Journal of Engineering Technology Vol. 2(1): 54-59, 2012
ISSN 2231-8798

Characterization of Bus Matrix

S. Bahisham[1], R. Suhairi[2], Z.M. Zain[1], N.M. Salih[1]
[1]Section of Electronics Technology
[2]Section of Electrical Technology
Universiti Kuala Lumpur, British Malaysian Institute, Gombak, Selangor

Corresponding email: suhaimib@bmi.unikl.edu.my

Abstract: Nowadays, more components are integrated into System on Chip (SoC) designs; caused inter-component communication often in the critical path of SoC design and is a very common source of performance bottlenecks.The need for very efficient bus becomes essential for meeting the high bit-rate demand of fundamental applications and also for low power constraint. These issues create unprecedented challenges in achieving SoC performance requirements of today's high bit-rate applications.It thus becomes imperative on system designers to focus on enhancing the performance of the whole system through the use of multi-layer bus matrix that provides efficient inter-module communication that leads to low power consumption.This paper discussed the different configurations of the AHB bus. Simulation is used to characterize the performance of the AHB bus. A customized AHB bus is proposed to avoid contention among competing peripherals over the bus access. With this customized AHB bus, it should be able to cater for high bit-rate applications utilizing the latest technology. With efficient bus,lower power consumption can be achieved.

Keywords: AHB, bus architecture, bus interconnects

1.0 INTRODUCTION

Currently, more components are integrated into these SoC designs to share the ever increasing processing load [1]. Over the years, SoC designs have evolved from fairly simple single processor, single-memory designs to massively complex multiprocessor systems with several on-chip memories, standard peripherals and custom blocks. There is a corresponding increase in the communication between these components. Inter-component communication is often in the critical path of a SoC design and is a very common source of performance bottlenecks. The concern with high performance SoC design is how to connect the components for efficient transfer of large amounts of data between components. The ultimate goal is to provide bottleneck-free integration for inter-module communications. Then through efficient bus architecture, lower power consumption can be attained due to a process can be completed in shorter time than non-efficient bus.

This paper will study exactly how different interconnection schemes affect the bus performance. Then this paper will propose a customized multilayer AHB to allow efficient and high-performance on-chip communications for high throughput that can be used in high bit-rate applications. This configuration extends the bus bandwidth in multiple masters system, improving SoC performance drastically. The customized multilayer AHB can accommodate a wide range of applications that need high processing power.

2.0 PROBLEM STATEMENT

The on chip buses have been the preferred interconnection method in most of the processor-based systems in the past years [6]. The single bus is a good choice when the number of connected components is small. Moreover this bus may not be the solution to the bandwidth problem because only one pair of master and slave can send and receive the data at a particular time [4]. It can not provide sufficient throughput for high bit rate applications due to many processes occurred at the same time. Also, contention is getting more serious in case of using the multi processors that require higher bandwidth. Multilayer AHB [5] will be used to resolve this bandwidth problem.

Multilayer AHB [6] provides parallel paths between the various masters and the slaves, giving improved overall system bandwidth. However multilayer AHB that are not conFig.d properly can lead to contention. Thus system designer needs to know how the SoC will function in the real application and to design the SoC properly to achieve the application required throughput.

Each master and slave has its own function [4]. System designer needs to know the functionality for each master and slave in the SoC system and then can determine how the bus should be arranged to minimize the contention and latency on the bus. Latency is caused by the access time to the bus until the bus is granted and also the latency introduced by the bus to transfer the data [2]. Effective high performance interconnect architecture depend on the specific target application domain [3].

In this paper, a customized multilayer AHB will be presented that decreases the latency time and provides better throughput. Three configurations have been developed and evaluated for the sake of comparison. The proposed bus configuration will be demonstrated with 2 test cases to evaluate the functionality and performance of the system.

3.0 BUS CONFIGURATIONS

There are three configurations are discussed and characterized in this paper: single layer configuration, multilayer configuration and proposed customize multilayer configuration. Presumably, in this SoC, there are 5 masters ARM9 Instruction Master (ARM_I), ARM9 Data Master (ARM_D), DMA Master 1 (DMA_1), DMA Master 2 (DMA_2), USB DMA Master (DMA_USB), 6 AHB slaves, memory controller (MCTL), interrupt controller (ICTL), USB, Arm gea2 interface unit (AGIU), Arm physical interface unit (APIU) and Advanced Peripheral Bus (APB).

3.1 CONFIGURATION 1 – SINGLE LAYER

For single layer, the bus interconnection is shown in Fig. 1. For Configuration 1, there are 5 masters on that layer. At one time, only one master can gain the bus ownership. If other masters need to use the bus, they have to wait for the current master to complete the job first then other masters can get the ownership. This will increase latency and decrease the throughput of the SoC performance. So, there will be a lot of contention occur in Configuration 1. This configuration definitely can't provide high throughput.

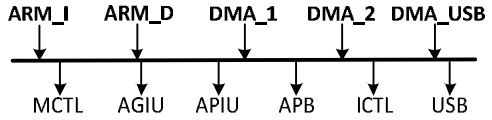

Fig. 1 Single Layer AHB Bus. Note: bold text is to designate Master, while plain text is to designate Slave

3.2 CONFIGURATION 2 – MULTILAYER

The multilayer bus consists of multiple physical buses that allow parallel paths between multiple masters and slaves. This gives the benefit of increased bandwidth on overall buses. Each AHB layer is connected to the shared slave by an interconnect matrix (BusMatrix). If two layers require access to the same slave at the same time, the arbitration within the interconnect matrix will determine which layer has highest priority. For multilayer, the bus interconnection is shown in Fig. 2.

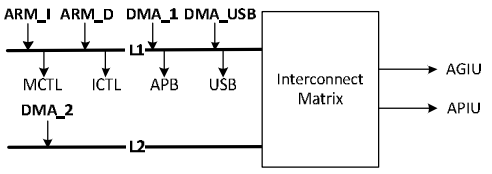

Fig. 2 Multilayer AHB Bus

For Configuration 2, there are two layers connected to an interconnect matrix to access the shared slaves between the layers. Since only ARM_I, ARM_D, DMA_1 and DMA_USB will access MCTL, MCTL has been placed as a local slave to Layer 1. The reason behind this is that there will be one clock cycle delay for each transfer that goes through BusMatrix [7]. In order to eliminate the unnecessary delay, MCTL has been placed as a local slave to Layer 1. In Layer 1(L1), there are 4 masters ARM_I, ARM_D, DMA_USB and DMA_1. Contention will occur on this layer if these 4 masters need to use the bus since each master has their own specific function and it is getting worse if ARM_I needs to fill the ARM9 cache. In this layer, DMA_1 has the highest priority, followed by DMA_USB, ARM_D and ARM_I is the lowest priority. In Layer 2(L2), there is only 1 master, DMA_2. Layer 2 has higher priority if these two layers want to access the shared slaves on the Interconnect Matrix. Masters that are less likely to access the bus at the same time should share the same layer.

3.3 CONFIGURATION 3 – CUSTOMIZED MULTILAYER

Configuration 3 is the proposed customized multilayer AHB. The bus interconnection is shown in Fig. 3. For Configuration 3, there are three bus layers. In Layer 1(L1), only ARM Data Master (ARM_D) is an AHB master. So, L1 will be conFig.d as AHB-Lite where ARM_D always gets the L1 bus ownership. ARM_D gets the access to all the components in the system.

In Layer 2(L2), there are 3 masters and all masters are accessing memory (DDR and FLASH) only through memory controller (MCTL). ARM_I master needs to fetch memory for ARM9 caches, DMA_USB master needs to do data transfer from/to memory to/from USB internally and DMA_1 master needs to transfer data from/to memory. Since masters access memory only, so they have been placed in one layer. It is useless to put these 3 masters in different layers because they only access memory. Contention still will occur if we put in a different layer because there is only a single memory controller to access the memory. On this layer, DMA_1 has the highest priority followed by DMA_USB and then ARM_I.

In Layer 3(L3), there is only one master, DMA_2. This master is to access data from/to the shared slaves. L3 also will be conFig.d as AHB-Lite. Using Configuration 3, it can provide higher bit rate system bus. For example, if stack needs to copy data from IR memory in DDR to IR memory in APIU, DMA_1 will access the IR memory in DDR and at the same time DMA_2 will transfer the data from DDR to IR memory in APIU. So, there will be no bottleneck for IR events. If we compare Configuration 3 with Configuration 1 and 2, there will be bus contention in Configuration 1 when we want to do IR event because DMA_2 and DMA_1 on the same layer. Only one master can get the ownership of the bus in a time. These worsen by the ARM_D and

ARM_I that need to access other components in the system. In Configuration 2, although DMA_1 and DMA_2 are at different layer, ARM_D and ARM_I will disrupt DMA_1 operation. For this configuration, L3 has the highest priority, followed by L2 and then L1. L3 will gain the access if other layers want to access the same shared slaves on the Interconnect Matrix.

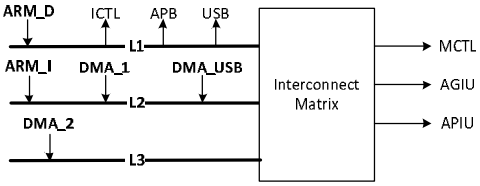

Fig. 3 Customized Multilayer AHB Bus

4.0 PERFORMANCE ANALYSIS

In this chapter, the method of the performance analysis on the bus configuration is described. The three bus configurations are implemented and their performances have been measured. For performance analysis, the performance has been characterized in terms of bus master request and total clock cycles to complete the testcases. The total clock cycle for each master bus request is measured. Through these results, we can see the occurrence of each master initiating bus request. These testcases have been developed to mimic similar applications in real time where a large data transfer will occur between the memory and AGIU/APIU.

The testcases have been developed using C language program to generate random traffic on the bus. Two types of testcases have been developed to characterize the performance for all bus configurations. The first type uses one DMA channel for data transfer between memory and AGIU only. The second type uses two DMA channels for data transfer between memory and AGIU/APIU. One channel was configured for data transfer from memory to AGIU and the other channel for data transfer from memory to APIU. These two channels will be enabled at the same time so that it will create more contention on the bus.

A Verilog testbench suite has been developed to run the simulation and measured the performance of the bus configurations. The bus configuration is instantiated in the testbench. Then the behavioral functional models are developed to drive the correct signals to the bus configuration. A DDR memory model has been used as the system memory. The C programs are compiled and a hex code is produced. This hex code is loaded into the DDR memory using the testbench. A behavioral functional model is used to control the reset of the testbench suite. After the reset signal has been de-asserted, the ARM_I will load the instructions set from DDR memory through the MCTL. After that, the C programs are executed as explained above. During simulation, the intended measured results are captured by the testbench. When a master initiate a bus request, the testbench will start counting the total clock cycle until the master de-asserted the bus request. This is done for each master involved in the bus configurations. After the simulation finished, the measured results are reported in a log file.

Each bus configuration is simulated and results are measured. From the results for all bus configurations, Table 1 and Table 2 have been created to summarize the results in appropriate manner.

5.0 RESULTS AND DISCUSSIONS

Table 1 shows the results of the simulation for one DMA channel, while Table 2 shows the results for two DMA channels. Simulations on various data Sizes for each type of testcase have been run. The '%' column is to indicate changes in percentage point compared to Configuration 1. All the figures in the tables below are in clock cycles unit.

Table 1 Results for using 1 DMA channel

6,000bytes

	Total	%	ARM_D	%	ARM_I	%	DMA_1	%	DMA_2	%
Configuration 1	134,424		27,565		90,240		8,458		5,964	
Configuration 2	133,956	-0.4	26,760	-3	88,225	-2	6,511	-23	1,501	-75
Configuration 3	112,452	-16	3,377	-88	80,229	-11	6,534	-23	1,501	-75

15,000bytes

	Total	%	ARM_D	%	ARM_I	%	DMA_1	%	DMA_2	%
Configuration 1	152,298		30,542		107,594		20,637		14,765	
Configuration 2	148,959	-2	28,725	-6	102,608	-5	16,006	-22	3,751	-75
Configuration 3	127,274	-16	3,660	-88	94,526	-12	16,020	-23	3,751	-75

45,000bytes

	Total	%	ARM_D	%	ARM_I	%	DMA_1	%	DMA_2	%
Configuration 1	213,829		40,904		167,236		63,343		44,610	
Configuration 2	200,041	-6	36,428	-11	151,728	-9	48,829	-23	11,251	-75
Configuration 3	177,067	-17	4,593	-89	143,067	-14	48,994	-23	11,251	-75

250,000bytes

	Total	%	ARM_D	%	ARM_I	%	DMA_1	%	DMA_2	%
Configuration 1	633,172		111,556		573,360		352,445		248,175	
Configuration 2	549,700	-13	87,822	-21	487,372	-15	272,103	-23	62,626	-75
Configuration 3	518,466	-18	11,032	-90	474,616	-17	272,746	-23	62,626	-75

750,000bytes

	Total	%	ARM_D	%	ARM_I	%	DMA_1	%	DMA_2	%
Configuration 1	1,610,524		280,423		1,518,524		1,030,005		726,176	
Configuration 2	1,383,378	-14	205,478	-26	1,287,309	-15	798,666	-22	187,501	-74
Configuration 3	1,334,167	-17	26,680	-91	1,266,225	-17	802,512	-22	187,501	-74

1,500,000bytes

	Total	%	ARM_D	%	ARM_I	%	DMA_1	%	DMA_2	%
Configuration 1	3,099,216		535,454		2,958,922		2,060,006		1,452,340	
Configuration 2	2,643,402	-15	385,824	-28	2,496,416	-16	1,597,367	-22	375,001	-74
Configuration 3	2,566,181	-17	50,180	-91	2,462,037	-17	1,605,021	-22	375,001	-74

Table 2 Results for using 2 DMA channels.

6,000bytes

	Total	%	ARM_D	%	ARM_I	%	DMA_1	%	DMA_2	%
Configuration 1	260,896		49,893		171,706		17,572		13,824	
Configuration 2	256,890	-2	50,328	+1	164,294	-3	10,698	-39	3,752	-73
Configuration 3	213,952	-18	6,311	-87	148,599	-13	10,695	-39	3,752	-73

15,000bytes

	Total	%	ARM_D	%	ARM_I	%	DMA_1	%	DMA_2	%
Configuration 1	291,194		50,587		200,565		43,715		34,408	
Configuration 2	275,815	-5	51,129	+1	181,914	-10	26,537	-39	9,380	-73
Configuration 3	231,999	-20	6,483	-87	165,820	-17	26,566	-39	9,380	-73

45,000bytes

	Total	%	ARM_D	%	ARM_I	%	DMA_1	%	DMA_2	%
Configuration 1	390,791		52,798		296,960		131,102		103,180	
Configuration 2	337,644	-14	53,553	+1	240,286	-19	79,643	-39	28,140	-73
Configuration 3	291,745	-25	7,014	-87	223,128	-25	79,653	-39	28,140	-73

250,000bytes

	Total	%	ARM_D	%	ARM_I	%	DMA_1	%	DMA_2	%
Configuration 1	1,073,330		67,799		957,223		730,093		574,800	
Configuration 2	761,121	-29	70,333	+4	640,158	-33	443,484	-39	156,646	-73
Configuration 3	701,140	-35	10,713	-84	615,661	-36	443,554	-39	156,646	-73

750,000bytes

	Total	%	ARM_D	%	ARM_I	%	DMA_1	%	DMA_2	%
Configuration 1	2,731,512		104,448		2,562,499		2,184,983		1,719,646	
Configuration 2	1,789,994	-34	110,745	+6	1,611,415	-37	1,327,337	-39	469,000	-73
Configuration 3	1,695,645	-38	19,704	-81	1,568,935	-39	1,327,245	-39	469,000	-73

1,500,000bytes

	Total	%	ARM_D	%	ARM_I	%	DMA_1	%	DMA_2	%
Configuration 1	5,221,658		159,421		4,972,167		4,369,949		3,439,282	
Configuration 2	3,335,016	-36	171,565	+8	3,070,022	-38	2,654,666	-39	938,000	-73
Configuration 3	3,189,133	-39	33,204	-79	3,000,688	-40	2,654,469	-39	938,000	-73

In Table 1, the results are for the total number of clock cycles consumed by master's request versus different configurations and for different masters. The second left most column represents total clock cycles needed to complete the testcase. The percentage column shows the improvement compared to Configuration 1. The fourth left most column represents total clock cycles of bus request for ARM_D master and the percentage column shows the improvement compared to Configuration 1. The sixth left most column represents total clock cycles of bus request for ARM_I master and the percentage column shows the improvement compared to Configuration 1. The fourth right most column represents total clock cycles of bus request for DMA_1 master and the percentage column shows the improvement compared to Configuration 1. The second right most column represents total clock cycles of bus request for DMA_2 master and the percentage column shows the improvement compared to Configuration 1. In Table 1, all the results are for using 1 DMA channel.

In Table 2, the results are for the total number of clock cycles consumed by master's request versus different configurations and for different masters. The second left most column represents total clock cycles needed to complete the testcase. The percentage column shows the improvement compared to Configuration 1. The fourth left most column represents total clock cycles of bus request for ARM_D master and the percentage column shows the improvement compared to Configuration 1. The sixth left most column represents total clock cycles of bus request for ARM_I master and the percentage column shows the improvement compared to Configuration 1. The fourth right most column represents total clock cycles of bus request for DMA_1 master and the percentage column shows the improvement compared to Configuration 1. The second right most column represents total clock cycles of bus request for DMA_2 master and the percentage column shows the improvement compared to Configuration 1. In Table 2, all the results are for using 2 DMA channels.

If we compare the total bus request for all masters, there are significant differences between these three configurations. As described in Chapter 3, L1 in Configuration 3 has only one master, whereas Configuration 2 has fewer masters compared to Configuration 1. This means ARM_D in Configuration 3 can perform its function faster than the other configurations. For Configuration 1 and 2, ARM_D needs to compete with other masters to obtain the bus ownership, subject to bus contention if two or more masters attempt to get the bus ownership at the same time. This is shown by the results for ARM_D in Table 1. The results for ARM_D in Table 1 show that there are significant differences between Configuration 1, Configuration 2 and Configuration 3. Total bus requests for ARM_D in Configuration 3 decrease about 90% compared to Configuration 1. Therefore, ARM_D in Configuration 3 can be used to carry out more data operation as needed. Although there is also decrement in terms of ARM_D for Configuration 2 compared to Configuration 1, however it is small compared to Configuration 3. These percentages prove that ARM_D in Configuration 3 is less prone to contention compared to Configuration 2 and Configuration 1.

From Table 1, it is noticeable that there are also decrement in total requests for ARM_I. DMA_1 and DMA_2 use the bus very minimal compared to ARM_D and ARM_I. This is shown in Table 1 by comparing the total bus requests among ARM_D, ARM_I, DMA_1 and DMA_2. Since DMA_1 and DMA_2 seldom use the bus, the contention happens on the bus is also minimal. After DMA_1 and DMA_2 completed the data transfer, ARM_D and ARM_I can perform its function faster. In Configuration 1 and Configuration 2, ARM_D and ARM_I still need to compete for the bus ownership, whereas in Configuration 3 ARM_I need not to compete for bus ownership after DMA_1 and DMA_2 completed the data transfer. These results are shown in the Table 1 for ARM_I section.

From Chapter 3, we knew that DMA_2 has the highest priority and DMA_1 has higher priority than ARM_D and ARM_I. In Configuration 1, since these 2 masters have higher priority, they can gain the bus ownership even though other masters are utilizing the bus. However, they still need to wait for the previous master transfer to finish first before they get hold of the bus if other master is still using the bus. As discussed in Chapter 3, in Configuration 1, there were lots of contention caused by ARM_D, ARM_I, DMA1 and DMA_2, whereas in Configuration 2, the contentions are only caused by ARM_D, ARM_I and DMA_1. This shows that less contention is happening in Configuration 2 and this is the reason for the decrement in total bus requests for DMA_1 and DMA_2 compared to Configuration 1. In Configuration 3, although ARM_D was on L1, while ARM_I and DMA_1 on L2, most of the contentions on the bus are caused by the ARM_I, DMA_1 and DMA_2. This shows that the contentions on L2 in Configuration 2 and Configuration 3 are almost the same. That is the reason that the decrement percentages are similar between Configuration 2 and Configuration 3.

From the results shown in Table 2, similar analysis applies to Table 1 except for ARM_D results. For DMA_1 results, there are significant differences compared to results in Table 1. The results show that the decrement percentage of DMA_1 is almost doubles that in Table 1. When using 1 DMA channel, DMA_1 and DMA_2 were not using the bus efficiently. So, ARM_I and ARM_D disrupted the DMA_1 transfer more frequently. By using 2 DMA channels, DMA_1 and DMA_2 uses the bus more efficiently compared to using only 1 DMA channel. The disruption from ARM_I and ARM_D will also be lesser when DMA_1 performs

the data transfer because of the priority. When DMA_1 are able to execute the transfer faster, consequently ARM_I will be able to perform its function faster after DMA_1 completed the data transfer. However, there will be some increment in the total bus request for ARM_D in Configuration 2. ARM_D is used to access all the datas from all the peripherals except for the memory. In this testcase, ARM_D will access AGIU and APIU for data access within AGIU and APIU. Unfortunately, there was one clock cycle delay when a transfer went through BusMatrix. As the contention on L1 in Configuration 2 are quite enormous, the one clock cycle delay in the Busmatrix will make the condition worst, causing ARM_D total bus requests increase compared to Configuration 1.

If we compare the results between Table 1 and Table 2 for Configuration 3 in terms of total clock cycle to finish the testcase, the results for small data transfer between the usage of 1 DMA channel and 2 DMA channel show not have much difference. However, when the data grow bigger, there are significant difference between using 1 DMA channel and 2 DMA channel. In real application, SoC will be transferring a lot of data for its application once we power up the SoC. Thus, Configuration 3 is the best configuration compared to other configurations. It will provide better latency and by using multiple DMA channel, the bus efficiency will increase significantly.

6.0 CONCLUSIONS

Three different configurations of SoC based on AHB bus have been presented and the performance for each configuration was evaluated. From the performance analysis on the three different bus configurations, the proposed configuration, Configuration 3, has better latency which will be the enabler for high bit-rate applications. It is also find out that by using multiple DMA channels, the SoC bus will be utilized more efficiently. Consequently, lower power consumption can be obtained using the Configuration 3 in a SoC. For future improvement, it is better to have multi ports memory controller. Lots of contentions and latencies in SoC are on the memory access.

REFERENCES

[1]. AMBA Specification (Rev 2.0), ARM Limited, 1999.J. Hennessy and C. A. A. Q. A. D. Petterson, Computer

[2]. M. Dubois, Y. Savaria and G. Bois, “A Generic AHB Bus For Implementing High-Speed Locally Synchronous Islands”, SoutheastCon 2005, Proceedings of IEEE.

[3]. M. Mitic and M. Stojcev, “An Overview of On-Chip Buses”, SER : ELEC. ENERG. Vol 19, no.3, page 405-428, December 2006.

[4]. Multi-layer AHB Overview, ARM Limited, 2004. http://www.arm.com/pdfs/DVI0045B_multilayer_ahb_overview.pdf

[5]. Rudolf Usselmann, OpenCoresSoC Bus Review,2001. www.opencores.org/projects.cgi/web/wishbone/soc_bus_comparison.pdf

[6]. S.Y. Hwang and K.S. Jhang, “An Improved Implementation Method of AHB BusMatrix”, SOC Conference 2005, Proceeding of IEEE.

[7]. W. Ho and T. Pinkston, “A design methodology for efficient application-specificon-chip interconnects,” IEEE Trans. On Parallel and Distributed Systems,vol. 17, no. 2, pp. 174–190, Feb. 2006.