International Journal of Technology 10(2): 309-319
ISSN 2086-9614

DESIGN AND IMPLEMENTATION OF A DIRECT MEMORY ACCESS CONTROLLER FOR EMBEDDED APPLICATIONS

Mohammed Altaf Ahmed[1*], Abdullah Aljumah[1], M. Gulam Ahmad[1]

[1]*Department of Computer Engineering, College of Computer Engineering & Sciences, Prince Sattam Bin Abdulaziz University, Alkharj 11942, Saudi Arabia*

(Received: October 2017 / Revised: September 2018 / Accepted: February 2019)

ABSTRACT

In this paper, we propose a design and implementation of a Direct Memory Access Controller (DMAC) as a part of an SoC. The main purpose of the DMAC design is to integrate it into a System on a Chip (SoC) for the exchange of a large volume of data between the memory and peripherals at high speed. The proposed DMAC works on Advanced Microcontroller Bus Architecture (AMBA) specifications. Internally, these specifications define two buses, Advanced High-performance Bus (AHB) and Advanced Peripheral Bus (APB). The Direct Memory Access (DMA) controller functions as the bridge between AHB and APB and allows them to work in parallel. It works either in buffer or non-buffer data transfer mode, according to the peripheral speed. This is synchronized with an asynchronous FIFO. Fast data reads can be achieved by using an AMBA based DMA controller with a processor in the SoC. This means that the DMAC provides a high volume of data transfer. Hence, the proposed DMAC is a better option for high volumes of data, as well as for timing. It can be concluded that if using this AMBA-based DMA controller the issues of high speed and high volume data have been resolved. Comparison is made with ARM processors, such as Cortex A8 and ZC702, and design comparison with Xilinx DMA is also made. The DMAC is viewed as a more appropriate choice.

Keywords: AMBA-based DMA; Data transfer rate; DMA; DMA Controller; FPGA; SoC

1. INTRODUCTION

The purpose of DMA is to reduce the load on the processor. As the term indicates, it accesses memory directly for peripheral devices. If DMA is used with a processor, then data access from the memory is made by the DMA instead of the processor. DMA permits peripheral devices to access the memory directly, without dependency on the processor. Hence, the processor can execute other tasks concurrently, while the DMA is accessing the memory. As such, the overall performance of the system is boosted (Aljumah & Ahmed, 2016). DMA appears to be an easy concept, but system implementation with other hardware subsystems is cumbersome. DMA has many other vital applications, such as network cards, graphics cards and disk drive controllers.

In computer systems, DMA plays a significant role in accessing the memory, and is a vital part or entity of an embedded system. Moreover, it plays a crucial role in SoC systems, providing fairly good speed for transferring data to externally connected peripheral devices. The performance of DMA improves when it works with a bus architecture. This is available in the design literatures of DMA using bus architectures. Intel designed the first DMA, which known as IC 8237. In 1981, IBM used DMA IC 8237 for the first time in its products.

*Corresponding author's email: m.altaf@psau.edu.sa, Tel. +966-115888347
Permalink/DOI: https://doi.org/10.14716/ijtech.v10i2.795

It uses bus architecture, with industry standard architecture (ISA) to improve its performance and was designed to transfer data between the system memory and peripherals (Zayati et al., 2012; Oded, 2012). The DMA design had four channels and transferred 1.6 megabytes of data every second. The individual channels had 64 kilobytes of memory address and were capable of transferring 64 KB of data with a single programming instruction (Barry, 1997). Initially, the system bus and ISA bus were identical. As the CPU of the IBM AT was cloned to work at higher frequencies than an ISA expansion bus, they were separated. An ISA bridge was used for separation (Hou, 2013).

In 1992, the Peripheral Component Interface (PCI), a new bus architecture, was introduced. Communication between the PCI and ISA was through the board. Subsequently, a PCI to ISA adapter was recommended (Jinbiao, 2013); for this reason, the basic architectural design used to contain an adapter block of logic. The hardware block, therefore, includes PCI bus interface circuit design, ISA bus interface circuit design and an I/O finding module logic block to find the peripherals (Hou, 2013). The PCI-bus architecture works on the principle of the master and the master will only have full control of the bus at a time. Only using certain arbitrary techniques, multiple devices can access the bus. An enhancement in the bus architecture took place when the concept of packet switching in full duplex mode was used to interface multiple devices and system memory. This enhanced the bus architecture and was termed PCI express (PCIe) (Li et al., 2009; Anand, 2013; Shengwei, 2016). It had an x_1 link pair in its architecture, which contained channels for transmitting and receiving separately. Therefore, bandwidth was doubled compared to the previous architecture.

Apart from these buses for DMA operation, embedded products employ a very useful specific bus architecture in SoCs, known as advanced microcontroller bus architecture (AMBA), which is a registered trademark (ARM, 2017) in the IC industry for Advanced Reduced Instruction Set Computer (RISC) Machines (ARM Ltd). Subsequently, by the end of 1997 the first native AMBA interfaces with cache memory cores were introduced. AMBA was an interconnect specification (on-chip), which was used for managing and connecting the various functional blocks under the SoC. It provided support to various controllers, processors, multiprocessor systems and peripherals and was an open standard system in the industry. Two types of bus system were defined in the specification of AMBA architecture, namely AHB and APB. Nowadays, AMBA is widely used in Application Specific Integrated Circuits (ASIC)-based and SoC-based modern mobile devices. Such products use state machines separately for transmission and reception, achieving moderate data transfer rate (Berawi, 2013; Ahmed et al., 2015; Ejidokun et al., 2018).

In this proposed research study, we designed a direct memory access controller (DMAC) for embedded system-based products, working on advanced microcontroller bus architecture (AMBA). DMA performance and data transfer speed for large volumes of data improve when it uses bus architecture such as AMBA. In this way, we first improved the performance of the DMA controller and then used this DMA engine in the embedded system to improve the performance of the system and of the processor. The performance characteristics of the embedded processor with the DMA controller are consistently better than those without the controller. These performance characteristics are presented in an article on Cypress Semiconductor (Gupta & Natarajan, 2010).

The proposed DMA engine was found to have superior speed and data transfer rates compared to the existing DMA controller used in Xilinx Embedded FPGA (Faezeh & Mohammad, 2017) and the Xilinx logic core IP XPS Central DMA Controller (Xilinx, 2010). A comparison is made in the discussion in section 4. Subsequently, it is explained how the proposed DMAC is better at transferring data compared to the existing ARM cortex-A8 processor and ZC 702 processor. The design frequency of the proposed DMAC is achieved as 478 MHz, with the use of only 167 LUTs

as area occupation in the FPGA Spartan 3 device. This is better than that of the existing DMA shown in Table 4 and of the xilinx and cortex-A8 processor. Therefore, the proposed design provides rich features, while keeping the gate count low. This DMA controller can be used for transferring data, while keeping the processor in a very light state when integrated with the SoC. The design is implemented in Field Programmable Gate Array (FPGA) Vertex family.

The paper is organized as follows: section 2 consists of the methodology, while section 3 presents the performance and results. Sections 4 makes an analysis and discussion to suggest the scope for further research work. Section 5 is the conclusion and is followed by the list of references.

2. METHODOLOGY

2.1. DMA Principle

Direct memory access (DMA) is a technique for transferring data between main memories and requesting input-output devices, and vice versa, independent of the processor. The principle of accessing data independently for the input-output device from the main memory is shown in Figure 1. DMA can be used for intra-chip data transfer in multi-core CPUs and for copying data transfer between the storage systems. The hardware entity which performs the address generation and reading and writing operations is called a DMA Controller (DMAC). For data transfer, the DMAC is configured by the processor, while continuing its individual function. At the instant the processor is granted the system bus, the DMAC performs all the functions of the input and output device.

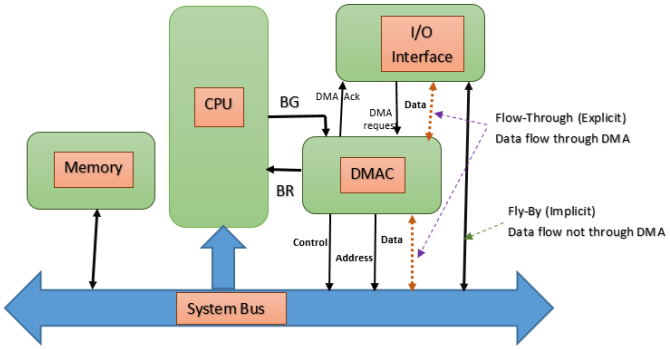

Figure 1 DMA controller principle. Data transfer between input output devices and memory flow-through and fly-by

During data transfer operations, DMAC sends one control signal to the processor, known as the bus request (BR) signal. In response to the BR, the processor completes its current job and sends a signal called the bus grant (BG). Once the BG is received, the DMAC takes hold of the CPU bus and initiates the signal required for the data transfer operation. DMA works in two modes: flow-through and fly-by. In the first of these, the data transfer between the memory and input-output devices is through the DMAC, while in the second mode the transfer is made when the DMAC writes the address and control signals onto the bus and gives access to the device for transfer. This will transfer data between the I/O ports and memory address, but not between more than one I/O port and memory location. Before the DMAC transfer is configured by the processor, the I/O address, read/write memory address, data size, and transfer modes are written in the DMAC register. A similar DMA principle is used in the proposed AMBA architecture-based system for embedded products.

2.2. Proposed Architecture

2.2.1. AMBA system

An AMBA based DMA controller for the SoC is proposed for embedded system products, in order to access the data between the memory and connected devices. Figure 2 shows a generic system based on AMBA, which consists of two types of bus: an Advanced High-Performance Bus (AHB) and an Advanced Peripheral Bus (APB) (Aljumah & Ahmed, 2015). AMBA is generally used in several SoCs as an on-chip system bus (Sinha et al., 2014). It is an open standard for the 32-bit embedded processor. Presently, it stands as a benchmark for the SoC model (Flynn, 1997). It provides various single transfer and bus transfer functions, in which the single data packet and the multiple data packets are exchanged. The ARM documentation is very useful for reference (ARM, 1999). Generally, an AMBA system is built using a processor, memory (on-chip RAM), AHB/APB bridge and certain peripherals, such as UART, timer, interrupt and GPIO (general-purpose input/output) devices.

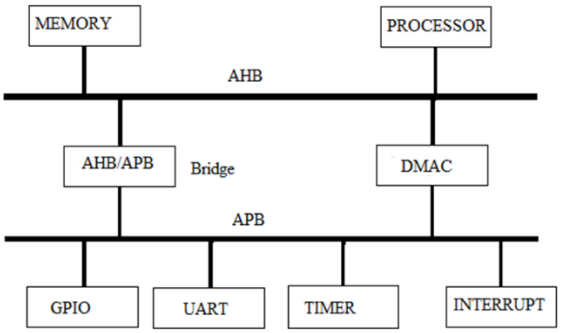

Figure 2 Generic AMBA-based system

a) Advanced High-performance Bus (AHB)

The AHB is used in high-performance systems, and sustains a maximum of 128 bit transfer. It can also support several bus masters. It generally comprises the AHB master, AHB slave, arbiter and decoder; it was introduced in AMBA 2, and subsequently improved in AMBA 3. In the simplest AMBA architecture, multiple masters are connected to multiple slaves. If more than one masters have to be used, arbiter will select one master at a time. One master can be connected to multiple slaves called AHB-Lite. This bus is placed at the processor side.

b) Advanced System Bus (ASB)

The Advanced System Bus (ASB) is a high-performance bus and is synchronous if multiple masters are connected to the arbiter, allowing access to the master using arbiter logic. The ASB works on a pipeline approach, in which the address and data can be transferred concurrently. This bus is also placed at the processor side.

c) Advanced Peripheral Bus (APB)

This is a low-performance bus, which is adapted for connecting SoC peripherals. It is interfaced with a system bus (such as the AHB), using a link between the two. It allows the AHB master to address one of the slaves of APB system. It only makes a connection between master and slave, but it is not error free. This bus is placed at the peripheral side.

2.3. Proposed DMA Block Diagram

The proposed AMBA-based DMA controller architecture for SoCs is shown in Figure 3. It is composed of a processor, embedded memory, AHB/APB bridge, peripherals, first-in first-out (FIFO- a buffer memory) and controller state machine. The DMA controller is divided into AHB and APB and functions in one master and many slaves mode (like the AHB-Lite protocol) and multiple masters and multiple slaves mode.

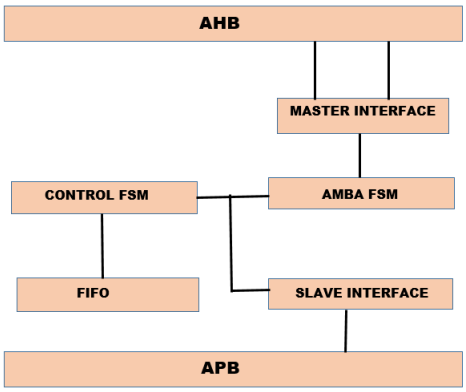

Figure 3 Proposed AMBA-based DMA controller for SoCs

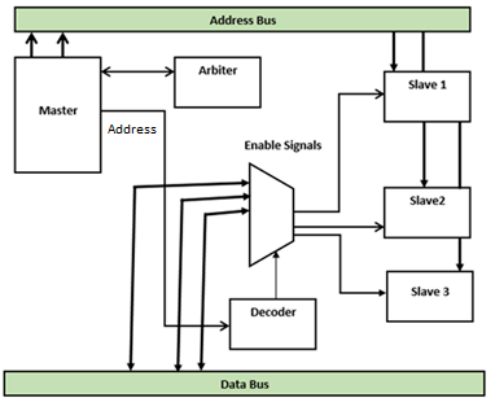

Figure 4 Single master connected to multiple slaves

As it works in both master and slave mode, the master and slave interface work separately. An AMBA finite state machine works to control the two modes, Master and Slave by using these two interfaces. The FIFO block is for the synchronization between the peripherals and the processor speed while exchanging data from the memory.

If we compare this architecture with the ARM Cortex A8 processor, the ARM cortex-A8 processor does not support the master slave mode and there is no arbitrary mechanism for accessing the bus during read-write operations in it. It is good in data transfer, but it is not an appropriate choice for high volume data transfer. The difference in architecture appears in the technical architectural product specification (ARM, 2019).

Hence, it is well suited to AHB-lite protocol, in which one master is connected to multiple slaves, as shown in Figure 4. It does not require an arbiter block, as there is only one master to select. Single master with multiple slave operations are presented for the proposed DMA in Figure 4.

A multiplexer is used to ensure that only one slave can hold or access the data bus at a single time from several slaves. The decoder selects the slave from the various options in order to perform the transfer operation. It also sets the selection input of the multiplexer at the same time. The multiplexer is used for selecting respective slaves for reading from and writing to the data bus. The DMA functions in multiple master and multiple slave mode. The format of this mode for the proposed DMA is illustrated in Figure 5.

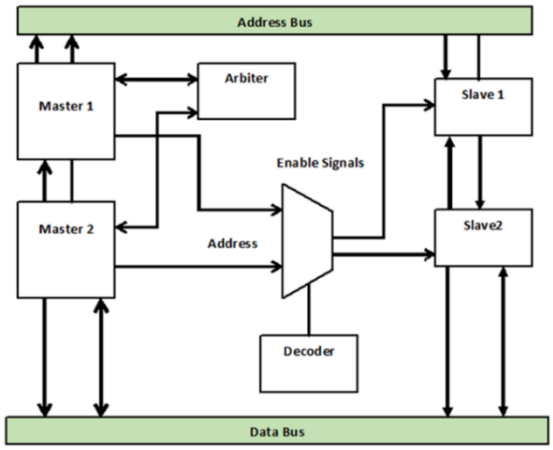

Figure 5 Multiple masters connected to multiple slaves

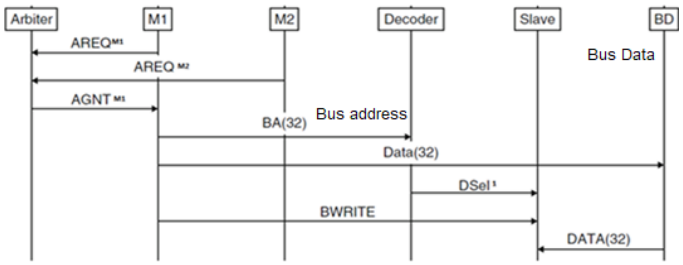

Figure 6 Master slave data transfer operations through address and data bus

Arbiter logic is used in this mode for selecting one or more masters from multiple options. In this logic, multiple masters may request the arbiter, but it only grants the request of one of these. A pipeline protocol is offered by this protocol, in which the transfer of address and data, as well as arbitration, can take place instantaneously. For example, in the Figure 5, there are two masters, Master 1 and Master 2. The arbiter grants the request of any one of the masters to access the bus. The AHB master then performs a burst transfer, in which multiple data element exchanges data from selected slave at a single time. The master-slave data transfer operations through the address and data bus for the proposed design are demonstrated in Figure 6. In this case, the processor initially configures the DMA, then the arbiter grants the request of an AHB master, which then obtains permission to use the bus. After gaining access, the data transfer into the AHB and FIFO is completed by the AHB master. This then requests hold or control of the APB, as arbitration is completed in association with the APB link. It obtains permission to use the bus and completes the data transfer process to the APB and FIFO. The APB and AHB operations are accomplished separately and independent of each other. Because of this, the DMAC is able to accomplish these two operations in parallel, as shown in Figure 3. The AHB side signals are presented in Table 1 and the APB ones in Table 2.

Table 1 AHB side signals

Name	Length (bits)	Description
CLK	1	System clock
RESET	1	Reset is active low
BUSREQ	1	Bus request used by the master to request the bus
GRANT	1	Indicates master has been granted the bus
ADDR	32	Address bus of 32-bits
TRANS	2	Four types of transfer: idle, non-sequential, sequential or busy
WRITE	1	Indicates write or read transfer. High indicates a write transfer, and low a read transfer
SIZE	3	Size of transfer: byte, half word or word
BURST	3	Indicates if the transfer forms part of a burst
PROT	4	Indicates if the transfer is an opcode fetch or data access
WDATA	32	Write operations, used for transferring data from master to slaves
RDATA	32	Read operations, used for transferring data from slaves to master
READY	1	High indicates transfer has finished
RESP	1	Status of transfer: okay, error, retry or split
SEL	1	Slave selected

Table 2 APB side signals

Name	Length (bits)	Description
CLK	1	The system clock
RESET	1	Reset signal is active low
ADDR	32	Address bus of 32-bits
SEL	16	Specifies the slave selected and that data transfer is desired
ENABLE	1	Enables signal
WRITE	1	Indicates read or write operations: high indicates write, and low indicates read
WDATA	32	Bus driven by peripheral bridge during write operations
RDATA	32	Driven by slave during read operations

3. PERFORMANCE AND RESULTS

The high-performance DMA controller was built using AMBA architecture and implemented in Verilog hardware descriptive language (HDL). The efficacy of the simulation and synthesis was performed with the Mentor Graphic's Modelsim tool and Xilinx, respectively. The simulation process was programmed for design strategy. It shows different test specifications for the reading and writing operations; Figure 7 displays the respective waveforms.

The read operations are performed when access to the bus is granted. This design is synthesized, while simulation with Modelsim was found to be error free. Virtex-5 was selected as a target device and the synthesis procedure was conducted on an FPGA device. The synthesis results are tabulated in Table 3 for the area, maximum frequency and time.

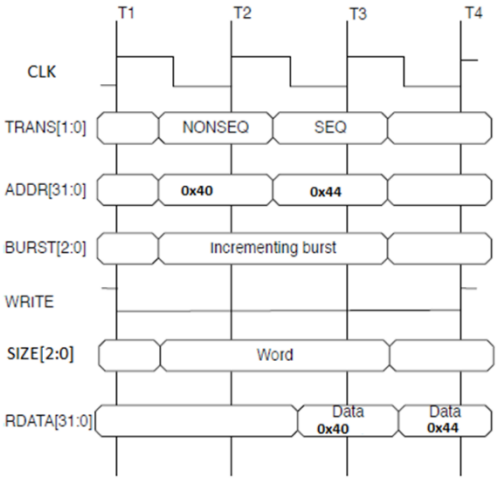

Figure 7 Read mode operations, 2-word transfer

Table 3 Proposed DMAC synthesis results

Design	Max Frequency (MHz)	Time (nsec)	LUTs	Cycles per sec
DMAC	478	2.092	167	478 mega

4. DISCUSSION

The DMA controller design on the Xilinx FPGA device (Faezeh & Mohammad, 2017) works at a maximum frequency of 100MHz with a 100 Mbytes/Sec data transfer rate. The design uses around 4000+ LUTs. There is another DMA design by Xilinx (Xilinx, 2010) and implemented on Virtex 5 FPGA as the target device. This design works at a maximum frequency of 170 MHz, occupying 801 LUTs. Similarly, the basic DMA 8237 which works on ISA bus architecture provides a data rate of 1.6 Mbytes/sec (Zayati et al., 2012; Oded, 2012), whereas the proposed DMA controller achieves a maximum frequency of 478 MHz, with the use of only 167 LUTs on the same Virtex 5 device. Therefore, the proposed DMA design is a better choice in terms of speed, data transfer rate and area utilization. A performance comparison is shown in Table 4.

From the results, a maximum frequency of 478 MHz, or 478,000,000 cycles/sec, is achieved. This means DMAC is able to transfer [4×478 M = 1912 megabytes/sec] 1912 MB/sec data. To better understand the role of DMA after integration into the SoC with the processor, we took two examples of the processors used in existing SoCs. An SoC-embedded processor such as the ARM ZC702 Zynq-700 Xilinx series processor works at a frequency of 200 MHz (Xilinx, 2017). The

data transfer rate of this processor with 32-bit data width results in 800 M bytes per second [4×200M = 800 megabytes/sec]. The second ARM series processor, the Cortex A8, is a 32-bit processor which works at a frequency range of 600 MHz to 1 GHz (ARM, 2019; Cadence, 2018). The average frequency can be taken as 800 MHz, which yields 3200 M bytes/sec [4×800 = 3200 megabytes/sec] data transfer. A comparison is made in Table 5.

Table 4 Performance comparison of proposed DMAC and existing DMA

Sr. No.	Design	Max Frequency (MHz)	LUT Utilization	Transfer Rate (Mbytes/Sec)
1	Proposed DMA	478	167	1912
2	DMA (Faezeh & Mohammad, 2017)	100	4000	100
3	Xilinx DMA (Xilinx, 2010)	170	801	680
4	DMA 8237	-	-	1.6

Table 5 Transfer rate with respect to time

Time Required for Transfer (msec)	DMAC	ZC702	Cortex A8
	Transfer Rate (Mbytes)	Transfer Rate (Mbytes)	Transfer Rate (Mbytes)
0.01	0.01912	0.008	0.032
0.1	0.1912	0.08	0.32
1	1.912	0.8	3.2
10	19.12	8	32
100	191.2	80	320
1000	1912	800	3200

The proposed DMAC works at 478MHz maximum frequency, whereas the ARM series processors ZC702 and Cortex 8 work at 200 MHZ and 800 MHz respectively. DMAC is a separate entity and will only perform data transfer operations. This results in better performance when transferring large volume of data. On the other hand, the ARM series processor will perform multiple operations in parallel. Any single interrupt can divert the processor from its transfer operation at hand and force to attempt the new requested interrupt. In this way, it is effective in transferring large volumes of data and is consequently a better solution. A performance comparison graph is shown in Figure 8.

Additionally, when DMAC is integrated with the processor in SoCs, it can off load the processor and perform fast and smooth operations, and simultaneously the processor can perform other operations side by side. In this way, system performance is boosted in terms of speed. The ARM Cortex processor works at higher frequency with respect to DMAC, hence it can transfer more bytes in the same time period. Nevertheless, DMAC is still able to send a considerable quantity of data. The data transfer rates between DMAC and ARM Cortex A8 (Roberts-Hoffman & Hegde, 2009) are compared in Figure 9.

The embedded processor Cortex A8 works at 800MHz but is not able to transfer a large volume of data at the same transfer rate. As the number of cycles increases, the transfer rate decreases. It is able to transfer data for 400 cycles or less at 800MHz and later it performs other concurrent operations. A comparison of the volume of data transfer by DMAC and Cortex A8 is tabulated in Table 6. The large volume of transfer is possible by using a choice of a DMA integration in

SoC with other entities including processor. A previous study has also focussed on this point concerning the ARM Cortex 8 processor (Roberts-Hoffman & Hegde, 2009) but considered it with an existing DMA. The DMAC inadvertently incorporated a significant volume of data transfer. Hence, using this DMAC we enjoy the benefit of a high volume of data transfer at increased transfer rate, which reduces stress on the processor.

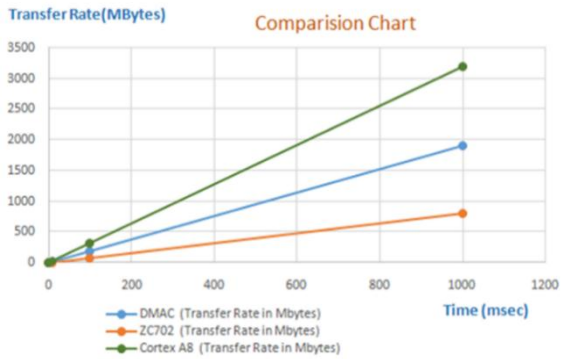

Figure 8 Performance graph of DMAC, ZC702 and cortex A8

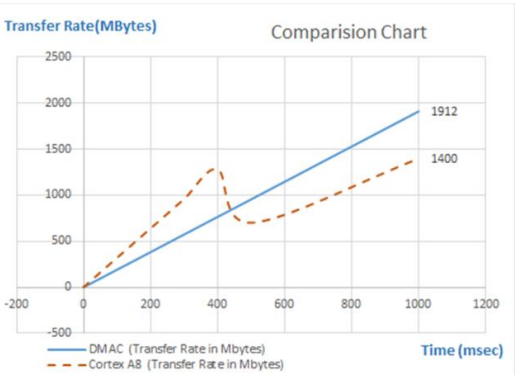

Figure 9 Rate of data transfer between DMAC and ARM cortex A8 for volume of data

Table 6 Performance comparision for volume of data between DMAC and cortex A8

Time Required to Transfer (msec)	DMAC Transfer Rate (Mbytes)	Cortex A8 Transfer Rate (Mbytes)
0.001	0.00191	0.0032
0.01	0.01912	0.032
0.1	0.1912	0.32
1	1.912	3.2
10	19.12	32
100	191.2	320
200	382.4	640
300	573.6	960
400	764.8	1280
500	956	700
1000	1912	1400

5. CONCLUSION

With regard to the performance characteristics of the proposed AMBA-based DMAC, based on the investigative observations we can state that it can be considered as a good alternative for SoC design. The volume of data transfer and timing are critical issues. This architecture is a good attempt at improving the characteristics of data transfer, and the DMAC has resolved both issues, which is highlighted by a comparison of the two cases. As illustrated in these, the suggested DMAC proves itself to be superior in the transfer of data at high speed; for example, in multimedia transfer operations. Future developments of this work could include extension to other peripherals, and the generation of a test bench in advanced verification language to stimulate the various peripheral modules connected.

6. CONFLICT OF INTEREST

The authors declare that there is no conflict of interest regarding the publication of this manuscript.

7. ACKNOWLEDGEMENT

This project was supported by the Deanship of Scientific Research, Prince Sattam bin Abdulaziz University, under research project No. 2017/01/7713.

8. REFERENCES

Ahmed, M.A., Rani, E.D., Syed, A.S., 2015. FPGA Based High Speed Memory Bist Controller for Embedded Applications. *Indian Journal of Science and Technology*, Volume 8(33), pp. 1–8

Aljumah, A., Ahmed, M.A., 2015. Design of High Speed Data Transfer Direct Memory Access Controller for System on Chip Based Embedded Products. *Journal of Applied Sciences*, Volume 15(3), pp. 576–581

Aljumah, A., Ahmed, M.A., 2016. AMBA Based Advanced DMA Controller for SoC. *International Journal of Advanced Computer Science and Applications*. Volume 7(3), pp. 188–193

Anand, S., 2013. Implementing a PCI-Express AMBA interface Controller on Spartan 6 FPGA. *Master's Thesis*, Integrated Electronic System Design, Chalmers University of Technology Sweden

ARM, 1999. AMBA Specification. 2.0. Available Online at http://www-micro.deis.unibo.it/~magagni/amba99.pdf., Accessed on December 17, 2016

ARM, 2019. ARM Developer the Products Category Processors Cortex-A. Available Online at https://developer.arm.com/products/processors/cortex-a/cortex-a8, Accessed on December 17, 2016

ARM., 2017. AMBA Trademark License. Available Online at http://arm.com/about/trademarks/arm-trademark-list/AMBA-trademark.php, Accessed on October 8, 2017

Barry, B., 1997. *The Intel Microprocessors Brey Architecture: Programming and Interfacing, Prentice-Hall International*. Inc. Fourth Edition 1997, pp. 469

Berawi, M.A., 2013. Modeling and Simulation in Engineering Design and Technology: Improving Project/Product Performance. *International Journal of Technology*, Volume 4(2), pp. 100–101

Cadence, 2018. Design Reuse by Cadence, Architecture and Implementation of the ARM Cortex-A8 Microprocessor. Available Online at https://www.design-reuse.com/articles/11580/architecture-and-implementation-of-the-arm-cortex-a8-microprocessor.html, Accessed on March 3, 2019

Ejidokun, T.O., Yesufu, T.K., Ayodele, K.P., Ogunseye, A.A., 2018. Implementation of an On-board Embedded System for Monitoring Drowsiness in Automobile Drivers. *International Journal of Technology*, Volume 9(4), pp. 819–827

Faezeh, S., Mohammad S., 2017. Area and Performance Evaluation of Central DMA Controller in Xilinx Embedded FPGA Designs. *In:* Iranian Conference on Electrical Engineering (ICEE), pp. 546–550

Flynn, D., 1997. ARM, AMBA: Enabling Reusable on Chip Designs. *IEEE Micro*, Volume 17(4), pp. 20–27

Gupta, S., Natarajan, L., 2010. Optimizing Embedded Applications using DMA Cypress Semiconductor Corp. *In:* EE Times Design (http://www.eetimes.com), pp. 1–6

Hou, J., 2013. Study on PCI Bus and ISA Bus Conversion Design. *International Journal of Digital Content Technology and its Applications (JDCTA)*, Volume 7(4), pp. 443–453

Jinbiao, H., 2013. Study on PCI Bus and ISA Bus Conversion Design. *International Journal of Digital Content Technology and its Applications (JDCTA)*, Volume 7(4), pp. 443–453

Li, Bo., Peng, Yu., Liu, Da-T., Peng, Xi-Y., 2009. A High Speed DMA Transaction Method for PCI Express Devices. *Journal of Electronic Science and Technology of China*, Volume 7(4), pp. 380–84

Oded, M., 2012. Optimizing DMA Data Transfers for Embedded Multi-Cores. *Master's Thesis*, Graduate Program, Université Grenoble Alpes, Grenoble, Frances

Roberts-Hoffman, K., Hegde, P., 2009. ARM Cortex-A8 vs. Intel Atom: Architectural and Benchmark Comparisons. University of Texas at Dallas EE6304 Computer Architecture Course Project – Fall. Available Online at http://caxapa.ru/thumbs/229665/armcortexa8vsintelatomarchitecturalandbe.pdf, Accessed on March 3, 2019

Shengwei, M., 2016. Design of a PCIe Interface Card Control Software. *In:* WDF IEEE Xplore: Instrumentation and Measurement, Computer, Communication and Control (IMCCC), Fifth International Conference, pp. 767–770

Sinha, R., Roop, P., Sinha, S.B., 2014. *Correct-by Construction Approaches for SoC Design, the AMBA SoC Platform*. First Edition, Springer book, pp. 11–23

Xilinx., 2010. LogiCORE IP Processor Local Bus (PLB), Product Specification, XilinxDS531, September 21, 2010. Available Online at https://www.xilinx.com/support/documentation/ip_documentation/plb_v46.pdf, Accessed on November 2, 2018

Xilinx., 2017. Xilinx Products Boards and Kits-SoC Evaluation Kit ZC702, Zynq-700. Available Online at http://www.xilinx.com/products/boards-and-kits/ek-z7-zc702-g.html#hardware, Accessed on January 16, 2019

Zayati, A., Biennier, F., Badr, M.M.Y., 2012. Towards Lean Service Bus Architecture for Industrial Integration Infrastructure and Pull Manufacturing Strategies. *Journal of Intelligent Manufacturing*, Volume 23(1), pp 125–139