Journal of Applied Sciences

ISSN 1812-5654

science alert

ANSInet
an open access publisher
http://ansinet.com

Journal of Applied Sciences 15 (3): 576-581, 2015
ISSN 1812-5654

ansinet
Asian Network for Scientific Information

RESEARCH ARTICLE OPEN ACCESS

DOI: 10.3923/jas.2015.576.581

Design of High Speed Data Transfer Direct Memory Access Controller for System on Chip Based Embedded Products

Abdullah Aljumah and Mohammed Altaf Ahmed
Department of Computer Engineering, College of Computer Engineering and Sciences, Salman Bin Abdulaziz University, Saudi Arabia

ARTICLE INFO
Article History:
Received: October 21, 2014
Accepted: January 05, 2015

Corresponding Author:
Mohammed Altaf Ahmed,
Department of Computer Engineering,
College of Computer Engineering and Sciences,
Salman Bin Abdulaziz University,
Saudi Arabia

ABSTRACT
This study proposes a method of high-speed data transfer by Advanced Microprocessor Bus Architecture (AMBA) based Direct Memory Access (DMA) controller using asynchronous first in first out (FIFO). Direct Memory Access Controller (DMAC) is designed to read/write data from/to dual ram. The DMA controller is a synthesizable Intellectual Property (IP) core for easy integration into System on Chip (SoC) implementation and it enables transferring the blocks of data from memory to peripheral and vice versa. This transfer of data appreciably reduces the load on the processor. The design is implemented in Verilog Hardware Description Language (HDL) and it may use in microprocessor based SoC for high-speed data transfer. It increases the data transfer speed as compared to original Microprocessor Unit (MPU) based architecture and provides more data transfer rate compare to conventional DMA method. With 162 Slice Look-Up Tables (LUTs) the design has achieved 306.24 Mhz maximum frequency. The proposed architecture is especially suitable for MPU based SoC for high speed large data transfer.

Key words: Advanced microprocessor bus architecture, direct memory access, controller, asynchronous first in first out, system on chip, verilog HDL

INTRODUCTION

Direct Memory Access (DMA) controller is an important entity of System on Chip (SoC) architecture. It plays a significant role in increasing the speed of data transfer between memory and peripherals via a special bus called Advanced Microprocessor Bus Architecture (AMBA) (Flynn, 1997). AMBA has become popular among all the on-chip industrial standard bus architecture.

The design DMAC has chosen AMBA specification for easy integration into SoC as per Tiwari and Dahigaonkar (2011) and Liang *et al.* (2000). Various architectures are available on DMAC, like flyby DMAC by Ganssle group, multi-channel transmission for multi users and dedicated channel for large data throughout. Flyby is a dedicated channel DMAC, works on non-buffer data transmission principle. The main advantage of it is to improve data transmission rate. But on the other side there is a fast data blocks movements in equipments, those are on the same part that is common in audio and video applications and need to buffer the data. Flyby mode is no longer applicable as per Olugbon *et al.* (2005) because, in non-buffering DMAC, it is not possible for writing operation just after reading in a single cycle. Therefore, flyby is not correct choice to get high-speed data transfer. The speed of flyby has achieved by the method described in the proposed research.

As AMBA is specified by advanced Reduced Instruction Set Computer (RISC) machines (ARM Ltd., 2007). Three buses are defined in AMBA specifications arm-amba specifications that are Advanced High Performance Bus, Advanced System Bus, Advanced Peripheral Bus (AHB, ASB and APB). It has described in AMBA based architecture by Flynn (1997), when DMAC transfers data, it is a master on AHB bus. When it realizes transfer of data between AHB slaves and APB peripheral, DMAC must buffer data and transfer to APB bridge to all visiting APB peripherals.

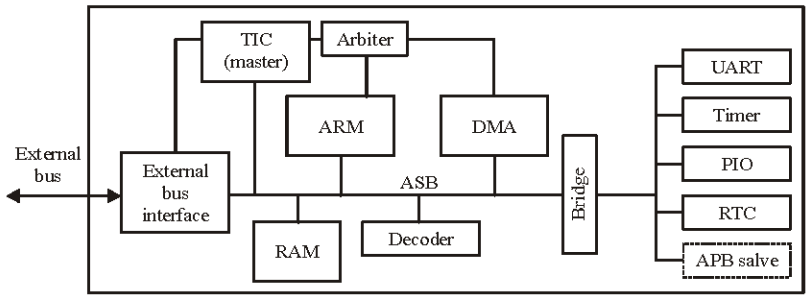

Fig. 1: DMAC AMBA based architecture

The buffering data reduces the data transfer rate which is shown in Fig. 1. For enhancing the data transfer rate, the AHB operation and APB operation should run in parallel, for that the DMAC architecture should lie in between AHB bus and APB bus with APB bridge functions, closely with the approach of Ma (2009). This architecture met the parallelism as per Hwang (1993). It can directly control address, data and control signals on APB bus. In Fig. 1 ASB is used instead of AMBA that is an older name of AMBA.

All the AMBA bus peripherals have to access the data from the memory directly; DMAC can do it. This research study introduces parallel reading/writing from/to dual RAM by buffering mechanism through asynchronous FIFO.

MATERIALS AND METHODS

In this section, the method of design functioning is explained by considering DMAC architecture. The function process of DMAC consists of two part. Transmitter DMA and received DMA engine to make writing and reading operation in parallel. In addition to this dual port RAM which is common for both transmitter and receiver as in Fig. 2.

Data movement from peripherals to memory and memory to peripherals through DMAC in both sides i.e., transmitter and receiver, DMAC has internal buffering mechanism through FIFO at both the side.

In transmitter side Fig. 3 shows, during write operations peripheral sends a request to DMAC. In respond to the request DMAC grants permission to write into FIFO, for every write the write pointer of Tx (transmitted) FIFO incremented by one. Once write operation completed, or while writing into transmitted FIFO, Tx DMA state machine has rights to read data from transmitted FIFO and for every read from Tx FIFO read pointer will increment by one.

Similarly, when the peripheral wants to read data from memory, it sends read request to receive DMAC as shown in Fig. 4. In response to a read request, the data reads from memory by receiving DMA engine and write into Rx (received) FIFO for any external peripheral. The FIFO pointer will increment appropriately for every read and write operations of received FIFO. For both transmitted and received DMA controller, the state machine has designed with three states each side.

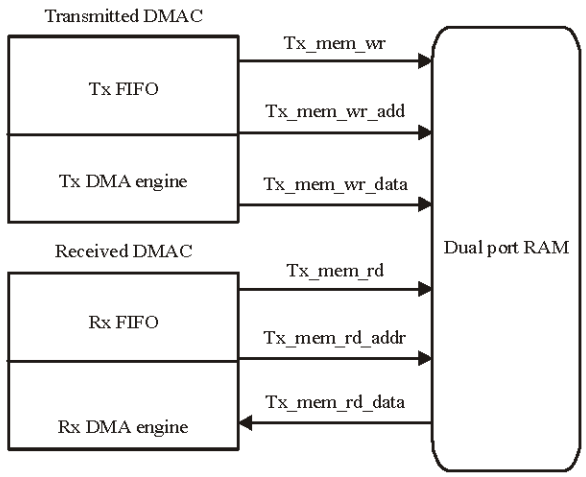

Fig. 2: Proposed architecture DMAC

DMA state machine: DMA state machine consists of two individual state machines for transmitter and receiver. On transmitter side state machine Fig. 5 shows, DMA engine is in idle state. Once Tx_req comes for writing data into memory. It jumps to the state Tx_FF_RD_MEM_WR_ST. In this state, DMA reads data from FIFO and writes back to the memory upon the request of external user. It will write to memory for the count equal to the signal Tx_dcnt. Three states at transmitter side are:

- TX_IDLE: Idle state for transmitter
- TX_FF_RD_MEM_WR_ST: In this state data reads from fifo and write into the memory at transmitter
- TX_ISSUE_STATUS_ST: In this state transmitter DMA issue the status for writing operation

On the other side, the received DMA engine also consists of three states shown in Fig. 6. In RX_IDLE, state DMA waits for the request. Once it receives it jumps to the memory read FIFO write state, i.e., RX_MEM_RD_FF_WR_ST. DMA engine in this state reads data from memory and writes back to the FIFO upon user request. Once it gets the signal TX_ff_ff_full it jumps to status state and displays the status of DMA state machine. Three states at receiver side are:

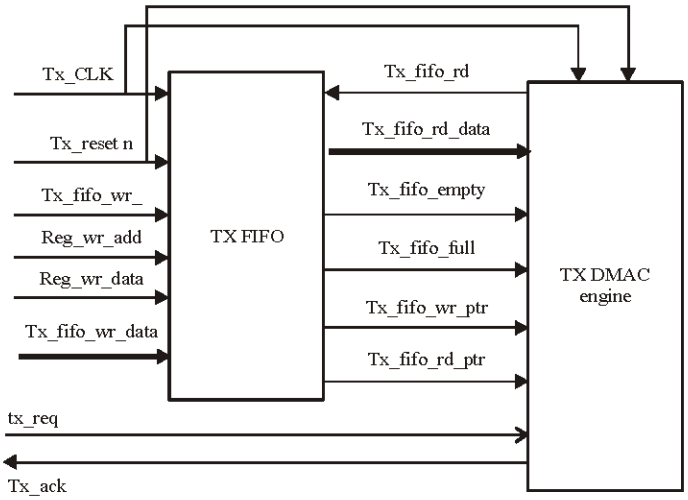

Fig. 3: Transmitted DMAC

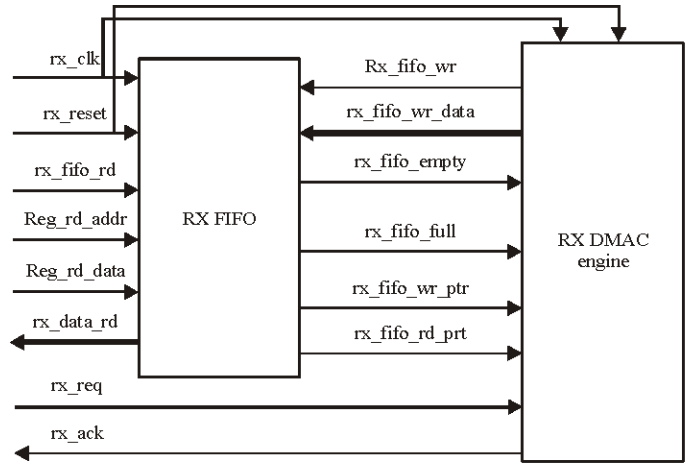

Fig. 4: Received DMAC

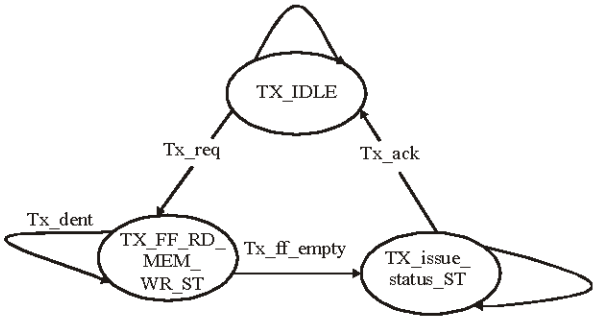

Fig. 5: Transmitted state machine

- RX_IDLE: Idle state for receiver
- RX_MEM_RD_FF_WR_ST: In this state data read from memory and write into fifo at receiver side
- RX_ISSUE_STATUS_ST: In this state receiver DMA issue the status for reading operation at receiver side

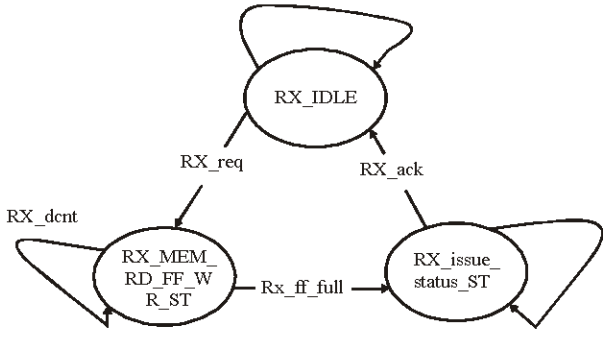

Fig. 6: Received state machine

Signals description of DMAC: All the input and output ports explained in Table 1. This, consists of the signals and their direction, width and function of respective signals.

Table 1: DMAC signal description

Pin	Pin direction	Width (bits)	Description
tx_clk	Input	1	This is the DMA clock input. All the internal state machines and hardware will get clocked with this clock
tx_reset	Input	1	Asynchronous reset; will reset the DMA state machine and internal hardware
tx_req	Input	1	This input sends by the external device to read or write into the memory
Tx_dcnt	Input	6	This is 6 bit input which indicates the No. of reads from fifo and writes to the memory
Tx_fifo_rd_data	Input	32	This is 32 bit input data bus which holds data read from fifo and applied to tx state machine to write into the memory
Tx_fifo_full	Input	1	This input indicates to the tx state machine that fifo is full, come and read data from fifo
Tx_fifo_empty	Input	1	This input indicates to tx state machine that fifo is empty. You cannot read data from fifo
Tx_fifo_wr_ptr	Input	6	This is 6 bit input which shows the No. of writes in the fifo
Tx_fifo_rd_ptr	Input	6	This is 6 bit input which shows the No. of reads from the fifo
Tx_fifo_rd	Output	1	This is the output of tx s.m which sends to tx fifo to read data from it (read enable to fifo)
Tx_ack	Output	1	This is an output send by tx s.m to indicate the data written into the memory successfully
Tx_dma_rd_err	Output	2	This is a two bit output, it asserts when request is zero. If request is '0' then it shows "01" and when it is '1' it gives "10" as output. For all error conditions DMA state machine will halt in idle state
Tx_fifo_rd_err	Output	1	This output indicates the fifo read error. If it is '1' that indicates the addition of dcnt and fifo write pointer is less than the write pointer value. For this condition if it ignores, data will be over write into the fifo and the already written data may be lost
Tx_dma_wr_err	Output	1	It indicates DMA write error. The two port memory we are using for this design, if it gets full and new request to write into the memory comes, then this shows DMA write error
Tx_mem_wr	Output	1	This is a write enable to memory
Tx_mem_wr_addr	Output	6	This is a 6 bit output to indicates the exact location of memory to write data in it
Tx_mem_wr_data	Output	32	32 bit output data bus, which carries data to write into the memory

RESULTS

Simulation results: DMAC design consists of asynchronous FIFO, DMA engine in both transmitted side (when peripherals want to write) and received side (when peripheral want to read) and a common dual port RAM. The whole design implemented in Verilog HDL for easy integration in SoC. Simulation has done for individual blocks and a complete top-level block architecture using Mentor Graphics tool Modelsim ISE by Mentor Graphics. The simulation result of AMBA based DMAC has achieved and indicates reading just after writing in a single clock.

Writing operation into FIFO by peripheral 1 indicated in Fig. 7 and 8 indicates reading operations at the same time by the request of any other peripheral 2 to access the bus. Hence, two peripherals in parallel requests controller and it grants the access of bus to both peripherals alternatively for writing and reading operations to/from dual RAM using buffer mechanism by FIFO. Here writing of peripheral is done first into FIFO and then DMAC reads data and writes into the dual RAM.

As soon as peripherals get access to bus, it sends the address, data and writes enable signals for writing operations and address, reads enable signals for reading operations to control the unit. The data is written into the memory and read from memory. Those addresses are specified by peripheral.

Synthesis results: Synthesis of the design is then carried out on Xilinx synthesis tool Xilinx 13.2 ISE by referring to Xilinx Corporation (1998). Device family has selected for synthesis is Virtex 7 and Target Device is xc7vx330t-3-ffg1157.

Table 2: DMAC design results for improved speed

Design	Area slice LUTs	Time (nsec)	Maximum frequency (Mhz)
DMAC	162	3.265	306.24

Results obtained by synthesis has tabulated in Table 2. It is shown in the Table 2 that design achieved 306.24 Mhz of maximum frequency by utilizing only 162 Slice LUTs on 3.265 nsec minimum time period.

Advanced synthesis report which shows the digital logic block has generated in implementation of DMAC. The synthesis tool also generates the net list. The logic block is shown in Fig. 9, indicates the top level of design that internally consist of FIFO and DMAC state machine.

DISCUSSION

This method introduces an improved DMA method for SoC applications. It increases the data transfer rate in SoC between processor and memory. The AMBA based implementation is used to enhance the data transfer speed using buffering mechanism with asynchronous FIFO. Asynchronous FIFO is use for buffering the data between processor and memory and it provides parallel reading and writing operations to improve speed of data access. As compared to the polling method, data transfer method in microprocessor based SoC and the interrupt method, the proposed methodology present the similar and more efficient method to enhance the speed of data transfer using AMBA interface. It extends the method of Ma (2009) and Hwang (1993). The design has achieved 306.24 MHz maximum frequency with 162 slice LUTs. Results show that the high-speed data rate is achieved with less utilization of chip area. Therefore, it is best suitable for embedded systems based SoC with RISC based processors.

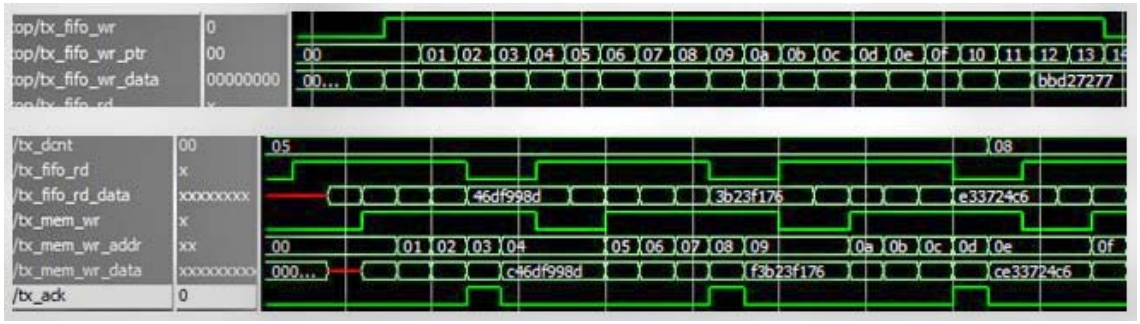

Fig. 7: Data writing to FIFO and reading from FIFO and then writing to memory

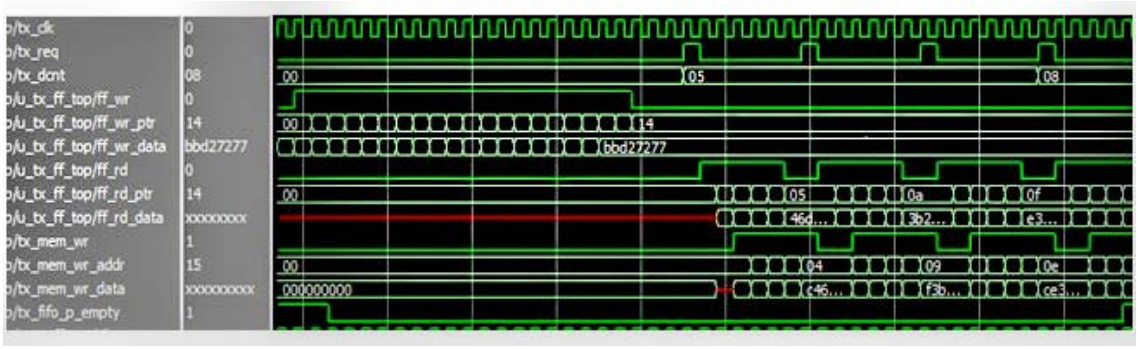

Fig. 8: Reading FIFO writing to memory

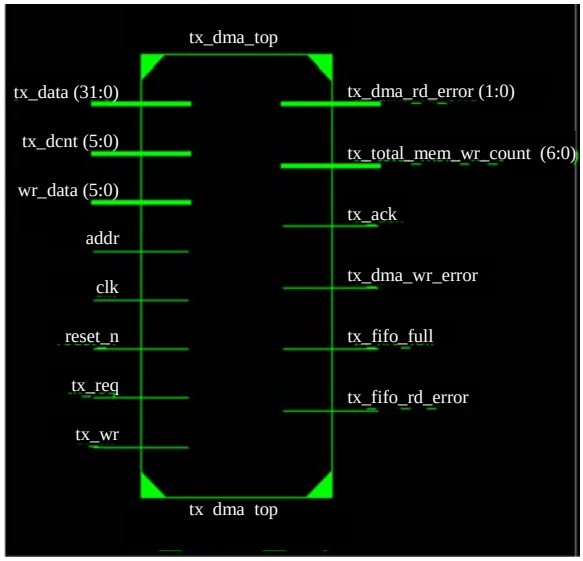

Fig. 9: Logic block generated for DMAC

CONCLUSION

AMBA based DMAC module is implemented in Verilog HDL. Simulation is done on Mentor graphics tool modelsim simulator and synthesis is done on Xilinx tool Xilinx 13.2 ISE synthesizer. The design has achieved AMBA based fast data rate DMAC core for easy integration into SoC product. This design describes how peripherals access

data from dual RAM and how it controls address, data and control signals independently and achieves AHB bus and APB bus to run parallel. The DMAC could adapt buffered data transferred mode according to the speed of the peripheral. Experimental results show DMAC has the advantage of high-speed transfer rate and is much suitable for integration in SoC products. It achieved the maximum frequency of 306.24 Mhz and minimum time period of 3.265 nsec.

ACKNOWLEDGMENT

This study project supported by the Deanship of Scientific Research at Salman Bin Abdul Aziz University, Ministry of Higher Education, Kingdom of Saudi Arabia.

REFERENCES

ARM Ltd., 2007. PrimeCell® DMA controller (PL330) technical reference manual. ARM DDI 0424A, ARM Ltd., Cambridge, UK.

Flynn, D., 1997. AMBA: Enabling reusable on-chip designs. IEEE Micro, 17: 20-27.

Hwang, K., 1993. Advanced Computer Architecture: Parallelism, Scalability, Programmability. McGraw-Hill, New York.

Liang, J., S. Swaminathan and R. Tessier, 2000. ASOC: A scalable, single-chip communications architecture. Proceedings of the International Conference on Parallel Architectures and Compilation Techniques, October 15-19, 2000, Philadelphia, PA., pp: 37-46.

Ma, G., 2009. Design and implementation of an advanced DMA controller on AMBA-based SoC. Proceedings of the 8th International Conference on ASIC, October 20-23, 2009, Changsha, Hunan, pp: 419-422.

Olugbon, A., S. Khawam, T. Arslan, I. Nousias and I. Lindsay, 2005. An AMBA AHB-based reconfigurable SoC architecture using multiplicity of dedicated flyby DMA blocks. Proceedings of the Asia and South Pacific Conference on Design Automation, Volume 2, January 18-21, 2005, New York, pp: 1256-1259.

Tiwari, A. and D.J. Dahigaonkar, 2011. AMBA dedicated DMA controller with multiple masters using VHDL. Int. J. Inform. Technol. Knowledge Manage., 4: 285-288.

Xilinx Corporation, 1998. Xilinx FPGAs: A technical overview for the first-time user. Application Note, XAPP 097 December 12, 1998 (Version 1.3).