Implementation of OCP-IP interface between IP Cores

Krishnaji Desai, Raghu Prasad Gudla, Srinivas Reddy Chirla
Krishnaji.Desai@utah.edu, Raghuprasad.Gudla@utah.edu, Reddy.Chirla@utah.edu

***Abstract*—This project aims at designing and implementation of a standard OCP-IP protocol between IP cores. The design and implementation of standard OCP-IP is done for clocked and asynchronous protocol versions. Basic memory read and write operations are implemented and simulated for verifying the correctness of the designed protocol. The asynchronous network interface designed using stoppable clock approach is extended to implement OCP-IP asynchronous protocol version.**

***Index Terms*—Open Core Protocol (OCP), Stoppable Clock system, IP Core, Synchronous communication, Asynchronous communication, Handshake process.**

I. Introduction

OPEN CORE PROTOCOL is a standard that defines a point to point interface between two communication entities such as IP cores. IP cores can act as master or slave or both master and slave [2]. Master IP core acts as the initiator for any transaction and slave IP core acts as the target. Communication medium between the cores can be network or a Bus interface module. Open Core Protocol – International Partnership (OCP-IP) is successfully implemented with synchronous communication for simple memory read and writes operations. The synchronous communication has a global clock for all the instances of Master/Slave OCP interface. Subset of OCP-IP signals are used for asynchronous network interface using stoppable clock system which generates the local clock for the IP cores. The external communication between the IP Cores is implemented by designing a request-acknowledge handshake protocol based on OCP-IP standard protocol. Further, this implementation is extended to handle end -end OCP-IP protocol.

II. Project scope

Implementation is carried out using behavioral VHDL in Modelsim simulation environment. Currently, the design is implements a simple memory read and write operations for the clocked interface between two IP cores and asynchronous network interface. The designed handshake protocol in case of asynchronous network interface comply with subset of OCP-IP handshake signals. In case of asynchronous network interface, the synchronous IP-cores are clocked using stoppable clock systems.

III. Basics of OCP-IP

A. Definition of OCP-IP

It is a standard that defines a point to point interface between two communication entities such as IP cores and thus maximizes the reusability of IP cores regardless of whether the communication is synchronous or asynchronous [2]. Hence, this protocol is interface dependent and highly configurable. The OCP-IP is the aggregation of signals and communication protocols that unify communication between IP cores.

B. OCP Signals

OCP Signals act as the interface between any two modules communicating[2]. They are grouped into dataflow, sideband, and test signals. The dataflow signals are divided into basic signals, simple extensions, burst extensions, tag extensions, and thread extensions. A small set of the signals from the basic dataflow group is required in all OCP configurations as shown in Table 1. Optional Signals can be configured to support additional core communication requirements. All sideband and test signals are optional.

Name	Width	Driver	Function
Clk	1	*varies*	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 1. Basic dataflow signals [2]

Clk: All interface signals are synchronous to the rising edge of clock. It is driven by external entity which drives both Master and Slave interface instances.

*MAddr:*It is the transfer address of the resource targeted by current transfer. Its width is configurable and driven by Master.

MCmd : It is the transfer command from the Master indicating the type of request. Command encoding is as shown below

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

Table 2. Command encoding [2]

MData: This signal carries the write data from master to the slave. Width of the signal is configurable and drive by master.
MDataValid: When set to 1, this signal indicates that the data present on the MData is valid.
MRespAccept: The master indicates that it accepts the current response from the slave with a value of 1 on the MRespAccept signal.
SCmdAccept: A value of 1 on the SCmdAccept signal indicates that the slave accepts the master's transfer request.
SData: This signal carries the requested read data from the slave to the master. Width of the data is configurable.
SDataAccept: The slave indicates that it accepts write data from the master with a value of 1 on SDataAccept. It is mainly useful in pipelined write.
SResp: Response field from the slave to a transfer request from the master. Response encoding is as shown below

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

Table 3. Response encoding [2]

IV. GALS AND STOPPABLE CLOCK SYSTEM

A. *GALS*

A stoppable clock system can be used to design a globally asynchronous locally synchronous (GALS) architecture. Communication between modules is done asynchronously using request/acknowledge protocols while computation is done synchronously within the modules using a locally generated clock [5] as shown in Fig 1.

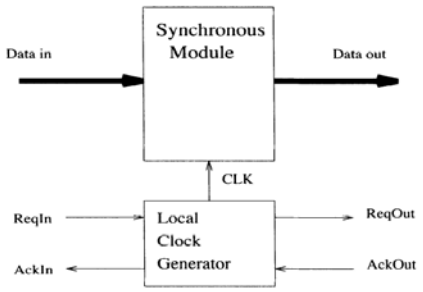


Fig 1. Basic GALS Architecture[5]

B. *Need for Stoppable Clock System*

GALS architecture is designed using stoppable clock systems [5]. To eliminate synchronization failures completely, it is necessary to be able to force the synchronous system to wait an arbitrary amount of time for a metastable input to stabilize. In order for the synchronous circuit to wait, it is necessary for the asynchronous module to be able to cause the synchronous circuit's clock to stop when it is either not ready to communicate new data or not ready to receive new data. A stoppable clock system (is constructed using a ring oscillator) as shown in Fig 2, is the solution for synchronization of handshake signals between synchronous cores.

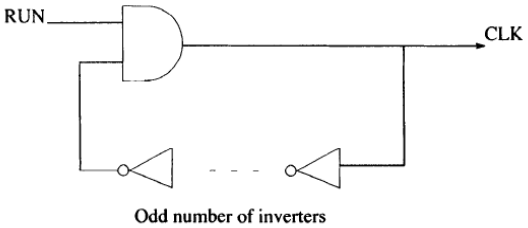


Fig 2. Stoppable Ring oscillator [5]

The stoppable clocks are designed in a manner to satisfy the set-up and hold time constraints with respect to the local clock. Usually clock is stopped synchronously and started asynchronously using Request-Acknowledge signals during asynchronous communication between two synchronous IP-cores/entities. Sufficient odd number of inverters are inserted in the feedback loop to provide required delay. The feedback loop in the stoppable clock system is designed in such a way that it does not delay the stopping of the clock as it is done synchronously. The starting of the clock is delayed by certain amount of delay as the clock is started asynchronously.

V. Design Approach and Implementaition

A. Basic Design Approach

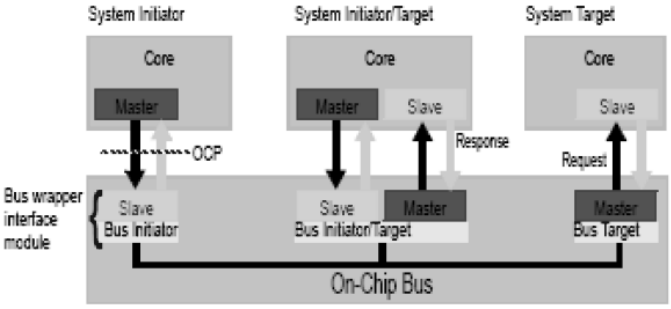


Fig 3. On chip Bus and OCP Interface instances [2]

As shown in Fig 3, a core can act as a mater enity or slave entity or both. Master is the entity which initiates transaction and also acts as is the controlling entity. The slave responds to commands, either by accepting data from the master, or sending data to the master. For two entities to communicate in a peer-to-peer fashion there need to be two instances of the OCP connecting them – OCP-master instance and OCP-slave instance.

The characteristics of the IP core determines whether the core needs master, slave, or both OCP instances. The bus wrapper interface acts as the complementary side of the OCP for each connected entity. A transfer across this system occurs as follows. A system initiator (as the OCP master) presents command, control, and possibly data to its connected slave (a bus wrapper interface module). The interface module places the request across the on-chip bus system. The OCP does not specify the embedded bus functionality. Instead, the interface designer converts the OCP request into an embedded bus transfer commands. The receiving bus wrapper interface module (as the OCP master) converts the embedded bus commands into a legal OCP commands. The system target (OCP slave) receives the command and responds to the requested action.

B. Clocked OCP-IP Interface between clocked IP Cores

Memory Read operation:

Fig 4. shows the block diagram for the Master-Slave interaction for a memory read operation. Here CPU acts as the master entity and RAM acts as the Slave entity.

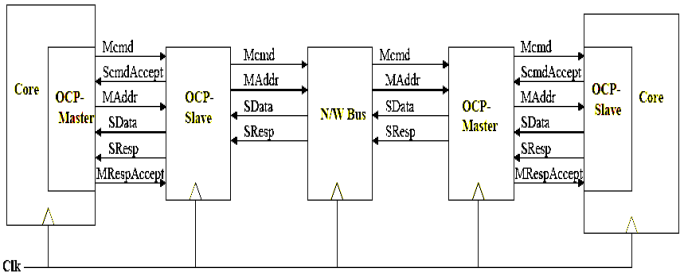


Fig 4. Master Slave interaction for memory read operation

Global clock is applied to each of the modules and all the transactions are synchronized to the leading edge of the clock signal. To start with the memory read operation at the positive edge of clock signal, OCP Master at the core initiates the transaction by placing the command encoding signal MCmd = '001' along with MAddr on read request from the core. OCP Slave instance responds to the request by setting the SCmdAccept signal to '1' and forwards the MCmd and MAddr signals to the network bus interface. After some delay, the command and address is forwarded to the OCP Master instance on the side of Target Core. OCP Master requests the OCP Slave instance with the MCmd and MAddr signal. OCP Slave instance acknowledges OCP master by sending the ScmdAccept signal. Slave core retrieves data from the memory using MAddr and then latches the data onto SData signal along with SResp = '01' for successful memory read. Once the OCP Master receives SResp = '01', it accepts the SData and forwards both the data and response encoded SResp signal back to the network bus module. Also, it acknowledges the Slave module with MRespAccept = '1' indicating the response encoded signal is accepted. After some delay, the response and the data are sent back to the OCP Slave interface. On receiving the SResp = '01' signal the OCP Slave interface accepts the SData and in turn sends back the same to the requested Master module which is waiting for the response and data. Now Master sees the SResp = '01' and reads in the SData and forwards the read data to the core and also acknowledges the OCP Slave module with MRespAccept set equal to '1' indicating the response is accepted. This completes the read transaction which happens through a series of handshakes between master and slave entities.

Memory Write operation:

Fig 5. shows block diagram for the Master-Slave interaction for memory write operation.

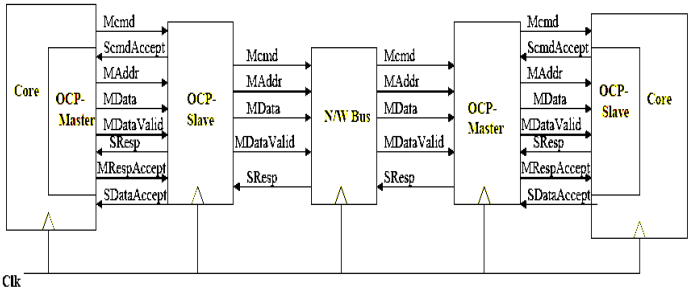


Fig 5. Master Slave interaction for memory write operation

All modules are synchronized with respect to the leading edge of the clock. Once the master core requests for write operation, the OCP Master instance at the core will set the command encoding signal MCmd = '010' along with MAddr and MData with MDataValid set equal to '1'. OCP Slave instance acknowledges the same with the SCmdAccept = '1' indicating the command is accepted. Also it sets the SDataAccept = '1' indicating the valid data has been accepted by OCP Slave. SDataAccept is especially useful in case of

pipelined write. OCP Slave forwards the request command along with the address and data with valid data indicator to the network bus module. After certain delay these signals are forwarded the OCP Master instance on the Target Core side. OCP Slave instance of target responds by acknowledging with SCmdAccept = '1' and SDataAccept = '1'. Once the MData is written into the appropriate MAddr successfully, it sets the SResp signal equal to '01'. OCP Master instance acknowledges the response signal with MRespAccept = '1' and forwards the response encoding signal SResp towards network bus side. After some delay the response encoded signal is sent back to the OCP Slave which in turn forwards it to the Master core which had requested the write operation. Once Master Core OCP instance receives the valid SResp signal it acknowledges the same with MRespAccept signal set to '1'. This completes the write transaction.

C. Asynchronous N/W Interface between two IP-Cores

Asynchronous Network Interface

In a GALS system, for external communication between the two synchronous cores an asynchronous network interface is required. Using the subset of OCP-IP handshake signals below interface is designed between two entities namely Master and Slave cores for external communication as shown in Fig 6.

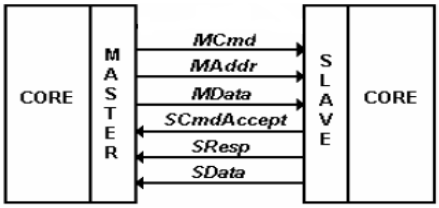


Fig 6. A basic Master Slave Core interface using OCP-IP [3]

Stoppable Clock Systems:

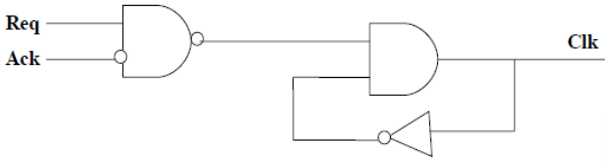


Fig 7. Stoppable clock for Master [2][5][6][8]

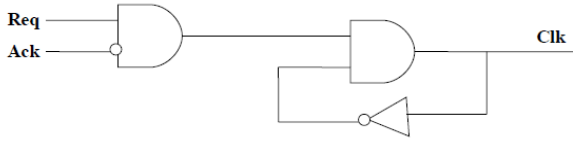


Fig 8. Stoppable clock for Slave [2][5][6][8]

Master core after initiating a transaction using request signal it has to stop its clock and remain stopped until it receives the respective acknowledge signal from the slave core [5] [2]. The Slave core which is assumed to be initially idle state will have its clock in stopped state. On receiving a request signal from the master, the slave has to start its clock, latch the data and do the requested transaction on the positive clock edge. After performing the requested transaction, it has to send out an acknowledge signal back to master and has to stop its clock. Once master receives the acknowledge signal from the slave it starts its clock and latches the data from slave data bus on positive edge of the clock. But overall, prerequisite is that the data should be readily available on the data bus signals ahead of request – acknowledge signals. The stoppable clock systems for master and slave have been designed as shown in Fig 7 and Fig 8.

Memory Read Transaction:

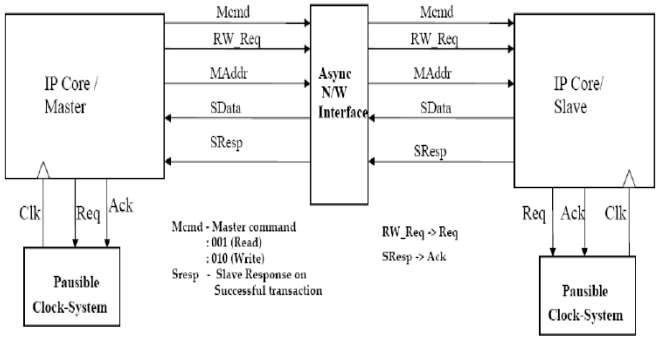


Fig 9. Master Slave interaction for memory read operation

External communication between Master and Slave entities, transaction is initiated by the master. As shown in Fig 9. In a memory read transaction Master cores initiates the read transaction by setting MCmd to "001"[2]. Before the MCmd signal is set the MAddr, master address bus signal should be readily available. RW_Req signal is set whenever MCmd is set to indicate the transaction requested by Master. Once the RW_Req is set the master core clock is stopped and it will remain stopped until it receives the acknowledge signal, SResp from the slave core.

On the other end slave core assuming to be in idle state which means its clock is in stopped state. On receiving the request transaction through RW_Req signal it will start its clock and checks for the requested transaction on MCmd signal. After starting the clock it latches the address from the address bus using the clock. Using the MAddr signal it retrieves the data from RAM memory array and latches that data on to SData signal. Once requested transaction is successfully done it sends out the SResp signal to the master core as an acknowledgment and stops its clock. On receiving the SResp signal from the slave core Master starts its clock and latches data from SData signal. This completes the successful read transaction.

Memory Write Operation:

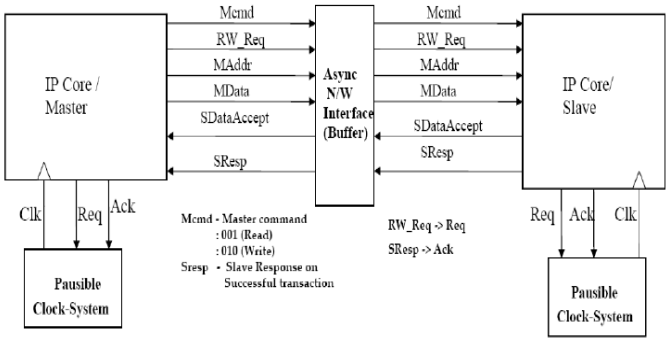


Fig 10. Master Slave interaction for memory write operation

Fig.12 shows the master slave interaction for memory write operation. Memory Write operation is very much analogous to the read transaction. There are extra signals MData and SDataAccept. MData, the master data signal is sent from the master to slave core to write it on to the slave core memory along with the MAddr. In this scenario MCmd is set to "010" indicating a memory write transaction. The stopping and starting of the clock is same as in read transaction.

On the slave core end it receives the RWReq, starts its clock and latches the MAddr, MData from both the bus signals. Once the data is received it sends out the SDataAccept signal to the maser core acknowledging that the data has been accepted successfully. As in this case the MCmd="010" which indicates write transaction, the slave core writes the data into the memory array on positive clock edge depending on MAddr. Once the write process is successful it sends out SResp signal to the master core acknowledging the successful write transaction. This completes the successful memory write operation.

Once the memory write transaction is successful, an external reset is used to flush out the data present on the data and address buses. During this time both the clocks are stopped. Asserting the external reset is handled in the code and this process continues for infinite number of times.

D. Extending the Asynchronous Network Interface to end to end OCP-IP protocol with clocked IP Cores

Memory Read Transaction:

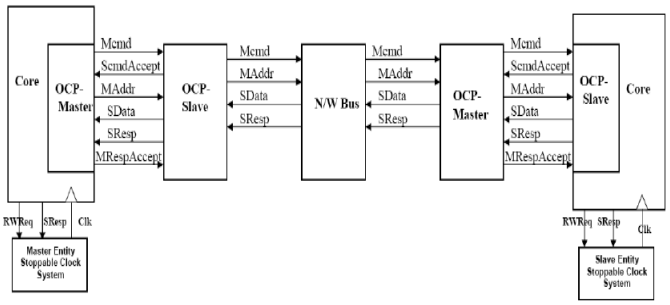


Fig 11. Master Slave interaction for memory read operation

External communication between Master and Slave entities, transaction is initiated by the master. As shown in Fig 11. In a memory read transaction Master cores initiates the read transaction by setting MCmd to "001". Before the MCmd signal is set the MAddr, master address bus signal should be readily available. RW_Req signal is set whenever MCmd is set to indicate the transaction requested by Master. Once the RW_Req is set the master core clock is stopped and it will remain stopped until it receives the acknowledge signal, SResp from the slave core. The requested read command along with memory address is acknowledged by setting SCmdAccept from OCP Slave module and is forwarded to the bus module. After certain delay the OCP Master instance at the target side receives the request and it in turn requests the Slave Core.

The slave core end, assumed to be in idle state which means its clock is in stopped state. On receiving MCmd signal it will start its clock and checks for the requested transaction. After starting the clock, it latches the address from the address bus using the clock. Using the address read from the MAddr bus signal, it retrieves the data from RAM memory array and latches that data on to SData signal. Once requested transaction is successfully done, it sends out the SResp signal to the OCP Master module which acknowledges the response accepted by MRespAccept. It forwards the read data and SResp signal back to network bus module. SResp signal for the Slave core acts as an acknowledgment and stops its clock. After certain delay, the OCP Slave module at the initiator side gets the valid SResp signal along with read data SData. It forwards the signals to the Master core which requested the transaction with SResp signal and the SData. Master on receiving the SResp signal from the slave core, starts its clock and latches data from SData bus. The response received is acknowledged with the MRespAccept signal. This completes the successful read transaction.

Memory Write Operation:

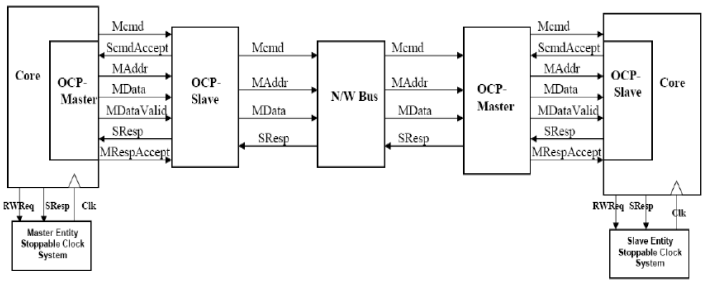


Fig 12. Master Slave interaction for memory write operation

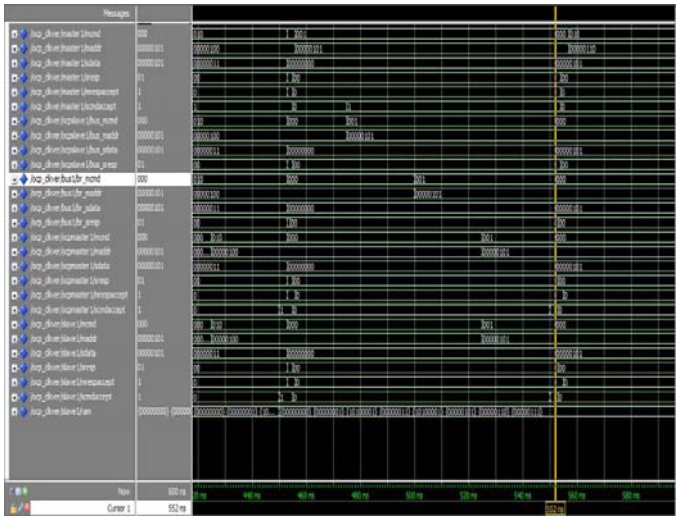
As aforesaid memory write operation is analogous to the read transaction. There are extra signals MData, MDataValid and SDataAccept. MData, the master data signal is sent from the master to slave core to write it on to the slave core memory along with the MAddr with MDataValid indicating the validity of MData. In this scenario MCmd is set to "010" indicating a memory write transaction. The stopping and starting of the clock is same as in read transaction. The requested transaction through the MCmd from Master Core OCP Instance is acknowledged by the OCP Slave instance

through SCmdAccept. SDataAccept is used to acknowledge the valid data being accepted. The request is forwarded to the Slave core as explained in the read operation. SDataAccept handshake happens at the Target Slave core end for the valid MData indicated by MDataValid signal.

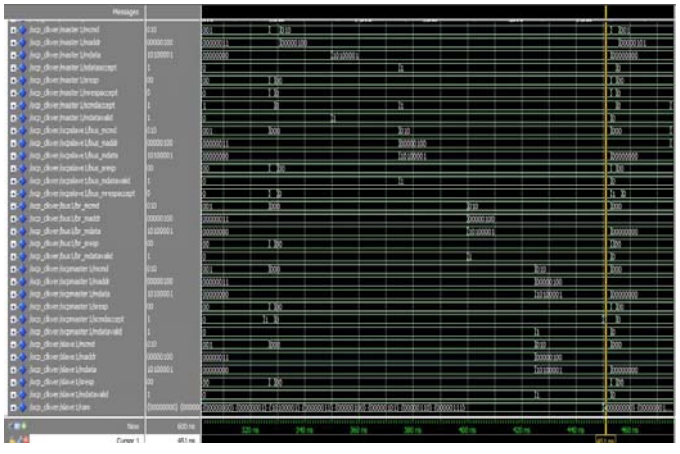
On the slave core end it receives the MCmd, starts its clock and latches the MAddr, MData from both the bus signals. Once the data is received it sends out the SDataAccept signal to the maser core acknowledging the data has been accepted successfully. As in this case the MCmd=”010” which indicates write transaction the slave core writes the data into the memory array on positive clock edge depending on the MAddr. Once the write process is successful it sends out SResp signal back to the OCP Master module and the same gets acknowledged with MRespAccept. Start and stop functions the same way based on the RWReq and SResp signals. This completes the successful memory write operation.

VI. Results

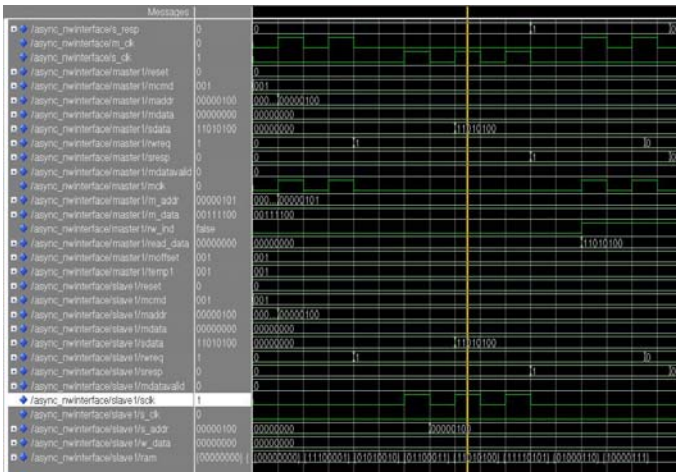
A. *Clocked OCP-IP Memory Read operation*



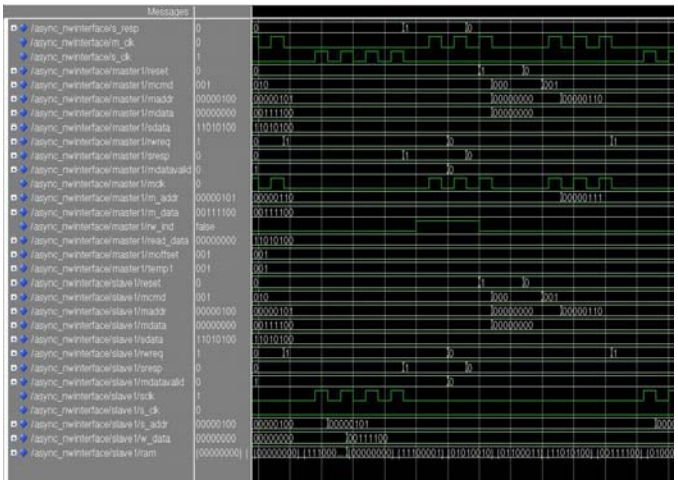
B. *Clocked OCP-IP Memory Write operation*



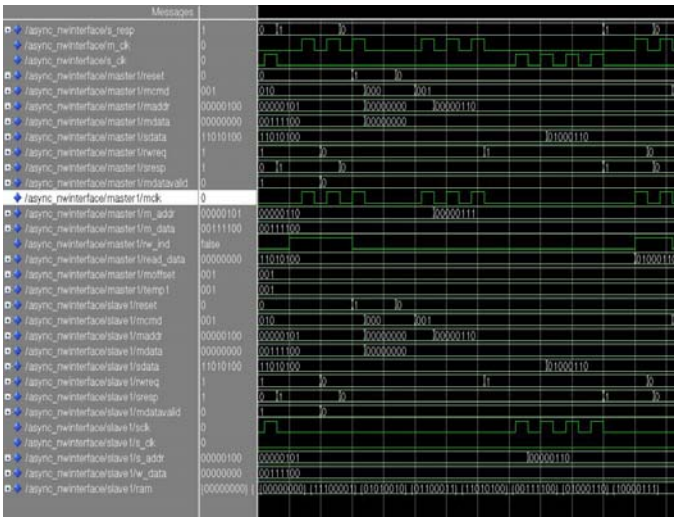
C. *Asynchronous N/W Interface Memory Read Operation*



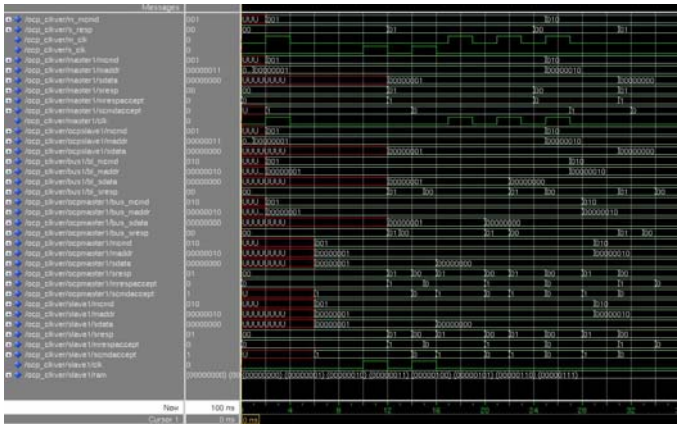
D. *Asynchronous N/W Interface Memory Write Operation*



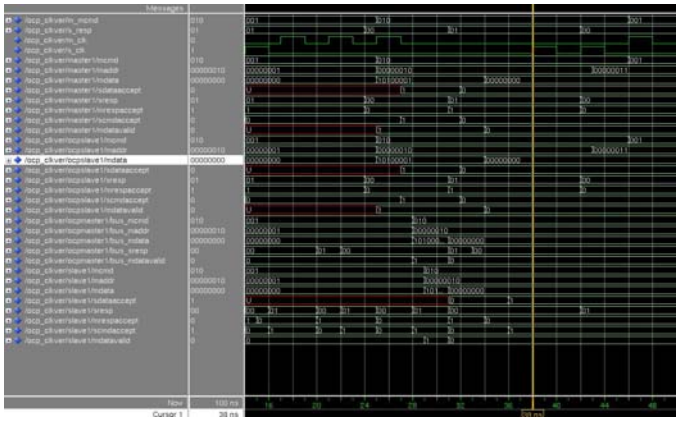
E. *Reset Period after back-back Read/Write transaction*



F. Asynchronous OCP-IP Interface Memory Read



G. Asynchronous OCP-IP Interface Memory Write



H. Synthesis

Initially, ATACS tool is used to synthesize the clocked version of OCP-IP interface code. In order to make compilation error free, the data plane had to be ignored. The synthesis is switched off at places where the data transactions were handled. Hence the synthesis was made free of compilation error only with the control plane. Using Synopsis DC compiler synthesis was run on individual modules and on the top level module. The wait statements were ignored in the process and all reports were mapped with generation of structural verilog.

VII. Conclusion

Initially, a synchronous OCP-IP interface between two synchronous IP cores is successfully designed and implemented for a basic memory read and write operations. An asynchronous network interface between two entities (master and slave module) is implemented using the stoppable clock system approach. Similarly memory read and write transactions are implemented using OCP-IP standard protocol. Finally, the asynchronous OCP-IP protocol version is implemented for the memory read and write transactions. Simulations were run on Modelsim. The functionality each of the protocol versions are successfully verified..

VIII. Future Scope

As part of this project we could successfully implement the Clocked and Asynchronous versions of standard OCP-IP protocol between two IP-cores. Elastic OCP-IP protocol version can also be implemented between the IP cores. Due to the time constraint we have implemented only a basic memory read/write transaction but it can be extended to handle Exclusive Memory Read, Read Linked, WriteNonPost and WriteConditional transactions [2].

This project can be extendable between more than two IP-cores interacting simultaneously using arbiter and Mutual exclusion elements. The asynchronous OCP-IP version was designed using stoppable clock approach but a much more efficient methodology of FMGALS can be employed to design.

References

[1] http://www.ocpip.org/socket/whitepapers/
[2] OCP-IP Specification 2.1 from http://www.ocpip.org
[3] OCP COMPLIANT ADAPTER FOR NETWORK-ON-CHIP by Juliana PeiZhou.
[4] Router Architecture for Connection-Oriented Service Guarantees in the MANGO Clock-less work-on-chip by Tobias Bjerregaard and Jen Sparso.
[5] Asynchronous Circuit Design by Chris J. Myers,University of Utah published by Wiley.
[6] W. Lim. Design methodology for stoppable clock systems. IEE Proceedings, Computers and Digital Techniques,133(1):65-69, January 1986.
[7] The MANGO clock-less Network-on-chip: Concepts and Implementation by Tobias Bjerregaard.
[8] K. Y. Yun and A. E. Dooply. Pausible clocking-based heterogeneous systems. IEEE Transactions on VLSI Systems,7(4):482-488, December 1999.