IMPLEMENTATION OF A INTERFACE BRIDGE FOR AXI TO OCP PROTOCOL

Nandipati Mounika, Jatroth Srinivas, M. Chandra Sheker
4.E.RAVINDER, 5.Dr.SARVANAN
nandipatimounika1806@gmail.com
Electronics and Communications Engineering
Ellenki College of Engineering,
Hyderabad, India

Abstract— **Protocols are commonly used today to connect IP blocks on structured SoCs. Generally Protocol is the back-bone of the SoC and its failure usually leads to a non-functional chip. In present market, various types of standard protocols are available and are used in SoC which requires a bridge to pass the information from one type of protocol to other type of protocol safely and without any data loss. Advanced eXtensible Interface (AXI) and ON chip protocol is widely used as the de facto standard SoC bus. In this work, the bus bridge was designed to interface these protocols which plays a vital role in SoC application such as it may lead to application failure, if it doesn't work properly. Initially basic AXI 4.0 and OCP protocols are modelled separately using VERILOG and are simulated. Basically Bus Bridge should convert command and data of AXI formats to acceptable OCP formats. This conversion does not ensure proper communication unless the timings of each protocol were met. Hence the interconnecting Bus Bridge wrapper between Advanced eXtensible Interface (AXI) and on chip was designed with proper timing delay.**

Keywords—SOC,AXI,OCP

I INTRODUCTION

There are many companies that develop core IP for SoC products. The interfaces to these cores can differ from company to company and can sometimes be proprietary in nature. The SoC developer then must expend time, effort, and money to create "bridge" or "glue" logic that allows all of the cores inside the SoC to communicate properly with each other. Incompatible interfaces are thus barriers to both IP developers and SoC developers. SoC integrated circuits envisioned by this subcommittee span a wide breadth of applications, target system costs, and levels of performance and integration.

Integrated circuits have entered the era of System-on-a-Chip (SoC), which refers to integrating all components of a computer or other electronic system into a single chip. It may contain digital, analog, mixed-signal, and often radio-frequency functions – all on a single chip substrate. With the increasing design size, IP is an inevitable choice for SoC design. And the widespread use of all kinds of IPs has changed the nature of the design flow, making On-Chip Buses (OCB) essential to the design.

Of all OCBs existing in the market, the AMBA bus system is widely used as the de facto standard SoC bus. On March 8, 2010, ARM announced availability of the AMBA 4.0 specifications. As the de facto standard SoC bus, AMBA bus is widely used in the high-performance SoC designs. The AMBA specification defines an on-chip communication standard for designing high-performance embedded microcontrollers.

ARM introduced the Advanced Microcontroller Bus Architecture (AMBA) 4.0 specifications in March 2010, which includes Advanced extensible Interface (AXI) 4.0 and Open Core Protocol Specification 2.2 which is configurable protocol interface. AMBA bus protocol has become the de facto standard SoC bus. That means more and more existing IPs must be able to communicate with AMBA 4.0 bus. Based on AMBA 4.0 bus and OCP bus, This design is an Intellectual Property (IP) core of AXI(Advanced extensible Interface) Lite to OCP(Open Core Protocol) Bridge, which translates the AXI4.0-lite transactions into OCP transactions. The bridge provides interfaces between the high-performance AXI bus and high-performance OCP domain.

The Advanced Microcontroller Bus Architecture (AMBA) is used as the on-chip bus in system-on-a-chip (SoC) designs. Since its inception, the scope of AMBA has gone far beyond microcontroller devices, and is now widely used on a range of ASIC and SoC parts including applications processors used in modern portable mobile devices like smart phones.

The AMBA protocol is an open standard, on-chip interconnect specification for the connection and management of functional blocks in a System-on-Chip (SoC). It facilitates right-first-time development of multi-processor designs with large numbers of controllers and peripherals.

SOC is a device which integrates all the computer devices on to one chip, which runs with desktop operating systems like windows, Linux etc.

The contrast with a microcontroller is one of degree. Microcontrollers typically have under 100 KB of RAM (often just a few kilobytes) and often really are single-chip-systems, whereas the term SoC is typically used with more powerful processors, capable of running software such as the desktop versions of Windows and Linux, which need external memory chips (flash, RAM) to be useful, and which are used with various external peripherals. In short, for larger systems system on a chip is hyperbole, indicating technical direction more than reality: increasing chip integration to reduce manufacturing costs and to enable smaller systems. Many interesting systems are too complex to fit on just one chip built with a process optimized for just one of the system's tasks.

When it is not feasible to construct an SoC for a particular application, an alternative is a system in package (SiP) comprising a number of chips in a single package. In large volumes, SoC is believed to be more cost-effective than SiP since it increases the yield of the fabrication and because its packaging is simpler.

II. AXI(Advanced eXtensible Interface)

AXI is the high-performance bus in the AMBA family. The architecture defines three write channels and two read channels. The write channels are address, write data, and response. The read channels are address and read data. The address channels include 32-bit address buses, AWADDR and ARADDR, but this could be extended in some implementations. The write and read data buses (WDATA and RDATA) may be defined under the specification as any 2n number, from 8-bit to 1024-bit. With the assumption that both the address and data buses are 32-bit, and that the data buses are 128-bit, the write address, write data, and write response channels would require 56, 139, and 8 I/O, respectively. The read address and read data channels would require 56 and 137 I/O, respectively. Thus, each 128-bit AXI master has 396 I/O total. AXI masters and slaves are connected together through a central interconnect, which routes master requests and write data to the proper slave, and returning read data to the requesting master. The interconnect also maintains ordering based on tags if, for example, a single master pipelines read requests to different slaves.

AXI uses a handshake between VALID and READY signals. VALID is driven by the source, and READY is driven by the destination. Transfer of information, either address and control or data, occurs when both VALID and READY are sampled high. AXI, the third generation of AMBA interface defined in the AMBA 3 specification, is targeted at high performance, high clock frequency system designs and includes features which make it very suitable for high speed sub-micrometer interconnect.

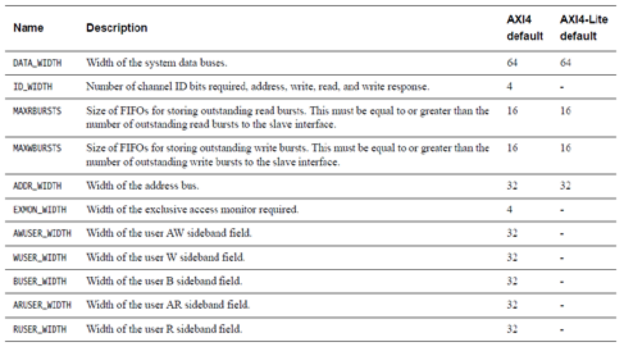
1.)separate address/control and data phases

2.)support for unaligned data transfers using byte strobes

3.)burst based transactions with only start address issued

4.)issuing of multiple outstanding addresses

5.)Easy addition of register stages to provide timing closure



Name	Description	AXI4 default	AXI4-Lite default
DATA_WIDTH	Width of the system data buses.	64	64
ID_WIDTH	Number of channel ID bits required, address, write, read, and write response.	4	-
MAXRBURSTS	Size of FIFOs for storing outstanding read bursts. This must be equal to or greater than the number of outstanding read bursts to the slave interface.	16	16
MAXWBURSTS	Size of FIFOs for storing outstanding write bursts. This must be equal to or greater than the number of outstanding write bursts to the slave interface.	16	16
ADDR_WIDTH	Width of the address bus.	32	32
EXMON_WIDTH	Width of the exclusive access monitor required.	4	-
AWUSER_WIDTH	Width of the user AW sideband field.	32	-
WUSER_WIDTH	Width of the user W sideband field.	32	-
BUSER_WIDTH	Width of the user B sideband field.	32	-
ARUSER_WIDTH	Width of the user AR sideband field.	32	-
RUSER_WIDTH	Width of the user R sideband field.	32	-

Table1 interface parameters for AXI-4 and AXI-4 LITE

Architecture: The AXI protocol is burst-based. Every transaction has address and control information on the address channel that describes the nature of the data to be transferred. The data is transferred between master and slave using a write data channel to the slave or a read data channel to the master. In write transactions, in which all the data flows from the master to the slave, the AXI protocol has an additional write response channel to allow the slave to signal to the master the completion of the write transaction. The AXI protocol enables: • address information to be issued ahead of the actual data transfer

• support for multiple outstanding transactions

• support for out-of-order completion of transactions.

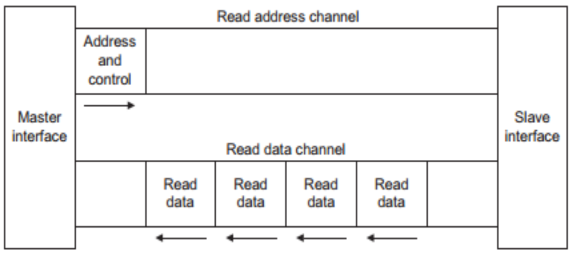


Fig1. channel architecture of reads

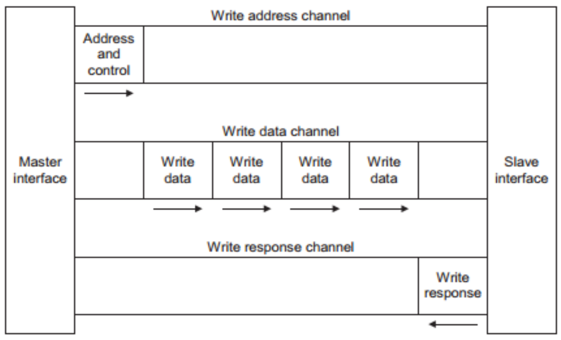


Fig2 channel architecture of writes

Channel definition:- Each of the five independent channels consists of a set of information signals and uses a two-way VALID and READY handshake mechanism. The information source uses the VALID signal to show when valid data or control information is available on the channel. The destination uses the READY signal to show when it can accept the data. Both the read data channel and the write data channel also include a LAST signal to indicate when the transfer of the final data item within a transaction takes place.

Read and write address channels:

Read and write transactions each have their own address channel. The appropriate address channel carries all of the required address and control information for a transaction. The AXI protocol supports the following mechanisms:

• variable-length bursts, from 1 to 16 data transfers per burst

• bursts with a transfer size of 8-1024 bits

• wrapping, incrementing, and non-incrementing bursts

• atomic operations, using exclusive or locked accesses

• system-level caching and buffering control.

Read data channel: The read data channel conveys both the read data and any read response information from the slave back to the master. The read data channel includes:

• the data bus, which can be 8, 16, 32, 64, 128, 256, 512, or 1024 bits wide

• a read response indicating the completion status of the read transaction.

Write data channel: The write data channel conveys the write data from the master to the slave and includes:

• the data bus, which can be 8, 16, 32, 64, 128, 256, 512, or 1024 bits wide

• one byte lane strobe for every eight data bits, indicating which bytes of the data bus are valid.

Write data channel information is always treated as buffered, so that the master can perform write transactions without slave acknowledgement of previous write transactions.

Write response channel: The write response channel provides a way for the slave to respond to write transactions. All write transactions use completion signaling. The completion signal occurs once for each burst, not for each individual data transfer within the burst.

Interface and inter connects:

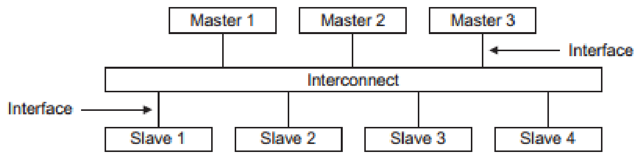


Fig3.interface interconnect.

The AXI protocol provides a single interface definition for describing interfaces:

- between a master and the interconnect
- between a slave and the interconnect
- between a master and a slave. The interface definition enables a variety of different interconnect implementations. The interconnect between devices is equivalent to another device with symmetrical master and slave ports to which real master and slave devices can be connected. Most systems use one of three interconnect approaches:
- shared address and data buses
- shared address buses and multiple data buses
- multilayer, with multiple address and data buses. In most systems, the address channel bandwidth requirement is significantly less than the data channel bandwidth requirement. Such systems can achieve a good balance between system performance and interconnect complexity by using a shared address bus with multiple data buses to enable parallel data transfers.

III OPEN CORE PROTOCOL

Point-to-Point Synchronous Interface:

To simplify timing analysis, physical design, and general comprehension, the OCP is composed of uni-directional signals driven with respect to, and sampled by the rising edge of the OCP clock. The OCP is fully synchronous (with the exception of reset) and contains no multi-cycle timing paths with respect to the OCP clock. All signals other than the clock signal are strictly point-to-point.

Bus Independence:

A core utilizing the OCP can be interfaced to any bus. A test of any busindependent interface is to connect a master to a slave without an intervening on-chip bus. This test not only drives the specification towards a fully symmetric interface but helps to clarify other issues. For instance, device selection techniques vary greatly among on-chip buses. Some use address decoders. Others generate independent device select signals (analogous to a board level chip select). This complexity should be hidden from IP cores, especially since in the directly-connected case there is no decode/selection logic. OCP-compliant slaves receive device selection information integrated into the basic command field.

Arbitration schemes vary widely. Since there is virtually no arbitration in the directly-connected case, arbitration for any shared resource is the sole responsibility of the logic on the bus side of the OCP. This permits OCPcompliant masters to pass a command field across the OCP that the bus interface logic converts into an arbitration request sequence.

Commands:

There are two basic commands, Read and Write and five command extensions. The WriteNonPost and Broadcast commands have semantics that are similar to the Write command. A WriteNonPost explicitly instructs the slave not to post a write. For the Broadcast command, the master indicates that it is attempting to write to several or all remote target devices that are connected on the other side of the slave. As such, Broadcast is typically useful only for slaves that are in turn a master on another communication medium (such as an attached bus).

The other command extensions, Read Exclusive, Read Linked and Write Conditional, are used for synchronization between system initiators. Read Exclusive is paired with Write or Write Non Post, and has blocking semantics. Read Linked, used in conjunction with Write Conditional has non-blocking (lazy) semantics. These synchronization primitives correspond to those available natively in the instruction sets of different processors.

Address/Data

Wide widths, characteristic of shared on-chip address and data buses, make tuning the OCP address and data widths essential for area-efficient implementation. Only those address bits that are significant to the IP core should cross the OCP to the slave. The OCP address space is flat and composed of 8-bit bytes (octets).

To increase transfer efficiencies, many IP cores have data field widths significantly greater than an octet. The OCP supports a configurable data width to allow multiple bytes to be transferred simultaneously. The OCP refers to the chosen data field width as the word size of the OCP. The term word is used in the traditional computer system context; that is, a word is the natural transfer unit of the block. OCP supports word sizes of power-of-two and nonpower-of-two as would be needed for a 12-bit DSP core. The OCP address is a byte address that is word aligned.

Transfers of less than a full word of data are supported by providing byte enable information that specifies which octets are to be transferred. Byte enables are linked to specific data bits (byte lanes). Byte lanes are not associated with particular byte addresses. This makes the OCP endianneutral, able to support both big and little-endian cores.

Pipelining:

The OCP allows pipelining of transfers. To support this feature, the return of read data and the provision of write data may be delayed after the presentation of the associated request.

Response:

The OCP separates requests from responses. A slave can accept a command request from a master on one cycle and respond in a later cycle. The division of request from response permits pipelining. The OCP provides the option of having responses for Write commands, or completing them immediately without an explicit response.

Burst:

To provide high transfer efficiency, burst support is essential for many IP cores. The extended OCP supports annotation of transfers with burst information. Bursts can either include addressing information for each successive command (which simplifies the requirements for address sequencing/burst count processing in the slave), or include addressing information only once for the entire burst.

In-band Information:

Cores can pass core-specific information in-band in company with the other information being exchanged. In-band extensions exist for requests and responses, as well as read and write data. A typical use of in-band extensions is to pass cacheable information or data parity

Tags:

Tags are available in the OCP interface to control the ordering of responses. Without tags, a slave must return responses in the order that the requests were issued by the master. Similarly, writes must be committed in order. With the addition of tags, responses can be returned out-of-order, and write data can be committed out-of-order with respect to requests, as long as the transactions target different addresses. The tag links the response back to the original request.

Tagging is useful when a master core such as a processor can handle out-oforder return, because it allows a slave core such as a DRAM controller to service requests in the order that is most convenient, rather than the order in which requests were sent by the master.

Out-of-order request and response delivery can also be enabled using multiple threads. The major differences between threads and tags are that threads can have independent flow control for each thread and have no ordering rules for transactions on different threads. Tags, on the other hand, exist within a single thread and are restricted to shared flow control. Tagged transactions cannot be re-ordered with respect to overlapping addresses. Implementing independent flow control requires independent buffering for each thread, leading to more complex implementations. Tags enable lower overhead implementations for out-of-order return of responses at the expense of some concurrency.

Threads and Connections:

To support concurrency and out-of-order processing of transfers, the extended OCP supports the notion of multiple threads. Transactions within different threads have no ordering requirements, and independent flow control from one another. Within a single thread of data flow, all OCP transfers must remain ordered unless tags are in use. Transfers within a single thread must remain ordered unless tags are in use. The concepts of threads and tags are hierarchical: each thread has its own flow control, and ordering within a thread either follows the request order strictly, or is governed by tags.

While the notion of a thread is a local concept between a master and a slave communicating over an OCP, it is possible to globally pass thread information from initiator to target using connection identifiers. Connection information helps to identify the initiator and determine priorities or access permissions at the target.

Interrupts, Errors, and other Sideband Signaling:

While moving data between devices is a central requirement of on-chip communication systems, other types of communications are also important.

Different types of control signaling are required to coordinate data transfers (for instance, high-level flow control) or signal system events (such as interrupts). Dedicated point-to-point data communication is sometimes required. Many devices also require the ability to notify the system of errors that may be unrelated to address/data transfers.

The OCP refers to all such communication as sideband (or out-of-band) signaling, since it is not directly related to the protocol state machines of the dataflow portion of the OCP. The OCP provides support for such signals through sideband signaling extensions.

Errors are reported across the OCP using two mechanisms. The error response code in the response field describes errors resulting from OCP transfers that provide responses. Write-type commands without responses cannot use the in-band reporting mechanism. The second method for reporting errors across the OCP uses out-of band error fields. These signals report more generic sideband errors, including those associated with posted write commands.

The Open Core Protocol™ (OCP) defines a high-performance, bus independent interface between IP cores that reduces design time, design risk, and manufacturing costs for SOC designs.

An IP core can be a simple peripheral core, a high-performance microprocessor, or an on-chip communication subsystem such as a wrapped on-chip bus.

The Open Core Protocol:

1.)Achieves the goal of IP design reuse. The OCP transforms IP cores making them independent of the architecture and design of the systems in which they are used

2.)Optimizes die area by configuring into the OCP only those features needed by the communicating cores

3.)Simplifies system verification and testing by providing a firm boundary around each IP core that can be observed, controlled, and validated

The Open Core Protocol interface addresses communications between the functional units (or IP cores) that comprise a system on a chip. The OCP provides independence from bus protocols without having to sacrifice high performance access to on-chip interconnects. By designing to the interface boundary defined by the OCP, you can develop reusable IP cores without regard for the ultimate target system.

Given the wide range of IP core functionality, performance and interface requirements, a fixed definition interface protocol cannot address the full spectrum of requirements. The need to support verification and test requirements adds an even higher level of complexity to the interface. To address this spectrum of interface definitions, the OCP defines a highly configurable interface. The OCP's structured methodology includes all of the signals required to describe an IP cores' communications including data flow, control, and verification and test signals.

Figure 1 shows a simple system containing a wrapped bus and three IP core entities: one that is a system target, one that is a system initiator, and an entity that is both.

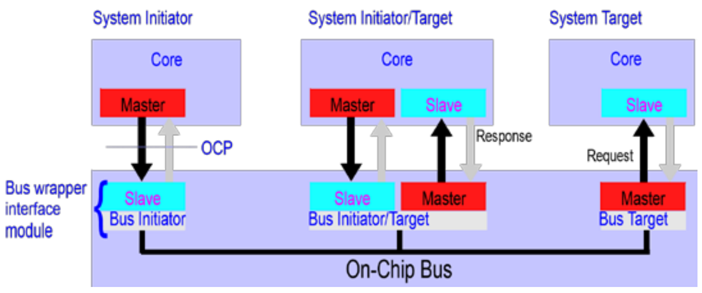
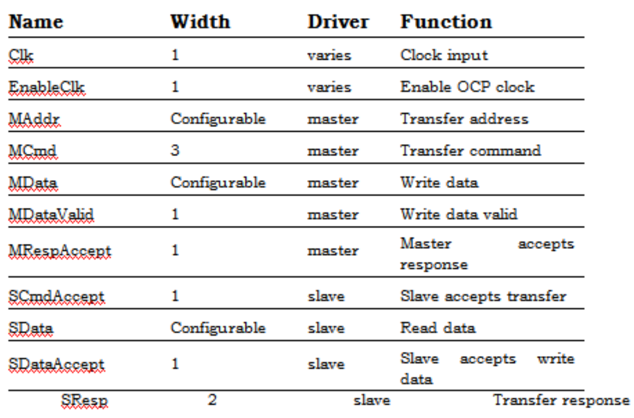


Fig.4 system showing wrapped bus and OCPinstances

Below table2 shows the basic signals of the OCP protocol. Only Clk and MCmd are required. The remaining OCP signals are optional.



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

Table 2basic OCP signals

III.SIGNAL CONNECTIONS

Figure 5 shows the component signal connections. The bridge uses:

• AMBA AXI-Lite and OCP signals as described in the AMBA AXI-Lite 4.0 protocol

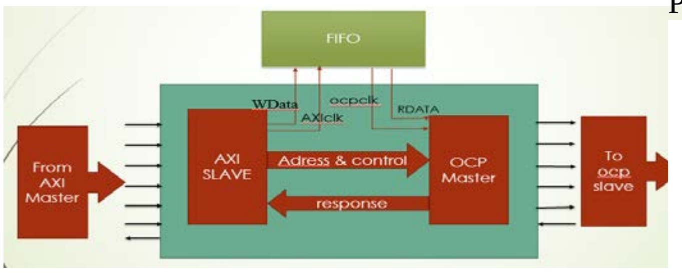


Fig5 signal connections

Handshaking mechanism of AXI&OCP

In AXI 4.0 specification, each channel has VALID and READY signals for handshaking. The source asserts VALID when the control information or data is available. The destination asserts READY when it can accept the control information or data. Transfer occurs only when both the VALID and READY are asserted. Figure6 Shows all possible cases of VALID/READY handshaking. Note that when source asserts VALID, the corresponding control information or data must also be available at the same time. The arrows in Figure. Indicate when the transfer occurs. A transfer takes place at the positive edge of clock. Therefore, the source needs a register input to sample the READY signal. In the same way, the destination needs a register input to sample the VALID signal. Considering the situation of Figure(c), we assume the source and destination use output registers instead of combination circuit, they need one cycle to pull low VALID/READY and sample the VALID/READY again at T4 cycle. When they sample the VALID/READY again at T4, there should be another transfer which is an error. Therefore source and destination should use combinational circuit as output. In short, AXI protocol is suitable register input and combinational output circuit.

The OCP Bridge buffers address, control and data from AXI4-Lite, drives the OCP peripherals and returns data and response signal to the AXI4-Lite. It decodes the address using an internal address map to select the peripheral. The bridge is designed to operate when the OCP and AXI4-Lite have independent clock frequency and phase. For every AXI channel, invalid commands are not forwarded and an error response generated. That is once a peripheral accessed does not exist, the OCP Bridge will generate DE CERR as response through the response channel (read or write). And if the target peripheral exists, but asserts PSLVERR, it will give a SLVERR response

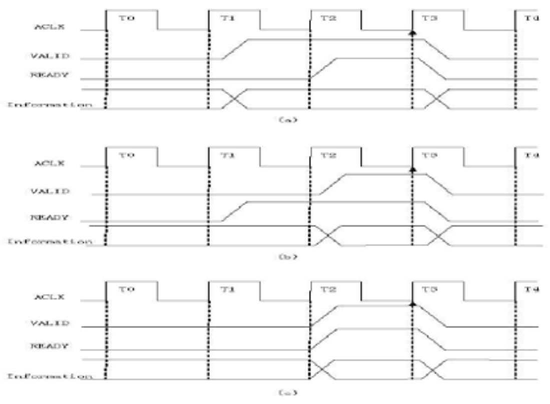


Fig 6 waveforms for handshaking mechanism.

IV FEATURES OF THIS WORK

The AXI to OCP Bridge translates AXI4-Lite transactions intoOCP transactions. The bridge functions as a slave on the AXI4-Lite interface and as a master on the OCP interface. The AXI to OCP Bridge main use model is to connect the OCP slaves with AXI masters. Both AXI4-Lite and OCP transactions are happened during rising edge of the clock.

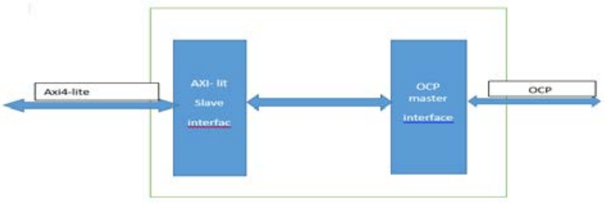


Figure 7 AXI to OCP block diagram

The AXI to OCP Bridge block diagram is shown in Figure 7 and described in subsequent sections.

AXI LITE SLAVE INTERFACE

The AXI4-Lite Slave Interface module provides a bi-directional slave interface to the AXI. The AXI address and data bus widths are always fixed to 32-bits and 1024bits. When both write and read transfers are simultaneously requested on AXI4-Lite, the write request is given more priority than the read request. This module also contains the data phase time out logic for generating OK response on AXI interface when OCP slave does not respond.

OCP MASTER INTERFACE

The OCP Master Interface module provides a bi-directional slave interface to the OCP. The OCP address and data bus widths are always fixed to 32-bits and 1024bits. When both write and read transfers are simultaneously requested on OCP, the write request is given more priority than the read request.

We provide an implementation of AXI4-Lite to OCP Bridge which has the following features:

1.)32-bit AXI slave and OCP master interfaces.
2.)AXI clock domain completely independent of OCP clock domain.
3.)Support up to 16 OCP peripherals.
4.)Suports increment,burst transfer,wrapping functions on data transfer.
5.)OCP signals Configuration.

V.CONCLUSION

Implemented An Intellectual Property (IP) core of AXI4(Advanced Extensible Interface) Lite to Open Core Protocol Bridge, which translates the AXI4.0-lite transactions into OCP Transactions. The bridge provides interfaces between the high-performance AXI bus and Configurable high performance OCP .In this work, the bus bridge was designed to interface these protocols which plays a vital role in SoC application such as it may lead to application failure, if it doesn't work properly. Initially basic AXI 4.0 and OCP protocols are modelled separately using VERILOG and are simulated. Basically Bus Bridge should convert command and data of AXI formats to acceptable OCP formats. This conversion does not ensure proper communication unless the timings of each protocol were met. Hence the interconnecting Bus Bridge wrapper between Advanced eXtensible Interface (AXI) and on chip was designed with proper timing delay.

VI REFERENCES

1. Design and Implementation of APB Bridge based on AMBA 4.0 (IEEE 2011), ARM Limited.
2.http://en.wikipedia.org/wiki/System_on_a_chip#Structure
3. Power.org Embedded Bus Architecture Report Presented by the Bus Architecture TSC Version 1.0 – 11 April 2008
4.ARM, "AMBA Protocol Specification 4.0", www.arm.com, 2010
5.ARM,AMBA Specification (Rev 2.0).
6.AMBA® 4 AXI4™, AXI4-Lite™, and AXI4-Stream™ Protocol Assertions Revision: r0p0 User Guide.
7.AMBA® APB Protocol Version: 2.0 Specifications.
8.ASB Example AMBA System Technical Reference Manual Copyright © 1998-1999 ARM Limited.
9. AHB to APB Bridge (AHB2APB) Technical Data Sheet Part Number: T-CS-PR-0005-100 Document Number: I-IPA01-0106-USR Rev 05 March 2007.