International Journal of Control and Automation
Vol. 10, No. 1 (2017), pp.13-22
http//dx.doi.org/10.14257/ijca.2017.10.1.02

The Efficient On-Chip Bus Architecture for High-Performance SoC Design

Fred Adu Kumi[1], Seungyong Park[2] and Kwangki Ryoo[*]

Department of Information and Communication Engineering, Hanbat National University, Daejeon 305-719, South Korea
[1]*kumiadufred@gmail.com*, [2]*srrr.kr@gmail.com*, and [*]*kkryoo@gmail.com*

Abstract

A high performance on-chip bus for system-on-chip design is presented in this paper. The proposed bus employs the use of two channels for data transfer. This allows for simultaneous transactions to be performed. Each of these channels is built as a full-duplex bus, allowing concurrent read and write transactions. The arbiter is designed so that it allocates the bus resources to requesting masters depending on the availability of the resources. The preferred arbitration scheme used for this bus is TDMA (Time Division Multiple Access) although any arbitration scheme will work fine. The internal workings of the bus are kept from the bus interface which makes the bus interface simple. The proposed bus is designed at RTL with Xilinx ISE 14.7 and simulated with Modelsim SE-64 10.1c. Various tests are performed and the results are compared with AMBA AHB. Experimental results show an increase in bus efficiency in terms of transfer time. A 10.67% reduction in transfer time is realized in test case 1, while a 49.80% and 58.95 for that of test case 2 and 3 respectively. A complete SoC platform with the proposed bus as the communication channel was designed and implement with Virtex-4 FPGA.

Keywords*: On-chip bus, FPGA, SoC*

1. Introduction

With the advancement of process technology, the frequency and amount of data communication that occurs between IPs increases considerably [1]. In a SoC, the overall system performance depends heavily on efficient communication among functional components and computation among them [2]. AMBA AHB [3] and AXI [4] are popular bus protocols which provide high-performance interconnection of various modules known as IPs on a single chip. AMBA AHB mainly uses a shared bus architecture in which various IP's communicate through the shared channel. As more and more IP cores are connected, the efficiency of the system is degraded due to the fact that the available bandwidth has to be shared by all the IPs [5]. The efficiency of the bus is affected due to factors such as long wait cycles, bus transfer cycle and priorities in bus arbitration [6]. Most often the shared bus architecture does not guarantee efficient communication since only one transaction can be performed at a time [1]. AXI uses a crossbar architecture where each master has a separate connection to each slave. Multiple simultaneous transactions can be performed and hence AXI provides higher data bandwidth. Despite this, the crossbar architecture comes at a higher cost. This owes to the complexity of the interconnection and the many wires and hardware resources need to implement this topology.

We propose a bus architecture for high-performance SoC designs. The proposed bus architecture uses two channels through which data can be routed through. It resembles a 2x2 crossbar architecture but without dedicated connections to any modules. The channels are only granted to a requesting IP module by an arbiter

ISSN: 2005-4297 IJCA

which controls access to the channels, and allocates bus resources provided they are available.

2. Bus Protocol

2.1. Overview

The proposed bus is made up of two channels, capable of performing simultaneous transactions. Each channel consists of separate read and writes channels which allows simultaneous read and write transfers. A read or write channel consists of address, controls and data buses. A single pair of handshake signals is used for both address and data buses. Bus allocation is done by the arbiter who also determines which channel a master is assigned to.

2.2. Architecture

Figure 1 shows the architecture of the proposed bus. A central arbiter is responsible for granting a master access to a channel. The channel to which master may be granted depends on the availability of the channel. A signal from the master is used to request for bus access. The master-to-slave mux is used to route address, control and write data signals from a master to a slave. Select signals from the arbiter are used to select the appropriate address, control and data information coming from the masters in both channels to the slave selected by the decoder. The srouter is used to route the signals to the appropriate slave only. It is made up of AND gates and OR gates which select the channel which has valid address control and data signal for the selected slave and routes the data to it. Similarly, the slave-to-master mux routes read data and response signals from the slave to the requesting master. The mrouter also ensures that read data and response signals from the right channel is selected and routed to the right master. Hence data is not broadcasted to all slaves or masters. The master of slave does not need to care about the channel through which data is routed. These details are shielded from the master and slave interfaces.

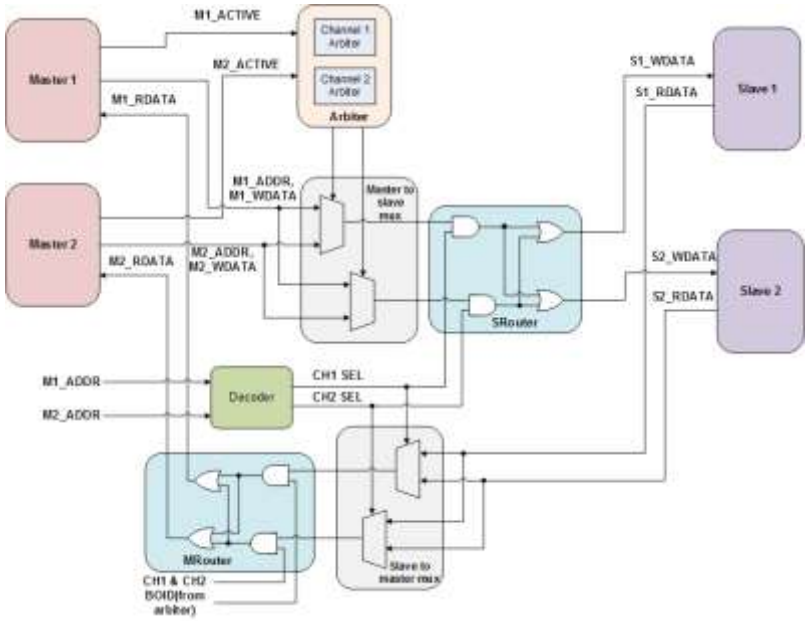

Figure 1. Bus Interconnection

2.3. Signal Description

Table 1 show the signals used in the proposed bus and a brief description of those signals.

Table 1. Signals Description

Signal	Source	Description
CLK	Clock source	All inputs are sampled on the rising edge of CLK
RESET_N	Reset source	Active low signal used to reset the bus
WR_ACTIVE	Master	Signals the arbiter that the master is ready to perform a write transaction and requires access to the bus.
WR_READY	Slave	Indicates that the slave is able to receive an address or data
WR_VALID	Master	Indicates that the master has valid address or data information on the channel
WR_ADDR	Master	Write address for initial transfer in a burst
WR_DATA	Master	Write data
WR_CTRL	Master	Write control signals. Includes burst length, size and type
RD_ACTIVE	Master	Signals the arbiter that the master is ready to perform a read transaction and requires access to the bus.
RD_READY	Slave	Indicates that the slave is ready to receive address. It also indicates that the slave has valid read data on the channel
RD_VALID	Master	Indicates that the master has valid address. It also indicates that the master is ready to receive read data
RD_ADDR	Master	Read address for initial transfer in a burst
RD_DATA	Slave	Read data
RD_CTRL	Master	Read control signals. Includes burst length, size and type
WR_BOID	Arbiter	Bus owner id. Shows the id of the master currently using the write bus
RD_BOID	Arbiter	Bus owner id. Shows the id of the master currently using the read bus
WR_SEL	Decoder	Indicates the selected slave for a write transaction and is used when routing information to the appropriate slave
RD_SEL	Decoder	Indicates the selected slave for a read transaction and is used when routing information to the appropriate slave

2.4. Arbitration

Arbitration is necessary to coordinate the activities of IPs in an on-chip bus. Different arbitration schemes have different performances in terms of data load and traffic [7]. The TDMA bus arbitration scheme used in this architecture is based on dynamic timeslot allocation used in [8] with some improvements. A master asserts its ACTIVE signal to request for the bus when it is ready to perform a transaction. When any of the channels is available, the arbiter grants the requesting master access to the available channel, and assigns a time slot to that master, with the

condition that the slave with which the master wants to communicate is not being accessed by another master on a different channel. With this, simultaneous transactions can occur on both channels provided different slaves are being accessed. Grant signals from the arbiter are used as selection signals in the master-to-slave mux. In this scheme, bus cycles are not wasted because the bus is only granted to masters who are ready to perform transactions. When the time allocated to a master is up and there is a current transaction which is yet to complete, the arbiter allows the transaction to complete before granting the bus to the next master. In the same way, if a transaction is completed before the time elapses, the master must de-assert its ACT signal to relinquish the bus access. The bus can then be allocated to the next master. *Figure 2* shows the block diagram of the arbiter.

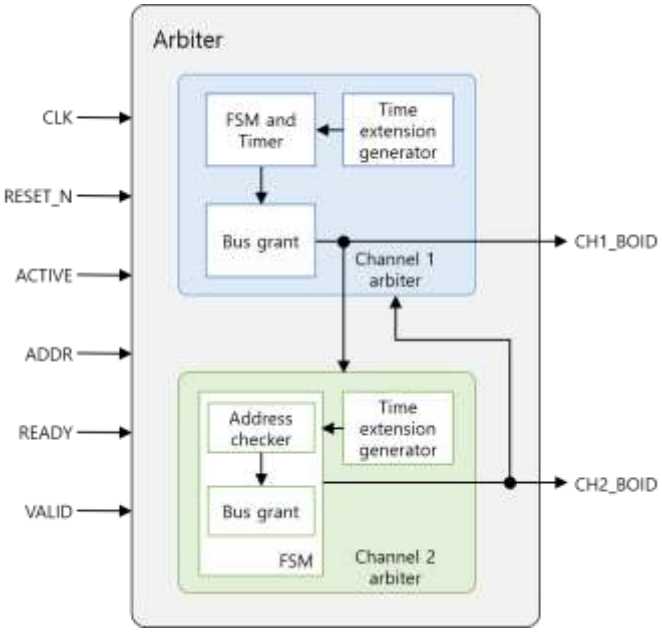

Figure 2. Block Diagram of Arbiter

In Channel 1 arbiter, an FSM (Finite State Machine) is used. The FSM begins by checking the ACTIVE signal of the first master, if it is high; it assigns the bus to the first master by setting the output CH1_BOID, to the id of that master. Once the master has access to the bus, it can perform any number of transactions as far as it still has the bus. The FSM moves to a state where a timer has begun. The timer begins counting till it reaches a pre-programmed number of cycles, which defines the time slot allowed for each master. When the time elapses, the arbiter checks to see if the master has begun another transaction which is yet to complete. If so, it allows the master to complete the transfer otherwise it moves on to the next bus master. If it's the turn of a master which does not have any transactions to perform, the arbiter moves to the next master in line in a round robin manner. This helps prevent wasted time slots since no time slot is generated for a master which is not ready to perform a transaction. The diagram in Figure 3 shows a simplified state diagram used in the arbiter. It shows arbitration between three masters. In state TS0, master 0 has the highest priority hence its ACTIVE signal is checked first before the rest of the masters. A state transition is not made until master 0 or any other master has its active signal high. It then moves to state WAIT0, where the timer starts counting. When the time elapses, master 1 which is the next in line is given the highest priority. If master 1 or none of the masters have their ACTIVE signal asserted, it moves to state TS1, otherwise the master is granted access immediately

and the state moves to state WAIT1. This happens in all the other states until it gets to state TS0 then it all begins again. A master may relinquish its access to the bus by de-asserting its ACTIVE signal. In this case, the bus is granted to the next requesting master. Bus masters are encouraged to relinquish bus access to improve the effectiveness of the entire system. Channel 2 arbiter checks for the bus owner of channel 1 and excludes it from its arbitration process. It then checks for the slave with which the requesting master wishes to communicate. If it is different from the slave in communication with the bus owner in channel 1, then the master is granted access to that slave through channel 2. The output of channel 1 and channel 2 arbiters is CH1_BOID and CH2_BOID respectively, which is the id of the masters occupying those channels at that point in time. These signals are used to route the appropriate address, control and data information from a master to the slave it wishes to communicate with.

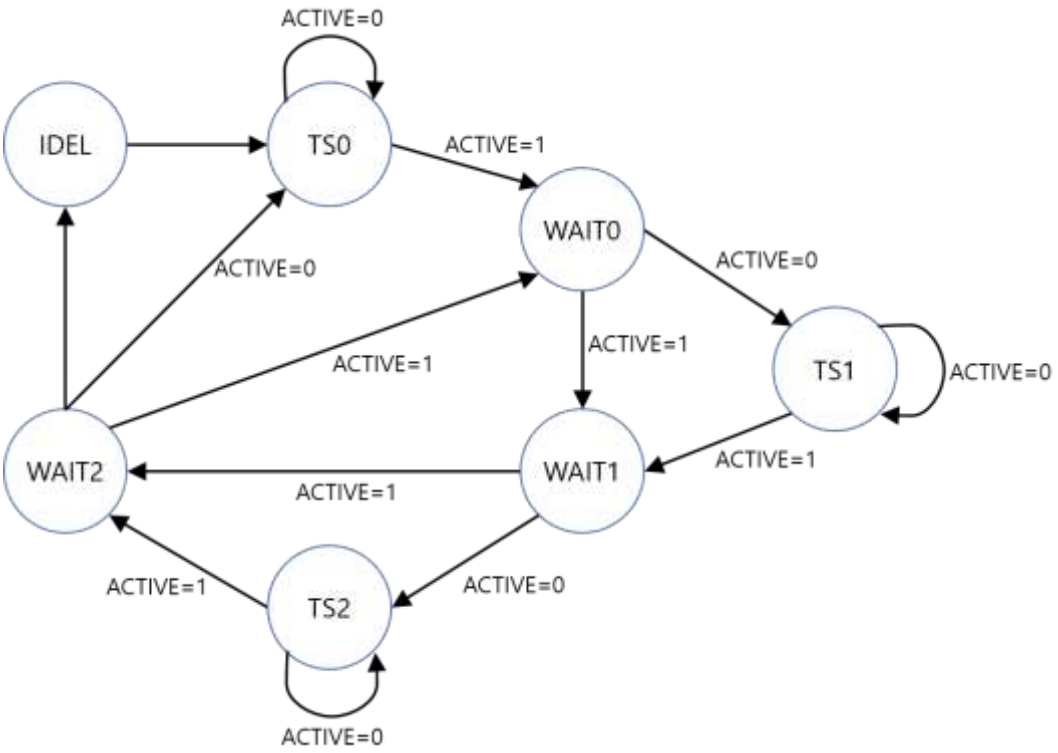

Figure 3. State Diagram of Arbiter

The master can only use the bus for the predetermined timeslot and can relinquish the bus if it has not exhausted all the timeslot before its transactions are completed. In the course of a transfer, if the timeslot is exhausted, the arbiter allows the transaction to complete before allocating the bus to another master. This prevents invalid data from been transferred. Grant signals from the arbiter are used as selection signals in the master-to-slave mux. In this scheme, bus cycles are not wasted because the bus is only granted to masters who are ready to perform transactions. Figure 4 shows a waveform for arbitration.

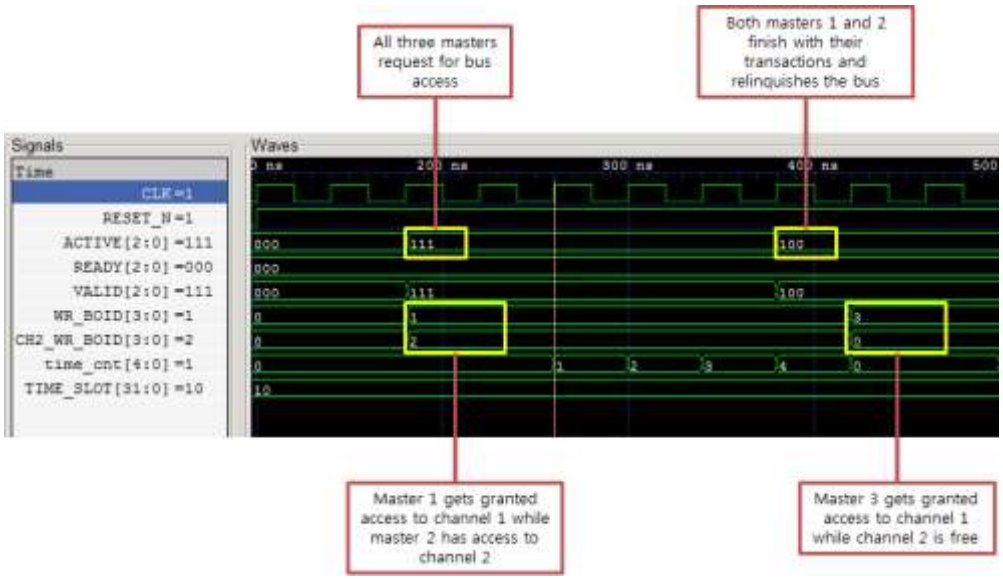

Figure 4. Waveform Showing Arbitration

2.5. Bus Operation

The proposed bus is made up of write address and control bus, read address and control bus, write data bus, read data and response bus. A simple handshaking process is used between a master and slave after the master has been granted access to the bus. The same handshaking signals (READY and VALID) are used for both address and data phase. Control signals include burst size, burst length and burst type. A bus operation is initiated by the master, by making a request to a slave. The slave, is responsible for responding to requests from the master. To perform a write transaction, the master drives address and control information on the bus and asserts its VALID signals. This begins the handshaking process for address and control information. The ACTIVE signal is also asserted to request for bus access from the arbiter. When access is granted to the master, its address and control information is routed to the target slave through one of the channels. The slave accepts the transfer and responds to the master. A READY signal from the slave completes the handshaking process. The master can then proceed with data transfer in a write transaction using the same handshaking signals. The READY signal is kept high during data transfer in a write transaction. Figure 5 show an example waveform for a write transaction.

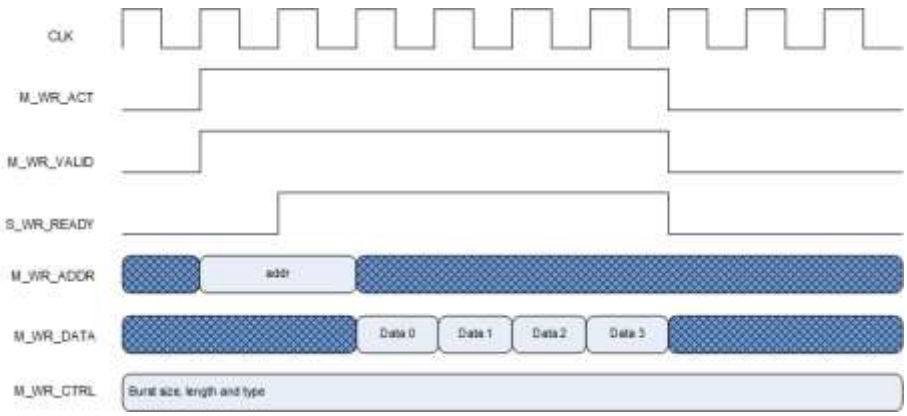

Figure 5. Waveform Showing a Write Transaction

In a read transaction, the VALID signal from the master, must remain high after the address is transferred in order to receive read data from the slave. The slave then uses its READY signal to indicate valid read data. A waveform for a read transaction is shown in Figure 6.

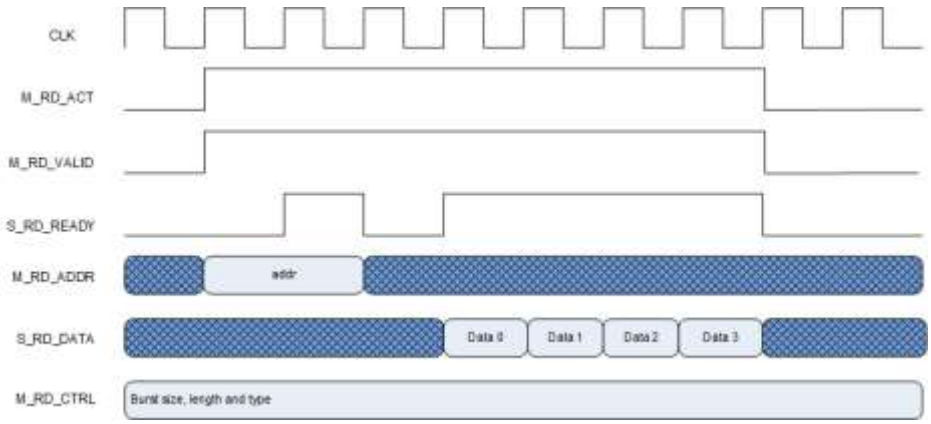

Figure 6. Waveform Showing a Read Transaction

3. Implementation and Results

The proposed bus was designed at RTL with Verilog HDL using Xilinx ISE 14.7 design tool and simulated with ModelSim SE-64 10.1c. A real hardware test was performed by implementing a SoC platform using the proposed bus as the underlying communication architecture in the SoC platform. The SoC architecture shown in Figure 7 was designed and simulated for testing purposes.

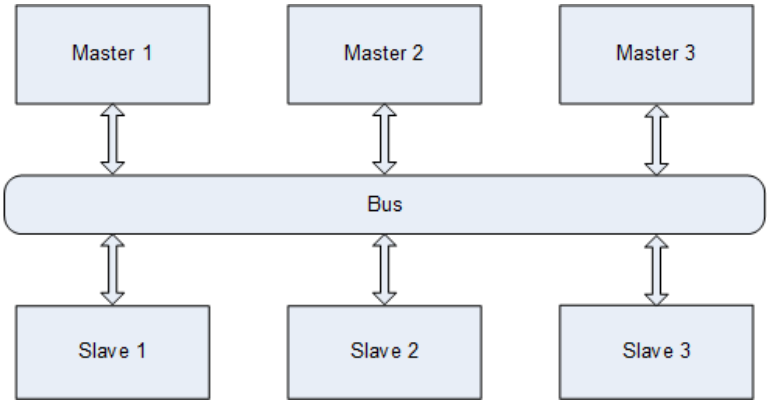

Figure 7. SoC Architecture for Simulation

The following test cases were used:

Test 1: Master 1 performs 256 byte read after write in burst of 4 and 8 to slave 3.

Test 2: Master 2 and master 3 perform 256 bytes write and read in burst of 4 and 8 to slave 1 and slave 2 respectively.

Test 3: Master 1 performs 256 bytes write and read in burst of 4 and 8 to slave 1 whiles master 2 and master 3 perform write and read in burst of 4 and 8 to slave 2.

All simulations were performed at a clock frequency of 50 MHz. The results of all the tests compared with AHB are shown in Table 2.

Table 2. Results of Test Cases

Test Case	AHB (ns)	Proposed Bus (ns)	Effieciency (%)
Test 1	16480	14720	10.67
Test 2	29320	14720	49.80
Test 3	45600	18720	58.95

Simulation results show that in all the test cases, the proposed bus has a greater efficiency than AHB. In test 3, it is seen that the transfer time is reduced by 58.96% which is approximately equal to 2.4 times that of AHB. This is due to master 1 using one channel whiles masters 2 and 3 use the other channel since master 1 accesses a different slave from both masters 2 and 3.

For the real board test, an image processing SoC platform, shown in Figure 8 was used. The image processing controller performs image rotation, scaling, changes to different color format and switching between displays. All these are controlled by the processor through the UART. All modules were designed at RTL with Xilinx ISE 14.7 and the proposed bus was used as the underlining communication channel. The design was implemented on HBE-SoC-IPD test board equipped with Virtex-4 FPGA device.

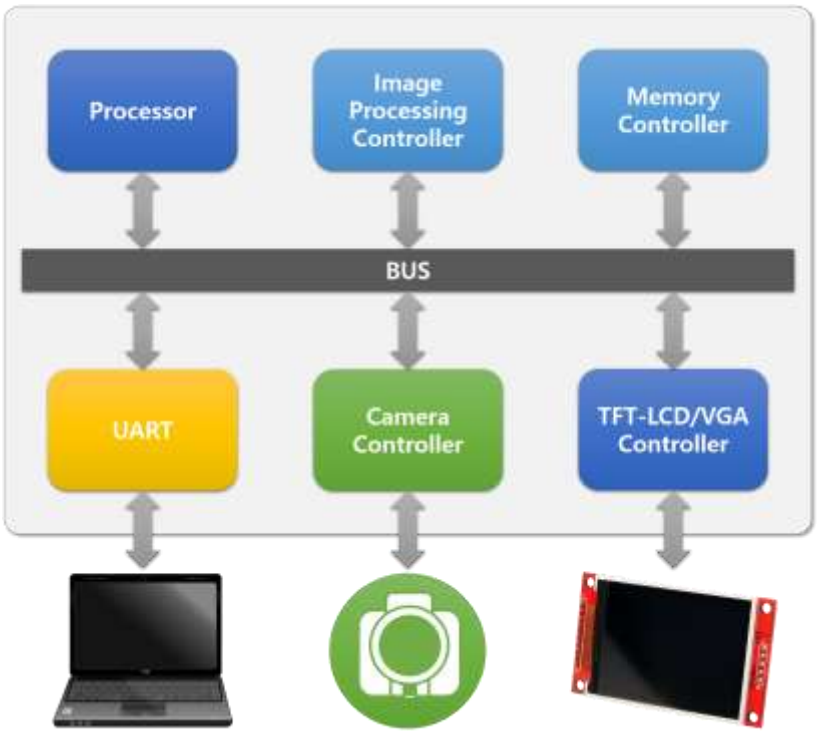

Figure 8. SoC Platform for Hardware Emulation

Some results of the hardware emulation are shown below.

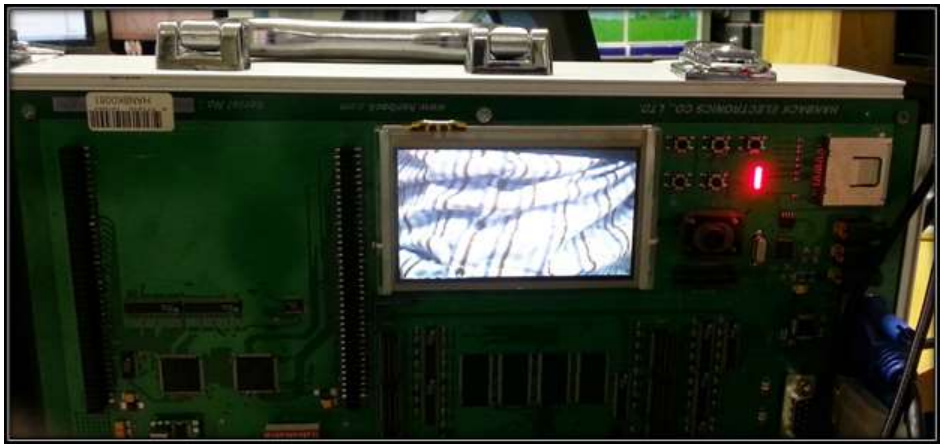

Figure 9. TFT-LCD Shows x2 Scaled Image

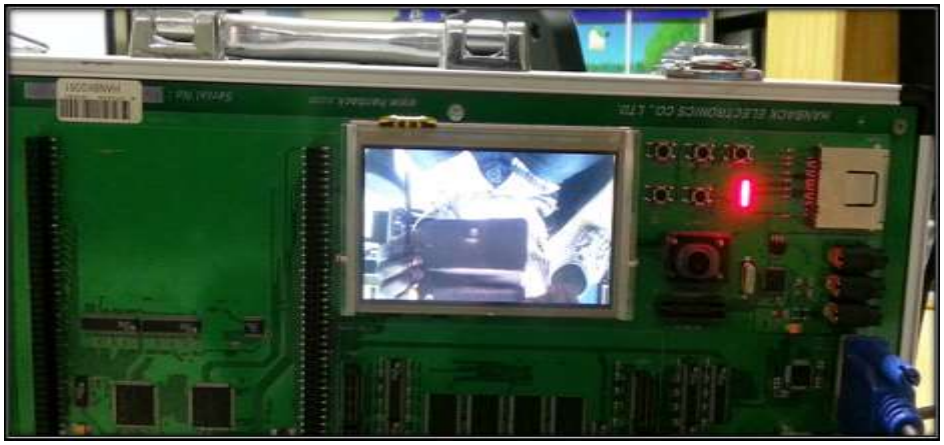

Figure 10. TFT-LCD Shows a Rotated Image

4. Conclusion

An on-chip bus architecture which provides high-performance communication architecture for SoC designs has been described in this paper. The proposed bus is made up of two independent channels, which allows simultaneous transactions to occur. The arbitration scheme used was TDMA where time slots (number of cycles) are assigned in a round robin manner to masters which are ready to perform a transaction. Simulation results have shown that the proposed bus used less time in the three tests that were performed compared to AHB, hence it has a higher efficiency than AHB. The proposed bus is suitable for small to medium scale SoC designs with data-intensive requirements and lower cost.

Acknowledgement

This research was supported by the MSIP (Ministry of Science, ICT and Future Planning), Korea, under the Global IT Talent support program (IITP-2016-R0134-16-1019) and Human Resource Development Project for Brain scouting program(IITP-2016-R2418-16-0007) supervised by the IITP(Institute for Information and Communication Technology Promotion).

References

[1] C. Y. Chang, Y. J. Chang, K. J. Lee, J. C. Yeh, S. Y. Lin and J. L. Ma, "Design of on-chip bus with OCP interface", VLSI Design Automation and Test (VLSI-DAT), 2010 International Symposium, Hsin Chu, Taiwan, **(2010)**.
[2] D. Shanthi and R. Amutha, "Design of efficient on-chip communication architecture in MpSoC", Recent Trends in Information Technology (ICRTIT), 2011 International Conference, Chennai, Tamil Nadu, India, **(2011)**.
[3] ARM Limited, "AMBA Specification", Rev. 2.0 Axis, Sunnyvale, CA, USA, **(1999)**.
[4] ARM Limited, "AMBA AXI Protocol Specification", Axis, Sunnyvale, CA, USA, **(2003)**.
[5] L. Ruibing, C. Aiqun and K. Cheng-Kok, "SAMBA-Bus: A High Performance Bus Architecture for System-on-Chips", Very Large Scale Integration (VLSI) Systems, IEEE Transactions, vol. 15, no. 1, (**2007**), pp. 69-79.
[6] K. P. Lee, Y. H. Joung and S. J. Kang, "Bus Arbitration Considering Waiting Cycle", Journal of the Korea Institute of Information and Communication Engineering, vol. 18, no. 11, **(2014)**, pp. 2703-2708.
[7] K. P. Lee and S. Y. Koh, "Performance Improvement of 2nd Arbitration in the Lotterry Bus Arbitration Method", Journal of the Korea Institute of Information and Communication Engineering, vol. 17, no.8, **(2013)**, pp.1879-1884
[8] T. D. Richardson, C. Nicopoulos, D. Park, V. Narayanan, Y. Xie, C. Das and V. Degalahal, "A hybrid SoC interconnect with dynamic tdma-based transaction-less buses and on-chip networks", Proceedings of the 19th International Conference on VLSI Design held jointly with 5th International Conference on Embedded Systems Design. Washington, DC, USA: IEEE Computer Society, **(2006)**.

Authors

Fred Adu Kumi, he received a BSc Degree in Computer Science from Kwame Nkrumah University of Science and Technology, Ghana, in 2012. He is currently pursuing a MENG Degree in Information and Communication Engineering at Hanbat National University, South Korea. His research interests include SoC Design and Verification Platforms and Communication Architectures.

Seungyong Park, he received a BS Degree and MENG Degree in Information and Communication Engineering from Hanbat National University, South Korea, in 2010 and 2012 respectively. He is currently pursuing a PhD Degree in Information and Communication Engineering at Hanbat National University, South Korea. His research interests include SoC Design and Verification Platforms, Image Signal Processing and Multimedia Codec Design.

Kwangki Ryoo, he received BS, MS and PhD Degrees in Electronic Engineering from Hanyang University, South Korea in 1986, 1988 and 2000 respectively. From 1991 to 1994, he was an Assistant Professor at the Military Academy in South Korea. From 2000 to 2002, he worked as a Senior Researcher at ETRI, South Korea. From 2010 to 2011, he was a Visiting Professor at University of Texas at Dallas. Since 2003, he has been a Professor at Hanbat National University, South Korea. His research interests include Engineering Education, SoC Design and Verification Platforms, Image Signal Processing and Multimedia Codec Design.

