Functional Verification of DMA Controller of an Image Processing SoC

Shouvik Saha[1], Shalini Mukhopadhyay[2], Saurabh Bhokre[3] and Basudeba Behera[4]

Department of Electronics & Communication Engineering

NIT Jamshedpur, Jamshedpur, 831014, India

E-mail:- *basudeb.ece@nitjsr.ac.in[4]*

DOI:- https://doi.org/10.47531/MANTECH/ECC.2021.52

Abstract

Technology in the field of VLSI is growing faster. After the advent of different standard hardware protocols& bus signal architectures, they are used for the interconnection of various modules on a System-on-a-Chip (SoC). In order to reduce the design time of the complete system, the SoC design became the most vital and integrated methodology. It has become challenging to verify those on-chip bus protocols as the traditional method is failing in the case of large and complex SoCs. In this paper, we are going to analyze the entire functionality of the DMA controller module of an image and video processing SoC. The DMA controller is based on AXI protocol, and so it is called AXI-DMA controller. A verification environment is built using Verilog and tested in Simics & have succeeded in verifying the AMBA AXI protocol, analyzing the read successfully and write operation for an incremental burst with saleable test bench architecture.

Keywords: - *AHB, APB, AXI, SoC (system-on-a-chip), VCS, Simics.*

INTRODUCTION

Direct Memory Access (DMA) is a feature in the modern computer subsystem that allows different types of subsystems of particular hardware or SoC (system on chip) to access memory and peripherals without using the main processing unit or the Central Processing Unit. Without the help of DMA (direct memory access), the main processing unit uses the programmed I/O to transfer data which is a very much time-consuming process as it takes an entire time in the read and writes operation.

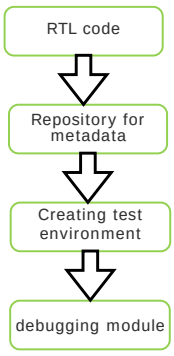

Figure 1: Workflow of verification

Thereby the CPU/main processing unit is devoid of doing other essential work. But with the help of DMA, the CPU/main processing unit does other arithmetical or logical activities through it. The workflow of the verification process is shown below in Figure 1. The SoC buses are used to interconnect all IP (intellectual property) core to the surrounding interfaces. The buses are not real in nature, but they are in the FPGA (Field Programmable Gate Array). The address bus is 32 bits wide, and the data bus width is 32, 64, 128 and 256 bytes.

AXI PROTOCOL

It supports the requirement of interfacing with a large variety of components. It is suitable for the design of systems with low response time and high bandwidth. They can provide a high frequency of operation without the use of the bridge.

It is mostly suitable for memory controllers whose initial access latency is very high. Compatible backwardly with the AHB and APB interfaces existing. The main procedure of the protocol is,

- Master and slave should handshake with each other.
- Transmission of control signal occurs in a separate channel.

- Due to burst type of communication, the continuous transfer can be accomplished.

A. AXI Architecture

The AXI protocol [1] [2] version 4 is entirely a burst based protocol, and it has five independent transaction channels

- Read Data channel.
- Read Address channel.
- Write Data channel.
- Write Address channel.
- Write Response channel.

Every transaction in the protocol stores the control and information of the address channel that contains the nature of the address that further stands for the nature of the data to be transferred. The master and slave communicate data by the use of a signal and thereby writing the data to the slave. AXI has additional responses named write response, where the data generally flows from master to slave during the write transaction. The extra write response channel, which AXI has, allows the slaves to indicate the compilation of the write signal. General formats of the system in which master and slaves are connected together with the help of some form of interconnect.

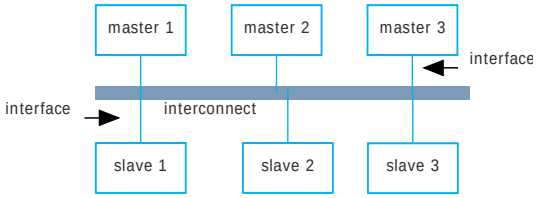

Figure 2: Master and slave interconnect

Figure 2 illustrates the typical interconnection of master and slaves through a common interconnect generally known as a bus. Figure 3 shows the typical block diagram of an AXI protocol [2] with its five independent transaction channels.

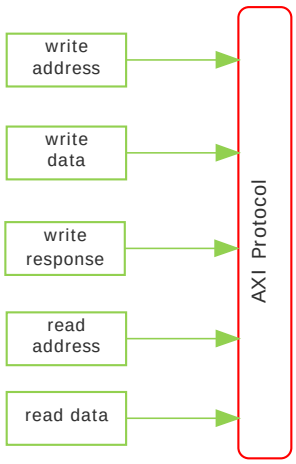

Figure 3: AXI protocol channels

- ***Write address:*** This channel has the required control and address information that describes the nature of data to be transferred.
- ***Write data:*** Transaction between master and slaves is made through this channel.
- ***Write response:*** This channel is used by the slave as an indication to the master that the transfer is completed.
- ***Read address:*** The channel has the required address and control information for a read path that describes the nature of transferred data.
- ***Read data:*** The channel is used to make transaction from slave to master.

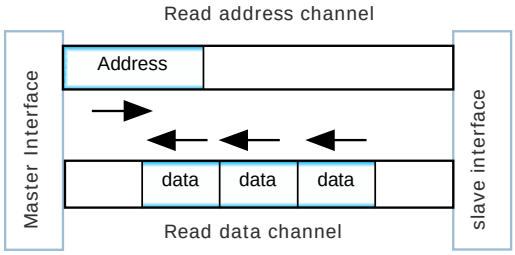

Figure 4: AXI read channel architecture

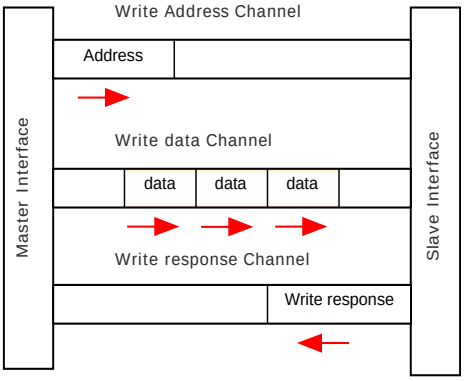

Figure 5: AXI write channel architecture

The above figures, i.e., Figure 4 and Figure 5, shows the read and write channel architecture of the AXI protocol.

B. Handshaking Signals

After the read and write transaction occurs separately through each channel, the handshaking takes place between master and slaves. Each channel has a pair of signals name VALID and READY [2] by which the handshaking takes place. Suppose the master asserts AWVALID on the positive edge of the clock, which indicates that the master is driving valid information on the write address channel. Now AWREADY, which is driven by a slave, is 1 indicates that the slave is ready to accept the control and address information provided by the master. A logical high

(=1) of both the above signals at the positive edge of the clock indicates the transfer is complete.

Table 1: Direction of handshaking signals

AXI channels	Direction of handshaking signals
Write address channel	AWVALID (m->s) AWREADY(s->m)
Write data channel	RVALID (m->s) RREADY (s->m)
Write response channel	BVALID (s->m) BREADY (m->s)
Read address channel	ARVALID (m->s) ARREADY (s->m)
Read data channel	RVALID (s->m) RREADY (m->s)

*master in short is m

*slave in short is s

C. AXI Protocol Write & Read transaction

a. Write Transaction

The transaction in AXI protocol takes place in five independent channels; the write transaction does so in 3 channels (Write address, write data, write response) to complete write transfer [1] [2]. Phase data is written based on the control signal that will decide the length of data (i.e. AWSIZE and AWLEN). The type of the burst is determined by AWBURST, and the last transfer is indicated by the WLAST signal. Now BRESP signal is set to 1 when the writing of data is complete. The transaction steps are given below,

- The address and control information is given by the master in the write address request.
- With the writing of data, the master gives the write data request.
- The write response status shows the response of slaves.

b. Read Transaction

The read transaction [1] [2] uses two channels in the AXI protocol, namely (Read address and Read data), to complete the read transfer. Like, write transaction here too, the read transaction is completed by going off the ARVALID and ARREADY as high. AWADDR and ARADDR must be the same, and the last transfer is shown by RLAST. The transaction steps are given below,

- The address and control information is given by the master with a read address request.
- The read data and read response are given by the slaves in a read data phase.

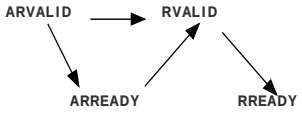

Figure 6: Read transaction dependencies

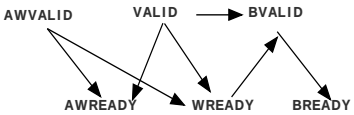

Figure 7: Write transaction dependencies

c. Relationship between channels

It is required to maintain the following while discussing AXI protocol,

- Only after the last write transaction has occurred, the write response occurs.
- The read data is always followed by address the data relates.
- There's supposed to be dependencies between the channel handshake signals.

CHANNEL FIFO

The transfer of data from the source peripheral of the channel to its destination peripheral is controlled by the transfer control logic. Data that comes from the source peripheral is temporarily stored into the FIFO [6] [11] before being transferred into the destination peripheral. Here the DMA (direct memory access) implements the logic it needs to pack and unpack the data accordingly to fit the FIFO size provided the source and destination peripheral uses different sizes (**arsize & awsize**) for data transfer. The FIFO also works in two different modes, namely asynchronous and synchronous FIFO. See **figure 8**.

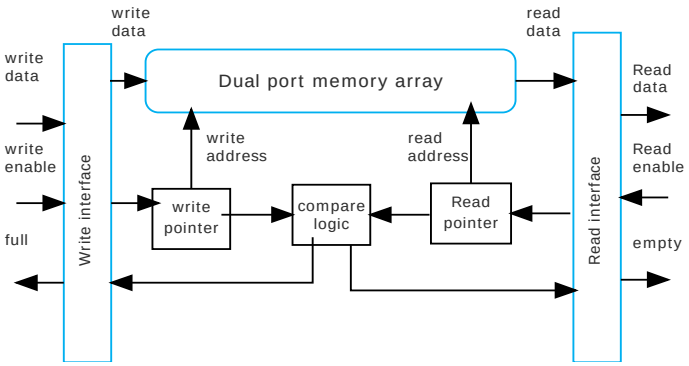

Figure 8: Structure of a FIFO

Synchronous FIFO: This is a First-In-First-Out memory queue with control logic that manages the read and write pointers, generates status flags, and provides optional handshake signals for interfacing with the user logic in the same clock domain.

Asynchronous FIFO: This is a FIFO logic where the data values are written to the FIFO buffer from one clock domain, and the data is read from the same FIFO buffer in a different clock domain, where the two clock domains are asynchronous to each other.

TOOLS USED

Simulation is performed by using the tools as follows,

a. Simics.
b. VCS (Verilog cluster simulation).
c. DVE (Discovery Visual Environment).

Simics [8] is a simulating software that can run unchanged production binaries of the target hardware at a very high-performance speed. Simics makes it easier to perform the experiment and test any new hardware setup, application software and different configuration of the platform before committing it to the original design of the system. It tests the way by which any application software behaves and scales by varying core count, processor speed and memory size, and the number of boards present in a network under variation of any circumstances.

A. Simulating with Simics

There are mainly two fundamental terms used in Simics, and those are “host” and “target”.

- “host” refers to the system on which Simics is running. It can also refer to the architecture or to the model of the computer in which the simulator is running, or it can refer to the combination of both architecture and operating system.
- “target” refers to the model or system which is simulated by Simics. It can also refer to the architecture or any model of the computer system.

B. DVE (Discovery visual environment)

DVE is an interactive graphical user interface that is used for debugging and simulating system Verilog, VHDL, Verilog and system C designs. A user can drag and drop various type of signals of the DUT (device under test) to view the signal source, trace drivers, compare waveforms and also view schematics.

C. Files required for DVE tools

VPD file

Table 2: Signals description of AXI channels

signal	source: master/global	Input/output	Description
aclk	Gobal	input	global clock sig
aresetn	Global	input	global reset sig
awid[3:0]	master	input	write address id
awaddr[31:0]	master	input	write address
awlen[3:0]	master	input	write burst length
awsize[2:0]	master	input	write burst size
awburst[1:0]	master	input	write burst type
awlock[1:0]	master	input	write lock type
awcache[3:0]	master	input	write cache type
awprot[2:0]	master	input	write protection type
wdata[31:0]	master	input	write data
arid[3:0]	master	input	read address id
araddr[31:0]	master	input	read address
arlen[3:0]	master	input	read burst length
arsize[2:0]	master	input	read burst size
arlock[1:0]	master	input	read lock type
arcache[3:0]	master	input	read cache type
arprot[2:0]	master	input	read protection type
rdata[31:0]	master	input	read data
wlast	master	input	write last
rlast	slave	output	read last
awvalid	master	output	write address valid
awready	slave	output	write address ready
wvalid	master	output	write valid
rvalid	slave	output	read valid
wready	slave	output	write ready
bid[3:0]	slave	output	write response id
rid[3:0]	slave	output	read response id
bresp[1:0]	slave	output	write response
rresp[1:0]	slave	output	read response
bvalid	slave	output	write response valid
bready	master	output	response ready
rvalid	slave	output	read valid

The **.vpd** files (design database files) are platform-independent versioned files that can dump the selected signals during testing or simulation. For the proper functionality of simulation, it is necessary that the VCS, which is generating the.vpd and the DVE to view the **.vpd**, both of these versions must be identical. The test set up for the project is as shown below

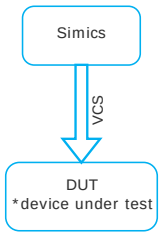

Figure 9: Test setup of verification

VERIFICATION PLAN

The test plan is such a great amount to be identified with equipment specs. It illustrates a portrayal of the situations that are to be examined

and also the coding styles of the test script that is implemented. It contains affirmation, formal evidence, copying, hardware/software co-confirmation, irregular or coordinated testing, and implementation of check IP. In general, the test case coding takes a longer time to run and analyze, but with a Simics environment, it has become a bit easier.

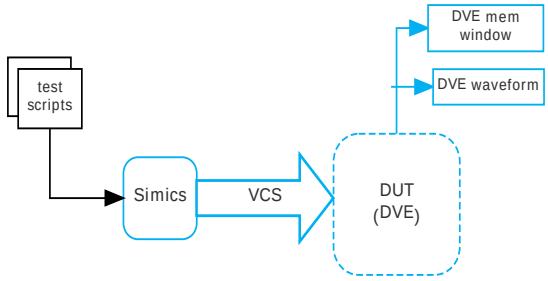

Figure 10: Verification plan

The advantages of using Simics in the test process are,

a) Save test case time development.

b) A proven model to fix the bug occurs in DUT during the verification.

c) Reconfigurable and adaptable to any kind of AXI interface.

d) Bug fixing is done effectively and can be reused at any time.

SIMULATION & RESULTS

Initializing all the registers of DMA, populating the source address register, and destination address register with their respective source address as well as destination address locations.

Table 3: List of important register

Register	Brief review
Source address register	Starting address location from where data is read
Destination address register	Starting address of the location where data is transferred
Block transfer register	The size of the block that is transferred
Configuration register	It control the transfer configuration
Control register	It is programmed to control the transfer of data through DMA
Status register	It indicates the status of the DMA transfer

Table 4: Initialization data

Source address	0x1000000000
Destination address	0x1800000000
Data_1	0xaa55aa55
Data_2	0xdeadbeef

Simulation Results

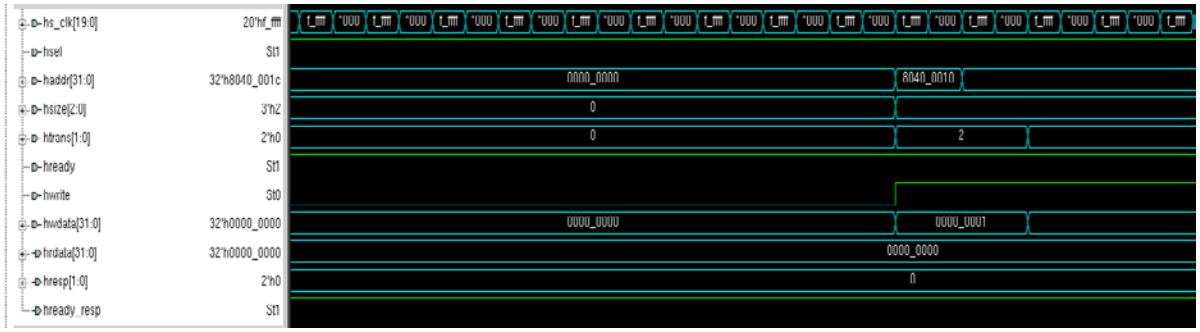

Figure 11: AXI/AHB Initialization

A. AXI-DMA Read Operation

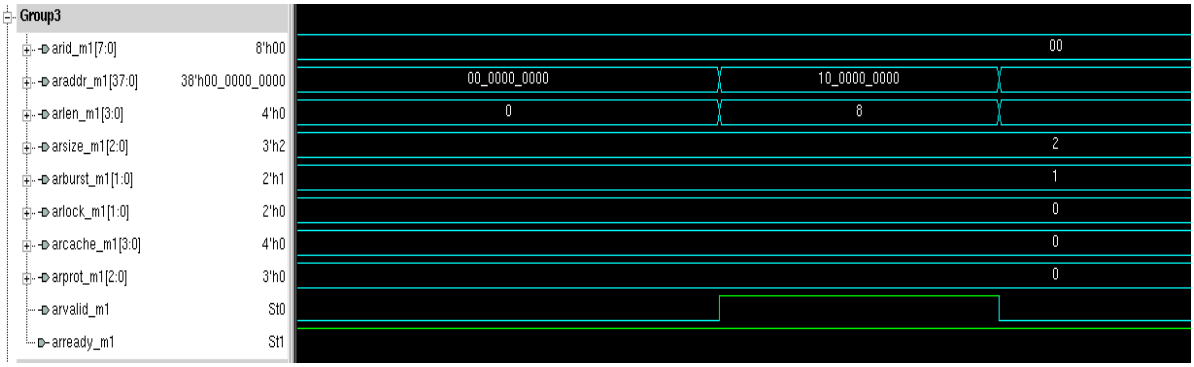

Figure 12: Address Read Channel

Figure 13: Data read channel

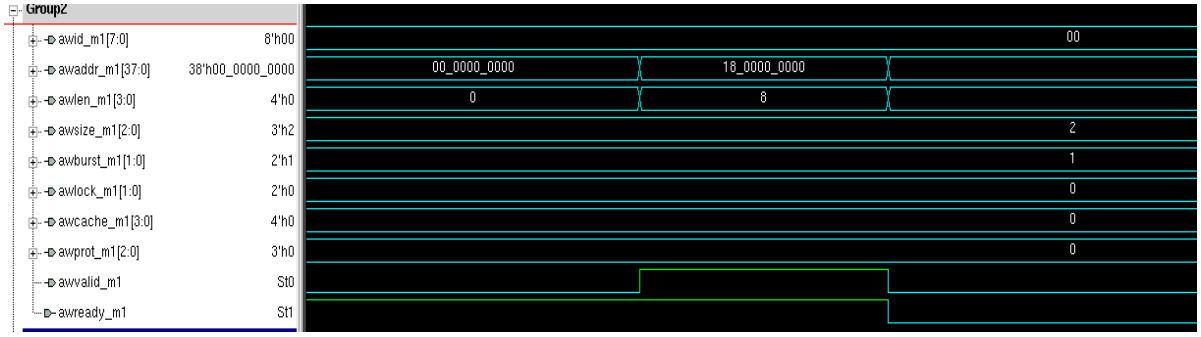

Figure 14: Address Write Channel

Figure: 12 and 13, as shown above, depicts Read operation of the AXI Channel and thereby showing the status of address and data read channel respectively.

B. AXI-DMA Write operation

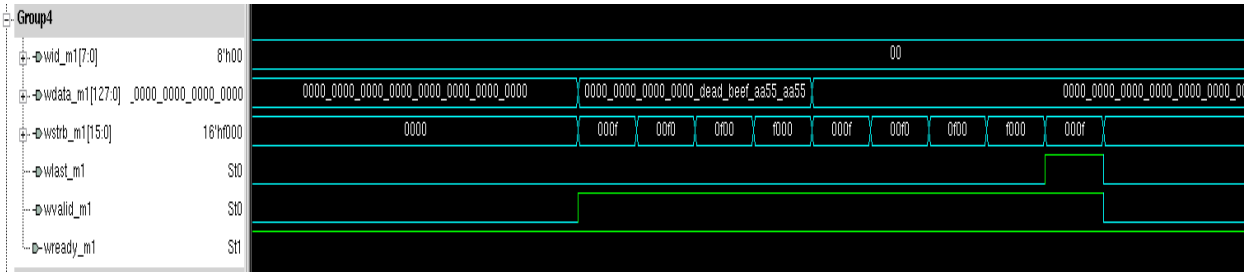

Figure 15: Data Write Channel

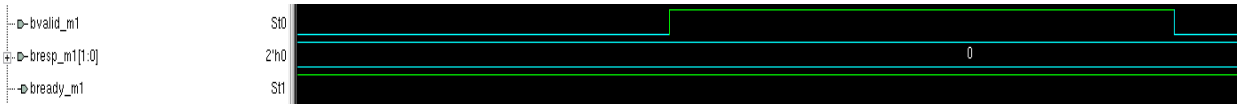

Figure 16: Write Response Channel

The above Figure: 14 and 15 show the respective Write operation of AXI-DMA. At first, it acknowledges the destination address, and then it writes (transfer) the data from the source address to that particular destination address location. The Write response channel is validated by the indication of the **bvalid** and **bready** signals. When both of these signals is high, it shows that DMA has successfully written the data into the channel. The result is shown in the above Figure: 16.

CONCLUSION

The functional characteristics of the AXI protocol are analyzed & verified using Verilog and Simics. The transfer of the data of AXI-DMA is also verified. A standard verification environment is built for AXI-DMA using Simics and Verilog, as shown in Figure: 10. The main idea is to simply perform the data transfer in between DMA and other memory modules such as DDR, CSRAM etc. and observe the read and write transaction for the incrementing type of burst feature of AXI. There are five independent channels. Each of the channels with its signals is simulated and analyzed from the waveform analysis. All the simulations are done using VCS-DVE tools by the synopsis. This advanced high-speed DMA controller is proposed to be the better alternative to high-speed data transfer in different application fields such as multimedia processing.

REFERENCES

1. N. Gaikwad and M. P. Deepu and R. Dhanabal, "Validation of transactions in AXI protocol using system verilog," 2017 *International conference on Microelectronic Devices, Circuits and Systems (ICMDCS)*, Vellore, 2017.

2. V. N. Patil, "Verification of AMBA AXI On-Chip Communication Protocol," 2018 Fourth *International Conference on Computing Communication Control and Automation (ICCUBEA)*, Pune, India, 2018.
3. V. Lakhmani, N. Ali and V. S. Tripathi, "AXI Compliant DDR3 Controller," 2010 *Second International Conference on Computer Modeling and Simulation*, Sanya, Hainan, 2010.
4. G. Mahesh and S. SM, "Verification of memory transactions in AXI protocol using system verilog approach," 2015 *International Conference on Communications and Signal Processing (ICCSP)*, Melmaruvathur, 2015.
5. AMBA AXI Protocol Specification, ARM Limited, 2003.
6. Guoliang Ma and Hu He, "Design and implementation of an advanced DMA controller on AMBA-based SoC," 2009 IEEE 8th *International Conference on ASIC*, Changsha, Hunan, 2009.
7. C. Chen, J. Ju and I. Huang, "A synthesizable AXI protocol checker for SoC integration," 2010 *International SoC Design Conference*, Seoul, 2010.
8. Wind River Simics, "Model Builder User's Guide," Wind River, 2010.
9. N. Tidala, "High Performance Network on Chip using AXI4 protocol interface on an FPGA," 2018 Second *International Conference on Electronics, Communication and Aerospace Technology (ICECA)*, Coimbatore, 2018.
10. Hang Yuan, Hongyi Chen and Guoqiang Bai, "An improved DMA controller for high speed data transfer in MPU based SOC," Proceedings. 7th *International Conference on Solid-State a*nd Integrated Circuits Technology, Beijing, China, 2004.
11. C. Sarojini and J. Thangaraj, "Implementation and Optimization of Throughput in High Speed Memory Interface Using AXI Protocol," 2018 9th *International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, Bangalore, 2018.