Introduction to OCP Protocol

Presented By Mitesh Khadgi

Date: 27/03/2025

Content

- Why verification protocol is needed?
- History and Evolution of OCP
- Problem and Solution
- Basics of OCP
- Ordering and Pipelining
- Sideband, Reset, Flags, and Test Signals
- SUPPORTING TOOLING
- Types of Encoding
- Future Opportunities and Summary

Why verification protocol is needed?

Why verification protocol is needed?

- Without a verification protocol, there is no clear standard for how to interact with the DUT, how data is transferred, or how to check whether the DUT responds correctly to various inputs.

- A protocol defines how components (like a master, slave, or communication bus) communicate with each other. For instance, protocols like AMBA, PCIe, or AXI need specific interactions between the transmitter and receiver, and a verification protocol ensures that these interactions are correctly modeled and verified.

Why verification protocol is needed? (continued…)

- A verification protocol is needed to:
- Ensure correct functionality of the design.
- Standardize the interactions between components, enabling integration and scalability.
- Automate the verification process, saving time and reducing human error.
- Enable reusability of testbenches across multiple designs.
- Detect errors and ensure compliance with timing and functional requirements.
- Support coverage-driven verification, ensuring complete and thorough testing.
- Enhance debugging and diagnostic capabilities, making issue resolution faster.
- Ensure compliance with industry standards and reduce risks associated with non-compliance.
- Accelerate time-to-market by streamlining the verification process and reducing design iterations.
- Verification protocols are the backbone of structured, effective, and scalable verification in hardware design. They not only ensure that a design functions correctly but also help in maintaining high-quality standards throughout the verification lifecycle.

History and Evolution of OCP

History of OCP

- The Open Core Protocol (OCP) was introduced in the early 2000s to provide a standardized communication framework for integrating different IP cores in System-on-Chip (SoC) designs. It was first released in 2003, with subsequent versions (2.0, 2.1, 3.0, 4.0) enhancing scalability, performance, and support for high-speed and multi-core systems. The protocol's goal was to simplify the design process by providing a flexible and reusable standard for communication between cores. Over time, it evolved to support complex, high-performance systems and has become integral to the development of modern SoCs, focusing on improving interconnects and data transfers.

Evolution of OCP

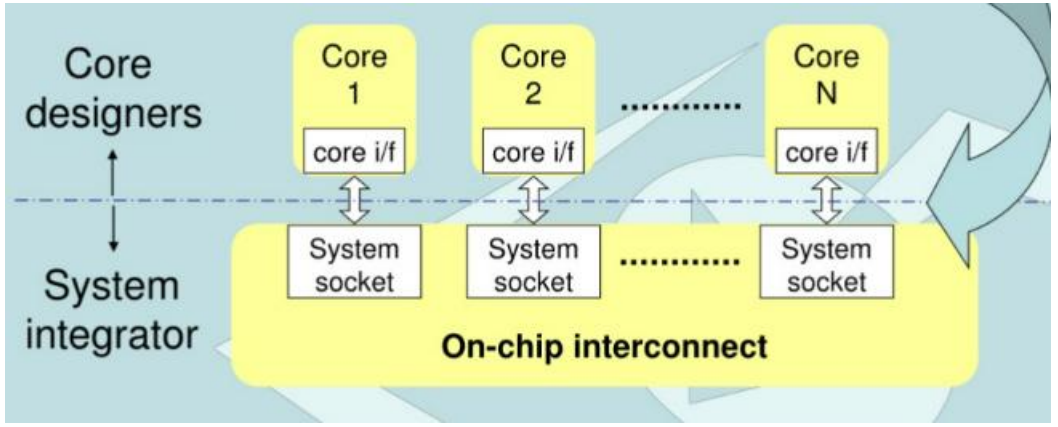
- The evolution of the Open Core Protocol (OCP) reflects the growing complexity of System-on-Chip (SoC) designs and the need for efficient, standardized communication between IP cores. Here's a brief overview of its evolution:

1. OCP 1.0 (2003)
    - Introduction: OCP 1.0 was the first version, designed to address the growing need for a standardized communication protocol between various IP cores in SoCs.
    - Features: Basic master-slave communication model, simple handshaking mechanism for data transfer.
2. OCP 2.0 (2004)
    - Enhancements: Introduced burst mode support, allowing multiple data units to be transferred in one transaction.
    - Features: Enhanced data transfer efficiency, flow control mechanisms, basic error detection.
3. OCP 2.1 (2005)
    - Improvements: Increased scalability and support for high-throughput systems.
    - Features: Introduced master-to-master communication, more advanced error handling, and additional error detection mechanisms.
4. OCP 3.0 (2007)
    - Major Overhaul: OCP 3.0 provided significant improvements for high-performance systems and multi-core designs.
    - Features: Improved transaction-level modeling, better synchronization, and enhanced error correction. Optimized for large, complex SoCs.
5. OCP 4.0 (2012)
    - Optimizations for Advanced Systems: This version focused on improving scalability and power efficiency.
    - Features: Support for high-bandwidth memory, low-power optimizations, and high-speed data transfers. Enhanced transaction ordering and priority control.
6. OCP 4.1 (2016)
    - Refinement: Minor updates focused on improved compatibility and performance.
    - Features: Enhanced error handling and integration with new interconnect technologies, such as PCIe and Ethernet.

Problem and Solution

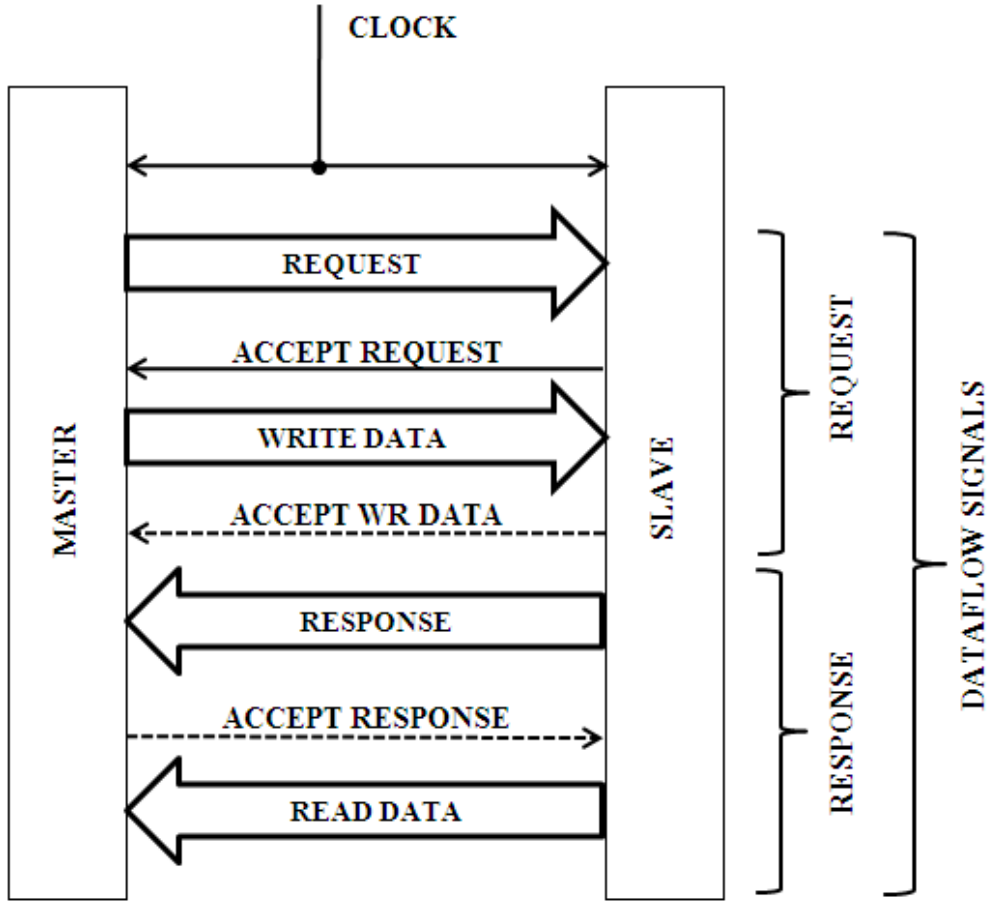
Problem & Solution

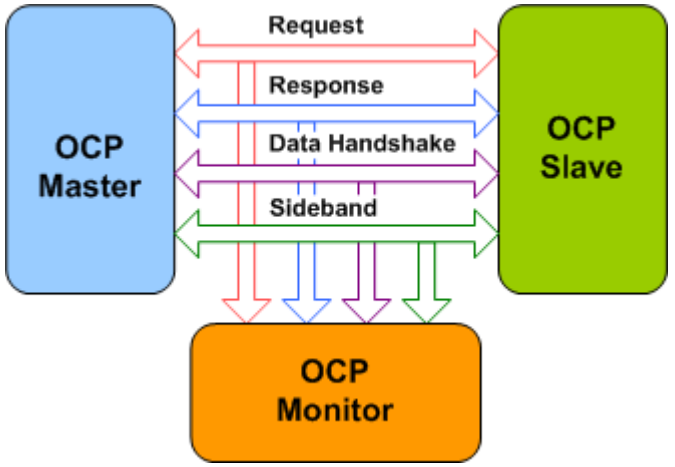
- SoC designers want to reuse IP cores to shorten development schedules.
- Problem:
- IP cores need to be re-adapted into each system design
- - 1000s of cores, 10s or 100s of interconnects/buses
- - must understand lots of interfaces
- - overwhelming amount of adaptation work
- Motivation:
- reuse without rework
- - Plug-and-Play: between cores and interconnects systems from different sources.
- Solution:
- Core/System Protocol
- What is needed is a standard, well-defined protocol for cores to talk to a system interconnect.

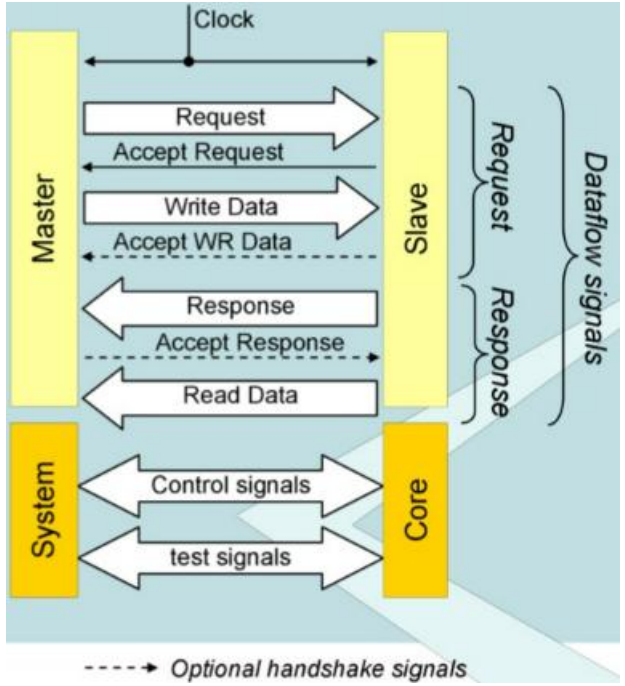


Basics of OCP

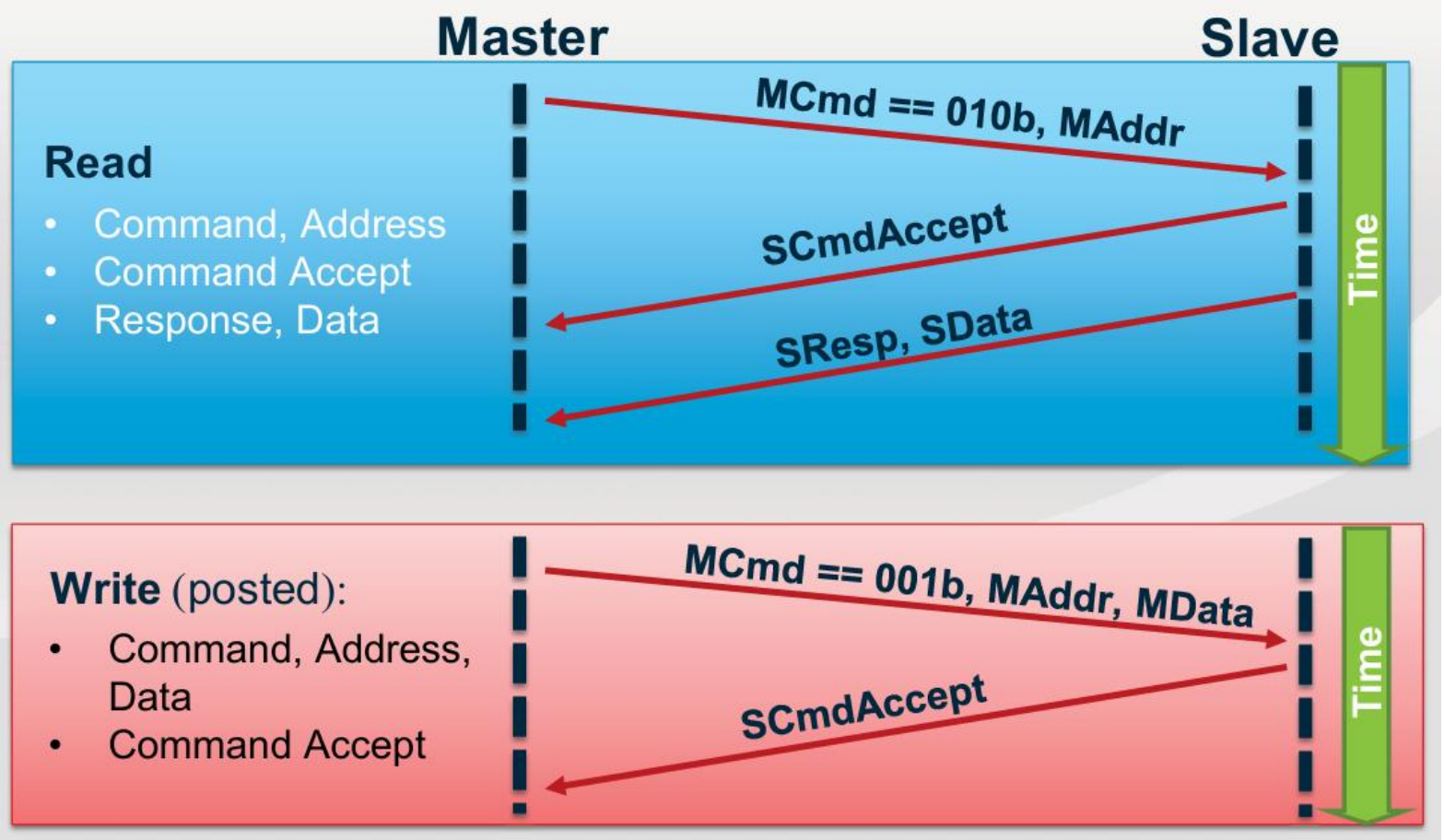
Basic OCP Interface



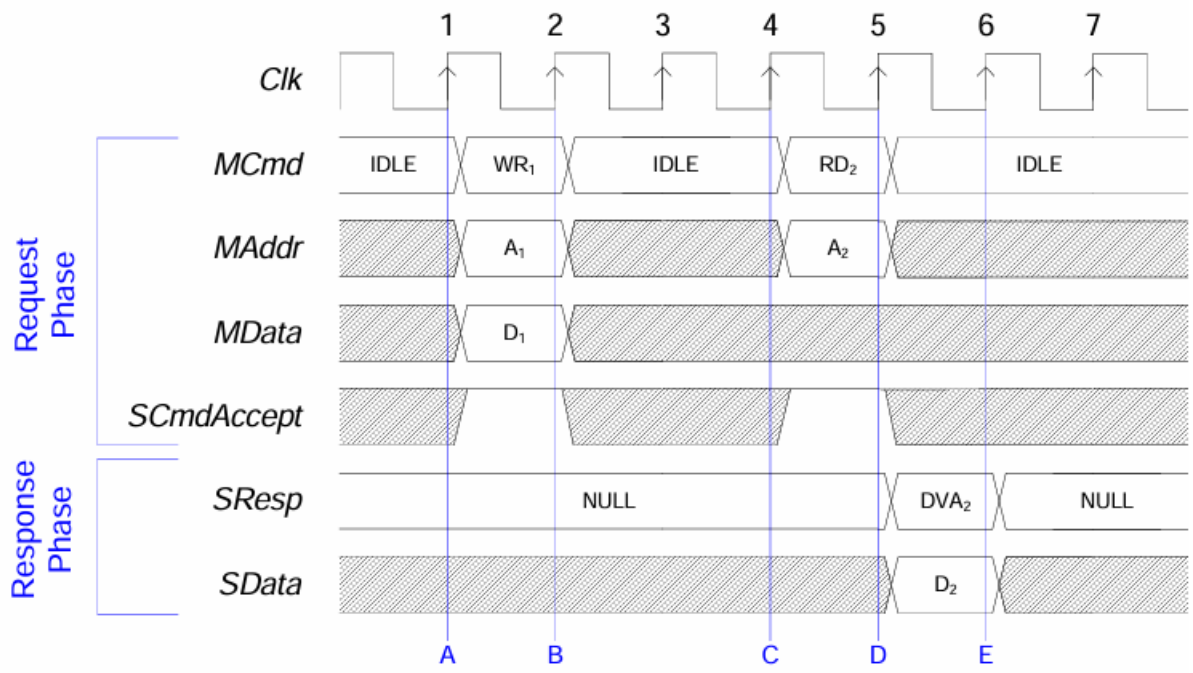




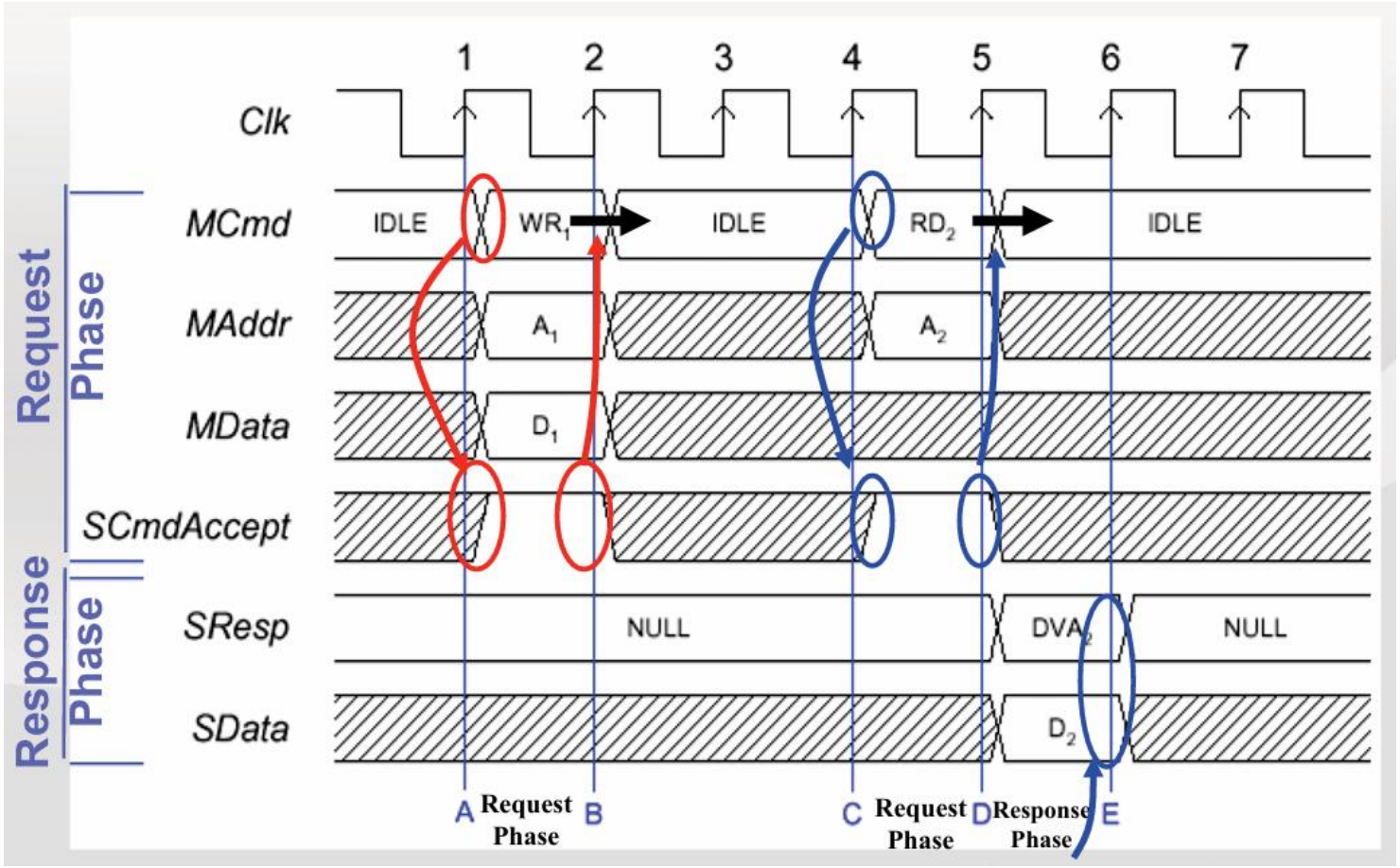
Phases of simple transfers: RD, WR



Simple Write and Read Transfer



Simple Write & Read Transfer



Data Accepted Automatically

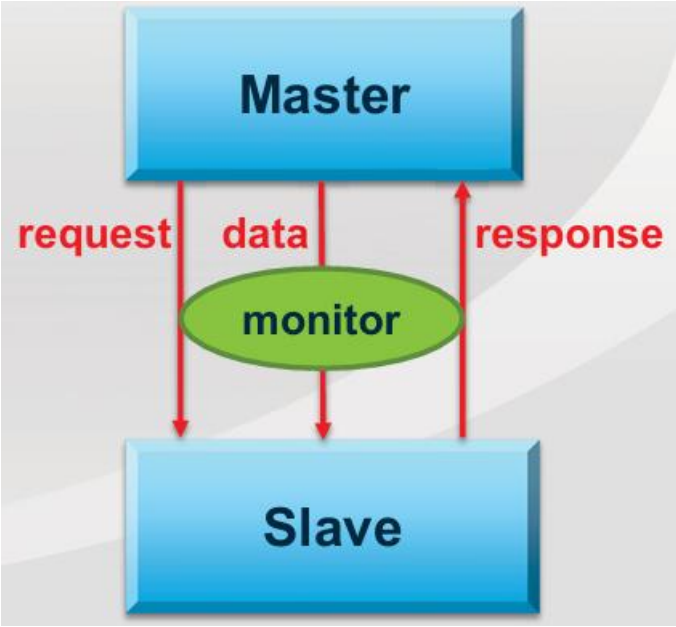
Basic OCP Signals

Name	Width	Driver	Function
Clk	1	varies	Clock input
EnableClk	1	varies	Enable OCP clock
MAddr	configurable	master	Transfer address
MCmd	3	master	Transfer command
MData	configurable	master	Write data
MDataValid	1	master	Write data valid
MRespAccept	1	master	Master accepts response
SCmdAccept	1	slave	Slave accepts transfer
SData	configurable	slave	Read data
SDataAccept	1	slave	Slave accepts write data
SResp	2	slave	Transfer response

These signals allow the master and slave components to communicate efficiently and handle various types of data transactions, such as reads, writes, bursts, and error management, making the Open Core Protocol an effective tool for interconnecting IP cores in complex SoCs.

Signal Groups

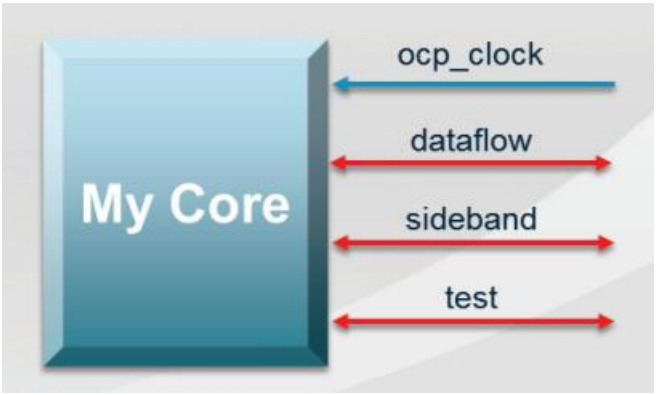
- OCP avoids phase interference by supporting each phase with its own independent set of signals (including flow control)
- - Request phase
- - Data phase signal group
- - Response phase signal group
- Arrows only show phase initiation direction
- - Each phase may have both inputs and outputs



OCP Transaction Types

- One or more transfers build a transaction
- Bursts
  - Reads
- - Writes: Posted (with or without response), and Non-posted
- - Broadcast
- Semaphores
- - Read-modify-write: Blocking, and 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

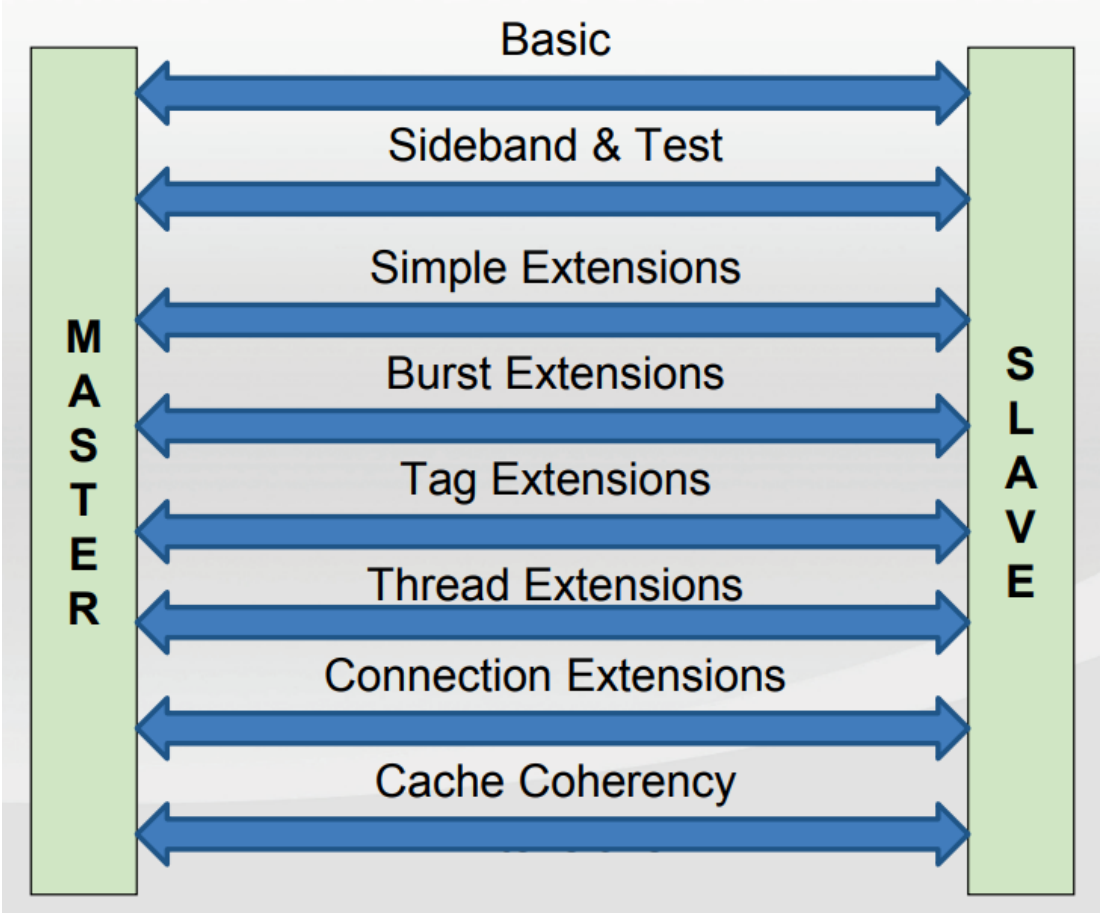
OCP - A Complete Socket



- 3 specs in 1: separate groups of signals
- - Dataflow: transfer of data with RD or WR
- - Sideband (control and status): Miscellaneous communication among cores and Interrupt, Flags, Errors, etc.
- - Test (scan, jtag, etc.)

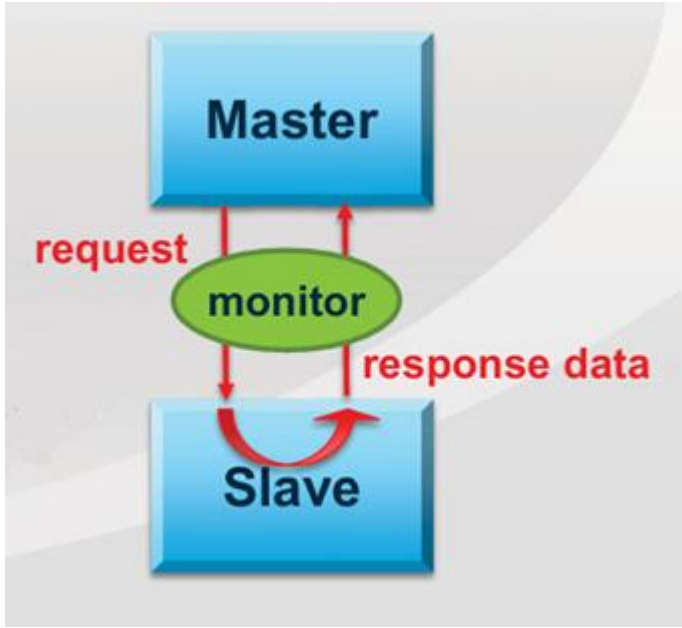
- All communication is synchronous to a clock input to the core

OCP Configurability



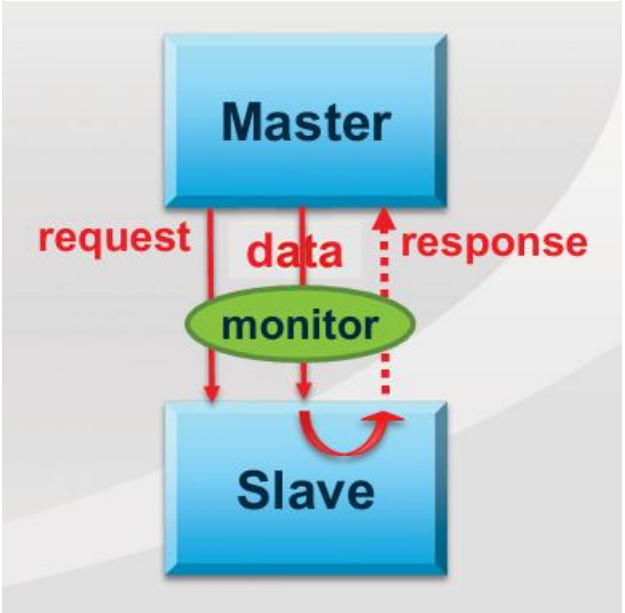
Note: Except Basic Signal Group, all other extensions are optional

OCP Read Transfer



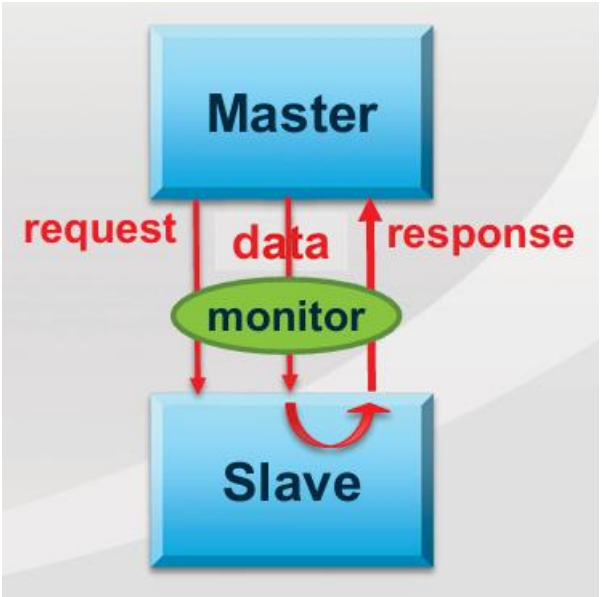
- Master sends a request to slave
- - Designates Read command
- - Specifies an address
- Slave returns a response to master
- - Indicates success or failure
- - Includes copy of the addressed data, if successful

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 monitor Slave
- 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 Master request response data monitor
- 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)

The Valid signals: MCmd, MDataValid and SResp

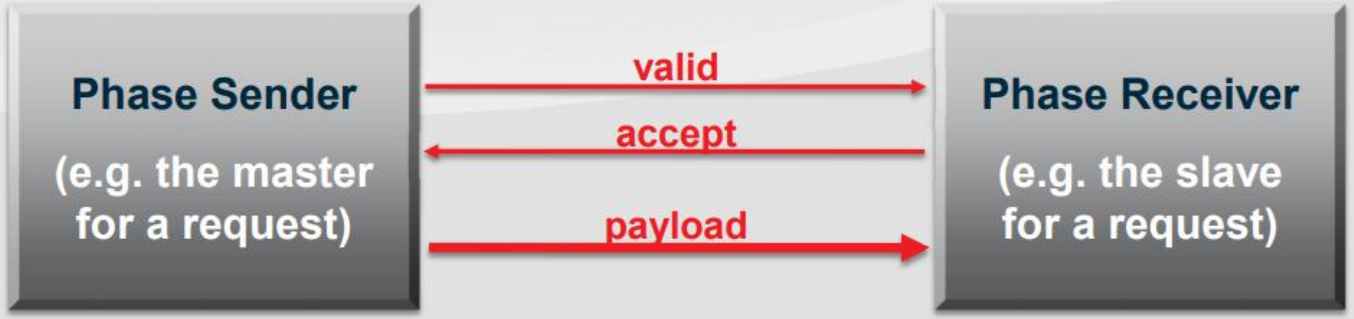
MCmd[2:0]	Command
000	IDLE
001	WRITE
010	READ
011	READEX
100	READ-LINKED
101	WRITE-NON-POST
110	WRITE-CONDITIONAL
111	BROADCAST

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

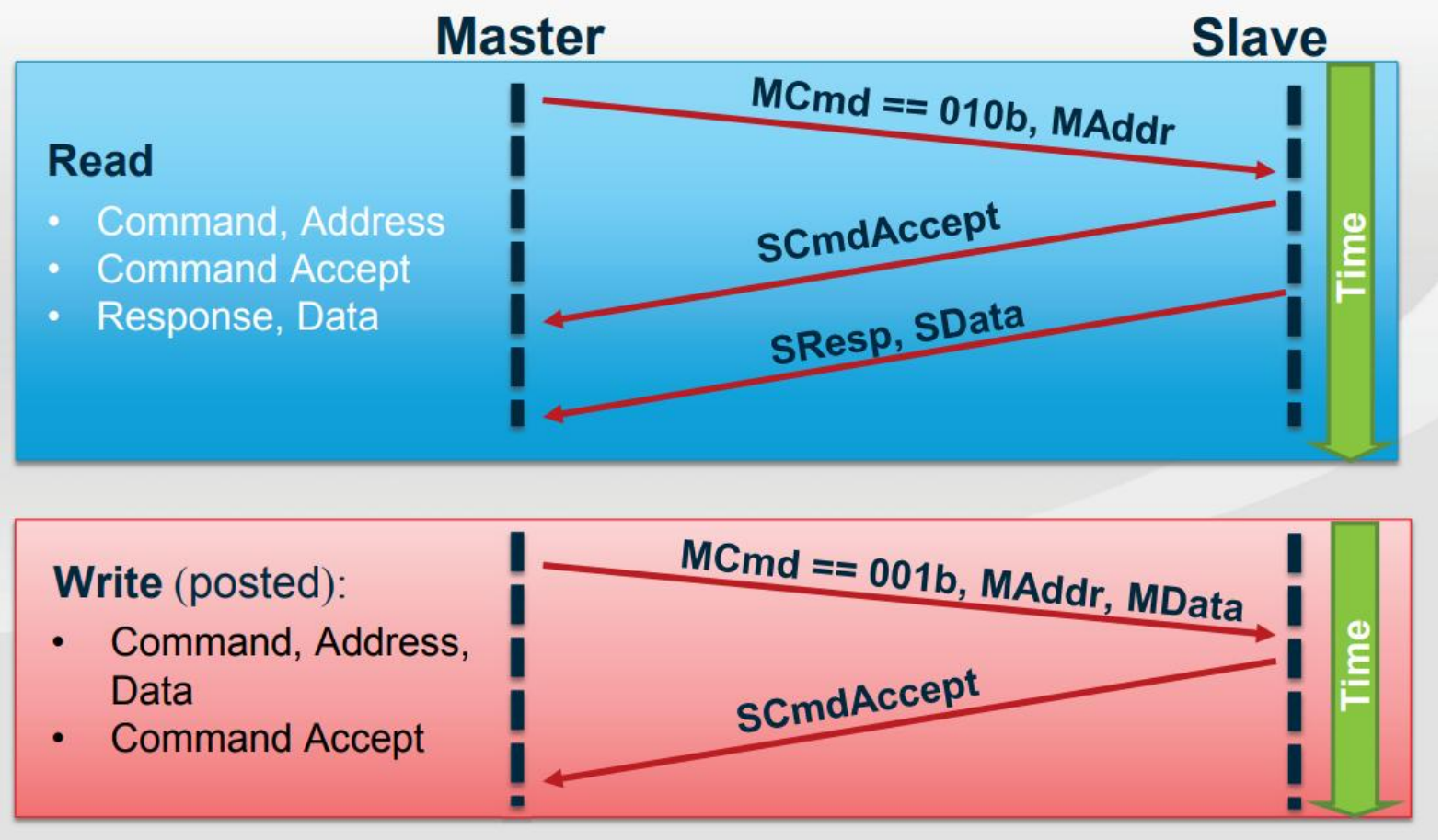
MDataValid	Write Data Phase
0	Write Data not Valid
1	Write Data Valid

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



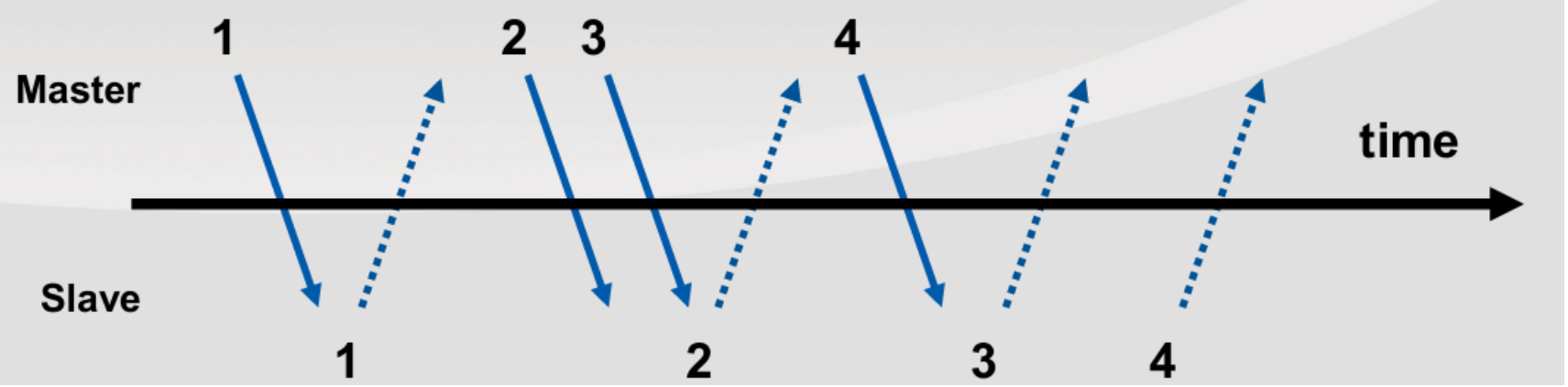
Phases of simple transfers: RD, WR



Ordering and Pipelining

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

- 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
- -- Single Request Multiple Data (packet) style
- -- Precise length without early termination
- - MRMD: Precise
- -- Precise length without early termination
- -- Multiple Request Multiple Data
- - MRMD: Imprecise
- -- Speculative length
- -- Since it's imprecise, it must be MRMD style
- -- Possible early termination
- ---- Network application detecting packet CRC error
- Burst sequence types
- - INCR, WRAP, STRM ….

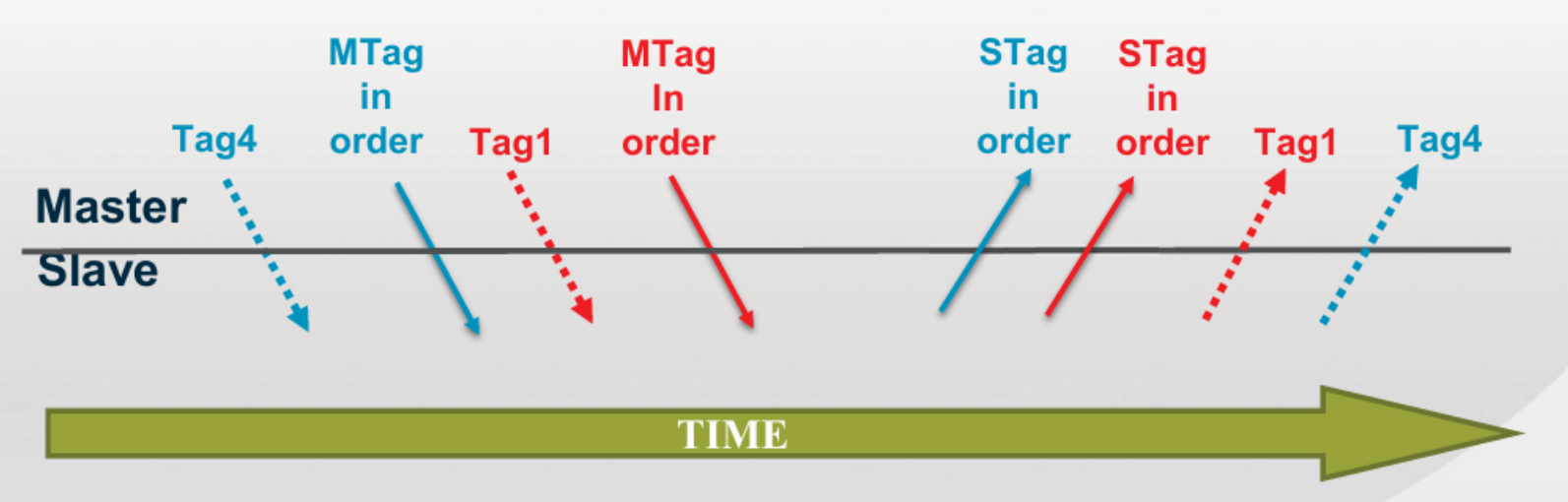
Bursts in OCP

- 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) – no-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.)

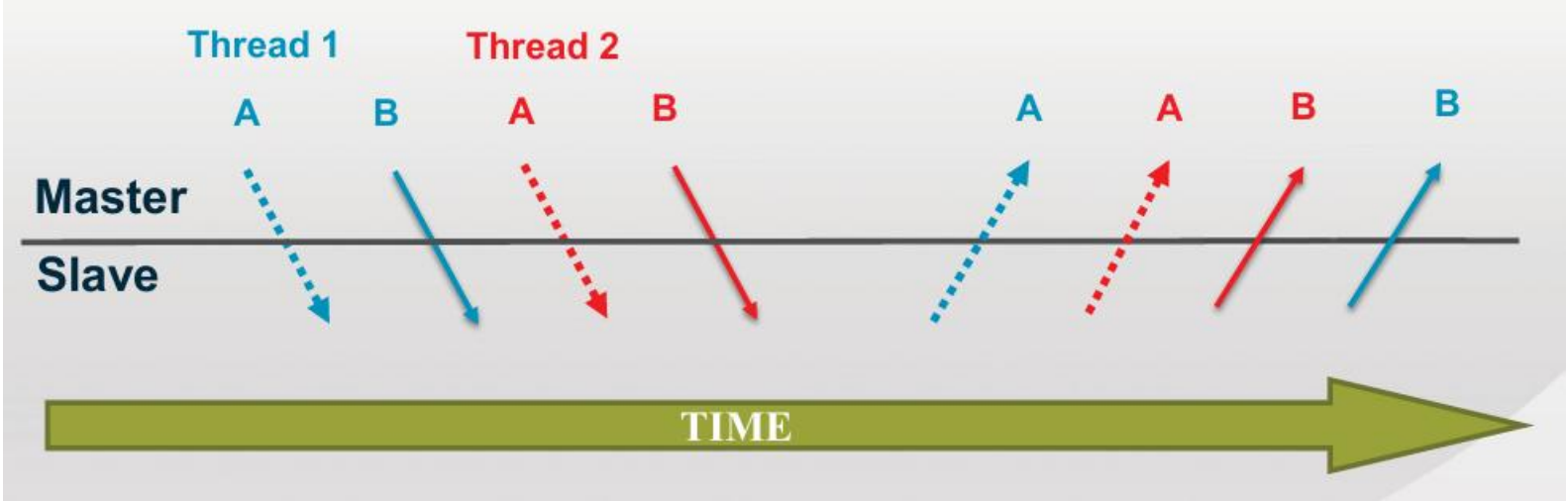
- Optional framing information
- - MReqLast, SRespLast, MDataLast indicate the end of a burst
- - If burst is imprecise, last transfer is indicated by MBurstLength=1
- - If burst is precise, use it to terminate a transaction without counting and, save some area
- If an initiator may change its burst 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

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

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

Sideband, Reset, Flags, and Test Signals

Sideband Signals

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

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)
- - ORcan be driven by the slave (SReset_n)
- - ORcan 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

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

Other Commercial Providers of OCP Verification Technology

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

Types of Encoding

Command Encoding

MCmd[2:0]			Command	Mnemonic	Request Type
0	0	0	Idle	IDLE	(none)
0	0	1	Write	WR	write
0	1	0	Read	RD	read
0	1	1	ReadEx	RDEX	read
1	0	0	ReadLinked	RDL	read
1	0	1	WriteNonPost	WRNP	write
1	1	0	WriteConditional	WRC	write
1	1	1	Broadcast	BCST	write

Response Encoding

SResp[1:0]		Response	Mnemonic
0	0	No response	NULL
0	1	Data valid / accept	DVA
1	0	Request failed	FAIL
1	1	Response error	ERR

Simple OCP Extensions

Name	Width	Driver	Function
MAddrSpace	configurable	master	Address space
MByteEn	configurable	master	Request phase byte enables
MDataByteEn	configurable	master	Datahandshake phase write byte enables
MDataInfo	configurable	master	Additional information transferred with the write data
MReqInfo	configurable	master	Additional information transferred with the request
SDataInfo	configurable	slave	Additional information transferred with the read data
SRespInfo	configurable	slave	Additional information transferred with the response

OCP Burst Extensions

Name	Width	Driver	Function
MAtomicLength	configurable	master	Length of atomic burst
MBlockHeight	configurable	master	Height of 2D block burst
MBlockStride	configurable	master	Address offset between 2D block rows
MBurstLength	configurable	master	Burst length
MBurstPrecise	1	master	Given burst length is precise
MBurstSeq	3	master	Address sequence of burst
MBurstSingleReq	1	master	Burst uses single request/ multiple data protocol
MDataLast	1	master	Last write data in burst
MDataRowLast	1	master	Last write data in row
MReqLast	1	master	Last request in burst
MReqRowLast	1	master	Last request in row
SRespLast	1	slave	Last response in burst
SRespRowLast	1	slave	Last response in row

MBurstSeq Encoding

MBurstSeq[2:0]			Burst Sequence	Mnemonic
0	0	0	Incrementing	INCR
0	0	1	Custom (packed)	DFLT1
0	1	0	Wrapping	WRAP
0	1	1	Custom (not packed)	DFLT2
1	0	0	Exclusive OR	XOR
1	0	1	Streaming	STRM
1	1	0	Unknown	UNKN
1	1	1	2-dimensional Block	BLCK

OCP Tag Extensions

Name	Width	Driver	Function
MDataTagID	configurable	master	Ordering tag for write data
MTagID	configurable	master	Ordering tag for request
MTagInOrder	1	master	Do not reorder this request
STagID	configurable	slave	Ordering tag for response
STagInOrder	1	slave	This response is not reordered

OCP Thread Extensions

Name	Width	Driver	Function
MConnID	configurable	master	Connection identifier
MDataThreadID	configurable	master	Write data thread identifier
MThreadBusy	configurable	master	Master thread busy
MThreadID	configurable	master	Request thread identifier
SDataThreadBusy	configurable	slave	Slave write data thread busy
SThreadBusy	configurable	slave	Slave request thread busy
SThreadID	configurable	slave	Response thread identifier

Sideband OCP Signals

Name	Width	Driver	Function
MConnect	2	master	Master connection state
MError	1	master	Master Error
MFlag	configurable	master	Master flags
MReset_n	1	master	Master reset
SConnect	1	slave	Slave connection vote
SError	1	slave	Slave error
SFlag	configurable	slave	Slave flags
SInterrupt	1	slave	Slave interrupt
SReset_n	1	slave	Slave reset
SWait	1	slave	Slave delays connection change
ConnectCap	1	tie-off	Connection capability tie-off
Control	configurable	system	Core control information
ControlBusy	1	core	Hold control information
ControlWr	1	system	Control information has been written
Status	configurable	core	Core status information
StatusBusy	1	core	Status information is not consistent
StatusRd	1	system	Status information has been read

Connection State Encoding

MConnect[1:0]		State	Mnemonic	Connected?
0	0	Disconnected by master	M_OFF	No
0	1	Waiting to transition	M_WAIT	Matches prior state
1	0	Disconnected by slave	M_DISC	No
1	1	Connected	M_CON	Yes

Slave Connection Vote Encoding

SConnect	Connection Vote	Mnemonic
0	Vote to disconnect	S_DISC
1	Vote to connect	S_CON

Slave Connection Change Delay Encoding

SWait	Function	Mnemonic
0	Allow connection status change	S_OK
1	Delay connection status change	S_WAIT

MReset Signal

MReset_n	Function
0	Reset Active
1	Reset Inactive

SReset Signal

SReset_n	Function
0	Reset Active
1	Reset Inactive

Test OCP Signals

Name	Width	Driver	Function
Scanctrl	configurable	system	Scan control signals
Scanin	configurable	system	Scan data in
Scanout	configurable	core	Scan data out
ClkByp	1	system	Enable clock bypass mode
TestClk	1	system	Test clock
TCK	1	system	Test clock
TDI	1	system	Test data in
TDO	1	core	Test data out
TMS	1	system	Test mode select
TRST_N	1	system	Test reset

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	n/a
Simple	MAddrSpace	addrspace	addrspace_wdth	0
	MByteEn[3]	byteen	data_wdth	all 1s
	MDataByteEn[4]	mdatabyteen	data_wdth	all 1s
	MDataInfo	mdatainfo	mdatainfo_wdth[5]	0
	MReqInfo	reqinfo	reqinfo_wdth	0
	SDataInfo[1]	sdatainfo	sdatainfo_wdth[6]	0
	SRespInfo[1]	respinfo	respinfo_wdth	0

OCP Signal Configuration Parameters (continued…)

Group	Signal	Parameter to add signal to interface	Parameter to control width	Default Tie-off
Burst	MAtomicLength[7]	atomiclength	atomiclength_wdth	1
	MBlockHeight[7,8]	blockheight	blockheight_wdth[9]	1
	MBlockStride[7,8]	blockstride	blockstride_wdth	0
	MBurstLength	burstlength	burstlength_wdth[10]	1
	MBurstPrecise[7, 11]	burstprecise	Fixed	1
	MBurstSeq[7]	burstseq	Fixed	INCR
	MBurstSingleReq[7, 12]	burstsinglereq	Fixed	0
	MDataLast[7, 13]	datalast	Fixed	n/a
	MDataRowLast[7, 8, 13, 14]	datarowlast	Fixed	n/a
	MReqLast[7]	reqlast	Fixed	n/a
	MReqRowLast[7, 8, 15]	reqrowlast	Fixed	n/a
	SRespLast[1, 7]	resplast	Fixed	n/a
	SRespRowLast[1, 7, 8, 16]	resprowlast	Fixed	n/a
Tag	MDataTagID[17]	tags>1 and datahandshake	tags	0
	MTagID	tags>1	tags	0
	MTagInOrder[18]	taginorder	Fixed	0
	STagID	tags>1 and resp	tags	0
	STagInOrder[19]	taginorder and resp	Fixed	0
Thread	MConnID	connid	connid_wdth	0
	MDataThreadID	threads>1 and datahandshake	threads	0
	MThreadBusy[1, 20]	mthreadbusy	threads	0
	MThreadID	threads>1	threads	0
	SDataThreadBusy[21]	sdatathreadbusy	threads	0
	SThreadBusy[22]	sthreadbusy	threads	0
	SThreadID	threads>1 and resp	threads	0

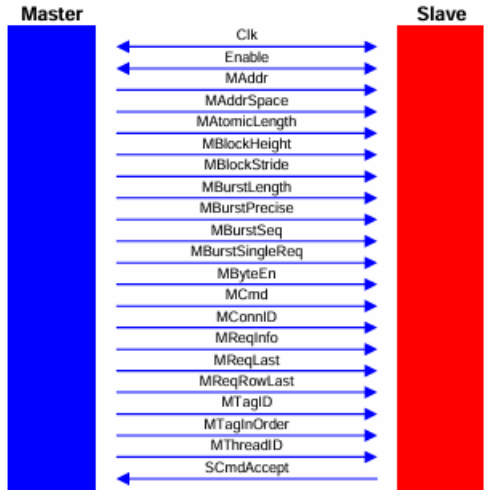
OCP Signal Configuration Parameters (continued…)

Group	Signal	Parameter to add signal to interface	Parameter to control width	Default Tie-off
Sideband	ConnectCap	connection	Fixed	n/a
	Control	control	control_wdth	0
	ControlBusy[23]	controlbusy	Fixed	0
	ControlWr[24]	controlwr	Fixed	n/a
	MConnect[25]	connection	2	M_CON
	MError	merror	Fixed	0
	MFlag	mflag	mflag_wdth	0
	MReset_n	mreset	Fixed	1
	SConnect[25]	connection	1	S_CON
	SError	serror	Fixed	0
	SFlag	sflag	sflag_wdth	0
	SInterrupt	interrupt	Fixed	0
	SReset_n	sreset	Fixed	1
	Status	status	status_wdth	0
	StatusBusy[26]	statusbusy	Fixed	0
	StatusRd[27]	statusrd	Fixed	n/a
	SWait[25]	connection	1	S_OK

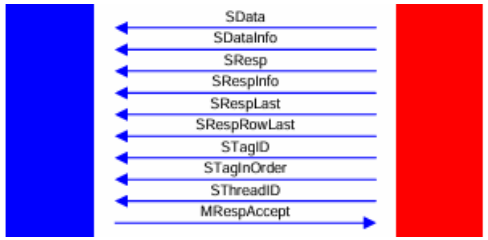
OCP Signal Configuration Parameters (continued…)

Group	Signal	Parameter to add signal to interface	Parameter to control width	Default Tie-off
Test	ClkByp	clkctrl_enable	Fixed	n/a
	Scanctrl	scanport	scanctrl_wdth	n/a
	Scanin	scanport	scanport_wdth	n/a
	Scanout	scanport	scanport_wdth	n/a
	TCK	jtag_enable	Fixed	n/a
	TDI	jtag_enable	Fixed	n/a
	TDO	jtag_enable	Fixed	n/a
	TestClk	clkctrl_enable	Fixed	n/a
	TMS	jtag_enable	Fixed	n/a
	TRST_N[28]	jtagtrst_enable	Fixed	n/a

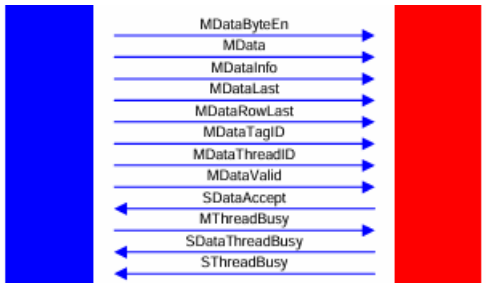
OCP Signal Summary (Request – Data Flow)



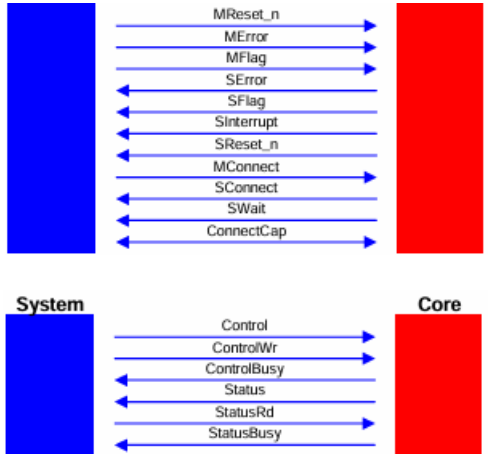
OCP Signal Summary (Response – Data Flow)



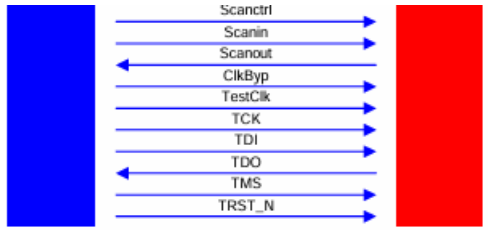
OCP Signal Summary (Data Handshake – Data Flow)



OCP Signal Summary (Sideband)



OCP Signal Summary (Test)



OCP Signal Groups

Group	Signal	Condition
Request	MAddr	always
	MAddrSpace	always
	MAtomicLength	always
	MBlockHeight	always
	MBlockStride	always
	MBurstLength	always
	MBurstPrecise	always
	MBurstSeq	always
	MBurstSingleReq	always
	MByteEn	always
	MCmd	always
	MConnID	always
	MData*	datahandshake = 0
	MDataInfo*	datahandshake = 0
	MReqInfo	always
	MReqLast	always
	MReqRowLast	always
	MTagID	always
	MTagInOrder	always
	MThreadID	always

OCP Signal Groups (Continued…)

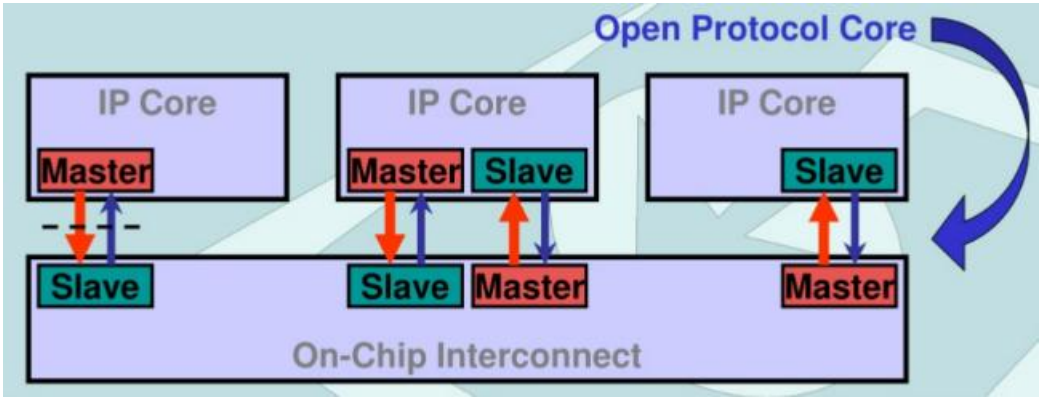
Group	Signal	Condition
Response	SData	always
	SDataInfo	always
	SResp	always
	SRespInfo	always
	SRespLast	always
	SRespRowLast	always
	STagID	always
	STagInOrder	always
	SThreadID	always
Datahandshake	MData*	datahandshake = 1
	MDataByteEn	always
	MDataInfo*	datahandshake = 1
	MDataLast	always
	MDataRowLast	always
	MDataTagID	always
	MDataThreadID	always
	MDataValid	always

Future Opportunities and Summary

System Usage of Open Core Protocol

- Can wrap any on-chip interconnect using OCP
- - Yields system ready for plug-and-play with IP cores

- Well-specified protocol allows interconnect wrapper-generation to be automated



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

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! ☺