Journal of Embedded Systems, 2014, Vol. 2, No. 2, 23-27
Available online at http://pubs.sciepub.com/jes/2/2/1

DOI:10.12691/jes-2-2-1

A Novel System-on-Chip (SoC) Integration Open Core Protocol (OCP) Bus with Multiple Master & Slave Support

Snigdharani Nath[1], Manas Ranjan Jena[2,*], Shilparani Panda[1]

[1]Deparment of ETC, SIET, DHENKANAL,ODISHA
[2]Deparment of ELTCE, VSSUT, BURLA, ODISHA
*Corresponding author: manas.synergy@gmail.com

Received May 10, 2014; Revised May 14, 2014; Accepted May 14, 2014

Abstract In this paper, we have designed a System-on-Chip (SoC) Integration with Open Core Protocol (OCP) both master and slave cores particularly in the burst and in the tag mode. The master core is responsible for initiating the communication on the bus. The slave core is the device that has been addressed by the master in order to establish the communication. Multiple OCP transfers can be linked into a burst transaction Cores such as DRAM controllers can supply the second related piece of data much faster than the first Bursts allow a target to know that there are more transfers coming, so it can pre-fetch. Tags allow out-of-order return of responses and out-of-order commit of write data. In IP core plug-and-play reuse, cores need to be coupled from one another using a clearly specified core interface protocol. The core must be portable from one SOC design to the next without integration rework. Taking advantage of the increasing density of IC process technologies remains extremely dependant on a formidable challenge. Adapting cores from chip design to chip design to make them fit with the rest of the system-on-a-chip (SOC) has become for a while a totally inefficient and unproductive methodology. Each time a core is to be integrated into a new system the system integrator is hampered by massive rework that reduces productivity.

Keywords: *System-on-Chip (SoC), interface, IPs, Open Core Protocol (OCP), FIFO*

Cite This Article: Snigdharani Nath, Manas Ranjan Jena, and Shilparani Panda, "A Novel System-on-Chip (SoC) Integration Open Core Protocol (OCP) Bus with Multiple Master & Slave Support." *Journal of Embedded Systems*, vol. 2, no. 2 (2014): 23-27. doi: 10.12691/jes-2-2-1.

1. Introduction

The Open Core Protocol, initially introduced by Sonics and widely known as OCP, has been pioneering that design methodology for several years. OCP is effectively core-centric and thus applicable to all on-chip interconnection systems. As a natural result of this success, the OCP specification has been moved under governance of the OCP-IP consortium. This association has been created in 2001 and is driven by a pool of semiconductor industry leaders, including Nokia, Texas Instruments, STMicroelectronics, UMC and Sonics, the original founder. A lot of other major players have joined the consortium since its creation. Hence today more than 70 member companies are merging their effort for the widest OCP adoption [1].

Multiple OCP transfers can be linked into a burst transaction Cores such as DRAM controllers can supply the second related piece of data much faster than the first. Tags are available in the OCP interface to control the ordering of responses [2].

1.1. Feature

1.1.1. 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 and contains no multi-cycle timing paths. All signals other than the clock are strictly point-to-point.

1.1.2. Bus Independence

A core utilizing the OCP can be interfaced to any bus. A test of any bus-independent 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 OCP compliant masters to pass a command field across the OCP that the bus interface logic converts into an arbitration request sequence [3].

The continuous innovation of semiconductor technology enables more complex System-on-Chip (SoC) designs. Tens, even hundreds of intellectual properties (IPs) are integrated into an SoC to provide various functions, including communications, networking, multimedia, storage, etc. An increasing number of electronic devices such as mobile phones, digital media players, digital TVs, are designed and manufactured using SoCs. The usually short market life cycle of these electronic devices, however, does not allow a long schedule for chip designers to integrate a such complex SoC [4].

1.2. Open Core Protocol

The OCP defines a point-to-point interface between two communicating entities such as IP cores and bus interface modules (bus wrappers). One entity acts as the master of the OCP instance, and the other as the slave. Only the master can present commands and is the controlling entity. The slave responds to commands presented to it, either by accepting data from the master, or presenting 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 - one where the first entity is a master, and one where the first entity is a slave [5].

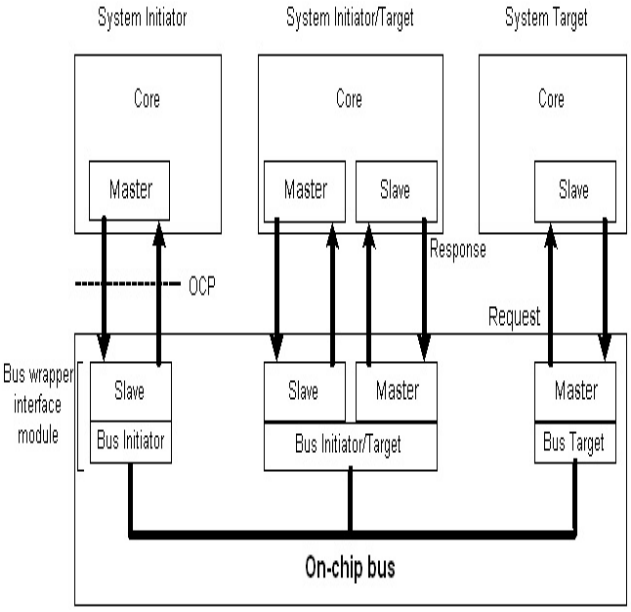

Figure 1.1. OCP Instances

2. Implementation & Verification

2.1. Block Diagram

Figure 2.1 shows the system block diagram. It shows how input is OCP interfaced with the memory. Here the data from the test bench is communicated through OCP interface towards destination. The destination is a RAM(Random Access Memory). The asynchronous FIFO will act as a buffer between OCP interface & memory.

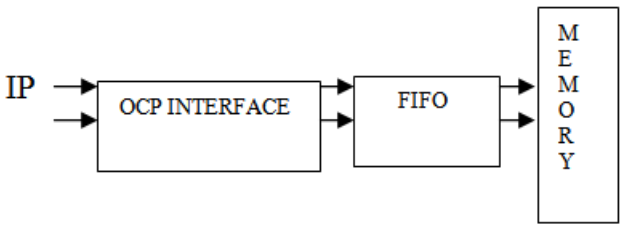

Figure 2.1. Block Diagram

2.1.1. OCP Interface

2.1.1.1. Sequential Master

The first example is a medium-throughput, high-frequency master design. To achieve high frequency, the implementation is a completely sequential (that is, Moore state machine) design.

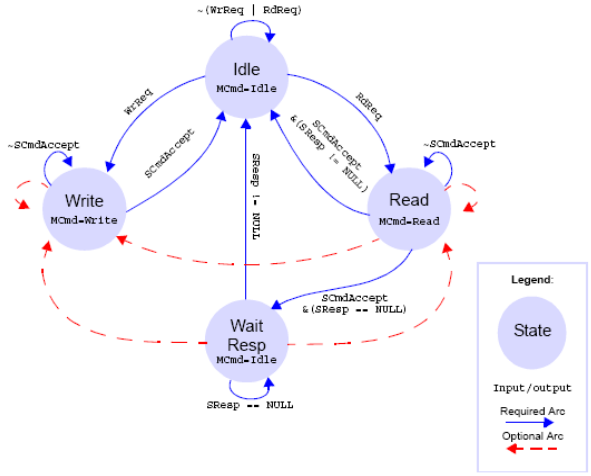

Figure 2.2. State Diagram of OCP Master

Since this is a Moore state machine, the outputs are only a function of the current state. The master cannot begin a request phase by asserting MCmd until it has entered a requesting state (either write or read), based upon the WrReq and RdReq inputs. In the requesting states, the master begins a request phase that continues until the slave asserts SCmd Accept. At this point (this example assumes write posting with no response on writes), a Write command is complete, so the master transitions back to the idle state. In case of a Read command, the next state is dependent upon whether the slave has begun the response phase or not.

Since MResp Accept is not enabled in this example, the response phase always ends in the cycle it begins, so the master may transition back to the idle state if SResp is asserted. If the response phase has not begun, then the next state is wait resp, where the master waits until the response phase begins. The maximum throughput of this design is one transfer every other cycle, since each transfer ends with at least one cycle of idle. The designer could improve the throughput (given a cooperative slave) by adding the state transitions marked with dashed lines. This would skip the idle state when there are more pending transfers by initiating a new request phase on the cycle after the previous request or response phase. Also, the Moore state machine adds up to a cycle of latency onto the idle to request transition, depending on the arrival time of WrReq and RdReq. The benefits of this design style include very simple timing, since the master request phase

outputs deliver a full cycle of setup time, and minimal logic depth associated with Sresp [6].

2.1.1.2. Sequential Slave

An analogous design point on the slave side is shown in Figure 2.3. This slave's OCP logic is a Moore state machine. The slave is capable of servicing an OCP read with one Clk cycle latency. On an OCP write, the slave needs the master to hold MData and the associated control fields steady for a complete cycle so the slave's write pulse generator will store the desired data into the desired location. The state machine reacts only to the OCP (the internal operation of the slave never prevents it from servicing a request), and the only non-OCP output of the state machine is the enable (WE) for the write pulse generator [7].

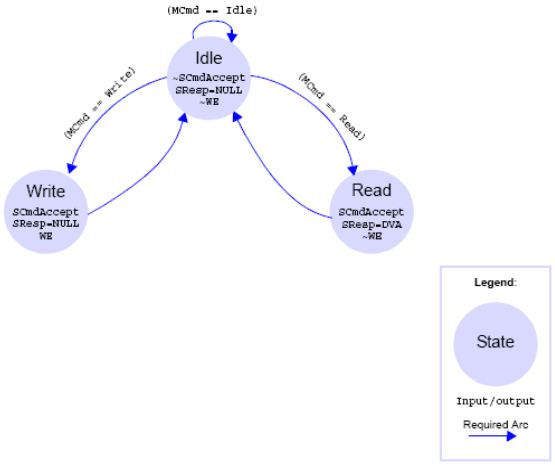

Figure 2.3. State Machine of OCP Slave

The state machine begins in an idle state, where it de-asserts Scmd Accept and SResp. When it detects the start of a request phase, it transitions to either a read or a write state, based upon MCmd. Since the slave will always complete its task in one cycle, both active states end the request phase (by asserting SCmd Accept), and the read state also begins the response phase. Since Mresp Accept is not enabled in this example, the response phase will end in the same cycle it begins. Writes without responses are assumed so SResp is NULL during the write state. Finally, the state machine triggers the write pulse generator in its write state, since the request phase outputs of the master will be held steady until the state machine transitions back to idle. Since the outputs depend upon the state machine, the sequential OCP slave has attractive timing properties. It will operate at very high frequencies (providing the internal logic of the slave an run that quickly). This state machine can be extended to accommodate slaves with internal latency of more than one cycle by adding a counting state between idle and one or both of the active states [8].

2.1.2. Asynchronous FIFO

An asynchronous FIFO refers to a FIFO design where data values are written to a FIFO buffer from one clock domain and the data values are read from the same FIFO buffer from another clock domain, where the two clock domains are asynchronous to each other. Asynchronous FIFOs are used to safely pass data from one clock domain to another clock domain.

The write pointer always points to the next word to be written; therefore, on reset, both pointers are set to zero, which also happens to be the next FIFO word location to be written. On a FIFO-write operation, the memory location that is pointed to by the write pointer is written, and then the write pointer is incremented to point to the next location to be written. Similarly, the read pointer always points to the current FIFO word to be read. Again on reset, both pointers are reset to zero, the FIFO is empty and the read pointer is pointing to invalid data (because the FIFO is empty and the empty flag is asserted). As soon as the first data word is written to the FIFO, the write pointer increments, the empty flag is cleared, and the read pointer that is still addressing the contents of the first FIFO memory word, immediately drives that first valid word onto the FIFO data output port, to be read by the receiver logic. The fact that the read pointer is always pointing to the next FIFO word to be read means that the receiver logic does not have to use two clock periods to read the data word. If the receiver first had to increment the read pointer before reading a FIFO data word, the receiver would clock once to output the data word from the FIFO, and clock a second time to capture the data word into the receiver. That would be needlessly inefficient. The FIFO is empty when the read and write pointers are both equal. This condition happens when both pointers are reset to zero during a reset operation, or when the read pointer catches up to the write pointer, having read the last word from the FIFO[9].

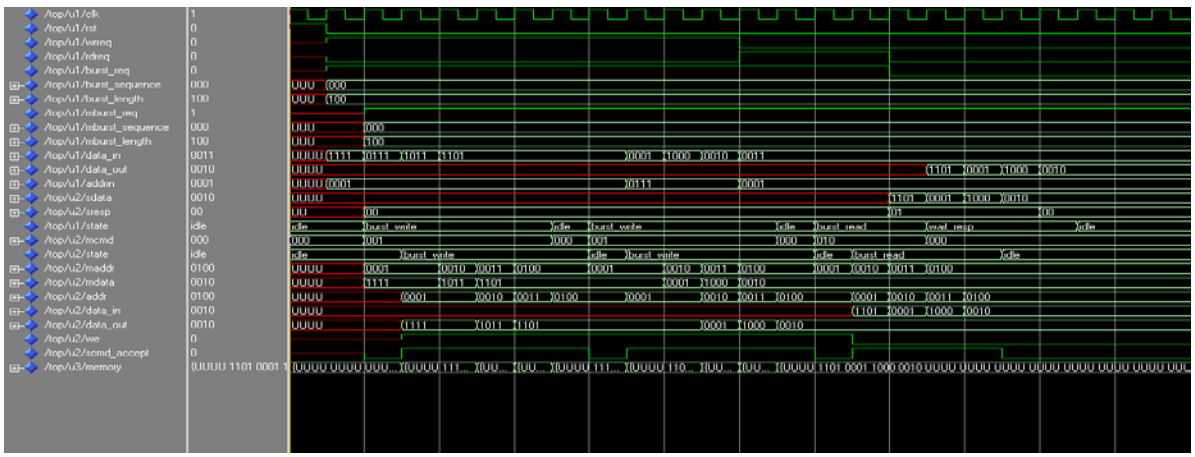

Figure 3.1. Simulation result of Burst Operation

A FIFO is full when the pointers are again equal, that is, when the write pointer has wrapped around and caught up to the read pointer. This is a problem. The FIFO is either empty or full when the pointers are equal. One design technique used to distinguish between full and empty is to add an extra bit to each pointer. When the write pointer increments past the final FIFO address, the write pointer will increment the unused MSB while setting the rest of the bits back to zero as shown in Figure 3.1 (the FIFO has wrapped and toggled the pointer MSB). The same is done with the read pointer. If the MSBs of the two pointers are different, it means that the write pointer has wrapped one more time that the read pointer. If the MSBs of the two pointers are the same, it means that both pointers have wrapped the same number of times [10].

3. Simulation Results & Analysis

The model is designed using VHDL code & simulated using Xilinx 11.1i. Figure 3.1 shows OCP burst mode operation which indicates simplified timing analysis that is useful for physical design. Figure 3.2 shows OCP tag mode operation which indicates reduced clock period which enhances speed of data transfer. Figure 3.3 shows asynchronous FIFO operation which indicates fastest data operation compared to above two cases.

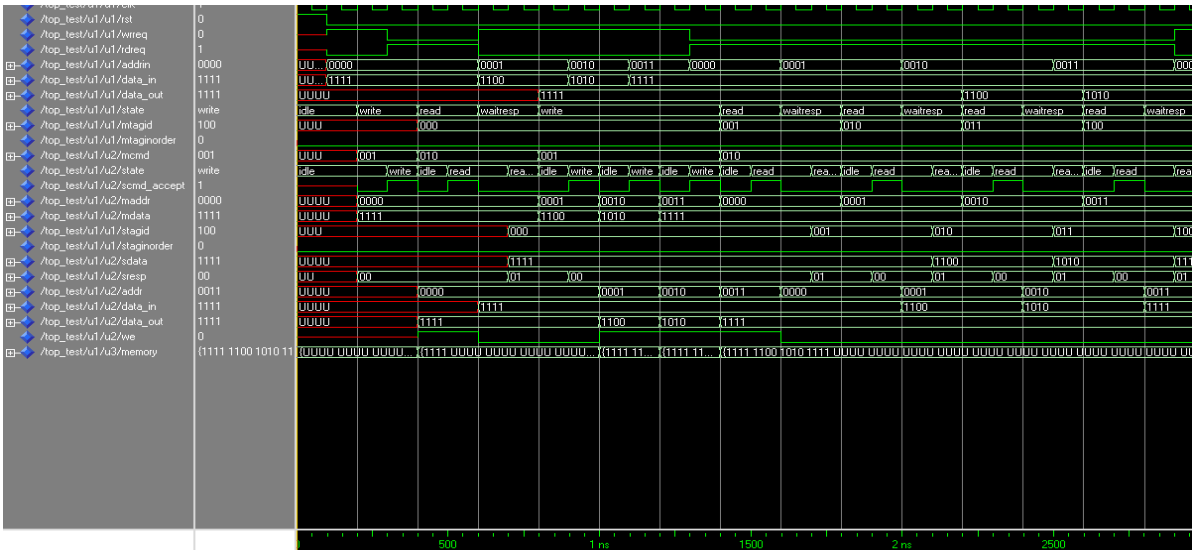

Figure 3.2. Simulation result of Tag Operation

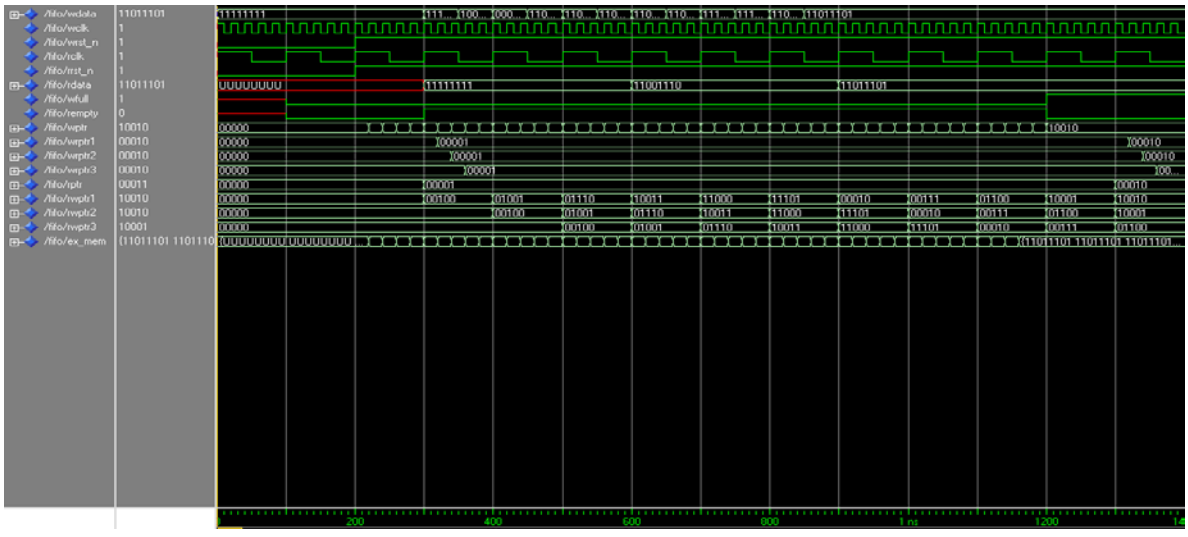

Figure 3.3. Simulation result of FIFO

4. Conclusion

The design of a large scale SoC is becoming challenging not only due to its complexity, but also the use of a large amount of IPs. A consensus interface standard for IP cores is becoming important and even inevitable for a successful SoC design. OCP, with its openness, collaborative, not-for-profit nature, and inherent large industry member base, is quickly becoming a feasible and preferable solution over a close or an in-house standard.

This paper presents the concepts behind the Open Core Protocol, a core interface aimed at capturing all of a core's system communication needs, allowing the core to be decoupled from the on-chip interconnect system. The Open Core Protocol is a simple synchronous, point-to-point interface between a core and the rest of the system. It is scalable and configurable to match the communication requirements associated with different IP cores.

Acknowledgement

The authors sincerely thank to the H.O.D & all the staff of Dept. of ETC, SIET, DHENKANAL, ODISHA for constant encouragement and support directly or indirectly.

References

[1] Jari Nurmi, "Processor design: System-on-chip computing for ASICS & FPGAS", Springer 1st edition, 2007.

[2] Sonics, Inc. Open Core Protocol Specification Reference Guide Version 2.0.

[3] Wolf-Dietrich Weber "Enabling Reuse via an IP Core-centric Communications Protocol: Open Core Protocol." Sonics, Inc.

[4] Farzad Nekoogar & Faranak Nekoogar "From ASICS to SOCS:A practical approach", Pearson Educaton, 2003.

[5] Satoshi Komatsu, Shota Watanabe "Protocol Transducer Synthesis using Divide and Conquer approach" IEEE, 1-4244-0630-7/07-2007.

[6] Douglas J Smith, HDL Chip Design.

[7] Chih-Wea Wang, Chi-Shao Lai, Chi-Feng Wu, Shih-Arn Hwang, and Ying-Hsi Lin "On-chip Interconnection Design and SoC Integration with OCP" IEEE 1-4244-1617-2/08-2008

[8] Bhaskar, J.A, "VHDL Primer", Pearson Education thirteenth edition-2004.

[9] Clifford E. Cummings "Simulation and Synthesis Techniques for Asynchronous FIFO Design" Sunburst Design, Inc.

[10] Carl Harmacture, Computer Architecture.