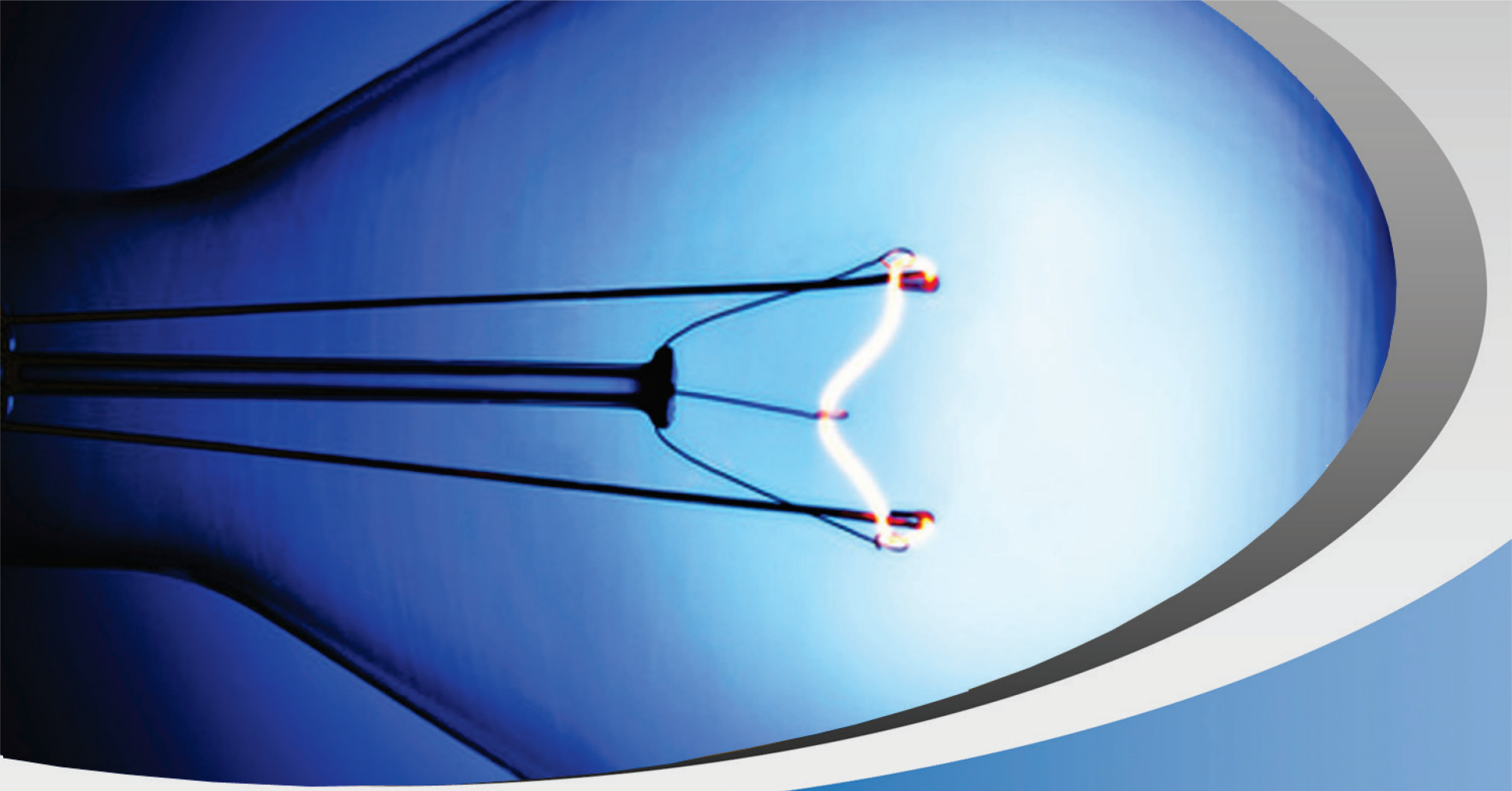
Results

Coverage Bins

accellera SYSTEMS INITIATIVE

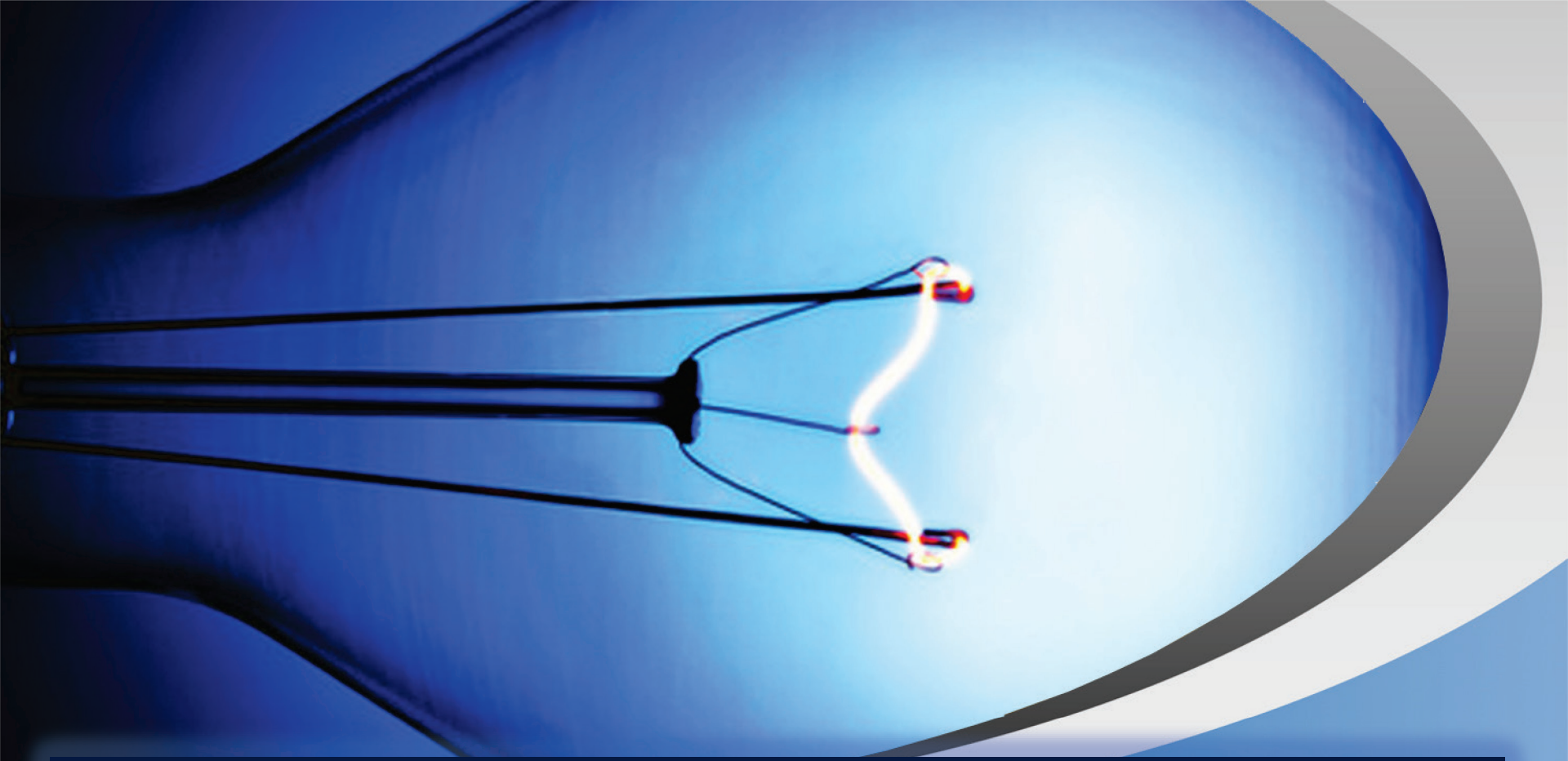
Conclusion

- **OCP is complex**
- **Protocol continues to be enhanced**
- **Commercial VIP enables complete, productive verification**
  - Aligned with customer demand for new versions of protocol
  - Support for new languages and methodologies
  - Enables productive debug
  - Includes coverage and planning



Thank you

accellera
SYSTEMS INITIATIVE



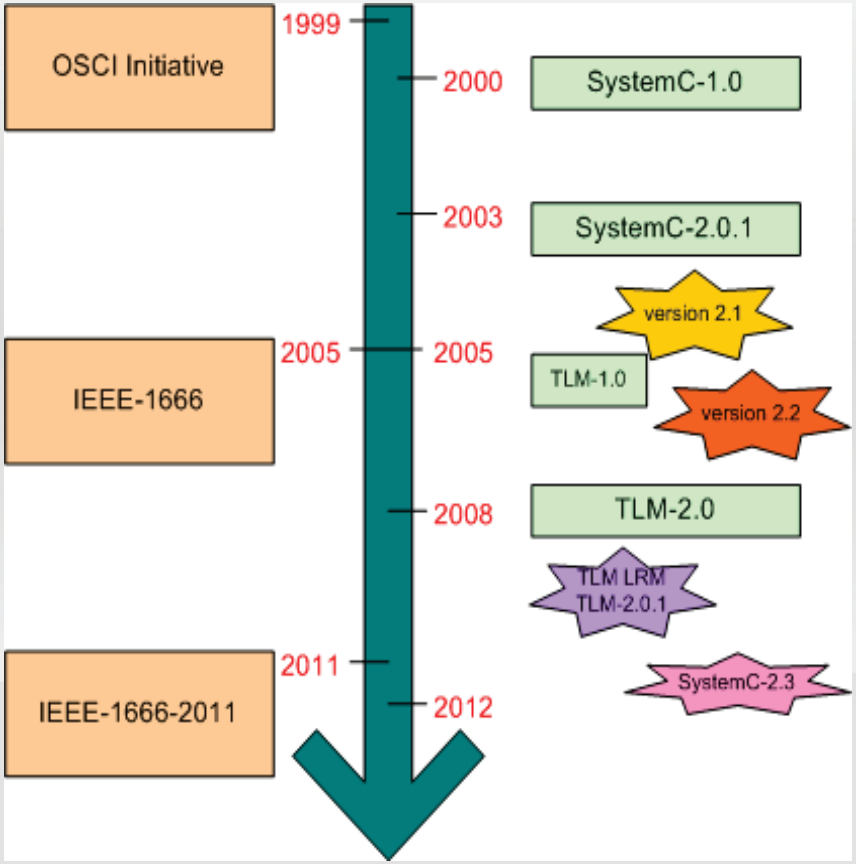
OCP: The Journey Continues

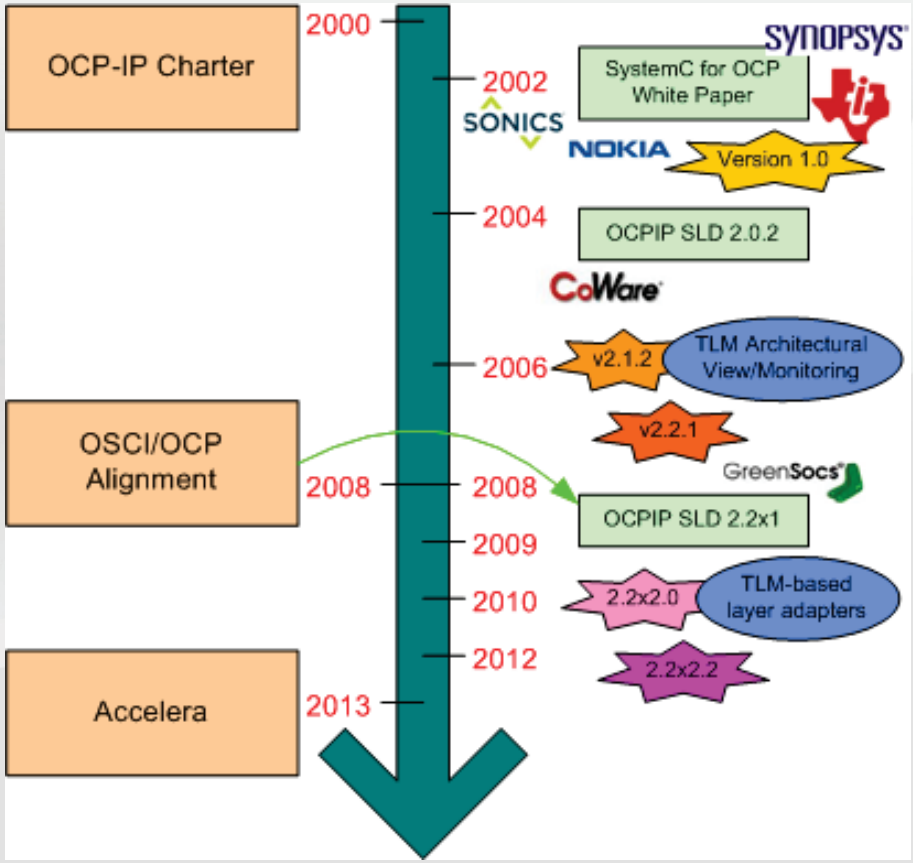
SystemC TLM 2.0 Support for OCP

Herve Alexanian, Sonics, Inc.

accellera
SYSTEMS INITIATIVE

OCP-IP and SystemC Brief History Comparative





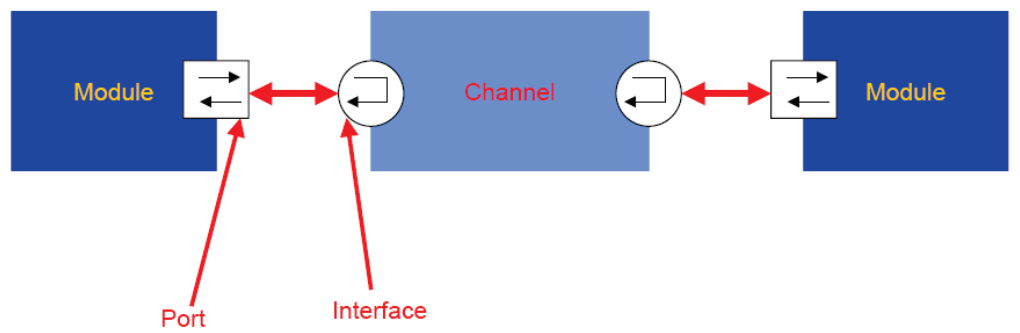
accellera SYSTEMS INITIATIVE

Modeling Layers – Terminology

- **OCP-IP coined these terms to identify layers of transaction representation**
  - TL4: Bursts transferred in one transport call. Blocking transport
  - TL3: Similar representation but non-blocking.
    Implies independent request and response paths
  - TL2: Hybrid between TL3 and TL1. Intra-burst Timing
  - TL1: Clock cycle-based. Every phase of the protocol is modeled
  - TL0: Pin-level, using systemc representation that can map to Verilog
- **Massive Confusion Alert!**
  - TL layers pre-date the OSCI TLM standards
  - OCP-IP TL1 **is not the same as** TLM-1.0
  - OCP-IP TL2 **is not the same as** TLM-2.0

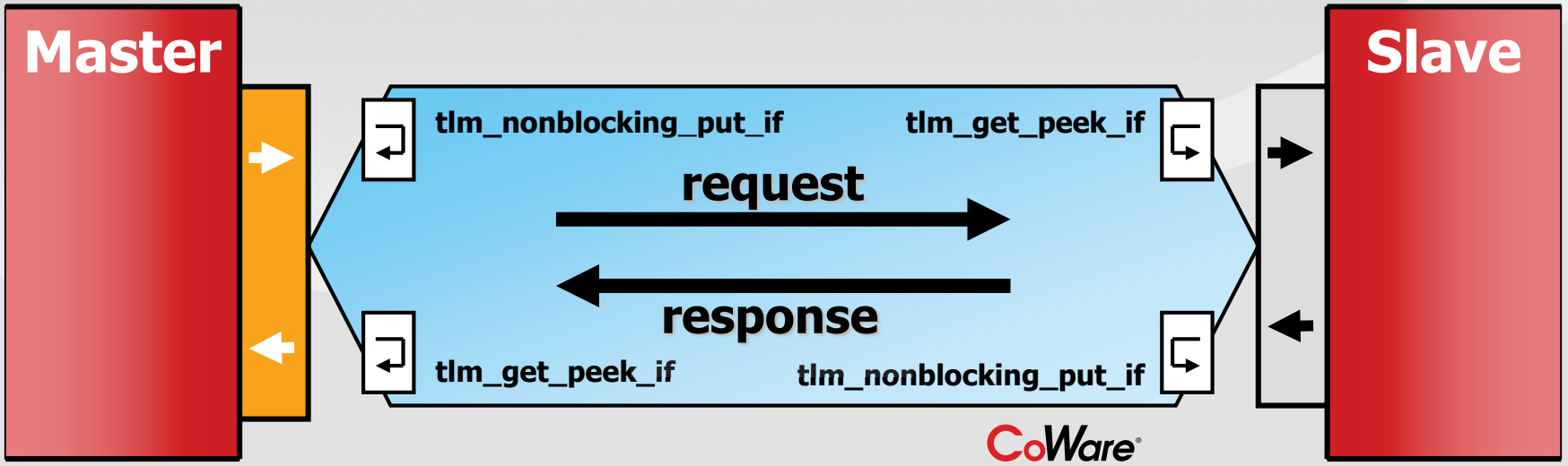
Channel-based (pre-TLM)

- **SystemC defined communication with interfaces, ports and channels**
- **OCP-IP defined custom Interfaces**
  - OCP_TL1_MasterIF, OCP_TL1_SlaveIF
  - OCP_TL1_MonitorIF
  - OCP_TL2_MasterIF, OCP_TL2_SlaveIF
- **Interfaces templated on OCP data structures**
  - OCPRequestGrp, OCPDataGrp, OCPResponseGrp
  - OCPTL2RequestGrp…
- **Functionality implemented in a channel**



OCP TL3 Channel

- **Higher levels of abstraction evolved to use TLM concepts**
  - Performance Monitoring, concept of multiple monitors
- **Using TLM-1.0 put/get interfaces**
- **Still a channel, still custom data structures**



TLM-2.0 Concepts

- **TLM-2.0 defines a modeling style and a number of concepts and classes**
- **Transport mechanism (required)**
  - Transactions passed by reference
  - Implemented in sockets (no more channels)
  - Defines abstract interface to pass transaction, phase and timing information

```
void b_transport( <TXN>&, sc_time& );
tlm_sync_enum nb_transport_fw( <TXN>&, tlm_phase&, sc_time& );
tlm_sync_enum nb_transport_bw( <TXN>&, tlm_phase&, sc_time& );
```

- **TLM generic payload (very important)**
  - Common representation for all modeling layers
  - Common basis with other protocols
  - Allows custom extensions without inheritance
- **Base protocol (BP)**
  - Defines 4 phases: BEGIN_REQ, END_REQ, BEGIN_RESP, END_RESP
  - Suitable for approximately timed modeling
  - Custom TLM phases can be defined

What Does OCP-IP Need to Do, Then?

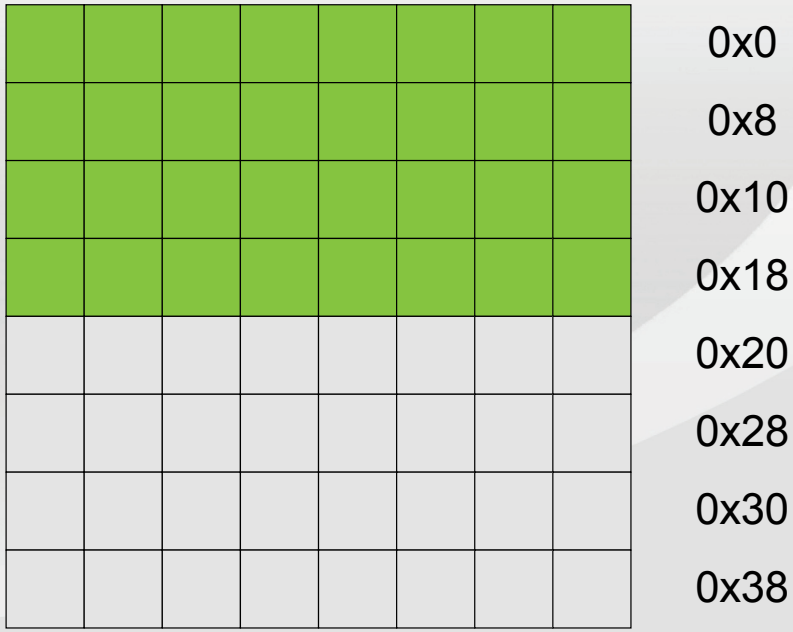
- **Define how to use TLM to model OCP transactions**
  - At higher levels of abstraction, not that much!
  - Represent OCP transactions (bursts) with TLM generic payload
  - Specify TLM extensions to TLM generic payload and Base protocol
    - Extensions: Threads, tags, ConnID, burst modifiers
    - Phases: Interface reset, data handshake, thread busy
- **Facilitate Integration**
  - Layer Adapters
    - TL0/TL1
    - TL1/TL3
    - Legacy adapter (to OCP-IP channel based modules)
  - Help access other OCP-IP deliverables
    - Tools (trace files, disassembly, debug)
    - Protocol Checking

OCP SystemC Next Generation Interface Standards

	OCP-IP SystemC Interface	OSCI TLM compatibility
TL0	Not specified separately for SystemC from other HDLs	None, this is the RTL level
TL1	OCP-IP TL1	Uses TLM-2 generic payload, sometimes with extensions. Uses different protocol phases and rules from OSCI TLM-2.0 BP. Uses nb_transport()
TL2	OCP-IP TL2	Uses TLM-2 generic payload, sometimes with extensions. Extensions are a subset of the extensions used at OCP-IP-TL1. Uses different protocol phases and rules from OSCI TLM-2.0 BP and from OCP-IP-TL1. Uses nb_transport()
TL3	OCP-IP TL3/TL4	Uses TLM-2 generic payload, sometimes with extensions. Extensions are a subset of the extensions used at OCP-IP-TL2. Uses the same protocol phases and rules as OSCI-TLM-2.0 BP. Extensions may be ignorable in which case OCP-IP-TL3 is directly interoperable with OSCI-TLM-2.0-BP. Uses nb_transport() and b_transport()
TL4		

4-beat Burst Transaction Example

- **OCP (data_wdth=64)**
  - MCmd=WR
  - MAddr=0x00
  - MBurstLength=4
  - MBurstSeq=INCR



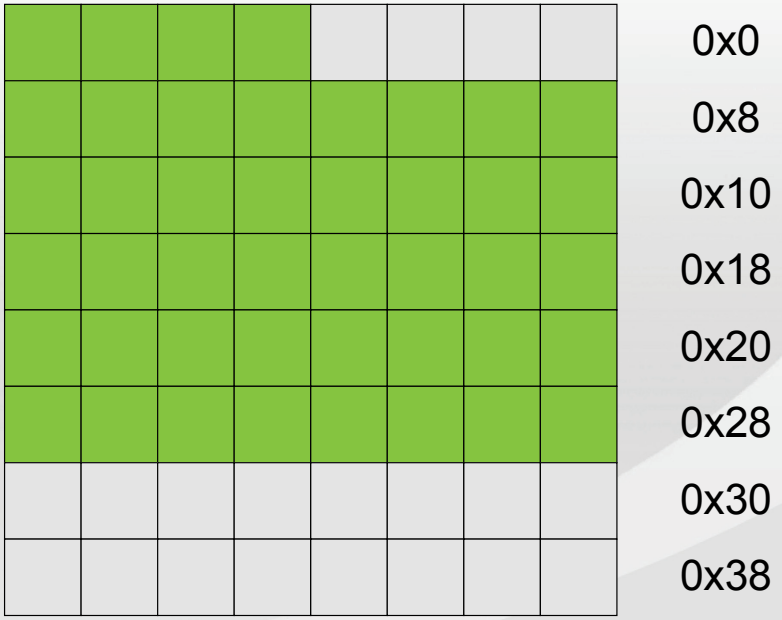

```
{
    // create a transaction
    tlm::tlm_generic_payload* txn = socket.get_transaction();
    txn->set_write();
    txn->set_address( 0 );
    uint32_t length = 4 * 8; // 4 beats on 8-byte wide socket
    txn->set_data_length( length );
```

Wait: nothing OCP here!

Misaligned Burst

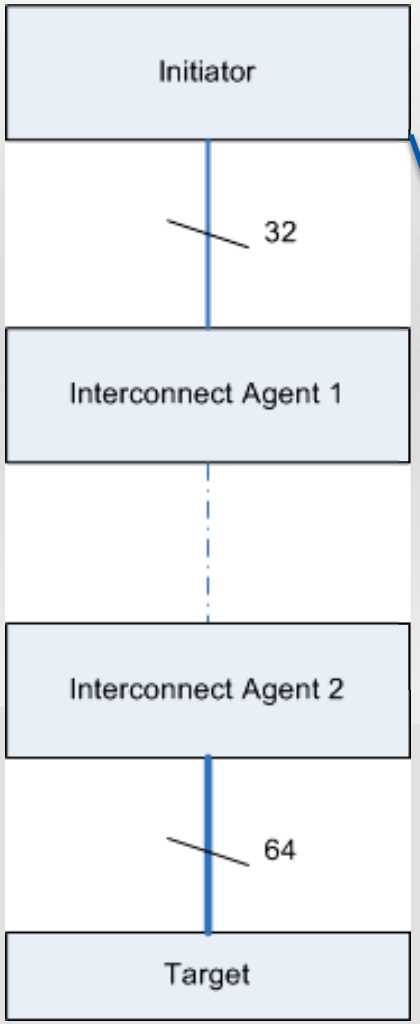
- **What about this transaction?**
  - txn->get_address() == 4;
  - txn->get_data_length() == 44;
- **Legal TLM, but not OCP**

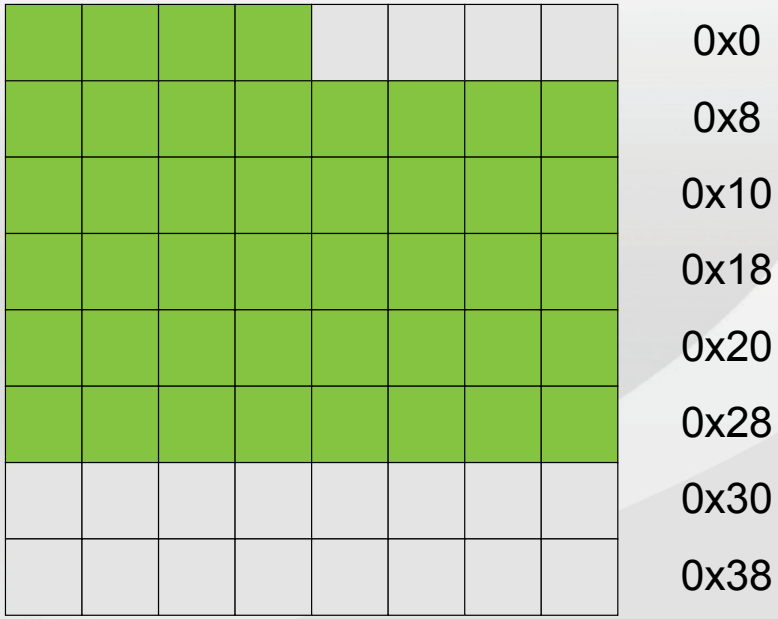
```
my_slave::nb_transport_fw( tlm::tlm_generic_payload& txn,
                           tlm::tlm_phase& phase,
                           sc_core::sc_time& time )
{
    // my socket is 64-bits, OCP addresses are aligned to 8 bytes
    assert( txn.get_address() % 8 == 0 );
```



Misaligned Burst

- **Remember, transactions are passed by reference!**



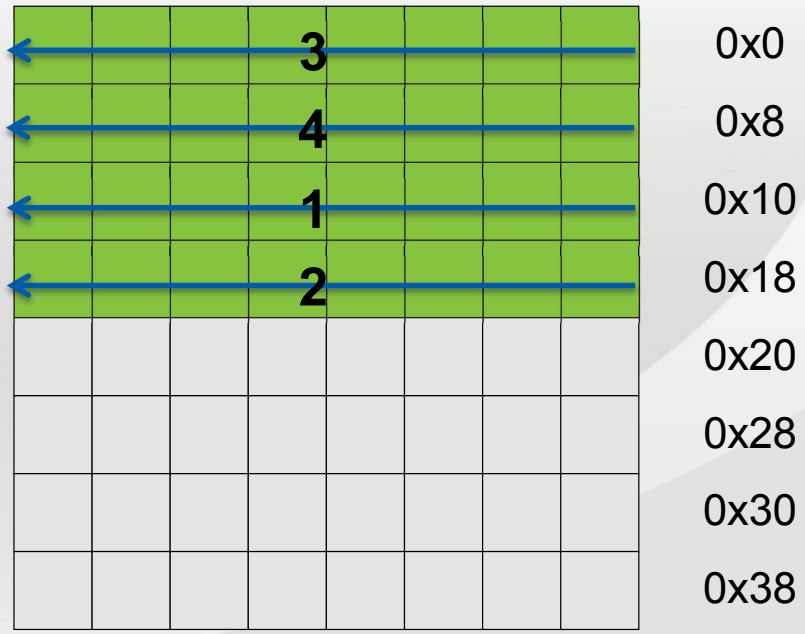


OCP (data_width=32)
MCmd=WR
Maddr=0x04
MBurstLength=11
MBurstSeq=INCR

If played on a 64-bit OCP, the effective OCP address would be 0, and the first transfer would have MDataByteEn=0xF0

4-beat WRAP Burst

- **OCP (data_wdth=64)**
  - MCmd=WR
  - MAddr=0x10
  - MBurstLength=4
  - MBurstSeq=WRAP



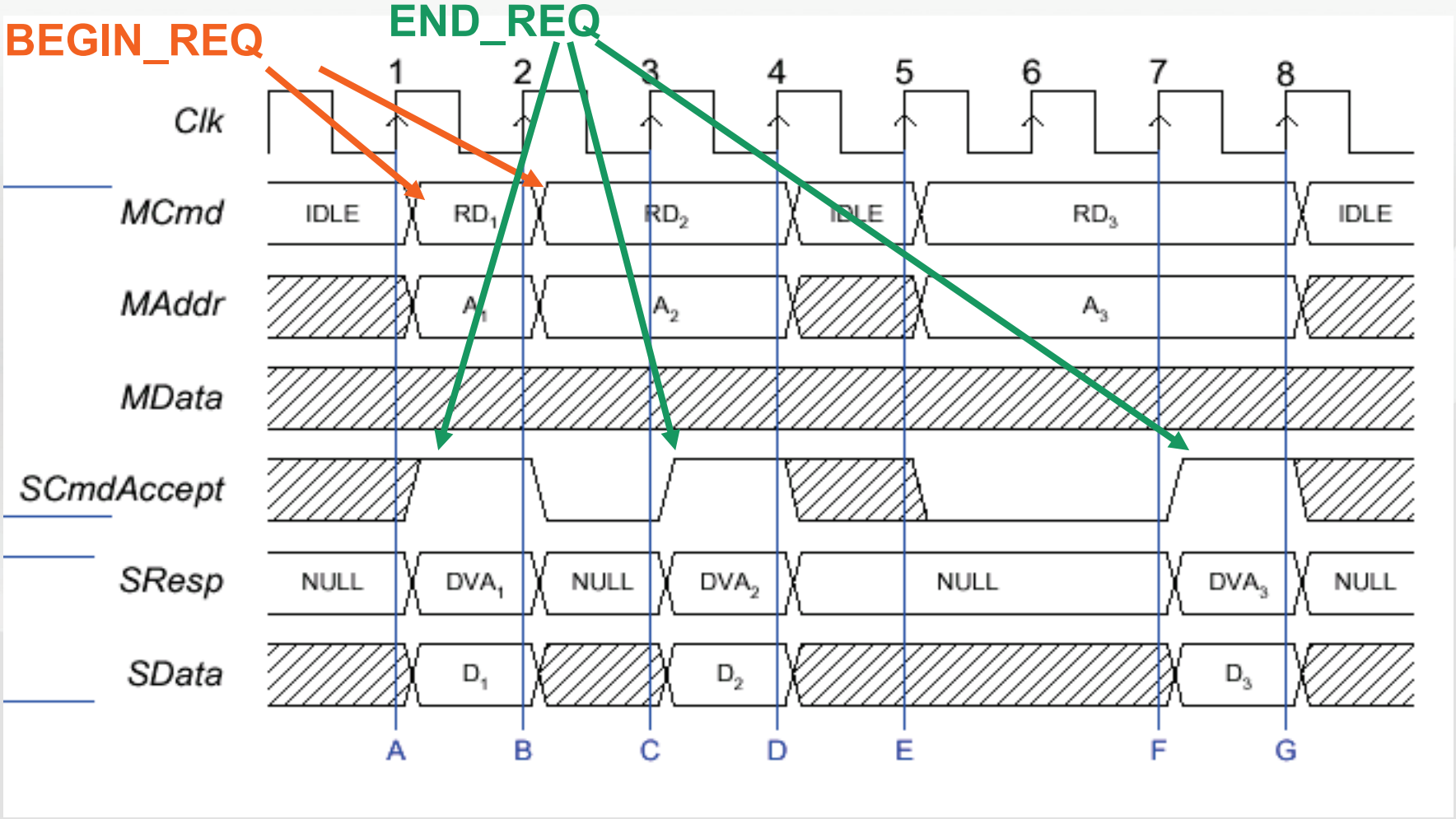

```
{
    // create a transaction
    tlm::tlm_generic_payload* txn = socket.get_transaction();
    txn->set_write();
    txn->set_address( 0 );
    uint32_t length = 4 * 8; // 4 beats on 8-byte wide socket
    txn->set_data_length( length );

    ocpip::burst_sequence* b_seq;
    ocpip::extension_api::get_extension<ocpip::burst_sequence>(b_seq, *txn);
    txn->get_extension(b_seq);
    b_seq->value.sequence=ocpip::WRAP;
    b_seq->value.xor_wrap_address = 0x10;
    ocpip::extension_api::validate_extension<ocpip::burst_sequence>(*txn);
```

And So On…

- **Bursts are codified in reference manual**
  - STRM bursts
  - XOR bursts
  - BLCK bursts (2D)
  - Legal Byte enable patterns
- **Other TLM payload extensions**
  - MThread
  - MTag
  - MReqinfo
  - MConnID
  - etc…

OCP TL1 Request Phase



TL1 Phases

- **TL1 only uses nb_transport_fw/nb_transport_bw**

```
tlm::tlm_sync_enum
nb_transport_fw( tlm::tlm_generic_payload&, tlm::tlm_phase&,
sc_time& );
```

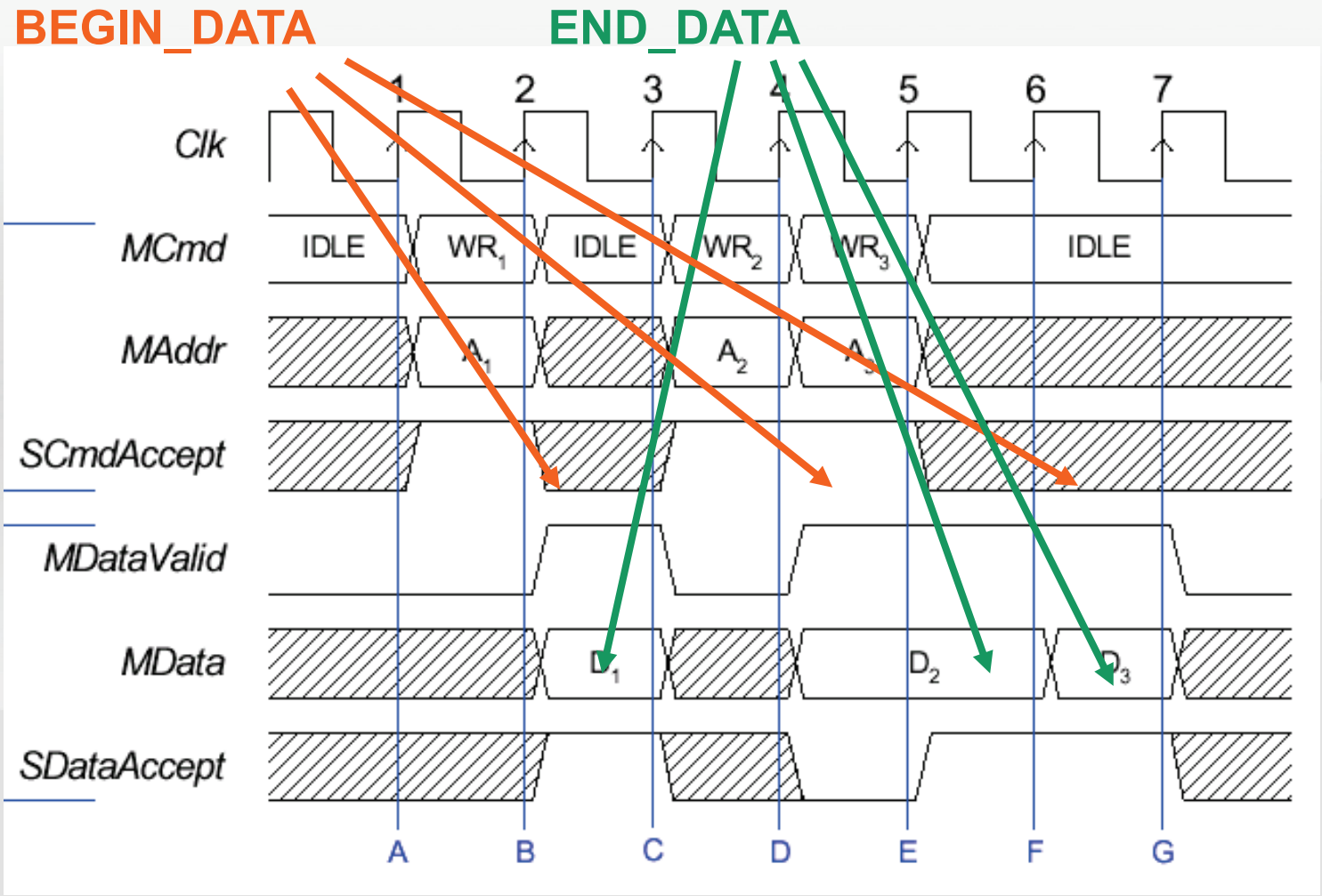
- **The sc_time argument is always 0.**
- **A phase begins with a nb_transport_* call with ph=BEGIN_***

```
tlm::tlm_phase reqPh = tlm::BEGIN_REQ;
sc_core::sc_time zero( 0, SC_NS );
tlm::tlm_sync_enum status = nb_transport_fw( txn, reqPh, zero );
tlm_::tlm_phase dataPh = ocpip::BEGIN_DATA;
status = nb_transport_fw( txn, dataPh, zero );
```

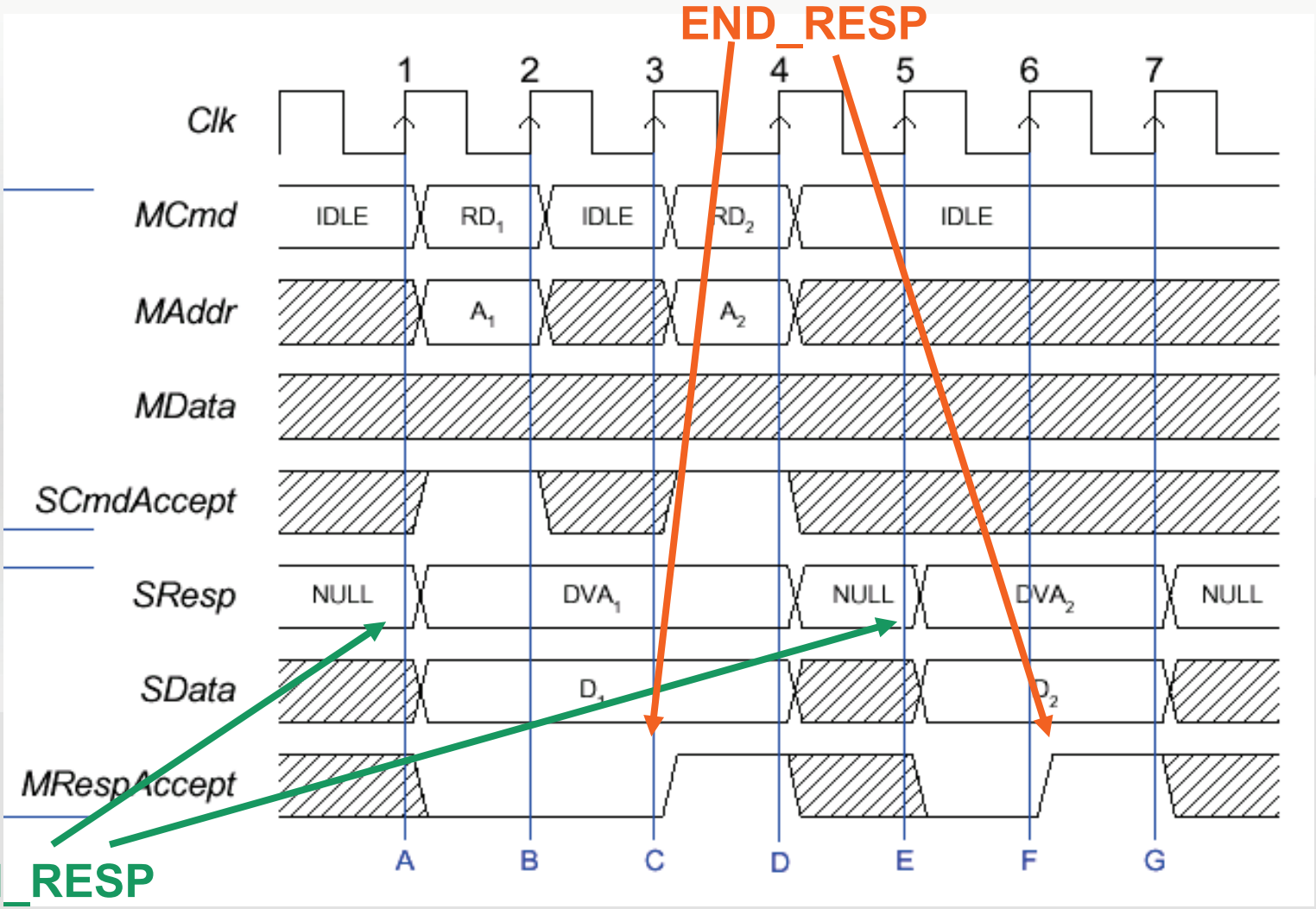
TL1 Phases

- **A TL1 phase ends in 2 possible ways**
- **the nb_transport_ call returns tlm::TLM_UPDATED**
  - The tlm::tlm_phase argument MUST be changed. The only valid change is to END_* (END_REQ/END_DATA/END_RESP)
  - This means the phase ends immediately. This is the equivalent of a having the accept signal at the same time as the valid is asserted.
  - This is the only legal way for an OCP phase with ThreadBusy
- **The nb_transport_call returns tlm::TLM_ACCEPTED**
  - This DOES NOT MEAN the phase is accepted! TLM_ACCEPTED simply means the receiver has accounted for the phase
  - The tlm::tlm_phase argument MUST NOT be changed. You can assert that!
  - The receiver will send a END_* phase later via
    - nb_transport_bw (to end a request or data phase)
    - nb_transport_fw (to end a response phase)

OCP TL1 Data Handshake Phase

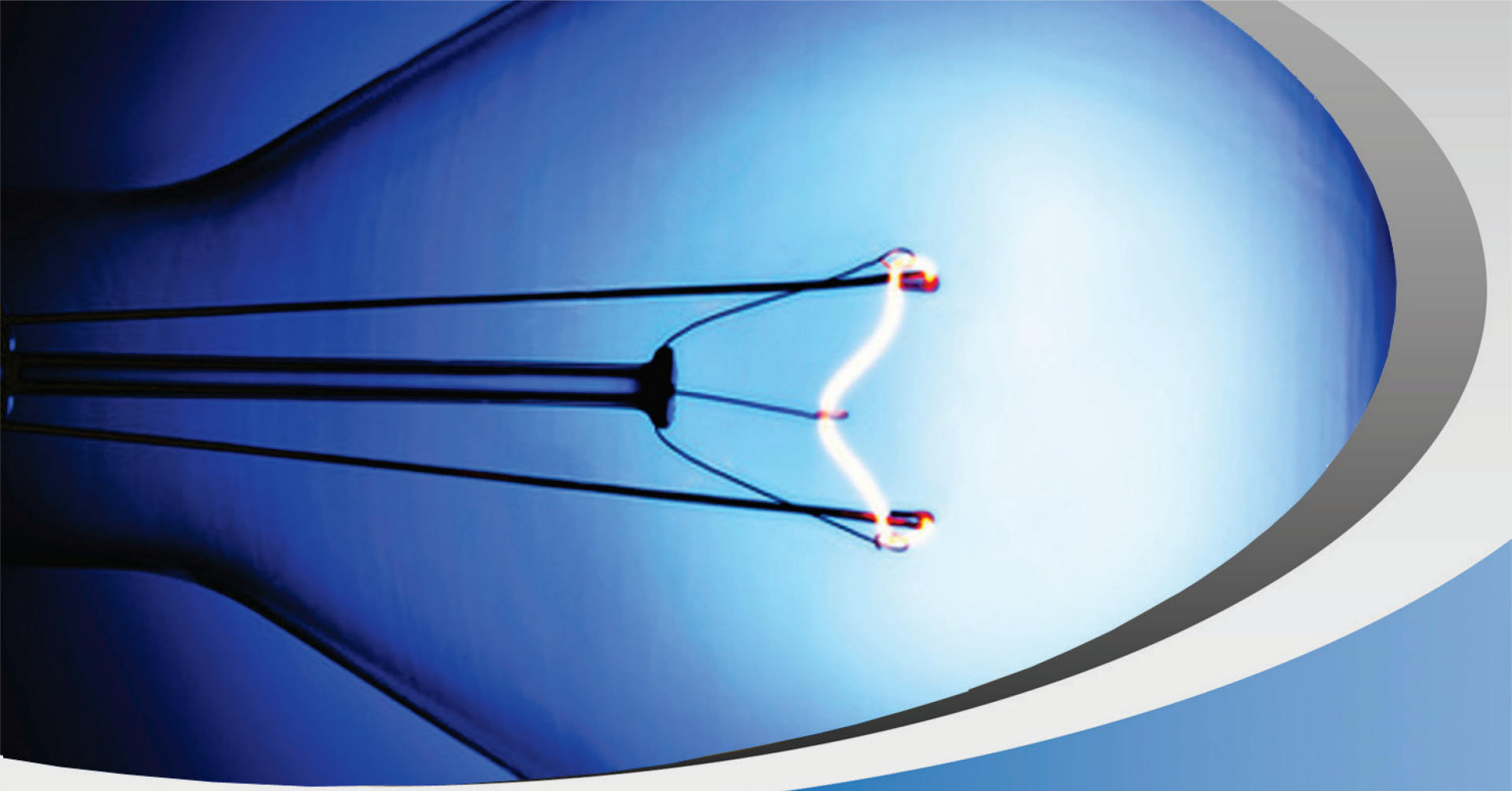


OCP TL1 Response Phase



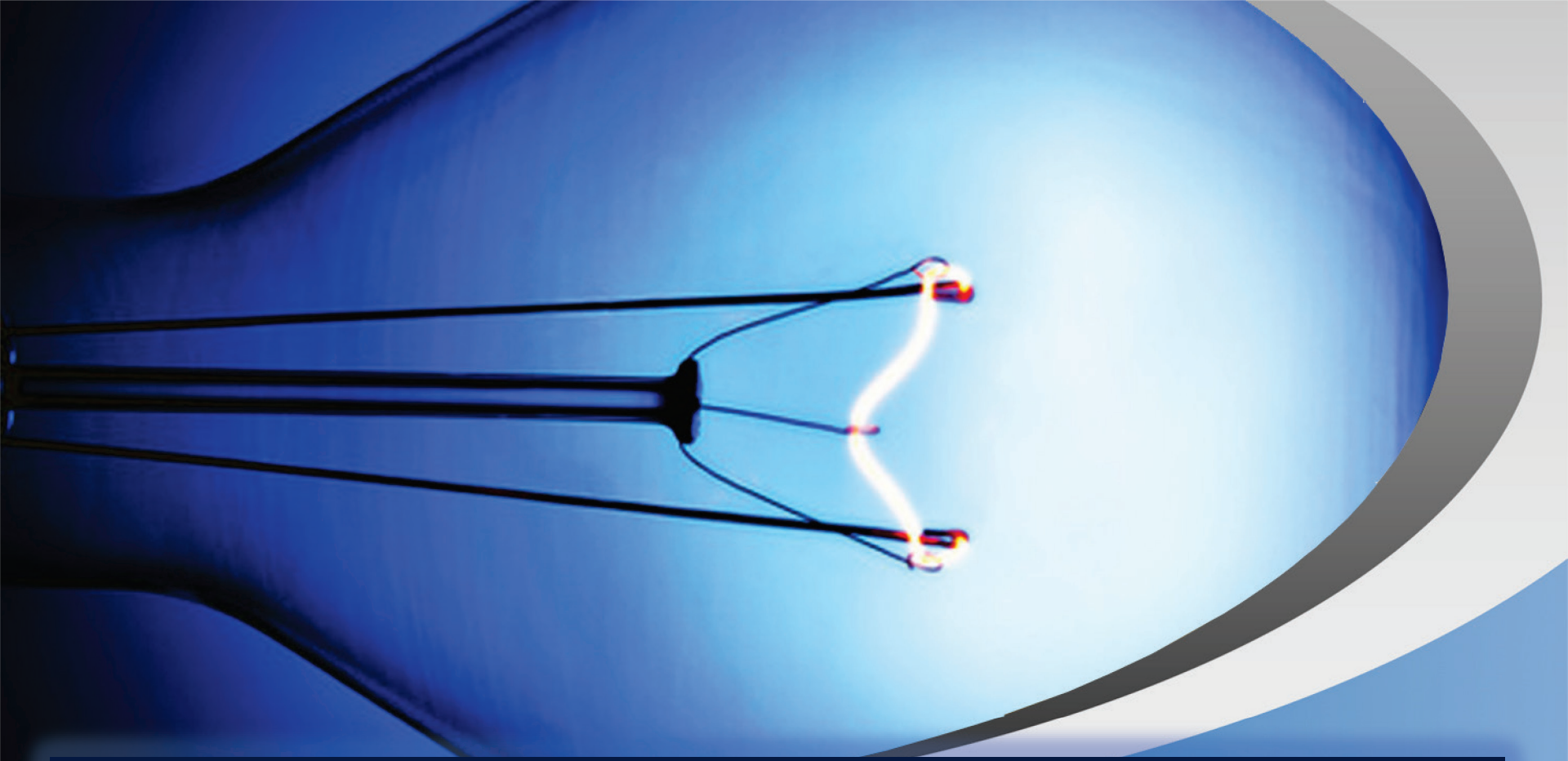
Non-dataflow Transfers

- **OCP interface reset (MReset_n/SReset_n) and ThreadBusy indications have to go through the socket**
- **This is done with custom TLM phases AND a dedicated transaction object stored on the socket**
- **Certain configurations of OCP allow combinatorial timing arcs between the master and slave. ThreadBusy is the most obvious example**
  - The modeling kit defines OCP specific sockets, with a timing contract facility
  - A component declares its phase specific “timing” at elaboration time
  - This facility was adapted from the channel-based OCP-IP modeling kit



Thank you

accellera
SYSTEMS INITIATIVE



OCP: The Journey Continues

IP-XACT Support for OCP

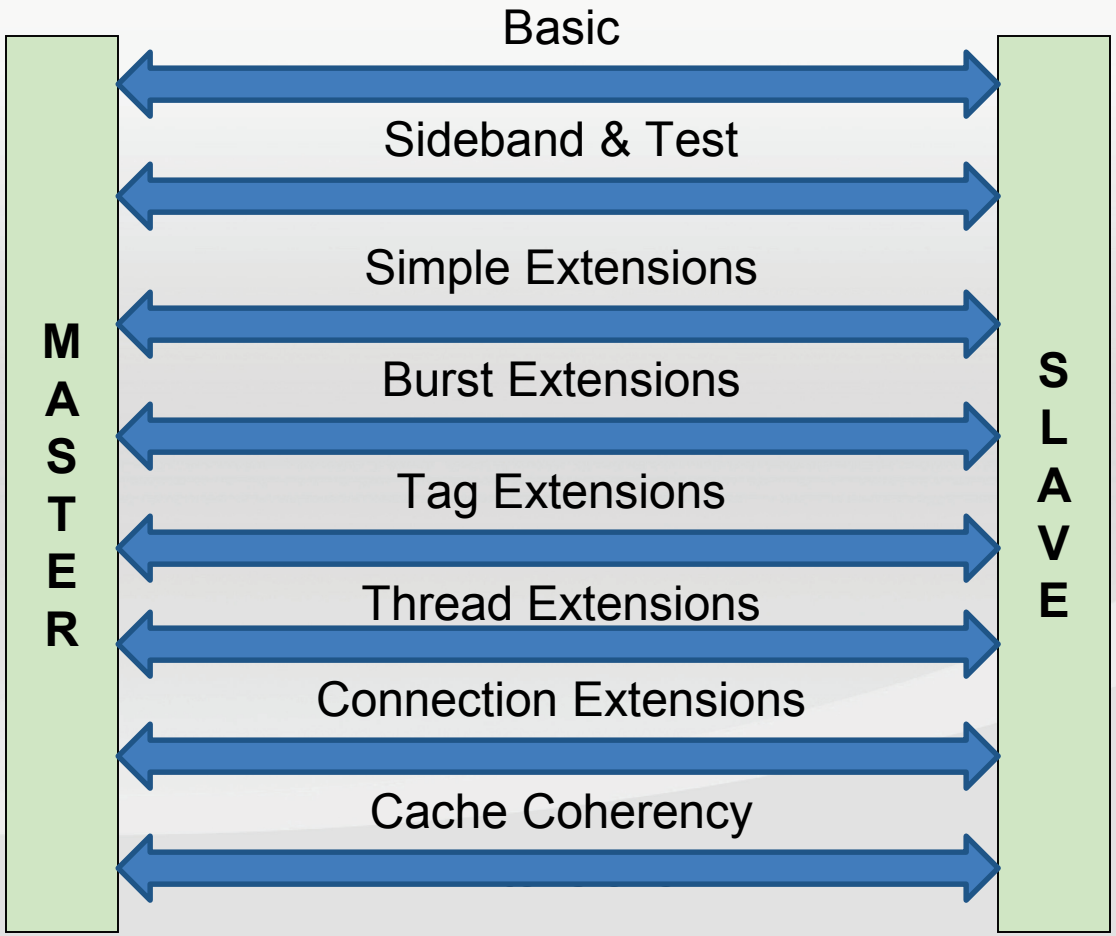
Prashant Karandikar, Texas Instruments

accellera
SYSTEMS INITIATIVE

Agenda

- Introduction
- OCP Vendor Extension
- OCP Checkers
- rtl.conf support
- IEEE1685-2014*
- Confluence of IEEE1685, IEEE1666 & OCP Standards

OCP Configurability



Except Basic Signal Group, all other extensions are optional

OCP Parameters

- **The OCP Specification defines, for each OCP interface, a certain number of parameters**

Table 13 OCP Signal Configuration Parameters

Group	Signal	Parameter to add signal to interface	Parameter to control width	Default Tie-off
Basic	Clk	Required	Fixed	n/a
	EnableClk	enableclk	Fixed	1
	MAddr	addr	addr_wdth	0
	MCmd	Required	Fixed	n/a
	MData	mdata	data_wdth	0
	MDataValid	datahandshake	Fixed	n/a
	MRespAccept[1]	respaccept	Fixed	1
	SCmdAccept	cmdaccept	Fixed	1
	SData[1]	sdata	data_wdth	0
	SDataAccept[2]	dataaccept	Fixed	1
	SResp	resp	Fixed	Null
Simple	MAddrSpace	addrspace	addrspace_wdth	0
	MByteEn[3]	byteen	data_wdth	all 1s

OCP Specification: Constraints

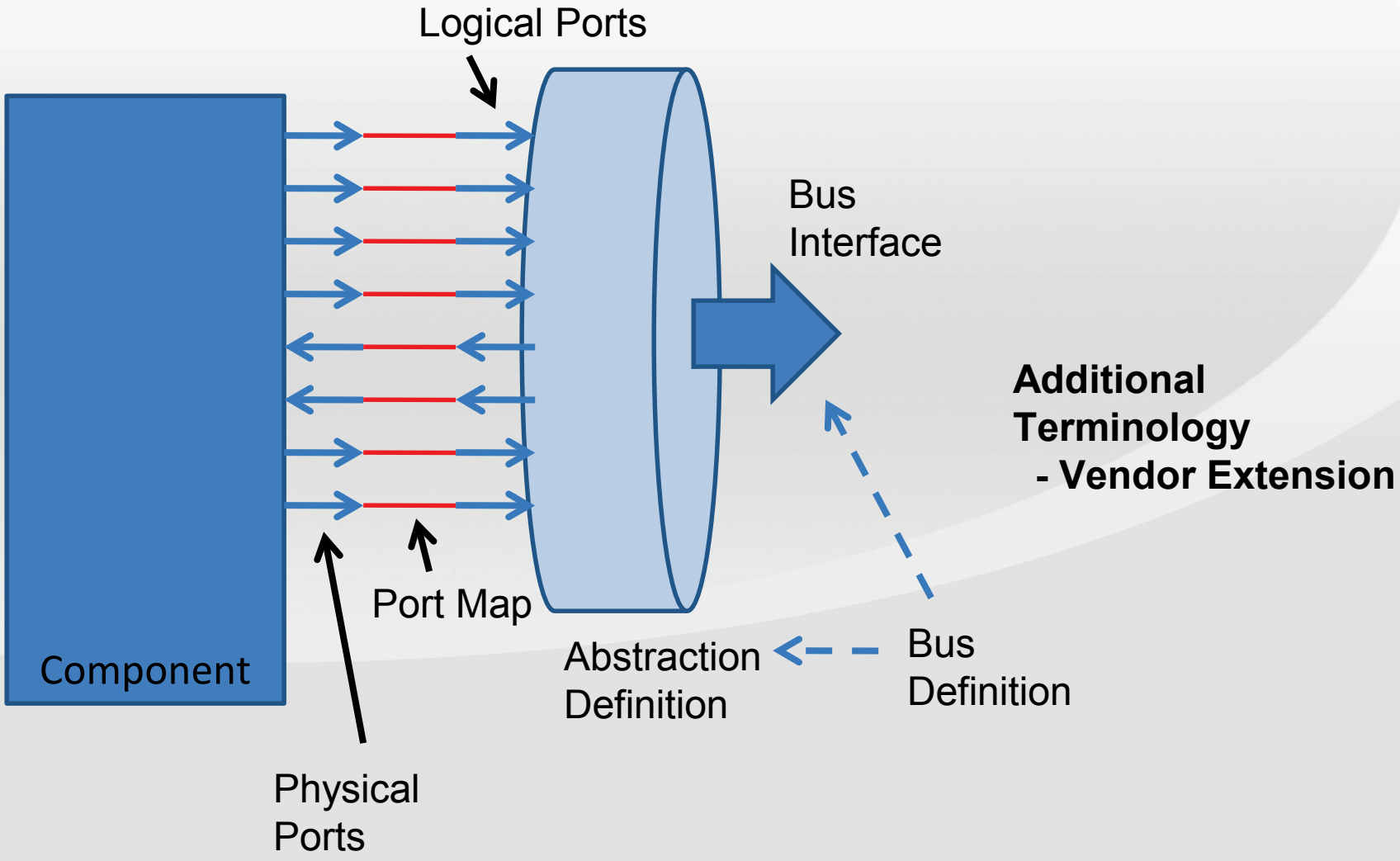
- **The OCP-IP Specification also defines a certain number of constraints between parameters checking the OCP compliancy**
- **Examples:**
  - 2.1.2: *byteen* can only be enabled if *sdata* or *mdata* is also enabled
  - 2.1.5: *byteen* is only supported when *data_wdth* is a multiple of 8
  - 2.1.23: The *burstlength_wdth* must be 0 if *burstlength* is disabled and must be greater than 1 if *burstlength* is enabled
  - …

IEEE1685 (2009 & 2014*)

- IP-XACT standard defines IP metadata description like interfaces, registers, bit fields in the form of an XML schema. It provides a common and language-neutral way to describe IP, compatible with automated integration techniques and enabling integrators to use IP from multiple sources with IP-XACT enabled tools.

* Yet to release

IP-XACT/IEEE1685 Terminology

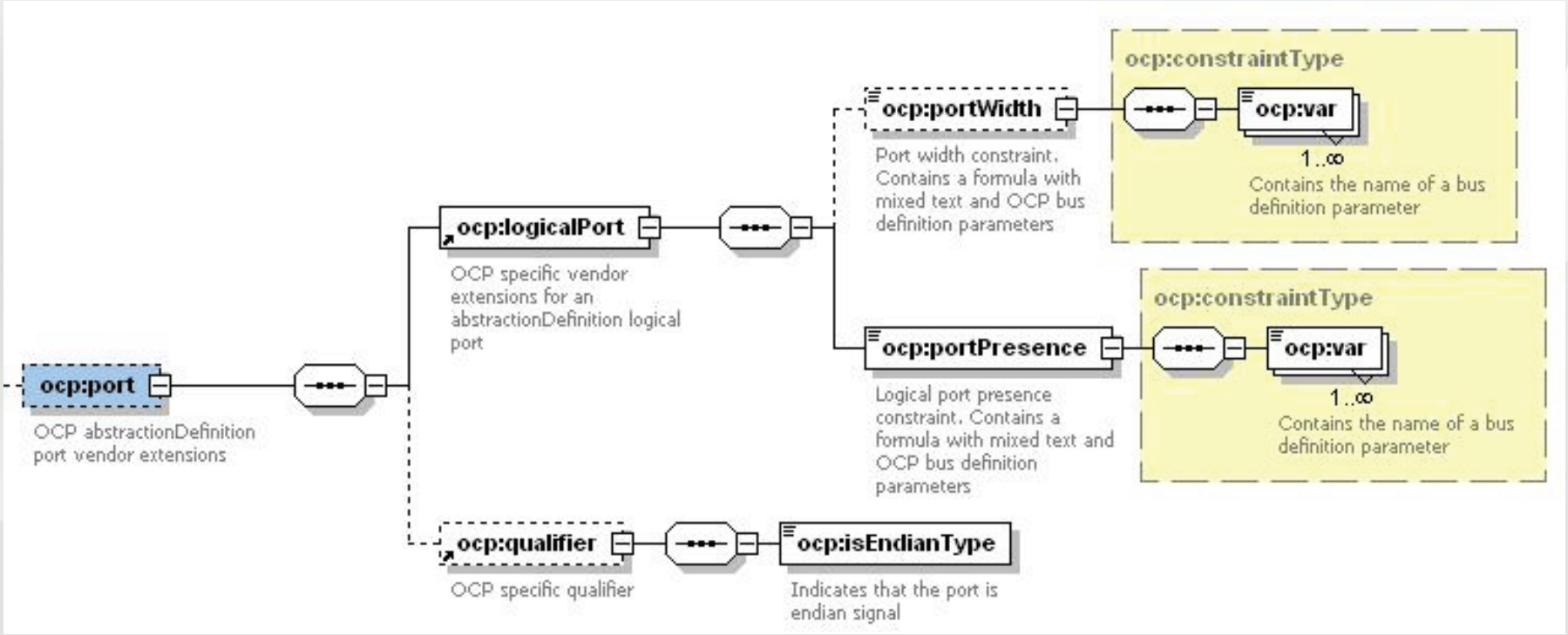


IEEE1685-2009 OCP Vendor Extensions

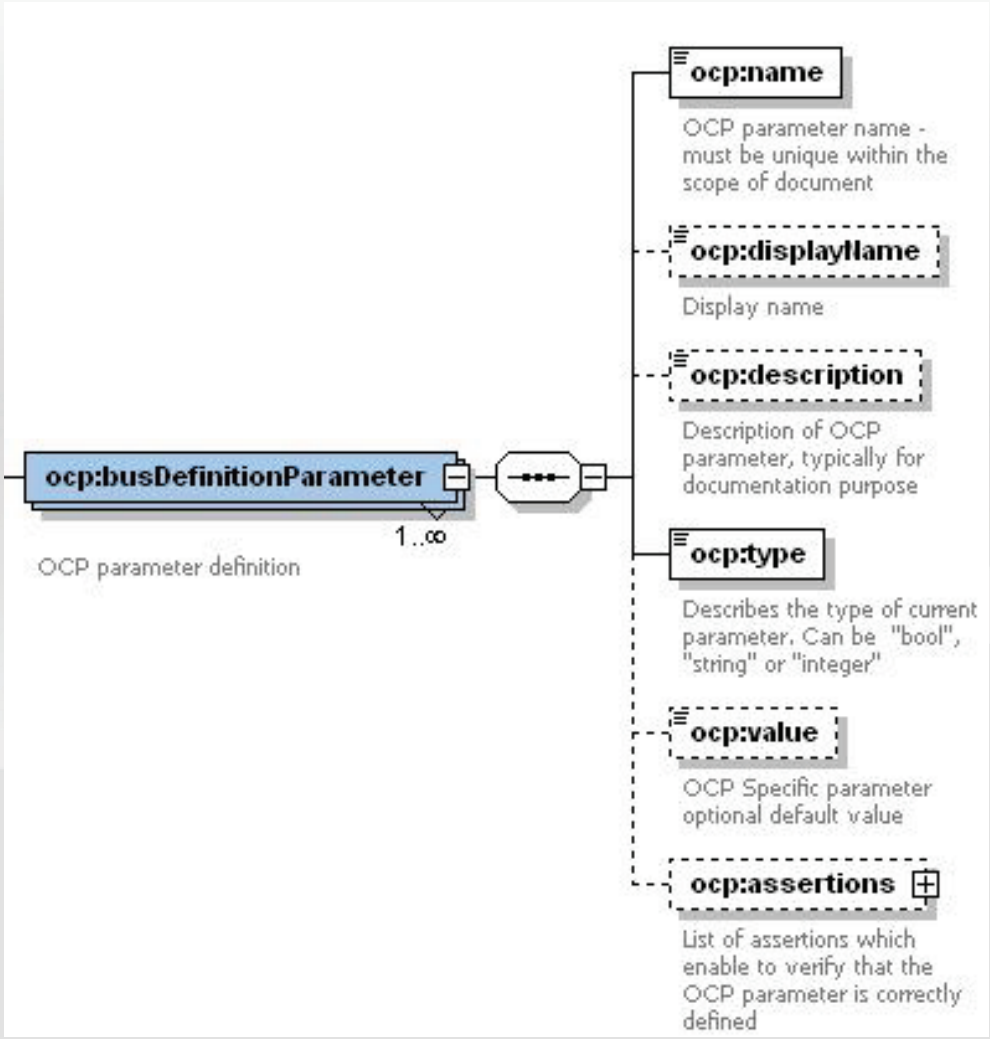
The OCP MDWG defined vendor extensions to allow the definition of these parameters and constraints:

- **busDefinition defines:**
  - OCP parameters (with description, type, default value and associated assertions)
- **abstractionDefinition defines:**
  - OCP logical ports and associated parameterized constraints:
    - portPresence: constraint indicating when this logical port should be mapped in an OCP busInterface, depending on OCP parameter value(s)
    - portWidth: width constraint of this logical port, depending on OCP parameter values
- **component/busInterface:**
  - Parameterization of OCP parameters (through standard IP-XACT busInterface parameters)

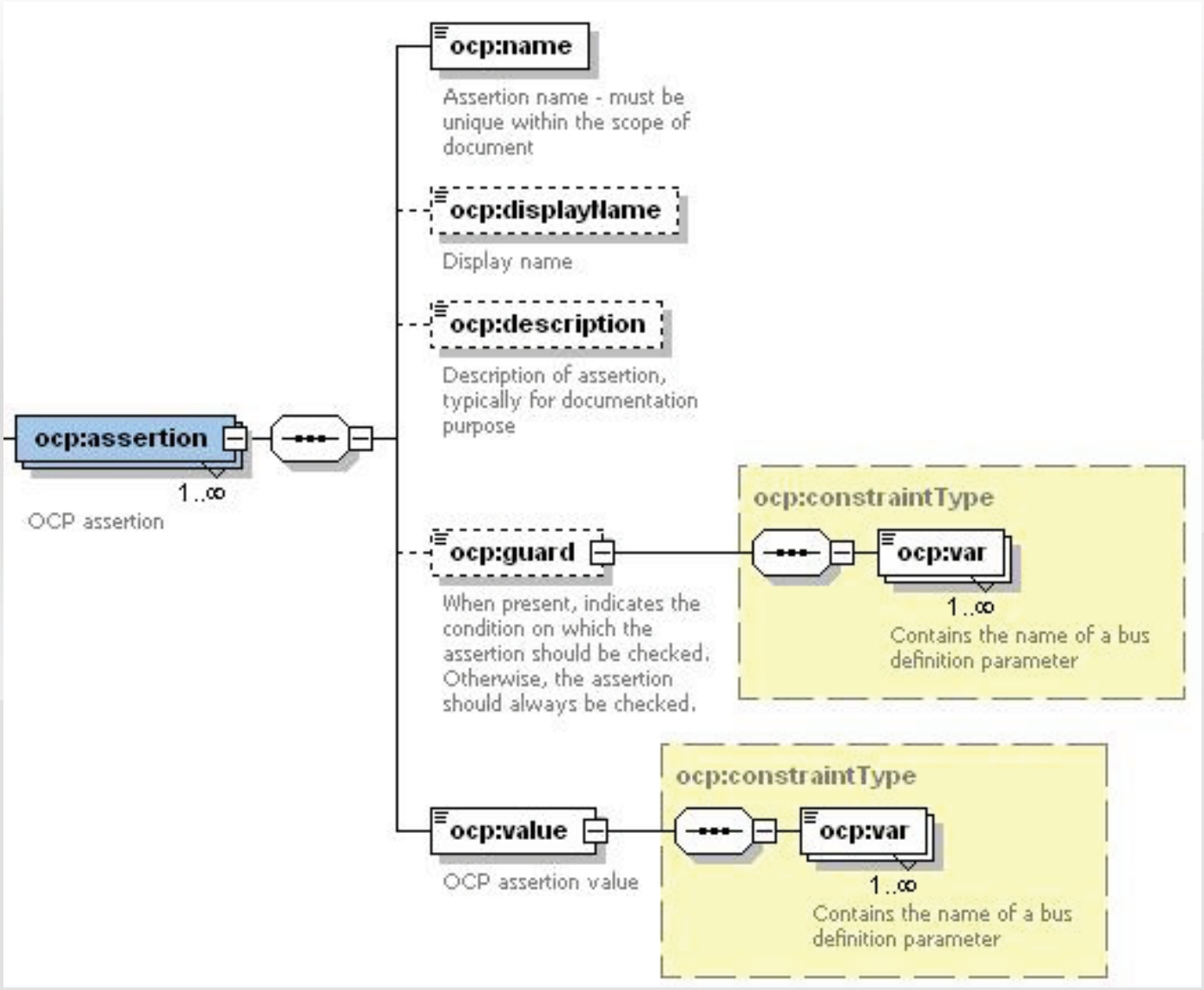
OCP Logical Port Definition (in abstractionDef)



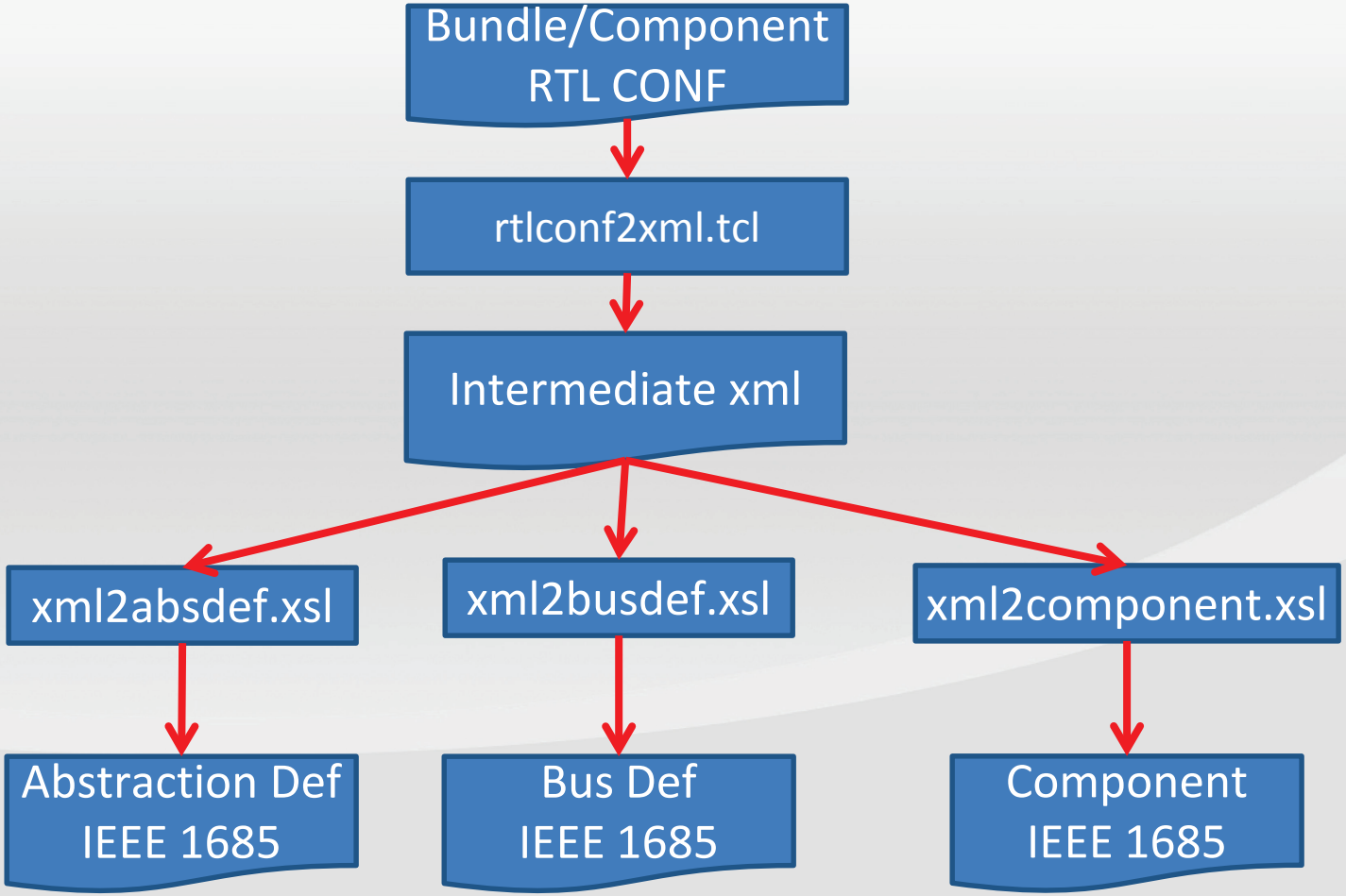
OCP Parameter Definition (in busDefinition)



OCP Parameter Assertions (in OCP parameter)



Support for RTL CONF to IEEE1685



IEEE1685-2009 Support for OCP

- **IEEE1685 Busdef & Abstractiondef for OCP2.2, OCP3.x**
- **Checkers**
  - OCP Interface configuration Checker
    - Format for OCP parameters
    - Port width constraint
    - Port Presence constraint
    - OCP Parameter assertions
  - OCP Interface Interoperability Checker
    - Master – Slave connection compatibility
- **rtl.conf ⇔ IP-XACT conversion to support migration**

One standard file per protocol version instead one abs/bus def per OCP configuration

Design-time Protocol Checkers

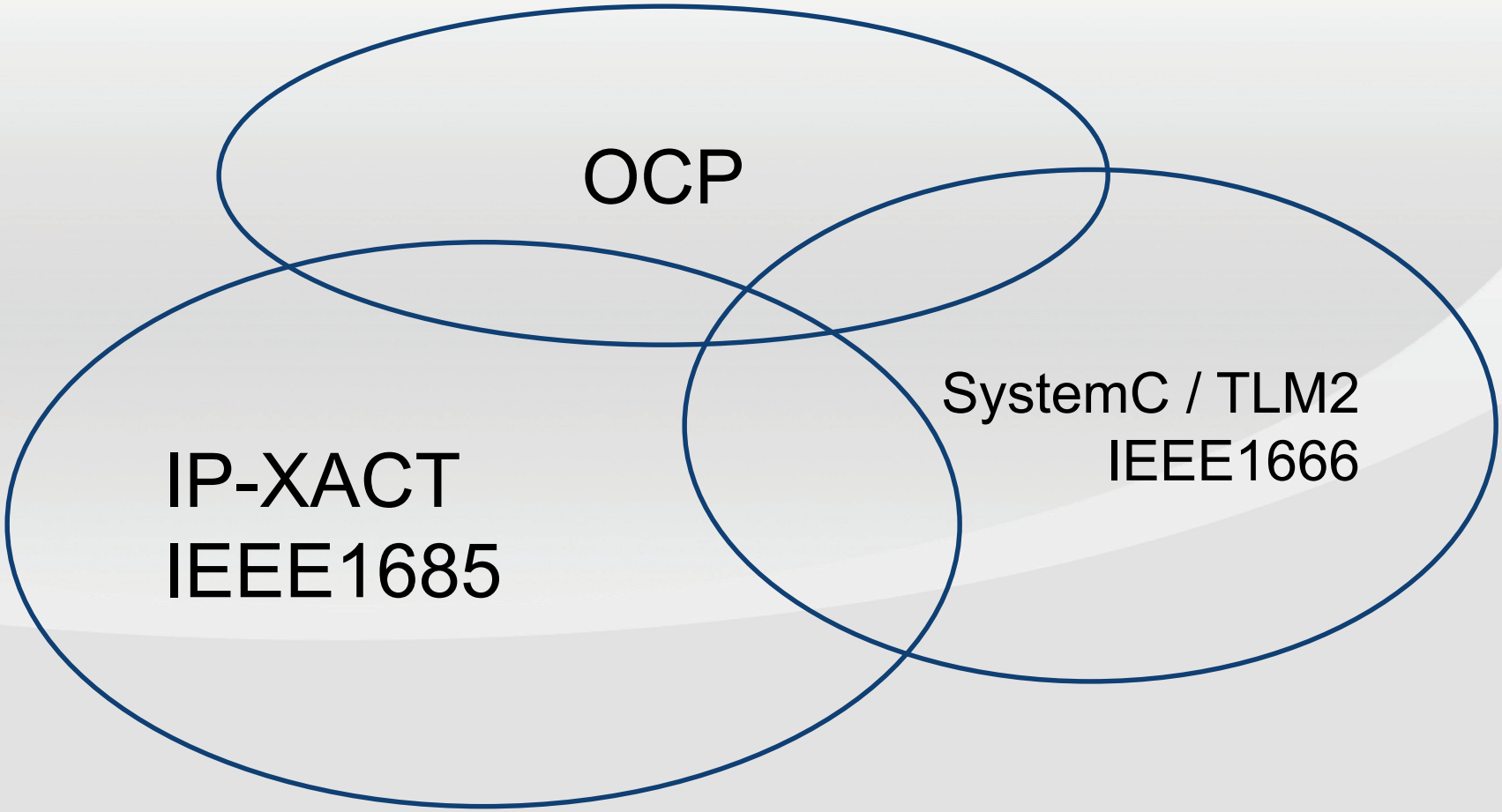
Legacy Support

IEEE1685-2014* Enhancements

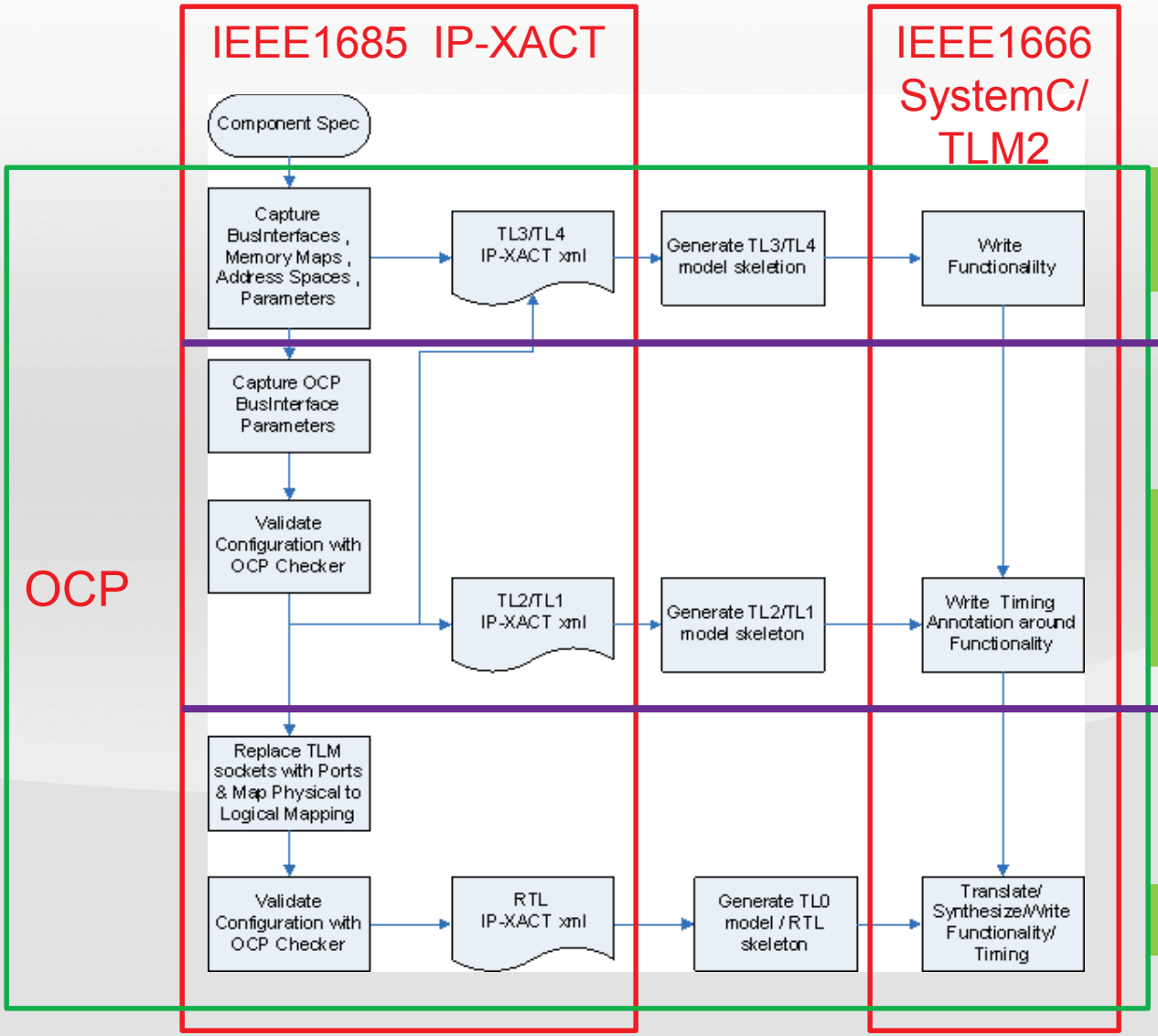
- Configurable Bus Interfaces
- Support for TLM2 sockets
- Multiple views of a component (RTL / TLM)

* Yet to release

Confluence of Standards



SystemC , IP-XACT & OCP

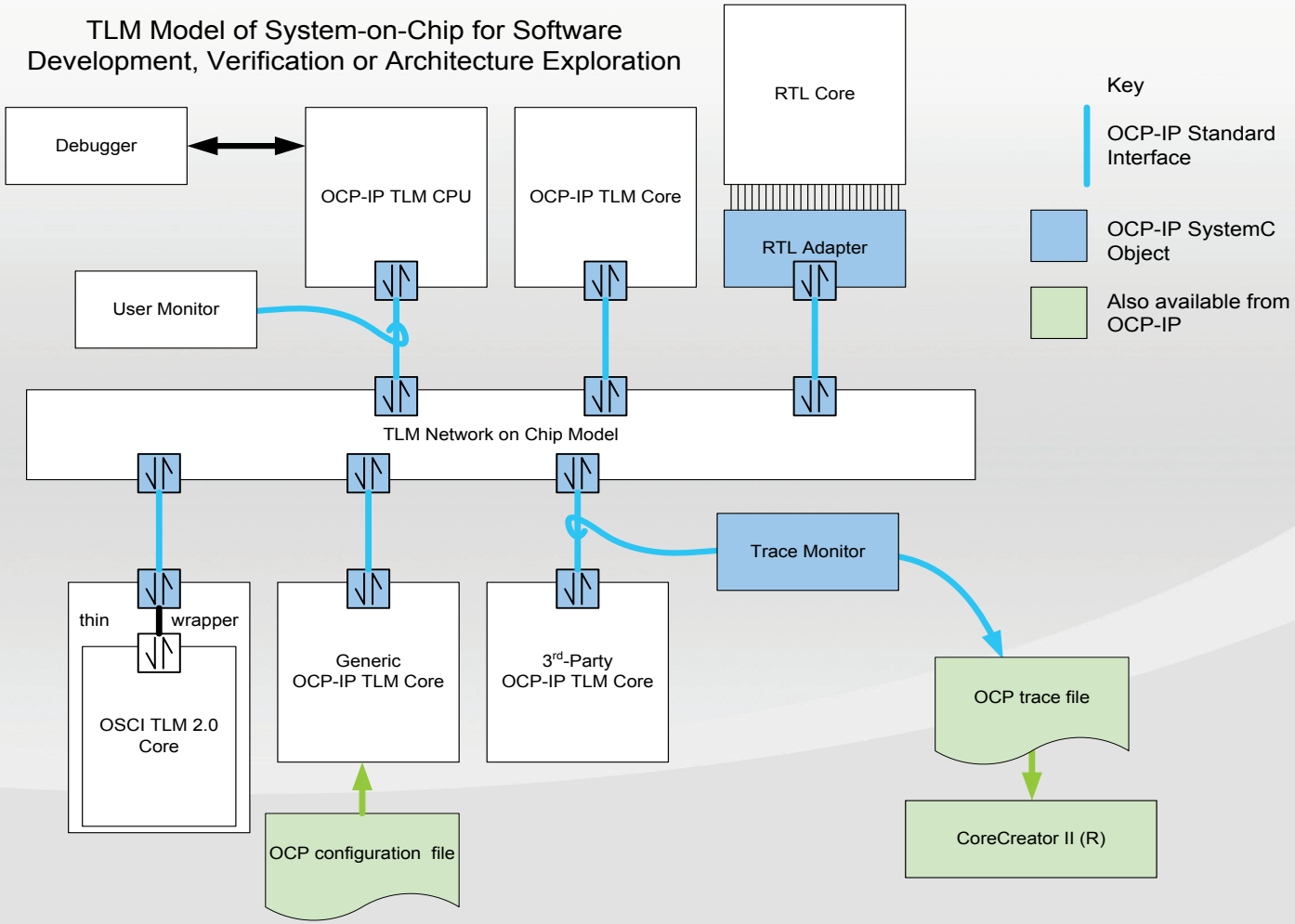


Virtual Platforms for Software Development

Cycle Accurate Platforms for Verification, Architecture Exploration

RTL

Platform Assembly

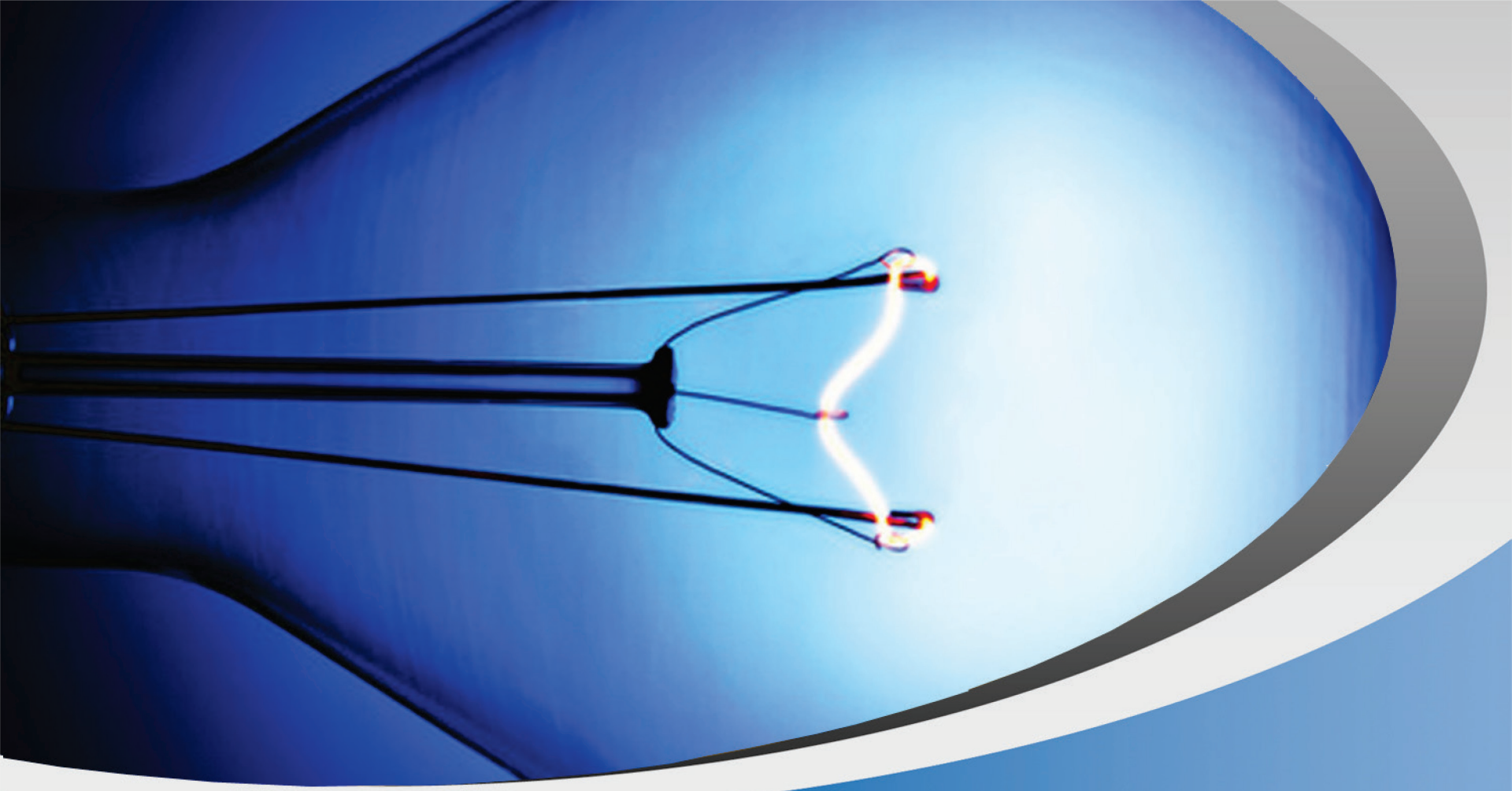


Assembly for multi abstraction models possible with IEEE1685-2009

Assembly for multi view (same component with multiple abstractions) would be possible in IEEE1685-2014

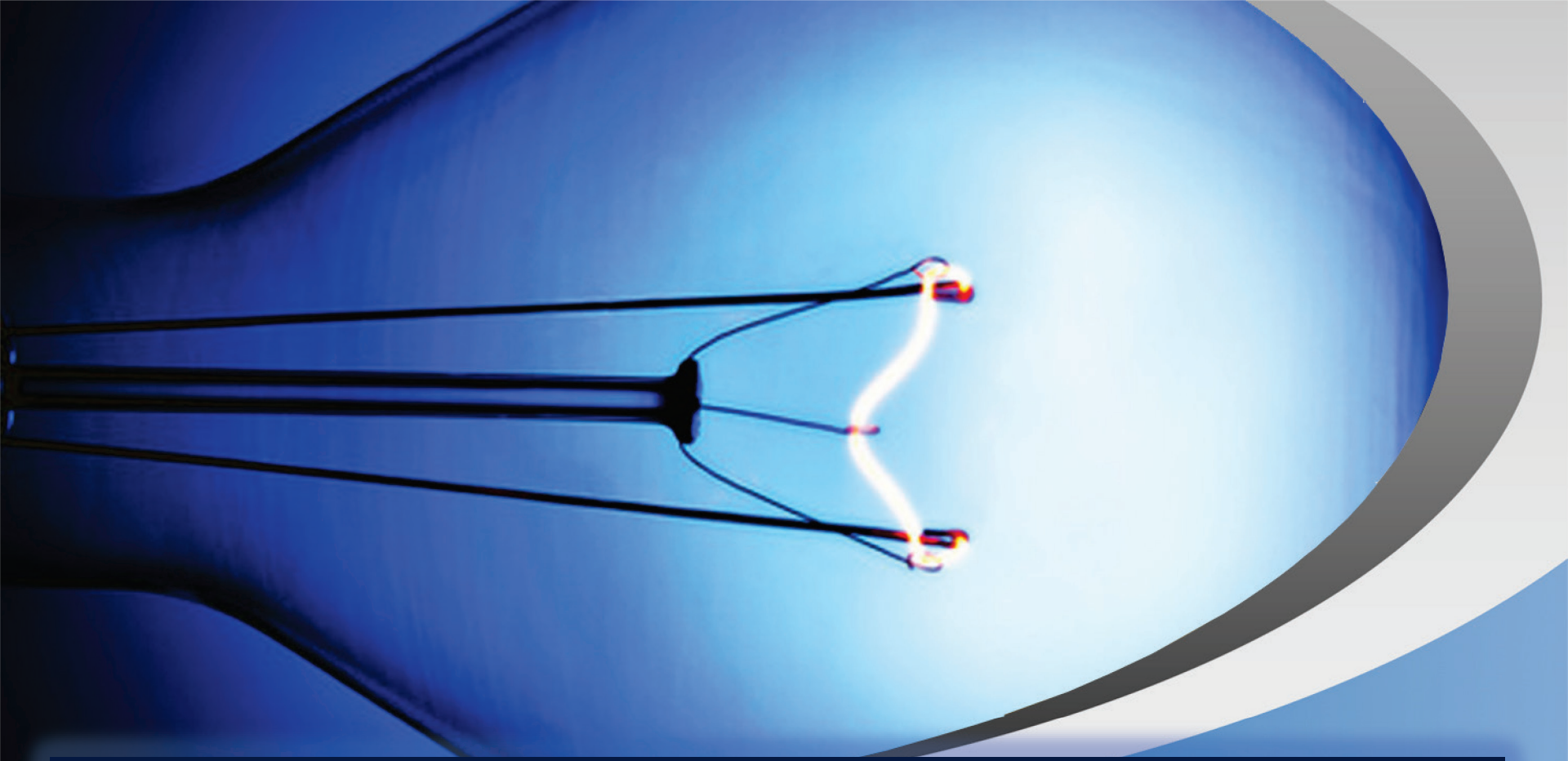
References

- ***OCP 3.0 Specification***
- ***Using OCP and Coherence Extensions to Support System-Level Cache Coherence*** **by Chien-Chun (Joe) Chou, Sonics, Inc.; Konstantinos Aisopos, EE Dept., Princeton University; David Lau, MIPS Technologies; Yasuhiko Kurosawa, Toshiba Corporation; and D. N. (Jay) Jayasimha, Sonics, Inc**
- **Spirit Consortium Documents** http://www.spiritconsortium.org/tech/docs/
- ***Viewpoint: Capture OCP systems in IP-XACT 1.4*** **by Stéphane Guntz** http://eetimes.eu/showArticle.jhtml?articleID=218101268&queryText=spec
- ***IPXACT_representation_of_OCP*** **by Stephane Guntz (Magillem Design Services), Christophe Amerijckx (STMicroelectronics), Pascal Chauvet /Kamil Synek (Sonics), Prashant Karandikar (Texas Instruments),Vesa Lahtinen (Nokia), Mark Noll (Synopsys)** http://www.ocpip.org/uploads/documents/IPXACT_representation_of_OCP_20081201.pps
- ***OCP TLM Kit practical implementation of TLM2.0*** http://iscug.in/pdf_download.php?file=1_OCP_SLD_ISCUG_2012.ppt



Thank you

accellera
SYSTEMS INITIATIVE



OCP: The Journey Continues

OCP Futures

Drew Wingard, Sonics, Inc.

accellera
SYSTEMS INITIATIVE

Current Status

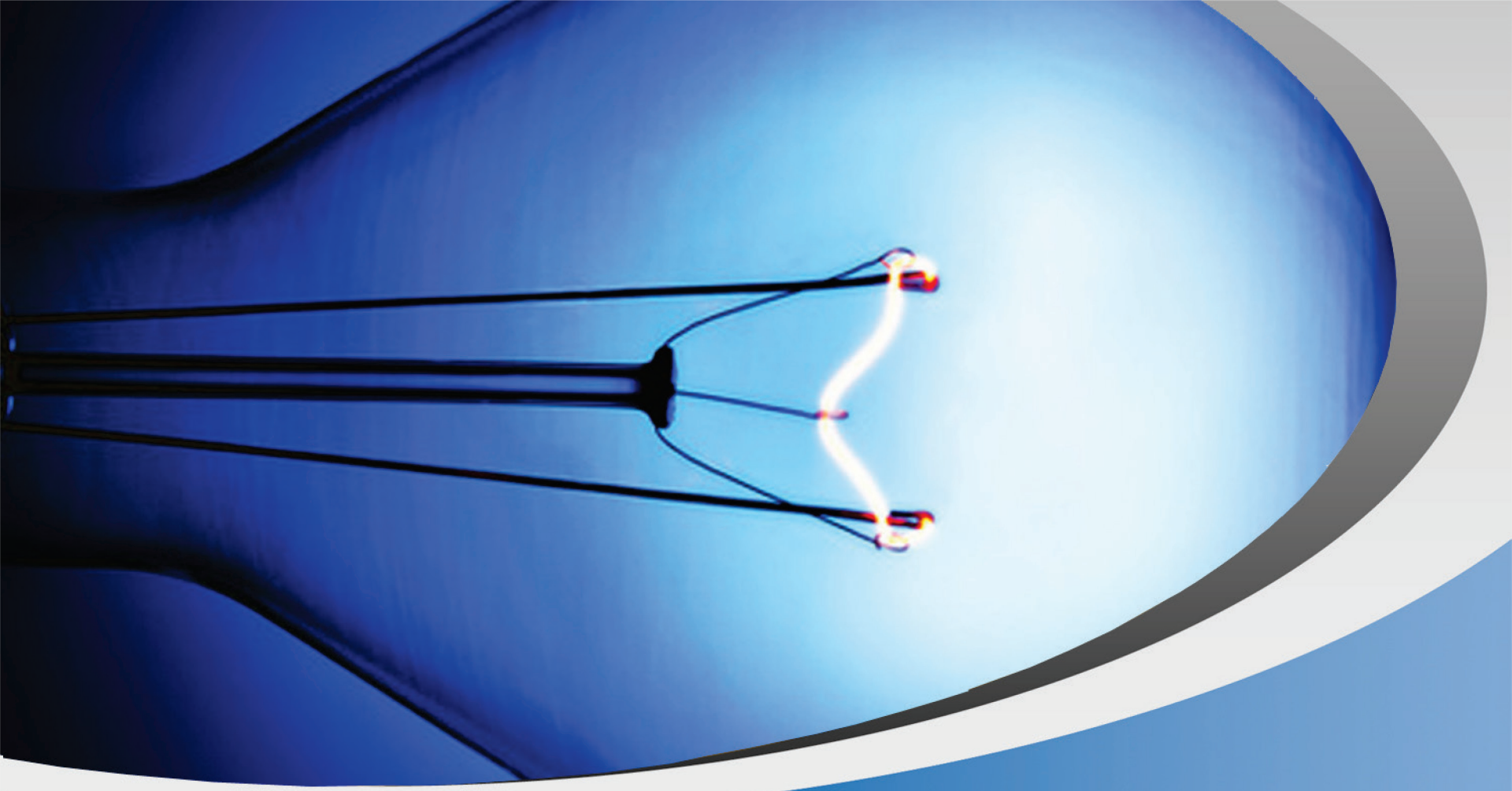
- **Specification of record: OCP 3.0**
  - New: cache coherence extensions
    - Proven for both snooping and directory-based system coherence
  - New: connection protocol
    - Enable master and slave to vote on when to safely disconnect (e.g., for power state changes)
- **OCP 3.1 completed Member Review before Accellera transfer**
  - New: compliance checks for coherence extensions
  - New: barrier transaction extensions
  - New: parameters to specify outstanding transaction counts
  - Transition from “rtl.conf” to IP-XACT for metadata
  - Awaiting release under Accellera procedures

Opportunities for Future Work

- **Add more performance-oriented parameters**
  - Minimum, maximum latency
  - Minimum, maximum throughput
  - Focus: enabling automated system analysis and formal verification
- **Create a serialized version of OCP**
  - Share signals between address/command and data
  - Keep same protocol semantics at transaction level
  - Provide better support for NoCs & 3D packages
- **Add power control interface**
  - Standardize power state control signaling
  - Include power management handshaking
- **If interested to participate, please contact me via Accellera:** ocp_specification-wg-chair@lists.accellera.org

Summary

- Open Core Protocol is the original, IP core-centric interface socket
- The high configurability of OCP allows it to span from very simple interfaces up through very complex ones, with many steps in between
- Excellent commercial IP and verification support exists for OCP from major EDA and IP suppliers
- OCP has strong support for Accellera TLM and IP-XACT standards



Thank you

accellera
SYSTEMS INITIATIVE