IMPLEMENTATION OF DRIVER FOR DMA IN OPEN POWER ARCHITECTURE A20 PROCESSOR

R.Jahnavi Ram [1],

M.Tech[1], CSE Dept[1], JNTUACEA[1], ANANTHAPURAMU[1] A.P[1], India[1]

Abstract— Direct memory access (DMA) is a hardware function within a computer system that is used to relieve the processor or coprocessor from the burden of copying large blocks of data. It allows hardware subsystems to access main system random-access memory independently of the central processing unit (CPU). DMA allows an input/output (I/O) device to send or receive data directly to or from the main memory. A device drivers provides a connection between interface to hardware devices. Device drivers are difficult to build as part of complicated operating system. They're often written in C language because it is a low level programming language. Device drivers developers must have a thorough grasp of the hardware and software systems they are working with. C code will be translated to binary, which will be sent into the processor to conduct operations with the desired module. Tools required for design and development of driver is GCC compiler. Code written in C will be converted into binary which is fed into processor to perform operation with desired module.This project goal is to design and develop a driver for DMA which is connected to A2O core through the AMBA AXI4 bus interface. The GCC compiler will be used to generate the drivers. Drivers will be written in the C programming language

Keywords- Advanced microcontroller bus architecture (AMBA) System-on-chip(SoC), Intellactual Property (IP), AMBA, AXI, VCS, Verilog

I. INTRODUCTION

In recent years due to the miniaturization of semiconductor process technology and computation for survival in the current market conditions constant customization is required. The semiconductor process technology is changing at a faster pace during 1971 semiconductor process technology was 10µm, during 2010 the technology is reduced to 32nm and future is promising for a process technology with 10nm. Intel, Toshiba and Samsung have reported that the process technology would be further reduced to 10nm in the future. So with decreasing process technology and increasing consumer design constraints SoC has evolved, where all the functional units of a system are modelled on a single chip. SoC buses [1] are used to interconnect an Intellectual Property (IP) core to the surrounding interface. These are not real buses, but they reside in Field Programmable Gate Array (FPGA). The AMBA [2] data bus width can be 32, 64, 128 or 256 byte, address bus width will be 32bits wide. The AMBA AXI4 [3] specification to interconnect different modules in a SoC was released in March 2010. A. AMBA AXI4 architecture AMBA AXI4 [3] supports data transfers up to 256 beats and unaligned data transfers using byte strobes. In AMBA AXI4 system 16 masters and 16 slaves are interfaced. Each master and slave has their own 4 bit ID tags. AMBA AXI4 system consists of master, slave and bus (arbiters and decoders). The system consists of five channels namely write address channel, write data channel, read data channel, read address channel, and write response channel. The AXI4 protocol supports the following mechanisms: • Unaligned data transfers and up-dated write response requirements. • Variable-length bursts, from 1 to 16 data transfers per burst. • A burst with a transfer size of 8, 16, 32, 64, 128, 256, 512 or 1024 bits wide is supported. • Updated AWCACHE and ARCACHE signalling details. Each transaction is burst-based which has address and control information on the address channel that describes the nature of the data to be transferred. The data is transferred between master and slave using a write data channel to the slave or a read data channel to the master. Table 1[3] gives the information of signals used in the complete design of the protocol. The write operation process starts when the master sends an address and control information on the write address channel as shown in fig. 1. The master then sends each item of write data over the write data channel. The master keeps the VALID signal low until the write data is available. The master sends the last data item, the WLAST signal goes HIGH. When the slave has accepted all the data items, it drives a write response signal BRESP[1:0] back to the master to indicate that the write transaction is complete. This signal indicates the status of the write transaction. The allowable responses are OKAY, EXOKAY, SLVERR, and DECERR. After the read address appears on the address bus, the data transfer occurs on the read data channel as shown in fig. 2. The slave keeps the VALID signal LOW until the read data is available. For the final data transfer of the burst, the slave asserts the RLAST signal to show that the last data item is being transferred. The RRESP[1:0] signal indicates the status of the read transfer. The allowable responses are OKAY, EXOKAY, SLVERR, and DECERR

The protocol supports 16 outstanding transactions, so each read and write transactions have ARID[3:0] and AWID [3:0] tags respectively. Once the read and write operation gets completed the module produces a RID[3:0] and BID[3:0] tags. If both the ID tags match, it indicates that the module has responded to right operation of ID tags. ID tags are needed for any operation because for each transaction concatenated input values are passed to module.

II. AMBA AXI4 PROTOCOL

AMBA AXI3 protocol has separate address/control and data phases, but AXI4 has updated write response requirements and updated AWCACHE and ARCACHE signaling details. AMBA AXI4 protocol supports for burst lengths up to 256 beats and Quality of Service (QoS) signaling. AXI4 has additional information on Ordering requirements and details of optional user signaling. AXI3 has the ability to issue multiple outstanding addresses and out-oforder transaction completion, but AXI4 has the ability of removal of locked transactions and write interleaving. One major up-dation seen in AXI4 is that, it includes information on the use of default signaling and discusses the interoperability of components which can't be seen in AXI3

A. AMBA AXI4 master

To perform write address and data operation the transaction is initiated with concatenated input of [awaddr, awid, awcache, awlock, awprot, awburst]. On the same lines for read address and data operations the concatenated input is [araddr, arid, arcache, arlock, arprot, arburst]. The addresses of read and write operations are validated by VALID signals and sent to interface unit.

B. AMBA AXI4 Interconnect

The interconnect block consists of arbiter and decoder. When two masters initiate a transaction simultaneously, the arbiter gives priority to access the bus. The decoder decodes the address sent by master and the control goes to one slave out of 16. The AMBA AXI interface decoder is centralized digital block. The decoder decodes the address sent by master and goes to one slave out of 16. 0-150 locations are meant for slave-1, next

151-300 addressable locations are meant for slave- and so on till slave-16.

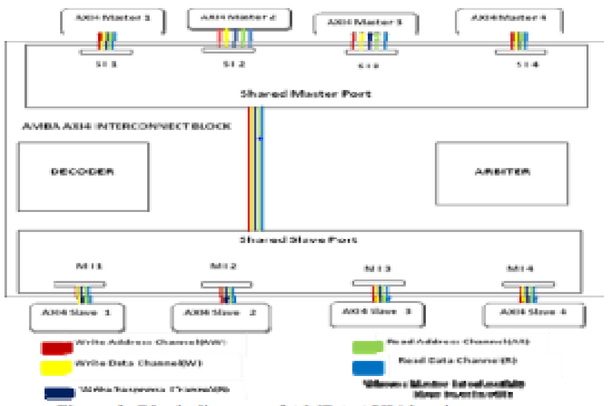

Figure 3: Block diagram of AMBA AXI4 bus interconnect.

C. AMBA AXI4 slave read/write block diagram

The AXI4 slave consists of common read/ write buffer which stores the read/ write address and data as shown in fig. 4. Pending read address register stores the remaining read addresses to be sent; pending write address register which stores the remaining write addresses to be sent and pending write data register which stores the remaining write data to be sent. The read/write state machines receive internal inputs from the read/ write buffer. The AXI4 slave test bench initiates the read or write transaction and the output from the AXI4 slave are standard read/write channel signals. The AXI4 slave receives the write data in the same order as address. Signals used to design slave module is shown in fig. 5. The test layer shown in the fig. 5 has 2 test cases.

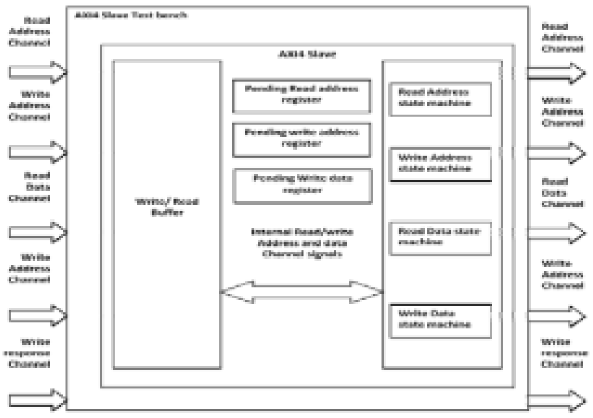

Figure 4: AMBA AXI4 slave Read/Write block Diagram.

III. RELATED WORK

In a SoC, it houses many components and electronic modules, to interconnect these a bus is necessary. There are many buses introduced in the due course some of them being AMBA [2] developed by ARM, CORE CONNECT [4] developed by IBM, WISHBONE [5] developed by Silicore Corporation, etc. Different buses have their own properties the designer selects the bus best suited for his application. The AMBA bus was introduced by ARM Ltd in 1996 which is a registered trademark of ARM Ltd. Later advanced system bus (ASB) and advanced peripheral bus (APB) were released in 1995, AHB in 1999, and AXI in 2003[6]. AMBA bus finds application in wide area. AMBA AXI bus is used to reduce the precharge time using dynamic SDRAM access scheduler (DSAS) [7]. Here the memory controller is capable of predicting future operations thus throughput is improved. Efficient Bus Interface (EBI) [8] is designed for mobile systems to reduce the required memory to be transferred to the IP, through AMBA3 AXI. The advantages of introducing Network-on-chip (NoC) within SoC such as quality of signal, dynamic routing, and communication links was discussed in [9]. To verify on-chip communication properties rule based synthesizable AMBA AXI protocol checker [10] is used.

IV PROPOSED WORK

AMBA AXI4 is the fourth generation of the AMBA interface specification from ARM. AXI is a Parallel high performance, synchronous, high frequency, multi-master multi-slave communication interface. AXI is a burst-based protocol that means there may be multiple data transfers (or beats) for a single request. It supports for burst lengths up to 256 beats. AMBA AXI4 is the bus that performs best in terms of Throughput latency and utilization for single or multiple channels AXI defines the protocol for the interface but not the interconnect. The interconnect does all kinds of protocol conversions and other features like bit mapping, width mapping and so forth. It is extendable for which open-ended to support future needs

Software connections between software programmes and hardware devices are known as device drivers. Device drivers are regarded exceedingly tough to build as part of a complicated operating system. They're often written in low-level programming languages like C. Device driver engineers must have a thorough grasp of the hardware and software platforms they are working with. C code will be translated to binary, which will be sent into the processor to conduct operations with the desired module. The goal of this project is to design and construct an AXI4 module driver that links all modules to an open power A2O core. The GCC compiler is necessary for driver design and development, and the C programming language will be used to construct drivers.

V RESULTS

Simulation is carried out in VCS tool and Verilog is used as programming language.

A. Simulation result for write operation

The AResetn signal is active low. Master drives the address, and the slave accepts it one cycle later. The write address values passed to module are 40, 12, 35, 42 and 102 as shown in fig. 8 and the simulated result for single write data operation is shown in fig. 9. Input AWID[3:0] value is 11 for 40 address location, which is same as the BID[3:0] signal for 40 address location which is identification tag of the write response. The BID[3:0] value is matching with the AWID[3:0] value of the write transaction which indicates the slave is responding correctly. BRESP[1:0] signal that is write response signal from slave is 0 which indicates OKAY

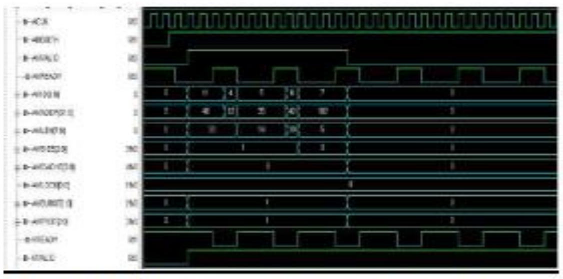
Figure 8: Simulation result of slave for write address operation.

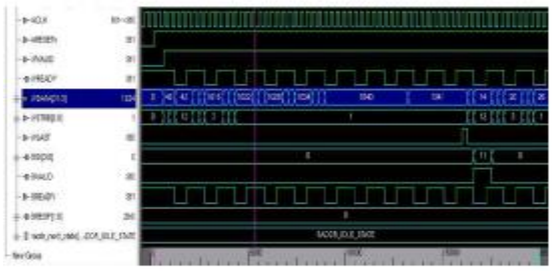
Figure 9: Simulation result of slave for single write data operation.

B. Simulation result for read operation

The read address values passed to module are 45, 12, 67, 98 as shown in fig. 11 and the simulated result for single read data operation is shown in fig. 12. Design of AMBA AXI4 protocol for System-on-Chip communication 42 Simulation result of slave for read address operation. Input ARID[3:0] value is 3 for 12 address location, which is same as the RID[3:0] signal for 12 address location which is identification tag of the write response. The RID[3:0] and ARID[3:0] values are matching, which indicates slave has responded properly. RLAST signal from slave indicates the last transfer in a read burst. Simulation result of slave for multiple read data operation is shown in fig. 13.

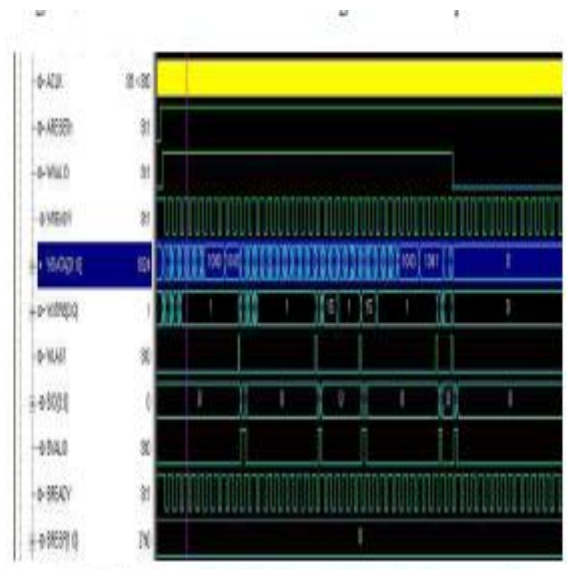
Figure 10: Simulation result of slave for multiple write data operation.

VI. CONCLUSION AND FUTURE SCOPE

Direct memory access (DMA) is a hardware function within a computer system that is used to relieve the processor or coprocessor from the burden of copying large blocks of data. It allows hardware subsystems to access main system random-access memory independently of the central processing unit (CPU). DMA allows an input/output (I/O) device to send or receive data directly to or from the main memoryDevice drivers are regarded exceedingly tough to build as part of a complicated operating system. They're often written in low-level programming languages like C. Device driver engineers must have a thorough grasp of the hardware and software platforms they are working with. C code will be translated to binary, which will be sent into the processor to conduct operations with the desired module. The developer can use any further language for the feasible accessible of the data in any format.

References :

1.https://japan.xilinx.com/support/documentation/ip_documentation/axi_iic/v2_0/pg090-axi-iic.pdf

2.https://www.researchgate.net/publication/257532859_Design_and_Analysis_of_Master_module_for_AMBA_AXI-4

3.https://www.interscience.in/ijcns/vol1/iss3/8/#:~:text=AMBA%20AXI4%20protocol%20system%20supports,compiler%20simulator%20(VCS