

Scalable On-Chip Bus and Thread Extension using Open Core Protocol

Manjunath N¹ Prof. Arun Raj S R²

¹M.Tech Student Digital Communication & Networking ²Assistant Professor

^{1,2}Department of Electronics Communication Engineering

^{1,2}UBDT College of Engineering, Davangere, India

Abstract— The technology is getting advanced to the future generation that requires most of the core function an on-chip bus having inter-operability interface irrespective of the core features. The issues that relate to SoC (System on a chip) are 1) a standard communication protocol between different cores. 2) Integration of different clocked domain. 3) The performance increase by system bus. This paper presents the open core protocol (OCP) interface with features such as Basic OCP signals, Simple OCP signals, Burst OCP signals, Tag OCP signals and Thread OCP signals. The Project is optimized with performance, area, power utility, and extended features. These combined improvements enhance the working of a SoC function. The project is designed with Xilinx ISE tool v14.7 in verilog HDL language and verification is done in modelsim.

Key words: OCP, Soc, OCP Interface, Verilog

I. INTRODUCTION

The Integration of different core technology requires an efficient and robust on-chip bus interface. The different core in a system-on-chip (SoC) need bus-independent design, therefore that core can operate without pertaining to on-chip bus functionality and communicate with any other core. An on-chip bus is implemented based on the OCP interface specification specified by the organization OCP-IP, OCP is a non-proprietary, openly licensed and a core-centric protocol which describes the system level integration requirements of core. The AMBA specification is defined by the ARM Company which contains the shared bus architecture details to interconnect more than one master or slave [1]. An OCP specification contains signals and timing details which are described by the members of OCP-IP [2]. In the paper [3] requirements of a standard on-chip bus protocol is discussed and an introduction to open core protocol is considered to fulfill the requisite requirements. In the paper [4] a design of internal crossbar bus architecture employed and adopts the well-defined interface standard Open Core Protocol. In the paper [5] the design and implementation of bus bridge with processor as OCP master and I2C controller as OCP slave is designed and discusses multi-voltage power analysis using synopsis design compiler. In the paper [6] design and implementation of memory interfacing system using Extended Simple OCP signals along with burst support is presented. In the paper [7] presents an OCP interface signals wrapper for cross-clock domain is designed using asynchronous FIFO, this shows that OCP is flexible and portable to other bus protocols. In paper [8] a design of OCP-AHB bus wrapper and built-in controller support for SOC verification. Wrapper configured for burst transaction. In paper [9] the asynchronous or synchronous domain interface wrapper is built. In this design asynchronous components produce the best results in terms of area, performance and energy per transfer.

A. Organization of the Paper:

In section-II description of OCP interface and paper contribution is discussed. In section-III the overall communication flow is revealed. In section-IV data flow signals communication strategy is explained. In section-V the result of synthesis report is presented. In section-VI is future scope. Section-VII is conclusion is discussed and section-VIII is Reference.

II. OCP INTERFACE

An OCP interface follows three types of interconnect as shown in Fig.1

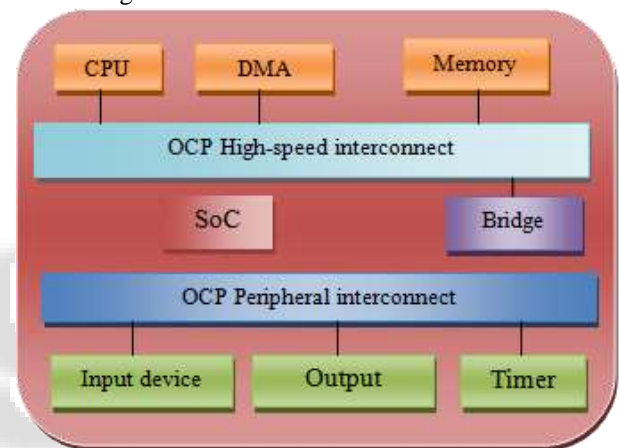


Fig. 1: Open Core Protocol Interconnect

First is the peripheral interconnect which interconnect timer, I/O device, Interrupt controller these are simple basic signals, support single accesses and not necessary to use burst-related signals, a simple read and write communication with added handshake signals may be used. Second is the high speed interconnect these are required for subsystem components that require high throughput such as processor, co-processors, memory controllers and DMA engines. These interconnect require burst-related signals to improve throughput and flow control. These can also carry Tag transaction to implement ordered sequence and out-of-order sequence. Third is the bridge interconnect this is intended to bridge to other interface protocols. The bridge can have either an OCP master or slave port.

A core is a logical block that must integrate the system master or system slave in order to obtain a bus independence interface, this establish a unique interface between any core block. A system master initiates the request and controls the operation, while system slave process the request and respond to the system master. The OCP master interconnects to the system slave and OCP slave interconnects to the system master this establishes an OCP interface. A single path interconnect from one core to another core (peripheral or high-throughput interconnect)

are as shown in the Fig.2. The OCP Bridge is either an OCP master or an OCP slave.



Fig. 2: Block Diagram of an OCP Interface

This paper focus on efficient utilization of shared bus architecture and verified simulation results with 4 masters and 4 slaves. The carried work includes Basic OCP transaction (write and read), Simple OCP transaction (extended signals), Burst OCP transaction (precise and imprecise signals), Tag OCP transaction (in-order and out-of-order signals), and Thread OCP transaction (two Threads). The paper shows the scalability of OCP with pack and unpack feature. The details are shown in rest of the section. The design and synthesis of OCP interface is run on Xilinx ISE tool v14.7 which is coded in verilog HDL language. Verification is done through waveform analysis tool modelsim 10.4.

III. AN OCP INTERFACE OPERATION DETAILS

A single Master and Slave interconnection with basic required signals and their bus width used in this paper are

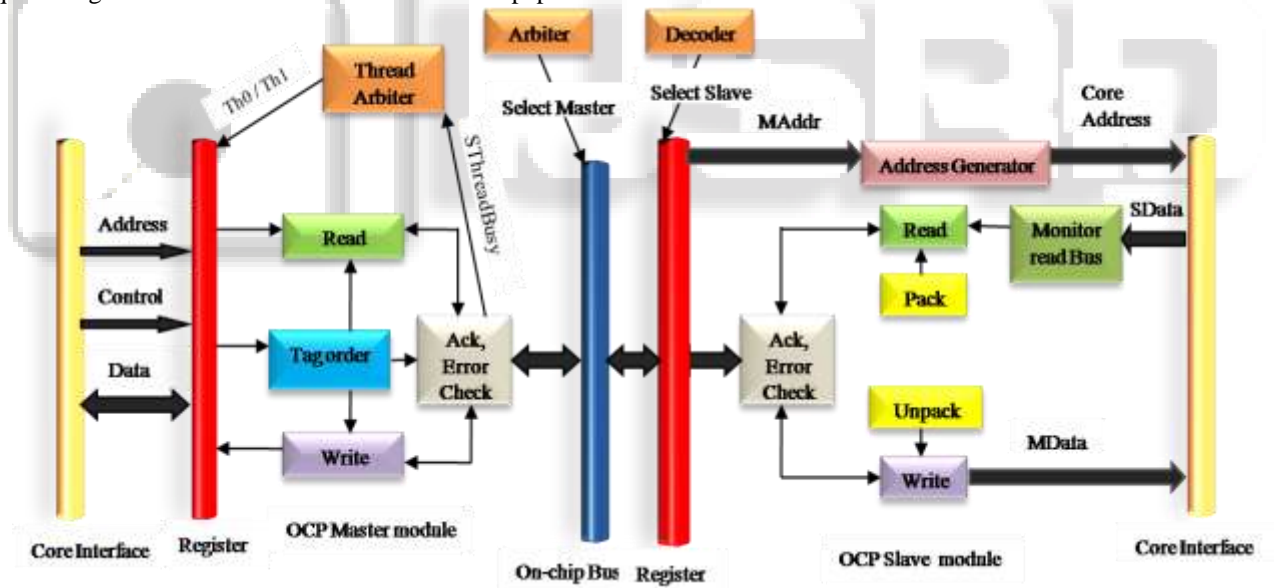


Fig. 4: Working Module in OCP Master and OCP Slave

A. Procedure of Transaction:

- 1) The command is checked at the raising edge of OCP clock by all the masters. If command is present a request to arbiter generated and wait for the grant of on-chip bus from arbiter.
- 2) The master that get grant signal will initiate either write or read request phase.
- 3) Thread arbiter will pass in-order request onto Thread0 and out-of-order request onto Thread1, if SThreadBusy signal from slave corresponding to selected Thread is busy then the request have to wait until successful or error response is got.
- 4) Tag order will present a Tag ID(binary identity value) according to the TagInOrder signal.
- 5) The selected slave from decoder will handle signals by saving into register for next cycle use and send Ack signal SCmdAccept.
- 6) Information regarding the transaction or Error check may be done using simple extension signals.
- 7) Address generator decode slave address.
- 8) In read cycle, core interface bus is monitored and received SData passed to master, if packing of data is

shown in Fig.3. The Datahandshake signals are optional signals that comprise MDataValid and SDataAccept signal used only to indicate the write transaction. While read transaction are coped with SResp and MRespAccept signals as required signals which represent as common signals. MAddr is divided into two parts lsb 2bit input to decoder which will select the individual slave and response signals, other higher bits used as slave address used to address core location. MData is master request phase data and SData is the slave response phase data.

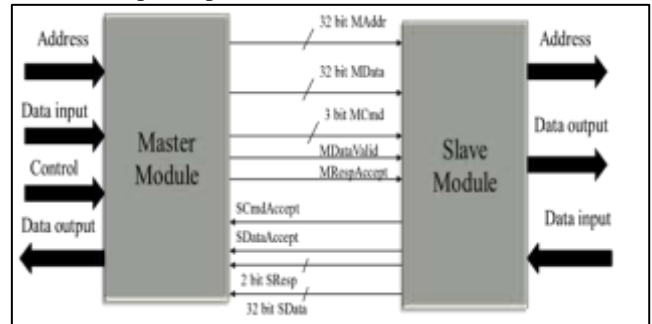


Fig. 3: Block Diagram of Single Master and Slave Module with Required Signals

Master module and Slave module behaviour is shown in the Fig.4 master read data and write data at the slave while slave read data and write data at master. Register are used to store data and signals for later clock cycle use.

required such as 8bit to 16bit or 8bit to 32bit or 16bit to 32bit is achieved.

- 9) In write cycle, MData is passed to core interface, if unpacking of data is required such as 32bit to 16bit or 32bit to 8bit or 32bit to 16bit is achieved.
- 10) Slave after completion of transaction Ack signal for successful or failure is sent by SResp(2bit) signal.
- 11) Master notice response signal then checks the tag ID, if its in-order compares the sequence required if its exact it accept and send Ack signal MRespAccept or else no Ack signal sent for response. If its out-of-order it accepts data and Ack signal is sent to slave.

B. Multiple Master and Slave Operation:

In a shared bus architecture needs a decision making arbiter and slave selector decoder. Arbiter must utilize bus more efficiently, make decision very fast, must have priority scheduling.

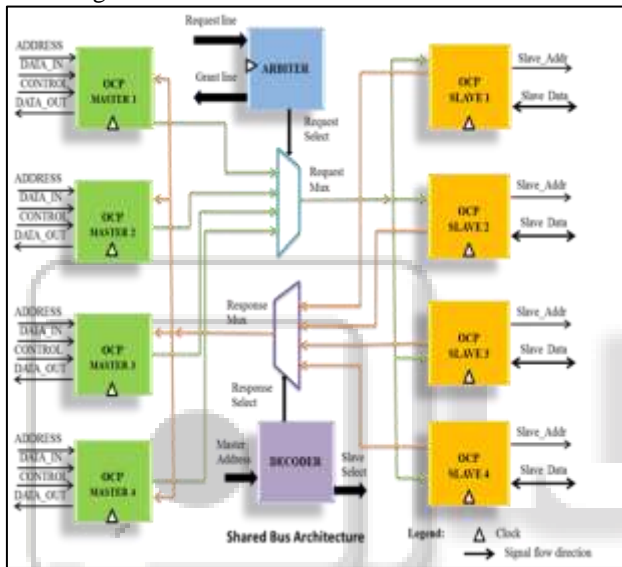


Fig. 5: Block Diagram of 4 Masters and 4 Slaves

The Fig.5 shows shared bus architecture where arbiter will decide which master should get grant of the on-chip bus for the current OCP clock cycle. All master get the command, address and data from its core interface each master check's if it has a request, if it has one master request arbiter for a grant signal. After grant accepted master perform its transaction through request mux and the corresponding slave is selected by decoder, the slave provide response to master request through response mux. Mux bypass the required and optional OCP signals from selected master or slave.

C. Working of Arbiter:

Arbiter is a logic component that allocates scarce resources. The algorithm used here is priority based selection algorithm. A 4bit priority register is used to hold the priority code and after every completion of the transaction the priority code is cyclically shifted once, so that the next succeeding master will have the highest priority. Zero(0) will have highest priority and 1 have lower priority and so on as shown in Table.1

Grant	M4	M3	M2	M1
G1	3	2	1	0

G2	2	1	0	3
G3	1	0	3	2
G4	0	3	2	1

Table 1: Row Shows Priority of Master and Grant of Bus Selected

The proposed arbiter is similar as the round robin priority based selection with added functionality that it can decide at the first level(Master 1) of the selection it needs grant or not without going to the next level. If master1 does not get grant then the decision passes to the next level. The arbiter is more efficient it does not deny any master with request and allow on-chip bus to be utilized by any one of the master, arbiter employ fast decision making circuit. Fig.6 shows the arbiter decision making dataflow diagram. Table.2 shows priority based selection example for the current clock cycle.

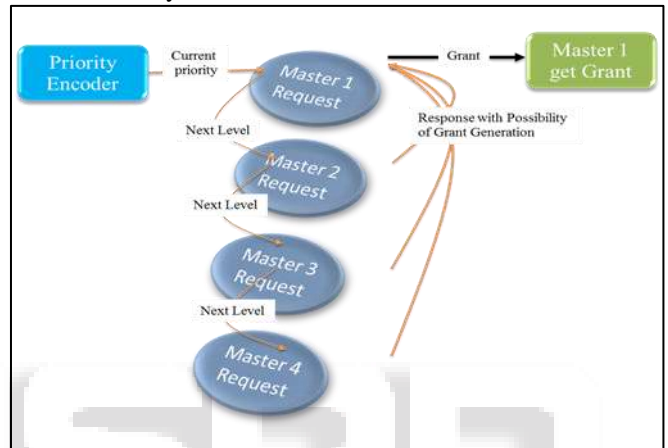


Fig. 6: Data Flow Diagram of Priority Based Selection Algorithm of Arbiter

Master	M4	M3	M2	M1
Priority	1	0	3	2
Request	0	0	1	1
Grant	X	X	----	G1

Table 2: Example of Priority Based Selection for a Clock Cycle

The arbitration is the key component to select the master and grant access to the corresponding slave. The arbiter in the example table.2 gives Grant(G1) to Master1 since highest priority holding master does not have any request therefore next priority is the master1 which gets the Grant. The arbiter is much faster and decision is taken at the raising edge of the OCP clock.

D. Overall Communication flow of OCP Interface:

The overall communication flow is shown in Fig.7. The process starts at the raising edge of the clock pulse all the master checks the command generated from system initiator, if command is not idle then master request grant. Arbiter decides which master should get grant. The master chooses the thread and process write or read transaction. Slave chooses the same thread and process the given request and send response back to master. The acknowledgement (Ack) is exchanged as a token of handshake.

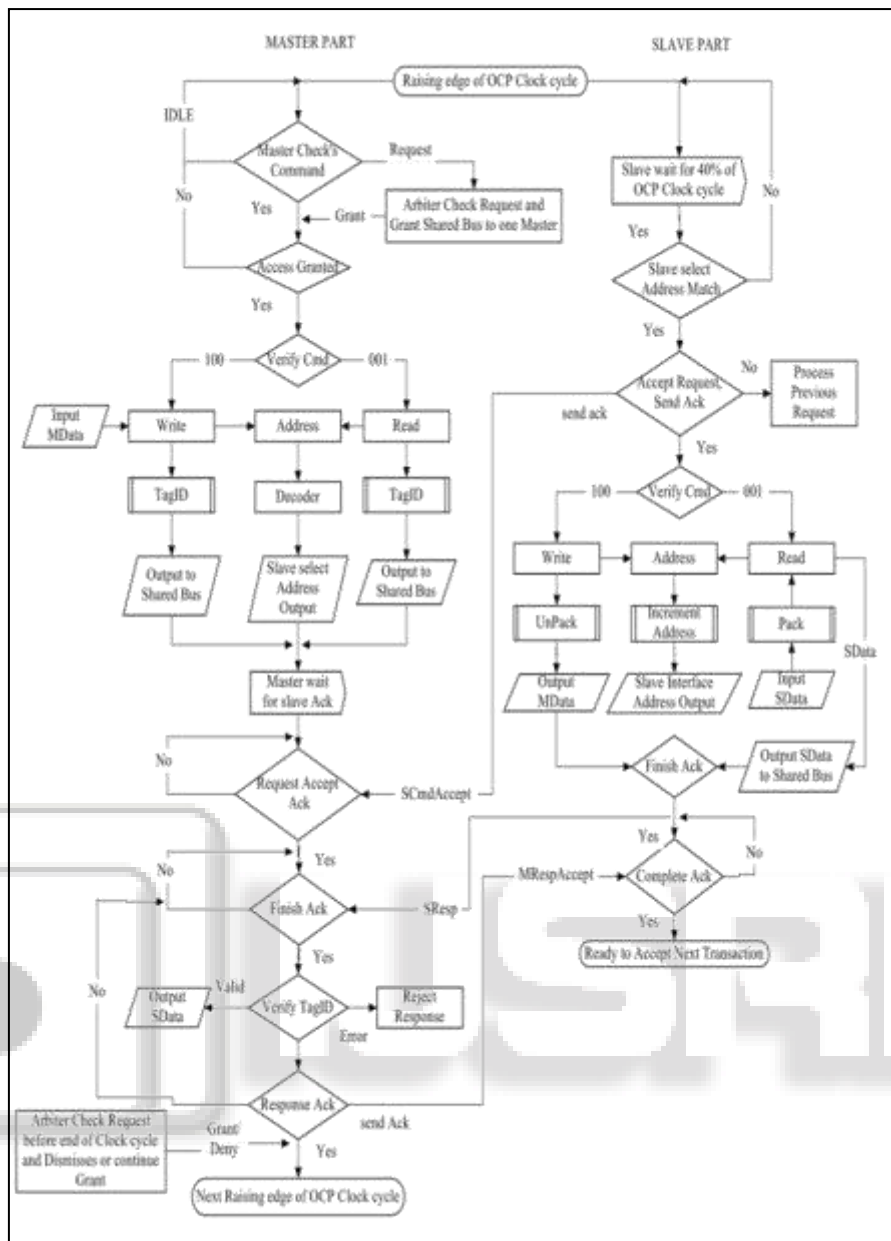


Fig. 7: Overall Communication Flow Chart

IV. DATA FLOW SIGNALS

The data flow signals are classified into five categories.

- 1) Basic OCP signals
- 2) Simple OCP signals
- 3) Burst OCP signals
- 4) Tag OCP signals
- 5) Thread OCP signals

A. Basic OCP Signals:

Basic OCP signals carry out common read and write operation this may include handshake signals. The command signal MCmd denotes the type of operation. WR(100) in the MCmd denote the write operation specified by OCP master. MAddr and MData are the address and data for communication. When OCP slave gets ready to accept data from OCP master the SCmdAccept signal goes high and MData is received by the OCP slave, after the data transfer is successful OCP slave sends SResp signal. Similarly if OCP master signals MCmd as RD(001) a read operation is performed. OCP master gives the address

(MAddr) and when OCP slave is ready (SCmdAccept = 1), the data read is available at the SData and successful transaction is indicated by SResp signal as Data valid/available (DVA). Once DVA is indicated OCP master read data from SData. The MDataValid signal from master and SDataAccept signal from slave represent handshake signal for write transfer. The Table.3 shows the command signals.

MCmd(2:0)	Command	Function
000	Idle	No operation
001	Read	Slave to Master read data
010	ReadEx	Read data from locked location
011	ReadLinked	Read data from unlocked location
100	Write	Master to Slave write data
101	WriteConditional	Write data to reserved location

Table 3: List of Command Signals

B. Simple OCP Signals:

Simple OCP signals are the extension signals to support basic signals, these signals are optional signals and used only if core interface requires.

MDataInfo	Core specific				Parity check			
	Higher byte				Lower byte			
MAddrSpace	0	0	0	0	0	0	0	1
	15	14	13	12	11	10	9	8

Say if 2⁸ is selected a 127byte space is accessed and if 2⁹ is selected then next 127byte space is accessed, so this allocate a range of address space.

MByteEn	0	0	1	1
	[31:24]	[23:16]	[15:8]	[7:0]

Fig. 8: Simple Extension Model

MAddrSpace is used to map the address space in a memory. MByteEn and SByteEn each bit are used to enable particular byte in an OCP wordsize. MDataInfo, SDataInfo are the extended information carrier related to Data, lower half of the signals contain parity check signal to notice loss of data. MReqInfo, SRespInfo can transmit extended request and response phase related data. Fig.8 shows the simple OCP signals

Scalability:- The OCP interface supports scallable bus. The transaction from wide bus to narrow bus and viceversa requires scalability to deliver and access data pertaining to different master or slave interface. MbyteEn bits enable the bytes corresponding to the bus width.

C. Burst OCP Signals:

Burst OCP signals are required for high throughput of transaction. Burst transactions allow multiple transfers. MBurstLength specifies the length of the transfer in a burst. MBurstPrecise signal classify multiple transfer into two types of transaction.

- 1) Precise Burst
- 2) ImPrecise Burst

Precise burst occur when MBurstPrecise is asserted the length of the burst is known at the starting of the burst. In imprecise burst MBurstPrecise signal is Low the length of the burst vary and when length is equal to one, burst ends.

MAddr hold the address. MData and SData are the 32bit data bus. MBurstSeq specifies type of burst operation. The MBurstSeq command INCR is used if the address to be incremented by its OCP wordsize and WRAP used for fixed memory size, if exceeds data put again from start address. All the burst transaction is same as the basic write and read operation except that end of the burst request is indicated by the signal MReqLast and its response by SRespLast.

Precise Burst

Request	Req1	Req2	Req3	Req4	Req5
Address	A1	A2	A3	A4	A5
MBurstLength	5				
ImPrecise Burst					
Request	Req1	Req2	Req3	Req4	Req5
Address	A1	A2	A3	A4	A5
MBurstLength	3	3	2	2	1

Fig. 9: Burst Precise and Imprecise Model

D. Out-of-Order OCP Signals:

Out-of-Order or Tag transactions are the sequence numbering to the transaction. There are two types of tag transaction.

- 1) In-Order
- 2) Out-of-Order

In-Order transaction send request in series and receive response in series. The request is processed sequentially. If responses to this request is received out-of-order these responses are not accepted. MTagInOrder signal is asserted for this type of transaction.

Out-of-Order transaction which uses the MTagID and STagID to identify the transaction, this transfer may be delayed since performance fall is acceptable to these transfers. MTagInOrder signal is deasserted for this type of transaction. Fig.10 show in-order and out-of-order model

In-Order transaction:

Request	Req1	Req2	Req3	Req4	Req5
MTagID	0	1	2	3	4
Response	Resp1	Resp2	Resp3	Resp4	Resp5
STagID	0	1	2	3	4

Out-of-Order transaction:

Request	Req1	Req2	Req3	Req4	Req5
MTagID	0	1	2	3	4
Response	Resp3	Resp2	Resp1	Resp5	Resp4
STagID	2	1	0	4	3

Fig. 10: In-Order and Out-Of-Order Model

Tag transfer allows flow control and priority based transfer. The write and read operation uses the identification number called Tag.

E. Thread Extension OCP Signals:

In this work there are two threads in each OCP master or OCP slave. Thread arbitration decides which thread must be used for the current transfer based on the received SThreadBusy signal, this signal indicate which thread is busy at slave module. Thread0 is chosen for in-order transfer and Thread1 for out-of-order transfer. When both the threads are busy at any instant the master switch the thread between clock cycles therefore any thread can complete first leaving other thread running.

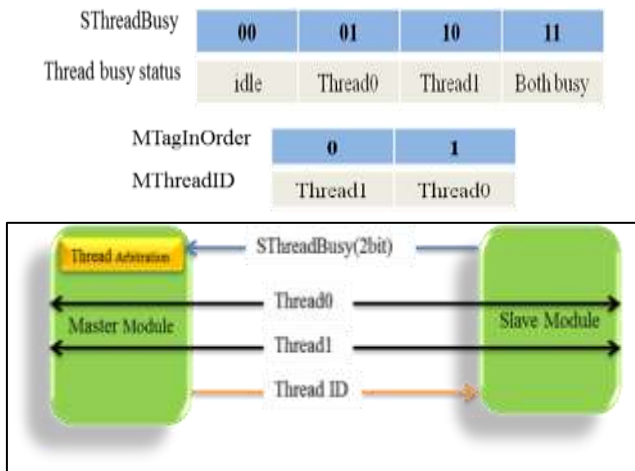


Fig. 11: Thread Extension Model

Thread extension are used to support complex core function. The thread function depends on the core requirements. MConnID signal identify the core interface. SThreadBusy signal indicate which thread is running currently. MThreadID denote the current working thread in current clock cycle. Fig.11 shows the thread extension model.

F. ReadEx transaction or Locked Transaction:

ReadEx is similar to the read operation and is combined with write. Once the ReadEx is performed the location is locked and next operation to the locked location is the write operation. The locked transaction is important when same location needs to be accessed by different master. The command restricts the access to only one master. The work carried to check overlapping of the locked location is removed and also shows no other thread can access the locked location.

G. Readlinked and Writeconditional Transaction:

ReadLinked is same as the read operation but with no lock on the location, a reservation is set for read location. The next operation is the WriteConditional(WRC) transaction which will check the reservation, if it is set then the write transaction is complete and clears reservation otherwise if reservation is not set the failure response is returned without write transaction. In this work all master can reserve and access the reserved location by WRC command and can't access other master reserved location with WRC but all master are unblocked by other commands to access reserved location and clears reservation.

V. SIMULATION RESULTS

The project is designed in schematic and verilog HDL code using Xilinx 14.7 ISE Tool. The design is verified by synthesis in Xilinx synthesizer and Xpower analyzer; waveform is analyzed using Modelsim 10.4.

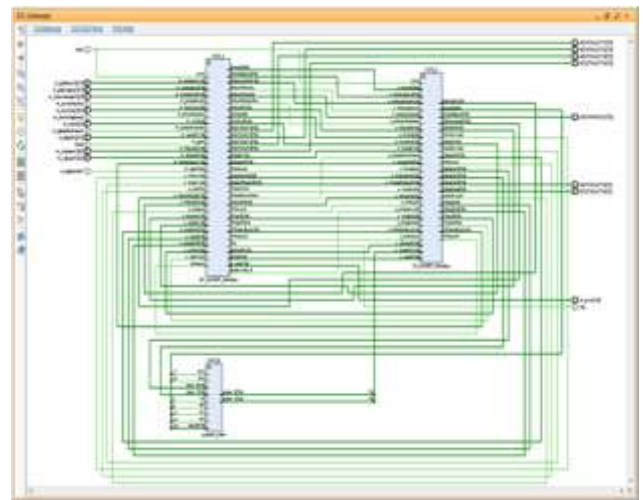


Fig. 12: RTL Schematic of the One Master and Slave Generated From Planahead

Supply Source	Summary Voltage	Total Current (A)	Dynamic Current (A)	Quiescent Current (A)
Vccint	1.200	0.013	0.003	0.010
Vccaux	2.500	0.010	0.000	0.010
Vcco25	2.500	0.002	0.000	0.002

Supply	Power (W)	Total	Dynamic	Quiescent
		0.045	0.004	0.041

Fig. 13: XPower Analyzer Result

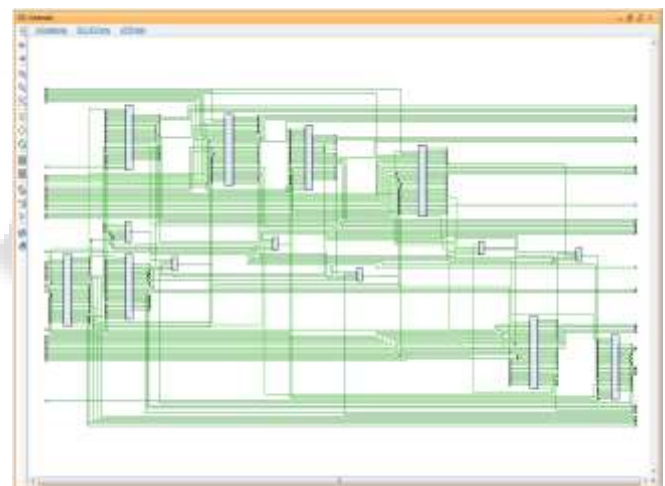


Fig. 14: RTL Schematic from Planahead of the Four Masters, Four Slave, Arbiter, Decoder and Mux

The OCP interface contains functions Basic, Burst, Tag, Thread, ReadEx-write and ReadLinked-WriteConditional and the Table.4 gives its device utilization summary of Spartan 3, XC3s200 family device, the summary listed for one master and one slave interfaced with the Asynchronous memory RAM of 64 locations of 16bit width. The arbiter design summary is also listed in the table. The comparison of the table shows that either packing or Thread usage will increase SoC performance. Packing is the pack and unpack feature used to pack lower width bus to higher width bus and unpack is vice versa. Packing and thread gives higher performance in pipelined transaction looking at the table packing can be achieved with slight reduction in frequency. Thread can be used in complex core communication when core have dual work in parallel.

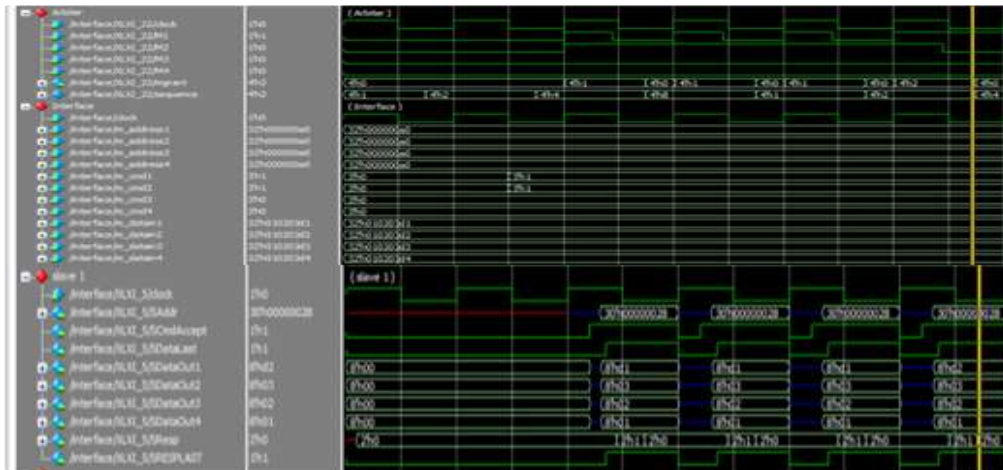


Fig. 15: Arbiter Operation

Logic Utilization	Number of Slice	Number of Slice flip-flop	Number of 4 input LUTs	Number of BRAM	Number of GCLKs
Used by two Thread	503	594	904	8	1
Used by two Thread and no Packing	370	490	543	8	1
Used by packing & no Thread	378	470	700	---	1
Used by no packing & no Thread	294	395	441	---	1
Used by Arbiter	11	8	20	---	1
Available	1920	3840	3840	12	8
Utilization by two Thread	26%	15%	23%	66%	12%
Utilization by 2Thread & no Packing	19%	12%	14%	66%	12%
Utilization by packing & no Thread	19%	12%	18%	---	12%
Utilization by no packing, no Thread	15%	10%	11%	---	12%
Utilization by Arbiter	0%	0%	0%	---	12%

Table 4: Device Utilization Summary (Estimated Values)

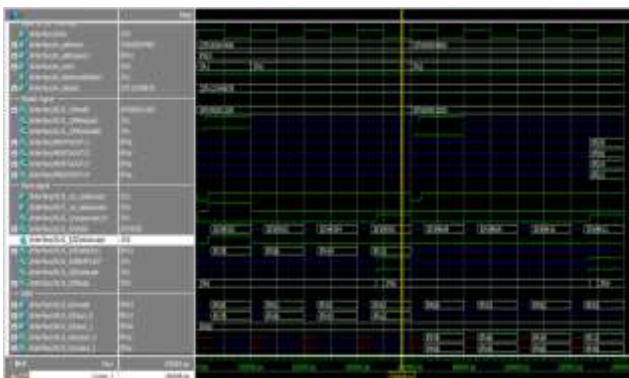


Fig. 16: Basic Write (32bit To 8bit) Unpack Operation And Read(8bit To 32bit) Pack Operation With Handshake Signals.



Fig. 17: Precise Burst Read of 16bit to 32bit Pack Operation

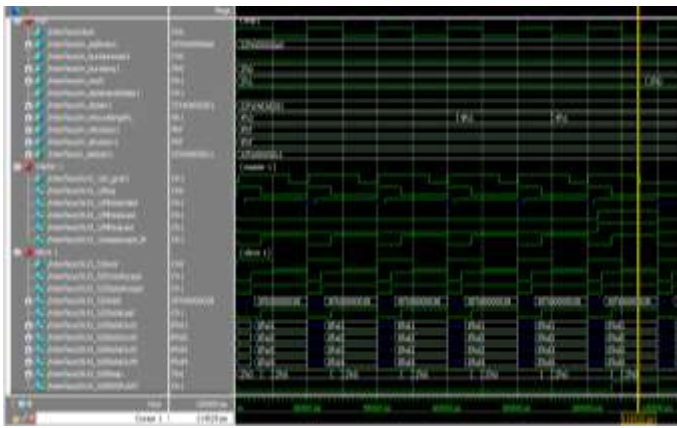


Fig. 18: Imprecise Burst Write Operation

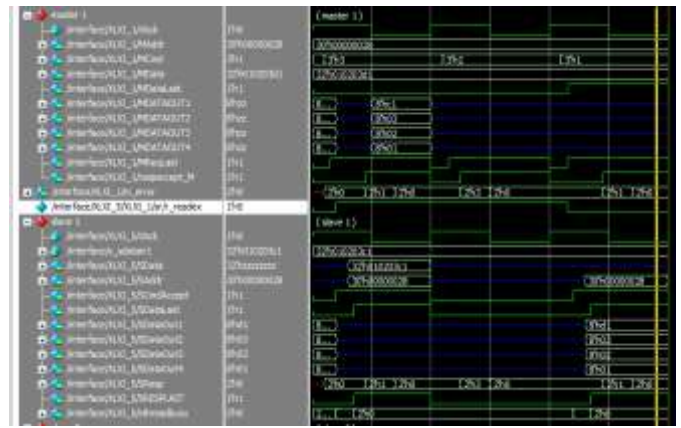


Fig.19: Readex and Write Transaction (Locked Transaction)

Quantity	Values for two Thread	Values for two Thread, no packing	Values for Packing, no Thread	Values for no Packing, no Thread	Values for Arbiter
Hardware platform	Xilinx Spartan3, XC3S200,-5				
Min period	9.284ns	6.697ns	8.068ns	6.628ns	1.648ns
Max frequency	107.708MHz	149.314 MHz	124 MHz	150.871MHz	606.962 MHz
Setup time	9.359ns	8.931ns	8.389ns	7.728ns	6.618ns
Hold time	10.019ns	9.405ns	8.440ns	8.440ns	6.141ns

Table 5: Observed Parameters of Frequency, Setup Time and Hold Time (Estimated Values)

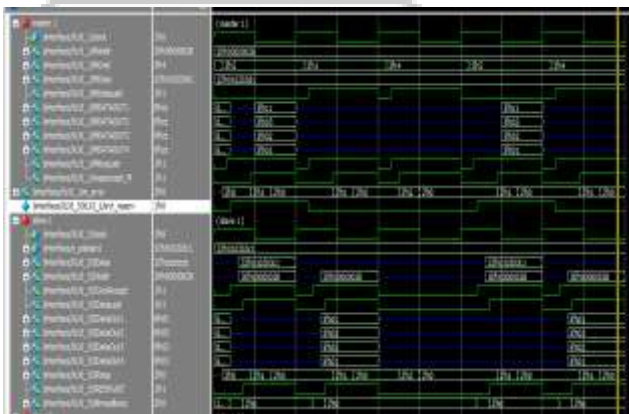


Fig. 20: Readlinked And Writeconditional Transaction

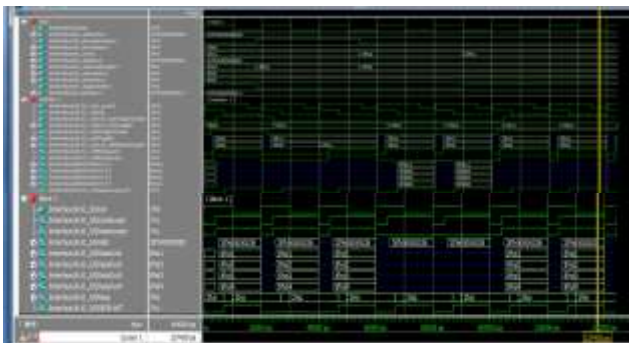


Fig.21: Tag Extension Signals

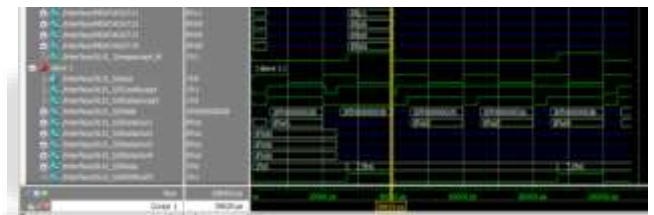


Fig. 22: Thread Extension OCP Signals

VI. FUTURE WORKS

The paper can be extended by pipeline transaction to be implemented for multi-master environment. Arbiter that have scheduler to control bandwidth constraint access need to be designed. The multi dimensional transaction can be implemented for large data processing. Simple extension signals can be completely utilized to carry information from all core's. sideband signals are to be used to handle external signal task this can be implemented for debugging the OCP interface. The project can be advanced to handle stream, broadcast and 2D block of data in burst cases.

VII. CONCLUSION

The OCP interface is successfully verified with Asynchronous Memory Interface and signal flow diagram for simple read and writes, burst read and write, Tag extension and Thread extension verified using Modelsim 10.4 Timing Analysis. The power estimation is obtained from the XPower Analyzer and synthesis from Xilinx ISE v14.7 tool. The Asynchronous 16bit RAM is Interfaced with the OCP Interface for its testing. The work is also carried for 4master and 4slave with high performance and efficient utilization of the on-chip bus by arbiter control. This paper puts packing in the bus utilization, the locked location is monitored for overlapping address when used in scalability condition. The error from the response is indicated to system

master. The results show that OCP as an on-chip bus can increase the performance of the SoC with added functionality like thread extension. The project shows less area utilization with packing function. The parameters observed are frequency, setup and hold time, area utilization and power. Most of the required cores can be interfaced with the functionality of OCP interface to communicate with peripheral devices or high performance required cores.

REFERENCE

- [1] Advanced Microcontroller Bus Architecture (AMBA) Specification Rev 2.0 & 3.0, <http://www.arm.com>.
- [2] Open Core Protocol (OCP) Specification 3.0, <http://www.ocpip.org/home>.
- [3] Wolf-Dietrich Weber "Enabling Reuse via an IP Core-centric Communications Protocol: Open Core Protocol" © 2000 Sonics, Inc.
- [4] chin-yaochang, yi-jiunchang, kuen-jonglee, jen-chiehyeh, shih-yin lin and jui-liangma, "Design of On-Chip Bus with OCP Interface" 2010 IEEE
- [5] Ramesh Bhakthavatchalu, Deepthy G R, et. al. "Analysis of Low Power Open Core Protocol Bridge Interface Using VHDL" 2011 IEEE
- [6] Elina Rajan Varughese, Rony Antony P "IMPLEMENTATION OF EXTENDED OPEN CORE PROTOCOL INTERFACE MEMORY SYSTEM USING Verilog HDL" 2013 IEEE
- [7] Shen-Fu Hsiao, Chi-Guang Lin, Po-Han Wu, and Chia-Sheng Wen "Asynchronous AHB Bus Interface Designs in a Multiple-clock-Domain Graphics System" 2012 IEEE
- [8] Cheng-Ta Wu, Feng-Xiang Huang, Kuan-Fu Kuo, Ing-Jer Huang "An OCP-AHB Bus Wrapper with Built-in ICE Support for SOC Integration" 2012 IEEE
- [9] Vikas S. Vij, Raghu Prasad Gudla, Kenneth S. Stevens "Interfacing Synchronous and Asynchronous Domains for Open Core Protocol" 2014 IEEE.